POLITECNICO DI TORINO

Master degree course in Computer Engineering

Master Degree Thesis

# Mitigating the catastrophic forgetting effect using generative models

**Supervisor**
prof. Elisa Ficarra

**Candidate**
Gianpaolo Bontempo
matricola: 243841

Anno accademico 2019-2020

# Acknowledgements

To those who have been close to me and supporting me.

## Abstract

The catastrophic-forgetting effect is the inability of a neural network to remember objects already seen when it encounters new classes as in an incremental learning approach. There have been many attempts to mitigate it such as increasing the number of parameters of a model or keeping the most significant samples in memory for each class. However, it requires too much memory. For the following reason, the goal of this research was to address the problem without external memory support but rather through the use of generative models such as VAE and PGGAN. Then a model is proposed that can explore the limits of memory replay combined with knowledge distillation. It consists in transferring knowledge from a teacher model to a student when it comes to pseudo-images. The research was conducted on several datasets such as MNIST, CIFAR10 and a biological dataset of different types of colorectal cancer

# Summary

## Introduction

In recent years, Deep Learning (DL) has become the state-of-the-art in many computer visions challenges. Conventionally a DL model elaborates a large dataset, learning how to retrieve relevant information, with the aim of properly classify the training data. The trained model is then applied to never seen samples, i.e. the test set. In such a learning process, the dataset fed to the model represents a domain of knowledge which does not change over time: given L classes, the trained model can classify them, but its behaviour in case of the new class L + 1, is not straightforward. Furthermore, in a real-world scenario, we need an incremental learning paradigm, able to process training examples that become available over time, modifying the learned knowledge accordingly. Think for example of biology. This is a field in constant evolution and training a DNN without mechanisms suitable for incremental learning can take longer to train compared to actual use. Furthermore, for privacy reasons, the cumulative saving of personal data is not possible for long periods. With Incremental Learning (IL), literature states exactly the DL research branch tailored to design methodologies and models able to integrate the previous acquired knowledge, with samples that became available dynamically, thus extending the domain of knowledge of the given model. The most intuitive way to achieve this behaviour could be a fine-tuning process on the new extended dataset, starting from the weight's configuration in the previous stage. Nevertheless, the alteration caused by weights update on the net parameters, affects the model globally, destroying existing features learned from earlier data. The inherent tendency of a neural network to lose the previously acquired knowledge, when new classes are added, is well-know in literature with the name of catastrophic forgetting (CF). The main purpose of the present thesis is first an investigation of the several state-of-the-arts method for IL, and then the design and development of a new IL strategy, based on the so-called generative models, which are going to be detailed in the following paragraph.

## Background

In the last years, many approaches have been investigated to mitigate CF. Then, the current research on IL may be summarized into three main families of methodologies, namely: i) architectural strategies, which means increasing the number of weights of the model; ii) regularization strategies, consisting in a modified version of the loss function, tailoring it to preserve the “old important” weights; iii) rehearsal strategies, namely using a few representative images of the old classes to preserve knowledge. In the latter context, the generative models are currently receiving more and more attention.With generative model we intend a Neural Network capable of generating new images by receiving random samples of distributions as input. In an incremental learning approach, they can be useful to replace the cumulative storage of samples. Among the most famous models, such as VAE and GAN, a recent and better model, named Progressive Growing of GANs (PGGAN), has been introduced. By training a generator to deceive a discriminator with false images, it is possible to generate good quality images by progressively increasing the resolution of the image.

## Proposed method

This thesis presents an IL solution which is a hybrid between rehearsal and regularization strategies, using images generated by generative models as pseudo-rehearsals of previous tasks. The contributions are: i) development of a progressive number of PGGAN generators to avoid the overlap of the features characterizing different classes; ii) creation of two different models performing the prediction and the memory replay; iii) development of a quality control module capable to filter among the generated images those more meaningful for the correspondent class. The result is then used to carry out the knowledge distillation.This technique requires that the judgment of a model trained on previous tasks, called teacher, is used as target to evaluate pseudo images. While, to learn new classes, a classic approach based on true images and labels is maintained.

## Results

The proposed method was tested on datasets characterized by different complexity, such as MNIST, CIFAR10 and a biological dataset created for the identification of colorectal cancer cells from other healthy cells. All the tests were performed in a context of class-Incremental Learning. Multi-layer perceptron (MLP) and pretrained ResNet networks were used as common feature extractor to perform fair comparisons. The results on MNIST were comparable with state-of-the-art methods. On other datasets, the proposed model performed better than the state-of-the art models that use generative models, but worse than those that maintain additional information on the class, such as the most significant exemplars. However, compared with the latest, the advantages of the proposed approach are: i) memory requirements independent of the size of the data; ii) network training capability on incremental classes without storing statistics or data used on previous training stages; this is particularly useful when the model is used by different users dealing with sensitive data, such as medical ones.

## Conclusion and future works

The proposed thesis investigated the applicability of generative models in an Incremental Learning context. For this purposed a method merging rehearsal and regularization strategies was designed and developed. In particular, images generated by PGGAN generative models were used as pseudo-rehearsals of previous tasks. In the proposed method the use of knowledge distillation in the solicitation of old tasks was a fundamental contribution. Although the results are not as good as hoped, these two components have great potential. Furthermore, filtering the generated images through the quality control module can be useful, especially when the dataset to be replicated is quite complex. Some future improvements could be: i) design of a dual memory approach by training the generators using only the images chosen from a faster model; ii) use better generative models in terms of quality and training speed.

# Contents

**List of Tables** 7

**List of Figures** 9

**1 Introduction** 11

**2 Background** 13
- 2.1 Neural Networks . . . 13
  - 2.1.1 Perceptron . . . 13
  - 2.1.2 Multi Layer Perceptron . . . 15
  - 2.1.3 Convolutional Neural Network . . . 16
- 2.2 Incremental Learning . . . 18
  - 2.2.1 Scenarios . . . 18
  - 2.2.2 Strategies . . . 19
  - 2.2.3 Architectural strategies . . . 19
  - 2.2.4 Regulation Strategies . . . 22
  - 2.2.5 Reharsal Strategies . . . 24
  - 2.2.6 Hybrids . . . 25
- 2.3 Tools for Memory replay . . . 29
  - 2.3.1 Auto Encoder . . . 29
  - 2.3.2 Generative models . . . 31

**3 Proposed model** 41
- 3.1 NVAE . . . 41
- 3.2 network of PGGAN . . . 43

**4 Results** 51
- 4.1 MNIST . . . 52
- 4.2 Cifar10 . . . 56
- 4.3 Bio-dataSet . . . 61

**5 Conclusions and future works** 65
5.1 Conclusion . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 65
5.2 Future Work . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 66

# List of Tables

2.1 Example the reconstraction of an Auto-encoder on biological dataset[Pon+19].reducing the latent space of an Auto-Encoder the quality of the image can be lower. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 30
2.2 This table presents a comparison between inputs (the first four lines) with the reconstructions (the last lines) of a variational auto-encoder. In particular, we want to rebuild cars and planes obtained by CIFAR10 with a and b respectively. . . . . . . . . . . . . . . . . . . . . 34
3.1 structure of a single vae . . . . . . . . . . . . . . . . . . . . . . . . . 42
3.2 Generation of all numbers using NVAE. For each generator, the input and output are illustrated by highlighting in yellow the cases in which they are the same . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 42
3.3 Convolutional AutoEncoder Structure . . . . . . . . . . . . . . . . . . 47
3.4 AutoEncoder Structure using resnet pretrained on imagenet . . . . 48
3.5 Structure of convolutional discriminator . . . . . . . . . . . . . . . . 48
3.6 Structure of MLP discriminator for MNIST dataset . . . . . . . . . 49
4.1 Results on different datasets using only generated images. . . . . . 51
4.2 Results on Mnist using a Multilayer Perceptron model as common discriminator on 5 task. . . . . . . . . . . . . . . . . . . . . . . . . . . 52
4.3 results on Mnist using a Multilayer Perceptron model as common discriminator on 2 task. . . . . . . . . . . . . . . . . . . . . . . . . . . 54
4.4 results on MNIST dataset using a MLP model as discriminator and only 5 tasks. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 54
4.5 results on Mnist using a convolutional model as discriminator. . . . 55
4.6 results on Mnist using a convolutional model as discriminator and only 2 tasks. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 55
4.7 results on MNIST dataset using a convoutional model as discriminator and only 5 tasks. . . . . . . . . . . . . . . . . . . . . . . . . . . 55
4.8 results on Cifar10 using pretrained resnet as common discriminator on 2 task. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 56
4.9 results on Cifar10 using pretrained resnet as common discriminator on 5 task. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 59

4.10 results on Cifar10 using a resnet model pretrained as discriminator and only 2 tasks.Using a quality control mechanism on the generated images the accuracy can improve. . . . . . . . . . . . . . . . . . . . 60
4.11 results on Cifar10 using a convulational model as discriminator . With this comparison, a degradation of performance was observed in the 5 tasks case, increasing the training time per task . . . . . . 61
4.12 results on Cifar10 using a convolutional model as discriminator and only 2 tasks. When the discriminator is not capable of extract all the features the quality controll is useless. . . . . . . . . . . . . . . . 62
4.13 results on Biological using pretrained resnet as common discriminator on 2 task. . . . . . . . . . . . . . . . . . . . . . . . . . . . . 62
4.14 results on Biological dataset using a resnet model as discriminator and only 2 tasks. . . . . . . . . . . . . . . . . . . . . . . . . . . . 63

# List of Figures

2.1 Example of perceptron[Hem+20] . . . . . . . . . . . . . . . . . . . 14
2.2 Example of multi-layer perceptron [Liu18] . . . . . . . . . . . . . . . 15
2.3 A visual representation of a convolutional layer([DV18] . . . . . . . 16
2.4 Example of residual block ([He+15]) . . . . . . . . . . . . . . . . . 18
2.5 the figure shows the accuracy of the CIFAR10 test set if the network is offline or without any incremental learning mechanism. it is therefore possible to see that the network is able to learn only the new classes without being able to remember anything of the previous ones 20
2.6 Generic Tree-CNN [RPR19] . . . . . . . . . . . . . . . . . . . . . . . 21
2.7 Illustration of NDL processing MNIST digits (orange/red circles indicate accuracte/inaccurate feature representations of the input; green indicate new nodes added via neurogenesis)[Dra+17]. . . . . . 26
2.8 FearNet[KK17] . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 27
2.9 mPFC [KK17] . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 28
2.10 image of an Undercomplete Auto-encoder([Cha+18] . . . . . . . . . 29
2.11 Comparison between VaE without reparametrization trick (left)and VAe using reparametrization trick(right)[Doe16]. They seems similar but only in the second the backpropagation works. . . . . . . . . . . 33
2.12 samples generated with Variational AutoEnconder On MNIST dataset. 35
2.13 Training of a PGGAN[Kar+18] model: It is based on training a gan by progressively adding multiple levels and simultaneously increasing the resolution of the image . . . . . . . . . . . . . . . . . . . . . . 37
2.14 Training of a PGGAN[Kar+18] model: In the phase of change between a lower resolution and a higher one, to make the transition gradual, the network is transformed into a residual network by attenuating the effect of the residual by an alpha coefficient (from 1 to 0). The figure shows the 3 different steps during this change. 2x and 0.5x are made with nearest neighbor filtering and average pooling, respectively.TORGB and FROMRGB are two layers that carry out a 1x1 convolution and are necessary to project the feature vector on the rgb channels and vice versa. . . . . . . . . . . . . . . . . . . . . . . 38
2.15 comparison with Biodataset [Pon+19] . . . . . . . . . . . . . . . . . 39

3.1 Proposed model trained on the first task. . . . . . . . . . . . . . . . . . 45
3.2 Proposed model trained on the n+1 task. . . . . . . . . . . . . . . . . . 45
3.3 scheme of the auto-encoder + discriminator . . . . . . . . . . . . . . 46
4.1 This diagram shows the progress of the accuracy on MNIST. In this case the dataset was divided into 5 different tasks which are given to the model individually.This result has been obtained using a MLP model. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 53
4.2 generated images using a Pgan model per class . . . . . . . . . . . . . 57
4.3 Accuracy On CIFAR10 using resnet pre-trained as encoder on 2 tasks. 58
4.4 Accuracy On CIFAR10 using resnet pre-trained as encoder on 5 tasks. 59
4.5 Icarl:Accuracy using resnet pre-trained on CIFAR10 on 5 tasks . . . 60
4.6 Icarl: Accuracy using resnet pre-trained on CIFAR10 on 2 tasks. . . 61

# Chapter 1

# Introduction

The human brain is able to recognize most of the objects from experience.
It can learn different characteristics of an object by interacting with it in different ways. This result is obtained from the collaborations of many neurons.
Each of these, receiving an electrical impulse, if strong enough, consequently generates another impulse which in turn will be received by another neuron. This happens until a single conclusion is reached.
Machine learning was designed to imitate these abilities. Thanks also to the advent of convolutional neural networks, which allowed to extrapolate characteristics in a hierarchical way, it was possible to have enormous successes in the field of image recognition.

In recent years, Deep Learning (DL) has become the state-of-the-art in many computer visions challenges. Conventionally a DL model elaborates a large dataset, learning how to retrieve relevant information, with the aim of properly classify the training data. The trained model is then applied to never seen samples, i.e. the test set. In such a learning process, the dataset fed to the model represents a domain of knowledge which does not change over time: given L classes, the trained model can classify them, but its behaviour in case of the new class L + 1, is not straightforward. However, there are still big gaps when we change the learning typology in a sequential way.
In fact there have been many attempts that have tried to have this approach without using cumulative savings of data but all have presented the same flaw in the end: the catastrophic forgetting.(Kirkpatrick et al.) It's the inability of a neural network to remember objects already seen when it encounters new classes.
Other approaches instead, exploiting a growth of the model(Progressive neural network [Rus+16] or using an external memory(as Icarl [Reb+16]), have managed to obtain better results with the only problem that the memory requirements can become unsustainable.

The ability to consolidate memories in such a way that objects can be recognized sequentially is the goal of Incremental Learning.
This branch aims to make inference with a sequential approach. All this requires a change to the ML models as they have been created specifically to simultaneously inference on multiple labels. For this reason, the evolution of machine learning is first explored.
Many applications require a neural network suitable for Continual Learning. In robotics([RLS]) the possibility of having to remember an object seen in the past is high, especially because a robot can move and not always detect the same object.
In biology, where knowledge is constantly evolving, a model is needed that can learn new types of cells without necessarily having to train again on old types already seen.
Many of the current models have taken inspiration from biological studies. Between these, however, what has remained of greatest interest is the brain. This is in fact made up of billions of neurons that communicate together through connections called synapses. When the brain has to learn something, this network of neurons is activated through electrical impulses. Thus, each neuron gradually learns how to behave on the next impulse.

Mathematical models such as neural networks have proven extremely useful in the field of biology. it is in fact essential to find a way to identify important pathologies hardly visible by other instruments from blood samples or cell images. Through machine learning it was possible to obtain excellent results, allowing non-invasive exams to be carried out.

However, although the enormous progress, there are still many critical issues. The knowledge of all pathologies is not complete but grows progressively. A model is therefore needed with the ability to learn new pathologies without forgetting previous memories without being forced to review the past as well. This would imply a big waste of time and would force to keep all the samples in memory. The need for memory becomes critical in many fields like in biology where the number of pathologies is very large.

# Chapter 2

# Background

## 2.1 Neural Networks

This is made up of a large number of interconnected units that work in parallel to obtain a result. Thanks to this structure it is therefore possible to find the connections between the different characteristics of the data so as to be able to distinguish them.
These structures can therefore be used for paradigms such as Supervised Learning (looking for a function that can combine input with the right output), Unsupervised Learning (looking for useful patterns to distinguish groups of data).

### 2.1.1 Perceptron

Perceptron can be considered as one of the first neural network models. This is in fact a single layer feed-forward neural network and is able to distinguish linearly separable characteristics.
Its algorithm simply consists of updating the weights of a learning rate whenever it is wrong to classify. This algorithm is functional if and only if the data are linearly separable as demonstrated by the convergence theorem([KS15]). The latter presents the existence of a finite number of iterations if the data are linearly separable.

Assuming $\mathbf{w} \in R^D$ a normal vector defining a hyperplane,using b as bias and $\mathbf{x} \in R^D$ as input sample, it is possible define a function f as

$$f(x) = h(b + \sum_{i=1}^{D} w_i x_i)$$

where h is the activation function which returns a value only if the input has exceeded a certain threshold
If there is also a label t = - 1,1 along with the input, then you can define an error

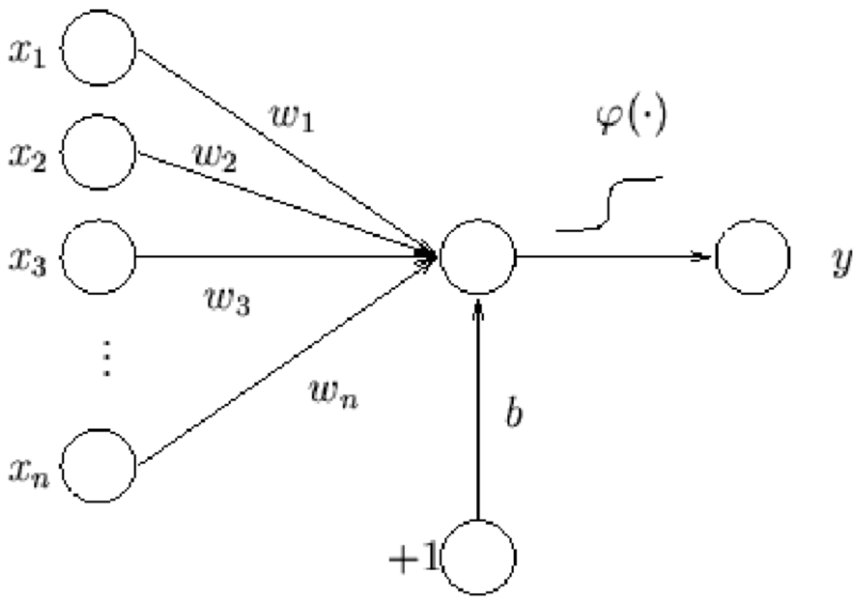


Figure 2.1. Example of perceptron[Hem+20]

function such as:

$$E(\mathbf{w}) = -\sum_{i\in\mathcal{M}} t_i \mathbf{w}^\top \mathbf{x}_i$$

Considering $\mathcal{M}$ as the set of all points incorrectly classified.

### Stochastic Gradient Descent

This is used to minimize the error function. Assuming to have a strictly convex function, it is possible to find out in which direction the minimum is found by using its derivative.
It is based on updating the weights each time an error is made.

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \eta \nabla E(\mathbf{w})$$

$\eta$ is the learning rate. It Represents how big the step is with each update. If large it makes the algorithm faster but at the same time it can prevent the algorithm from converging. If we consider the Error function as the mean square error between the target t and the predicted label o then:

$$\frac{\partial E_D[\vec{w}])}{\partial w_j} = \frac{\partial}{\partial w_i}\frac{1}{2}\sum_d (t_d - o_d)^2$$

By deriving it is possible to separate the derivable part from what is not

$$= \frac{1}{2} \sum_d 2 \left( t_d - o_d \right) \frac{\partial}{\partial w_i} \left( t_d - o_d \right)$$

$$= \sum_d \left( t_d - o_d \right) \left( - \frac{\partial o_d}{\partial w_i} \right)$$

however it is possible to solve the equation by making a change of variables

$$= - \sum_d \left( t_d - o_d \right) \frac{\partial o_d}{\partial net_i} \frac{\partial net_d}{\partial w_i}$$

$$= - \sum_d \left( t_d - o_d \right) o_d \left( 1 - o_d \right) x_d^i$$

## 2.1.2 Multi Layer Perceptron

When, on the other hand, the data are not linearly separable, there is a need for an equal non-linear function. Its realization is possible through a neural network consisting of several layers arranged sequentially. In this way the output of a layer becomes the input of the next layer. In this case, to understand how to update the

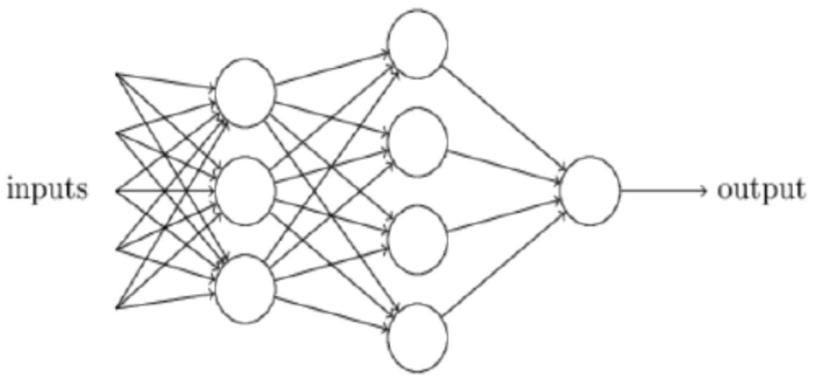


Figure 2.2. Example of multi-layer perceptron [Liu18]

weights of all the layers, it is necessary to take the mean squared error as a cost function.

### 2.1.3 Convolutional Neural Network

The structure has been of particular interest in the area of image recognition.
Using this structure it is possible to hierarchically extrapolate different types of characteristics from the lowest to the highest levels.
A peculiarity is that it uses the same parameters (which make up a kernel) on different areas of the input.
After extracting the characteristics, a fully-connected layer is used as the last layer in order to predict the class the image belongs to.

#### Convolutional Layer

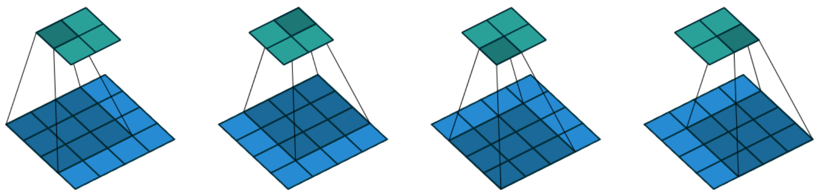


Figure 2.3. A visual representation of a convolutional layer([DV18]

$$h_j^n = \max\left(0, \sum_{k=1}^{K} b_k^{n-1} * w_{kj}^n\right)$$

#### Pooling Layer

It has the role of downsample the input, consequently allowing positive effects such as: r

- greater generalization of the image preventing the problem of overfitting
- reducing the parameters necessary for the next layer

#### Activation Function

As previously mentioned, the activation function plays a very important role in the network. However, depending on the choice you want to make, you can get different effects. Although the most obvious function may be the sigmoid function $f(x) = (1 + e^{-x})^{-1}$, this could attenuate very important nonlinear characteristics. For this reason, the relu function is more useful $f(x) = \max(0, x)$.

### Batch Normalization

In the intermediate layers, together with the activation function it is good practice to normalize the values in order to control overfitting by increasing the generalization. It is also useful for preventing gradients from becoming too large.
Considering in fact $I_{b,c,x,y}$ and $O_{b,c,x,y}$ inputs and outputs respectively, c the channels, b the batch and x and y two spatial dimensions then

$$O_{b,c,x,y} \leftarrow \gamma_c \frac{I_{b,c,x,y} - \mu_c}{\sqrt{\sigma_c^2 + \epsilon}} + \beta_c \quad \forall b, c, x, y$$

### Transposed Convolutional layer

it is usually used when an output larger than the input is desired. Its use is used in the construction of decoders to reconstruct images.

### ResNet

In the previously discussed model, increasing the level of depth means increasing the levels of abstraction.
However, after a certain point, a certain type of degradation occurs, not caused by overfitting but precisely by the depth of the model.
Discovered this problem, a new approach was proposed residual learning. The results turned out to be excellent, even winning the first place in the ILSVRC 2015 classification competition.
Main difference from one from a convolutional network is the addition of shortcut connections by creating residual blocks in the structure.[He+15]

This mechanism involves combining the output of each layer with its input. When these layers have a shortcut far from zero, this becomes a residual function. A peculiarity of this method is that to obtain a good accuracy it is not necessary to have a model of an excessive number of layers.
A residual function is defined as

$$F(x) := H(x) - x$$

where H represents the result of a few stacked layers.The output y can be obtained as

$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + \mathbf{x}$$

In this way the solvers have the possibility of bringing the weights initially to zero towards identity mappings. When, for various reasons, the size of the input is different from that of the output, a linear projection is performed

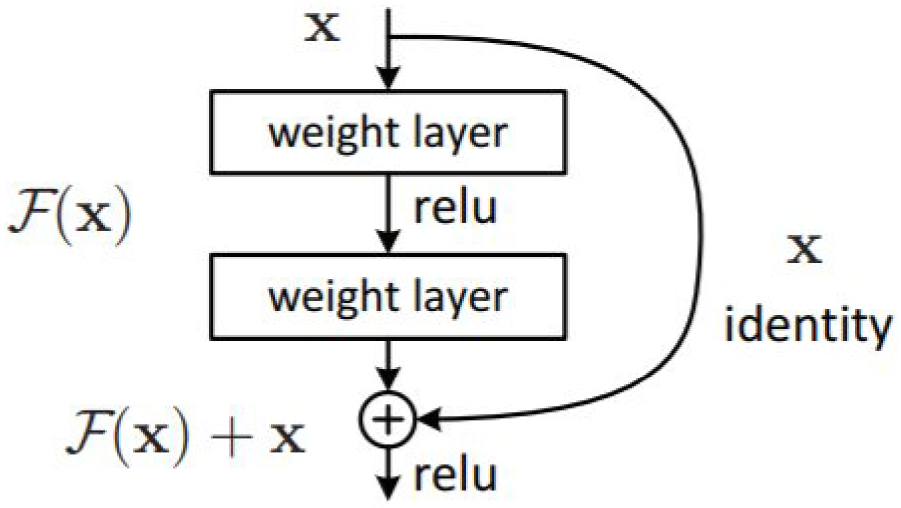

x
weight layer
relu
weight layer
$\mathcal{F}(\mathbf{x})$
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Figure 2.4. Example of residual block ([He+15])

## 2.2 Incremental Learning

### 2.2.1 Scenarios

Classes are addressed in task groups. As suggested by [VT18]and [ML19] The main scenarios are of three types :

- Task-incremental learning
- Domain-incremental learning
- class-incremental learning

#### Task Incremental Learning

Given the task it is required to predict which class of the task the sample belongs to.

#### Domain Incremental Learning

Typical examples of this scenario are protocols whereby the structure of the tasks is always the same, but the input-distribution is changing. A relevant real-world example is an agent who needs to learn to survive in different environments, without the need to explicitly identify the environment it is confronted with.

#### Class Incremental Learning

This turns out to be the most complicated of scenarios because inference is required on all classes without any information regarding the task. However, this case is the one that most represents reality.

### 2.2.2 Strategies

This type of research has been developed mainly in recent years thanks to the advent of DNN. For this reason it is still a developing branch. In any case, until now three main types of strategies have been formed:**architectural, by regulation or by rehearsal**. The first of these is proposed as a method capable of completely removing the effect of catastrophic forgetting at the cost of exponentially increasing the number of parameters. The remaining two instead seek a compromise between time and memory, mitigating the catastrophic forgetting.

#### Full Rehearsal

This type of technique involves the accumulative saving of all classes. This is then combined with the new task and the model is trained again on the newly created dataset. Although it can be defined as upper-bound in terms of precision, it is inefficient in terms of memory and time and it is difficult to consider it as a method for continual learning

#### Naive

This approach involves the use of a standard classification algorithm trained sequentially on different classes. Obviously, since it has no mechanism to prevent catastrophic forgetting, its ability to remember will be hopelessly limited.This therefore represents the lower-bound of continual learning.

### 2.2.3 Architectural strategies

Architectural techniques involve changing the architecture of the model such as the number of parameters, connections and their freezing in order to slow down the model to forget old classes.

#### PNN

PNN ([Rus+16]is one of the best architectural models immune to catastrophic forgetting and with explicit support to tackle different sequences of tasks in a sequential manner (multiple tasks).One of the main applications for which it has been developed is reinforcement learning. It starts from a single column made of a deep neural network having L layers. When switching to a new task the parameters

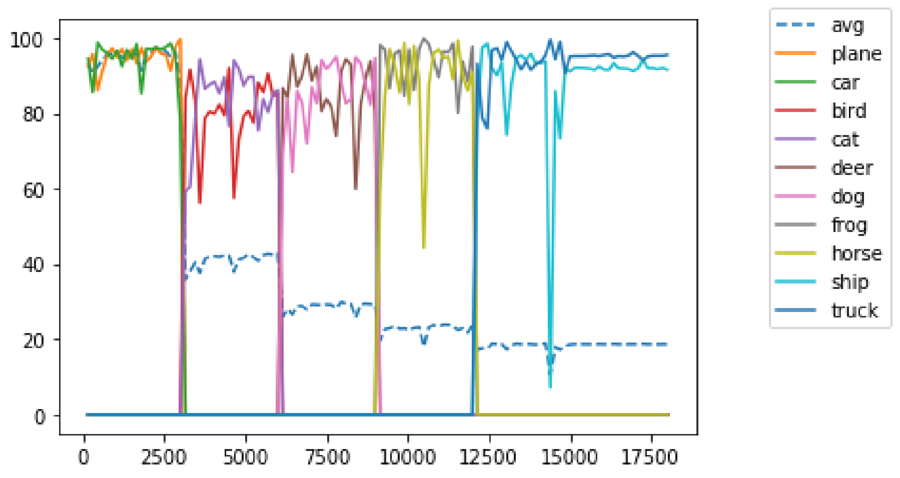


Figure 2.5. the figure shows the accuracy of the CIFAR10 test set if the network is offline or without any incremental learning mechanism. it is therefore possible to see that the network is able to learn only the new classes without being able to remember anything of the previous ones

are frozen and a new column is istantiated where hidden layers receive receive input from both column layers via lateral connection.

$$h_i^{(k)} = f\left(W_i^{(k)} h_{i-1}^{(k)} + \sum_{j<k} U_i^{(k:j)} h_{i-1}^{(j)}\right)$$

with:

- f = max(0,x)
- k task
- $W_i^{(k)}$ weight matrix of layer i of column k
- $U_i^{(k:j)}$ lateral connections from layer i - 1 of column j, to layer i of column k
- $h_i^(K)$ layer i of column k

Through this algorithm it is possible to predict K independent tasks quickly and avoiding forgetting.
The real problem with this network is that the number of parameters increases exponentially with the number of tasks it learns.

### Tree-CNN

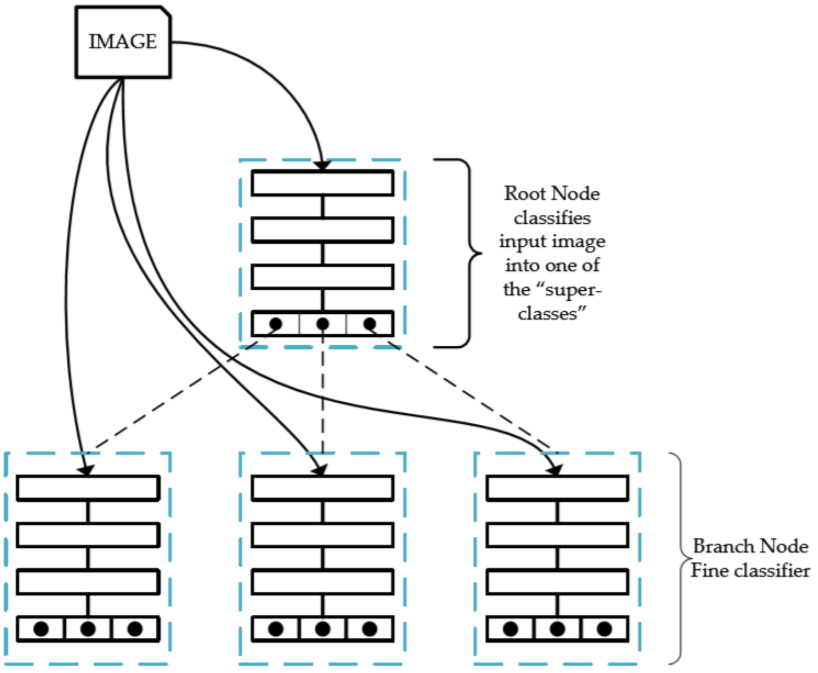


IMAGE
Root Node classifies input image into one of the "super-classes"
Branch Node Fine classifier

Figure 2.6. Generic Tree-CNN [RPR19]

The Tree-CNN is made of nodes connected as a directed a-cyclic graph.Each node acts a classifier that outputs a label for the input image. As per the label, the image is then passed on to the next node which further classifies the image, until we reach a leaf node, the last step of classification.
The network starts from a single root node and gradually grows as a direct a-cyclic graph.
When a new class arrives at a non-leaf node for each child the probability of class membership is calculated. If this probability is higher than a limit then the new class can be added to the existing child, if there are two children with the same probability then the two children and the class are combined to generate a new child node otherwise if no one exceeds the threeshold then a new child is created.

Below is presented how the likelihood is calculated:

$$O_{avg}(k,m) = \sum_{i=1}^{I} \frac{O(k,m,i)}{I}$$

$$L(k,m) = \frac{e^{O_{avg}(k,m)}}{\sum_{k=1}^{K} e^{O_{avg}(k,m)}}$$

$$\max_k (l_{m1}(k)) >= \max_k (l_{m2}(k)) >= \ldots >= \max_k (l_{mM}(k))$$

where

- $O_{avg}(k,m)O_{avg}(k,m)$ is the average output of the class m for the k child
- $L(k,m)$ is the likelihood of the class m for the child k

### 2.2.4 Regulation Strategies

The regularization techniques include the addition of a term during the training so as to attenuate the change to the network weights when a new task is faced. In EWC this is done by penalizing excessive changes to the fisher matrix during task switching. In many others, Knowledge distillation is carried out.
The distillation (introduced by Hilton []) uses an original model called Teacher that teaches how to behave when old tasks are introduced as an input to a copy named student. This is compared to the teacher for the old tasks and the ground truth for the current task.
Among those that use knowledge distillation are Lwf (Learning without forgetting), Icarl and RtF

#### Elastic weight consolidation -EWC

[Kir+16] The following model wants to imitate the synapses by reducing the plasticity of the basic network parameters for previous activities, therefore trying to keep them always close to the same values.

In a nutshell, when the aim is to learn B after having already learned A, we try to protect the area in which the parameters have been allocated so as to keep the error on the previous task low.
To find out which are the most important parameters, a probabilistic analysis is then carried out.
It can be assumed that after learning a task, all its information has been absorbed by the posterior distribution. The latter also contains all the information on the arrangement of the weights of the network.

So to be able to extrapolate it is sufficient to calculate an approximation( approximation by Mackay[]) using a Gaussian distribution with mean and covariance obtained from the fisher matrix.[Kir+16]

$$L = L_{\text{cross}}(y, t = y) + \lambda/2 \cdot \sum_k F_{\text{k}}(\theta_{\text{k}} - {\theta^*}_{\text{k}})^2$$

The dataset used is none other than the Mnist dataset. Each task was created by applying a noise (i.e. a fixed random permutation) on all the classes. In this way it is possible to predict the task on which it was applied (Task incremental learning).

There are two problems related to this model: firstly, maintaining the Fisher matrix can be expensive in terms of memory and, as a second point, it works well in the case of Task incremental learning but the same cannot be said in the case of class- incremental learning.

### Synaptic Intelligence

[ZPG17] It is a variant of EWC since the fisher matrix is too expensive

this is proposed as a structural regularizer developed online and implemented in each synapse. a synapse is actually a parameter (weights and biases) of the neural network. Its goal is to penalize the change of important parameters to remember the old memories

$$L(\theta(t)+\delta(t)) - L(\theta(t)) \approx \sum_k g_k(t)\delta_k(t)$$

$$g = \frac{\partial L}{\partial \theta}$$

$$\int_C g(\theta(t))d\theta = \int_{t_0}^{t_1} g(\theta(t)) \cdot \theta'(t)dt$$

$$\int_{t^{\mu-1}}^{t^\mu} g(\theta(t)) \cdot \theta'(t)dt = \sum_k \int_{t^{\mu-1}}^{t^\mu} g_k(\theta(t))\theta_k'(t)dt$$

$$\equiv -\sum_k \omega_k^\mu$$

the attempt to reduce the loss on the new tasks leads to an increase in the cost on the previous ones. To avoid this it is necessary to understand how each parameter is important and how much it moves with each update.

So it was decided to change the loss by adding a surrogate term which approximates the loss of the previous tasks

$$\tilde{L}_\mu = L_\mu + \underbrace{c\sum_k \Omega_k^\mu \left(\tilde{\theta}_k - \theta_k\right)^2}_{surrogateloss}$$

$$\Omega_k^\mu = \sum_{\nu<\mu} \frac{\omega_k^\nu}{(\Delta_k^\nu)^2 + \xi}$$

### Learning without Forgetting

[LH17] It is one of the first methods that uses knowledge distillation to mitigate the CF. This involves copying the trained model on the previous tasks.
When it receives a new task as input, it performs the cross-entropy loss between its prediction and the ground truth. Instead, when it receives an old task as an input, it compares its prediction with the copied model using the distillation loss[HVD15].its operation is based both on the use of: parameters shared between tasks $\theta_s$, and of task-specific parameters $\theta_o$.
When new tasks arrive, the response from the original model trained on the old tasks is recorded first.Then, randomly initialized parameters are added and, after freezing the parameters already trained, the complete network is trained to minimize the loss.
The latter is the union of lnew (loss generated by the comparison of the new tasks with the ground truth) and of lold (based on the comparison with the original network). In addition to the loss comes a regularization term made with the weight decay.

$$\mathcal{L}_{new}\left(\mathbf{y}_n, \hat{\mathbf{y}}_n\right) = -\mathbf{y}_n \cdot \log \hat{\mathbf{y}}_n$$

$$\mathcal{L}_{old}\left(\mathbf{y}_o, \hat{\mathbf{y}}_o\right) = -H\left(\mathbf{y}'_o, \hat{\mathbf{y}}'_o\right) = -\sum_{i=1}^{l} y_o'^{(i)} \log \hat{y}_o'^{(i)}$$

$$y_o'^{(i)} = \frac{\left(y_o^{(i)}\right)^{1/T}}{\sum_j \left(y_o^{(j)}\right)^{1/T}}, \quad \hat{y}_o'^{(i)} = \frac{\left(\hat{y}_o^{(i)}\right)^{1/T}}{\sum_j \left(\hat{y}_o^{(j)}\right)^{1/T}}$$

$$\theta_s^*, \theta_o^*, \theta_n^* \leftarrow \hat{\theta}_s, \hat{\theta}_s, \hat{\theta}_n argmin \left(\lambda_o \mathcal{L}_{old}\left(Y_o, \hat{Y}_o\right) + \mathcal{L}_{new}\left(Y_n, \hat{Y}_n\right) + \mathcal{R}\left(\hat{\theta}_s, \hat{\theta}_o, \hat{\theta}_n\right)\right)$$

## 2.2.5 Reharsal Strategies

Here the idea is to re-use the most representative tests of the previous classes to maintain an understanding of the past. These samples can be kept in external buffers of fixed size or can be replicated through generators. These replicas can be useful both to classify and to continually solicit the model with old features

### Ex-stream

[HCK18] It involves the use of a buffer per class. This is filled until it reaches a limited number of samples. Subsequently it is updated by removing or compressing the elements already present in the buffer. In addition to these buffers, it also contains counters (c) that indicate how many samples are present for each cluster. Furthermore, once a buffer is full and a new sample arrives $(x_t, y_t)$ the two clusters closest to the image are joined.

$$\mathbf{w}_i \leftarrow \frac{c_i\mathbf{w}_i + c_j\mathbf{w}_j}{c_i + c_j}$$
$$c_i \leftarrow c_i + c_j and \mathbf{w}_j \leftarrow \mathbf{x}_t and c_j = 1$$

When the buffers are ready, all the images in them are used to retrain the DNN.

### 2.2.6 Hybrids

Another type of method involves using regularization techniques combined with pseudo-rehearsal images. Usually these take advantage of the knowledge distillation combined with the memory replay. The model of art is iCarl obtaining good results on both Cifar10 and Cifar100. This however provides for the rescue of the most exemplary samples for each class. Subsequently, a dual-memory model like Fearnet was created but this involves saving statistische per class to generate the images. To avoid saving samples of all kinds, the first attempts to use generative models began. In this case there is the Rtf[VT18]model trained on MNIST, Neurogenesys Deep Learning[Dra+17], MerGAN[Wu+19] on L-SUN and CQFG

#### Icarl

The following model consists of a classifier, a feature extractor and a progressive number of exemplar sets per class. The latter are the most representative samples of the classes previously seen and their size depends on the memory constraints imposed.
To select the most representative images this makes use of methods such as: nearest-mean-of-examplers, herding or knowledge distillation. The classification is therefore based simply on the extraction of the features from all the exemplars to then calculate the average distance of each set from the sample[Reb+16]

$$y^* = argmin_{y=1,\dots,t} \left\| \varphi(x) - \mu_y \right\|$$

Furthermore, to respect memory constraints, when new classes are available the previous sets must be updated in order to have fewer images. To do this, each specimen is chosen by trying to minimize the distance of the samples of each set from their average.
For each examplar k we select x in such a way as to create a set of mean $\mu$ using features extractor $\varphi()$

$$argmin_{x \in X} \| \mu - \frac{1}{k}\left[\varphi(x) + \sum_{j=1}^{k-1} \varphi\left(p_j\right)\right] |$$

Subsequently, the classifier and the characteristic extractor are trained simultaneously trying to optimize both the classification loss for the new classes and the distillation loss to reproduce the same results for the old classes.

### Neurogenesys Deep Learning

this model is inspired by how the hippocampus, to keep memory of past events, exploits the mechanism of neurogenesis to generate new neurons which gradually learn to code new information.[Dra+17]
The model is able to realize when it receives images of new classes thanks to the

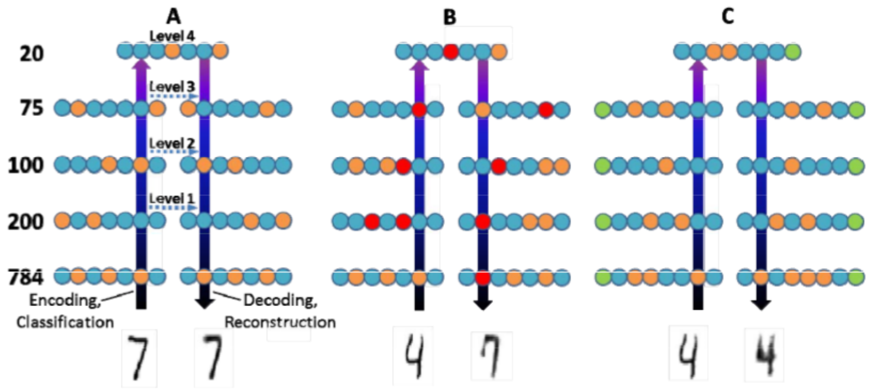


Figure 2.7. Illustration of NDL processing MNIST digits (orange/red circles indicate accuracte/inaccurate feature representations of the input; green indicate new nodes added via neurogenesis)[Dra+17].

error of reconstruction of an auto-encoder which is optimized to reconstruct the images. This peculiarity allows to have a high reconstruction error if the class has never been recognized. In this case, new weights are added by freezing / slowing down those already trained in such a way as to also be able to reconstruct the new images without losing information on old features.
Finally, to stabilize the network with all weights, it is chosen to perform fine-tuning using the images of the current task together with images called "intrinsic replay" IR and generated by the Auto-encoder itself.
To generate them, statistics (mean and Cholesky factorization of the covariance) are stored, not of images but of the intermediate product of the autoencoder in which it is smaller.

$$\mu_E = Mean(E), Ch_E = Chol(Cov(E))$$

To have a certain variability for each class at the time of generation, the product of the cholesky factorization is added to the average vector with a random sampling of a normal distribution.

$$IRImages = Decode\left(\mu_E + N(0{,}1) * Ch_E\right)$$

However, the network does not have a classification method but is only able to determine whether or not it has already seen a class through the reconstruction error. It is also forced to keep statistics for each class that can be excessively heavy (covariance matrix) and this depends on the size of the intermediate space.

### Fear-Net

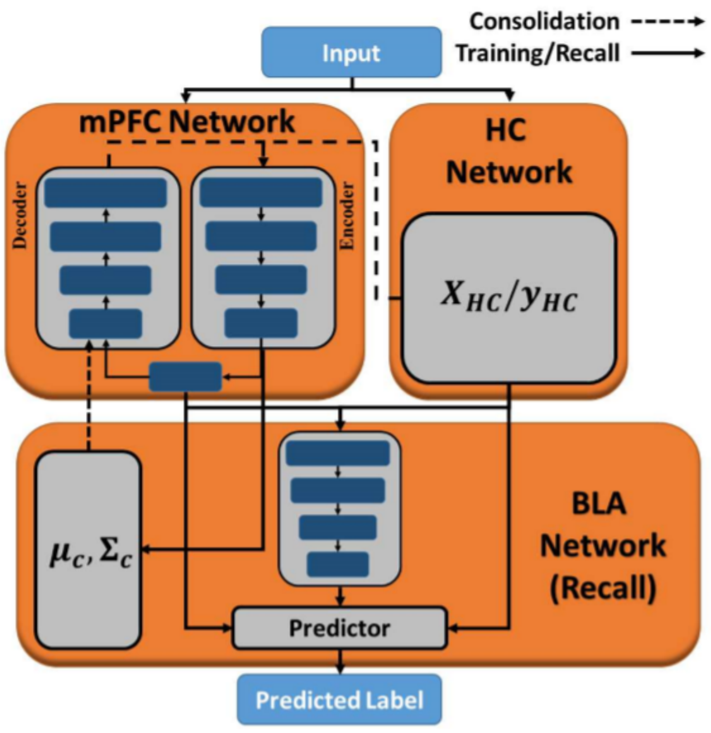


Figure 2.8. FearNet[KK17]

FearNet has three brain-inspired sub-systems: HC as recent memory system, mPFC for long-term storage and third component able to predict if it is more usefull select HC or mPfc.
Hc is an alternative of a probabilistic neural network

$$P_{\mathrm{HC}}(C = k|x) = \frac{\beta_{\mathrm{k'}}}{\sum_{\mathrm{k'}} \beta_{\mathrm{k'}}}$$

beta in this case represents the distance from the image more similar for each class k.
Mpfc was instead created with an autoencoder which not only is able to reconstruct-generate images but is also able to predict the class to which it belongs

$$\mathcal{L}_{\mathrm{mpfc}} = \mathcal{L}_{\mathrm{recon}} + \mathcal{L}_{\mathrm{pred}}$$

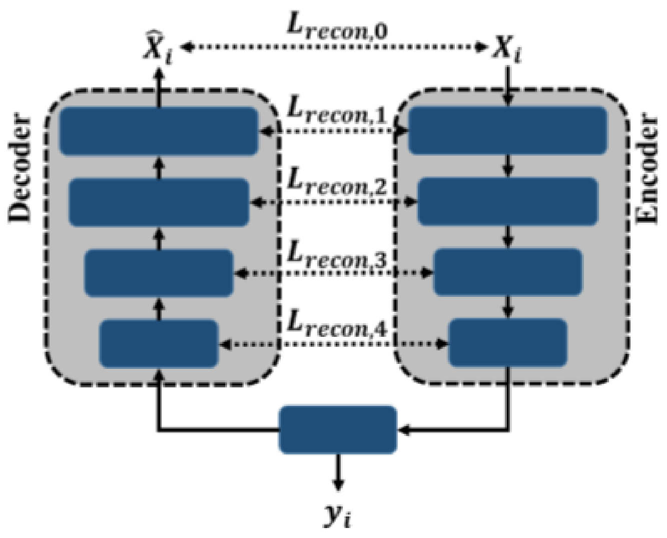


Figure 2.9. mPFC [KK17]

$$\mathcal{L}_{recon} = \sum_{j=0}^{M} \sum_{i=0}^{H_j-1} \left\| h_{encoder,(i,j)} - h_{decoder,(i,j)} \right\|_2^2$$

### Rtf

[VT18]The Rtf model is similar to the FearNet Mpfc. The difference is that instead of using an autoencoder with an external support it was preferred to use a generative model like a VAE. This allows you to generate different types of images knowing in advance which input to give to the decoder without keeping a storage.
The network is a symmetrical VAE with an additional softmax layer from the hidden layer of the encoder. The loss is calculated as follows: For the current task the loss consists of 3 terms: the reconstruction loss, the classification loss and the KL loss.

$$\mathcal{L} = \mathcal{L}_{\text{VAE}} + \mathcal{L}_{\text{pred}} = \mathcal{L}_{\text{recon}} + \mathcal{L}_{\text{KL}} + \mathcal{L}_{\text{pred}}$$

For the replicated images instead the classification term is replaced by a distillation term already seen in algorithms like LWf.
The model works very good on MNiST-digits which is relatively easy.

### Mergan

It is a conditional GAN framework that integrates a memory replay generator.
The approach is based on a discriminator and a generator. the editor creates false images starting from a sampling of a Gaussian distribution and then the

discriminator has the role to recognize the fake images from the real images. In this way the generator become more good to generate real images. A Mergan integrates also a memory replay generator.
This replay mechanism resembles the role of the hippocampus in replaying memories during memory consolidation.
A first approach is to combine the dataset of the new class to be learned with the replicas generated by the generator. The second approch is synchronize both the replay generator and the current one

## 2.3 Tools for Memory replay

This chapter presents the tools to carry out the memory replay.The latter is a technique that aims to use pseudo-rehearsals to solicit the memory of the model specially in cases like Incremental Learning.
To do this, a tool capable of generating images is required. In the past this mechanism was created through AutoEncoder and later also by generative models such as Variational AutoEncoder and gan

### 2.3.1 Auto Encoder

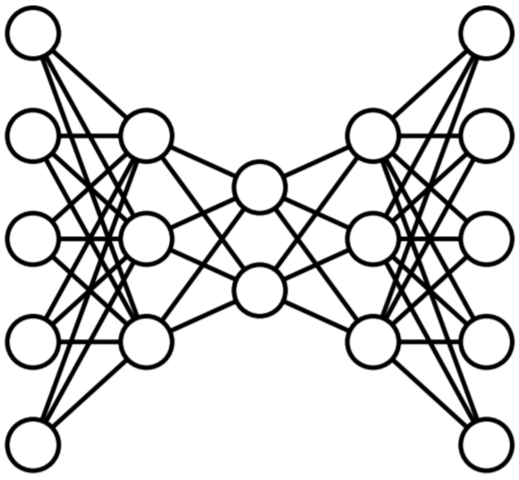


Figure 2.10. image of an Undercomplete Auto-encoder([Cha+18]

An auto encoder is an effective tool to perform dimensionality reduction. This is realized through an encoder that produces a code much smaller than the input and a decoder which, receiving the code, is able to reconstruct the image.

### Structure

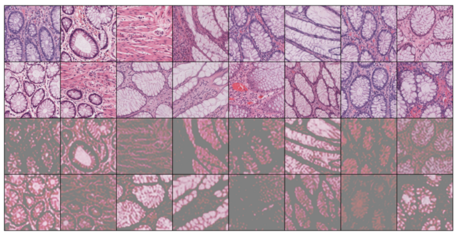
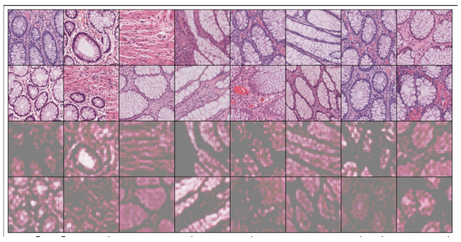
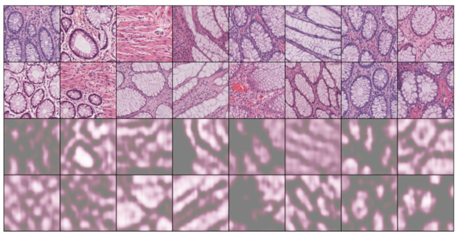
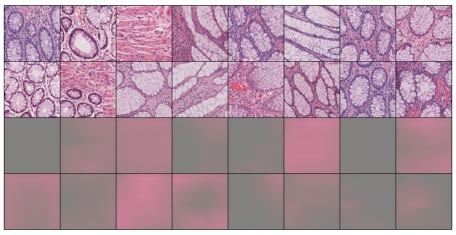
| Nr. | a | b |
|---|---|---|
| 1 |  |  |
| 2 |  |  |

Table 2.1. Example the reconstraction of an Auto-encoder on biological dataset[Pon+19].reducing the latent space of an Auto-Encoder the quality of the image can be lower.

Both are trained to minimize the rebuild error and can have different implementations. Its main use is to make a sort of feature fusion, that is to combine features. In this way it is possible to remove redundant features and thus also reduce the size of the dataset.

Different groups of Auto-encoders can be differentiated according to the structure of the model, including: Undercomplete Autoencoder and Overcomplete Autoencoder. they differ from the size of the intermediate space between encoder and decoder. In fact, the under-complete AE has a smaller input size thus forcing the model to achieve a more compact display of the data. On the contrary in the over-complete AE the size is equal to or greater than the input size allowing to learn the identity function.

### AE for memory replay

In the past, the autoencoder has been used by several models to make a sort of memory replay. It was used in FearNet[KK17] and NDL[Dra+17] as a first attempt. In these cases, statistics by class were saved on the encoder output (mean vector and covariance matrix).
Then,to allow the variability of the reconstructions, the decoder was given a result

code by the sum of the mean vector and by the eigenvalues of the covariance matrix multiplied by a random value (so as to vary the main characteristics of the class).In this case, a classifier was added to the penultimate layer of the encoder with the role of identifying the class.The problem with this technique is that reconstructing images of complex datasets (CIFAR) only from mean vector and covariance is not at all simple.
It also forces the storage of a mean vector and of the covariance matrix for each class and the latter can require a lot of memory.

### 2.3.2 Generative models

Generative modeling is a vast area of machine learning that works using P (X) distributions on a larger set of X data. X can be any type of data as well as an image of different resolution. The generative model therefore has the task of capturing the correlations between the different characteristics of the image and elaborating the importance of each of these. Then you can use this feature to reproduce new images similar to the initial database but not exactly the same.This is very useful when the dataset is small and there is a need to increase its size.
Generative models exploit the concept of latent variable z. It is called this because it is not necessary to know which parameters are more important than this to generate a specific image.
However, it must be ensured that it exists a setting for each element X that can allow you to reconstruct something very similar to X.
The goal is therefore to sample z from P (z) known a priori and optimize theta in such a way that f (z, theta) can be similar to the X dataset

$$P(X) = \int_z P(X|z)P(z)dz$$

Variational autoencoders (VAEs) are generative models. This is called as autoencoder only because it consists of an encoder and a decoder and recalls the structure of a traditional autoencoder. However his work is completely different. In fact z is extracted from a simple distribution known a priori with mean 0 and variance to 1. Then it is possible to deduce the probability of obtaining X given z as:

$$P(X|z;\theta) = \mathcal{N}\left(X|f(z;\theta), \sigma^2 * I\right)$$

Through this formula, if we consider f as a neural network, then it is possible to see how the encoder can map the input into latent variables and then be used by the decoder to generate new images. However it is necessary to penalize how different this variable can be from a normal distribution P (z).

$$P(z) \sim \mathcal{N}(0, I)$$

KL Divergence

Since P (X | z) is very small, its contribution to estimate P (X) is almost nonexistent. We therefore need a function Q (z | X) that allows us to find a distribution on z that is more likely to reproduce X. It is therefore necessary to correlate P (X) and EzsimQ P (X | z) through the Kullback-leibler divergence

$$\mathcal{D}_{KL}[Q(z|X) \| P(z|X)] = E_{z\sim Q(z|Z)}[\log Q(z|X) - \log P(z|X)]$$

Applying the rules of bayes:

$$P(z|X) = \frac{P(X|z)P(z)}{\int_z P(X|z)P(z)dz}$$

Then

$$E_{z\sim Q(z|X)}\left[\log Q(z|X) - \log \frac{P(X|z)P(z)}{P(X)}\right] =$$

$$E_{z\sim Q(z|X)}[\log Q(z|X) - \log P(X|z) - \log P(z)] + \log P(X) =$$

$$\mathcal{D}_{KL}[Q(z|X) \| P(z)] - E_{z\sim Q(z|X)}[\log P(X|z)] + \log P(x)$$

Ordering

$$\log P(X) - \mathcal{D}_{KL}[Q(z|X) \| P(z|X)] = E_{z\sim Q(z|X)}[\log P(X|z)] - \mathcal{D}_{KL}[Q(z|X) \| P(z)]$$

The right part represents what we want to optimize (P (x)) instead the left part represents what we can optimize through the neural network.

$$\mathcal{D}_{KL}\left[\mathcal{N}\left(\mu_0, \Sigma_0\right) || \mathcal{N}\left(\mu_1, \Sigma_1\right)\right] = \frac{1}{2}\left[\left(\mu_0 - \mu_1\right)^\top \Sigma_1^{-1}\left(\mu_0 - \mu_1\right) + Tr\left(\Sigma_0\Sigma_1^{-1}\right) - \log\frac{|\Sigma_0|}{|\Sigma_1|} - n\right]$$

reparameterization trick A problem that occurs, however, is to propagate the error in sampling z from Q (z | X) which is an operation without a gradient. The reason is that back-propagation only works on stochastic units. So, to solve this problem, a solution called reparameterization trick was devised.
Obtained mu and var as the encoder output, the trick is to extrapolate z as

$$z = \mu(X) + \Sigma^{1/2}(X) * \epsilon$$

considering $\epsilon \sim \mathcal{N}(0, I)$

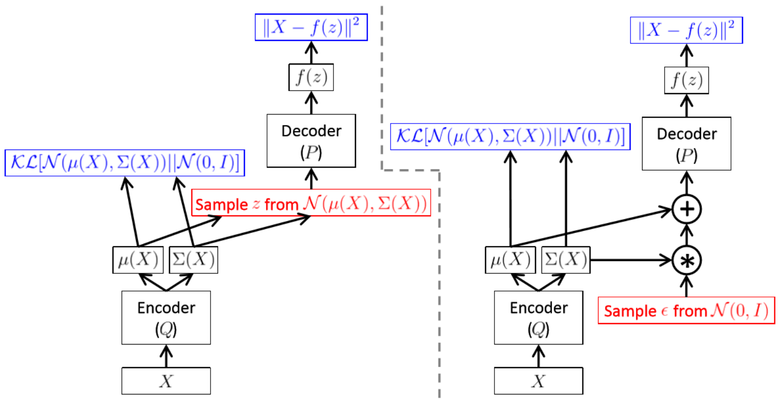

$\|X - f(z)\|^2$
$f(z)$
Decoder (P)
$\mathcal{KL}[\mathcal{N}(\mu(X), \Sigma(X))||\mathcal{N}(0, I)]$
Sample $z$ from $\mathcal{N}(\mu(X), \Sigma(X))$
$\mu(X)$
$\Sigma(X)$
Encoder (Q)
$X$
$\|X - f(z)\|^2$
$f(z)$
$\mathcal{KL}[\mathcal{N}(\mu(X), \Sigma(X))||\mathcal{N}(0, I)]$
Decoder (P)
$\mu(X)$
$\Sigma(X)$
Encoder (Q)
$X$
Sample $\epsilon$ from $\mathcal{N}(0, I)$

Figure 2.11. Comparison between VaE without reparametrization trick (left)and VAe using reparametrization trick(right)[Doe16]. They seems similar but only in the second the backpropagation works.

Tests on MNIST and CIFAR10
Once the model has trained itself to reconstruct a dataset, it can be verified that the most significant and probable images are present in the latent space in the neighborhood of the distribution N (0,1). Furthermore, speaking of Mnist, all the forms of each number can be grouped into sets of similar characteristics.
However, when the model is tested on cifar10 the reconstruction is blurred. This is probably due to an over generalization in an attempt to group the different sets.

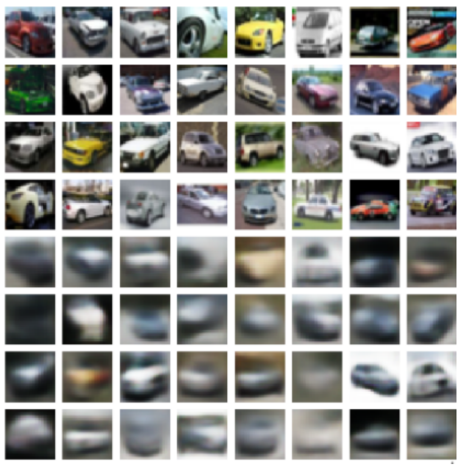
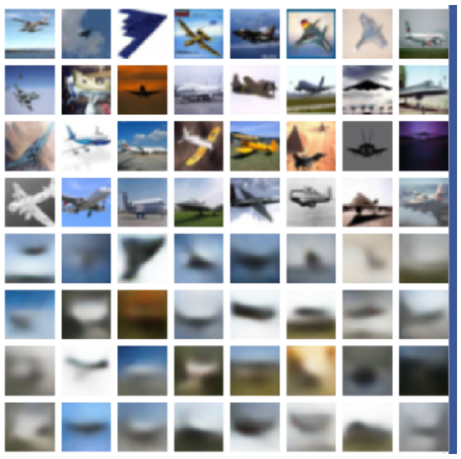
| a | b |
|---|---|
|  |  |

Table 2.2. This table presents a comparison between inputs (the first four lines) with the reconstructions (the last lines) of a variational auto-encoder. In particular, we want to rebuild cars and planes obtained by CIFAR10 with a and b respectively.

## GAN

The idea is based on the simple game of two components. The first is called discriminator and has the task of receiving images and being able to divide them into two classes (fake or real). The second is called generator and is trained to deceive the discriminator. This means that the generator must learn to reconstruct samples more and more similar to the original dataset to make the discriminator's job more and more difficult.
First we consider $\theta$ as the parameters of the model, G as the function that describes the generator, D as that to describe the discriminator.Then we can say that both components want to minimize their cost function using only their parameters.
The generator receives as input z, i.e. the result obtained from a random sampling of a distribution known a priori.
The training of this model is very simple: at each step two minibatches are obtained (from the original dataset and from the generation of the G generator) and two gradients are processed simultaneously. The first is used to optimize theta (G) and the other to improve theta (D), obviously using SGD techniques.

In fact the cost of the discriminator can be considered as

$$J^{(D)}\left(\theta^{(D)}, \theta^{(G)}\right) = -\frac{1}{2}E_{x\sim p_{data}} \log D(x) - \frac{1}{2}E_z \log(1 - D(G(z)))$$

$$\min_G \max_D DE[\log(D(x))] + \tilde{x} \sim P_g E[\log(1 - D(\tilde{x}))]$$

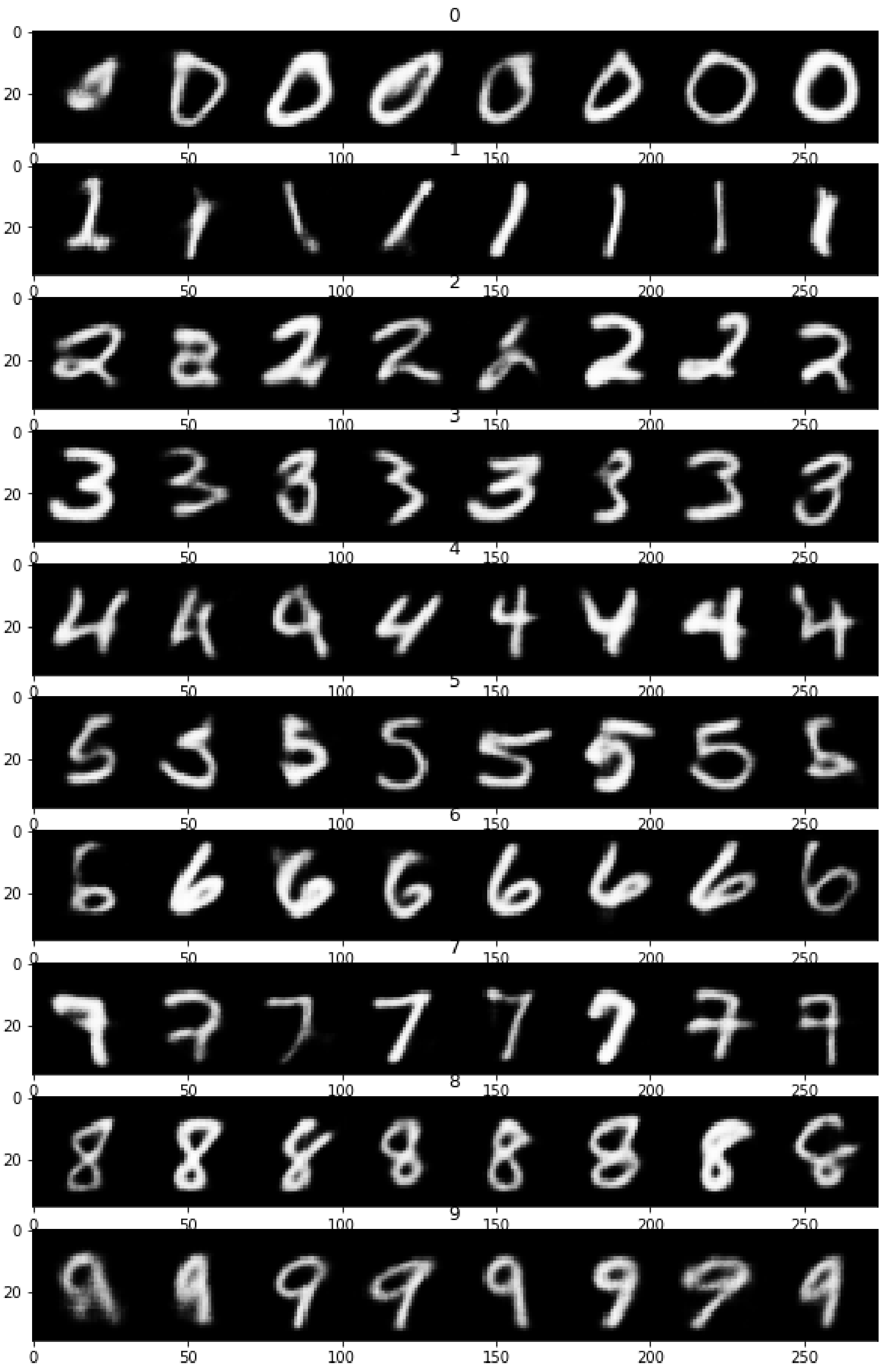


Figure 2.12. samples generated with Variational AutoEnconder On MNIST dataset.

This is cross-entropy considering as labels at 1 for the images of the original dataset

and labels equal to 0 for the images generated by the generator. The function of the cost to optimize the generator is different. In this case, it is necessary to reward the cases in which the generator managed to make the discriminator wrong. So, it would be enough to change the sign to the previous formula on the generated images.
Since however in this way the work of one would defeat that of the other, then it is much more preferable to change the target label. In this way it is possible to maximize the probability that the discriminator will be wrong. This is a simple game but unfortunately it is not fast. In fact, while one manages to make improvements the other simultaneously can get worse. In this case there is a problem of non-convergence.
An important problem related to gan is called mode collapse and it happens when multiple values of z correspond to the same image. It is a problem that limits the use of the GAN which can only produce a small number of distinct images.

Considering the generator G, the discriminator D, $P_r$ the data distribution and $P_g$ the model distribution with x '= G (z) and z a random noise generated by sampling a uniform or Guassian distribution

### PGGAN-Progressive Growing of GANs

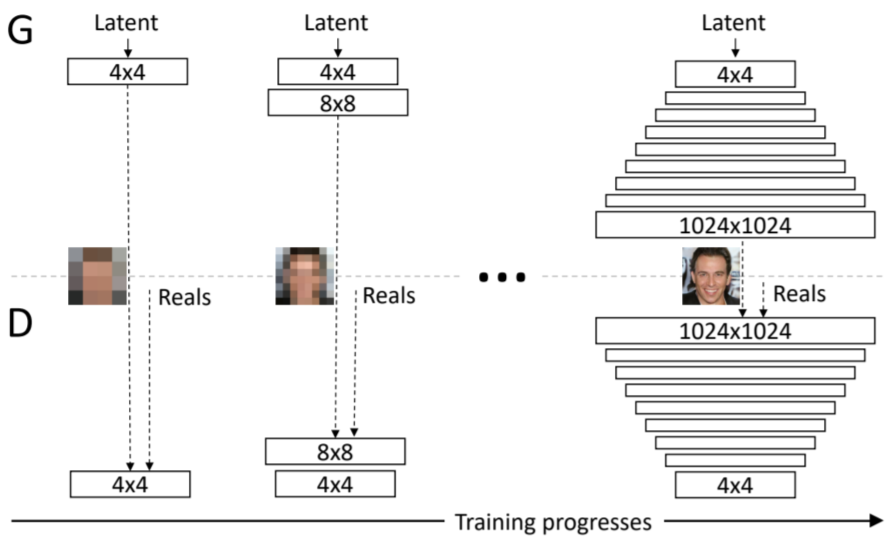


Figure 2.13. Training of a PGGAN[Kar+18] model: It is based on training a gan by progressively adding multiple levels and simultaneously increasing the resolution of the image

The idea is to take a Generative adversarial network trained with the Wasserstein loss[Gul+17] and gradually increase the resolution and layers of the network. This network was born because it was noticed that a normal GAN manages to generate a subset of the variations of the training set.
Since this is due to the fact that the statistics of the characteristics are connected not to the single image but to the minibatch, the proposal of the PPGAN was to calculate the standard deviation for each characteristic in each spatial location. Finally the average on all features is calculated to obtain a single value. The training is done using the improved wasserstein loss.

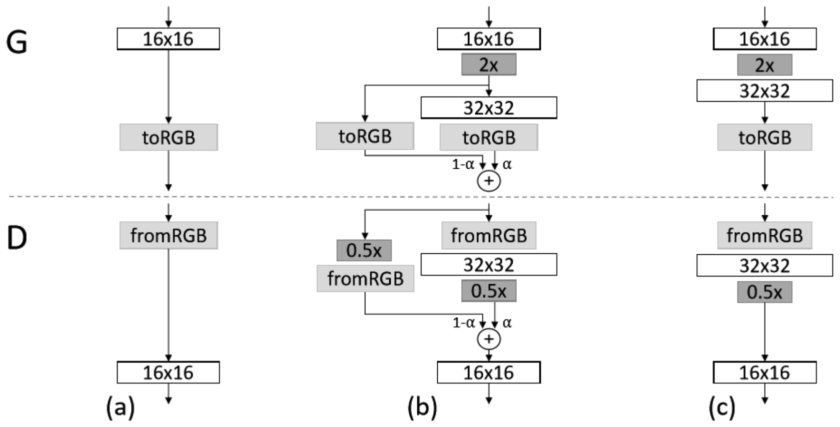


Figure 2.14. Training of a PGGAN[Kar+18] model: In the phase of change between a lower resolution and a higher one, to make the transition gradual, the network is transformed into a residual network by attenuating the effect of the residual by an alpha coefficient (from 1 to 0). The figure shows the 3 different steps during this change. 2x and 0.5x are made with nearest neighbor filtering and average pooling, respectively.TORGB and FROMRGB are two layers that carry out a 1x1 convolution and are necessary to project the feature vector on the rgb channels and vice versa.

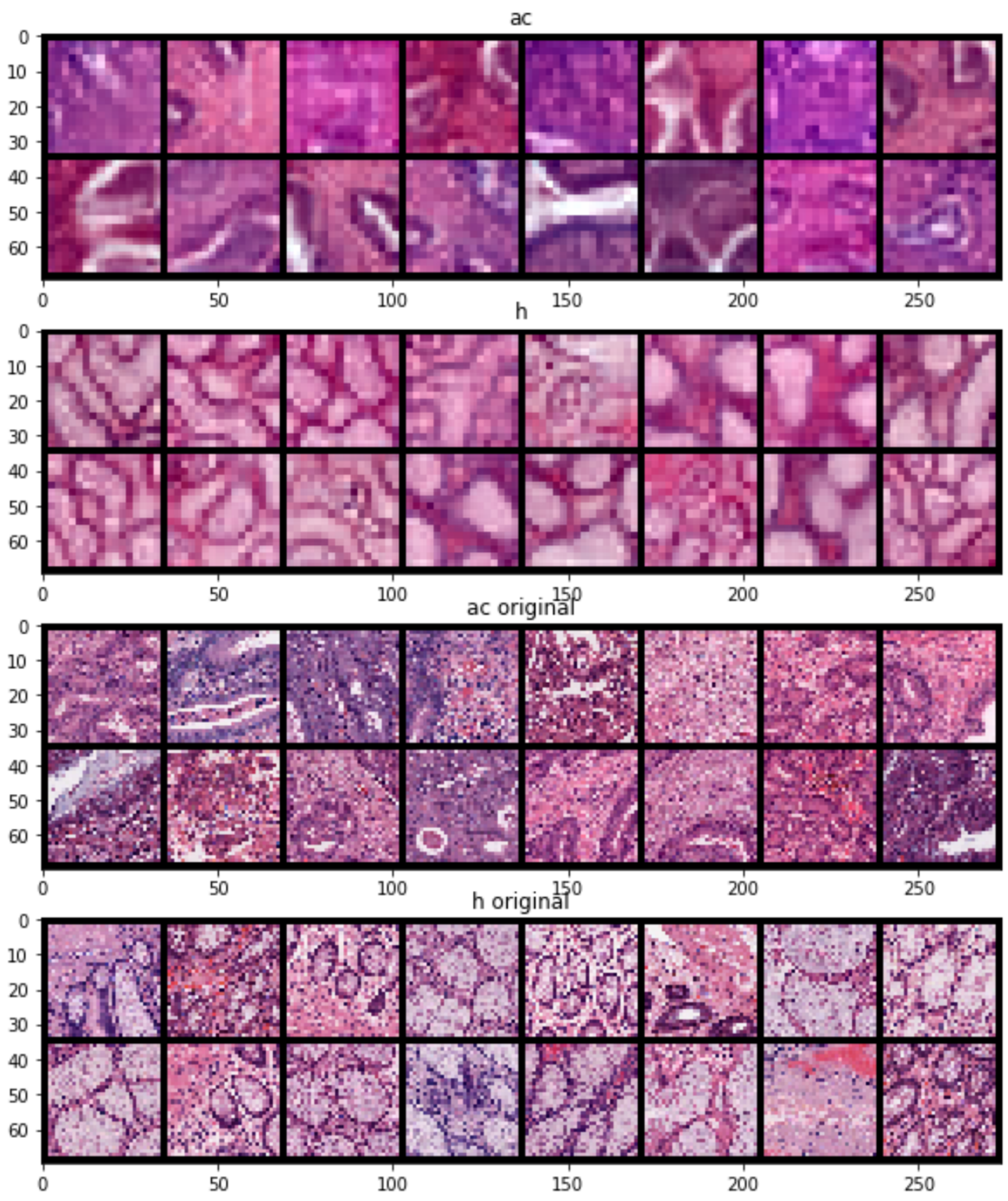


Figure 2.15. comparison with Biodataset [Pon+19]

# Chapter 3

# Proposed model

A first experiment aims to analyze the potential of a generative model such as a VAE in a system such as RTF using convolutional networks.
Always using a VAE as generator, a progressive network of generators has been created using only the reconstruction error to classify on MNIST.
Finally, given that the previous models use a single generator for all classes, as a last experiment a network of generators (PGGAN) has been created, built progressively, maintaining a class incremental learning approach from the data point of view.

## 3.1 NVAE

During the study of the generative models, the Variational Auto-Encoder model was examined. This, in addition to being able to have a light structure, can also function as an anomaly detector. This is possible because after being trained to rebuild a given class, when he receives an image he generates a new one of that class more similar to the input. If therefore this is very different from normal, the reconstruction error is high. Given this, it was decided to create a network of VAE generators of a size proportional to the number of classes encountered. The classification was therefore based on selecting the generator with the lowest reconstruction error.

$$y_{pred} = min_y(E_{recon})$$

the accuracy obtained on MNIST was on average 94.5%. On Cifar10, however, performance has dropped drastically due to the fact that the Vae generates images that are too blurry.For this reason this first model was discarded for subsequent comparisons.For this reason they have not been used to make comparisons.

| type | layer | input | output | kernel size | padding | stride |
|---|---|---|---|---|---|---|
| encoder | | | | | | |
| | 1 | (1,32,32) | (16,16,16) | (3,3) | (1,1) | 2 |
| | 2 | (16,16,16) | (32,8,8) | (3,3) | (1,1) | 2 |
| fully connected | | (32*8*8) | 100 | | | |
| fully connected | | 100 | (32*8*8) | | | |
| decoder | | | | | | |
| | 1 | (32,8,8) | (16,16,16) | (3,3) | (1,1) | 2 |
| | 2 | (16,16,16) | (1,32,32) | (3,3) | (1,1) | 2 |

Table 3.1. structure of a single vae

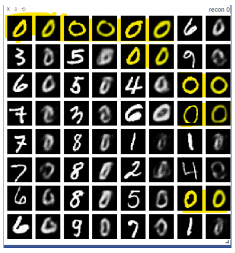
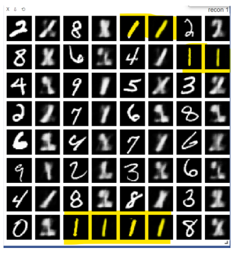
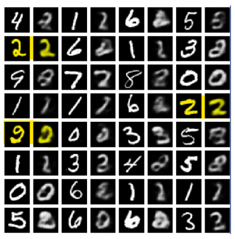
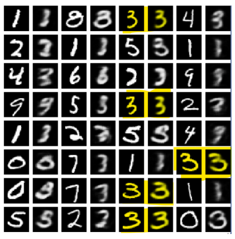
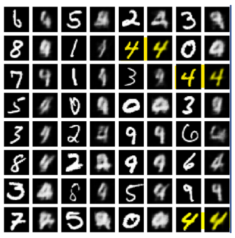
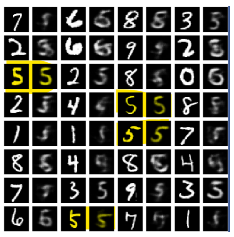
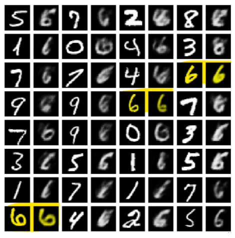
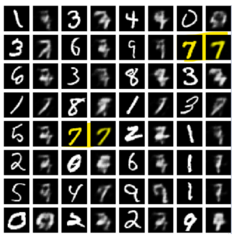
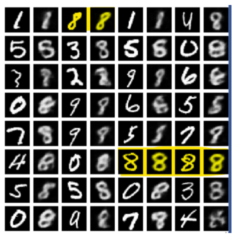
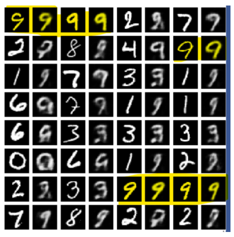
| Nr. | a | b | c | d |
|---|---|---|---|---|
| 1 |  |  |  |  |
| 2 |  |  |  |  |
| 3 |  |  | | |

Table 3.2. Generation of all numbers using NVAE. For each generator, the input and output are illustrated by highlighting in yellow the cases in which they are the same

## 3.2 network of PGGAN

The aim of this research is focused on learning different kind of objects in a sequential way mitigating at the same time the catastrophic forgetting effect with pseudo-Images.In essence we want to make a model capable of dreaming and use these dreams to consolidate memory.
In the past many approaches has been discovered and we have decided to take the more challenging way.
Indeed it is first of all necessary to create a stable and plastic neural network. This means that it should be enough stable to consolidate previous memory and at the same time It should enough malleable to learn new object without breaking it.

**Algorithm 1:** Incremental Train

```
Result: Write here the result
S=data;
S'= generated images;
y'= target scores;
y=label;
G=generator;
D= discriminator;
for k in tasks do
    for e in epochs do
        S'= $\emptyset$;
        y'= $\emptyset$;
        for j in (0,k) do
            OutPut'$_j$ = $G_j(noise)$;
            y'$_j$ = $D_{k-1}(S'_j)$;
            if y'$_j$ == j then
                S' = S' + OutPut'$_j$;
                y'$_{target}$ = $y'_j$; end
            end
            $y_{pred} = D_k(S_k)$;
            y'$_{pred} = D_k(S')$;
            $L_{distill}$ =-$\sum_{i=1}^{l} y'^{(i)}_{target} \log y'^{(i)}_{pred}$;
            $L_{cross} = CrossEntropyLoss(y, y_{pred})$;
        end
    end
    for class in k do
        train G$_{class}$;
    end
```

we want to do all this without using any type of external memory and instead of keeping samples I wanted to generate them every time.
The incremental approach used is the class incremental learning. This consists in using only the images of the current classes grouped in tasks and the model must be able to predict the image label without knowing which task it belongs to.
Finally, we decided to use knowledge distillation. This technique consists in transmitting knowledge from a larger model, the teacher model, to a smaller one, the student model. To do this, in our case it is necessary that the teacher initially elaborates the pseudo-image to obtain a result subsequently used by the student as a target. At the same time the original images arrive and in that case the student model uses ground truth as a target.

A point that we would like to analyze further are the generated pseudo-images used by existing incremental learning models. Generated by Auto-encoder plus statistics by class or Variational Auto-Encoder these images were often very similar to the original images but unfortunately still visibly different. This evidence becomes larger as the complexity of the dataset increases.At the same time, overuse of these can be risky, causing the model to lose contact with reality.
However these images, being extremely rich in features, have great potential in soliciting the memory of the model.

Like a normal autoencoder the purpose of a variational auto-encoder is to reconstruct the image received in input. However, the essential change was to make the latent space look like a distribution known a priori. In this way, by randomly sampling the same distribution it is possible to obtain different types of images.
One of its known characteristics is also that of being a good anomaly detector. In fact, if it receives as input an image very different from the one on which you trained the reconstruction error is very high.

however, the fact that the variational auto-encoder generates blurry images on complicated datasets becomes a limitation.

For this reason we have chosen to use in my research a more recent generative model capable of optimally generating even more complicated images such as those present in Cifar10. PGGAN. it was published in 2018 obtaining the inception score record (8.8) on a complicated dataset such as Cifar10.
Like a simple Gan his training is based on a simple game between a discriminator and an image generator. while the discriminator has the task of recognizing the real images from the fakes, the generator improves its image generation by trying to deceive the discriminator. As an additional thing, however, its incremental increase in resolution is particularly interesting. in fact during the passage from lower to higher resolution the network becomes very similar to a residual network.

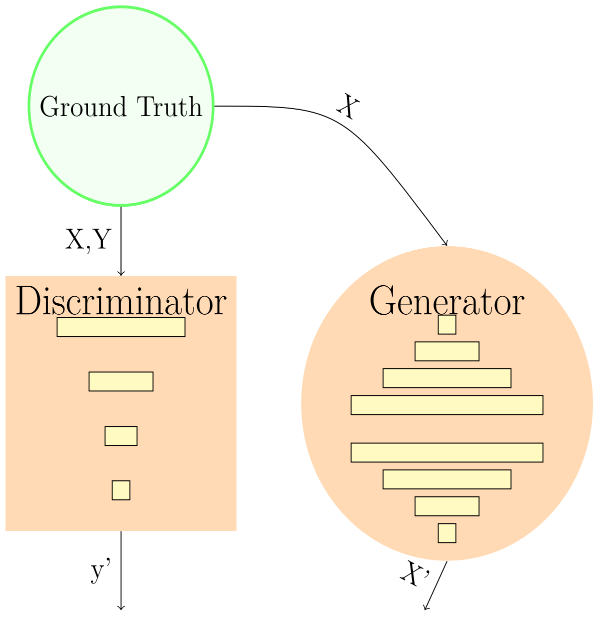


Figure 3.1. Proposed model trained on the first task.

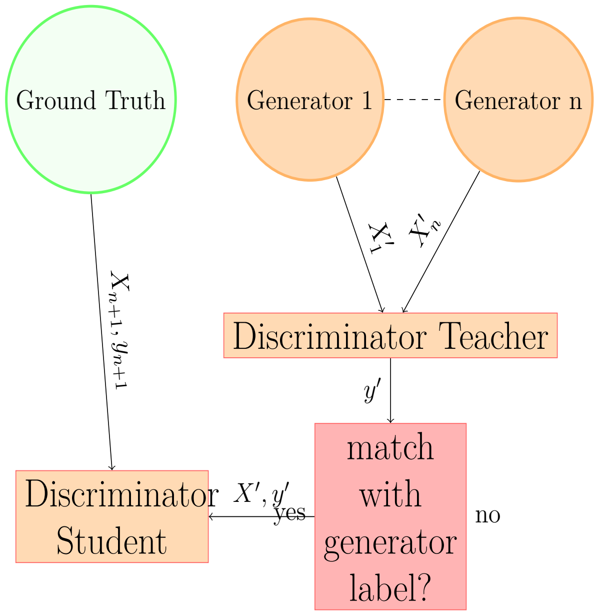


Figure 3.2. Proposed model trained on the n+1 task.

So as to be able to study the true potential of the memory replay associated with knowledge distillation within an incremental learning environment, we decided to use a generative model for each class encountered although it may be an excessive

cost in terms of memory requirements . In this way I have the possibility to avoid overlapping characteristics belonging to different classes in the same image.
Moreover, this architectural structure allows me to carry out a quality control on the images generated before they are used for training. The images of a generator trained on a class A are therefore only used if the teacher's judgment on that image is A. However, to prevent it from being too blocking when the teacher has already been the victim of catastrophic forgetting, this check is carried out a limited number of times.
In this research, given that in an incremental approach it is an important variable also the time, we have not trained the PGGAN models with all the iterations suggested by the manufacturer but only with those necessary to replicate images similar to the originals (2 days per class )

Finally the protagonist of my study is the discriminator. this must face the classes incrementally, also receiving pseudo-images from the network of generative models. this consists of an autoencoder and a CNN to be able to extract the characteristics of the image in a hierarchical way. This is essential to reduce the difference between the original images and the fake images.
It is then trained to reduce the reconstruction error, the prediction error on the current task and the distillation loss on the pseudo-images

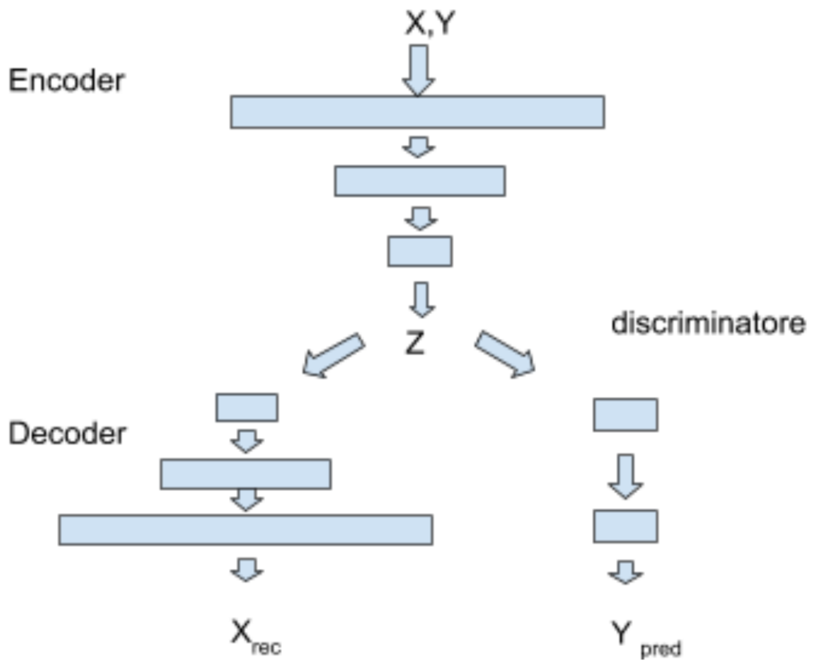


Figure 3.3. scheme of the auto-encoder + discriminator

| type | layer | input | output | kernel size | padding | stride |
|---|---|---|---|---|---|---|
| encoder-cov | | | | | | |
| | conv | (3,32,32) | (10,32,32) | (3,3) | (1,1) | 1 |
| | ELU | | | | | |
| | batchNorm | | | | | |
| | MaxPool | (10,32,32) | (10,16,16) | | | |
| | ELU | | | | | |
| | batchNorm | | | | | |
| | dropout | | | | | |
| | conv | (10,16,16) | (20,16,16) | (3,3) | (1,1) | 1 |
| | ELU | | | | | |
| | batchNorm | | | | | |
| | MaxPool | (20,16,16) | (20,8,8) | | | |
| | ELU | | | | | |
| | batchNorm | | | | | |
| | dropout | | | | | |
| encoder-mean | | | | | | |
| | $\top conv$ | (3,32,32) | (10,32,32) | (3,3) | (1,1) | 1 |
| | ELU | | | | | |
| | batchNorm | | | | | |
| | MaxPool | (10,32,32) | (10,16,16) | | | |
| | ELU | | | | | |
| | batchNorm | | | | | |
| | dropout | | | | | |
| | conv | (10,16,16) | (20,16,16) | (3,3) | (1,1) | 1 |
| | ELU | | | | | |
| | batchNorm | | | | | |
| | MaxPool | (20,16,16) | (20,8,8) | | | |
| | ELU | | | | | |
| | batchNorm | | | | | |
| | dropout | | | | | |
| decoder | | | | | | |
| | conv | (20,8,8) | (10,16,16) | (3,3) | (1,1) | 1 |
| | ELU | | | | | |
| | conv | (10,16,16) | (3,32,32) | (3,3) | (1,1) | 1 |
| | ELU | | | | | |
| | Sigmoid | | | | | |

Table 3.3. Convolutional AutoEncoder Structure

| type | layer | input | output | kernel size | padding | stride |
|---|---|---|---|---|---|---|
| encoder | | | | | | |
| | resnet | (3,32,32) | 512 | | | |
| | fully connected | 512 | (512) | | | |
| decoder | | | | | | |
| | conv | (8,8,8) | (10,16,16) | (3,3) | (1,1) | 1 |
| | ELU | | | | | |
| | conv | (10,16,16) | (3,32,32) | (3,3) | (1,1) | 1 |
| | ELU | | | | | |
| | Sigmoid | | | | | |

Table 3.4. AutoEncoder Structure using resnet pretrained on imagenet

| type | layer | input | output | kernel size | padding | stride |
|---|---|---|---|---|---|---|
| discriminator | | | | | | |
| | conv | (20,8,8) | (20,8,8) | (3,3) | (1,1) | 1 |
| | ELU | | | | | |
| | batchNorm | | | | | |
| | MaxPool | (20,8,8) | (20,4,4) | | | |
| | ELU | | | | | |
| | batchNorm | | | | | |
| | dropout | | | | | |
| | conv | (20,4,4) | (20,4,4) | (3,3) | (1,1) | 1 |
| | ELU | | | | | |
| | batchNorm | | | | | |
| | MaxPool | (20,4,4) | (20,2,2) | | | |
| | ELU | | | | | |
| | batchNorm | | | | | |
| | dropout | | | | | |
| | fully connected | (20,2,2) | 10 | | | |

Table 3.5. Structure of convolutional discriminator

| Layer | input | output |
|---|---|---|
| fully connected | (1,32,32) | (100) |
| fully connected | (100) | (100) |
| fully connected | (100) | (100) |
| fully connected | (100) | (10) |

Table 3.6. Structure of MLP discriminator for MNIST dataset

# Chapter 4

# Results

In this section the various results obtained on datasets such as MNIST, CIFAR10 and biological dataset are presented. The context addressed is class-incremental learning. Everything was developed using the famous Pytorch framework. The comparisons made with other state-of-art were made using MLP and pre-trained resnet as main components. Further experiments were developed using a convolutional model consisting of an AutoEncoder.
The comparisons were made considering the model without mechanisms suitable for incremental learning as the lower limit and the model trained using the entire dataset as the upper limit.
Icarl was chosen as comparison models, because it is the state of the art between the regularization and rehearsal strategies, and RtF, one of the first models of this type to use generative models. Finally, comparisons were made using the same approach but varying the number of tasks, the presence of a quality control and also the duration on which each task is trained.

Table 4.1 shows how effective the generated images are to replace the original

| | MNIST | CIFAR10 | Biological Dataset |
|---|---|---|---|
| Only pseudo-images | 96,20 | 35,62 | 15,20 |

Table 4.1. Results on different datasets using only generated images.

dataset during the training phase. Visibly there are 3 very different difficulty levels.

### statistics-based models

In models such as NDL and FearNet statistics are saved for each class and then reused to generate pseudo-reharsal images through the use of an autoencoder. This, if performed as NDL, involves saving an average vector and a covariance matrix, which can be a heavy element at the computational level. In an attempt to replicate this phenomenon, it was possible to replicate simple datasets on MNIST while on other datasets, where the correlation of the characteristics is lower, the replicas were difficult to obtain.Since therefore, probably some information was missing from the literature, it was not possible to replicate the results of models that use statistics.

## 4.1 MNIST

This dataset represents the basis from which all models start. In fact, it is relatively easy to recognize each number. They are also low resolution images (1x28x28).This allows simple generative models such as vae to generate images almost identical to the original dataset.For this reason the results on this dataset are excellent with any model.
The peculiarity of the anomaly detection of the vae works well on this dataset.When the input is the number it is trained on, the reconstruction remains faithful. In the case of a mismatch, on the other hand, there is a high reconstruction error.
peculiarity of the anomaly detection of the vae works well on this dataset. As shown in the table 3.2, when the input is the number it is trained on, the reconstruction remains faithful. In the case of a mismatch, on the other hand, there is a high reconstruction error.

The diagram 4.1 presents a first positive approach to how to mitigate catastrophic forgetting. In fact from one task to another the accuracy remains almost constant. This is precisely because the images generated replace the initial dataset almost perfectly, without letting the model understand that there have been actual differences.However, it can be seen that there is an error propagation and it grows as new tasks are faced up until the end.

| | i-carl | RtF | NG |
|---|---|---|---|
| lower bound | 19,9 | | |
| 1 epoch | 95,30 | 91,51 | 92,75 |
| 2 epoch | 96,15 | 92,87 | 91,16 |
| 5 epoch | 96,45 | 91,15 | 92,54 |
| upper bound | 97,20 | | |

Table 4.2. Results on Mnist using a Multilayer Perceptron model as common discriminator on 5 task.

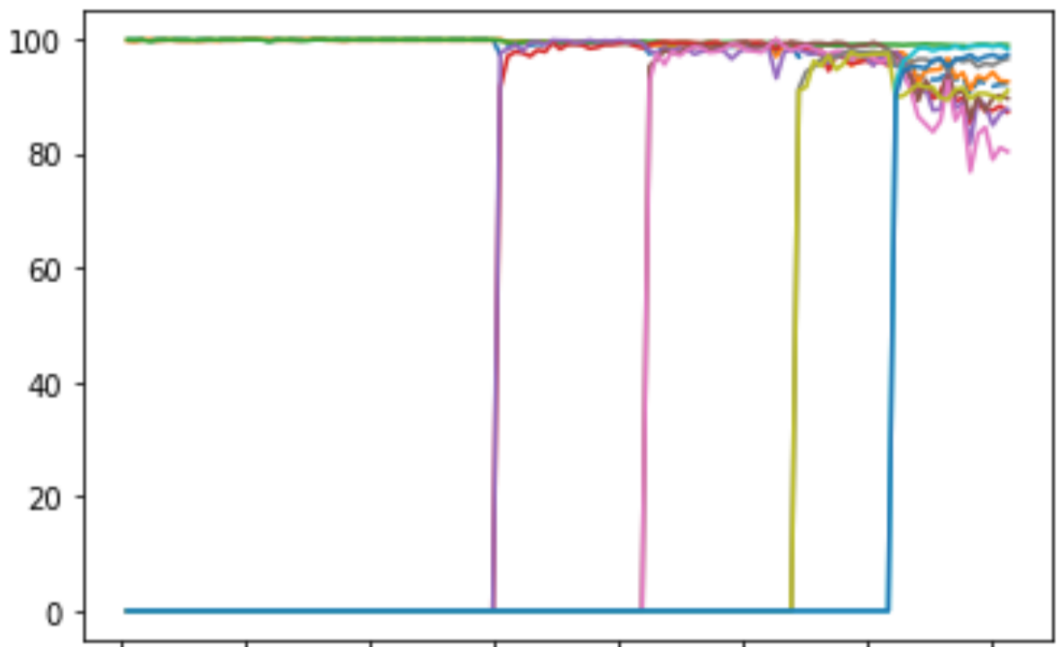


Figure 4.1. This diagram shows the progress of the accuracy on MNIST. In this case the dataset was divided into 5 different tasks which are given to the model individually.This result has been obtained using a MLP model.

| name | i-carl | rtf | NG |
|---|---|---|---|
| lower bound | 49,9 | | |
| | | | |
| 1 epoch | 92,20 | 90,21 | 93,85 |
| 2 epoch | 94,30 | 94,66 | 93,11 |
| 5 epoch | 95,48 | 95,05 | 94,78 |
| | | | |
| upper bound | 97,20 | | |

Table 4.3. results on Mnist using a Multilayer Perceptron model as common discriminator on 2 task.

From the tables 4.2 and 4.3 it is easy to understand that in an incremental learning environment, models that exploit image generators are unlikely to obtain better results than those that keep the best exemplars in memory as Icarl. Furthermore, it can be seen if the same problem is addressed with few but large tasks, the accuracy improves.

| name | no quality | with quality control |
|---|---|---|
| lower bound | 19,9 | |
| 1 epoch | 91,85 | 92,75 |
| 2 epoch | 90,15 | 91,16 |
| 5 epoch | 89,43 | 92,54 |
| upper bound | 97,20 | |

Table 4.4. results on MNIST dataset using a MLP model as discriminator and only 5 tasks. .

Add a quality control can improve the situation, as shown in the table 4.4Quality control filters the images used for the training phase. The improvement that this allows compared to not using it represents the proportion of significant images compared to all those generated. When the generated images are all incomprehensible, the quality control is ineffective and blocking.
In this case a slight improvement can be noticed even if, by increasing the number of epochs per class, it becomes less and less effective.

Subsequently, some tests(4.5,4.6,4.2) were performed using a convolutional model

| name | 2 task | 5 task |
|---|---|---|
| lower bound | 49,2 | 19,9 |
| 1 epoch | 97,22 | 95,15 |
| 2 epoch | 96,89 | 92,18 |
| 5 epoch | 92,54 | 88,42 |
| upper bound | 99,48 | |

Table 4.5. results on Mnist using a convolutional model as discriminator.

| name | no quality | with quality control |
|---|---|---|
| lower bound | 49,2 | |
| 1 epoch | 97,22 | 97,20 |
| 2 epoch | 96,89 | 96,55 |
| 5 epoch | 88,42 | 92,78 |
| upper bound | 99,48 | |

Table 4.6. results on Mnist using a convolutional model as discriminator and only 2 tasks. .

| name | no quality | with quality control |
|---|---|---|
| lower bound | 19,9 | |
| 1 epoch | 95,15 | 95,75 |
| 2 epoch | 92,18 | 96,15 |
| 5 epoch | 88,42 | 85,51 |
| upper bound | 99,48 | |

Table 4.7. results on MNIST dataset using a convoutional model as discriminator and only 5 tasks. .

consisting mainly of an Auto-encoder.A convolutional model can lead to better results, above all thanks to an extractor of features such as the Auto-Encoder. However, it suffers from the same problems and indeed accentuates them.
As can be seen, in fact, even with such a model, there continues to be a greater error when the number of tasks per dataset increases. Furthermore, there is a visible overfitting on the generated images, losing contact with reality, increasing the number of epochs.
This, of course, can also be caused by the structure of the network, such as the number of layers or an excessive presence of pooling layers.
For this reason, in the next datasets, much more complicated than MNIST, "resnet18" has been chosen to make correct comparisons with the other models.

## 4.2 Cifar10

This dataset is, although small, much more complicated. With a resolution of 32x32 it has first 3 input channels to represent all possible colors. Each class does not have a unique shape and color but is very variable.
Furthermore, the subject is almost never centered (something that often happens in works of generative models). So we can basically say that the correlation of features of different images belonging to the same class is very low.
For this reason, the generated images, as illustrated in 4.2, are different from the original dataset. However, they maintain some important peculiarities that characterize the class. To measure how different these images were from the original dataset, a special training was performed, using only the images generated during the training. In this way, using a pre-trained resnet18, it was possible to reach 35% accuracy on the test set.

| | i-carl | rtf | NG |
|---|---|---|---|
| lower bound | 44,23 | | |
| | | | |
| 1 epoch | 55,30 | 41,32 | 52,31 |
| 2 epoch | 58,14 | 40,45 | 55,15 |
| 5 epoch | 72,36 | 42,53 | 49,45 |
| | | | |
| upper bound | 82,20 | | |

Table 4.8. results on Cifar10 using pretrained resnet as common discriminator on 2 task.

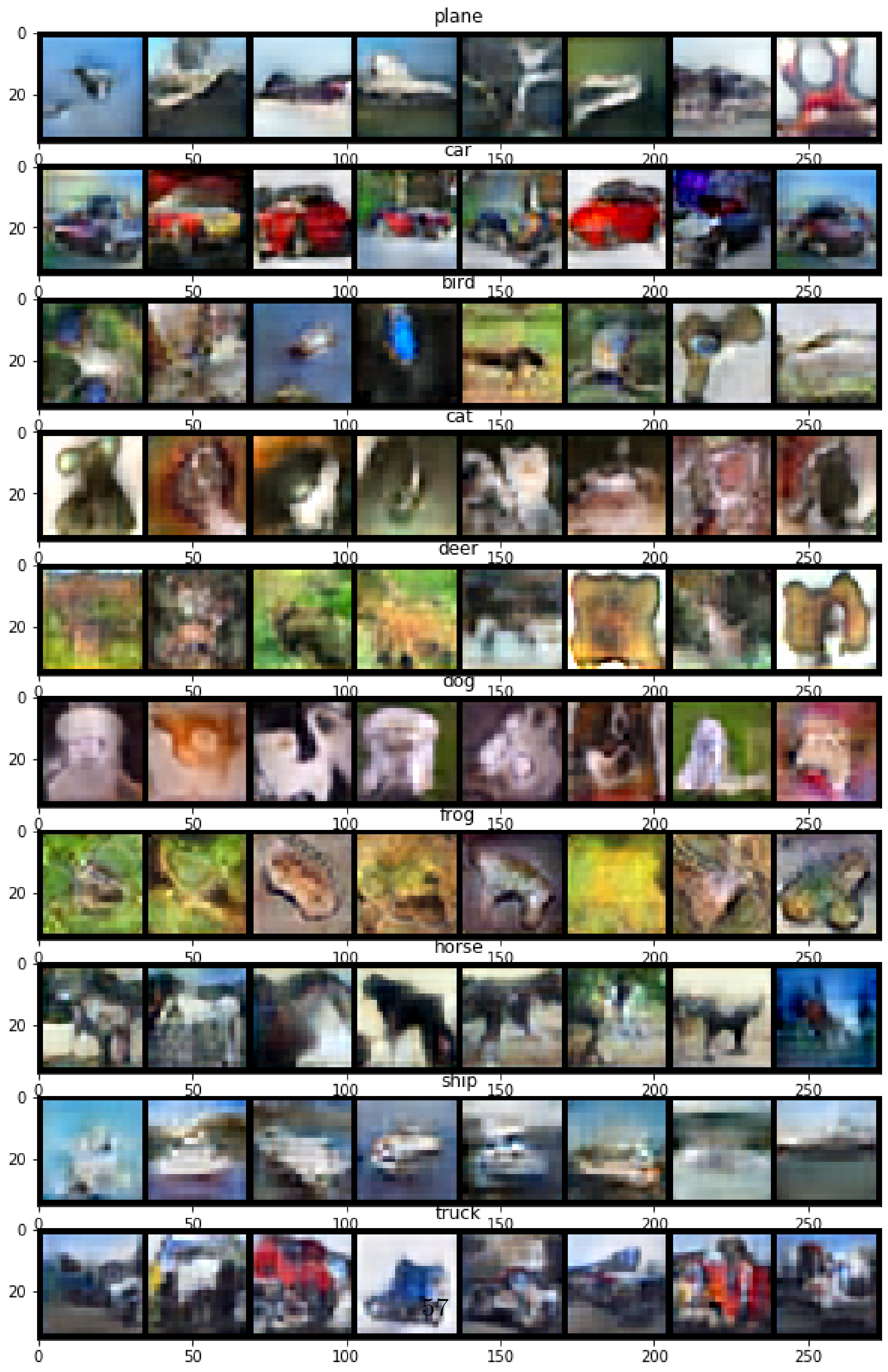


Figure 4.2. generated images using a Pgan model per class

For this reason the results shown in table 4.8 are much lower than a standard workout where classes are given simultaneously. In this case, however, we can see that compared to RtF (which generates blurred images) the performance is better. In particular, it is possible to see that, while Icarl learns to better extract the features in the exemplars that it has in memory, our model suffers from overfitting, adapting too much to a new date completely different from the original one.
As the diagram 4.3 shows, the first task is absorbed fairly well. Subsequently,

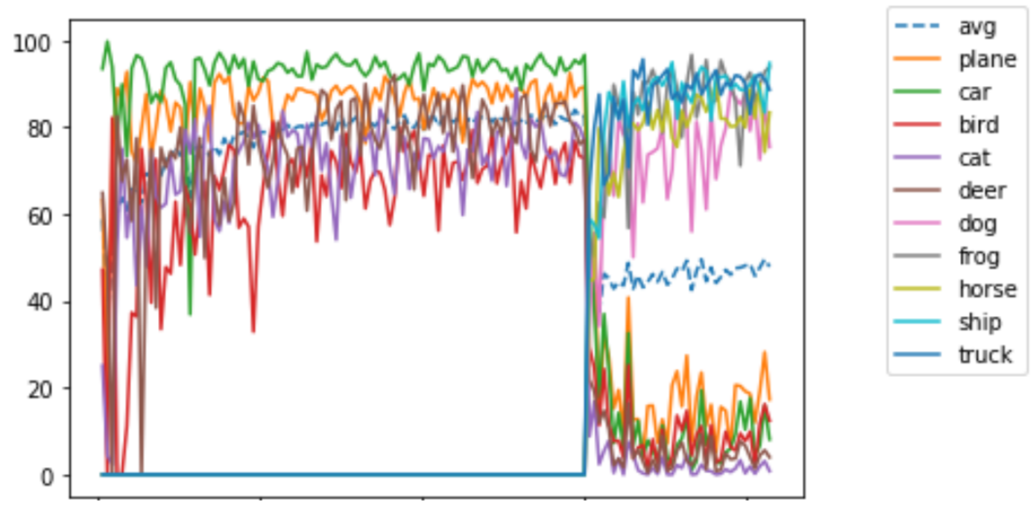


Figure 4.3. Accuracy On CIFAR10 using resnet pre-trained as encoder on 2 tasks.

however, when facing the new task, most of the characteristics are forgotten and the classification deteriorates very quickly. it should be noted, however, that the model learns the new task with an accuracy> 80%. This may mean that the structure is still well designed and the fundamental problem lies in the generated images. From this we can understand that we should have a model capable of putting dirty and original images on the same level.

When the number of tasks increases, the results worsen even more. This affects all models including I-carl(table 4.9). One of the possible reasons is that when faced with smaller tasks, the discriminator manages to obtain excellent results during training. As a consequence, the loss remains low and the model does not work enough to be able to separate the classes substantially. For example, if it is enough to distinguish two classes for a color, then it is not necessary to take into consideration all the other details that determine them.
In fact, shown by the graph 4.4 at the end of the fourth task, the only two classes that the model can remember only planes, usually the sky as a background, and the machines. The fifth task unfortunately contains two classes very similar to the ones mentioned above. This makes losing the little memory left. In this case however

| | i-carl | rtf | NG |
|---|---|---|---|
| lower bound | 18,5 | | |
| | | | |
| 1 epoch | 51,37 | 21,32 | 21,31 |
| 2 epoch | 56,78 | 20,45 | 22,15 |
| 5 epoch | 60,86 | 22,53 | 21,45 |
| | | | |
| upper bound | 82,20 | | |

Table 4.9. results on Cifar10 using pretrained resnet as common discriminator on 5 task.

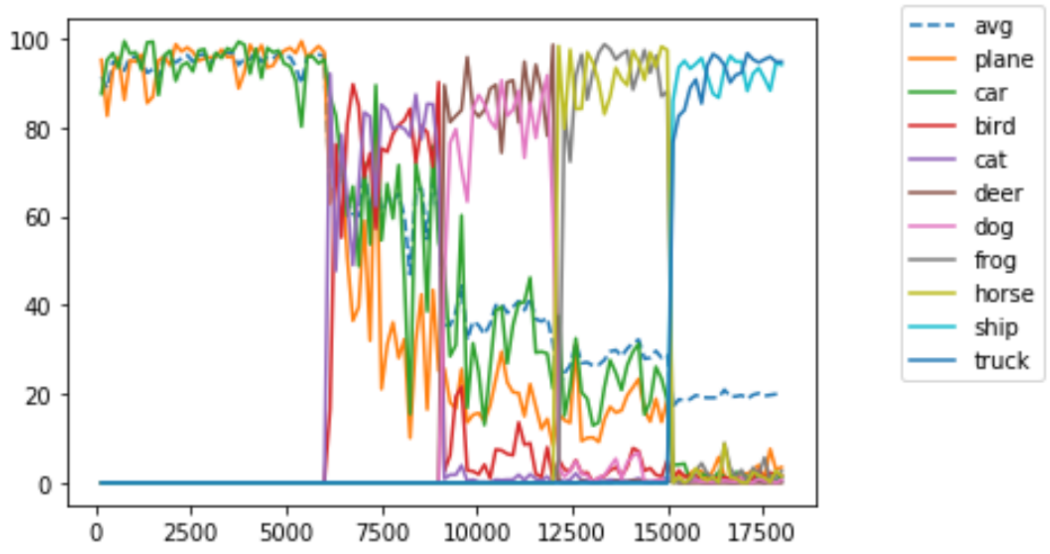


Figure 4.4. Accuracy On CIFAR10 using resnet pre-trained as encoder on 5 tasks.

we can note that inserting a quality control mechanism will actually improve performance (table 4.10). However, this experiment was only carried out on 2 tasks, because carrying out it on 5 tasks can be too blocking, This happens because if the teacher becomes a victim of catastrophic forgetting he is no longer in line with the ground truth.

By carrying out tests also on Icarl 4.6 4.5, it is possible to note that this model also suffers from catastrophic forgetting, especially when the number of tasks is greater. Subsequently other attempts were made using a convolutional Autoencoder. As can be seen in table 4.11 the first epochs of this model manage to be better than a pre-trained model when the number of epochs is greater. The possible reason is that an autoEncoder is an unsupervised feature extractor. His training is

| name | no quality | with quality control |
|---|---|---|
| lower bound | 43,2 | |
| 1 epoch | 45,98 | 52,31 |
| 2 epoch | 46,25 | 55,15 |
| 5 epoch | 46,88 | 49,45 |
| Upper bound | 82,2 | |

Table 4.10. results on Cifar10 using a resnet model pretrained as discriminator and only 2 tasks.Using a quality control mechanism on the generated images the accuracy can improve.

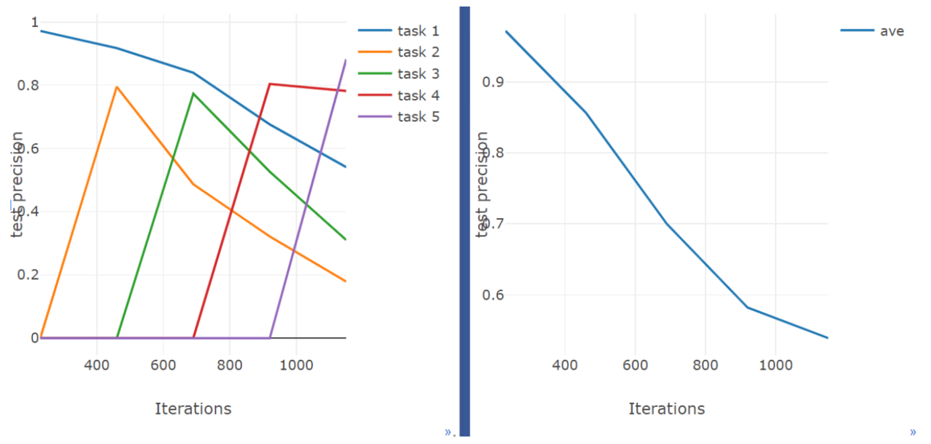


Figure 4.5. Icarl:Accuracy using resnet pre-trained on CIFAR10 on 5 tasks

based on how good it is to reconstruct the image. So adding new classes changes it less. Furthermore, being a feature extractor, it can put generated images and original dataset on the same level.

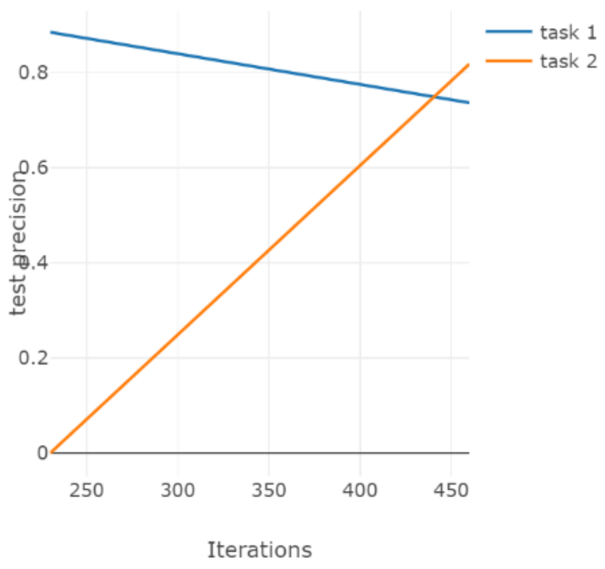


Figure 4.6. Icarl: Accuracy using resnet pre-trained on CIFAR10 on 2 tasks.

| name | 2 task | 5 task |
|---|---|---|
| lower bound | 38,74 | 18,41 |
| 1 epoch | 42,84 | 30,18 |
| 2 epoch | 42,07 | 23,61 |
| 5 epoch | 43,36 | 22,84 |
| Upper bound | 77,05 | |

Table 4.11. results on Cifar10 using a convulational model as discriminator . With this comparison, a degradation of performance was observed in the 5 tasks case, increasing the training time per task

## 4.3 Bio-dataSet

This dataset, kindly provided by the author of [Pon+19], was born as a need to create a convolutional model capable of distinguishing adenocarcinomas from healthy tissues. In this case the images of the entire dataset share the same colors and the cells are never centered.
The overlap of the characteristics of different classes is so high that simply classifying them normally can be difficult. The replicas obtained from the generative models are very generalized, managing to distinguish them slightly with naked eyes thanks to the shape.
However, by performing a training on the generated images only, the discriminator cannot distinguish them (<20% accuracy on the test set). For this reason, very low

| name | no quality | with quality control |
|---|---|---|
| lower bound | 38,74 | |
| 1 epoch | 42,84 | 42,36 |
| 2 epoch | 42,07 | 41,81 |
| 5 epoch | 41,36 | 41,40 |
| upper bound | 77,05 | |

Table 4.12. results on Cifar10 using a convolutional model as discriminator and only 2 tasks. When the discriminator is not capable of extract all the features the quality controll is useless.

results should be expected.

| | I-carl | rtf | NG |
|---|---|---|---|
| lower bound | 33,3 | | |
| | | | |
| 1 epoch | 33,53 | 33,3 | 33,3 |
| 2 epoch | 33,91 | 33,3 | 42,81 |
| 5 epoch | 44,23 | 33,3 | 34,64 |
| | | | |
| upper bound | 72,2 | | |

Table 4.13. results on Biological using pretrained resnet as common discriminator on 2 task.

| name | no quality | with quality control |
|---|---|---|
| lower bound | 33,33 | |
| 1 epoch | 33,3 | 33,3 |
| 2 epoch | 39,08 | 42,81 |
| 5 epoch | 34,2 | 34,64 |
| upper bound | 71,2 | |

Table 4.14. results on Biological dataset using a resnet model as discriminator and only 2 tasks. .

# Chapter 5

# Conclusions and future works

## 5.1 Conclusion

From the results obtained we can see that it is not easy to get good results from generative models. When the correlation of the characteristics of different images belonging to the same class is high, the generative model knows how to replicate excellent results. When this is low with more complex datasets such as Cifar10 the few characteristics captured are not sufficient to be able to recognize the class.

The difficulty lies primarily in the choice of the generative model, an element that is always evolving. However, the difference between original and generated images can be balanced by a good feature extractor. In fact, a pre-trained model has proven to have better results than a generic convolutional model.

It was then noted that different results can also be obtained on the basis of the number of classes that are faced at a time. For this reason, it is convenient to try to have large rather than small tasks.

This technique was also unstable with a decrease in accuracy increasing the training iterations. This is mainly due to the quality of the generated images: if they are of low quality the model adapts to that format forgetting the original dataset. The speed with which this happens is high and therefore a low number of epochs is preferable. However, if you use a pre-stretched model this memory loss is lower allowing you to learn new classes too.

To improve this phenomenon, a quality control of the training images used with knowledge distillation can be useful. In fact, using a generative model, more significant images can be generated than others. The added value from quality control represents the percentage of good quality images out of all the generated images. If adding quality control there is no improvement it means that the generated images are the same or completely different to the original dataset

In the presence of multiple tasks the structure of auto-encoder + discriminator can be useful. This is because in the transition from one task to another, unlike a normal model, the initial layers remain unchanged as you want to minimize the reconstruction error.

The fact that it requires the use of a few epochs reveals a strong connection with the requirements imposed by incremental learning. In fact, it allows you to learn new classes without having to waste too much time reviewing all previous classes

The results obtained are different from RtF because this has exploited VaE, not excellent for complex datasets, and above all because rtf uses a single model to generate and predict. However this work is still far behind models such as Icarl which, however, keeps real samples by class

## 5.2 Future Work

Generative models have great potentional skills but It is not convenient using only them in the role of consolidation. One solution would be to try using them in a dual memory approach .This would involve building a model similar to fearnet but using generative models in memory consolidation instead of class statistics. The generators could therefore be trained with the most characteristic images of the class chosen through the criteria of the second model. Another case study path could be increasing the resolution of the images during training (the work carried out focused on 32x32 low resolution images) for more complex datasets such as the biological one. Since a quality control has taken effect, further control should be carried out without the need to have a generator per class but only 1.Finally, it is obvious that a better image generator is needed especially in cases of complex datasets

# Bibliography

[He+15] Kaiming He et al. «Deep Residual Learning for Image Recognition». In: *arXiv:1512.03385 [cs]* (Dec. 10, 2015). arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385 (visited on 01/05/2020).

[HVD15] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. «Distilling the Knowledge in a Neural Network». In: *arXiv:1503.02531 [cs, stat]* (Mar. 9, 2015). arXiv: 1503.02531. URL: http://arxiv.org/abs/1503.02531 (visited on 01/10/2020).

[KS15] Kristo and er Stensbo-Smidt. «The Perceptron Algorithm». In: 2015.

[Doe16] Carl Doersch. «Tutorial on Variational Autoencoders». In: *arXiv:1606.05908 [cs, stat]* (Aug. 13, 2016). arXiv: 1606.05908. URL: http://arxiv.org/abs/1606.05908 (visited on 01/27/2020).

[Kir+16] James Kirkpatrick et al. «Overcoming catastrophic forgetting in neural networks». In: *arXiv:1612.00796 [cs, stat]* (Dec. 2, 2016). arXiv: 1612.00796. URL: http://arxiv.org/abs/1612.00796 (visited on 10/11/2019).

[Reb+16] Sylvestre-Alvise Rebuffi et al. «iCaRL: Incremental Classifier and Representation Learning». In: *arXiv:1611.07725 [cs, stat]* (Nov. 23, 2016). arXiv: 1611.07725. URL: http://arxiv.org/abs/1611.07725 (visited on 10/11/2019).

[Rus+16] Andrei A. Rusu et al. «Progressive Neural Networks». In: *arXiv:1606.04671 [cs]* (June 15, 2016). arXiv: 1606.04671. URL: http://arxiv.org/abs/1606.04671 (visited on 10/11/2019).

[Dra+17] T. J. Draelos et al. «Neurogenesis deep learning: Extending deep networks to accommodate new classes». In: *2017 International Joint Conference on Neural Networks (IJCNN)*. 2017 International Joint Conference on Neural Networks (IJCNN). May 2017, pp. 526–533. DOI: 10.1109/IJCNN.2017.7965898.

[Gul+17] Ishaan Gulrajani et al. «Improved Training of Wasserstein GANs». In: *arXiv:1704.00028 [cs, stat]* (Dec. 25, 2017). arXiv: 1704.00028. URL: http://arxiv.org/abs/1704.00028 (visited on 01/09/2020).

[KK17] Ronald Kemker and Christopher Kanan. «FearNet: Brain-Inspired Model for Incremental Learning». In: *arXiv:1711.10563 [cs]* (Nov. 28, 2017). arXiv: 1711.10563. URL: http://arxiv.org/abs/1711.10563 (visited on 10/11/2019).

[LH17] Zhizhong Li and Derek Hoiem. «Learning without Forgetting». In: *arXiv:1606.09282 [cs, stat]* (Feb. 14, 2017). arXiv: 1606.09282. URL: http://arxiv.org/abs/1606.09282 (visited on 01/09/2020).

[ZPG17] Friedemann Zenke, Ben Poole, and Surya Ganguli. «Continual Learning Through Synaptic Intelligence». In: *arXiv:1703.04200 [cs, q-bio, stat]* (Mar. 12, 2017). arXiv: 1703.04200. URL: http://arxiv.org/abs/1703.04200 (visited on 10/11/2019).

[Cha+18] David Charte et al. «A practical tutorial on autoencoders for nonlinear feature fusion: Taxonomy, models, software and guidelines». In: *Information Fusion* 44 (Nov. 2018), pp. 78–96. ISSN: 15662535. DOI: 10.1016/j.inffus.2017.12.007. arXiv: 1801.01586. URL: http://arxiv.org/abs/1801.01586 (visited on 01/27/2020).

[DV18] Vincent Dumoulin and Francesco Visin. «A guide to convolution arithmetic for deep learning». In: *arXiv:1603.07285 [cs, stat]* (Jan. 11, 2018). arXiv: 1603.07285. URL: http://arxiv.org/abs/1603.07285 (visited on 01/06/2020).

[HCK18] Tyler L. Hayes, Nathan D. Cahill, and Christopher Kanan. «Memory Efficient Experience Replay for Streaming Learning». In: *arXiv:1809.05922 [cs, stat]* (Sept. 16, 2018). arXiv: 1809.05922. URL: http://arxiv.org/abs/1809.05922 (visited on 10/13/2019).

[Kar+18] Tero Karras et al. «Progressive Growing of GANs for Improved Quality, Stability, and Variation». In: *arXiv:1710.10196 [cs, stat]* (Feb. 26, 2018). arXiv: 1710.10196. URL: http://arxiv.org/abs/1710.10196 (visited on 01/08/2020).

[Liu18] Yu Liu. «Feature Extraction and Image Recognition with Convolutional Neural Networks». In: *Journal of Physics: Conference Series* 1087 (Sept. 1, 2018), p. 062032. DOI: 10.1088/1742-6596/1087/6/062032.

[VT18] Gido M. van de Ven and Andreas S. Tolias. «Generative replay with feedback connections as a general strategy for continual learning». In: *arXiv:1809.10635 [cs, stat]* (Sept. 27, 2018). arXiv: 1809.10635. URL: http://arxiv.org/abs/1809.10635 (visited on 10/11/2019).

[ML19] Davide Maltoni and Vincenzo Lomonaco. «Continuous Learning in Single-Incremental-Task Scenarios». In: *arXiv:1806.08568 [cs, stat]* (Jan. 22, 2019). arXiv: 1806.08568. URL: http://arxiv.org/abs/1806.08568 (visited on 02/26/2020).

[Pon+19] Francesco Ponzio et al. «Dealing with Lack of Training Data for Convolutional Neural Networks: The Case of Digital Pathology». In: *Electronics* 8.3 (Mar. 2019), p. 256. DOI: 10.3390/electronics8030256. URL: https://www.mdpi.com/2079-9292/8/3/256 (visited on 01/27/2020).

[RPR19] Deboleena Roy, Priyadarshini Panda, and Kaushik Roy. «Tree-CNN: A Hierarchical Deep Convolutional Neural Network for Incremental Learning». In: *arXiv:1802.05800 [cs, eess, stat]* (Sept. 8, 2019). arXiv: 1802.05800. URL: http://arxiv.org/abs/1802.05800 (visited on 01/27/2020).

[Wu+19] Chenshen Wu et al. «Memory Replay GANs: learning to generate images from new categories without forgetting». In: *arXiv:1809.02058 [cs]* (Sept. 23, 2019). arXiv: 1809.02058. URL: http://arxiv.org/abs/1809.02058 (visited on 01/27/2020).

[Hem+20] Mehrali Hemmatinezhad et al. «PREDICTING THE SUCCESS OF NATIONS IN ASIAN GAMES USING NEURAL NETWORK». In: (Jan. 27, 2020).

[RLS] Javier Ruiz-del-Solar, Patricio Loncomilla, and Naiomi Soto. «A Survey on Deep Learning Methods for Robot Vision». In: (), p. 43.