TECHNICAL UNIVERSITY OF CARTAGENA

DEPARTMENT OF INFORMATION AND COMMUNICATION TECHNOLOGIES

Bachelor Thesis in Telecommunications Engineering

PERFORMANCE ASSESSMENT OF CAPSULE NETWORKS ON DIFFERENT APPLICATION SCENARIOS

Author: **Antonio Oliva Aparicio**

Mentor: **Ph.D. Jorge Larrey Ruiz**

Cartagena, September 2019

Acknowledgments

I would like to thank my supervisor PhD Jorge Larrey for the opportunity to work in such an interesting field and for his support along this project. My thanks extend, as it could not be otherwise, to my family and closer friends, both have been an inspiration throughout my life and have helped me get this far.

Abstract

Capsule networks are newborn deep neural networks that substitute traditional artificial neurons by vectors of them called 'capsules'. This new entities are thought to allow systems to understand the instantiation parameters of objects and induce the construction of an abstracted geometry from them. Thus, capsules enable to achieve a deeper knowledge of the object whose extrapolation procures a further generalization. These new architectures have been compared to the traditional Convolutional Neural Networks (CNNs) on four different scenarios to study their performance. Three of this scenarios are the commonly used datasets: MNIST, CIFAR-10 and SmallNORB; while the other is a set of retinographies containing non-pathological cases and others presenting glaucoma. The goal is to determine whether any of these networks is viable for an early detection of glaucoma. In addition, there is a discussion about the capacity of the structures based on capsules considered to understand and reconstruct the input images.

Resumen

Las redes de cápsulas son un tipo de redes neuronales profundas de reciente creación cuya esencia reside en la sustitución de las tradicionales neuronas artificiales por una versión vectorizada conocida como "cápsula". Esta nueva entidad está pensada para permitir que los sistemas puedan comprender los parámetros de instanciación de los objetos para así permitir la creación de una geometría abstraída de los mismos. De este modo, las cápsulas habilitan la consecución de un conocimiento más profundo de los objetos, y extrapolar este conocimiento conduce a una mejor generalización. Estas nuevas arquitecturas han sido comparadas con las Redes Neuronales Convolucionales (CNNs, por sus siglas en inglés) en cuatro escenarios diferentes para poder evaluar su rendimiento. Tres de estos escenarios son bases de datos utilizadas comúnmente: MNIST, CIFAR-10 y SmallNORB; mientras que la última es un conjunto de retinografías que contiene casos con glaucoma y casos no patológicos. El objetivo es determinar si alguna de estas redes es viable para una detección temprana del glaucoma. Además, se discute la capacidad de las estructuras basadas en cápsulas para entender y reconstruir las imagénes de entrada.

Contents

Part I

Introduction

The field of Artificial Intelligence is experimenting a golden moment in research and development of new systems and algorithms that allow to perform some tasks which were impossible to do little time ago. One of the main application fields is the so-called **Machine Learning**. This discipline tries to create new structures of autonomous learning so that the systems are capable of learning by themselves and improving from experience without being explicitly programmed. The inflection point in the take-off of Machine Learning has possibly been the rise of **Deep Learning**. It has become such an important topic because of recent theoretical research but also the improvement in the computer processing and memory capabilities. This sub-field basically divides complex problems in simpler ones by adding new steps to reach the solution, establishing a hierarchy of concepts along the network.

Regarding image processing application, the typically used structure is the *Convolutional Neural Network* (CNN), which has brought the best performance compared to its opponents, dominating the state-of-the-art. These systems are based on the use of filters or kernels with trainable values. These kernels are able to detect some features that a high-reasoning deeper stage can interpret. Nevertheless, a new competitor has appeared onto the stage: the *Capsule Network* (CapsNet). It introduces some innovative changes in traditional techniques used in CNNs that are rising it to the state-of-the-art in image classification. The most important concept is the idea of capsules itself. Capsules are groups of neurons whose dimensionality is associated to an upper level of abstraction, which is needed for achieving a solid knowledge that improves later generalization.

This thesis contains an overall approach to some theoretical concepts that must be understood before facing a later discussion of the results. In the next section, a description of the used methodology to compare the networks is given. Then, a presentation of the datasets employed as well as the two CNN architectures that are considered to compete against CapsNet and the CapsNet itself. Finally, a presentation of the results obtained and the conclusions extracted.

Part II

Theoretical background

Along this chapter some fundamental concepts of machine and deep learning are covered. In addition, the theory behind Convolutional Neural Networks (CNNs) and Capsule Networks (CapsNets) is also explained.

1 Machine Learning basics

Machine Learning is a subfield of Artificial Intelligence whose target is the development of computer structures capable of achieving autonomous learning. In particular, we are interested in the so-called **supervised learning**, where these structures are fed with data whose properties are known a priori by the programmer, so that it can be taught which output is desired for each input signal.

1.1 Task

The task is the learning goal of the algorithm. There are many kind of tasks according to the purpose of the network; however, the only task performed in this thesis is **classification**, so it is the single one to be covered here. Classification consists in mapping the input vector X to one specific label within a finite set of K possible labels or classes. Mathematically, it shall be synthesized:

$$f : \mathbb{R}^n \mapsto \{1, ..., K\} \tag{1}$$

1.2 Training, Validation & Test

Training is the process by which the algorithm tries to get knowledge from experience in the form of a set of samples. In the image classification task, the network is fed with input images and the output is compared to the label it was supposed to be assigned. The weights or parameters of the networks are updated so as to get a better approximation to the desired output. Thus, the goal is to iteratively minimize the"distance" between the real output and the desired one, quantizing a cost or loss function. Because of this process, the programmer does not need to adjust the values of any parameter/filter; instead, an architecture must be properly designed and trained with data.

Nevertheless, there are some parameters the programmer can adjust so as to improve the performance of the network: they receive the name of *hyperparameters*. In order to provide the programmer with some unbiased information about the results that training is achieving, there may be used a dataset for **validation**. At the end of each epoch of training, the network is fed with this dataset and some metrics are calculated. Then, the developer or the program itself can determine whether to stop the process and fit the hyperparameters or not. The key-point is that the network does not have access to the real label so it does not learn here, only the program and the programmer can learn how it is working.

Finally, the network is subjected to a **test** set. The purpose is to estimate the error when the architecture must face an unknown input data. The metrics obtained here shall determine the viability of the system for unseen data. Again, the network does not learn with this set. We are only interested in quantizing its later performance when making predictions.

1.3 Underfitting & Overfitting

The two aims of the processes of the previous section are reducing the training error and narrowing the gap between training and test error, what is called *generalization error*. The former is associated to a **underfitting**, the latter to **overfitting**. Underfitting occurs when the model is not capable of reaching a satisfactory low error on the training set, while overfitting occurs when the generalization error is too large.

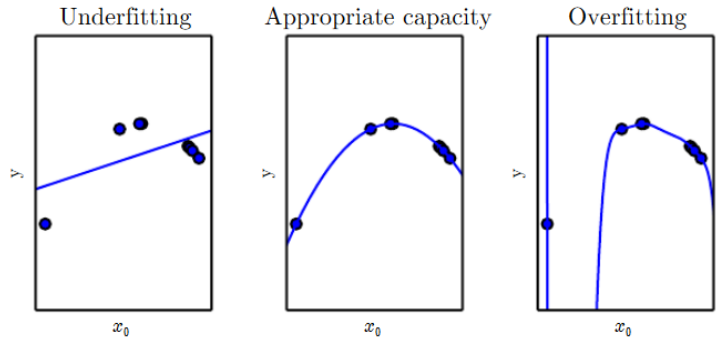


Figure 1: Graphical illustration of underfitting and overfitting.

It could be said that the larger the set of functions a network is able to fit, the greater the **capacity** of that network. Thus, an optimal model capacity shall be the one which ensures the minimum training and generalization error. Below the optimal, the model may not be able to fit the training set. Above it, the model might memorize features of the training set that are not extrapolated to the test set.

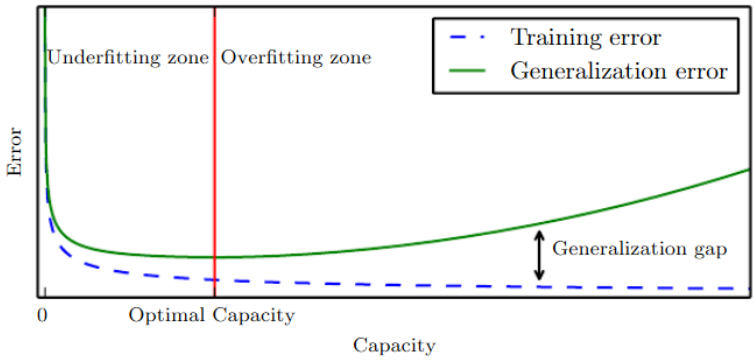


Figure 2: Graphical illustration of typical relationship between capacity and error.

1.4 Regularization

All Deep Learning systems strive to avoid the ghost of overfitting. **Regularization** is a helpful technique used in this struggle. An overfitted network has poor prediction and high generalization error, providing that it sticks too much to the data and probably is learning the background noise.

One way of adjusting the capacity of the model is by calculating the bias and the variance of the algorithm. The bias is the difference between the expected value of the prediction and the real value. The variance describes random variations between training sets generated by random nature of the variables in the algorithm and noise in the training data. Many regularization methods consist in penalizing the loss function adding a term in order to compensate overfitting. The calculation of this term depends on the chosen method.

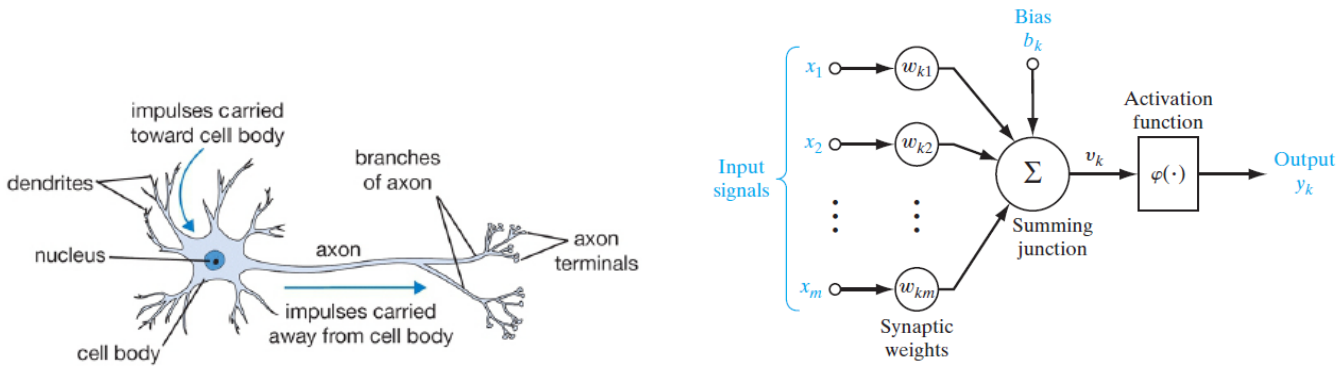
2 Artificial Neural Networks

The fundamental structure in Machine Learning is the *Artificial Neural Network*. The name comes from the neurons in the brain, providing that human brain is the most complex and intelligent system currently known. Thus, the purpose commonly consists in the imitation of human brain. The problem is that we do not have much information of the learning process inside our brain. The very few things we know is that the brain is composed of processing units called *neurons*, the relationship among them is extremely complex and the information is processed in parallel and in a non-linear way. These are the basics for the construction of an Artificial Neural Network (ANN).

The model of a neuron, is composed of a set of *synapses*, or *connecting links*, each of them is weighted to differ its importance for the output and these weighted input signals are then summed in an *adder* so that the output contains information from all the inputs. Finally, an *activation function* is applied to the result. This function is necessary in order to adequate the range of the output signal to some finite permissible values and to include non-linear operations. In the figure 3 there is a drawing of this model: each signal x_j at the input of synapse j connected to neuron k is multiplied by the synaptic weight ω_{kj}. Sometimes it is recommended the use of bias b_k in terms of an *affine transformation* to the output of the adder. Mathematically, we could describe this model with a single equation:

$$y_k = \varphi(\sum_{j=1}^{m} \omega_{kj} x_j + b_k) \tag{2}$$

where x_1, x_2, ..., x_m are the input signals; ω_{k1}, ω_{k2}, ..., ω_{km} are the synaptic weights of neuron k; b_k is the bias of neuron k; $\varphi(\cdot)$ is the *activation function* and y_k is the output signal.



(a) Natural neuron.

(b) Model of a neuron.

Figure 3: Comparison between a natural neuron (a) and its mathematical model (b).

Neural networks are built with a certain number of neurons distributed in layers and connected among them. The distribution of the layers, the way they are interconnected and the activation function characterize the network itself. There could also be feedback links but they are not interesting in our case of application so we will just skip this explanation. Regarding the number of outputs, it is just a matter of the information we want to extract from, such as the number of admissible labels in classification tasks.

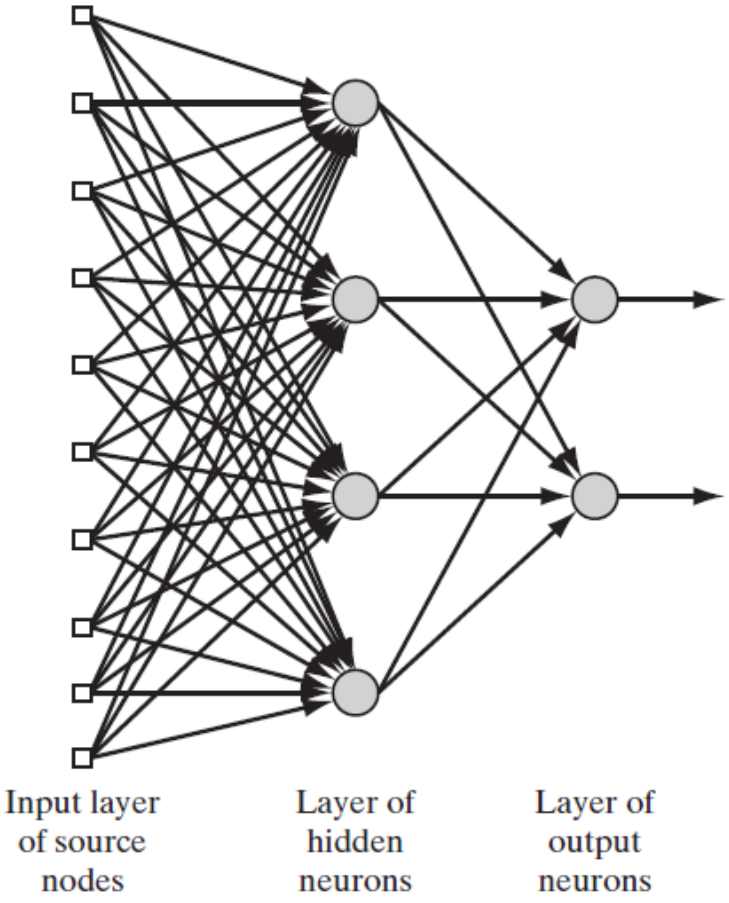


Figure 4: Model of a neural network with one hidden layer.

It is common to have one or more layers between input and output, these are called *hidden layers*. Normally, with more than one single hidden layer it is said to be Deep Learning. Therefore, 'deep' is just because we have increased the number of layers, the deepness of the network. The main drawback relies on the fact that the number of parameters shoots up, this is why it was mentioned before that the improvement in the computer processing and memory capabilities was so determinant for the rise of Deep Learning. However, in many cases the relationship between input and output is so complex that we need to fractionate it in the form of new hidden layers so that the understanding turns simpler.

2.1 Activation functions

As it was mentioned before, activation functions map the resulting values of the combiner in between a certain range. The traditional activation function used is the **Sigmoid** (or **logistic**) function. The main reason is because it compresses the values in between 0 and 1, providing a probability distribution.

$$\varphi(x) = \frac{1}{1 + exp(-x)} \tag{3}$$

However, the most common choice for an activation function in current deep neural networks is the so-called *rectifier*. A unit with this activation has the name of Rectified Linear Unit (**ReLU**). This function has proven to obtain better training results for deep networks. ReLUs are optimized quicker than other units with non-vanishing and higher order derivatives, since its derivative is either 0 or a positive constant value. One disadvantage of ReLUs is the fact that when activation is zero, gradient-descent methods are not able to learn and training stagnates.

$$\varphi(x) = x^{+} = max(0, x) \tag{4}$$

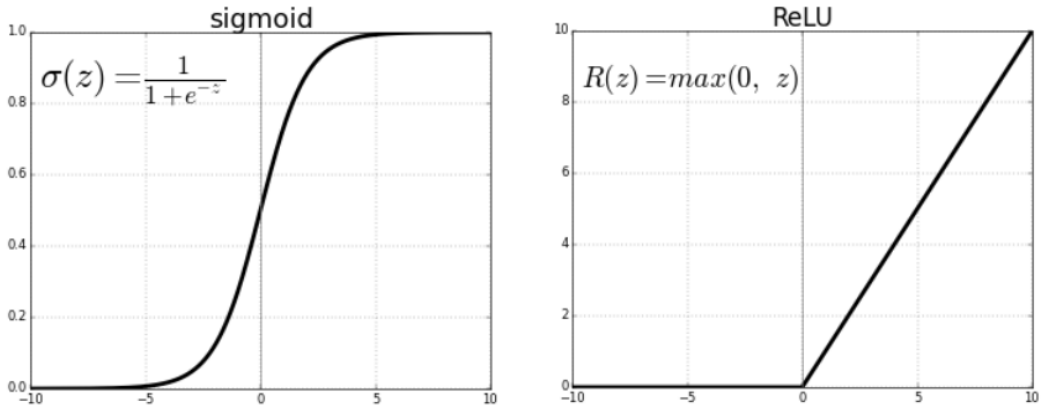


Figure 5: Sigmoid and ReLU plots.

3 Convolutional Neural Networks (CNNs)

CNN architectures widely domain the state-of-the-art in object recognition due to their outstanding performance. The key idea behind this kind of networks is the application of **feature detectors** to extract spatial information from pixels. These detectors are based on 2-D filters called *kernels*, whose values are adjusted during training. This technique allows to massively reduce the number of parameters to train if it is compared with a fully-connected layer, since the same kernels are applied to the whole image instead of having neurons completely connected to each other in consecutive layers. The operation performed between kernel and input is the convolution, which gives CNNs their well-known name. Each kernel produces a different *feature map*, so the set of feature maps that will be passed to the next layer will contain information of different aspects (features) of the input image. Convolution is a particular case of linear operation between functions of real valued argument that is denoted with the symbol $*$. An example of one-dimensional convolution $y(\xi)$ with a generic independent variable ξ between two functions $x(\xi)$ and $\omega(\xi)$ is defined as the integral of the dot product of these functions:

$$y(\xi) = x(\xi) * \omega(\xi) = \int_{\tau=-\infty}^{+\infty} x(\tau)\omega(\xi - \tau)d\tau \tag{5}$$

Providing that continuous operations are impossible to realize in computers because they are constrained by a finite precision, the discrete version of the above formula is expressed as follows:

$$y[n] = x[n] * \omega[n] = \sum_{k=-\infty}^{+\infty} x[k]\omega[n - k] \tag{6}$$

One single independent variable is being considered; however, images are defined in two-dimensional space. In fact, there is an extra dimension for the color coding in different channels but it does not affect really much the definition of convolution. Thus, the equivalent 2-D approach could be expressed with the following formula:

$$y[n_1, n_2] = x[n_1, n_2] * \omega[n_1, n_2] = \sum_{k_1=-\infty}^{+\infty} \sum_{k_2=-\infty}^{+\infty} x[k_1, k_2]\omega[n_1 - k_1, n_2 - k_2] \tag{7}$$

The convolution stage will be mostly determined by three parameters: the kernel size, which is the amount of pixels in width and height to consider for every position; the stride, which is the distance between consecutive positions where kernel is applied; and padding, which is the addition or not of zeros beyond the limits of the axis so as to be able to reach the boundaries. The remaining important element to design is the activation function performed to the output of the convolution. The default activation is usually ReLU.

After convolution, another operation is executed so as to achieve a certain level of translational invariance of a feature map's output, the so-called **pooling layer**. Another consequence of pooling is the reduction in the number of inputs to the next layer, which means a reduction of the size of feature maps and the amount of computation required, this is why pooling is also called *subsampling*. There might be considered several arithmetic functions that help us to get an approximately invariant output to small translations, scalings, skewing and other forms of distortion of the input. The most popular consists in computing the average or the maximum activation from a group of neurons, like a kernel. *Max Pooling* has proven to be the most successful method. This operation discards information that might be important; although, in some cases, it is not so significant to know specifically where the legs of a horse are as much as whether these legs are in the image, in order to determine the presence of the horse itself.

The remaining basic layer is a **dense** or **fully-connected layer**. Usually, this type of layer is placed at the deepest levels of the CNN so as to use the features extracted in the maps for a high-level reasoning. The structure is simple: each neuron in a dense layer is connected to every neuron in the previous layer. The output of this neurons is just described by the expression (2). Although their computation is quite straightforward, the size of the set of trainable parameters becomes too large in high dimensional data input like images; hence the need of a parameter sharing alternative like convolutional filters.

The final output of the network in classification must be a single label describing the object. Thus, the system needs a conversion of the activations of the last layer to this expected label. One common way of doing this is with the *softmax* function. It "shrinks" arbitrary real values z of a K-dimensional vector in a vector with the same dimensionality, but whose values are constrained in a range [0,1] and they must sum 1 all together. Both restrictions satisfy a probability distribution.

$$\varphi(z) = \frac{exp(z_j)}{\sum_{k=1}^{K} exp(z_k)} \; for \; j = 1, ..., K. \tag{8}$$

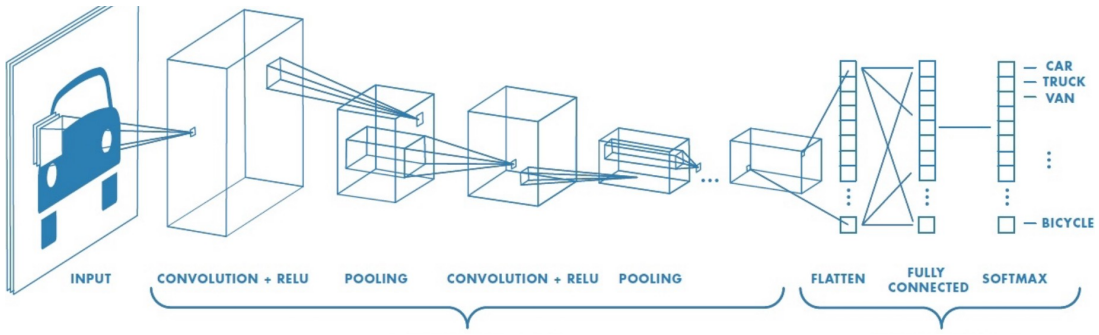


Figure 6: Architecture of a Convolutional Neural Network.

The basic CNN architecture contains all explained before and is represented in figure 6. There can be used some additional techniques to improve the performance of the networks. For instance, **Dropout** is a technique introduced by Hinton in 2012 [9] with the purpose of struggling against overfitting. It consists in dropping neurons during training with a certain probability, so that it does not contribute to neither forward nor backward propagation. Every input is forced to go through a different path, aiming to result into a different network. This provides of some robustness against memorizing, therefore against overfitting. It is basically a type of regularization method.

3.1 Limitations of CNNs

The most important drawback of CNN is that they are not capable of modeling spatial relationships satisfactorily. The only knowledge they can extract is from the data itself. For instance, if we want to recognize any object in 3D from any viewpoint, we would need images from all these viewpoints; otherwise, the network could not recognize precisely the concrete object. This is because CNNs are not able to generate an internal representation of the geometrical constraints of the data.

In addition, when applying *Max Pooling*, the system may be able to detect a feature, though it shall not determine the exact location due to the discarding of the lower activations within the range of the pooling kernel. A typical example used to understand this is illustrated in figure 7. Despite CNNs have a high activation of the different features or parts of human face, they are not able to comprehend that the image of the right is not a human face since its attributes are not located where they should be.

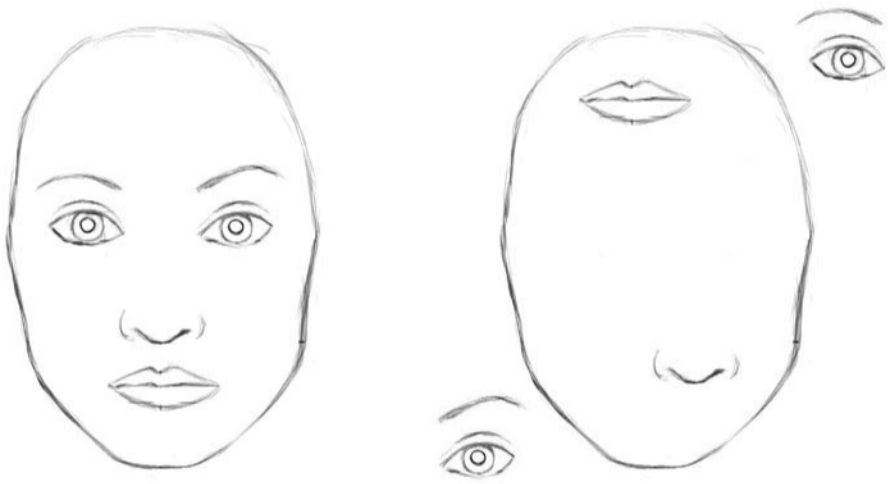


Figure 7: Example of limitation of a CNNs moving parts from a human face.

4 Capsule Networks

In 2011, Hinton et al. proposed an alternative to CNNs in *transforming auto-encoders* [2]. The most remarkable idea is the evolution of neurons as the basic elements of the architecture to "capsules" of neurons. The main difference is that capsules work with vectors in input and output instead of scalars. This change encourages a further comprehension of the feature learning in terms of its possible deformations and viewing conditions. Six years later, Hinton enhanced this alternative with a new paper where he explained the routing between layers of capsules and showed that the results of the first *Capsule Network* (CapsNet) could be comparable to state-of-the-art architectures on MNIST (handwritten digit dataset) and other common datasets [1]. The performance was particularly outstanding with overlapping digits and proved higher robustness against affine transformations.

4.1 Capsules

A capsule is merely a group of neurons whose outputs are associated to different properties of an entity in the input data, such as an object or a part of it in an image. Neurons provide a scalar activation while capsules use several neurons to form an activity vector that encodes the instantiation parameters of that entity. For instance, pose, brightness or thickness are encoded in the orientation of the vector and the probability of the presence of that entity relies on its length. As in neural networks, capsule networks are built with several layers composed of elementary units, though they are capsules in this case.

4.2 Transforming Auto-encoder

Transforming Auto-encoder was the first proposal with capsules as the essential architectural element [2]. In this primary design, the probability of the existence of a particular entity is encoded in one output value of the capsule and the instantiation parameters in the rest. This structure is in charge of generating the instantiation parameters for the capsules. The paper shows how it can be modified the pose of the objects in images, proving that the use of capsules permits a better understanding of the pose. This concept was promising but still quite limited because of the fact that the pose of the objects needed to be supplied externally.

4.3 Dynamic Routing between Capsules

The most remarkable success comes in 2017 with a new Hinton's publication [1]. It contains the model of the real first capsule network, called *CapsNet*, and the state-of-the-art performance results against typical datasets such as MNIST. The main advantage presented in the paper of CapsNet might be its robustness against affine transformations and the especially low error rate detecting overlapping digits. The key-point in this paper is that the pose of the objects does not need to be supplied externally anymore.

Hinton suggests substituting pooling operation of CNNs by a new kind of routing between capsule layers, called *dynamic routing-by-agreement*. For all but the last layer of capsules the output u_i of capsule i in layer l is forwarded to every capsule in the next layer $l+1$. The capsules of the higher level apply a weight matrix (W_{ij}) to u_i:

$$\hat{u}_{j|i} = W_{ij} u_i \tag{9}$$

where $\hat{u}_{j|i}$ is called "prediction vector", since it can be understood as the prediction from the capsules in the layer below. Then, the algorithm calculates a weighted sum s_j with the prediction vectors of previous capsules, where the coupling coefficients c_{ji} are determined by the iterative dynamic routing procedure.

$$s_j = \sum_i c_{ij} \hat{u}_{j|i} \tag{10}$$

In this architecture, the probability of the presence of the entity is encoded in the length of the vector. This can be obtained straightforward by applying a non-linear "**squashing**" function so that short vectors get contracted to almost zero length and long vectors get close to a length of 1. This function is analogous to the activation function in neural networks.

$$v_j = squash(s_j) = \frac{||s_j||^2}{1 + ||s_j||^2} \frac{s_j}{||s_j||} \tag{11}$$

where v_j is the vector output of capsule j and s_j is its total input. The coupling coefficients between capsule i and every capsule j in the higher level must sum 1. Their value is given by a *softmax* function whose arguments are the logits b_{ij}; these are the log prior probabilities that a capsule i should be coupled to capsule j.

$$c_{ij} = \frac{exp(b_{ij})}{\sum_k exp(b_{ik})} \tag{12}$$

The agreement a_{ij} is defined as the scalar product between the current output v_j of each capsule j and the prediction $\hat{u}_{j|i}$ performed by the capsule i from the previous layer. The log priors b_{ij} are updated every routing iteration by simply adding a_{ij} to its previous value: $b_{ij} = b_{ij} + \hat{u}_{j|i} \cdot v_j$. This procedure is continued as many iterations as wanted, starting with an initial value of 0 for every b_{ij}.

4.3.1 Margin loss function

The proposed loss function for this first CapsNet is *Margin loss*. It computes one value of loss for each class or label, based on the length of the output vector of the digit capsule. The total loss is just the sum of these individual values.

$$L_k = T_k \; max(0, m^+ - ||v_k||)^2 + \lambda(1 - T_k) \; max(0, ||v_k|| - m^-)^2 \tag{13}$$

where $T_k = 1$ if an object with label k is present and 0 otherwise, $m^+ = 0.9$ and $m^- = 0.1$. The λ down-weighting, by default 0.5, prevents the initial learning from shrinking the activity vectors of all classes. The interpretation of this formula might be something like: "if an entity with label k is present, then $||v_k||$ should be over 0.9. Otherwise, it should be below 0.1".

4.3.2 Architecture

This early architecture contains a first stage of feature detectors based in convolution, followed by a first layer of capsules called *PrimaryCaps* and a second capsule layer whose name is *DigitCaps*. The final stage is just a calculation of the probabilities associated to each digit capsule so as to determine the presence of the object(s). We will go step by step explaining the operations performed between each pair of consecutive layers considering a gray-scale 28×28 MNIST image as an input example.

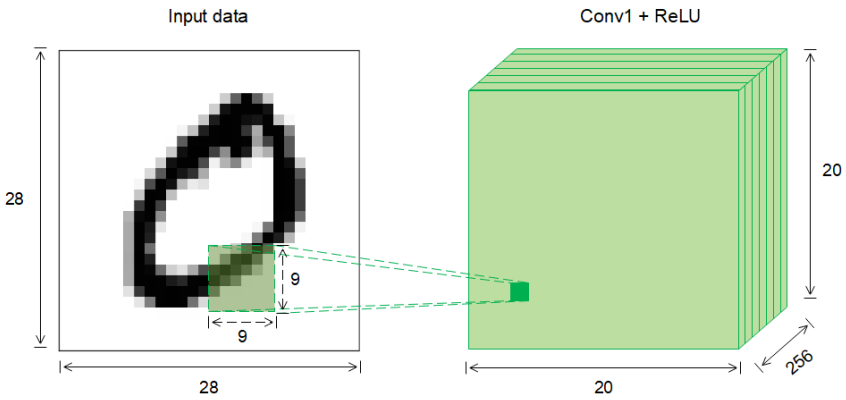


Figure 8: First feature detector in CapsNet.

In the figure 8, it can be seen that the first convolutional layer (Conv1) applies a kernel of 9×9 pixels to each color channel of each input image. Therefore, for gray-scale images (e.g. MNIST), the shape of the filters is 9×9×1. Since no zero padding is considered and the stride of this layer is 1, there is a reduction of the 2D size from input to feature maps of 9 - 1 = 8, resulting in maps of 20×20 pixels for MNIST. The chosen number of filters is 256, so the shape of this layer is 20×20×256. Finally, ReLU activation is used, although this does not change the shape of the layer.

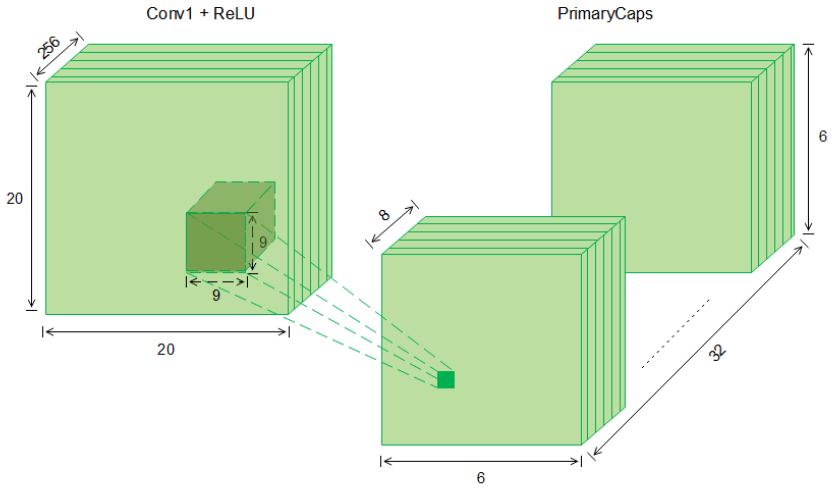


Figure 9: Second feature detector and reshaping to PrimaryCaps in CapsNet.

The following stage is illustrated in figure 9. A second feature detection is applied. At this time, the kernels are again 9×9 pixels, but for every map of the first convolutional layer, so their real size is 9×9×256. Again, no zero padding is considered, although the stride is 2 for width and height. Therefore, the size is scaled down by 9 - 1 = 8, and then, divided by 2. The final shape of the second convolutional layer is 6×6×256. In addition, ReLU is used anew. PrimaryCaps are just formed by reshaping this second convolutional layer, dividing the 256 maps into 32 *channels* of 8D capsules. In total, PrimaryCaps contains 32×6×6 = 1152 capsule vectors of length 8.

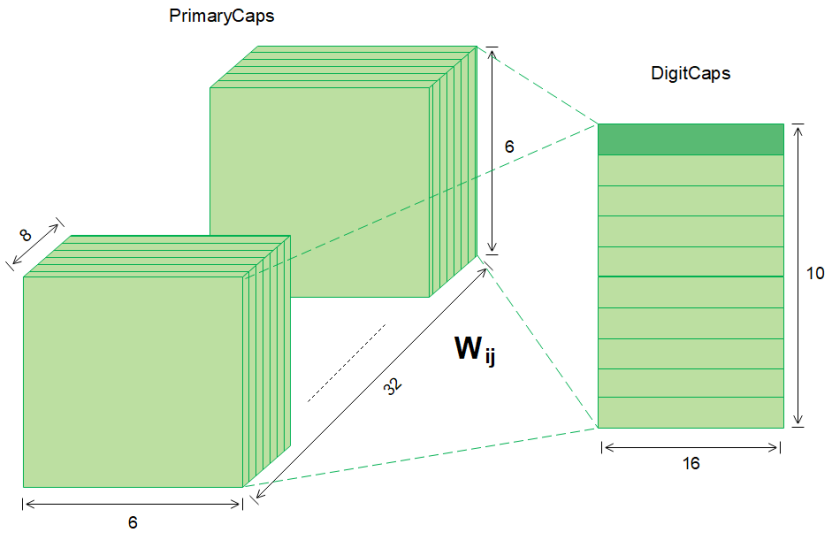


Figure 10: Routing between PrimaryCaps and DigitCaps in CapsNet.

Next stage connects the first (and last) two consecutive capsule layers. It is time now for the *dynamic routing-by-agreement*. As we said before, there are 1152 primary capsules with 8 dimensions, so we have 1152 vectors u_i of shape 1×8. DigitCaps contains 10 capsules (one per digit), with 16 dimensions each, so there are 10 vectors v_j of shape 1×16. Nevertheless, we need one prediction vector $\hat{u}_{j|i}$ per primary and digit capsules; therefore, 1152×10 = 11520 vectors. Thus, we also need 11520 weight matrices W_{ij} of shape 8×16. As we know, digit vectors are obtained from squashing the weighted sum s_j over the 1152 primary capsules.

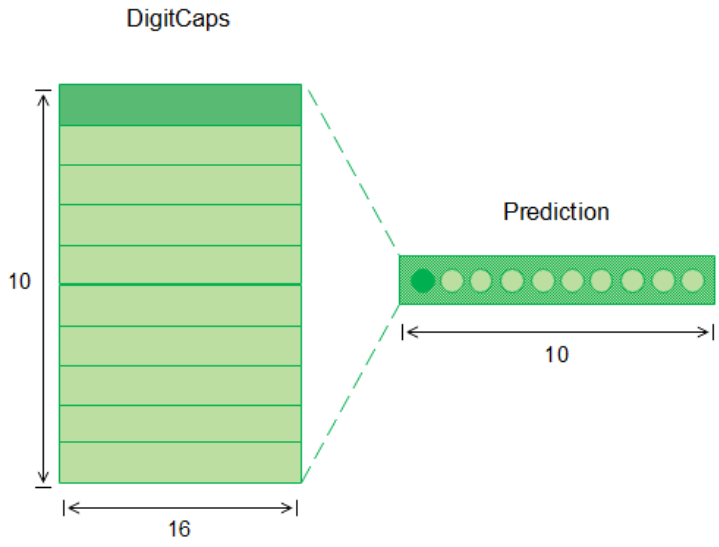


Figure 11: Predictions from DigitCaps in CapsNet.

The remaining stage of CapsNet architecture is just a calculation of the length of the digit vectors v_j in terms of their norm, so there are 10 values per image in which the highest activation determines the present digit. It is illustrated in figure 11. In addition, a perspective of the entire architecture is shown in figure 12.

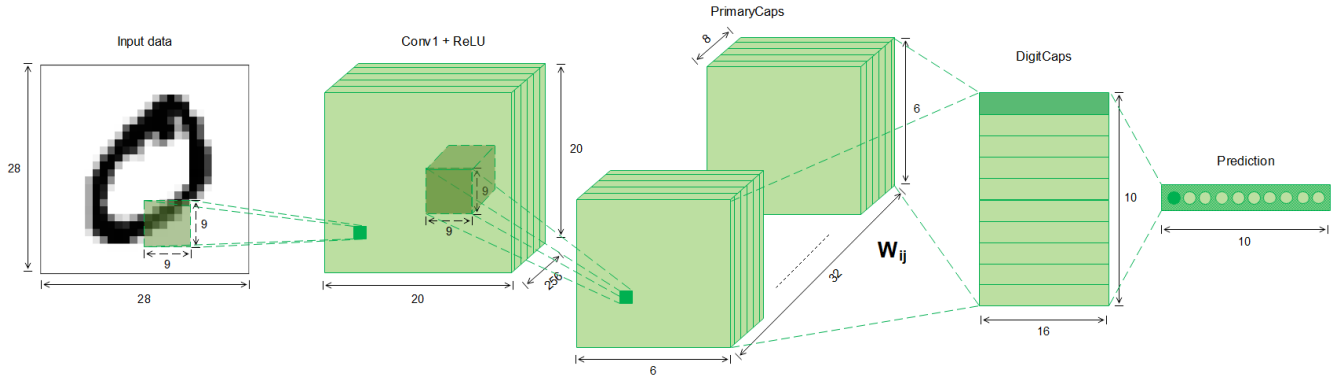


Figure 12: CapsNet architecture for MNIST input data.

4.3.3 Reconstruction

This technique is a sort of regularization method and is helpful for encouraging the digit capsules to encode the instantiation parameters of the input digit. During training, all the activity vectors are masked out with 0's except the one associated to the correct label. Therefore, the input to this stage is the whole set of digit vectors, although only the vector of the digit capsule associated to the correct label produces activation; all the rest are negligible. A decoder is added to the output of the digit capsules, consisting of 3 fully-connected layers modeling pixel intensities. The reconstruction loss is calculated as the squared difference between the input pixel intensities (x_{in}) and the output of the decoder (x_{out}). This cost function is added to the margin loss, although it is first multiplied by 0.0005 so that it does not dominate the margin loss:

$$L_{total} = \sum_{k} L_k + 0.0005 \cdot \sum_{r} (x_{in,r} - x_{out,r})^2 \tag{14}$$

where r denotes the indexes to go through these vectors and L_k is the margin loss calculated for each class k.

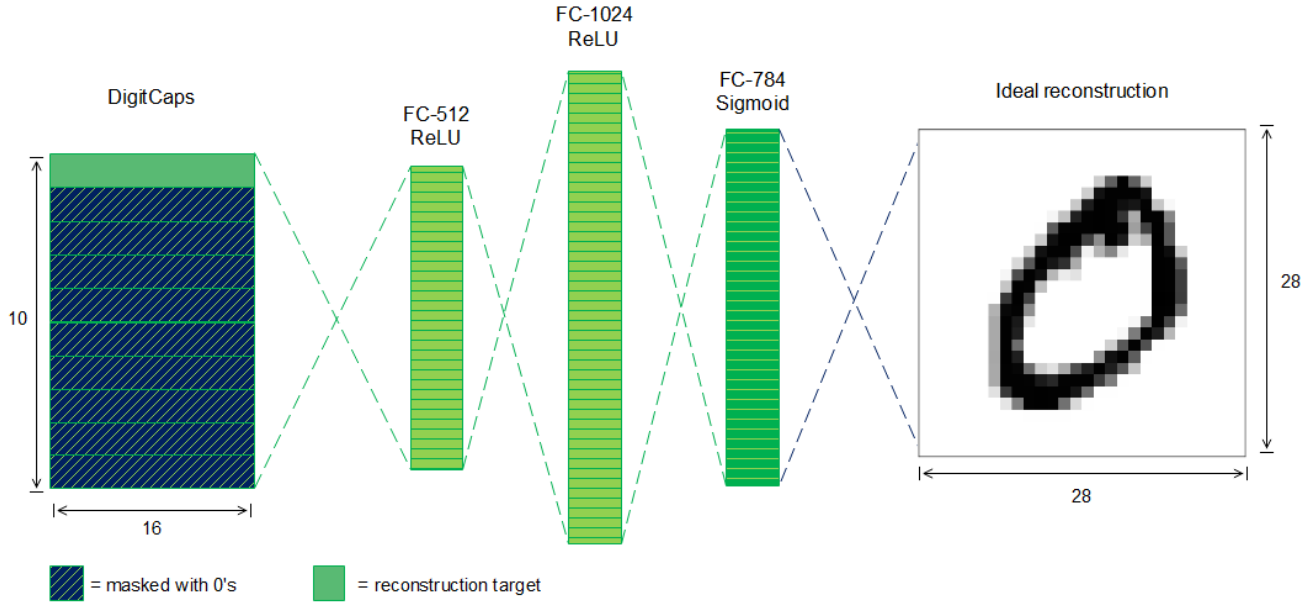


Figure 13: Decoder structure for reconstruction stage.

4.4 Equivariance

A system is said to be **invariant** with respect to a transformation when it is not possible to detect the change at its output. The target chased when applying sub-sampling is, apart from the reduction in the number of trainable parameters, making the neural activities invariant to small transformations. This a misconception, since the system would need to be fed with the whole pack of viewpoints in order to be able to detect the object from any viewpoint. Structures based on capsules apply another method so as to achieve "viewpoint invariance": the *equivariance*.

A system is said to be **equivariant** with respect to a transformation when a change at the input is detectable at the output. Human perceptual system is able to understand that all the pictures in figure 14 shows the same object but from different viewpoints. This is only an example of rotation, but the same would be applicable with lighting conditions, translation or thickness. The idea behind capsules is that they achieve to encode this properties.

The goal now is the procurement of a set of viewpoint-invariant weight matrices, capable of transforming the 3D mesh objects into 2D views in the same way as in Computer Graphics. Like this, systems are able to do the inverse operation and construct a geometrical abstraction of the object from the images of any viewpoint. Then, it becomes quite easy for a model to understand that the object it sees now is merely something it has seen before from another point of view. This would strongly reduce the number of needed data to train models based on capsules in comparison with traditional CNNs.

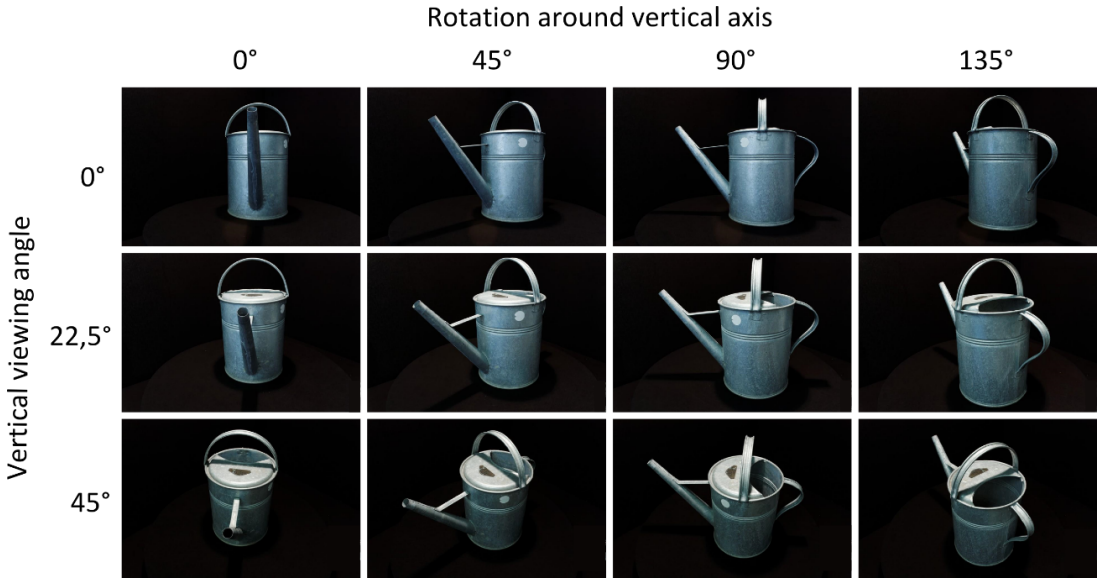


Figure 14: Different viewpoints of an object.

In Computer Graphics, these matrices are organized hierarchically to encode the relationship between a whole entity and its parts. This is what routing-by-agreement does. Coming back to figure 7, capsules in the lower level would represent the parts of the face (mouth, nose, eye and so on). Considering the face at the left, the agreement among the parts would be quite high, because the location of the parts and distance between them is coherent with the locations and distances learned during the training process. Nevertheless, the agreement in the case of the face at the right side would be quite low, because of the opposite reason. With capsules, the probability of a certain part-entity to be present is not modified if we rotate or move it, only the orientation of the vector changes; so that the agreement with other capsules changes. Understanding the relationships among the parts of a whole entity, we have achieved an internal representation of the geometry of the entity itself.

Part III

Experimental Setup

This section covers the setup of all the experiments whose results are shown in this thesis, the Deep Learning architectures that have been used as well as the datasets they have faced and the metrics used to evaluate them. The majority of simulations and codes have been developed in the Python programming language, concretely using Tensorflow library. MATLAB environment has also been used for image processing of the retinographies dataset.

5 Datasets

There have been considered four datasets: the first three are commonly used to compare deep neural networks while the other is a case of real application.

5.1 MNIST

The Modified National Institute of Standards and Technology (**MNIST**) [3] is a database of handwritten digits consisting of 60,000 samples for training and 10,000 for test. In our case, the training set is splitted into 55,000 images for training and 5,000 for validation. The size of the images is 28×28 in gray-scale coding.

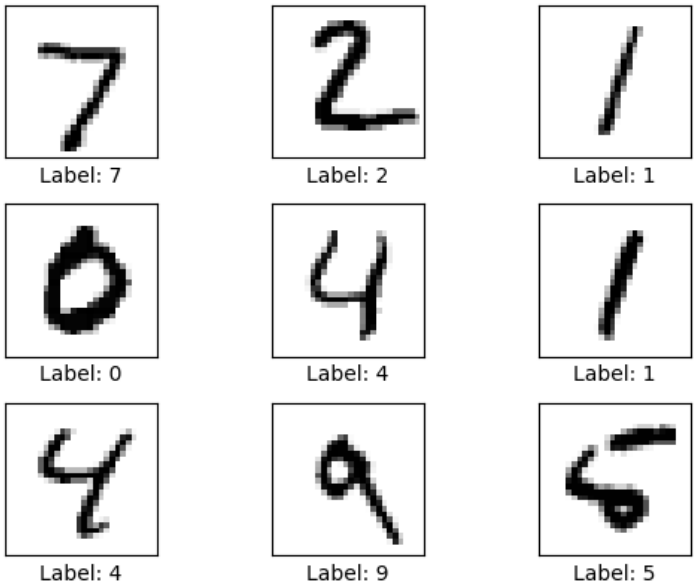


Figure 15: Examples of MNIST dataset.

5.2 CIFAR-10

The Canadian Institute For Advanced Research (**CIFAR-10**) is a collection of 32×32 color images in 10 classes, with 6,000 samples per class. There are 50,000 images are used for training, 5,000 for validation and 5,000 for test. The labels are numbers associated to “airplane”, “automobile”, “bird”, “cat”, “deer”, “dog”, “frog”, “horse”, “ship” and “truck”.

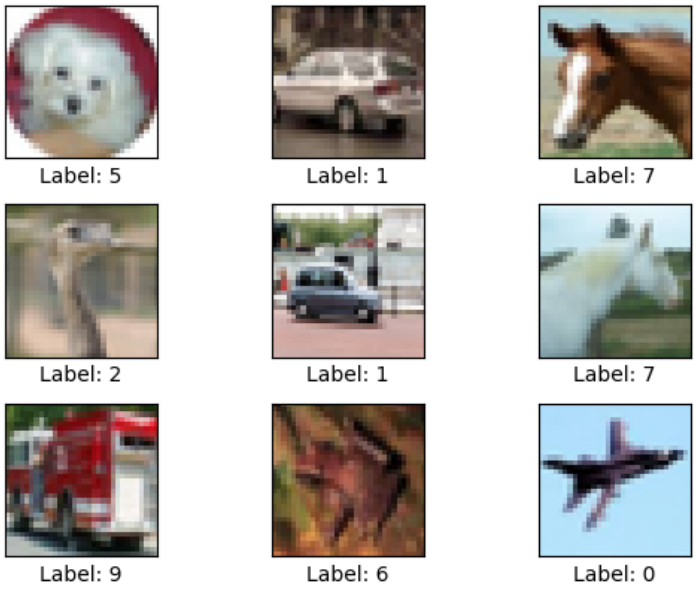


Figure 16: Examples of CIFAR-10 dataset.

5.3 SmallNORB

SmallNORB is the reduced version of NYU Object Recognition Benchmark (NORB). It contains images of 50 toys classified in 5 generic categories: "four-legged animals", "human figures", "airplanes", "trucks" and "cars", labeled with 0, 1, 2, 3 and 4 successively. These toys were photographed by means of two cameras under 6 lighting conditions and from different angles: 9 elevations and 18 azimuths, which makes this dataset specially indicated for 3D object recognition. It is originally separated into 48,600 samples for training and another 48,600 for testing, although we have splitted the last one into 23,300 for validation and the remaining 23,300 for testing. Images are 96×96 in gray-scale color coding. Before entering the network, they have been rescaled to 48×48. Then, random crops of 32×32 have been made for training while these crops have been taken in the center of the images for validation and test.

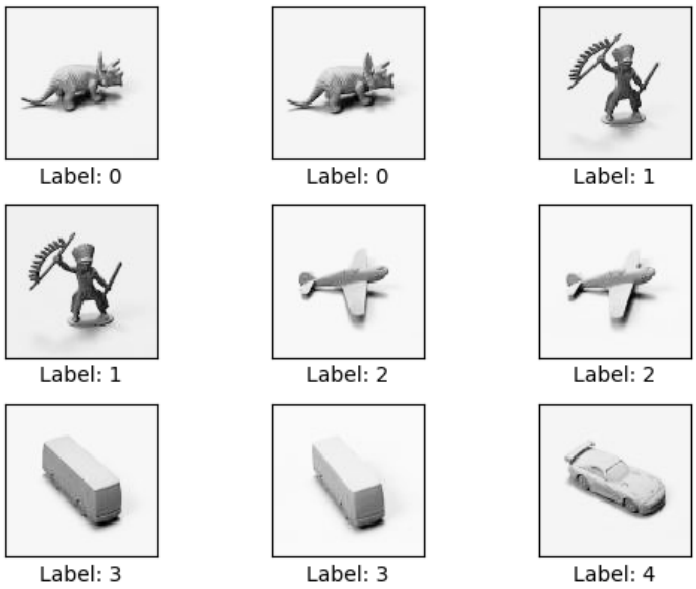


Figure 17: Examples of SmallNORB dataset.

5.4 Retinographies

Probably the main inspiration to begin this thesis was studying an alternative that allows an early detection of glaucoma from a patient's retinography. This alternative is based on deep learning architectures, capable of computing the probability of the presence of glaucoma in the retinography; so that doctors are able to work directly with people that have a high risk of suffering it. In the creation of this system, the following collaborate jointly: the TDAM research group, from the Technical University of Cartagena, and the Reina Sofía University General Hospital (HURS) in Murcia, Spain.

The original set of retinographies is composed of 123 cases with glaucoma and 133 non-pathological. Regarding the later, 23 are obtained from HURS and 110 from DRIONS-DB database [5], while all the glaucoma cases are obtained from HURS. The useful information is extracted from the optical nerve, so we obtain this part of the images by cropping them. In addition, they have been resized to 50×50 pixels so that the number of parameters is not too high. Finally, we have normalized the intensity range to [0,1]. Providing that the dataset is quite small, we have horizontally flipped the processed images to double the samples as a data augmentation technique. Some of the resulting images were not accurate because the nerve was not easy to distinguish, so they have been discarded. After balancing the dataset for both glaucoma and non-pathological cases, the whole set is composed of 242 cases for each, 484 in total.

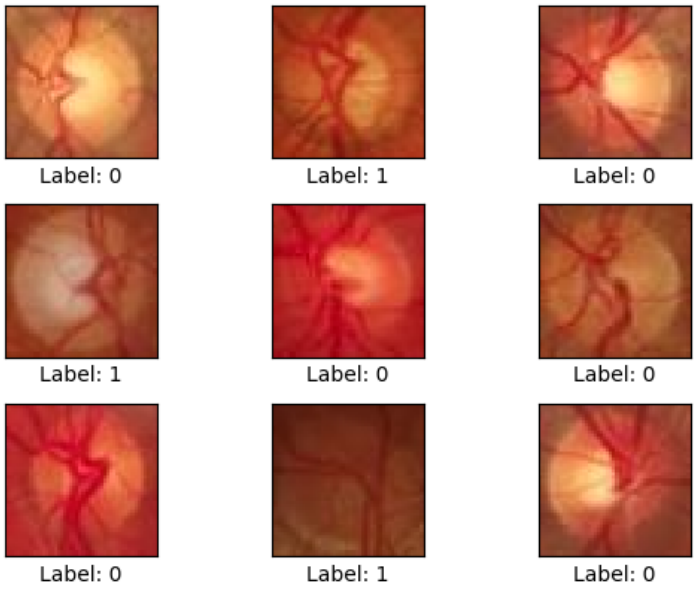


Figure 18: Example of original retinography against the processed one.

For splitting the data, we have followed a distribution 70%, 15% and 15% that, considering the approximation to the closest integer, has resulted into 338 for training, 72 for validation and 74 for test. All these sets are balanced in the number of each class.

6 Deep networks

The aim is to compare the performance of CapsNet in different scenarios with other typical architectures. The first one is a baseline CNN, since it is reasonable to compare CapsNet, as a newborn network, with a basic version of CNNs. The second one is the well-known AlexNet, an improved version of CNNs, which provides state-of-the-art results. This section presents all the layers and loss functions used in every one of them as well as the calculation of the number of parameters to train. The explanation of how to obtain this number is developed in a later subsection dedicated to this.

6.1 CNN baseline

This network is composed of two convolutional layer with max pooling and ReLU activation, followed by two fully-connected layers. Both convolutional and max pooling filters are applied using zero padding, so there is no more size reduction than with the stride. The first fully-connected layer uses ReLU activation while the second uses softmax to take an output based of probabilities.

CNN baseline	Conv1	Pool1	Conv2	Pool2	FC	Out
MNIST	16 of 5×5×1	stride=2	36 of 5×5×16	stride=2	128	10
CIFAR-10	16 of 5×5×3	stride=2	36 of 5×5×16	stride=2	128	10
SmallNORB	16 of 5×5×1	stride=2	36 of 5×5×16	stride=2	128	5
Retinographies	16 of 5×5×3	stride=2	36 of 5×5×16	stride=2	128	2

From previous table and the size of the input, we can calculate the total number of parameters to train.

Parameters	Conv1	Conv2	FC	Out	TOTAL
MNIST	16×(5×5×1+1)	36×(5×5×16+1)	(36×7×7+1)×128	10×(128+1)	242,062
CIFAR-10	16×(5×5×3+1)	36×(5×5×16+1)	(36×8×8+1)×128	10×(128+1)	311,982
SmallNORB	16×(5×5×1+1)	36×(5×5×16+1)	(36×8×8+1)×128	5×(128+1)	310,537
Retinographies	16×(5×5×3+1)	36×(5×5×16+1)	(36×13×13+1)×128	2×(128+1)	794,790

The cost function employed is the Cross-Entropy (CE) loss:

$$CE = -\sum_{n}^{N} t_n log(s_n) \tag{15}$$

where t_n is the target label, and s_n is the result of the softmax operation for label n in the set of classes N.

6.2 AlexNet

AlexNet is one of Deep Learning's flagship boats. Its deepness and other improvement techniques made it win the ILSVRC in 2012, a reference competition in image classification. The one we have used is a reduced version of AlexNet, provided that the original one was designed for input data of 256×256×3 while the images of the datasets considered are much smaller. All the convolutional and fully-connected layers use ReLU activation except the output dense layer, which implements softmax. Convolutional layers also use dropout as regularization method, with a probability to keep units of 0.8. All the filters use zero padding and the pooling layers use max pooling. The cost function is also Cross-Entropy (CE).

AlexNet	Conv1	Pool1	Conv2	Pool2	Conv3	Pool3	FC1	FC2	Out
MNIST	64 of 3×3×1	2	128 of 3×3×64	2	256 of 3×3×128	2	1024	1024	10
CIFAR-10	64 of 3×3×3	2	128 of 3×3×64	2	256 of 3×3×128	2	1024	1024	10
SmallNORB	64 of 3×3×1	2	128 of 3×3×64	2	256 of 3×3×128	2	1024	1024	5
Retino.	8 of 5×5×3	2	16 of 7×7×8	2	32 of 9×9×16	2	256	256	2

From previous table and the size of the input, we can calculate the total number of parameters to train.

AlexNet	Conv1	Conv2	Conv3
MNIST	64×(3×3×1+1)	128×(3×3×64+1)	256×(3×3×128+1)
CIFAR-10	64×(3×3×3+1)	128×(3×3×64+1)	256×(3×3×128+1)
SmallNORB	64×(3×3×1+1)	128×(3×3×64+1)	256×(3×3×128+1)
Retinographies	8×(5×5×3+1)	16×(7×7×8+1)	32×(9×9×16+1)

AlexNet	FC1	FC2	Out	TOTAL
MNIST	(4×4×256+1)×1024	1024×(1024+1)	10×(1024+1)	5,624,842
CIFAR-10	(4×4×256+1)×1024	1024×(1024+1)	10×(1024+1)	5,625,994
SmallNORB	(4×4×256+1)×1024	1024×(1024+1)	5×(1024+1)	5,619,717
Retinographies	(7×7×32+1)×256	256×(256+1)	2×(256+1)	516,370

6.3 CapsNet

The architecture of CapsNet has previously been explained; however, there are some changes between the structure used for one dataset and another.

Layers	Conv1	Conv2	Channels	PrimaryCaps	DigitCaps	Decoder
MNIST	256 of 9×9×1	256 of 9×9×256	32	8	16	512-1024-784
CIFAR-10	256 of 9×9×3	256 of 9×9×256	32	8	16	512-1024-3072
SmallNORB	256 of 9×9×1	256 of 9×9×256	32	8	16	512-1024-1024
Retino.	16 of 7×7×3	16 of 7×7×16	4	4	8	32-128-7500

From previous table and the size of the input, we can calculate the total number of parameters to train.

Parameters	Conv1	Conv2	Routing	Total
MNIST	256×(9×9×1+1)	256×(9×9×256+1)	10×(1152×8×16)	6,804,224
CIFAR-10	256×(9×9×3+1)	256×(9×9×256+1)	10×(2048×8×16)	7,992,576
SmallNORB	256×(9×9×1+1)	256×(9×9×256+1)	5×(2048×8×16)	6,640,384
Retinographies	16×(7×7×3+1)	16×(7×7×16+1)	2×(1444×4×8)	107,344

Parameters	Decoder	TOTAL
MNIST	512×(16×10+1)+1024×(512+1)+(28×28×1)×(1024+1)	8,215,568
CIFAR-10	512×(16×10+1)+1024×(512+1)+(32×32×3)×(1024+1)	11,749,120
SmallNORB	512×(16×5+1)+1024×(512+1)+(32×32×1)×(1024+1)	8,256,768
Retinographies	32×(8×2+1)+128×(32+1)+(50×50×3)×(128+1)	1,079,612

6.4 Calculation of the number of parameters

In a convolutional layer, we only have the parameters of the kernels, so its calculation is the number of kernels multiplied by the size of the kernels plus one because of the biases. In fully-connected layers, the number of parameters is the number of outputs times the number of inputs plus one; again, because of the biases. The number of inputs to the first fully-connected layer is computed in terms of the size reduction caused by the convolutional and pooling kernels. When zero padding is used, there is no more reduction than the division by the stride; when it is not used, there is a reduction equal to the width and height of the kernel minus one; then, the division by the stride. For example, in the figure 12, which shows the whole CapsNet architecture, from Conv1 to PrimaryCaps, the 20×20 maps are scaled down to 6×6 by subtracting 9 - 1 = 8 and dividing by a stride of 2.

Previous explanation is enough to compute the number of parameters in both CNNs; however, in CapsNet there are other variables to train. As it was shown in the section of the architecture of CapsNet, the number of primary capsules is the size of the grid multiplied by the number of channels. So, for MNIST, we have 6×6×32 = 1152. Thus, we have to compute 1152 W_{ij}, whose shape is 8×16, for each digit capsule. Therefore, in the case of MNIST, routing-by-agreement needs 10×(1152×8×16) parameters. In addition to this, we have to sum the parameters of the reconstruction stage. Since it is composed of fully-connected layers, there is no need of a further explanation but the input and output of the decoder. The input is the dimension of one digit capsule multiplied by the number of classes, since we take one digit vector for each label. The output is a flatten vector whose length is the total size of the input data; in MNIST, 28×28×1 = 784.

7 Metrics

The most common way of comparing the performance of Machine Learning systems is the calculation of the *test accuracy*. This value shows the number of successfully classified images over the total in an unknown dataset for the network. It is an estimation of the generalization behavior of the network when it faces new data. Since this value can suffer changes every time a simulation is run due to the random nature of the optimization process, the results we provide are the average and the standard deviation over several values of the accuracy. They are obtained from different simulations: 50 in the case of retinographies and only 2 for the rest, because computing many simulations with large datasets demands too much time for a basic CPU as the one that has been used.

$$accuracy = \frac{1}{N}\sum_{i=1}^{N}(p_i == t_i) \tag{16}$$

where p_1, p_2, ..., p_N are the predicted labels in the a set of N input images, t_1, t_2, ..., t_N are the target or true class labels of that set and " == " is the logic equal operand, whose result is 1 when the condition is satisfied.

8 Adam optimizer

Adaptive Moment Estimation (**Adam**) is a kind of adaptive learning rate algorithm that computes individual learning rates particularized to each parameter. Since it was first introduced in 2014 [10], it has become one of the most common methods of optimization for deep neural networks.

The *i-th* moment of a random variable is defined as the expected value of that random variable to the power of i. Mathematically:

$$M_i = E[X^i] \tag{17}$$

where M_i is the *i-th* moment of the random variable X. It is reasonable to consider the gradient of a loss function in a neural (or capsule) network as a random variable g_t, because it is commonly calculated from small random batches of data, a.k.a. *minibatches*. Adam uses estimations of the first and second moments of the gradient in order to adapt the learning rate. These moments are successively the mean m_t and the uncentered variance υ_t. The expression for their estimation is shown in (18).

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1-\beta_1) g_t \\ \upsilon_t &= \beta_2 \upsilon_{t-1} + (1-\beta_2) g_t^2 \end{aligned} \tag{18}$$

where β_1 and β_2 are hyperparameters known as *decay rates* and t refers to the current minibatch. Providing that m_t and υ_t are initialized to all 0's, the authors observed that the estimator is biased to zero. The target is an **unbiased estimator**, so the authors applied a correction of the bias.

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$$
$$\hat{v}_t = \frac{v_t}{1-\beta_2^t} \tag{19}$$

Once these calculations have been performed, we just need to update each weight with the following expression:

$$w_t = w_{t-1} - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \tag{20}$$

where η is the learning rate, ϵ is a small number used to avoid division by zero and w_t refers to any weight parameter of the network. Default values for β_1 and β_2 are 0.9 and 0.999 respectively. The ϵ parameter is 10^{-8} by default; however, this might make the training process unstable, so in some cases this value has been changed. It makes the training process longer but more stable. The following table shows the election of learning rates and ϵ for each network and dataset. In some cases where the convergence was more difficult to achieve we have used a learning rate decay, which means that the initial η is scaled down by a factor of 0.95 when it has iterated as many times as the number established by *global step* (g-step).

Network / Dataset	CNN baseline			AlexNet			CapsNet		
-	η_{ini}	g-step	ϵ	η_{ini}	g-step	ϵ	η_{ini}	g-step	ϵ
MNIST	10^{-4}	-	10^{-8}	10^{-4}	-	10^{-8}	10^{-4}	10000	10^{-5}
CIFAR-10	10^{-4}	-	10^{-8}	10^{-4}	-	10^{-4}	$2 \cdot 10^{-4}$	1000	10^{-6}
SmallNORB	10^{-4}	-	10^{-8}	10^{-4}	-	10^{-8}	$5 \cdot 10^{-5}$	-	10^{-8}
Retinographies	10^{-4}	50	10^{-5}	10^{-3}	5000	10^{-7}	10^{-4}	100	10^{-8}

Batch size is the amount of input samples we use to feed the algorithm, depending on the dataset and network this number has been modified in order to maximize the accuracy results, this converts it into a hyperparameter. On the one hand, a smaller batch usually provides a faster convergence of the gradient at the expense of the stability of the optimization process, since the algorithm is updating the weights more frequently. On other hand, a bigger batch requires more memory usage, which is limited by a RAM capacity of 4 GB in these experiments. The following table shows the value used in each case:

Network / Dataset	CNN baseline	AlexNet	CapsNet
MNIST	128	128	32
CIFAR-10	128	8	32
SmallNORB	100	10	10
Retinographies	128	8	128

Part IV

Results

The final results are presented through this section. It contains the final values for test accuracy, plots that show the evolution of the validation accuracy and also graphs with the progress of both margin and reconstruction loss functions as well as the output of the decoder for capsule networks.

The discussion about them is done later while the explanation of how these results have been obtained was developed in the preceding section.

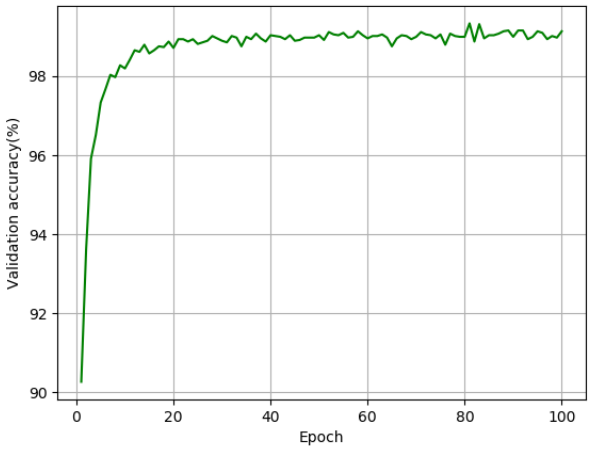
9 Test accuracy

The following table contains the average and standard deviation (std) of the values of test accuracy obtained from the different simulations.

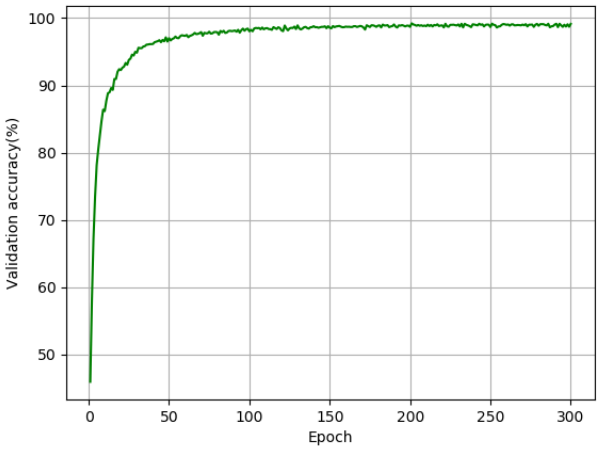
Network / Dataset	CNN baseline		AlexNet		CapsNet	
-	Average (%)	Std (%)	Average (%)	Std (%)	Average (%)	Std (%)
MNIST	99.01	0.03	99.10	0.04	99.31	0.01
CIFAR-10	73.39	0.11	72.24	1.79	72.42	0.21
SmallNORB	76.04	0.32	66.95	1.98	88.69	0.16
Retinographies	94.19	2.82	93.27	3.23	94.08	2.60

10 Validation accuracy along training epochs

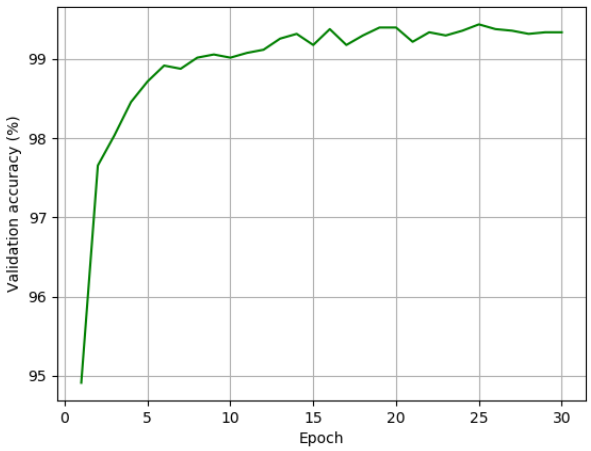
Here it is presented the evolution of the values of accuracy obtained during validation against the number of epochs of training. These charts provide an idea of how the weights are being adjusted so that the model becomes more and more accurate until the mean stagnates, this is called convergence of the optimization process.



(a) Baseline.

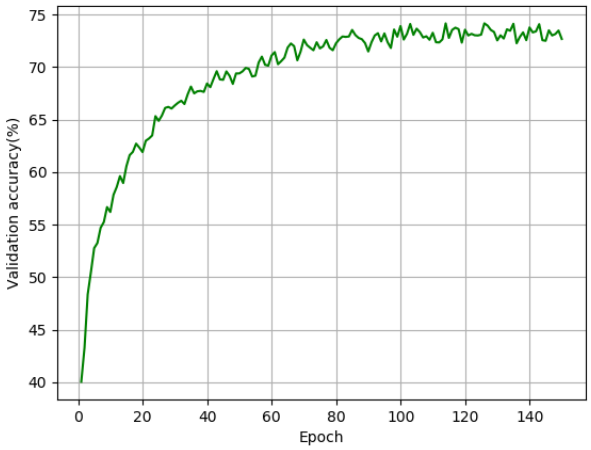


(b) AlexNet.

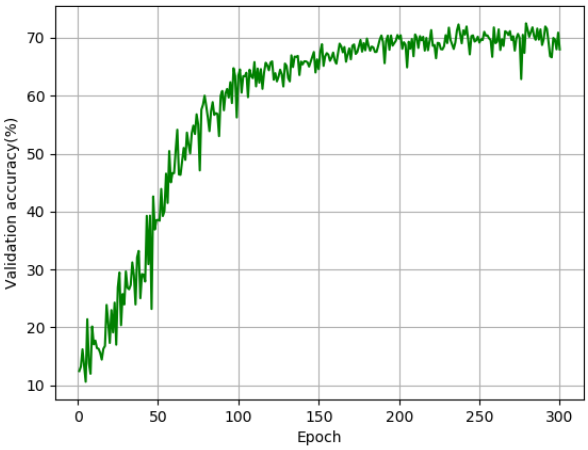


(c) CapsNet.

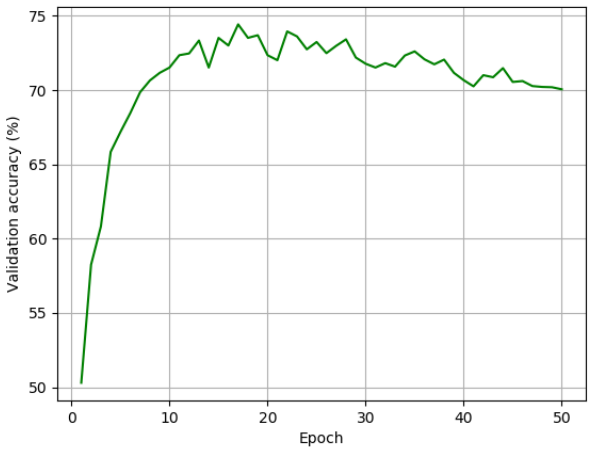
Figure 19: Validation accuracy on MNIST.



(a) Baseline.

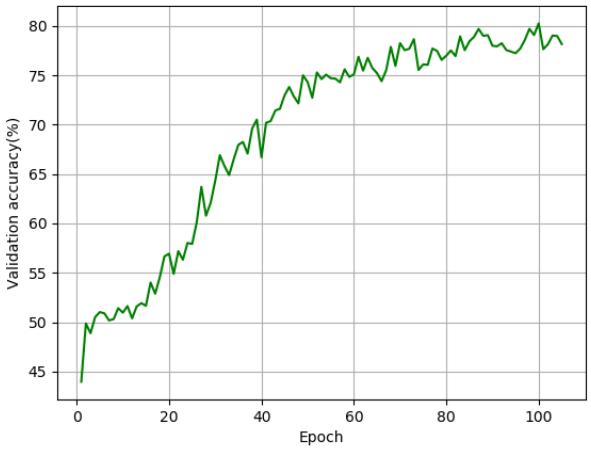


(b) AlexNet.

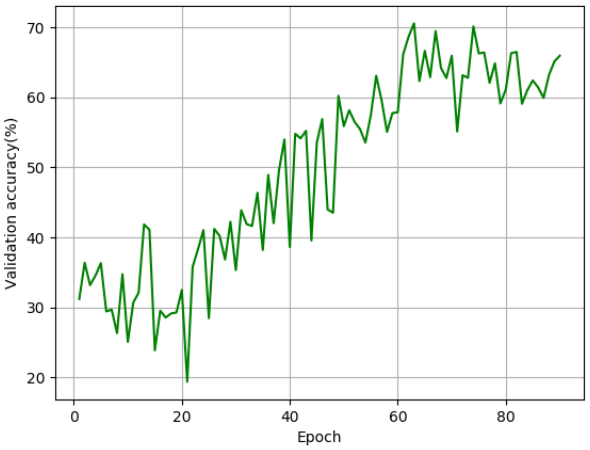


(c) CapsNet.

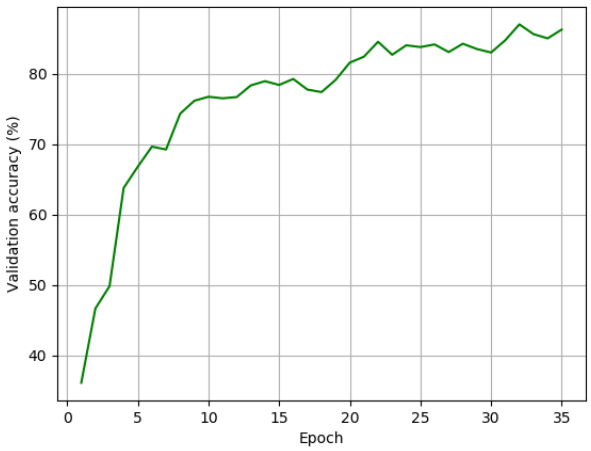
Figure 20: Validation accuracy on CIFAR-10.



(a) Baseline.

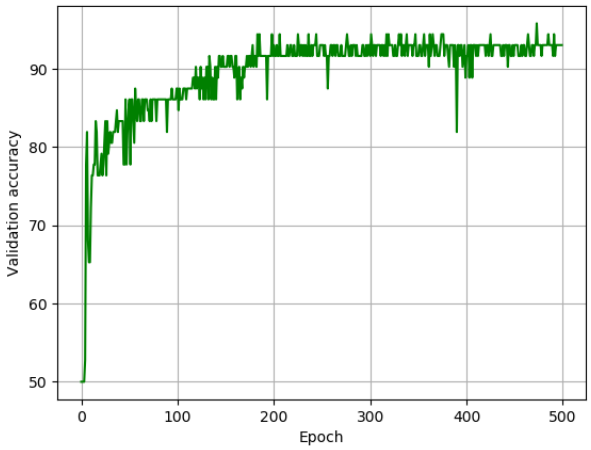


(b) AlexNet.

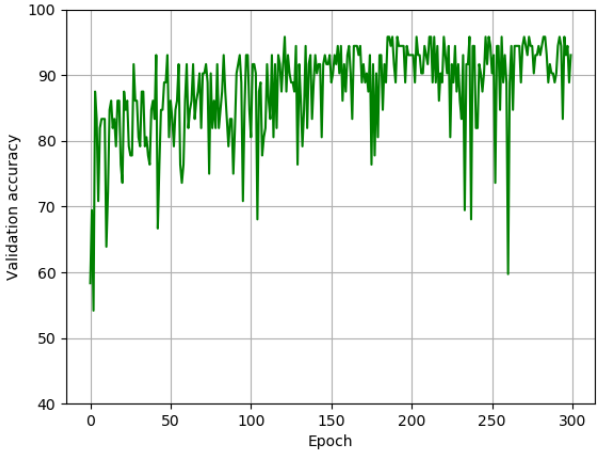


(c) CapsNet.

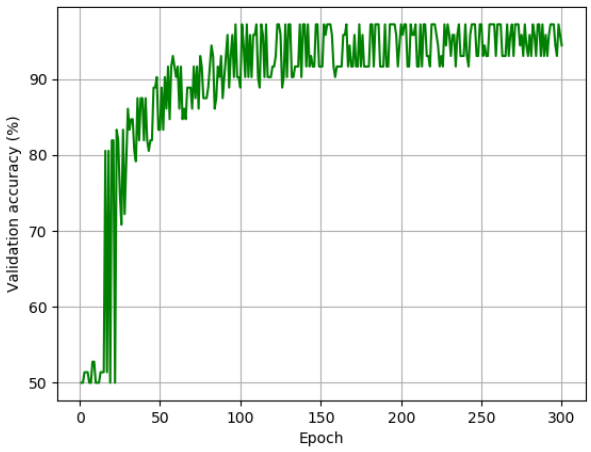
Figure 21: Validation accuracy on SmallNORB.



(a) Baseline.



(b) AlexNet.



(c) CapsNet.

Figure 22: Validation accuracy on retinographies.

11 Margin and reconstruction loss evolution with CapsNet

This subsection presents plots of the margin and reconstruction losses in order to be able to see the evolution of both and understand how reconstruction is affecting the optimization process. The total loss is not shown, though with the factor 0.0005 multiplying, as Hinton's paper suggests, its value is barely the margin loss.

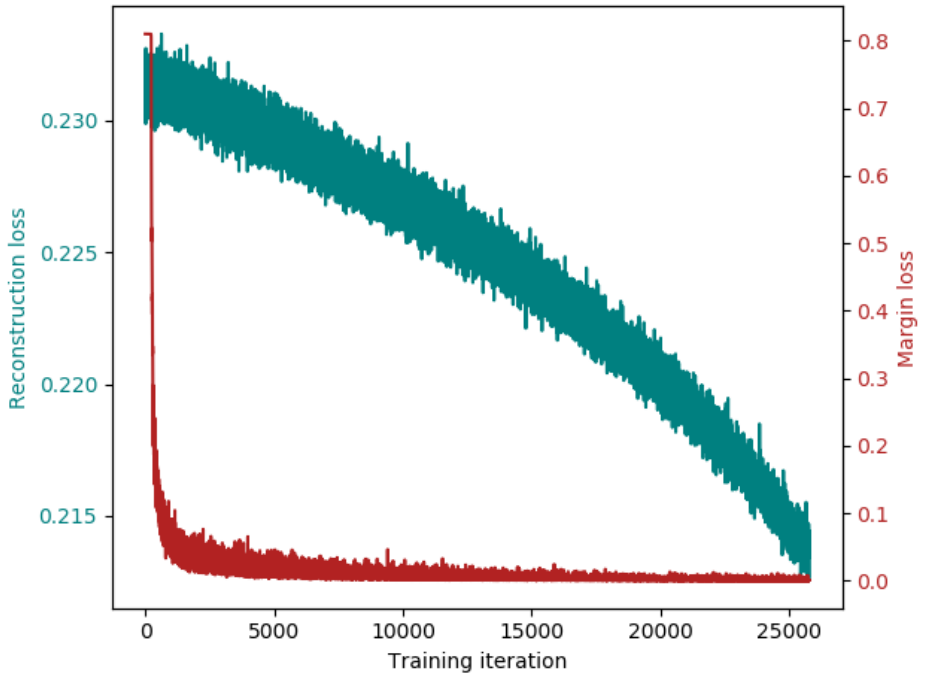


Figure 23: Margin and reconstruction loss on MNIST.

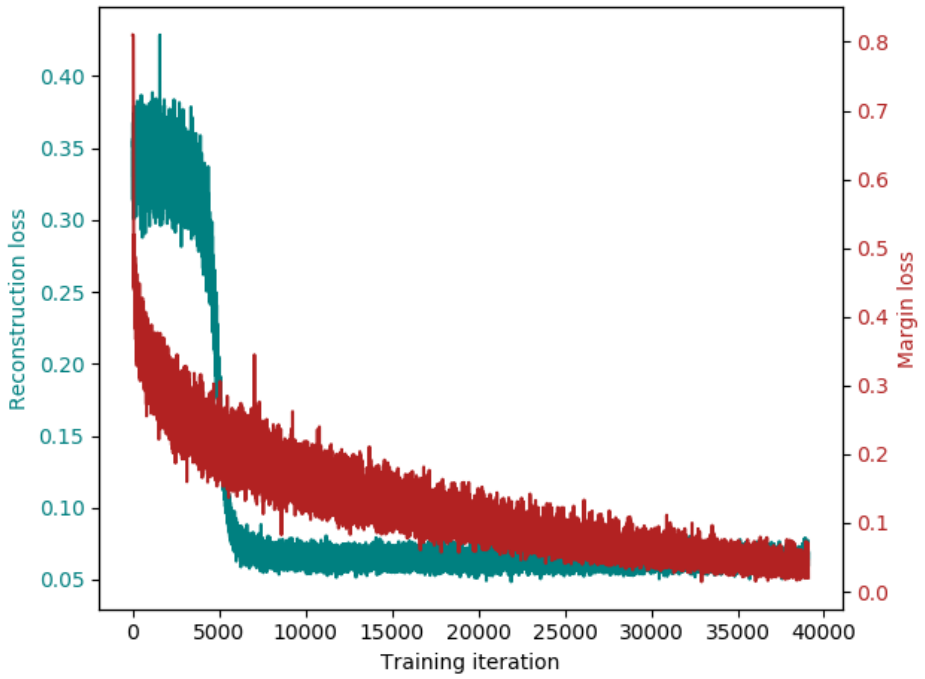


Figure 24: Margin and reconstruction loss on CIFAR-10.

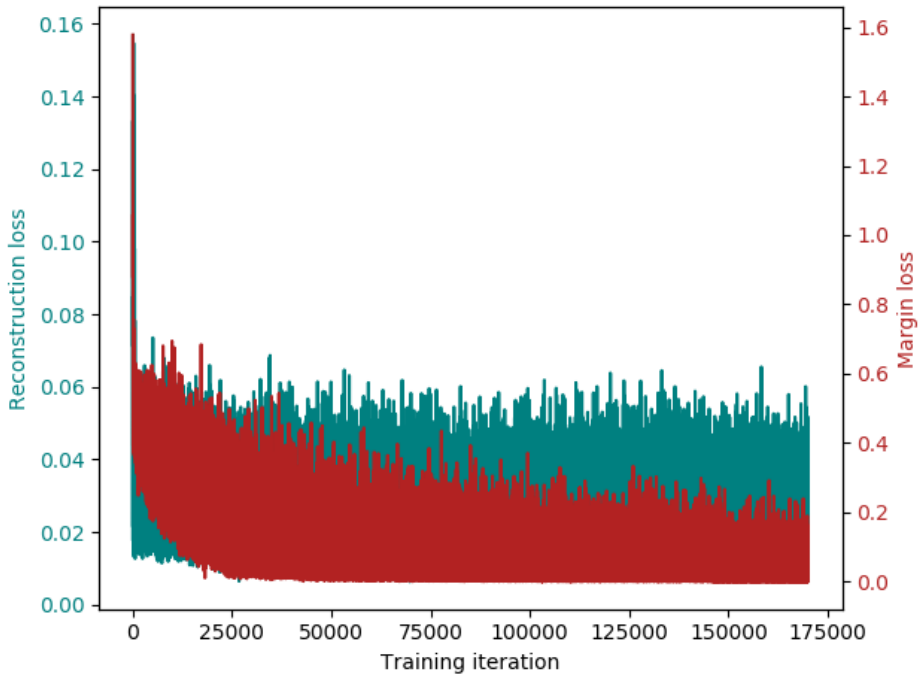


Figure 25: Margin and reconstruction loss on SmallNORB.

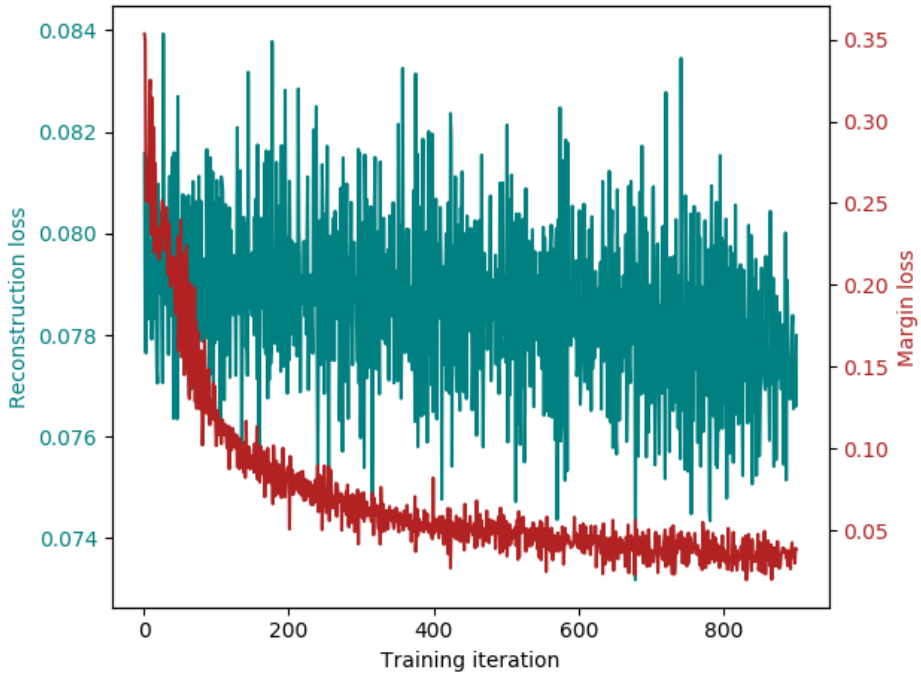


Figure 26: Margin and reconstruction loss on retinographies.

12 Reconstructed images

Here it can be observed the output of the CapsNet decoder on the different datasets. This might help the comprehension of reconstruction stage and whether it is achieving at least a blurred outline so that there silhouettes seem understandable for human eye. Since somehow the goal is to model human vision and reasoning with artificial machines, obtaining images that are easily classified by human beings shall be considered as a success.

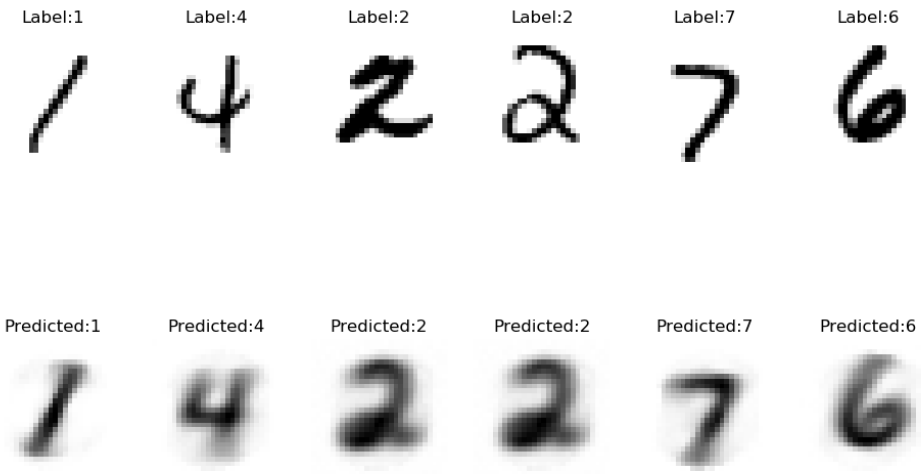


Figure 27: Original input images and their reconstruction on MNIST.

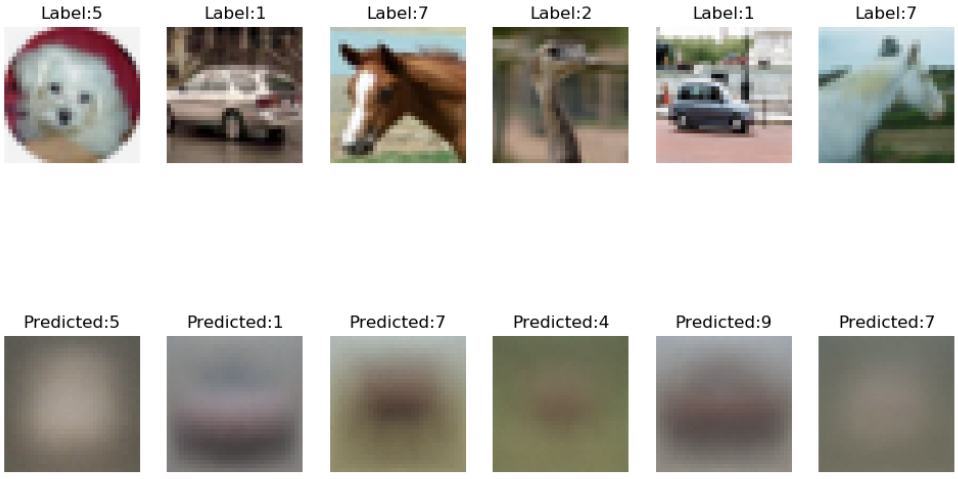


Figure 28: Original input images and their reconstruction on CIFAR-10.

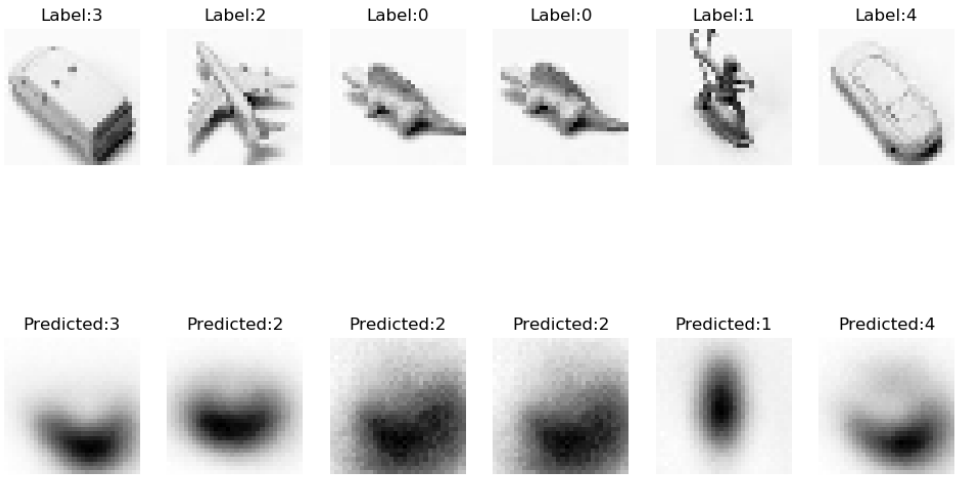


Figure 29: Original input images and their reconstruction on SmallNORB.

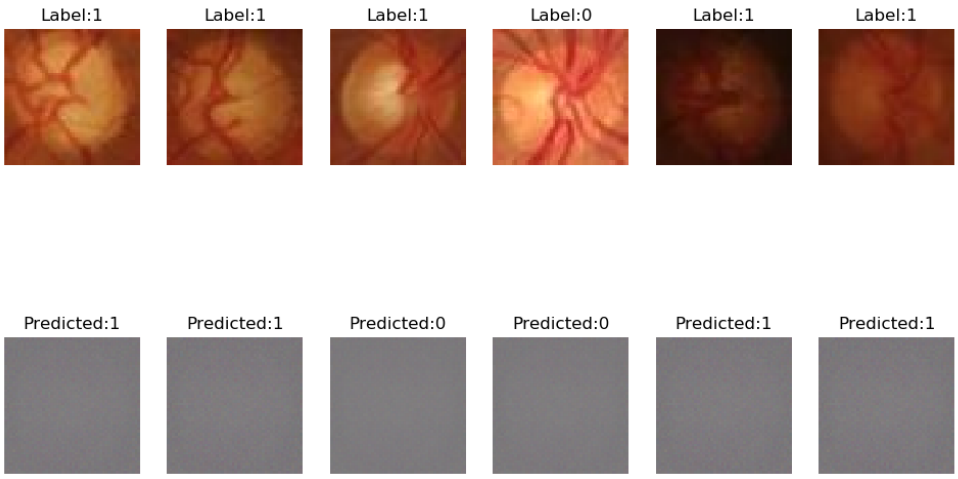


Figure 30: Original input images and their reconstruction on retinographies.

Part V

Discussion

The starting point is MNIST, all the architectures that have been employed reach a high performance on it. This is not a huge surprise, assuming that this dataset is considered a "toy problem" used to present newborn structures to the scientific community. However, the staging of CapsNet should not be underestimated: it has achieved an outstanding test accuracy that leaves its competitors behind. Regarding the comparison between both convolutional networks, the reduced version of AlexNet overtakes its peer. The reason why CapsNet does not reach the 99.61% revealed on Hinton's paper [1] (without rotation and scaling) might be the lack of data augmentation techniques employed to train the network on the cited paper. Validation accuracy plots showcase that all the networks have come to converge, so the test accuracy difficultly could be improved unless hyperparameters are readjusted or data augmentation is used. About the reconstructed images, it can be observed that they are a bit blurred but legible, which was actually the aim. This shows that capsules have managed to acquire an internal representation of the digits.

Now, it is time for a more difficult challenge: the classification and reconstruction of color images on CIFAR-10. This dataset is sort of headache for all the networks that have been employed, since the accuracy at this time is quite humble for all of them. However, some conclusions may be extracted from here. First of all, it is clear that our version of AlexNet has lagged behind this time, because it has not even been able to beat the baseline; probably the answer could be in the hyperparameters. In particular, the probability to keep units that defines the amount of dropout applied may be increased, since the system is providing improvable validation results. This is a symptom of underfitting while dropout is designed to reduce the phenomenon of overfitting, so it might be hampering the optimization process. Focusing now on the two winners, the CNN baseline provides the best results closely chased by CapsNet. The variability of backgrounds has possibly made the performance of CapsNet be reduced, as this kind of structures does not discard information with pooling; instead, it tends to understand everything in the image. On its behalf, the baseline applies a humble level of reasoning due to its limited number of parameters and layers. Sometimes it is better to apply Occam's razor and prioritize simple models. On the other hand, Hinton's paper [1] shows a far better result of "capsules" on CIFAR-10: 89.40% of accuracy. Nevertheless, it is specified there that they assembled 7 models with patches of 24×24 to achieve that value and included an additional label known as "none-of-the-above" that would have been helpful, so it demands augmentation of the architecture with assembled models and the data with patches. Reconstructions in this case are much more blurred, in some cases we can even figure out what kind of object or animal is present in the image; however, it is quite difficult in general. Therefore, this decoder has not been able to reconstruct satisfactorily, although it must be taken into account that obtaining a good reconstruction of this dataset is infinitely more complicated than MNIST digits.

The following dataset is an opportunity for CapsNet to emphasize its characteristics. The way in which SmallNORB samples are introduced into the networks fosters those structures that are able to identify an object from different points of view even though they have not seen it from every angle. Back to theory, CapsNets are designed with this specific purpose, as opposed to convolutional networks that need to be trained with all possible points of view. This might be observed, for instance, at the evolution of validation accuracy. Figure 21 reveals that CapsNet abruptly rises in accuracy while both CNNs showcase a more progressive learning as they are being fed with new points of view. At the end, CapsNet unreservedly beats its rivals with a large difference with respect to the runner-up in test accuracy. The battle between both convolutional networks has ended with a new defeat for our reduced version of AlexNet. The latter is thought to show strength in challenges in which it is required to understand an amalgam of different points of view, because of its depth allows a higher level of reasoning. However, more layers and parameters demand a finer adjustment of the hyperparameters, both those related to the structure of the network and the optimization process. Since computer resources are quite limited in this thesis, and also the main focus is on early detection of glaucoma, we have not performed the proper adjustment that this network requires and deserves. Directing the focus towards reconstructions now, it can be observed that the decoder is not too effective here. In spite of the shapes change according to the input object, like the stretched shape for "human figures", it does not provide reconstructions that might be classified by human beings.

Finally, we will discuss the results of glaucoma detection from retinographies. The podium in test accuracy is distributed as follows: CNN baseline closely followed by CapsNet and the reduced version of AlexNet below them. This dataset might not be a favorable battlefield for CapsNet as there is not a clear geometry to encode with capsules. This fact is demonstrated by the reconstructed images of retinographies in figure 30: all of them are constant gray pictures despite the predicted labels are mostly right. This shows that they are not able to reconstruct them and the optimization process was fallen to a "comfortable" state in which all the outputs of the decoder are set to the mean value of intensity. Figure 26 also reveals that reconstruction loss does not improve along training. This leads us to the conclusion that we would need a larger network capable of understanding the more complex geometry of retinographies; however, this would mean much more parameters to train, which demands more samples in order to avoid underfitting. Capsules see their power reduced by decreasing their dimensionality, although this is a necessary step so as to reach the best test accuracy obtained with this kind of structures. We have tried with different configurations varying the dimensions of capsules, the number of maps and the amount of units in the decoder. Nevertheless, we have been limited by a modest computing capacity, so we have not been able to perform a complete hyperparameter adjustment study that might bring a CapsNet model capable of surpassing the basic CNN architecture. Again, the baseline has reached a higher accuracy than their competitors. Apart from the hyperparameter adjustment that has already been mentioned, it might be also a consequence of that tendency we discussed with CIFAR-10 of CapsNet to account for everything in the image. Retinographies are really changing backgrounds; thus, CapsNet is striving to manage all the information provided by each pixel. This property that may be an asset in certain cases, may end up being a ballast when the system is not capable of seeing the relationships among the parts.

References

[1] S. Sabour, N. Frost and G. E. Hinton,"Dynamic routing between capsules". In: *Advances in Neural Information Processing Systems*. 2017, pp. 3856-3866.

[2] G. E. Hinton, A. Krizhevsky and S. D. Wang, "Transforming auto-encoders". In: *International Conference on Artificial Neural Networks*. 2011, pp. 22-51.

[3] Y. Lecun et al., "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE 86.11*. 1998, pp. 2278-2324.

[4] Alex Krizhevsky, "Learning Multiple Layers from Tiny Images". 2009.

[5] E.J. Carmona, M. Rincón, J. García-Feijoo and J. M. Martínez-de-la-Casa, "Identification of the optic nerve head with genetic algorithms". In *Artificial Intelligence in Medicine*. 2008, vol. 43(3), pp. 243-259.

[6] Tensorflow.
`https://github.com/tensorflow/tensorflow`

[7] A. Krizhevsky, I. Sutskever and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks". In *Advances in Neural Information Processing Systems 25*. 2012, pp. 1097-1105.

[8] Y. Lecun, F.J. Huang and L. Bottou, "Learning Methods for Generic Object Recognition with Invariance to Pose and Lighting". In: *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*. 2004.

[9] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever and R. Ruslan, "Improving neural networks by preventing co-adaptation of feature detectors". 2012.

[10] Kingma, Diederik and Ba, Jimmy, "Adam: A Method for Stochastic Optimization". In *International Conference on Learning Representations*. 2014.

[11] Haykin, Simon. "Neural Networks and Learning Machines". Hamilton, Ontario, Canada. 2009.

[12] Sebastian Raschka. "Python Machine Learning". Birmingham, UK. 2015.

[13] Ian Goodfellow, Yoshua Bengio and Aaron Courville. "Deep Learning". MIT Press, 2016.
`http://www.deeplearningbook.org`

[14] Kevin P. Murphy. "Machine Learning". The MIT Press, Cambridge, Massachusetts. 2012.

[15] Aurélien Geron.
`https://github.com/ageron`

[16] Jeremy Ellis.
`https://github.com/hpssjellis`

[17] Hvass-Labs.
`https://github.com/Hvass-Labs`

[18] Johannes Mohr `http://info.ni.tu-berlin.de/photodb/`