SAPIENZA
UNIVERSITÀ DI ROMA

Generating handwritten text images with GANs for data augmentation

Faculty of Information Engineering, Informatics, and Statistics

Corso di Laurea Magistrale in Data Science

Candidate

Gianmarco Ursini

ID number 1635956

Thesis Advisor

Prof. Simone Scardapane

Co-Advisor

Dr. Pierfrancesco Veltri

Academic Year 2021/2022

Generating handwritten text images with GANs for data augmentation
Master's thesis. Sapienza – University of Rome

© 2022 Gianmarco Ursini. All rights reserved

This thesis has been typeset by LaTeX and the Sapthesis class.

Version: October 16, 2022 Thesis defended on: October 28, 2022

Author's email: ursini.1635956@studenti.uniroma1.it

Abstract

State of the art architectures used to perform Handwritten Text Recognition have achieved impressive performances onto public benchmarks. Due to the high variability between handwriting styles, such models have to be trained onto huge volumes of manually labeled training data, which is by itself a highly time-consuming problem. Also, due to the conceptual gaps (e.g. language and alphabet variation, content's semantic differences) existing between the above mentioned public benchmarks and the real-world data that those architectures are deployed to recognize, tremendous drops of performances are often noticed at inference time. In order to mitigate those problems, data augmentation is occasionally executed via the generation of new data points. Nevertheless, positive influences of such an augmentation technique over training pipelines are greatly affected by the goodness of the implemented generative model and by the domain of the real world data. Accordingly to the ENEL proposed internship, the present thesis is devoted to the implementation of two different GANs able to generate handwritten text images given an user inputted text content and handwriting style. A benchmark database of handwritten images will therefore be augmented with synthetic images, so to assess if the ENEL corporate OCR (Optical Character Recognition) model training can benefit by this GAN based data augmentation. Results show that under the conditions of the conducted experiments, no meaningful improvement of the OCR model performances is achieved, suggesting that better GAN performances or other strategies are required to attain OCR model enhancements.

Contents

Chapter 1

Introduction

Handwritten Text Recognition [1] aims to the digitalization of handwritten text. HTR can be performed in *online* or *offline* fashion. In Online-HTR, characters are individually isolated at run time (this is possible thanks to the usage of digitalizing hardware e.g. PDAs or tablets), allowing also the gathering of information about time at which each character has been written. On the other hand, in Offline-HTR, the data is captured as a static image once the writing operation has been completed, so removing the information about the identification of the single entities (i.e. characters) to be predicted. Challenges regarding HTR arise by the inner nature and complexity of hand written text: indeed, analyzing the features that can be reasonably used to describe writing styles, a huge variability in terms of slant, width of the pen line, style of the characters (among the others) is often encountered. For this reason, in order to obtain machine learning models able to capture and identify the identity of characters and words regardless of the style they have been written with, huge amounts of labeled training data are required. Due to the fact that the label recording of this kind of data is a time consuming operation, the framework of ML generative models for the automatic production of handwritten data is often leveraged, aiming to the improvement of the training pipelines of those HTR systems.

1.1 Scope and discussion

Aim of this thesis work is to assess whether ENEL corporate OCR training can benefit by a GAN based data augmentation mechanism. The implementation of generative models able to synthesize realistic handwritten text images is carried out. Those generated images will then be used to enlarge an handwritten benchmark database, aiming to the improvement of the performances of the corporate OCR model used to scan and digitalize ENEL's customers hand-filled forms et similia. The main goal is achieved after the evaluation of the generated data's influences over the corporate HTR system training pipeline.

1.2 Structure of thesis

A brief introduction to all the technologies, architectures and general knowledge needed for the task is performed in Chapter 2. Chapter 3 contains the chronography of the generative model's framework, reporting the papers that mostly affected and influenced the growth of the field. Chapter 4 will technically introduce all the datasets used to train SOTA models and generate the final handwritten dataset. An accurate

description of the corporate OCR architecture is provided in Chapter 5. Chapter 6 will focus on the detailed characterization and description of the generative model's architectures. All the experiments executed and the empirical results obtained will be reported in Chapter 7, while the final Chapter 8 will try to identify meaningful and interesting extensions of the present thesis work.

Chapter 2

Technologies and framework

2.1 M.L. algorithms and supervised learning

Machine learning algorithms can be broadly characterized as **unsupervised** or **supervised** depending on the kind of the experience they have during the learning process. Indeed, most of the algorithms can be understood as being able to experience a **data set**, i.e. a collection of data points. In the supervised learning framework (the only one that will be addressed during this thesis work), algorithms experience data sets where each i_{th} data point is a tuple containing the features x_i (e.g. a textual sentence) and the respective label y_i (e.g. the sentiment associated to that particular sentence). In such a sample task, analyzing the data set the supervised learning algorithm is able to learn how to classify sentences to sentiment's classes. In general, aim of machine learning is also to obtain general representations of the dynamics underlying the associative process of a feature point to its label, allowing the supervised learning algorithm to correctly associate labels to unseen feature points. The associative process is reconstructed by machine learning architectures (often made up of Artificial Neural Networks, presented in detail in section 2.2), that simply represent highly parameterized models $f_\theta()$ ideally able to correctly map features to labels (i.e. $f_\theta(x_i) = y_i$).

Gradient descent: the core algorithm to learn

The right set of the θ parameters is often obtained via the resolution of optimization tasks. Optimization refers to the task of maximizing or minimizing a function $L(s)$ altering s (in the first case $L(s)$ is usually referred to as a gain function, in the latter $L(s)$ it's usually defined cost or loss function). Up to now it is sufficient to now that $L()$ will act as a measure of the knowledge over the data set's underlying feature-label associative process achieved by the model itself. Detailed probabilistic interpretations of the most common loss functions will be provided in section 2.3. Suppose $L() : \mathbb{R}^n \to \mathbb{R}$. $\nabla_x L(\mathbf{x})$ denotes the gradient of $L()$ evaluated in $\mathbf{x}$, i.e. the vector $\in \mathbb{R}^n$ that can be proven to be [2] the direction of steepest ascent of $L()|_{\mathbf{x}}$. For sufficient small ϵ then:

$$L(\mathbf{x} + \epsilon\nabla_{\mathbf{x}}) \geq L(\mathbf{x}) \qquad \text{or equivalently} \qquad L(\mathbf{x} - \epsilon\nabla_{\mathbf{x}}) \leq L(\mathbf{x}) \tag{2.1}$$

Iterating the execution of those infinitesimal steps towards the direction of steepest descent is known as the **gradient descent** algorithm, able to minimize (if the value of ϵ is properly adjusted) any arbitrary strictly convex function Figure 2.1. It is worth to highlight that the above described procedure suffers by the presence of

critical points (i.e. points where $\nabla_{\mathbf{x}} L(\mathbf{x}) = \mathbf{0}$, where $\mathbf{0}$ denotes the null vector $\in \mathbb{R}^n$). As obvious, the module of the step undertaken would be zero, typically causing the halting of the gradient descent algorithm in those points. The introduction of *momentum* techniques (informally, the size and direction of the step undertaken from $\mathbf{x}$ will not depend simply by $\nabla_{\mathbf{x}} L(\mathbf{x})$, but also from the gradient computed in the previous position) typically dampen those issues. Given a data set containing m data points, and recalling that the ML architecture is just an highly parameterized function, in the ML framework the loss function is typically a function of the model's parameters θ and of the data set features and labels $\{x_i, y_i\}_{i=1}^{i=m}$. Since as said machine learning aim to generalize models so to make them suitable to predict never seen data, the function that actually needs to be minimized is said to be the **risk function** $J(\theta)$ (i.e. the expected value of the loss function computed over the empirical probability distribution function of the data), solely function of the model's parameters θ:

$$J(\theta) = \frac{1}{m} \sum_i L(x_i, y_i, \theta) \tag{2.2}$$

The above descrived task is known as **empirical risk minimization**, and the optimal parameters $\theta^* = \underset{\theta}{\text{argmin}}\ J(\theta)$ can so be found iterating the gradient descent algorithm:

$$\theta^{new} = \theta - \epsilon \nabla_\theta J(\theta) \qquad \text{with} \qquad \nabla_\theta J(\theta) = \frac{1}{m} \sum_i \nabla_\theta L(x_i, y_i, \theta) \tag{2.3}$$

ϵ is often called *learning rate* and it modulates the size of the parameter's updating step, the actual module of which is equal to $\epsilon \|\nabla_\theta J(\theta)\|_2$. If on the one hand estimating $J(\theta)$ as the expected value over the whole data set allows for the most accurate estimate of the risk function, on the other hand each single learning step would require a full scan of the data set, and the gradient descent algorithm time complexity would linearly depend by the number of data points. Furthermore, data points typically contain natural noise that has to be disregarded and kept unmodeled if good generalization properties over unseen data points want to be achieved [2]. For those reasons, this vanilla gradient descent algorithm often leaves room to the **stochastic gradient descent** algorithm. The whole data set is divided into subsets (a.k.a. mini batches) containing a defined number n of data points. In the literature n is often referred to as the batch size. During each iteration, $\nabla_\theta J(\theta)$ is estimated solely over the data points belonging to a single mini batch. A different mini batch is picked at each iteration, and once $K = \frac{m}{n}$ iterations are executed, a training epoch is said to be completed. Injecting momentum to the stochastic gradients eventually reduce the noise implied by the choice of the batch size. Stochastic gradient descent significantly improves the efficiency of the learning procedure: said d the cardinality of the θ parameters set, SGD reduces the $\mathcal{O}(m \cdot d)$ time complexity of the GD algorithm to an $\mathcal{O}(d)$ time complexity [3]. SGD algorithm it is also proven to reach parameter's configurations that allow the parameterized models to retain better generalization capabilities [4].

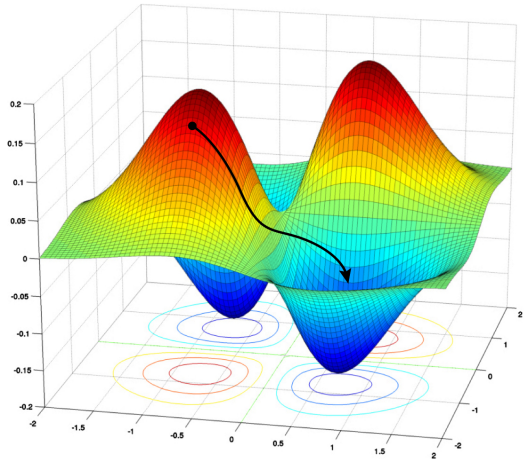

Figure 2.1. Gradient descent over a non-convex function $L() : \mathbb{R}^2 \to \mathbb{R}$ [5]

2.2 Artificial Neural Networks

The expression *Artificial Neural Network* is inspired by the networks of biological neurons that can be observed in the cerebral cortex. Neurons constitute the fundamental bricks of the animal nervous system: complexly connected by synapses, they are capable to the deploy and transmit electrical signals that, on the overall, allow the general performance of the cerebral activity. In an analogous fashion, Artificial Neural Networks (commonly shortened to ANNs) are built with fundamental units and, depending on the task that has to be faced, different kind of ANN's architectures can be implemented. Despite the fact that the number of neurons used to build modern ML models lately overcame the average brain size of 100 billion neurons [6] (e.g. the Megatron-Turing NLG 530B, made up of 530 billions parameters [7]), we find reasonable to adopt this analogy just because the architectures of ANNs try to structurally resemble the human brain. We care to highlight that neither a capability nor a complexity comparison should be carried out since ANNs are still far from reaching the performances achieved by the animal brain in all the tasks that typically require power of expression, generalization ability, creativity and *true* understanding.

2.2.1 Neurons and Multi Layer Perceptrons

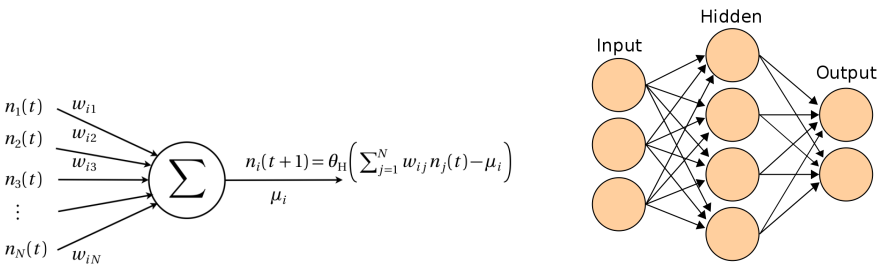

Figure 2.2. McCulloch-Pitts neuron and a shallow neural network. [8] [9]

The McCulloch-Pitts neuron is the fondamental unit used to build ANNs. In its simplified version, each neuron can be either switched on or off if the argument of its *activation function* (i.e. the function determining the dynamic of the neuron, an Heaveside Θ in Figure 2.2). The chronology of the information flow through the network (built with parallel layers each containing a defined number of neurons) can be counted via temporal unitary steps measured by the variable t. In the case of a fully connected network (i.e. a net where each neuron is connected to all the neurons of the previous and following layer), the J neuron's outputs $n_j(t)$ at time t will be act as the input of the single i_{th} neuron at time step $t+1$: it output can be computed as:

$$n_i(t+1) = \theta_H(\sum_{j=1}^{j=N} w_{ij} n_j(t) - \mu_i) \tag{2.4}$$

Where the w_{ij} *weights* the outputs of the neurons coming from the previous layer. Stacking multiple parallel layers of neurons it is so possible to build Multi Layer Perceptrons (a.k.a. feedforward networks). Given an arbitrary non-linear activation function σ_l, the output of the l_{th} MLP's layer can be represented in matrix notation as:

$$\boldsymbol{x}_{l+1} = \sigma_l\left(\boldsymbol{W}_l \boldsymbol{x}_l\right) \tag{2.5}$$

Where weights of the layer l are encoded in the matrix $\boldsymbol{W}_l$, and $\mathbf{x}_l$ recall the input of the l_{th} layer (i.e. the output of the $(l-1)_{th}$ layer). Looking at this matrix representation it is worth to notice that once multiple parallels layers are used, the non-linearities adopted prevent the collapse of the whole the model into a single naive linear model.

2.2.2 The universal approximation theorem

The capability of those architectures to approximate any arbitrary functional form $f(x)$ is guaranteed by a fundamental theorem in ML, i.e. the **universal approximation theorem** [10]. It states that, given a σ function which is continuous and sigmoidal ($\sigma_{x\to\infty} \to 1; \sigma_{x\to-\infty} \to 0$), said I_n the n dimensional unitary hypercube $[0,1]^n$ and $C(I_n)$ the set of continuous functions over I_n, the finite sums in the shape:

$$F(x) = \sum_{j=1}^{j=N} \alpha_j \sigma(y_j^T x + \theta_j) \qquad \text{with } \alpha_j, \theta_j \in \mathbb{R},\ y_j^T \in \mathbb{R}^n \tag{2.6}$$

Are dense in $C(I_n)$, i.e. $\forall\ f(x) \in C(I_n)$ and given an $\epsilon > 0$:

$$\exists F(x):\ |F(x) - f(x)| < \epsilon \quad \forall x \in I_n \tag{2.7}$$

It is worth to notice that even if the theorem prove the existence of a net able to approximate the generic $f(x)$, it doesn't claim anything about the complexity of its implementation, i.e. how big should N (the number of neurons used) be.

2.2.3 The back-propagation algorithm

The training of a neural network's procedure deeply relies on the gradient descent algorithm introduced in section 2.1. Indeed, due to the fact that non-linearities are introduced among the layers of the multi perceptrons, most of the commonly used cost functions result to be non-convex w.r.t. the model's parameters θ. Closed form solutions to find the optimal parameter's set become so inapplicable: the stochastic

gradient descent algorithm, on the other hand, iteratively adjusts the weights to reduce the risk function, even if no guarantees that the optimal parameter's configuration will be reached can theoretically be given. Being ANNs arbitrarily deep and complex, an analytic derivation and/or computation of the gradients often results into a very challenging and error prone task. The **back propagation** algorithm, an efficient differentiation technique to compute gradients, is below introduced.
As an input $\boldsymbol{x}$ is fed to the model, its output $f_\theta(\boldsymbol{x}) = \hat{\boldsymbol{y}}$ is computed via the application of elementary operations (e.g. matrix multiplications, activation function's applications, additions). The knowledge of the order in which those elementary operations are executed inside the neural network allows the construction of the **computational graph** Figure 2.3. The flow of the input information from the input node towards the output node is known as **forward propagation**. Let's also recall the chain rule of calculus, used to compute derivatives of function obtained via the composition of other functions with known derivatives. Suppose $y = g(x)$ and $z = f(g(x)) = f(y)$, the chain rule states that [2]:

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx} \tag{2.8}$$

If $\boldsymbol{x} \in \mathbb{R}^m$, $\boldsymbol{y} \in \mathbb{R}^n$, $z \in \mathbb{R}$, $\boldsymbol{y} = g(\boldsymbol{x})$ and $z = f(\boldsymbol{y})$, then Equation 2.8 translates to:

$$\nabla_{\mathbf{x}} z = \left(\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}\right)^{\intercal} \nabla_{\boldsymbol{y}} z \qquad \text{with} \qquad \frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}} \in \mathbb{R}^{n \times m} \tag{2.9}$$

Back propagating through the computational graph and leveraging dictionaries containing the derivative expression of several standard operations, gradients of the empirical risk function w.r.t. the model's parameters can easily and efficiently be obtained at each layer of the network.

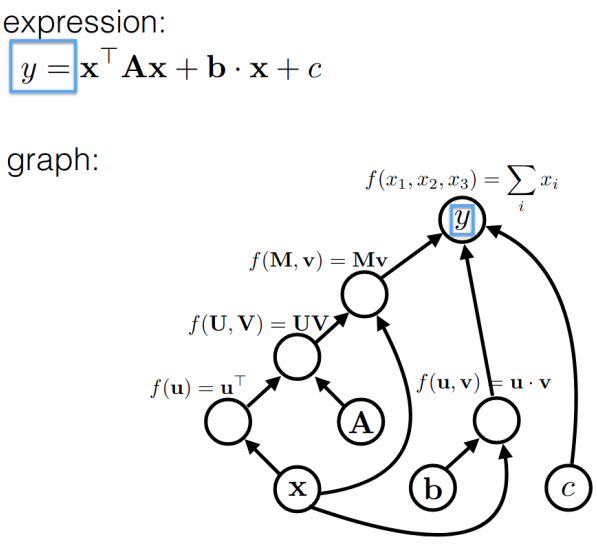

Figure 2.3. An example of computational graph built to compute the output of a defined sample function [11]

2.2.4 Activation functions

As said, the introduction of non-linearities in MLPs (and, in general, in all ANNs) via the application of activation functions allows to approximate arbitrarily complex functional forms and avoids the collapse of those models into mere linear models. The choice of those activation functions can significantly impact the behaviour and the performance of the built model. Each activation function presents pros and cons, most of them often related to the gradients computation and the dynamic of the gradient's flow through the network. One of the most commonly adopted activation function is the **sigmoid**, often indicated with the symbol $\sigma()$. Given an input x, it squashes its values in the range $(0, 1)$ and is defined as:

$$\sigma(x) = \frac{1}{1 + \exp(-x)} \tag{2.10}$$

As clear from Figure 2.4, once the input reaches the saturation areas, $\frac{\partial \sigma(x)}{\partial x}$ get close to zero. Furthermore, Figure 2.4 shows that $\frac{\partial \sigma(x)}{\partial x}$ is bounded in the interval $[0, 0.25]$. Being able to build models that can be arbitrarily deep, and since the chain rule is leveraged to compute the gradients of the risk w.r.t. the model's parameters, the chain multiplication of many values close to 0 typically cause the *vanishing gradient* problem. The gradients w.r.t. the model's shallowest weights (i.e. the ones parameterizing the closest layers to the input one) turn out to be not sufficiently robust to properly update those parameters, causing the weight's updating slowdown. ReLu (which stands for Rectified linear unit) is able to solve those issues: given an input x, it's output is computed as:

$$\text{ReLU}(x) = \max(0, x) \tag{2.11}$$

Neurons never got saturated and gradients hardly vanish when computed with back propagation. On the other hand, its outputs (like the sigmoid's ones) are not zero centered. Let see what happens when this always positive output is fed to a neuron belonging to a deeper layer. If the neuron is equipped with an activation function $f()$, its output will be equal to $y = f(\sum_i w_i \cdot \sigma(x_i)) = f(j)$. The gradient of the cost function $L()$ w.r.t. its parameters $\boldsymbol{w}$ can so be computed as:

$$\frac{\partial L}{\partial \boldsymbol{w}} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial j}\frac{\partial j}{\partial \boldsymbol{w}} \qquad \text{with} \qquad \frac{\partial j}{\partial \boldsymbol{w}} = \text{ReLU}(\boldsymbol{x}) \geq 0 \tag{2.12}$$

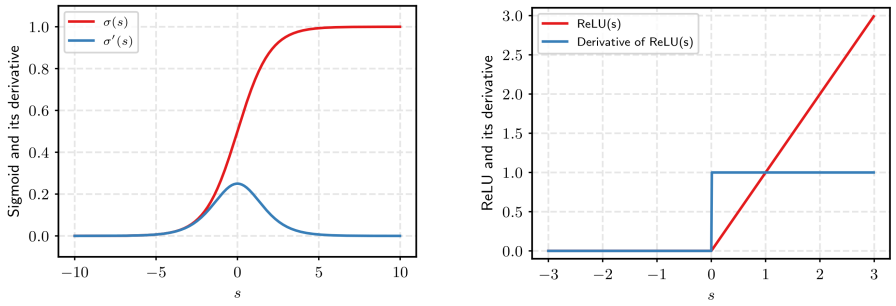

Figure 2.4. Visualization of Sigmoid and ReLu activation functions with their first derivatives.[12]

Being $\frac{\partial y}{\partial j} = \text{ReLU}'(j) \in \{0, 1\}$, all the gradient's values of $\frac{\partial L}{\partial \boldsymbol{w}}$ will either be all positive or negative depending on the sign of $\frac{\partial L}{\partial y}$, causing the weight's update to proceed in a zigzag fashion [13] if the optimal $\boldsymbol{w}$ configuration is sitting, if $\boldsymbol{w} \in \mathbb{R}^2$, in the 2^{nd} or 3^{rd} quadrant Figure 2.5. Suitable properties arising by the zero-centering of the non-linearity's outputs can be obtained via the tanh() activation function, outputs of which can be computed as:

$$\tanh(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)} \tag{2.13}$$

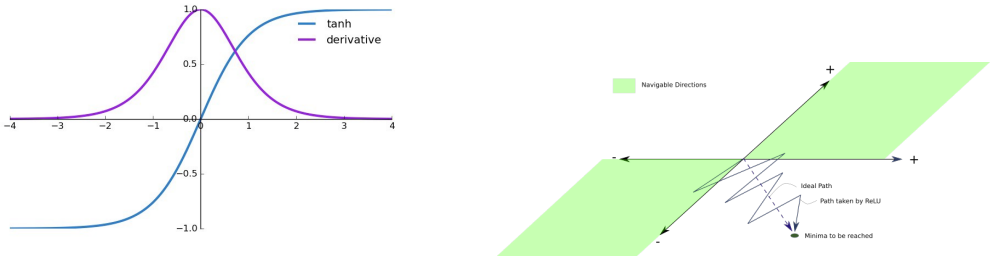

Figure 2.5. Visualization of the tanh() activation function with its first derivative and the zigzag gradient descent path due to always positive inputs.[14] [15]

Arbitrarily more complex and dynamic activation functions can be adopted, the introduction of which is beyond the scope of this thesis work.

2.2.5 Exploiting data's structure priors with Convolutional Neural Networks

During the introduction of multi layer perceptrons no prior assumption has been made on the structure of the inputted data. On the other hand, for all the cases where a grid-like topology can be identified in the data (e.g. temporal series which can be thought as 1D grids, or images, represented as 2D grids), specialized ANNs can be used: Convolutional Neural networks [2]. The term *convolutional* recalls the **convolution** mathematical operation that acts (in its most general form) on two functions $x()$ and $w()$ having real-valued arguments. The convolution $s(t)$ between $x()$ and $w()$ evaluated in t is defined as:

$$s(t) = (x * w)(t) = \int x(a)w(t-a)da \tag{2.14}$$

Being our data euclidean domains often (if not always) discrete, Equation 2.14 can be re-defined as:

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a) \tag{2.15}$$

or, if the convolution is executed on multiple axis at the same time, as:

$$s(t,s) = (x * w)(t,s) = \sum_{a}\sum_{b} x(a,b)w(t-a, s-b) \tag{2.16}$$

In the convolutional network terminology , x is often referred as the *input*, while w as the *kernel*. The convolution results to be commutative thanks to the flipping of the kernel w.r.t. the input (i.e. in the sum, the index of the input increases while the one of the kernel decreases). Since the commutative property can be neglected into neural networks where the identification of input and kernels is always maintained, the analogous **cross-correlation** (identical to the convolution but without kernel flipping) can be computed as:

$$s(t,s) = (x * w)(t,s) = \sum_a \sum_b x(a,b)w(t+a, s+b) \tag{2.17}$$

The spatial size of the step used during the convolution operation (i.e. the number of pixels which the kernel is shifted) is the stride, Managing its value, the size of the kernels and the number of padding pixels inserted in the input it is possible to fix the spatial dimensions of the activation maps (a.k.a. feature maps, i.e. the output of the convolution operation of a single kernel on the input).

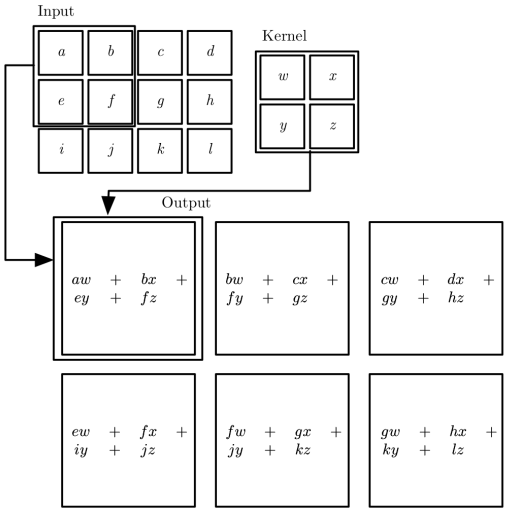

Figure 2.6. An example of 2D convolution without kernel flipping [2]

Convolutional layers can so be built: they foresee the convolution of a set of trainable kernels on top of the input. Sharing the same weights of each kernel while convolving across all the input's positions results into the construction of translational equivariant layers. Indeed, if identical patterns are found in the input, the corresponding activation map's partitions for will be identical among them self. In words, translational equivariance make us capable to recognize patterns in the input *regardless* of their location. After the application of the convolutional layer, a non linear activation function is applied element-wise to the feature maps. Subsequently, the features flow through a pooling layer: pooling kernels are scanned on the feature maps without superposition, and their output provide a summary statistic of their receptive field (i.e. the portion of input pixels contributing to the computation of the kernel's output) Figure 2.7. Among the most common pooling functions we find **max pooling** and **average pooling**: applying a max pooling kernel, the maximum value found in its receptive field is outputted to its feature map, while applying an average pooling kernel the average of the values found in

its receptive field is outputted to its feature maps. Pooling layers helps to obtain representation approximately invariant to small translations of the input and, as we go deeper in the convolutional neural network architecture, increase the receptive field of the kernels, allowing for the extraction of high-level (i.e. more complex) features Figure 2.7. The sum of a convolutional layer, an activation layer and a pooling layer make what is know as a convolutional block. After the application of an adequate number of convolutional blocks, a linear classifier head is typically applied on the deepest feature's representations in order to compute the final network's output.

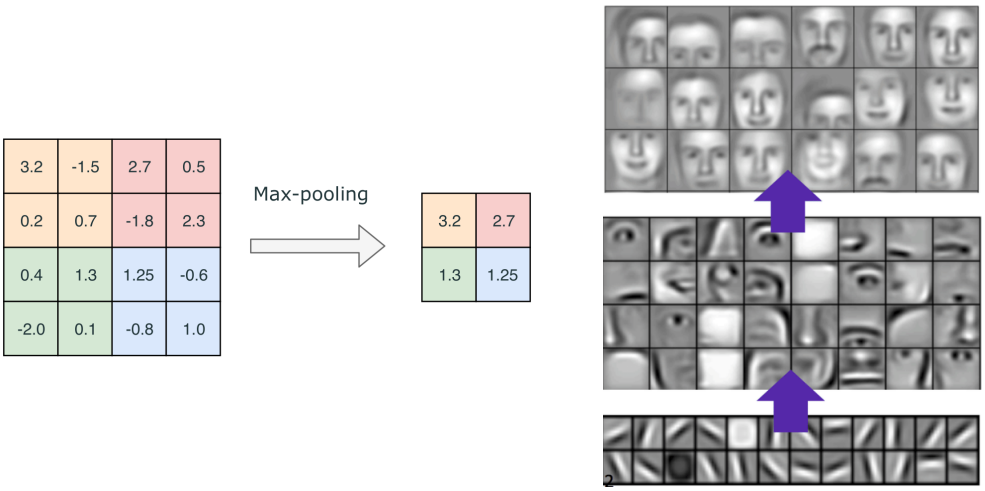

Figure 2.7. On the left: max-pooling kernel with spatial size equal to $2 * 2$ in action. On the right: visualization of the learned kernels as we go from the first to the deepest layers of a CNN trained on faces data set.[12][16]

2.2.6 Sequence modeling with Recurrent Neural Networks

Recurrent neural networks (a.k.a. RNNs) are a family of neural networks naturally built for processing sequential data. Different kind of requirements arise by the sequence's related task it needs to be addressed Figure 2.8 : one-to-many modeling (e.g. in an image captioning task, in which a textual description of a single image has to be provided), many-to-one modeling (e.g. in a sentiment analysis task, in which a sentiment has to be associated to an inputted sequence of words), or many-to-many modeling (e.g. in a machine translation task, in which a sequence of words has to be translated to a chosen language). The key intuition that made RNNs capable to analyze sequences of variable (and possibly never seen during training) length is to share the same parameters across different parts of the model. As seen in CNNs, in RNNs tha parameters are shared by the model across several time steps. Indeed, it is possible to model sequences even with 1D convolutions. The problem arising by this approach is mainly computational: in order to obtain a receptive field that fully covers the input sentence, the number of convolutional layers to be used grows linearly with the length of the sequence. The problem can be dampened via the adoption of dilated convolutions, where the kernel is not applied to all the elements of the sequence, but temporal holes of size n (i.e. the dilation value) are left uncolvolved. The dilated convolution at temporal position t is so computed as

[17]:

$$s(t) = (x *_n w)(t) = \sum_{a+n\cdot b=t} x(a)w(b) \tag{2.18}$$

In this way, the order of convolutional layers needed grows in a logarithmic fashion w.r.t. the sequence length. On the other hand, RNNs leverage internal hidden states h_j computed via the application of linear transformations (parameterized by trainable weights) on top of the input element of the temporal sequence. Indeed, let $\mathbf{x}$ the input sequence and $\hat{\mathbf{y}}$ be the output sequence. At each time step t, the internal state h_t and the output $\hat{y}_t$ of the RNN cell can be determined via the use of the following recurrence formula:

$$h_t = f_W(h_{t-1}, x_t) = \tanh(W_{hh}h_{t-1} + W_{hx}x_t) \qquad \hat{y}_t = W_{hy}h_t \tag{2.19}$$

As been said, both W_{hh}, W_{hx} and W_{hy} represent matrices which don't depend by the temporal step, i.e. are shared across all the RNN architecture. If dealing with a many-to-many sequence modeling task, Equation 2.19 can be applied straightforward. In case of one-to-many or many-to-one sequence modeling of length T, instead, input enforcing (i.e. $\forall\, t > 1,\, x_t = \hat{y}_{t-1}$) or the computation of $y_t \iff t = T$ can respectively be executed.

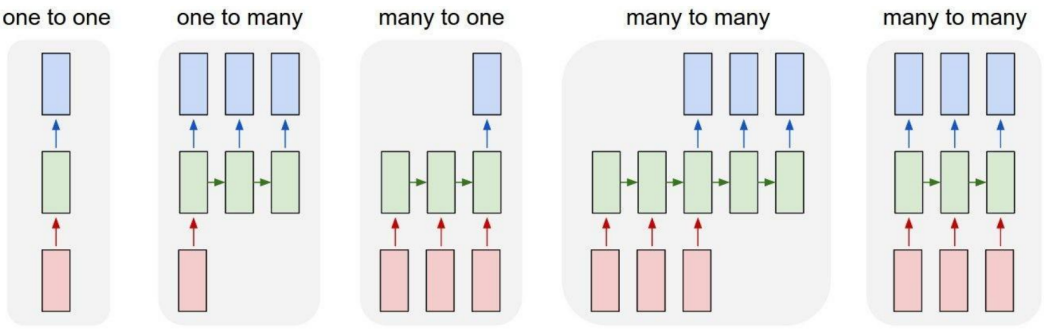

Figure 2.8. Different types of sequence modeling. Red cells indicates input elements, green cells indicate hidden states, blue cell indicate output elements.[13]

Training RNNs

The training of RNNs typically suffers of instability issues. They mainly arise by the fact that e.g. in the case of a one-to-many sequence modeling, producing a wrong output at a particular time step would affect all the subsequent RNN's predictions or, since the modeled sequences can be arbitrarily long, by the evidence that iterating multiplicative operations of derivatives across the length of the temporal sequence often leads to the shrinkage or explosion of the gradients. Indeed, e.g. in the case of many-to-many sequence modeling, it is possible to see Figure 2.9 that at each time step the computation of the gradients (for this kind of architectures executed with a differentiation technique a.k.a. **back propagation through time**) is affected both by the gradient coming from the following time step hidden state and from the actual time step prediction. To counteract the lack of input for all the time steps where $t > 1$, input enforcing has already been introduced. In this case, in order to remove the dependence by wrongly predicted output and said $\mathbf{y}$

the ground truth output sequence, at training time input enforcing is often replaced by teacher enforcing (i.e. $\forall t > 1, x_t = y_{t-1}$. On the other hand, in order to reduce the gradient explosion's issue, **gradient clipping** can be performed: the maximum value allowed for the gradient norms is fixed to a threshold value s, and for all the gradients computed with the back propagation through time technique the gradient re-scaling $\nabla' = \min\left(1, \frac{s}{\|\nabla\|_2}\right)\nabla$ is applied. In order to reduce the issue of vanishing gradients, instead, **truncated** back propagation through time can be executed: the time-unwrapped RNN is divided into chunks and the loss contributions (as also gradient back-flows) are solely computed over the cell belonging to a single chunk per time in favor of the diminution of the number of therms multiplied during the chain rule application. Even if those heuristics typically protect against the knock backs of those RNN's weaknesses, no theoretical guarantees about the network capabilities to retain dependencies over inputs further-in-time fed to the model can be provided.

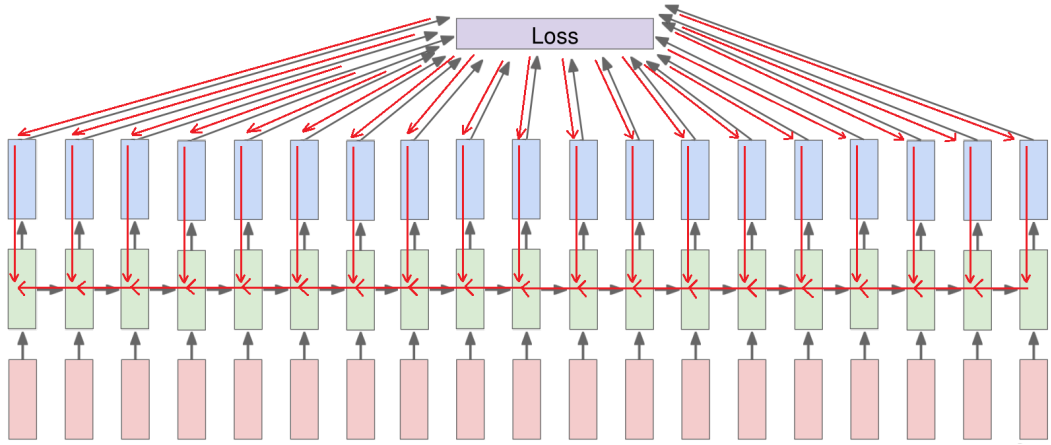

Figure 2.9. Many-to-many sequence modeling: forward flow and loss contributions described by the grey arrows. Backward gradients flow described by the red arrows.[13]

2.2.7 Attention is all you need

Attention is a complex cognitive function that is indispensable for human beings [18]. A key property of perception is that information is not processed in its whole entirety at once: instead, in order to efficiently perceive reality, humans selectively concentrate on single portions of the whole information set that turn out to be the most relevant for the addressed task, simultaneously ignoring, on the other hand, the rest of the perceivable information [19] [20]. For instance, in the framework of visual perception, only specific portions of the visual field are taken into account: if driving a car, for instance, humans typically learn that attention has to be mainly devoted to particular objects in sight (e.g. directions, traffic lights, road markers). This is a way for humans to quickly search for high-value information spots among the huge amount of information typically fed to the brain that have to be relied on if dealing with a certain kind of task, so to increase the efficiency and accuracy of information processing. In line with those results, attention mechanism has been enforced in many deep learning frameworks. It allows the modeling of dependencies among items in the input regardless of their relative distance in a parallel fashion, so

for instance reducing the typical long/short memory capacity of RNNs as also their computational knock backs due to the sequential nature of those architectures.[21].

Self-attention

Self-attention (a.k.a. intra-attention) is an attention mechanism relating different items of a single data point in order to compute a task-based attentioned representation for that data point. It has been adopted and implemented into networks dealing with a wide a variety of tasks covering in most of AI branches (e.g. image captioning in computer vision [22], machine translation in NLP [23]) providing both performances improvements and a new tool to interpret ANNs behaviours and decision-making policies. Let's dive into the working principle of self-attention for sequential data, where each one of n elements in the sequence is a vector $\in \mathbb{R}^k$. In a NLP task this can be achieved via the extraction of dense embeddings for the individual tokens. The input sequence $X \in \mathbb{R}^{n \times k}$ can then be regarded in parallel as the *query* sequence (denoted by Q), the *key* sequence (denoted by K) and the *value* sequence (denoted by V). Indeed, the attention function can be described as mapping a query and a set of key-value pairs to an output. A similarity comparison of the elements in the sequence can be expressed via the computation of the dot product $QK^\intercal$, which is then fed to a Softmax() acting row-wise to obtain probabilistic scores (refer to section 2.3 for further details over the Softmax). Since the L_2 norm of vector $\in \mathbb{R}^k$ generally tends to increase as we increase k [24], a re-scaling of $QK^\intercal$ is first executed. The output of the self-attention mechanism is finally equal to:

$$\text{Attention}(Q, K, V) = \text{Softmax}\left(\frac{QK^\intercal}{\sqrt{k}}\right)V \quad \text{with} \quad \begin{matrix} Q, K, V \in \mathbb{R}^{n \times k} \\ QK^\intercal \in \mathbb{R}^{n \times n} \end{matrix} \tag{2.20}$$

As clear, the above introduced self-attention mechanism doesn't contain trainable weights. In order to obtain task-adaptive representations of the Q, K and V matrices, they can eventually be individually and linearly transformed by 3 different matrices of trainable parameters W_q, W_k and $W_v \in \mathbb{R}^{k \times k}$ first to be sent to the self-attention pipeline:

$$Q' = QW_q \qquad K' = KW_k \qquad V' = VW_v \tag{2.21}$$

Both W_q, W_q and W_q elements are optimized during the model's learning pipeline. In order to let the model jointly attend to information from different representation sub spaces at different positions, multi-head attention can be implemented [21]. A defined amount h of multiple-head is fixed: each i_{th} head is provided with a triplet of trainable matrices $W_q^{(i)}, W_k^{(i)}, W_v^{(i)} \in \mathbb{R}^{k \times \frac{k}{h}}$. A final linear transformation is applied on the concatenated output of the multi-head attention Figure 2.10. The whole operation is summarized by the following expression:

$$\begin{aligned} &\text{MultiHead}(Q, K, V) = (\text{head}_1 \| \text{head}_2 \| ... \| \text{head}_h)W_o \\ &\text{with} \\ &\text{head}_i = \text{Attention}(QW_q^{(i)}, KW_k^{(i)}, VW_v^{(i)}) \end{aligned} \tag{2.22}$$

It is worth to notice that using trainable matrices $\in \mathbb{R}^{k \times \frac{k}{h}}$ allows to keep the size of this multi-head attention mechanism comparable to the one of the single head attention. Even if for some precise tasks the influence over the performances of multiple heads appears not to be statistically significant [25], it's granted that the presence of multiple attention heads can significantly speed up the training time leveraging parallel computations if h is properly set according to the number of available cores.

The transformer architecture

The **transformer** can finally be implemented, mainly made up of the *encoder* module and the *decoder* module Figure 2.10. The encoder module is made up N identical layers. In each of those layers the input is fed to the multi-head attention layer, summed to the plain input via a residual connection and then layer normalization is executed. The output is fed to a simple feed forward network, after which the same residual connection and layer normalization process is carried out. The output of the encoder module will so be an highly token-level-context aware representation of the inputted sequence. The decoder module is almost identical to the encoder one, except for the introduction of an additional multi-head attention layer that leverages the output of the encoder module to compute the key and value representations. The input of the decoder module is either the ground truth target sequence that has to be predicted (teacher enforcing is executed at training time) or the decoder's output itself. This input is used to compute the query vector needed by the MHA layer. Clearly, in both cases, a special token (e.g. `<START>`) needs to be inserted at the beginning of the encoder's input sequence in order to let the model start the inference process without any knowledge over the actual ground truth token that has to be predicted. At each time step, the decoder is solely able to output a sequence element: in order to avoid the decoder to be able to attend to subsequent sequence's items, a binary mask is applied to its first multi-head attention layer. At the top of the decoder module, a simple linear layer followed by a softmax activation maps the decoder representation to probabilities over the target vocabulary. It is worth to notice, since all the architecture turn out to be invariant under permutations of the input, that a *positional encodings* needs to be added to both the input and the output sequence. The most common way to produce positional encodings (vectors in $\mathbb{R}^k$, the same dimension of the sequence's items vector representation) is through the usage of sinusoidal functions: given a sequence item at position t, the j_{th} element PE_t^j of its positional encoding vector PE_t can be computed as;

$$\begin{aligned} \mathrm{PE}_t^{j=2i} &= \sin\left(\frac{t}{10000^{\frac{2i}{k}}}\right) \\ \mathrm{PE}_t^{j=2i+1} &= \cos\left(\frac{t}{10000^{\frac{2i}{k}}}\right) \end{aligned} \qquad \text{with} \qquad i \in \{0, 1, ..., \frac{k}{2}\} \tag{2.23}$$

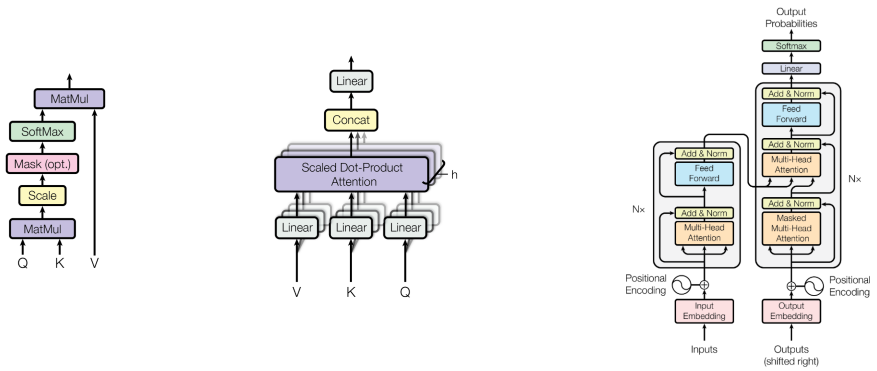

Figure 2.10. On the left: a non-trainable single head self-attention visualization. In the middle: a trainable h multi-headed self-attention visualization. On the right: full transformer architecture visualization.[21]

2.2.8 The expressiveness of latent codes

Given a supervised learning data set of feature points x_i and labels y_i, generative models aim to the learning of the joint probability distribution $p(x, y)$. When used for instance in a classification framework, they are then able to obtain the conditional distribution $p(y|x)$ via the Bayes rule, that can eventually be used to predict the most probable y for a specific x. Discriminative models (e.g. discriminative classifiers), on the other hand, aim to the direct modeling of the conditional distribution $p(y|x)$ (i.e. to the parametric reconstruction of the so called *decision boundary*) without any knowledge over the joint distribution $p(x, y)$ [26]. Generative models are also able to model the probability distribution of features $p(x)$ if no labels are provided. Even if the training of a generative model implies the solution of a much more general and difficult problem [27], a naive sampling of the learned target probability distribution allows for the execution of a useful task: generating of new data points.

Deterministic and variational auto-encoders

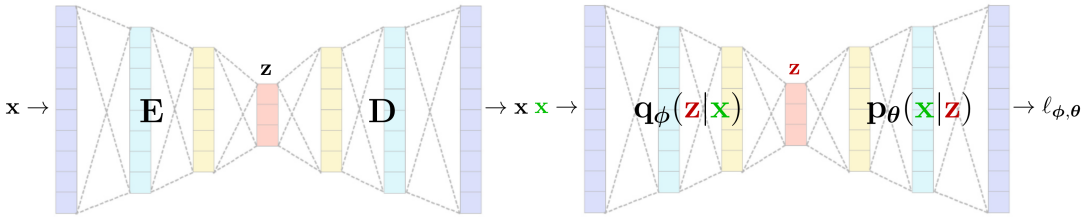

Figure 2.11. On the left: visualization of the DAE architecture. On the right: visualization of the VAE architecture. [28]

Deterministic auto-encoders (a.k.a. DAEs) and variational auto-encoders (a.k.a. VAEs) are among the most common generative models used, both of them basically built of an encoder-decoder pair Figure 2.11. In DAEs the encoder send the inputted information $\boldsymbol{x}$ to a bottle neck in order to compress it to latent representation (a.k.a. *code*) $\boldsymbol{z}$, which is then decompressed into $\tilde{\boldsymbol{x}}$. The optimal θ^* DAE's parameters are typically found via the minimization of a reconstruction loss $J(\theta)$: denoting with $E_\theta()$ the encoder's mapping and with $D_\theta()$ the decoder's mapping it is defined as:

$$J(\theta) = \sum_i \|\boldsymbol{x}_i - \tilde{\boldsymbol{x}}_i\| = \sum_i \|\boldsymbol{x}_i - D_\theta(\boldsymbol{z}_i)\| = \sum_i \|\boldsymbol{x}_i - D_\theta(E_\theta(\boldsymbol{x}_i))\| \tag{2.24}$$

Once properly trained, feeding a random sample of the latent space to the decoder module is enough to generate a new data point. It has been observed that [29] data points get sparsely clustered in DAE's latent spaces depending on their most representative features Figure 2.12. Sparsity of those clusters implies that the model won't be properly able to reconstruct a code belonging to poorly populated latent space's subsets, e.g. in all the cases when a generated data point with a mixture of representative features is desired. Enforcing the probability distribution of data points in the latent space to resemble a known and dense high dimensional probability distribution (e.g. a multivariate standard normal distribution) can be executed to smooth the transition among those clusters. To this purpose, VAEs kick in: the latent code $\boldsymbol{z}$ is indeed assumed as a set of parameters modeling an high dimensional probability distribution (as said usually a multivariate standard normal distribution). Once a random occurrence $\overline{\boldsymbol{z}}$ is sampled from such a distribution, it is

analogously decompressed to $\tilde{\boldsymbol{x}}$. The optimal θ^* VAE's parameters are found via the minimization of a loss $L(\theta, \phi)$ containing both a reconstruction term $J(\theta, \phi)$ and a regularization term $R(\theta, \phi)$ that force the encoder to map inputs towards the center of the latent space. The regularization term is computed via the Kullback-Leibler divergence, further analyzed in section 2.3. Denoting with $q_\phi(|x)$ the probability to observe z given x (modeled by the encoder's set of parameters ϕ), with $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ the probability to observe x given z (modeled by the decoder's set of parameters θ) and with $p(\boldsymbol{z})$ the multivariate standard normal distribution, the target loss is expressed as:

$$L(\theta, \phi) = J(\theta, \phi) + R(\theta, \phi) = -\sum_{\mathbf{z}} q_\phi(\mathbf{z}|\mathbf{x}) \log p_\theta(\mathbf{x}|\mathbf{z}) - \sum_{\mathbf{z}} q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{p(\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})} \quad (2.25)$$

Once the generation of a new data point is needed, it is sufficient to sample a point from the learned probability distribution of the latent space (in a portion of the latent space characterized by the presence of the features that are desired into the output) and feed it to the decoder to proceed with the reconstruction.

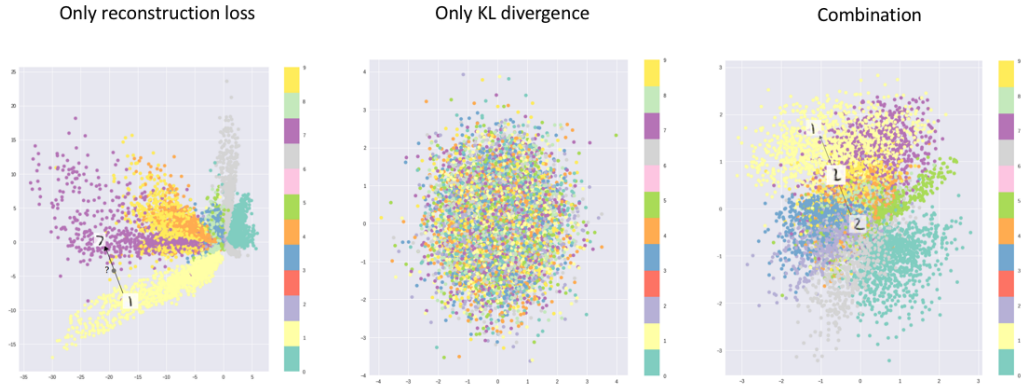

Figure 2.12. Visualization of latent spaces obtained under the minimization of the solely reconstruction loss (in DAE's fashion-like training, on the left), of the solely KL divergence (in the middle) and of their combination (in VAE's fashion-like training, on the right). At the side of the three plots, a colour bar provides the ground truth class for each of the data points. Even though training has been performed in an unsupervised fashion, data points automatically cluster depending on their label. [29]

Generative adversarial networks

As clear from the DAEs and VAEs introduction, no explicit guarantee can be given about the capability of the generative model to learn the true underlying probability distribution of the data $p_{\text{data}}(\boldsymbol{x})$. In the *adversarial networks* framework, instead, the generative model is pitted against an *adversary* Figure 2.13, i.e. a discriminative model that simply learns to distinguish whether a sample is coming from the generative model distribution or the true data distribution. The two networks are trained one in competition of the other. A common analogy is to think to the generative model as an art forger, aiming to produce realistic art pieces. The discriminative model can on the other hand be thought as an art expert:

receiving both forgeries and real art he aims to tell them apart. The competition in this game drives both networks to improve their methods until the forgeries are indistinguishable from the genuine art pieces [30] [31].

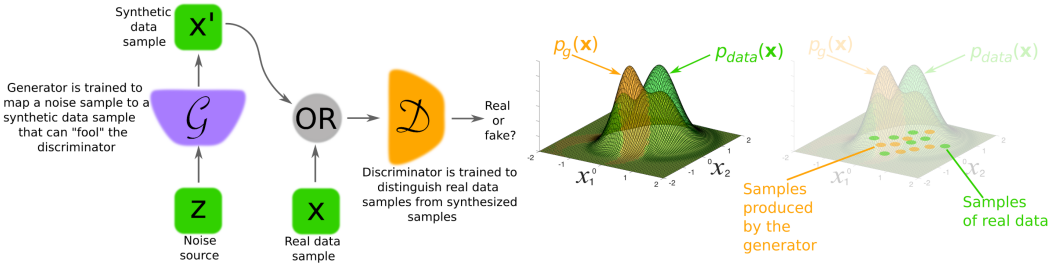

Figure 2.13. On the left: visual schema of the GANs setup. On the right: $p_g(\boldsymbol{x})$ approximate $p_{\text{data}}(\boldsymbol{x})$ better and better as the GAN training goes on. [30]

Let's denote with $p_z(\boldsymbol{z})$ the probability distribution over which random instances $\boldsymbol{z}$ are sampled. The generator can be interpreted as a differentiable mapping $\mathcal{G}_{\theta_g}(\boldsymbol{z})$ of those random instances $\boldsymbol{z}$ to the data space. The generator mapping is parameterized by the trainable weights θ_g. $p_g(\boldsymbol{x})$ denotes the probability distribution of the data points generated by $\mathcal{G}_{\theta_g}$. The discriminator $\mathcal{D}_{\theta_d}(\boldsymbol{x})$, on the other hand, can be interpreted as a differentiable mapping (parameterized by the set of trainable weights θ_d) of data points to a single scalar value. Being a binary classification framework, $\mathcal{D}_{\theta_d}(\boldsymbol{x})$ can be conventionally assumed as the probability that $\boldsymbol{x}$ belongs to the real data. $\mathcal{D}_{\theta_d}$ is so trained to maximize the probability of assigning the correct label to both training examples and generated samples, while $\mathcal{G}_{\theta_g}$ is trained simultaneously to minimize $\log(1 - \mathcal{D}_{\theta_d}(\mathcal{G}_{\theta_g}(z)))$. The training involves solving the min-max game:

$$\min_{\mathcal{G}_{\theta_g}} \max_{\mathcal{D}_{\theta_d}} V(\mathcal{G}_{\theta_g}, \mathcal{D}_{\theta_d}) \tag{2.26}$$

Where the value function $V(\mathcal{G}_{\theta_g}, \mathcal{D}_{\theta_d})$ is:

$$V(\mathcal{G}_{\theta_g}, \mathcal{D}_{\theta_d}) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}\left[\log\left(\mathcal{D}_{\theta_d}(\boldsymbol{x})\right)\right] + \mathbb{E}_{\boldsymbol{z}\sim p_z(\boldsymbol{z})}\left[\log\left(1 - \mathcal{D}_{\theta_d}(\mathcal{G}_{\theta_g}(\boldsymbol{z}))\right)\right] \tag{2.27}$$

During the training phase, alternatively, the parameters of one model are updated while the parameters of the other are kept fixed. Ideally, the discriminator is trained until the optimal configuration (that can be proven [31] to be $\mathcal{D}_{\theta_d}(\boldsymbol{x}) = \frac{p_{\text{data}}(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_g(\boldsymbol{x})}$) with respect to the current generator; then, the generator is again updated. It can also be proven [31] that given the value function $V(\mathcal{G}_{\theta_g}, \mathcal{D}_{\theta_d})$ and freezing the discriminator's parameters according to the min-max game, the generator aims to the minimization of the Kullback-leibler divergences sum $D_{KL}(p_{data} \| p_{data} + p_g) + D_{KL}(p_g \| p_{data} + p_g)$, which occurs only if $p_g = p_{data}$. Once properly trained, the generator will be capable to generate data points with an impressive degree of affinity to the true training data points Figure 2.14. This loopy training pipeline can be modified training the discriminator not until optimality but just for few iterations, simultaneously updating generator. Despite the theoretical existence of unique solutions, GANs training is often challenging and unstable. The quality (interpreted as the affinity with real world data) of the generated data points can be evaluated with different metrics, one of which (in the framework of image generation) is the Frèchet Inception Distance. More details and further explanations are provided in section 2.3.

Figure 2.14. Examples of images produced by the generator model after GANs training over the CELEBA-HQ data set. [32]

2.3 Loss functions, metrics and other utils

As seen in the previous section, training ANNs involves the minimization of risk functions, computed as empirical expectations over the available data points of relevant loss functions. Even though the choice of such losses might seem driven by the heuristic, they usually can be interpreted in a probabilistic fashion. The following section briefly introduce the most relevant loss functions, as also other utility functions often used while developing ML models.

2.3.1 Mean squared error loss

Let's take for example the MSE loss, measuring the average L_2 norm of the difference between the ground truth label y_i and the net response $\hat{y}_i = f_\theta(x_i)$ computed across each i_{th} data point (θ denotes the set of weights that parameterize the network mapping). Having a data set built with m data points of labels $\boldsymbol{y}$ and features $\boldsymbol{x}$ and assuming that $y_i = f(x_i) + \epsilon_i$ (where $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$ ideally collect all the sources of noise and the unmodeled behaviours), then $p(y_i|x_i, \theta) = p(\epsilon_i) = \mathcal{N}(y_i - f(x_i), \sigma^2)$. The likelihood function can so be expressed as:

$$\mathcal{L}(\theta|\boldsymbol{y}, \boldsymbol{x}) = \prod_{i=1}^{m} p(y_i|x_i, \theta) = \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{(y_i - f_\theta(x_i))^2}{2\sigma^2}\right] \tag{2.28}$$

Analogously the log-likelihood $l(\theta|\boldsymbol{y}, \boldsymbol{x})$:

$$l(\theta|\boldsymbol{y}, \boldsymbol{x}) = \log\left(\mathcal{L}(\theta|\boldsymbol{y}, \boldsymbol{x})\right) = m\log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right) - \frac{1}{2\sigma^2}\sum_{i=1}^{m}(y_i - f(x_i))^2 \tag{2.29}$$

That present a minimum in the point $\theta^* = \underset{\theta}{\text{argmin}} \sum_{i=1}^{m}(y_i - f(x_i))^2$. So, the minimization of the MSE is equivalent to providing a Maximum likelihood estimate for the parameters θ under the gaussian noise assumption made [33].

2.3.2 Entropy, Kullback-Leibler divergence and cross entropy loss

The *information content* (a.k.a. Shannon information) carried by the outcome x of a random variable $X \sim p_X(x)$ with support $\mathcal{X}$ can be thought as the amount of surprise of a particular outcome [34]. It can be expressed as $I(x) = -\log(p_X(x))$. Its expected value computed over the probability distribution $p_X(x)$, encoding the average level of information/surprise inherent to the X possible outcomes, it is known as the entropy of p_X, defined as:

$$H(p) = -\sum_{x \in \mathcal{X}} p_X(x) \log(p_X(x)) \tag{2.30}$$

The Kullback-Leibler divergence (a.k.a. relative entropy and denoted by $D_{KL}(P\|Q)$), on the other hand, is a statistical distance which measures how much a probability distribution P is different from a reference distribution Q. Equivalently, it quantifies the amount of information lost when modeling Q with P [34]. For discrete probability distributions having the same support $\mathcal{X}$ it is defined as:

$$\begin{aligned} D_{KL}(P\|Q) &= \sum_{x \in \mathcal{X}} P(x) \log\left(\frac{P(x)}{Q(x)}\right) = -\sum_{x \in \mathcal{X}} P(x) \log\left(\frac{Q(x)}{P(x)}\right) = \\ &= -\sum_{x \in \mathcal{X}} P(x) \log(Q(x)) - H(P) = H(P,Q) - H(P) \end{aligned} \tag{2.31}$$

Where H(P,Q) is referred to as the cross-entropy, a quantity closely related to the KL divergence: minimizing $H(P,Q)$ w.r.t. Q is by fact equivalent to minimize $D_{KL}(P\|Q)$, since the omitted term does not depend by Q [2]. In all the classification tasks where a data point's label y_i needs to be associated to the data point's features x_i, typically the model $f_\theta()$ outputs a probability distribution $Q_{\theta i}$ having support over the C possible label's classes. On the other hand, the ground truth label y_i can be represented by a degenerate probability distribution P_i (again, having support over the possible label's classes) having all the probability mass in the actual label's class. Indeed, once the *cross-entropy* loss is chosen, the minimization of the corresponding risk function (computed over the whole amount m of data set's points) $J(\theta)$:

$$J(\theta) = -\sum_{i=1}^{m} \sum_{j=1}^{C} P_i^j \log\left(f_\theta(x_i)^j\right) = -\sum_{i=1}^{m} \sum_{j=1}^{C} P_i^j \log\left(Q_{\theta i}^{\ j}\right) = -\sum_{i=1}^{m} \log\left(Q_{\theta i}^{\ y_i}\right) \tag{2.32}$$

Leads to a parameter's set θ that enforces the model's capability to output probability distributions $Q_{\theta i}$ statistically similar to the degenerate probability distributions P_i. Recalling that at inference time the predicted label $\hat{y}_i = \underset{y}{\text{argmax}}\ Q_{\theta i}$, the model will be ideally able to perform correct associations among features and labels.

2.3.3 Connectionist Temporal Classification

Many sequence learning tasks require the prediction of sequences of labels from unsegmented input data. Unsegmentation typically implies that no alignment among the input and the output sequence is provided. This issue affect also HTR: given a picture containing handwritten text, there is no way to priorly assume which slice of the image correspond to which character unless under the adoption of naive methods (e.g. assuming a fixed rate-of-characters-occurrence w.r.t. the width of the

picture's slice), which turn out to be very error prone. A Connectionist Temporal Classification network [35] allows to get around this issue. The output of a CTC network is provided by a softmax layer with an extra unit w.r.t. the target vocabulary that express the probability to observe a *blank* (i.e. the probability to observe no labels, which should not to be confused with the *white space* character). Let $\boldsymbol{x}$ be an input sequence of length T, $\boldsymbol{y}$ the corresponding network's output sequence and y_k^t the probability of observing label k at time t. If the target alphabet has a cardinality (i.e. the number of allowed characters) equal to L, having added an extra unit the output sequence $\boldsymbol{y} \in (0,1)^{(L+1)\times T}$. A path $\pi = (\pi_1, \pi_2, ..., \pi_T)$ is a way through the output matrix $\boldsymbol{y}$ (going from the first to the last frame and selecting one character π_t per frame with probability $y_{\pi_t}^t$) that essentially encodes the predicted labelling. The probability to observe a path π given an inputted sequence $\boldsymbol{x}$ is:

$$p(\pi|\boldsymbol{x}) = \prod_{t=1}^{T} y_{\pi_t}^t \tag{2.33}$$

A collapsing function $\mathcal{B}()$ subsequently map paths π to labellings $\boldsymbol{l}$ first removing duplicates in the path, then removing the blank characters. Denoting with "_" the blank character, then for instance $\mathcal{B}$(h_eee_l_ll_o) = (hello), while $\mathcal{B}$(h_e_ee_ll_oo) = (heelo). As clear, the lenghts of the labelings $\boldsymbol{l}$ will be $\leq T$. The probability to observe a labeling $\boldsymbol{l}$ can therefore be obtained summing the probabilities of all the paths that turn into $\boldsymbol{l}$ under the application of $\mathcal{B}$:

$$p(\boldsymbol{l}|\boldsymbol{x}) = \sum_{\pi|\mathcal{B}(\pi)=\boldsymbol{l}} p(\pi|\boldsymbol{x}) \tag{2.34}$$

The network's inferred labelling $h(\boldsymbol{x}) = \boldsymbol{l}^*$ will then be chosen as the most probable labelling, i.e.:

$$\boldsymbol{l}^* = \underset{\boldsymbol{l}}{\text{argmax}}\ p(\boldsymbol{l}|\boldsymbol{x}) \tag{2.35}$$

Since no optimal algorithm is known to execute the above mentioned labeling identification (a.k.a. *decoding* step), several heuristics can be applied, the simplest of which (a.k.a the *best path decoding* heuristic) make the assumption that:

$$h(\boldsymbol{x}) \approx \mathcal{B}(\pi^*) \qquad \text{with} \qquad \pi^* = \underset{\boldsymbol{\pi}}{\text{argmax}}\ p(\pi|\boldsymbol{x}) \tag{2.36}$$

Even though this heuristic allows for the quickest evaluation of $h(\boldsymbol{x})$, as in the example depicted in Figure 2.15, the best path approach happens to fail in the correct identification of $\boldsymbol{l}^*$: indeed, under the best path approximation, the optimal labeling inferred is $\boldsymbol{l}^*$ = " ", while the actual optimal labeling is $\boldsymbol{l}^*$ = "a". Also, it is desired for our network to solely output meaningful/known words. Due to those reasons, a more complex algorithm known as *word beam search* needs to be implemented [36]. Let's first introduce the notion of *prefix tree*, a tree-data structure made up of edges and nodes. Each of the edges is labeled by a character and points to the next node: a node is word-flagged if that node represents a word. Because of that a word is encoded in the prefix tree by a path starting at the root node and following edges labeled with the corresponding characters of the word. The implementation of a prefix tree simplify several tasks. For instance, if querying the characters which can follow a given prefix is desired, it is enough to identify the node corresponding to the given prefix: all the outgoing edge labels determine the allowed subsequent characters. Or, if all the words which contain a given prefix

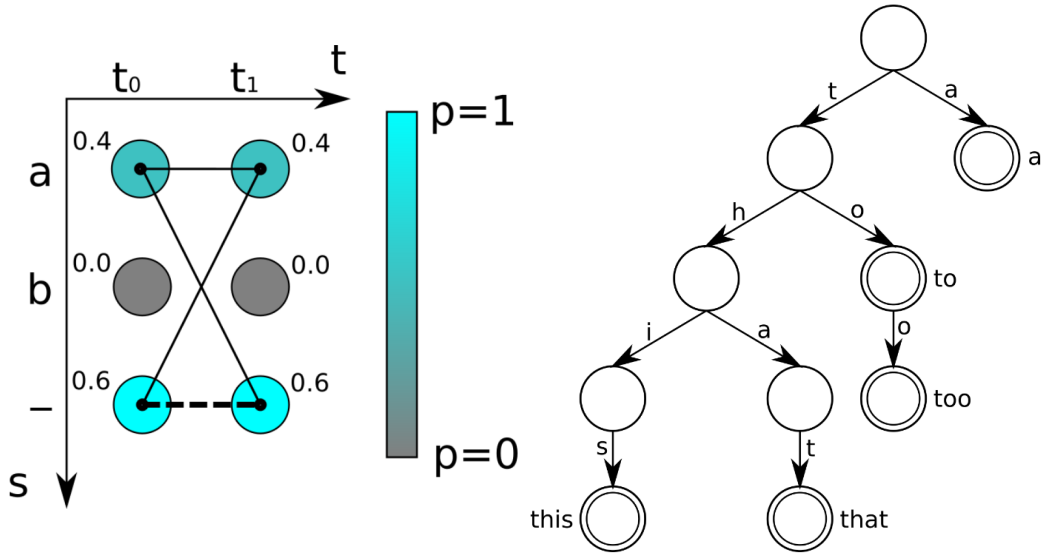

Figure 2.15. On the left: setup in which the best path approach fails in the correct identification of $\boldsymbol{l}^*$. The path π_1 (identified by the dashed line) turns into the wrong maximum probability labeling " ", while both the paths π_2 and π_3 (identified by the thin lines) turn into the correct maximum probability labeling "a". On the right: visualization of the prefix tree. Nodes representing words can be identified by the word flag, i.e. an inner circle in the node. [37]

want to be identified, it is enough to collect all the descendant word-flagged nodes starting from the node corresponding to the given prefix.

As mentioned, the key desired property for our word beam search algorithm is both to be able to output arbitrary characters to allow the output of digits and punctuation marks (a distinguishing feature of the *vanilla beam search* algorithm) and to constrain the outputted words to a given dictionary to avoid mispellings (a distinctive characteristic of the *token passing* algorithm. More details about those two basic algorithms are provided in [37].). Word beam search iterates through the ANN output and iteratively creates *beams* (i.e. text candidates): starting from an empty beam, all the possible characters are added during the first iteration (eventually, in order to reduce the algorithm spatial complexity, removing the ANN lowest scored ones). At this point, each of the beam is either in a *word-state* (i.e. adding precise characters to that beam leads to the construction of a word. For instance the beam "H" eventually leads to "$Hello$" under proper choice of characters.) or in a *non-word-state* (e.g. the beam "3"). Once a beam is in word-state, leveraging a prefix tree built over a large dictionary, the solely selection of characters leading to prefix-tree's word-flagged nodes. This is equal to descend only the prefix-tree paths leading to words which contain a given prefix. The transition from word-state beams to non-word-state beams is allowed only when a word is completed, while the opposite is always allowed. The process is iterated until the whole ANN output is processed.

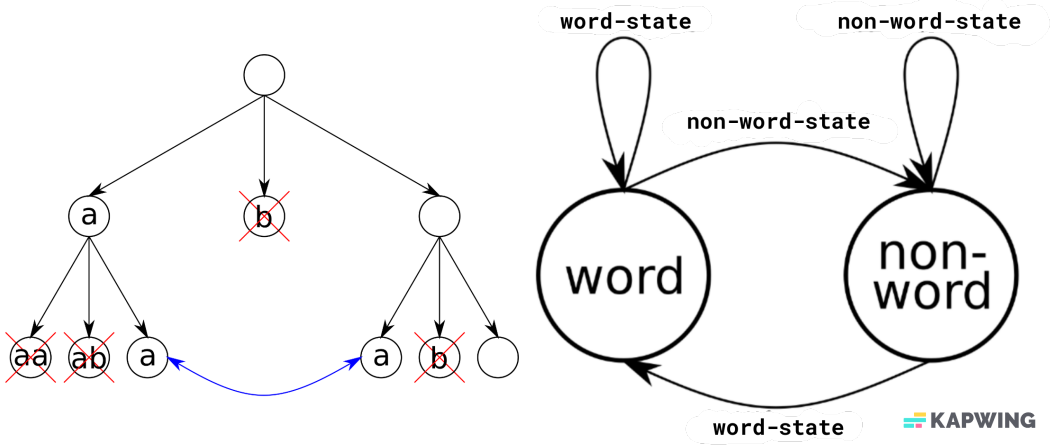

Figure 2.16. On the left: visualization of two iterations of the beam construction process. ANN lowest scored characters are removed by the tree. On the right: visualization of the transitions among word-state/non-word-state beams. [37]

Given a ground truth labeling $\boldsymbol{l}^{\text{true}}$, the network can now be trained via the minimization of the CTC loss, defined as the negative log probability of correctly labelling the sequence:

$$\text{CTC}(\boldsymbol{l}^{\text{true}}, \boldsymbol{x}) = -\log(p(\boldsymbol{l}^{\text{true}}|\boldsymbol{x})) \tag{2.37}$$

Further details over the training algorithm and the other heuristics can be found in [35].

2.3.4 Wasserstein metric and Frichèt Inception Distance

The Wasserstein distance (a.k.a. Kantorovich-Rubinstein metric) is a distance function defined over probability distributions [38]. Aiming to transform one probability distribution $\mu(x)$ into another probability distribution $\nu(x)$, it expresses the minimum amount of probability mass that has to be moved times the mean distance (w.r.t. the probability distribution support) it has to be moved. Let's denote with $\gamma(x, y)$ the function describing the amount of probability mass that has to be moved from $\mu(x)$ to $\mu(y)$. For consistency reasons, both of the following needs to be verified:

$$\int \gamma(x, y)dy = \mu(x) \qquad \int \gamma(x, y)dx = \nu(y) \tag{2.38}$$

Both of the above (respectively stating that the amount of probability mass removed from $\mu(x)$ needs to be equal to the amount of probability mass that was in $\mu(x)$ originally, and the amount of probability mass brought to $\mu(y)$ needs to be equal to the amount of probability mass that was in $\nu(x)$ originally) leads to the equivalent assumption that $\gamma(x, y)$ is the joint probability distribution having as marginal distributions $\mu(x)$ and $\nu(y)$. Being the infinitesimal mass transported from x to y equal to $\gamma(x, y)dxdy$ and $c(x, y)\gamma(x, y)dxdy$ the the cost of this infinitesimal movement, the total cost C of the transport plan can so be expressed as:

$$C = \int\int c(x, y)\gamma(x, y)dxdy = \int c(x, y)d\gamma(x, y) \tag{2.39}$$

Denoting with $\Gamma(x, y)$ the set of joint probability distributions having as marginals $\mu(x)$ and $\nu(y)$ and being the transport plan not unique, the optimal cost can be

identified as:

$$C^* = \inf_{\gamma \in \Gamma(x,y)} \int c(x,y) d\gamma(x,y) \tag{2.40}$$

This is equivalent to the definition of the $W_1(\mu, \nu)$ Wasserstein distance between μ and ν. For a general p, the W_p distance is defined as:

$$W_p(\mu, \nu) := \left(\inf_{\gamma \in \Gamma(x,y)} \int c(x,y)^p d\gamma(x,y) \right)^{\frac{1}{p}} \tag{2.41}$$

Let's now introduce a metric used to assess the quality (in terms of affinity to the real world images) of generated images [39], and consider the computation of the squared $W_2(\nu_1, \nu_2)$ Wasserstein distance (where $\nu_1 = \mathcal{N}_n(\mu_1, \Sigma_1)$ and $\nu_2 = \mathcal{N}_n(\mu_2, \Sigma_2)$. If ν_1 represents the distribution of some neural network internal representations of generated images, while ν_2 the distribution of the same network internal representations of real world images, then the $W_2(\nu_1, \nu_2)$ is equal to the **Fréchet inception distance**, the standard metric to assess the quality of generative models. The FID can so be computed as:

$$\text{FID} = W_2(\nu_1, \nu_2) = \|\mu_1 - \mu_2\|_2^2 + \text{Trace}\left(\Sigma_1 + \Sigma_2 - 2\left(\Sigma_1^{\frac{1}{2}} \cdot \Sigma_2 \cdot \Sigma_1^{\frac{1}{2}} \right)^{\frac{1}{2}} \right) \tag{2.42}$$

Inception V3 trained on ImageNet is often chosen as a reference neural network, used to extract the above mentioned probability distributions.

2.3.5 Luce's choices and the softmax function

In probability theory, the Luce's choice axiom defines the probability to choose an item i over a pool containing J items. It states that:

$$\mathbb{P}(i) = \frac{w(i)}{\sum_{j=1}^{J} w(j)} \tag{2.43}$$

Where $w(i)$ is a positive score associated to the i_{th} item. The softmax function (a.k.a. normalized exponential function [40]) leverages the Luce's choice axiom: it is a generalization of the logistic function to multiple dimensions [41], and is often used as the last activation function of a network to normalize the output to a probability distribution over predicted output classes. Having an input vector $z \in \mathbb{R}^K$, the exponential function is first applied (in order to obtain a positive score for each element $\in z$), subsequently followed by a normalization operation. The softmax function $\text{Softmax} : \mathbb{R}^K \to (0,1)^K$ is so defined:

$$\text{Softmax}(z)_i = \frac{\exp(z_i)}{\sum_{j=1}^{K} \exp(z_j)} \qquad \text{s.t.} \qquad \sum_{j=1}^{K} \text{Softmax}(z)_j = 1 \tag{2.44}$$

2.3.6 Levenshtein distance and the character error rate

In information theory, linguistics, and computer science, the Levenshtein distance is a string metric for measuring the difference between two sequences. Given a ground truth sequence of characters y of length $|y|$, a predicted sequence of characters $\hat{y}$ of length $|\hat{y}|$, denoting with $y[n]$ the n_{th} character of sequence y and with $\text{tail}(y)$ the y sequence deprived of the first character $y[0]$, the Levenshtein distance between the two sequences $\text{lev}(y, \hat{y})$ is defined as:

$$\text{lev}(y,\hat{y}) = \begin{cases} |y| & \text{if } |\hat{y}| = 0 \\ |\hat{y}| & \text{if } |y| = 0 \\ \text{lev}(\text{tail}(y), \text{tail}(\hat{y})) & \text{if } y[0] = \hat{y}[0] \\ 1 + \min \begin{cases} \text{lev}(\text{tail}(y), \hat{y}) \\ \text{lev}(y, \text{tail}(\hat{y})) \\ \text{lev}(\text{tail}(y), \text{tail}(\hat{y})) \end{cases} & \text{otherwise} \end{cases} \tag{2.45}$$

The computation of the character error rate among the two sequences $\text{CER}(y, \hat{y})$ is now straight forward, being it equal to:

$$\text{CER}(y,\hat{y}) = \frac{\text{lev}(y,\hat{y})}{|y|} \tag{2.46}$$

2.3.7 t-SNE projection

The t-distributed stochastic neighbor embedding (t-SNE) [42] is a statistical non linear dimensionality reduction method originally developed by Sam Roweis and Geoffrey Hinton to visualize high-dimensional data by giving each data point a location in a two or three-dimensional space. Denoting with $\mathcal{X} \in \mathbb{R}^{N \times d} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_N\}$ the collection of our data points (where N recalls the number of data points and d the dimensionality of the space in which each data point is embedded), the t-SNE method firstly foresee the computation of the conditional probabilities $p_{i|j}$:

$$p_{i|j} = \frac{\exp\left(\frac{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2}{2\sigma_i^2}\right)}{\sum_{k \neq i} \exp\left(\frac{\|\boldsymbol{x}_i - \boldsymbol{x}_k\|^2}{2\sigma_i^2}\right)} \tag{2.47}$$

Where each σ_i indicates the bandwidth of each Gaussian kernel adapted to the data density (i.e. smaller values of σ_i are used in denser parts of the data space). It can be easily verified that $\sum_i p_{i|j} = 1 \; \forall j$. Next, the computation of the symmetric probability distribution $p_{ij} = \frac{p_{i|j} + p_{j|i}}{2N}$ is executed. The t-SNE method finally aims to learn a map $f() : \mathbb{R}^d \mapsto \mathbb{R}^k$ (where k is commonly $\in \{2, 3\}$) reflecting the symmetric probability distribution p_{ij} as well as possible in the reduced space $\mathbb{R}^k$. Such a goal is achieved minimizing the Kullback-Leibler divergence $\text{D}_{\text{KL}}(P\|Q)$ through gradient descent, where $\text{D}_{\text{KL}}(P\|Q)$ is defined as:

$$\text{D}_{\text{KL}}(P\|Q) = \sum_{i \neq j} p_{ij} \log\left(\frac{p_{ij}}{q_{ij}}\right) \quad \text{with} \quad q_{ij} = \frac{\left(1 + \|f(\boldsymbol{x}_i) - f(\boldsymbol{x}_j)\|^2\right)^{-1}}{\sum_l \sum_{k \neq l} \left(1 + \|f(\boldsymbol{x}_k) - f(\boldsymbol{x}_l)\|^2\right)^{-1}}$$

It is worth to notice that unlike other dimensionality reduction mechanism no meaningful association can be performed between the k axis representing the reduced representational space and the d axis representing the original embedding space.

2.3.8 Adam optimizer

Unlike vanilla Stochastic Gradient Descent, the Adam optimization algorithm [43] exploits estimates of first and second moments of the gradients so to compute adaptive learning rates for the optimized parameters. Indeed, the name *Adam* derives from *Adaptive Moment estimation*. Initializing $m_t|_{t=0} = v_t|_{t=0} = 0$, at training time step $t+1$ the updated weights θ_{t+1} are computed as follows:

$$
\begin{aligned}
v_t &= \beta_2 v_{t-1} + (1-\beta_2)\nabla_{\theta_t} J(\theta)^2 \\
\widehat{v_t} &= \frac{v_t}{1-\beta_2^t} \\
m_t &= \beta_1 m_{t-1} + (1-\beta_1)\nabla_{\theta_t} J(\theta) v_t \\
\widehat{m_t} &= \frac{m_t}{1-\beta_1^t} \\
\theta_{t+1} &= \theta_t - \widehat{m_t}\left(\frac{\alpha}{\sqrt{\widehat{v_t}+\epsilon}}\right)
\end{aligned}
$$

Where:

- α denotes the learning rate;
- ϵ is a small constant exploited to avoid division by 0;
- m_t denotes the biased gradient first moment estimate;
- $\widehat{m_t}$ denotes the bias-corrected gradient first moment estimate;
- v_t denotes the biased gradient second moment estimate;
- $\widehat{v_t}$ denotes the bias-corrected gradient second moment estimate;
- $J(\theta)$ denotes the function being minimized;
- β_1, β_2 denote the running average/ exponential decay rate parameters;

Chapter 3

Related works

Generative models have been in existence for many decades. Even though those models tackle a task which is by its inner nature *harder* then the one tackled by analogous discriminative models, leveraging the technologies and practices of deep learning they are currently allowing the generation of hyper realistic data samples. Some of those advances have now reached the public sphere via the wide spreading success of generated realistic images, voices, or movies: the so called deep fakes. Despite these successes, several mathematical and practical issues limit the broader use of those deep generative models. Other than the already mentioned generative-problem hardness, the identification or definition of suitable and theoretically robust metrics able to quantify the effectiveness and quality of generative models regardless of their domain of application turned out to be hard too. The lack of those general measures of performances still poses as a reason for delay into the development of better generative models, as far as it's not yet clear when and whether a model's architecture has to be preferred over another. Nevertheless, the usage of generated data often occurs in the attempt to improve tasks like:

- Missing parts of data completion: missing data is obviously a problem while aiming to evaluate a model correctly. With generative models, it is possible to generate data in those unexplored regions of the data distributions;
- Generation of data with pre-defined/user inputted content;
- Data sets augmentation, inducing improvements of discriminative model's performances and training pipelines;

Among the generative model's architectures which turned out to mostly affect the above mentioned tasks, two main branches can be identified: **autoencoder/ variational autoencoder** based methods and **generative adversarial networks** based methods [44].

3.1 AE/VAE based methods

With the aim to improve the vanilla autoencoder architecture, several modifications have been introduced. Denoising Autoencoder (DAE) [45] model have been introduced to obtain robust representation after executing the learning pipeline during which giving corrupted data as input. Stacked Denoising Autoencoder (SDAE) [46] is obtained stacking several DAEs into a single network. To reduce the overfitting and the sparsity of latent codes as also to increase the control over desired generated

features, Variational Autoencoder (VAE) [47] have been developed. Adversarial Autoencoder (AAE) [48] mixes the benefits of VAE with the ones of adversarial training. Denoising Variational Autoencoder (DVAE) [49] combines VAE and DAE approaches to get a robust model via the stochastic injection both at the input and at the hidden layer level. β Variational Autoencoder (β-VAE) [50] is proposed to improve the disentanglement capabilities of the VAE via the application of a β hyperparameter to tune the effect of the KL term in the loss.

3.2 GANs based methods

After the introduction of the GANs framework [31], Conditional Generative Adversarial Net (CGAN) [51] has extended the original GANs paradigm to generate facial images conditioning the generated image to present user desired features. IC-GAN [52] leverages similar conditioning to generate images similar to the ones of the COCO dataset. Deep Convolutional Generative Adversarial Network (DC- GAN) [53] both the generator and the discriminative models of the vanilla GAN framework are deep convolutional networks. Improved GAN [54] deeply analyze the GAN training pipeline in order to improve it. Bidirectional GAN (BiGAN) [55] proposes a way to perform a mapping of the generated data back to the latent space, i.e. an inverse mapping w.r.t. the one performed by a GAN. Another attractive study going one further step in terms of image generation conditioned on inputted sentences is GAN-INT-CLS [56]: GAN trains conditional text features obtained by a recurrent encoder, subsequently used to generate the images. In particular, in the framework of deep learning-based offline handwritten text generation [57], Haines et al. [58] proposes a way to generate new handwritten text in a distinct style extracted by source images, even though limited to generate characters that are in the source images. Chang [59] used CGAN to morph images of isolated handwritten characters of Chinese language from a given style to another. ScrabbleGAN, introduced by Fogel et al. [60], synthesizes handwritten word using a fully convolutional architecture. Here, the characters generated have similar receptive field width. Zhang, Goodfellow et al. introduced the SAGAN [61], a GAN equipped with self-attention modules to model long-range dependencies when generating images. Davis et al. [62] introduced a model that generates handwritten text conditioned to both user inputted text and style. Similarly, Gan et Wang presented HiGAN [63], which allows for a more flexible control of the handwriting styles. Bhunia et al. [57] enriched their GAN with a transformer architecture that enables the disentanglement of content and style both at character, word and sentence level.

Chapter 4

The data set

In order to train the chosen generative SOTA models, properly described in Chapter 6, the IAM handwritten database has been chosen. It is worth to notice that, for the purposes of the generative task, it was desired to implement and train models being able to generate handwritten text given an arbitrary English, Italian or Spanish query text input. Since the English alphabet (and consequently IAM) vocabulary lacks of special characters used, on the other hand, in the Italian and Spanish language, a collection of extra handwritten text has been carried out among Enel employees and La Sapienza università di Roma Data Science students in order to enlarge the above mentioned benchmark databases. The corporate ENEL production data set (named ENEL_{DS} for privacy related reasons) is also presented. Furthermore, three distinct textual data sets have been chosen to be used as query inputs for the final generative stage. Details of the mentioned data sets are reported in the following sections and in Table 4.2.

4.1 IAM data set

The IAM Handwriting Database [64] [65] (firstly published at the 1999 International Conference on Document Analysis and Recognition) contains forms of handwritten English text, manually labeled both at sentence, text line and word level. Being one of the largest handwritten data sets available, it is often considered as the reference point once it comes to handwritten-related tasks. The main features of its latest version (IAM 3.0) are summarized below:

- 657 writers contributed samples of their handwriting;
- Built over 1'539 pages of scanned text;
- 5'685 isolated and labeled sentences;
- 13'353 isolated and labeled text lines;
- 115'320 isolated and labeled words;

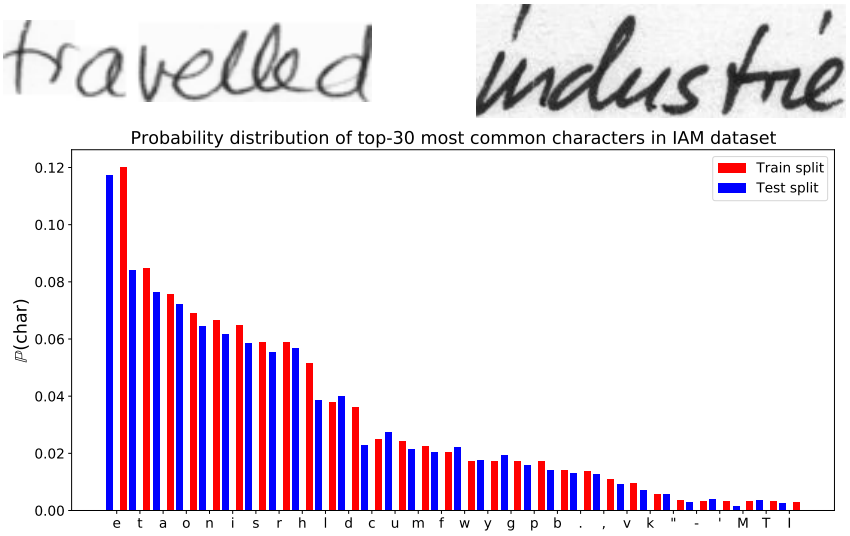

Figure 4.1. Above: Visualization of IAM sample images. The associated labels are, respectively, "travelled" and "industrie". Below: probability distribution of the top-30 most common characters in IAM train and test splits.

4.2 ENEL$_{DS}$

The ENEL$_{DS}$ handwriting data set is composed of 50'383 handwritten Italian words written by Enel customers. It is mostly composed of extremely dirt images depicting out-of-vocabulary words (i.e. codes, VAT numbers, identifiers, dates, addresses, etc.). For privacy related reasons, no sample image can be visualized in the present thesis. An aggregated description of the most common character's distribution is provided in Figure 4.2.

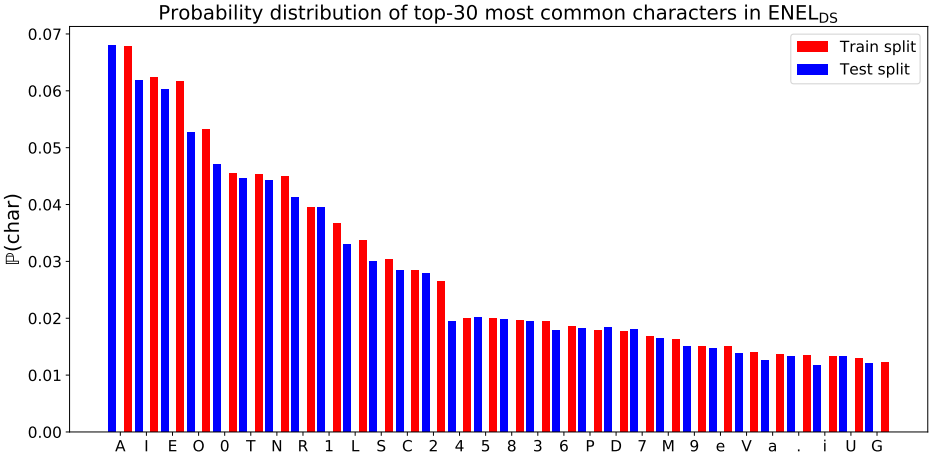

Figure 4.2. Probability distribution of the top-30 most common characters in ENEL$_{DS}$ train and test splits.

4.3 Italian & Spanish special characters collection

As mentioned in the present chapter introduction, the main obstacle faced towards the implementation of a generative model being able to generate text in an arbitrary language among English, Spanish and Italian was the absence in the English alphabet of special characters often encountered in the Italian and Spanish language. The missing characters are listed below for each of the languages:

- **Italian alphabet**: à, è, é, ì, ò, ù;
- **Spanish alphabet**: á, é, í, ó, ú, ü, ñ;

Being the IAM handwritten database in lack of those accented characters, any implemented architecture trained upon it would be naturally incapable of generating handwritten text images given a textual input containing those characters (e.g. "perchè", "niño" and so on). Two distinct forms have so been built and distributed across the Data Science MSc and Enel colleagues in order to collect handwritten text images for 60 different words containining the listed characters (in total, 120 unique words shown in Table 4.1). The special character distribution, shown in Figure 4.3, highlights the fact that, for the purposes of handwritten text generation, the main focus has been set on the Italian special set of characters.

perché	più	encontrarás	ragù	falò
comò	quotidianità	sì	sennò	número
ciò	niño	ambigüa	dì	bignè
è	perciò	advertí	virtù	mañana
mengüe	giù	público	nazionalità	scimpanzé
emù	giovedì	tabù	purè	bilingüe
separé	revolución	menú	ovvietà	supplì
finché	condición	bebè	sé	está
aquí	tiramisù	casinò	lunedì	bensì
biodegradabilità	embè	cioè	observación	frangibilità
magnetizzabilità	sorprendí	aggressività	pedalò	né
frappé	così	caffè	panamá	señor
venerdì	paltò	podestà	pagherò	giurì
ñublar	signorsì	lassù	espressività	fábrica
zulù	sufflè	caribù	púrpura	macché
perù	pingüino	indù	tribù	relè
dúctil	aliña	sfottò	bistrò	pressocché
baño	opción	extensión	fraternità	ánimo
núcleo	mezzodì	síntesis	solución	simpático
crème	sanità	energía	ahimè	bohème
onírico	portascì	rondò	tournée	rosé
martedì	lassù	accessibilità	umiltà	lacchè
granché	rococò	buondì	karkadè	flambé
ilarità	trentatré	lengüeta	risciò	güimba

Table 4.1. Selected words to build the 2 distinct collection forms.

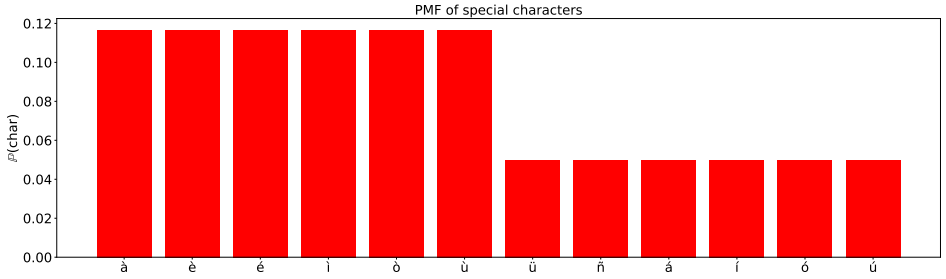

Figure 4.3. Probability mass function computed over the 120 words corpus.

Both of the forms have been distributed sharing an Allocate Monster link: this allowed to randomly assign one of the two forms to each of the collection participants. 48 different persons participated: after the response's collection, all the pages have been fed to the Tesseract Open Source OCR Engine [66] in order to automatically produce meaningful bounding boxes around the words. The Tesseract pre-processing output has then been fed to the VGG Image Annotator [67] in order to manually verify and adjust both the bounding boxes and the textual label associated to each of the word patches.

Figure 4.4. Visualization of samples collected by the Data Science and Enel colleagues

4.4 Most common English, Italian & Spanish words

After the execution of the training pipeline for all the selected SOTA architectures, a big corpus of English, Italian and Spanish words is needed to proceed with the generation stage. The "Oxford 5000" [68], the "Italian Frequency Dictionary for Learners" [69] and the "A Frequency Dictionary of Spanish: Core Vocabulary for Learners" [70] have so been chosen. They respectively contain 5'000, 10'000 and 5'000 most common words for each of the languages. Indeed, since the generative stage is needed to produce a dataset that will be subsequently used to retrain the Enel corporate OCR model, the choice of corpuses containing the most used words for each of the languages seemed a meaningful choice. It is worth to notice that both the "English Common Dictionary capitalized" and the "Italian Common Dictionary without special characters" have been manually created: the first, adding 500 words randomly sampled from "Oxford 5000" (and subsequently capitalized) to the original corpus. The latter, removing all the words containing special characters from the original corpus.

Dataset	**# Chars**	**# U.W.** Train	**# U.W.** Test
IAM	79	13'170	1'989
ENEL_{DS}	77	14'272	2'715
English Common Dictionary	54	4'715	N.A.
English Common Dictionary (Capitalized)	63	5215	N.A.
Italian Common Dictionary	59	10'105	N.A.
Italian Common Dictionary (Without special characters)	53	9'878	N.A.
Spanish Common Dictionary	49	4'977	N.A.

Table 4.2. Data sets statistics. **# U.W.** recall the number of unique words.

Chapter 5

Handwritten OCR

The corporate handwritten Optical Character Recognition system is fully inspired to the architecture proposed by B. Shi, X. Bai et al. [71]: an end-to-end trainable Convolutional Recurrent Neural Network which naturally handles sequences of arbitrary lengths. The proposed architecture has been implemented with the **Tensorflow** ML framework and it is basically made up of three main component, i.e a Convolutional module, a Recurrent module and a Transcription module.

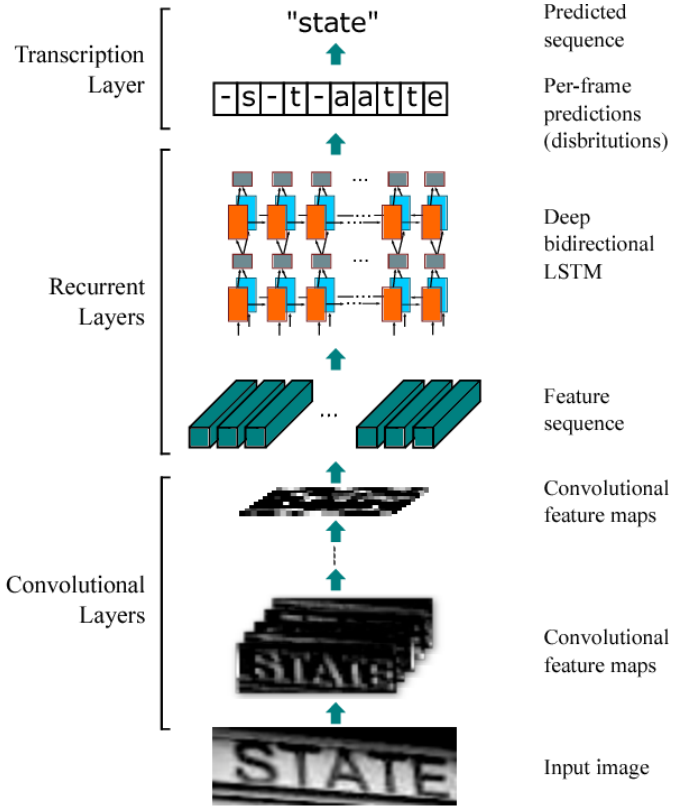

Figure 5.1. Visualization of the OCR architecture. [71]

5.1 Convolutional module

The convolutional module leverages 5 convolutional blocks where the information flows in the classic feed-forward fashion. Each of the convolutional block is respectively made up of a convolutional layer, a batch normalization layer, a ReLu layer and a MaxPooling layer. The hyperparameter's sets used to define layers in each convolutional block are reported in Table 5.1. Looking at the convolutional module output size, it is worth to notice that the width value of the input image has been compressed from 128 to 32, i.e. the temporal steps leveraged by the subsequent recurrent module. The height value of the original image, on the other hand, has been fully compressed from 32 to 1. Each output vector $\in \mathbb{R}^{256}$ corresponding to time step $t \in \{1, ..., 32\}$ is denoted with c^t.

Layer type/ID	Description	Output size
Input	gray-valued images	(B, 128, 32, 1)
Convolutional + BN/1	N_F=32, F=(5, 5), S=(1, 1, 1, 1)	(B, 128, 32, 32)
MaxPooling/1	F=(2, 2), S=(1, 2, 2, 1)	(B, 64, 16, 32)
Convolutional + BN/2	N_F=64, F=(5, 5), S=(1, 1, 1, 1)	(B, 64, 16, 64)
MaxPooling/2	F=(2, 2), S=(1, 2, 2, 1)	(B, 32, 8, 64)
Convolutional + BN/3	N_F=128, F=(3, 3), S=(1, 1, 1, 1)	(B, 32, 8, 128)
MaxPooling/3	F=(1, 2), S=(1, 1, 2, 1)	(B, 32, 4, 128)
Convolutional + BN/4	N_F=128, F=(3, 3), S=(1, 1, 1, 1)	(B, 32, 4, 128)
MaxPooling/4	F=(1, 2), S=(1, 1, 2, 1)	(B, 32, 2, 128)
Convolutional + BN/5	N_F=256, F=(3, 3), S=(1, 1, 1, 1)	(B, 32, 2, 256)
MaxPooling/5	F=(1, 2), S=(1, 1, 2, 1)	(B, 32, 1, 256)

Table 5.1. Sizes of the convolutional blocks leveraged by the convolutional module. B refers to the batch size, N_F to the number of learnable filters used, F indicates the filter spatial sizes, and S the stride values adopted.

5.2 Recurrent module

As mentioned, the output of the convolutional module can be seen as a temporal sequence spread over 32 temporal steps, each containing 256 different features. The RNN leveraged by the recurrent module is a 2 layered Bi-directional Long Short Therm Memory. Indeed, as mentioned in section 2.2, *vanilla* RNNs often fail to propagate the information through longer distances. On the other hand the LSTM cell internally leverages *gates* Figure 5.2, the activation of which depends by trainable weights and the inputted sequence's elements: if a particular configuration of the so called forget gate f and input gate i is achieved during training (i.e. if $f = 1, i = 0$), a residual connection (i.e. a short circuit among adjacent RNN's temporal states) is virtually introduced, allowing the uninterrupted forward flow of the information and backward flow of the gradients [13]. It's worth to notice that even if obtaining such configuration is theoretically possible, no guarantee about its achievement can be provided. 2 different layers are leveraged: two LSTM cells are indeed stacked one on top of the other (i.e. the *deeper* LSTM cell accepts as input the sequence of hidden states outputted by the *shallower* one. The output is only provided by the deeper cell). Bi-directionality, on the other hand, refers to the fact that the input sequence is analyzed in both forward (i.e. its elements are fed to the RNN from the first to the last) and backward (i.e. its elements are fed to the RNN from the

last to the first) fashion. The output of the convolutional model is so fed to the RNN. Identifying with r^t the Bi-LSTM's outputted vector $\in \mathbb{R}^{512}$ corresponding to time step t, a point wise convolution is executed in order to map each r^t to a vector $r'^t \in \mathbb{R}^C$, where $C = 80$ recalls the cardinality of the character set (including the *blank* character needed by the CTC loss as explained in section 2.3). Indeed, being the spatial size of the filters adopted to perform such a convolution point-like, each r'^t can be simply thought as a linear projection of the corresponding r^t. The final Softmax application allows for the extraction of probability scores (Figure 5.2). Table 5.2 reports the sizes of the layers used to implement the recurrent module.

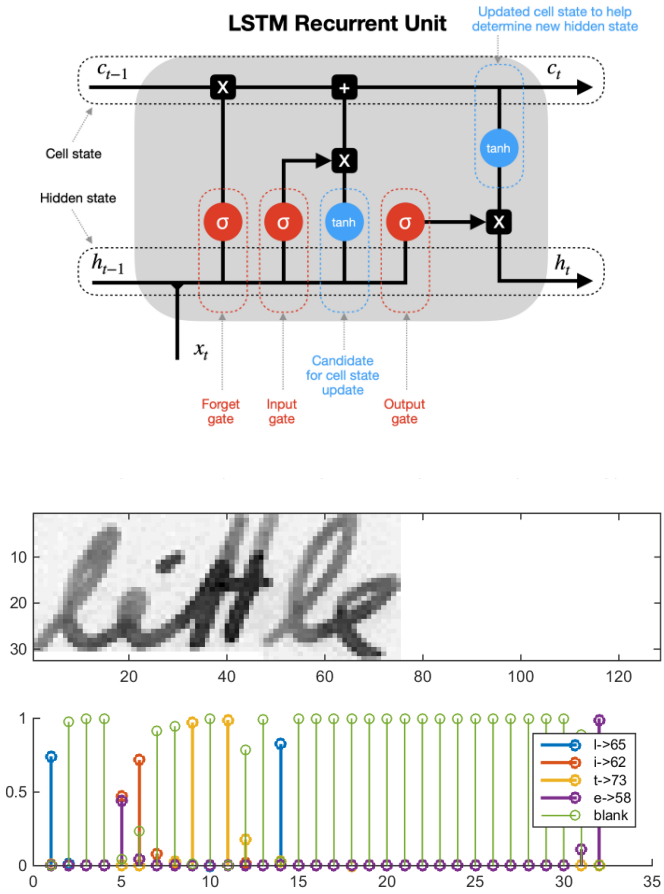

Figure 5.2. Upper: representation of a LSTM cell and its inner gates. In the middle: visualization of a sample input. Lower: visualization of the probability distributions on the character set outputted at each time step. [72]

Layer type/ID	Description	Output size
Input	output of conv. module	(B, 32, 256)
Bi-LSTM	H=256, L=2,	(B, 32, 2 $\cdot H$ =512, 1)
Point-wise convolution	N_F=C, F=(1, 1), S=(1, 1, 1, 1)	(B, 32, C)

Table 5.2. Sizes of the layers leveraged by the recurrent module. C refers to the cardinality of the character set and H to the dimension of the LSTM hidden state.

5.3 Transcription module

The transcription module deals with the conversion of the frame predictions made by the 2 layered Bi-LSTM into highest probability labelings. As seen in section 2.3, 2 different CTC decoding algorithms (i.e. the best-path and the word-beam-search methods) have been introduced. Both of them have been separately adopted to train the corporate OCR system: respectively denoting with **CER** and with **WA** the character error rate (computation of which is described in section 2.2) and the word accuracy, performances of the present corporate OCR model are reported in Table 7.5.

Decoding algorithm	**Train set**	**Val. set**	**CER**	**WA**
Best path	IAM	IAM	6.6 %	82.6%
Best path	IAM	ENEL_{DS}	87.0%	3.7 %
Best path	IAM + ENEL_{DS}	ENEL_{DS}	19.5%	57.2%
Beam search	IAM	IAM	10.9%	26.4%
Beam search	IAM	ENEL_{DS}	79.8%	4.2%
Word beam search	IAM	IAM	4.3%	92.0%

Table 5.3. Performances of the corporate OCR architecture.

Chapter 6

Handwritten text image generation

For the purposes of this thesis work, the architectures proposed by two different papers have been selected among the most relevant ones listed in section 3.2 to generate realistic handwritten text images: the HiGAN (Handwriting Imitation Conditioned on Arbitrary-Length Texts and Disentangled Styles) model, proposed by Gan & Wang [63], and the HWT (Handwriting Transformers), proposed by Bhunia et al. [57]. Indeed, when it comes to synthesizing handwritten text images, several challenges need to be addressed: it is in fact desirable to inject into the generative model the capability to:

- Synthesize variable-sized images (since handwriting images of variable length texts should have different sizes);
- Generate images, given a language alphabet, conditioned on arbitrary texts with no constraint to any corpus;
- Disentangle calligraphic styles from reference images and then imitate the extracted styles for synthetic images;
- Capture the long and short range relationships within the reference style examples;

Despite many of the papers listed in section 3.2 faced some of those challenges, almost none of them succeeded in simultaneously fulfilling all those requirements except for the above mentioned and selected papers. Both of them contain references to their respective repositories, containing the source code used to implement the generative models and the weights of their pre-trained models (HiGAN, HWT). During this chapter, a detailed and technical analysis of both the architectures will be carried out.

6.1 Handwriting imitation GAN - HiGAN

Both the training and the evaluation of the HiGAN architecture has been executed over the IAM benchmark data set. Said $\mathcal{X}$ the set of handwritten images, $\mathcal{Y}$ the set of their text labels and $\mathcal{W}$ the set of the corresponding writer ids, the goal is to train a generator G able to synthesize realistic handwriting images conditioned on the arbitrary text $\boldsymbol{y} = [y_1, \ldots, y_L]$ of length L and any style feature s which can be

either sampled from a prior distribution $\mathcal{N}(0, 1)$ or disentangled from a reference image $x \in \mathcal{X}$ with a pre-trained style encoder E (i.e. $s = E(x)$).

6.1.1 Architecture

The proposed architecture can be sub-divided into five main components, the interaction of which can be visualized in the following Figure 6.2:

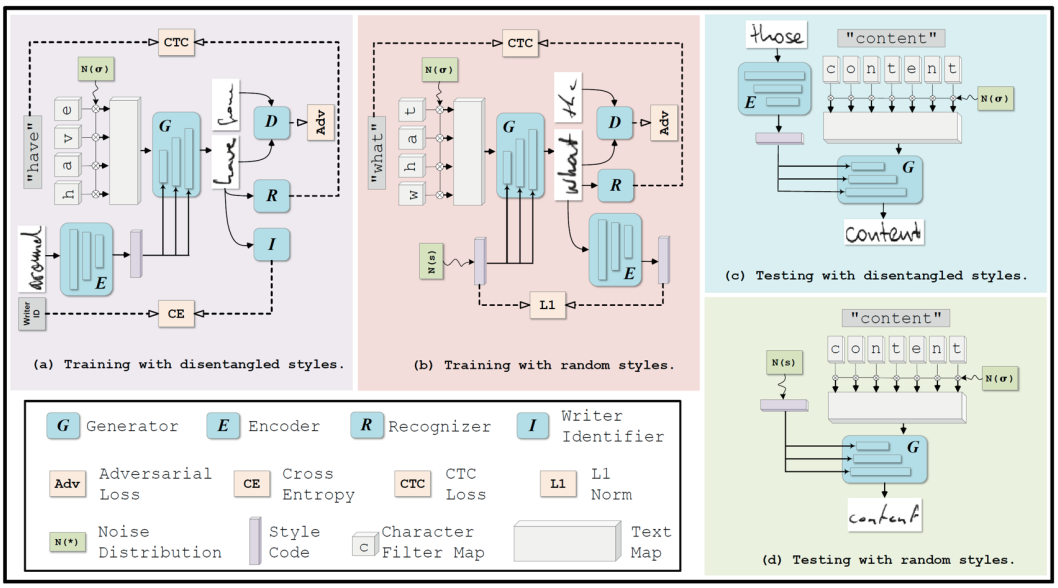

Figure 6.1. Schematization of the HiGAN architecture. While training, the model learns to (a) disentangle calligraphic styles from real images and (b) generate fake images that are indistinguishable from real ones. During test time, on the other hand, the model can either generate realistic images imitating a given style disentangled from a reference sample (c) or generate realistic images conditioned over a randomly sampled style (d).[63]

Generator

The generator G deals with the generation of variable lengths handwritten text images. Given an alphabet $\mathcal{A}$, let $\mathcal{F} = \{f_c | c \in \mathcal{A}\}$ be a set of character filter maps (f_c denotes the filter map associated with character c). Given a textual input $\boldsymbol{y} = [y_1, \ldots, y_L]$, it can be embedded into multiple feature maps as $\mathcal{F}(\boldsymbol{y}, \sigma) = [f_{y_1} \otimes \sigma, \ldots, f_{y_L} \otimes \sigma]$, where σ denotes injected random noise modulating filter maps and inducing stochastic distortions. Finally, the style invariant text-map $\mathcal{M}$ is obtained by concatenating in an horizontal fashion those filter maps. A convolutional network is then used to up-sample the text-map $\mathcal{M}$: instead of vanilla Batch Normalization layers, it leverages Conditional Batch Normalization layers [73]. They provide a natural and efficient way to meaningful inject the style feature s in the G module output: indeed, variations $\Delta\gamma$ and $\Delta\beta$ to the batch normalization's trainable scalars γ and β are outputted by a trainable multi-layer perceptron which is fed with style features s. Due to the spread of the activations fields (which overlap over adjacent letters)

naturally inherited by the convolutional operation, the generator is automatically capable to draw the ligatures needed between adjacent letters.

Discriminator

Exploiting convolutional layers, the discriminator D mainly learns a binary classification problem, i.e. whether an image x is coming from the training set of it is generated by G. Both the generator and the discriminator modules inherit their architecture by the BigGAN model [74], more details of which can be found in the original paper.

Style encoder

The style encoder E faces the disentanglement of hand writing styles s from a reference image x without accessing to the ground truth corresponding writer identity.

Writer identifier

The identifier I is exploited to address the writer classification problem. Indeed, it is desirable to enforce the style encoder E to clusters embeddings for the images of similar styles and to push apart the ones belonging to images of different styles. Therefore, I classifies each image x to the writer which produced the handwritten text. Of course, being the writer identity known only during the analysis of training data points, the I usage (and consequently the above described embedding spatial distribution enforcement) is discarded at test time.

Recognizer

The recognizer R aims at the prediction of a textual label $\boldsymbol{y}$ given in handwritten image sample x: in the training process, R enforces the generator G to produce handwritten images which contains the ground truth textual content $\boldsymbol{y}$. Since it is desired to obtain a generator capable to perform OOV word generation, the implicit injection of a language model into R has to be avoided, and this can be obtained preferring a fully convolutional architecture rather than a recurrent one to implement the recognizer. At test time, G will be able to output a realistic handwritten image with a textual content ideally matching the arbitrary user inputted query textual content.

6.1.2 Training process

The training is executed leveraging the classical min-max GAN game over an objective function, which takes contributions by different terms identified by Gan & Wang [63]:

Adversarial loss

Given an arbitrary text content $\tilde{\boldsymbol{y}} \in \mathcal{C}$ (where $\mathcal{C}$ is an arbitrary open corpus s.t. $\mathcal{Y} \subset \mathcal{C}$) and a style feature s sampled from a prior $\mathcal{N}(0,1)$ G learns to produce a realistic and discriminator's indistinguishable fake image $G(\tilde{\boldsymbol{y}}, s)$ via the minimization of:

$$\mathcal{L}_{\text{adv}_1} = \mathbb{E}_x[\log D(x)] + \mathbb{E}_{\tilde{\boldsymbol{y}},s}[\log(1 - D(G(\tilde{\boldsymbol{y}}, s)))] \tag{6.1}$$

Furthermore, the synthesized image needs to be conditioned to the disentangled style $E(x)$. That can be enforced via the additional minimization of:

$$\mathcal{L}_{\text{adv}_2} = \mathbb{E}_x[\log D(x)] + \mathbb{E}_{\tilde{\boldsymbol{y}},s}[\log(1 - D(G(\tilde{\boldsymbol{y}}, E(x))))] \tag{6.2}$$

The total adversarial loss can so be quantified as:

$$\mathcal{L}_{\text{adv}} = \mathcal{L}_{\text{adv}_1} + \mathcal{L}_{\text{adv}_2} \tag{6.3}$$

Text recognition loss

As mentioned, a key property of the generator is that it should be capable to synthesize images preserving an inputted textual content. Other than the visual similarities enforced by $\mathcal{L}_{\text{adv}}$ and said $\boldsymbol{y}$ the ground truth textual label associated with x, this textual conditioning can be obtained first optimizing R w.r.t. the CTC loss $\mathcal{L}^D_{\text{ctc}}$:

$$\mathcal{L}^D_{\text{ctc}} = \mathbb{E}_{x,\boldsymbol{y}}[-\log(p(\boldsymbol{y}|R(x))] \tag{6.4}$$

Furthermore, R can guide G to generate images containing an arbitrary textual content $\tilde{\boldsymbol{y}}$ if G is also optimized to minimize:

$$\mathcal{L}^G_{\text{ctc}} = \mathbb{E}_{\tilde{\boldsymbol{y}},s}[-\log(p(\tilde{\boldsymbol{y}}|R(G(\tilde{\boldsymbol{y}}, s)))] \tag{6.5}$$

It is worth to highlight that the R's parameters are kept freezed when G is optimized to minimize $\mathcal{L}^G_{\text{ctc}}$.

Style disentangling loss

In order to disentangle calligraphic styles form handwritten images and to enforce the model ability to reconstruct the style of any arbitrary generated image, L_1 norm is exploited to define the latent style reconstruction loss:

$$\mathcal{L}_{\text{style}} = \mathbb{E}_{\tilde{\boldsymbol{y}},s}[\|s - E(G(\tilde{\boldsymbol{y}}, s)\|_1] \tag{6.6}$$

Denoting with $\{x, w\}$ the ground truth image-writer id pairs (recall that $w \in \mathcal{W}$) and in order to enforce the identifier I capability to distinguish writer's identities, I is trained to minimize the classical cross entropy:

$$\mathcal{L}^D_{\text{id}} = \mathbb{E}_{x,w}[-\log I(x)_w] \tag{6.7}$$

Where the subscript w denotes that only the probability score assigned by I to the writer identity class corresponding to the ground truth writer identity contributes to the loss term. Furthermore, to enforce the G capability to generate images with a similar style compared to the reference image x, G and E are also optimized to minimize the following additional loss term:

$$\mathcal{L}^G_{\text{id}} = \mathbb{E}_{x,w,\tilde{\boldsymbol{y}}}[-\log I(G(\tilde{\boldsymbol{y}}, E(x)))_w] \tag{6.8}$$

I's parameters are kept freezed when G and E are optimized to minimize $\mathcal{L}^G_{\text{id}}$. Finally the encoded latent space is regularized via the Kullback-Leibler, pushing its distribution towards the Gaussian prior:

$$\mathcal{L}_{D_{KL}} = \mathbb{E}_x[D_{KL}(E(x)\|\mathcal{N}(0, 1))] \tag{6.9}$$

Full objective

The loss contributions identified by Gan & Wang can be summarized as follows: the discriminator D, the recognizer R and the identifier I are respectively optimized w.r.t.:

$$\mathcal{L}_D = -\mathcal{L}_{\text{adv}} \qquad \mathcal{L}_R = \mathcal{L}_{\text{ctc}}^D \qquad \mathcal{L}_I = \mathcal{L}_{\text{id}}^D \tag{6.10}$$

Clearly in this stage both the generator G and style encoder E parameters are kept fixed once optimizing D w.r.t. $\mathcal{L}_D$. On the other hand, G and E are jointly optimized w.r.t:

$$\mathcal{L}_{G,E} = \mathcal{L}_{\text{adv}} + \lambda_{\text{ctc}}\mathcal{L}_{\text{ctc}}^G + \lambda_{\text{id}}\mathcal{L}_{\text{id}}^G + \lambda_{\text{style}}\mathcal{L}_{\text{style}} + \lambda_{D_{KL}}\mathcal{L}_{D_{KL}} \tag{6.11}$$

All the modules are trained from scratch in an end-to-end fashion.

6.1.3 Performances

The performance of the HiGAN can be summarized by the FID value, computed in an experimental setup similar to the one depicted in [75]. The basic idea is to compute the average FID score over 25k of real-generated handwritten image pairs, where in each pair the textual content and the style is kept fixed. Results and comparison with other architectures reported in Table 6.1.

Model	**FID** $\downarrow$
GANwriting, *Kang et al.* [76]	120.07
ScrabbleGAN, *Fogel et al.* [60]	23.78
HiGAN, *Gan & Wang* [63]	17.28

Table 6.1. FID comparison across different handwritten text generation architectures. [63]

6.2 Handwriting Transformers - HWT

Both the training and the evaluation of the Handwriting Transformer architecture has been solely executed over the IAM benchmark data set. Let's denote a particular writer with $i \in \mathcal{W}$, where $\mathcal{W}$ identify the set including a total of M writers. A set $\boldsymbol{X}_i^s = \{\boldsymbol{x}_{ij}\}_{j=1}^P$ of P handwritten word images is given for each writer: s denotes the use of the set as a source of handwritten style, which is transferred to the target set $\tilde{\boldsymbol{X}}_i^t$ having the same handwritten style but different textual content. The textual content is denoted as the set $\mathcal{A} = \{\boldsymbol{a}_j\}_{j=1}^Q$, where each $\boldsymbol{a}_j$ comprise an arbitrary number of characters from the character set $\mathcal{C}$.

6.2.1 Architecture

The model is composed by:

- The conditional generator G_θ, in charge of synthesizing handwritten images;
- The discriminator D_ψ, which enforce the generation of realistic images;
- The recognizer R_ϕ, which helps in preserving inputted textual content, fully inspired to the recurrent CNN implemented in [71];
- The style classifier S_η, which pushes the correct transfer of calligraphic style;

Where the different subscripts denote the set of trainable weights which parameterize each of the modules.

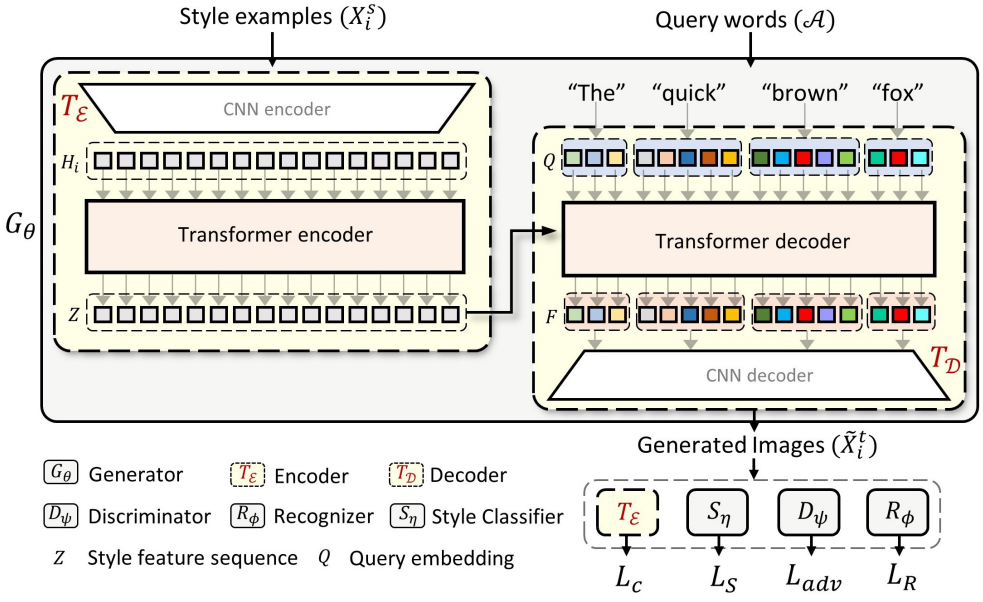

Figure 6.2. Schematization of the HWT generator architecture, built with a encoder/decoder paradigm. Both the modules combine the strengths of CNN and transformer based models, i.e. the capability to model complex relationship leveraging a limited amount of handwritten training images.[57]

Generator

The G_θ generator module is made up of two Convolutional/MultiHeadAttention hybrid networks: an encoder network $T_\epsilon : \boldsymbol{X}_i^s \mapsto \boldsymbol{Z}$ (which produces a sequence of feature embeddings $\boldsymbol{Z} \in \mathbb{R}^{N \times d}$ given the i_{th} set of style examples $\boldsymbol{X}_i^s$) and a decoder network $T_D : (\boldsymbol{Z}, \mathcal{A}) \mapsto \tilde{\boldsymbol{X}}_i^t$ (which transforms the input word strings $\boldsymbol{a}_j \in \mathcal{A}$ into a set of realistic handwritten images $\tilde{\boldsymbol{X}}_i^t$ with a style consistent with $\boldsymbol{X}_i^s$.
The **encoder** T_ϵ leverages a CNN ResNet18 [77] based backbone which generates lower spatial resolution activation maps $\boldsymbol{h}_{ij} \in \mathbb{R}^{h \times w \times d}$ for each style image $\boldsymbol{x}_{ij}$. Those activation's maps spatial dimension is then flattened so to obtain $\boldsymbol{h}'_{ij} \in \mathbb{R}^{n \times d}$, where $n = h \cdot w$. Concatenating the flattened activation maps computed over all the P examples of the i_{th} writer (i.e. $\boldsymbol{h}'_{ij} \forall j \in \{1, \dots, P\}$) the single tensor $\boldsymbol{H}_i \in \mathbb{R}^{N \times d}$ is obtained, where $N = P \cdot n = P \cdot h \cdot w$. A MHA attention layer leveraging J attention heads is then recursively applied L times over $\boldsymbol{H}_i$ in order to obtain $\boldsymbol{Z} \in \mathbb{R}^{N \times d}$, a representation which highly expresses the local and global dependencies in $\boldsymbol{H}_i$ naturally modeled by the MHA layers.
The **decoder** T_D firstly leverages the standard transformer's decoder module: the key and value vectors needed by the MHA layers are obtained by the means of trainable linear projections on top of the encoder output $\boldsymbol{Z}$, while a query embedding $\boldsymbol{Q}_{\boldsymbol{a}j} = \{\boldsymbol{q}_{c_k}\}_{k=1}^{m_j}$ is computed for each input word $\boldsymbol{a}_j$ of length m_j. Recursively iterating the information flow through multiple decoder layers, the final representation $\boldsymbol{F}_{\boldsymbol{a}j} \in \mathbb{R}^{m_j \times d}$, both style and text content aware, is obtained. Random noise sampled from $\mathcal{N}(0, 1)$ is added in order to take into account the natural variability of writing. After a further linear transformation is applied to $\boldsymbol{F}_{\boldsymbol{a}j}$, a CNN decoder is used to obtain the final text and style conditioned handwritten images.

6.2.2 Training process

The training pipeline follows the traditional GAN paradigm, where the discriminator D_ψ tells apart the samples generated from the generator G_θ from the real ones.

Adversarial hinge loss

The $\mathcal{L}_{\text{adv}}$ adversarial hinge loss enforces G_θ to produce realistic images, even though it doesn't take into account any textual or stylish conditioning. It is defined as:

$$\mathcal{L}_{\text{adv}} = \mathbb{E}[\max(1 - D_\psi(\boldsymbol{X}_i^s), 0)] + \mathbb{E}[\max(1 + D_\psi(G_\theta(\boldsymbol{X}_i^s, \mathcal{A})), 0)] \tag{6.12}$$

Recognizer loss

In order to preserve the textual content in the generated images, a recognizer loss is introduced so to push G_θ to produce synthetic images which preserve the inputted textual content. The CTC loss is adopted defined as:

$$\mathcal{L}_R = \mathbb{E}_{\boldsymbol{x} \sim \{\boldsymbol{X}_i^s, \tilde{\boldsymbol{X}}_i^t\}, y}[-\log(p(y|R_\phi(\boldsymbol{x})))] \tag{6.13}$$

Where y denotes the ground truth textual content of $\boldsymbol{x} \sim \{\boldsymbol{X}_i^s, \tilde{\boldsymbol{X}}_i^t\}$. It is worth to notice that R_ϕ is trained only once a real image $\boldsymbol{x} \in \boldsymbol{X}_i^s$ is sampled.

Style classifier loss

A style classifier network S_η is used to enforce the G_θ's synthesizing of images conditioned to the right handwriting style: leveraging the classical cross-entropy and recalling that S_η is trained only if a real image is sampled, the additional style classifier loss is defined as:

$$\mathcal{L}_S = \mathbb{E}_{\boldsymbol{x}\sim\{\boldsymbol{X}_i^s,\tilde{\boldsymbol{X}}_i^t\},y}[-\log(S_\eta(\boldsymbol{x}))_y] \tag{6.14}$$

Where the subscript y denotes that only the probability score assigned by S_η to the writer identity class corresponding to the ground truth writer identity contributes to the loss term.

Style consistency loss

Since as said $\boldsymbol{Z}$ solely needs to express the style content of a given set of sample images, it is expected by the generator encoder outputs $T_\epsilon(\boldsymbol{X}_i^s)$ and $T_\epsilon(\tilde{\boldsymbol{X}}_i^t)$ to be similar. In order to enforce the extraction of the sole style representation $\boldsymbol{Z}$, a further style consistency loss is adopted. Leveraging the L_1 distance, it is defined as:

$$\mathcal{L}_{S.C.} = \mathbb{E}[\|T_\epsilon(\boldsymbol{X}_i^s) - T_\epsilon(\tilde{\boldsymbol{X}}_i^t)\|_1] \tag{6.15}$$

Furthermore, Bhunia et al. [57] realized how balancing the gradients computed w.r.t. the S_η and R_ϕ network's parameters (i.e. normalizing them in order to obtain the same standard deviation $\sigma_{\nabla_{D,G}}$ of the gradients flowing by the adversarial loss) stabilized the training procedure. The normalization is executed in an online fashion as follows:

$$\nabla S_\eta \leftarrow \left(\frac{\sigma_{\nabla_{D,G}}}{\sigma_{\nabla_S}}\nabla_{S_\eta}\right) \qquad \nabla R_\phi \leftarrow \left(\frac{\sigma_{\nabla_{D,G}}}{\sigma_{\nabla_R}}\nabla_{R_\phi}\right) \tag{6.16}$$

6.2.3 Performances

As for the HiGAN, the performance of the HWT can similarly be summarized by the FID value. Results and comparison with other architectures reported in Table 6.2.

Model	**FID ↓**
ScrabbleGAN, *Fogel et al.* [60]	23.78
Text and style conditioned gan for generation of offline handwriting lines, *Davis et al.* [62]	20.65
HWT, *Bhunia et al.* [57]	19.40

Table 6.2. FID comparison across different handwritten text generation architectures. [57]

Chapter 7

Experiments and results

In this chapter, an earlier description of the training process executed for both of the implemented models is provided, reporting the qualitative evaluation of their performances. Next, both of the models will be used to generate new data sets, exploiting the corpus described in section 4.4. All these generated data sets will be used to re-train the corporate OCR system, comparing their influence on the performances of the above mentioned Handwritten Text Recognition model.

7.1 GANs Training

After the introduction of the collected data points (i.e. the one containing Italian and Spanish accented characters), it was mandatory to enlarge the original alphabets [63] [57] with those extra characters. Indeed, since via the augmentation of the original IAM data set also new style identities have been introduced, the output layers belonging to the style recognizer modules for both the architectures had to be increased in the same appropriate way.
It is also worth to notice that none of the implemented SOTA models leveraged the full IAM data set. Indeed, in each of the original papers [63] [57] it is possible to assess how they removed different portions of the original data set. This is probably the result of a data pre-processing and filtering pipeline which is described for none of the models. Since the reproduction of the results achieved in both approaches is the first step that needs to be addressed for the purposes of this thesis work, the naive decision to enlarge those filtered data sets with our own collected data points has been taken.

7.1.1 Setup

Because of the long training times required by the 2 architectures and the need to carry out these trainings on GPU, all the training processes have been executed leveraging AWS services. In particular, Sagemaker elaboration processes allow to use GPU instances only for the actual time required by the training algorithm. After a cost/efficiency evaluation of the available EC2 instances, the G4dn.2xlarge (equipped with an NVIDIA T4 GPU, an Intel Cascade Lake CPU and 32 GB of RAM) has been selected as the most suitable to the training purposes.

7.1.2 HiGAN

In order to stay consistent with the training executed in [63], the same hyper-parameter and training setup has been adopted. It foresees the usage of the Adam optimizer (with β_s running average parameters $\beta_1 = 0.5$, $\beta_2 = 0.999$) with an original learning rate $\gamma = 2 \times 10^{-4}$. The learning lasts for 70 epochs, after 25 of which a linear learning rate decay starts in order to reach $\gamma = 2 \times 10^{-6}$ at epoch 70 ($\approx$ 20 hours on the G4dn adopted instance). The batch size has been set to 16. The number of trainable parameters and memory usage of each model's sub-module is reported in Table 7.1, while the visualization of the intra-learning architecture's modules losses is provided in Figure 7.1.

Sub-module	**Params**	**Memory**
Generator	11'732 K	44.8 MB
Discriminator	9'227 K	35.2 MB
Style encoder	1'363 K	5.2 MB
Writer identifier	1'392 K	5.3 MB
Text recognizer	1'638 K	6.3 MB
HiGAN	25'082 K	96.8 MB

Table 7.1. Size of involved sub-modules in the HiGAN architecture.

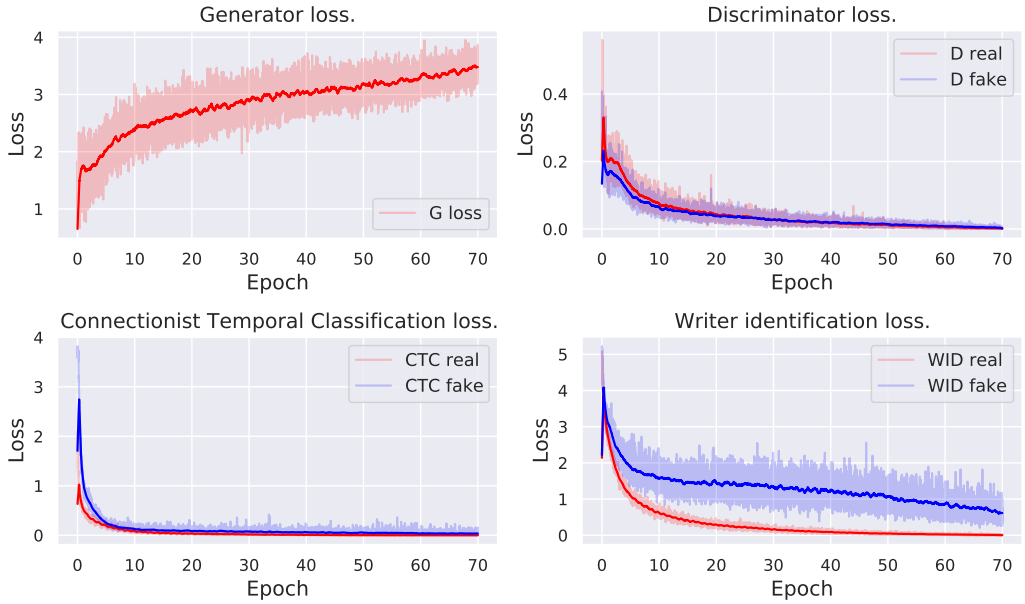

Figure 7.1. Intra-learning monitoring of HiGAN sub-module's losses.

7.1.3 Handwriting Transformers

As in HiGAN training, the tightest consistency with the original training pipelines [57] was desired. The Adam optimizer has been initialized with β_s running average parameters $\beta_1 = 0.4$, $\beta_2 = 0.999$. The learning rate has been set to 5×10^{-5}. No learning rate decay has been leveraged. The original training has been executed for 4'000 epochs, leveraging a batch size equals to 8. Given the performances of the G4dn used instance, such a training has been estimated to last for ≈ 192 hours. Seen the cost of such a computational provisioning, increasing the batch size to 16 is been our sole modification to the original hyper-parameters setup, which led to a training pipeline lasting ≈ 110 hours. The number of trainable parameters and memory usage of each model's sub-module is reported in Table 7.2, while the visualization of intra-learning architecture's modules losses is provided in Figure 7.2.

Sub-module	**Params**	**Memory**
Generator	40'738 K	155.4 MB
Discriminator	36'383 K	138.8 MB
Writer identifier	36'776 K	140.1 MB
Text recognizer	5'599 K	21.5 MB
Handwriting Transformers	119'497 K	455.8 MB

Table 7.2. Size of involved sub-modules in the Handwriting Transformers architecture.

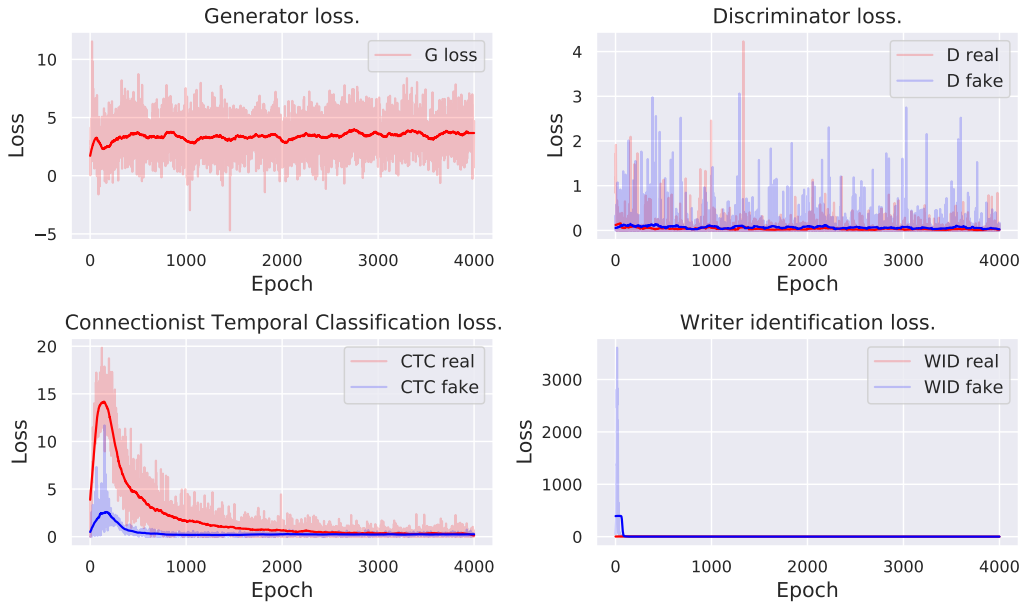

Figure 7.2. Intra-learning monitoring of Handwriting Transformers sub-module's losses.

7.2 GANs performances

The main difficulty faced in the development of the result's analysis for the present thesis was the lack of suitable and reliable metrics to carry out a formal quantitative evaluation. Indeed, even if both of the original papers [63] [57] mainly base their analysis on the evaluation of the FID score (the computation of which is described in subsection 2.3.4), it's worth to recall that such a metric is grounded on the computation of a distance among the embeddings of a generated and a real set of images. Those embeddings are commonly extracted from the deepest layer of an InceptionV3 network trained on the ImageNet database, mainly containing natural images. The bias induced by the nature of ImageNet images lead to an FID evaluation which can't be guaranteed to be reliable nor suitable for handwritten (i.e. non-natural) image's goodness evaluation. Furthermore, while the standard cross-entropy loss commonly turns out to be a good proxy for performance's metrics like the Accuracy, the same can't be said about the adversarial loss. Indeed, even if the minimization of such a loss ideally leads to Generators being able to parameterize the real data probability distribution, such an optimal convergence is hardly achieved during GANs training. Even if several approaches have been implemented to qualitatively evaluate the goodness of the generated images, their definitive quality evaluation will therefore be left to section 7.6, deduced by the performances of the OCR model trained on each generated data set.

7.2.1 HiGAN

The best FID score attained is 10.46. This score depict the quality of the generated images, which turn out to be consistent (if not improved) w.r.t. the one achieved in the original paper [63]. Samples generated with the trained model are reported in Figure 7.3.

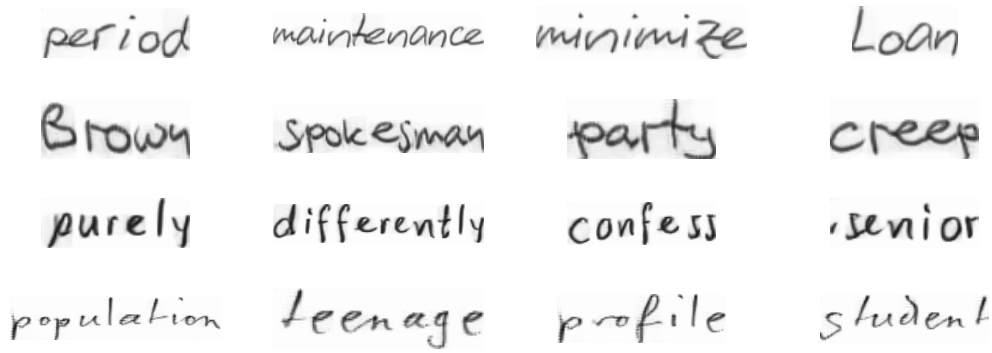

Figure 7.3. Visualization of HiGAN generated samples, produced leveraging 4 different input styles. The style consistency of HiGAN architecture can therefore be visually appreciated.

On the generation of special characters

Despite the fact that the trained HiGAN Generator is capable of generating images for words without special characters which turn out to be visually satisfying from the quality point of view, it still struggles if the inputted text contains special characters. A visualization of samples generated under the input of text containing

those characters is provided in Figure 7.4. It is visually clear that the quality of those generated images is not adequate to the purposes of the HiGAN implementation for the generation of words containing special characters.

Figure 7.4. Visualization of sample images generated with the HiGAN Generator via the input of the ground truth words "oscenità", "fácil", "bebè", "estético", "martedì", "crítico", "però", "presentación", "vudù", "estúpido", "nicaragüense", "dueña".

In order to better understand such a lack of performances, the deepest Discriminator embeddings $\in \mathbb{R}^{512}$ (i.e. the representations obtained at the hidden layer preceding the final binary classification layer) have been extracted for three different sets (namely R, G_s and G_n) of images. Each of this sets respectively contains 200 random real images, 200 generated images containing special characters and 200 generated images without special characters. A t-SNE 2D representation of those deep representations is given in Figure 7.5.

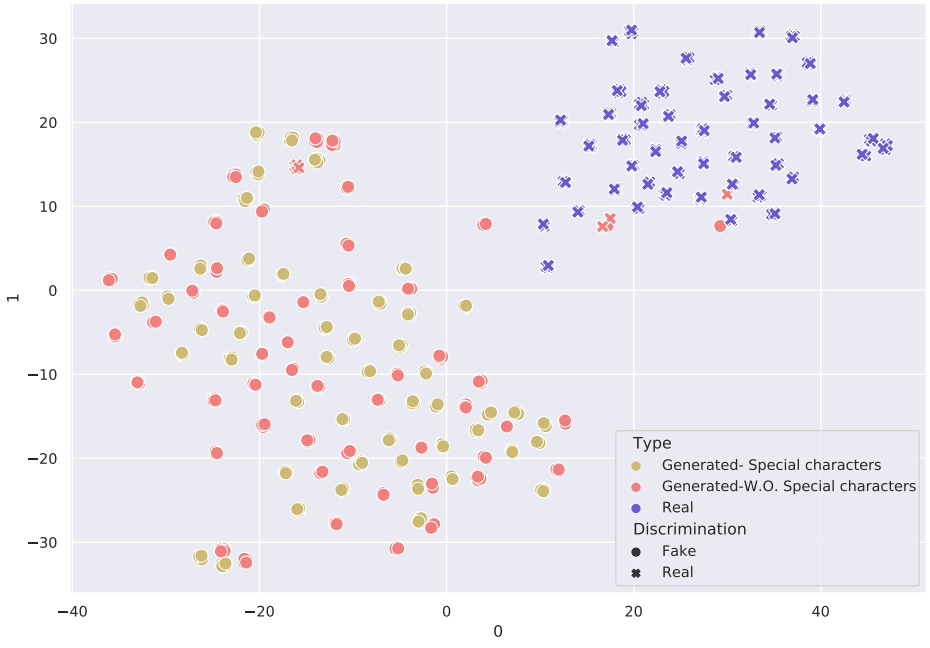

Figure 7.5. t-SNE 2D representation of the Discriminator embeddings for the images belonging to the 3 different sets R, G_n and G_s.

Such a visualization is consistent with both the Generator and Discriminator losses depicted in Figure 7.1. Indeed, the monotonic increase of the Generator loss shows that the generator can't keep up with the increasing capability of the Discriminator to distinguish between real and generated images. If just few of the samples generated without special characters are able to trick the Discriminator, all the images generated with special characters fail in that task. In the same fashion as the FID score is calculated, the distance between R and G_s embeddings (i.e. $d(R^E, G_s^E)$) and the distance between R and G_n embeddings (i.e. $d(R^E, G_n^E)$) have been computed in the original Discriminator embedding space (i.e. $\mathbb{R}^{512}$):

$$d(R^E, G_s^E) = 156.68 \qquad\qquad d(R^E, G_n^E) = 152.62$$

Those results confirm that the Discriminator representations for the generated images $\in G_n$ are slightly closer to the real images $\in R$ w.r.t. the generated images $\in G_s$, highlighting that the generated images $\in G_n$ appear to be more realistic to the Discriminator w.r.t. the generated images $\in G_s$.

7.2.2 Handwriting Transformers

The best FID score attained is 15.52, which again depict the quality of the generated images and the consistency w.r.t. the results obtained in the original paper [57]. Samples generated with the trained model are reported in Figure 7.6.

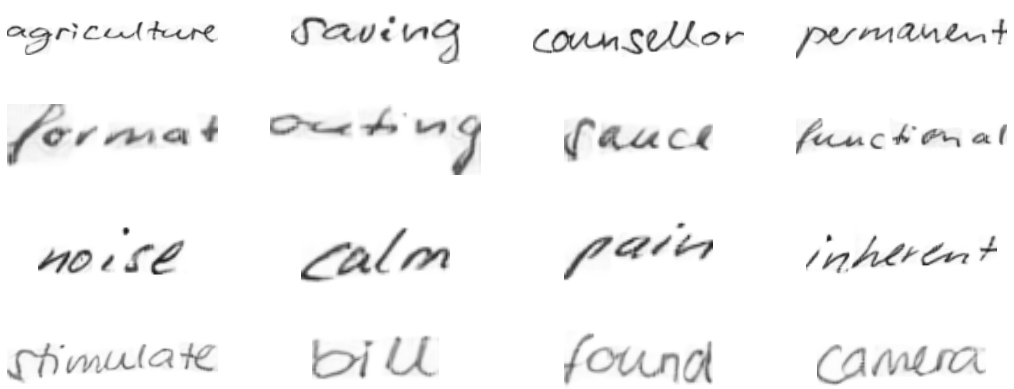

Figure 7.6. Visualization of HWT generated samples, depicting the style consistency attained by the Handwriting Transformers architecture.

On the generation of special characters

Similarly to the HiGAN model, it turns out that the Handwriting Transformers Generator also struggles in generating text containing special characters. A visualization of those generated samples is provided in Figure 7.7. Also in this case, it is visually clear that the quality of those generated images do not even satisfy visual realism requirements.

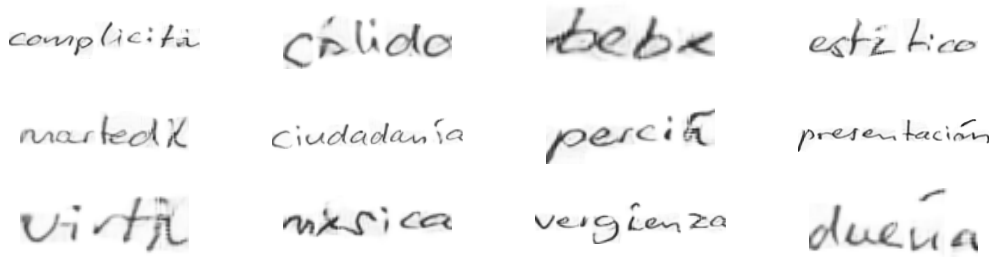

Figure 7.7. Visualization of sample images generated with the Handwriting Transformers Generator via the input of the ground truth words "complicità", "cálido", "bebè", "estético", "martedì", "ciudadanía", "perciò", "presentación", "virtù", "música", "vergüensa", "dueña".

A Discriminator deepest embeddings $\in \mathbb{R}^{1024}$ analysis have also been carried out. Similarly in what have been done for the HiGAN model, those deep representations have been extracted for three different sets of images (namely R, G_s and G_n) built in the same fashion as in section 7.2.1. The t-SNE 2D representation of those embeddings is provided in Figure 7.8.

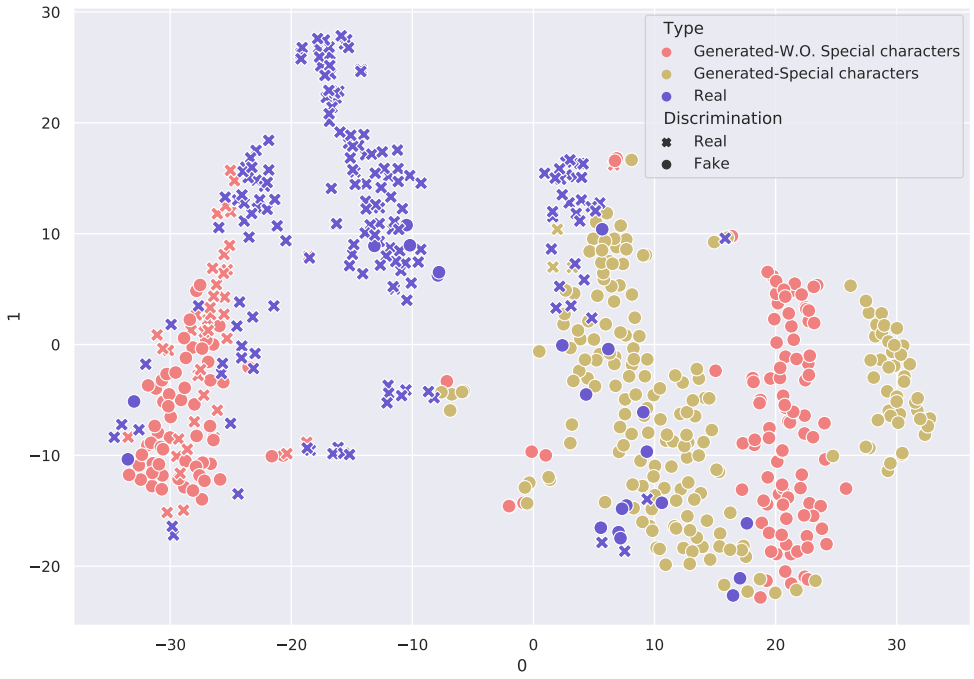

Figure 7.8. t-SNE 2D representation of the Discriminator embeddings for the images belonging to the 3 different sets R, G_n and G_s.

The first noticeable fact is that the Handwriting Transformers Discriminator representations are far less sparse w.r.t. the ones built by the HiGAN Discriminator. Furthermore, the embeddings of the generated images (without special characters) mostly lie in portion of the latent Discriminator space where an high density of real images representations is observed. Also, about 25% of the images $\in G_n$ are discriminated as real images: this evidence actually reflect the capability of the Generator to fool the Discriminator module, as highlighted by both the Generator and the Discriminator losses shown in Figure 7.2. Indeed, the constant trend of such adversarial losses express the fact that the min-max game is equally played by both modules. On the other hand, the greatest portion of the generated images $\in G_s$ are correctly discriminated as fake, highlighting the Generator incapability to produce realistic images containing special characters. As in section 7.2.1, both $d(R^E, G_s^E)$ and $d(R^E, G_n^E)$ have been computed:

$$d(R^E, G_s^E) = 82.98 \qquad\qquad d(R^E, G_n^E) = 69.36$$

Again, those results confirm that the Discriminator representations for the generated images $\in G_n$ are closer to the real images $\in R$ w.r.t. the generated images $\in G_s$, highlighting that the generated images $\in G_n$ appear to be more realistic to the Discriminator w.r.t. the generated images $\in G_s$.

7.2.3 Comparison

A preliminary comparison of the performances of both the HiGAN and HWT Generator modules has been carried out as follows. 100 different words (reported in Table 7.3) have been randomly sampled by the Italian and Spanish corpus, enforcing the sampling of 4 words for each of the 12 special characters. The probability distribution of characters $p_{\text{input}}()$ has so been computed on this set of random words. Each word has then used to generate an handwritten image both with the HiGAN and the HWT Generator exploiting handwritten styles randomly selected upon the ones provided by the IAM data set. Both of the generated set of words have then be manually labeled, and the character probability distributions $p_{\text{HiGAN}}()$ and $p_{\text{HWT}}()$ have been subsequently computed.

estético	presentación	negoziatore	suceder	casualidad
tampoco	fácil	esaurito	complicità	fedeltà
rotundo	paralizar	martedì	exportar	fatto
perciò	hospital	cálido	metrò	privilegio
episodio	nicaragüense	tecnología	ciudadanía	ciondolare
nocturno	comida	paloma	finalidad	horror
schiamazzo	facultad	posa	jurar	lontananza
venerdì	giovedì	vudù	toro	esclusivo
colaborar	piede	humillar	antigüedad	però
disabile	empresarial	intérprete	affinità	dirottare
música	aggiustare	virale	presidir	decente
estúpido	lucertola	biología	crítico	così
aldea	cimitero	amistad	aún	bebè
apparecchiare	bolletta	pierna	común	montañoso
dueña	lassù	instalar	matemática	debajo
poderoso	sparto	ciò	niña	frutto
a	vergüenza	placa	tabù	huésped
alemán	distraer	avión	cheque	bebé
contemplare	escalón	concentrarsi	virtù	difettare
oscenità	tres	manchado	español	profesión

Table 7.3. Sampled words to execute preliminary comparison between the HiGAN and the HWT generators performances.

A visualization of the above mentioned probability distributions is provided in Figure 7.9. Despite the fact that both $p_{\text{HiGAN}}()$ and $p_{\text{HWT}}()$ underline one more time the difficulty faced by the two generators in producing handwritten images for text containing special characters, the probability masses associated with few special characters (e.g. "ó", "í") highlights that the Handwriting Transformer Generator deals better with such a task w.r.t. the HiGAN Generator. Furthermore, the Kullback-Leibler divergence has been computed between $p_{\text{input}}()$ and $p_{\text{HiGAN}}()$ as also between $p_{\text{input}}()$ and $p_{\text{HWT}}()$:

$$D_{KL}(p_{\text{input}} \| p_{\text{HiGAN}}) = 0.0338 \qquad D_{KL}(p_{\text{input}} \| p_{\text{HWT}}) = 0.0227$$

These results show that, under the adoption of the HWT Generator, less information is lost at character level once handwritten images are generated.

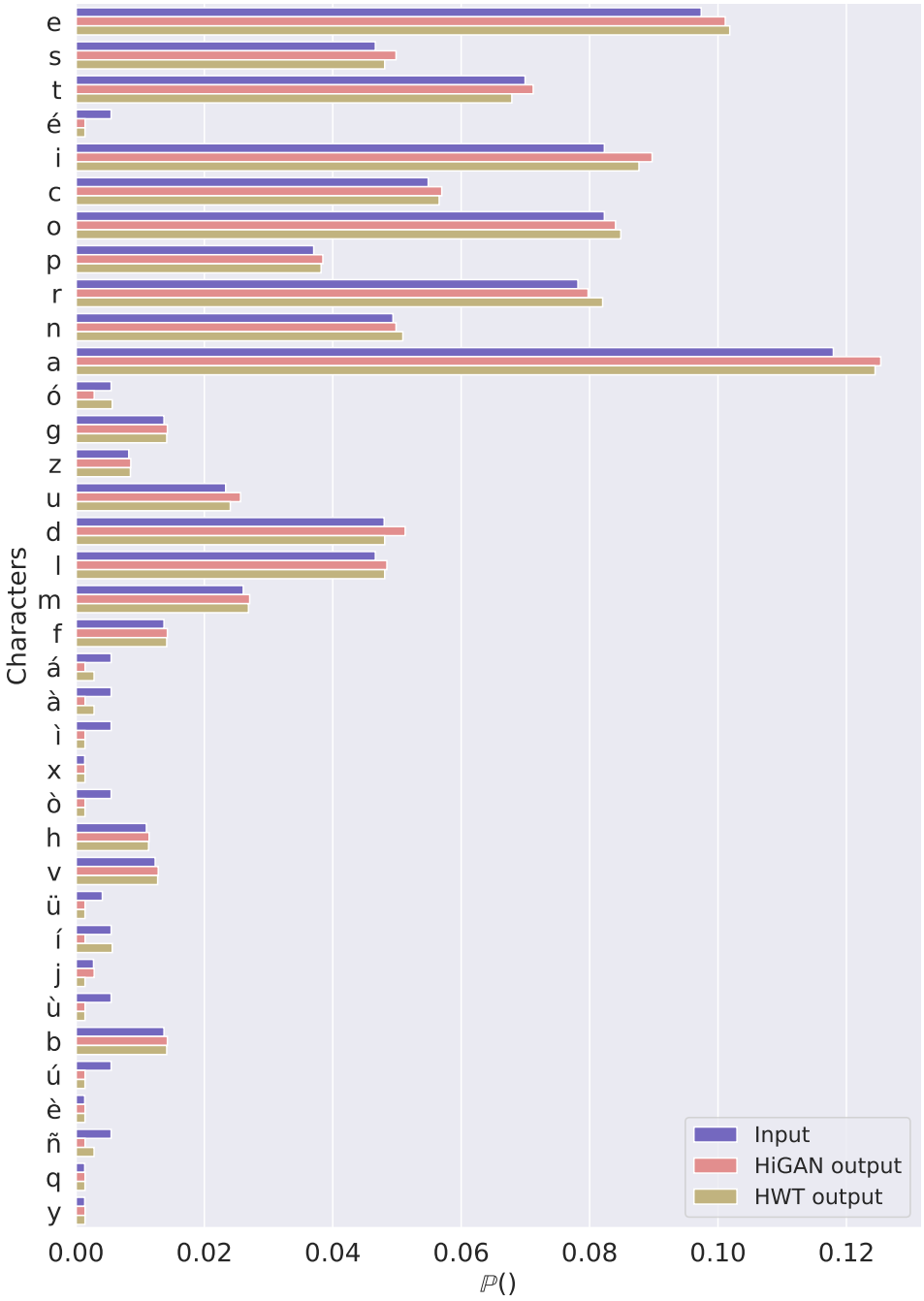

Figure 7.9. Visualization of the character probability distributions $p_{\text{input}}()$, $p_{\text{HiGAN}}()$ and $p_{\text{HWT}}()$.

7.3 Generating new data

In order to generate new data sets (leveraged for the corporate OCR training) and seen the poor results obtained by both the HiGAN and HWT Generators when generating handwritten images containing special characters, it's been decided to omit the generation of words containing those characters. Because of that, among the corpus described in section 4.4, only the "English Common Dictionary capitalized" and the "Italian Common Dictionary without special words" have been adopted as English and Italian corpus.

To generate the English data set, the English corpus has been split in 5 batches, each containing approximately 1000 unique words. 80 distinct writing styles have been assigned to generate all the images contained in each of those batches (i.e. all the words contained in a corpus batch will be generated with 80 unique styles. 4 other disjoint sets each having 80 unique styles will be leveraged to generate the other 4 corpus batches). The corpus is then fed to both the HiGAN and the HWT Generators: since the size of those generated data sets is approximately the 400% of the original IAM size, the two generated data sets will be respectively named $\text{HiGAN}^{400}_{\text{ENG}}$ and $\text{HWT}^{400}_{\text{ENG}}$.

On the other hand, to generate the Italian data set, the Italian corpus is split in 8 batches, each containing approximately 1250 unique words. To each of those batches, 50 distinct writing styles have been assigned for the generation task. Being in this case the size of those generated data sets approximately the 500% of the original IAM size, the two generated datasets will be respectively named $\text{HiGAN}^{500}_{\text{ITA}}$ and $\text{HWT}^{500}_{\text{ITA}}$. The usage of a fully generated data set in a different language allows both to evaluate the suitability of an artificial data set to re-train the OCR model, and to assess the OCR system capability to implicitly learn a language model.

Several percentages of augmentation had to be tested: even if an online augmentation would result into a memory efficient training pipeline (i.e. without the need to store a different data set for each amount of augmentation), the latency induced over the OCR model training by both SOTA model's generators forward passes encouraged the choice of an offline augmentation. Starting from the English generated sets being 400% the size of IAM, a random sampling over words have been executed so to sample the 25%, 12% and the 5% of the generated words for each of the styles. Recalling the meaning of the numeric superscript, these samplings respectively led to the creation of the $\text{HiGAN}^{100}_{\text{ENG}}/\text{HWT}^{100}_{\text{ENG}}$, $\text{HiGAN}^{50}_{\text{ENG}}/\text{HWT}^{50}_{\text{ENG}}$ and $\text{HiGAN}^{20}_{\text{ENG}}/\text{HWT}^{20}_{\text{ENG}}$ data sets. Their features are reported in Table 7.4.

Data set	#Samples	#Train samples	#Val. samples
$\text{HiGAN}^{400}_{\text{ENG}}/\text{HWT}^{400}_{\text{ENG}}$	417'200	396'340	20'860
$\text{HiGAN}^{500}_{\text{ITA}}/\text{HWT}^{500}_{\text{ITA}}$	493'900	469'205	24'695
$\text{HiGAN}^{400}_{\text{ENG}}/\text{HWT}^{400}_{\text{ENG}}$+IAM	527'520	501'144	26'376
$\text{HiGAN}^{20}_{\text{ENG}}/\text{HWT}^{20}_{\text{ENG}}$	19'900	18'905	995
$\text{HiGAN}^{20}_{\text{ENG}}/\text{HWT}^{20}_{\text{ENG}}$ + IAM	130'220	123'709	6'511
$\text{HiGAN}^{50}_{\text{ENG}}/\text{HWT}^{50}_{\text{ENG}}$	49'815	47'324	2'491
$\text{HiGAN}^{50}_{\text{ENG}}/\text{HWT}^{50}_{\text{ENG}}$ + IAM	160'135	152'128	8'007
$\text{HiGAN}^{100}_{\text{ENG}}/\text{HWT}^{100}_{\text{ENG}}$	99'695	94'710	4'985
$\text{HiGAN}^{100}_{\text{ENG}}/\text{HWT}^{100}_{\text{ENG}}$ + IAM	210'015	199'514	10'501
$\text{HiGAN}^{50}_{\text{ENG}}/\text{HWT}^{50}_{\text{ENG}}$+IAM+$\text{ENEL}_{\text{DS}}$	210'518	199'992	10'526

Table 7.4. Features of the different data sets used for the OCR training. 0.95 is the chosen value to perform the train-validation sets split.

7.4 Style consistency analysis

In order to understand whether the style consistency has been preserved by the GAN's Generators during the production of the generated data sets, a visual analysis of the t-SNE projections of the words generated with each handwriting style has been carried out. 4 different synthetic images have been selected for each handwriting style. Given a word of length L, each GANs generator produces synthetic images of shape $H \times W = 32 \times 16 \cdot L$ pixels. Since for the purposes of such analysis all the dependencies by the textual content of those generated images wanted to be removed, only images which depict a word starting with the character "v" are selected. Next, for each image, only the image's slice corresponding to the first character have been used to carry out the rest of the analysis. The 2D t-SNE projection of those image patches is therefore executed both for the $\text{HiGAN}^{400}_{\text{ENG}}$ and the $\text{HWT}^{400}_{\text{ENG}}$ generated set and reported in Figure 7.10 and in Figure 7.11. In both plots, all the 400 handwriting styles are visualized, each of them being identified by a unique triplet of marker color, marker shape, and marker size. The style consistency achieved by the HiGAN generator appears pretty clear: all the 4 image's patches extracted for each handwriting style appear to be well clustered. The HWT generator, on the other hand, struggles to preserve the handwriting features for each style: despite several clusters can be identified, HWT generated handwriting styles appear to be more homogeneous and similar among each other w.r.t. the one obtained with the HiGAN generator. It is worth to underline that the homogeneity among the HWT generated handwriting styles experienced in Figure 7.11 might also be induced by the naive slicing mechanism adopted (i.e. slicing images with a fixed window width of 16 pixels) to select solely the first character in each generated image.

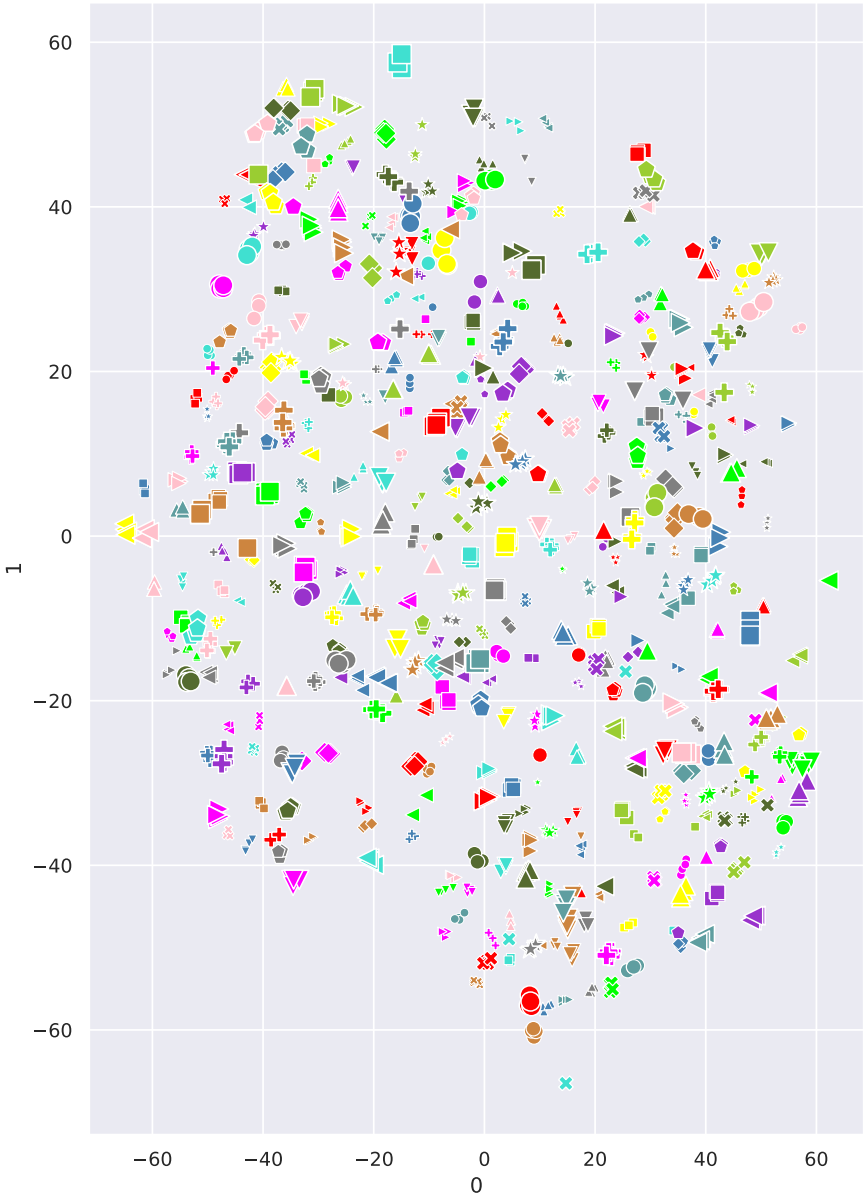

Figure 7.10. 2D t-SNE projection of $\text{HiGAN}^{400}_{\text{ENG}}$ generated image's slices depicting the character "v". Each handwriting style is identified by a unique triplet of marker color, marker shape and marker size.

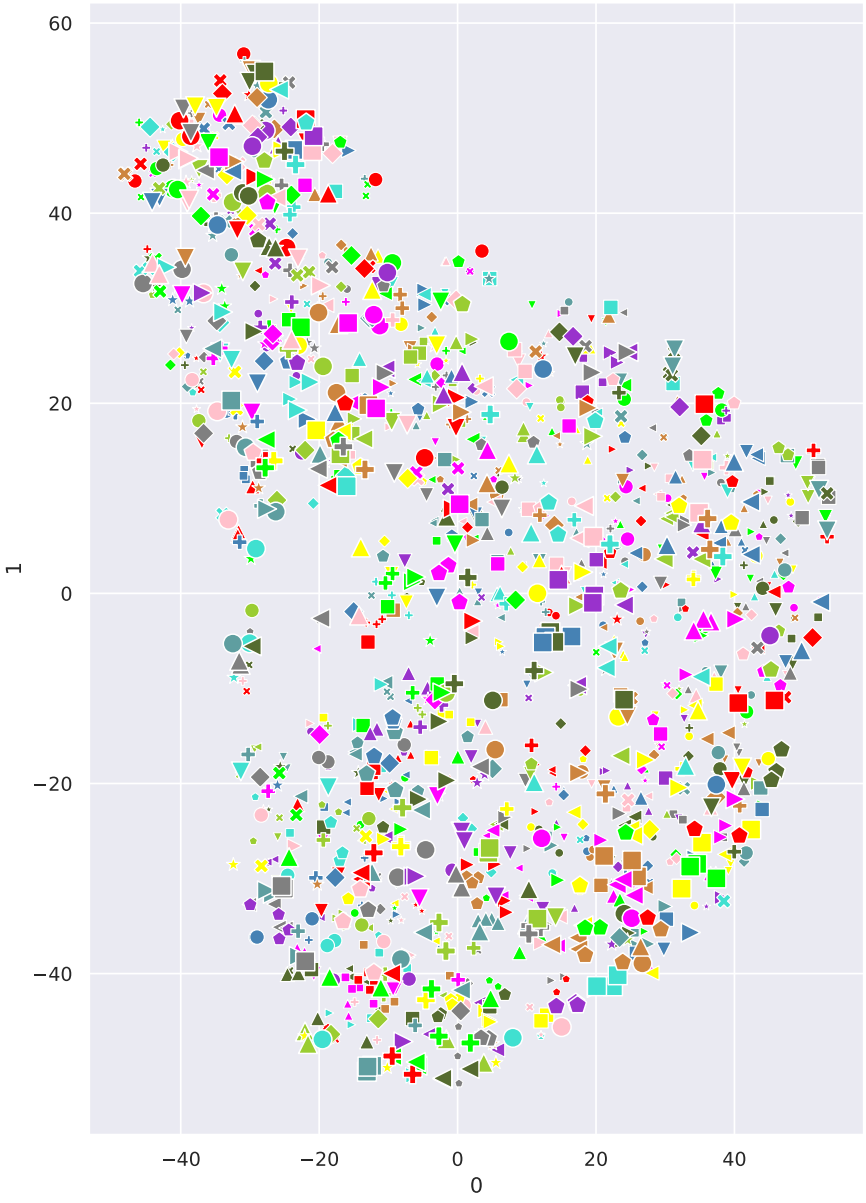

Figure 7.11. 2D t-SNE projection of $\text{HWT}^{400}_{\text{ENG}}$ generated image's slices depicting the character "v". Each handwriting style is identified by a unique triplet of marker color, marker shape and marker size.

7.5 OCR model training

In order to retrain Enel corporate OCR model, the Adam optimizer (with β_s running average parameters $\beta_1 = 0.9$, $\beta_2 = 0.999$) has been selected, initialized with a learning rate $\gamma = 1 \times 10^{-3}$. The learning is executed for 50 epochs and leverages an early stopping which monitor performances over the validation set with a patience value equals to 10 epochs. The batch size is set to 500. Beyond the naive augmentation induced by the inclusion of generated data points, other common online image augmentation techniques are leveraged. Every training image can indeed be blurred with gaussian filters, eroded (i.e. the pen stroke gets thinned), dilated (i.e. the pen stroke gets thickened), darkened, noised with standard gaussian noise and chiaroscuro inverted. Each data augmentation mechanism is independently applied with probability:

- $\mathbb{P}$(Blurring)=0.25
- $\mathbb{P}$(Erosion)=0.25
- $\mathbb{P}$(Dilation)=0.25
- $\mathbb{P}$(Darkening)=0.25
- $\mathbb{P}$(Noising)=0.25
- $\mathbb{P}$(Chiaroscuro inversion)=0.1

Visualization of the above mentioned standard augmentations is provided in Figure 7.12. As mentioned in chapter 5, the corporate OCR model's output allows to adopt the Connectionist Temporal Classification loss. Since the main OCR system re-training purpose is to understand whether the generated data sets positively affect its performance, the sole and simple "Best Path" decoding algorithm have used. CTC loss values observed during the OCR model training are reported in figure Figure 7.13. A visualization of the validation character error rate and word accuracy trends is provided in Figure 7.14.

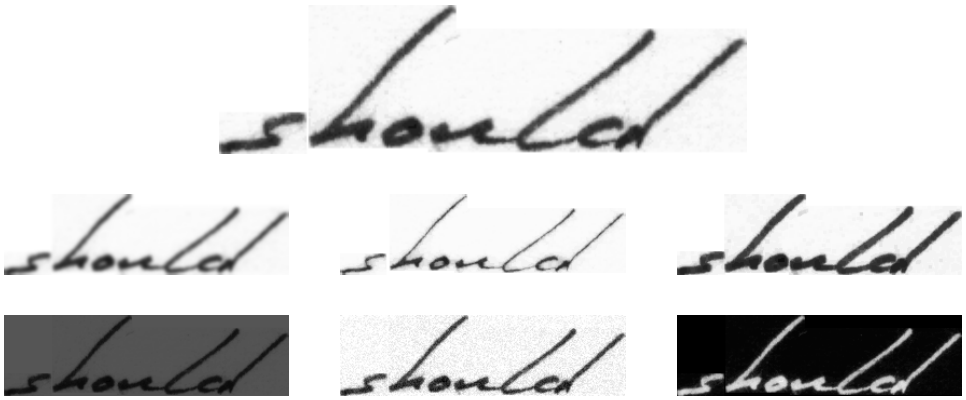

Figure 7.12. Standard augmentations leveraged for the OCR training. At the top: visualization of the original images. Below, from left to right, visualization of the augmented image after: blurring, erosion, dilation, darkening, noising, chiaroscuro inversion.

7.6 OCR model performances

Looking at Table 7.5, it is straightforward to assess how the introduction of the handwritten generated data sets in the OCR system training pipeline on average degrades the OCR model performances when inferring over the IAM and the ENEL_{DS} test sets. It is also worth to notice that training the OCR model on fully generated sets (i.e. the experiments executed over the $\text{HWT}^{500}_{\text{ITA}}$ and the $\text{HiGAN}^{500}_{\text{ITA}}$ sets) result in a catastrophic drop of performance when inferring over both the IAM and the ENEL_{DS} test sets. Indeed, according to Figure 7.13, the OCR system training over both of the generated sets results in the steepest observed decrease of the CTC loss and in the steepest increase of the validation character error rate. A meaningful interpretation is that generated data points contain similarities easily distinguishable by the OCR model, and such a data bias results in a catastrophic degrading of the model generalization capabilities. Despite the model trained over the $\text{HWT}^{20}_{\text{ENG}} + \text{IAM}$ set achieves the best character error rate on the ENEL_{DS} test set w.r.t. all the other models which don't experience the ENEL_{DS} at training time, such a 1% relative improvement can't be considered significant and might simply be due to the randomness of the train-test split process.

Training set Without ENEL_{DS}	**CER** ↓ [IAM | ENEL_{DS}]	**WA** ↑ [IAM | ENEL_{DS}]
$\text{HWT}^{500}_{\text{ITA}}$	54.7 % | 98.7 %	11.8 % | 0.4 %
$\text{HiGAN}^{500}_{\text{ITA}}$	49.7 % | 98.4 %	14.2 % | 1.1 %
$\text{HWT}^{400}_{\text{ENG}}$+IAM	8.1 % | 91.8 %	79.7 % | 2.5 %
$\text{HiGAN}^{400}_{\text{ENG}}$+IAM	7.4 % | 90.8 %	81.3 % | 3.5 %
$\text{HWT}^{20}_{\text{ENG}} + \text{IAM}$	7.7 % | 86.4 %	80.1 % | 2.8 %
$\text{HiGAN}^{20}_{\text{ENG}} + \text{IAM}$	7.7 % | 91.4 %	80.1 % | 3.3 %
$\text{HWT}^{50}_{\text{ENG}} + \text{IAM}$	7.6 % | **86.0** %	80.5 % | **3.7** %
$\text{HiGAN}^{50}_{\text{ENG}} + \text{IAM}$	7.7 % | 93.0 %	80.1 % | 3.2 %
$\text{HWT}^{100}_{\text{ENG}} + \text{IAM}$	7.2 % | 93.3 %	81.4 % | 2.9 %
$\text{HiGAN}^{100}_{\text{ENG}} + \text{IAM}$	7.5 % | 93.9 %	80.8 % | 3.0 %
IAM	**6.6** % | 87.0 %	**82.6** % | **3.7** %

Training set With ENEL_{DS}	**CER** ↓ [IAM | ENEL_{DS}]	**WA** ↑ [IAM | ENEL_{DS}]
$\text{HWT}^{50}_{\text{ENG}} + \text{IAM} + \text{ENEL}_{\text{DS}}$	7.8 % | 19.7 %	79.7 % | 55.9 %
$\text{HiGAN}^{50}_{\text{ENG}}$+IAM+$\text{ENEL}_{\text{DS}}$	8.0 % | 20.5 %	79.5 % | 54.7 %
$\text{IAM} + \text{ENEL}_{\text{DS}}$	**7.0** % | **19.5** %	**80.1** % | **57.2** %

Table 7.5. Performances of the corporate OCR architecture retrained over augmented data sets. Each entry report the performances (i.e. the character error rate **CER** and the word accuracy **WA**) over IAM test set / ENEL_{DS} test set.

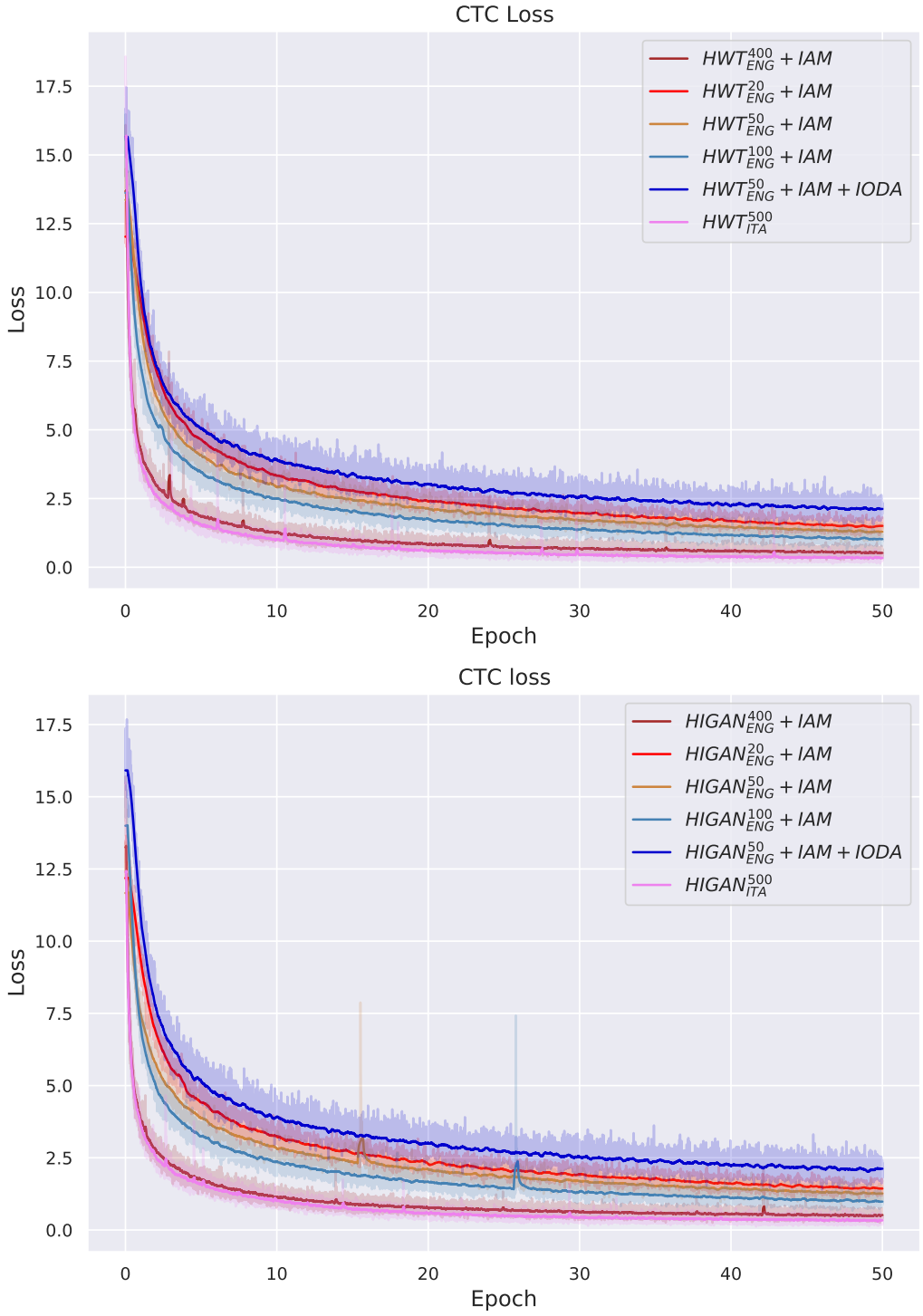

Figure 7.13. Visualization of the training CTC loss trend. At the top: training executed over HWT related data sets. At the bottom: training executed over HiGAN related data sets.

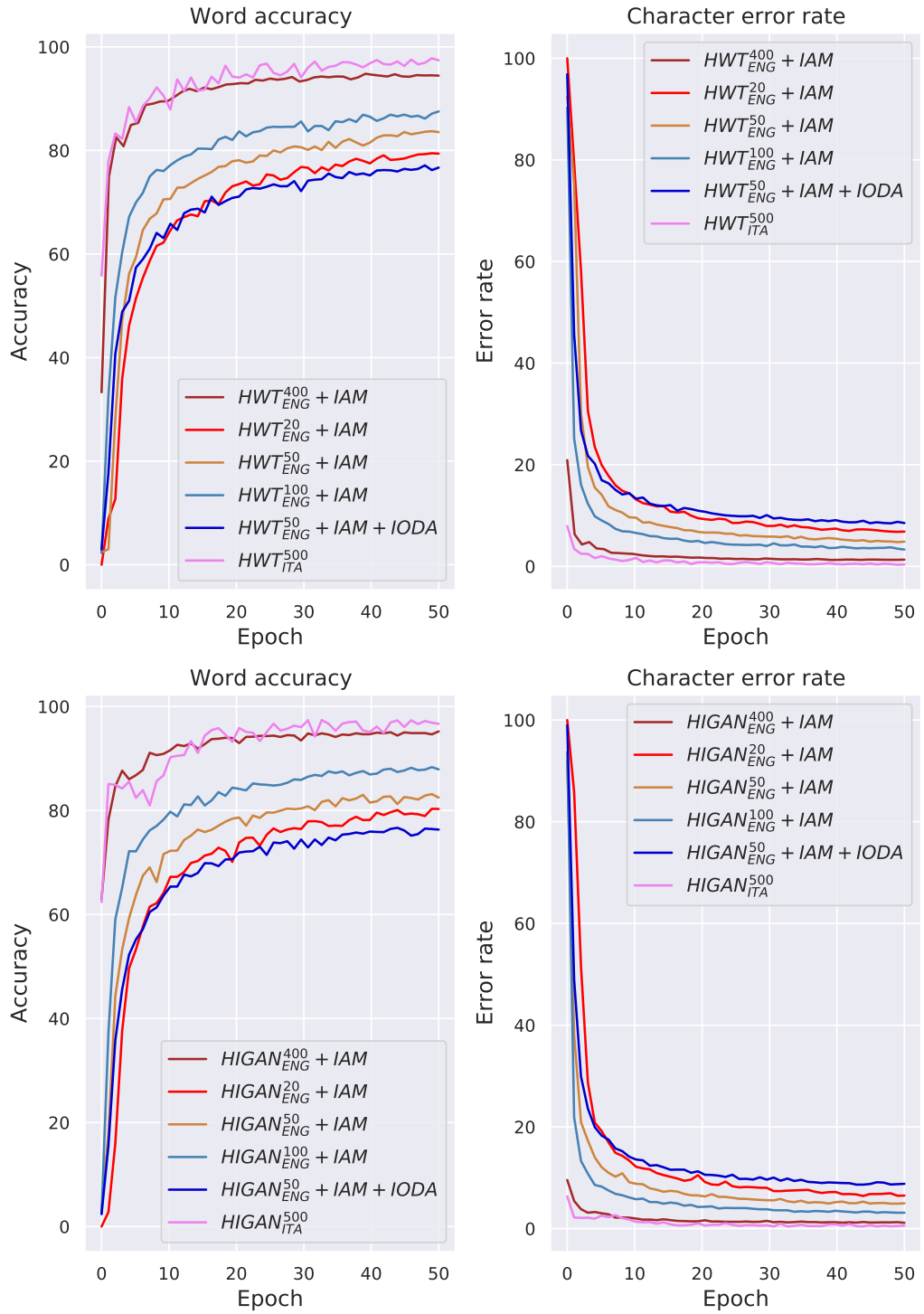

Figure 7.14. Visualization of the validation character error rate and word accuracy trends. At the top: training executed over HWT related data sets. At the bottom: training executed over HiGAN related data sets.

Chapter 8

Conclusions and future works

Accordingly to the ENEL proposed internship, the present thesis was devoted to the implementation of two different GANs able to generate handwritten text images, so to assess whether the ENEL corporate OCR model training could benefit by a GAN based data augmentation. It started with a literature research among state-of-the-art models for handwritten text images generation. Because of the competitive performances reported in their presentation papers, the Handwriting imitation GAN (HiGAN) and the Handwriting-Transformers (HWT) models have been chosen. Both of the models were originally trained on the IAM database, which solely contains characters belonging to the English alphabet. Because of that, a collection of handwritten text images containing special characters (i.e. à, è, é, ì, ò, ù, á, í, ó, ú, ü, ñ) have been carried out among ENEL and Data Science colleagues. Augmenting the IAM data set with the collected images, the aim was to re-train both the HiGAN and the HWT, ideally obtaining generative models able to produce handwritten text images in English, Italian or Spanish. Once trained, both the models have been exploited to produce synthetic data sets of various sizes. Despite the lack of suitable metrics to quantitatively evaluate the quality of generated data points, a detailed qualitative analysis of the synthetic images goodness have been carried out. Generated data sets have then been exploited to augment the IAM database and to retrain the ENEL corporate OCR model. Several experiments have been conducted to assess whether the OCR system training pipeline can benefit by the proposed GAN based augmentation.

The first point that needs to be addressed is the incapability of the implemented GANs to generate handwritten images depicting words containing special characters. Indeed, as clear from Figure 7.4 and Figure 7.7, both of the architectures solely got a vague knowledge about special character's generation. Reasonably it is due to the great character class imbalance observed in the IAM set augmented with the special characters data collected only from only 48 colleagues. Still, the results obtained look promising and worth to be elaborated towards the implementation of multi language handwritten text images generative models. The results presented in Table 7.5 highlights that the tested GANs based augmentation is neither able to increase the ENEL corporate OCR model robustness nor its generalization capability. Despite the positive outcome of the qualitative evaluation led over the generated images and the consistency of the implemented GANs performances with the one obtained in the reference papers [63] [57], the missed improvement of the OCR system character error rate and word accuracy over the IAM test set indicate that the quality of synthetic data points might simply not be sufficient to consider them as valuable OCR training samples. Furthermore, patterns inherited by the generative model structure might be shared across images generated with the same textual

input but with different handwriting styles. This assumption is also consistent with the CTC loss trend shown in Figure 7.13 and with the validation metrics presented in Figure 7.14: fully generated data sets seems to be learnt by the OCR model with extreme easiness. This suggests that the OCR system is able to exploit these similarities so to naively memorize the textual content of synthetic data points. The presence of those patterns would clearly result to a poor exploration of the data space during training and also to the introduction of a data bias. Both of them appear to be meaningful causes of the OCR model performance's degradation upon the insertion of generated data points in its training set. The lack of OCR system performance improvement once the inference over the ENEL_{DS} is executed can be imputed to the concept drift. Even if the exploited decoding algorithm "Best Path" prevents the OCR model from explicitly using a language model, such a concept drift is surely reflected both at style level (i.e. the handwriting styles observed in the ENEL_{DS} differs from the ones observed in the IAM database) and at noise level. Indeed, all the images belonging to the ENEL_{DS} turned out to be way noisier than the one observed in IAM due to the presence of the underlying customer form structures (e.g. dots and dashes on top of which handwriting is performed by ENEL customers). If on one hand the results highlight that generating images with GANs solely trained on IAM is not the most efficient way to counteract the above mentioned effects due to the test domain shift, on the other hand they suggest that broadening the domain of the GANs training sets might be the key to address this issue.

Addressing the HiGAN and the HWT incapability to produce handwritten images depicting words containing special characters, the introduction of more labeled special character handwritten text images in the GANs training sets might lead to multi-language generative models effectively being able to produce handwritten images containing arbitrary words from the English, Spanish and Italian vocabulary. As said, no experiment lead to the implementation of an effective GAN based augmentation. Augmenting the selected GANs training sets with other real handwritten text databases might result into generative models being able to produce higher quality synthetic data points exploitable as valuable OCR training samples. Furthermore, the implementation of the proposed GAN based augmentation in an online fashion (i.e. synthetic data points are generated on the fly during the training execution) might help to properly tune the amount of artificial images to be inserted in the real data set so to counteract the mentioned memorization effect. The introduction of an image de-noiser acting over ENEL_{DS} images might help to counteract the concept drift side effects. Such a future work has actually already been identified by the ENEL NLP team, being the topic of another proposed internship. Upon the adoption of such a de-noiser, the ENEL_{DS} might also be included in the GANs training set, aiming to the capability to generate data according to the probability distribution over which the ENEL_{DS} is sampled. Indeed, such a GANs training set's augmentation is currently not possible solely due to the extreme sensibility of a proper GANs convergence to the noise contained in its training data points.

Bibliography

[1] Wikipedia contributors. *Handwriting recognition — Wikipedia, The Free Encyclopedia*. [Online; accessed 17-June-2022]. 2022. URL: https://en.wikipedia.org/w/index.php?title=Handwriting_recognition&oldid=1086761434.

[2] Ian J. Goodfellow, Yoshua Bengio, and Aaron C. Courville. "Deep Learning". In: *Nature* 521 (2015), pp. 436–444.

[3] Léon Bottou and Olivier Bousquet. "The Tradeoffs of Large Scale Learning". In: *Advances in Neural Information Processing Systems*. Ed. by J. Platt et al. Vol. 20. Curran Associates, Inc., 2007. URL: https://proceedings.neurips.cc/paper/2007/file/0d3180d672e08b4c5312dcdafdf6ef36-Paper.pdf.

[4] Samuel L. Smith, Erich Elsen, and Soham De. *On the Generalization Benefit of Noise in Stochastic Gradient Descent*. 2020. DOI: 10.48550/ARXIV.2006.15081. URL: https://arxiv.org/abs/2006.15081.

[5] Richard Mintick. *Stochastic gradient descent*. URL: https://sciencesprings.wordpress.com/tag/stochastic-gradient-descent/.

[6] Suzana Herculano-Houzel. "The remarkable, yet not extraordinary, human brain as a scaled-up primate brain and its associated cost". In: *Proceedings of the National Academy of Sciences* 109.supplement_1 (2012), pp. 10661–10668. DOI: 10.1073/pnas.1201895109. eprint: https://www.pnas.org/doi/pdf/10.1073/pnas.1201895109. URL: https://www.pnas.org/doi/abs/10.1073/pnas.1201895109.

[7] Shaden Smith et al. *Using DeepSpeed and Megatron to Train Megatron-Turing NLG 530B, A Large-Scale Generative Language Model*. 2022. DOI: 10.48550/ARXIV.2201.11990. URL: https://arxiv.org/abs/2201.11990.

[8] Bernhard Mehlig. "Artificial neural networks". In: *arXiv preprint arXiv:1901.05639* (2019).

[9] Wikimedia Commons. *File:Artificial neural network.svg — Wikimedia Commons, the free media repository*. [Online; accessed 9-December-2020]. 2020. URL: https://commons.wikimedia.org/w/index.php?title=File:Artificial_neural_network.svg&oldid=494529927.

[10] George Cybenko. "Approximation by superpositions of a sigmoidal function". In: *Mathematics of control, signals and systems* 2.4 (1989), pp. 303–314.

[11] Yoav Artzi. *Notes and slides of the Natural Language Processing's course, A.Y. 2017/18, Cornell's Computer Science department.*

[12] Simone Scardapane. *Notes and slides of the Neural Networks for Data Science application's course, A.Y. 2021/22, MSc in Data Science, la Sapienza.*

[13] Fabio Galasso. *Notes and slides of the Advanced Machine Learning's course, A.Y. 2021/22, MSc in Computer Science, la Sapienza.*

[14] Nallagoni Omkar. *Activation Functions with Derivative and Python code: Sigmoid vs Tanh Vs Relu*. URL: https://medium.com/@omkar.nallagoni/activation-functions-with-derivative-and-python-code-sigmoid-vs-tanh-vs-relu-44d23915c1f4.

[15] Ayoosh Katuria. *Intro to Optimization in Deep Learning: Vanishing Gradients and Choosing the Right Activation Function*. URL: https://blog.paperspace.com/vanishing-gradients-activation-function/.

[16] Muhammad Asghar et al. "Assessment of Deep Learning Methodology for Self-Organizing 5G Networks". In: *Applied Sciences* 9 (July 2019), p. 2975. DOI: 10.3390/app9152975.

[17] Fisher Yu and Vladlen Koltun. *Multi-Scale Context Aggregation by Dilated Convolutions*. 2015. DOI: 10.48550/ARXIV.1511.07122. URL: https://arxiv.org/abs/1511.07122.

[18] Zhaoyang Niu, Guoqiang Zhong, and Hui Yu. "A review on the attention mechanism of deep learning". In: *Neurocomputing* 452 (2021), pp. 48–62. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2021.03.091. URL: https://www.sciencedirect.com/science/article/pii/S092523122100477X.

[19] Ronald Rensink. "The Dynamic Representation of Scenes". In: *Visual Cognition* 7 (Jan. 2000), pp. 17–42. DOI: 10.1080/135062800394667.

[20] Maurizio Corbetta and Gordon Shulman. "Control of Goal-Directed and Stimulus-Driven Attention in the Brain". In: *Nature reviews. Neuroscience* 3 (Apr. 2002), pp. 201–15. DOI: 10.1038/nrn755.

[21] Ashish Vaswani et al. *Attention Is All You Need*. 2017. DOI: 10.48550/ARXIV.1706.03762. URL: https://arxiv.org/abs/1706.03762.

[22] Kelvin Xu et al. *Show, Attend and Tell: Neural Image Caption Generation with Visual Attention*. 2015. DOI: 10.48550/ARXIV.1502.03044. URL: https://arxiv.org/abs/1502.03044.

[23] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. *Sequence to Sequence Learning with Neural Networks*. 2014. DOI: 10.48550/ARXIV.1409.3215. URL: https://arxiv.org/abs/1409.3215.

[24] Charu Aggarwal, Alexander Hinneburg, and Daniel Keim. "On the Surprising Behavior of Distance Metric in High-Dimensional Space". In: *First publ. in: Database theory, ICDT 200, 8th International Conference, London, UK, January 4 - 6, 2001 / Jan Van den Bussche ... (eds.). Berlin: Springer, 2001, pp. 420-434 (=Lecture notes in computer science ; 1973)* (Feb. 2002).

[25] Paul Michel, Omer Levy, and Graham Neubig. "Are Sixteen Heads Really Better than One?" In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach et al. Vol. 32. Curran Associates, Inc., 2019. URL: https://proceedings.neurips.cc/paper/2019/file/2c601ad9d2ff9bc8b282670cdd54f69f-Paper.pdf.

[26] Andrew Ng and Michael Jordan. "On Discriminative vs. Generative Classifiers: A comparison of logistic regression and naive Bayes". In: *Advances in Neural Information Processing Systems*. Ed. by T. Dietterich, S. Becker, and Z. Ghahramani. Vol. 14. MIT Press, 2001. URL: https://proceedings.neurips.cc/paper/2001/file/7b7a53e239400a13bd6be6c91c4f6c4e-Paper.pdf.

[27] Vladimir Vapnik. *The nature of statistical learning theory*. Springer science & business media, 1999.

[28] Emanuele Rodolà. *Notes and slides of the Deep Learning and Artificial Intelligence's course, A.Y. 2021/22, MSc in Computer Science, la Sapienza.*

[29] Irhum Shafkat. *Intuitively Understanding Variational Autoencoders.* URL: https://towardsdatascience.com/intuitively-understanding-variational-autoencoders-1bfe67eb5daf.

[30] Antonia Creswell et al. "Generative Adversarial Networks: An Overview". In: *IEEE Signal Processing Magazine* 35.1 (Jan. 2018), pp. 53–65. DOI: 10.1109/msp.2017.2765202. URL: https://doi.org/10.1109%2Fmsp.2017.2765202.

[31] Ian J. Goodfellow et al. *Generative Adversarial Networks.* 2014. DOI: 10.48550/ARXIV.1406.2661. URL: https://arxiv.org/abs/1406.2661.

[32] Tero Karras et al. *Progressive Growing of GANs for Improved Quality, Stability, and Variation.* 2017. DOI: 10.48550/ARXIV.1710.10196. URL: https://arxiv.org/abs/1710.10196.

[33] Kevin P Murphy. *Machine learning: a probabilistic perspective.* Cambridge, MA, 2012.

[34] Wikipedia contributors. *Kullback–Leibler divergence — Wikipedia, The Free Encyclopedia.* [Online; accessed 27-June-2022]. 2022. URL: https://en.wikipedia.org/wiki/Kullback%E2%80%93Leibler_divergence.

[35] Alex Graves et al. "Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural 'networks". In: vol. 2006. Jan. 2006, pp. 369–376. DOI: 10.1145/1143844.1143891.

[36] Harald Scheidl. *Word Beam Search: A CTC Decoding Algorithm.* URL: https://towardsdatascience.com/word-beam-search-a-ctc-decoding-algorithm-b051d28f3d2e.

[37] Harald Scheidl. "Handwritten text recognition in historical documents". PhD thesis. Wien, 2018.

[38] Wikipedia contributors. *Wasserstein metric — Wikipedia, The Free Encyclopedia.* [Online; accessed 27-June-2022]. 2022. URL: https://en.wikipedia.org/w/index.php?title=Wasserstein_metric&oldid=1093376527.

[39] Wikipedia contributors. *Fréchet inception distance — Wikipedia, The Free Encyclopedia.* [Online; accessed 27-June-2022]. 2022. URL: https://en.wikipedia.org/wiki/Fr%C3%A9chet_inception_distance.

[40] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning.* Vol. 4. 4. Springer, 2006.

[41] Wikipedia contributors. *Softmax function — Wikipedia, The Free Encyclopedia.* [Online; accessed 27-June-2022]. 2022. URL: https://en.wikipedia.org/w/index.php?title=Softmax_function&oldid=1091841696.

[42] Geoffrey E Hinton and Sam Roweis. "Stochastic Neighbor Embedding". In: *Advances in Neural Information Processing Systems.* Ed. by S. Becker, S. Thrun, and K. Obermayer. Vol. 15. MIT Press, 2002. URL: https://proceedings.neurips.cc/paper/2002/file/6150ccc6069bea6b5716254057a194ef-Paper.pdf.

[43] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization.* 2014. DOI: 10.48550/ARXIV.1412.6980. URL: https://arxiv.org/abs/1412.6980.

[44] Ceren Guzel Turhan and H. Bilge. "Recent Trends in Deep Generative Models: a Review". In: Sept. 2018. DOI: 10.1109/UBMK.2018.8566353.

[45] Pascal Vincent et al. "Extracting and composing robust features with denoising autoencoders". In: Jan. 2008, pp. 1096–1103. DOI: 10.1145/1390156.1390294.

[46] Pascal Vincent et al. "Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion". In: *J. Mach. Learn. Res.* 11 (2010), pp. 3371–3408.

[47] Diederik P Kingma and Max Welling. *Auto-Encoding Variational Bayes*. 2013. DOI: 10.48550/ARXIV.1312.6114. URL: https://arxiv.org/abs/1312.6114.

[48] Alireza Makhzani et al. *Adversarial Autoencoders*. 2015. DOI: 10.48550/ARXIV.1511.05644. URL: https://arxiv.org/abs/1511.05644.

[49] Daniel Jiwoong Im et al. "Denoising Criterion for Variational Auto-Encoding Framework". In: *AAAI*. 2017.

[50] Irina Higgins et al. "beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework". In: *ICLR*. 2017.

[51] Jon Gauthier. "Conditional generative adversarial nets for convolutional face generation". In: *Class project for Stanford CS231N: convolutional neural networks for visual recognition, Winter semester* 2014.5 (2014), p. 2.

[52] Arantxa Casanova et al. *Instance-Conditioned GAN*. 2021. DOI: 10.48550/ARXIV.2109.05070. URL: https://arxiv.org/abs/2109.05070.

[53] Alec Radford, Luke Metz, and Soumith Chintala. *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks*. 2015. DOI: 10.48550/ARXIV.1511.06434. URL: https://arxiv.org/abs/1511.06434.

[54] Tim Salimans et al. "Improved Techniques for Training GANs". In: *ArXiv* abs/1606.03498 (2016).

[55] Jeff Donahue, Philipp Krähenbühl, and Trevor Darrell. *Adversarial Feature Learning*. 2016. DOI: 10.48550/ARXIV.1605.09782. URL: https://arxiv.org/abs/1605.09782.

[56] Scott Reed et al. "Generative Adversarial Text to Image Synthesis". In: *Proceedings of The 33rd International Conference on Machine Learning*. Ed. by Maria Florina Balcan and Kilian Q. Weinberger. Vol. 48. Proceedings of Machine Learning Research. New York, New York, USA: PMLR, 20–22 Jun 2016, pp. 1060–1069. URL: https://proceedings.mlr.press/v48/reed16.html.

[57] Ankan Kumar Bhunia et al. "Handwriting Transformers". In: (2021). DOI: 10.48550/ARXIV.2104.03964. URL: https://arxiv.org/abs/2104.03964.

[58] Tom S. F. Haines, Oisin Mac Aodha, and Gabriel J. Brostow. "My Text in Your Handwriting". In: *ACM Transactions on Graphics (TOG)* 35 (2016), pp. 1–18.

[59] Bo Chang et al. "Generating Handwritten Chinese Characters Using CycleGAN". In: *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE, Mar. 2018. DOI: 10.1109/wacv.2018.00028. URL: https://doi.org/10.1109%2Fwacv.2018.00028.

[60] Sharon Fogel et al. *ScrabbleGAN: Semi-Supervised Varying Length Handwritten Text Generation*. 2020. DOI: 10.48550/ARXIV.2003.10557. URL: https://arxiv.org/abs/2003.10557.

[61] Han Zhang et al. *Self-Attention Generative Adversarial Networks*. 2018. DOI: 10.48550/ARXIV.1805.08318. URL: https://arxiv.org/abs/1805.08318.

[62] Brian Davis et al. *Text and Style Conditioned GAN for Generation of Offline Handwriting Lines*. 2020. DOI: 10.48550/ARXIV.2009.00678. URL: https://arxiv.org/abs/2009.00678.

[63] Ji Gan and Weiqiang Wang. "HiGAN: Handwriting Imitation Conditioned on Arbitrary-Length Texts and Disentangled Styles". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 35.9 (May 2021), pp. 7484–7492. URL: https://ojs.aaai.org/index.php/AAAI/article/view/16917.

[64] fki - Computer Vision and Artificial Intelligence. *IAM Handwriting database*. URL: https://fki.tic.heia-fr.ch/databases/iam-handwriting-database.

[65] Urs-Viktor Marti and H. Bunke. "The IAM-database: An English sentence database for offline handwriting recognition". In: *International Journal on Document Analysis and Recognition* 5 (Nov. 2002), pp. 39–46. DOI: 10.1007/s100320200071.

[66] Ray Smith. *Tesseract Open Source Optical Character Recognition Engine*. URL: https://github.com/tesseract-ocr/tesseract.

[67] Ankush Gupta Abhishek Dutta and Andrew Zisserman. *VGG Image Annotator (VIA)*. URL: https://annotate.officialstatistics.org/.

[68] Oxford University press. *Oxford 5000*. URL: https://www.oxfordlearnersdictionaries.com/wordlists/oxford3000-5000.

[69] E. Kool. *Italian Frequency Dictionary for Learners: Practical Vocabulary - Top 10.000 Italian Words*. CreateSpace Independent Publishing Platform, 2017. ISBN: 9781977899729. URL: https://books.google.it/books?id=yym4swEACAAJ.

[70] M. Davies and K.H. Davies. *A Frequency Dictionary of Spanish: Core Vocabulary for Learners*. Taylor & Francis, 2017. ISBN: 9781134874460. URL: https://books.google.it/books?id=emUPEAAAQBAJ.

[71] Baoguang Shi, Xiang Bai, and Cong Yao. *An End-to-End Trainable Neural Network for Image-based Sequence Recognition and Its Application to Scene Text Recognition*. 2015. DOI: 10.48550/ARXIV.1507.05717. URL: https://arxiv.org/abs/1507.05717.

[72] Saul Dobilas. *LSTM Recurrent Neural Networks — How to Teach a Network to Remember the Past*. URL: https://towardsdatascience.com/lstm-recurrent-neural-networks-how-to-teach-a-network-to-remember-the-past-55e54c2ff22e.

[73] Harm de Vries et al. *Modulating early visual processing by language*. 2017. DOI: 10.48550/ARXIV.1707.00683. URL: https://arxiv.org/abs/1707.00683.

[74] Andrew Brock, Jeff Donahue, and Karen Simonyan. *Large Scale GAN Training for High Fidelity Natural Image Synthesis*. 2018. DOI: 10.48550/ARXIV.1809.11096. URL: https://arxiv.org/abs/1809.11096.

[75] Eloi Alonso, Bastien Moysset, and Ronaldo Messina. *Adversarial Generation of Handwritten Text Images Conditioned on Sequences*. 2019. DOI: 10.48550/ARXIV.1903.00277. URL: https://arxiv.org/abs/1903.00277.

[76] Lei Kang et al. *GANwriting: Content-Conditioned Generation of Styled Handwritten Word Images*. 2020. DOI: 10.48550/ARXIV.2003.02567. URL: https://arxiv.org/abs/2003.02567.

[77] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. DOI: 10.48550/ARXIV.1512.03385. URL: https://arxiv.org/abs/1512.03385.