

**ED 548**

**[LIS]**

**THÈSE** présentée par :  
**Guillaume SANCHEZ**

soutenue le : 17 mai 2022

pour obtenir le grade de Docteur en Informatique  
Spécialité : Vision par ordinateur

**Creating and exploiting metadata for video  
content recommendation**

**THÈSE dirigée par :**

**M. BOUCHARA Frédéric**

**M. MARXER Ricard**

Maître de conférences HDR, LIS

Maître de conférences HDR, LIS

**Co-encadrée par :**

**Mme. GUIIS Vincente**

Ingénieur de recherche, LIS

**JURY :**

**Mme. GODIN Christelle**

**M. CHERUBINI Andrea**

**M. JOLY Philippe**

Directeur de recherche CEA, Examineur

Professeur, LIRMM, Rapporteur

Professeur, IRIT, Rapporteur

# Abstract

Deep Learning applied to computer vision has been shown to be able to extract many kinds of semantic information. From classification to localization, or pixel-level semantic segmentation, those new algorithms improved on the state-of-the-art of many tasks and many domains. The company I have been working with provides video streaming platforms for many customers. One of them wants to compete with other actors who have been investing in deep learning in order to improve their user experience. We aim at extracting semantic information that was not accessible before in order to make better personalized suggestions, emphasize on high quality content and propose new content browsing and exploration features. As such, in this work, we explore tasks such as face identification, activity recognition and recommender systems with an emphasis on latency and the ability to deploy at scale. Our contributions were made by developing three datasets from our industrial content. The first one is a study on data augmentation and pretrained models to train a classifier from an activity dataset for our data domain. Our second contribution is a survey on learning classifiers in presence of label noise. The next contributions revolve around face recognition. We propose a new loss function, the Threshold-Softmax, aiming to learn from negative samples, that is, faces whose identity is just known not to be one of the other classes. We revert back from metric learning to standard classifiers and explore four loss functions for exploiting further negative learning, using a dataset of faces labeled with their identity, of people famous in our customer's domain. We also contribute a face swapping model based on the Vector-Quantized Variational Auto-Encoder (VQVAE), along with a new algorithm to improve the vector quantization algorithm. Finally, we use the browsing history of premium users in order to learn a recommender system based on metadata, aiming to mitigate the cold start problem for both users and items.

## Résumé

Le Deep Learning appliqué à la vision par ordinateur s'est révélé capable d'extraire de nombreux types d'informations sémantiques. De la classification à la localisation, ou à la segmentation sémantique au niveau du pixel, ces nouveaux algorithmes ont amélioré l'état de l'art de nombreuses tâches et de nombreux domaines. L'entreprise dans laquelle je travaille fournit des plates-formes de streaming vidéo à de nombreux clients. L'un d'entre eux souhaite concurrencer d'autres acteurs qui ont investi dans l'apprentissage profond afin d'améliorer leur expérience utilisateur. Notre objectif est d'extraire des informations sémantiques qui n'étaient pas accessibles auparavant afin de faire de meilleures suggestions personnalisées, de mettre l'accent sur le contenu de haute qualité et de proposer de nouvelles fonctionnalités de navigation et d'exploration du contenu. Ainsi, dans ce travail, nous explorons des tâches telles que l'identification de visage, la reconnaissance d'activité et les systèmes de recommandation en mettant l'accent sur la latence et la capacité de déploiement à grande échelle. Nos contributions ont été réalisées en développant trois jeux de données à partir de notre contenu industriel. La première est une étude sur l'augmentation des données et les modèles pré-entraînés pour entraîner un classificateur à partir d'un ensemble de données d'activité pour notre domaine de données. Notre deuxième contribution est une étude sur l'apprentissage de classifieurs en présence de bruit d'étiquettes. Les contributions suivantes portent sur la reconnaissance des visages. Nous proposons une nouvelle fonction de perte, le Threshold-Softmax, visant à apprendre à partir d'échantillons négatifs, c'est-à-dire des visages dont l'identité n'est pas celle d'une des autres classes. Nous revenons de l'apprentissage métrique aux classifieurs standards et explorons quatre fonctions de perte pour exploiter davantage l'apprentissage négatif, en utilisant un jeu de données de visages étiquetés avec leur identité, de personnes célèbres dans le domaine de notre client. Nous proposons également un modèle d'échange de visages basé sur le Vector-Quantized Variational Auto-Encoder (VQVAE), ainsi qu'un nouvel algorithme pour améliorer l'algorithme de quantification vectorielle. Enfin, nous utilisons l'historique de navigation des utilisateurs premium afin d'apprendre un système de recommandation basé sur les métadonnées, visant à atténuer le problème du démarrage à froid pour les utilisateurs et les vidéos.

## Remerciements

Mes premiers remerciements vont à Hexaglobe, en particulier à Franck COPPOLA et Pierre-Alexandre ENTRAYGUES, et au client que je ne peux nommer, qui m'ont fait confiance et ont parié sur moi. Qui ont laissé place au travail académique et compris les difficultés et incertitudes inhérentes à la recherche, qu'elle soit appliquée ou pas. Ainsi qu'au LIS pour m'avoir accueilli, nommément Elisabeth MURISASCO et Eric BUSVELLE.

J'aimerais tout autant remercier Vincente GUIIS pour sa relecture si assidue et dévouée, pour son suivi indéniablement méticuleux, Ricard MARXER et Frédéric BOUCHARA pour l'aide fournie pendant ces années de thèse et avoir accepté cet encadrement et avoir confronté et guidé mes idées.

Merci à la science et tous les humains qui l'ont faite progresser et nous léguer une discipline aussi épanouissante, un monde aussi riche, et une compréhension du monde aussi vaste que nous l'avons maintenant. Merci aux pères fondateurs du Deep Learning pour avoir créé une discipline qui m'a autant intéressé.

Mes remerciements vont également à mes amis et collègues Maxence FERRARI, Marion POUPARD et Paul BEST (merci d'avoir fait gaffe aux outils) qui m'ont supporté dans les jours difficiles comme dans les jours de travail jovial, accompagné de ma Air Guitar et de Britney Spears. Qui ont partagé mes réflexions, au développement et à la critique de mes idées, enrichissant grandement ma compréhension, compétence, et mes qualités humaines. (Merci aussi pour le fromage).

Mes pensées chaleureuses vont également à tous les amis qui ont ensoleillé ce parcours. D'abord ceux du Bâtiment X : Baptiste DOMPS, Anatole GROS-MARTIAL, Manon SCHOLIVET, William BRUSCH, Nathan CARRIOT, Alexandre LUTZ, dont certains avec qui j'ai pu partagé des parties de jeu de rôle mémorables. Je n'oublie pas non plus mes amis externes à l'environnement académique : Christophe et Margaux, Camille et Gautier, les jumeaux ainsi qu'Horgix qui m'accompagne dans mes pérégrinations codistiques depuis plus de 10 ans.

Ma famille également, qui m'a présenté son soutien à bien des moments. Mes petits parents et mon frère, toujours adorables.

Et, plus que tout, exprimer ma reconnaissance envers ma tendre épouse Aurélie qui a été un soutien et un encouragement de chaque jour. Dans les yeux aimants et admiratifs de laquelle j'ai puisé ma force et ma détermination.

# Table of Contents

|   |             |
|---|-------------|
| <b>Abstract</b>   | <b>ii</b>   |
| <b>Remerciements</b>  | <b>iv</b>   |
| <b>List of Tables</b>   | <b>vii</b>  |
| <b>List of Figures</b>  | <b>viii</b> |
| <b>1 Introduction</b>   | <b>1</b>    |
| <b>2 Industrial Context</b>   | <b>3</b>    |
| 2.1 Introduction . . . . .  | 3           |
| 2.2 Problems and motivations . . . . .  | 3           |
| 2.3 Data . . . . .  | 4           |
| 2.4 Machines . . . . .  | 8           |
| 2.5 Conclusion . . . . .  | 8           |
| <b>3 Machine Learning</b>   | <b>10</b>   |
| 3.1 Introduction . . . . .  | 10          |
| 3.2 What is Machine Learning . . . . .  | 10          |
| 3.3 Neural Networks . . . . .   | 11          |
| 3.4 $\Rightarrow$ Practical Example: A classifier for HActions . . . . .          | 21          |
| 3.5 Conclusion . . . . .  | 23          |
| <b>4 <math>\Rightarrow</math>Contribution: Label noise for face recognition</b>   | <b>25</b>   |
| 4.1 Introduction . . . . .  | 25          |
| 4.2 Deep Learning Classification with Noisy Labels . . . . .                      | 26          |
| 4.3 Overview of techniques . . . . .  | 26          |
| 4.4 Experimental Setups . . . . .   | 28          |
| 4.5 Approaches . . . . .  | 29          |
| 4.6 Discussion . . . . .  | 31          |
| 4.7 Conclusion . . . . .  | 32          |
| <b>5 Face Recognition</b>   | <b>34</b>   |
| 5.1 Introduction . . . . .  | 34          |
| 5.2 Standard systems . . . . .  | 35          |
| 5.3 Metric learning . . . . .   | 36          |
| 5.4 $\Rightarrow$ Contribution: Face Recognition with Threshold-Softmax . . . . . | 38          |

|          |  |            |
|----------|--|------------|
| 5.5      | ⇒ <i>Contributions</i> : distractor-robust face recognition for a closed-set of identities . . . . | 41         |
| 5.6      | Conclusion . . . . .   | 53         |
| <b>6</b> | <b>Fixing Datasets with generative models</b>  | <b>54</b>  |
| 6.1      | Introduction . . . . .   | 54         |
| 6.2      | Generative models for building invariants . . . . .  | 54         |
| 6.3      | Problem definition . . . . .   | 54         |
| 6.4      | Generative Models . . . . .  | 55         |
| 6.5      | Latent-Variable Models and Variational AutoEncoders . . . . .                                      | 57         |
| 6.6      | Generative Adversarial Networks (GANs) . . . . .   | 62         |
| 6.7      | Conditional Modeling . . . . .   | 68         |
| 6.8      | Constrained Modeling . . . . .   | 70         |
| 6.9      | Controlled Unconditional GANs . . . . .  | 72         |
| 6.10     | Evaluation . . . . .   | 72         |
| 6.11     | ⇒ <i>Contributions</i> : Expiration for Vector-Quantized (VQ) codebook . . . . .                   | 73         |
| 6.12     | ⇒ <i>Contribution</i> : A Latent Variable Model for facial pose generation . . . . .               | 79         |
| 6.13     | Conclusion . . . . .   | 82         |
| <b>7</b> | <b>Recommender System</b>  | <b>84</b>  |
| 7.1      | Introduction . . . . .   | 84         |
| 7.2      | Recommender systems are hard to build . . . . .  | 85         |
| 7.3      | Types of recommender systems . . . . .   | 87         |
| 7.4      | Baseline algorithms . . . . .  | 88         |
| 7.5      | ⇒ <i>Project</i> : Hexaglobe’s RecSys . . . . .  | 91         |
| <b>8</b> | <b>⇒<i>Project</i>: Torchelie</b>  | <b>102</b> |
| 8.1      | Introduction . . . . .   | 102        |
| 8.2      | Overview . . . . .   | 102        |
| 8.3      | Design Principles . . . . .  | 105        |
| 8.4      | Discussion . . . . .   | 114        |
| 8.5      | Conclusion . . . . .   | 115        |
| <b>9</b> | <b>Conclusion</b>  | <b>116</b> |
| <b>A</b> | <b>Notations, conventions and acronyms</b>   | <b>132</b> |
| <b>B</b> | <b>Face recognition models additional plots</b>  | <b>135</b> |
| <b>C</b> | <b>ConvNeXt experiments</b>  | <b>138</b> |

## List of Tables

|     |   |     |
|-----|---|-----|
| 3.1 | Test accuracy on HActions according to various configurations . . . . .   | 22  |
| 4.1 | Approaches according to annotations in the dataset. Notes: TIMIT is a speech to text dataset, "NLP" is a set of natural language processing datasets (Twitter, IMDB and Stanford Sentiment Treebank), "face rec" denotes classical face recognition datasets (LFW, CALFW, AgeDB, CFP) . . . . . | 33  |
| 5.1 | Accuracy on face verification for LFW and FGLFW image pairs for various loss functions. Best rejection angular threshold selected for each method. . . . .  | 40  |
| 5.2 | Various metrics for each model, rejection threshold selected at maximal <i>total</i> accuracy. <b>Best</b> and <i>second best</i> results are highlighted . . . . .   | 47  |
| 5.3 | Various metrics for each model, rejection threshold selected at maximal F1. <b>Best</b> and <i>second best</i> results are highlighted . . . . .  | 47  |
| 5.4 | F1 AUC for the models evaluated. Value for ArcFace is normalized for comparability (0.26 to 0.34) . . . . .   | 47  |
| A.1 | Notations and conventions . . . . .   | 132 |

## List of Figures

|     |   |    |
|-----|---|----|
| 2.1 | Distribution of samples per classes in HActions. . . . .  | 5  |
| 2.2 | A few samples from HActions, with class label "none" . . . . .  | 5  |
| 2.3 | Number of examples per identities in the photos subset of HFaces (sorted). Most identities have between 10 and 100 samples. Each bar represents a different class, sorted from least to most populated. . . . .   | 6  |
| 2.4 | Samples from HFaces. Top row: extracted from pictures. Bottom row: extracted from videos. . . . .   | 6  |
| 2.5 | Number of views per videos (sorted). Each video is a thin vertical bar. . . . .   | 7  |
| 3.1 | A (very simplified) biological neuron and an artificial neuron (Perceptron) . . . . .   | 12 |
| 3.2 | A neural network training loop . . . . .  | 13 |
| 3.3 | Example of data augmentation. The original image is transformed to artificially generate new training examples. In this case, AutoAugment is used ; it combines multiple transformations and its settings depend on the dataset and the task. Figure from torchvision's documentation. Each row shows the set of augmentations used for those datasets on a sample image. . . . .   | 14 |
| 3.4 | AlexNet architecture, built from convolutions (CONV), pooling operations (POOL), and linear layers (FC). Figure from Krizhevsky et al. [88] . . . . .   | 15 |
| 3.5 | GoogLeNet / Inception-v1. It uses complex building blocks, aggregating different design decisions. Figure from <a href="https://medium.com/@RaghavPrabhu/cnn-architectures-lenet-alexnet-vgg-googlenet-and-resnet-7c81c017b848">https://medium.com/@RaghavPrabhu/cnn-architectures-lenet-alexnet-vgg-googlenet-and-resnet-7c81c017b848</a> 16   | 16 |
| 3.6 | VGG network. Grey: 3x3 convolution layers; red: pooling layers; blue: linear (or 1x1 convs) layers; green: softmax. Each activation is described as Height $\times$ Width $\times$ Channels. 17   | 17 |
| 3.7 | In ResNets, an identity path is added every two convolutions, so that the gradient can flow up to the first layers untouched. Figure from [59] . . . . .  | 18 |
| 3.8 | Gradient field of the L2 loss function over a $\mathbb{R}^2$ plane. The black line shows the minimization trajectory from the black dot. We observe that this loss penalizes each variable proportionally to its value. . . . .   | 19 |
| 3.9 | Gradient field of the L1 loss function over a $\mathbb{R}^2$ plane. The black line shows the minimization trajectory from the black dot. We observe that this loss penalizes equally each variable, encouraging sparsity. . . . .   | 20 |
| 5.1 | Figure 17 from Wang and Deng [165]. The comparison of different training protocol and evaluation tasks in FR. In terms of training protocol, FR can be classified into subject-dependent or subject-independent settings according to whether testing identities appear in training set. In terms of testing tasks, FR can be classified into face verification, close-set face identification, open-set face identification. . . . . | 36 |



|      |  |    |
|------|--|----|
| 5.2  | Training of classification-based metric learning algorithm. The algorithm is trained to classify faces in different identities, with a margin in softmax and representation constraints. . . . .   | 36 |
| 5.3  | Test time usage of feature vectors. Two images' representations are compared under a distance metric. A distance under a predefined threshold indicates the same identity. . .   | 37 |
| 5.4  | Comparison of the angular softmax, ArcFace and the proposed Threshold-Softmax. In ArcFace, the margin (in blue) is fixed but the width of the arcs of each class can be arbitrarily wide (or narrow), since there is no constraint on them. In threshold-softmax, there are no enforced margins but the decision boundaries have a fixed width. An artificial class is predicted outside of those trusted cones. Figure derived from [163]. . .  | 39 |
| 5.5  | Threshold-softmax with negative samples: crosses are negative samples. We do not know their identities, we just know they do not belong to any of the known identities. Threshold-Softmax naturally uses those samples by placing their identities outside of the known classes decision boundaries, ie, predicting class $\Omega$ . . . . .   | 40 |
| 5.6  | Accuracy on LFW and FGLFW according to the hyper parameter threshold value $m$ for Threshold-Softmax. . . . .  | 41 |
| 5.7  | Rank 1 identification performance for contestants on Megaface's Facescrub challenge under various quantities of distractors. Most models have their performance degrading quickly even with 100 distractors only. Figure from [79]. . . . .  | 42 |
| 5.8  | Each class (abbreviated to the first letter of the person's name) is placed on this grid depending on its precision and recall scores. Top right is best, bottom left is worst. We aim to find strategies that help moving each point right and up. (Produced by the DCE model, Section 5.5.5) . . . . .   | 44 |
| 5.9  | A, B, and C are three fictitious identities for illustration purposes. We compute some metrics by selecting various meaningful subsets from the confusion matrix: <b>distractor accuracy</b> (blue rectangle), <b>identification accuracy</b> (orange rectangle), <b>kept accuracy</b> (violet rectangle), and <b>total accuracy</b> (green rectangle). For each subset, the metric is computed as the sum of the green cells it contains divided by the sum of all cells it contains. . . . . | 45 |
| 5.10 | Identification accuracy and total accuracy for a true acceptance rate of 95% (top) and 99% (bottom). CE: Cross-Entropy, DCE: Cross-Entropy+Distractors, ME: Cross-Entropy+MaximumEntropy, zlog=Zero-Logits. . . . .  | 48 |
| 5.11 | Various metrics for the a) ArcFace b) CE c) DCE d) ZLog e) ME model as predictions are set as distractors under various threshold values. The black bar traverses all plots at the best total accuracy, the dotted bar is located at maximum F1. As we sweep over the threshold values and reject more samples as distractors, we look at the variations on the metrics. . . . .   | 49 |
| 5.12 | Calibration plots for the a) CE b) DCE c) ZLog d) ME models. See Section 5.5.2 for more details about calibration. . . . .   | 50 |
| 6.1  | $G$ is a generator that turns a person identifier $y_i$ and a latent variable $z_i$ into an image. . . . .   | 55 |
| 6.2  | Conditional probability graph of an autoregressive model. Each pixel depends on the previous ones, iteratively. . . . .  | 56 |
| 6.3  | A PixelCNN sampling a pixel value for the current pixel from its surrounding context. White pixels are still undetermined grey pixels have already been sampled. Shown in red is the softmax output describing the probability distribution of the current pixel values conditioned on the context window. Image from Kolesnikov and Lampert [83] . . . . .  | 56 |

|      |   |    |
|------|---|----|
| 6.4  | Conditional probability graph of a Latent Variable Model (LVM). The whole image $x$ is sampled at once from a lower dimensional encoding $z$ . . . . .  | 57 |
| 6.5  | Training a latent variable model for colorization. There are multiple possible colorizations for a single greyscale input. A latent extractor $h$ extracts the information solving the ambiguity between those multiple answers ; an information bottleneck prevents the latent extractor from encoding all of the target and short-circuiting the task. The colorizer $f$ resolves ambiguous cases using the latent. . . . .   | 60 |
| 6.6  | Figure from [155] developping the quantization process. . . . .   | 61 |
| 6.7  | <b>top:</b> Training a Vector-Quantized Variational AutoEncoder (VAE) (VQ-VAE) stage 1: a quantized encoder and decoder are trained in an autoencoding fashion. <b>bottom left:</b> Training a VQ-VAE stage 2: the encoder is frozen and an autoregressive prior is learnt on the extracted latents. <b>bottom right:</b> Sampling from a VQ-VAE: We generate a latent variable from the prior model and decode it to a full picture . . . . .  | 61 |
| 6.8  | Training a standard GAN. <b>top left:</b> $G$ is kept frozen, we teach $D$ to classify a fake sample as a fake image with a Binary Cross-Entropy (BCE) loss $\text{BCE}(D(x_f, 0))$ . <b>top right:</b> $D$ is taught to classify a real sample with $\text{BCE}(D(x_r, 1))$ . <b>bottom:</b> we train $G$ to produce images that are classified as true by $D$ with $\text{BCE}(D(x_f, 1))$ , $D$ is kept frozen. . . . .  | 63 |
| 6.9  | Interpreting $D$ as a trainable loss giving low values to real samples and high values to fake samples. $G$ learns to minimize the loss $D$ represents. Gradients of fake samples represented as white arrows. . . . .  | 63 |
| 6.10 | An example of GAN training collapse. The generated samples suddenly ceases converging towards realistic samples, and the GAN never escapes this degenerate state. Image source: <a href="https://www.mathworks.com/help/deeplearning/ug/monitor-gan-training-progress-and-identify-common-failure-modes.html">https://www.mathworks.com/help/deeplearning/ug/monitor-gan-training-progress-and-identify-common-failure-modes.html</a> . . . . .   | 65 |
| 6.11 | Effect of regularizers. <b>Top:</b> $D$ is trained without a regularizer. The loss landscape might be noisy and hard to optimize against. There are strong peaks and valley because of the unregulated Lipschitzness. <b>Bottom:</b> $D$ is trained with R1 or Wasserstein GAN with Gradient Penalty (WGAN-GP) regularizers, smoothing the surface around real data points or just controlling $D$ 's Lipschitzness. The gradients are more predictive of the correct optimization direction, the loss is easier to optimize against, the peaks and valley are smoother than the unregulated version. Note: these surfaces are just for illustrative purposes and are not visualizations of actual loss surfaces. . . . . | 67 |
| 6.12 | A cGAN. The discriminator and generator are both conditioned on $y$ . . . . .   | 68 |
| 6.13 | Examples of image translation from the original pix2pix paper [72]. $x$ is a real image, $y$ a label, and $G(y, z)$ a fake sample produced by the generator. . . . .  | 69 |
| 6.14 | In BigGAN, $G$ generates samples from features and $E$ generates features from samples. Both pairs are discriminated, forcing $G$ and $E$ to reciprocate each other. Figure fom [38].   | 70 |
| 6.15 | In the InfoGAN, the generator is fed with a random noise $z$ and random categorical and continuous random codes $c$ . The discriminator pushes the generator towards real samples. $Q$ tries to guess $c$ and $G$ cooperates, ideally leading to $G$ utilizing $c$ in an interpretable way so that $Q$ can identify them back in the generated samples. Figure from [97]. . . . .   | 70 |
| 6.16 | The CycleGAN architecture. Image from <a href="https://towardsdatascience.com/image-to-image-translation-using-cyclegan-model-d58cfff04755">https://towardsdatascience.com/image-to-image-translation-using-cyclegan-model-d58cfff04755</a>   | 71 |

|      |   |    |
|------|---|----|
| 6.17 | Precision/Recall estimation: white dots are 2D representations of generated samples and black dots are real samples representations. Top figure shows the fake manifold estimation (in blue), the ratio of black dots inside the manifold shows the recall, ie, the ratio of the real dataset covered by the generator. Below, we show the manifold of the real dataset. The ratio of white dots inside the blue zone is the precision, ie, the ratio of generated samples that look like real samples. The spheres are drawn from each point in the manifold to estimate to its $k$ -th nearest neighbor. For those visualization we set $k = 2$ . . . . . | 74 |
| 6.18 | Vector Quantization illustration. Black points are codebooks prototypes. They divide the space into Voronoi cells. White points are input vectors, quantized to the prototype of the Voronoi cell they fall in. (1) shows the commitment loss as white arrows, bringing the input vectors closer to the prototype they have been assigned to. The prototype in cell (2) is not used in this iteration, its unused age is incremented. When, like in cell (3), that prototype has not been used for too long (more iterations than <code>limit</code> ), it is resampled to a random input vector and its age is reset to 0. . . . .                         | 75 |
| 6.19 | Histogram of age (time since last use) of each VQ layer codepoint after 20k training iteration. <i>left</i> : Without the expiration process, the optimization is harder and the net fails to use the codebook to optimize the loss. A lot of the codes remained unused for at least 2k iterations, presumably dead. <i>right</i> : Expiring and resampling code allows for exhaustive use of the codebooks and controllable entropy. Even if the maximum age is set to 250 iterations, the codebook has a much lower age on average. . . . .   | 77 |
| 6.20 | Experiment comparing test loss (left) and codebook usage Perplexity (PPL) (right) with a ReLU layer before quantization. Expiration VQ achieves lower loss and the perplexity scales correctly. . . . .   | 78 |
| 6.21 | Experiment comparing test loss (left) and codebook usage PPL (right) for a codebook of 32 code points. Horizontal axis: training iterations, blue: VQ with expiration, orange: VQ without expiration. . . . .   | 78 |
| 6.22 | Our proposed controllable face generator. A target face is encoded to a latent, decoded back with the identity label to the original picture. An information bottleneck in the encoder discourages the latent variable to contain any information about the person's identity, thus not leaking any identity specific geometry and encoding parameters not recoverable from the identity alone: lighting, pose, makeup, etc. At inference time, one can use any latent from any sample or sample a latent from a prior distribution to reenact anyone's face. . . . .   | 80 |
| 6.23 | Four batches of not curated samples. Rows 1, 3, 5, 7 are reconstructed samples. Identity is randomly swapped in rows 2, 4, 6, 8 but latent vector is kept untouched. . . . .  | 82 |
| 7.1  | Long tail. A few popular items (the head) get a significant higher number of view than the majority (the tail). Exploiting only head items results in neglecting most items. The Y axis is clamped at 2k views but the highest video count is 8k. . . . .   | 87 |
| 7.2  | In collaborative filtering, we aim to guess the ratings one user would give to an item given the rating similar users gave. Would she like Shrek because she liked The Dark Knight like user 1, or dislike Shrek because she liked Memento like user 2? Picture from <a href="https://developers.google.com/machine-learning/recommendation/collaborative/basics">https://developers.google.com/machine-learning/recommendation/collaborative/basics</a> . . . . .  | 88 |

|     |   |     |
|-----|---|-----|
| 7.3 | This imaginary app store has 3 apps: a science app, a robot game, and a dentist appointment finder. Those apps and John's interests are annotated by a set of tags shown above the table. Based on John's past interests, the first item, the science app, seems to be a good recommendation: John's and this app's feature vector share the greater similarity.  | 89  |
| 7.4 | The ratings matrix is decomposed as the inner product of user latent factors and movies latent factors, discovered during learning. They can be inspected to find semantically meaningful features. Image source: <a href="https://developers.google.com/machine-learning/recommendation/collaborative/basics">https://developers.google.com/machine-learning/recommendation/collaborative/basics</a> . . . . .   | 90  |
| 7.5 | Overview of the model. We sample a user, randomly sample a video from the watch history, and cut the history at its watch timestamp $t$ . The user is embedded by a user network while videos are encoded with a video network. The dot product of their embedding is computed and fed to a softmax + negative log likelihood loss, trained to predict the next video watched. The $\cdot$ denotes the dot product / matrix multiply operation.         | 92  |
| 7.6 | Illustrating word2vec training. A linear model trains word embeddings either by predicting the center word of a context window, or the context words of a context window from the center words. . . . .   | 95  |
| 7.7 | We train and evaluate our recommender system in a contrastive way. Batches of 256 pairs of histories and their next viewed video are loaded; the encoders learn to embed them so that the dot product of the real pair is greater than the ones of the other possible pairs formed in the batch. In other words, the encoders are learned so that $u_i \cdot v_i > u_i \cdot v_j$ with $i \neq j$ . . . . .   | 96  |
| 7.8 | Experiments on video encoder for a fixed user encoder. We aim to understand how features contribute to the classification information and build a model from this information. Test Top1 accuracy is indicated. . . . .   | 97  |
| 7.9 | Experiments on user encoder for a fixed video encoder. We aim to understand how features contribute to the classification information and build a model from this information. Test Top1 accuracy is indicated. Unless indicated otherwise, the history length $H$ is set to 2 . . . . .  | 99  |
| 8.1 | Visualization of <code>torchelie.hyper</code> hyperparameter search. The user can select hyper parameters to sample (and how to sample them), and target metrics. Once ran, the results appear in this visualization. In this case, we highlighted via the interface the three runs with the best resulting accuracy. . . . .   | 104 |
| 8.2 | The <code>ClassificationInspector</code> allows to see live the performance of the classifier. It reports the samples that are provide the best, worst, and most confused answers from the classifier. The bar below the images is green when the prediction is correct, red otherwise; the width reflects the confidence score of the prediction. This allows eyeballing the datasets, strengths and weaknesses of the model, and build intuition. . . | 105 |
| 8.3 | Live confusion matrix provided automatically when the number of classes is not too big to make it unreadable (less than 25 classes). . . . .  | 106 |
| 8.4 | Gradient of the loss wrt the input on the current batch. The per-pixel norm of the gradient weighs each pixel's intensity. This helps figuring out what the model looks at in the picture in order to make its predictions . . . . .  | 107 |
| B.1 | Additional plots for the ArcFace model (Section 5.5.5) . . . . .  | 135 |
| B.2 | Additional plots for the CE model (Section 5.5.5) . . . . .   | 136 |
| B.3 | Additional plots for the DCE model (Section ??) . . . . .   | 136 |
| B.4 | Additional plots for the ZLog model (Section 5.5.5) . . . . .   | 137 |

|     |  |     |
|-----|--|-----|
| B.5 | Additional plots for the ME model (Section 5.5.5) . . . . .  | 137 |
| C.1 | Incremental improvement process from ResNet to ConvNext. Figure extracted from Liu et al. [102]. . . . . | 138 |

# 1 Introduction

This thesis summarizes four years of work and research in collaboration with Hexaglobe for my PhD. This took place in the Université de Toulon, Laboratoire Informatique et Système. The section I am in focuses on solving problems with automated statistical approaches, commonly called Machine Learning. Machine Learning uses algorithms that are able to learn patterns from data in order to make predictions on new data. In the last decade, neural networks, a class of those algorithms, got a lot of traction. Researchers managed to stack many layers of neural networks, an approach now dubbed Deep Learning. Computer Vision, treats images in order to understand or process them in various way. Tremendous progress was achieved in Computer Vision thanks to new Deep Learning techniques, which will be the main focus of this work.

Hexaglobe is a company providing video distribution platforms to many customers. The work plan was to dedicate the research and innovation efforts to a single customer willing to invest in order to lead its market. This customer has huge quantities of data, possibility to label datasets, and can provide hardware, making it a convenient deep learning research environment. As such, there is a strong emphasis on applied research as the problems treated are motivated by industrial challenges. The modern computer vision developments, started in 2015, were seen as a opportunity to modernize the underlying software of the video platform, enriching user experience through semantic analysis of the content.

The customer and Hexaglobe were motivated by the numerous press articles emphatic about computer vision astonishingly fast progresses in the deep learning era. They wanted to investigate how useful could deep learning be for a video streaming platform. Could new computer vision algorithms extract semantic information from pixels, useful for enhancing the user experience? Can deep learning outperform the recommender system currently in place, based on manual heuristics and popularity scores, using semantic information instead? Can we recognize and annotate persons famous in our domain, at scale (both in number of samples to label and identities to recognize)? Can deep learning be used to **create and exploit metadata for video content recommendation**?

A two steps work plan was made: first, extract face recognition metadata from videos, then build a recommendation engine using them and other available features and metadata. Chapter 2 will proceed to give more context about Hexaglobe, and the customer's datasets. Then, Chapter 3 will outline how modern computer vision algorithms work: define the main components and exemplify with an image classification project. Chapter 4 acknowledges that our face recognition training dataset has noisy label issues and investigates state of the art methods for detecting and mitigating this issue. Chapter 5 deals with face recognition in itself. We will see how one can leverage modern generative model with the intent to reinforce face recognition models in chapter 6. Chapter 7 lays out how we are building our customer's recommender system. Finally, Chapter 8 will present the code framework developed supporting both research and industrial work.

This document also highlights contributions:

1. a survey on label noise in the context of deep image classification (Chapter 4), that was contributed in SANCHEZ et al. [138];
2. a novel loss function for metric learning applied to face recognition, Threshold-Softmax (Section 5.4);
3. a system for using and detecting distractors in the context of open-set face recognition (Section 5.5);
4. improvements over the original Vector Quantized information bottleneck (Section 6.11), one being contributed in Łańcucki et al. [90] and another described in this chapter;
5. a latent variable model for face swapping (Section 6.12);
6. a study of various design options for designing our customer's recommender system (Section 7.5);
7. a novel framework for deep learning work (Chapter 8), publicly available at <https://github.com/Vermeille/Torchelie>.

## 2 Industrial Context

### 2.1 Introduction

In order to understand the work that has been done during this PhD, it is preferable to first contextualize it. Hexaglobe will be presented first, along with their problems and motivations. Then, Section 2.3 will introduce the datasets developed internally in order to approach the problems that we aim to solve.

Hexaglobe provides all types of companies in the modern media landscape with technologies and professional services covering the entire process from video ingest to delivery.

Customers are numerous and diverse, from TV channels to radios and Video On Demand (VOD) producers. Hexaglobe takes care of the whole video life cycle: uploading, storage, metadata extraction and management, encoding, referencing, searching, and serving.

### 2.2 Problems and motivations

While classical software engineering is the right tool for encoding, managing, and serving the videos, it almost forces to consider them as impenetrable data blobs. Classical software engineering is very limited in the understanding and semantics that can be extracted from those data.

This is unfortunate. Being able to access the semantics of the videos would help improve the user experience in many ways [94]: it would make search more accurate, would allow for fine grain classification or help with recommendations. It would also be useful for the customers that would have more accurate insight into the type of videos that get more views, and would help data analytics with more intelligence. This is especially true for collaborative platforms, akin to YouTube, where the videos are uploaded by non professional people and the user given metadata cannot be relied on.

Very recent development have seen models such as Contrastive Language-Image Pretraining (CLIP) [127] able to query images from natural languages, confirming the feasibility of the task.

The customer I have been affected to has a massive database of 9M+ user uploaded videos since 2006. That data has very little annotations and metadata, and it is fundamental that those videos can get recommended in a more impactful way, get more accurate categorization and more precise search. All in all, what is needed is to ease browsing and exploit that massive data and make sure nothing high value lies dormant or drowns in this data ocean.

It would also be interesting to modernize the older content with quality enhancement and super resolution techniques, a thing I kept in mind while learning about generative models. This, however, is beyond the scope of this thesis.



## 2.3 Data

Throughout the thesis, three datasets have been created and are continuing to be developed as an ongoing process. Those datasets need to be refined, completed, and changed in order to fit the ever growing industrial needs. They are used for training and evaluating models before deployment.

1. HActions is an image dataset for activity recognition. It has been tailored to 12 popular activities in our domain;
2. HFaces is a face recognition dataset for the celebrities in our domain;
3. HHistory contains the browsing history of our premium users.

Due to the confidential nature of the datasets, the nature of the data and example samples cannot be revealed.

### 2.3.1 HActions

This first dataset has been manually labeled on my own. It contains pictures of natural images of people going on about what are supposed to be the 12 most popular activities on the website’s videos, labeled A to M, plus an extra class that represents any other activity and another one for title screens / text screens showing no humans. The distribution of data samples is shown in Figure 2.1 and a few samples in Figure 2.2.

In many activity recognition tasks, the environment can be extremely informative. For instance, karting, canyoning, sailing, driving, climbing, all take place in different environments and there are activity clues scattered all over the picture. However, in our situation, those activities are decided by body position rather than environment. In some sense, sleeping, singing, dining, watching TV or playing games all take place in a domestic environment and a classifier would have very little clues in the environment to sort them out. Special care would be needed to make sure the classifier does not overfit on spurious background elements for instance.

Since training and inference on full videos would require a lot of compute, we instead assume that still pictures taken from the videos contain enough information for the classification task. 300 evenly spaced frames are extracted throughout videos that lasts more than one minute. That way the computation budget remains fixed and controlled. For comparison sake, if we were to use all the video frames with a temporality aware model, 300 frames would represent a video of 10s only. Even if a video aware model performed much better, the associated costs would be too high, and reducing the frame rate in training and inference would need another pass of video transcoding, which already is expensive.

Instead of randomly splitting the pictures among train / validation / test sets, we randomly split the *videos* in those sets. Not doing this would create validation and test sets containing examples very similar to the ones in the train sets as nearby frames look similar. We aim to have a model that generalizes to videos outside the training set, not frames of a closed set of videos, thus the validation/test sets must contain frames of unseen videos.

### 2.3.2 HFaces

Automatic face recognition would bring very valuable metadata to our content. In our business, identifying celebrities in our domain is one of the core feature that the website’s visitors would find valuable.

The dataset contains 8938 identities, but is growing every day as we wish to recognize more people. The distribution of the number of pictures per identities is shown in Figure 2.3 and a few samples are

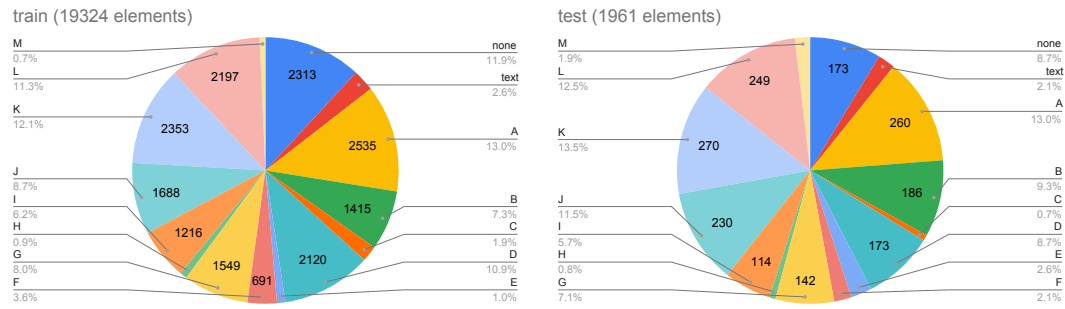


Figure 2.1: Distribution of samples per classes in HActions.



Figure 2.2: A few samples from HActions, with class label "none"

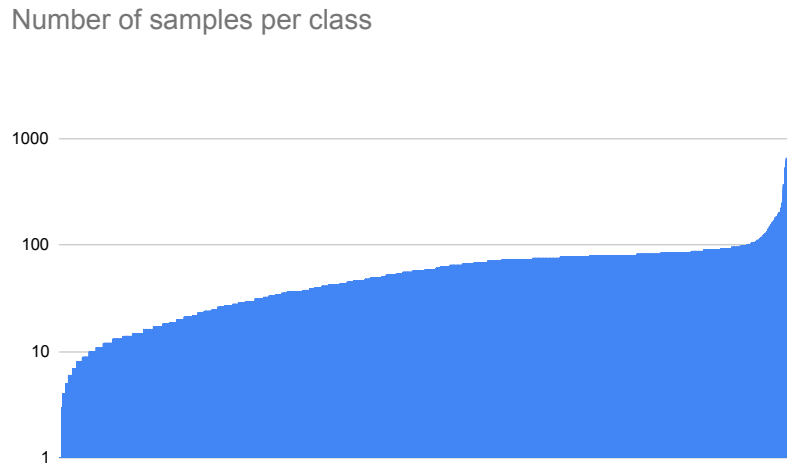


Figure 2.3: Number of examples per identities in the photos subset of HFaces (sorted). Most identities have between 10 and 100 samples. Each bar represents a different class, sorted from least to most populated.

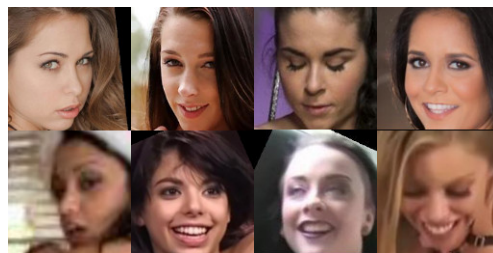


Figure 2.4: Samples from HFaces. Top row: extracted from pictures. Bottom row: extracted from videos.

shown in Figure 2.4. The dataset has been built with balance in mind. After collecting few very famous identities with web scraping, the labelers have been tasked to manually collect 100 pictures per identity when possible.

Our face dataset is divided into two data sources: faces coming from promotional pictures and videos. Faces coming from pictures are easier to collect but are prone to domain mismatch with videos as they are cleaner: the faces are usually not occluded, there is no motion blur, lighting is good and people tend to smile. In videos, none of this might hold true. We collected many pictures from photos and a smaller bunch from videos in order to evaluate and eventually mitigate the domain mismatch between both.

The dataset has been complemented with MS1Mv2 [54] to provide the so-called "distractors", developed in section 5.5.

### 2.3.3 HHistory

Finally, there is a dataset of premium users' browsing history. This dataset is here to provide historical data for building a recommender system, analyze trends, study semantic proximity of items, etc. This could be useful in various way : videos frequently watched together can serve as a contrastive / metric learning dataset to learn about visual features that makes them similar, can learn about tag proximity as well, etc. This dataset can be of many uses and the only metric really lacking is the lenght of watch time per video in order to assess the interest of the user.

## Number of views per video

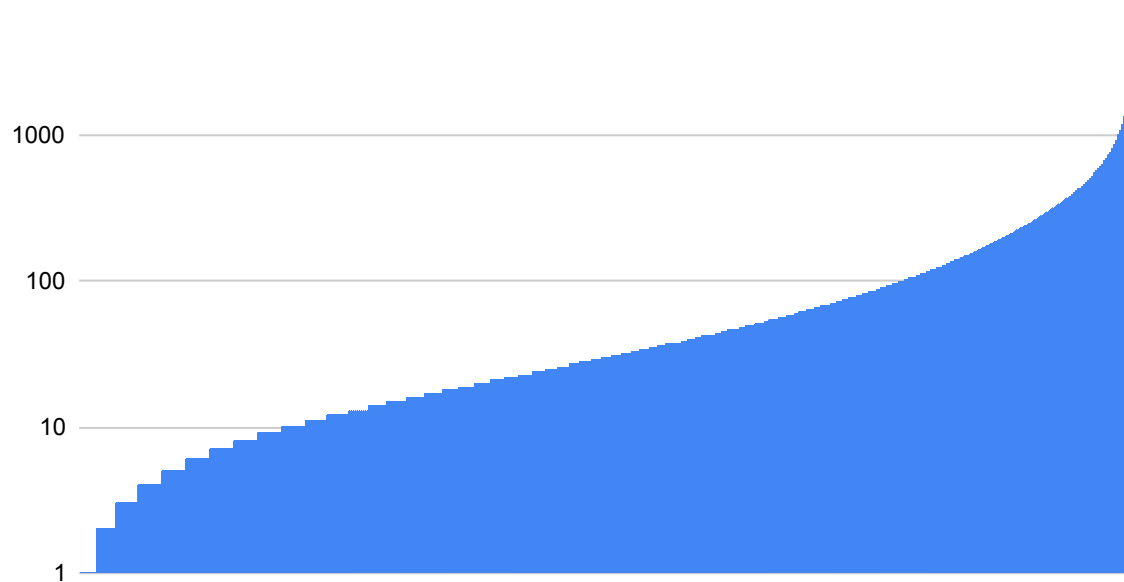


Figure 2.5: Number of views per videos (sorted). Each video is a thin vertical bar.

Premium content is only a subset of the whole content. The dataset has 135,984 users (content uploader), 3,673 channels and 139,711 videos. The distribution of views is in Figure 2.5. It contains various user information, many being optional:

1. user ID
2. username
3. country / region
4. gender
5. liked videos (and the like action timestamp)
6. disliked videos (and the dislike action timestamp)
7. favorited videos
8. watched videos (and their watch timestamp)
9. channels subscriptions

Similarly, metadata about videos are:

1. video ID
2. main category
3. the uploader's channel ID

4. people appearing in the video (from the face recognition model or the previous system)
5. title (possibly in multiple languages)
6. various free form tags chosen by the uploader
7. video encoding quality
8. upload timestamp

Finally, each channel has popularity scores per category and region.

### 2.3.4 A note about open datasets

The reader used to academic benchmark and comparisons might object and argue that there are open datasets covering the kind of problems dealt with here. Indeed none of those tasks are new. That being said, each use case is unique in its own way and models fitted on those datasets can't be straightly used for the industrial needs.

HActivity can be thought to be similar to YouTube-8M [3]. The actions in HActivity have zero overlap with YouTube-8M, so it definitely cannot be used there. Also, as discussed before, the actions in YouTube-8M are often highly informed by environmental clues and much less by body pose.

HFaces resembles CelebA [103] / MS1M [54]. Both those datasets have a vast majority of neutral faces facing the camera. In our situation, the faces are quite often non neutral and highly expressive, like faces during sports could be, are quite often not facing the camera, suffer from motion blur, occlusions, and MPEG encoding artifacts.

HHistory is harder to relate to MovieLens [57] and Netflix [13]. Those datasets are centered around ratings. While HHistory has some, they are both binary and scarce, and most of the training supervision has to be extracted from watch history instead.

## 2.4 Machines

To work, Hexaglobe provides me three machines, hosted by our Cloud computing provider.

1. gpu1 has 2 NVIDIA GTX 1080 Ti and is used mainly for production and inference.
2. gpu2 has 4 NVIDIA RTX 2080 and is used for my main experiments. It allows me to iterate quickly during development by using large batch sizes.
3. gpu3 has 2 NVIDIA RTX 2080. This machine is used for runs that I don't mind waiting for or side experiments.

The university also provided several clusters that I could use when working on public datasets.

## 2.5 Conclusion

In this chapter, we presented the company who supported the thesis and their industrial needs: they provide video platforms to customers. In order to improve their products, they would like to explore what Deep Learning has to offer for extracting metadata. Those metadata would be used to enrich searchability, sorting, user experience and recommendation. It then appears coherent to first focus on

extracting activity and face recognition to collect the most important features, then build a recommender system using them.

We are developing three datasets, one for each task: HActivity for activity recognition, HFaces for face recognition, and HHistory for the recommender system.

In the next chapters, we will be exploiting those datasets in order to develop and improve models.

## 3 Machine Learning

### 3.1 Introduction

As an introductory material, this chapter introduces what Machine Learning is and how it works. This chapter will illustrate the notions through the lens of activity recognition in images. However, the foundations laid out here are not limited to this use case and are indeed more general. The explanations will alternate between the specific use cases, to ease the intuitive understanding, and the more abstract concepts in order to make the generality of the techniques clear.

At the end of this chapter we will illustrate the foundations laid here with a practical example. This will allow us to introduce the basics of model training, evaluation, data augmentation, and fine-tuning a pretrained model.

### 3.2 What is Machine Learning

#### ML as functions

Let  $\mathcal{X}$  be the set of images we are interested in, and  $\mathcal{Y}$  be the set of activities. In order to solve the problem of activity recognition, we wish to find a function  $f : \mathcal{X} \mapsto \mathcal{Y}$  that maps an image from  $\mathcal{X}$  to its correct activity.

While researchers have tried to write themselves algorithms that decipher natural images, applications such as "search images by text" in Google Photos, image colorization, natural image generation or captioning remained out of reach. Indeed, researchers found more success writing algorithms that *search* for  $f$  rather than finding  $f$  themselves. Image understanding is now dominated by *machine learning* approaches, that is, algorithms finding  $f$  from examples.

In fact,  $\mathcal{X}$  does not have to be images, and  $\mathcal{Y}$  class labels. As long as there is a relationship between  $\mathcal{X}$  and  $\mathcal{Y}$ , the former can be considered a random variable and the later a dependent variable. Machine Learning is about getting better at a task (with respect to a chosen metric) based on experience (or "examples", or "training samples").

#### Parameters and hyperparameters

In machine learning, we distinguish two kinds of values: parameters and hyper-parameters.

The values or decisions that are searched during learning for representing  $f$  are the *parameters*. For neural networks (defined later) they could be the individual neurons' synapses values, or *weights*.

Sometimes, there are configurations for those algorithms that cannot be learnt during training, like the number of neurons or layers, the maximum depth of a decision tree, which are the aforementioned *hyperparameters*.

## Train, validation, and test datasets

Let a dataset  $\mathcal{D}_t = (x_0, y_0), \dots, (x_N, y_N)$  where the  $x_i \in \mathcal{X}$  are the images and the  $y_i \in \mathcal{Y}$  are their class labels (most of the time provided by humans), usually an integer between 0 and  $|\mathcal{Y}| - 1$ . Machine Learning (ML) aims at finding a function  $f$  that, based on the examples provided in  $\mathcal{D}_t$ , can approximate a correct  $f$ . This is usually called *training or learning a model*.

Unfortunately, those functions found by ML techniques build complex relations hard to interpret by humans, sometimes known to be coarse approximations. Deep Learning solutions, and sometimes classical ML solutions are often considered black boxes. If we were able to understand it or not use approximations, we would have probably been able to write the function in the first place. It is often not possible to verify that the model solves the task in a meaningful way by checking its inner workings. Checking a ML model usually involves having a second set of annotated data  $\mathcal{D}_v$  and checking the performance (accuracy, for instance) on this set of data, called *validation set*. We have to rely on the validation set performance in order to trust that if the model solves them, then it has found an appropriate solution.

Finally, there is another catch. Some algorithms need to be configured with some *hyperparameters* before learning on the training set. However, throughout a project's life, the model and algorithms evolve during development. In order to make sure those hyperparameters and this exact algorithm generalize well and are not just conveniently working for this validation set, there is usually a third set, the *validation set* that is used in place of the test set during both development and hyperparameters selection. The test set  $\mathcal{D}_e$  is used as little as possible, only when trying to have a clear evaluation of the performance of the model. The test set, in order to be predictive of the model's final performance, should be representative of the real world data.

Every time the developer or the learning algorithm tunes parameters or hyperparameters, there is a risk that they become tailored to this dataset exactly. This is called overfitting: decreasing generalization and improving the performance on the training set. The test set exists to test whether the learning algorithm overfitted parameters on the training set.

The developer adjusts various hyperparameters (model size, learning rate, etc) in order to maximize the performance on the validation set. In order to make sure that those hyperparameters decisions generalize beyond the test set, we need a third set, the validation set. Unfortunately, as we use more and more the validation set to test various models, there is a risk of overfitting it as well.

Indeed, some datasets are used so commonly by researchers that there is a risk that the models and algorithms tested on those don't generalize as well and are accidentally tuned to those datasets specifically [130].

## 3.3 Neural Networks

### 3.3.1 Neural Networks in computer vision

While there are many algorithms that are able to learn from examples, computer vision is nowadays largely dominated by neural networks. They gained a lot of traction in 2012 when the ImageNet challenge [33] was won by a large margin with neural networks [88] rather than classical image processing techniques using classical machine learning (kNN, SVM, decision trees, etc [16]). The next year, in 2013, all competitors used neural networks. In 2015, they started gaining traction in the industry as well. I can remember, that year, when interning in Google, Sundar Pichai pitching Google Photos and explaining how big resnets [59] made it possible, less than a full year after their discovery.



### 3.3.2 Neurons

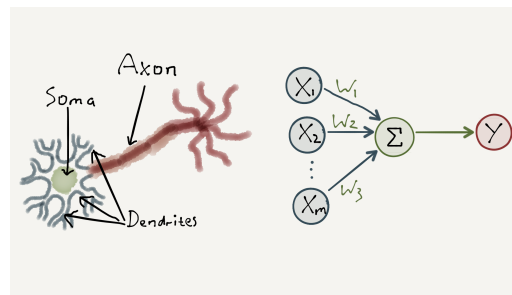


Figure 3.1: A (very simplified) biological neuron and an artificial neuron (Perceptron)

Before understanding neural networks, let's focus on a single artificial neuron. Artificial neurons are loosely inspired by biological neurons. They represent a neuron taking some input electrical signals, transmitting those by weaker or stronger dendrites (connections), summed up in the soma, firing through the axon if the total amount is above a certain threshold. Cf Figure 3.1

In computer science, this is approximated by a linear combination  $\mathbf{w}$  - a weighted sum - of the input  $\mathbf{x}$ , followed by a threshold  $b$  and an activation function  $\sigma$ . An artificial neuron is then  $\sigma(\mathbf{w}^T \mathbf{x} + b)$ , with  $\sigma$  being any non-linear function. *Training a neuron is finding the weights  $\mathbf{w}$ ,  $b$  that works best for solving a given problem.* This is sometimes still called a Perceptron, in reference to the original paper by "Rosenblatt [134].

While today we often chose the *Rectified Linear Unit* (ReLU)  $\sigma(x) = \max(0, x)$  for its good numerical and computational properties [116], researchers initially used a sigmoid, tanh or step function. Developing on activation functions is well beyond the scope of this manuscript, but it is still an active area of research with new propositions, despite none of them outperforming the performance / computational cost tradeoff of ReLUs.

### 3.3.3 From single neurons to neural networks

Once we have a neuron, there are two ways we can add more to build a *neural network*: either by parallel neurons (the *width*) or by stacking them in successive *layers* (the *depth*). The Universal Approximation Theorem [66] states that a neural network with 2 layers and enough neurons [9] can theoretically approximate any computable function. Unfortunately, possible does not equate practically feasible and we needed decades before turning them into useful algorithms. The deception lies in the "enough" part, which can be intractable for any practical purpose.

Fortunately, it has been shown that width can be traded with depth [112], allowing neural networks to work in successive abstractions [92, 174], thus not needing stellar amount of data and computation. Deepening neural networks was not possible immediately, and required new tools (among which the aforementioned ReLU), explaining why neural nets remained shallow for so long.

### 3.3.4 Training neural networks

After describing what neural networks *are*, it is necessary to describe how they *work*. Neural networks are stacks of non-linear layers, composed of many tunable weights  $\theta$  computing data transforms. How do we train them?

Neural Networks are built from differentiable components, making  $f_\theta(\cdot)$ , the function they compute, differentiable as well. In the simplest case, the training dataset provides an input  $x$  and expected

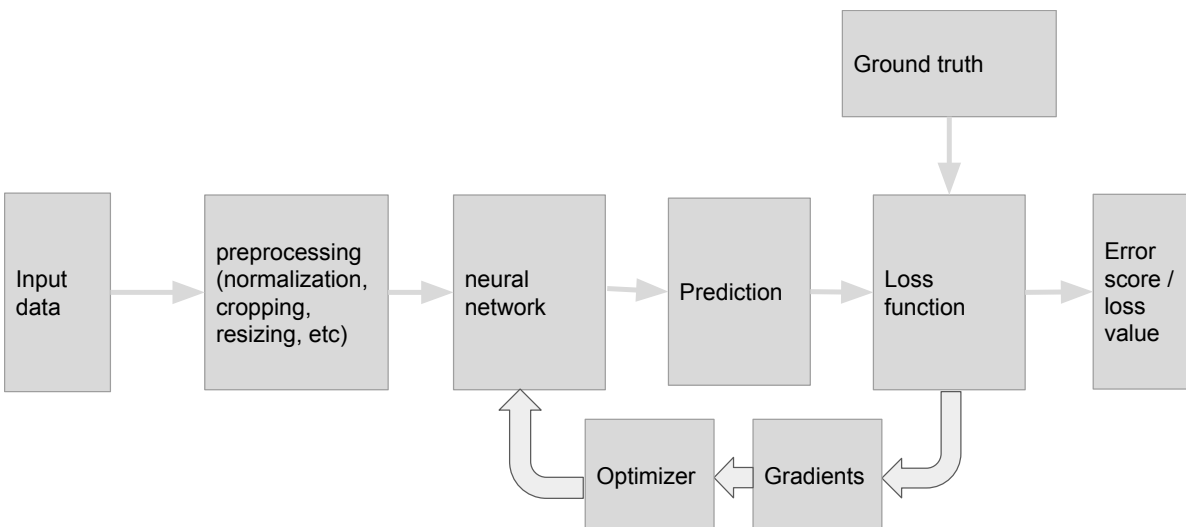


Figure 3.2: A neural network training loop

correct output  $y$ . We compute  $\hat{y} = f(x)$  the output of the network. Then, a differentiable predefined function  $\mathcal{L}(\hat{y}, y)$  computes the distance between the generated output and the expected correct output. By differentiating this distance wrt to the parameters  $\theta$ , we get the direction in which to move  $\theta$  to reduce the error.  $\theta$  is moved a bit (a step of size  $\alpha$ ) in this direction and the iterative process continues, and  $f$  learns to provide outputs closer and closer to the target output. This process is called *Gradient Descent*. In Deep Learning, however, *Stochastic Gradient Descent* (SGD) is used. The stochasticity comes from the fact that we compute the gradient on a random subset of the dataset (a *batch* or mini-batch), each iteration. This helps regularizing the model (thus providing greater generalization), and descending on the full dataset is often not practically doable anyway since they are usually too big to fit in memory along with the gradient information. Each iteration thus fundamentally computes

$$(3.1) \quad \theta := \theta - \alpha \nabla_{\theta} \mathcal{L}(f_{\theta}(x), y)$$

However, as we will later see, this update equation can be made more sophisticated in order to reduce the number of steps needed to converge and / or to improve generalization. These equations are called *optimizers*.

This process is summarized in Figure 3.2.

### 3.3.5 Data Augmentation

Last but not least, neural networks have the capacity to fit and memorize even big datasets. When the networks start to memorize the training set, the model overfits and its generalization abilities decrease. One way to fight against that overfitting is to add more training data, but this is often expensive or hard. Instead, one can create new artificial examples by modifying the training set. For images, this can be done by flipping the pictures, rotating them, changing the illumination, etc. This also injects domain knowledge into the algorithm about the existing invariances. This is called *data augmentation*.

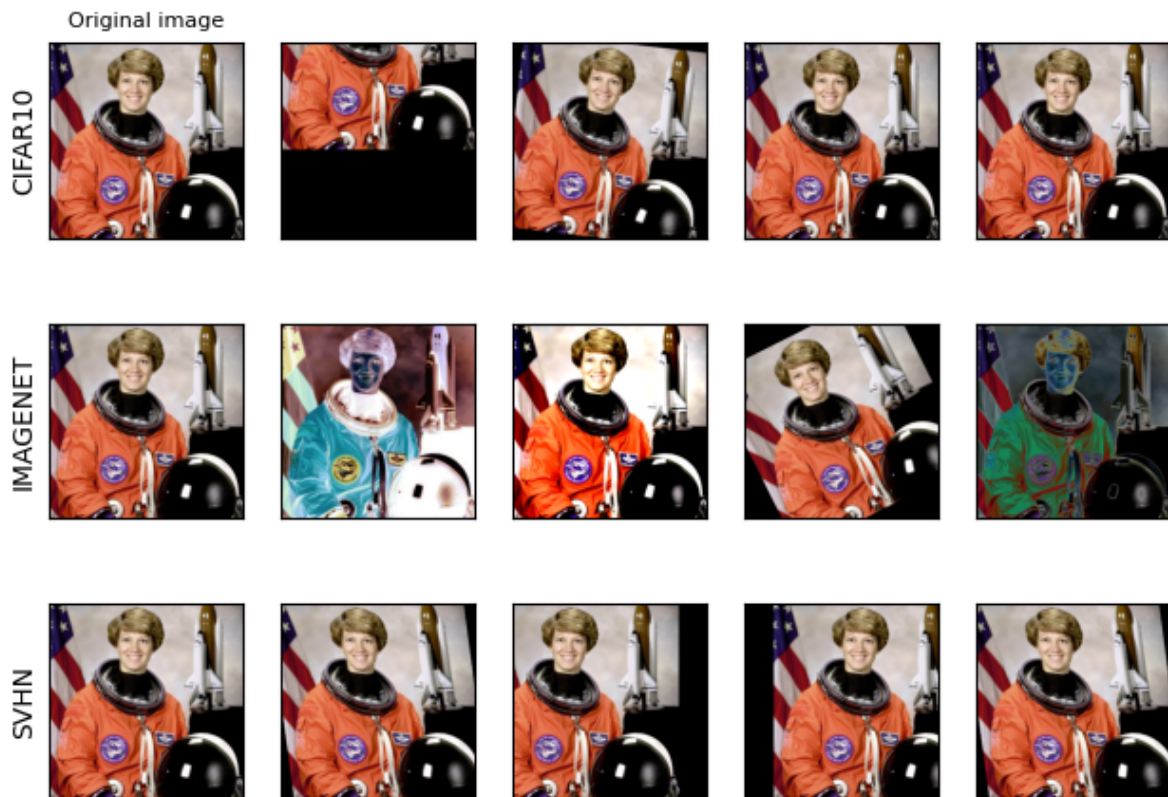


Figure 3.3: Example of data augmentation. The original image is transformed to artificially generate new training examples. In this case, AutoAugment is used ; it combines multiple transformations and its settings depend on the dataset and the task. Figure from torchvision’s documentation. Each row shows the set of augmentations used for those datasets on a sample image.

Tremendous progress has been made thanks to augmentation strategies [178, 29, 180, 36]. Examples from AutoAugment from Cubuk et al. [28] are shown in Figure 3.3.

For instance [36] randomly erases rectangle parts of the image in order to encourage robustness, by forcing models to rely on several and less salient features.

In this work, we will mainly use TrivialAugment [115]. It is a recent and very simple augmentation algorithm from which we can learn that despite a lot of complicated methods to automatically find augmentation policies (AutoAugment [28], RandAugment [29], ...), the simplest method performs comparably or better.

TrivialAugment defines an augmentation as ”a function mapping an image  $x$  and a discrete strength parameter  $m$  to an augmented image”. It uses a collection of predefined classic image transformations (color, contrast, rotation, ...). It randomly selects one of those operations per sample, and randomly samples its strength parameter  $m$ . ”The strength parameter is not used by all augmentations, but most use it to define how strongly to distort the image.”

### 3.3.6 Architecture

What those layers of neurons compute and how they are arranged defines an *architecture* or *topology*.

When two or more Perceptron layers are stacked, it is called a Multi-Layer Perceptron (MLP).

However, Perceptrons do not perform well on natural image data. They are too general and some-

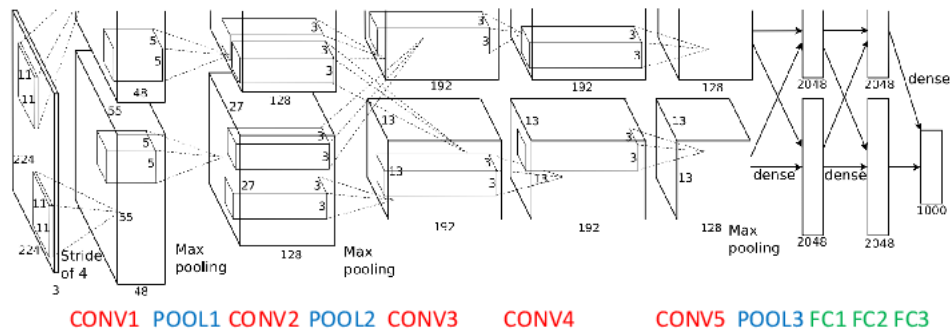


Figure 3.4: AlexNet architecture, built from convolutions (CONV), pooling operations (POOL), and linear layers (FC). Figure from Krizhevsky et al. [88]

how too powerful for computer vision: a Perceptron treats each and every input as a separate variable, but pixel values are not independent variables. First, they are spatially correlated as the world is compositional, and exhibits many invariances in translation, scale, and orientation.

Perceptrons have to learn to perform the same operations everywhere on the picture, and, in practice they don't. Replacing Perceptrons with convolutions with learnable weights gave birth to Convolutional Neural Networks (CNNs).

A convolutional layer is a Perceptron in disguise. It is just applied repeatedly and similarly on small spatial patches of the input. The size of that input patch is called *kernel size*. We compute many different convolutions on the same input - the *width* of the convolutional layer. Perceptrons compute multiple different linear combinations of the input in the same way -, each resulting spatial map is called a *feature map* or *channel*. Finally, the convolution kernel can jump some inputs, we call this *strided* convolutions, in order to downsample the input resolution.

Convolutions instead of Perceptrons build into the model what are called *inductive biases*: they perform local operations - addressing compositionality - and performs them the same way everywhere on the picture - addressing translational invariance. CNNs got fame in 2012 when they beat their competitors in the ImageNet classification challenge by a large margin and created the deep learning hype we know today [88].

The convnet that made deep learning attractive by winning ImageNet is **AlexNet** [88] (see Figure 3.4). It contains 5 conv layers, 2 max pooling layers, and 3 linear layers (also called Fully Connected layers). Max Pooling are meant to reduce the spatial size in order to reduce both memory and computation consumption, and increase the working region of each convolution. As the image is processed through the convnet, it produces layers of activations, that get abstracted in neural representations. Deep representations have a low spatial resolution but rich semantics [92, 174].

The design of convnets raises many questions: how many channels? what kernel shape? what stride? Should we pool? It is hard to answer those, so, when designing **GoogLeNet / Inception** [146] (Cf Figure 3.5), the Google team stacked many layers of complex building blocks. Each of those blocks is composed of parallel paths taking different design decisions. At train time, the net can learn to use each of those paths to its best.

However, those questions might not be that important after all. The **VGG** net [141] (Cf Figure 3.6) uses only 3x3 convolutions, and doubles the number of channels after each pooling operation. Its simplicity encouraged researchers to invest time into simple designs with better fundamental principles rather than complex models.

He et al. [59] observed that VGG nets could not be made very deep (no more than 20 layers). They hypothesized that the gradients quickly lose their supervision quality going back through the layers, failing to update the first layers. From this, they chose to design *residual* networks, where convs

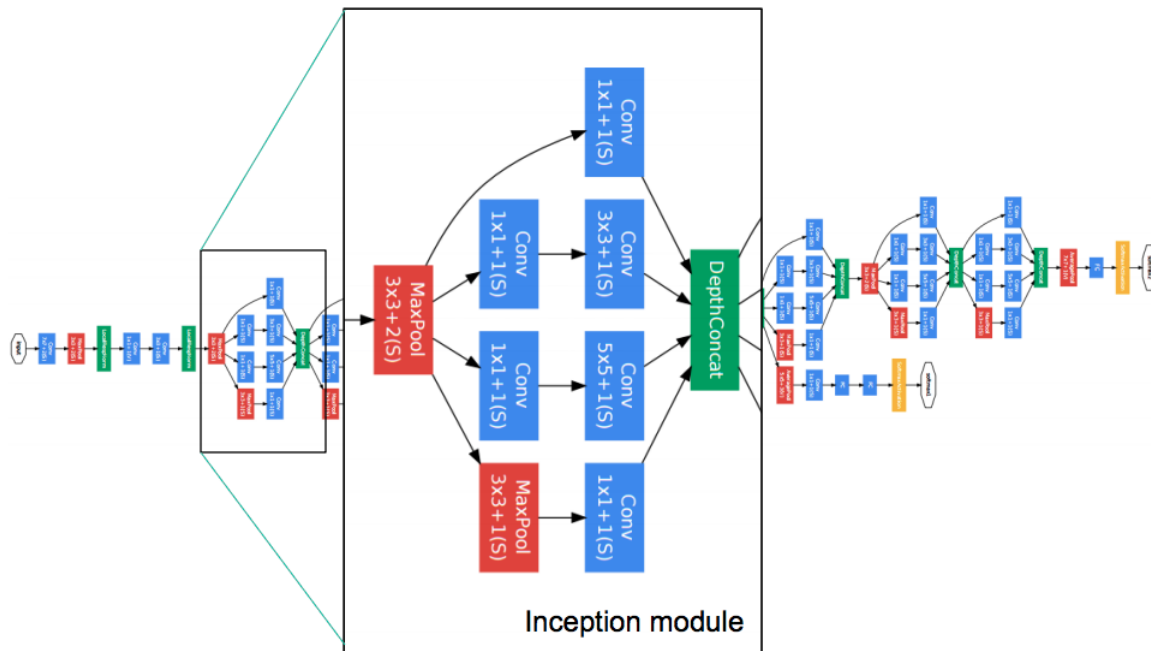


Figure 3.5: GoogLeNet / Inception-v1. It uses complex building blocks, aggregating different design decisions. Figure from <https://medium.com/@RaghavPrabhu/cnn-architectures-lexnet-alexnet-vgg-googlenet-and-resnet-7c81c017b848>

compute additive residual transformations. The gradients can then flow unchanged along the identity path and keep its informative quality. The **Residual Network** (ResNet, cf Figure 3.7) can go at least up to 1000 layers and still learn, despite reaching diminishing returns after 150 layers. The 50 layers variant is the most widely used because of its compute cost / performance tradeoff. ResNets were such an improvement that most of the following architectures integrated residual connections.

Note: Perceptrons are making their great comeback in image processing [153], mostly through Transformers (Perceptrons with an attention mechanism) [158, 40]. However, it does not invalidate what was said previously: while their performance scales better wrt big data quantity than State Of The Art (SotA) CNNs, they perform worse than CNNs on smaller training sets (ImageNet-1k being considered "small" in this context). Indeed, despite limiting convnets at scale, the convolutional inductive biases embody some knowledge about natural images. Transformers based architectures need some more data to discover those invariances and close the gap, but are able to surpass CNNs with even more data. There are works trying to suggest inductive biases to vision transformers that the network can un-learn if needed, in order to make them perform on par or better than convnets in lower data regimes [32, 31] and always benefit from their scalability.

### 3.3.7 Optimizers for Deep Learning

Developing optimizers is way beyond the scope of this work, but they cannot be kept unspoken of either. We briefly mention the main optimizers used for training deep networks.

#### Stochastic Gradient Descent

When gradient descent is done on a random subset of the training data, it becomes Stochastic Gradient Descent (SGD). SGD is akin to walking the steepest direction in a valley in order to reach a minimum.

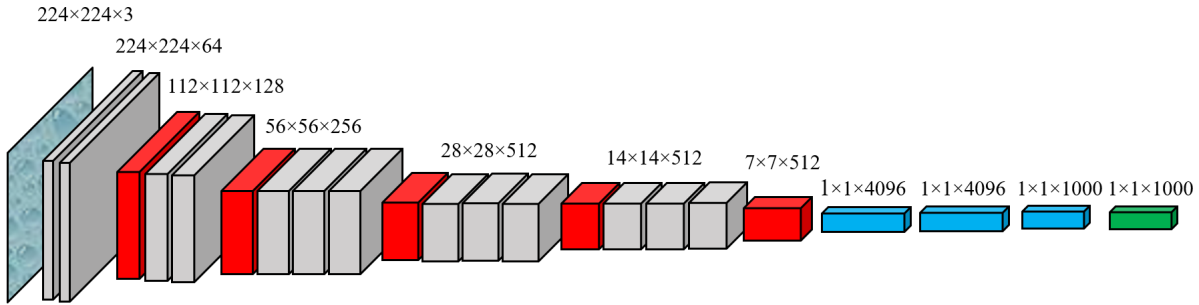


Figure 3.6: VGG network. Grey: 3x3 convolution layers; red: pooling layers; blue: linear (or 1x1 convs) layers; green: softmax. Each activation is described as Height  $\times$  Width  $\times$  Channels.

SGD is slow and prone to underfitting as it has no mechanism to escape local minima. For weights  $\theta$ , gradient of the loss  $\nabla J$ , and a positive hyperparameter learning rate  $\alpha$ :

$$(3.2) \quad \theta := \theta - \alpha \nabla J$$

### Stochastic Gradient Descent with Momentum

Instead, SGD is almost always used in its momentum variant. Taking the same analogy, Stochastic Gradient Descent with Momentum (SGDM) is like *rolling* in the steepest direction. While we roll, we accumulate velocity  $v$  that might help escape small local minima. Even if the valley is irregular, momentum helps escape those irregularities and reach the bottom. It adds a new positive momentum hyperparameter  $\beta$ .

$$(3.3) \quad \begin{aligned} v &:= v + \beta \nabla J \\ \theta &:= \theta - \alpha v \end{aligned}$$

Another notable variant is Nesterov momentum. Here, the gradient evaluation is performed after the momentum step of the parameters, contrarily to SGDM.

### Adam

Another family of optimizers, the *adaptive optimizers*, made popular by *Adaptive Moment Estimation (Adam)* is said to be less sensitive to hyperparameters and especially to the learning rate. It works by scaling the learning rate by the rolling variance of the weight in recent history. It introduces new hyperparameters,  $\beta_1$  and  $\beta_2$ , respectively defaulted to 0.9 and 0.999. It also considers  $t$ , the number of the current iteration, and  $\epsilon$ , a constant for numerical stability.

$$(3.4) \quad \begin{aligned} m &:= \beta_1 m + (1 - \beta_1) \nabla J \\ v &:= \beta_2 v + (1 - \beta_2) \nabla J \\ \hat{m} &:= m / (1 - \beta_1^t) \\ \hat{v} &:= v / (1 - \beta_2^t) \\ \theta &:= \theta - \alpha \hat{m} / \sqrt{\hat{v} + \epsilon} \end{aligned}$$

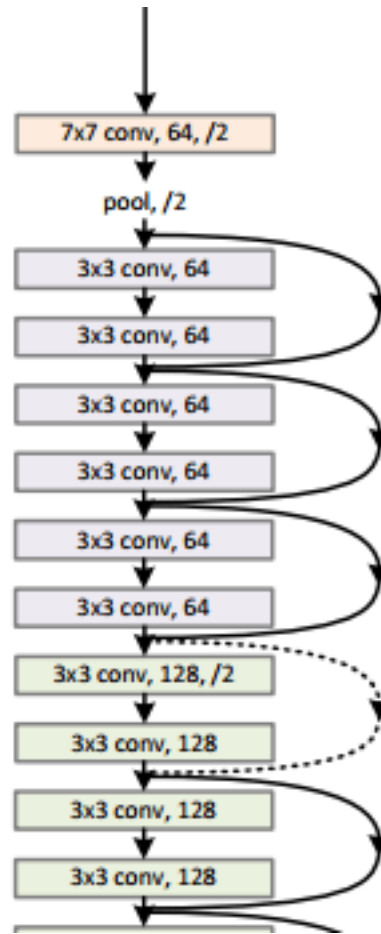


Figure 3.7: In ResNets, an identity path is added every two convolutions, so that the gradient can flow up to the first layers untouched. Figure from [59]

Despite its advantages in convergence speed and hyperparameters tuning, it has not fully took precedence over SGD as it converges to results only close to SGDM performance.

The literature explored many variants, including AdamW, RAdam, AdaMax, AMSGrad, AdaBelief, etc.

### 3.3.8 Loss functions

Any differentiable function that measures the error of the prediction can be used as a loss function. Choosing or developing loss functions tailored to the problem at hand is one fundamental part of designing a ML system for a task. We will quickly lay out the most commonly used loss functions.

#### Cross-Entropy

Cross-entropy measures the difference between two discrete probability distributions  $p_a$  and  $p_b$  for a random variable  $A$  with realizations  $a$ .

$$(3.5) \quad H(p_a, p_b) = - \sum_a p_a(x) \log p_b(x)$$

We can use it as a loss function. This implies that  $\hat{y} = f(x)$ , our neural network, must be interpreted as the conditional probability distribution  $p_f(\hat{y}|x)$  and the target outputs as a probability distribution  $p(y|x)$ . Minimizing the cross-entropy (that is, using it as a loss function) reaches its optimum when the two distributions are identical.

In order to train a classifier, we consider the special case with the target distribution  $p(y|x)$  defined as a categorical distribution over the possible classes. This distribution assigns a probability mass of 1 for the correct class, 0 otherwise. Minimizing the cross-entropy in this situation is strictly equivalent to maximizing the predicted probability of the target class, or minimizing the negative log likelihood of the target class. The later formulation is preferred.

A neural network does not produce calibrated probability distributions on its own, but unconstrained scalars. They are often called *logits*, as unnormalized log parameters of the categorical distribution, interpreted as the output of the logit function. Those logits  $z_i$  can be normalized into the categorical distribution parameters  $o$  with the softmax function.

$$(3.6) \quad o_i = \frac{\exp(z_i \tau)}{\sum_j \exp(z_j \tau)}$$

Where  $\tau$  is an optional *temperature* parameter that controls the sharpness / entropy of the distribution.

While not being totally accurate, some people call the cross-entropy loss the *softmax loss*.

### Mean-Squared Error

When trying to predict continuous values, one's default loss function choice is the L2 distance, or Mean-Square Error (MSE). It penalizes the prediction more as the difference to the target grows (Figure 3.8). When there is ambiguity, optimizing the MSE will result into predicting the mean value over the possible targets.

$$(3.7) \quad \mathcal{L}(y, \hat{y}) = (\hat{y}_i - y_i)^2$$

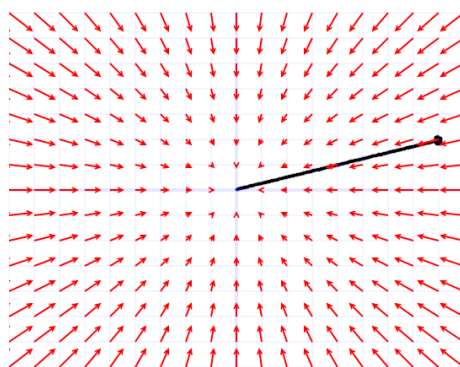


Figure 3.8: Gradient field of the L2 loss function over a  $\mathbb{R}^2$  plane. The black line shows the minimization trajectory from the black dot. We observe that this loss penalizes each variable proportionally to its value.



### L1 Loss

L1 might be used as well. L2 makes sure that predictions are not diverging too much from the target value; L1 rather encourages the number of exact predictions by penalizing equally any amount of divergence (Figure 3.9). In there is ambiguity, optimizing the L1 distance will result into predicting the median over the possible target values.

$$(3.8) \quad \mathcal{L}(y, \hat{y}) = |\hat{y}_i - y_i|$$

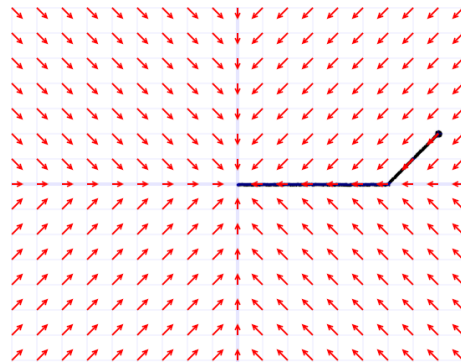


Figure 3.9: Gradient field of the L1 loss function over a  $\mathbb{R}^2$  plane. The black line shows the minimization trajectory from the black dot. We observe that this loss penalizes equally each variable, encouraging sparsity.

**Note: Numerical Considerations** When working with probabilities, computers might run into issues. Probabilities  $p_i$  are often multiplied together and can become small, to the point where there could be representation issues with standard IEEE754 32 bits floats. For this reason, when possible, we instead manipulate log probabilities, taking advantage of this property:

$$(3.9) \quad \prod_i p_i = \exp \sum_i \log p_i$$

### 3.3.9 Common datasets

Some datasets are commonly used in computer vision in order to develop new techniques, compare models, or leverage knowledge from big datasets.

#### MNIST

MNIST [91] is dataset of 28x28 greyscale images representing handwritten digits labeled with the ground truth digit. It contains 60k training pictures and 10k testing pictures. Classifying those digits is such a simple task that a SVM reaches 0.8% error. In deep learning, MNIST is used as a sanity check for debugging or to introduce novel ideas, for instance with artificially introduced label noise.

#### CIFAR-10/100

CIFAR-10 [87] contains 50k training 32x32 color images of 10 classes (airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck). There are 10k test images. This dataset is often used for developing

new strategies. Optimizers, architectures, augmentations etc are often tested and calibrated on CIFAR-10 before being battle-tested on ImageNet.

CIFAR-100 is similar but extended to 100 classes, 600 images each. It is not as commonly used as CIFAR-10.

### **ImageNet-1K / ILSVRC12**

ImageNet (sometimes called ILSVRC for Image Large-Scale Visual Recognition Challenge) [136, 33] is a large scale database of natural images crawled from the web, divided into 1k classes. It contains about 1M training pictures (1k per class) and 50k testing images.

Since 2012, ImageNet has been the gold standard dataset for evaluation and comparison of classifiers. It is also commonly used to extract knowledge from natural images in order to build feature extractors or backbones for assembling models together [85]. An expanded version of ImageNet, ImageNet-21k has been released. Models able to ingest lots of data are usually pretrained on it, before being tested on ImageNet-1k.

## **3.4 ⇒ *Practical Example: A classifier for HActions***

In this section we show how a standard image recognition problem can be solved with the tools laid out in this chapter. This is illustrated on activity recognition on the HActions dataset.

Activity recognition aims to identify what people are doing in still pictures or videos. At Hexaglobe, we are interested in activity tags in order to add metadata to our customers' content. They can be fed to search engines and recommender systems for more relevant results, as well as explicit categorization for a better user experience.

Most of the time, a still picture is enough to infer the activity: people sitting at a table with food are likely eating, people holding a microphone are most probably singing, etc.

Labeling frames with activities can be used in order to improve user browsing and experience. First, the extracted activities can be added as tags to videos so that they become searchable. Then, when a user searches for or is recommended an activity, a relevant thumbnail from the video can be selected instead of a random one. Finally, those activities can be annotated directly on the video player's timeline so that the users can skip to the part they are interested the most

### **3.4.1 The model**

For industrial settings such as this one, training speed is more important than accuracy, so the tradeoff should favor iterative development. For this reason, I chose a ResNet-18 as our network. It has less parameters (11M) than the popular ResNet-50 (24M) for the sacrifice of few accuracy points.

As the HActions dataset (see section 2.3.1) has at most 2400 examples per class, training a model from scratch is likely to be suboptimal. In those situations where the dataset is relatively small, it helps to use *transfer learning*. I chose to leverage a ResNet-18 from torchvision, pretrained on ImageNet-1k. The fact that both datasets are made of natural images suggests that the pretrained knowledge will be useful.

### **3.4.2 Transfer Learning**

Natural images share common properties: neighboring pixels are correlated, the distribution of colors and brightness is not uniform, and some shapes occur frequently. When training a model on a large

| Config                 | Test accuracy (%)     |
|------------------------|-----------------------|
| A: from random weights | 37.82                 |
| C1: A + random flip    | 43.87 (+10.05)        |
| B: pretrained weights  | 52.44 (+14.62)        |
| C2: B + random flip    | 55.67 (+3.23)         |
| D: C2 + random crop    | 55.67 (+0)            |
| E: D + TrivialAugment  | <b>58.19 (+ 2.52)</b> |

Table 3.1: Test accuracy on HActions according to various configurations

dataset such as ImageNet, the model would learn generic patterns, filters, shapes or objects that would be useful for other datasets or other tasks as well.

The parameters and architecture of the net is then kept untouched (or "frozen") but the last layer(s) that are trained on the new task. The frozen layers are used as a fixed feature extractor. This way, we only learn a small model from semantically rich features, instead of a bigger model from raw pixels. This allows to reuse natural images knowledge extracted from ImageNet. It helps preventing overfitting on meaningless spurious patterns in the case of small training sets, and to reuse knowledge for a different task. In addition to the performance advantage, this is much faster than training the whole thing as the features can be pre-computed only once.

After these last layer(s) have been retrained, it sometimes helps to *fine tune* all the parameters of the model, by iterating a little more on the dataset, this time training the whole network with a very small learning rate. This allows the model to extract a few more useful or specific features that were not learned during the pretraining, while trying not to overfit. When possible, it helps to know how the base network has been trained as regularizers can improve ImageNet accuracy but reduce transferability [85, 86].

### 3.4.3 Experiments

In order to demonstrate what we laid down so far we run a quick set of typical experiments with a ResNet-18. All experiments are conducted with SGDM, weight decay (regularizing the square of the weights) is set to  $1e-3$ , momentum is set to 0.9, batch size to 128, split across 4 GPUs. The learning rate is searched in  $\{0.3, 0.1, 0.01, 0.001, 0.0001\}$ . We train for 40 epochs. The learning rate ( $lr$ ) decays linearly from its initial value to 0 at the end of the training as [95] shows this to be a sensible choice in common scenarios. The initial data augmentation includes random horizontal flipping and the pictures are resized to  $128 \times 227$  which is unusual but keeps the 16:9 aspect ratio of the frames. We use a standard cross-entropy loss. A run takes about 1h.

We wish to verify and exemplify the benefits of a pretrained model and data augmentation, known to be among the top strategies to improve a natural images classifier.

The results, summarized in Table 3.1, show the power of pretraining and data augmentation, allowing performance boost with fast experiments.

**Config-A** trains from scratch. The best learning rate found is 0.1. It reaches an accuracy of 37.82%.

**Config-B** verifies that weights pretrained on ImageNet actually boost the accuracy. All the Batch-Norm layers are kept in inference mode during training and the running statistics are not updated. The learning rate for all the layers but the last is set to 0 during the first 4 epochs and divided by 100 for

the following iterations. The best learning rate found is 0.01. This significantly boosts the accuracy by 15%, reaching 52.44%.

**Config-C** Config B is found to perform significantly better than A. Config-C adds random horizontal flipping of input images. Adding this transformation brings the random initialization (C1) to 43.87%, and Config-B to 55.67% (C2).

**Config-D** adds inception-style random cropping to Config-C. It crops a random area from 80% to 100% of the image and resizes it to the input size.

**Config-E** adds TrivialAugment (Section 3.3.5). The accuracy moves up to 58.19%.

### 3.4.4 Results and interpretation

We trained a standard modern activity classifier from still pictures on HActions. Leveraging data augmentation and pretraining allowed us to jump from 38% to 58%. Although 58% may seem low, the dataset happens to be fairly small for classes that are loosely defined and somehow semantically hard to distinguish. For instance, if we were to classify dance move frames, the first and last few frames of each move, when the move starts and ends and is not clearly identifiable, this can legitimately be confused with no move at all. Upon manual inspection of the prediction on the test set, we indeed find that most of the errors can be attributed to ambiguity. The remaining errors can be largely attributed to a lack of data as they exhibit patterns not covered in training.

For the purpose of annotating frames and thumbnails in order to enrich user experience, those ambiguities are not a problem. Ambiguous frames can be ignored and we can focus on the frames that are not ambiguous. None of the use case we have for those labels need all frames to be annotated with big accuracy. Moreover, since there was no activity based user experience features before, any system that gives reliable predictions will provide more value than none.

We would certainly solve many of those uncertainties by analyzing videos and neighbouring frames instead of independent frames, but that would come at a prohibitive cost. Instead we will consider those samples from a label noise perspective, hoping to construct a dataset without confusing training samples and reject ambiguous samples at inference time.

## 3.5 Conclusion

We demonstrated how the building blocks explained in this chapter (Figure 3.2) can be assembled in order to train and utilize a learning algorithm. Pre-trained models for most used classification models are readily available and the data augmentation strategies are based on simple image manipulations. We show how one can leverage those for quick starting a project with little compute and data.

We trained a standard modern deep image classifier, leveraging battle tested techniques. Yet, the accuracy we obtained are usable for our use case, but somewhat below what can be expected from ImageNet capable models. We are facing problems that arise with "real world" usages: low data availability, labeling cost, ambiguity in samples and class semantics, etc. For instance, while ImageNet is a dataset of natural images, there are some peculiarities that might not be considered "real world", notably the data collection protocol that biased the data (keywords from search engines), or the fact that the class is an object centered and occupying the majority of the image surface.

In the next chapters, we apply the same standard classifier to the problem of face recognition, but quickly observe the same phenomenon: some data dependent problems are starting to arise, and

need mitigation: there are some specific types of label errors that need to be fixed, the face extraction pipeline sometimes extracts non-face images, and the data bears its own set of hard features (very narrow demographics, extreme facial expressions, varying makeup, etc). The next chapter investigates dealing with label noise so as to mitigate the most damaging aspect of our training set: erroneous training samples.

## 4 ⇒ *Contribution*: Label noise for face recognition

### 4.1 Introduction

Deep Learning systems have shown tremendous accuracy in image classification, at the cost of big, manually labeled, image datasets. Collecting such amounts of data can lead to labelling errors in the training set. Indexing multimedia content for retrieval, classification or recommendation can involve tagging or classification based on multiple criteria. In our case, we train face recognition systems for actors identification with a closed set of identities while being exposed to a significant number of distractors (actors unknown to our database). Face classifiers are known to be sensitive to label noise. We review recent works on how to manage noisy annotations when training deep learning classifiers, independently from our interest in face recognition.

Our client wishes to extract as much metadata as possible from their content. For the video content we host, identifying the actors is very valuable. Indeed, those videos have recurring actors that are worth identifying, among many unknown people that should be ignored. Users get to search for the content from the same actors, or similar actors. In order to extract these data, we build a face recognition system. We collect a dataset of face recognition with our celebrities. First we selected the 50 most popular celebrities on the platform and scrape pictures automatically from the internet, quickly verifying them manually resulting in 1k+ pictures per identity. In a second phase, we collect data for all the lesser known actors we wish to recognize. As those second-phase actors are less known, it is harder to find data for them, making automatic web scraping unreliable, and we shift to a strategy that is both more reliable and less time consuming. Human annotators are tasked to manually download between 10 and 100 pictures per identity, by descending order of popularity.

Bootstrapping and managing a large-scale dataset for face recognition requires either a lot of manual collection and labelling or scraping data from internet. Either way, the data is complex and the process is prone to error, and, when analyzing the data, we observe some recurrent error types:

- some people might be lookalikes and end up mixed up by human annotators or web resources (eg Vin Diesel and Dwayne Johnson);
- some might share a similar name and get scraped together, either by an automatic process or a human annotator (like two women named Alexa);
- some others might appear frequently together and collecting one would probably get pictures of the others as well (like Eric Judor and Ramzy Bédia). It might also be hard to collect pictures where each person of interest is alone, and we might also end up collecting people sharing the shot.

For this reason, it became important to detect and mitigate label noise. I wrote a survey in order to

learn the state of the art in detecting and handling label noise and its impact on image classification in general, keeping face recognition in mind.

This section is a contributed review paper published in ICME2020 [138].

## 4.2 Deep Learning Classification with Noisy Labels

Learning a deep classifier requires building a dataset. Datasets in media are often situation dependent, with different looking sets or landscape or exhibiting various morphologies, even non-human for face recognition, especially in fantasy and sci-fi contexts. It becomes tempting to use search engines to build a dataset or sort large image sets based on metadata and heuristics. Those methods are not perfect and label noise is introduced.

It is widely accepted that label noise has a negative impact on the accuracy of a trained classifier. Several works have started to pave the way towards noise-robust training. The proposed approaches range from detecting and eliminating noisy samples, to correcting labels or using noise-robust loss functions. Self-supervised, unsupervised and semi-supervised learning are also particularly relevant to this task since those techniques require few or no labels.

In this chapter, we propose a review of recent research on training classifiers on datasets with noisy labels. We will reduce our scope to the data-dependant approaches estimating or correcting the noise in the data. It is worth mentioning that some works aim to make learning robust by designing new loss functions [5, 183] without inspecting or correcting the noisy dataset in any way. Those approaches are beyond the scope of our study.

In the following sections, we first define label noise and summarize the different experimental setups used in the literature. We conclude by presenting recent techniques that rely on datasets with noisy labels. This work is inspired by [44], extending it to deep classifiers.

## 4.3 Overview of techniques

All the techniques presented will vary in different ways defined and presented briefly in this section. They can differ on the noise model they build upon, and whether they handle open or closed noise, presented in subsection 4.3.1, and based on [44]. Those noise models might need some additional human annotations in the dataset in order to be estimated, introduced in subsection 4.3.2. Subsection 4.3.3 will shortly enumerate approaches used for noisy samples detection, when needed. Once noisy samples have been detected, they can be mitigated differently, as outlined in subsection 4.3.4.

The various combinations taken by the approaches reviewed here are summed up in Table 4.1.

### 4.3.1 Problem definition

#### Models of label noise

In the datasets studied here, we posit that each sample  $x_i$  of a dataset has two labels: the true and unobservable label  $y_i$ , and the actual label observed in the dataset  $\hat{y}_i$ . We consider the label noisy whenever the observed label is different from the true label. We aim to learn a classifier  $f(x_i)$  that outputs the true labels  $y_i$  from the noisy labels  $\hat{y}_i$ . We denote a dataset  $D$  as  $D = \{(x_0, \hat{y}_0, y_0), \dots, (x_n, \hat{y}_n, y_n)\}$  and  $\hat{y}_i, y_i \in C$ , the set of classes. As presented in [44] the dataset label noise can be modeled in three way in descending order of generality.

1) The most general model is called **Noise Not At Random** (NNAR). It integrates the fact that corruption can depend on the actual sample content and actual label. It requires complex models to predict the corruption that can be expected. This model introduces  $P(y = c'|x)$  as the probability of

sample  $x$  having a true label  $y$  of class  $c$ , and a complex corruption model for  $\hat{y}$ , depending on both  $y$  and  $x$ ,  $P(\hat{y} = c|y = c', x)$ .

$$(4.1) \quad P(\hat{y} = c|x) = \sum_{c' \in C} P(\hat{y} = c|y = c', x)P(y = c'|x)$$

2) **Noise At Random** (NAR) assumes that label noise is independent from the sample content and occurs randomly for a given label. Label noise can be modeled by a confusion matrix  $\mathbf{C} \in \mathbb{R}^{|C| \times |C|}$  that maps each true label to labels observation probabilities. It implies that some classes  $y = c'$  may be more likely to be corrupted to  $\hat{y} = c$ . It also allows for the distribution of resulting noisy labels not to be uniform, for instance in naturally ambiguous classes. In other words, some pairs of labels may be more likely to be switched than others.

$$(4.2) \quad \begin{aligned} P(\hat{y} = c|x) &= P(\hat{y} = c) \\ &= \sum_{c' \in C} P(\hat{y} = c|y = c')P(y = c') \end{aligned}$$

3) The least general model, called **Noise Completely at Random** (NCAR), assumes that each erroneous label is equally likely and that the probability of an error is the same among all classes. For an error probability  $E$ , it corresponds to a confusion matrix with  $P(E = 0)$  on the diagonal and  $P(E = 1)/(|C| - 1)$  elsewhere. The probability of observing a label  $\hat{y}$  of class  $c$  among the set of all classes  $C$  is

$$(4.3) \quad \begin{aligned} P(\hat{y} = c|x) &= P(\hat{y} = c) \\ &= P(E = 0)P(y = c) \\ &\quad + P(E = 1)P(y \neq c) \end{aligned}$$

### Closed-set, open-set label noise

We distinguish **open-set** and **closed-set** noise. In closed-set noise, all the samples have a true label belonging to the classification taxonomy. For instance, a chair image is labeled "table" in a furniture dataset. In open-set noise this might not be the case, in the way a chair image labeled "chihuahua" in a dog races dataset has no correct label.

The second and third models discard open-set label noise by definition.

### 4.3.2 Types of additional human annotations

While training is done on a dataset with noisy labels, a cleaned test set is needed for evaluating the performance of the model. Those clean labels can be acquired from a more trusted yet limited source of data or via human correction.

We may also assume that a subset of the training set can be cleaned. A trivial approach in such cases, is to discard the noisy labels and perform semi-supervised learning using the validated ones and the rest of data as unlabeled. In noisy label training, one aims to exploit the noisy labels as well.

We can imagine a virtual metric, the complexity of annotation of a dataset, determined by factors such as the number of classes, the ambiguity between classes and the domain knowledge needed for labelling. A medical dataset could be hard to label even if it has only two classes while a more general



purpose dataset could have a hundred classes that can easily be discriminated if they are all different enough. When the dataset is simple, true label correction can be provided without prohibitive costs. When it is not, a reviewer can sometimes provide a boolean verification saying that the label is correct or not, which might be easier than recovering the true labels.

A dataset can then provide (1) no annotations, (2) corrected labels or (3) verified labels for a subset of its labels.

### 4.3.3 Detecting the noisy labels

When working on a per-sample decision basis, we often perform noisy samples detection. There are several sources of information to estimate the relevance of a sample to its observed label. In the analyzed papers, four families of methods can be identified. Most of them manipulate the classifier learned, either through its performance or data representation.

1) Deep features are extracted from the classifier during training. They are analyzed with Local Outlier Factor (LOF) [17] or a probabilistic variant (pLOF). Clean samples are supposed to be in majority and similar so that they are densely clustered. Outliers in feature space are supposed to be noisy samples.

2) The samples with a high training loss or low classification confidence are assumed to be noisy. It is assumed that the classifier does not overfit the training data and that noise is not learned.

3) Another neural network is learned to detect samples with noisy labels.

4) Deep features are extracted for each sample from the classifier. Some prototypes, representing each class, are learnt or extracted. The samples with features too dissimilar from the prototypes are considered noisy.

### 4.3.4 Strategies with noisy labels

Techniques mitigating noise can be divided in 4 categories. One is based on the NAR model, using statistical methods depending only on the observed labels. The three other methods use NNAR and need a per sample noise evaluation. These techniques are summed up in the Strategy column of Table 4.1.

1) One can **re-weight the predictions** of the model with a confusion matrix to reflect the uncertainty of each observed label. This is inherently a closed-set technique as the probability mass of the confusion matrix has to be divided among all labels.

2) Instead of re-weighting the predictions, we can **re-weight their importance** in the learning process based on the likelihood of a sample being noisy. Attributing a zero weight to noisy samples is a way to deal with open-set noise.

3) Supposedly erroneous samples can be **unlabeled**. The sample is kept and used differently, through semi-supervised or unsupervised techniques.

4) Finally, we can try to **fix the label** of erroneous samples and train in a classical supervised way.

## 4.4 Experimental Setups

While **CIFAR-10** [87] (section 3.3.9) remains one of the most used datasets in image classification due to its small image sizes, relatively small dataset size, and not-too-easy taxonomy, it has clean labels that are unsuitable for our works. CIFAR-10 contains 60000 images evenly distributed among 10 classes such as "bird", "truck", "plane" or "automobile". It is still largely employed in noisy label training with artificial random label flipping, in a controlled manner to serve whichever method is shown. However, synthetically corrupting labels fails to exhibit the natural difficulties of noisy labels due to ambiguous,

undecidable, or out of domain samples. MNIST [91] can be employed under the same protocols, with a reduced size of classes of handwritten digits, each composed of 1000 images.

**Clothing1M** [171] contains 14 classes of clothes for 1 million images. The images, fetched from the web, contain approximately 40% of erroneous labels. The training set contains 50k images with 25k manually corrected labels, the validation set has 14k images and the test set contains 10k samples. This scenario fits our low annotation complexity situation where labels can be corrected without too much difficulty, but the size of the dataset makes a full verification prohibitive.

**Food101-N** [93] has 101 classes of food pictures for 310k images fetched from the internet. About 80% of the labels are correct and 55k labels have a human provided verification tag in the training set. This dataset rather describes the high annotation complexity scenario where the labels are too numerous and semantically close for an untrained human annotator to correct them. However, verifying a subset of them is feasible.

Finally, **WebVision** [96] was scraped from Google and Flickr in a big dataset mimicking ILSVRC-2012 [33] (1k classes, 1.2M training samples), but twice as big. It contains the same categories, and images were downloaded from text search. Web metadata such as caption, tags and description were kept but the training set is left completely uncurated. A cleaned test set of 50k images is provided. WebVision-v2 extends to 5k classes and 16M training images.

When working on image data, all the papers used classical modern architectures such ResNet [59], Inception [146] or VGG [141].

## 4.5 Approaches

### 4.5.1 Prediction re-weighting

Given a softmax classifier  $f(x_i)$  for a sample  $x_i$ , prediction re-weighting mostly implies estimating the confusion matrix  $\mathbf{C}$  in order to learn  $\mathbf{C}^T f(x_i)$  in a supervised fashion with the noisy labels. Doing so will propagate the labels' confusion in the supervising signal to integrate the uncertainty about label errors. The main difference between the approaches lies in the way  $\mathbf{C}$  is estimated.

In **Noisy Label Neural Networks (NLNN)** [10], noisy labels are assumed to come from a real distribution observed through a noisy channel. The algorithm performs an iterative Expectation Maximization algorithm. In the Expectation step, correct labels  $y_i$  are guessed through  $\mathbf{C}^T f(x_i)$  while in the Maximization step,  $\mathbf{C}$  is estimated from the confusion matrix between guessed labels  $\tilde{y}_i$  and dataset labels  $\hat{y}_i$ . Finally,  $f(x_i)$  is trained on guessed labels  $\tilde{y}_i$ . The process is repeated until convergence.

Taking a more direct approach, **(Xiao et al, 2015)** [171] directly approaches  $\mathbf{C}$  by manually correcting the labels of a subset of the training set. Then, a secondary neural network  $g(x_i)$  is defined, giving to each sample a probability  $P(z_{1,i}, z_{2,i}, z_{3,i}|x_i)$  of being either ( $z_1$ ) noise free, that is  $\hat{y}_i = y_i$ , ( $z_2$ ) victim of completely random noise (NCAR), ie  $P(\hat{y}_i|y_i) = (U - I)y_i$  such that the matrix  $U$  is uniform and all rows of  $U - I$  sums to 1, or ( $z_3$ ) confusing label noise (NAR),  $P(\hat{y}_i|y_i) = \mathbf{C}^T \hat{y}_i$ . Finally,  $f(x_i)$  is trained on the noisy labels so as to minimize  $L_{CE}(z_{1i}f(x_i) + z_{2i}(U - I)f(x_i) + z_{3i}\mathbf{C}^T f(x_i), \hat{y}_i)$  with  $L_{CE}$  the cross entropy loss function.

**(Hendrycks et al, 2018)** [62] first train a model on the dataset with noisy labels. This model is then tested on a corrected subset and its predictions errors are used to build the confusion matrix  $\mathbf{C}$ . Finally  $f(x_i)$  is trained on the corrected subset and  $\mathbf{C}^T f(x_i)$  is trained on the noisy subset.

### 4.5.2 Sample importance re-weighting

For a softmax classifier  $f(x_i)$  trained with a loss function such as cross-entropy  $L_{CE}$ , sample importance re-weighting consists in finding a sample weight  $\alpha_i$  and minimizing  $\alpha_i L_{CE}(f(x_i), \hat{y}_i)$ . For a value  $\alpha_i$

close to 0, the example has almost no impact on training.  $\alpha_i$  values larger than 1 emphasize examples. If  $\alpha_i$  is exactly 0, then it is analogous to removing the sample from the dataset.

**Co-mining** [167] investigates face recognition where correcting labels is unapproachable for a large number of identities, and most likely a situation of open-set noise. Two neural nets  $f_1$  and  $f_2$  are given the same batch. For each net, the losses  $l_{1i} = L(f_1(x_i), \hat{y}_i)$  and  $l_{2i} = L(f_2(x_i), \hat{y}_i)$  are computed for each sample and sorted. The samples with the highest loss for both nets are considered noisy and are ignored. The samples  $s_{1,i}$  and  $s_{2,i}$  that have been kept by  $f_1$  and  $f_2$  are considered clean and informative: both nets agreed. Finally, the samples kept by only one net are considered valuable to the other. Backpropagation is then applied, with clean faces weighted to have more impact, valuable faces swapped in order to learn  $f_1$  with  $s_{2,i}$  and  $f_2$  with  $s_{1,i}$ , and low quality samples are discarded.

**CurriculumNet** [53] trains a model on the whole dataset. The deep features of each sample are extracted, and from the Euclidean distance between features vectors, a matrix is built. Densities are estimated, 3 clusters per class are found with k-means, and ordered from the most to least populated. Those three clusters are used for training a classifier with a curriculum, starting from the first with weight 1, then the second and third, both weighted 0.5.

**Iterative learning** [168] chooses to operate iteratively rather than in two phases like Curriculum-Net. The deep representations are analyzed throughout the training with a probabilistic variant of Local Outlier Factor [17] for estimating the densities. Local outliers are deemed noisy. The unclean samples importance is reduced according to their probability of being noisy. A contrastive loss working on pairs of images is added to the cross entropy. It minimizes the euclidean distance between the representation of samples considered correct and of the same class, and maximizes the Euclidean distance between clean samples of different class or clean and unclean samples. The whole process is repeated until model convergence.

We can also employ meta-learning by framing the choice of the  $\alpha_i$  as values that will yield a model better at classifying unseen examples after a gradient step. (Ren et al, 2018) [131] performs a meta gradient step on  $L = \alpha_i L_{CE}(f(x_i), \hat{y}_i)$  then evaluate the new model on a clean set. The clean loss is backpropagated back through  $L$ , for which the gradient  $\eta$  gives the contribution of each sample to the performance of the model on the clean set after the meta step. By setting  $\alpha_i = \max(0, \eta_i)$ , the samples that impacted the model negatively are discarded, and the positive samples get an importance proportional to the improvement they bring.

**CleanNet** [93] learns what it means for a sample to come from a given class distribution, utilizing a correct / incorrect tag provided by human annotators. A pretrained model extracts deep features of the whole dataset. Then, they run a per-class K-Means, and find the images with features closest to the centroids as a set  $v_c$  of reference images for that class  $c$ . A deep model  $g(v_c)$  encodes the set into a single prototype. A third deep model  $h(x_i)$  encodes the query image  $x_i$  in a prototype. We learn to maximize  $w_{ci} = \cos(g(v_c), h(x_i))$  if  $x_i$  has a correct class  $c$ , and to minimize it otherwise. This relevance score is used to weigh the importance of that sample when training a classifier with  $\max(0, w_{\hat{y}_i}) L_{CE}(f(x_i), \hat{y}_i)$ .

Instead of getting a consistent wrong information from an erroneous label, **NLNL** [80] (not to be confused with NLNN) samples a label  $\tilde{y}_i \neq \hat{y}_i$  and uses negative learning, a negative cross-entropy version that minimizes the probability of  $\tilde{y}_i$  for  $x_i$ . As the number of classes grows, the more likely the sampled label  $\tilde{y}_i$  is indeed different of  $y_i$  and noise is mitigated, despite being less informative. Then only samples with a label confidence above  $1/|C|$  are kept and used negatively in a second phase called Selective Negative Learning (SelNL). Finally, examples with confidence over a high threshold (0.5 in the paper) are used for positive fine-tuning with a classical cross entropy and their label  $\hat{y}_i$ .

### 4.5.3 Unlabeling

**Iterative Noise Filtering** [117]: A model is trained on the noisy dataset. An exponential moving average estimate of this model is then used to analyze the dataset. Samples classified correctly are considered clean, while the label is removed for those classified incorrectly. The model is further trained with both a supervised and unsupervised objective for labeled and unlabeled samples. The samples with labels are used with a cross entropy loss. For each unlabeled sample, we maximize  $\max_c f(x_i)_c$  in order to reinforce the model’s prediction, while maximizing the entropy of the predictions over the whole batch to avoid degenerate solutions. After each epoch, the dataset’s labels are evaluated again according to the average model.

### 4.5.4 Label fixing

A few methods already listed above try to fix the labels as part of their approach. While listed as a sample re-weighting method, **NLNL** [80] also employs a sort of label fixing procedure by using the negative labels. Similarly, **NLNN** [10] attempts to fix the labels while estimating the confusion matrix. Finally, **Iterative Noise Filtering** [117], assumes that the class with the highest prediction for the unlabeled examples is correct.

**Deep Self-Learning** [55] learns an initial net on noisy labels. Then, deep features are extracted for a subset of the dataset. A density estimation is made for each class and the most representative prototypes are chosen for each cluster. The similarity of all samples to each set of prototypes is computed to re-estimate correct labels  $\tilde{y}_i$ . The model training continues with a double loss balancing learning from the original label or the corrected one  $L = \lambda L_{\text{CE}}(f(x_i), \hat{y}_i) + (1 - \lambda)L_{\text{CE}}(f(x_i), \tilde{y}_i)$  with hyper-parameter  $\lambda \in [0, 1]$ . We iterate between label correction and weighted training until convergence. Note that contrarily to sample weighting techniques that weigh the contribution of each sample in the loss, all samples have an equal importance, but we place a cursor as a hyper-parameter to balance between contribution from the noisy labels and corrected labels.

## 4.6 Discussion

Those approaches cover a wide variety of use cases, depending on the dataset: whether it has verified or corrected labels or not, and the estimated proportion of noisy labels. They all have different robustness properties: some might perform well in low noise ratio but deteriorate quickly while others might have a slightly lower optimal accuracy but not deteriorate as much with high noise ratio.

Re-weighting predictions performs better on flipped labels rather than uniform noise as shown in the experiments on CIFAR-10 in **Hendrycks et al, 2018** [62]. As noise becomes close to a uniform noise, the entropy of the confusion matrix **C** increases, labels provide more diffused information, and prediction re-weighting is less informative. CIFAR-10 being limited to 10 classes, **NLNN** [10] is shown to scale with a greater number of classes on TIMIT.

Noisy samples re-weighting scales well: CurriculumNet [53] scales in number of samples and classes as the experiments on WebVision shows, Co-Mining [167] is able to scale to face recognition datasets and open-set noise at the expense of training two models, CleanNet generalizes its noisy samples detection by manually verifying a few classes.

However, **NLNL** [80] may not scale as the number of classes grows: despite having negative labels that are less likely to be wrong, they also become less informative.

We can expect unlabeling techniques to grow as the semi-supervised and unsupervised methods gets better, since any of those can be used once a sample had its label removed. One could envision

utilizing algorithms such as MixMatch [14] or Unsupervised Data Augmentation [172] on unlabeled samples.

Similarly, the label fixing strategies could benefit from unsupervised representation learning to learn prototypes that makes it easier to discriminate hard samples and incorrect samples. Deep self-learning [55] is shown to scale on Clothing1M and Food-101N. It would be expected that those approaches become less accurate as the number of classes grows or as the classes get more ambiguous. Some prior knowledge or assumptions about the classes could be used explicitly by the model. Iterative Noise Filtering [117] in its entropy loss assumes that all the classes are balanced in the dataset and in each batch.

## 4.7 Conclusion

We explored the situation where a deep classifier has to be learnt on data with label noise, that is, containing erroneous target labels. We explored the literature and showed that the approaches can be sorted into four main categories: reweighting predictions using a noise model, reweighting the importance of training samples based on their assessed probability of having a wrong label, unlabel the suspicious samples and use them with unsupervised training, or fixing the suspicious labels with new guesses.

Training a deep classifier using a noisy labeled dataset is not a single problem but a family of problems, instantiated by the data itself, noise properties, and provided manual annotations if any. As types of problems and solutions will reveal themselves to the academic and industrial deep learning practitioners, deciding on a single metric, a more thorough and standardized set of tests might be needed. This way, it will be easier to answer questions about the use of domain knowledge, generality, tradeoffs, strengths and weaknesses, of noisy labels training techniques depending on the use-case.

In the face recognition system that we are building, label noise has varying causes: persons with similar names; confusion with lookalikes; related persons that appear together; erroneous faces detected on signs or posters in the picture; errors from the face detector that are not faces; and random noise. All those situations represent label noise with different characteristics and properties that must be handled with those algorithms. We believe those issues are more general than this scenario and find an echo in the broader multimedia tagging and indexing domain.

From this study we mainly keep that samples with label errors produce higher losses than correctly labeled ones. The next chapter will explore how to leverage this in order to manually curate our training set. Furthermore, as we will show, a face recognition dataset contains pictures of many identities, but can also include pictures that do not belong to any of the known identities. This situation is reminiscent of the open-set label error (when the correct label does not belong to any of the known label) or unlabeled samples.

|                                 | Strategy                                  |  |                                     |                                  | Annotations   |                  |                 | Detection            |                            |       |                          | Datasets                              |                                |                                  |                           |
|---------------------------------|---|--|-------------------------------------|----------------------------------|---------------|------------------|-----------------|----------------------|----------------------------|-------|--------------------------|---------------------------------------|--------------------------------|----------------------------------|---------------------------|
|                                 | Reweight predictions<br>(NAR, Closed-set) | Reweight or remove samples<br>(NNAR, Open-set) | Unlabel samples<br>(NNAR, Open-set) | Fix labels<br>(NNAR, Closed-set) | No correction | Corrected labels | Verified labels | Local Outlier Factor | High loss / Low confidence | Model | Similarity to prototypes | CIFAR-10 / MNIST<br>(Synthetic noise) | Food-101N<br>(Verified labels) | Clothing1M<br>(Corrected labels) | WebVision<br>(Raw labels) |
| NLNL [80]                       |   | ✓  |                                     | negative labels                  | ✓             |                  |                 |                      | ✓                          |       |                          | ✓                                     |                                |                                  |                           |
| Iterative Noise Filtering [117] |   |  | without entropy loss                | with entropy loss                | ✓             |                  |                 |                      | ✓                          |       |                          | ✓                                     |                                |                                  |                           |
| Ren et al, 2018 [131]           |   | ✓  |                                     |                                  | ✓             |                  |                 |                      |                            | ✓     |                          | ✓                                     |                                |                                  |                           |
| Iterative learning [168]        |   | ✓  |                                     |                                  | ✓             |                  |                 | ✓                    |                            |       |                          | ✓                                     |                                |                                  |                           |
| NLNN [10]                       | ✓   |  |                                     | ✓                                | ✓             |                  |                 |                      |                            |       |                          | ✓                                     | & TIMIT                        |                                  |                           |
| Hendrycks et al, 2018 [62]      | ✓   |  |                                     |                                  |               | ✓                |                 |                      |                            |       |                          | ✓                                     | & NLP                          |                                  |                           |
| Deep Self-Learning [55]         |   |  |                                     | ✓                                | ✓             |                  |                 |                      |                            | ✓     |                          | ✓                                     | ✓                              |                                  |                           |
| CleanNet [93]                   |   | ✓  |                                     |                                  |               |                  | ✓               |                      |                            | ✓     | ✓                        | ✓                                     | ✓                              |                                  |                           |
| Xiao et al, 2015 [171]          | ✓   |  |                                     |                                  |               | ✓                |                 |                      |                            | ✓     |                          |                                       | ✓                              |                                  |                           |
| CurriculumNet [53]              |   | ✓  |                                     |                                  | ✓             |                  |                 |                      | ✓                          |       |                          |                                       |                                |                                  | ✓                         |
| Co-Mining [167]                 |   | ✓  |                                     |                                  | ✓             |                  |                 |                      | ✓                          |       |                          | face rec                              |                                |                                  |                           |

Table 4.1: Approaches according to annotations in the dataset. Notes: TIMIT is a speech to text dataset, "NLP" is a set of natural language processing datasets (Twitter, IMDB and Stanford Sentiment Treebank), "face rec" denotes classical face recognition datasets (LFW, CALFW, AgeDB, CFP)

## 5 Face Recognition

### 5.1 Introduction

As previously mentioned, a face recognition system embedded on the customer’s platform would extract metadata that could be interesting for our recommendation engine and valuable for the user experience.

We will first highlight how our use case is different from the general case, then present how face recognition is usually tackled, and finally explore our system currently in production. We will emphasize our contributions: a new metric-learning loss function, the threshold-softmax loss, and a study of several methods for exploiting unlabeled faces during training.

In our industrial setting, we wish to identify some people of interest (“VIPs”) among many unknown people in videos with unconstrained facial pose, illumination, expression and occlusion. The set of identities of interest is known ahead of time but a lot of the input pictures, if not most, are unknown people that must be rejected by the system. This particular setting makes this an instance of a subject-dependent open-set protocol, which we observe to be an understudied case, not even considered in Wang and Deng [165] (Figure 5.1).

We believe this setting is particularly common under some industrial settings in which we are interested in some people, like celebrities, for which we can acquire datasets if needed, among many unknown test-time distractors. One such example would be reidentifying famous YouTubers in fan compilations, clips or video reuses. Another would be to reidentify famous actors in movies among extras.

We call *distractors* faces whose identity is unknown to the system. It is expected that the system rejects those faces and is aware that they are unknown. The presence or absence of distractors is what differentiates open-set and close-set settings. Depending on the data and task, the ratio of distractors might be largely greater or lesser than that of VIPs. For us, the distractors dominate.

Hexaglobe’s Face Recognition system has some specificities compared to the algorithms laid in the literature:

1. Contrarily to most, this work is focused on identifying a known set of identities among distractors. Most works instead deal with verifying if a query face is the same person as a key face, for instance, verifying if the person at the customs is the same as the one on the passport being presented.
2. Most works on classifiers focus on having a correct best guess, but this system does not have to make a prediction for every input. The model rejecting some inputs because of uncertainty is a better outcome than a failed prediction. Knowing that we don’t know is crucial here.
3. Our work is deployed at scale where it analyzes hundreds of user uploaded videos per day. Those videos present real world faces in the wild, with occlusions, variable lighting conditions, image

resolution, quality and scale, not properly aligned, sometimes with extreme facial poses, making this data more challenging than most used datasets currently used in publications [54].

However, our setting allows some relaxations :

1. As a public video platform using the system for tagging videos with identities for search improvements, we are only interested in tagging the most popular people. These people can be decided ahead of time and our problem becomes a semi-open set face recognition problem: the set of identities to recognize is known ahead of time but there are unknown distractors to reject.
2. All errors are not equal. It is more harmful to add a wrong tag than to miss one. In our case, precision is more important than recall. This makes it possible, even needed, to let the model express its lack of confidence and not act on uncertain predictions.
3. As we are interested in overall video tagging and not per frame tagging, per frame errors are not harmful if they can be smoothed out.

Besides, considering the volume of existing videos (approx 8M) and the number of new uploads per day, processing a video should not take more than 5 minutes. In order to guarantee this processing time, we extract 300 frames evenly spaced throughout the video.

Finally, since our system is meant to have a quickly growing set of persons of interest, we favor fast iterations, both in model training time and time needed to add a subject to the set of known persons.

## 5.2 Standard systems

Face recognition (FR) systems based on deep learning have been under heavy research these past years. As outlined in Wang and Deng [165], FR systems can be characterized by two major features: whether we expect the train and test sets to contain the same identities or not (subject-dependent vs subject-independent protocol), and the evaluation task: verification (whether a picture matches the identity of another), closed-set identification (all input test identities are known to the system) and open-set identification (some test inputs do not belong to any known identity and must be rejected).

We observe that a lot of work went into the subject-independent closed-set protocol thanks to MegaFace [79]. Contrarily, the subject-dependent protocol has mostly been considered in the closed-set evaluation and deemed to be solved with a simple image classifier, not receiving much attention. Figure 5.1 does not even consider other settings.

### 5.2.1 Open-set/closed-set and subject-dependent/subject-independent recognition

As explicated in Figure 5.1, face recognition systems, when focusing on the task of open-set verification, have to decide whether two pictures of people unseen during training time are the same person or not. In order to generalize to unseen identities, the systems are trained under a similarity learning objective to produce dense semantic face representations that should be similar for same identities and dissimilar for different identities under cosine or Euclidean distance (for the general case). The Triplet Loss [65, 63] explicitly enforces those properties with a negative and positive pair relative to an anchor face. While it marked the beginning of scalable face recognition systems, the long training time drove the community to more data-efficient and time-efficient techniques. Present state-of-the-art approaches such as SphereFace [101], CosFace [164], ArcFace [34], AdaCos [182] mostly build on classification losses such as Cross-Entropy with margin and representation constraints such as normalizing L2 norm (Figure 5.2). Current datasets for those approaches have about 10k of identities in order to learn good



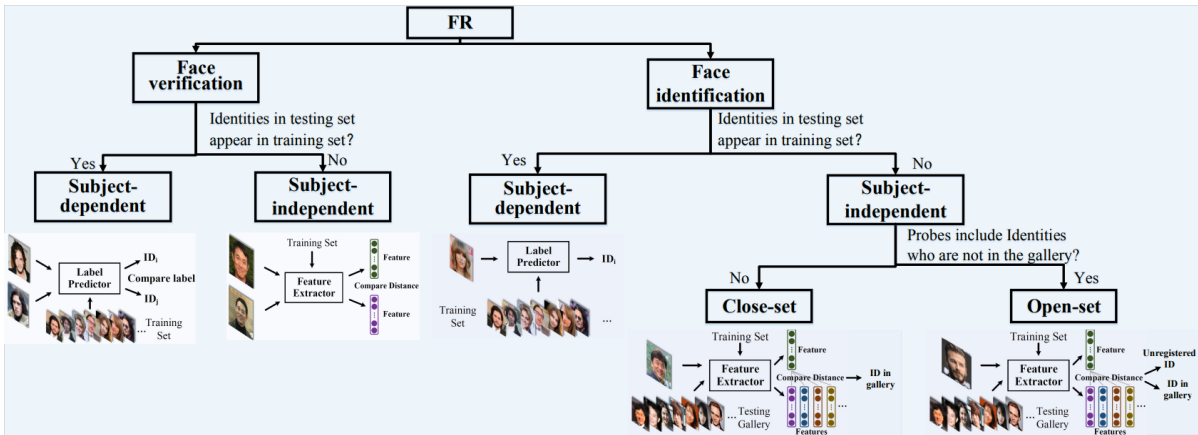


Figure 5.1: Figure 17 from Wang and Deng [165]. The comparison of different training protocol and evaluation tasks in FR. In terms of training protocol, FR can be classified into subject-dependent or subject-independent settings according to whether testing identities appear in training set. In terms of testing tasks, FR can be classified into face verification, close-set face identification, open-set face identification.

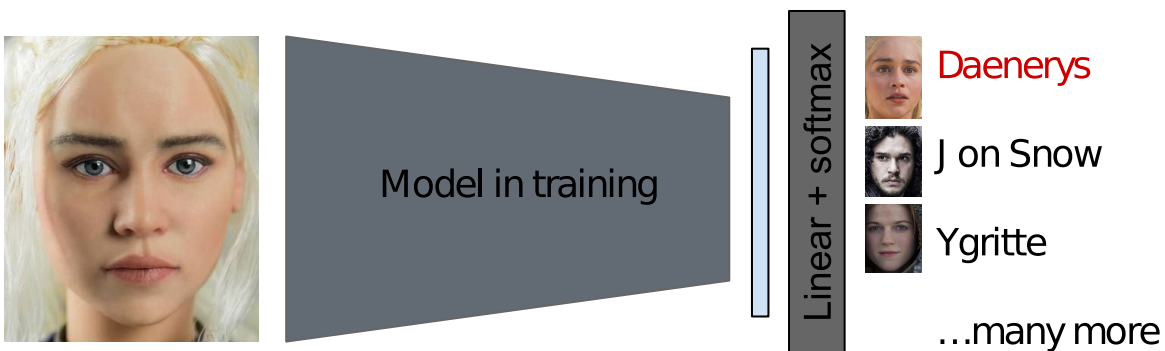


Figure 5.2: Training of classification-based metric learning algorithm. The algorithm is trained to classify faces in different identities, with a margin in softmax and representation constraints.

discriminative features that generalize well (Megaface [79], VGGFace2 [21], MS1M [54], ImdbFace [162], IJB-A,B,C [82, 169, 106]).

Those representations are then extracted from reference images and compared independently with the input image’s representation (Figure 5.3).

When tasked to identify, the input image (the “probe”) is encoded in a feature vector and compared to all the encoded reference picture vectors (the “gallery”), aiming for minimum distance on the correct identity. This strategy has some shortcomings outlined in [82]: it is unclear how to aggregate various feature vectors from several images of the same identity, and current systems are not precise enough to reject all the images in the gallery for unknown probe identity when the gallery has many pictures, triggering false detections. For testing, LFW [71] is a common test set as well.

### 5.3 Metric learning

In order to generalize to unseen faces, FR systems are usually seen as a metric learning (or similarity learning) problem instance. In general, metric learning aims to learn an embedding space that is semantically meaningful wrt to a chosen distance function.

Under this framework, we aim to obtain a neural network embedding a face into similar vectors for

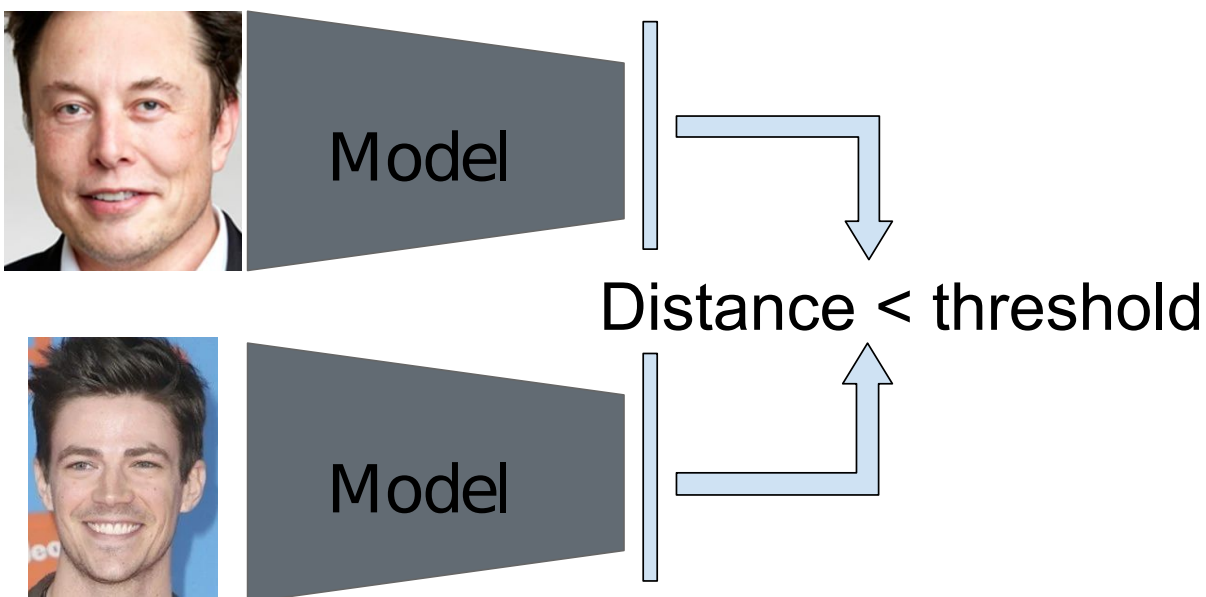


Figure 5.3: Test time usage of feature vectors. Two images' representations are compared under a distance metric. A distance under a predefined threshold indicates the same identity.

the same person and dissimilar vector for different people. The similarity measure is often euclidean or angular.

### 5.3.1 Triplet loss

The most direct translation of this goal is the Triplet Loss [63, 157, 65]. For a neural net  $f$  that takes a face as an input and outputs a vector, and a distance function  $d$ , we want to minimize

$$(5.1) \quad \mathcal{L} = d(f(A_i), f(A_j)) - d(f(A_i), f(B_k))$$

where  $A_i$  and  $A_j$  are two different pictures of a person A, and  $B_k$  is a picture of a different person B.

While this works, it requires a lot of time to converge.

### 5.3.2 Softmax

Posing the problem as a classification task allows to leverage the efficiency of the cross-entropy loss and the discriminative capability of neural networks. In order to frame  $f$  as a classifier, we need to define

$$(5.2) \quad h(x) = \text{softmax}(f(x) \cdot W^T)$$

Here,  $W$  is a learnable  $k \times d$  matrix where  $k$  is the number of identities to classify and  $d$  is the dimension of the embedding row vector produced by  $f$ . The  $i$ th row of  $W$  can be interpreted as the prototypical embedding for class  $i$ .

At inference time,  $W$  is discarded and a distance function is used to compare the embeddings. It is hoped that  $f$  was trained on enough identities in order to make a rich, semantically meaningful vector

space. During training, as the embeddings were discriminated with a linear classifier, a parameter-free distance function should be able to discriminate embeddings from people not seen during training.

### 5.3.3 L2-softmax

The softmax has some notable drawbacks: it does not enforce positive pairs to remain together and negative pairs further; it is biased towards the training distribution unbalance; and uncertain samples produce decisions with low confidence that are poorly penalized.

Ranjan et al. [129] (Figure 5.4a) proposes to fix all of those by enforcing a L2 constraint on both the output vector of  $f$  and each row of  $W$ . Instead of maximizing the softmax output with maximum inner product between the correct row of  $W$  and  $f(x)$ , this aims to maximize (minimize) the cosine similarity between the correct (incorrect) row of  $W$  and  $f(x)$ . If we accept the notation  $\langle n \rangle$  for a matrix  $n$  with each row vector normalized to unit length, the L2-softmax is:

$$(5.3) \quad h(x) = \text{softmax}(s \langle f(x) \rangle \cdot \langle W \rangle^T) = \text{softmax}(s \cos(f(x), W)) = \text{softmax}(s \cos \theta)$$

where the scalar  $s$  can be either interpreted as a radius or the temperature of the softmax. The higher the  $s$ , the more the softmax will resemble a hard argmax. As  $\cos$  is bounded in  $[-1; 1]$ , it is necessary to control the sharpness of the softmax with a temperature in order to control the gradient magnitude.  $\theta$  is the vector of the angles between  $f(x)$  and each row of  $W$ .

Note that in the equation above,  $\cos$  is applied independently to each row of  $W$  and  $f(x)$ , also making  $\theta$  a vector of size  $k$  representing the angles between  $f(x)$  and each row of  $W$ .

### 5.3.4 ArcFace

ArcFace [34] (Figure 5.4b) builds on this by enforcing a margin between classes. The L2-softmax (Eq. 5.3) emits a valid high probability as soon as  $f(x)$  has its smallest angle with the prototype of the correct class in  $W$ . They thought this was not enough and wanted to add another guarantee, that small perturbations to  $f(x)$  due to input distribution shifts or hard samples would not be predicted as another class. As such, they add an hyperparameter margin  $m$  in the equation to repel the embedding by  $m$ . By adding  $m$  radians to the angle of the correct class the angular distance with other classes is forced to be greater than the margin:

$$(5.4) \quad h(x)_y = \text{softmax}(s \cos(\theta_y + m))$$

where  $y$  is the index of the target class.

Some other margins were proposed prior to ArcFace such as [101] or [164] but showed less accuracy at test time.

## 5.4 $\Rightarrow$ Contribution: Face Recognition with Threshold-Softmax

### 5.4.1 Principle

ArcFace constrains the representation so that the angular distance to an incorrect class is at least of size  $m$ , but does not enforce any absolute requirement on the intra-class angles. I wanted to explore the opposite view and constrain the representation vector in the other way, so that the angle between samples of the same class is *at most*  $m$ , making it an absolute requirement.

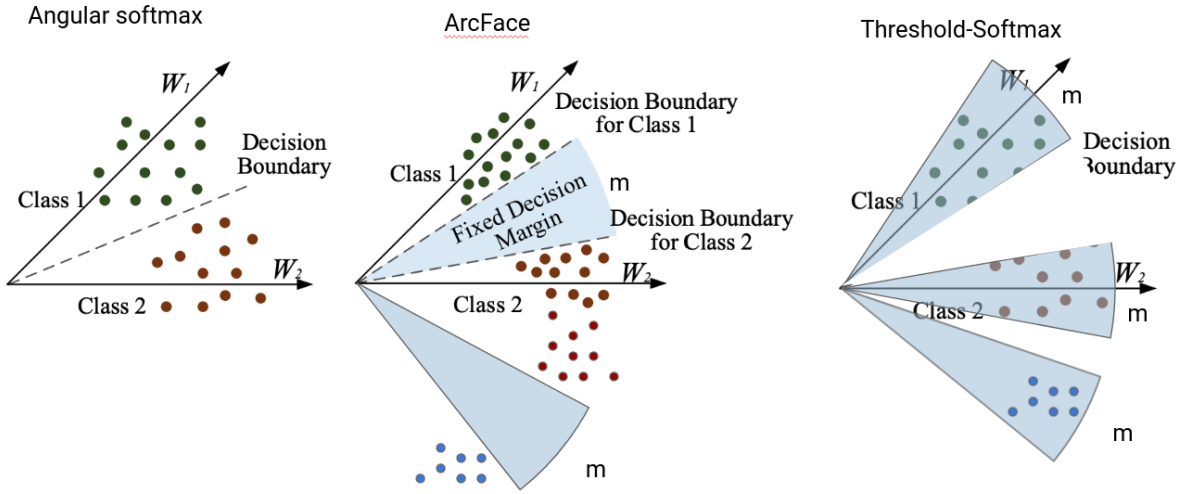


Figure 5.4: Comparison of the angular softmax, ArcFace and the proposed Threshold-Softmax. In ArcFace, the margin (in blue) is fixed but the width of the arcs of each class can be arbitrarily wide (or narrow), since there is no constraint on them. In threshold-softmax, there are no enforced margins but the decision boundaries have a fixed width. An artificial class is predicted outside of those trusted cones. Figure derived from [163].

I achieved this by concatenating an artificial entry  $\cos(m)$  to  $\theta$ . We say this entry has class  $\Omega$ . If all the angles in  $\theta$  are greater than  $m$ , then this artificial entry is maxed out by the softmax function, leading to a high loss until the correct angle is finally smaller than  $m$ .  $m$  being a hyperparameter. The loss function can be expressed as:

$$(5.5) \quad h(x) = \text{softmax}(s[\cos(f(x), W); \cos m])$$

Figure 5.4 highlights the difference with standard angular softmax, ArcFace, and Threshold-Softmax.

### 5.4.2 Extension: Threshold-Softmax with negative samples

The Threshold-Softmax loss function offers us a clear definition of "negative space", aka all vector that the softmax would classify as  $\Omega$ . This can be naturally leveraged by having "negative samples" (ie samples not belonging to the classes set), and aiming to classify them as class  $\Omega$ . That way, we can collect cheap samples as this set of negative samples just has to be cleaned of overlaps with the positive samples, and use them as an additional source of supervision. Negative faces might have people looking like some positive identities and the loss would enforce the network to distinguish them so as to place those in the negative space. This is visualized in Figure 5.5.

### 5.4.3 Evaluation

In the subject independent scenario, the dataset LFW [71] test set is commonly used to assess the quality of face representation. It consists of 6k image pairs, half being faces of the same identity, half being of different ones, formed with 13233 source images from 5749 people. The model has to decide whether a pair of pictures belong or not to the same identity.

State-of-the-art accuracy on this set surpassed 99% which drove the creation of newer and harder sets.

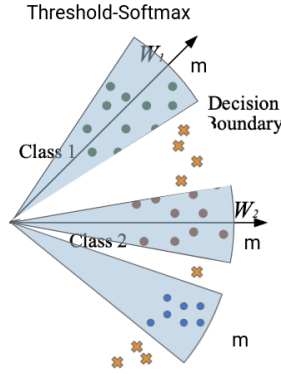


Figure 5.5: Threshold-softmax with negative samples: crosses are negative samples. We do not know their identities, we just know they do not belong to any of the known identities. Threshold-Softmax naturally uses those samples by placing their identities outside of the known classes decision boundaries, ie, predicting class  $\Omega$ .

|                         | LFW (accuracy, %) | FGLFW (accuracy, %) |
|-------------------------|-------------------|---------------------|
| L2-Softmax              | 97.66             | 88.88               |
| ArcFace                 | 98.63             | <b>95.58</b>        |
| Threshold-Softmax       | 98.88             | 93.51               |
| Threshold-SM + Negative | <b>98.93</b>      | 95.23               |

Table 5.1: Accuracy on face verification for LFW and FGLFW image pairs for various loss functions. Best rejection angular threshold selected for each method.

FGLFW [35] reuses pictures from LFW but selected harder pairs. DeepFace [148] has an accuracy of 92.87% on LFW but 78.78% on FGLFW.

The Megaface challenge [79] extends this beyond pairs. They propose an input "probe" picture and a gallery of "candidate" pictures. The model has to identify which picture in the gallery is of the same person as the probe.

IJB-A,B,C propose a collection of challenges including verification and identification in both pictures and videos.

#### 5.4.4 Experimental study

We experiment with our new loss function by training on MS1Mv2 [54]. It contains about 5.8M images of 85k subjects. We test on LFW and FGLFW. We compare L2-Softmax / Angular Softmax, ArcFace, Threshold-Softmax, Threshold-Softmax with negative samples, at various data availability.

In order to cut training times down, we train on 5% of the training data for 10 epochs, with a Resnet18 pretrained on ImageNet.

We conduct an experiment with extensive hyperparameter search (for  $m$ , the learning rate and the weight decay) and sum up our results in Table 5.1. In the Threshold-Softmax with negative samples case, we add another 5% of the dataset with a single negative label. We assess the performance of our proposed loss according to the threshold value in Figure 5.6.

We see that the Threshold-Softmax is competitive with ArcFace and better than the raw L2-Softmax. Furthermore, training with negative samples indeed boost the performance of Threshold-Softmax.

Further testing should be conducted with bigger data and epochs budget in order to compare those algorithms in realistic training situations.

## Accuracy and threshold value

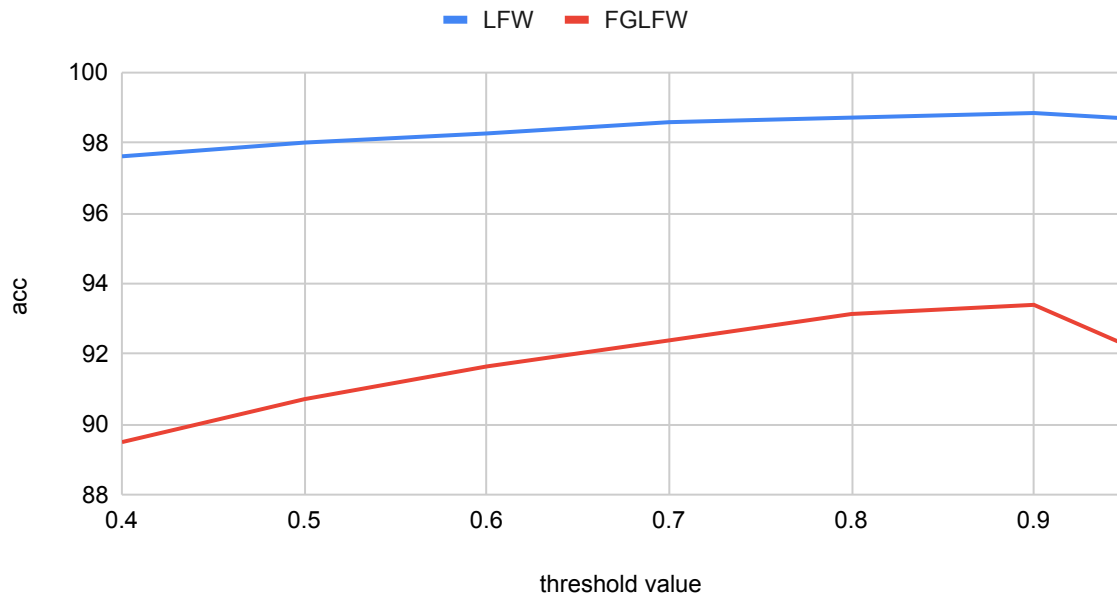


Figure 5.6: Accuracy on LFW and FGLFW according to the hyper parameter threshold value  $m$  for Threshold-Softmax.

It is interesting to note that ArcFace’s margin is not mutually exclusive with the cones of trust of Threshold-Softmax and the two could be combined. This is left as future work.

### 5.4.5 Conclusion

We proposed the Threshold-Softmax loss function that is able to use negative samples that are cheaper to collect. The Threshold-Softmax proposes to learn face embeddings fitting a cone with an absolute maximum angle, rather than imposing an angular margins between classes. Negative samples are forced in the negative space: outside of the regions allocated for the positive classes. We experimented this loss on MS1Mv2 and compared it to the state of the art ArcFace. The Threshold-Softmax is competitive but not always superior to ArcFace, but presents the ability to learn from unlabeled negative samples (unknown people not belonging to any positive class), halving the error rate in our tests on LFW and FGLFW. Those cones are not mutually exclusive with ArcFace’s margins and future works could include adding margins to the Threshold-Softmax.

## 5.5 ⇒ *Contributions*: distractor-robust face recognition for a closed-set of identities

The following sections will highlight our contributions:

1. a comparison of ArcFace and cross-entropy classifiers in the context of closed-set face recognition (Sections 5.5.1, 5.5.2, 5.5.5)

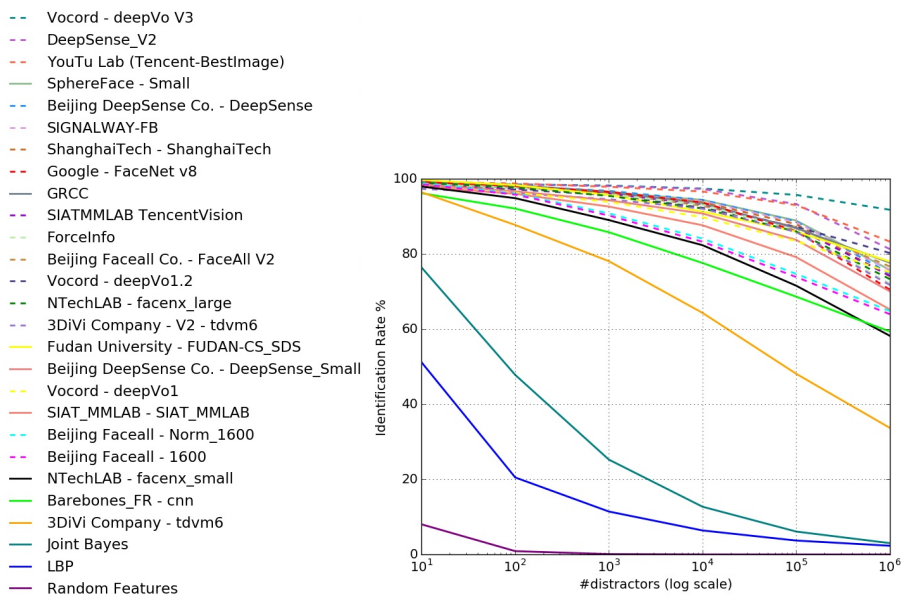


Figure 5.7: Rank 1 identification performance for contestants on Megaface’s Facescrub challenge under various quantities of distractors. Most models have their performance degrading quickly even with 100 distractors only. Figure from [79].

2. a search for a strategy to manually deal with label noise, curate and expand a face recognition dataset for closed-set scenarios (Sections 5.5.3, 5.5.4)
3. a set of losses and their evaluations allowing to leverage unlabeled negative faces and make face recognition system more robust to distractors (Section 5.5.5)

### 5.5.1 The limits of metric learning

Our first attempt to address face recognition in our industrial context was based on a publicly available face detection and face recognition pre-trained CNNs. The problem was posed as an open-set face recognition task, and consisted in matching a feature vector extracted from an input photo against each reference image in the database feature vectors, as customary in metric learning. Unfortunately, despite the impressive number, having an accuracy of 99.38% on LFW means that identifying someone by pair matching among 4000 identities (as we wished initially) triggers 25 positive matches. In those 25 positive matches, at least 24 are obviously false, as one person has only one identity, which may or may not be in the database. The performance scored in our setting was substantially worse due to distribution mismatch between the training domain and our application domain.

This shows that the metric learning strategy reached its limits in this setting and is not robust enough to scale to many identities and many distractors. Metric learning aims at learning a powerful embedding function, which is too ambitious for our problem: filtering out distractors, and recognizing a handful of persons of interest. Filtering out distractors better describes the task at hand, allows the net to focus on learning only the facial features for the set of the persons of interest, plus a rejection criterion or margin for rejecting distracting faces.

Indeed, Figure 5.7 shows that most of the models which participated in the MegaFace challenge are sensitive to the number of distractors and their performance quickly degrades under 95%. A performance under this value is out of the acceptance threshold for our industrial application.

## 5.5.2 Back to simple classifiers

In fact, as the identities to recognize are known ahead of time and part of the training set, it is possible to cast the project as a classification problem, using a regular cross entropy loss. We would be performing the classification from the softmax probabilities, just like the action classification (see Section 3.4).

### Calibration

Models trained with a cross entropy loss should exhibit a useful property: calibration. A model is said to be calibrated when the predicted probability aligns with the actual correctness rate of the prediction. For instance, labels predicted with probability 0.3 indeed have a 30% chance of being correct: *the predicted probability is equal to the true probability*. Calibration is extremely important in some systems: we could ignore predictions with confidence lower than 0.95 if would not tolerate more than 5% of errors, or a medical system could perform automatic labeling on high confidence predictions and require a human expert label on lower scores.

Unfortunately, modern neural networks are poorly calibrated [52]. They have a high capacity and end up overfitting on the training loss, predicting only high confidences. Guo et al. [52] proposes to fix this by learning a temperature parameter before the softmax on a separate set. In our tests, this approach was both extremely simple and efficient, for a negligible computation cost and a low code complexity. Figure 5.12 shows calibration plots.

Computing a calibration plot is easy: one can bin predicted probabilities on the validation set, and evaluate the actual correctness ratio in each bin. i.e.: For samples predicted with confidence 0.9-0.95, the predicted label should be correct 90-95% of the time.

### Rejection

Hendrycks and Gimpel [61] claim that Out of Distribution (OoD) samples have a lower softmax probability than in-distribution samples. This is something we have not observed to be true or sufficient to reliably reject distractors. Moreover, uncalibrated models make thresholding rejection confidence score hard. The predicted probabilities lose their meaning and become detached from the semantics they represent, just reflecting the overfitting level of the model. The best threshold value might then change between training runs and its value is not interpretable.

## 5.5.3 Dataset denoising and distractors

We propose a strategy to incrementally clean a dataset from label noise by expanding the set of identities to recognize. We explicitly train on negative samples (“distractors”) and select a subset of HFaces’ identities as a positive training set and the rest as negative set. This allows to grow the positive set in a controlled way, so that the negative set can be denoised progressively. If we had to investigate a classifier among thousands of identities at once, it would be extremely hard to gain insight into the dataset and take actions. Instead, we use as a positive set the top  $N$  most popular identities, learn a model, evaluate it, clean or complete the training set, then add another  $N$  positive identities when the performances are good for our setting, and iterate. For our dataset, iterating by chunks of  $N = 20$  is convenient.



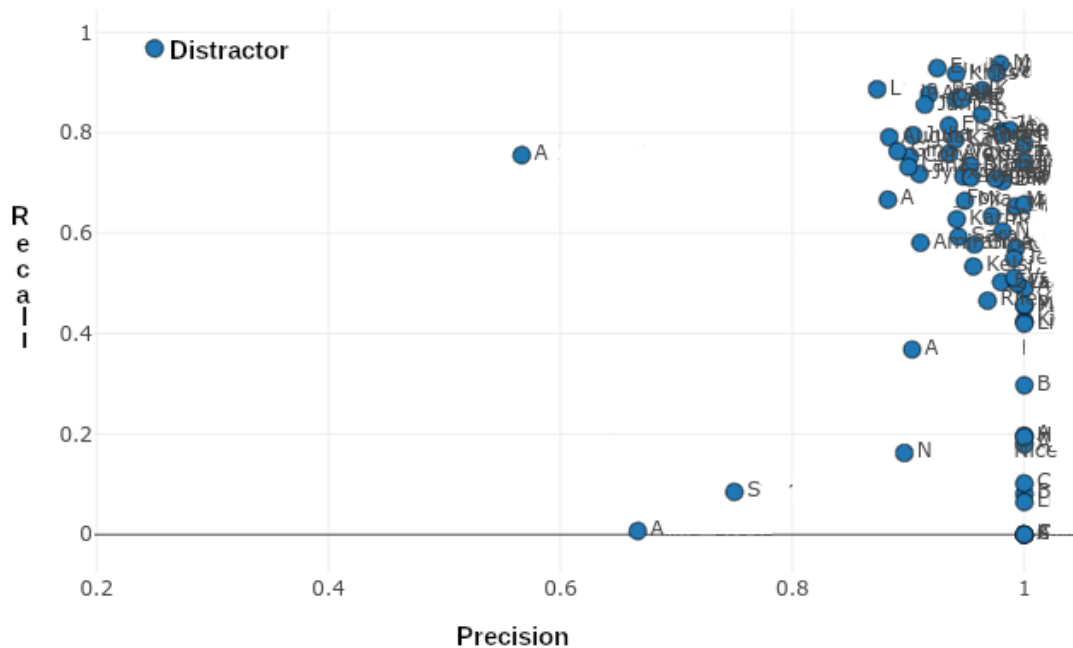


Figure 5.8: Each class (abbreviated to the first letter of the person’s name) is placed on this grid depending on its precision and recall scores. Top right is best, bottom left is worst. We aim to find strategies that help moving each point right and up. (Produced by the DCE model, Section 5.5.5)

### 5.5.4 Precision / Recall for dataset construction

#### Motivation

As the training data can be modified as well in order to address the task, we need a strategy in order to grow the dataset in a principled way. Mainly, at any point in time, there are 3 possible situations and their associated response:

1. Everything works well enough: grow the set of VIPs
2. Some VIPs seem to cause too much confusion and trigger too many false positives: rework the data for those identities (looking for errors, low quality or confusing samples) in order to identify harmful samples
3. Some VIPs are not detected in tests (false negatives) : add samples. Those identities have insufficient data to generalize correctly.

Which one of those actions to take is crucial in order to control the growth of the model and its performance.

#### Principles

Diagnosing the current state of the dataset in order to choose what to do next happens to be easy thanks to simple metrics. By computing precision / recall by class we get a view of the model performance.

| Pred \ True | A | B | C | Distractor |
|-------------|---|---|---|------------|
| A           |   |   |   |            |
| B           |   |   |   |            |
| C           |   |   |   |            |
| Distractor  |   |   |   |            |

Figure 5.9: A, B, and C are three fictitious identities for illustration purposes. We compute some metrics by selecting various meaningful subsets from the confusion matrix: **distractor accuracy** (blue rectangle), **identification accuracy** (orange rectangle), **kept accuracy** (violet rectangle), and **total accuracy** (green rectangle). For each subset, the metric is computed as the sum of the green cells it contains divided by the sum of all cells it contains.

In order to easily visualize which class need work, we plot each class on a grid, seen in Figure 5.8. We empirically hypothesize that classes with low precision suffer from noisy training data and/or labels, while classes with low recall indicate a lack of training data.

These assumptions bring insight into the model’s state, are easy to implement, interpret and use but bear the drawback that some additional effort must be taken in order to build a meaningful test set for each identity. As we observe strong outliers on the precision/recall plots while building the dataset, we hypothesize that the performance across identities is not correlated (or not enough), and therefore we believe we cannot use a random subset of identities to estimate the overall performance.

In our situation, we accept trading recall for precision as false positives damage the user experience with erroneous suggestions or search results. With false negatives, no labeling is performed, and the user browsing is not impacted.

## Results

In order to investigate these hypotheses, we select a class which performs well in both precision and recall and run three experiments (with the DCE model, Section 5.5.5):

1. A base run gives 94% precision and 99% recall.
2. We subtract 50% of the training data for that class, obtaining 98% precision and 86% recall. Here, it is clear that removing data deteriorates recall. By reducing the amount of training data, the identity is more narrowly represented, thus less sensible to accept other faces as belonging to it: the precision slightly increases.
3. We introduce label noise by integrating someone else’s samples into this class’ training data (up to 25%). We obtain 94% precision and 98% recall.

This data point seem to suggest that recall indeed correlates with the amount of data for a class. Unfortunately, precision does not negatively correlate with label noise. The classifier seems to overfit the noise rather quickly and learn a multimodal class. The question of detecting noisy classes remains unanswered by this approach.

### 5.5.5 Experiments

We devise a set of experiments in order to evaluate the progress done on the task of face recognition on HFaces. We emphasize the industrial context: this tool is to be used in order to label videos on the

customer’s platform. As such, it is better not to annotate a video than to predict a wrong label. Having the product owner or production engineers being able to set a false acceptance rate in production would be a very interesting feature as we could decorrelate the model training from production settings.

We train our model on a subset of the 105 most popular VIPs among 8k identities in our dataset, the remaining ones are set as distractors. The model, a standard resnet18, pretrained on ImageNet, is trained for 40k iterations with a batch size of 1024, Stochastic Gradient Descent, a linearly decaying learning rate starting at 0.1 and a weight decay of  $5e-4$ . The test set contains 119k pictures (resized to 96x96), with 24% of them being distractors.

For each experiment, we measure:

**Distractor accuracy** accuracy on the distractor set: among all distractors how many have been predicted as such, ie, the true rejection rate (See Figure 5.9, blue rectangle).

**Identification accuracy** the recognition accuracy among non-distractors (See Figure 5.9, orange rectangle).

**Total accuracy** accuracy on all test samples (See Figure 5.9, green rectangle).

**Mean F1** mean of the F1 score of each class

**Kept accuracy** accuracy of kept (non rejected) predictions. We consider that we reject predictions classified as distractors (See Figure 5.9, violet rectangle). This allows us to evaluate how many false identification will be performed on the platform, and choose our precision / recall tradeoff.

**True Positive Rate (TPR)@95, TPR@99** When techniques give scores with predictions and the model performs reasonably well, wrong predictions are given a lower score and correct predictions a higher score. This allows us to trade precision for recall, as we can find a threshold value which rejects predictions with low scores until a chosen true positive ratio. Thus, we also compute the identification and total accuracy for 95% and 99% true positives, giving us an estimate of the recall for both test sets at those true positive rate goals.

**F1 Area Under Curve (AUC)** We also compute the area under the F1 curve as a function of the threshold, as it captures the sensitivity of the model to the thresholding value. A high value (close to 1) indicates that the model has a maximum F1 value irregardless of the threshold, whereas a value of 0.5 indicates that there is a high threshold sensitivity, a bad mean F1 score, or both, which are all unsatisfactory.

**Calibration** Finally, we compute the calibration curves.

All these metrics are shown for the tested models in Figure 5.11 and Figure 5.12.

The total accuracy highly depends on the ratio of distractors in the test set, which is, in this situation, arbitrary. While it remains an interesting metric, the mean F1 more closely captures our expectations: if all the classes are given equal importance, what is the best precision / recall we can reach? Generalizing this question to all possible thresholds (all possible VIP / distractor decision boundaries) gives us the F1 AUC which will be our metric of choice *given that the model has a satisfying calibration curve*. If this condition is not met, the precision / recall tradeoff (and thus F1 score) can’t be set based on production needs. A model able to control this tradeoff by thresholding the predicted scores, is said to be *amenable to thresholding*.

We compare several models: ArcFace, and cross-entropy classifiers. We train a simple classifier (CE) and also explore several techniques for exploiting distractors from the training set, and rejecting them at inference: an extra class for distractors (DCE), maximizing entropy on distractors (ME),

| Model                | distractor accuracy (%) | identification accuracy (%) | total accuracy (%) | mean F1     | kept accuracy (%) |
|----------------------|-------------------------|-----------------------------|--------------------|-------------|-------------------|
| ArcFace              | 78.38                   | 43.93                       | 52.03              | 0.40        | 73.11             |
| Cross-Entropy (CE)   | 79.65                   | 73.51                       | 74.96              | 0.65        | 85.40             |
| CE with Distractors  | <b>95.17</b>            | 65.35                       | 72.36              | 0.62        | <b>92.82</b>      |
| CE + Zero-Logits     | 79.09                   | 78.67                       | 78.77              | 0.66        | 84.92             |
| CE + Maximum-Entropy | 80.31                   | <b>81.45</b>                | <b>81.18</b>       | <b>0.67</b> | 85.89             |

Table 5.2: Various metrics for each model, rejection threshold selected at maximal *total* accuracy. **Best** and *second best* results are highlighted

| Model                | distractor accuracy (%) | identification accuracy (%) | total accuracy | mean F1     | kept accuracy (%) |
|----------------------|-------------------------|-----------------------------|----------------|-------------|-------------------|
| ArcFace              | 98.89                   | 25.33                       | 42.62          | 0.45        | 99.42             |
| Cross-Entropy (CE)   | 97.02                   | 58.05                       | 67.21          | 0.70        | 97.19             |
| CE with Distractors  | 99.26                   | 54.65                       | 65.14          | 0.69        | 98.84             |
| CE + Zero-Logits     | 97.84                   | <b>62.58</b>                | <b>70.87</b>   | 0.75        | 97.98             |
| CE + Maximum-Entropy | <b>99.48</b>            | 59.63                       | 69.41          | <b>0.78</b> | <b>99.36</b>      |

Table 5.3: Various metrics for each model, rejection threshold selected at maximal F1. **Best** and *second best* results are highlighted

minimizing logits on distractors (ZLog). We report, in Table 5.2, the various metrics *at the maximum accuracy* and at maximum mean-F1 in Table 5.3. The F1 AUC is reported in Table 5.4. As we shall see, all models tested here exhibits satisfying calibration. Therefore, we present their performance when selecting a confidence threshold giving 99% and 95% of true positives in Figure 5.10 as an additional informative signal.

### ArcFace

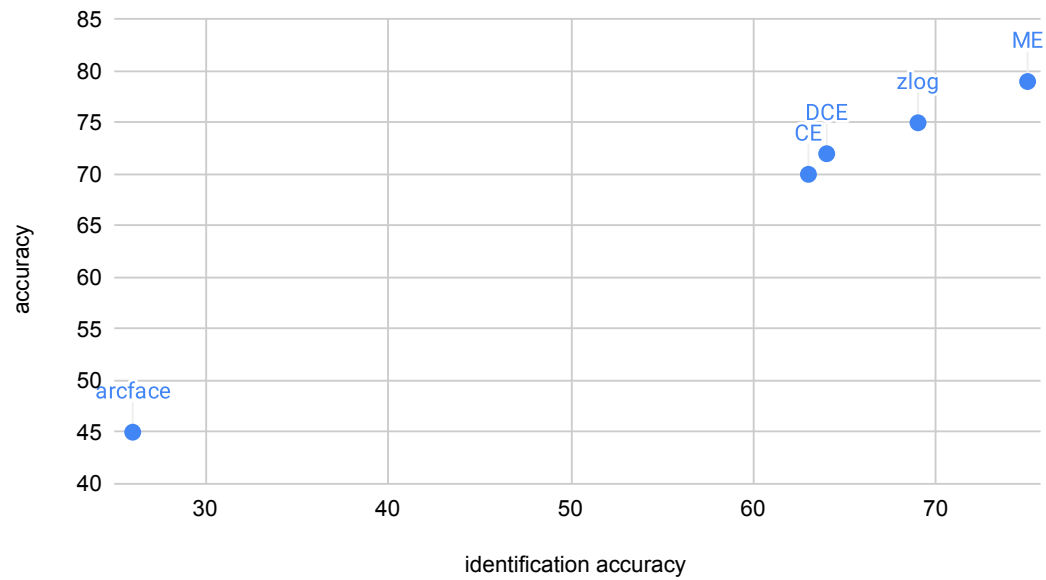
We first verify our claim that metric learning is not robust in our setting. We reuse the pretrained ArcFace model published by <https://github.com/foamliu/InsightFace-v2>. The important things to note are:

- this model has been trained for a face-independent protocol, that is, it has not been trained on the people it was meant to recognize;
- this model has several data biases: our distribution includes more females than males, very few above the age of 50. The training set is broader than this narrower distribution of ours. The model

| Model                | F1 AUC      |
|----------------------|-------------|
| ArcFace              | 0.34*       |
| Cross-Entropy (CE)   | 0.64        |
| CE with Distractors  | 0.59        |
| CE + Zero-Logits     | 0.68        |
| CE + Maximum-Entropy | <b>0.72</b> |

Table 5.4: F1 AUC for the models evaluated. Value for ArcFace is normalized for comparability (0.26 to 0.34)

TPR@95



TPR@99

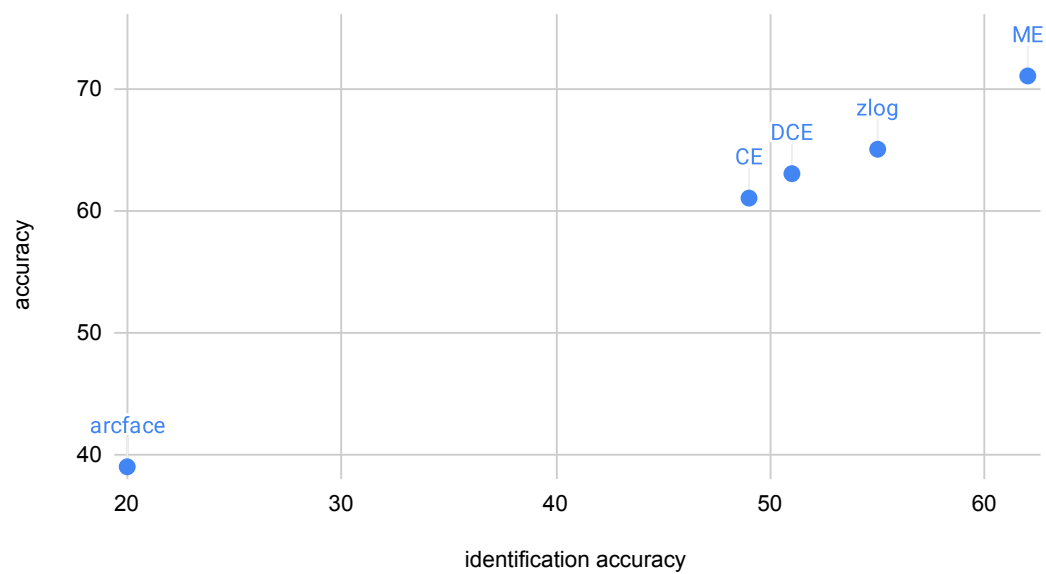


Figure 5.10: Identification accuracy and total accuracy for a true acceptance rate of 95% (top) and 99% (bottom). CE: Cross-Entropy, DCE: Cross-Entropy+Distractors, ME: Cross-Entropy+MaximumEntropy, zlog=Zero-Logits.

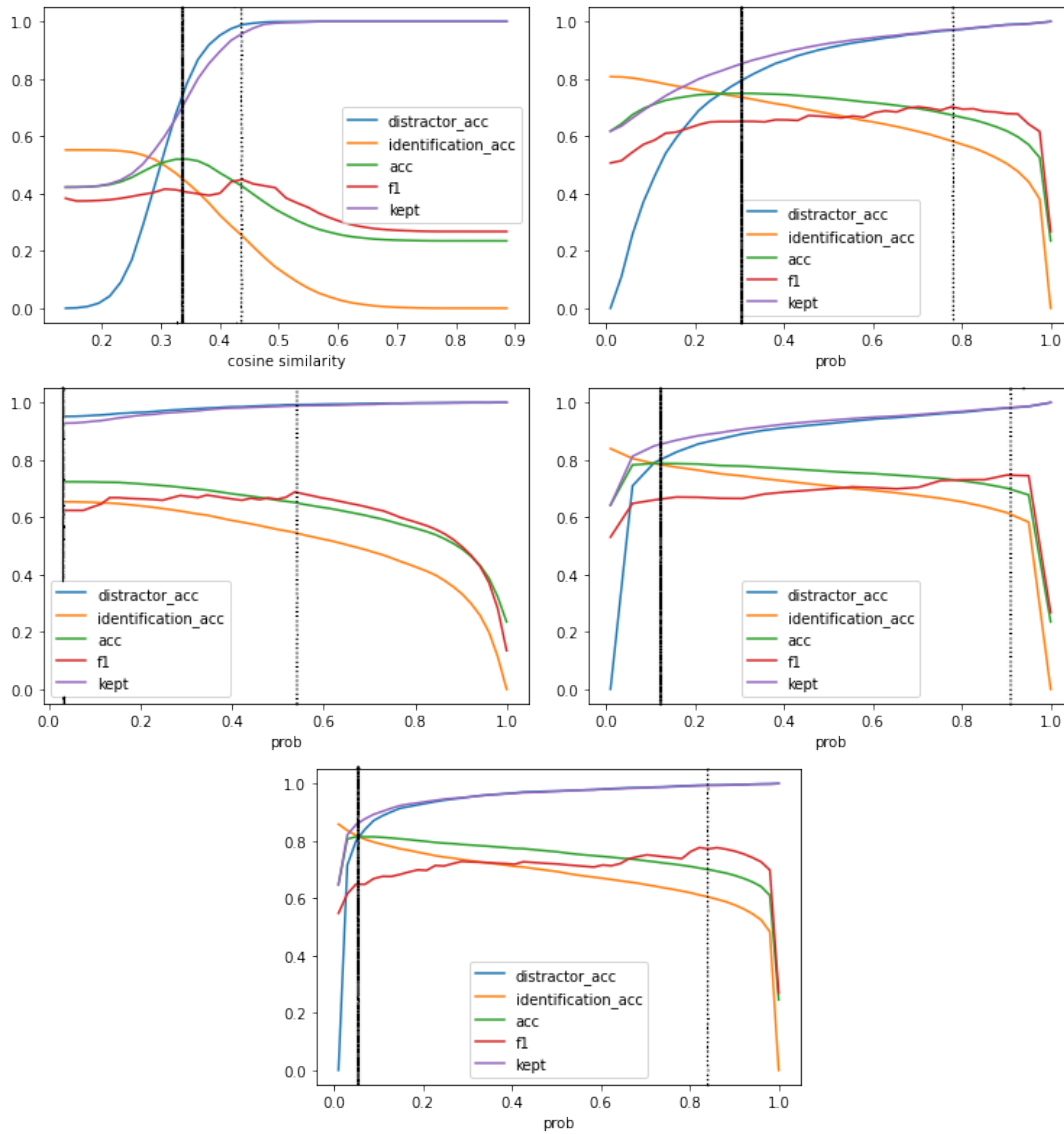


Figure 5.11: Various metrics for the a) ArcFace b) CE c) DCE d) ZLog e) ME model as predictions are set as distractors under various threshold values. The black bar traverses all plots at the best total accuracy, the dotted bar is located at maximum F1. As we sweep over the threshold values and reject more samples as distractors, we look at the variations on the metrics.

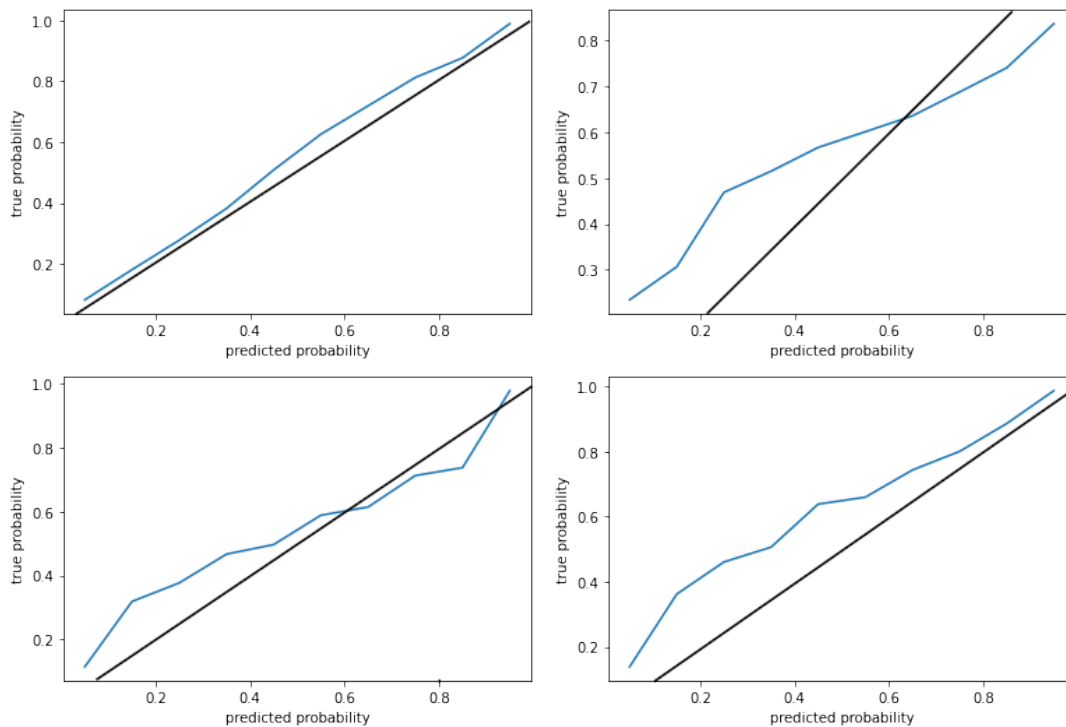


Figure 5.12: Calibration plots for the a) CE b) DCE c) ZLog d) ME models. See Section 5.5.2 for more details about calibration.

is faced with a narrower and more fine-grained task than intended;

- Metric learning makes it hard to know when we don't know, thus rejecting distractors is not easy;
- ArcFace protocol is about computing the cosine similarity between the representation of the input image and a reference image. For identification, we randomly choose one reference image per class. The choice of that reference image could have been selected with respect to a validation set for minor gains. We made sure that the different metrics do not dramatically change between each run.
- We set a distractor threshold. If all reference images have a cosine similarity lower than this threshold, we reject this input as a distractor. We compute our metrics sweeping over threshold values.

Figure 5.11a shows different metrics and plots on this experiment. We see that the rejection threshold maximizing total accuracy is about 0.35. the first plot shows that all distractors are concentrated between a cosine similarity of 0.25 and 0.45, which unfortunately overlaps with VIPs, between 0.25 and 0.6: there is no clear boundary to separate both and rejecting distractors implies rejecting correctly identified VIPs too. The kept accuracy show that detections above 0.46 are all correct, and there is no point setting a higher threshold value.

The plots show a best accuracy value of 52.03% for a distractor accuracy of 78%, an identification accuracy of 44% and F1 score of 0.40.

### Cross-Entropy (CE)

We now compare the advanced ArcFace loss with a standard softmax+cross entropy loss, trained on the persons of interest only. This model has no built-in way of signaling distractors. We are going to assume that the distractors produce predictions with lower confidence, and threshold them on low confidence scores.  $H(\cdot, \cdot)$  is the cross entropy function. In equations, we note  $\mathbf{x}^+$  and  $\mathbf{y}^+$  the non-distractors training examples (positive examples, persons of interest) and their associated label, and  $\mathbf{x}^-$  the set of distractors in the training set. A learned softmax classifier is noted  $f(\cdot)$ , and its logit predictions  $\log f(\cdot)$ . The loss function is defined by:

$$(5.6) \quad \mathcal{L}_{CE} = H(f(\mathbf{x}^+), \mathbf{y}^+)$$

Figure 5.11b indicates that this heuristic holds some truth. It seems that around 90% of distractors have a confidence score below 0.42 and half of the people of interest get a confidence score over 0.95. Unfortunately this heuristic is not entirely satisfying. As the rejection threshold increases, the overall accuracy slowly decreases. Fortunately, the increasing F1 score further indicates that there is a possible precision / recall tradeoff that can be used, and that the model is actually able to separate to some extent VIPs and distractors based on the confidence values. The calibration plot in Figure 5.12a is also very close to the ideal one.

It is unsurprising that the base identification accuracy of 81% is much greater than the 55% of ArcFace. This model gets a peak accuracy of 75%, for a distractor accuracy of 80%, an identification accuracy of 73%, and a F1 score of 0.65. The mean F1 score peaks to 0.70 and the F1 AUC bumps from 0.34 to 0.64. This model is already indisputably more suited for our task.

### Cross-Entropy with Distractors (DCE)

Aiming to improve both the representational power of our model and teaching it the ability to reject distractors, we add distracting faces to our training data that have to be classified as their own distractor class. The loss function is defined by:

$$(5.7) \quad \mathcal{L}_{DCE} = H(f(\mathbf{x}^+), \mathbf{y}^+) + H(f(\mathbf{x}^-), \mathbf{distractor})$$

It is important to notice that this model is the only one presented here able to disambiguate between uncertain VIP classification and distractors: it is able to give a high confidence score to the Distractor class or predict low scores, indicating lack of confidence in any VIP or distractors. However, being able to distinguish both cases is not useful to our industrial scenario, and we consider this as a convenient feature rather than a requirement.

Similarly to the previous model, we evaluate how the different metrics behave when we also consider as distractors predictions below a given threshold in Figure 5.11. The peak accuracy of 72% is below the one of the simple classifier, explained by a lower ability to recognize the people of interest (from 73% to 65%) but a much greater ability to detect distractors (from 80% to 95%). Overall the models have a comparable maximum F1 score of 0.62 but the F1 AUC that deteriorates to 0.59 and is less amenable to thresholding as the flat F1 and deteriorated calibration plots suggests. However, the model exhibits slightly greater recall for a given error rate, as shown in Figure 5.10.

### Cross-Entropy + ZeroLogits (zlog)

Inspired by Vaze et al. [159] and [129], we hypothesize that distractors can be sorted by a lower logit value than persons of interest. Distractors are not their own class anymore, but are discriminated from



the low logits they produce. We propose penalizing their squared logits, further encouraging them to zero, and classify only the persons of interest with a cross-entropy loss. The loss function is defined by:

$$(5.8) \quad \mathcal{L}_{ZLog} = H(f(\mathbf{x}^+), \mathbf{y}^+) + \text{MSE}(\log f(\mathbf{x}^-), \mathbf{0})$$

We find that this brings some improvements in identification accuracy and total accuracy, bringing the F1 score at maximum accuracy to 0.66 and the best F1 to 0.75. The F1 and kept plots of Figure 5.11d show that this model is amenable to thresholding, and the F1 AUC of 0.68 performs preferably to the CE model (0.64). This model gets either the best or second best scores at both max accuracy and max F1, and performs better at fixed true positive rate.

However, contrarily to Vaze et al. [159], we don't observe logits to be more informative than softmax probabilities. We tried thresholding distractors with logits in each model, every time bringing equivalent or lower results. For this reason, we do not report those results, and threshold only on softmax confidence scores.

### Cross-Entropy + Maximum Entropy (ME)

Finally, observing that the zlog model was not working as well on logits than on probabilities, despite the loss working directly with them, we envision working on probabilities. Model CE shows that distractors naturally produce lower probabilities, so we aim to predict a maximum entropy (flat) distribution for distractors. Semantically, the DCE model tells us that distractors are persons that have to be recognized as not belonging to other identities. The ME model is less powerful and states that a distractor is someone that produces maximal confusion. From a learning point of view, now that the model does not have to remember distractors in order to classify them with high confidence, it can dedicate its full capacity to learning only accurate decision boundaries around each VIP.  $H(\cdot)$  is the entropy function,  $H(\cdot, \cdot)$  is the cross entropy function. The loss function is defined by:

$$(5.9) \quad \mathcal{L}_{ME} = H(f(\mathbf{x}^+), \mathbf{y}^+) - H(f(\mathbf{x}^-))$$

We evaluate this model at maximum accuracy, and get a best identification accuracy of 78.93%, total accuracy of 81.43%, and mean F1 of 0.72 shown in Table 5.2. Thresholded for maximal F1, this model also obtains the best F1 score and is still highly competitive with ZLog in Table 5.3. It gets the best F1 AUC (Table 5.4) by a significant margin, since its F1 curve is strictly above ZLog's. Figure 5.11e shows that the model is amenable to thresholding from the increasing F1 plot and the kept accuracy / distractor accuracy curves strictly better than ZLog's. AT 95% and 99% TPR (Figure 5.10) this model dominates all others. Finally, its calibration plot in Figure 5.12e displays a close to perfect calibration for confidence values above 0.6, which is the range of values that will be of interest for a high precision deployment scenario.

### Discussion

Overall, the ME approach results in the best way to leverage distractors in order to build our industrial system, as can be seen on the TPR plots (Figure 5.10) and our F1 AUC metric. It results in the best total and identification accuracy and F1, while being amenable to thresholding. The ZLog technique scores second but is inferior in every way.

The DCE method attracts our attention as well, scoring the highest distractor accuracy and a nice semantic interpretation. However, its identification and total accuracy, as well as its F1 are the

lowest among the cross-entropy classifier, showing that it might just favor detecting distractors rather than recognizing VIPs. We conclude that this strategy just over emphasizes classifying as distractor, decreasing identification recall too strongly. Its distractor/kept curves dominating ME's as well as the TPR plots further indicates that this model overemphasizes precision over recall. All considered, The ME is a better compromise as we will have a slightly higher error ratio (that can be negotiated with thresholding) but a much better recall, hence predicting many more correct labels.

We highlight how observing that OoD samples naturally score lower softmax probabilities lead us to the ME loss that encourages this behavior, giving us the best results we obtained in these experiments. Indeed we can observe on Figures 5.11b and 5.11e that the metrics follow the same trends with similar shapes, but ME strictly improves on them.

## 5.6 Conclusion

We showed that off-the-shelf face recognition classifiers, trained with ArcFace, were not satisfying for our industrial scenario. We first explored remaining in the Metric Learning realm and proposed the Threshold-Softmax loss function that is able to use negative samples that are cheaper to collect. The Threshold-Softmax proposes to learn face embeddings fitting a cone with an absolute maximum angle, rather than imposing an angular margins between classes. Negative samples are forced in the negative space: outside of the regions allocated for the positive classes. We experimented this loss on MS1Mv2 and compared it to the state-of-the-art ArcFace. The Threshold-Softmax is competitive but not always superior to ArcFace, but presents the ability to learn from unlabeled negative samples (unknown people not belonging to any positive class), halving the error rate in our tests on LFW and FGLFW. However, Threshold-Softmax remains a technique for subject-independent face verification and identification.

We moved away from metric learning and went back to cross-entropy classifiers as our system only has to recognize identities known ahead of time, in a subject-dependent, open-set fashion. After showing the inefficacy of ArcFace in our situation, we explored various ways of making it robust to distractors, unknown people that the system must learn to discriminate. We explored various techniques to reject distractors at inference time and use distractors at training time, and found that maximizing the entropy for distractors to be the best performing strategy we tried among regularizing the logits or adding a Distractor class. We further showed that all models were quite satisfyingly calibrated and amenable to thresholding, making production engineers able to decide on the precision/recall that is best for the product. The ME model is used in production today at Hexaglobe.

We search for a principled way to refine and extend the training set, leveraging precision/recall plots. While our hypothesis that recall correlates with the amount of data for that class looks promising, we still don't know how to manipulate precision and our hypothesis that noise decreases precision has been disproved.

The final system is currently used in production, labeling 15k videos a day, and the extracted labels are used as planned to enrich the user experience.

## 6 Fixing Datasets with generative models

### 6.1 Introduction

Deep Learning has proven to be successful at generating natural images. Antoniou et al. [6] see in this ability an opportunity to improve datasets by generating more data and shows performance improvements in classifiers when using generated data as a supplement to the training data.

Using generative models in the context of face recognition is appealing. Many pictures per identities are needed in order to teach a classifier that it should be invariant to lighting, pose, makeup, haircuts, etc. However, as we grow the number of identities that the system has to recognize, there is a risk that the classifier does not learn the invariants for identities with less variations in training data. In other words, we fear that the classifier learns useful features only for the identities with many diverse pictures and overfit the case with little training data.

Generating data gives us the opportunity to create the diversity of pose, lightning etc for the identities with the least diverse identities.

In this chapter we lay out a review of the different techniques of generative models we explored before settling on one. We will explore GANs and VAEs, and more specifically the VQ-VAE for which we will present our contribution: an expiration process for the codebook in order to improve its training dynamics and performance. We then introduce our chosen system for data augmentation in the context of face recognition.

### 6.2 Generative models for building invariants

In our dataset, some people were always facing the camera, or never smiling, while others exhibited larger variations in pose, exposition or image quality. We hypothesized that this could lead the classifier to learn some shortcuts like "This is not Person A because that person is smiling, A never does". One obvious way to teach invariants to classifier is to feed them proper data exhibiting those invariants. In face recognition, that would translate in making sure that all identities have various level of illuminations and a wide variety of poses so that the classifier does not learn shortcuts. We aim to complement the dataset with the missing variations of each identity by training a generative model. Hopefully, the model would learn the general concepts of face geometry, disentangle it from facial identity, and could reenact anyone's face into any pose.

### 6.3 Problem definition

Let  $\mathcal{D}$  be a dataset containing some face pictures  $x_i$  and their identity label  $y_i$  such that  $(x_i, y_i) \in \mathcal{D}$ . Let  $p_i \in \mathcal{P}$  be an unknown semantic latent vector representing pose, lighting etc, containing no information

about  $y_i$ , such that a powerful generative model  $G$  could hold  $G(y_i, p_i) = x_i$  (see figure 6.1).

Those faces can be considered samples of an underlying "face photo" manifold with dimensions describing semantic variations such a lighting, pose or identity. We would like to learn a generator  $G(y_i, z)$ ,  $z \sim p_z(z)$ .  $G$  learns to interpret  $z$  as a  $p_i$  and decode it as a pose / illumination / etc vector that doesn't include any identity information. Ideally,  $z$  is a probability distribution that is easy to sample from (eg standard normal). We could then reenact any identity  $y$  by sampling  $z$  vectors at will.

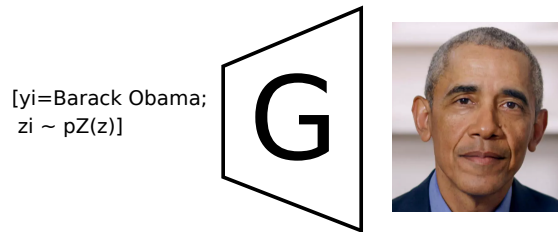


Figure 6.1:  $G$  is a generator that turns a person identifier  $y_i$  and a latent variable  $z_i$  into an image.

## 6.4 Generative Models

### 6.4.1 Fundamentals

When trying to generate data, we wish to model  $p(X)$  for any data distribution  $X$  in terms of its individual components. For instance, for image data, we learn to model images  $x \in \mathcal{X}$  by modeling the color probability distribution of each of its pixels  $x_{1..H,1..W}$ , by assuming independence.

$$(6.1) \quad p(X = x) = \prod_{i=1}^H \prod_{j=1}^W p(X_{i,j} = x_{i,j})$$

With such a simple model,  $p(X_{i,j})$  is left as an arbitrarily sophisticated or simple distribution of our choice, such as a categorical distribution over discretized pixels values (with parameters  $\theta_{i,j}$ )

$$(6.2) \quad p(X_{i,j}) = \text{Cat}(X_{i,j}; \theta_{i,j})$$

or Gaussian distributions over values (with mean parameters  $\mu_{i,j}$  and standard deviation parameters  $\sigma_{i,j}$ )

$$(6.3) \quad p(X_{i,j}) = \mathcal{N}(X_{i,j}; \mu_{i,j}, \sigma_{i,j})$$

Once the individual pixel probabilities parameters (ie  $\theta_{i,j}$  or  $\mu_{i,j}, \sigma_{i,j}$ ) have been estimated from data, one could sample a value for each pixel and get an image.

However, this modeling is trivial and would produce pictures that looks nothing like real images because it considers each pixel as independent and does not take into account patterns and spatial correlations.

## 6.4.2 Auto-regressive models

This components independence assumption leads to very poor results, especially in image generation. Instead of sampling each component independently, we could sample each component one after another, in any predefined order, based on some of the previously sampled values. In which case  $p(X_{i,j})$  becomes a distribution conditioned on the previous  $k$  components. The graphical model is illustrated in figure 6.2.

For image data, those individual components are pixels, and a full image is sampled pixel by pixel. Each sampling operation operates on a context window constituted of the previous samplings. As we consider bigger context windows  $k$ , the models needed become more complex and bigger and that is when Deep Learning comes into play. This image is progressively sampled following an ordered set of pixel coordinates  $\Psi$  (usually left to right and top to bottom, but not limited to).

$$(6.4) \quad p(X) = \prod_{i=0}^{|\Psi|} p(X_{\Psi_i} | x_{\Psi_{i-1}}, \dots, x_{\Psi_{i-k}})$$

This is the approach coined by PixelCNN [83], PixelRNN [156], PixelCNN++ [137] or PixelSNAIL [24]. At inference time, we sample pixels one by one, each requiring a model forward pass. This exhibits the major drawback of auto-regressive models for image synthesis: they require  $H \times W$  forward passes, making it extremely slow and computationally intensive. Moreover, as the images grow bigger, not only more forward passes are needed, bigger models are needed as well in order to grow their receptive fields and context windows accordingly. A single pass of PixelCNN is shown in figure 6.3.

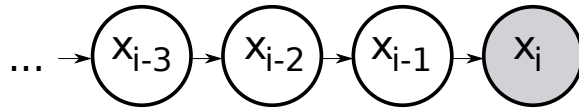


Figure 6.2: Conditional probability graph of an autoregressive model. Each pixel depends on the previous ones, iteratively.

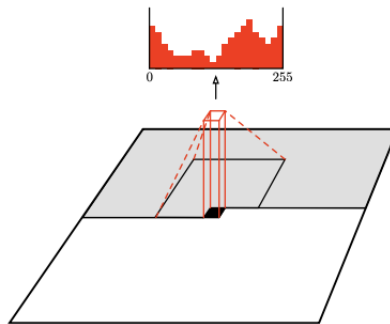


Figure 6.3: A PixelCNN sampling a pixel value for the current pixel from its surrounding context. White pixels are still undetermined grey pixels have already been sampled. Shown in red is the softmax output describing the probability distribution of the current pixel values conditioned on the context window. Image from Kolesnikov and Lampert [83]

Modern auto-regressive models such as the VQ-VAE [155] or Vector-Quantized GAN (VQ-GAN) [42] try working around this complexity by only sampling small pictures or small representations,

leaving the actual high quality rendering to another method, such as convolutional upsamplers, convolutional decoders, or GANs. The VQ-VAE will be described in section 6.5.4.

## 6.5 Latent-Variable Models and Variational AutoEncoders

### 6.5.1 Principles

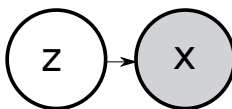


Figure 6.4: Conditional probability graph of a Latent Variable Model (LVM). The whole image  $x$  is sampled at once from a lower dimensional encoding  $z$ .

Instead of having a long chain of random variable dependency (ie the previous components), we can assume that there is a lower dimensional explicative random variable  $z \sim p_z(z)$ , and that a powerful function could decode from it all the components at once (compare figure 6.2 and 6.4). For images, this embodies the idea that the pixels of an image can be reduced to a much denser amount of semantic information such as "a child sitting on a bench and eating ice cream in a park" or a low dimensional feature vector. We can thus model the data probability density as the probability of a data point  $x$  decoded by all possible codes  $z$ :

$$(6.5) \quad p(x) = \int p(x|z)p_z(z)dz = \int p_z(z) \prod_{i=1}^H \prod_{j=1}^W p(x_{i,j}|z)dz$$

$$(6.6) \quad -\log p(x) = -\log \int p(x|z)p_z(z)dz$$

In our case, this code  $z$  is considered unknown and has to be discovered by the training procedure as well. We often chose  $z$  to be a continuous feature vector and  $p_z(z)$  to be a standard Gaussian as it is easy to sample from.  $p(X|Z)$ , called "decoder", generates the data components from the code. It usually is a neural network suited for the data type.

The integral inside Eq 6.6 can be rewritten as an expectation:

$$(6.7) \quad -\log p(x) = -\log \mathbb{E}_{z \sim p_z(z)}[p(x|z)]$$

The expectation outside the log is unfortunate: the log of the expected value would need many samples in order to be accurate, and all those probability multiplications would be numerically unstable.

Thankfully, Jensen's inequality gives us a useful lower bound:  $f(\mathbb{E}[x]) \leq \mathbb{E}[f(x)]$  for any convex function  $f$ . So, the log can be moved inside the expectation at the cost of a lower bound.

$$(6.8) \quad -\log p(x) \leq \mathbb{E}_{z \sim p_z(z)}[-\log p(x|z)]$$

Learning this model with Maximum Likelihood Estimate for large datasets is impractical as we would first need to sample a  $z$ , find a sample in the dataset that is best explained by the decoder for that  $z$ , and perform the MLE step.

The VAE [81] proposes to solve this with more neural networks. The simple fix is to train an "encoder"  $q$  that learns which  $z = q(x)$  explains  $x$  the best. We can then take a training sample, encode it to a  $z$ , decode it back, and optimize for reconstruction and penalize the log probability of  $z$  according to the prior  $p_z(z)$  as well.

This, however, would not make a good generative model as there is no incentive that the encoder covers the whole volume of  $p_z(z)$ . Instead of encoding  $z = q(z|x)$  as a deterministic mapping,  $q(z|x)$  can be turned into probability density parameters from which we can sample  $z$ . We can thus read  $z \sim q(z|x)$ , and instead of penalizing the log probability of  $p_z(z)$ , we penalize the Kullback-Liebler divergence  $D_{KL}(q(z|x)||p_z(z))$ . The KL divergence measures the dissimilarity between two probability distributions. With  $q$  being pushed to resemble the prior, we hope to enforce full utilization of the prior probability space.

Formally, if we use this surrogate distribution  $q(z|x)$  to ease smart sampling from  $p_z(z)$ , we are doing *importance sampling*, and get  $\mathbb{E}_{z \sim p_z(z)}[p(x|z)] = \mathbb{E}_{z \sim q(z|x)}[\frac{p_z(z)}{q(z|x)}p(x|z)]$ . Taking the log and applying Jensen's inequality, we obtain

$$\begin{aligned}
 -\log p(x) &= -\log \mathbb{E}_{z \sim p_z(z)}[p(x|z)] \\
 &= -\log \mathbb{E}_{z \sim q(z|x)}[p(x|z) \frac{p_z(z)}{q(z|x)}] \\
 (6.9) \quad &\leq \mathbb{E}_{z \sim q(z|x)}[-\log p(x|z) - \log \frac{p_z(z)}{q(z|x)}] \\
 &\leq -\mathbb{E}_{z \sim q(z|x)}[\log p(x|z)] + \mathbb{E}_{z \sim q(z|x)}[-\log \frac{p_z(z)}{q(z|x)}]
 \end{aligned}$$

This formalizes the Evidence Lower BOund (ELBO):

$$(6.10) \quad -\log p(x) \leq ELBO(x) = -\mathbb{E}_{z \sim q(z|x)} \log p(x|z) - D_{KL}(q(z|x)||p_z(z))$$

We usually interpret this loss as two terms that must be minimized: a reconstruction term and a prior divergence term. We aim to learn an encoder that produces representations whose distribution is similar to an isotropic gaussian, and a decoder that is able to decode any sample from the  $\mathcal{N}(0, \mathbf{I})$  prior into a realistic data sample. While this helps this intuitive explanation is the source of some misconceptions that are beyond the scope of this document.

### The reparameterization trick

It is important to note that sampling is a discrete operation. As  $z$  is sampled from  $q$ , it is not natural to learn  $q(z|x)$  by backpropagation. In order to be able to backpropagate into  $q$  we have to make it a differentiable operation.

In order to do so we model  $q$  as an isotropic Gaussian of  $D$  dimensions. We make the neural network modeling  $q$  output two sets of values:  $\boldsymbol{\mu}(z|x)$  and  $\boldsymbol{\sigma}(z|x)$ , ie the mean and standard deviation of each dimension of the Gaussian. We observe that sampling from  $\mathcal{N}(\boldsymbol{\mu}(z|x), \boldsymbol{\sigma}(z|x))$  is the same as sampling from  $\boldsymbol{\mu}(z|x) + \boldsymbol{\sigma}(z|x)\mathcal{N}(0, \mathbf{I})$ . Elementwise addition and multiplication are differentiable, therefore this form allows to backpropagate into  $q$ . This is known as the reparameterization trick.

### 6.5.2 Limits

It is to be noted that the shortcomings of the VAE are well known:

1. The KL term and sampling operation prevents the decoder from having an accurate latent variable to decode. Thus, the produced samples are notoriously blurry.
2. The reconstruction term and divergence term balance in counter-intuitive ways. The ultimate VAE goal is not to learn a meaningful latent vector but to assign the correct probability density to the data distribution. When possible, the encoder ignores the input sample, produces exactly the prior distribution (turning the KL term to zero), and decodes samples at random. This is to be expected, especially when powerful decoders are used.

### 6.5.3 Information Bottleneck

#### General concept

The reparameterization trick and the KL term in order to fit a noise distribution lead to the variational information bottleneck, used as a layer. Through this layer, only the information necessary for minimizing the task's loss would go through in a compressed way. All the unnecessary information would be eliminated to resemble the prior noise distribution.

#### Benefits as regularization

Alemi et al. [4] inspect how models with an information bottleneck generalize. It happens that those models are less prone to overfitting and adversarial attacks, and generalize better overall.

#### Information Bottlenecks In Latent Variable Models

This information bottleneck is also useful in tasks with multiple possible correct outputs. For instance, in image colorization (illustrated in figure 6.5), naive supervised training is unable to represent that many colors might fit an object, and the neural net would produce grey-ish pictures without vibrance, actually outputting the average color of the possible responses.

While GANs (Section 6.6) fit this issue, they come with their own difficulties. Instead, we can still use a Maximum Likelihood Estimate framework by extracting a latent variable from the target. An information bottleneck on that latent variable prevents the target to bleed through and ensure it only contains the information to complete the task that can't be deduced from the input.

At inference time, we can either use latent-variables extracted from predefined samples, sample from the prior noise distribution, or learn a prior from the extracted train latents (such as a Gaussian mixture model).

Designing an efficient information bottleneck is challenging: it must not leak any redundant or useless information, must not filter out the needed information, and it is interesting to be able to sample from it.

### 6.5.4 VQ-VAE

The VQ-VAE considers that a discrete latent variable could be used in place of a continuous one.

#### Architecture

The VQ-VAE [155] takes the VAE from another perspective. They propose to train an auto-encoder then, in a second stage, fit an auto-regressive model on the latent representation as a prior to sample from. In order to both ease the job of the prior network and control the amount of information that can be transmitted, the latent is encoded as discrete tokens.



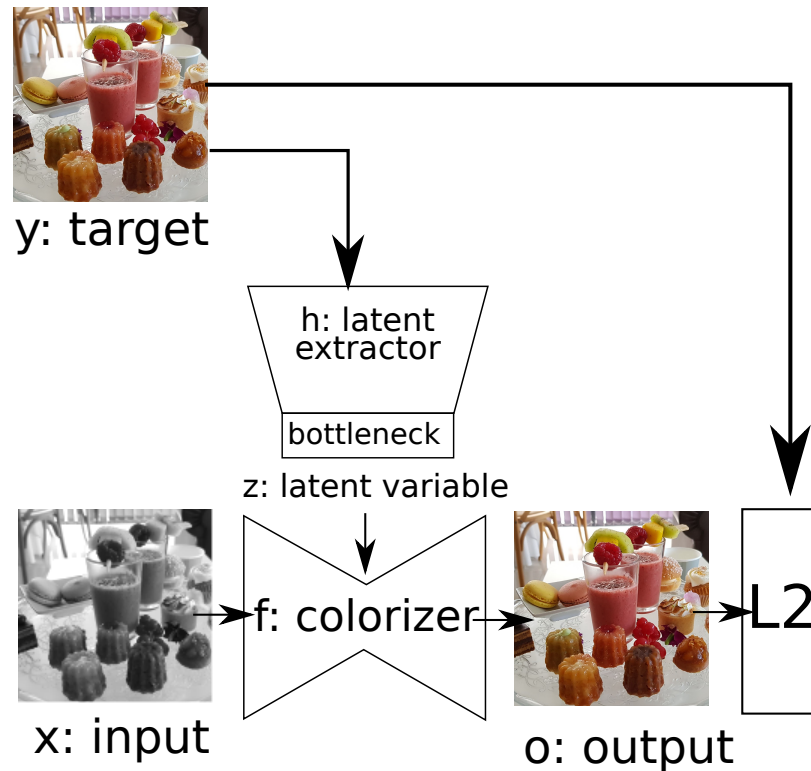


Figure 6.5: Training a latent variable model for colorization. There are multiple possible colorizations for a single greyscale input. A latent extractor  $h$  extracts the information solving the ambiguity between those multiple answers ; an information bottleneck prevents the latent extractor from encoding all of the target and short-circuiting the task. The colorizer  $f$  resolves ambiguous cases using the latent.

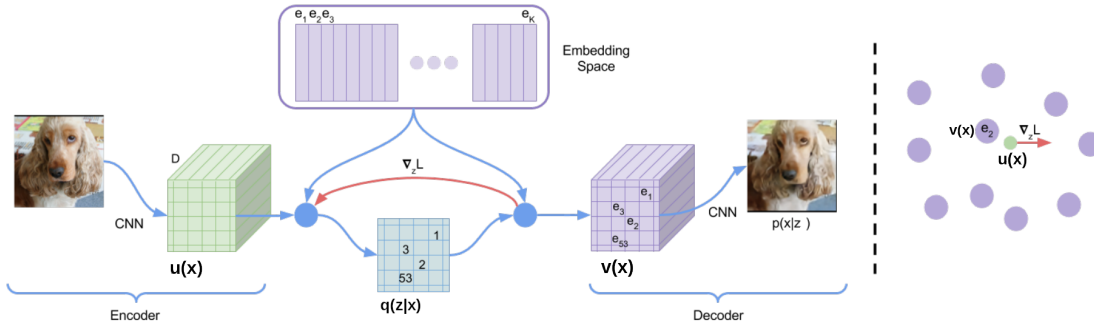
For image data, the prior network usually is a PixelCNN or a variation of it. The approach is summed up in figure 6.7 and Figure 6.6. A CNN auto-encoder with a VQ bottleneck is trained, then a prior model able to model discrete sequences (PixelCNN, LSTM, Transformer, etc) fits the latent distribution generated by the bottleneck.

From a VAE perspective, the encoder is (deterministic) categorical one-hot distribution and by defining our prior as a uniform categorical distribution, we obtain a KL divergence constant and equal to  $\log K$ ,  $K$  being the size of the codebook (better explained in the next section). This dispenses us from computing this term at all and it can be removed from the loss.

### Backpropagating through quantization

Besides those architectural novelties, the main contribution of [155] was to provide a backpropagation-friendly discretization operation.

In order to discretize, the quantization layer maintains a codebook  $e$  of  $K$  vectors,  $K$  being a hyperparameter. The continuous activation vectors (unquantized values)  $u(x)$  compared to the ones in the codebook. The closest code to each activation vector is selected, producing  $q(z|x)$ , the deterministic one-hot categorical distribution predicting the quantized value. Then, we produce the value  $v(x)$  replacing the unquantized vectors by their closest neighbor in the codebook, quantizing with a resolution



Left: A figure describing the VQ-VAE. Right: Visualisation of the embedding space. The output of the encoder  $u(x)$  is mapped to the nearest point  $e_2$ . The gradient  $\nabla_z L$  (in red) will push the encoder to change its output, which could alter the configuration in the next forward pass.

Figure 6.6: Figure from [155] developing the quantization process.

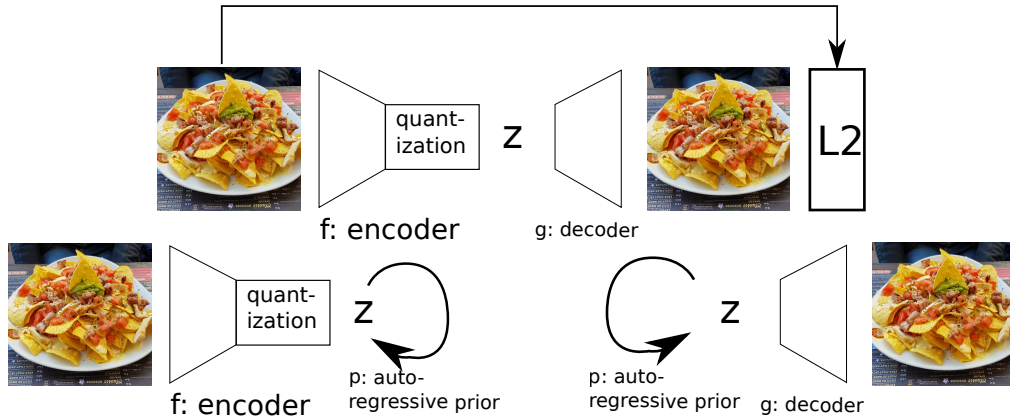


Figure 6.7: **top**: Training a VQ-VAE stage 1: a quantized encoder and decoder are trained in an autoencoding fashion. **bottom left**: Training a VQ-VAE stage 2: the encoder is frozen and an autoregressive prior is learnt on the extracted latents. **bottom right**: Sampling from a VQ-VAE: We generate a latent variable from the prior model and decode it to a full picture

of  $K$ . (See Figure 6.6).

$$(6.11) \quad q(z = i|x) = 1 \text{ if } i = \underset{j}{\operatorname{argmin}} \|u - e_j\|_2 \text{ else } 0$$

$$(6.12) \quad v(x) = e_k, \text{ where } k = \underset{j}{\operatorname{argmin}} \|u(x) - e_j\|_2$$

Since the operation is not differentiable, gradients have to be approximated manually.

1. A straight-through estimator is used. It is assumed that the gradients from the upper layers, computed from the quantized codes, are good approximations for the gradients of the pre-quantized, continuous values.

2. In order to keep this approximation relevant and learn the codebook, we move (in a L2 sense) each prototype towards the center of mass of the continuous vectors that were assigned to it. The prototypes follow the input values.
3. Finally, we reinforce the approximation and strengthen training dynamics. We add a "commitment" term encouraging the pre-quantized values to get closer (in a L2 sense) and aggregate around their assigned quantized prototype. The strength of this parameter is controllable through a parameter  $\beta$ . This value is defaulted to 0.25 and rarely touched.

The final VQ-VAE loss is

$$(6.13) \quad \mathcal{L} = \log p(x|v(x)) + \|\text{sg}[u(x)] - e\|_2^2 + \beta \|u(x) - \text{sg}[e]\|_2^2$$

We can reidentify, in order: the reconstruction term, the codebook update term, and the commitment term.  $\text{sg}$  indicates the "stop gradients" operator which zeroes partial derivatives.

## Benefits

As the latent variables usually have a lower dimensionality than the data points, it is faster to train and sample latents from an autoregressive model, than to train and sample from an autoregressive model on the data points directly.

The generated samples are also of much greater quality than ones of a standard VAE. First, The prior distribution is much more complex, hence much more expressive. Secondly, the component-by-component, conditional, sampling, instead of sampling all the latent at once like a standard VAE allows for a much more precise latent.

## As an information bottleneck

When designing a quantization layer in a neural network, we can choose how many codebooks and quantized values per codebook we want. This allows to set a very accurate and hard limit on the maximum amount of information that can be transmitted. For instance, with 8 codebooks with 32 codepoints, we can transmit exactly  $8 \log 32 = 8 \times 5 = 40$  bits of information.

## 6.6 Generative Adversarial Networks (GANs)

### 6.6.1 Principles

#### GANs as competition

Another completely different family of generative model are GANs. GANs do not model  $p(X)$  explicitly, neither do they compute the log likelihood, exact or approximated of the data. In the literature, GANs are presented as two neural networks competing against each other. A Discriminator ( $D$ ) learns to discriminate samples from the real data distribution  $p_{\text{data}}$  and the fake samples from distribution  $p_{\text{fake}}$  produced by a Generator ( $G$ ). The two networks are trained in an alternating and opposite objectives,  $D$  learning to discriminate better while  $G$  learns to fool  $D$  by gradient ascent. The optimal state is reached when  $p_{\text{fake}} = p_{\text{data}}$ .

While a lot of engineering went into designing better generators for image synthesis of various kind [75, 76, 78, 74], discriminators got most of the theoretical work as they provide the signal the generator trains against, and are the only components in contact with the true data distribution.

Those three training steps are illustrated in figure 6.8.

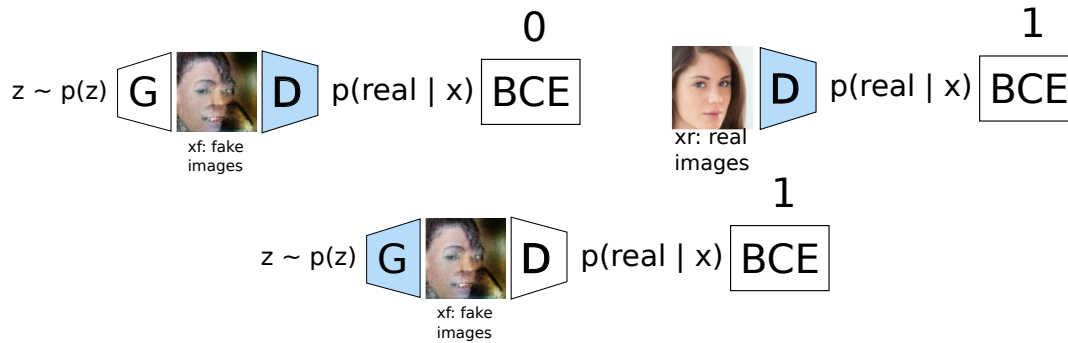


Figure 6.8: Training a standard GAN. **top left:**  $G$  is kept frozen, we teach  $D$  to classify a fake sample as a fake image with a BCE loss  $\text{BCE}(D(x_f, 0))$ . **top right:**  $D$  is taught to classify a real sample with  $\text{BCE}(D(x_r, 1))$ . **bottom:** we train  $G$  to produce images that are classified as true by  $D$  with  $\text{BCE}(D(x_f, 1))$ ,  $D$  is kept frozen.

### GANs as learnable loss

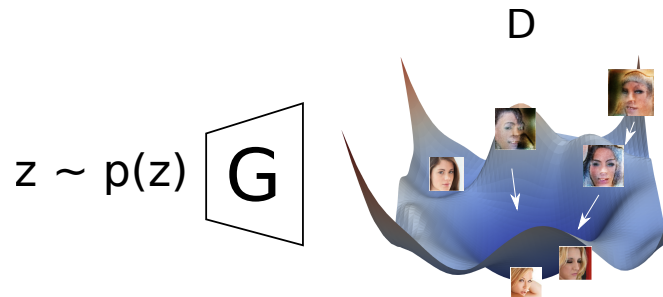


Figure 6.9: Interpreting  $D$  as a trainable loss giving low values to real samples and high values to fake samples.  $G$  learns to minimize the loss  $D$  represents. Gradients of fake samples represented as white arrows.

Alternatively, they can be viewed as a simple but rich idea: training a neural network as loss function, modeling the manifold of the data distribution. This loss-neural-network learns to give high logits to samples coming from the real data distribution and low logits to samples produced by a generator network. The generator networks is trained to maximize the discriminator's output, and convergence is reached when it perfectly mimics the data distribution, making a flat logits surface. As we shall see, correctly shaping this energy surface is of crucial importance and can make GANs simple to work with or very difficult to train. Figure 6.9 shows a generator learning to reduce the loss modeled by  $D$ .

### Formal Definition

An unconditional Generator learns a mapping  $G(z)$  from a distribution  $z \sim p_z(z)$  that is easy to sample from to the data distribution  $x \sim P_{\text{data}}(x)$  [49]. We call the distribution produced by  $G$   $p_{\text{fake}}$ . We aim for  $p_{\text{fake}} = p_{\text{data}}$  and often chose  $p_z(z)$  to be a standard Gaussian distribution.

It is often stated that  $G$  and  $D$  play a min-max game on the value function  $V$ .  $V$  has initially been defined like a binary cross entropy loss on  $D$ . However, instead of ascending the gradient on  $G$ , which would be really small if  $D$  makes confident choices,  $G$  is learned with gradient descent with reversed targets (referred to as Non-Saturating GAN (NSGAN)).

$$(6.14) \quad \min_G \max_D V(G, D) = \mathbb{E}_{x \sim p_{\text{data}}(x)} [\log D(x)] + \mathbb{E}_{z \sim p_z(z)} [\log(1 - D(G(z)))]$$

Goodfellow et al. [49] proved in the seminal paper that the optimal  $G$  for an optimal  $D$  mimics the data distribution perfectly and that the system minimizes the Jensen-Shannon (JS) divergence between  $p_{\text{fake}}$  and  $p_{\text{data}}$ .

$$(6.15) \quad \begin{aligned} JS(p_{\text{fake}}, p_{\text{data}}) &= \frac{1}{2} D_{KL}(p_{\text{fake}} || Q) + \frac{1}{2} D_{KL}(p_{\text{data}} || Q) \\ Q &= \frac{1}{2} (p_{\text{data}} + p_{\text{fake}}) \end{aligned}$$

The global optimum of the JSD is given by the Nash equilibrium reached when  $p_{\text{fake}} = p_{\text{data}}$  in the case of generator and discriminators of unlimited capacity and unlimited training data.

This game can converge to various points:

- $G$  is overpowered by  $D$  and generates poor results, sometimes leading to mode collapse;
- $D$  is overpowered by  $G$ ,  $G$  tries to satisfy  $D$  but cannot, and the samples are of poor quality;
- $G$  and  $D$  are both able to generate and learn the data distribution, the optimization process does not diverge, and  $G$  produces a distribution close to  $p_{\text{data}}$ .

Alternatively and more classically, one can view  $D$  as a classifier modeling  $p(\text{real}|x)$ , and training  $G$  is maximizing  $p(\text{real}|G(z))$ , using  $D$  as a differentiable loss. Several alternatives were proposed, such as a regression or a hinge loss for  $D$  instead of a BCE loss. Viewed as an energy based model, all those alternatives are similar as they train  $D$  to model a loss surface that  $G$  optimizes against.

## 6.6.2 Failures

GANs were said to have numerous problems

- Sensibility to architecture:  $G$  and  $D$  had to be symmetrical for one not to overpower the other, and they had to be carefully tuned
- Training collapse: one of the two networks can collapse and end the convergence, producing unrealistic samples (Figure 6.10).
- Mode collapse:  $G$  can collapse to a single output, often unrealistic.
- Rotational dynamics: mode collapse can be rotational as well, meaning that  $G$  moves from mode to mode as training goes.

Most of those difficulties are now mitigated thanks to gradient penalties introduced by WGAN-GP [51] and later improved into various regularizers such as R1 [108] or R0 [151]. They all bear the same idea: control the Lipschitzness of  $D$ , aka its smoothness, to prevent strong gradients and give  $G$  an easy and stable descent into the loss surface.

## 6.6.3 Advances in GANs

### Wassertstein distance

Instead of optimizing the JS divergence which suffers from vanishing gradients and suboptimal behavior that are developed in Arjovsky et al. [7], it has been proposed to use the Wassertstein distance between



Figure 6.10: An example of GAN training collapse. The generated samples suddenly ceases converging towards realistic samples, and the GAN never escapes this degenerate state. Image source: <https://www.mathworks.com/help/deeplearning/ug/monitor-gan-training-progress-and-identify-common-failure-modes.html>

two distributions  $p_a$  and  $p_b$  instead, noted  $W(p_a, p_b)$ . Also called "Earth-Mover Distance", it represent the optimal cost of transporting the probability mass to transform one distribution into another.

This requires complex transportation algorithms to solve in low dimensionality and becomes intractable in high dimensions. Instead, Arjovsky et al. [7] devise a variational approach using the Kantorovich-Rubinstein duality [160]:

$$(6.16) \quad W(p_a, p_b) = \sup_{\|f\|_L < 1} \mathbb{E}_{x \sim p_a}[f(x)] - \mathbb{E}_{x \sim p_b}[f(x)]$$

That is, for a function  $f$  that has a maximum Lipschitzness of 1 and gives the highest (lowest) possible scores to the samples from  $p_a$  ( $p_b$ ), the Wasserstein distance between two distributions is the difference of the average score for each distribution.

### Lipschitzness

The *Lipschitzness* of a function  $f$  is the maximum L2-norm of its gradient. We say that  $f$  is K-Lipschitz if its Lipschitzness is equal to or less than K.

$$(6.17) \quad \text{Lip}(f) = \max_x \|\nabla_x f(x)\|_2$$

### Wasserstein GAN (WGAN)

This variational approach makes it very convenient to use a neural network as  $f$  that would serve as a Discriminator.  $f$  would be trained to maximize its score on real samples and minimize it on fake

samples, which is a trivial task for today’s neural networks. However, the way to enforce the Lipschitz constraint is not trivial.

WGAN [7] proposes as a first rough solution to clip the weights of  $D$  to small absolute values. Thus, the value function optimized is:

$$(6.18) \quad \min_G \max_D V(G, D) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[D(x)] - \mathbb{E}_{x \sim p_z(z)}[D(G(z))]$$

### WGAN-GP

WGAN-GP [51] approximates the Lipschitzness by measuring the L2 gradient norm of  $D$  on a linear path from real samples to fake samples.

$$(6.19) \quad \begin{aligned} \text{Lip}(D) &\approx \mathbb{E}_x \|\nabla_x D(x)\|_2, \text{ where} \\ x &= \alpha x_r + (1 - \alpha)x_f \\ \alpha &\sim \mathcal{U}(0, 1) \\ x_r &\sim p_{\text{data}} \\ x_f &\sim p_{\text{fake}} \end{aligned}$$

From this, they devise the 1-GP regularizer:  $R_{1\text{-GP}}(D) = (\text{Lip}(D) - 1)^2$ . It encourages  $D$  to be 1-Lipschitz.

### Spectral Normalization GAN

Another approach has been introduced in Spectral Normalization GAN (SNGAN) [111], by bounding the Lipschitzness of  $D$  by controlling the spectral norm of the weight matrices in  $D$ . While computationally cheap, this approach comes with its own set of issues, such as spectral collapse [19], sometimes provoking training collapse, for which a regularizer has been proposed [99] but diminishing the computational costs benefits of the approach.

### R1 regularizer

Mescheder et al. [108] exhibits that  $R_{1\text{-GP}}$  brings rotational dynamics that slows down or totally hinders convergence. The system would oscillate around the convergence point as the gradients do not effectively point towards it but spiral around it. They present the  $R_1$  regularizer that flattens the surface around real data points, effectively turning them into attractive points. Figure 6.11 illustrates a regularized vs an unregularized loss landscape.

$$(6.20) \quad R_1(D) = \mathbb{E}_{x \sim p_{\text{data}}} \|\nabla_x D(x)\|^2$$

This strategy was successful enough to be used in and make the glory of StyleGAN [76]. Though, this regularizer does not enforce anything about  $D$ ’s Lipschitzness and diverges from the Wasserstein GAN framework.

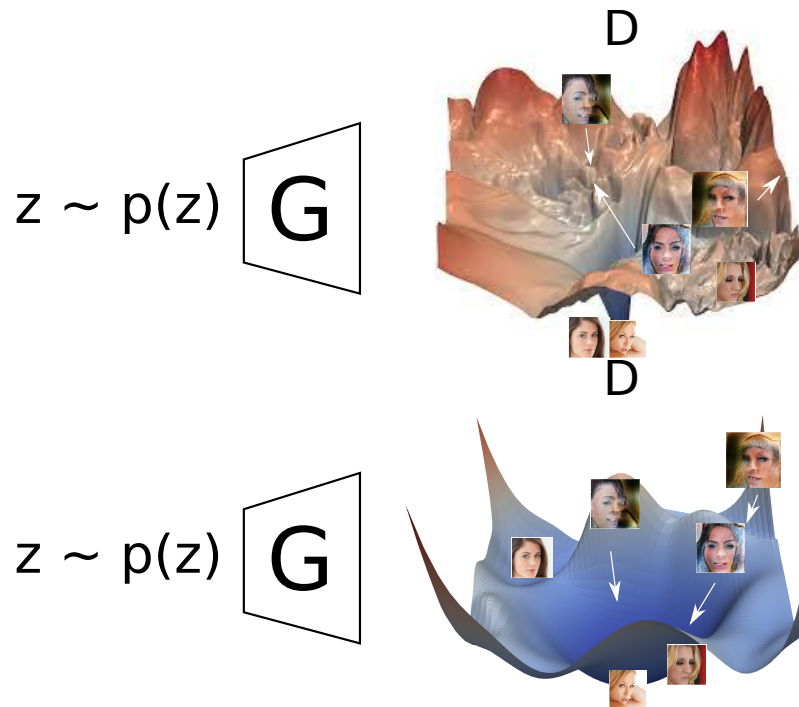


Figure 6.11: Effect of regularizers. **Top:**  $D$  is trained without a regularizer. The loss landscape might be noisy and hard to optimize against. There are strong peaks and valley because of the unregulated Lipschitzness. **Bottom:**  $D$  is trained with R1 or WGAN-GP regularizers, smoothing the surface around real data points or just controlling  $D$ 's Lipschitzness. The gradients are more predictive of the correct optimization direction, the loss is easier to optimize against, the peaks and valley are smoother than the unregulated version. Note: these surfaces are just for illustrative purposes and are not visualizations of actual loss surfaces.

### Beyond Wasserstein

Despite paving the way towards reliable GAN convergence, the WGAN is not the pinnacle of training algorithms. As shown in Lucic et al. [104], no loss for  $D$  can ensure proper convergence. Qin et al. [125] shows that any loss would work with a good Lipschitz regularization as those loss functions would be constrained in a linear regime anyway. This explains why Mescheder et al. [108], despite not being rooted in WGAN, shows better theoretical and empirical convergence than WGAN-GP's 1-GP regularizer. Thanh-Tung et al. [151] pushes this idea further for greater generalization by flattening the path from real samples to fake samples. Those works continue investigating further regularizations with success.

### Image Synthesis

Advances specific to image synthesis were mostly brought in the form of architectural refinements in  $G$ , starting from the Deep Convolutional GAN [126], residual GANs introduced with SNGAN [111], progressively grown GAN [75], or with multiscale noise inputs and adaptive scaling [122, 76, 78] as seen in fast style transfer [47].



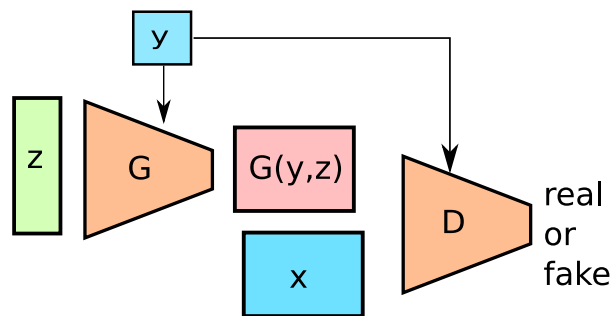


Figure 6.12: A cGAN. The discriminator and generator are both conditioned on  $y$ .

## 6.7 Conditional Modeling

The GANs seen so far are unconditional. Without some more machinery, there is no way of deciding what to produce. What if we trained a GAN on ImageNet and produce pictures of a specific class? What if you want to specify whether our medical images generator has to produce benign or malign tumors? What if we want to control what identity to produce with our face generating GAN? Those models are conditioned on some variable.

Conditional generative models don't learn to replicate the full, unconditional  $p(x)$  but instead learn to reproduce  $p(x|y)$  with condition  $y$ , usually a class label or another data point. For instance, one might want to change pictures of satellites views into schematics for maps services or colorize edge sketches.

In most conditional techniques,  $x$  and  $y$  are known, and the mapping is unknown. For this reason, they are often called "supervised" techniques since there is an input and at least one known target output.

*Generative modelling allows to model not a single output but a distribution of outputs.* There is more than one way to smile, and there are many possible pictures with class label "car". Common supervised techniques fail to acknowledge those situations.

### 6.7.1 Conditional GANs (cGANs)

#### cGANs

Conditional GANs were first introduced by Mirza and Osindero [110]. They propose to concatenate the condition  $y$  to the noise vector  $z$  in  $G$  and  $y$  to the fake images  $x_f$  and real images  $x_r$  in the Discriminator (see Figure 6.12). That way,  $D$  learns to discriminate whether  $x$  and  $y$  are in accordance and  $G$  is taught to produce data points  $x_f$  in accordance with  $y$ .

In Mirza and Osindero [110],  $y$  is a one-hot class label encoding. They generate class conditioned MNIST and CIFAR samples.

#### Pix2Pix

Isola et al. [72] conditioned images based on images and was highly successful at supervised image translation, that is, *transforming pictures from one domain to another*. Back then, gradient penalties were unknown and Lipschitzness was not a concern, thus Pix2pix and its evolution Pix2PixHD [166] had to bake in several stabilization techniques and convergence helpers.

First, Pix2Pix restricts the discriminator's receptive field so that it sees patches of the image only and produces a real/fake signal per patch. They name this approach *PatchGAN* and *Patch Discriminator*.

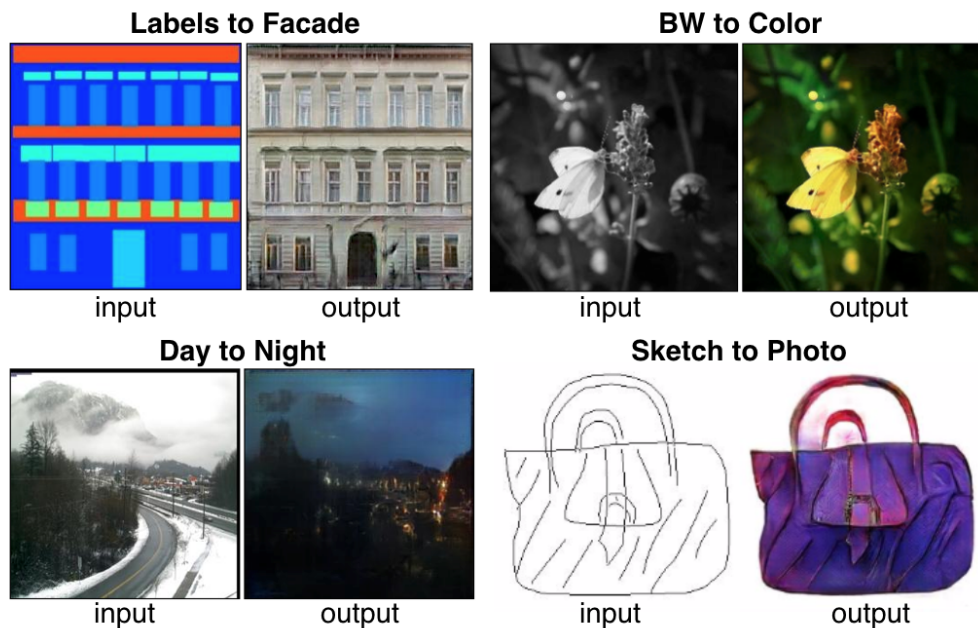


Figure 6.13: Examples of image translation from the original pix2pix paper [72].  $x$  is a real image,  $y$  a label, and  $G(y, z)$  a fake sample produced by the generator.

This design change provides multiple benefits:

1. this forces the discriminator to learn more about textures and patches rather than discriminating on global coherence;
2. it simulates a bigger training set since each patch is seen independently, fighting against discriminator overfitting.

Then, they add a L1 pixel loss to the adversarial loss, in order to tackle low frequency supervisions and large image regions bigger than a patch. This also helps guiding the generator, stabilizes training, and helps converging to better results than this unregularized discriminator alone would. The generator is based on a U-Net architecture with skip connections.

Figure 6.13 shows examples from the original paper.

**Pix2PixHD** also shows the ongoing research on generator architectures and uses a ResNet based generator instead. Besides, they change the NSGAN loss to a Least-Square GAN (LSGAN) loss, which replaces the BCE targets with MSE targets. They found this loss to be more stable. They also replace the L1 pixel loss with a feature matching loss, matching deep features of the discriminators under a L1 constraint. They use 3 patch discriminators working on images resized to different sizes, and some more subtle differences.

## BiGAN

Unconditional GANs show that they have a semantic interpretation of their latent variable  $z$ . BiGAN [38] proposes to jointly learn a generator  $G : z \mapsto x$  and an encoder  $E : x \mapsto z$  using a conditional discriminator  $D$  that learns to discriminate  $D(z, G(z))$  against  $D(E(x), x)$  (Figure 6.14). They prove that  $D$  can be fooled only if  $G = E^{-1}$ . They then use  $E$  as a feature extractor.

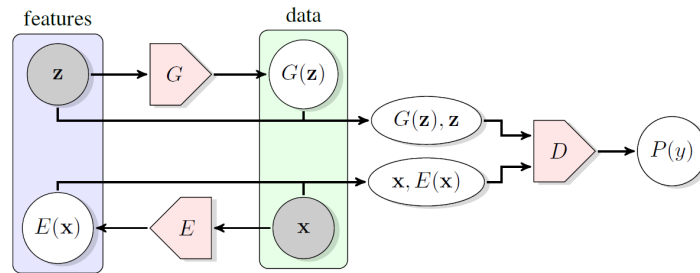


Figure 6.14: In BigGAN,  $G$  generates samples from features and  $E$  generates features from samples. Both pairs are discriminated, forcing  $G$  and  $E$  to reciprocate each other. Figure from [38].

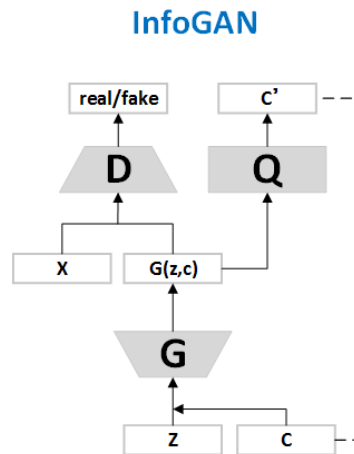


Figure 6.15: In the InfoGAN, the generator is fed with a random noise  $z$  and random categorical and continuous random codes  $c$ . The discriminator pushes the generator towards real samples.  $Q$  tries to guess  $c$  and  $G$  cooperates, ideally leading to  $G$  utilizing  $c$  in an interpretable way so that  $Q$  can identify them back in the generated samples. Figure from [97].

## 6.7.2 Conditional VAEs (CVAEs)

Sohn et al. [143] introduced a CVAE, aiming to learn a conditional decoder  $p(x|z, y)$ . They replace the standard VAE encoder  $q(z|x)$  with  $q(z|x, y)$  and decoder  $p(x|z)$  with  $p(x|z, y)$  (see Section 6.5 for details on VAEs). The conditioning variable  $y$  can be changed at will to control the produced samples.

## 6.8 Constrained Modeling

Sometimes though, the pairs  $(x, y)$  are unknown or the end goal is not suitable for conditional modeling. In those situations, it is possible to use constrained modeling, that is, generative modeling with additional constraints represented as additional loss terms. The generator must then compromise between the distribution matching (loss enforced by the discriminator) and satisfy the additional constraints losses. Sometimes, the constraints and distribution matching do not share a common minimum.

### 6.8.1 InfoGAN

One such example is the InfoGAN [23] aiming to learn a controllable generator with disentangled input features. They learn a generator  $G(z, c, f)$  with  $z \sim p_z(z)$  the latent variable distribution,

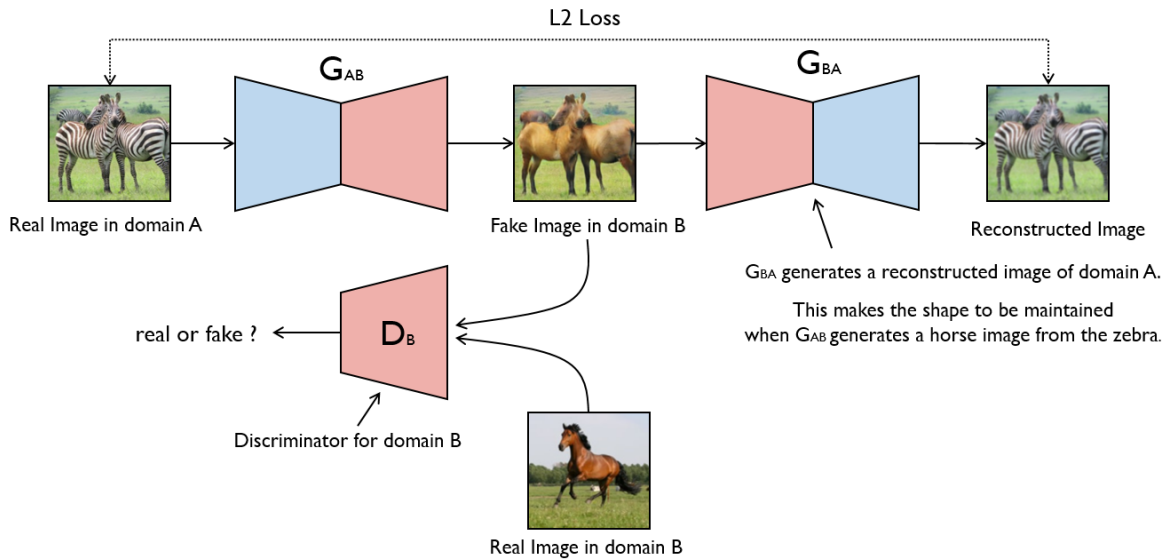


Figure 6.16: The CycleGAN architecture. Image from <https://towardsdatascience.com/image-to-image-translation-using-cyclegan-model-d58cfff04755>

$c \sim C$  with  $C$  a user-chosen distribution of categorical random variables, and  $f \sim F$  with  $F$  a user-chosen distribution of continuous random variables.  $G$  is trained against an unconditional discriminator  $D(x)$  like in unconditional modeling, and another recognition network aiming to guess  $c$  and  $f$  from the generated sampled.  $G$  and  $Q$  collaborate in order to maximize the mutual information  $I(Q(c, f|G(z, c, f)); G(z, c, f))$  (See Figure 6.15). We aim for  $G$  to learn to use  $c$  and  $f$  as discoverable latent variables in the generative process, while enforcing the generated samples distribution  $p_{\text{fake}}$  to be similar to  $p_{\text{data}}$ .

$$(6.21) \quad \min_G \max_D V_I(D, G) = V(D, G) - \lambda I(c; G(z, c))$$

$Q$  can then be used as a disentangled features extractor, or  $G$  as a controllable generator.

## 6.8.2 CycleGAN

Zhu et al. [184] wanted to take Pix2Pix one step further and perform image translation *in case pairs are not available*. We have real data points from real distribution  $x_{a, \text{real}} \sim A$  and real data points from real distribution  $x_{b, \text{real}} \sim B$ . We wish to learn a generator  $G_{A \rightarrow B} : x_{a, \text{real}} \mapsto x_{b, \text{fake}}$ . They propose to learn two generators,  $G_{A \rightarrow B}$  and  $G_{B \rightarrow A}$ , which respectively perform distribution matching against their own discriminator, respectively  $D_B$  and  $D_A$ . This alone would ensure that both generators would produce realistic samples of their target distribution. However, we also want the output to bear some similarity with the input. This additional constraint is added as a cycle loss aiming for  $x_a == G_{B \rightarrow A}(G_{A \rightarrow B}(x_a))$  and  $x_b == G_{A \rightarrow B}(G_{B \rightarrow A}(x_b))$  and is modeled with a L2 pixelwise similarity constraint. See Figure 6.16.

While being a breakthrough, CycleGAN exhibits two major flaws. First, the L2 pixel-wise similarity constraint prevents the GAN from performing geometric heavy changes. Second, the cycle loss prevents any transformation that would lose information. For example, CycleGAN can't be used correctly for sunglasses removal as removing the sunglasses in a convincing way would make it impossible

to recreate the exact same glasses to complete the cycle. In those situations, CycleGAN adds artifacts in order to be able to complete the cycle.

### 6.8.3 Contrastive Unpaired Translation

CUT [123] used contrastive learning to enforce the similarity constraint and improve on CycleGAN. They have a generator  $G_{A \rightarrow B}$ , a discriminator  $D_B$  that ensures realistic samples of  $B$  are produced, and a matcher  $M$ .  $M$  compares real and fake pairs of source image patches and output image patches with a contrastive loss,  $G$  cooperates to ensure  $M$  gets low loss.

This allows destructive transformations as there is no cycle loss, and allows shape transform as the constraint is not enforced on pixels but on deep features.

## 6.9 Controlled Unconditional GANs

Finally, while unconditional GANs are supposed to decode a random noise into a sample ( $x = G(z)$ ), a growing set of work focus on deciphering the random noise space. It has been observed that the generators actually organize the random noise input into semantically meaningful information [38, 37].

*These methods would enable reusing massive GANs that are expensive to train, such as StyleGAN2 [78] or BigGAN [19], to fit different scenarios.*

Some works try to learn, a posteriori, a mapping from labels to latent subspaces, from few shots. This allows to create a controlled label-conditioned GAN from less annotated data than needed by a conditional GAN. Shen et al. [139] learn latent directions from labels. Härkönen et al. [56] discovers disentangles directions from self-supervised learning.

Some other work focus on casting back real samples to the latent space, called *GAN inversion*, by modeling  $z = G^{-1}(x)$ . They then transform the latent vector, and decode it back, in order to transform the original sample. Some optimize  $z$  from a single sample, like Creswell and Bharath [27] while others like Guan et al. [50] learn encoders.

A fair amount of work in this framework tackles the inversion problem in order to make  $G(G^{-1}(x))$  as close to  $x$  as possible, and into identifying semantically meaningful and disentangled latent directions. Concretely, in the context of a face generation GAN, let  $f_{\text{smile}}(z)$  be a function that maps a latent vector into its smiling face counterpart. The method is  $G(f_{\text{smile}}(G^{-1}(x)))$ , and the challenge lies into creating good algorithms to provide  $G^{-1}$  and  $f_{\text{smile}}$ . Several transformation methods have been proposed. Brock et al. [18] allow painting a target result by optimizing a per pixel L2 loss, Shen et al. [140] identify latent directions with an auxiliary attribute classifier, Roich et al. [133] first optimize  $z$  then tune  $G$  for better reconstruction with this  $z$ .

## 6.10 Evaluation

Evaluating GANs is difficult and must account for two key elements: image quality and distribution matching. In unconditional GAN, the Kernel Inception Distance (KID) [15], Fréchet Inception Distance (FID) [64], and slightly obsolete Inception Score (IS) [152] are used to evaluate both elements at once. Those can also be evaluated separately with metrics such as the Precision / Recall developed by Kynkäänniemi et al. [89]. Unfortunately it is quite unclear as of today how to evaluate a conditional GAN, especially in the unpaired setting.

### 6.10.1 FID

The Fréchet Inception Distance gained a lot of traction to evaluate image GANs. It works by fitting a multivariate normal distribution on the output vectors of an Inception-V3, encoding the real and generated images, and computing the Fréchet distance [41] between both. For two Gaussians  $X$  and  $Y$ , the Fréchet distance is expressed as

$$F(X, Y) = \|\mu_X - \mu_Y\|^2 + \text{tr} \left( \Sigma_X + \Sigma_Y - 2\sqrt{\Sigma_X \Sigma_Y} \right)$$

where  $\mu$  and  $\Sigma$  are the mean and co-variance of the subscripted Gaussian. This captures both the realism of the generated images and the coverage of the modes and variance of the real distribution.

While the FID is widely used, it has some drawbacks. It is biased and as such limited for small datasets, is not easily interpretable, and is meant for evaluation of unconditional GANs.

### 6.10.2 KID

The Kernel Inception Distance measures the asymmetry between two distributions of samples. Contrarily to the FID, it does not assume a parametric form and has a different mathematical expression that makes it unbiased. It uses the polynomial kernel  $k(\mathbf{x}, \mathbf{y}) = \left(\frac{1}{d}\mathbf{x}^T\mathbf{y} + 1\right)^3$ , where  $d$  is the number of dimensions of  $\mathbf{x}$  and  $\mathbf{y}$ , which are the Inception representations of the images. Not having to compute this metric on 50k samples as is traditionally done for the FID makes it more suitable for smaller datasets. It has been recently used in pair with the FID for comparing methods.

### 6.10.3 Precision / Recall

Perhaps more relevant to our work is the precision / recall metrics developed by [89], that separate the evaluation of image quality and target distribution coverage.

Both those metrics use a density estimation technique to approximate the manifold of the real images and generated images. The ratio of generated images included in the real manifold is called precision and correlates with image fidelity. The ratio of real images included in the fake manifold is the recall and expresses how much of the real data has coverage in the fake manifold.

In order to estimate a manifold, we draw a sphere from each sample to its  $k$ -th nearest neighbor. That is, its radii is the distance to the current point to its  $k$ -th nearest neighbor. A point inside one of those spheres is considered inside the manifold. For image data, we don't use the pixel values but deep features from a VGG or inception network and usually set  $k = 2$  or  $k = 3$ . Figure 6.17 illustrates this algorithm.

Precision reflects the image quality without accounting for the distribution difference in the conditional setting. Recall measures how much of the training data is covered by the generated distribution.

All those metrics are implemented in our Torchélie library (presented in Chapter 8).

## 6.11 $\Rightarrow$ Contributions: Expiration for Vector-Quantized (VQ) codebook

**In a previous contribution** , Łańcucki et al. [90], we observed that VQ-VAEs fail to manipulate efficiently the entirety of the quantized vectors available, some remaining unoptimized for the entirety of the training procedure. In this earlier work, we explored different initialization strategies or periodic resampling of quantized vectors by k-means.

**We now propose** , in this section, a simpler and lightweight algorithm that even allows choosing the entropy of the quantized vectors usage.

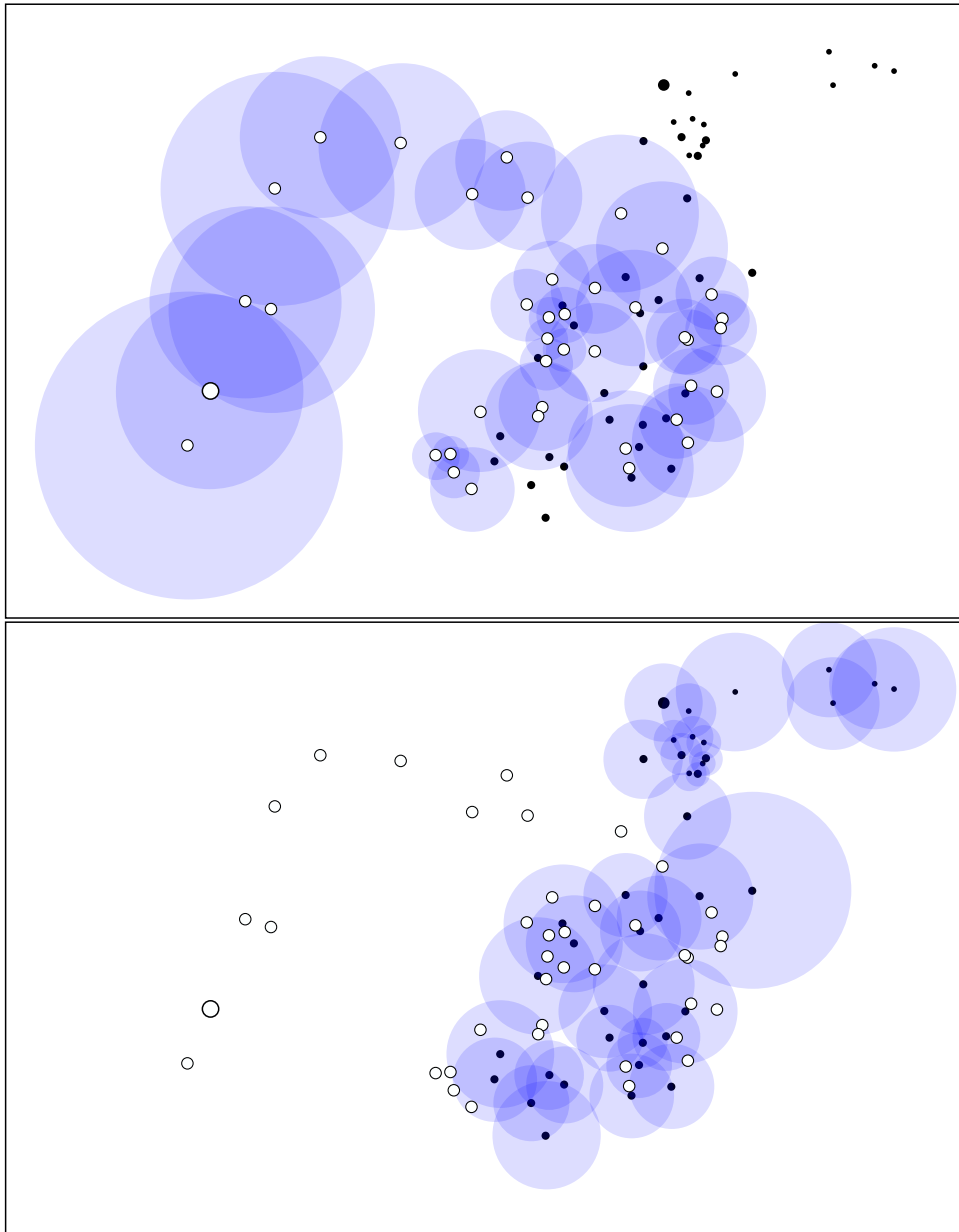


Figure 6.17: Precision/Recall estimation: white dots are 2D representations of generated samples and black dots are real samples representations. Top figure shows the fake manifold estimation (in blue), the ratio of black dots inside the manifold shows the recall, ie, the ratio of the real dataset covered by the generator. Below, we show the manifold of the real dataset. The ratio of white dots inside the blue zone is the precision, ie, the ratio of generated samples that look like real samples. The spheres are drawn from each point in the manifold to estimate to its  $k$ -th nearest neighbor. For those visualization we set  $k = 2$

### 6.11.1 Principles

It became quickly clear that the quantized vectors in the codebook are updated only via weight decay or receive gradients only if input vectors are quantized to them. If initialized improperly, a significant part of the codebook might not be ever used and just lost. Some codes might be lost as well during training if the updates of the codebook and input vectors get out of synchronization.

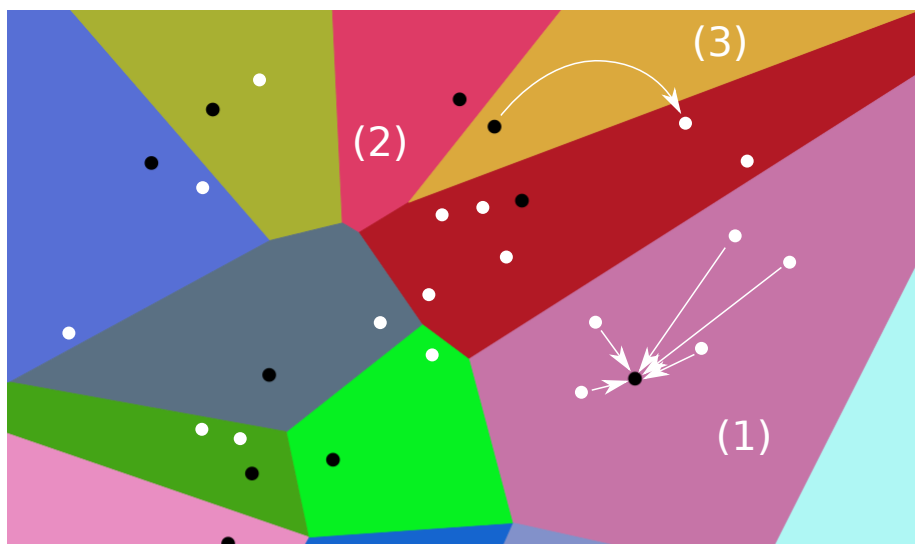


Figure 6.18: Vector Quantization illustration. Black points are codebooks prototypes. They divide the space into Voronoi cells. White points are input vectors, quantized to the prototype of the Voronoi cell they fall in. (1) shows the commitment loss as white arrows, bringing the input vectors closer to the prototype they have been assigned to. The prototype in cell (2) is not used in this iteration, its unused age is incremented. When, like in cell (3), that prototype has not been used for too long (more iterations than `limit`), it is resampled to a random input vector and its age is reset to 0.

To overcome this issue and ensure a full usage of the codebook, and thus of the available bandwidth, Łańcucki et al. [90] chose to resample every so often the codebook based on  $k$ -means centroids of the previous input vectors. Others have proposed continuous relaxation [144] or soft assignments [135].

Instead, we propose a simpler algorithm. Prototypes that have not been used for more than `limit` iterations are said to *expire* and are resampled to a random vector of the current batch. This allows to directly set a lower bound on the entropy of the assignments: a lower expiration limit will push towards uniform assignments while a higher limit would allow for stronger unbalance and preferences. Figure 6.18 illustrates this resampling operation, Listing 6.1 is a pseudo-python simple implementation.

Alternatively, we call *age* the number of iterations spent since last resampling.

Our approach is implemented in Torchélie (Chapter 8) and supports distributed training.

### 6.11.2 Experiments

We run several experiments in order to demonstrate that our VQ with expiration has better training dynamics than the vanilla version of van den Oord et al. [155]. That is, we expect our experiments to converge faster, show greater performance, and / or exhibit better utilization of the codebook.

Following the ideas in van den Oord et al. [155], we auto-encode 128x128 Imagenette [68].

#### Settings.

The encoder and decoder are fully convolutional. Each encoder layer  $LN$  contains a  $3 \times 3$  convolution with  $N$  output channels, a batchnorm, and ReLU. MaxPool is noted  $M$ . The encoder full architecture is L64-M-L128-M-L256-L256-M-L512-L512-M. The decoder is L512-U-L512-U-L256-U-L128-U-L64-L3 with U the bilinear upsampling operation; the last layer does not uses batchnorm and replace ReLU with a sigmoid activation. Between the encoder and decoder, the activations are quantized.



```

1 class VQ(nn.Module):
2     """
3     Quantization layer from *Neural Discrete Representation Learning*
4     Args:
5         latent_dim (int): number of features along which to quantize
6         num_tokens (int): number of tokens in the codebook
7         limit (int): maximum number of iterations before unused codepoints
8             get resampled.
9     """
10    def __init__(self, latent_dim: int, num_tokens: int, limit: int):
11        self.codebook = Array(num_tokens, latent_dim).gaussian_init()
12        self.age = Array(num_tokens).fill(limit)
13        self.limit = limit
14
15    def forward(self, x: torch.Tensor):
16        if self.training:
17            for i in range(len(self.age)):
18                if self.age[i] => self.limit:
19                    self.codebook[i] = random.choice(x)
20                    self.age[i] = 0
21
22        quantized, used_indices = quantize(x, self.codebook)
23
24        if self.training:
25            for i in range(len(self.age)):
26                self.age[i] += 1
27
28            for i in used_indices:
29                self.age[i] = 0
30
31        return quantized

```

Listing 6.1: VQ with expiration (pseudo python)

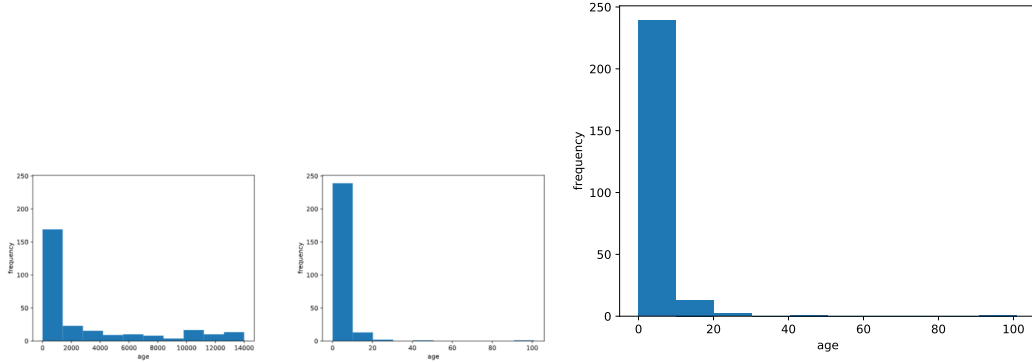


Figure 6.19: Histogram of age (time since last use) of each VQ layer codepoint after 20k training iteration. *left*: Without the expiration process, the optimization is harder and the net fails to use the codebook to optimize the loss. A lot of the codes remained unused for at least 2k iterations, presumably dead. *right*: Expiring and resampling code allows for exhaustive use of the codebooks and controllable entropy. Even if the maximum age is set to 250 iterations, the codebook has a much lower age on average.

This encodes input images into a spatial map of 8x8 codes. The number of available codes varies through experiments.

### Metrics.

We evaluate the proposed algorithm under various codebook sizes, in both training and testing. Perplexity is used to estimate the codebook usage, as well as the age of the different codes. Perplexity is defined as  $PP(p) = 2^{H(p)}$  ( $H(p)$  is the entropy function) where a perplexity of  $k$  indicates an entropy similar to the one of a  $k$ -way discrete uniform distribution. Finally, the influence on the test loss is considered as well; the test set contains 512 pictures.

### Training and age.

Figure 6.19 shows the age of the code points after 20k training iterations (50 epochs). As expected, with expiration, the codes have all been recently utilized with a median age of 0 and a mean age of 4. Even the oldest code is 150, less than the expiration period (250 iterations). Without the expiration strategy, and despite the Batch Normalization preceding it [90], most code points have not been used at all, showing an age of 20k. In this situation, the bottleneck is actually much stronger than expected, making it hard to design and reason about its size.

Figure 6.20 displays the loss and PPL with a ReLU instead of a batch normalization layer before the quantization. Our expiration process allows to quickly replace the codes initialized in the negative area. Our study [90] showed that initializing carefully the codebook and inserting a BN layer before the quantization improved the model. This present work shows that the Expiring VQ allows an even greater robustness to initialization and architecture, being more error tolerant as well.

### One codebook of size N

In the next series of experiments, the bottleneck is reduced to a single codebook of N code points, varying N. The quantized vector has 8 dimensions. We budget each experiment with the number of iterations needed for the expiration strategy to converge. Figure 6.20 shows the test loss and PPL for increasing codebook size.

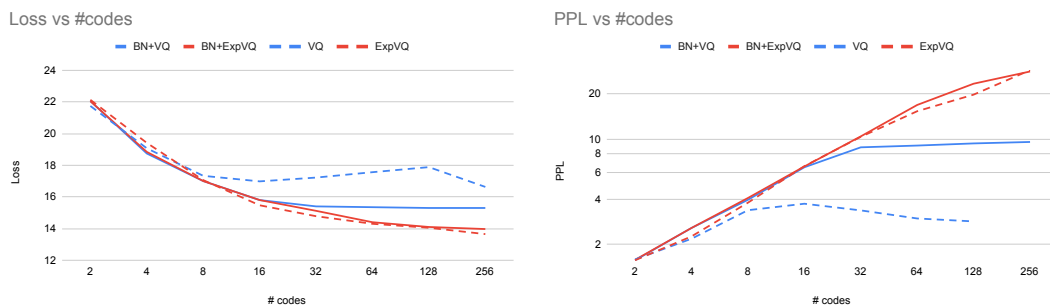


Figure 6.20: Experiment comparing test loss (left) and codebook usage Perplexity (PPL) (right) with a ReLU layer before quantization. Expiration VQ achieves lower loss and the perplexity scales correctly.

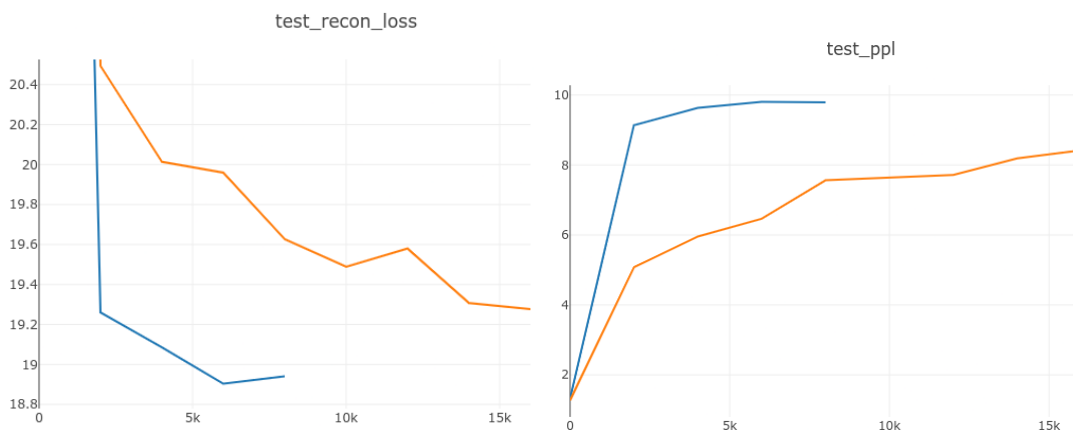


Figure 6.21: Experiment comparing test loss (left) and codebook usage PPL (right) for a codebook of 32 code points. Horizontal axis: training iterations, blue: VQ with expiration, orange: VQ without expiration.

**For small codebooks.** A deeper experiment for 32 code points shows in Figure 6.21 that, with a relatively small codebook, more training without expiration manages to gradually recover full usage of the codebook, although much more slowly than its expiration counterpart. In fact, training twice as long does not suffice to reach the the same loss.

**For big codebooks.** Taking this experiment to more code points yields a different result: most of the codes remain unused, and, contrarily to the previous results, are not "recovered" thanks to more training iterations. In this situation, the expiration strategy becomes necessary in order to control the effective bottleneck size and avoid wasting unused parameters.

**Follow up.** We hypothesize that those results are amplified for dimensions greater than 8 because of the curse of dimensionality, but this is still to be verified experimentally.

### 6.11.3 Conclusion

We proposed a simple and lightweight algorithm that allows setting a lower bound on the entropy of the codebook usage in VQ-VAEs. Codes that have not been used for more training iterations than a set threshold are resampled, preventing dead codes that receive no updates. Experimental evidence suggest that this strategy yields improvements over the baselines that grows with the size of the codebook. Our results show no notable inferiority scenario, and can be used as a default safely.

## 6.12 $\Rightarrow$ *Contribution*: A Latent Variable Model for facial pose generation

### 6.12.1 Problem setting

We saw in the latent variable model (Section 6.5.3) a way to disentangle known factors from the rest in a generative process. In the situation we are interested in, face generation with controllable identity, we wish to learn a generative process able to disentangle facial geometry, specific to a given identity, from other factors such as pose, pixels location in picture, lighting, saturation. Some other factors are more ambiguous such as makeup, hair style, or age that may or may not change according to identity and can sometimes help recognize someone.

We evaluate our system under face swap quality and image quality.

### 6.12.2 Methods

#### Architecture

The system we propose is a latent variable model in the form of an encoder-bottleneck that extracts latent variables from a picture, and sends them to the decoder along with an identity embedding. We aim to reconstruct the input image under a L2 pixel-wise loss and a VGG loss (also called Perceptual loss, using a VGG16). The VGG loss is the L2 difference of the deep features of a pretrained VGG network, extracted after every ReLU.

We choose simple convolutional encoders and decoders following a VGG style for both the encoder and decoder. The encoder (a VGG11 with BN up to the linear layers) uses quantification bottleneck as we have seen it produces crispier pictures. The decoder is the same as the one used for experiments in Section 6.11.

Figure 6.22 shows our proposed approach.

We emphasize the importance of image quality since the produced samples are to be used as training data.

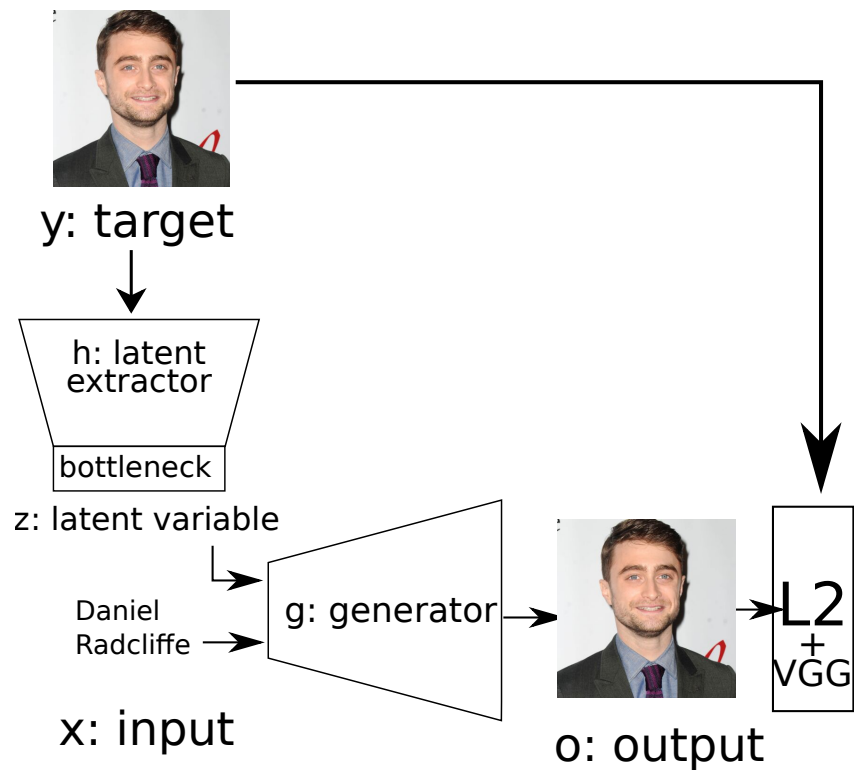


Figure 6.22: Our proposed controllable face generator. A target face is encoded to a latent, decoded back with the identity label to the original picture. An information bottleneck in the encoder discourages the latent variable to contain any information about the person’s identity, thus not leaking any identity specific geometry and encoding parameters not recoverable from the identity alone: lighting, pose, makeup, etc. At inference time, one can use any latent from any sample or sample a latent from a prior distribution to reenact anyone’s face.

### Training

We train the model on aligned faces from Hexaglobe with RAdamW. The algorithm is implemented with Torchélie.

At inference time, we reuse latent variables from the training batch but randomly shuffle identity vectors within the batch. Not only is this simple but also aligns with our goal of performing face swap.

1. While not ideal, we score the quality of the face swap features by the ability of a face classifier to recover the new sampled identity.
2. We compute the KID [15] (Section 6.10.2) as our image quality metric. This is only an indicative number as the KID is a *distribution matching* metric. While the precision outlined in Section 6.10.3 was a better fit for image quality, it was not yet implemented in Torchélie. We argue that since we exchange identities in a batch instead of randomly sampling them uniformly, the swapped distribution does not diverge much from the ground truth distribution, making KID a usable image quality metric in this situation.

### 6.12.3 Negative results

- We tried training the model with a Gaussian prior information bottleneck. The image quality was terrible (blurry), which is unsurprising from a standard VAE approach. Moreover, the manifold

of the latent variations was too complex to fill the whole Gaussian volume, resulting into many "holes" in the prior spaces. Sampling latents from the prior lead to results only slightly looking like faces, far worse than reusing extracted latents.

- We found the L2 pixel loss not sufficient and too harsh to generate meaningful images. This loss considers that all pixels are equal in the image, which is not true. Some pixels of the face, like face contours, bear more semantics than the others, like background pixels. A VGG loss captures this pixel importance and emphasize the importance of those pixel structures, while relaxing the need to reconstruct the target picture in a pixel-perfect fashion. The VGG loss compares image semantics rather than pixel intensities.

#### 6.12.4 Results and Conclusion

Our generator exhibits a good face swap quality: 92% of the time, a face classifier trained on the identities recognizes the target identity, showing that our model is successfully disentangling identity latents and pose latents. We reach a KID of 0.044. The generated images are somehow "too clean" and the background is not captured, showing a washed out "average background color" without any pattern. This is due to the L2 loss that encourages to produce an average (in this case, blurry) response when patterns fail to be captured.

The results from our classification metric have to be discussed. This metric kind of contradicts itself: an excessively low result would indicate that the swapping fails, but a very high result would signal that our classifier perfectly recognizes the swapped identities. This would sound like a good thing, but it would actually show that the proposed data augmentation is useless. Indeed, this would indicate that our face classifier perfectly disentangles pose and face features, without learning shortcuts or overfitting on spurious elements. Whether or not the score of 92% is an indication of the latter can only be known by computing the actual impact as a data augmentation, which is still to be done.

Figure 6.23 presents some resulting samples. The image quality is, as expected from a VQ-VAE, fairly good. Notice how the images from each couple of rows look identical but, upon closer inspection, actually display different identity-specific facial features (hair color, nose size, skin tone, eyes shape, etc).

#### 6.12.5 Future Work

In our tests, we reuse pose latents extracted by the encoder. If we wanted to generate new poses as well, we could train an autoregressive model on the latent variables of the training set. Generating new poses is not an objective of this work, but it is left as a future work. For now, the training set is already large enough to propose a large diversity of latent variables.

More important to us is the ability to scale the model to new identities. Ideally, we want to generate as many training pictures from as few ground truth pictures as possible. Future work will include exploring directions similar to the ones used for inverting StyleGAN [27, 133]. Similarly to these works we could either:

1. learn an identity encoder network that predicts an identity embedding from a face picture;
2. extract the pose latents from a specific picture then optimize the identity embedding under a VGG or pixel reconstruction loss;
3. optimize the decoder and latents to reconstruct the image.

The possibilities are not limited to this list.

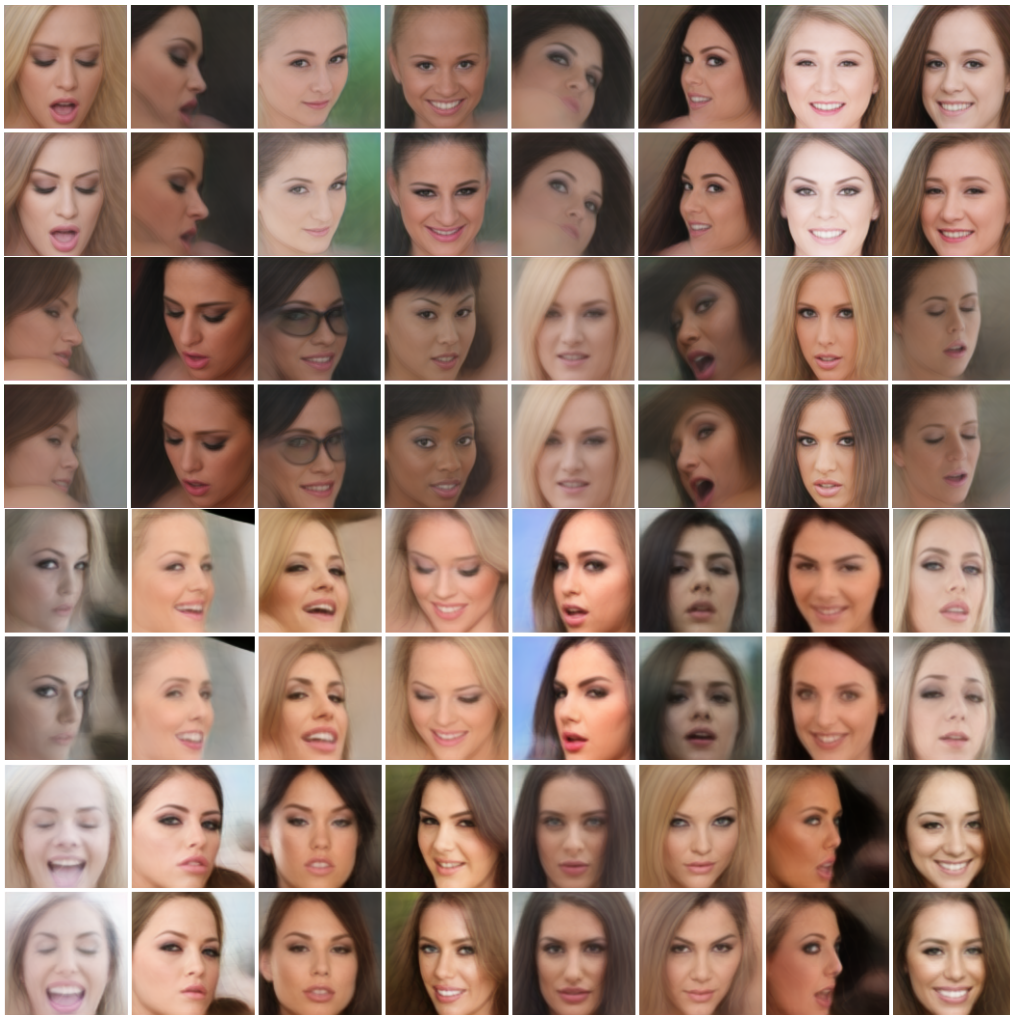


Figure 6.23: Four batches of not curated samples. Rows 1, 3, 5, 7 are reconstructed samples. Identity is randomly swapped in rows 2, 4, 6, 8 but latent vector is kept untouched.

## 6.13 Conclusion

In this chapter we explored various ideas about how to augment a face recognition dataset with invariants. We hypothesize that enriching our dataset with a face swap tool would enforce a face recognition model to really exploit facial geometry features, and would reduce the risk of overfitting on backgrounds, or makeup, that would be transferred by the swapping. With this goal in mind, we presented various generative models: autoregressive models, VAEs with a focus on the VQ-VAE, and GANs. We presented various problems and their proposed solutions in the literature, such as Spectral Normalization or R1 regularizer.

We then presented conditional and controlled modelling allowing control over the generated samples, which is necessary in our situation as we want to generate pictures of a specific person from another picture of someone else. We presented various algorithms: cGANs, Pix2Pix, Pix2PixHD, BiGAN, CVAE, InfoGAN, CycleGAN and CUT.

We contributed an improved VQVAE. Codes that have not been used for more training iterations than a set threshold are resampled, preventing dead codes that receive no updates. Experimental evidence suggest that this strategy yields improvements over the baselines that grows with the size of the

codebook. Our algorithm shows no notable inferiority scenario, and can be used as a default safely.

From this VQVAE we proposed a face swap model. Our generator shows a 92% face swap success rate on our tests. The images exhibit good quality. They show good facial transfer while keeping other features untouched, as expected. The impact of this augmentation strategy for training our face recognition model is still to be evaluated and left as future work. Inverting this model, getting inspiration from GAN inversion, is another step to take in order to explore this model and its usability.

Now that we have extracted metadata from the videos with our activity recognition and face recognition models, we will build our recommender system.



## 7 Recommender System

### 7.1 Introduction

Now that we learnt how to extract features from videos, those will be fed to a recommender system. In this chapter, we will explore various notable ways of building recommender systems, then dive into how we built ours. This section features various experiments showcasing the importance of various features and augmentation strategies.

When browsing on YouTube for instance, the landing page proposes some content. This content is tailored and suggested according to the visitor. Suggesting videos at random or just the most recent videos would do for a terrible user experience as most of them would not be relevant to the user's interests.

In order to extract the maximum value from the available videos, relevant videos should be favored and presented individually to each different user, and give each one of them a tailored YouTube experience. Selecting those videos that are a good fit for a given user is the role of the recommender system.

#### 7.1.1 Lexicon

More formally, a recommender system setting involves several entities.

**The items** are the objects of interest of the application domain. For YouTube, these are videos, for Amazon these are products and for Spotify these are songs.

**The platform** is the application. It can be Spotify, Facebook, YouTube, etc. The platform hosts items to recommend. They have their own set of goals, usually maximizing revenue.

**Content creators** are the ones creating new items on the aforementioned platforms. Musicians for Spotify, user profiles for Facebook, shops for Amazon. Some platforms, such as Last.fm (music recommendation) or MovieLens (movie recommendation) have no content creator. Instead, they have a catalog of items provided by the platform. The content creators have their own incentive for using that platform: they make revenue based on views, they sell items, they get exposure, etc. If the platform does not care enough about them, they might stop creating content and the platform loses its value.

**Users** are the ones exploring the items and interacting with them. They are YouTube's viewers, Spotify's listeners, Amazon's shoppers, etc. Users find value in the platform for various reasons: it hosts items they value, it allows discovering new items they like, etc. If the platform does not have their interest at heart, they might leave, decreasing revenue to both the platform and content creators.

**The recommender system** is a key element of the platform that has to solve a tripartite equation: maximizing its own business goals while maximizing the incentive for content creators to remain on this platform and to create more valuable content, and ensuring that users will get in contact with it. In other words, the recommender system has to find a mapping from content to users that is optimal for the three actors.

### 7.1.2 At Hexaglobe

Hexaglobe provides platforms for their clients. One major way a user interacts with content is by searching. In order to provide a good search engine, the platform has to extract features from behavioral patterns or from the content itself (description text, computer vision, etc); or to ask content creators for metadata about the item. Face recognition, for the client I am working for, provides metadata that is valuable to index items, and this is complemented by other tags.

However, in some cases, we want to be able to suggest the videos the user wants even before they search for it. This is typical on landing pages where we want to suggest videos the user will enjoy. This is known as a recommender system as it recommends content to a given user, sometimes based on its profile info and/or browsing history.

I have been tasked with giving a shot at writing a recommender system for my client.

## 7.2 Recommender systems are hard to build

Recommender systems are hard to build. They face many challenges. We all have friends who bought a toilet seat once because the current one broke and then Amazon wanted them to buy more toilet seats for months like they were on some weird toilet seat collection spree. We all got angry at Spotify for playing a song we hate, at YouTube for recommending for the billionth time that video we kept ignoring purposefully. We try to give a few explanations of the challenges.

### No clear ground truth

The main reason is that there is no clear answer or goal to maximize or even evaluate. The complex interactions outlined above, between the platform's, content creators' and users' interest has usually been disregarded as they are too hard to express, quantify and too business dependent. Instead, the researcher community has sought to develop business agnostic frameworks, the most common one being to model user preferences through ratings or an item's relevance to a user.

It's impossible, by design, to work out an unambiguously labeled dataset for a strict supervised task. The algorithms introduced later that look unambiguous and completely supervised. However, predicting a rating is not predicting what is best to display. Sure, that viewer seems to like romantic comedy, but if the system recommends romantic comedy ad nauseam, he will probably get bored and angry at that recommendation bubble. That's even worse if that user liked ONE romantic comedy and the recommendation engine interprets too strongly that spurious signal.

Jannach and Adomavicius [73] analyse several ways recommender systems can serve different purposes, both from the users' and the platform's point of view. As user purposes, they exemplify: show alternatives, show accessories, help exploration, entertainment, etc. For platform's purpose: create more demand, increase business success, increase activity, increase discoverability of items, learn about customers, etc.

The algorithms presented here follow the trend and present recommendation as analogous to rating prediction, but this is a fairly severe assumption.

### Recommendation bubble

Recommender systems might have the tendency to not balance enough exploitation of safe or known items and exploration of other items. This locks users in a fairly limited subset of items, and limits their ability to discover content, or even giving the false impression that there is no other content.

You just bought a shelf, and now Amazon wants you to buy every other shelf. You listen to rock music, you've never ever been recommended a single rap song, etc. Those are recommendation bubbles. You are getting recommended only items that are similar to things you already know and like, without any exploration or novelty. Sometimes, this is expected and a good thing: if you hate some type of music, it's okay not being exposed to it. Sometimes, you are missing out: you're not being recommended this great movie just because your past watch session inclines you more towards movies of lesser popularity matching a bit more the features you like. Finally, it can also be plain detrimental: being a flat earther whose recommendations include more flat earth fake news and not a single scientifically valid and informative video. Having political point of views that keeps you from being exposed to opinions opposed to yours, but gets you recommendations of more and more extreme content [118, 121].

Recommendation bubbles must be at least considered when building a recommender system, and whether to fight against them or not by adding some randomness or exploration bias to the system.

### Bias towards previous system

Training a recommender system on historical data will bias predictions towards the old one. The new system might learn to recommend a popular item only because the previous system was biased and made it popular. Inversely, the new system might learn to ignore some content only because that content was ignored by the previous iterations and reached zero popularity. There are mitigating strategies to debias the learning procedure to some extent [26].

### Cold start

Finally, what should be done with a new user or item? A new user might not have enough implicit or explicit feedback or information for the system to gauge their interest. A new item has not been seen yet and it's hard to know who it can be appealing to or if it is ever going to be popular. This is known as the *cold start* problem.

### Fairness

Related to the cold start problem is *fairness*. Fairness might not matter for Netflix but might be of primal importance when the content is proposed by users, such as Amazon Marketplace or YouTube. Taking fairness into account is making sure that the system is not overly biased towards old and/or popular items, so that newcomers or smaller artists can still get recommended. Failing to account for fairness is making the system useless for the long tail items and narrowing the recommendations to top popular items. Users might benefit from more diversity and content creators get some benefit getting involved into the platform. [2] inspects popularity bias, [22] explores long tail items in music recommendation.

The long tail items are the ones - usually the majority - that get a much lower popularity than the top popular items. For some platforms, the value lies in exploiting the long tail items through personalization. It ensures that a valuable majority of items does not remain dormant. Figure 7.1 shows the popularity distribution of the client's items. Failing to recommend long tail items would make it a bad platform for content creators and just a waste of hard drive space.

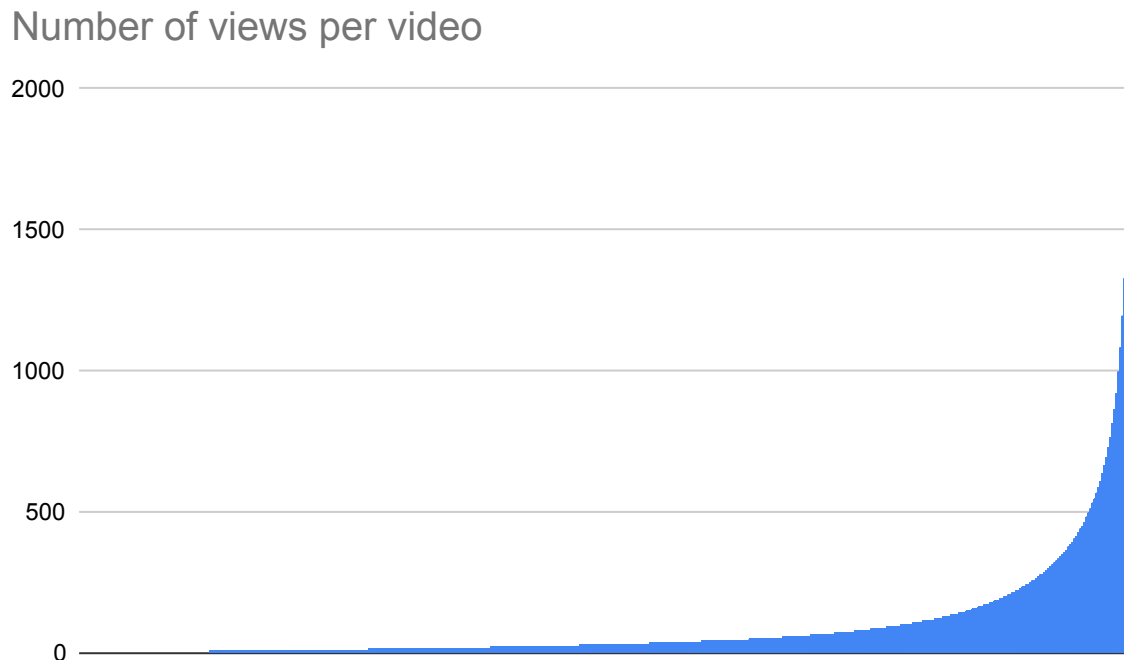


Figure 7.1: Long tail. A few popular items (the head) get a significant higher number of view than the majority (the tail). Exploiting only head items results in neglecting most items. The Y axis is clamped at 2k views but the highest video count is 8k.

## 7.3 Types of recommender systems

### 7.3.1 Definitions and notations

Deciding what to recommend is difficult. To my knowledge, most recommender systems use ratings prediction as a proxy task. The datasets used contain user-to-items ratings from the platform history and the systems aim to predict how would users rate items they have not rated from the statistical knowledge extracted from previous ratings. They assume that the items with the best predicted ratings are the one to recommend.

Those algorithms assume that a given user  $u_i \in \mathcal{U}$  emits explicit or implicit feedback ratings  $r_{i,j}$  to content  $c_j \in \mathcal{C}$ . We have a sparse matrix of rating  $r$  where row  $r_{i,\cdot}$  represents ratings given by user  $u_i$  and column  $r_{\cdot,j}$  represents all ratings given to content  $c_j$ . From this, they want to predict  $\hat{r}_{i,j}$ , the rating for content  $c_j$  that user  $u_i$  would give.

### Collaborative filtering

We can leverage information about a collection of user preferences. It is assumed that if a user  $u_i$  shares opinion about some content with user  $u_j$ , then he should also share a similar opinion about another item that  $u_i$  has not rated yet.

This method is often presented as "user who watched this video also watched", "users like you also liked". Figure 7.2 displays a user rating matrix. The unknown value is predicted from the ratings of other users.

|                     | Harry Potter | The Triplets of Belleville | Shrek | The Dark Knight Rises | Memento |
|---------------------|--------------|----------------------------|-------|-----------------------|---------|
| User 1 (Red hat)    | ✓            |                            | ✓     | ✓                     |         |
| User 2 (Black hair) |              | ✓                          |       |                       | ✓       |
| User 3 (Red hair)   | ✓            | ✓                          | ✓     |                       |         |
| User 4 (Grey hair)  |              |                            | ?     | ✓                     | ✓       |

Figure 7.2: In collaborative filtering, we aim to guess the ratings one user would give to an item given the rating similar users gave. Would she like Shrek because she liked The Dark Knight like user 1, or dislike Shrek because she liked Memento like user 2? Picture from <https://developers.google.com/machine-learning/recommendation/collaborative/basics>

## Content-based filtering

In content-based filtering, a representation of items is extracted from some available features and metadata. An interest feature vector is extracted from a user’s watch history and ratings, and the database of content is queried with it. A baseline algorithm uses Term Frequency–Inverse Document Frequency (TF-IDF) [128] for featurization and a dot-product for determining content relevance. Figure 7.3 shows a user’s interests featurized in the same feature space as the items. Features could include keywords left in comments, main geographical region of users, and metadata might be categories and tags provided by the uploader. We recommend an item based on the user’s similarity with candidate content. Contrarily to collaborative filtering, note that no other user is considered in making a decision. Platform presents content-based filtering such as “items matching your interests”, or “similar to your recent history”.

## 7.4 Baseline algorithms

### 7.4.1 Deep learning is disappointing

As in other areas, deep learning has been applied to recommendation for years, paper after paper. However Dacrema et al. [30] ran a fair comparison between DL methods and carefully optimized baselines on common benchmark, and found out that DL methods were consistently outperformed. Those baseline work better and are orders of magnitude cheaper to run. For those reasons, the DL methods will not be detailed here. Ludewig et al. [105] made similar findings for session-based recommender systems. It seems that deep learning models miss features-features interactions that make traditional methods work so well.





|   | education | casual | Health | ... | game | science | healthcare |
|---|-----------|--------|--------|-----|------|---------|------------|
|  | ●         |        |        | ... |      | ●       |            |
|  |           | ●      |        |     | ●    |         |            |
|  |           |        | ●      |     |      |         | ●          |
|  | ●         |        |        | ... |      | ●       | ●          |

Figure 7.3: This imaginary app store has 3 apps: a science app, a robot game, and a dentist appointment finder. Those apps and John's interests are annotated by a set of tags shown above the table. Based on John's past interests, the first item, the science app, seems to be a good recommendation: John's and this app's feature vector share the greater similarity.

## 7.4.2 Nearest neighbors

### Base algorithm

The  $k$ -Nearest-Neighbors algorithm allows to classify or regress to a value given a labeled training set. Let us lay a quick explanation of the method before applying it to our problem in the next paragraphs.

Let  $(x_0, y_0), \dots, (x_N, y_N)$  be a dataset of  $N + 1$  examples  $x_i$  and their labels  $y_i$ . Classifying a query  $q$  involves finding the  $k$  most similar  $x_i$  to  $q$  under distance  $d(\cdot, \cdot)$ , and taking a majority vote of their class label  $y_i$ . For instance, if  $k = 1$ ,

$$\text{kNN}(q) = y_a \text{ with } a = \underset{i=0..N}{\operatorname{argmin}} d(x_i, q)$$

Here, we will use the cosine distance as the distance  $d(\cdot, \cdot)$ . In case of a regression task, the labels of the  $k$  most similar items is averaged.

### kNN applied to recommendation

To predict the ratings  $\hat{r}_{i,j}$  that a user  $u_i$  would give to a content  $c_j$ , *UserkNN* takes the  $k$  users most similar to  $u_i$  and who have rated  $c_j$ . Their ratings value for item  $c_j$  is averaged.

*ItemkNN* takes the problem the other way around. We search for the  $k$  items  $c_k$  most similar to  $c_j$  that had been rated by the active user  $u_i$ . The ratings that user gave to those similar items is averaged to predict  $\hat{r}_{i,j}$ .

There are multiple possible variations depending on how we decide to represent the users and items for the kNN:

- The *Collaborative filtering* uses the rating row  $r_{i,\cdot}$  (column  $r_{\cdot,j}$ ) to represent an user (item).
- The *Content-Based Filtering* represents users and items by a set of features extracted from items metadata or user profile.
- An *hybrid* method concatenates both metadata features and rating vector.

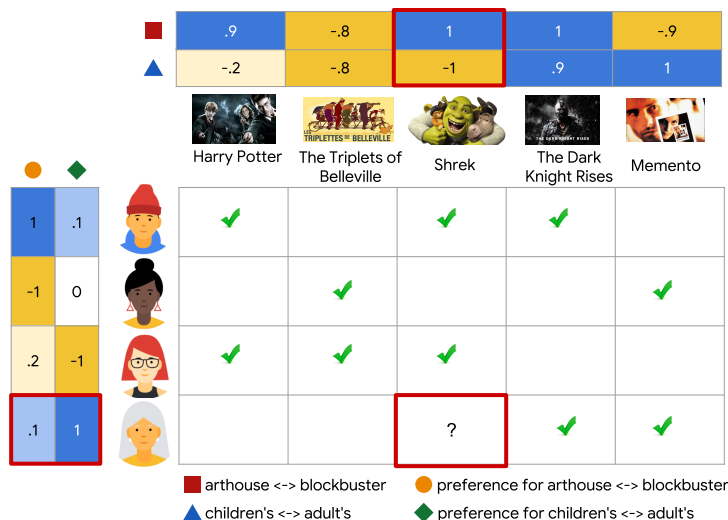


Figure 7.4: The ratings matrix is decomposed as the inner product of user latent factors and movies latent factors, discovered during learning. They can be inspected to find semantically meaningful features. Image source: <https://developers.google.com/machine-learning/recommendation/collaborative/basics>

### 7.4.3 Matrix Factorization

Funk [45] proposes a *Matrix Factorization* approach based on Singular Value Decomposition (SVD) that interprets the ratings matrix  $r$  as the multiplication of a user matrix  $A$  and a content matrix  $B$  such that  $r = AB^T$ . Those matrices are of shape  $|\mathcal{U}| \times k$  and  $|\mathcal{C}| \times k$ , where  $k$  is a free parameter of the implementer's choice which represents the number of latent dimensions used to represent each user and content. We learn them when  $r_{i,j}$  is known by minimizing the mean square error while regularizing both matrices

$$\min_{A,B} (r_{i,j} - A_i \cdot B_j)^2 + \lambda_A \|A\|^2 + \lambda_B \|B\|^2$$

where  $\lambda$  is the hyperparameter controlling the regularization strength. The unknown values can then be predicted  $\hat{r}_{i,j} = A_i \cdot B_j$ .

The greater  $k$ , the more latent factors can be learnt about users and items, the smaller the training error is but the overfitting risk increases. There are various ways to increase this model complexity and account for various biases or features.

Those embeddings can be analyzed to find semantically meaningful latent dimensions, like movie genre or target age group (Figure 2 of [84]). Illustrative examples are shown in figure 7.4.

Pure MF methods need a full retrain at each new user, content or new interaction and suffer from a cold start problem : each user and content is treated independently.

### 7.4.4 SLIM: Sparse Linear Methods

Instead of learning latent factors, one can learn how items recommend other items. Ning and Karypis [119] propose to learn a  $W \in \mathbb{R}^{|\mathcal{C}| \times |\mathcal{C}|}$  matrix representing the relation of items.  $W$  is learnt by

minimizing

$$(7.1) \quad \min_W (r_{i,j} - r_i \cdot W_j)^2 + \lambda \|W\|_2^2 + \beta \|W\|_1$$

subject to

$$W \geq 0, \text{diag}(W) = 0$$

where  $\beta$  and  $\lambda$  are the hyperparameters regularizing, respectively, the  $L_1$  aiming for a sparse  $W$  and the  $L_2$  norm preventing overfitting. The diagonal of  $W$  is forced to zero so that items can't recommend themselves and fall into that trivial solution. Given that both  $r$  and  $W$  are very sparse, computing  $\hat{r} = rW$  can be done very fast with sparse aware math libraries.

#### 7.4.5 Embarrassingly Shallow AutoEncoder (in Reverse order) (EASER)

When the matrix  $W$  of SLIM fits in memory, Steck [145] proposes to remove the sparsity constraint and the positivity constraint. They devise a closed-form solution that allows for much faster learning. The results are on par or better than SLIM.

### 7.5 ⇒ *Project: Hexaglobe's RecSys*

#### 7.5.1 Problem setting

The client I am working for has 7M videos to index and recommend. They need recommendations in two places: on the homepage, and on a video page, as "continue watching" suggestions. The current homepage system is a country-wise top popular pick refreshed every hours mixed with some new uploads in order to fight the cold start problem. The "continue watching" suggestions are gathered from videos having similar search terms.

Despite many conversations, we could not fix an unambiguous single business objective to optimize. Do we want to optimize for retention? user fidelity? number of videos watched? watch percentage per video? They proposed subjective relevance evaluation with manual testing from the managers and client leader.

We could not agree on such a business centric metric, so we reverted back to a model metric, and judge our model based on its ability to recommend what the user watched next in the top-k recommendation. This is a Precision@k metric.

What should be further emphasized is that the baseline algorithms outlined above were very hardly applicable here. Indeed, the ratings are so scarce and unreliable on our client's platform that predicting ratings is not the option of choice here, despite its success. Instead, having profusion of users' watch history and metadata, it was easier to devise the problem as a sequence prediction setting.

#### 7.5.2 Constraints

The system has some constraints to operate under in order to be useful in our production environment:

##### **Latency**

The system must return a recommendation response in under 30 ms.



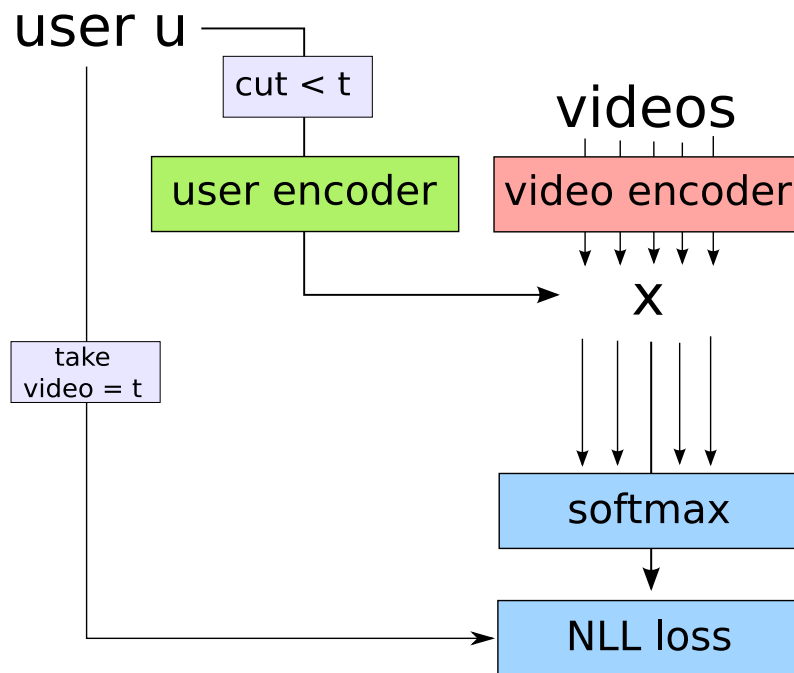


Figure 7.5: Overview of the model. We sample a user, randomly sample a video from the watch history, and cut the history at its watch timestamp  $t$ . The user is embedded by a user network while videos are encoded with a video network. The dot product of their embedding is computed and fed to a softmax + negative log likelihood loss, trained to predict the next video watched. The  $x$  denotes the dot product / matrix multiply operation.

### Cold start for videos

Around 15k videos are uploaded each day. Without a manual shuffling from our part or serendipity aware recommender algorithm, many of those videos would not get a single view, turning them into dead items. The system must not rely on collaborative features only for representing videos.

### Cold start for users

Most users are not logged in when visiting the website. There is some cookie tracking done but users are not tracked between devices and many users browse our website in incognito mode. These constraints make it impractical to write a highly personalized engine. For these reasons, it has been decided to deploy the resulting engine on *premium users* only first. Premium users should see the benefits of paying for their subscription as soon as possible or there is a risk of losing them. This is a tight requirement that encouraged me considering seriously the user cold start problem as well.

### Fairness

Even if it is secondary for the customer's business, dominated by big content creators, they want to encourage indie content creators. If it is not detrimental to recommendations, fairness is a desirable property to have.

### 7.5.3 System design

#### Overview

Back when I had to work on the recommender engine for Hexaglobe, the study of Dacrema et al. [30] was not out yet and I chose to base my work on the Deep YouTube engine laid out by Covington et al. [26]. This work proposes to learn a softmax classifier  $f$  that predicts what a user  $u$  will watch next. The user is represented by the profile and browsing history, while the videos are learnable embeddings in the last dense layer. Their function is designed to predict  $y = f(u)$ .

We propose avoiding the cold start problem by learning a user embedding function  $f(u)$  and a video embedding function  $g(v)$  for videos  $v$ . We predict the next video with

$$(7.2) \quad y = \text{softmax}(f(u) \cdot g(v)^T)$$

Which is similar to the previous method but computes the video embeddings from a model instead of learning them as weights. This enables to use video metadata to generate embeddings, and possibly be more data efficient:  $g$  has access to metadata to know which video are similar or not, instead of having to discover that from user behavior alone. Moreover, when a new video arrives on the platform, we can predict an embedding, instead of it being a dead item without user feedback, making it impossible to learn an embedding for it.

It is interesting to note that this particular approach is in itself a Matrix Factorization parameterized by a neural network instead of learnt embeddings. At its core still lays the idea of maximizing the dot product between the user's and the relevant videos' embeddings. Interestingly, Dacrema et al. [30] acknowledges the work done by Rendle et al. [132], proposing this architecture. Instead, it focuses only another proposed architecture, which does not perform as well, in which the explicit dot product is replaced by an Multi-Layer Perceptron (MLP). Rendle et al. [132] even dives into explanations as to why the explicit dot product performs favorably. We are in this situation, left out by Dacrema et al. [30].

We should also emphasize again that traditional recommender system are trained on ratings. When user provided ratings are not available, implicit ratings are derived from user activity, based on domain knowledge, or heuristics. Predicting the next seen videos alleviate this need as well, and the aforementioned problem of cold start. The downside is that traditional models run fast, which is needed for real time recommendation, and neural networks usually need significantly more compute. The latency of the system has to be carefully taken care of, or the website would take more time to load, impacting the user experience very negatively.

Figure 7.5 exhibits the model. It can be seen that there are two main components: a user encoder and a video encoder. The next sections will dive into those in detail.

#### User network

The user network is a 2-layers MLP with 1024 hidden units and 64 output units. All categorical variables have an associated learnt embedding using pytorch's `nn.Embedding` layer which effectively transforms a discrete vector into a dense, learnable, continuous vector in latent space. The embedding of the  $n$ -th value of a categorical variable is  $W^T \mathbf{one\_hot}(n)$ , where  $W$  is a learnable matrix (basically extracting the  $n$ -th row of  $w$ ). When a categorical variable can take multiple values at the same time (like video tags), the embeddings of all tags are averaged, thanks to pytorch's `nn.EmbeddingBag`. It encodes various features:

- **Profile features:** country and gender. Categorical variables like those use learnt embedding before going into the MLP.

- **History features:** for the number  $H$  of previously seen, liked, favorited and disliked videos: their tags, their category, their main actors, their uploader's ID. The embeddings for all videos are concatenated. Sequences shorter than  $H$  are padded with zero embeddings. All the videos share the same embedding layers; there is no reason the tags from the last seen video must be encoded differently than the second to last. However, there is a reason to keep each video encoding separate, so that the network can estimate if the user's history is diversified or not, what was the previous seen videos so that it knows it can willingly decide to recommend it again or not, etc.
- **Request features:** for now, the category the user is browsing the website for. It was crucial making sure that the system would not recommend videos from another category as it displeases users.

The dimension of the `nn.Embedding` vectors is equal to  $\min(512, \text{round}(1.6n^{0.56}))$  with  $n$  being the number of possible values for this variable (following FastAI library's empirical rule of thumb). We set tag embeddings to have 64 dimensions.

### Video network

The video network is similar to the user network. The input features are:

- **Popularity features:** the popularity scores, summed over all categories.
- **Metadata features:** the category, the video tags, the main actors, the uploader's ID.
- **File features:** the video's encoding quality.
- **Optional: visual features:** convolutional features extracted for some images from the video.

The embedding layers for tags, actors, categories, and uploader ID are shared among both networks.

### 7.5.4 Training strategies

The data is sparse. There are a lot of different entities to embed, most of them being used rarely. In practice, this is a problem. The network could easily overfit and avoid learning a general recommendation function. Because they are rarely appearing, the model may easily learn that tags 12, 68 and 98 identify video #9786, and recognize user #1234 from the history. The net would essentially learn nothing but a big table giving the ID of the next video given some user and timestamp and never form any semantic understanding.

In order to avoid such overfitting we propose to separately pretrain as much as the entities' embeddings as possible, and heavily regularize training.

#### Pretraining tags and words embeddings with Word2Vec

**Word2Vec** was proposed by Mikolov et al. [109]. They learn words embeddings in a language-model fashion, exploiting words co-occurrence. They either predict the probability of a center word  $w_t$  at position  $t$  its  $K$  left and right context words, or the opposite.  $P(\text{center}|\text{context})$  is known as the Continuous Bag Of Words model while  $P(\text{context}|\text{center})$  is known as the skip-gram model.

Once learned, those embedding display semantic properties, like similar words being clustered together or words showing linear relations. For instance, when training Word2Vec embeddings on general English corpus, we find that in this semantic space the vector from France to Paris is roughly the same as from Germany to Berlin.

Figure 7.6 illustrates both CBOW and Skip-Gram models.

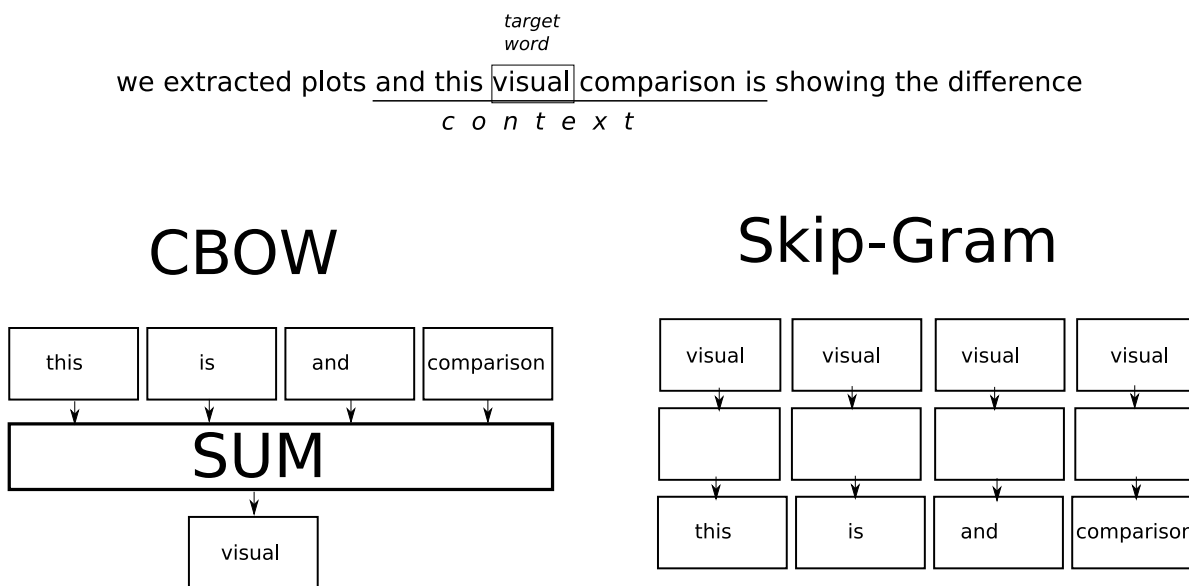


Figure 7.6: Illustrating word2vec training. A linear model trains word embeddings either by predicting the center word of a context window, or the context words of a context window from the center words.

**Tags** of a video, contrarily to words in a sentence, however, have no order. Deciding on a "center word" makes no sense, nor does "context window". A user's watch history is a natural ordering. We thus considers all tags on a video to be center tags, and use the tags of previous and next videos as context. The target word is then a randomly picked word in the center video; tags from center and neighbouring videos are considered context as well. This already biases the embeddings towards recommendation. We use a context window of 5, that is, we use 2 videos before and after the center video as context.

We use the same strategy for words in the titles, also sampling context words, when applicable, from their multilingual translations. Stop words (pronouns, determiners, etc) that I am able to identify (French, English, Spanish) are removed.

Inspecting the embeddings with a technique such as Uniform Manifold Approximation and Projection (UMAP) [107] shows great insight about our tags. For instance, anime tags are effectively clustered together, such as various groups of tags for various genres, scene types, or famous actors. From this alone, computing the Word2Vec embeddings has value. One could envision running a k-means algorithm on those embeddings to create invisible meta-tags that help even more with the indexing, merging different tags but with the same meaning.

Those embeddings are used, frozen, in the recommendation model.

### Regularization and data augmentation

In order to both regularize the model and augment the data, dropout-based strategies are employed:

- **DropTags:** There is a probability  $p$  to drop each tag in a video metadata.
- **RandLang:** Only one language is selected at random when there are multilingual data for the title
- **DropPop:** the popularity info is randomly dropped. This encourages fairness as well as the system learns not to recommend only videos with high popularity scores.

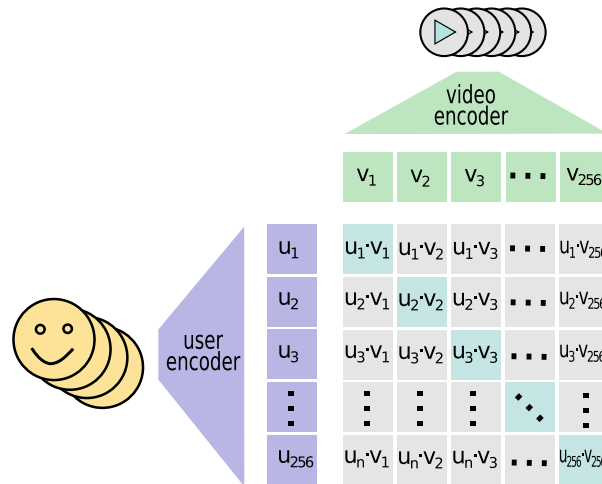


Figure 7.7: We train and evaluate our recommender system in a contrastive way. Batches of 256 pairs of histories and their next viewed video are loaded; the encoders learn to embed them so that the dot product of the real pair is greater than the ones of the other possible pairs formed in the batch. In other words, the encoders are learned so that  $u_i \cdot v_i > u_i \cdot v_j$  with  $i \neq j$ .

- **DropUploader:** Sometimes drops the uploader ID. The system learns not to rely only on popular uploaders.

### 7.5.5 What's next?

Following the findings in Dacrema et al. [30], a simpler matrix factorization approach, not based on deep learning, might work better or just as well for a fraction of the compute. However, as mentioned earlier, those shallow approaches suffer from a generalization / cold start problem.

We could then learn a shallower system such as Matrix Factorization or build a collaborative filtering matrix for a user-knn or item-knn approach, disregarding (almost) inactive users and unpopular videos.

For computing a score for a user or video (or both) not learned in a shallow model, a deep embedder net finds the closest known user / item, and the rating is taken from the corresponding entries in the shallow model.

### 7.5.6 Experimenting with the model

In order to gain insight into our dataset presented in Section 2.3.3, we choose our hyperparameters and refine the model, we run a collection of experiments. The model is trained with batch size 256 and Adam for 40k iterations. The learning rate warms up for 5% of the training iterations then linearly decays. We train our model in a contrastive fashion: the batch contains 256 queries  $x_0, \dots, x_{255}$  and their 256 associated next videos watched  $y_0 \dots y_{255}$ . All queries and videos are embedded and the dot product is computed for all pairs so that the matrix  $a_{ij}$  contains the dot product of the embedding of  $x_i$  and  $y_j$ . A row-wise softmax is performed on  $a$ , and the cross entropy loss is computed so that row  $i$  must predict label  $i$  (see Figure 7.7).

This strategy naturally samples according to the items' popularity. Sampling by popularity creates a harder prediction problem than uniformly sampling candidates items. It directly embeds the popularity bias of the items, forcing the system to rely on the features rather than learning which items are popular and finding them among long tail items. Top1 accuracy is used as a metric.

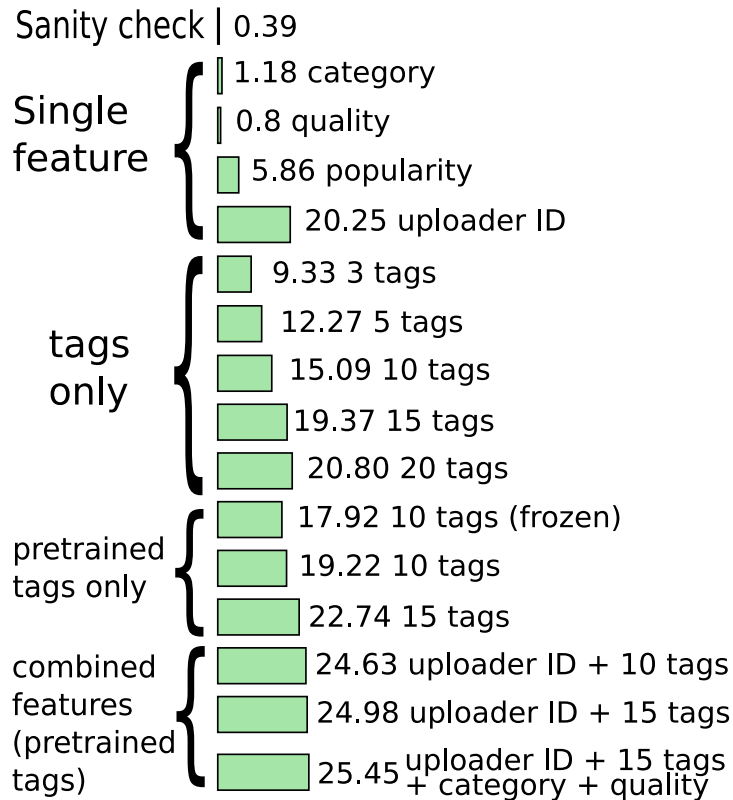


Figure 7.8: Experiments on video encoder for a fixed user encoder. We aim to understand how features contribute to the classification information and build a model from this information. Test Top1 accuracy is indicated.

Reusing the other examples in the batch as negative targets prevents manually inserting user-videos interaction features such as: has the user already seen this video? How older than the request is that video?. This however is not a bad thing: for latency reasons, it is better to precompute the embeddings of the videos, which is not possible if they depend on the user and/or the request.

For testing, we use an identical setting as for training. The only difference is that the testing set consists in an isolated set of users that have not been seen in training. However, the set of videos is shared between training and testing as it would be extremely hard to find a user split that also splits the seen videos in two disjoint sets.

### Candidate videos encoder

We arbitrarily fix an history length of 2 with all user features as a starting point for running experiments. We add or ablate various features in the video encoder. Those experiments are also reported in Figure 7.8. We report the Top1 test accuracy for each experiment.

1. A featureless experiment as a sanity check, making sure the accuracy is  $1/256=0.39\%$ , which gives 0.39%, passing the test.
2. Only the category of the next requested item and of the candidate videos, giving 1.18%. This is not surprising as there are only 3 main categories, which are notably unbalanced. It is important to recommend items within the correct category. This is a typical case where false positives are to be avoided. Despite not being very helpful in a Top1 test accuracy sense, this feature is of crucial importance.

3. Only the video encoding quality of the videos, giving 0.8%. The video quality does not seem to explain much of the user behavior.
4. Only the popularity scores, giving 5.86%. This is quite surprising and would indicate that the dataset might not be enormously biased towards popular items. This hypothesis could not be verified, as the scores were computed some time ago and the code is not available anymore. Thus, the popularity scores may have been processed in an unhelpful way.
5. Only the uploader's ID, giving 20.25%. It seems that this gives a lot of valuable information about the content, suitable for recommendation. A projection and visualization of the learnt uploader ID embeddings would help understand the semantic space they are organized into, and help get insight into the information extracted from the uploader ID.
6. Only 3 tags per video, giving 9.33%.
7. Only 5 tags per video, giving 12.27%. There are substantial gains adding more tags.
8. Only 10 tags per video, giving 15.09%. Adding more tags still helps.
9. Only 15 tags per video, giving 19.37%. Adding even more tags still helps.
10. Only 20 tags per video, giving 20.80%. This yields diminishing returns, 15 looks good for tests.
11. Only 10 tags per videos, tags are pretrained with Word2Vec, giving 17.92%. This is a notable improvement from the random initialization. Does it help unfreezing the embeddings?
12. Only 10 tags per videos, tags are pretrained and trainable, giving 19.22%. Unfreezing tags embeddings help. Since we are challenging the performance with 15 tags, try with 15 pretrained and trainable tags.
13. Only 15 tags per videos. Tags are pretrained and trainable, giving 22.74%. The gains are still important.
14. Uploader ID and 10 tags per videos. Tags are pretrained and trainable, giving 24.63%. The uploader ID and tags do not bear totally redundant information.
15. Uploader ID and 15 tags per videos. Tags are pretrained and trainable, giving 24.98%. The uploader ID seem to bear enough information so that more tags yield diminishing returns.
16. Uploader ID, 15 pretrained and trainable tags, category, quality, gives 25.45%. This value serves as an upper bound when all info is available.

We now have a pretty clear view of the importance of candidate video features. We settle to 15 pretrained and trainable tags, not using popularity scores (favoring unbiased information), with uploader ID, main category and quality.

### User encoder

Now, we run feature experiments on the user encoder, summed up in Figure 7.9.

1. We remove all features from users, giving 0.3905%. Sanity check passes, without features, we get 1/256 chances of being right.

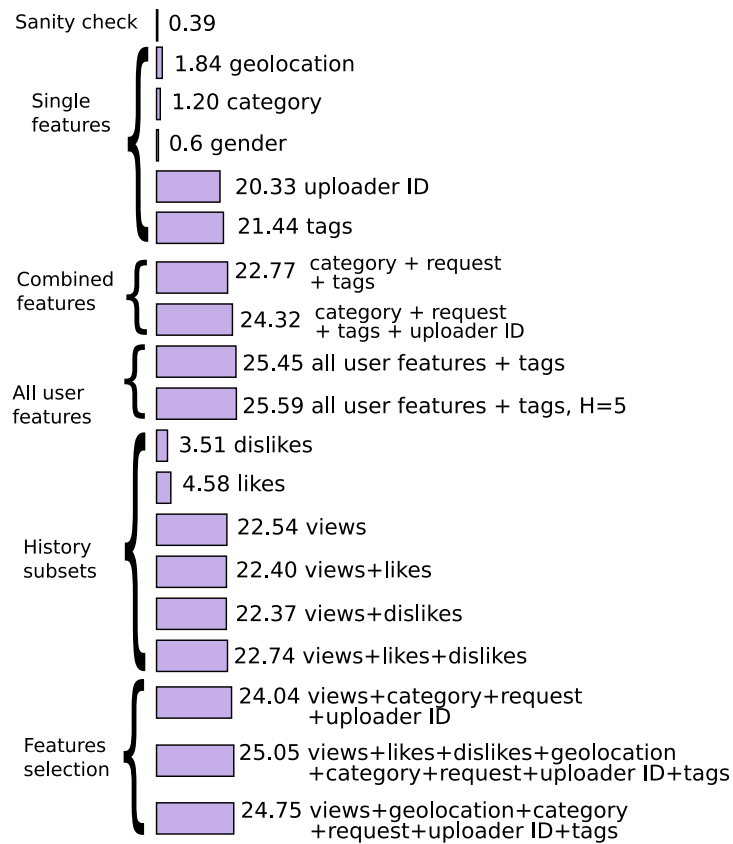


Figure 7.9: Experiments on user encoder for a fixed video encoder. We aim to understand how features contribute to the classification information and build a model from this information. Test Top1 accuracy is indicated. Unless indicated otherwise, the history length  $H$  is set to 2



2. Only profile geolocation features, gives 1.84%. This is surprising as location correlates with language which can be inferred, to some extent, from tags. We know that language does not constitute a major feature of our videos, but the accuracy gain still looks quite low.
3. Only requested category gives 1.20%. This matches the previous results with main category only in candidate videos.
4. Only user profile's gender feature gives 0.6%. Unsurprisingly, gender is a low indicator of user preference.
5. Only Uploader ID from history gives 20.33%.
6. Only 15 video tags from history gives 21.44%. This jumps accuracy close to its maximum observed value.
7. 15 video tags from history, video categories, request category, gives 22.77%.
8. 15 Tags, category from history, uploader IDs from history, and request category gives 24.32%.
9. Enabling all user features, growing from 2 to 5 elements in history increases from 25.45% to 25.59%. This is not enough for the increased computation.
10. Using all user features, only 15 tags in candidate video encoder but only viewed videos in history, gives 22.54%. Only dislikes yields 3.51%. Only likes 4.58%. Views and like, 22.40%. Views and dislikes, 22.37%. All of them, 22.74%. We see that, contrarily to intuition, likes and dislikes bears little information compared to views. Moreover, this information is not only redundant but maybe slightly misleading, suggesting not using likes and dislikes in the model.
11. Using only views, categories, uploader ID, we reach 24.04%.
12. Using all data but gender (ie tags, categories, geolocation, uploader, request, tags), with only past views from history, we reach 24.75%. We consider that adding geolocation, even if it does not bring tremendous improvement, is very cheap and assume that it biases favorably the suggestions for a new user with localization.
13. 25.05% with views, likes and dislikes, confirming the improvement is marginal.

We settle with a user encoder that uses all data but gender (ie tags, categories, geolocation, uploader, request, tags), with only past views from history, reaching 24.75%.

## Transforms

We now evaluate our transforms.

1. We activate DropTags with probability 0.25, effectively randomly dropping a quarter of the tags during training. It gives 24.78%, making no difference. Raising to 0.5 probability yields 23.89%, impairing the training. DropTags is better kept off.
2. We activate DropUploader with probability 0.25 gives 24.63%. Probability 0.5 gives 24.37%.

We designed those transforms inspired by CutOut [36] (Section 3.3.5). The results suggest that randomly removing some input features during training does not make the model more robust as expected, but impairs learning instead.

As it is sometimes the case with data augmentation, we experiment with less regularization and longer training schedule. We double the training time and reduce the weight decay from 0.1 to 0.01.

1. Doubling the training iterations brings our baseline to 25.11%. Adding DropTags with probability 0.25 yields 25.35%.
2. Reducing the weight decay brings our baseline to 24.91%. Adding DropTags with probability 0.25 yields 24.05%.

It seems that DropTags starts to be moderately useful with more training iterations. Time and efforts would be better spent on feature engineering as suggested by all the previous experiments.

### **Further experiments**

We finally want to evaluate the contribution of face recognition to the recommender system. Adding the detected identities as features goes to 26.84%, demonstrating a noticeable effect in recommending content. The identities embeddings could be pretrained with word2vec, in a similar fashion to tags embeddings, maybe yielding even greater improvements.

### **7.5.7 Discussion**

Starting from Covington et al. [26], we designed our own recommender system, guided with experimental results. We found out that tags are the most important features to be used, followed by uploader ID as each uploader has its own type of content. We expected our model to overfit and devised some data augmentation techniques; the experiments showed that overfitting was not a problem but DropTags still managed to be helpful. We also showed that pretraining embedding with Word2Vec noticeably helps (approximately +4% going from random to pretrained tag embeddings). The next steps to be taken for maximum rewards are probably pretraining identity and uploader embeddings as well. This could be done by predicting the associated tags distribution or their co-occurrences in users' watch history. The practical use of this model now has to be tested in real world condition, and A/B testing [176] is needed in order to compare it to the current recommender system to verify its superiority in actually understanding users' preferences.

## 8 ⇒ *Project: Torchélie*

### 8.1 Introduction

Back in Montréal, working for JSALT2019 in order to publish [25], I wanted to gather all the deep learning related code I wrote so far for my thesis. The initial motivation to realize this project was clear: while deep learning code is often short to write, it is also extremely easy to get wrong, and even harder to diagnose and debug. In standard software engineering, most mistakes end up crashing the application or raising exceptions, making them obvious to uncover. In deep learning, they damage the results, often in very subtle ways.

Besides, many code bases would share common patterns: training and testing loops, alternating training for G and D in GANs, measuring accuracy, layers or blocks, etc. In most public code, the training code is often made of raw for-loop instrumented with many ifs, and as the need to monitor new quantities grow, the code gets harder to maintain and error prone. As we try new incremental ideas, the ifs switches grow out of hand, readability suffers, and more bugs arise.

Torchélie is a software engineering take to tackle this problem and provide tools to build both experiments and production-ready code that is fast to write and easy to maintain, from battle tested building blocks. It is based on PyTorch that it extends.

Torchélie is a twofold contribution:

1. first, as a library and toolbox for pytorch, providing many utilities. We aim to mimic PyTorch's style as close as possible, hoping to make it a seamless experience;
2. second, as a set of design principles that can be followed even outside of Torchélie in order to ease iterative development.

### 8.2 Overview

The first contribution Torchélie brings is a Python package based on PyTorch that contains multiple tools extending Pytorch horizontally:

**`torchelie.datasets`** implements new datasets, new dataset transforms, and dataset utils. It contains utilities like `FastImageFolder` which caches pytorch's `ImageFolder` file list, making big datasets loading much faster; `PairedDatasets` sampling pairs from two datasets at a time, which is useful for tasks like image translation or augments like `Mixup`; `MixupDataset` which samples from a dataset and uses `Mixup` to augment the sampled image and its classification target; `Subset` which allows using only a random (but reproducible) subset of dataset, quantified either as a fraction or a number of samples. It also provides datasets loaders and downloaders such as `MS1M` which is able

to load MS1M despite its file format encoded for MXnet, `Pix2PixDataset` loading from NVidia's servers the datasets used for Pix2Pix [72], Imagenette, and Imgewoof.

**`torchelie.distributions`** adds Logistic distribution, LogisticMixture, GaussianMixture and Truncated Normal to `torch.distributions`, used mainly in `PixelCNN(++)` [83, 137].

**`torchelie.loss`** contains many losses and regularizers, noticeably for GANs (Hinge loss for BigGAN [19], R1 regularizer from [108], R0 regularizer from [150], WGAN loss from [51]), face recognition (AdaCos [182], ArcFace [34]) and style transfer. It implements the `BitemperedLoss` from Amid et al. [5] that is said to be more robust to label noise, `DeepDreamLoss` from Mordvintsev et al. [113], Style Transfer loss from Gatys et al. [46], `FocalLoss` from Lin et al. [98]. It contains a normalized VGG network for computing perceptual losses [39] with equal layer importance as suggested in [46]. We extend PyTorch's cross-entropy with `continuous_cross_entropy` allowing for non one-hot target distributions or `smoothed_cross_entropy` for cross-entropy with label smoothing [147].

**`torchelie.lr_scheduler`** complements `torch.optim.lr_scheduler`, providing linear decay [95] (with or without warmup), a configurable `CurriculumScheduler`, `OneCycle` from Smith [142], and `HyperbolicTangentDecay` from Hsueh et al. [69].

**`torchelie.optim`** implements optimizers that I wanted to experiment with. It contains `DeepDreamOptim` used for Mordvintsev et al. [113, 114], `AddSign` from Bello et al. [11], `AdaBelief` a variant of Adam from Zhuang et al. [185], `RAdam` from Liu et al. [100] and `Lookahead` from Zhang et al. [181].

**`torchelie.transforms`** contains reimplementations of usual transforms in order to make them differentiable for Karras et al. [78] and other data augmented GANs, `RandAugment` from Cubuk et al. [29], `TrivialAugment` from Müller and Hutter [115] and other minor utilities.

**`torchelie.utils`** contains many utility functions, some for weights initialization, accessing layers / weights by name, computing things like Gram matrices, linear interpolation, or distributed training helpers.

**`torchelie.nn`** is one of the biggest part of `torchelie` and contains layers and blocks from various papers and architectures. Listing them all would be tedious and not very informative. We should say that it contains the `VQ` as a layer transparently handling backpropagation, layers for `StyleGAN2` [78], `PixelCNN(++)` [83, 137], blocks for `ResNets` and `ResNet`-likes [59, 173], `SE` blocks [70], and some more utilities. An original addition is the `ModuleGraph` allowing to describe a model as a computational graph.

**`torchelie.models`** reimplements `AlexNet` [88], `Attention-56` [161], `AutoGAN`'s generator and discriminator [48], `EfficientNet` [149], `hourglasses` from [154], `MLP-Mixer` [153], `Pix2Pix` and `Pix2PixHD`'s generator and discriminator [72], `PixelCNN` [83], `ResNets` [59, 60, 173, 179, 58], `SNGAN`'s discriminator [111], `StyleGAN2`'s generator and discriminator [78] and `UNets`. Contrarily to all general purpose known libraries to date, `Torchélie` would be the first one to propose models

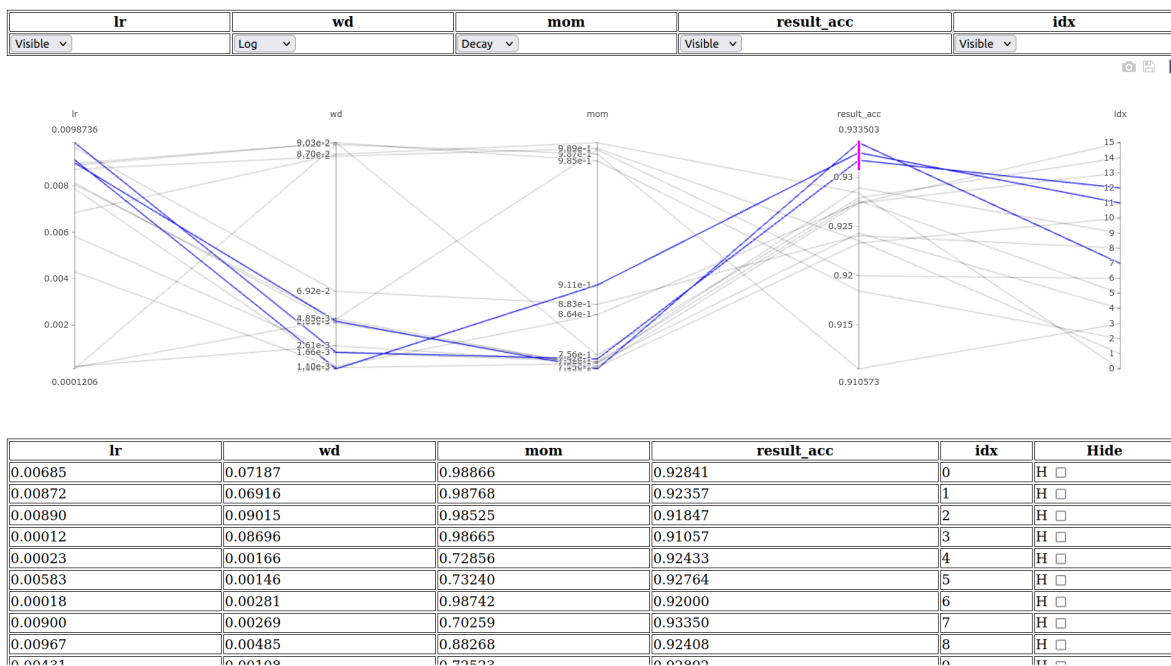


Figure 8.1: Visualization of `torchelie.hyper` hyperparameter search. The user can select hyper parameters to sample (and how to sample them), and target metrics. Once ran, the results appear in this visualization. In this case, we highlighted via the interface the three runs with the best resulting accuracy.

pretrained on other datasets than ImageNet, by embedding, in a near future, models pretrained on face recognition and face detection tasks. We will talk more about model design in Section 8.3.1.

Torch elie also contains utilities that are not within PyTorch’s scope:

`torchelie.nn.utils` is part of the different design philosophy and contains utilities aimed to edit models, in order to insert, remove, or replace layers, activations, etc.

`torchelie.data.learning` provides tools to infer data via backpropagation, used for instance in style transfer or feature visualization. It provides learnable images, in pixel space or fourier space, with uncorrelated color space or not.

`torchelie.hyper` provides tools for automatic hyperparameter exploration. It allows random search or search guided with Gaussian Processes, and provides a visualization of the results (Figure 8.1).

`torchelie.recipes` a model is useless without training and inference code. Recipes are Torch elie’s ways of building programs utilizing models. We provide ready to use Recipes such as CUT [123], DeepDream [113], Feature visualization for CNNs [120], vanilla GAN training [49], image manipulations algorithms from Ulyanov et al. [154], neural style transfer [46], Pix2Pix [72], StyleGAN2 [78], and standard cross-entropy classification. Most of those recipe provide a commandline interface to complement their Python API.

**torchelie.callbacks** are callbacks that can be inserted in recipes in order to instrument them with metrics, logging, additional steps, etc. They are a very powerful tool keeping the code modular and clean. We provide `Checkpoint` that saves models on disk during training, `ClassificationInspector` providing live visualization for classification models (Figure 8.2), `ConfusionMatrix` show in Figure 8.3, various averaging strategies for logging metrics, `GANMetrics` (providing KID [15], FID [64], precision-recall [89]), `ImageGradientVis` showing the gradient magnitude of the loss wrt the input image, effectively showing the image parts that the model relied on (Figure 8.4), `Polyak` (exponential) averaging of models used for instance in StyleGAN2, `Throughput` displaying the number of forward passes done per second during training in order to keep models' latency under scrutiny, `Optimizer` running an optimizer step with configurable features (such as gradient accumulation, lr logging, gradient centralization, gradient clipping), `LRSched` for lr schedulers, and some more utilities.



Figure 8.2: The `ClassificationInspector` allows to see live the performance of the classifier. It reports the samples that are provide the best, worst, and most confused answers from the classifier. The bar below the images is green when the prediction is correct, red otherwise; the width reflects the confidence score of the prediction. This allows eyeballing the datasets, strengths and weaknesses of the model, and build intuition.

### 8.3 Design Principles

The second contribution Torch lie proposes is a set of design patterns thought to be convenient for research and modular / incremental development and experimentation. Torch lie follows few design principles:

1. **Independent** As much as possible, features must be independent and one must be able to use exactly the feature they need without difficulty. For instance, one must be able to use a layer or a model in their project and use them as much as possible as Pytorch components in their Pytorch project. One might be able to train a vanilla Pytorch model with a training loop from Torch lie without any adaptation.

| Pred\True  | airplane | automobile | bird | cat  | deer | dog  | frog | horse | ship | truck |
|------------|----------|------------|------|------|------|------|------|-------|------|-------|
| airplane   | 2071     | 2223       | 2311 | 2126 | 2205 | 2325 | 1721 | 2200  | 2211 | 1941  |
| automobile | 458      | 126        | 325  | 118  | 357  | 125  | 192  | 140   | 429  | 104   |
| bird       | 3        | 0          | 1    | 1    | 0    | 0    | 2    | 0     | 3    | 1     |
| cat        | 310      | 636        | 1018 | 1160 | 1382 | 1107 | 1862 | 1275  | 210  | 665   |
| deer       | 1        | 0          | 0    | 3    | 2    | 2    | 2    | 0     | 0    | 0     |
| dog        | 6        | 14         | 8    | 4    | 3    | 3    | 6    | 2     | 10   | 6     |
| frog       | 61       | 141        | 203  | 214  | 319  | 138  | 584  | 140   | 33   | 77    |
| horse      | 1        | 0          | 2    | 1    | 1    | 2    | 1    | 1     | 0    | 0     |
| ship       | 1942     | 1808       | 1094 | 1342 | 688  | 1277 | 607  | 1229  | 1866 | 2163  |
| truck      | 147      | 52         | 38   | 31   | 43   | 21   | 23   | 13    | 238  | 43    |

Figure 8.3: Live confusion matrix provided automatically when the number of classes is not too big to make it unreadable (less than 25 classes).

2. **Compositionality** Features are built by composing small and independent building blocks. All blocks must do one thing and do it well. Greater components must be designed by combining smaller components and keep the logic simple. The user should be able to easily write their own components and easily compose them to the set of feature that is looked for. For instance, it must be easy to write a new type of layer and use it in the provided models with minimal effort; to write a new metric to integrate in the training loop; to write a new training logic code; etc. This is, for instance, why our implementation of the VQVAE embodies its gradients estimator in its backward function: using this layer is now a totally local decision, there is no need to alter the code somewhere else, like in the loss function, to make it usable.
3. **Build standard and modify** This *delta approach* is the new take Torch lie proposes on developing libraries. Trying to build models, training recipes, layers, with options for customization inside the object constructors leads to two kind of major issues. First, despite the best efforts, since it is very hard to know what the developer will want to control, it is unlikely that our parameters would address every implementation details. Second, the code either lacks customizability or becomes unmaintainable as one adds switches and parameters. Moreover, each layer option must be ported as well to the model's options, and the parameters and switches grow totally out of hand. In training code, this happens the same way as one might want to change a loss function, add one, etc. Instead, we should provide simple ways to build standard components, and give editing features. Instead of building customized, it is hypothesized that thinking in terms of deltas from the standard blocks scales better: the construction code remain simple and straightforward as it builds just one standard thing, and the edition functions remain external and self



Figure 8.4: Gradient of the loss wrt the input on the current batch. The per-pixel norm of the gradient weighs each pixel's intensity. This helps figuring out what the model looks at in the picture in order to make its predictions

contained.

Training code should follow the same guidelines. For instance, complex losses such as the Pix2pixHD loss should be simple but composable so that a user can easily add, remove, change or replace one term.

A deep learning practitioner mainly works in incremental changes, as we evolve from a standard algorithm and evaluate their performance; let's keep these deltas expressed as deltas in the code as well.

### 8.3.1 Models

Torchélie provide many architectures, that are yet to be pretrained. It provides many resnet variants and GANs architectures such as Pix2pix or StyleGAN2.

It becomes straightforward to write code in Torchélie style. The model or block is implemented in its most basic, canonical form. Derivative models are implemented in terms of transforms from those basic blocks.

Let's illustrate this first with a simple example, without using Torchélie. One wants to experiment with SE-ResNets [70]. In simple terms, the SE-ResNet is a ResNet which adds a Squeeze-and-Excite block in every residual layers.

A typical way to implement this, shown in Listing 8.1, would be invasive changes to the ResNet definition. Implementing more and more variants would bloat the code and make it harder and harder to maintain as the code grows with switches.

Instead, we repeat that *the SE-ResNet is a ResNet which adds a Squeeze-and-Excite block in every residual layers*. This is exactly the way this should be expressed in code. The resulting code, in Listing



```

1 class ResNet(nn.Module):
2     def __init__(self, arch, use_se=False):
3         # create input stem
4
5         for n_blocks in arch:
6             for i in range(n_blocks):
7                 self.add_module(ResBlock(..., use_se=use_se))
8         # create end of trunk
9
10    class ResBlock(nn.Module):
11        def __init__(self, ..., use_se=False):
12            # create branch and skip convs if any
13
14            if use_se:
15                self.branch.add_module(SE(...))
16
17    se_resnet = ResNet(..., use_se=True)

```

Listing 8.1: Typical SE-ResNet implementation. Note how `use_se` has to be passed down. Details are left out for clarity and brevity.

```

1 resnet = ResNet(...)
2
3 for m in resnet.modules():
4     if isinstance(m, ResBlock):
5         m.branch.append(SE(...))

```

Listing 8.2: Delta implementation of SE-ResNet. There are no needs for intrusive changes.

8.2, is shorter, more readable, local and non invasive. There are no risks inserting bugs into vanilla resnets that were perfectly functioning before our intervention.

This is not cherry picking, and examples are plenty:

- a Wide ResNet [179] is "a ResNet with wider kernels". We can take a resnet, iterate on its residual blocks and replace the layers with wider ones.
- ResNet-v2 [60] has "a ResNet with three 3x3 convs in the input stem", "a ResNet where stride 2 convolutions are the 2nd one in the branch rather than the first one", and "a ResNet that uses average pooling rather than strides in the identity path".
- ResNeXt [173] uses "grouped convolutions instead of full convolutions".
- PoolFormer [177] is a Vision Transformer [40] "using average pooling instead of self attention" [158].
- ReZero [8] "adds a learnable scalar multiplier, initialized to 0, at the end of every residual branch".
- etc...

Those are deltas that are easily expressed in code, and more explicitly exhibit the intent. Those deltas are fundamental for research and iterative development. Having those deltas expressed as *external* transforms rather than *intrusive* instrumentation is very important. To some extent, it allows modifying an architecture provided by a library without needing to alter its code. It ensures that the

new delta the user is about to try does not introduce bugs into previous models and does not interfere with any other part of the code. It scales trivially as incremental changes can be layered in the exact same way, or explored in parallel, without introducing uncontrollably growing complexity with switches.

The role of **Torchélie** is to ease delta programming. This is first done by providing carefully designed interfaces:

1. Models are totally described as editable computational graphs (`nn.Sequential` or `torchelie.nn.ModuleGraph`). They can be externally modified. For this, we avoid hard coded logic and hard coded design choices (there are few exceptions to this, decided with common sense). Actually, the SE-ResNet example could not have been done using `torchvision`'s definition because the contents of the residual branch is not editable in their implementation.
2. Layers bear a semantic name, so that they can be easily and robustly indexed.
3. Layers that form a semantic units are grouped, so that models are not a flat sequence of fundamental operations, but a (fully editable) hierarchy exhibiting design choices; for instance, Conv-BN-ReLU are grouped together in a `torchelie.nn.ConvBlock`.

Second, Torchélie provides utilities easing delta edition. For instance, in `torchelie.nn.utils`, we provide:

**insert\_after**(`base`, `key`, `new`, `new_name`) that inserts `new` with name `new_name` into model `base` after the module indexed by `key`

**insert\_before**(`base`, `key`, `new`, `new_name`) works similarly

**make\_leaky**(`m`) that changes ReLUs in `m` into LeakyReLUs and accordingly adapts the initialization the layer before it (if applicable)

**edit\_model**(`model`, `f`) that recursively transforms every module `m` of `model` to `f(m)`, etc.

### 8.3.2 Algorithms

Those design guidelines apply to algorithm implementation as well. This is not as straightforward as for models since we are already used to models being described as data structures. For code, we need carefully designed code architecture.

Let's study examples of how that would work out on concrete cases.

- StyleGAN2 [78] (besides architectural changes) "adds a Perceptual Path Length regularizer and a R1 regularizer to a standard NSGAN".
- StyleGAN2-ADA [77] "inserts differentiable data augmentation before feeding real and fake images to the discriminator".
- ResNet-RS [12] propose to improve resnets by "architecture improvements" (dealt with model delta programming) and "better regularization methods: adding model exponential averaging [124] (Polyak averaging), label smoothing [147], RandAugment [29] and a lower weight decay".
- Similarly, ResNet-SB proposes to "replace cross-entropy with BCE, add mixup [180] and cutmix [178], use the LAMB optimizer [175] with bigger batch size, and a few other hyper parameter differences"

```

1 (aa=None, amp=False, apex_amp=False, aug_splits=0, batch_size=32, bn_eps=None,
  bn_momentum=None, bn_tf=False, channels_last=False, clip_grad=None,
  color_jitter=0.4, cooldown_epochs=10, crop_pct=None, cutmix=0.0, cutmix_minmax=
  None, data_dir='../imagenette2-320', dataset='', decay_epochs=30, decay_rate
  =0.1, dist_bn='', drop=0.0, drop_block=None, drop_connect=None, drop_path=None,
  epochs=200, eval_metric='top1', gp=None, hflip=0.5, img_size=None,
  initial_checkpoint='', input_size=None, interpolation='', jsd=False, local_rank
  =0, log_interval=50, lr=0.01, lr_cycle_limit=1, lr_cycle_mul=1.0, lr_noise=None
  , lr_noise_pct=0.67, lr_noise_std=1.0, mean=None, min_lr=1e-05, mixup=0.0,
  mixup_mode='batch', mixup_off_epoch=0, mixup_prob=1.0, mixup_switch_prob=0.5,
  model='resnet101', model_ema=False, model_ema_decay=0.9998, model_ema_force_cpu
  =False, momentum=0.9, native_amp=False, no_aug=False, no_prefetcher=False,
  no_resume_opt=False, num_classes=None, opt='sgd', opt_betas=None, opt_eps=None,
  output='', patience_epochs=10, pin_mem=False, pretrained=False, ratio=[0.75,
  1.3333333333333333], recount=1, recovery_interval=0, remove='const', reprob
  =0.0, resplit=False, resume='', save_images=False, scale=[0.08, 1.0], sched='
  step', seed=42, smoothing=0.1, split_bn=False, start_epoch=None, std=None,
  sync_bn=False, torchscript=False, train_interpolation='random', train_split='
  train', tta=0, use_multi_epochs_loader=False, val_split='validation',
  validation_batch_size_multiplier=1, vflip=0.0, warmup_epochs=3, warmup_lr
  =0.0001, weight_decay=0.0001, workers=4)

```

Listing 8.3: Arguments to the training script of `timm`.

- Pix2Pix [72] ”adds a L1 per pixel loss to the adversarial loss of a standard cGAN [110]”.
- Pix2PixHD [166] ”removes the L1 pixel loss but adds a feature matching loss and changes the NSGAN loss to a LSGAN loss”.
- etc

Again, all those paper are defined in deltas from another paper or standard procedure. And even if they propose significant changes (like ResNet-SB), they came to those lists of changes by incremental trials, building deltas one on top of another, often made explicit through ablation studies. ConvNeXt [102] is a very demonstrative example showing how research advances by stacking deltas (cf. Figure C.1).

The reasoning is the same as for handling model variants: as one adds options and wants to experiment, the code grows out of hands, and all those intrusive changes risk introducing regression bugs. Listing 8.3 shows how `timm` [170], a famous library providing vision models and their training code, grew an enormous list of parameters by not following a delta approach. Developing new ideas outside of what is permitted by those is not possible without intrusive intervention inside `timm`’s code. However, humility is needed as `timm` sprung many works and supports a much much larger number of projects than Torchélie does.

Torchélie proposes tooling in order to describe code as data structures and ease delta programming of algorithms. Our currently proposed tool is the `Algorithm` class. It is an improved sequence of named functions. Those functions are called in order when executing the `Algorithm`. The names allow to easily manipulate the sequence in order to replace or remove existing functions, or insert new ones. That way, algorithms can be described in atomic steps that can be incrementally manipulated to grow more sophisticated or fork variants.

Let’s illustrate how a standard conditional GAN algorithm can be manipulated to create new derivative algorithms, namely WGAN-GP, Pix2Pix and Pix2PixHD with small deltas.

```

1 class ConcatConditionalGANLoss:
2     def __init__(self, G: nn.Module, D: nn.Module) -> None:
3         self.G = G
4         self.D = D
5         self.gan_loss = tch.loss.gan.standard
6
7         G_alg = Algorithm()
8
9         @G_alg.add_step('fake')
10        def G_fake_pass(env, real, target):
11            # Generate some fakes
12
13        @G_alg.add_step('adversarial')
14        def G_adv_pass(env, real, target):
15            # compute the discriminator loss
16
17        @G_alg.add_step('backward')
18        def backward(env, real, target):
19            # backward the discriminator loss
20
21        self.G_alg = G_alg
22
23        ... # D's pass to be implemented here

```

Listing 8.4: The generator pass for a ConcatConditionalGAN. The implementation details are removed in order to focus on the design principles. The discriminator’s pass is in Listing 8.5

A conditional GAN concatenates the source and target image to the discriminator. The Generator (G) pass is made in two steps: generate some fakes and compute the loss; then backpropagate to the generator. Listing 8.4 illustrates our delta programming approach using `Algorithm` for the generator. We can see that a conditional GAN loss is defined as a pass for G and the Discriminator (D). G’s include one ‘fake’ step, generating fake samples, ‘adversarial’ discriminating those fakes, and ‘backward’ running the back propagation. Same goes for D.

Listing 8.5 illustrates the steps involved in training the discriminator: ‘gen\_fakes’ generates fake samples from G, ‘fake\_adversarial’ computes the discriminator’s output on them, ‘fake\_backward’ backpropagates the fake loss, ‘real\_adversarial’ computes the discriminator’s output on real samples and ‘real\_backward’ backpropagates the loss for them. These sequences of operations are now manipulable as data from `G_alg` and `D_alg` members.

A first variant, a conditional WGAN, could use the WGAN-GP regularizer [51]. This could be achieved by adding a gradient penalty computation before performing the backpropagation on D, as shown in Listing 8.6. We first make sure that the GAN’s `Algorithm` is using a standard BCE loss, we define our gradient penalty, and we insert a new step in the algorithm ‘D\_gp’, right before the ‘real\_adversarial’ step. The actual implementation of the gradient penalty is totally irrelevant to the point shown here: our cWGAN-GP is grown by adding a simple term, a simple delta, to our vanilla cGAN.

We can try further and implement Pix2Pix’s loss instead of WGAN-GP. Pix2Pix extends the standard conditional GAN with a L1 loss between the generated picture and the actual target picture. Inheritance can achieve this and is another valid way of implementing our goal. Listing 8.7 exhibits this implementation, adding the L1 term to our vanilla cGAN, the ‘l1’ step, after ‘real\_adversarial’.

Pix2PixHD can be implemented as a derivative work that mostly incorporates a feature matching loss to the standard conditional GAN setting and switches to the Least-Squares GAN loss they

```

1 class ConcatConditionalGANLoss:
2     def __init__(self, G: nn.Module, D: nn.Module) -> None:
3         ... # G's pass to be implemented here
4
5         D_alg = Algorithm()
6
7         @D_alg.add_step('gen_fakes')
8         def gen_fakes(env, real, target):
9             # Generate fake images, store them in env['fake_image']
10
11        @D_alg.add_step('fake_adversarial')
12        def D_fake(env, real, target):
13            # Compute the discriminator loss for fake samples.
14
15        @D_alg.add_step('fake_backward')
16        def D_backward(env, real, target):
17            # Backpropagate the fake loss. Return some metrics
18
19        @D_alg.add_step('real_adversarial')
20        def D_real(env, real, target):
21            # Compute the discriminator loss for real samples.
22
23        @D_alg.add_step('real_backward')
24        def D_backward(env, real, target):
25            # Backpropagate the real loss. Return some metrics
26
27        self.D_alg = D_alg
28
29        def G_step(self, real: torch.Tensor, target: torch.Tensor) -> dict:
30            return self.G_alg(real, target)
31
32        def D_step(self, real: torch.Tensor, target: torch.Tensor) -> dict:
33            return self.D_alg(real, target)

```

Listing 8.5: The discriminator for a ConcatConditionalGAN. The implementation details are removed in order to focus on the design principles. The generator pass is in Listing 8.4

```

1 def to_wgangp(cgan, gp_gamma):
2     cgan.gan_loss = tch.loss.gan.standard
3     gradient_penalty = GradientPenalty(gp_gamma)
4
5     @cgan.D_alg.insert_before('real_adversarial', 'D_gp')
6     def D_gp(env, real, target):
7         g_norm = gradient_penalty(cgan.D, env['real_pair'], env['fake_pair'])
8         return {'g_norm': g_norm}

```

Listing 8.6: ConcatConditionalGAN with WGAN-GP regularizer

```

1 class Pix2PixLoss(ConcatConditionalGANLoss):
2     def __init__(self, G: nn.Module, D: nn.Module, l1_gain: float) -> None:
3         super().__init__(G, D)
4         self.l1_gain = l1_gain
5
6         @self.G_alg.insert_after('real_adversarial', 'l1')
7     def G_l1_pass(env, real, target):
8         loss = self.l1_gain * F.l1_loss(env['fake_image'], target)
9         env['loss'] += loss
10        return {'l1_loss': loss.item()}

```

Listing 8.7: Pix2Pix from ConcatConditionalGAN

```

1 class Pix2PixHDLoss(ConcatConditionalGANLoss):
2
3     def __init__(self, G: nn.Module, D: nn.Module, l1_gain: float):
4         super().__init__(G, D)
5         self.gan_loss = tch.loss.gan.ls
6         self.l1_gain = l1_gain
7
8         D_with_acts = tnn.WithSavedActivations(D.module)
9
10        @self.G_alg.insert_before('fake_adversarial', 'extract_features')
11    def G_features(env, real, target):
12        # Generate fake samples, compute D's activations and its final
13        # predicted probability
14
15        @self.G_alg.insert_after('extract_features', 'feature_matching')
16    def G_featmatch(env, real, target):
17        # Compute the feature matching loss
18
19        @self.G_alg.override_step('fake_adversarial')
20    def G_adv(env, real, target):
21        # add the actual adversarial loss

```

Listing 8.8: Pix2PixHD

find more stable. Listing 8.8 implements Pix2PixHD’s loss by adding ‘extract\_features’ and ‘feature\_matching’ before ‘fake\_adversarial’, which extract deep features from the discriminators, compute the deep feature matching loss. It finally replaces ‘fake\_adversarial’ in order to compute the discriminator loss from the features instead of the default step that runs a full forward pass on D, saving compute.

We believe this programming paradigm highlights the incremental nature of the works, and make them easily manipulable. Easy experiments can be conducted if one wishes to remove, replace, or add elements to losses or more complex algorithms.

Torchélie’s Algorithm class is certainly not a silver bullet, but while we believe our implementation has flaws and certainly has cases that it does not handle in a comfortable and elegant way, we are confident that this delta programming is an important paradigm for research and deep learning work in general.

## 8.4 Discussion

### 8.4.1 The future of Torch  lie

It first important to remind that Torch  lie is a tool tailored to and built from *my* needs but could be useful to the community thanks to its MIT license. Having a bigger user base and more regular contributors would be very nice, but that would make modifying the library to adapt my needs harder and slower. I would now have to worry about API stability, breaking changes, etc. This *is* envisioned, but for now, I consider some Torch  lie parts still unstable and in alpha, often modified with design changes that would invalidate work based on it. The main cause of instability is the ongoing exploration for better tools to help delta programming for algorithms. Actually, the algorithms / recipes implemented *before* the rise of the delta paradigm in Torch  lie will be rewritten, and their current API will be broken. Delta programming for models is already stable and fully functional.

The library is also in need of a new scope definition. Initially, I saw Torch  lie as "everything computer vision", which is now unsustainable. While it was doable at its birth to follow along the major publications in order to implement a wide range of architectures, this is not doable anymore. Torch  lie needs to focus on its *raison-d'  tre*, which is research for applied industrial problems and its constraints: models trainable in a small amount of time on commodity hardware, with small or moderately sized datasets (fine tuning is another topic). This means that Torch  lie only needs to implement models, tools and algorithms that passed the hard test of time, instead of the latest trend; applied to Vision Transformers [40] for instance this means implementing only the milestone models that resisted industrial testing, time, and are commonly used in benchmarks. It certainly means lagging behind, but it also means providing healthy defaults.

This project has no end, but the near future to-do list contains a lot already. First and foremost, Torch  lie has focused a lot on convolutional architectures which has become less and less relevant: ResNets work well, and every other architecture I tested and implemented did not satisfy me more than ResNets. Some were just slower, and others did not deliver the expected gains. That said, another component that has been overlooked in Torch  lie is in great need of implementation: the library is not up to date with modern training settings and need to implement those in convenient ways. Besides, as Python is a quite volatile language due to its dynamic typing, annotating type hints and checking them with `mypy` [1] is an ongoing work.

### 8.4.2 The competition

Torch  lie lives in the land of deep learning libraries, having already many competitors. The use case for Torch  lie could be questioned now that other libraries such as `timm` [170], `Fastai` [67], `Ignite` [43], or `PyTorch-Lightning` are getting traction.

1. `timm` provides a very large quantity of models, most of which are also pretrained on ImageNet. However, its scope remains focused on training image classifiers.
2. `Fastai` is closer in its scope to Torch  lie. `Fastai` has some different design choices that makes it feel like learning an entirely novel library. `Fastai` mostly sits on top of `Pytorch` and bring its own design principles, rather than extending `PyTorch` horizontally.
3. Finally, `Ignite` and `PyTorch-Lightning` are source of inspiration and close to what I envision. Torch  lie tries to be more task oriented at the cost of flexibility, and fit my needs better.

In the long run, the broad scope of Torch  lie does not seem sustainable and should either narrow its scope (which has been discussed) or integrate at least parts of the aforementioned libraries in order

to remain relevant with a sustainable work load. The field is advancing at an increasing pace and more means are needed today to keep up with the pace than when it started.

## 8.5 Conclusion

In this chapter, we presented Torch elie, a PyTorch library gathering the code (layers, losses, algorithms, training loops, etc) under a consistent and coherent Python package. We presented the double contribution that Torch elie is: 1) a toolbox for the deep learning practitioner (especially in the domain of computer vision) with a brief presentation of its content, and 2) a programming paradigm that we call *Delta programming* encouraging code and models designed as data structures for extrusive and incremental modifications; illustrated with a GAN example.

We discussed the work that remained to be done on Torch elie in order to keep it relevant in the near future and more easily maintainable while maximizing its usefulness. I expressed my decision to limit the scope of new additions to models and algorithms that succeed the test of time and invest on them rather than chasing every new development in the literature. This is undoable and brings little value in the not so uncommon case where there are no influential subsequent work in my line of work. This would also help finding Torch elie’s place in a rapidly growing ecosystem, maybe reusing and delegating some of its component to other packages in the PyTorch landscape like Albumentations [20] for image augmentation.

The main thing that could benefit Torch elie is having a community. This would alleviate the weight of maintaining the code.

All in all, Torch elie is an invaluable tool for both my academic and industrial work. There is a virtuous double-way feedback loop at play: Torch elie helps my works and its use allows me to figure out the tools and designs needed in the library. This symbiotic flow also allows to proof test the code and ensure its correctness and performance.



## 9 Conclusion

During this thesis, we explored how Deep Learning applied to Computer Vision could bring value to a video platform. We started with a simple idea: extract semantic information from video content in order to improve user experience (through content categorization and searchability for instance) and content suggestion and personalization. This naturally lead to two main parts: extracting information from videos, and creating a recommender system. We settled on some information that would be interesting: activity classification could help categorize the content and choose a fitting thumbnail when showing search results, and face recognition for similar purposes. In a second time, we would analyze the challenges presented with a recommender system in our domain and propose a model.

The systems presented here are used in production or about to be, bringing value as improved user experience. This allows us to conclude on several points. First and foremost, deep learning is able to deliver on our data type and domain, which corroborates the large amount of testimonies of businesses positively impacted by the usage of deep learning.

**Semantic activity information** could be extracted and leveraged as see in Section 3.4. We could analyze the videos fairly easily and reliably enough for the way we want to exploit them. We were able to develop a model, evaluating several options including randomly initialized models, pretrained models and data augmentation. We show that data augmentation and pretrained models were able to significantly increase our accuracy up to a point where the model could be good enough for some non critical labeling, improving the user experience.

**Negative training samples** were leveraged with standard softmax classifiers in order to use so that our face recognition classifier becomes more robust to unknown people (Chapter 5). We first set the face recognition problem as a standard metric learning problem, showed that it does not scale for our problem. Since the people we want to recognized are known at training time, we simplified our problem down to a standard classifier in need an out-of-domain rejection ability. We compared several loss functions in order to implement this rejection capability: standard cross-entropy, cross-entropy with an additional out-of-domain class, cross-entropy augmented with a logit regularizer on distractors, and cross-entropy augmented with a maximum entropy objective for distractors. Each model improved on the previous one, the maximum entropy loss being the best we tried. We also showcased that all those models exhibit satisfying calibration, and the rejection threshold could be set by the production engineers in order to specify the error rate tolerance by trading recall for precision.

**Recommender system** were our subject of study in Chapter 7, when we have no user ratings but user browsing histories instead. We compared about fifty models trained on different feature subsets. It was shown that providing tag embeddings pretrained with Word2Vec lead to significant gains, making it one

of the most important features, followed by learnable uploader embeddings. This model needs to be A/B compared with the current system in production, based on manual heuristics.

**Torchélie** is a framework based on pytorch that was introduced in Chapter 8, underpinning all experiments and industrial developments done during the thesis. We presented how its design, based on deltas and patches, diverged from current famous frameworks. We implemented in it various contributions, including the proposed VQ layers. The pertinence of those design principles were supported by examples exhibiting how a standard resnet can be patched to produce its variants, and how a conditional GAN algorithm can be gradually patched to produce the Pix2Pix algorithm then Pix2PixHD.

Those achievements were accomplished thanks to **three datasets** that we put together, explained in Section 2.3. HActions is composed of frames extracted from videos, sorted into 13 classes, representing various activities. HFaces is our face recognition dataset, containing 8938 identities with an average of 50 pictures per identity. We ran our experiments on the 105 most popular identities, using the rest as distractors that have to be rejected by the classifier. Finally, we leveraged HHistory, the browsing history of our premium users for our recommendation engine. It contains several metadata, 3673 uploaders, 140k videos and 136k users. We searched for ways to grow those dataset in a principled ways in Section 5.5.4.

Besides those industry focused goals, we explored the metric learning framework and proposed the **Threshold-Softmax**, a new loss function able to learn from negative examples (in Section 5.4). The threshold-Softmax proposes to learn face embeddings fitting a cone with an absolute maximum angle, rather than imposing angular margins between classes. Negative samples are forced in the negative space: outside of the regions allocated for the positive classes. We experimented this loss on MS1Mv2 and compared it to the SotA ArcFace. The Threshold-Softmax is competitive but not always superior to ArcFace, but presents the ability to learn from unlabeled negative samples (unknown people not belonging to any positive class), halving the error rate in our tests on LFW and FGLFW.

During the ongoing exploration on generative models based on the VQ-VAE, we proposed to improve the efficiency and control of the quantization layer thanks to our **expiring codebook** in Section 6.11. An expiration mechanism is added to the codebook. When a code has not been used for more than a fixed number of training iterations, it is resampled to an input data point. This threshold is an hyper parameter that allows controlling the entropy of assignments. Experiments showed that this algorithm lead to better training dynamics that consistently outperformed the original VQ algorithm and trained faster.

We used this to build a **face swap** model (Section 6.12) based on a VQ bottleneck. A face picture is encoded, constrained with a VQ bottleneck, and decoded back. We also provide a learnable embedding of the identity label to the decoder. In order to produce an accurate reconstruction despite the bottleneck, the model has to learn as much as possible about facial geometry and store that knowledge in the embedding. The bottleneck can be used to transport the remaining information: colors, lighting, pose, etc. We showed that this model produces crisp pictures, and that the face swap is successful, actually fooling a face recognition system.

**Some future work** remain to be done and questions to be answered. We mainly need to put all those systems together and test whether the metadata extracted from videos are able to improve the recommender system. The impact on the face recognition system of the samples generated from the face swap algorithm is still to be evaluated. This VQ-VAE system can be compared to a GAN based model, as they are famous for the great image quality they produce. The recommender system has to be evaluated in production and A/B tested to assess its real world performance. Torchélie needs to find a community to maintain it and keep it up to date as the needs for deep learning grow too much for

a single developer. Most importantly, Torch lie must now get up to date with modern training recipes (involving Mixup, CutMix, TrivialAugment, etc) while mitigating the cost they incur, often needing longer training schedules or smaller batch sizes, if we want to keep our extreme applicability focus.

**The perspectives** that were opened by this work are numerous.

There could be many interesting ways to categorize videos. Instead of building big datasets for classifiers with human defined label sets, we could envision training a model with unsupervised techniques on our videos and use small datasets for few shot classification, allowing faster iterations on ways to categorize the data with respect to their business impact. Models like CLIP [127] could also be interesting for the zero-shot classification ability they provide, but our domain has no such model built for use, and no dataset of captioned images; could we develop a CLIP-like model with constrained data resources, or what would it take to fine tune CLIP?

The Threshold-Softmax can be extended to integrate ArcFace’s margins as well, maybe able to improve on it. Experiments should be lead to understand precisely how ArcFace utilizes its latent space and how Threshold-Softmax organizes the negative samples in the latent space. Those insights could help improving both, and maybe Metric Learning in general. The same detailed experiments could be done for the four losses we explored for negative training (vanilla cross-entropy, with a Discriminator class, penalizing the logits, maximizing the entropy).

Our face swap model lacks the ability to generalize to new identities, which is very important for data augmentation. Some work paved the ways in reversing GANs, which could be an inspiration to reverse the proposed architecture. This would allow editing pictures in latent space, and open a wide range of new possibilities.

Some of it is priority work as the industrial interests are highly convergent with this exploration.

## Bibliography

- [1] <http://mypy-lang.org/>, 2014.
- [2] Himan Abdollahpouri, Masoud Mansoury, Robin Burke, Bamshad Mobasher, and Edward Malt-house. User-centered evaluation of popularity bias in recommender systems. In *Proceedings of the 29th ACM Conference on User Modeling, Adaptation and Personalization, UMAP '21*, page 119–129, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450383660. doi: 10.1145/3450613.3456821. URL <https://doi.org/10.1145/3450613.3456821>.
- [3] Sami Abu-El-Haija, Nisarg Kothari, Joonseok Lee, Apostol (Paul) Natsev, George Toderici, Balakrishnan Varadarajan, and Sudheendra Vijayanarasimhan. Youtube-8m: A large-scale video classification benchmark. In *arXiv:1609.08675*, 2016. URL <https://arxiv.org/pdf/1609.08675v1.pdf>.
- [4] Alexander Amir Alemi, Ian S. Fischer, Joshua V. Dillon, and K. Murphy. Deep variational information bottleneck. *ArXiv*, abs/1612.00410, 2017.
- [5] Ehsan Amid, Manfred K. Warmuth, Rohan Anil, and Tomer Koren. Robust bi-tempered logistic loss based on bregman divergences. *ArXiv*, abs/1906.03361, 2019.
- [6] Anthreas Antoniou, Amos Storkey, and Harrison Edwards. Data augmentation generative adversarial networks, 2018. URL <https://openreview.net/forum?id=S1Auv-WRZ>.
- [7] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *International conference on machine learning*, pages 214–223. PMLR, 2017.
- [8] Thomas Bachlechner, Bodhisattwa Prasad Majumder, Henry Mao, Gary Cottrell, and Julian McAuley. Rezero is all you need: Fast convergence at large depth. In *Uncertainty in Artificial Intelligence*, pages 1352–1361. PMLR, 2021.
- [9] Andrew R Barron. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information theory*, 39(3):930–945, 1993.
- [10] Alan Joseph Bekker and Jacob Goldberger. Training deep neural-networks based on unreliable labels. *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2682–2686, 2016.
- [11] Irwan Bello, Barret Zoph, Vijay Vasudevan, and Quoc V Le. Neural optimizer search with reinforcement learning. In *International Conference on Machine Learning*, pages 459–468. PMLR, 2017.

- [12] Irwan Bello, William Fedus, Xianzhi Du, Ekin D Cubuk, Aravind Srinivas, Tsung-Yi Lin, Jonathon Shlens, and Barret Zoph. Revisiting resnets: Improved training and scaling strategies. *arXiv preprint arXiv:2103.07579*, 2021.
- [13] J. Bennett and S. Lanning. The netflix prize. In *Proceedings of the KDD Cup Workshop 2007*, pages 3–6, New York, August 2007. ACM. URL <http://www.cs.uic.edu/~liub/KDD-cup-2007/NetflixPrize-description.pdf>.
- [14] David Berthelot, Nicholas Carlini, Ian G Goodfellow, Nicolas Papernot, Avital Oliver, and Colin Raffel. Mixmatch: A holistic approach to semi-supervised learning. *ArXiv*, abs/1905.02249, 2019.
- [15] Mikołaj Bińkowski, Danica J Sutherland, Michael Arbel, and Arthur Gretton. Demystifying mmd gans. In *International Conference on Learning Representations*, 2018.
- [16] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.
- [17] Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. Lof: Identifying density-based local outliers. In *SIGMOD Conference*, 2000.
- [18] Andrew Brock, Theodore Lim, James Millar Ritchie, and Nicholas J Weston. Neural photo editing with introspective adversarial networks. In *5th International Conference on Learning Representations 2017*, 2017.
- [19] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale GAN training for high fidelity natural image synthesis. In *International Conference on Learning Representations*, 2019. URL <https://openreview.net/forum?id=B1xsqj09Fm>.
- [20] Alexander Buslaev, Vladimir I. Iglovikov, Eugene Khvedchenya, Alex Parinov, Mikhail Druzhinin, and Alexandr A. Kalinin. Albumentations: Fast and flexible image augmentations. *Information*, 11(2), 2020. ISSN 2078-2489. doi: 10.3390/info11020125. URL <https://www.mdpi.com/2078-2489/11/2/125>.
- [21] Q. Cao, Li Shen, Weidi Xie, Omkar M. Parkhi, and Andrew Zisserman. Vggface2: A dataset for recognising faces across pose and age. *2018 13th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2018)*, pages 67–74, 2018.
- [22] Òscar Celma Herrada et al. *Music recommendation and discovery in the long tail*. Universitat Pompeu Fabra, 2009.
- [23] Xi Chen, Yan Duan, Rein Houthoofd, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, pages 2180–2188, 2016.
- [24] Xi Chen, Nikhil Mishra, Mostafa Rohaninejad, and Pieter Abbeel. Pixelsnail: An improved autoregressive generative model. In *International Conference on Machine Learning*, pages 864–872. PMLR, 2018.
- [25] Jan Chorowski, Nanxin Chen, Ricard Marxer, Hans Dolfing, Adrian Łańcucki, Guillaume Sanchez, Tanel Alumäe, and Antoine Laurent. Unsupervised neural segmentation and clustering for unit discovery in sequential data. In *NeurIPS 2019 workshop-Perception as generative reasoning-Structure, Causality, Probability*, 2019.

- [26] Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM conference on recommender systems*, pages 191–198, 2016.
- [27] Antonia Creswell and Anil Anthony Bharath. Inverting the generator of a generative adversarial network. *IEEE transactions on neural networks and learning systems*, 30(7):1967–1974, 2018.
- [28] Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. Autoaugment: Learning augmentation strategies from data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 113–123, 2019.
- [29] Ekin D Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V Le. Randaugment: Practical automated data augmentation with a reduced search space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 702–703, 2020.
- [30] Maurizio Ferrari Dacrema, Simone Boglio, P. Cremonesi, and D. Jannach. A troubling analysis of reproducibility and progress in recommender systems research. *ACM Transactions on Information Systems (TOIS)*, 39:1 – 49, 2021.
- [31] Zihang Dai, Hanxiao Liu, Quoc V Le, and Mingxing Tan. Coatnet: Marrying convolution and attention for all data sizes. *arXiv preprint arXiv:2106.04803*, 2021.
- [32] Stéphane d’Ascoli, Hugo Touvron, Matthew L. Leavitt, Ari S. Morcos, Giulio Biroli, and Levent Sagun. Convit: Improving vision transformers with soft convolutional inductive biases. *CoRR*, abs/2103.10697, 2021. URL <https://arxiv.org/abs/2103.10697>.
- [33] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.
- [34] Jiankang Deng, J. Guo, and S. Zafeiriou. Arcface: Additive angular margin loss for deep face recognition. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4685–4694, 2019.
- [35] Weihong Deng, Jiani Hu, Nanhai Zhang, Binghui Chen, and Jun Guo. Fine-grained face verification: Fglfw database, baselines, and human-dcmn partnership. *Pattern Recognition*, 66:63–73, 2017.
- [36] Terrance DeVries and Graham W Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017.
- [37] Jeff Donahue and Karen Simonyan. Large scale adversarial representation learning. *Advances in Neural Information Processing Systems*, 32:10542–10552, 2019.
- [38] Jeff Donahue, Philipp Krähenbühl, and Trevor Darrell. Adversarial feature learning. *arXiv preprint arXiv:1605.09782*, 2016.
- [39] Alexey Dosovitskiy and Thomas Brox. Generating images with perceptual similarity metrics based on deep networks. *Advances in neural information processing systems*, 29:658–666, 2016.
- [40] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2020.

- [41] DC Dowson and BV Landau. The fréchet distance between multivariate normal distributions. *Journal of multivariate analysis*, 12(3):450–455, 1982.
- [42] Patrick Esser, Robin Rombach, and Bjorn Ommer. Taming transformers for high-resolution image synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12873–12883, 2021.
- [43] V. Fomin, J. Anmol, S. Desroziere, J. Kriss, and A. Tejani. High-level library to help with training neural networks in pytorch. <https://github.com/pytorch/ignite>, 2020.
- [44] Benoît Fréney and Michel Verleysen. Classification in the presence of label noise: A survey. *IEEE Transactions on Neural Networks and Learning Systems*, 25:845–869, 2014.
- [45] Simon Funk. Netflix update: Try this at home, 2006.
- [46] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2414–2423, 2016.
- [47] Golnaz Ghiasi, Honglak Lee, Manjunath Kudlur, Vincent Dumoulin, and Jonathon Shlens. Exploring the structure of a real-time, arbitrary neural artistic stylization network. *Proceedings of the British Machine Vision Conference 2017*, 2017. doi: 10.5244/c.31.114. URL <http://dx.doi.org/10.5244/C.31.114>.
- [48] Xinyu Gong, Shiyu Chang, Yifan Jiang, and Zhangyang Wang. Autogan: Neural architecture search for generative adversarial networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3224–3234, 2019.
- [49] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.
- [50] Shanyan Guan, Ying Tai, Bingbing Ni, Feida Zhu, Feiyue Huang, and Xiaokang Yang. Collaborative learning for faster stylegan embedding. *arXiv preprint arXiv:2007.01758*, 2020.
- [51] Ishaan Gulrajani, Faruk Ahmed, Martín Arjovsky, Vincent Dumoulin, and Aaron C. Courville. Improved training of wasserstein gans. In *NIPS*, 2017.
- [52] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. On calibration of modern neural networks. In *International Conference on Machine Learning*, pages 1321–1330. PMLR, 2017.
- [53] Sheng Guo, Weilin Huang, Haozhi Zhang, Chenfan Zhuang, Dengke Dong, Matthew R. Scott, and Dinglong Huang. Curriculumnet: Weakly supervised learning from large-scale web images. *ArXiv*, abs/1808.01097, 2018.
- [54] Yandong Guo, Lei Zhang, Yuxiao Hu, X. He, and Jianfeng Gao. Ms-celeb-1m: A dataset and benchmark for large-scale face recognition. In *ECCV*, 2016.
- [55] Jiangfan Han, Ping Luo, and Xiaogang Wang. Deep self-learning from noisy labels. *ArXiv*, abs/1908.02160, 2019.
- [56] Erik Härkönen, Aaron Hertzman, Jaakko Lehtinen, and Sylvain Paris. Ganspace: Discovering interpretable gan controls. In *IEEE Conference on Neural Information Processing Systems*, 2020.

- [57] F. Maxwell Harper and Joseph A. Konstan. The movielens datasets: History and context. *ACM Trans. Interact. Intell. Syst.*, 5(4), December 2015. ISSN 2160-6455. doi: 10.1145/2827872. URL <https://doi.org/10.1145/2827872>.
- [58] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer, 2016.
- [59] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. doi: 10.1109/CVPR.2016.90.
- [60] Tong He, Zhi Zhang, Hang Zhang, Zhongyue Zhang, Junyuan Xie, and Mu Li. Bag of tricks for image classification with convolutional neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 558–567, 2019.
- [61] Dan Hendrycks and Kevin Gimpel. A baseline for detecting misclassified and out-of-distribution examples in neural networks. *Proceedings of International Conference on Learning Representations*, 2017.
- [62] Dan Hendrycks, Mantas Mazeika, Duncan Wilson, and Kevin Gimpel. Using trusted data to train deep networks on labels corrupted by severe noise. In *NeurIPS*, 2018.
- [63] A. Hermans, Lucas Beyer, and B. Leibe. In defense of the triplet loss for person re-identification. *ArXiv*, abs/1703.07737, 2017.
- [64] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems*, 30, 2017.
- [65] E. Hoffer and Nir Ailon. Deep metric learning using triplet network. In *SIMBAD*, 2015.
- [66] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.
- [67] Jeremy Howard. Fastai. <https://github.com/fastai/fastai>, 2019.
- [68] Jeremy Howard. Imagenette, 2019. URL <https://github.com/fastai/imagenette/>.
- [69] Bo-Yang Hsueh, Wei Li, and I-Chen Wu. Stochastic gradient descent with hyperbolic-tangent decay on classification. In *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 435–442. IEEE, 2019.
- [70] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018.
- [71] Gary B. Huang, Manu Ramesh, Tamara Berg, and Erik Learned-Miller. Labeled faces in the wild: A database for studying face recognition in unconstrained environments. Technical Report 07-49, University of Massachusetts, Amherst, October 2007.
- [72] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. *CVPR*, 2017.



- [73] Dietmar Jannach and Gediminas Adomavicius. Recommendations with a purpose. In *Proceedings of the 10th ACM conference on recommender systems*, pages 7–10, 2016.
- [74] Animesh Karnewar and Oliver Wang. Msg-gan: Multi-scale gradients for generative adversarial networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7799–7808, 2020.
- [75] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. In *International Conference on Learning Representations*, 2018.
- [76] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun 2019. doi: 10.1109/cvpr.2019.00453. URL <http://dx.doi.org/10.1109/CVPR.2019.00453>.
- [77] Tero Karras, Miika Aittala, Janne Hellsten, Samuli Laine, Jaakko Lehtinen, and Timo Aila. Training generative adversarial networks with limited data. In *Proc. NeurIPS*, 2020.
- [78] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of stylegan. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun 2020. doi: 10.1109/cvpr42600.2020.00813. URL <http://dx.doi.org/10.1109/cvpr42600.2020.00813>.
- [79] Ira Kemelmacher-Shlizerman, Steven M Seitz, Daniel Miller, and Evan Brossard. The megaface benchmark: 1 million faces for recognition at scale. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4873–4882, 2016.
- [80] Youngdong Kim, Junho Yim, Juseung Yun, and Junmo Kim. Nlnl: Negative learning for noisy labels. *ArXiv*, abs/1908.07387, 2019.
- [81] Diederik P. Kingma and M. Welling. Auto-encoding variational bayes. *CoRR*, abs/1312.6114, 2014.
- [82] B. Klare, B. Klein, Emma Taborsky, Austin Blanton, J. Cheney, K. Allen, P. Grother, Alan Mah, M. Burge, and Anil K. Jain. Pushing the frontiers of unconstrained face detection and recognition: Iarpa janus benchmark a. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1931–1939, 2015.
- [83] Alexander Kolesnikov and Christoph H. Lampert. Pixelcnn models with auxiliary variables for natural image modeling. In *ICML*, 2017.
- [84] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, August 2009. ISSN 0018-9162. doi: 10.1109/MC.2009.263. URL <https://doi.org/10.1109/MC.2009.263>.
- [85] Simon Kornblith, Jonathon Shlens, and Quoc V Le. Do better imagenet models transfer better? In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2661–2671, 2019.
- [86] Simon Kornblith, Honglak Lee, Ting Chen, and Mohammad Norouzi. What’s in a loss function for image classification? *arXiv preprint arXiv:2010.16402*, 2020.

- [87] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, CIFAR, 2009. URL <https://www.cs.toronto.edu/~kriz/cifar.html>.
- [88] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.
- [89] Tuomas Kynkäänniemi, Tero Karras, Samuli Laine, Jaakko Lehtinen, and Timo Aila. Improved precision and recall metric for assessing generative models. *CoRR*, abs/1904.06991, 2019.
- [90] Adrian Łańcucki, Jan Chorowski, Guillaume Sanchez, Ricard Marxer, Nanxin Chen, Hans JGA Dolfing, Sameer Khurana, Tanel Alumäe, and Antoine Laurent. Robust training of vector quantized bottleneck models. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7. IEEE, 2020.
- [91] Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *ATT Labs [Online]*. Available: <http://yann.lecun.com/exdb/mnist>, 2, 2010.
- [92] Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th annual international conference on machine learning*, pages 609–616, 2009.
- [93] Kuang-Huei Lee, Xiaodong He, Lei Zhang, and Linjun Yang. Cleannet: Transfer learning for scalable image classifier training with label noise. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5447–5456, 2017.
- [94] Michael S. Lew, Nicu Sebe, Chabane Djeraba, and Ramesh Jain. Content-based multimedia information retrieval: State of the art and challenges. *ACM Trans. Multimedia Comput. Commun. Appl.*, 2(1):1–19, February 2006. ISSN 1551-6857. doi: 10.1145/1126004.1126005. URL <https://doi.org/10.1145/1126004.1126005>.
- [95] Mengtian Li, Ersin Yumer, and Deva Ramanan. Budgeted training: Rethinking deep neural network training under resource constraints. In *International Conference on Learning Representations*, 2019.
- [96] Wen Li, Limin Wang, Wei Li, Eirikur Agustsson, and Luc Van Gool. Webvision database: Visual learning and understanding from web data. *ArXiv*, abs/1708.02862, 2017.
- [97] Xiaoqiang Li, Liangbo Chen, Lu Wang, Pin Wu, and Weiqin Tong. Scgan: Disentangled representation learning by adding similarity constraint on generative adversarial nets. *IEEE Access*, PP:1–1, 09 2018. doi: 10.1109/ACCESS.2018.2872695.
- [98] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017.
- [99] Kanglin Liu, Guoping Qiu, Wenming Tang, and Fei Zhou. Spectral regularization for combating mode collapse in gans. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, Oct 2019. doi: 10.1109/iccv.2019.00648. URL <http://dx.doi.org/10.1109/ICCV.2019.00648>.

- [100] Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the variance of the adaptive learning rate and beyond. In *Proceedings of the Eighth International Conference on Learning Representations (ICLR 2020)*, April 2020.
- [101] Weiyang Liu, Y. Wen, Zhiding Yu, Ming Li, B. Raj, and Le Song. Sphereface: Deep hypersphere embedding for face recognition. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6738–6746, 2017.
- [102] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. *arXiv preprint arXiv:2201.03545*, 2022.
- [103] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.
- [104] Mario Lucic, Karol Kurach, Marcin Michalski, S. Gelly, and O. Bousquet. Are gans created equal? a large-scale study. In *NeurIPS*, 2018.
- [105] Malte Ludewig, Noemi Mauro, Sara Latifi, and Dietmar Jannach. Performance comparison of neural and non-neural approaches to session-based recommendation. In *Proceedings of the 13th ACM conference on recommender systems*, pages 462–466, 2019.
- [106] Brianna Maze, J. Adams, J. A. Duncan, Nathan D. Kalka, T. Miller, Charles Otto, Anil K. Jain, W. T. Niggel, J. Anderson, J. Cheney, and P. Grother. Iarpa janus benchmark - c: Face dataset and protocol. *2018 International Conference on Biometrics (ICB)*, pages 158–165, 2018.
- [107] L. McInnes, J. Healy, and J. Melville. UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction. *ArXiv e-prints*, February 2018.
- [108] Lars M. Mescheder, Andreas Geiger, and S. Nowozin. Which training methods for gans do actually converge? In *ICML*, 2018.
- [109] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.
- [110] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.
- [111] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. In *International Conference on Learning Representations*, 2018. URL <https://openreview.net/forum?id=B1QRgziT->.
- [112] Guido Montúfar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. On the number of linear regions of deep neural networks. *arXiv preprint arXiv:1402.1869*, 2014.
- [113] Alexander Mordvintsev, Christopher Olah, and Mike Tyka. Inceptionism: Going deeper into neural networks, 2015. URL <https://ai.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html>.
- [114] Alexander Mordvintsev, Christopher Olah, and Mike Tyka. Deepdream, 2015. URL <https://www.tensorflow.org/tutorials/generative/deepdream>.
- [115] Samuel G. Müller and Frank Hutter. Trivialaugment: Tuning-free yet state-of-the-art data augmentation, 2021.

- [116] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Icml*, 2010.
- [117] Duc Tam Nguyen, Thi-Phuong-Nhung Ngo, Zhongyu Lou, Michael Klar, Laura Beggel, and Thomas Brox. Robust learning under label noise with iterative noise-filtering. *ArXiv*, abs/1906.00216, 2019.
- [118] Tien T Nguyen, Pik-Mai Hui, F Maxwell Harper, Loren Terveen, and Joseph A Konstan. Exploring the filter bubble: the effect of using recommender systems on content diversity. In *Proceedings of the 23rd international conference on World wide web*, pages 677–686, 2014.
- [119] Xia Ning and George Karypis. Slim: Sparse linear methods for top-n recommender systems. In *2011 IEEE 11th International Conference on Data Mining*, pages 497–506. IEEE, 2011.
- [120] Chris Olah, Alexander Mordvintsev, and Ludwig Schubert. Feature visualization. *Distill*, 2017. doi: 10.23915/distill.00007. <https://distill.pub/2017/feature-visualization>.
- [121] Eli Pariser. *The Filter Bubble: What the Internet Is Hiding from You*. Penguin Group , The, 2011. ISBN 1594203008.
- [122] Taesung Park, Ming-Yu Liu, Ting-Chun Wang, and Jun-Yan Zhu. Semantic image synthesis with spatially-adaptive normalization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019.
- [123] Taesung Park, Alexei A. Efros, Richard Zhang, and Jun-Yan Zhu. Contrastive learning for unpaired image-to-image translation. In *European Conference on Computer Vision*, 2020.
- [124] Boris T Polyak and Anatoli B Juditsky. Acceleration of stochastic approximation by averaging. *SIAM journal on control and optimization*, 30(4):838–855, 1992.
- [125] Yipeng Qin, Niloy Mitra, and Peter Wonka. How does lipschitz regularization influence gan training? *Lecture Notes in Computer Science*, page 310–326, 2020. ISSN 1611-3349. doi: 10.1007/978-3-030-58517-4\_19. URL [http://dx.doi.org/10.1007/978-3-030-58517-4\\_19](http://dx.doi.org/10.1007/978-3-030-58517-4_19).
- [126] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *CoRR*, abs/1511.06434, 2016.
- [127] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. *arXiv preprint arXiv:2103.00020*, 2021.
- [128] Juan Ramos. Using tf-idf to determine word relevance in document queries, 1999.
- [129] Rajeev Ranjan, Carlos D. Castillo, and R. Chellappa. L2-constrained softmax loss for discriminative face verification. *ArXiv*, abs/1703.09507, 2017.
- [130] Benjamin Recht, Rebecca Roelofs, Ludwig Schmidt, and Vaishaal Shankar. Do imagenet classifiers generalize to imagenet? In *International Conference on Machine Learning*, pages 5389–5400. PMLR, 2019.
- [131] Mengye Ren, Wenyan Zeng, Bin Yang, and Raquel Urtasun. Learning to reweight examples for robust deep learning. *ArXiv*, abs/1803.09050, 2018.

- [132] Steffen Rendle, Walid Krichene, Li Zhang, and John Anderson. Neural collaborative filtering vs. matrix factorization revisited. In *Fourteenth ACM Conference on Recommender Systems*, pages 240–248, 2020.
- [133] Daniel Roich, Ron Mokady, Amit H Bermano, and Daniel Cohen-Or. Pivotal tuning for latent-based editing of real images. *arXiv preprint arXiv:2106.05744*, 2021.
- [134] Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65:386–408, 1958.
- [135] Aurko Roy, Ashish Vaswani, Niki Parmar, and Arvind Neelakantan. Towards a better understanding of vector quantized autoencoders. *ArXiv*, 2018.
- [136] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y. URL <https://image-net.org/>.
- [137] Tim Salimans, Andrej Karpathy, Xi Chen, and Diederik P. Kingma. Pixelcnn++: A pixelcnn implementation with discretized logistic mixture likelihood and other modifications. In *ICLR*, 2017.
- [138] Guillaume SANCHEZ, Vincente GUIZ, Ricard MARXER, and Frederic BOUCHARA. Deep learning classification with noisy labels. In *2020 IEEE International Conference on Multimedia Expo Workshops (ICMEW)*, pages 1–6, 2020. doi: 10.1109/ICMEW46912.2020.9105992.
- [139] Yujun Shen, Jinjin Gu, Xiaoou Tang, and Bolei Zhou. Interpreting the latent space of gans for semantic face editing. In *CVPR*, 2020.
- [140] Yujun Shen, Ceyuan Yang, Xiaoou Tang, and Bolei Zhou. Interfacegan: Interpreting the disentangled face representation learned by gans. *IEEE transactions on pattern analysis and machine intelligence*, 2020.
- [141] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL <http://arxiv.org/abs/1409.1556>.
- [142] Leslie N Smith. Cyclical learning rates for training neural networks. In *2017 IEEE winter conference on applications of computer vision (WACV)*, pages 464–472. IEEE, 2017.
- [143] Kihyuk Sohn, Honglak Lee, and Xinchen Yan. Learning structured output representation using deep conditional generative models. *Advances in neural information processing systems*, 28: 3483–3491, 2015.
- [144] Casper Kaae Sønderby, Ben Poole, and Andriy Mnih. Continuous relaxation training of discrete latent variable image models. In *Bayesian Deep Learning workshop, NIPS*, volume 201, 2017.
- [145] Harald Steck. Embarrassingly shallow autoencoders for sparse data. In *The World Wide Web Conference*, pages 3251–3257, 2019.

- [146] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.
- [147] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.
- [148] Yaniv Taigman, Ming Yang, Marc’Aurelio Ranzato, and Lior Wolf. Deepface: Closing the gap to human-level performance in face verification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1701–1708, 2014.
- [149] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pages 6105–6114. PMLR, 2019.
- [150] Hoang Thanh-Tung, Truyen Tran, and Svetha Venkatesh. Improving generalization and stability of generative adversarial networks. In *International Conference on Learning Representations*, 2018.
- [151] Hoang Thanh-Tung, Truyen Tran, and Svetha Venkatesh. Improving generalization and stability of generative adversarial networks. In *International Conference on Learning Representations*, 2019. URL <https://openreview.net/forum?id=ByxPYjC5KQ>.
- [152] L. Theis, A. van den Oord, and M. Bethge. A note on the evaluation of generative models. In *International Conference on Learning Representations*, Apr 2016. URL <http://arxiv.org/abs/1511.01844>.
- [153] Ilya O. Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Daniel Keysers, Jakob Uszkoreit, Mario Lucic, and Alexey Dosovitskiy. Mlp-mixer: An all-mlp architecture for vision. *CoRR*, abs/2105.01601, 2021. URL <https://arxiv.org/abs/2105.01601>.
- [154] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Deep image prior. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 9446–9454, 2018.
- [155] Aaron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. Neural discrete representation learning. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 6309–6318, 2017.
- [156] Aaron Van Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. In *International Conference on Machine Learning*, pages 1747–1756. PMLR, 2016.
- [157] Daniel Ponsa Vassileios Balntas, Edgar Riba and Krystian Mikolajczyk. Learning local feature descriptors with triplets and shallow convolutional neural networks. In Edwin R. Hancock Richard C. Wilson and William A. P. Smith, editors, *Proceedings of the British Machine Vision Conference (BMVC)*, pages 119.1–119.11. BMVA Press, September 2016. ISBN 1-901725-59-6. doi: 10.5244/C.30.119. URL <https://dx.doi.org/10.5244/C.30.119>.
- [158] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

- [159] Sagar Vaze, Kai Han, Andrea Vedaldi, and Andrew Zisserman. Open-set recognition: A good closed-set classifier is all you need. *arXiv preprint arXiv:2110.06207*, 2021.
- [160] Cédric Villani. *Optimal transport: old and new*, volume 338. Springer, 2009.
- [161] Fei Wang, Mengqing Jiang, Chen Qian, Shuo Yang, Cheng Li, Honggang Zhang, Xiaogang Wang, and Xiaoou Tang. Residual attention network for image classification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3156–3164, 2017.
- [162] Fei Wang, L. Chen, Cheng Li, Shiyao Huang, Yanjie Chen, Chen Qian, and Chen Change Loy. The devil of face recognition is in the noise. In *ECCV*, 2018.
- [163] Feng Wang, Jian Cheng, Weiyang Liu, and Haijun Liu. Additive margin softmax for face verification. *IEEE Signal Processing Letters*, 25(7):926–930, 2018.
- [164] H. Wang, Yitong Wang, Z. Zhou, Xing Ji, Zhifeng Li, Dihong Gong, Jingchao Zhou, and Wenyu Liu. Cosface: Large margin cosine loss for deep face recognition. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5265–5274, 2018.
- [165] Mei Wang and Weihong Deng. Deep face recognition: A survey. *Neurocomputing*, 429:215–244, Mar 2021. ISSN 0925-2312. doi: 10.1016/j.neucom.2020.10.081. URL <http://dx.doi.org/10.1016/j.neucom.2020.10.081>.
- [166] Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Andrew Tao, Jan Kautz, and Bryan Catanzaro. High-resolution image synthesis and semantic manipulation with conditional gans. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.
- [167] Xiaobo Wang, Shuo Wang, Jun Wang, Hailin Shi, and Tao Mei. Co-mining: Deep face recognition with noisy labels. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9358–9367, 2019.
- [168] Yisen Wang, Weiyang Liu, Xingjun Ma, James Bailey, Hongyuan Zha, Le Song, and Shu-Tao Xia. Iterative learning with open-set noisy labels. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8688–8696, 2018.
- [169] Cameron Whitelam, Emma Taborsky, Austin Blanton, Brianna Maze, J. Adams, T. Miller, Nathan D. Kalka, Anil K. Jain, J. A. Duncan, K. Allen, J. Cheney, and P. Grother. Iarpa janus benchmark-b face dataset. *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 592–600, 2017.
- [170] Ross Wightman. Pytorch image models. <https://github.com/rwightman/pytorch-image-models>, 2019.
- [171] Tong Xiao, Tian Xia, Yi Yang, Chang Huang, and Xiaogang Wang. Learning from massive noisy labeled data for image classification. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2691–2699, 2015.
- [172] Qizhe Xie, Zihang Dai, Eduard H. Hovy, Minh-Thang Luong, and Quoc V. Le. Unsupervised data augmentation. *ArXiv*, abs/1904.12848, 2019.
- [173] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1492–1500, 2017.

- [174] Jason Yosinski, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson. Understanding neural networks through deep visualization. In *Deep Learning Workshop, International Conference on Machine Learning (ICML)*, 2015.
- [175] Yang You, Jing Li, Sashank Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. Large batch optimization for deep learning: Training bert in 76 minutes. In *International Conference on Learning Representations*, 2020. URL <https://openreview.net/forum?id=Syx4wnEtvH>.
- [176] Scott WH Young. Improving library user experience with a/b testing: Principles and process. *Weave: Journal of Library User Experience*, 1(1), 2014.
- [177] Weihao Yu, Mi Luo, Pan Zhou, Chenyang Si, Yichen Zhou, Xinchao Wang, Jiashi Feng, and Shuicheng Yan. Metaformer is actually what you need for vision. *arXiv preprint arXiv:2111.11418*, 2021.
- [178] Sangdoon Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Youngjoon Yoo. Cutmix: Regularization strategy to train strong classifiers with localizable features. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6023–6032, 2019.
- [179] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *British Machine Vision Conference 2016*. British Machine Vision Association, 2016.
- [180] Hongyi Zhang, Moustapha Cisse, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *International Conference on Learning Representations*, 2018. URL <https://openreview.net/forum?id=r1Ddp1-Rb>.
- [181] Michael R. Zhang, James Lucas, Geoffrey Hinton, and Jimmy Ba. *Lookahead Optimizer:  $\tilde{\eta}K/\tilde{\eta}$  Steps Forward, 1 Step Back*. Curran Associates Inc., Red Hook, NY, USA, 2019.
- [182] X. Zhang, R. Zhao, Y. Qiao, Xiaogang Wang, and Hongsheng Li. Adacos: Adaptively scaling cosine logits for effectively learning deep face representations. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10815–10824, 2019.
- [183] Zhilu Zhang and Mert R. Sabuncu. Generalized cross entropy loss for training deep neural networks with noisy labels. In *NeurIPS*, 2018.
- [184] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, 2017.
- [185] Juntang Zhuang, Tommy Tang, Yifan Ding, Sekhar C Tatikonda, Nicha Dvornek, Xenophon Papademetris, and James Duncan. Adabelief optimizer: Adapting stepsizes by the belief in observed gradients. *Advances in Neural Information Processing Systems*, 33, 2020.



## A Notations, conventions and acronyms

Unless otherwise specified in the text, these are the default notations and conventions used throughout the manuscript.

|                               |  |
|-------------------------------|--|
| $x$                           | $x$ is either a scalar value or of an unspecified or irrelevant type, depending on the context     |
| $\mathbf{x}$                  | the vector $\mathbf{x}$  |
| $X$                           | the matrix $X$ or set $X$  |
| $x \sim p(x)$                 | $x$ is sampled according to the probability distribution $p(x)$                                    |
| $\mu, \sigma$                 | Mean and standard-deviation  |
| $\mathcal{N}(\mu, \sigma)$    | Normal distribution of mean $\mu$ and standard-deviation $\sigma$                                  |
| $\mathcal{N}(x; \mu, \sigma)$ | Likelihood of $x$ according to a normal distribution of mean $\mu$ and standard-deviation $\sigma$ |
| $\text{Cat}(x; \theta)$       | Probability of event $x$ for categorical distribution with parameters $\theta$                     |
| $ A $                         | cardinal of set $A$  |
| $H(p)$                        | Entropy of distribution $p$  |
| $H(p, q)$                     | Cross-Entropy of distributions $p$ and $q$   |

Table A.1: Notations and conventions

**Adam** Adaptive Moment Estimation  
**AUC** Area Under Curve  
**BCE** Binary Cross-Entropy  
**CLIP** Contrastive Language-Image Pretraining  
**CNN** Convolutional Neural Network  
**convnet** Convolutional Neural Network  
**CVAE** Conditional VAE  
**D** Discriminator  
**EASE<sup>R</sup>** Embarrassingly Shallow AutoEncoder (in Reverse order)  
**ELBO** Evidence Lower Bound  
**FID** Fréchet Inception Distance  
**G** Generator  
**GAN** Generative Adversarial Network  
**IS** Inception Score  
**JS** Jensen-Shannon  
**KID** Kernel Inception Distance  
**lr** learning rate  
**LSGAN** Least-Square GAN  
**MLP** Multi-Layer Perceptron  
**MSE** Mean-Square Error  
**NLNN** Noisy Label Neural Networks  
**NSGAN** Non-Saturating GAN  
**OoD** Out of Distribution  
**PPL** Perplexity  
**SGD** Stochastic Gradient Descent  
**SGDM** Stochastic Gradient Descent with Momentum  
**SNGAN** Spectral Normalization GAN  
**SotA** State Of The Art  
**SVD** Singular Value Decomposition

**TPR** True Positive Rate

**TF-IDF** Term Frequency–Inverse Document Frequency

**UMAP** Uniform Manifold Approximation and Projection

**VAE** Variational AutoEncoder

**VOD** Video On Demand

**VQ** Vector-Quantized

**VQ-GAN** Vector-Quantized GAN

**VQ-VAE** Vector-Quantized VAE

**WGAN** Wasserstein GAN

**WGAN-GP** Wasserstein GAN with Gradient Penalty

## B Face recognition models additional plots

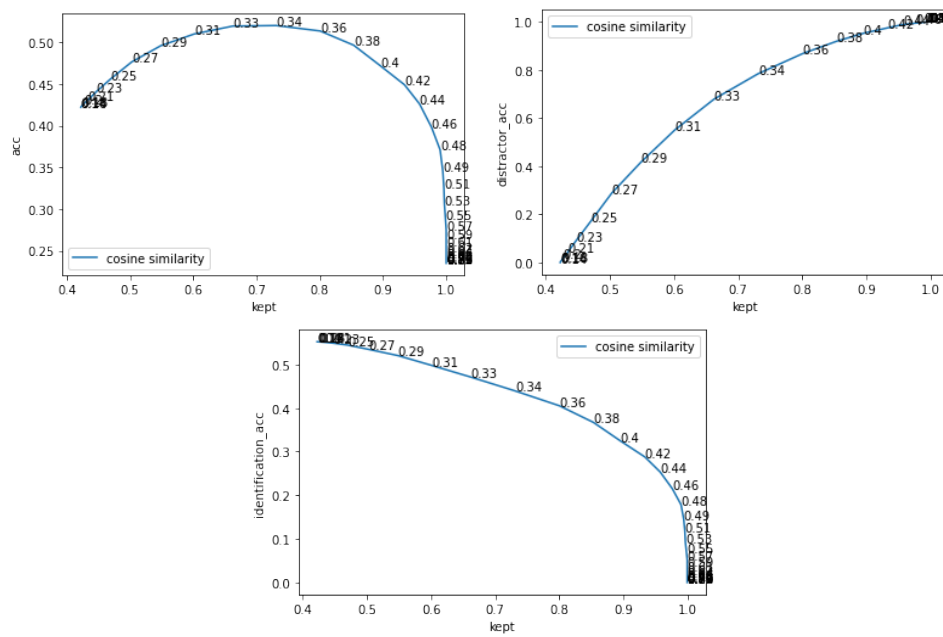


Figure B.1: Additional plots for the ArcFace model (Section 5.5.5)

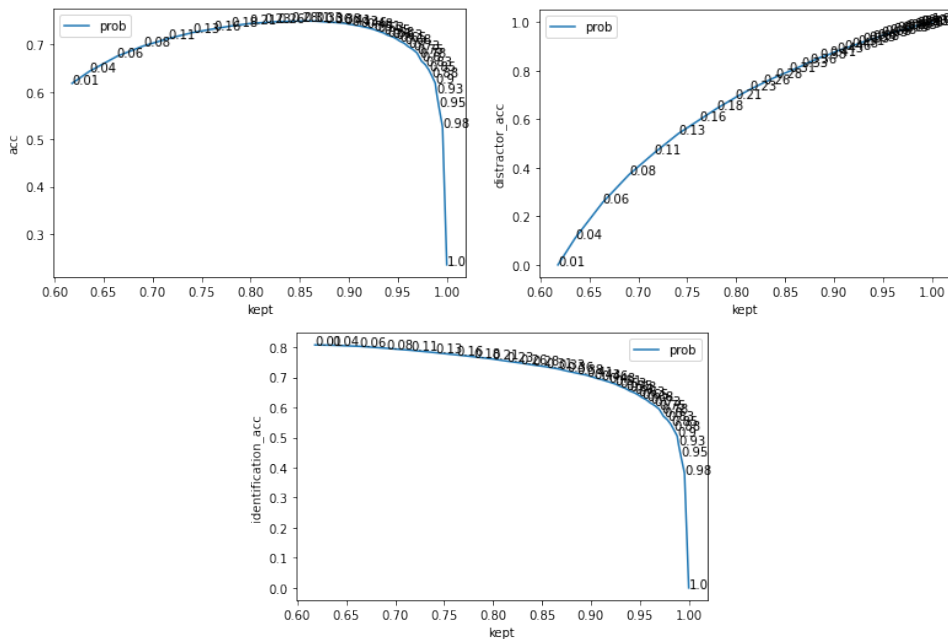


Figure B.2: Additional plots for the CE model (Section 5.5.5)

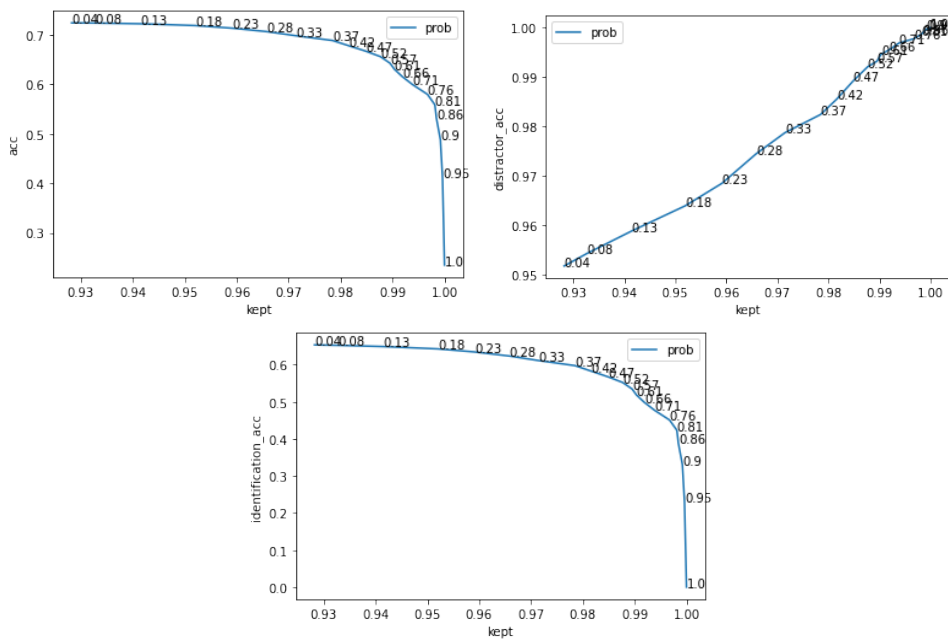


Figure B.3: Additional plots for the DCE model (Section ??)

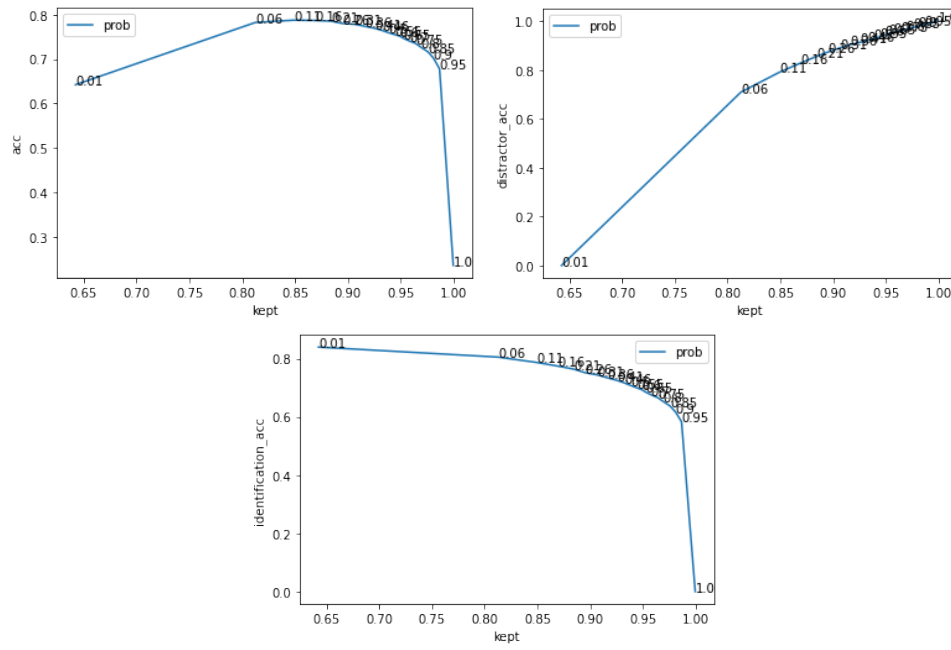


Figure B.4: Additional plots for the ZLog model (Section 5.5.5)

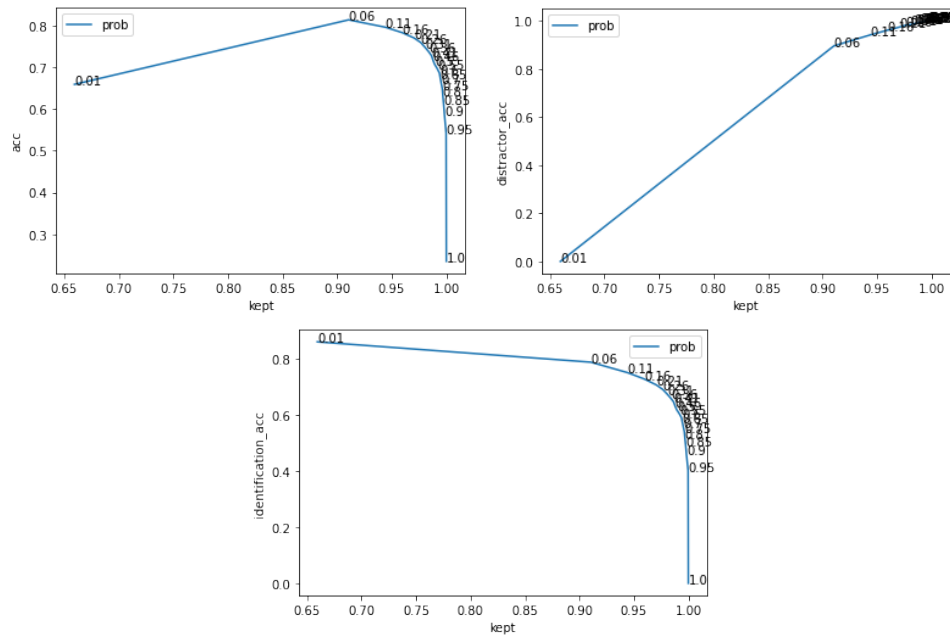
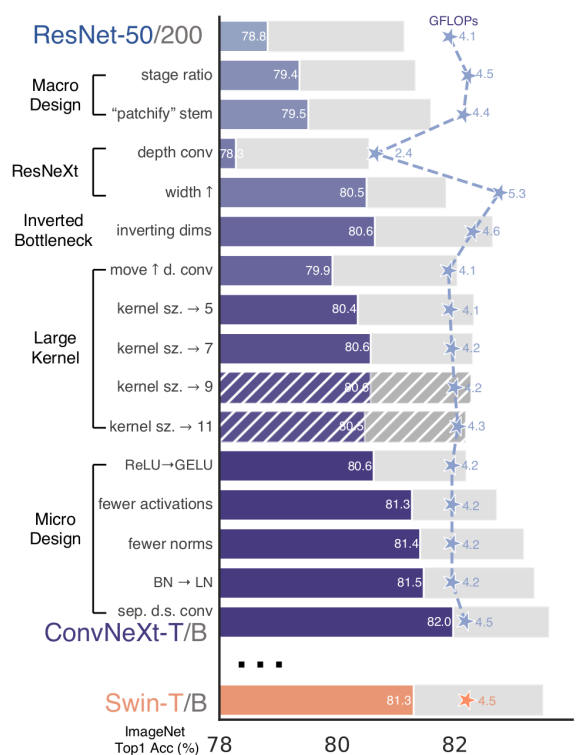


Figure B.5: Additional plots for the ME model (Section 5.5.5)

## C ConvNeXt experiments



We modernize a standard ConvNet (ResNet) towards the design of a hierarchical vision Transformer (Swin), without introducing any attention-based modules. The foreground bars are model accuracies in the ResNet-50/Swin-T FLOP regime; results for the ResNet-200/Swin-B regime are shown with the gray bars. A hatched bar means the modification is not adopted. Detailed results for both regimes are in the appendix. Many Transformer architectural choices can be incorporated in a ConvNet, and they lead to increasingly better performance. In the end, our pure ConvNet model, named ConvNeXt, can outperform the Swin Transformer.

Figure C.1: Incremental improvement process from ResNet to ConvNext. Figure extracted from Liu et al. [102].