ALMA MATER STUDIORUM
UNIVERSITÀ DEGLI STUDI DI BOLOGNA

Scuola di Ingegneria e Architettura
Corso di Laurea Magistrale in Ingegneria e Scienze Informatiche

FROM ARTIFICIAL INTELLIGENCE TO ARTIFICIAL ART: DEEP LEARNING WITH GENERATIVE ADVERSARIAL NETWORKS

Tesi di Laurea Magistrale di:
RICCARDO BENEDETTI

Relatore:
Prof. GIANLUCA MORO
Correlatore:
Ing. ROBERTO PASOLINI

ANNO ACCADEMICO 2018-2019
SESSIONE II

KEYWORDS

Generative

Adversarial

Network

Art

Deep Learning

DEDICATA AI MIEI GENITORI
E A MIA SORELLA ANNA

Contents

Abstract

Neural Networks had a great impact on Artificial Intelligence and nowadays the Deep Learning algorithms are widely used to extract knowledge from huge amount of data.
This thesis aims to revisit the evolution of Deep Learning from the origins till the current state-of-art by focusing on a particular prospective. The main question we try to answer is: can AI exhibit artistic abilities comparable to the human ones?
Recovering the definition of the Turing Test, we propose a similar formulation of the concept, indeed, we would like to test the machine's ability to exhibit artistic behaviour equivalent to, or indistinguishable from, that of a human.
The argument we will analyze as a support for this debate is an interesting and innovative idea coming from the field of Deep Learning, known as **Generative Adversarial Network (GAN)**.
GAN is basically a system composed of two neural network fighting each other in a zero-sum game. The "bullets" fired during this challenge are simply images generated by one of the two networks.
The interesting part in this scenario is that, with a proper system design and training, after several iteration these fake generated images start to become more and more closer to the ones we see in the reality, making indistinguishable what is real from what is not.
We will talk about some real anecdotes around GANs to spice up even more the discussion generated by the question previously posed and we will present some recent real world application based on GANs to emphasize their importance also in term of business.
We will conclude with a practical experiment over an Amazon catalogue of clothing images and reviews with the aim of generating new never seen product starting from the most popular existing ones.

Abstract (italian version)

Da sempre le reti neurali hanno avuto una grande rilevanza nel campo dell'Intelligenza Artificiale e oggigiorno gli algoritmi di Deep Learning vengono ampiamente utilizzati per estrarre conoscenza da grandi quantità di dati.
Questa tesi ha come obiettivo quello di ripercorrere l'evoluzione del Deep Learning dalle orgini fino ad arrivare all'attuale stato dell'arte, affrontando però l'argomento da un punto di vista differente. La domanda a cui cercheremo di dare una risposta è: quant'è lontana, ad oggi, l'Intelligenza Artificiale dal mostrare capacità artistiche paragonabili a quelle di un normale essere umano?
Questa domanda può essere vista come una riformulazione della definizione di "Test di Turnig", in particolare vogliamo testare l'abilità di una macchina di emulare l'artista in modo tale che i suoi risultati appaiano equivalenti, o indistinguibili, da quelli di un artista umano.
Lo strumento che intendiamo analizzare a supporto di questo dibattito è un'idea innovativa e rivoluzionaria che viene dall'area del Deep Learning ed è conosciuta col nome di **Rete Antagonista Generativa (GAN)**.
Una GAN è un sistema che vede due reti neurali combattere l'un l'altra in un gioco a somma zero. Le "armi" di questo combattimento sono semplicemente immagini generate da una delle due reti.
La parte interessante di questo scenario è che, con un'opportuna progettazione delle reti, scelta del dataset e relativo addestramento, è possibile arrivare a una situazione in cui le immagini generate iniziano a sembrare talmente verosimili da rendere indistinguibile ciò che è veramente reale da ciò che non lo è.
Vi racconteremo inoltre alcuni aneddoti riguardanti le GAN che potrebbero accendere ulteriormente la discussione attorno alla domanda posta

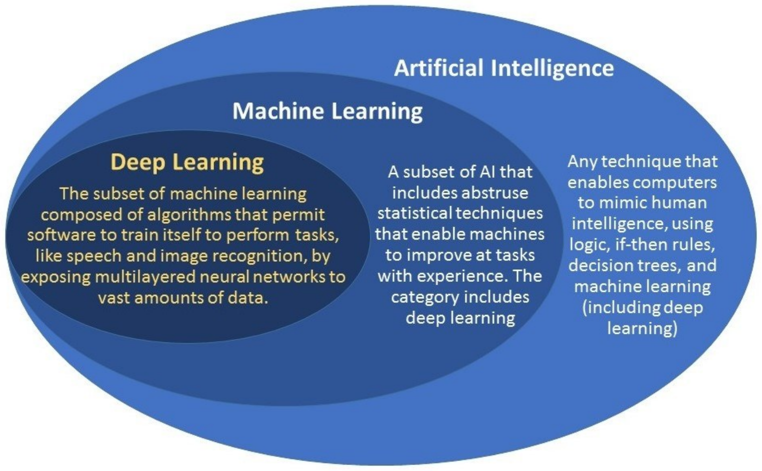
in precedenza e presenteremo alcune applicazioni esistenti basate su GAN per enfatizzare la loro importanza anche nel mondo del business. Infine, concluderemo la tesi con un pratico esperimento, utilizzando un dataset Amazon di immagini provenienti da cataloghi di recensioni e prodotti di moda, con lo scopo di generare immagini di nuovi prodotti mai visti in precedenza, sfruttando le caratteristiche di quelli piú popolari e piú venduti.

Chapter 1

A Deep Learning overview

1.1 AI, Machine Learning and Deep Learning

In the field of Computer Science, these three concepts are often erroneously employed in an almost identical manner, sometimes even interchanged. The truth is they have a very different meaning.
In order to understand the difference between **Artificial Intelligence**, **Machine Learning** and **Deep Learning**, can be useful to see them as a series of concentric circles.



Artificial Intelligence
Machine Learning
Deep Learning
The subset of machine learning composed of algorithms that permit software to train itself to perform tasks, like speech and image recognition, by exposing multilayered neural networks to vast amounts of data.
A subset of AI that includes abstruse statistical techniques that enable machines to improve at tasks with experience. The category includes deep learning
Any technique that enables computers to mimic human intelligence, using logic, if-then rules, decision trees, and machine learning (including deep learning)

Figure 1.1: The AI hierarchy of concepts

Into the large outer circle we have the **Artificial Intelligence**. Artificial Intelligence is the goal we would like to achieve. More specifically, we want to create software or hardware able to think and solve problems just like a human being. The problems we are talking about are those that most affect our everyday life and the humans are used to figure out pretty easily. Unfortunately, machines are not provided of a native ability to handle this kind of challenges, with some including for instance text interpretation or image recognition.
The overall objective is a general AI, a highly smart system as the ones we can see in science fiction, like C-3PO from Star Wars, which is able to emulate the human behaviour through and through. At the moment, we are not even nearly close to this kind of artificial intelligence, so in practice we have to refer to it as a (very)long-term objective.
The example we have just been discussing is also referred as **Strong AI**. Instead, what we have nowadays can be identified only as **Weak AI**, which is the use of techniques and algorithms with the aim to solving just some individual tasks.
Inside this big circle of the Artificial Intelligence we can find a smaller circle that is the circle of **Machine Learning**. Machine Learning concerns the use of big data set and a series of classification algorithms that overturns the common approaches of computer programming. A traditional programmer, in fact, is used to write algorithms increasingly complex but in a way he knows exactly how it works every single statement.
The basic idea of creating a classifier is quite different. The first step consists in retrieving a significant amount of data, and then to create a bunch of functions that can help us to understand which of these data we are really interested in. This approach should lead a gradual improvements of the results we get back and, moreover, the possibility to obtain a system capable of making decisions based on the available data, without writing all the specific algorithm.
Most of these classifiers are based on well-known mathematical functions, like linear functions, polynomial functions, clustering of various types, statistical functions and so on. Many classifiers can also predict some kind of future trends by exploiting time series, like the price of a product, the forecast average selling estimates for the following

months or even some indications about financial investments.
Nested inside the Machine Learning circle exists another sub-branch called **Deep Learning**. All the Deep Learning techniques are based on the artificial neural networks, a part of software that tries to replicate, with some respects, the functioning of the brain cells.
From 2012 onwards, Google researchers efforts have been completely revolutionized the state of art of Deep Learning. The term "deep" refers to the number of levels, or layers, that characterizes the depth of the neural network architecture. The way a neural network works is even more interesting. It's indeed the machine itself who determines the classifiers by exploiting a set of data usually much larger than the ones used in the ordinary Machine Learning. In other words, the machine chooses and defines the specific classifiers to be used. These classifiers are, in short, not pre-selected by the researchers, but seem to have a significantly higher impact from a point of view of desirable results we would like to expect.
When we talk about the three terminology: Artificial Intelligence, Machine Learning and Deep Learning; we must be aware of their very different meanings. The Artificial Intelligence is the goal we wish to achieve, the Machine Learning is one of the possible approach to dealing with it, and the Deep Learning is a particular advanced technique of Machine Learning the we could exploit, maybe now the most promising one.

1.2 The need for Deep Learning

Deep Learning is a branch of Artificial Intelligence (in particular of the Machine Learning) that deals with solving some kind of problems through the use of **multilayer neural networks**.
It represents an innovative approach compared to the first Shallow Networks that were composed by a low number of layers (typically three: input, hidden and output layer) where the most part of the neurons was situated in the broad intermediate.
Although there are studies that demonstrate how the Shallow Networks are able to solve the same problems as Deep Networks [1], the layered approach is more desirable for a multitude of reasons. The

most relevant one is about **parameter reduction**. It has been proven the fact that states the representation power of a `k`-layer neural network with polynomial many neurons needs to be expressed with exponentially many neurons if a `(k-1)`-layer structured is used. Further supports come from the comparison with the human brain visual cortex both in terms of structure, human neural network is effectively a deep architecture, and in terms of behaviour, humans tend to represent concepts at one level of abstraction as the composition of concepts at lower levels.

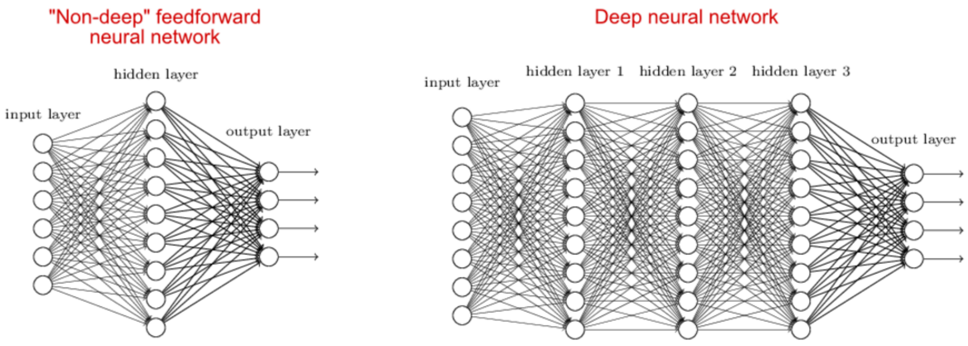


Figure 1.2: Shallow network Vs. Deep network

Relying on the previous assertions, some researchers [2] refer to the Deep Learning also with the term “Hierarchical Feature Learning” because the main skill of these networks is to model high level abstractions starting from the lowest level characteristics, producing in practice an hierarchical structure of features. Deep learning algorithms, in fact, seek to exploit the unknown structure in the input distribution in order to discover good representations, often at multiple levels, with higher-level learned features defined in terms of lower-level features. The objective is to make these higher-level representations even more abstract, with their individual features more invariant respect to most of the variations that are typically present in the training distribution, while collectively preserving as much as possible of the information of the input.
The research fields involved range from the images recognition and computer vision, to the voice recognition, to the discovery of new

drugs, text analysis and much more.
Nowadays all the main technological giants invest in Deep Learning for research and innovation purposes. In the forefront we find for example Google, Facebook, Microsoft, Amazon, etc.

1.3 The cognitive system: from Aristotle to Hebb

[3] The modern progresses about Artificial Neural Networks have been made possible thanks to the human ambition of understandings the functionality of our cognitive system. The first attempt is attributable to Aristotle (300 B.C.) who ispired the Associationism Theory through the following thought:

"When, therefore, we accomplish an act of reminiscence, we pass through a certain series of precursive movements, until we arrive at a movement on which the one we are in quest of is habitually consequent. Hence, too, it is that we hunt through the mental train, excogitating from the present or some other, and from similar or contrary or coadjacent. Through this process reminiscence takes place. For the movements are, in these cases, sometimes at the same time, sometimes parts of the same whole, so that the subsequent movement is already more than half accomplished."

Inspired by Plato, Aristotle examined the processes of remembrance and recall and brought up with four laws of association:

- Contiguity: Things or events with spatial or temporal proximity tend to be associated in the mind.
- Frequency: The number of occurrences of two events is proportional to the strength of association between these two events.
- Similarity: Thought of one event tends to trigger the thought of a similar event.

- Contrast: Thought of one event tends to trigger the thought of an opposite event.

These ideas have been revisited and redrafted by many philosophers or psychologists, especially from the 16th century onwards.
In addition to existing laws, David Hartley (1705-1757), as a physician, proposed his argument that memory could be conceived as smaller scale vibrations in the same regions of the brain as the original sensory experience. These vibrations can link up to represent complex ideas and therefore act as a material basis for the stream of consciousness. This idea potentially inspired Hebbian Learning Rule, introduced by Donald O. Hebb in his work *The Organization of Behavior.*
In 1949, Hebb stated the famous rule: "Cells that fire together, wire together", which emphasized on the activation behavior of co-fired cells. More specifically, in his book, he wrote that:

"When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that As efficiency, as one of the cells firing B, is increased."

This archaic paragraph can be re-written into modern machine learning languages as the following:

$$\Delta w_i = \eta x_i y$$

where Δw_i stands for the change of synaptic weights (w_i) of Neuron i, of which the input signal is x_i. y denotes the postsynaptic response and η denotes learning rate. In other words, Hebbian Learning Rule states that the connection between two units should be strengthened as the frequency of co-occurrences of these two units increase.
Even though Hebb is considered the pioneer of neural network, it should be noted that this rule raises some issues which can lead to a form of unstableness: as co-occurrences appear more, the weights of connections keep increasing and the weights of a dominant signal will increase exponentially.
This problem has been solved by Erkki Oja (1982), who updated the

HLR introducing a normalization factor. His updated learning rule would later demonstrated that a neuron does nothing but approximating the behavior of a Principal Component Analyzer (PCA).

1.4 Artificial neural networks over the years

The first important research in the ANN field dates back to 1943, when a neurophysiologist, Warren McCulloch, and a mathematician, Walter Pitts, speculated the inner workings of neurons and molded a primitive neural network [4]. Their neural model was inspired by the behaviour of electronic circuits, where the binary outputs are determined from a linear weighted combination of inputs. Unlike modern ANNs, this model was characterised by fixed weights, then non-adjustable over the time. This learning aspect was introduced only from 1949, with the Hebbian Learning Rule.
In 1958, Rosenblatt was investigating the area of vision systems and in this context he implemented the first electronic device called Perceptron [5]. Perceptron has represented the crucial transition from biological to artificial systems.
Basically, the representation power of Rosenblatt Perceptron concerns the definition of linear decision boundaries in order to handle basic logical operations, such as AND, OR and NOT. Here, it has emerged also a critical shortcoming highlighted by Minski and Papert (1969) [6]. They showed how a single Perceptron cannot solve XOR and NXOR since a non-linear decision boundary is required to properly separate binary output classes.
From 1974 onwards, the breakthrough came from the introduction of backpropagation and MultiLayer Perceptron (MLP). The neuron weights are no longer perceived as fixed parameter and backpropagation put into practice the concept of learning by providing the network a self-improvement approach.
It has also been proven how a simple MLP architecture can easily figure out the XOR problem. Furthermore, MLP allows to model cognitive processes such as classification, recognition, sensorimotor behaviours, association and memorization attitudes.

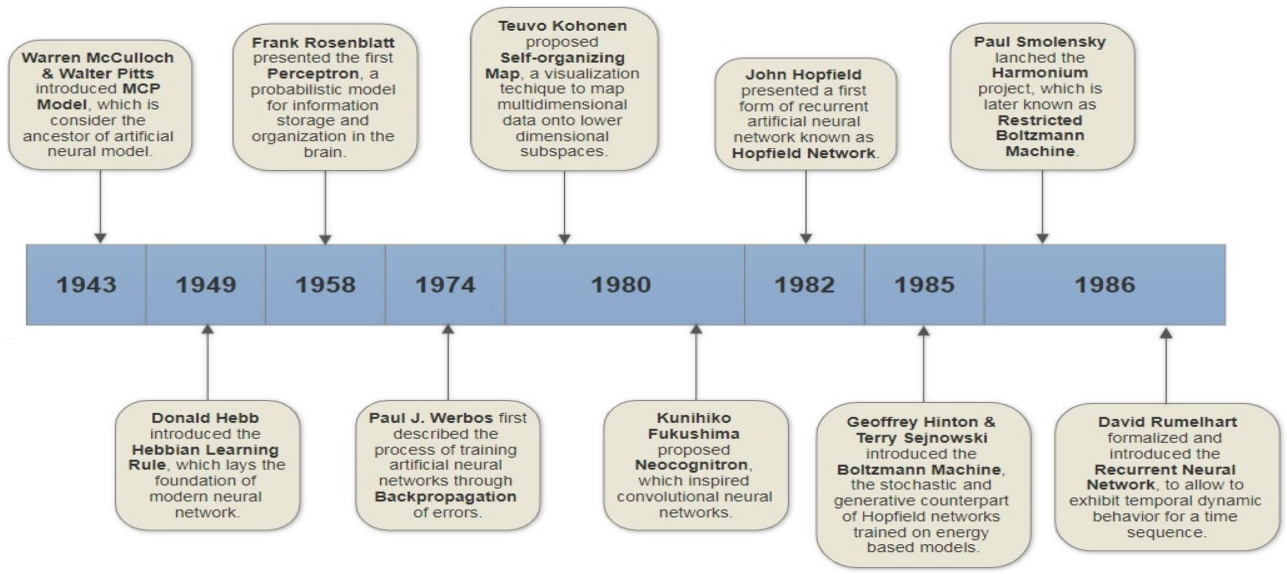

Warren McCulloch & Walter Pitts introduced MCP Model, which is consider the ancestor of artificial neural model.
Frank Rosenblatt presented the first Perceptron, a probabilistic model for information storage and organization in the brain.
Teuvo Kohonen proposed Self-organizing Map, a visualization techique to map multidimensional data onto lower dimensional subspaces.
John Hopfield presented a first form of recurrent artificial neural network known as Hopfield Network.
Paul Smolensky lanched the Harmonium project, which is later known as Restricted Boltzmann Machine.
1943
1949
1958
1974
1980
1982
1985
1986
Donald Hebb introduced the Hebbian Learning Rule, which lays the foundation of modern neural network.
Paul J. Werbos first described the process of training artificial neural networks through Backpropagation of errors.
Kunihiko Fukushima proposed Neocognitron, which inspired convolutional neural networks.
Geoffrey Hinton & Terry Sejnowski introduced the Boltzmann Machine, the stochastic and generative counterpart of Hopfield networks trained on energy based models.
David Rumelhart formalized and introduced the Recurrent Neural Network, to allow to exhibit temporal dynamic behavior for a time sequence.

Figure 1.3: Deep Learning milestones - Part 1

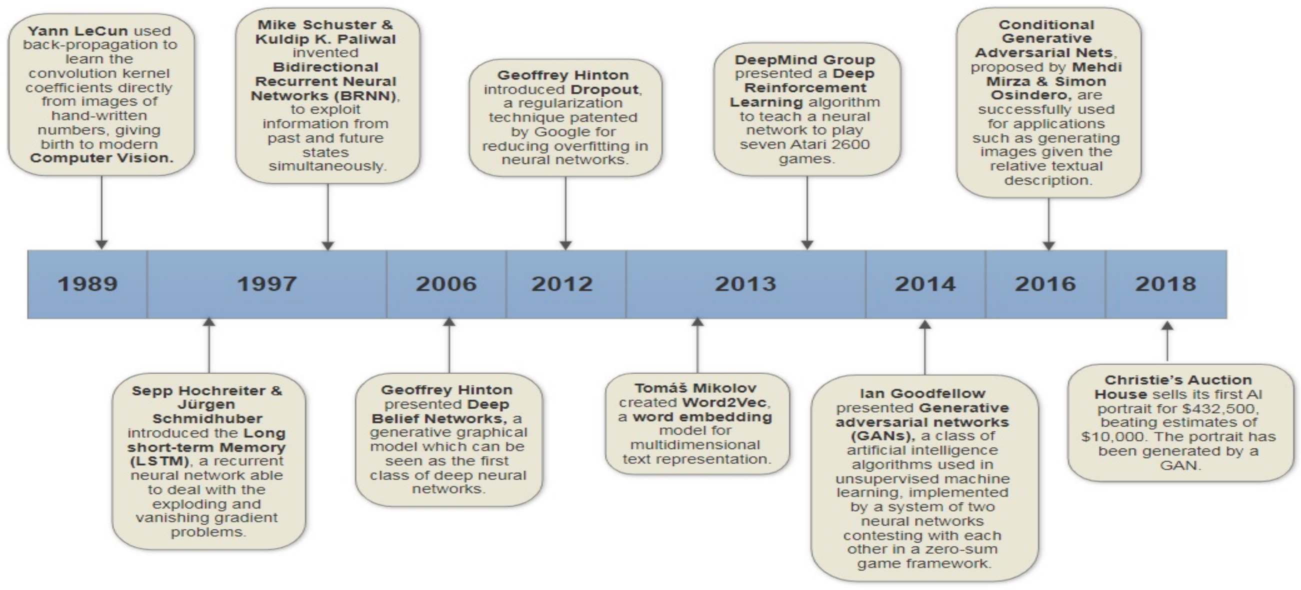

Yann LeCun used back-propagation to learn the convolution kernel coefficients directly from images of hand-written numbers, giving birth to modern Computer Vision.
Mike Schuster & Kuldip K. Paliwal invented Bidirectional Recurrent Neural Networks (BRNN), to exploit information from past and future states simultaneously.
Geoffrey Hinton introduced Dropout, a regularization technique patented by Google for reducing overfitting in neural networks.
DeepMind Group presented a Deep Reinforcement Learning algorithm to teach a neural network to play seven Atari 2600 games.
Conditional Generative Adversarial Nets, proposed by Mehdi Mirza & Simon Osindero, are successfully used for applications such as generating images given the relative textual description.
1989
1997
2006
2012
2013
2014
2016
2018
Sepp Hochreiter & Jürgen Schmidhuber introduced the Long short-term Memory (LSTM), a recurrent neural network able to deal with the exploding and vanishing gradient problems.
Geoffrey Hinton presented Deep Belief Networks, a generative graphical model which can be seen as the first class of deep neural networks.
Tomáš Mikolov created Word2Vec, a word embedding model for multidimensional text representation.
Ian Goodfellow presented Generative adversarial networks (GANs), a class of artificial intelligence algorithms used in unsupervised machine learning, implemented by a system of two neural networks contesting with each other in a zero-sum game framework.
Christie's Auction House sells its first AI portrait for $432,500, beating estimates of $10,000. The portrait has been generated by a GAN.

Figure 1.4: Deep Learning milestones - Part 2

The learning aspect of MLP is implemented on the basis of Delta Rule giving rise to first form of supervised learning, where the network can improve itself by changing its weight according to the errors committed (represented as the difference between the predicted output and the expected output).
Other remarkable works were carried out also in the context of unsupervised learning. Kohonen (1981) introduced the Self-Organizing Maps, a new kind of network that didn't need of a external supervisor but by exploiting the statistical properties of data can identify a series of categorical patterns.
Researches about unsupervised learning proceed with Hopfield Networks and Boltzmann Machines, culminating in Deep Boltzmann Machines and Deep Belief Networks. In particular the self-organization approach enables the modelling of hierarchical structure of features where the abstraction level grows up over hidden layers. Then, by investigating the latent data representation in these less wider hidden layers, another interesting idea was proposed: can neural network be used also as autoencoder? The answer is yes and in particular these autoencoders can be successfully employed for task such as dimensionality reduction and data denoising.
Around the 2000s, the enthusiasm about neural networks research and their practical applications gradually faded due to the unavailability of vast data sets and adequate computational resources.
Only in 2009, when the first results about speech recognition raised up, neural networks were reconsidered and expectations come back to growth. In 2013, Convolutional Neural Networks were successfully employed in Computer Vision tasks. In 2014, Recurrent Neural Networks were used for the first automatic translation models.
A further boost of encouragement came from the spread of Big Data and low-cost increasingly fast GPUs. These two contributions represent in fact the crucial solutions to all the problems that had caused the latest crisis, years earlier.
Currently we are able not only to analyze Big Data, but also to use them to make predictions and extract further knowledge.
With a view into the future, we are now so building the launch pad towards the Strong AI.

1.5 Learning approaches

In Machine Learning, all known learning approaches basically share the same principle: trying to learn what is the best output `o` to produce for each possible input `i` [7].
The previous assertion can be more accurately expressed in mathematical terms: trying to find a function `F: I` $\rightarrow$ `O` as efficiently as possible, where `I` is the set of possible inputs, and `O` the set of possible outputs.
Beyond this principle, some machine learning approaches raised up taking inspiration by real-life learning approaches, like training a pet or teaching a child. These examples usually involve an alternation of training and testing sessions, where the latter has the purpose of checking the progresses achieved after training.
Unfortunately, not all the problems cannot be solved adopting a unique standard approach and this often depends by the data we have at our disposal. We can have examples of data where we have both the inputs and outputs: `(i,o)`. In other cases, we can only have the inputs i. Sometimes, instead, we have no direct access to the correct output, but we can get some measure of the quality of an output o following input i.
The approach used to deal with first kind of problems is also called **supervised learning**. The idea consists in exploiting a large set of training data encoded as pairs `(i,o)`. The matching between input and output is represented by a function which often depends by a large number of parameters. The machine goal is to find an optimal assignment for those parameters and, in case of neural networks, this can be achieved by adjusting the neuron weights through backpropagation. Typical examples are spam detectors which are trained on data set on explicitly labelled spam and non-spam emails.
The welcomed outcome is to obtain a model able to generalize on fresh unseen data, or in other words, the learning process should avoid to overfit on training set.
The sore point of supervised learning is mainly related to the construction of the data set. Since the data are required to be properly hand labelled, it usually needed a huge manual effort. In general, most of the problems have just input data without prior knowledge about

output. Furthermore, many data mining scenarios start with no idea about the kind of output are looking for. Often, the most desired output is what you don't expect and this can be discovered by deeply investigating on input data, such as searching for correlation between features or detecting outliers with clustering techniques. One refers to these approach also as **unsupervised learning**.
In the field of natural language processing and text classification, in particular, the idea of word embedding represents one of the most relevant unsupervised models.
For sake of completeness, we briefly discuss also about a third kind of learning which in recent years is leading impressing results. It covers the category of problems in where it is possible to obtain an estimate of the output quality. The approach we are talking about is known as **reinforcement learning**.
Csaba Szepesvári [8] explained reinforcement learning as a learning paradigm concerned with learning to control a system so as to maximize a numerical performance measure that expresses a long-term objective. What distinguishes reinforcement learning from supervised learning is that only partial feedback is given to the learner about the learner's predictions. Moreover, the predictions may have long term effects through influencing the future state of the controlled system. Thus, time plays a special role. The goal in reinforcement learning is to develop efficient learning algorithms, as well as to understand the algorithm merits and limitations. Reinforcement learning is of great interest because of the large number of practical applications that it can be used to address, ranging from problems in artificial intelligence to operations research or control engineering.

Chapter 2

The Art of Training

2.1 Standard Backpropagation

Back in 1998, Yann LeCun wrote “backpropagation can seem more of an art than a science” [9].
The reason of that assertion is related to the fact that there is still no a well-defined best practice to apply this technique. The parameters to consider when designing a neural network are several, such as nodes, layers, learning rates, dropout, epochs, training and test sets, and so forth.

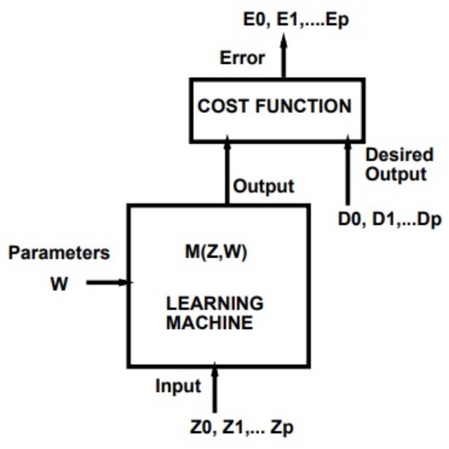


Figure 2.1: Gradient-based Learning Machine

Backpropagation is the core of gradient-based learning methods.
The main goal of the training process is to minimize, through the iterations, a cost function which keeps track of the deviation from the estimated output to the expected.
While the cost function decreases, the network should be able to better **generalize** its predictions, from the training examples to new fresh unseen.
So far the most widely used cost functions are the Mean Square Error (also known as Sum Squared Error or Maximum Likelihood) and the Cross Entropy (also known as Binary Cross Entropy or Bernoulli Negative Log-Likelihood). The first one is preferred in linear regression contexts, for example in the forecasting models used to predict the future trend of time series, while the Cross Entropy is more suitable for classification problems, therefore it is commonly used in the area of Image Recognition at the end of Convolutional Neural Networks to determine the category of the objects detected [10].
The number of possible cost functions is not limited to these two. The list includes also the Kullback-Leibler divergence (a measure of how one probability distribution is different from a second), Generalized Kullback-Leibler divergence, Exponentional cost, Hellinger distance, Itakura-Saito distance [11].
Another important consideration concerns the ways we feed the data into the model. There are two kind of approaches known as **Batch Learning** and **Stochastic Learning**, both of them with benefits and drawbacks.
Stochastic Learning is an online technique preferred over Batch because generally leads to faster computation and better results. Furthermore, while Batch tends to converge directly to a minimum which could not be the global one, Stochastic behaviour produces fluctuations very useful in order to avoid to get stuck in local minima.
A third kind of hybrid approach has been proposed by G.B. Orr [12]. The idea is to provide adaptive mini-batches as an alternative to fixed batch sizes, starting with one examples (pure stochastic) and then gradually increase over the epochs until it reaches the full training size (pure batch).
The advantage of mini-batches consists of removing the noise that can be accumulated after many stochastic iterations. But this is not the

only way, indeed it can also be achieved decreasing step by step the learning rate (learning rate decay).

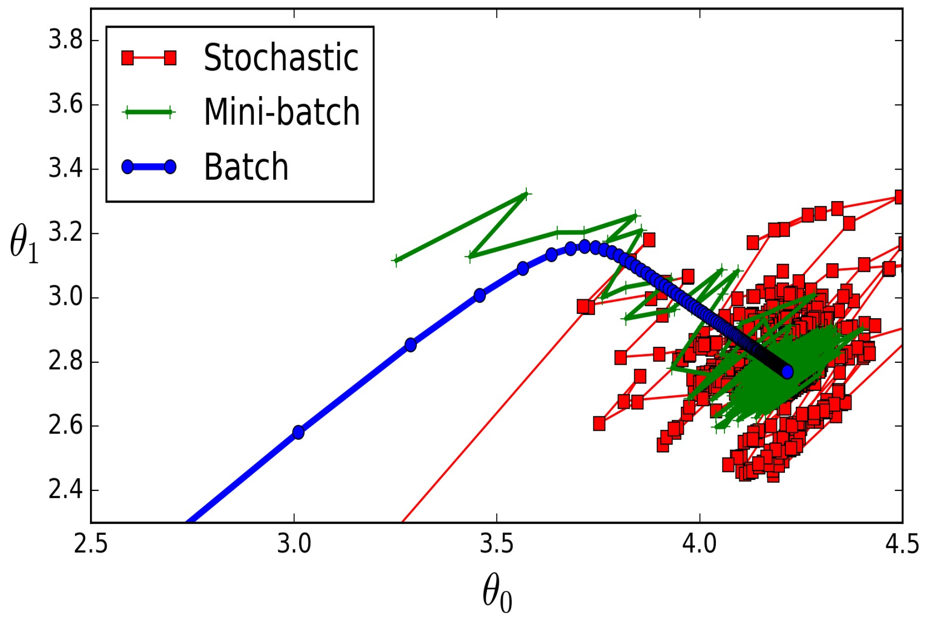


Figure 2.2: Stochastic, Batch and Mini-Batch
(credits: https://i.stack.imgur.com/lU3sx.png)

As the training proceed, some training examples can present very large errors respect to the other. This can be an indication that the model needs to be trained better on these data. In such cases it exists a method called "emphasizing scheme" which prioritize the examples with maximum information content, or in other words the most relevant for the network. So basically, the frequency with which an occurrence appears is proportional to the error previously committed by the model on that occurrence.
On the other side, it is also important to point out that such a method should be applied very carefully since may prove extremely counterproductive in case are present some outliers inside the dataset.

2.2 Input preprocessing

It happens very rarely that the input are fed the neural network "as is". In order to fast the convergence it is recommended to perform a series of transformations on all the input variables.
The most common transformation is called "normalization" and is performed combining two-step: shifting and scaling. The expected result is to arrange each input close to zero with also uniform covariances, through scaling. This is known as "mean cancellation" since the overall mean should become close to zero.
Then it is required a more tricky step: remove eventual linear correlation between the inputs, making them as independent as possible. The solution lies in Principal Component Analysis, or Karhunen-Loeve transformation, which by the way finds also application in image compression [13].

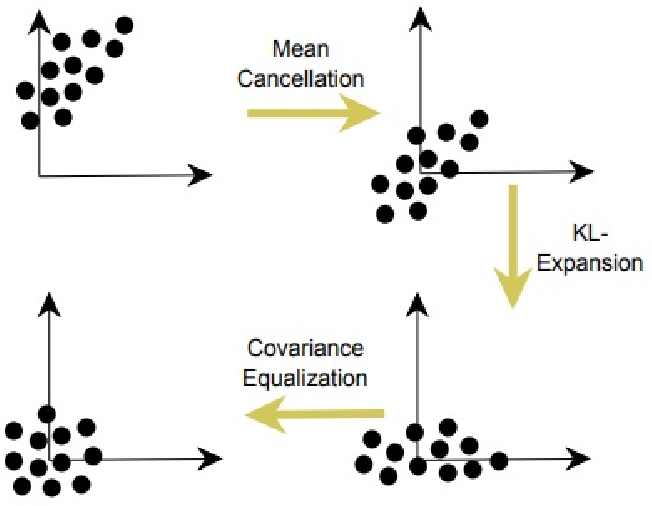


Figure 2.3: Input transformation

As mentioned above, all these transformations are applied to the input data in order to feed the input layer. But the general property of a fully connected neural network states that the output of a layer

becomes the input of the next one. Therefore, the output of a layer should be even produced in an appropriate format in the interests of the next layer.
This problem is addressed by the correct choice of a nonlinear activation function, such as the standard logistic function and the hyperbolic tangent.

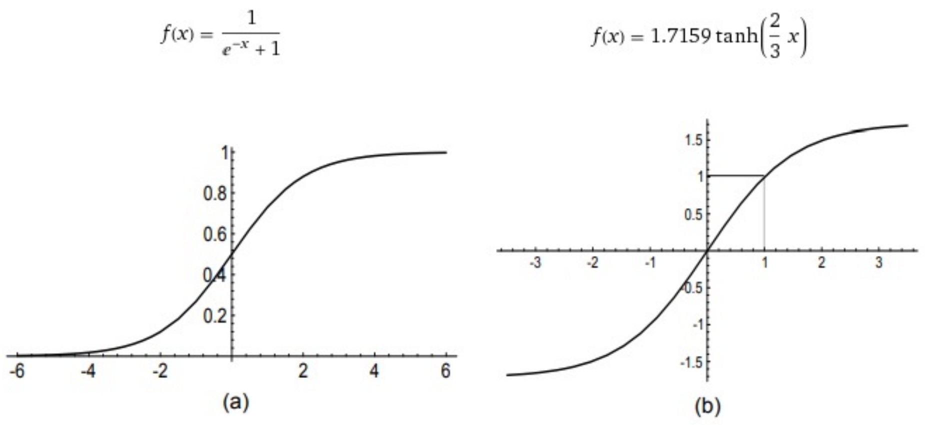


Figure 2.4: (a) Standard logistic function, (b) Hyperbolic tangent

Usually it is preferable an activation function symmetric about the origin, like the hyperbolic tangent: $f(x) = 1.7159 \tanh\left(\frac{2}{3}x\right)$. The constants in the formula are experimentally determined to work properly with the transformations discussed previously.

2.3 Hyperparameters initialization and tuning

2.3.1 Weights

Before starting the training phase of a neural network, it is necessary a proper initialization of a series of hyperparameters.
Let's start by considering the connections between neurons in adjacent layers. As we know, the synapses in a ANN are transmitted by exploiting this network of connections and each one is associated to a particular weight which plays a crucial role during the overall training process.
The weight set, usually represented as a multi-dimensional vector, is the most dynamic hyperparameter of a neural network as a consequence of applying backpropagation.
In order to avoid bad convergence, the weight must be initialized following some recommendations. First of all, they shouldn't be neither too small nor too large (this last case would cause the sigmoid saturation). In both cases it will end up in very small gradients and therefore very slow learning.
The best practice is given by the following rule: assuming you just have a training set normalized and you picked the proper sigmoid, the weights initialization should be randomly sampled from a distribution with zero mean and standard deviation $\sigma_w = m^{-1/2}$ (where m is the number of connections feeding into the node, or fan-in).

2.3.2 Learning Rate

The learning speed is controlled by another hyperparameter so called learning rate, η.
As well as for the weight initialization, a similar consideration can be applyed about the learning rate which concerns the risk of picking a too small or a too large value. Large value can lead to undesiderable divergence while small value can strongly slow down the convergence. The effect of different learning rates is pretty comprehensible with a simple example of a one dimensional cost function and it is even easy to decide the optimal learning rate η_{opt} for that.

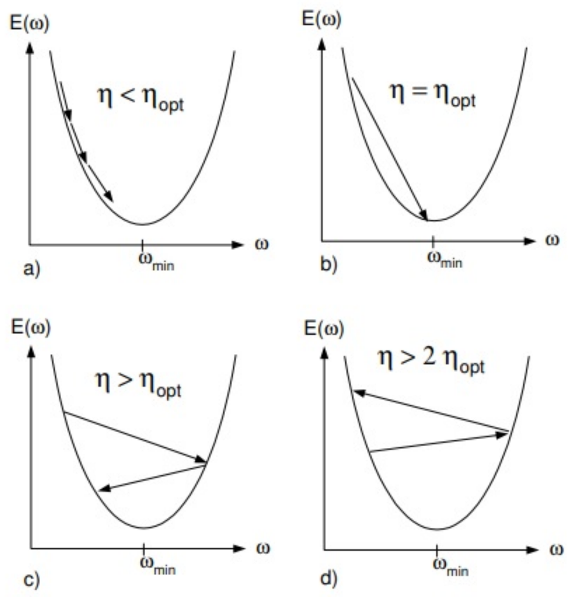


Figure 2.5: Learning rates comparison

Some empirical attempts suggest to treat the learning rate not as a global constant but as a decreasing function. One can refer to this approach as "learning rate decay" or "adaptive learning rate" [14].
The idea is to start with a reasonable large value and then gradually dump it over the training epochs.
Some modern adaptive learning rate methods are nowadays widely used: AdaGrad [15], ADAM [16] and RMSprop.
Another popular technique used along with stochastic gradient descent is the Momentum. Intuitively, Momentum helps the convergence by considering the previous iterations when calculating the new gradient. This implies the capability of moving quickly towards the minima and meanwhile smoothing eventual oscillations.

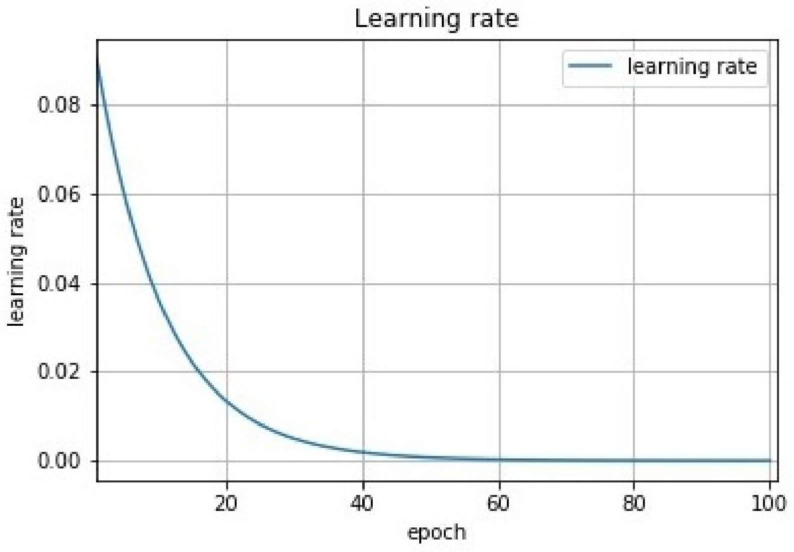


Figure 2.6: Example of an exponential decay schedule (credits: https://towardsdatascience.com/learning-rate-schedules-and-adaptive-learning-rate-methods-for-deep-learning-2c8f433990d1)

2.4 Rectifying activation function

The argument that motivated the research in the field of neural networks comes from the need of artificially emulating the mammalian visual cortex behaviour.
Even if some activation functions like the logistic sigmoid or the hyperbolic function work well, the studies coming from computational neuroscientists highlight that the neurons rarely reach their maximum saturation regime and generally the portion of neurons activate at the same time is between the 1 and 4% of the total.
This means the networks become highly sparse and more likely to be linearly separable.
In order to inject this behaviour to the artificial neural networks, an alternative kind of activation function had been proposed [17]. The Rectified Linear Units, often indicated as ReLU, instead of enforcing the sign symmetry or antisymmetry like the hyperbolic tangent, drops

to zero the negative inputs by applying the function $f(x) = \max(0, x)$. Statistically speaking, after an uniform weight initialization, the first rectification should drop the 50% of hidden units output, which will even more increase after eventual L1 regularization.

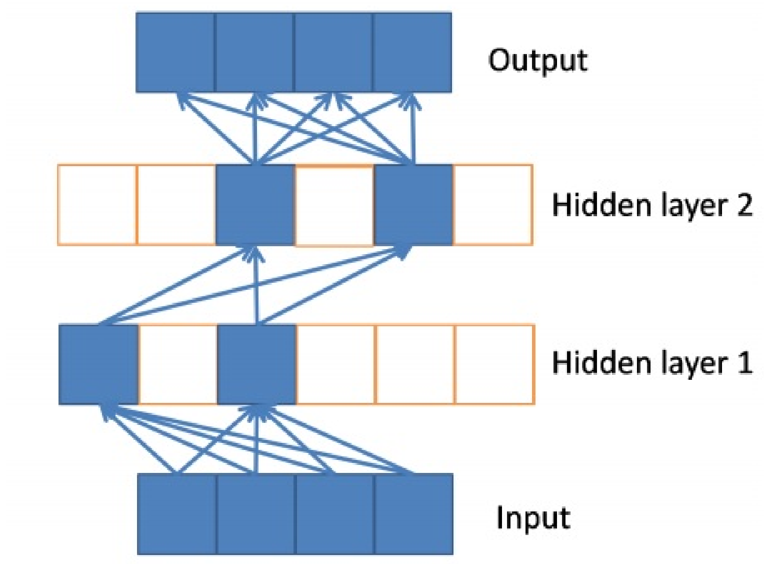


Figure 2.7: Example of sparse hidden activations produced by ReLU

The mathematical advantages are the elimination of the risk to incur in the vanishing gradient, the gradient flow will be faster and so the computation cheaper.
ReLU is widely used in convolutional neural network for object detection but some noticed that the impact on the activations is quite aggressive and may lead some drawbacks concerning the optimization performance.
For this reason it exists also a smoother version named Softplus, $f(x) = \log(1 + e^x)$, which reduce the exact sparsity. However, some experimental results belie this hypothesis and confirm the effective power of ReLU, which still the best choice.

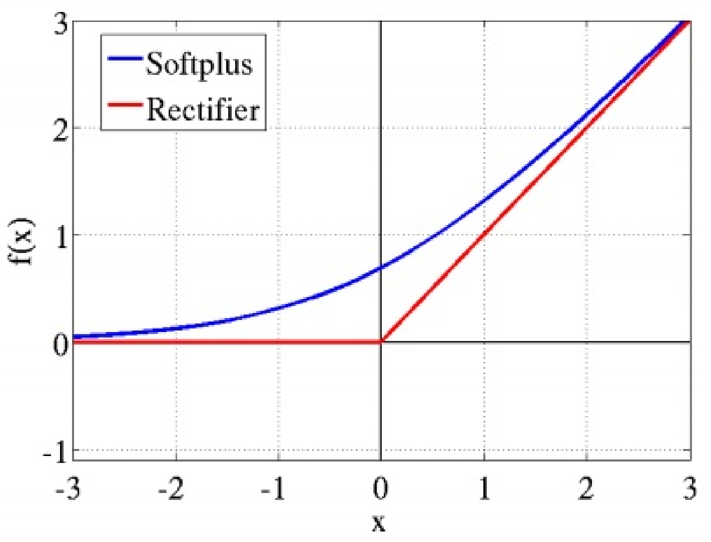


Figure 2.8: ReLU and Softplus

Chapter 3

Convolutional Neural Networks

3.1 Limitations of MLPs in image processing contexts

So far, we have seen some best practices to keep in mind when training fully-connected multi-layer networks useful for tasks such as general supervised classification. We have also introduced the ReLU activation function emphasizing the main advantages over the classical logistic sigmoid and hyperbolic tangent, and we presented it as an efficient choice for Convolutional Neural Networks.
In this chapter we will dive into the Convolutional Neural Networks, in particular analyzing why they are so important in the field of computer vision and image processing and what is the added value they bring over the multi-layer perceptron.
The ability of multi-layer networks trained with gradient descent to learn complex, high-dimensional, non-linear mappings from large collections of examples makes them obvious candidates for image recognition tasks but there are problems.
First of all, in order to feed an image to a neural network we need the number of units of the input layer must be equal to the number of image pixels. Thus, fully-connect the input layer with the adjacent hidden layer requires about several tens of thousand of connections, and

so weight parameters. While the network capacity grows, the hardware computation dramatically blows up and the number of training examples also should be increased (the network needs to learn more information).
Another important limitation of fully-connected networks is that they do not provide built-in invariance over the feature extraction phase. The main challenge in image classification, OCR and face recognition is to detect object which can appear with different position, scale level, dimension and quality. Furthermore, since an object is represented by a subset of adjacent pixel, the neural network should take care about the topology of the input and the spatial features correlation.
The solution provided from the Convolutional Neural Network to handle invariance and locality properties consists in sharing the weight across the 2D space. This reduce drastically the number of independent parameters to tune during the training.
So in conclusion, the usage of fully-connected architectures is reasonable for those cases in which the order of the inputs is irrelevant for estimating the output. Indeed, the possible correlations between the inputs can be seen under two aspects: space and time.
Video processing is a typical context in which both should be considered. In that case, for example, a CNN can address the spatial aspect about the single frames content while Recurrent Neural Networks (like LSTM [18] and GRU [19]) can be used to consider the temporal sequence of frames.

3.2 An uniform representation for heterogeneous data

Tabular data, images, text, video, audio and in general every kind of data can be stored in the disk in a specific format. However it is absolutely out of the question the possibility that a neural network takes in input these files directly as they are.
Beyond the consideration about input preprocessing done in the 2.2, each programming language or deep learning framework requires an input representation independent from the nature of the data.
In general, each training examples is always represented as a numeri-

cal vector or a matrix. These concept are generalized to higher-order matrices, or more commonly **tensors**.
The tensor representation of an image is quite intuitive: two dimensions are associated to the width and the height while the third one depends from the specific color notation. Generally is RGB or BGR, which indicates the red, green and blue channels.
In synthesis a color image is converted in an order 3 tensor of [0 to 255] values. In case of greyscale image the tensor degenerates in an order 2 tensor, or in other terms a simple matrix.
Less obvious, instead, is how represent other kind of data always in a numerical tensor format. Think for example about text data and in particular the concept of word. In such case indeed, the representations should be able to express even more sophisticated kind of correlations, like semantics and syntactic, and consider also the context in which the words occur [20]. Some works are focused in dealing text data as one-dimensional raw signal, just like occurs for image channels. In the paper [21], for example, the authors have been employed Deep Convolutional Neural Network on a sequence of characters. The main advantages of this approach are that don't require the knowledge about the syntactic or semantic structure of a language, works independently from the specific language and may naturally learn abnormal character combinations, such as misspellings and emoticons.
The encoding process, also known as character quantization, covers an alphabet of 70 characters (26 letters, 10 digits, 33 other chars and the new line):

```
a b c d e f g h i j k l m n o p q r s t u v w x y z
0 1 2 3 4 5 6 7 8 9 - , ; . ! ? : ' " / \
| _ @ # $ % ^ & * ~ ` + - = < > ( ) [ ] { }
```

Here, the character quantization has been performed via one-hot vectors, the simplest raw technique to transform categorical data to binary values vectors where vectors has all the elements set to zero except for one.
However there are many obvious reasons to avoid the usage of one-hot encoding and choose more advanced techniques such as word embeddings.

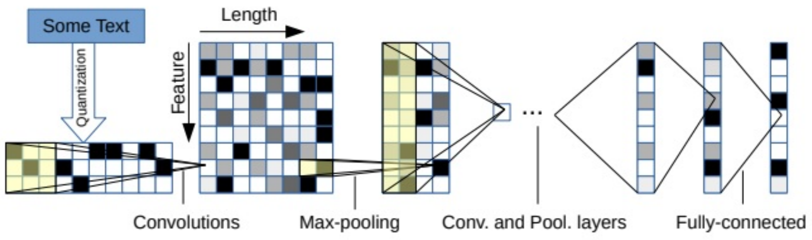


Figure 3.1: Deep ConvNet Character-level for text classification

3.3 CNN structure

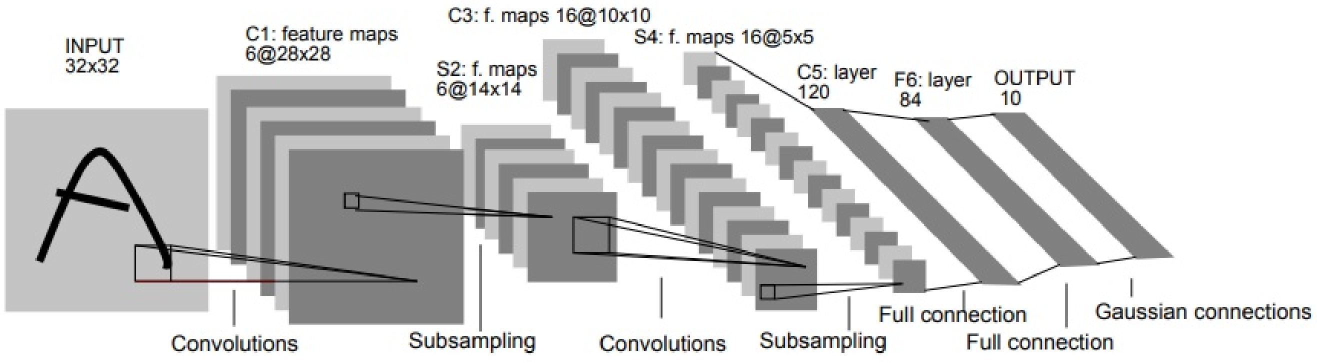


Figure 3.2: LeNet-5 CNN architecture

The godfather of **Convolutional Neural Network (CNN)** is Yann LeCun who in 1998 introduced the first prototype named LeNet-5 for digits recognition. Yann LeCun is currently the director of the Facebook AI Research Departement, in New York [22].
From 1998 to nowadays, several model has been proposed such as LeNet, AlexNet, ZFNet, GoogLeNet, VGGNet. One of the most impressive and cutting-edge model is probably the Inception-V3 [23].

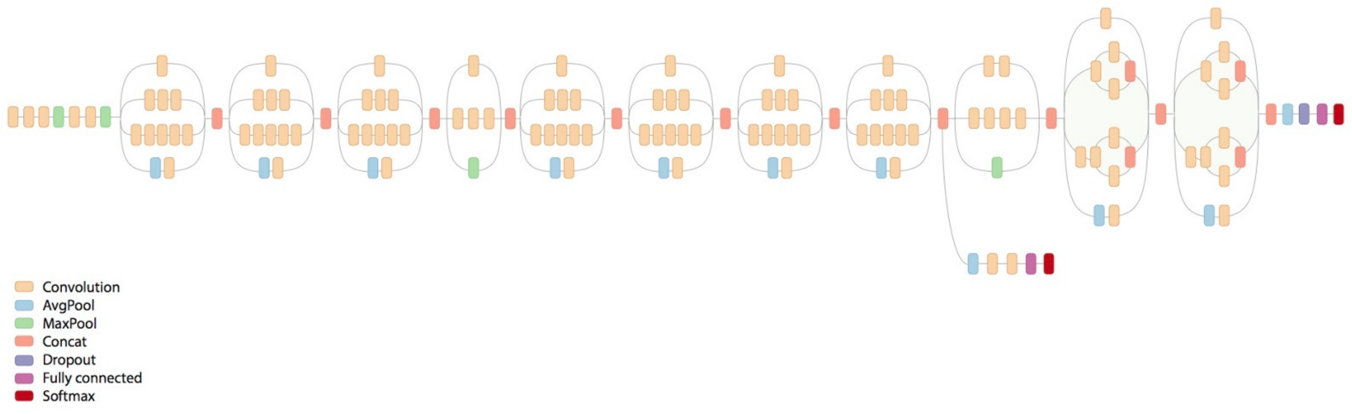


Figure 3.3: Inception-V3 CNN architecture

By observing the structure of the most famous CNNs, it's possible to identify 4 main categories of layer that are always employed for different purposes:

- **Convolutional Layers**
- **Pooling Layers**
- **Flattening Layers**
- **Fully-Connected Layers**

3.3.1 Convolutional Layer

Convolutional layers represent the core of this kind of networks.
A convolutional operation can be expressed in pure mathematical terms as:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)\, g(t - \tau)\, d\tau$$

This formula is very common in digital signal processing where the integration concerns the continuous time domain. In our case the image represents a discrete signal of pixel over the 2D space.

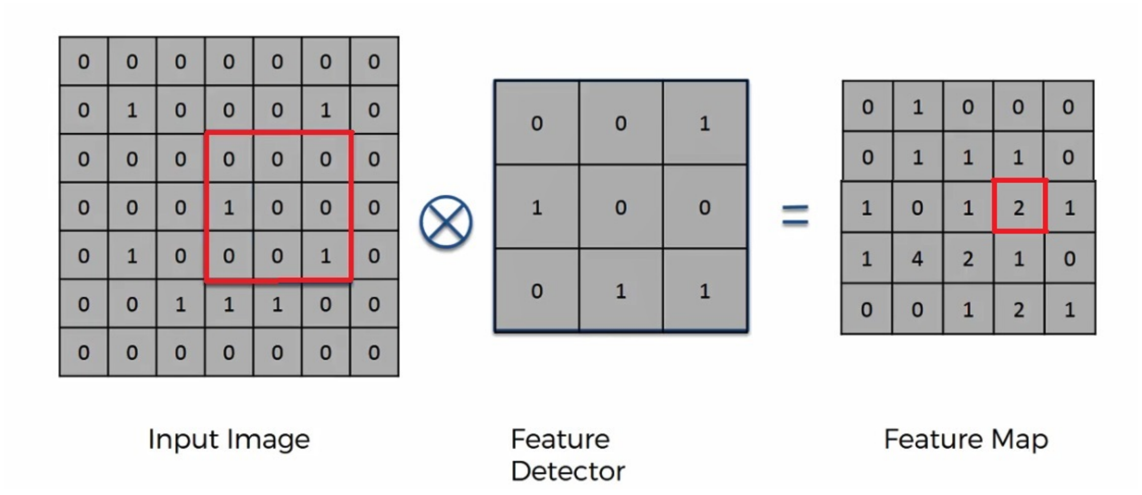


Figure 3.4: Convolution in CNN
(credits: https://www.superdatascience.com/computer-vision/)

The picture 3.4 shows a snapshot of a convolutional operation. During the execution, a **feature detector** (also known as filter or kernel) performs a scan through the whole input image producing what is called a **feature map** (also known as convolved feature or activation map).
The feature map is the result of an element-wise matrix product and as you can see the dimension of the input image has been reduced. The reduction rate depends from a parameter called **stride** which defines how many pixels the kernel has to be moved from the current position at each step (in the example the stride is set to one).
The benefits of convolution are to implicitly emphasize certain feature meanwhile discarding the worthless and also data reduction implies less computation.
This process resembles in a much more similar way the human brain behaviour, indeed when we look an image we don't focus on the single pixels but in a portion of pixel, an object or a set of features.
In practical, the convolutional process is even more widespread be-

cause the feature detector is typically not just one, like in the example, but are several and each one is applied on the input image and so produces a different feature map.
The values of the kernel matrix are not randomly. They are studied and used by almost all the image processing applications to apply some specific filtering techniques: blur, Gaussian blur, motion blur, find edges, sharpen, emboss, mean and median filter and so on [24].
Finally, since the images are often characterized by high non-linearity, especially on the border lines between distinct objects, it is a common practice to apply the ReLU activation function to the generated feature maps in order to break those linearities [25].
In the 2.4 we briefly discussed about Softplus activation function as an alternative of ReLU but we have seen that in the experimental context Softplus brings no added value. ReLU is still the best choice but other variants have been proposed in order to get even better accuracy.
The [26] investigates successfully about the usage of the PReLU, a revisited ReLU with non-constant negative part where the coefficient a is adaptively learned.

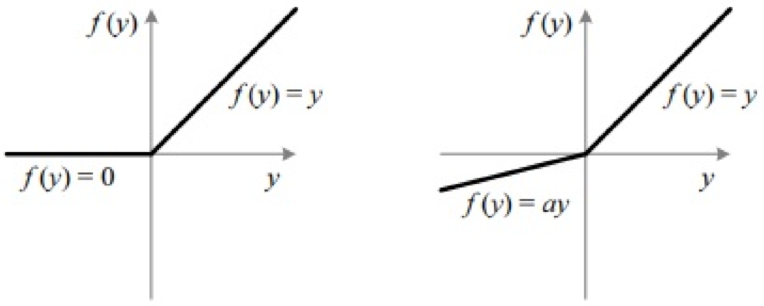


Figure 3.5: ReLU Vs. PReLU

In practical applications we often can see PReLU with the a small and fixed coefficient a. This kind of activation function is commonly known as Leaky ReLU.

3.3.2 Pooling Layer

The first stage of a deep CNN is typically characterized by a series of alternated convolutional and pooling layers.
Pooling operation works likewise convolution but with some important difference. The main strength is to injects to the neural network the spacial invariance property.

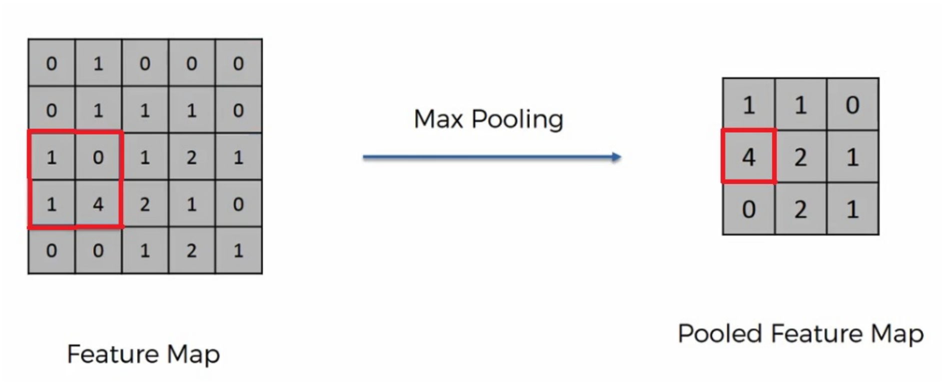


Figure 3.6: Max Pooling in CNN
(credits: https://www.superdatascience.com/computer-vision/)

The operation is performed on the feature maps produced after the convolution and, as for convolution, the input is scanned from top-left to bottom-right by a square window that aggregate a set of pixel into one. Again, the shift is regulated with a stride parameter and the resulting feature map is reduced in favour of computation speed.
By contrast, now we don't have to deal with feature detector and the mapping function is determined by the specific kind of pooling operation used (the most common one is Max Pooling which filter out only the maximum pixel value inside the window, as illustrated in Figure 3.6).
Intuitively, the information lost after the reduction are the very ones which keep track of the spacial variance and potential noises, enabling to focus only on the necessary features, therefore preventing overfitting and leading to generalization.
Regarding the possible pooling operations, there are obviously some alternative proposed over the Max Pooling. The reason why we high-

lighted Max Pooling is not causal, indeed researches [27] demonstrate Max Pooling is vastly superior for capturing invariances in image-like data, compared to other operation like Subsampling, which instead takes care of all pixel by averaging them (Average Pooling).

3.3.3 Flattening Layer

After getting through the sequence of convolution and pooling transformations, the input is ready to be classified.
For this purpose, a classical CNN always ends with a multi-layer perceptron composed by one or more fully-connected layers.
To make a bridge between the two stages and so feed the first fully-connected layer, the feature maps have to be converted into one dimensional vectors.

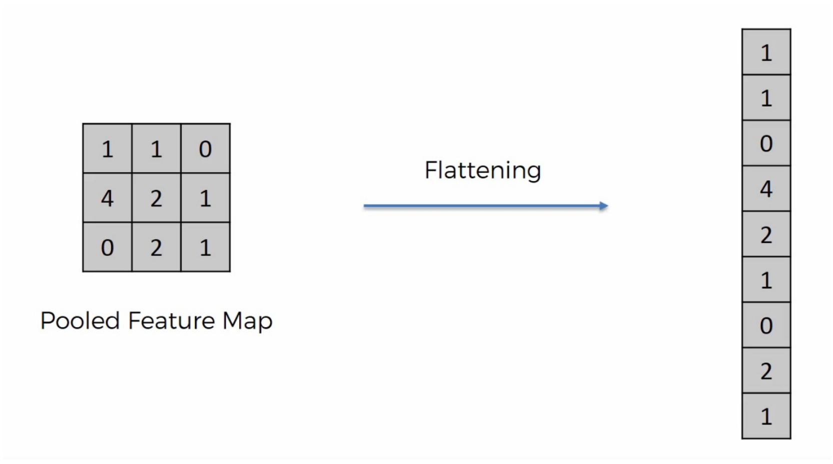


Figure 3.7: Flattening in CNN
(credits: https://www.superdatascience.com/computer-vision/)

This consist of concatenating the matrix rows one after another and finally transposing everything in a column. As we can see, the operation is pretty straightforward and doesn't need any further explanation.

3.3.4 Fully-Connected Layers

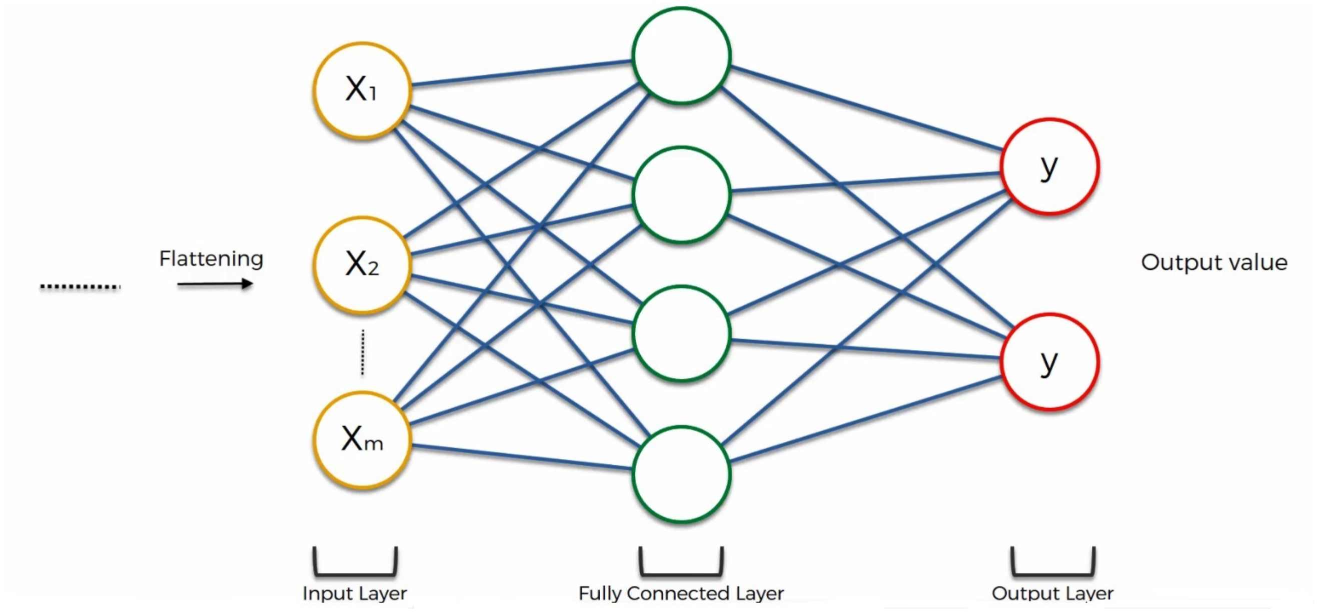


Figure 3.8: Fully-Connected part of CNN
(credits: https://www.superdatascience.com/computer-vision/)

The last part of a CNN resembles most of the concepts already seen in Chapter 2.
The objective is to reuse all the feature maps produced at the end of the first stage and translate the information content into a classification. The final classification should fire the neuron associated to the detected object.
Assuming we are in a supervised learning context, the classification is not immediate but may require a proper training with several forward passes and backpropragation.
The Figure 3.4, 3.6, 3.7 and 3.8 are too simplistic. Real models normally consists in several millions of parameter and require high performing GPU in order to accomplish the training in a reasonable time. To make an example, let's take a look to the ImageNet Challenge 2014 winning models, the deep CNNs VGG16 and VGG19 [28]. For the classification task, images must be classified into one of 1000 different

categories.
The VGG16 for example has a total of 138 millions of parameters. Focusing only to the last 3 fully-connected:

Layer	Number of Weights	Number of Biases
fc1	$(512 \times 7 \times 7) \times 4096$	4096
fc2	4096×4096	4096
fc3	4096×1000	1000

Take also in consideration that VGG state of the art has been nowadays overcame.

3.4 Autopsy of a CNN

This section is dedicated to a very amazing tool which is not only funny to explore but to comprehend what happens under the hood of a CNN, have a clear view of each layer, visualize in details every single feature map and so on.
The name of this application is "3D Visualization of a Convolutional Neural Network", developed by Adam Harley [29] and available at `http://scs.ryerson.ca/~aharley/vis/conv/`.
Visualizing a CNN is important not only for having a detailed understanding about how the model behaves but also to easily experiment with the input and observe the result of the experimentation immediately, underlying the strengths and the weaknesses. Furthermore, this tool provide a full view about the hierarchical abstraction layer-by-layer of the input till the output.
The CNN employed is trained on an augmented version of the MNIST dataset which consist of classify 28×28 pixel images of handwritten digits.
The structure resembles exactly the sequence of layers described in the previous section, indeed the input is processed through two couples of convolutional-downsampling layers followed by two fully-connected. Hence, the flattening layer is implicit.
The interactive functionality allows to explore every single layer, feature map and unit. For each unit is possible to check its weighted

input, the activation function applied and the relative output.
The user has simply to draw a 0 to 9 number into the canvas and the classification is automatically performed. The correct prediction appears as “first guess” and, since the output is a distribution of probabilities across the overall possible ten digits, it’s also possible to check the “second guess”.
We conclude the section with two screenshots and let the reader enjoy the tool by following the link indicated at the beginning.

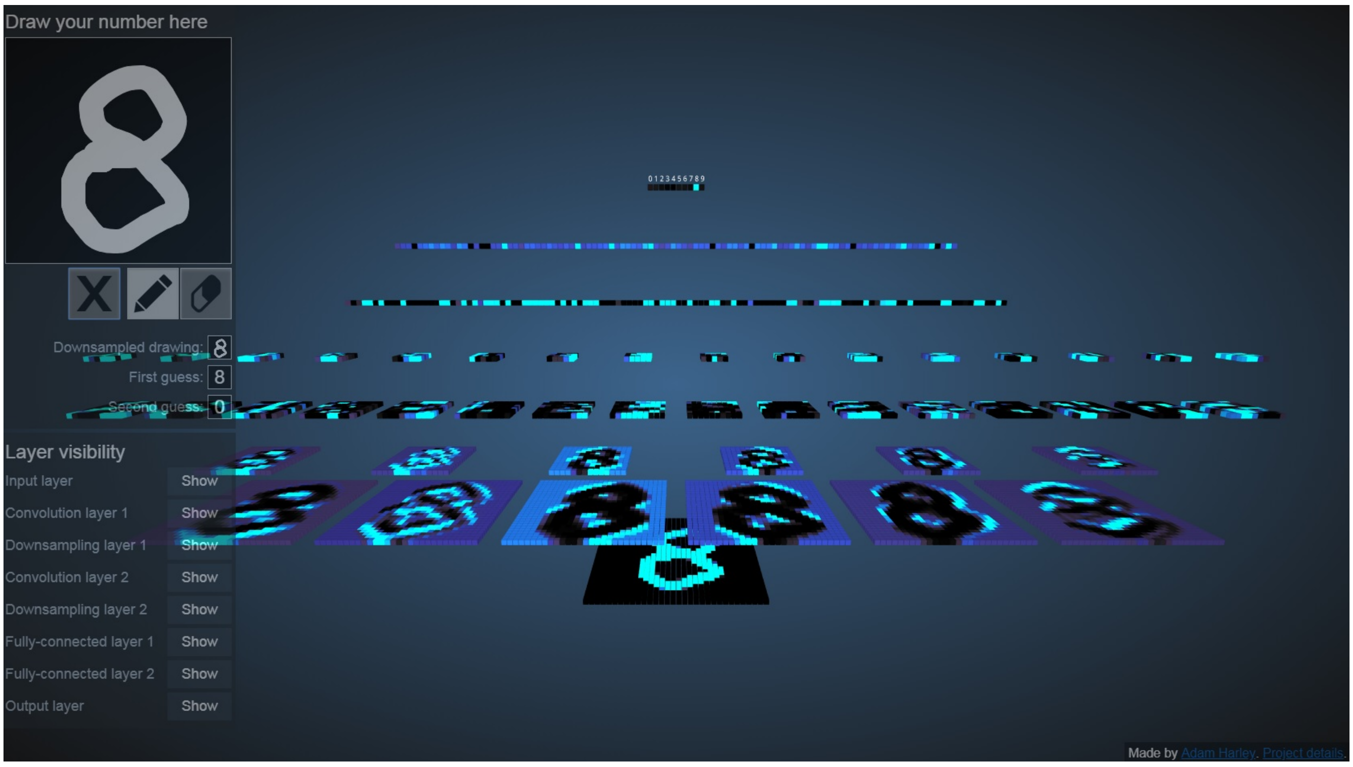
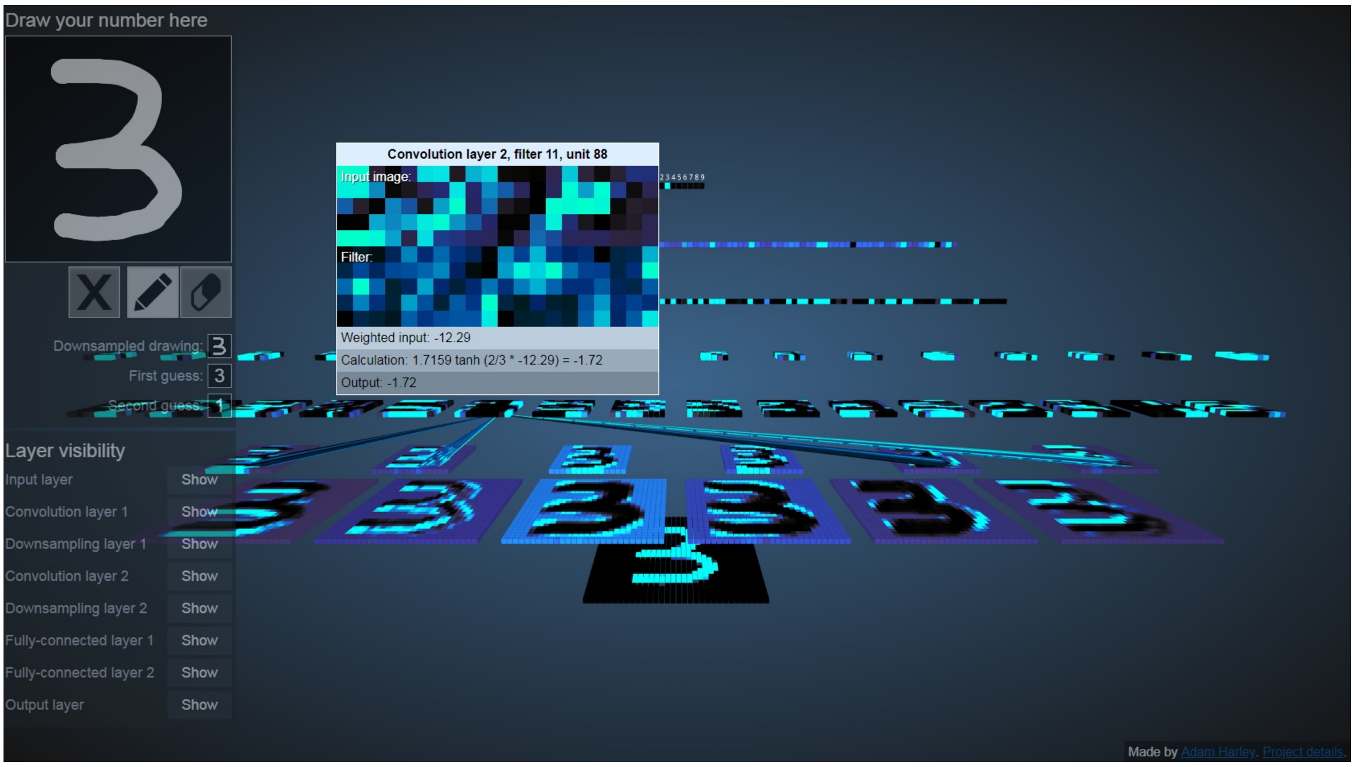


Figure 3.9: 3D Visualization of a Convolutional Neural Network - part 1



Convolution layer 2, filter 11, unit 88

Weighted input: -12.29

Calculation: 1.7159 tanh (2/3 * -12.29) = -1.72

Output: -1.72

Figure 3.10: 3D Visualization of a Convolutional Neural Network - part 2

Chapter 4

Generative Adversarial Networks

4.1 The artificial artist

Generative Adversarial Networks first appeared in 2014 as a revolutionary idea of the Google Brain researcher Ian J. Goodfellow [30].
Before going into the details of the GAN architecture, it could be useful to understand what is the main idea behind this networks, which are the expectations, which are the most impressing results obtained so far and why GANs are also recognized as "'artificial artists".
The first positive feedback come from the illustrious opinion of Yann LeCun, who referred to GANs as *"the most interesting idea in the last 10 years in Machine Learning* [31]".
The capacity of a GAN is to be able to generate new "never seen" data relying to the distribution of real world data. This potential is not restricted only to the image domain but also text, music, video and so on.
From a philosophical point of view, comparing a neural network with an artist can be consider inappropriate and a terminological abuse. An human being possesses native skills that are not provided to a machine, such as intuition, creativity, imagination and consciousness.
All these qualities lead to non-deterministic behaviours which is the main prerogative of an artist.
Modern processors, which we use to train such neural networks, are

built on the basis of the Von Neumann architecture and on these processors, the training processes run as an algorithm. Computational theory refers to algorithm as a sequence of imperative instructions which leads to a deterministic result.
Under these assumptions, a GAN is far away from being consider as an artist in the true sense of the term. However, this adjective still widely used because of the outcomes of GAN, which are often really impressive.

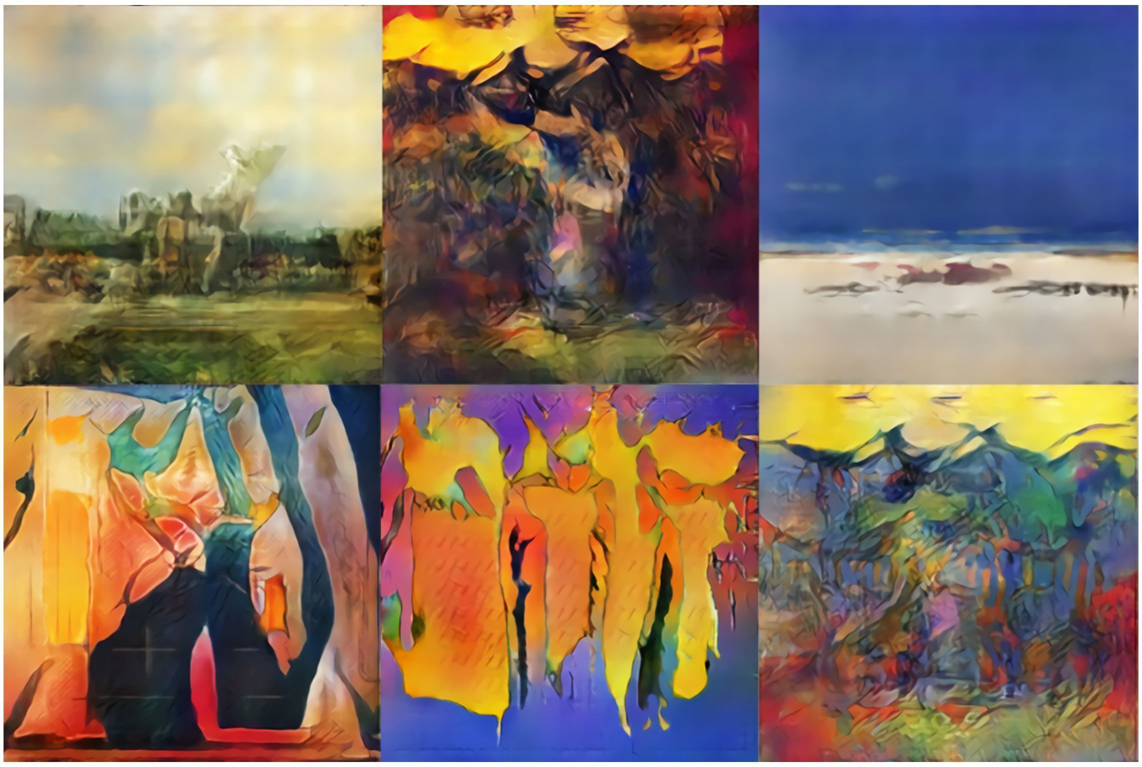


Figure 4.1: Example of images generated via GANs

The New York Times in 2017 published an article titled “How A.I. Is Creating Building Blocks to Reshape Music and Art” [32] to present the world the work of some Google researches about GANs.

Figure 4.2: A GAN that generate imagescapes from existing photography
(credits: DeepDream - https://deepdreamgenerator.com)

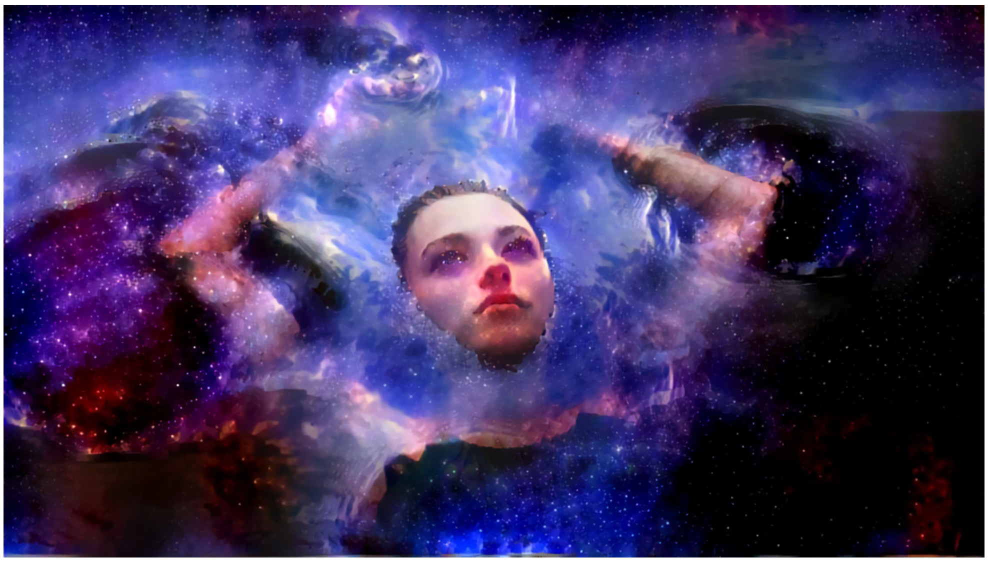


Figure 4.3: An artwork created using DeepDream
(credits: DeepDream - https://deepdreamgenerator.com)

A very interesting and recent anecdote that involves GAN occurred on 25th October 2018.
On 23th October, the network media The Verge announced [33]: Christie's, an auction house with its headquarter in Paris, conducted a very unusual sale. As part of a three-day Prints & Multiples event, it's auctioning off the *Portrait of Edmond Belamy*, a canvas in a gold frame that shows the smudged figure of what looks like an 18th century gentleman. It's expected to fetch a modest price, somewhere between \$7,000 and \$10,000, but the artwork's distinguishing feature is that it was "created by an artificial intelligence," says Christie's. "And when it goes under the hammer, [it] will signal the arrival of AI art on the world auction stage."

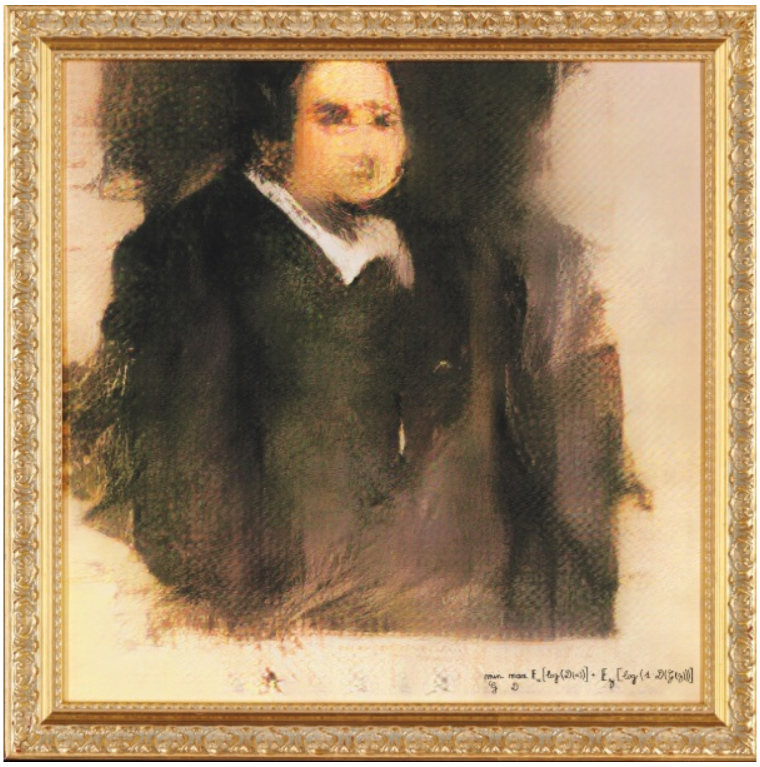


Figure 4.4: Portrait of Edmond Belamy, 2018, created by GAN

On 25th October, after the event, it came up the final sentence [34]: Christie's sells its first AI portrait for \$432,500, beating estimates of

$10,000. The image was created using a machine learning algorithm that scanned historical artwork.
The artwork was created by a collective named Obvious. The three members of Obvious, a trio of 25-year-old French students, used a GAN to create the picture [35]. The network was trained on a dataset of historical portraits, and then it tried to create one of its own. Obvious printed the image, framed it, and signed it with the objective function used by the algorithm (as you can see in the botton-right corner of the picture).
This event is destined to lights up even more the debate around the concept of art in artificial intelligence.

4.2 The GAN system

4.2.1 Structure

The domain model is composed by two main actors: a **Generative Model (G)** and a **Discriminative Model (D)**.
Alongside the generator, the system needs to be supported with a large database of real world images.

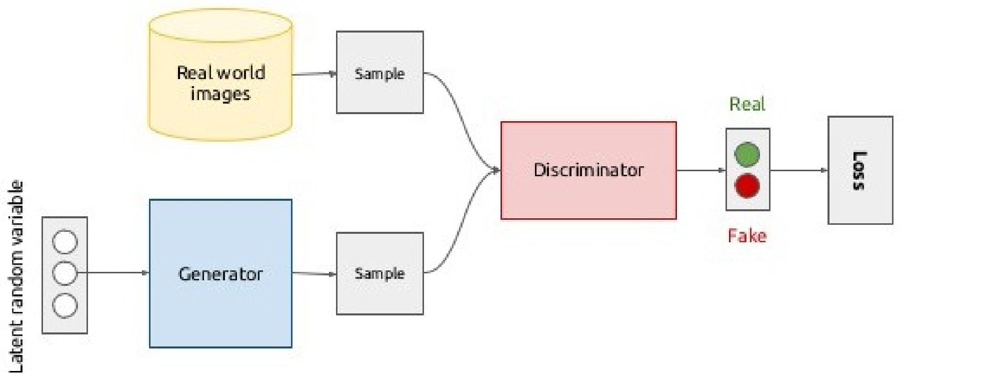


Figure 4.5: GAN conceptual architecture (https://hackernoon.com/how-do-gans-intuitively-work-2dda07f247a1)

Let's start visualizing the structure of the individuals components.
Both the Generator and Discriminator are Deep Convolutional Neural Networks. The duality G-D holds an important symmetric property, indeed, they are mirror images of each other. In particular, the Generator is better recognized as a Deconvolutional Neural Network.
A Deconvolutional Neural Network is nothing else than a reverse CNN in which the layers are also called transposed convolutional layers.

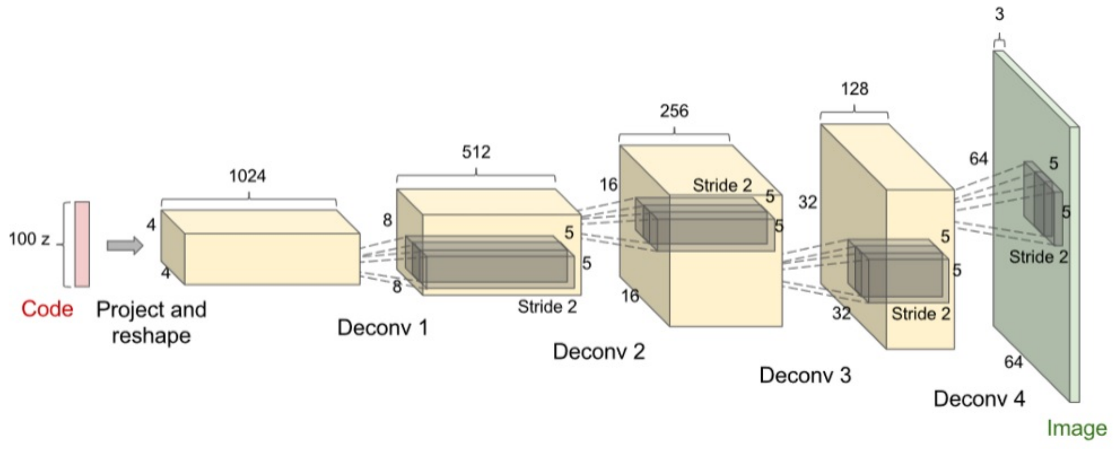


Figure 4.6: A DCGAN generator (https://hackernoon.com/how-do-gans-intuitively-work-2dda07f247a1)

The Discrimination, as mentioned before, is the convolutional counterpart of the Generator.

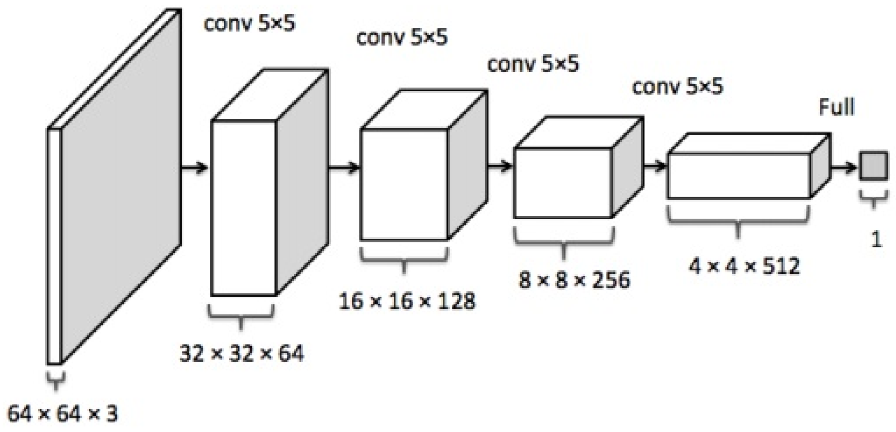


Figure 4.7: A DCGAN discriminator (https://hackernoon.com/how-do-gans-intuitively-work-2dda07f247a1)

4.2.2 Interaction

The “fight” between G and D represents the classical zero-sum game. The exact definition of zero-sum game from Wikipedia is:

“In game theory and economic theory, a zero-sum game is a mathematical representation of a situation in which each participant’s gain or loss of utility is exactly balanced by the losses or gains of the utility of the other participants. If the total gains of the participants are added up and the total losses are subtracted, they will sum to zero. [36]”

In the specific case of GANs, the gain and loss correspond exactly to the cost funtions of the two partecipants.
Generator and Discriminator are constantly in touch and work in tandem. The first one (as the name should suggest) is responsible of generating a stream of data, such as images.
These images are then submitted one by one to the Discriminator which basically has to answer to the following question: how much is this image likely to be similar to what we can see in the real world?

In short, is this real or a fake?
In order to understand what is real, the Discrimination needs to be aware of the concept of reality. That's why it must be provided a dataset of real images alongside the Generator.
For example, let's suppose our target is dogs. The dataset should contain a large collection of photos of dogs of different races, in different positions, different prospective, colors and so on... and the Generator should produce new images that look as much as possible like dogs.
While the Discriminator get the data from both sources, it has to label each image as **fake** (coming from the Generator) or **real** (coming from the dataset).
This classification, whatever is right or wrong, always favours the rise of learning. Intuitively, every time the Discriminator guesses the prediction the Generator knows that must improve itself, so the next time has more chance to successfully fool his opponent. Alternatively, it is Discriminator that needs to improve itself.
[NB: It must be emphasised that in this context, the verb "improve" means backpropagation and parameter tuning.]
The whole interaction can also be modelled with the Actor-Critic paradigm [37], as well as for Q-Learning in Reinforcement Learning[38]. Actor-Critic and GANs work in a very similar way from the prospective of the parallel training (which in GAN involves both G and D). Training a GAN is indeed a very difficult task which needs to consider even more aspects compared to the traditional learning practices discussed in Chapter 2.
The reason is pretty obvious; before we had single models with their own private loss function, while now we have two which moreover they need to be synchronized in order to accomplish the zero-sum game. A lack of synchronization may degenerate in the mode collapse scenario, discussed in the 4.3.
The main similarity between AC and GAN concerns the information flow. One model produces data and pass it to the other model which perform an evaluation.

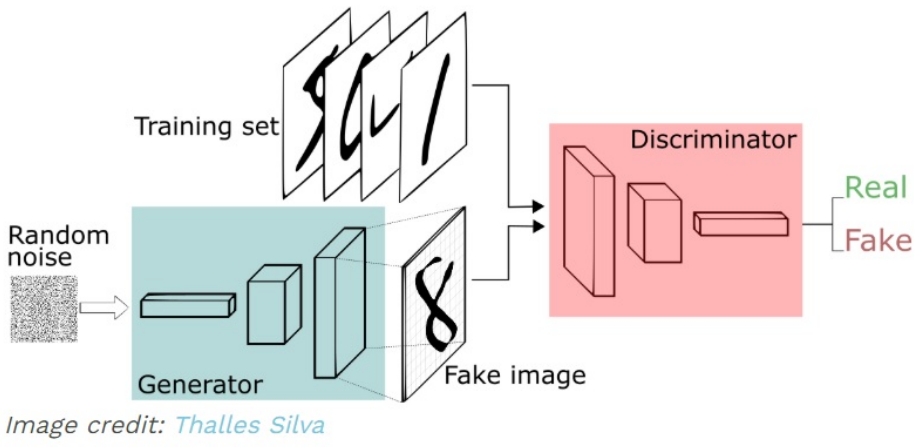


Figure 4.8: GAN components interaction with the MNIST dataset

In order to better understand the idea behind the scenario is good to make an analogy with a real world example. Quoting the words of Ian J. Goodfellow:

"The generative model can be thought of as analogous to a team of counterfeiters, trying to produce fake currency and use it without detection, while the discriminative model is analogous to the police, trying to detect the counterfeit currency."

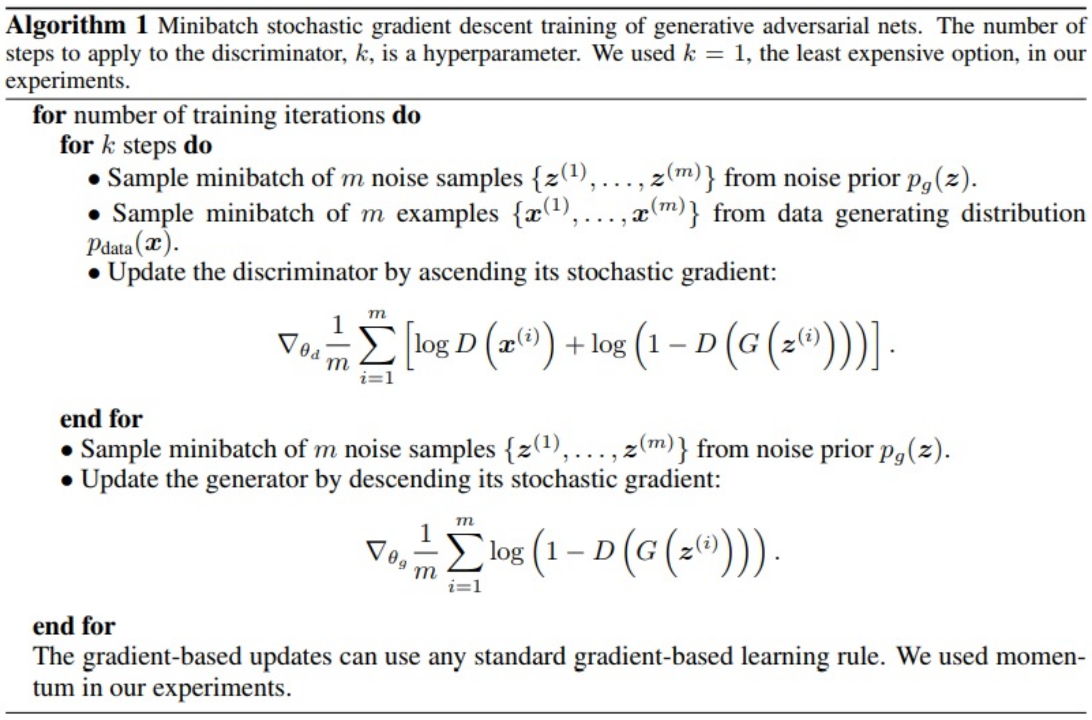
4.2.3 Behaviour

Let's describe in more depth the algorithm governing the GAN framework.
The minimax game is represented by the following objective function:

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

The Generator G is associated to a distribution p_g over data x and its input noise vector is $p_z(z)$. $D(x)$ defines the classification made by the Discriminator, a floating value between 0 and 1 which measures the probability that x came from the real dataset rather than p_g.
The reason of $\max_D$ is straightforward, we have to train D to maximize

the probability of assigning the correct label to both training examples and samples from G.
Since we are playing to the zero-sum, we expect a counterpart to minimize and that's $\min_G$, which is expressed as $\log(1 - D(G(z)))$.
In the objective function $\mathbb{E}$ indicates the binary cross-entropy loss function, so that the overall goal is to maximize the cross-entropy of the Discriminator and minimize the cross-entropy of the Generation.
In his paper, Goodfellow provides a pseudo-algorithm to describe the training process.



Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

```
for number of training iterations do
    for $k$ steps do
        • Sample minibatch of $m$ noise samples $\{z^{(1)}, \dots, z^{(m)}\}$ from noise prior $p_g(z)$.
        • Sample minibatch of $m$ examples $\{x^{(1)}, \dots, x^{(m)}\}$ from data generating distribution $p_{\text{data}}(x)$.
        • Update the discriminator by ascending its stochastic gradient:
            $\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[ \log D\left(x^{(i)}\right) + \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right) \right].$
    end for
    • Sample minibatch of $m$ noise samples $\{z^{(1)}, \dots, z^{(m)}\}$ from noise prior $p_g(z)$.
    • Update the generator by descending its stochastic gradient:
        $\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right).$
end for
The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.
```

Figure 4.9: GAN pseudo-algorithm

4.3 Mode collapse: The Helvetica Scenario

As mentioned before, the process of training a GAN is nowadays subject of many debates and researches.
In particular the hardest part is related to how to train the generative model than the discriminative.

It is easier to recognize a Monet's painting than drawing one [39]

In order to avoid unpleasant effects, during the whole training it is fundamental to keep a proper synchronization between the two actors.
The image below illustrates the backpropragation procedure on both the discriminator and the generator.

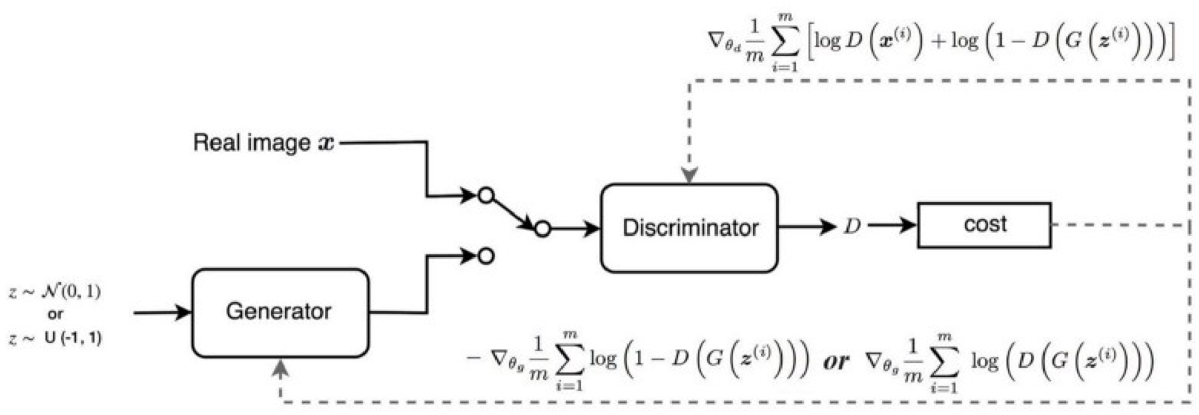

$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(x^{(i)}\right) + \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right)\right]$
Real image x
Discriminator
D
cost
$z \sim \mathcal{N}(0,1)$
or
$z \sim$ U (-1, 1)
Generator
$-\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right)$ or $\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(D\left(G\left(z^{(i)}\right)\right)\right)$

Figure 4.10: Discriminator and generator training via gradient descent

Due to these formulas, the main causes which can lead to bad convergence of the model are:

- **Vanishing gradient:** the discriminator improves too fast respect to the generator. Therefore it starts to guess all the prediction so that the generator gradient dramatically fades out causing the impossibility of learning.

- **Non-convergence:** Highlighted in research paper by Arjovsky [40], he attempted to solve the vanishing gradient problem including a reverse KL-divergence term to the original cost function. During his experiments he tried to freeze the generator and train the discriminator from scratch, obtaining a fluctuating gradient and so an unstable model.

- **Mode collapse**

One of the most critical event to solve in GAN is known as **mode collapse**.
Inspired by the parodistic 2002 science tv show, Look Around You, Goodfellow compared the mode collapse in GAN as an equivalent of the Helvetica Scenario.
In the first episode of the serie, the Helvetica Scenario is descibed as an hypothetical phenomena which should be triggered by altering the nucleus of a particle of calcium, leaving it in an unstable state and causing an apocalyptic chain reaction [41].
In the GAN context, we refer to mode collapse the situation in which the generator starts to produce always the same output and so cheating the discriminator forever. Thus, the overall training has to be consider a flop, since the generator fails to learn the real-world data distribution and gets stuck in a small space with extremely low variety.
A practical example can be observed in the experimental section of the paper "Unrolled GAN" [42], where the author compared the result of their model, applied on the MNIST dataset, with the same result obtained using a standard GAN. This latter case shows a situation of "Helvetica Scenario" in which the model collapse generating continuously the same digit, an handwritten six.

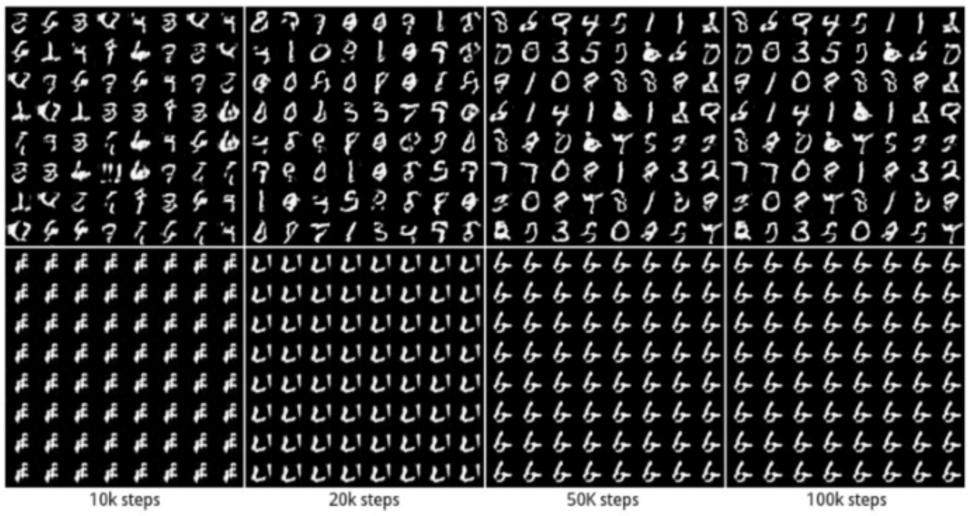


Figure 4.11: GAN mode collapse with MNIST dataset

Mathematically speaking, mode collapse occurs when the model reaches a spacial condition called Local Nash Equilibrium (LNE) associated with a sub-optimal generative performance [43].
The objective of the training is instead to converge to a Global Nash Equilibrium (GNE), which often can be arbitrary far from the obtained LNE.
Quite solution has been proposed in literature to prevent as much as possible the risk of mode collapse, such as GANGs (Generative Adversarial Network Games) [44] and Coulomb GANs [45].
The first starts by formalizes the adversarial networks in a finite games setting, finite GANGs. This kind of solution theoretically do not suffer from LNE but in practice it is computationally intractable since it intends to analyze all the possible candidate strategies which end in a LNE. Therefore the authors came up with a solution based on Resource-Bounded Best-Responses (RBBR) that looks for resource-bounded NE (RB-NE), in particular it focuses to a subset of strategies and using an heuristic search it try to deliver to new promising strategies expanding the research space.
Regarding Coulomb GANs, the idea is to model the learning problem as a potential field in which the generated examples repel each other in order to avoid output replication.

The inspiration comes from the electrostatic physics where the electrical charges situated in the space generate each one an electric field which affects the whole system. In GAN, the possible generations are the equivalent of the charges and to maintain a situation of global equilibrium (in this case a Global Nash Equilibrium) should be disposed in a proper configuration keeping proper distances.

Chapter 5

Practical applications of GANs

Several applications originally developed in research context have nowadays found a place in the world as dominant technologies in favour of the human progress, providing also a breakthrough in term business and marketing. It's the case for instance of the World Wide Web developed by Tim Barners-Lee in the laboratories of CERN and today representing the greatest means of communication of ever.
The question we intend to give an answer in this chapter is: what the future holds for GANs? Can GANs be used to bring an added value to the Artificial Intelligence?
If we try to sell to companies a product presenting it as a random image generator it would be quite difficult to find buyers willing to invest on it.
In this respect, recent papers propose very interesting solutions which involve the usage of GANs, in the hope that such ideas may help to move towards a business direction.
We present four practical application of GANs analyzing aspects, implementations, strengths, weaknesses.

5.1 Text to image generation

Although most of the works around GANs are focused on the image domain, we mentioned that neural network can handle even different kind of data under a proper tensor representation.
The paper "Generative Adversarial Text to Image Synthesis" [46] proposes a novel solution in order to translate text in the form of single-sentence human-written descriptions directly into image pixels.
The approach described take into account two main sub-tasks. The first one is focused on learning the text features and the second aims to properly use these features to generate an images which can be able to fool the discriminator.
Connecting the two tasks is not straightforward for the reason that the relation between the text description and the relative image representation is typically multimodal. This means it doesn't exist a one-to-one relation among the two domains. As long as there can be different descriptions for one image, at the same time there can be several pixel configurations for one textual description. This property can be exploited by a GAN-based solution providing the discriminator with a "smart" adaptive loss function.
The problem of learning text representation is addressed through a combination between a character-level CNN (a solution similar to the one already mentioned in the 3.2) and a recurrent neural network (see LSTM for instance).
In such way, the generator is fed with a embedding encoded representation of the natural language description alongside a random sample from the noise vector. This input passes along the deconvolutional layers of the network and output an image. The generator process can be denoted as $G : \mathbb{R}^Z \times \mathbb{R}^T \rightarrow \mathbb{R}^D$ where T, D and Z are respectively the dimension of the text, image and noise vectors.
It's important to point out that the same text vector associated to different noise samples will lead to different output images, therefore encouraging the generator to model the multimodal property mentioned before.
Discriminator side we have a convolutional network (whose structure reflects symmetrically the generator) denoted as $D : \mathbb{R}^D \times \mathbb{R}^T \rightarrow \{0, 1\}$. It basically receives the images produced by the generator supported

with the original text description and, as well as every GAN discriminator does, it sentences how much the input seems real or fake, expressing the classification with a floating value between 0 and 1.

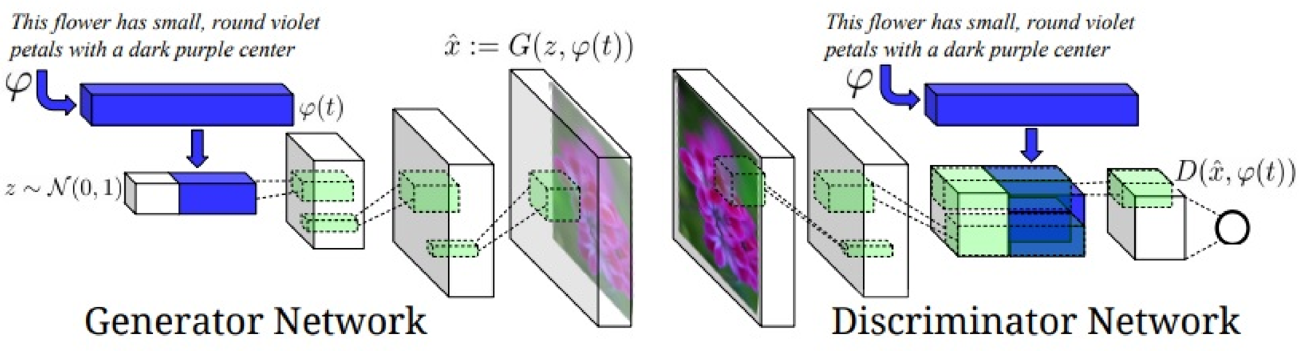
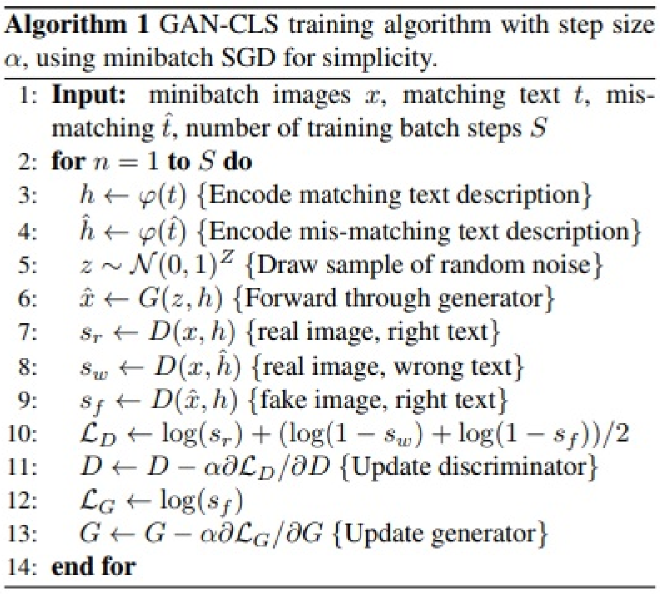

This flower has small, round violet petals with a dark purple center
φ
$\varphi(t)$
$z \sim \mathcal{N}(0,1)$
$\hat{x} := G(z, \varphi(t))$
This flower has small, round violet petals with a dark purple center
φ
$D(\hat{x}, \varphi(t))$
Generator Network
Discriminator Network

Figure 5.1: GAN architecture for text to image synthesis

From the previous explanation and by looking at the Figure 5.1 you may notice something missing. It is the dataset of real images. In the experimental part have been used two dataset: the Oxford-102 [47] (8189 images of flowers of 102 categories) and the CUB [48] (11788 images of birds of 200 different categories).
Now, how these dataset have been employed to train the GAN? The solution proposes two main approaches.
The first is called Matching-aware discriminator (GAN-CLS), where the discriminator receives three possible combination of image-text pairs: a real image coming from the datasets with its correct description, a real image with a mis-matching description, and a fake image from the generator with its correct description.
After these information it's the case to introduce the pseudo-algorithm which defines the training process:



Algorithm 1 GAN-CLS training algorithm with step size α, using minibatch SGD for simplicity.

```
 1: Input: minibatch images $x$, matching text $t$, mis-matching $\hat{t}$, number of training batch steps $S$
 2: for $n = 1$ to $S$ do
 3:     $h \leftarrow \varphi(t)$ {Encode matching text description}
 4:     $\hat{h} \leftarrow \varphi(\hat{t})$ {Encode mis-matching text description}
 5:     $z \sim \mathcal{N}(0, 1)^Z$ {Draw sample of random noise}
 6:     $\hat{x} \leftarrow G(z, h)$ {Forward through generator}
 7:     $s_r \leftarrow D(x, h)$ {real image, right text}
 8:     $s_w \leftarrow D(x, \hat{h})$ {real image, wrong text}
 9:     $s_f \leftarrow D(\hat{x}, h)$ {fake image, right text}
10:     $\mathcal{L}_D \leftarrow \log(s_r) + (\log(1 - s_w) + \log(1 - s_f))/2$
11:     $D \leftarrow D - \alpha \partial \mathcal{L}_D / \partial D$ {Update discriminator}
12:     $\mathcal{L}_G \leftarrow \log(s_f)$
13:     $G \leftarrow G - \alpha \partial \mathcal{L}_G / \partial G$ {Update generator}
14: end for
```

Figure 5.2: GAN-CLS training algorithm for text to image synthesis

An alternative contribution to GAN-CLS is supported by GAN-INT, which consists in a text embedding data augmentation obtained by interpolating training set captions in the embedding space.
The implementation of GAN-INT is made by altering the generator objective with an additional term. Recalling the minimax function proposed by Goodfellow in the original GAN paper, the generator now assumes the following form:

$$\mathbb{E}_{t_1, t_2 \sim p_{data}}[\log(1 - D(G(z, \beta t_1 + (1 - \beta) t_2)))]$$

where t_1 and t_2 are the two text embedding to interpolate and β is the interpolation coefficient, which experiments recommend to set to 0.5 .
We leave the reader some images of qualitative results on both datasets, birds and flowers, with a comparison between basic GAN, GAN-CLS, GAN-INT and an hybrid of the last two.

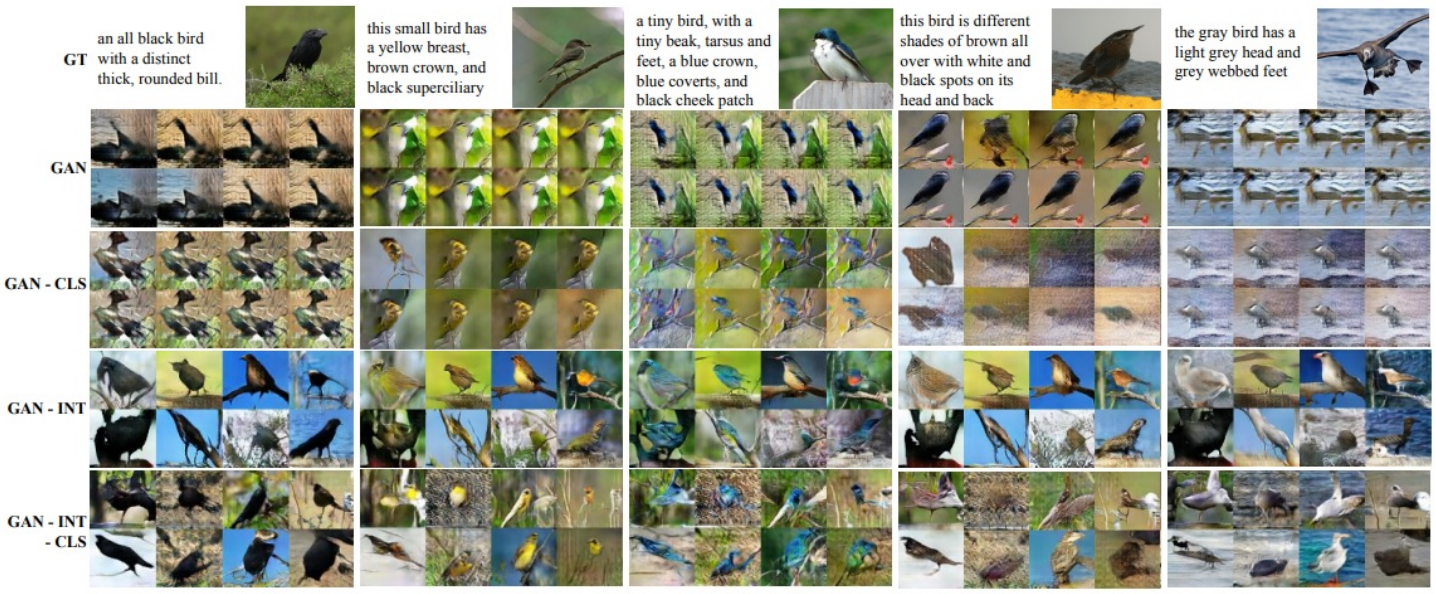

GT
an all black bird with a distinct thick, rounded bill.
this small bird has a yellow breast, brown crown, and black superciliary
a tiny bird, with a tiny beak, tarsus and feet, a blue crown, blue coverts, and black cheek patch
this bird is different shades of brown all over with white and black spots on its head and back
the gray bird has a light grey head and grey webbed feet
GAN
GAN - CLS
GAN - INT
GAN - INT - CLS

Figure 5.3: GAN results for text to image synthesis on CUB dataset

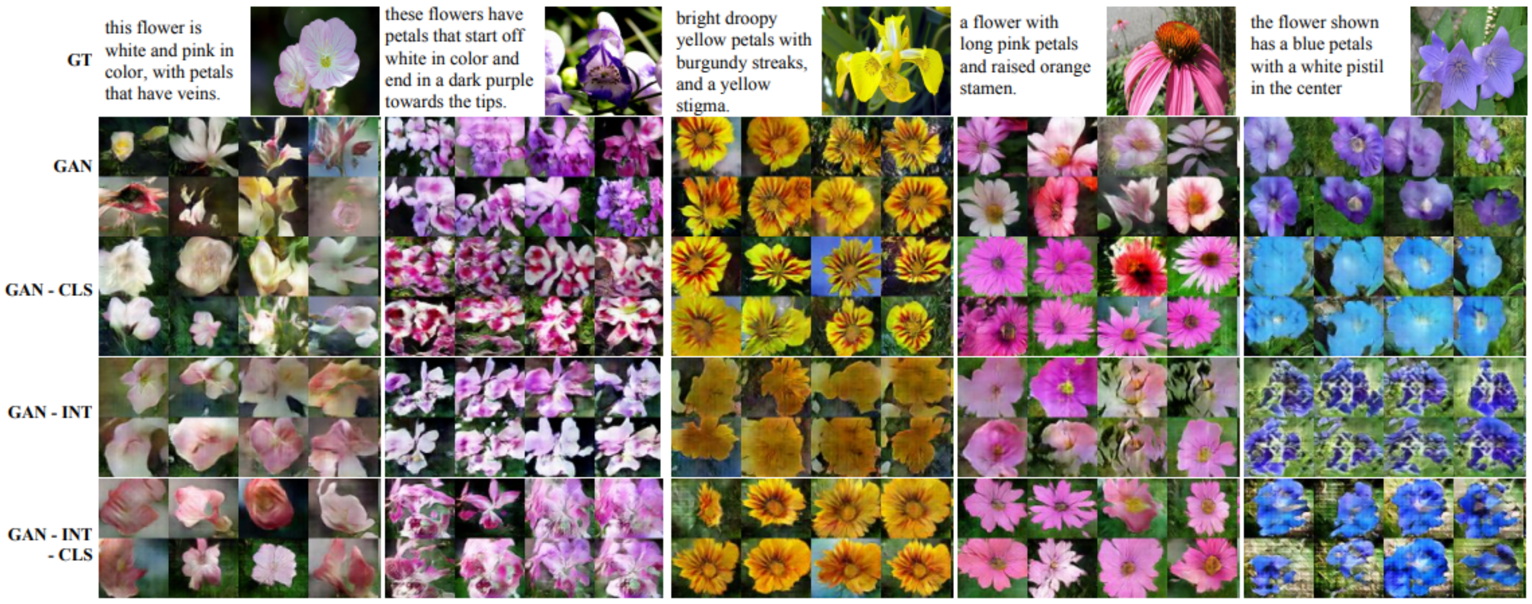

GT
this flower is white and pink in color, with petals that have veins.
these flowers have petals that start off white in color and end in a dark purple towards the tips.
bright droopy yellow petals with burgundy streaks, and a yellow stigma.
a flower with long pink petals and raised orange stamen.
the flower shown has a blue petals with a white pistil in the center
GAN
GAN - CLS
GAN - INT
GAN - INT - CLS

Figure 5.4: GAN-CLS results for text to image synthesis on Oxford-102 dataset

To prove the generalizability of the model, the authors also tried to generate images with multiple objects and variable backgrounds obtaining even more impressive results.
Text to image synthesis is an application which can attract quite interest in many Business-To-Customer domains especially those companies which aim to gain and keep clients backing to the design of their product.
Consider for instance furniture, automotive, clothing sectors and so on. Collecting opinions, interviews, reviews expressed in natural language and readily propose different visual prototypes in order to meet the wishes of the crown is indeed one of the main ambitions of those companies.

5.2 Image to image translation

Before delving into this next application, it is necessary to briefly discuss about a particular kind of GANs: the **conditional GANs (cGANs)** [49].
Previously we referred to GAN as a "random image generator" since the multi-modal nature of the generation process cannot be controlled in advance. Despite to this consideration, Conditional GAN were created with the idea to somehow inject this capability and reduce as much as possible the vagueness of the output distribution by providing an additional information to drive the generation towards a more specific target.
Given y, vector representation of a class label, it is possible to condition the generation process by feeding y into the both the discriminator and generator as additional input layer. The objective function of a cGAN will consequently change in:

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

Taking for instance the classical MNIST dataset, y should be the one-hot encoded vector representing the 10 possible digits. Then, by conditioning the model with one specific class, the generator should ideally output different samples but each one belonging to that class.

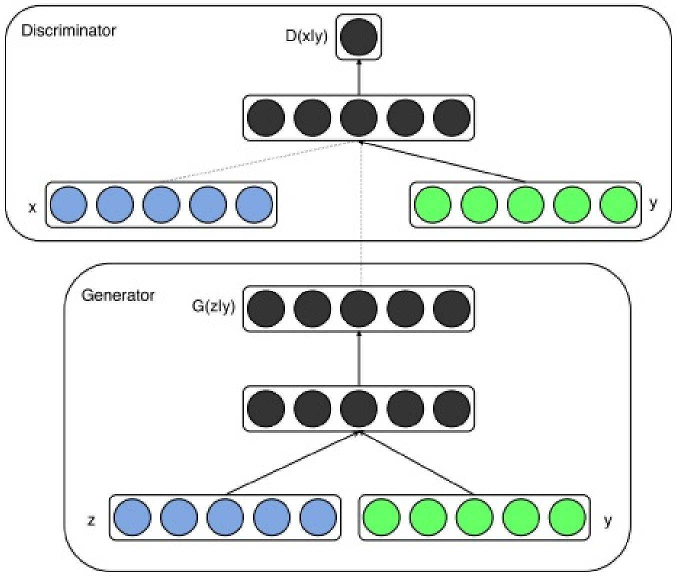


Figure 5.5: Conditional Adversarial Network

The paper ”Image-to-Image Translation with Conditional Adversarial Networks” [50] proposes another GAN-based solution which differs from the previous for the fact that it focuses only on the image domain. However, a general objective still hold: produce images on the basis of certain input conditions.
These “input conditions” justify in a sense the need for Conditional GAN.
The idea is to map input to output images where the input can be raw quality, hand-drawn or architectural patterns, and the output is an image as much as indistinguishable from the reality.
The solution proposed exploits a mix between standard GAN and cGAN (justified by experimental evidences) in which the input image is used as the conditioner of the output. An interesting peculiarity lies in the noise modelling which is not the typical noise vector but it’s implicitly injected in form of dropout across all generator network layers. The standard GAN contribution is led by L1 regularization on the generator side, indeed L1 results in less blurring compared to L2. Next up, some experimental results on building facades which compare L1, cGAN and a combination L1-cGAN with respect to the original

ground truth.

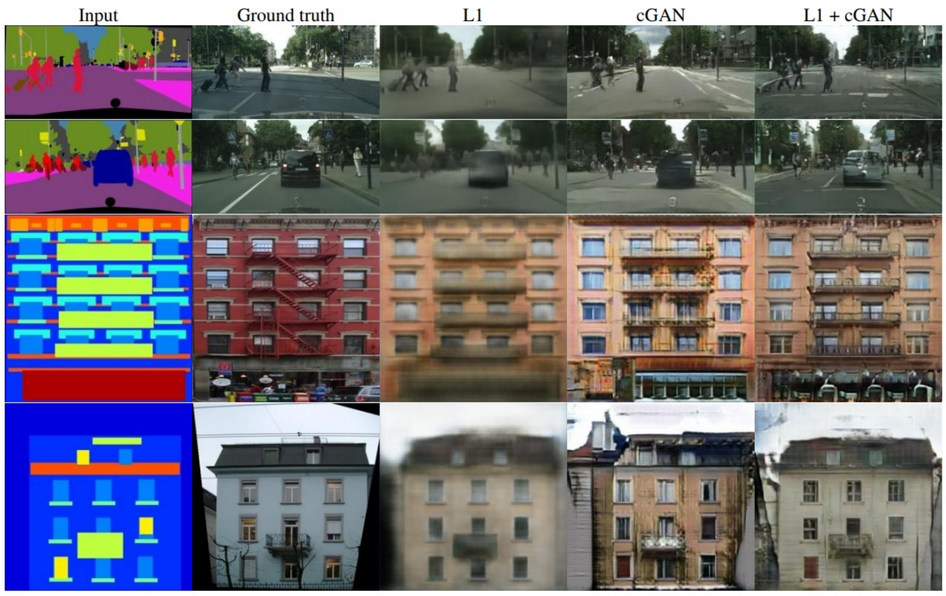


Figure 5.6: GAN results for image to image translation on a facades dataset

The project related to this paper is called **pix2pix** and the open source code is available on `https://github.com/phillipi/pix2pix` [51]. On Youtube can also be found a funny demonstration of the pix2pix project titled "ARTIST Vs. PIX2PIX - Is this HUMOR or HORROR?!" [52] in which a young youtuber submits to the model some hand-drawn picture made by himself obtaining very borderline results. In the video, the model he uses is trained with a ground truth of human faces.
Further improvements has been made and even more implementation has been released for different domains using the Tensorflow framework. At `https://affinelayer.com/pixsrv/` [53] it is possible to play with them choosing between cats, shoes, handbags and facades.

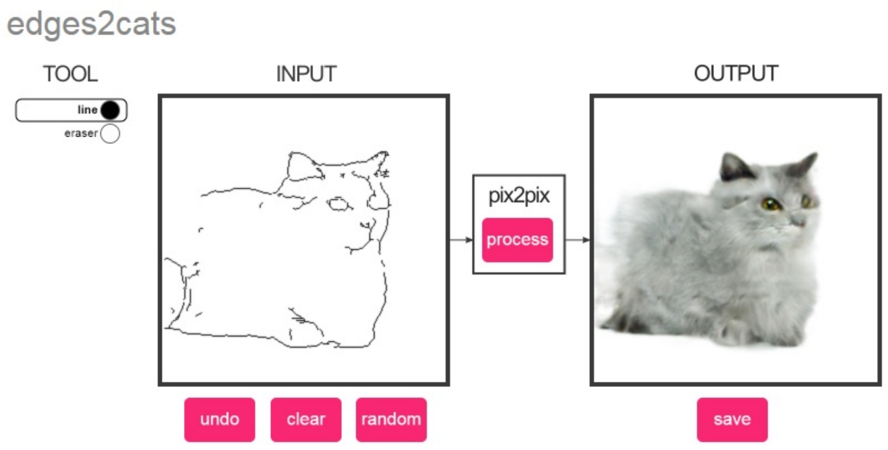


Figure 5.7: edges2cats UI
(credits: pix2pix project)

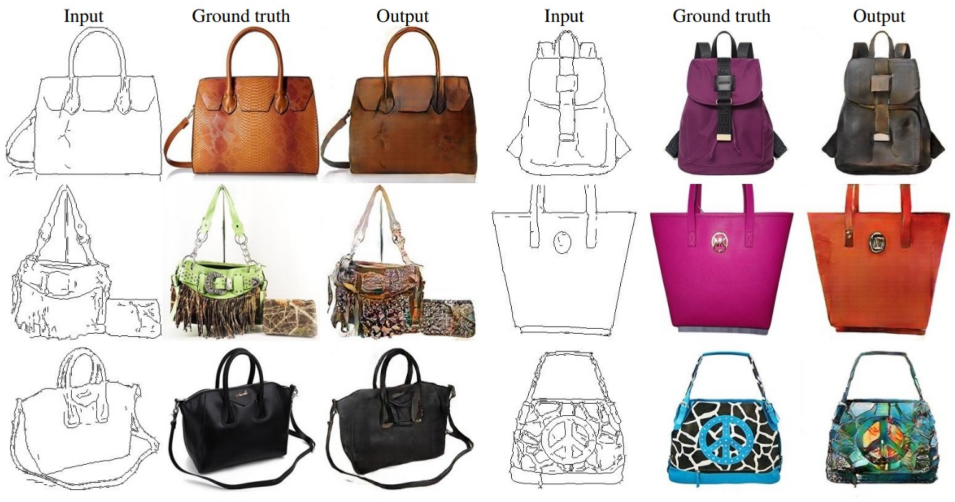


Figure 5.8: edges2bags results
(credits: pix2pix project)

This tool can be seen as an alternative to the "text to image synthesis", indeed for an user point of view, the only difference is the way to provide the input. Real world applications are so more or less the

same as before but now with a different prospective.
The version trained on human faces can represent, for instance, a solution that supports police sketch artists in order to produce the identikit of a suspect on the basis of the witness statements.

5.3 Increasing image resolution

In the field of Computer Vision, the term Super-Resolution refers to the technique used to improve the resolution of digital images.
The researches about Super-Resolution are motivated by the huge number of possible practical application in many sectors, from medical diagnosis, in order to help doctor to have a better view of the image of the patient labs, till satellite images analysis.
In the past, some solution have been proposed to perform Super-Resolution such as trivially reducing the pixel size through CMOS image sensors or using stochastic approach based on Bayesian estimators [54].
Recently, some contributions coming from Artificial Intelligence achieved the breakthrough. The solution we want to explore in this section is once again a GAN-based solution, from the paper "Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network" [55].
In order to have a good comparison for their results, the authors started from already high-resolution images (HR). As a first step they performed a downsampling operation on the HRs applying a Gaussian filter and obtaining the relative low-resolution images (LR).
The objective of their GAN is to train a generating function that estimates for a given LR input image its corresponding HR counterpart (SR). On the other side, the discriminator has to distinguish between the real HR and the generated SR. In such a way, the generator is encouraged to learn to create SRs that are highly similar to real HRs and thus difficult to classify by the discriminator.
In that work they proposed a **Super-Resolution Generative Adversarial Network (SRGAN)** for which they employ a deep residual network (ResNet) with skip-connection for the generator.
Residual learning is useful when the CNN is particularly deep and so

there is the risk of information degradation and accuracy saturation. Skip-connections allow the first layers to be connected not only to the next ones but also to those far away. In typical encoder-decoder architecture, skip-connections are represented as connection from each encoder layer to the relative decoder layer (the layer of the same dimension), bypassing the bottleneck in the middle.
The advantage of this approach is to minimize the possible loss of information across the network passing them directly to the last layers as a support for data reconstruction.

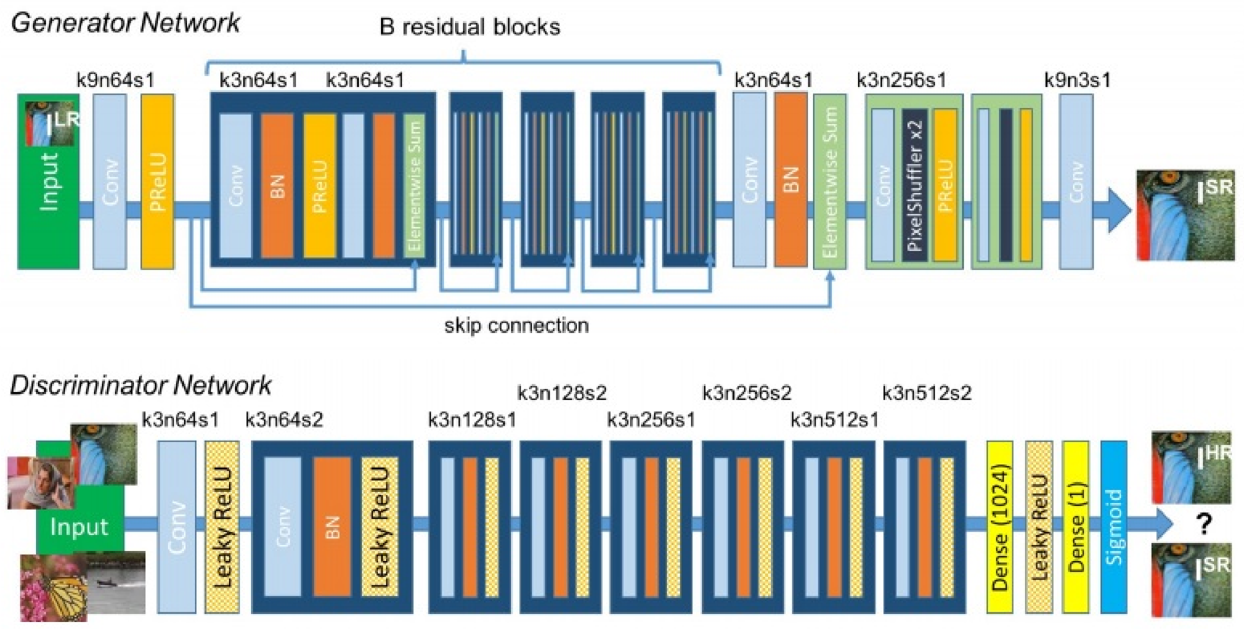


Figure 5.9: SRGAN architecture

In the Figure 5.9, the layer description is in the following notation: k = kernel size, n = number of feature maps and s = stride.
The related work which led to the SRGAN solution is based on the studies of Super-Resolution technique with Deep CNN, where the common practice consists not only in minimizing the MSE but at the same time maximizing the Peak Signal-To-Noise Ratio (PSNR). The problems arise when the image presents very high texture details which

are hardly captured by MSE and PSNR metrics. This effect becomes pretty evident in the image 5.11 if we look closely at the ornamental motifs in the clothes of the little girl.
The results are presented as a comparison between the LR (bicubic kernel with downsampling factor r = 4.), standard SRResNet optimized for MSE, SRGAN and the original HR image.

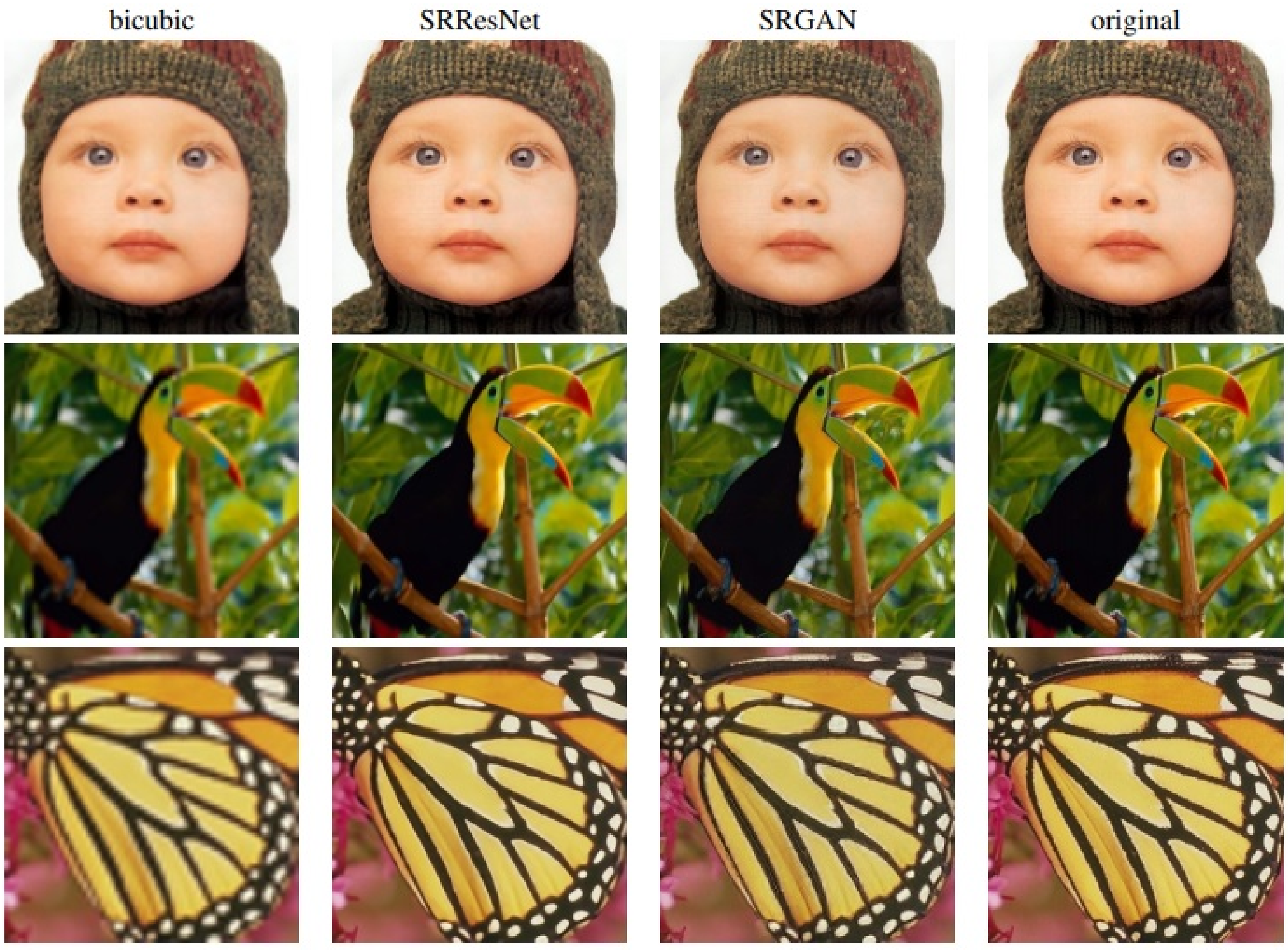


Figure 5.10: SRGAN results - Part 1

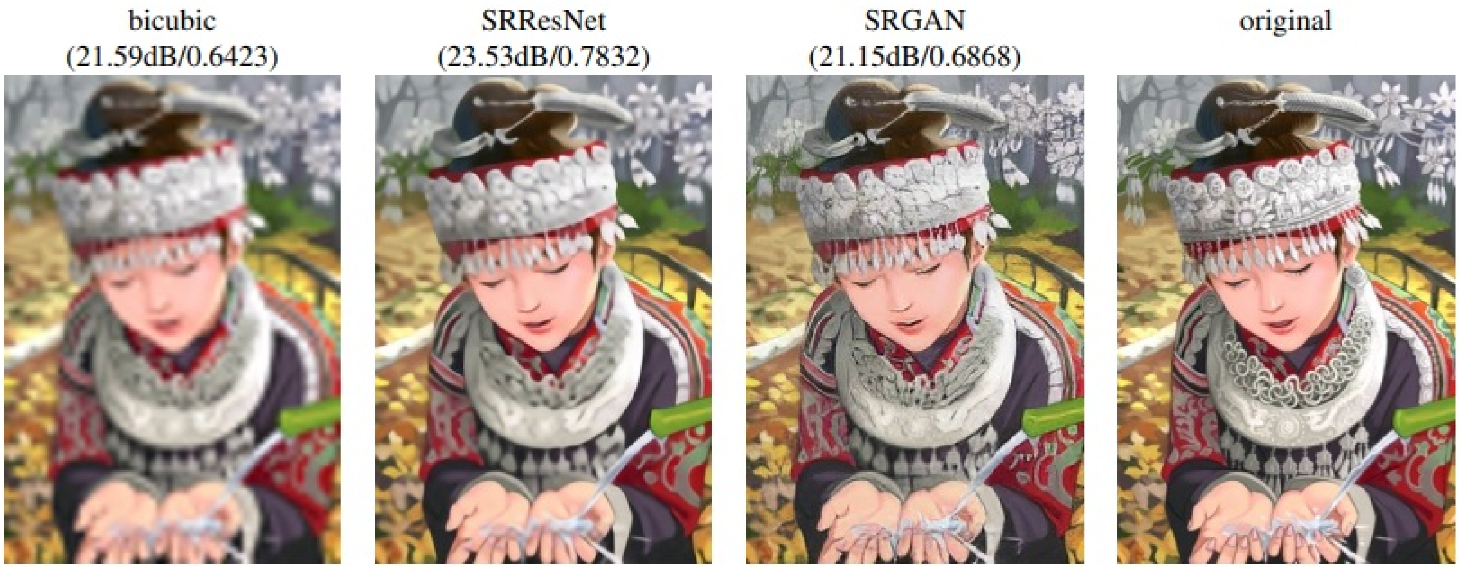


Figure 5.11: SRGAN results - Part 2

5.4 Predicting next video frame

GAN solutions are even employed not only for strictly generative tasks but also in predictive generative context.
An interesting case is when the input data are not independent of each other but are correlated for instance along the time dimension. That's exactly the case of video data seen under the point of view of the sequence of frames.
A lot of applications are indeed designed with the objective of predicting the next state of a system relying on what already exists, while others attempt to patch missing information which have been not captured for some reasons by a monitoring device or by a sensor.
In order to exploit these time correlations most Deep Learning solutions employ Recurrent Neural Networks, as already mentioned in the 3.1.
In this section we explore the solution proposed in the paper "Unsupervised Learning of Visual Structure using Predictive Generative

Networks" [56] where the authors designed a GAN-based architecture integrated with LSTM in order to generate the prediction of the possible next frame of a video.
The motivation that led the author to choose a GAN solution is because traditional methods based on MSE tend to produce quite blurred images, such as already seen in the previous section.
Considering that, their solution, called **Predictive Generative Network (PGN)**, is designed by exploiting a weighted combination of MSE and Adversarial Loss.

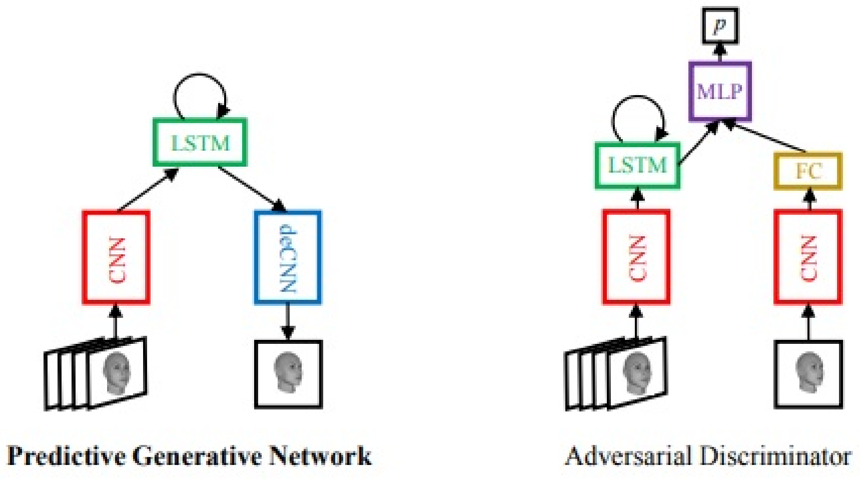


Figure 5.12: Predictive GAN architecture

The parts which compose the PGAN are illustraded in Figure 5.12.
The Generative part takes in input the sequence of frames and encodes them in a lower-dimensional feature space through a first CNN.
The temporal correlation between the frames is captured by a Long-Short Term Memory, usually preferred over the simple RNN since it helps to avoid the vanishing gradient. This choice becomes even more important in a GAN context to reduce the risk of possible mode collapse.
After the LSTM, a Deconvolutional NN performs the generation of the image which should candidate to be the next frame.
On the other hand, the Discriminative counterpart starts exactly like

the Generative, taking in input the same sequence of frame and processing it again through a CNN and an LSTM. After that, the Discriminative receives also both the encoded representations of the generated frame and the real one.
All these input are finally passed to a Multilayer Perceptron which has to assign a probability of how much the final frame seems to come from the ground truth, as well as it should minimize this value when detects fake frames.
The model has been validated on a dataset of video showing face rotations and the results demonstrate the effectiveness of the MSE/AL approach over the standard MSE.

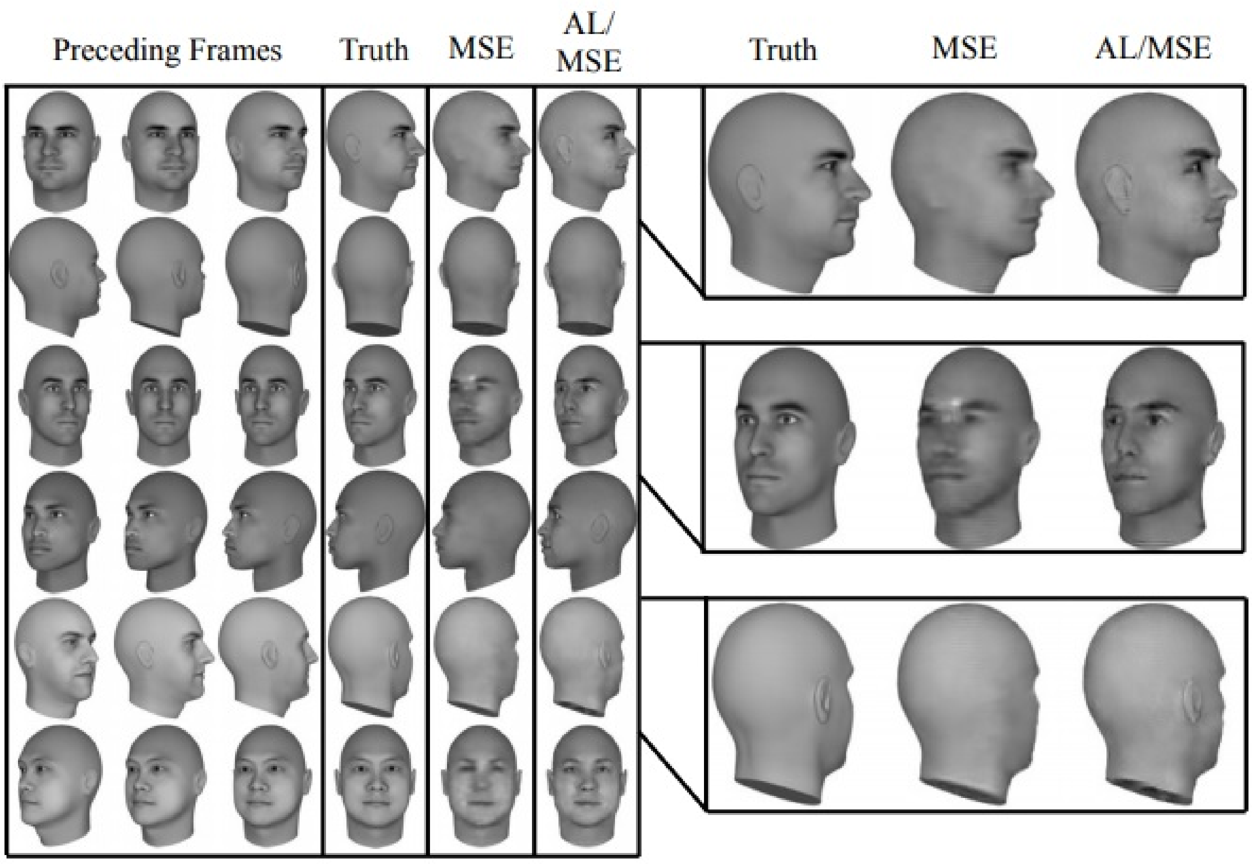


Figure 5.13: Predictive GAN results

Chapter 6

Experimental Case: Generating new Amazon products

6.1 Overview, goal and datasets

We conclude this thesis by proposing a possible application which involves the usage of GAN.
The idea is to generate new images candidate to be new possible revolutionary and innovative products which in a later stage can be designed and sold by fashion companies. Our target is so clothing products like t-shirt, pants, shoes and so on.
In order to accomplish our objective we need to know which are the features that make a product successful, what is the trend of the public demand, which is the market orientation and even more statistical information. These indication can be often obtained by extracting reviews and ratings of already sold products from the well known big e-commerce platforms, like Amazon, eBay, ePRICE...
By collecting the images of the most popular products we intend to use these data to support the generator and discriminator of our GAN to learn to produce images of not-yet-existing products that can be used as a guidance for stylists and designers.
In every Machine Learning task, one of the most critical part is related on how to obtain a dataset which reflects as well as possible the char-

acteristics we want to exploit, represented both in implicit or explicit ways. Moreover, the dataset should be large enough to consider all the possible cases and avoid the model to overfit only on a subset of them.
In our case, all of this has been possible thanks to the Professor Julian McAuley of the Computer Science Department of the University of California San Diego (UCSD), who kindly provided the dataset we used for our experiments, but originally built by himself for research purposes in the context of recommender system [57, 58].
This dataset contains product reviews and metadata from Amazon, including 142.8 million reviews spanning May 1996 - July 2014. For our experiments we focused only on the "Clothing, Shoes and Jewelry" category.
The data we used were split in two files, one containing 5,748,920 reviews and the other 1,503,384 products.
Each **review** is represented by the following attributes:

- `reviewerID`: ID of the reviewer (e.g. "A2SUAM1J3GNN3B")
- `asin`: ID of the product (e.g. "0000013714")
- `reviewerName`: name of the reviewer (e.g. "J. McDonald")
- `helpful`: helpfulness rating of the review (e.g. [2, 3])
- `reviewText`: text of the review (e.g. "I bought this for my husband who plays the piano. He is having a wonderful time playing these old hymns. The music is at times hard to read because we think the book was published for singing from more than playing from. Great purchase though!")
- `overall`: rating of the product from 0 to 5 (e.g. 4.0)
- `summary`: summary of the review (e.g. "Heavenly Highway Hymns")
- `unixReviewTime`: time of the review in unix time (e.g. 1252800000)
- `reviewTime`: time of the review in raw format (e.g. "09 13, 2009")

And for each **product metadata**:

- `asin`: ID of the product (e.g. "0000013714")
- `title`: name of the product (e.g. "Girls Ballet Tutu Zebra Hot Pink")
- `price`: price in US dollars at time of crawl (e.g. 3.17)
- `imUrl`: url of the product image (e.g. "`http://ecx.images-amazon.com/images/I/51fAmVkTbyL._SY300_.jpg`")
- `related`: asins of the related products split in four nested sub-categories (also bought, also viewed, bought together, buy after viewing)
- `salesRank`: sales rank information (e.g. "Toys Games": 211836)
- `brand`: brand name (e.g. "Coxlures")
- `categories`: list of categories the product belongs to (e.g. [["Sports Outdoors", "Other Sports", "Dance"]])

6.2 Data pre-processing

By taking a closer look to the original datasets we figured out many critical aspect which require an accurate selection, cleaning and transformation opererations before we proceed towards the GAN solution.
The first step is to merge the two datasets with a join operation on the shared primary key `asin`. We have gone through the entire **reviews** archive performing a group by `asin` and calculating number of review and average rating for each distinct product.
From **product metadata** we extracted the relative `imUrl` attribute in order to be able to retrieve the image of the product.
After that, we got left 1,136,004 products, obtaining so a first significant reduction from the initial amount of 1,503,384. This is due to the fact that 367,380 products appear to have zero reviews or no image associated.

This attribute selection let us focus only on the information we really need and with a simpler data structure:

- `asin`: see previous section
- `total_revs`: total number of reviews associated to the product (e.g. 302)
- `avg_rating`: average rating of the product from 0 to 5 on the entire reviews (e.g. 4.2)
- `imUrl`: see previous section

The products we left are characterized by very different ratings and number of reviews, while for our purpose we have to distinguish between the popular and successful ones from the unpopular and unattractive ones.
We defined two threshold values to filter the best products: a minimum average rating of 4.00 and a minimum number of reviews set to 12. The reason why we have chosen these is to preserve an adequate amount of images, not too little but not too much. Moreover, we have to take into account a minimum number reviews in order to avoid unfair cases where, for instance, a product with one 5 stars review might overtake a 4.9 with 100 reviews.
This filtering operation has further reduced the number of products to 50,201 and, by exploiting the `imUrl` attribute, we started to collect the images running a script to download one by one in a jpg format. The final number of images collected is 49,774. Indeed, during the download few urls seem to be unreachable.
Other inconvenient has been encountered also after the download. We found some duplicated and some default Amazon images which state "No image available".
To remove duplicated and empty images we reuse a script [59] which compute a hash for every file, allowing us to find and delete duplicated even though their names are different.

6.3 GAN solution

We built our solution starting with an already existing GAN which operates on the Cifar10 dataset [60]. The CIFAR-10 dataset [61] consists of 60000 colour images in 10 classes, with 6000 images per class (airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck).
Starting from this solution we were able to analyze the qualitative and quantitative results and then to make some proper considerations in order to obtain a satisfactory result even on our dataset.
Because of long computational training time, we loaded and resized all the images in a 64×64 pixel format. For the same reason we also fed the model with mini-batches of 128 images (see Chapter 2.1).
The layered structure of the Generative and Discriminative convolutional networks is resumed in the following pytorch notation:

Generative Network

1. `ConvTranspose2d(input_shape=100, output_shape=512, kernel_size=(4, 4), stride=(1, 1), bias=False)`
2. `BatchNorm2d(512, eps=1e-05, momentum=0.1)`
3. `ReLU()`
4. `ConvTranspose2d(input_shape=512, output_shape=256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)`
5. `BatchNorm2d(input_shape=256, eps=1e-05, momentum=0.1)`
6. `ReLU()`
7. `ConvTranspose2d(input_shape=256, output_shape=128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)`
8. `BatchNorm2d(input_shape=128, eps=1e-05, momentum=0.1)`
9. `ReLU()`
10. `ConvTranspose2d(input_shape=128, output_shape=64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)`

11. `BatchNorm2d(input_shape=64, eps=1e-05, momentum=0.1)`
12. `ReLU()`
13. `ConvTranspose2d(input_shape=64, output_shape=3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)`
14. `Tanh()`

Discriminative Network

1. `Conv2d(input_shape=3, output_shape=64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)`
2. `LeakyReLU(negative_slope=0.2)`
3. `Conv2d(input_shape=64, output_shape=128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)`
4. `BatchNorm2d(input_shape=128, eps=1e-05, momentum=0.1)`
5. `LeakyReLU(negative_slope=0.2)`
6. `Conv2d(input_shape=128, output_shape=256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)`
7. `BatchNorm2d(input_shape=256, eps=1e-05, momentum=0.1)`
8. `LeakyReLU(negative_slope=0.2)`
9. `Conv2d(input_shape=256, output_shape=512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)`
10. `BatchNorm2d(input_shape=512, eps=1e-05, momentum=0.1)`
11. `LeakyReLU(negative_slope=0.2)`
12. `Conv2d(input_shape=512, output_shape=1, kernel_size=(4, 4), stride=(1, 1), bias=False)`
13. `Sigmoid()`

The most relevant parts of these networks are the shapes of the inputs and the outputs. For the generative part, the input is a 100-dimensional noise vector of random values and the output is a 3-dimensional vector representing the RGB channels of the 64×64 image generated. Discriminative side, the input obviously must match the output of the generator and the final outcome is a single value classification between 0 and 1.
The activation functions Tanh, ReLU, Leaky ReLU and Sigmoid and the parameters Kernel Size, Stride, Padding and Negative Slope are set experimentally and their meaning have been discussed in Chapter 2 and 3.

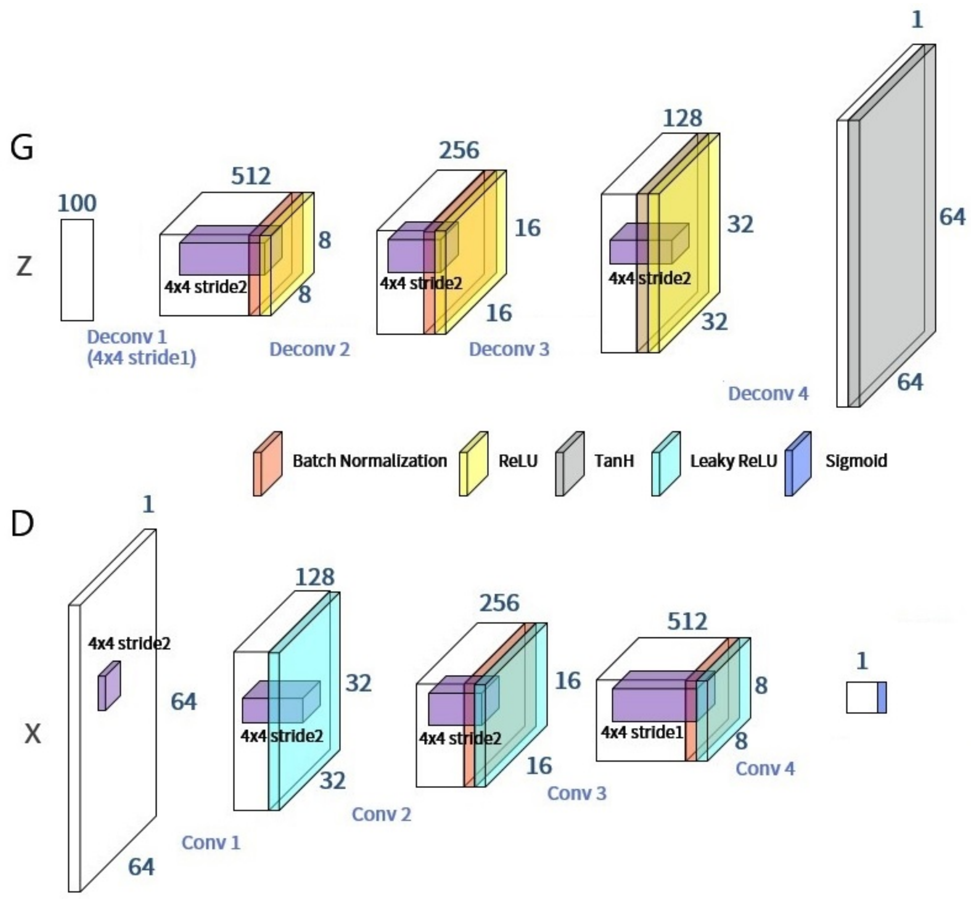


Figure 6.1: Experimental GAN architecture

6.4 Setup of the experiment

We start by providing two samples from the two dataset in order to have a view of the images we deal with:

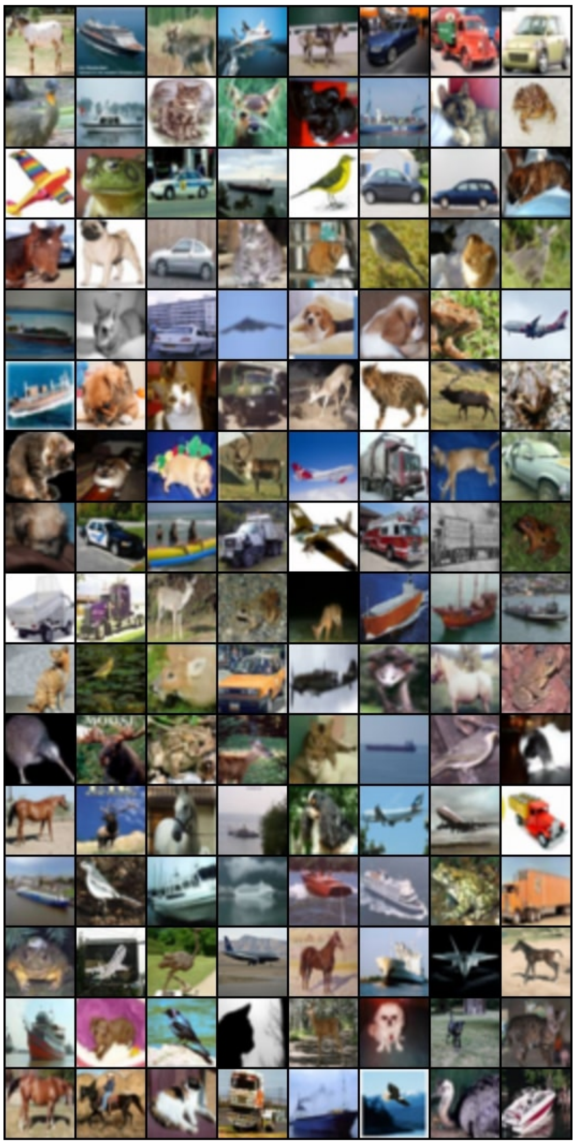


Figure 6.2: A sample of real images from the Cifar10 dataset

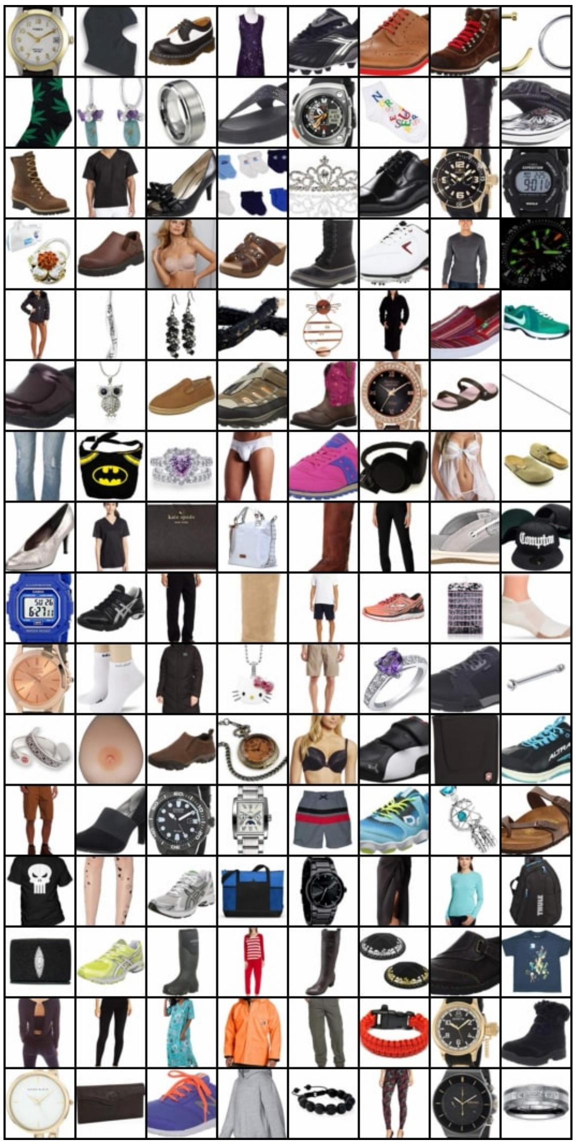


Figure 6.3: A sample of real images from the Amazon product dataset

We ran our GAN twice: one time using the images of the Cifar10 dataset and the second one using our Amazon Dataset. Following this procedure we can prove the legitimacy of our results by comparing them to which have already been technically demonstrated with suc-

cess.
As we can see, our images show an higher quality respect to the Cifar10. Luckily, the resizing operation helps to alleviate this difference and the GAN has shown the ability to manage it without particular issues.
Recalling the overall GAN loss function, our target is to find a solution for the following:

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

In theory, the solution to this mini-max game is where $p_g = p_{data}$, and the discriminator guesses randomly if the inputs are real or fake. However, the convergence theory of GANs is still being actively researched and in reality models do not always train to this point.
A possible convergence can be visualized over the training epochs by plotting some statistics on 2D graphs. This is what we will do in the next section where we show the **quantitative results**. In particular, we focus on four indicators:

- **Loss_D**: discriminator loss calculated as the sum of losses for the all real and all fake batches. Mathematically speaking, $\log D(x) + \log(1 - D(G(z)))$
- **Loss_G**: generator loss calculated. Mathematically speaking, $\log(D(G(z)))$
- **D(x)**: average output (across the batch) of the discriminator for the all real batch. This should starts around 1 and converge to 0.5 as G improves, so that D become unsure that real images are “really” real.
- **D(G(x))**: average discriminator outputs for the all fake batch. This should starts around 0 and converge to 0.5 as G improves, so that D is led to consider some fake images as real.

Regarding the **qualitative results** we will simply collect and show some fake generated samples from the first to last training epoch. We

will see the evolution of these samples which are expected to be noisy and confused at the beginning and they will get better and better towards the end.
We let the reader judge if these generated fakes can be considered good enough to accomplish our goal.
The source code has been written for the pytorch library. We setup the network parameters by initializing the weights of the convolutional layers from a Normal distribution with mean=0 (stdev=0.2) and the batch normalization layers with mean=1 (stdev=0.2). We used the Binary Cross Entropy as a criterion for the loss funtion. We provided two optimizer for D and G using Adam algorithm (an extension of stochastic gradient descent) with learning rate = 0.0002, beta1 = 0.5 and beta2 = 0,999.
The training process is performed across a proper number of epochs in which the dataset is split in mini-batches of 128 size, performing for each epoch a number of iterations equal to (dataset length / 128). For each iteration we submit the discriminator a mini-batch of real images and a mini-batch of fake. Then, we update the discriminator in order to maximize Loss_D and the generator to minimize Loss_G (or equivalently maximizing $\log(1 - (D(G(z)))))$.
All the experiments are carried out on a machine of the University of Bologna (DISI department), equipped with two GPUs NVidia Titan Xp.

6.5 Quantitative Results

The proper number of epochs has been determined in this way: first of all, we ran the model on both datasets on a very large amount of epochs (100 in our case); then we analyzed the trend of the indicators, described in the previous section, over this long training process and we looked for the specific epoch for which these indicators reached their best values.
In a second step, we reran again the model on both datasets but this time we set the epochs in a reasonable way, relying on the observations above.

We start this quantitative analysis showing the charts on the Cifar10 over 100 epochs. Here, another interesting fact is that not only we have found the best epoch but also we found the exact moment where the model suddenly fell in the mode collapse scenario.
We provide three kinds of chart: one to compare loss D and loss G over the single iterations (1 iteration = 1 mini-batch = 128 images), the second one is a more clear and aggregated prospective of the first one but over the epochs (where all the iteration values have been averaged) and the last one compares D(x) and D(G(z)) to show the accuracy of the discriminative in classifying fake and real images.

Cifar10 dataset

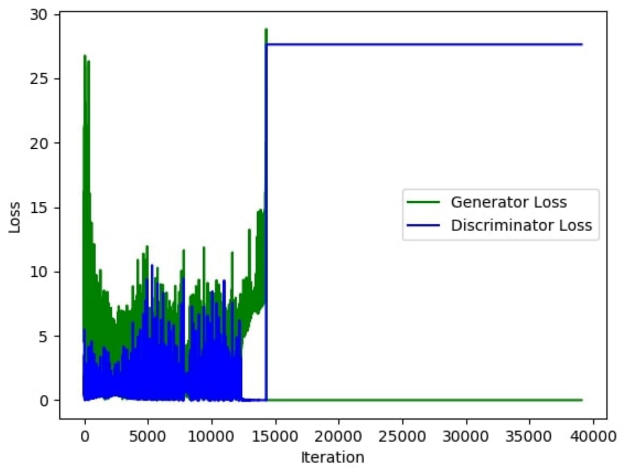


Figure 6.4: Cifar loss chart over 40000 iterations

The best epoch is located between 5000 and 10000 iterations, where the generator loss reaches his local minima (which can be seen as a global minima if we do not consider the case of mode collapse).
Mode collapse occurs towards 15000 iteration, where the discriminator loss explodes and stabilises between 25 and 30. On the other side, the generator consequently falls to zero. This means the generator found a way to fool the discriminator forever simply proposing each time

the same identical image (in the qualitative analysis we will show the guilty image).
Unfortunately the previous chart doesn't allow us to identify exactly the best epoch, so we need to replot it with in an epoch prospective.

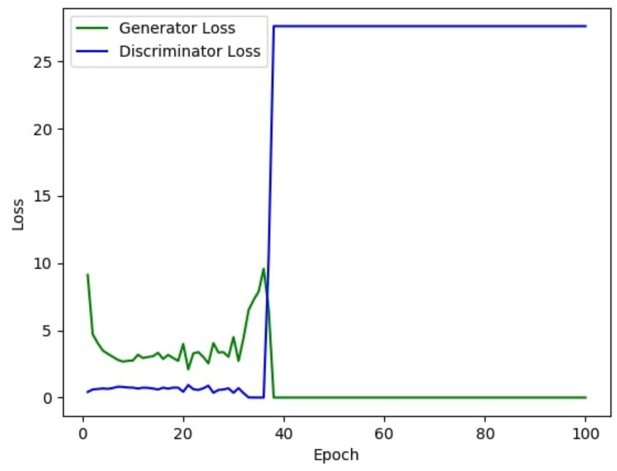


Figure 6.5: Cifar loss chart over 100 epochs

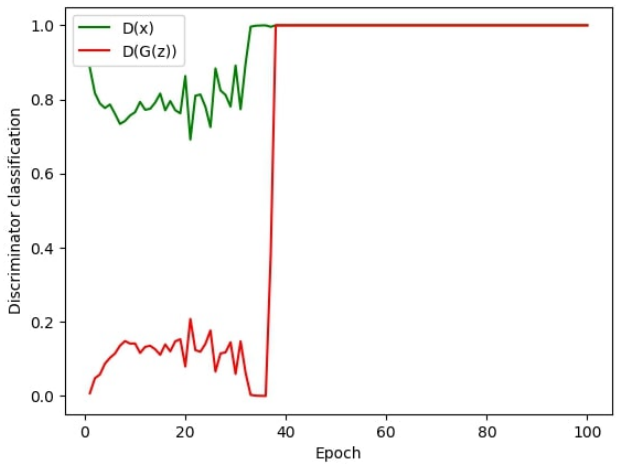


Figure 6.6: Cifar discriminator chart over 100 epochs

With this new kind of visualization we can say with more certainty that the best number of training epochs is 21 an the mode collapse happens above the 39.
The second chart reinforces our conviction since at 21 epochs we see the discriminator classifications get closer to 0.5. Also note that in the mode collapse all the fake images (or to be more precise "the fake image", since it is always the same) are recognized as real with a 100% certainty.
Now we can rerun the model and show again this three charts, but this time we resized the training to 21 epochs.

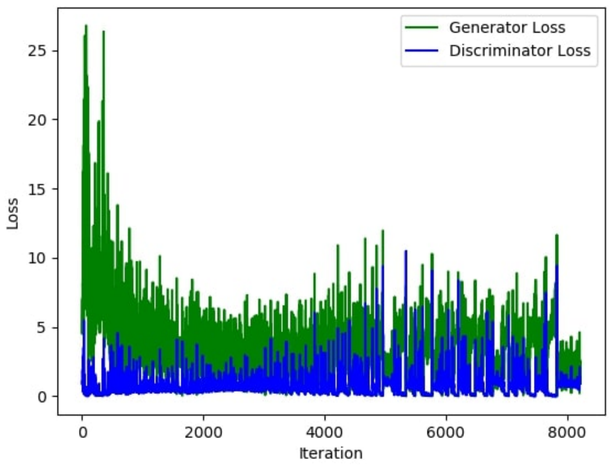


Figure 6.7: Cifar loss chart over 8000 iterations

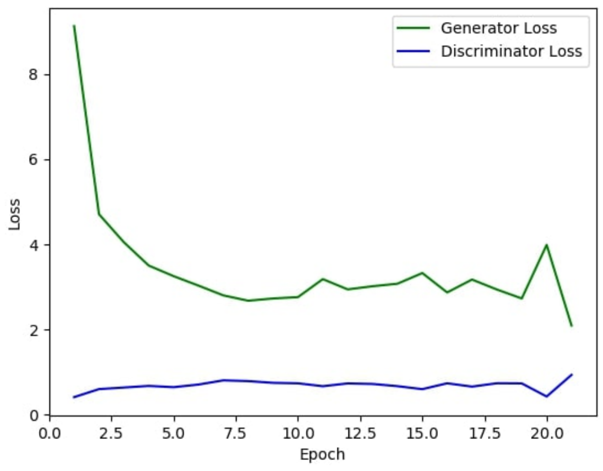


Figure 6.8: Cifar loss chart over 21 epochs

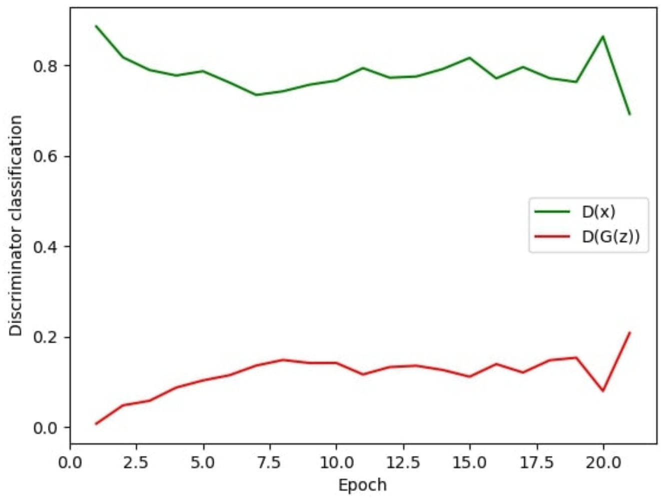


Figure 6.9: Cifar discriminator chart over 21 epochs

Our Amazon products dataset

We can now replicate the same procedure on our dataset. It is time to use the images of the best product obtained from the Amazon dataset.

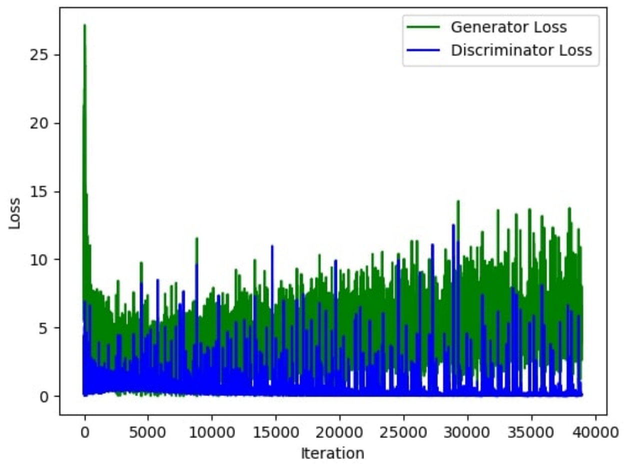


Figure 6.10: Amazon loss chart over 40000 iterations

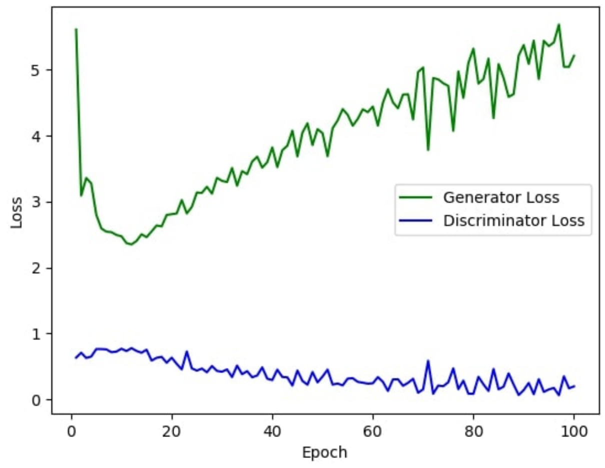


Figure 6.11: Amazon loss chart over 100 epochs

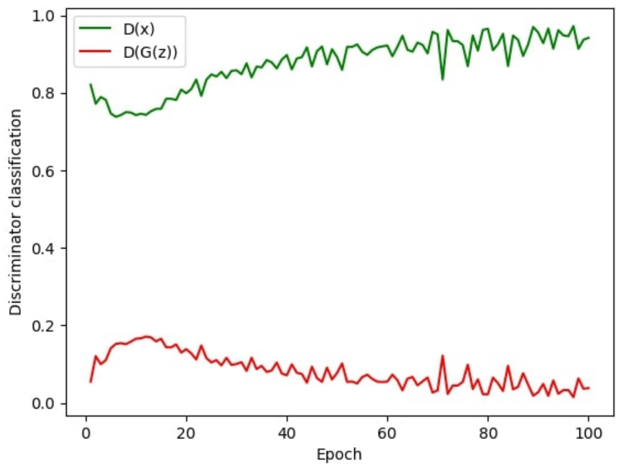


Figure 6.12: Amazon disciminator chart over 100 epochs

Thanks to the results and charts about the Cifar dataset we can not only select the best epoch for out dataset, but also make a comparison between the two.
The main two evident aspects are the absence of mode collapse and the need of less epochs to obtain good qualitative results. This doesn't mean the mode collapse will never occur but probably it requires more and more training epochs compared to the Cifar case.
Second question: why do we need less epochs? Since the quality of the images is better, one can think we should need more epochs and not less. Well, the reason is that more quality usually means more texture details to learn for the discriminator but also it is required a greater effort from the generator to represent and then generate those details.
In this context we have that the discriminative network starts to overfit faster than the generative and that will bring to lose the synchronization between the two, leading to a situation where the generative loses any chance to beat the discriminative. We will discuss in the last section about some possible advanced solutions to overcome the problem.
Replying the approach adopted for the Cifar, we estimated the best number of training epochs up to 12, and we provide the refactored charts.

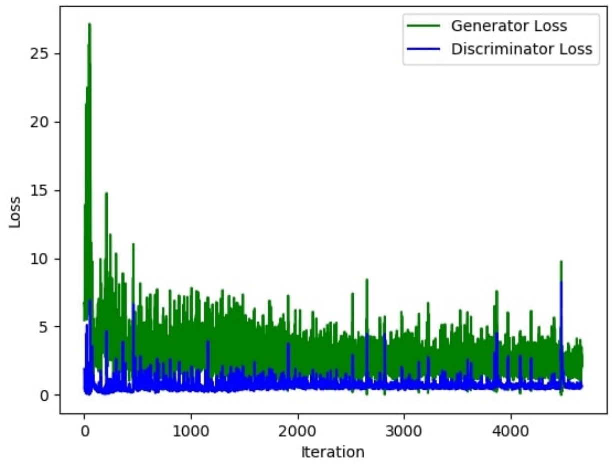


Figure 6.13: Amazon loss chart over 5000 iterations

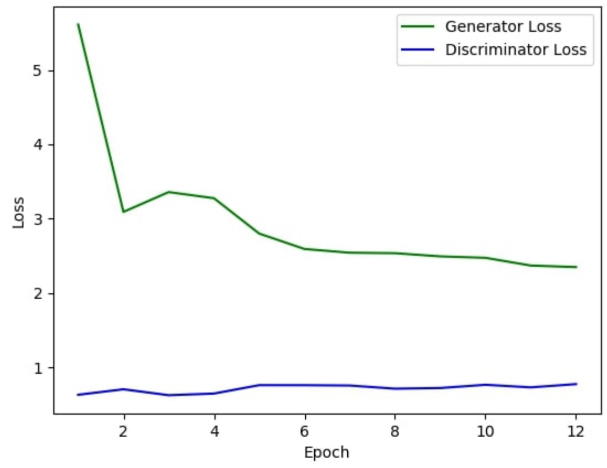


Figure 6.14: Amazon loss chart over 12 epochs

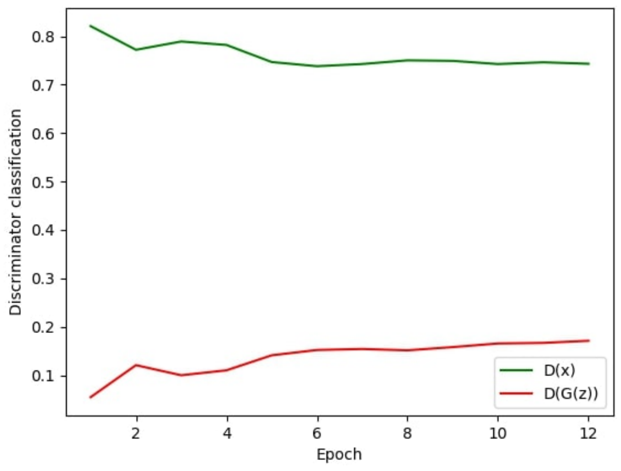


Figure 6.15: Amazon discriminator chart over 12 epochs

Most of the quantitative analysis about Machine Learning make use of metrics like accuracy, precision, recall and F1 score.
Are those metrics are also suitable in the context of GANs? Since the generative performance is mainly measurable in qualitative terms, let's focus for a moment on the discriminative network.
One interesting observation comes up if we compare the behaviour of our discriminative network with the behaviour of traditional supervised classifiers.
In a traditional classification task we have that usually the model starts with a very low accuracy and epoch by epoch it improves via standard backpropagation of the errors, leading more and more in a higher accuracy.
In GANs, the discriminative behaves in a complete opposite way. That's why the training process starts with a poor generator, so at the beginning the discriminative shouldn't have any particular difficulties in distinguish fake images from the reals. As soon as the generative improves, the discriminative loses his strength and decrease his accuracy getting more and more confused.
A GAN indeed has not the objective of minimizing an overall loss

function or to maximize the accuracy of the discriminative because in that way the generative wouldn't have any chance to compete with his counterpart. Instead, the intention is to keep a proper synchronization and an equilibrium between his internal parts so the disciminative tends to reach a situation of confusion, instability and inefficiency.
This argumentation justifies in a sense the alternative approach we used to evaluate our quantitative results. This practice is also adopted in many GANs papers, studies and experiments.
This analysis leads us to the following conclusion: relying on the observations related to the chart of Figure 6.15, we can assert that after 12 epochs of training on our Amazon dataset, the GAN reaches a state where around 20% of the generated images are classified as real from the discriminative network.

6.6 Qualitative Results

We conclude the analysis of the result with a rich illustration of the "fake" images obtained from the generative network.
Regarding the Cifar10 dataset, we present some samples obtained at epoch 21 and, as promised, we show what we get as soon as the model started to collapse.
Next, we will switch to our Amazon dataset of products. We will see the evolution of the GAN training from the first epoch to the twelfth one in order to have a full understanding of how the GAN improved epoch by epoch.
We have already provided before the samples from the original datasets, so the reader can compare these with those.
We would point out again that all the images you will see have been resized to 64×64 pixels in order to reach a compromise between having reasonable computation times and reasonable quality results.

Cifar10 dataset

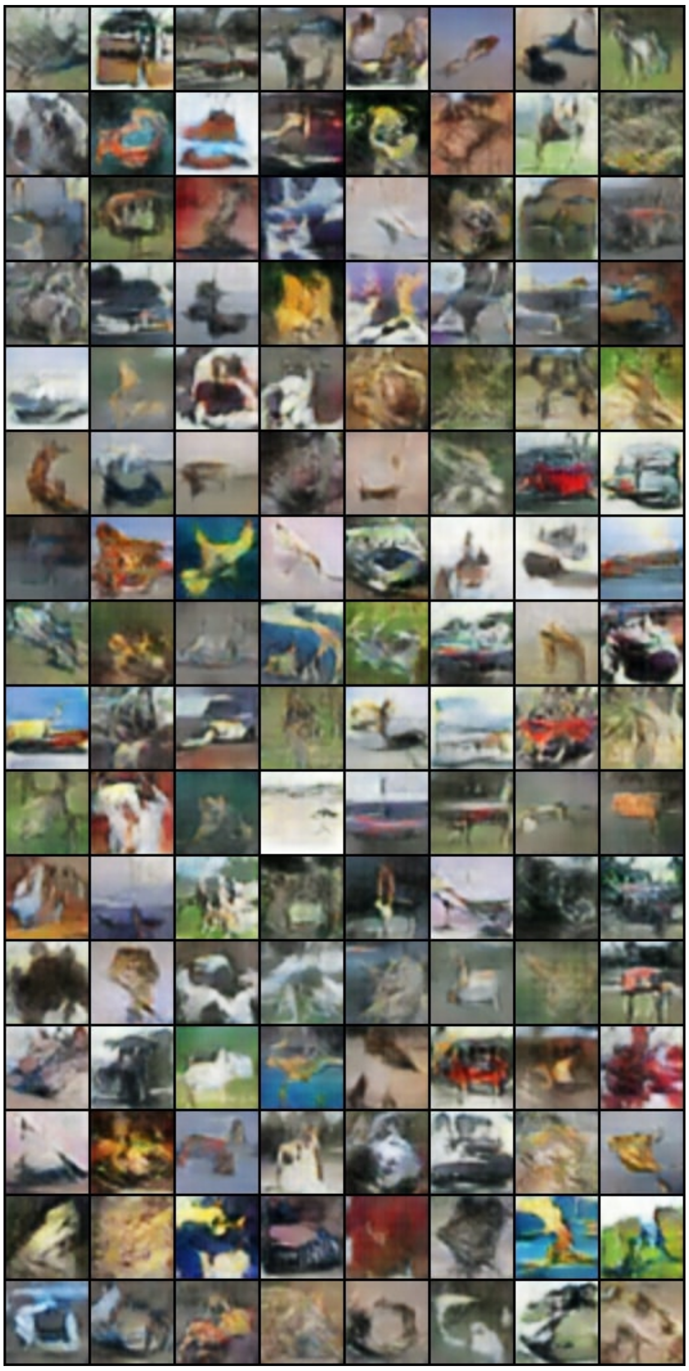


Figure 6.16: Cifar fake samples at epoch 21

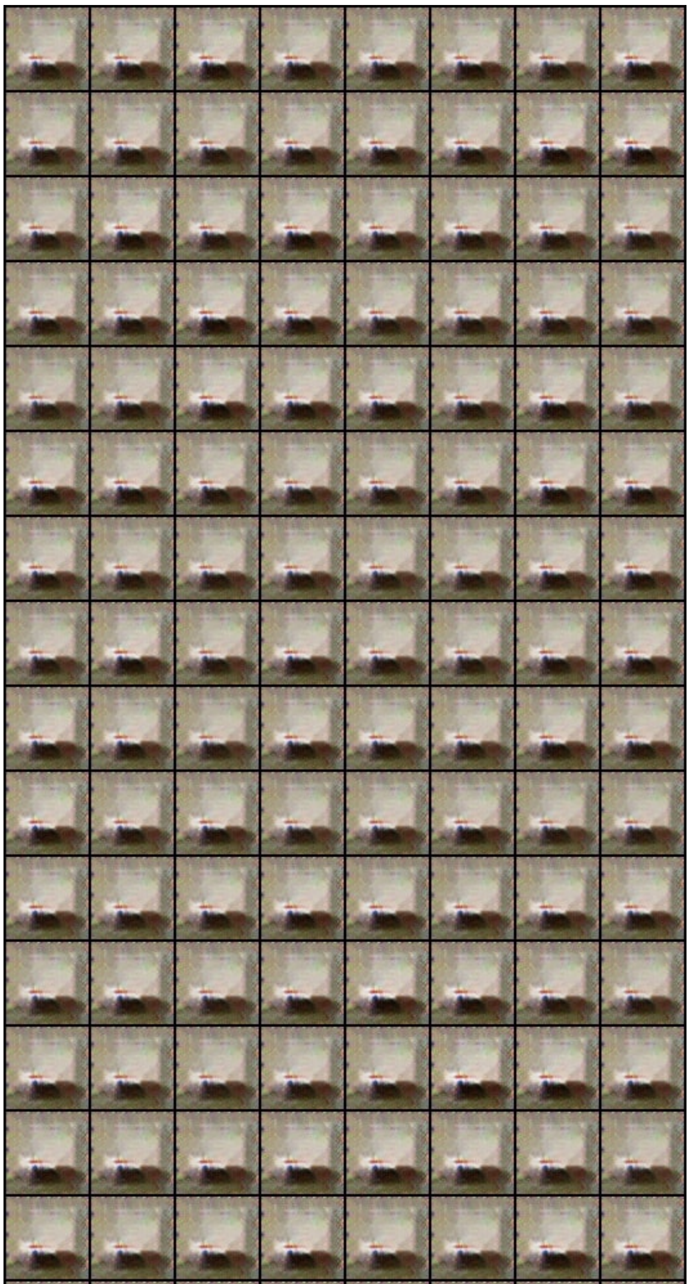


Figure 6.17: Cifar mode collapse after epoch 39

Our Amazon products dataset

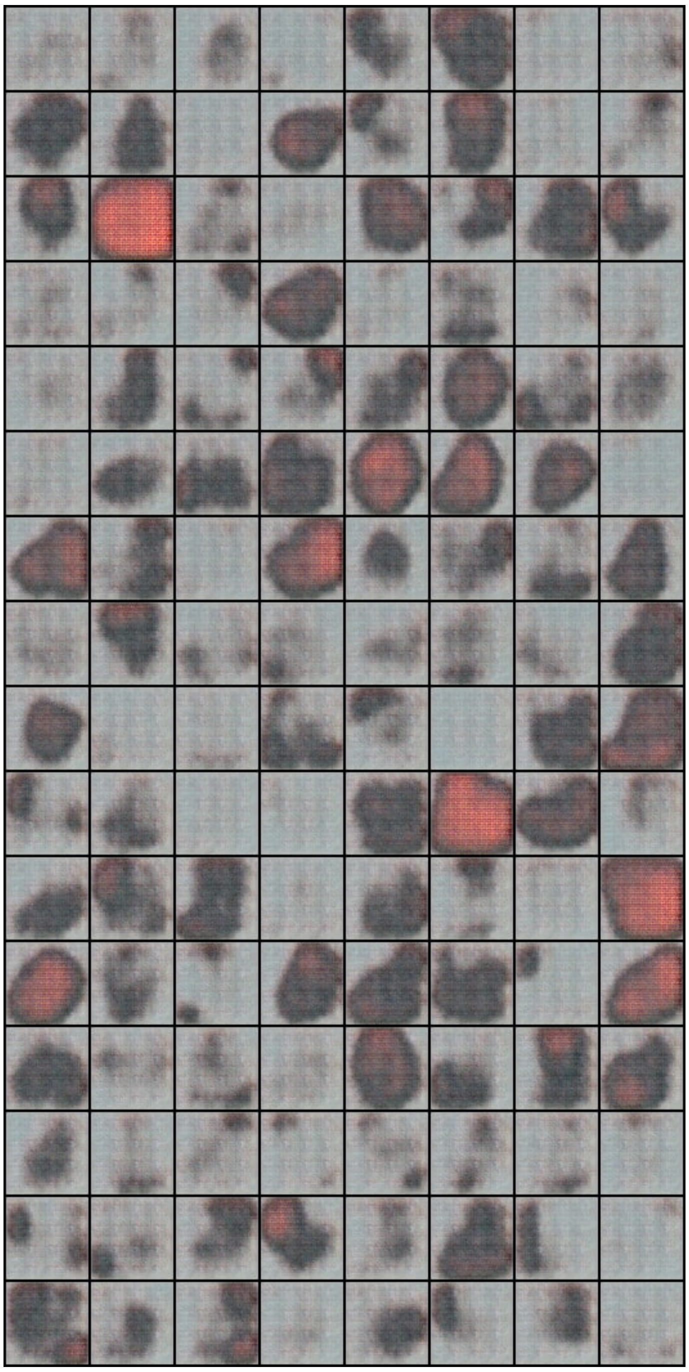


Figure 6.18: Amazon products fake samples at epoch 1

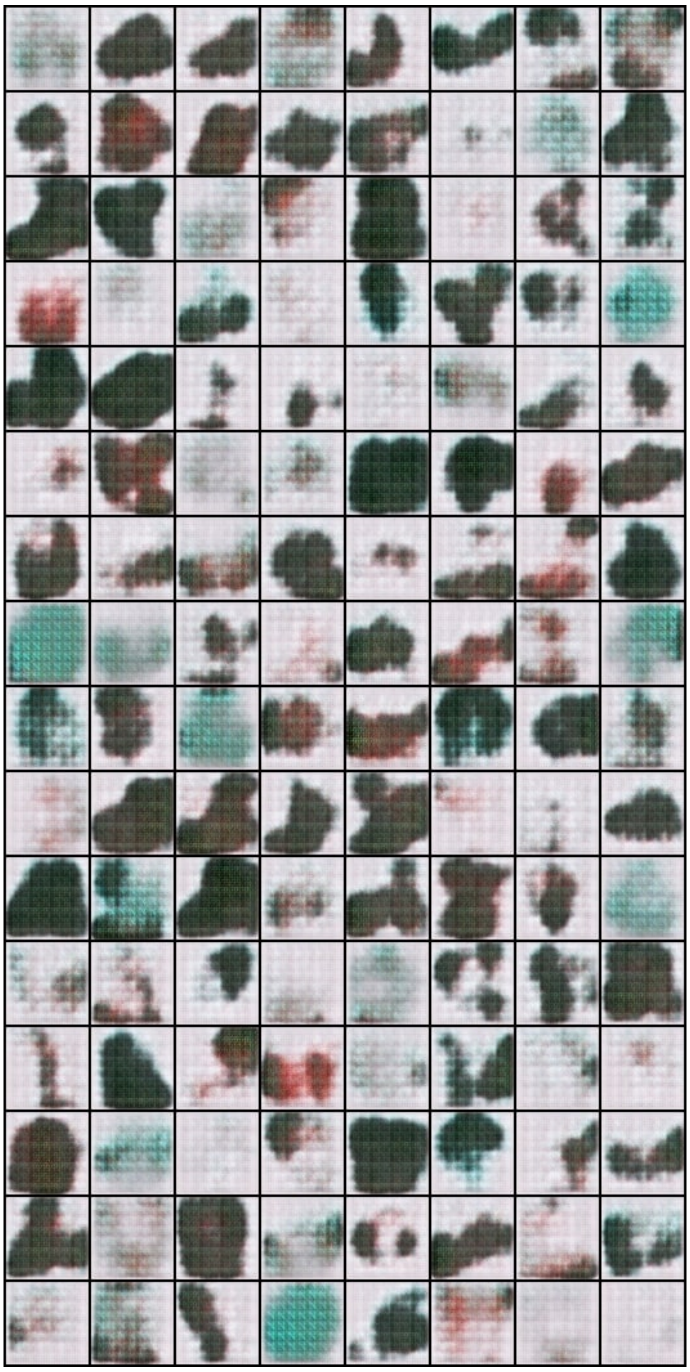


Figure 6.19: Amazon products fake samples at epoch 2

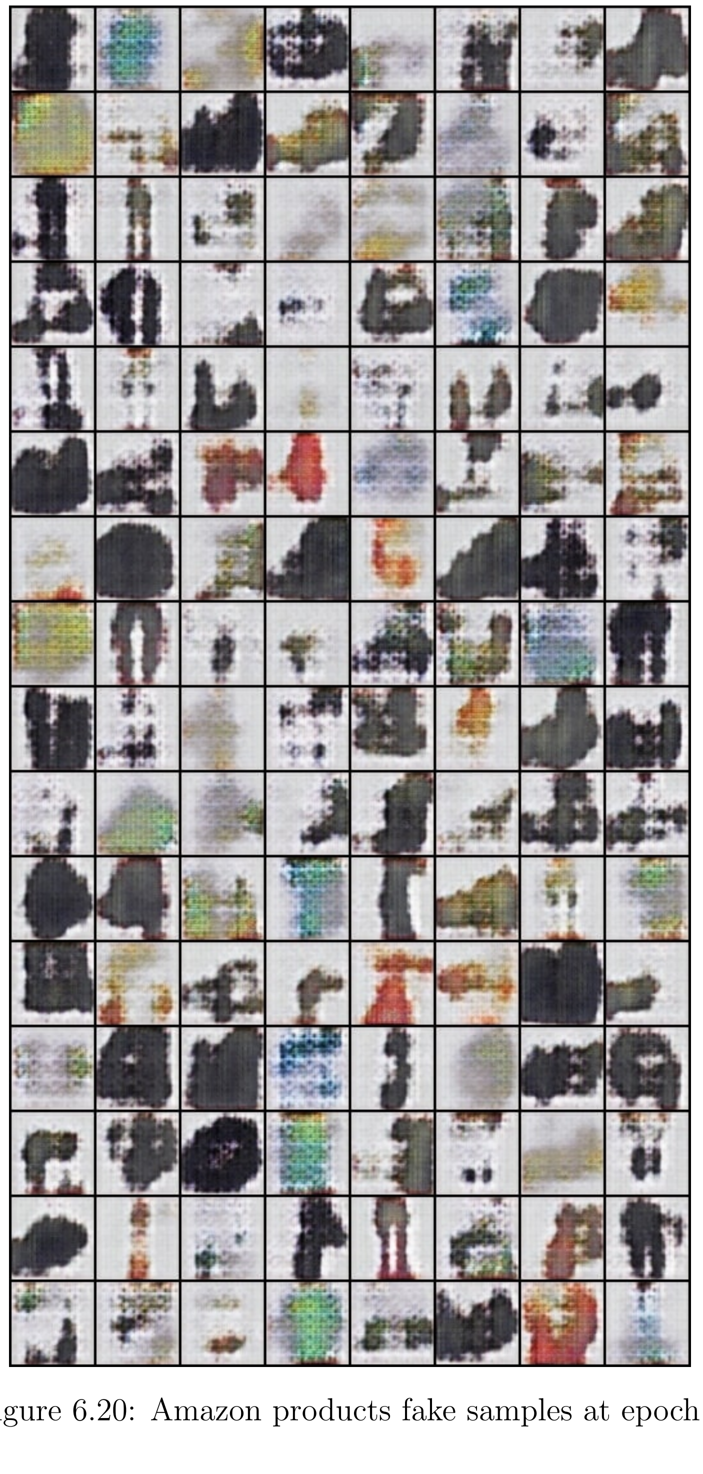


Figure 6.20: Amazon products fake samples at epoch 3

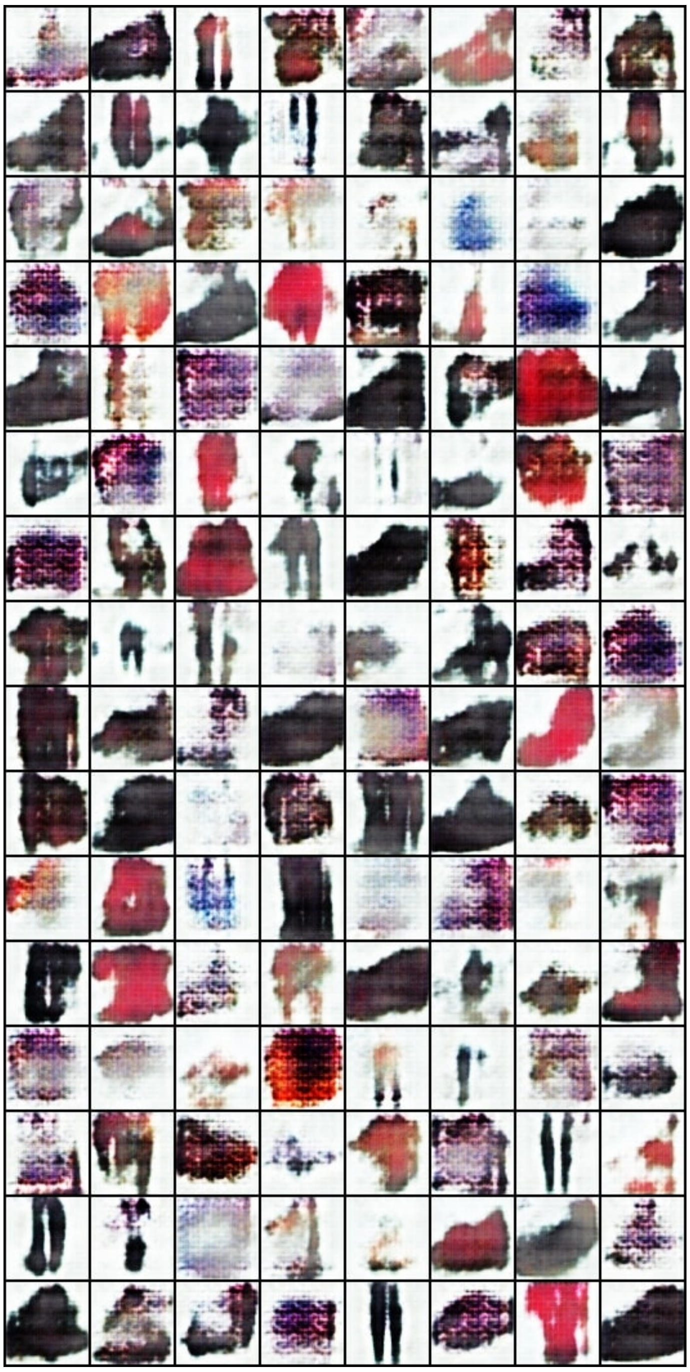


Figure 6.21: Amazon products fake samples at epoch 4

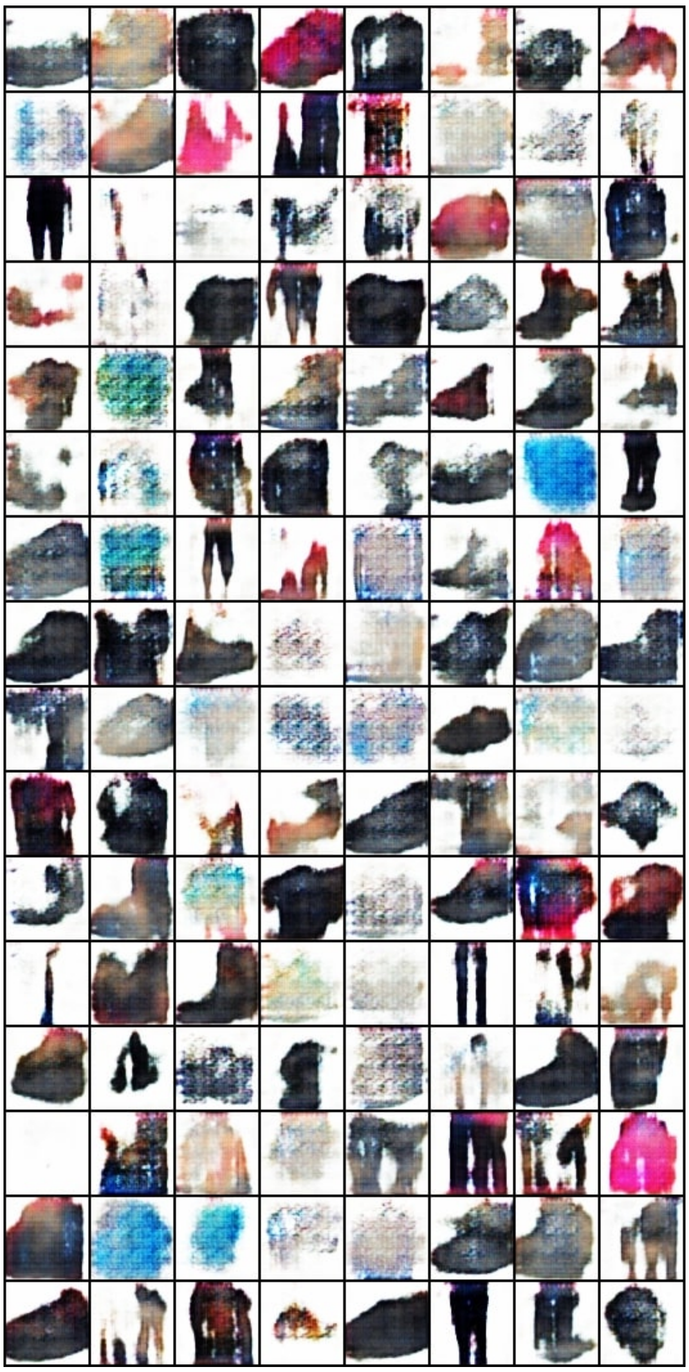


Figure 6.22: Amazon products fake samples at epoch 5

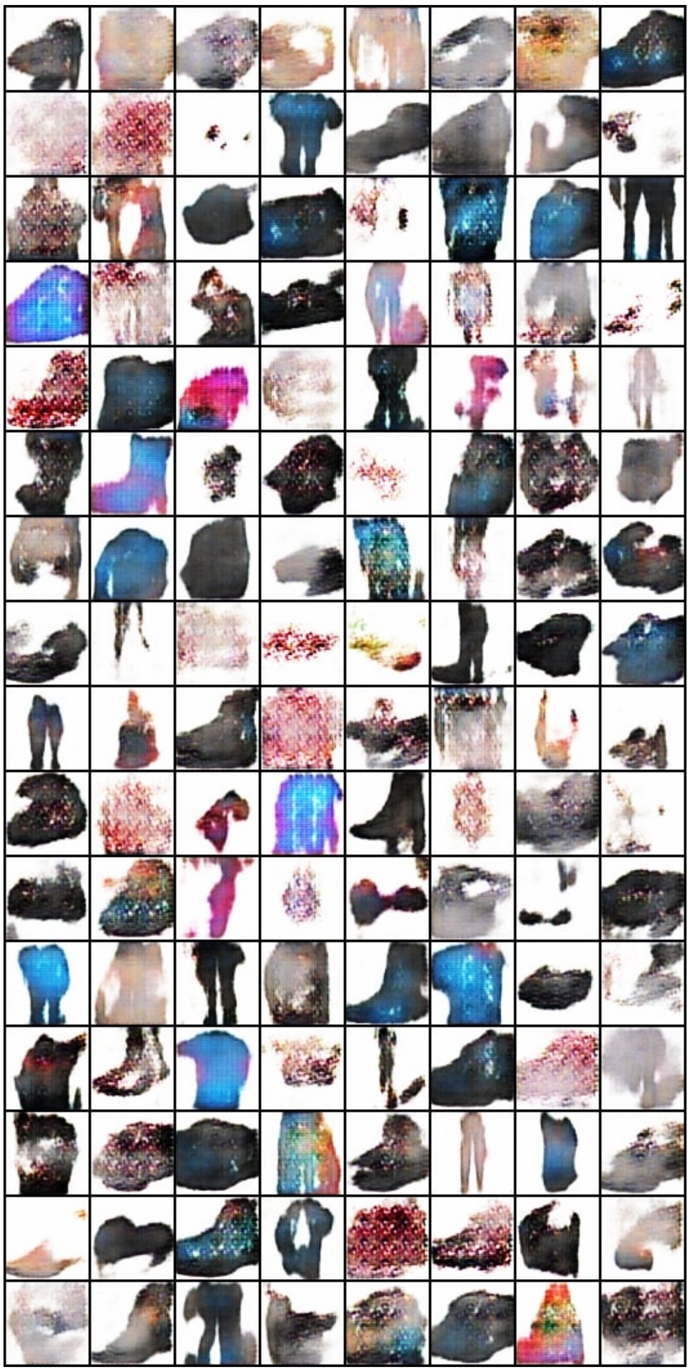


Figure 6.23: Amazon products fake samples at epoch 6

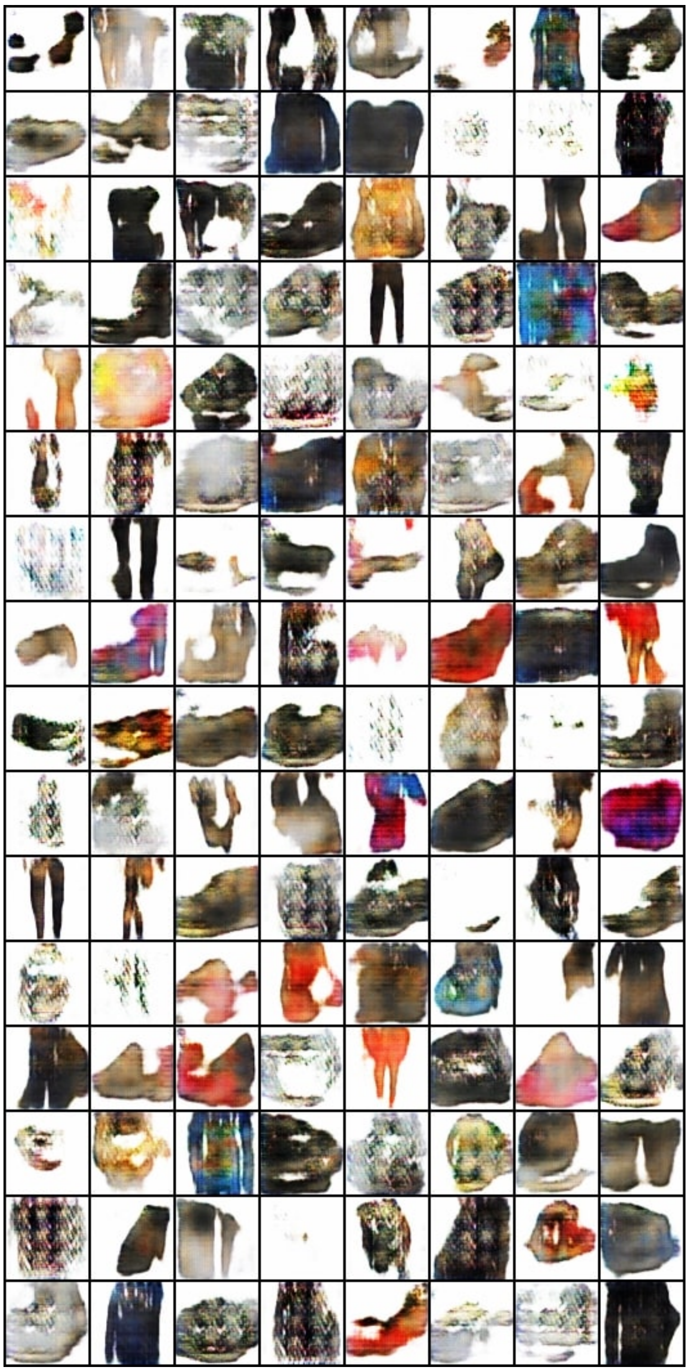


Figure 6.24: Amazon products fake samples at epoch 7

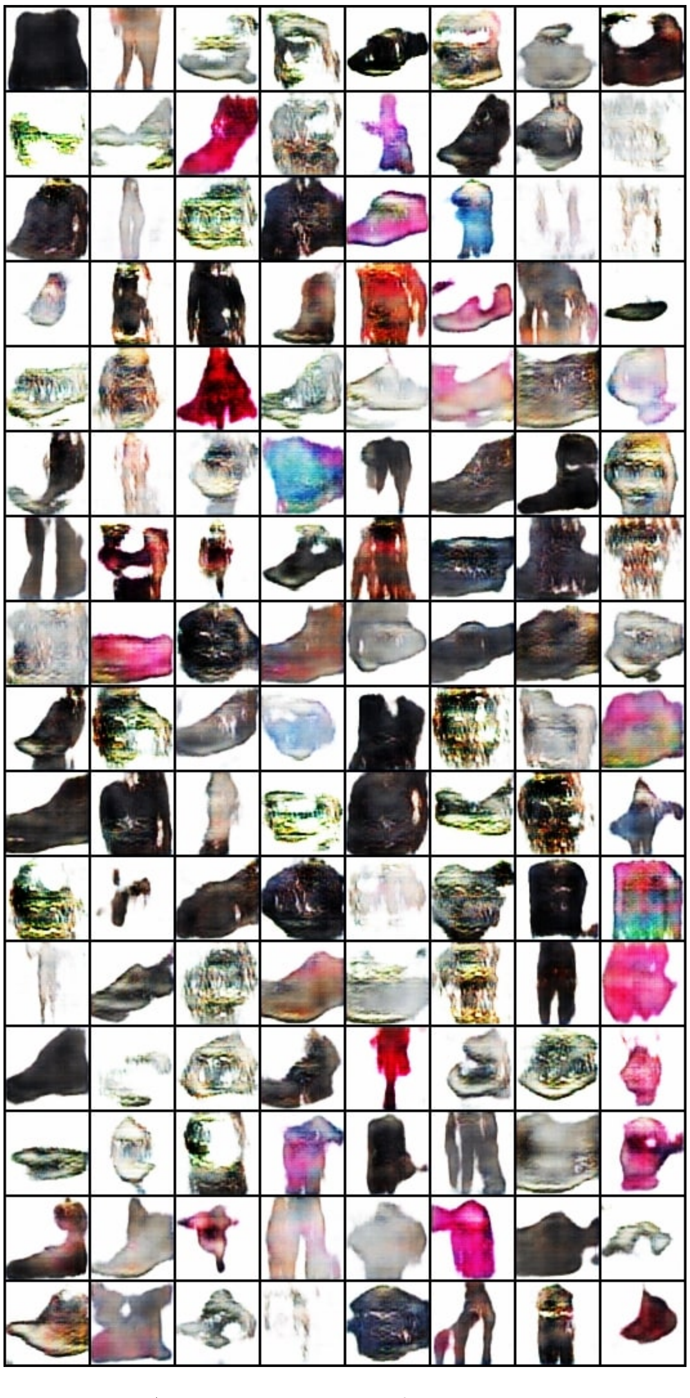


Figure 6.25: Amazon products fake samples at epoch 8

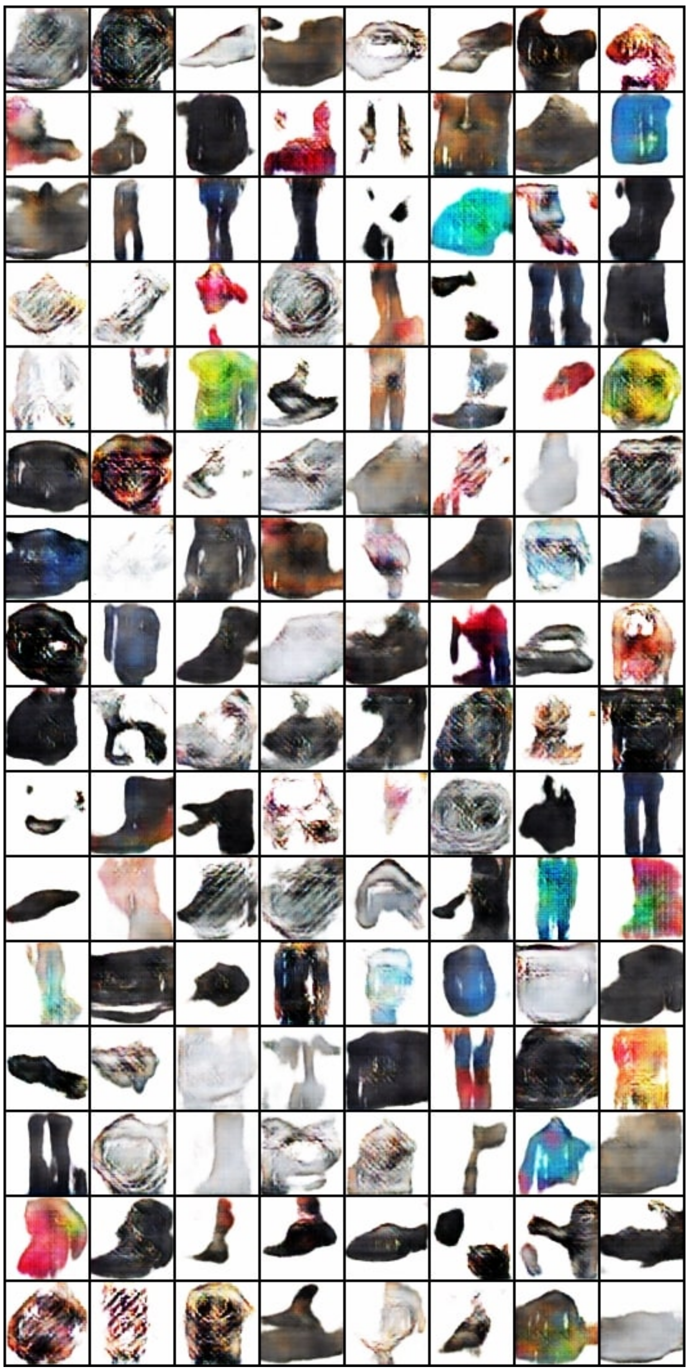


Figure 6.26: Amazon products fake samples at epoch 9

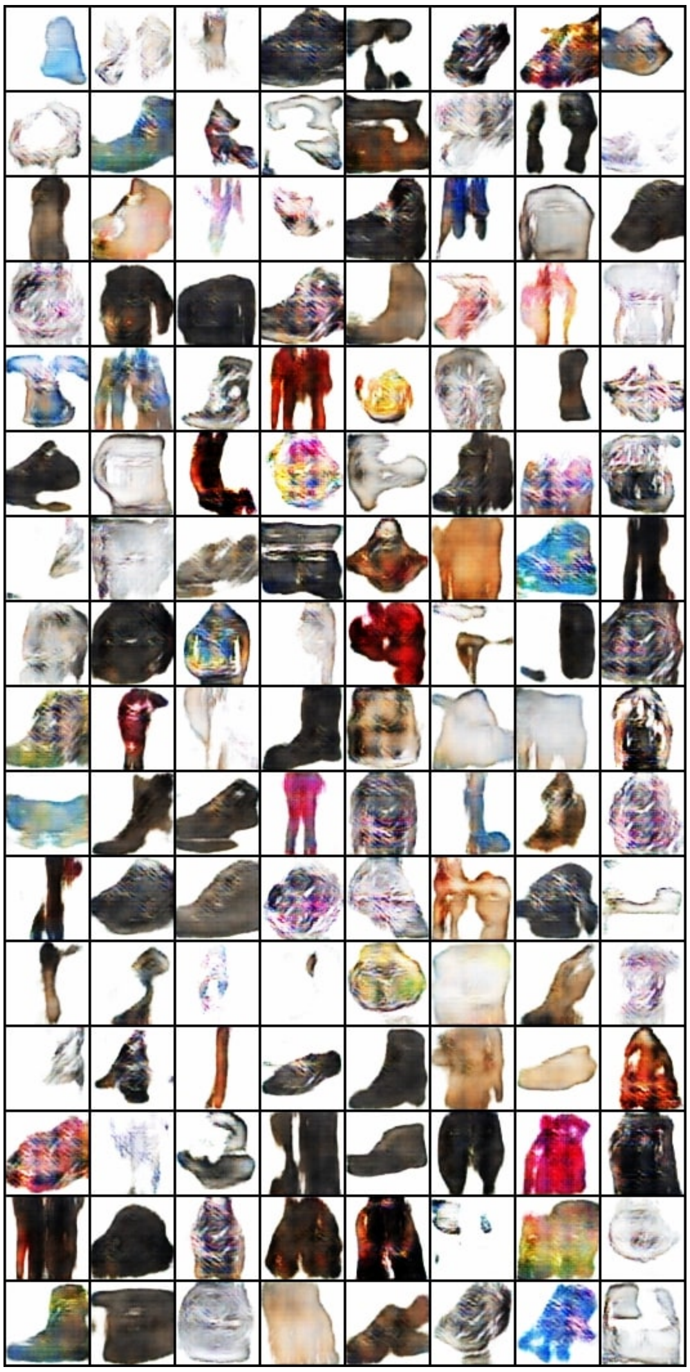


Figure 6.27: Amazon products fake samples at epoch 10

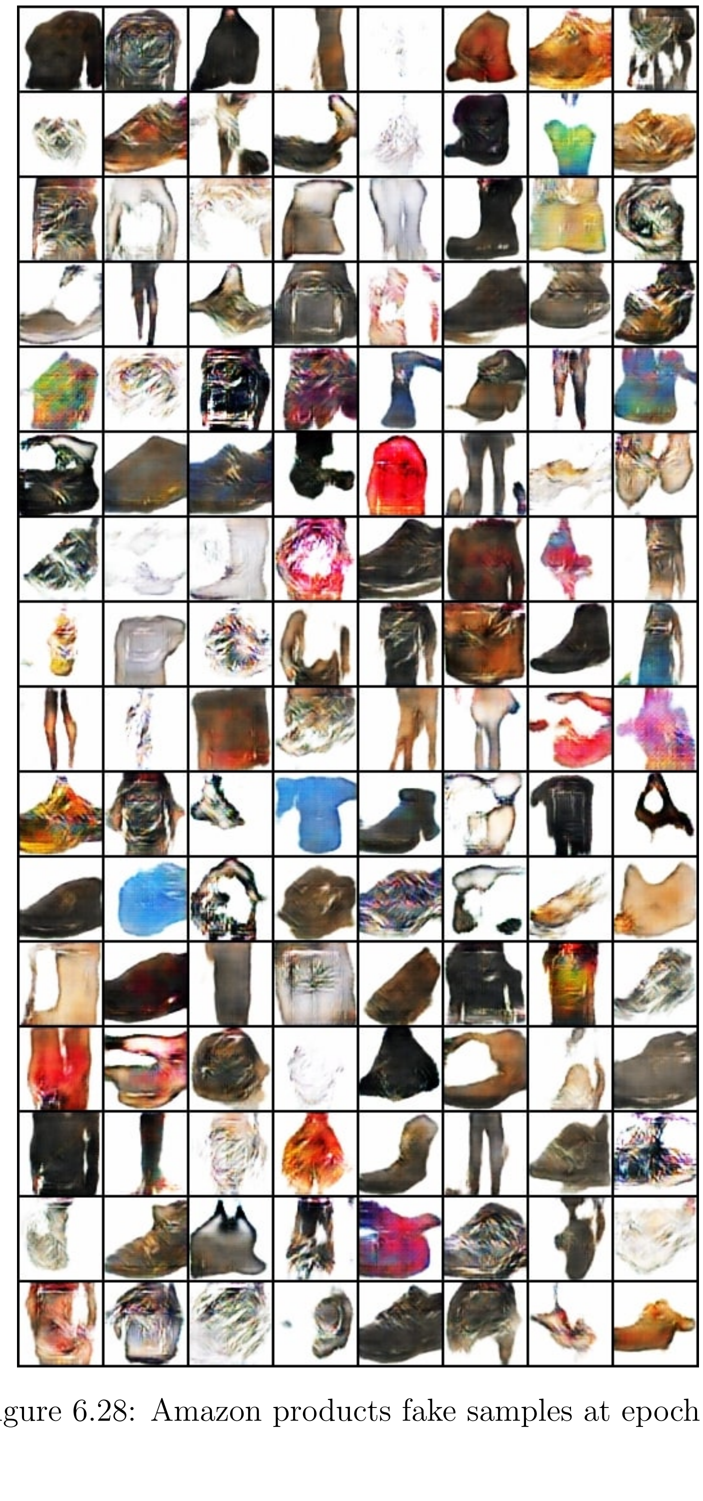


Figure 6.28: Amazon products fake samples at epoch 11

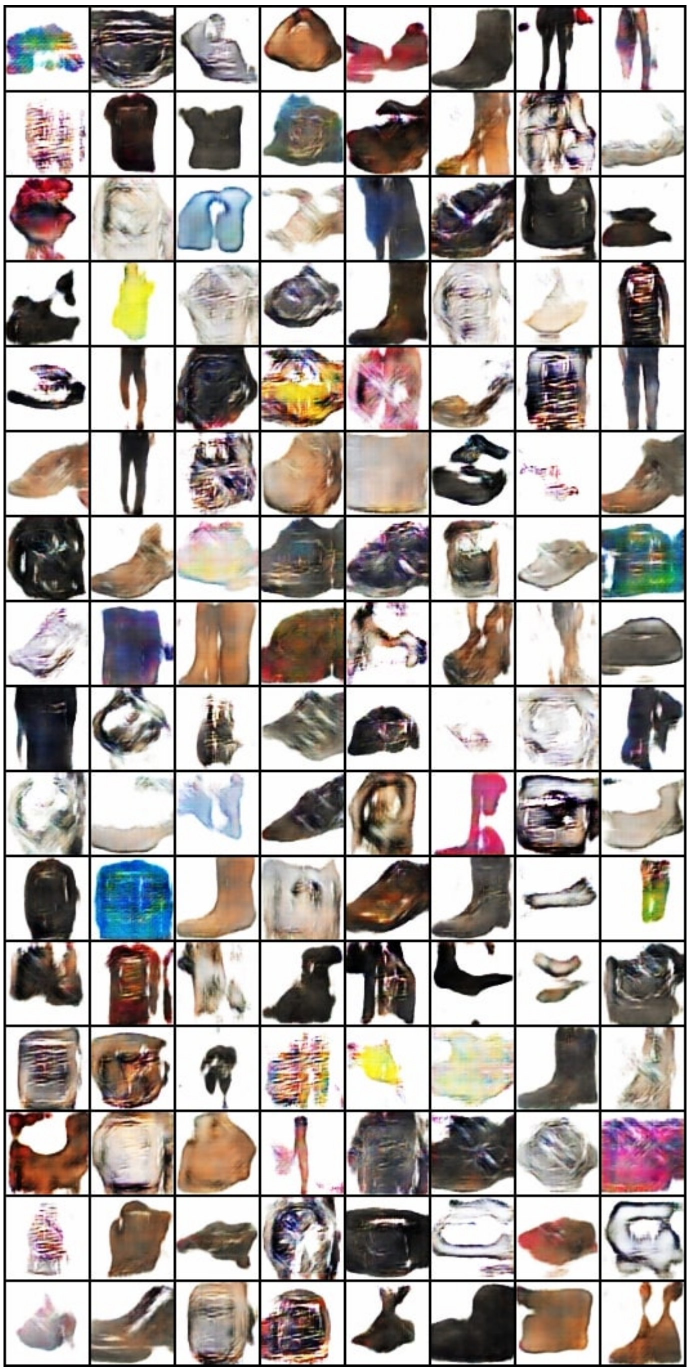


Figure 6.29: Amazon products fake samples at epoch 12

6.7 Possible future improvements with Style GAN

Cifar10, together with the MNist, are the most famous datasets because of their simplicity and indeed are almost always used in the simpler Deep Learning application. Moreover, a lot of papers which propose novel solutions attempt to demonstrate the effectiveness of these solutions on those datasets.
With this experiment we provided a simple GAN application and we tried to prove that our GAN can obtain good results even in a different context. In order to achieve these results, we processed and transformed our original dataset to reflect a structure close to the one of Cifar, in term of dataset size, image dimension and quality. In such a way we obtained a compromise between a reasonable training time and a reasonable quality of the results.
By exploiting more complex and advanced solutions it is possible to obtain even better result. For sake of completeness we want to briefly discuss about one new interesting kind of GAN which is the one applied in the online demo `https://www.thispersondoesnotexist.com/` [62], where each time you refresh the site, the algorithm will generate a new facial image from scratch.
The algorithm behind this demo comes from a very recent paper (6^{th} February 2019) and it's based on the idea of Style GANs [63], a solution motivated from the style transfer literature.
A Style-based generator differs from traditional generators in many aspects. First of all, traditional generator can be treated as black boxes in which the input is simply the noise vector (alongside a target label in case of Conditional GANs) and the output is an image.
In Style-based generator, the input is not submitted only to the first layer but it is re-submitted at each convolutional layer, so that also the intermediate layers are conditioned by external stimuli.
Style GANs, such as Conditional GAN or InfoGANs, need for a latent code in order to prevent from entangled representations. This latent code first passes through a non-linear mapping network composed of eight fully connected layers.
This operation allows to map the latent code into an intermediate latent space W and extract learned affine transformations which are fed

to the synthesis network generator before each convolution.
The Gaussian noise comes into play after the convolutions, just before evaluating the non-linearity.

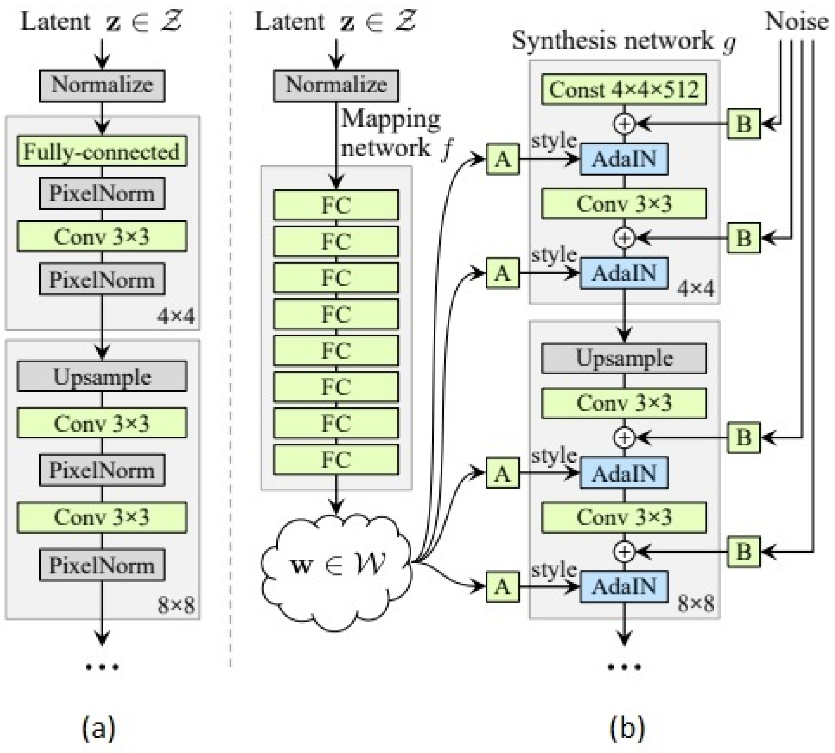


Figure 6.30: Traditional GAN (a) vs. Style GAN (b)

Some interesting outcomes from the Style GAN solution can be useful also for our Amazon dataset. In particular, the generated images exhibit a better quality respect to the inputs.
Furthermore, by running two latent codes it is also possible to perform some kind of style mixing effects. This can be interesting in case we want to obtain a new product as a mixed hybrid of two popular products.
Style GAN has been tested on different dataset, especially human faces, and one of the demo is also available online, as stated above.

We conclude showing some interesting face combinations coming from the experiments about style mixing using two latent codes.

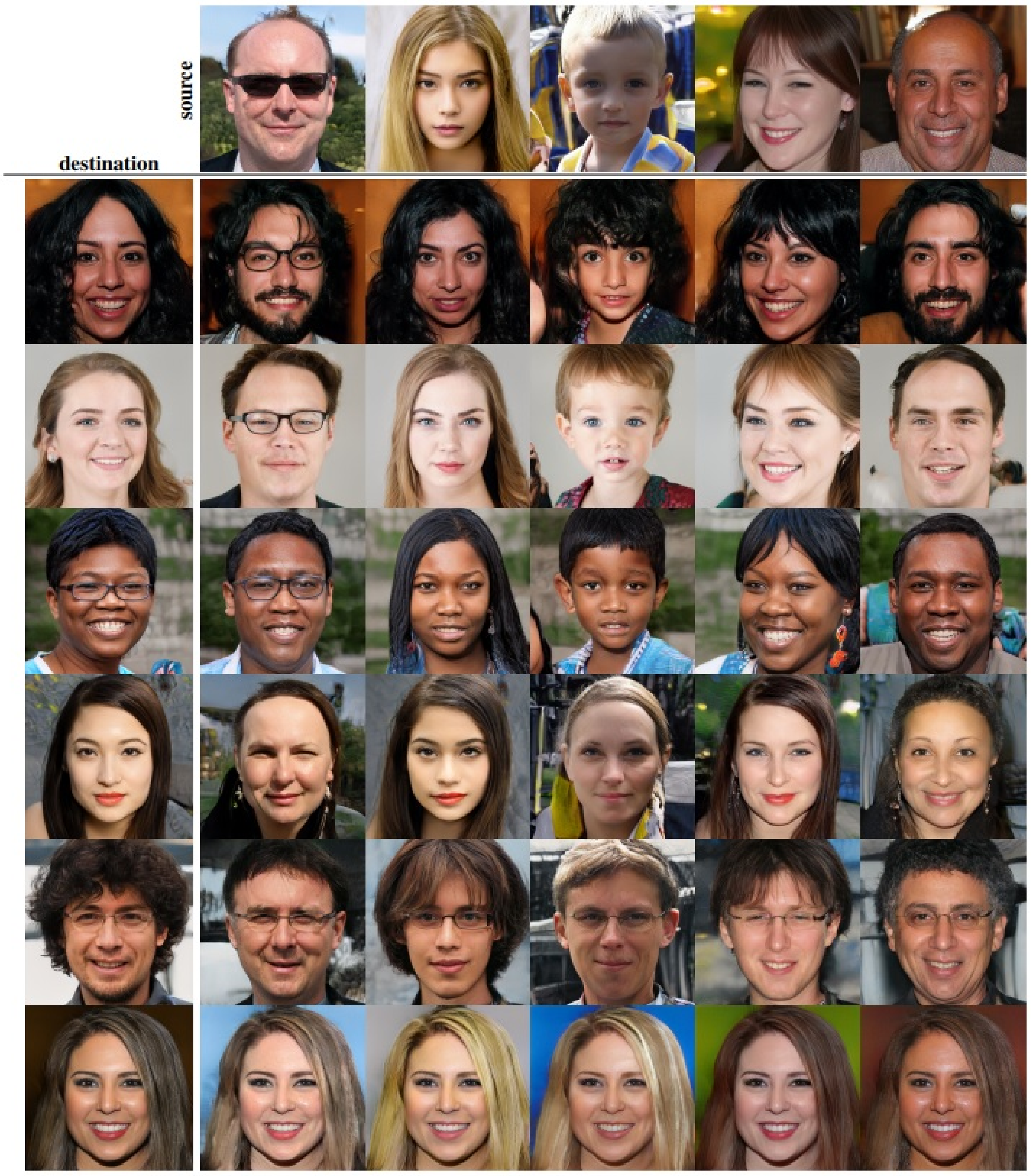


Figure 6.31: Style GAN results for face mixing

Conclusion

Can AI exhibit artistic abilities comparable to the human ones?
This is the question we posed at the introduction of the thesis.
The argument can be addressed from very different prospective such as philosophical, mathematical, aesthetic and so on.
We do not presume to give an answer to a so delicate question and we would like to leave the reader to take its own position about it.
With this thesis we hope we provided to the reader the means to think about the possible implication of GAN for the progress of Artificial Intelligence.
But before concluding we invite you to think about the following scenario. Take the picture of the "Portrait of Edmond Belamy" (Figure 4.4) and show it to any person who has never seen it. Probably the first question you will hear is "Who is the painter?".
The form of the question would suggest that the person is taking for granted the artist is a human being.
That's the point.
The answer is not a "who", but a "what".

Acknowledgements

This thesis represents for me the bridge between my academic path and my professional career.
For this reason I would like to mention both the University of Bologna and the AKKA Technologies Group.
I wish to thank my thesis advisor, Professor Gianluca Moro, for the support, the encouragement and the professionalism demonstrated in these months.
A special thank also to the DISI (Dipartimento di Informatica - Scienza e Ingegneria) department for having provided me the best hardware resources, and to the Professor Julian McAuley of the Computer Science Department of the University of California San Diego (UCSD), who kindly provided the Amazon datasets, all this essential to accomplish the experimental part.
Many acknowledgements also to my company, AKKA Technologies Toulouse, which allowed me to complete this thesis in France. In particular, I would like to mention my business managers, Xavier Jalabert and Helene Posbic, and the AKKA Research department for having given me the great opportunity to expand my technical skills in the field of Data Science and Artificial Intelligence.
Thanks in addition to my french flatmates for putting up with me in the last stressful months.
Finally, last but not least, my parents and my sister Anna, to whom I dedicate this work. Thank so much for the love and the support during these years.

References

[1] Yoshua Bengio,
Learning Deep Architectures for AI,
Dept. IRO, Université de Montréal

[2] Yoshua Bengio,
Deep Learning of Representations for Unsupervised and Transfer Learning,
Dept. IRO, Université de Montréal

[3] Haohan Wang, Bhiksha Raj
On the Origin of Deep Learning,
Language Technologies Institute, School of Computer Science, Carnegie Mellon University

[4] Warren S. McCulloch, Walter Pitts
A Logical Calculus of the Ideas Immanent in Nervous Activity,
The bulletin of mathematical biophysics, Volume 5, 1943

[5] F. Rosenblatt
The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain,
Cornell Aeronautical Laboratory
Psychological Review Vol. 65, No. 6, 19S8

[6] Marvin L. Minsky, Seymour A. Papert
Perceptrons: An Introduction to Computational Geometry,
MIT Press, Cambridge, 1969

[7] Pierre Lison
An Introduction to Machine Learning,

Language Technology Group (LTG)
Department of Informatics
HiOA, October 3 2012

[8] Csaba Szepesvári
Algorithms for Reinforcement Learning,
Draft of the lecture published in the Synthesis Lectures on Artificial Intelligence and Machine Learning series by Morgan Claypool Publishers
June 9, 2009

[9] Yann LeCun, Léon Bottou, G. B. Orr, Klaus Robert-Muller
Efficient BackProp,
1998

[10] Rohan Varma
Picking Loss Functions - A comparison between MSE, Cross Entropy, and Hinge Loss,
2018

[11] G. B. Orr
A list of cost functions used in neural networks, alongside applications,
`https://stats.stackexchange.com/questions/154879/`
`a-list-of-cost-functions-used-in-neural-networks-alongside-applications`
CrossValidated, 2015

[12] G. B. Orr
Removing noise in on-line search using adaptive batch sizes,
Department of Computer Science
Willamette University, 1997

[13] Luminita State, Catalina Lucia Cocianu, Vlamos Panayiotis
Network for Principal Component Analysis with Applications in Image Compression,
2007

[14] Robert A. Jacobs
Increased Rates of Convergence Through Learning Rate Adaptation,

Department of Computer Information Science
University of Massachusetts, 1987

[15] John Duchi, Elad Hazan, Yoram Singer
Adaptive Subgradient Methods for Online Learning and Stochastic Optimization,
Journal of Machine Learning Research, 2011

[16] Diederik P. Kingma, Jimmy Lei Ba
ADAM: A Method for Stochastic Optimization,
ICLR, 2015

[17] Xavier Glorot, Antoine Bordes, Yoshua Bengio
Deep sparse rectifier neural networks,
Université de Montréal, 2011

[18] Sepp Hochreiter, Jurgen Schmidhuber
Long short-term memory,
Neural Computation Vol. 9, 1997

[19] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, Yoshua Bengio
Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation,
2014

[20] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, Jeffrey Dean
Distributed Representations of Words and Phrases and their Compositionality,
2013

[21] Xiang Zhang, Junbo Zhao, Yann LeCun:
Character-level Convolutional Networks for Text Classification,
Courant Institute of Mathematical Sciences, New York University, 2016

[22] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner:
Gradient-Based Learning Applied to Document Recognition,
Proc. of the IEEE, 1998

[23] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, Zbigniew Wojna:
Rethinking the Inception Architecture for Computer Vision,
2015

[24] Lode Vandevenne:
Image Filtering,
Lode's Computer Graphics Tutorial

[25] C.-C. Jay Kuo:
Understanding Convolutional Neural Networks with A Mathematical Model,
Ming-Hsieh Department of Electrical Engineering
University of Southern California, Los Angeles, 2016

[26] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun:
Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification,
Microsoft Research, 2015

[27] Dominik Scherer, Andreas Muller, Sven Behnke:
Evaluation of Pooling Operations in Convolutional Architectures for Object Recognition,
20th International Conference on Artificial Neural Networks (ICANN)
Thessaloniki, Greece, September 2010

[28] Karen Simonyan, Andrew Zisserman:
Very Deep Convolutional Networks for Large-Scale Image Recognition,
Visual Geometry Group
Department of Engineering Science
University of Oxford, 2015

[29] Adam W. Harley:
An Interactive Node-Link Visualization of Convolutional Neural

Networks,
Department of Computer Science
Ryerson University
Toronto, Canada, 2015

[30] Ian J. Goodfellow:
Generative Adversarial Nets,
Departement d'informatique et de recherche opérationnelle
Universite de Montréal, 2014

[31] Yann LeCun:
What are some recent and potentially upcoming breakthroughs in deep learning?,
`https://www.quora.com/What-are-some-recent-and-potentially-upcomi`
2016

[32] Cade Metz:
How A.I. Is Creating Building Blocks to Reshape Music and Art,
The New York Times, 2017

[33] James Vincent:
How Three French Students Used Borrowed Code To Put The First AI Portrait In Christie's,
The Verge, 2018

[34] James Vincent:
Christie's sells its first AI portrait for $432,500, beating estimates of $10,000,
The Verge, 2018

[35] Robbie Barrat:
art-DCGAN,
`https://github.com/robbiebarrat/art-DCGAN`

[36] *Zero-sum game*,
`https://en.wikipedia.org/wiki/Zero-sum_game`

[37] David Pfau, Oriol Vinyals:
Connecting Generative Adversarial Networks and Actor-Critic

Methods,
Google DeepMind, 2017

[38] Christopher J.C.H. Watkins, Peter Dayan:
Q-Learning,
Machine Learning, 8, 279-292, 1992

[39] Jonathan Hui:
GAN - Why it is so hard to train Generative Adversarial Networks!,
Medium, Data Science, 2018

[40] Martin Arjovsky, Léon Bottou:
Towards Principled Methods for Training Generative Adversarial Networks,
ICLR, 2018

[41] Look Around You: : Season 1 Pilot - Calcium (2002)
`https://www.youtube.com/watch?v=FBaVwwuErmU`

[42] Luke Metz, Ben Poole, David Pfau, Jascha Sohl-Dickstein:
Unrolled Generative Adversarial Networks,
ICLR, 2018

[43] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, Sepp Hochreiter:
GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium,
LIT AI Lab & Institute of Bioinformatics,
Johannes Kepler University Linz
Austria, 2017

[44] Frans A. Oliehoek, Rahul Savani, Jose Gallego, Elise van der Pol, Roderich Groβ:
Beyond Local Nash Equilibria for Adversarial Networks,
2018

[45] Thomas Unterthiner, Bernhard Nessler, Calvin Seward, Gunter Klambauer, Martin Heusel, Hubert Ramsauer, Sepp Hochreiter:
Coulomb GANs: Provably Optimal Nash Equilibria via Potential

Fields,
LIT AI Lab & Institute of Bioinformatics,
Johannes Kepler University Linz
Austria, 2018

[46] Scott Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele, Honglak Lee:
Generative Adversarial Text to Image Synthesis,
University of Michigan, Ann Arbor, MI, USA
Max Planck Institute for Informatics, Saarbrucken, Germany
2016

[47] Maria-Elena Nilsback, Andrew Zisserman:
Automated flower classification over a large number of classes,
Proceedings of the Indian Conference on Computer Vision
Graphics and Image Processing, 2008

[48] Wah C., Branson S., Welinder P., Perona P., Belongie S.:
The Caltech-UCSD Birds-200-2011 Dataset,
Computation & Neural Systems Technical Report
CNS-TR-2011-001

[49] Mehdi Mirza, Simon Osindero:
Conditional Generative Adversarial Nets,
Departement d'informatique et de recherche opérationnelle
Universite de Montréal, 2014

[50] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, Alexei A. Efros:
Image-to-Image Translation
with Conditional Adversarial Networks,
Berkeley AI Research (BAIR) Laboratory,
UC Berkeley, 2018

[51] Phillip Isola:
pix2pix,
`https://github.com/phillipi/pix2pix`

[52] Josiah Alan Brooks (Jazza):
ARTIST Vs. PIX2PIX - Is this HUMOR or HORROR?!,
`https://www.youtube.com/watch?v=9cgFPttB_RQ`

[53] Christopher Hesse :
Image-to-Image Demo
Interactive Image Translation with pix2pix-tensorflow,
`https://affinelayer.com/pixsrv/`

[54] Sung Cheol Park, Min Kyu Park, Moon Gi Kang:
Super-Resolution Image Reconstruction: A Technical Overview,
IEEE Signal Processing Magazine
Volume: 20, Issue: 3, May 2003

[55] Christian Ledig, Lucas Theis, Ferenc Huszar, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, Wenzhe Shi:
Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network,
2017

[56] William Lotter, Gabriel Kreiman, David Cox:
Unsupervised Learning of Visual Structure using Predictive Generative Networks,
Harvard University of Cambridge
ICLR 2016

[57] R. He, J. McAuley:
Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering,
WWW, 2016

[58] J. McAuley, C. Targett, J. Shi, A. van den Hengel:
Image-based recommendations on styles and substitutes,
SIGIR, 2015

[59] Andres Torres:
Finding Duplicate Files with Python,
`https://www.pythoncentral.io/`
`finding-duplicate-files-with-python/`

[60] Hadelin de Ponteves, Kirill Eremenko:
Image Creation with GANs,

`https://www.superdatascience.com/pages/computer-vision`

[61] Alex Krizhevsky:
Learning Multiple Layers of Features from Tiny Images,
Department of Computer Science
University of Toronto, Canada, 2009

[62] James Vincent:
ThisPersonDoesNotExist.com uses AI to generate endless fake faces,
The Verge, 15th Feb 2019

[63] Tero Karras, Samuli Laine, Timo Aila:
A Style-Based Generator Architecture for Generative Adversarial Networks,
NVIDIA, February 2019

List of Figures