University of Liège

Faculty of Science

Master thesis conducted in the research unit Topological Orders in Matter (TOM)

Topological phases identification using artificial intelligence

Author:
Xavier BOSCH

Supervisor:
Bertrand Dupé
Advisor:
Sebastian Meyer

August 20, 2024

Abstract

This master thesis presents the development and application of neural networks for the recognition of topological structures, specifically magnetic skyrmions and anti-skyrmions. Topological phases of matter, characterized by their geometric properties rather than local order parameters, have significant implications in spintronics as they have special transport properties. For example, skyrmions can be moved by electrical currents and serve as potential information carriers. These potential applications require a precise analysis of their dynamical properties. However, traditional methods to locate and analyse topological structures dynamics are involving complex algorithm and are only valid at 0 Kelvin. To overcome these limitations, this research leverages deep learning techniques, including Feedforward Neural Networks, Convolutional Neural Networks, and UNet architectures. Databases of synthetic magnetic structures were generated and labelled, serving as the foundation for training these networks. The results demonstrate the effectiveness of these models in accurately classifying topological structures with over 97% accuracy on unseen data. Despite the success, challenges such as bias in the training datasets and model overfitting were identified, suggesting the need for further refinement. This work lays the groundwork for future research on the dynamic behaviour of skyrmions and their potential applications in next-generation spintronic devices.

Résumé

Cette thèse de master présente le développement et l'application de réseaux neuronaux pour la reconnaissance de structures topologiques, en particulier les skyrmions et anti-skyrmions magnétiques. Les phases topologiques de la matière, caractérisées par leurs propriétés géométriques plutôt que par des paramètres d'ordre locaux, ont des implications significatives en spintronique en raison de leurs propriétés de transport particulières. Par exemple, les skyrmions peuvent être déplacés par des courants électriques et servir de porteurs d'informations potentiels. Ces applications potentielles nécessitent une analyse précise de leurs propriétés dynamiques. Cependant, les méthodes traditionnelles pour localiser et analyser la dynamique des structures topologiques impliquent des algorithmes complexes et ne sont valides qu'à 0 Kelvin. Pour surmonter ces limitations, cette recherche exploite des techniques d'apprentissage profond, y compris les réseaux neuronaux Feed-forward, les réseaux de neurones convolutionnels et les architectures UNet. Des bases de données de structures magnétiques synthétiques ont été générées et étiquetées, servant de base à l'entraînement de ces réseaux. Les résultats démontrent l'efficacité de ces modèles pour classifier avec précision les structures topologiques avec une précision supérieure à 97% sur des données inédites. Malgré ce succès, des défis tels que les biais dans les ensembles de données d'entraînement et le surapprentissage du modèle ont été identifiés, suggérant la nécessité d'affinements supplémentaires. Ce travail jette les bases de futures recherches sur le comportement dynamique des skyrmions et leurs applications potentielles dans les dispositifs spintroniques de nouvelle génération.

Acknowledgement

First of all, my sincere thanks go to Bertrand Dupé for shaping a master thesis that aligned with my aspirations. His unwavering availability, coupled with some light-hearted mutual teasing and the complete freedom he offered in approaching the project, created a stimulating, enjoyable and supportive work environment. 'What a bully!'

Next, I must express my gratitude to Sebastian Mayer for his constant support and expertise throughout the year. His guidance allowed me to stay confident, improve my plots, and ultimately reach the target.

A big thank you also goes to everyone in the solid-state physics group and my classmates Axel and Cyril, who were always ready to lend a helping hand whenever possible.

I am also deeply appreciative of Pr. Ngoc Duy Nguyen, Pr. Geoffroy Lumay and Dr. François Damanet for unhesitatingly agreeing to form my reading committee. My gratitude also extends to Joséphine Van Hulle, who introduced me to Dr. Pauline Nyssen and Dr. Julie Hottechamps, both of whom had agreed to join the committee. Their enthusiasm, kindness, and encouraging words should be highlighted.

Finally, heartfelt thanks to all those close to me; you know who you are. Your support has meant the world to me.

Contents

List of Figures

List of Tables

Nomenclature

AI: Artificial Intelligence
DNN: Deep Neural Network
IoT: Internet of Things
MLP: MultiLayer Perceptron
NN: Neural Network
RNN: Regular Neural Network
RU: Research Unit
TL: Training Loss
VL: Validation Loss

Preamble

The manuscript is divided as follows: In Chapter 1, the reader will find the state of the art, an example of topological structure recognition algorithm without artificial intelligence (AI) done in TOM research unit and a brief summary of first research conducted in order to get acquainted with AI for this interdisciplinary project. Chapter 2 is devoted to the topological structure generation and the labelling. For this part, a script was created to automate those processes. In the next chapters, NNs are constructed going from simpler to more complicated ones. Chapter 3 describes how a regular deep NN works and the results obtained with the one created for the project. In Chapter 4, a second NN is created in the form of a convolutional NN (CNN). A description is given, then the results are analysed.

We will then conclude the report with a summary of the results and make an opening for further research with the development of a UNet. This network does not show the same level of accuracy as those in the previous chapters yet, but it is a significant step forward toward the analysis of skyrmion dynamics.

I wish you a pleasant reading.

Chapter 1

Introduction

1.1 Magnetic Skyrmions

Topological phases of matter are states of matter that cannot be described by a local order parameter [1]. Instead, they are characterised by their geometrical properties, i.e. topological integers, -2,-1,1,2, etc. In magnetism, these phases are formed by magnetic moments. On a large scale, magnetic moments can build a variety of structures, such as a ferromagnet (Fig. 1.1 (a)), domain walls (Fig. 1.1 (b)), skyrmions (Fig. 1.1 (c)), or anti-skyrmions (Fig. 1.1 (d)). While the former two are called topologically trivial, magnetic skyrmions are topologically protected, defined by the non-zero topological charge Q that prevents them from spontaneously collapse into a trivial state. Due to this property, they are particularly interesting for their potential applications in spintronics [2, 3].

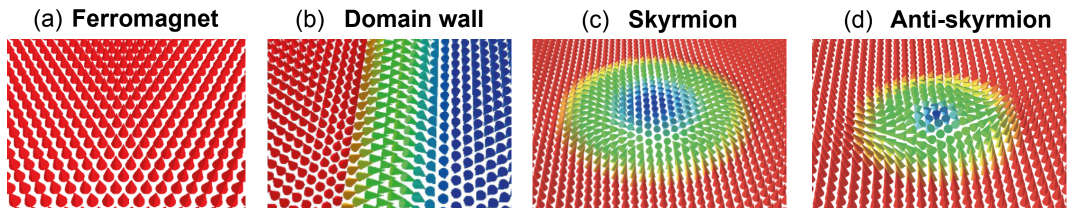


Figure 1.1: Examples of magnetic structures: (a) Ferromagnet: all spins (red cones) point in the same direction - here, up. (b) Domain wall: two oppositely magnetised ferromagnetic domains are separated by a region, where the direction of spins is gradually changing (here, from up - red to down - blue) (c) Skyrmion: round shaped structure in which the spins tilt from up in the ferromagnetic surrounding to down in the core. Note that for skyrmions, the sense of rotation for all spins is equal. (d) Anti-skyrmion: Similar to the skyrmion structure in (c), but with two senses of rotation. Figure adapted from [4].

One major application of skyrmions would be to use them as information carriers in spintronics, i.e. as bits. Implementing spintronics based on skyrmions could lead to several breakthroughs. The storage density would be greatly enhanced partly due to the fact that magnetic skyrmions can have sizes under 10 nm, while domain walls reach a limit around 30 nm. Moreover, skyrmions can be moved with very low current density, less than $10^6\mathrm{A/m}^2$. By comparison, the current density required to move domain walls is estimated to be above $10^{12}\mathrm{A/m}^2$ in racetrack memory [5]. These improvements could counter the slowdown in Moore's law and reduce the electrical consumption of all electronic and spintronic devices.

One of the challenges arising with skyrmion-based spintronics devices is identifying the structures. Indeed, the topological charge is a non-local quantity. Two opposite topological structures with different topological charges will cancel each other out when evaluating the topological charge of the sample. Thus, if a skyrmion and its opposite, an anti-skyrmion, are present on the same sample, it is not straightforward to find an effective method to determine the number and the type of structures. Such a method is a key for research on creation, destruction and topological protection of skyrmion as it would allow systematic detailed simulation analysis in complex environment [6].

1.2 Topological structure recognition without AI

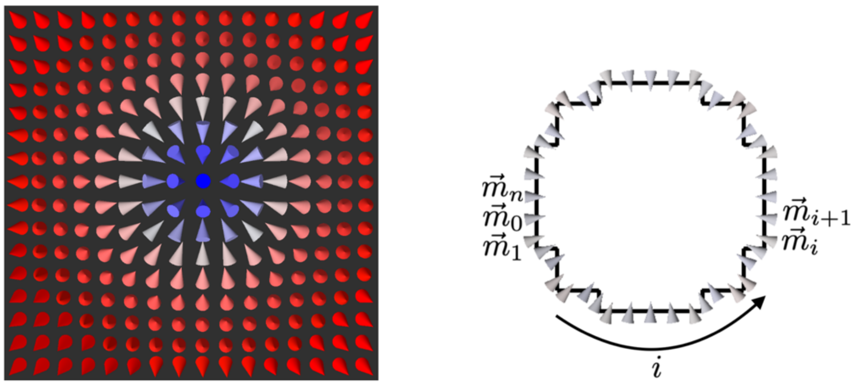
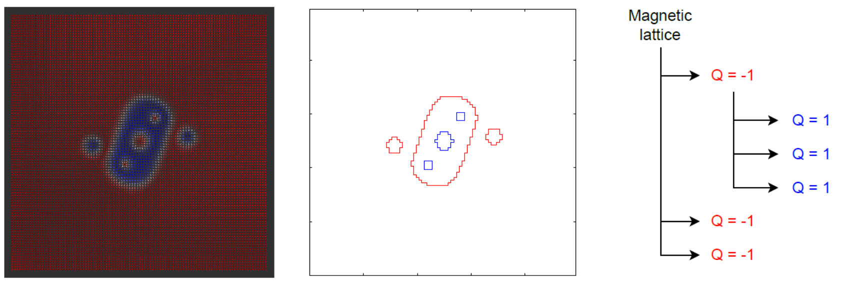


Figure 1.2: Figure from [7]. The contour is defined as the ensemble of magnetic moments which are pointing in plane. Then the inner part is defined as the ensemble of magnetic moments pointing down and the outter part as the ensemble of magnetic moments points up. However, this very simple method is only effective for a single type of topological textures at 0 Kelvin.



(a) Spin lattice

(b) Skyrmion contours: blue curves have a charge of 1 an the red ones a charge of -1

(c) Tree structure representation

Figure 1.3: Figure from [7]. Example of complex contours within another and corresponding tree-like graph that is compared to a database to determine the structure type.

One possible method to recognise topological structure is to use an algorithm such as the one developed by Maxime Mignolet [7]. This algorithm works in a few steps: It defines the contour of structures based on the sign change of the magnetisation along the z-component as in Figure 1.2. Then it computes the topological charge of the region defined by the contour. After that, it determines if structures are embedded into others. From this, it builds a tree-like graph which is compared to a pre-generated database to identify the structure. An example of these steps is shown in Figure 1.3

Although this algorithm is well designed, two requirements limit its use to advanced purposes: First, it needs a ferromagnetic background (so no thermal disorder should be present). Second, the magnetic state should be a local minimum of energy (a metastable state) at least. This last requirement does not allow for the dynamical identification of skyrmions.

1.3 Purpose, goals and contents using AI

To counter the problems faced with the previous algorithm, a possible solution could be Artificial Intelligence (AI) as it has rapidly evolved into a vast and dynamic field in different technologies and applications. Recently, efforts have been made to use AI for the detection of different magnetic structures, ranging from simple supervised algorithms [8], over ML approaches to identify structures in experimental measurements [9] to classifying magnetic textures based on Monte-Carlo simulations [10]. Here, our goal is to specifically pay attention to creating a starting point of a deep learning algorithm to lay the groundwork for future more complex magnetic structures recognition and their respective researches.

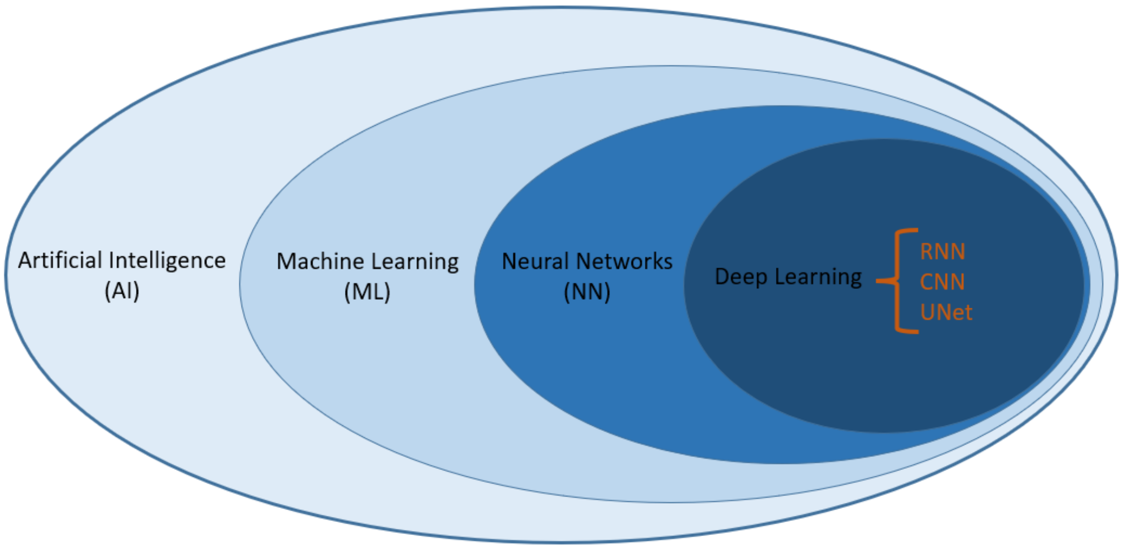


Figure 1.4: Venn diagram showing relations between AI theories. The NNs created during the project are placed in orange.

To begin with, a definition of AI is welcome. Peter Norvig and Stuart Russell define the field of AI as what *is concerned with not just understanding but also building intelligent entities — machines that can compute how to act effectively and safely in a wide variety of novel situations* [11]. AI itself is more complex to define. There is a wide variety of intelligences. Hence, there is also a wide variety of definitions. However, if we are satisfied with the definitions of the field, we have a definition of the intent without looking at the means to achieve it. The means can be classified (Figure 1.4) into two groups depending on whether the machine is end-to-end programmed or it can learn. In the first case, the best known field is the internet of things (IoT) which is now almost always coupled with scripts to improve the use of the objects. The second case is called machine learning (ML). It is the domain in which the machine is able to become better to realise some tasks. A wide variety of algorithms form the domain of ML (e.g. Random forest, decision trees, support vector machines...). For the project which is skyrmion pattern recognition with AI, we decided to focus on neural networks (NNs) to be able to learn directly on physical quantities (magnetic moments[1]) instead of features computed by other methods. As those quantities are represented by lots of components for a small sample surface, we created more advanced NNs which can be classified in the deep learning field, a sub-part of the NN field. We successfully created two deep NNs to classify single magnetic structures: a regular neural network (RNN) and a convolutional neural network (CNN), then we started to take a step further with a so-called UNet.

[1]Local magnetic densities would also have been possible.

To parameterise these NNs, we carried out several steps. First, a database containing skyrmions, anti-skyrmions and empty examples must be created. Then, to do supervised learning, labels are required for each corresponding example. Once the labelled database has been created, the next step is to do the link between the simulations and the neural networks. From that point, the project is focused on machine learning in the hope of good results on structure recognition with the underlying idea of creating increasingly complex algorithms to move towards dynamics with multiple magnetic structures.

From this first acquired knowledge, two main goals were defined:

- Create a first skeleton for the AI in the TOM research unit (RU)
- Create and use NN to recognise topological structures

We are using the program Matjes, which is an atomistic simulation tool developped in the TOM group. Matjes is made to simulate topological textures in 1D, 2D and 3D based on the generalised Heisenberg model [12]. Using Matjes, the first goal consists in automating the creation of databases for AI. Then, this database is linked with the NNs in an efficient manner. The challenge is now to recognise the different textures and to improve the level of recognition accuracy. This level of recognition is unpredictable and the material resources needed are unknown.

Chapter 2

Database generation and labelling

2.1 Topological structure generation

In spintronics and magnetism, topological textures are defined by the symmetry of solution of the differential equation as given by Bogdanov et al [13, 14]:

$$E = \int \frac{1}{2} A \left(\frac{\partial \mathbf{m}}{\partial \mathrm{x}} \right)^2 - \frac{1}{2} K \mathbf{m}_z^2 - \mu_0 \mathbf{m} \cdot \mathbf{H} - \frac{1}{2} \mathbf{m} \cdot \mathbf{H}_\mathrm{m} + H_\mathrm{DM} + \epsilon_\mathrm{DM} dV \tag{2.1}$$

where $\mathbf{m}$ is the unitary magnetic vector, A is the exchange energy density, K is the anisotropy, $\mathbf{H}_\mathrm{m}$ is the magnetostatic energy and ϵ_DM is the energy density of the Dzyaloshinskii-Moriya interaction. The general solution of this equation is a three-dimensional magnetic texture which can be expressed in spherical coordinate [15] as:

$$\mathbf{m} = (\cos \Phi (\phi) \sin \Theta (\mathbf{r}), \sin \Phi (\phi) \sin \Theta (\mathbf{r}), \cos \Theta (\mathbf{r})) \tag{2.2}$$

where Θ and Φ are the polar angles and ϕ depends on the symmetry of the DM interaction. Topological textures are then characterised, based on 3 order parameters. The first one is the topological charge or skyrmion number

$$Q = \frac{1}{4\pi} \int_S \mathbf{m}(x, y) \cdot \left(\frac{\partial \mathbf{m}}{\partial x} \times \frac{\partial \mathbf{m}}{\partial y} \right) dS \tag{2.3}$$

where $\mathbf{m}(x, y)$ is the unitary magnetisation vector field and S is the integration surface. This simple order parameter allows a simple identification of the topological texture. By convention, $Q = 1$ is called a skyrmion (SK), $Q = -1$ is an antiskyrmion (ASK) and $Q = 0.5$ is a vortex or a meron. However, this order parameter lacks the information on the chirality for example.

The chirality reveals the sense of rotation for the skyrmion. Both left and right rotating skyrmions will have a charge of 1 although being of different symmetry. To evaluate this quantity, the helicity is usually computed. The helicity can be identified by the Φ angle:

$$\Phi = h\phi + \gamma \tag{2.4}$$

where m is the helicity and γ is a phase factor which corresponds to the direction of the reference magnetisation. For example, $(h = 1, \gamma = \pi, Q = 1)$ corresponds to the typical skyrmions that is seen everywhere, as shown in Figure 2.1. Other simple topological textures are also represented in Figure 2.1 for comparison.

In Matjes [12], the general solution of this equation was coded to obtain all possible magnetic textures. It takes the form:

$$\mathbf{m} = \left(\cos\Phi\left(h_x\phi + \gamma_x\right)\sin\Theta\left(\mathbf{r}\right), \sin\Phi\left(h_y\phi + \gamma_y\right)\sin\Theta\left(\mathbf{r}\right), \cos\Theta\left(\mathbf{r}\right)\right) \tag{2.5}$$

where h and γ are defined for x and y separately.

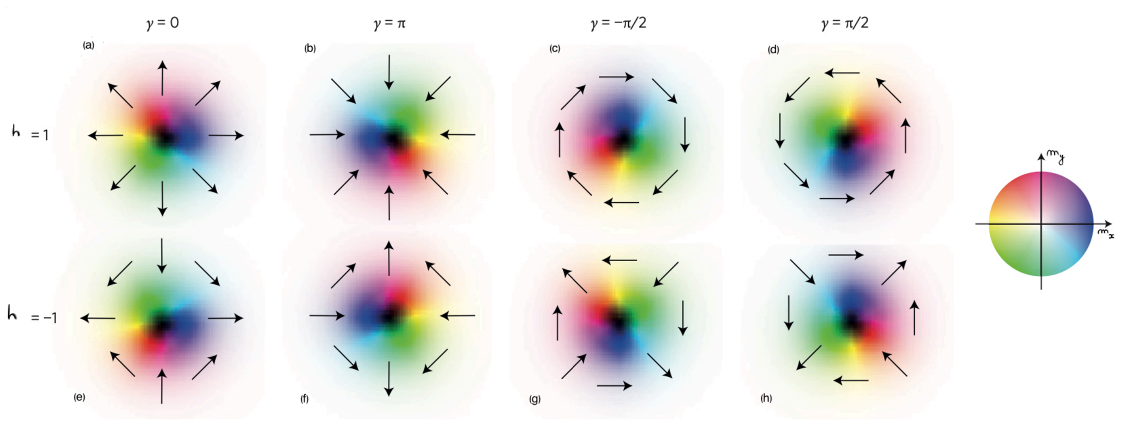


Figure 2.1: "*Skyrmion structures with varying h and γ. The arrows indicate the direction of the in-plane spin component, and the brightness indicates the normal component to the plane, with white denoting the up direction and black the down direction. All the structures of anti-skyrmions (h = -1) are equivalent in rotation in the* x − y *plane*" Figure and text adapted from [15].

2.2 Labelling

To generate the database, we considered a restriction of the above case. For simplicity, we have taken $\gamma = 0$ and change helicity along the x and y directions. In the following, we will label all the cases with (h_x, h_y). In order to get the labelled samples, the helicities are fixed. Then, the other parameters are changed. Changing sizes and/or positions of the structures does not impact the topological charge. However, it can greatly change the final representation if the skyrmion is located at the edge of the supercell. Then the skyrmion will appear cut in half due to periodic boundary conditions. Hence, computing once the topological charge for different helicities at fixed size and position allows to predict the structure that will be generated for all the sizes and all the positions. Restricting the labelling to first order SK and ASK, we obtain:

- $(-1, -1) \rightarrow$ SK
- $(-1, \ \ 1) \rightarrow$ ASK
- $(\ \ 1, -1) \rightarrow$ SK
- $(\ \ 1, \ \ 1) \rightarrow$ ASK
- Otherwise $\rightarrow$ None of the listed

To be able to generate as large as possible databases to allow the training with labelled samples, a library was created. This library creates numbered directories corresponding to the ID of the sample. It also generates and puts the input files in the directories. Then, it executes Matjes in each folder and finally, saves the parameters and the structure label in a single file classified by IDs. The structure types are encoded as numbers: $0 = none$, $1 = SK$, $2 = ASK$. Proceeding like this allows easy modifications in the future to add structures, change the labels and facilitates data import to neural networks. However, this procedure generates around tens of Gb of data for the recognition of only two topological textures.

Chapter 3

Feedforward Neural Network

This chapter is devoted to feedforward Neural Networks which are fully connected. They are also called multi-layer perceptrons (MLP). MLP are a generalisation of the perceptron invented by Warren S. McCulloch and Walter Pitts [16]. This single perceptron model is only able to do binary classification on linearly separable data. Hence, non-linear multi-class analyses for topological structure recognition required a more advanced model.

This chapter will give the basics needed to understand how NNs work. Even if very complicated NNs structures are now used in our every day life through GPTs, image search engines, automatic pilots in cars and so on, a simple RNN is a powerful tool allowing good results for the task discussed in this report. It will also allow to understand more complicated models discussed in the following chapters.

3.1 Creation and operation of a deep NN

3.1.1 Neural Network Structure

The structure of a NN consists in interconnected layers. The first layer is the input layer. This layer is the one that will receive the data and must be set accordingly. The last one is the output layer. It is specifically designed for a purpose. Between those two layers there is a set of layers called hidden layers (see Figure 3.1 (a)). That's the place where the non-linearities can be dealt with. In the case of a single hidden layer, the network is called a NN. If there are at least two hidden layers, the network is called a deep neural network. The input and output layers contain a number of neurons that is chosen to map the input and the desired output. The hidden layers must contain enough neurons to fit a function representing the problem of interest.

The connections between layers are not random. In our case of a fully connected NN, the layers follow one another and there is no direct connection from two non-adjacent layers. Let's take a neuron in the first hidden layer as in Figure 3.1 (b). This neuron, sometimes called a node, is connected to all neurons in the previous layer. Each connection has a specific weight which defines the impact of neurons from the first layer onto the one selected in the second layer. Some connections may not play a role if the weight is 0. The connections going to the selected neuron give the inputs from which an output is calculated (Figure 3.1 (b). Then, (Figure 3.1 (c)) this output is given to every neuron in the next layer in the same manner. In repeating the described connection process to all neurons in the network, we obtain a fully connected NN as represented in Figure 3.1 (d).

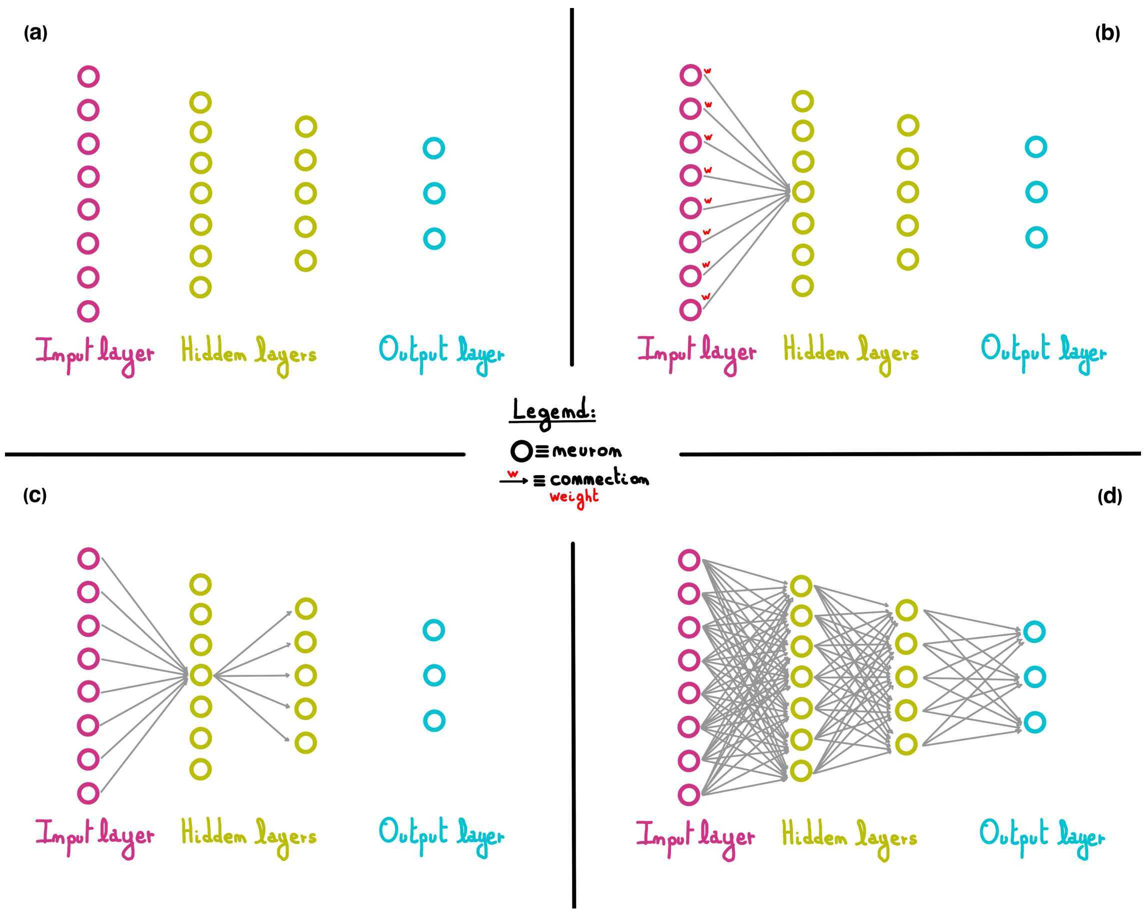


Figure 3.1: Construction of a fully connected NN. (a) The network is composed of layers. Each layer contains a specific number of neurons. (b) Between two adjacent layers, each neuron is connected. Each connection has a weight. (c) A neuron in a layer is connected to all neurons in the previous layer as well as the ones in the next layer. The weights are omitted. (d) Connecting all neurons in the same way as in (c), we obtain a fully connected NN.

3.1.2 Forward propagation

As the general NN structure is now established, we will explain how the information is treated by the network. The information contained in the input will go from the input layers, through the hidden layers, to the output layer. This is called forward propagation. We will start by describing the functioning of a neuron, then do the same for the different layers.

Neuron functioning

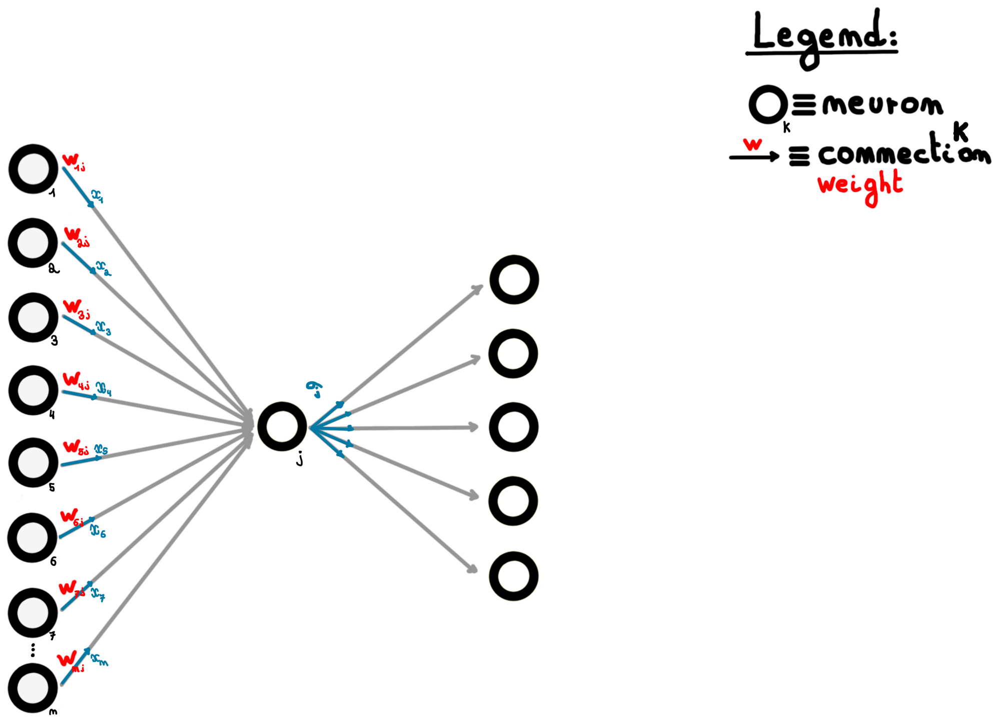


Figure 3.2: Functioning of a neuron j. Neurons K in the first layer give an output x_k through a weighted connection to neuron j. From all these inputs, neuron j gives an output to all the neurons in the next layer.

To understand forward propagation, we first need to understand how a neuron calculates an output. As we explained in the NN structure, a neuron receives data from the previous layer through the weighted connections. From those inputs, the neuron will calculate a net weighted input:

$$net_j = z_j = \sum_{k=1}^{n} (w_{kj} \cdot x_k) + b \tag{3.1}$$

In this equation, the *net* input for a neuron j is calculated by summing products and adding a bias b. The products terms are the weight w_{Kj} of the connection from neuron K to j and the output x_K of neuron K. This product can be interpreted as the importance of the activation of a first neuron onto the next one as it would be with two "real" neurons. Notations are represented in Figure 3.2. Equation 3.1 is called the transfer function.

After calculating the net input, the neuron will apply a function σ to it:

$$a_j = \sigma(z_j) = \sigma\left(\sum_{k=1}^{n}(w_{kj} \cdot x_k) + b\right) \tag{3.2}$$

This sigma function is called the activation function. The scalar value calculated with the activation function is the output of the neuron j. Through the connection to neurons in the next layer, it will also be weighted in the same manner we just described here, this output is now a x_K for neurons in the next layer. Activation functions are mandatory for the network and must not be linear. As the net input is linear, using linear activation functions would result in a linear NN. The non-linearity of the NN comes from non-linear activation functions. These functions will help the algorithm to represent more complex ones to map the input to the output.

Input layer

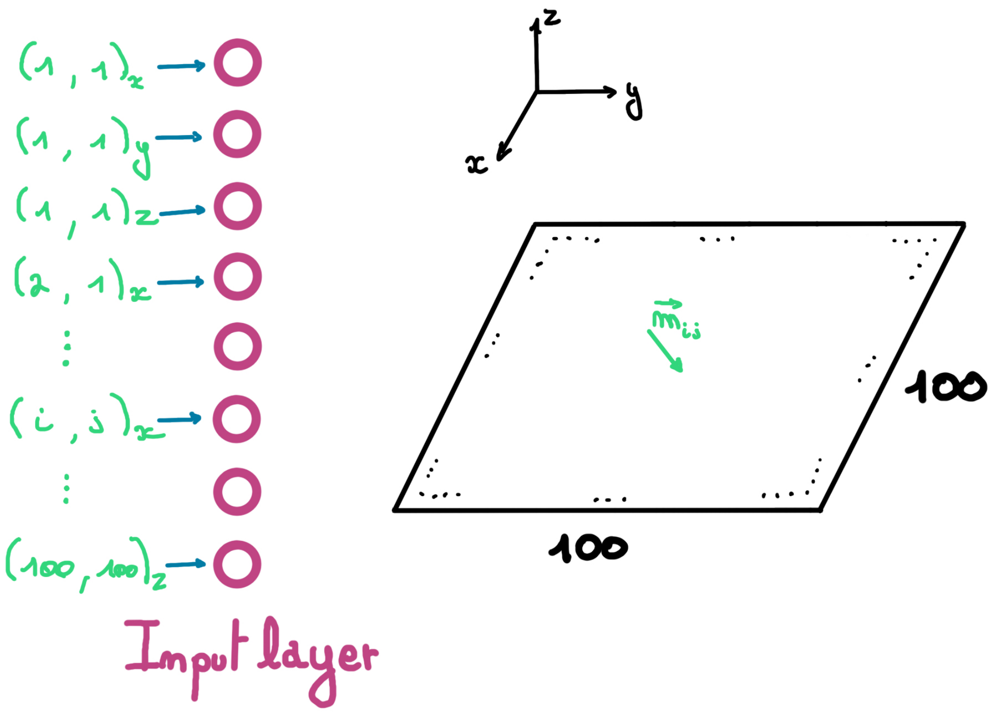


Figure 3.3: Input Layer working. The sample is made of 100 times 100 discrete points. At each point, there are three components for the magnetic moment. These components are given to the network point by point, line by line. A single neuron corresponds to a single magnetic moment component.

The input layer (Figure 3.3) is the link between the data and the network. From our script to generate databases, we can create samples with skyrmions, anti-skyrmions or with non-stable structures. A sample is composed of discrete points forming a square. For this network, we use grids of 100 per 100. At each point, Matjes gives three components[1] $\in [-1, 1]$ for the magnetic moment

[1]As a thumb rule, the data should be normalised or have values contained in $[-1, 1]$. This helps the network to converge as all values start with the same weight.

resulting in 30 000 input parameters for the network. For each parameter of data, there is a single neuron. The order of parameters given to the neurons matter. This classification contains the spatial relations of the magnetic moment components. As such, it enables to distinguish between magnetic structures. The order must be the same for all the samples given to the network. For convenience, we kept a simple classification consisting in giving the three components of the spin along x, y and z line by line[2]. For the first layer, there is no activation function, the weights are set to 1 and the biases to 0. The purpose of this layer is simply to load the information into the network.

Hidden layers

The analysis of the information contained in the sample is done in the hidden layers. The more hidden layers there are, the deeper the network is. However, the number of neurons in a layer, i.e. the width, is also important. The wider and deeper the network is, the more parameters the network has to map the data to the expected output.

We saw in Equation 3.2 that the net inputs of neurons get into an activation function and explained the need of using non-linear ones to improve the network performances. The classical activation functions for the hidden layers are the step, sigmoid, ReLU, leaky ReLU and tanh functions [17]. Their definitions are the following:

- step function:
$$\text{Step}(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases} \tag{3.3}$$
- sigmoid function:
$$\text{Sigmoid}(x) = \frac{1}{1 + e^{-x}} \tag{3.4}$$
- ReLU, the rectifier linear unit:
$$\text{ReLU}(x) = \max(0, x) \tag{3.5}$$
or equivalently by
$$\text{ReLU}(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases} \tag{3.6}$$
- leaky ReLU:
$$\text{Leaky ReLU}(x) = \begin{cases} \alpha x & \text{if } x < 0,\ \alpha \in \mathbb{R}^+ \\ x & \text{if } x \geq 0 \end{cases} \tag{3.7}$$
- tanh:
$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{3.8}$$

They are all plotted in Figure 3.4. Two interesting applications using these functions are speech recognition with the sigmoid [18] and image treatment with the ReLU function [19] with impressive results. A comparison between ReLU function types showed better results for leaky ReLU [20] on MINST dataset. For this project, we started with the ReLU activation function as it is the recommended one in modern neural network [21] and we didn't change it as our results were already above our expectations.

[2]For this NN, the order of the parameters given to the NN should not matter as long as it stays the same. The weights between the input layer and the first hidden layer should just be exchanged accordingly to have the same net input for the neurons in the first hidden layer. However, this will not be possible for CNN as explained in the next chapter.

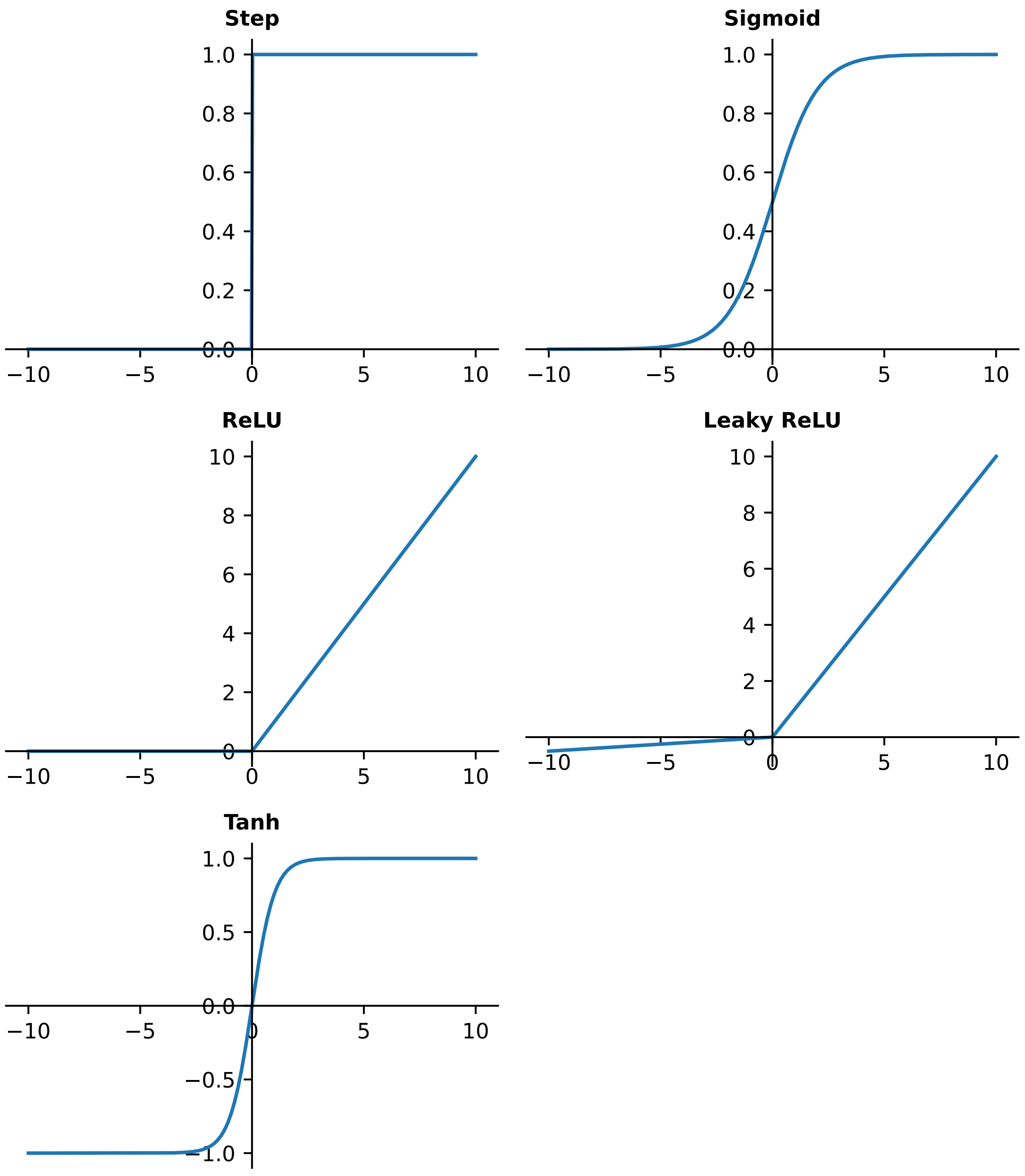


Figure 3.4: Mainly used activation functions. All the functions are non-linear. The step, ReLU and leaky ReLU are not continuously differentiable functions, unlike the sigmoid and the hyperbolic tangent.

Output layer

The output layer has a specific purpose. This is where the overall function depending on the input of the network will give a result. Hence, it needs to be set in accordance to the task. In our case, for each sample, this result must be either one of the magnetic structures categorised in our database or none of the labelled ones. This is not a binary classification. Indeed, we have three different answers. Each answer will be given by a neuron. Therefore, the output layer is made of three neurons as in Figure 3.5.

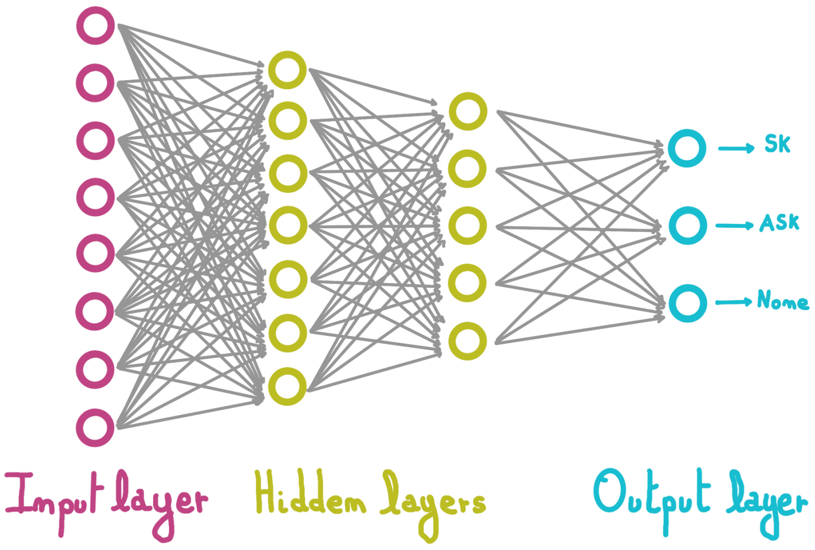


Figure 3.5: Output layer

The neurons in the output layer work in the same manner as described before. They follow Equation 3.2, so they receive a weighted input and apply an activation function on it. Now, we need to chose an activation function that corresponds to our purpose. This purpose applies to each neuron. From the evaluation of our sample, the neuron will calculate an output. From this output, the presence of the structure represented by the neuron must be deducible. Moreover, the answer should offer a degree of certainty. The prediction should be converted between $[0, 1]$ to represent a probability. Considering this, the step, relu and tanh functions are not good candidates. In our mainly used activation function the best one would be a sigmoid. Indeed, the sigmoid function maps all real values giving an answer between 0 and 1. Our database has the specificity that the 3 categories (SK, ASK, None) are exclusive. Ideally, we would prefer an output normalised for the different categories. Fortunately, there is a specific function for that purpose: a generalisation of the sigmoid: The softmax funtion. The softmax function is defined by:

$$\text{Softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}} \tag{3.9}$$

The softmax function takes a vector $\vec{x} \in \mathbb{R}^K$ of components x_i, applies the exponential function to its components, then normalises the vector.
In the case $K = 1$, it is trivially found that the result is 1 whatever the value. In the case $K = 2$

with one component $x_i = 0$, let's say $x_2 = 0$, we have:

$$\text{Softmax}\begin{pmatrix} x_1 \\ 0 \end{pmatrix} = \begin{pmatrix} \frac{e^{x_1}}{e^{x_1}+e^{x_2}} \\ \frac{e^{x_2}}{e^{x_1}+e^{x_2}} \end{pmatrix} = \begin{pmatrix} \frac{e^{x_1}}{e^{x_1}+1} \\ \frac{1}{e^{x_1}+1} \end{pmatrix} \tag{3.10}$$

Then slightly modifying the last vector, we easily find:

$$\text{Softmax}(x_1) = \frac{e^{x_1}}{e^{x_1}+1} = \frac{1}{1+\frac{1}{e^{x_1}}} = \frac{1}{1+e^{-x_1}}$$

which is Equation 3.4, the standard logistic function also called the sigmoid function, and

$$\text{softmax}(x_2) = \text{softmax}(-x_1)$$

Finally, let us take an example in the case of $K = 3$, which corresponds to the number of classes we have to classify our magnetic structures:

$$\text{Softmax}\begin{pmatrix} 1 \\ 2 \\ 4 \end{pmatrix} = \begin{pmatrix} 0.0420 \\ 0.1142 \\ 0.8438 \end{pmatrix}$$

This example shows that the softmax puts all values in $[0, 1]$, amplifies the larger values and normalises the results. This last point allows to interpret the results as probabilities to have the structure present in the sample or not. Indeed, the samples generated contain single structures -if there is any. Hence, there is only one correct answer that can be chosen by taking the higher value of the normalised predictions using the argmax function.

3.1.3 Metrics to evaluate NNs

Up to now, we have seen how neurons work, how to shape the input layer, we have a set of functions to use as activation for the neurons in the hidden layers and we have shaped the output layer for topological structure recognition regarding the database created during this project.
Let's predict a few structures from samples of our database and put the results in Table 3.1. Each sample generated is represented in the second column and has its own ID. With this ID, we can look at the sample label in the label list. The labels are encoded with numbers (cf Chapter 2): $0 = none$, $1 = SK$, $2 = ASK$. Finally, from sample data, the network return probabilities[3]. These probabilities are reported in the last column.

Taking a look at Table 3.1, two problems can be observed. First, by choosing the higher values for the prediction, all the predictions do not match their labels (cf red label boxes). This is a fundamental problem as we want to be able to make correct predictions. Second, the probabilities given for a sample may be close to one another. This means that the NN is not able to differentiate between the structures. Ideally, we would like a probability of 1 for the correct label. This would mean a perfect classification for the structures.

To resolve these two problems, we need to introduce some metrics. To begin with, there is an easy metric to characterise the fact that the predictions match with the labels or not: the accuracy. We simply have:

$$accuracy = \frac{\#correct\ prediction}{\#total\ prediction} \tag{3.11}$$

To measure the discrepancy between the ideal probability and the one obtained by the network, we will start with a function called Categorical Cross Entropy (CCE). This function is sometimes called *softmax loss* because it is the commonly recommended to use with the softmax activation

[3]Examples at this stage.

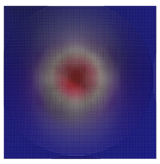
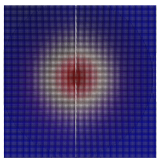
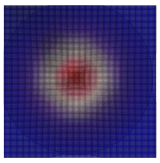
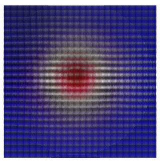
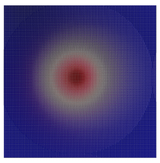
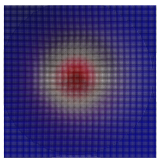
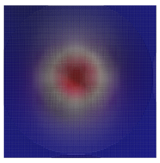
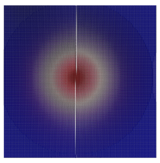
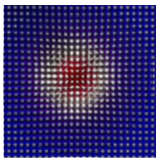
Sample ID	**Sample**	**Label (y_i)**	**Predictions ($P_{i,j}$)**
576		🟩 1	[0.10, 0.75, 0.15]
577		🟩 0	[0.80, 0.10, 0.10]
578		🟩 2	[0.10, 0.20, 0.70]
579		🟥 0	[0.20, 0.50, 0.30]
580		🟩 0	[0.60, 0.30, 0.10]
581		🟥 0	[0.20, 0.40, 0.40]
582		🟥 1	[0.40, 0.30, 0.30]
583		🟥 0	[0.30, 0.20, 0.50]
584		🟩 2	[0.25, 0.35, 0.40]

Table 3.1: Examples of predictions using forward propagation. Samples 577 and 583 show a small discontinuity (white line). Therefore, they are not topologically protected. Predictions are a probability for each category. We expect the NN to have a higher probability at the index of the label.

function in the last layer. The CCE loss function is defined by:

$$\text{CCE} = -\sum_{i=1}^{S}\sum_{j=1}^{C} y_{ij}\log(P_{ij}) \tag{3.12}$$

where:

- S is the number of samples
- C is the number of categories
- y_{ij} is the probability (one-hot encoded) for the sample i and category j
- P_{ij} is the probability for the sample i given by the network for the category j

The one-hot encoding for the labels consists in giving a value of 1 to the correct category and value 0 for the others. For example if we have some images of cats and dogs, we can encode them as [1, 0] and [0, 1]. We can also go a bit further and give the ratio of each category in the sample. For example, if we want to have a NN which finds the ratio of red, blue and green pixels turned on to display picture i, we could have a label encoded as:

$$[P_{i,red}, P_{i,blue}, P_{i,green}] = [0.42,\ 0.21,\ 0,37]$$

This is sometimes called continuous categorical [22]. This kind of encoding can be used in facial expression recognition [23] where each face may have more than a single expression (e.g. Happiness and surprise at the same time). However, in the case of topological structure recognition with single structure in the sample, we have only one category at a time. Hence, there is only one $y_{ij} \neq 0$ and this $y_{label} = 1$. This leads to the possibility to use a simplification of the CCE: the sparse categorical cross entropy (SCCE) defined as:

$$\text{SCCE} = -\sum_{i=1}^{S}\log(P_{i,label}) \tag{3.13}$$

where:

- S is the number of samples
- $P_{i,label}$ is the probability for the sample i given by the network for the index given by the label value.

This is the metric that is used for the networks in this project as it is easy to encode the labels, it requires less memory and less computation time to convert or create the labels in one-hot. This is in continuity with the idea of producing an efficient model that can be calculated on a small computer.

Now that we have some metrics to evaluate our results, we can compute the SCCE of the predictions given in Table 3.1. In details, we have:

- Sample 576: Label: 1, Prediction: [0.10, 0.75, 0.15] $\rightarrow L_1 = -\log(0.75) = 0.29$
- Sample 577: Label: 0, Prediction: [0.80, 0.10, 0.10] $\rightarrow L_2 = -\log(0.80) = 0.22$
- Sample 578: Label: 2, Prediction: [0.10, 0.20, 0.70] $\rightarrow L_3 = -\log(0.70) = 0.36$
- Sample 579: Label: 0, Prediction: [0.20, 0.50, 0.30] $\rightarrow L_4 = -\log(0.20) = 1.61$
- Sample 580: Label: 0, Prediction: [0.60, 0.30, 0.10] $\rightarrow L_5 = -\log(0.60) = 0.51$

- Sample 581: Label: 0, Prediction: [0.20, 0.40, 0.40] $\rightarrow L_6 = -\log(0.20) = 1.61$
- Sample 582: Label: 1, Prediction: [0.40, 0.30, 0.30] $\rightarrow L_7 = -\log(0.30) = 1.20$
- Sample 583: Label: 0, Prediction: [0.30, 0.20, 0.50] $\rightarrow L_8 = -\log(0.30) = 1.20$
- Sample 584: Label: 2, Prediction: [0.25, 0.35, 0.40] $\rightarrow L_9 = -\log(0.40) = 0.92$

Summing all the individual costs, we have:

$$SCCE = 7.92$$

If we had always had the correct prediction with a probability of 1, the logarithms would have been equal to 0. Hence, the loss would also have been 0. The question is then: How can we improve these results? Our NN structure is defined, we know how forward propagation works in details. Hence, we can make predictions but we haven't learned how to train our NN yet. This will be the object of the next section.

3.2 Training with gradient descent

This section is dedicated to the modifications needed in order to improve the results of our NN. Previously, we fixed our NN structure through the layer sizes and activation functions regarding our database. This allowed forward propagation for the samples in Table 3.1. We just changed the inputs by using different samples and we obtained different probabilities to find a structure depending on the sample. To calculate these probabilities, the NN calculates the activation a_j layer by layer for each neuron j:

$$a_j = \sigma(z_j) = \sigma\left(\sum_{k=1}^{n}(w_{kj} \cdot x_k) + b\right) \tag{3.2}$$

where the weights and biases are fixed and the x_k depends at first only on the input of the NN. This leads to different predictions depending on the sample used.

Now, we need to introduce some modifications in our network. These modifications will not take place in the structure of the NN. Otherwise, we would have different NNs. We would like to keep the same structure but still do some modifications to obtain better predictions whatever the sample. To apply these modifications, it will be the opposite of forward propagation procedure. We will keep the sample fixed and modify the weights and biases in order to improve the predictions regarding the labels. For this purpose, the outputs are also fixed.

As we saw in the previous section, having better results leads to a reduction in the loss function, our metric which assesses the accuracy and certainty of responses from the NN. Let's first neglect the biases. To calculate this loss, we can use Equation 3.13. This equation depends on $P_{i,label}$. This probability depends on the weights between the last layer L and the previous one $(L-1)$, the activation function a^L and the x_i through Equation 3.2. If we continue to make the terms explicit, we obtain the loss expression as:

$$Loss\left(y_{label}, a^L\left(W^L a^{L-1}\left(W^{L-1} \;...\; a^2\left(W^2 a^1\left(W^1 x\right)\right) \;...\; \right)\right)\right) \tag{3.14}$$

This equation highlights two specificities: First, if we keep the NN structure, the expected output and the input x fixed, the only editable parameters are the weights. Second, a change in a weight within the input layer or the hidden layer modifies the input values of all the next activation functions. This leads to a lot of calculus each time we will modify a weight and we already know that we have a lot of them as there are already 30.000 neurons in the first layer fully connected to the next ones. From these observations, a more efficient way to modify the weights is needed to avoid time consumption.

3.2.1 Back-propagation

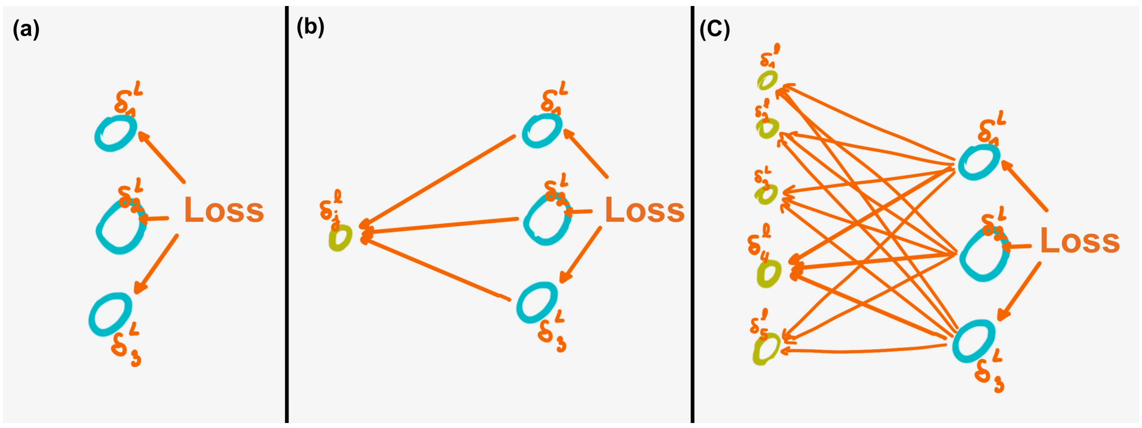


Figure 3.6: Back-propagation: (a) The errors δ_j^L are calculated from the loss for each neuron in the output layer. (b) The errors are propagated to the previous layer for a neuron. (c) The errors in layer $l-1$ are calculated from the errors of layer l.

To begin with, we have a well-known tool to properly change parameters instead of doing random tries: the derivative. We will need to calculate the derivatives of the loss regarding the weights. We just saw that a modification of a weight in the first layers leads to a lot of computation. This will be the case if we calculate the derivative through the computation of $\frac{\partial C}{\partial w_i} = \frac{C(w+\epsilon)-c(w)}{\epsilon}$. To solve this problem, we will use an algorithm called back-propagation introduced in 1986 by David E. Rumelhart, Geoffrey E. Hinton and Ronald J. Williams [24]. This algorithm consists in calculating the derivatives of the loss regarding the weights using the chain rule for each neuron. We have:

$$\frac{dC}{da^L} \cdot \frac{da^L}{dz^L} \cdot \frac{dz^L}{da^{L-1}} \cdot \frac{da^{L-1}}{dz^{L-1}} \cdot \frac{dz^{L-1}}{da^{L-2}} \cdot \ldots \cdot \frac{da^1}{dz^1} \cdot \frac{dz^1}{dx} \tag{3.15}$$

where:

- C is the cost function
- a^L is the activation function
- z^L is the net input of a neuron depending on the weights
- x is the input

Using Equation 3.2, $\frac{da^L}{dz^L}$ is simply the derivative of the activation function. In the case of the ReLU, we take the derivative at 0 equal to 0. Using Equation 3.1, we see that $\frac{dz^{L-1}}{da^{L-2}}$ is the matrix of weights. Let's get deeper into the details with a brief summary of chapter 2 of *Neural Network and Deep Learning* [25]:

Let's define the error δ_j^l of the neuron j in a layer l as:

$$\delta_j^l = \frac{\partial C}{\partial z_j^l} \tag{3.16}$$

The error caused by the neurons in the last layer can be calculated using the chain rule (Figure 3.6(a)):

$$\delta_j^L = \frac{\partial C}{\partial a_j^L} \frac{\partial a_j^L}{\partial z_j^L} = \frac{\partial C}{\partial a_j^L} \sigma'\left(z_j^L\right) \tag{3.17}$$

This error can be recursively propagated to a neuron in layer l (Figure 3.6(b)) if we express the error δ_j^l in terms of δ_k^{l+1}. Starting with the definition of δ (Equation 3.16), using the chain rule, then the definition of δ again, we successively have:

$$\delta_j^l = \frac{\partial C}{\partial z_j^l} = \sum_k \frac{\partial C}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial z_j^l} = \sum_k \frac{\partial z_k^{l+1}}{\partial z_j^l} \delta_k^{l+1} \tag{3.18}$$

Let's develop $\frac{\partial z_k^{l+1}}{\partial z_j^l}$. We start with the expression of z_k^{l+1} using Equation 3.2 then differentiate the expression:

$$z_k^{l+1} = \sum_j w_{kj}^{l+1} a_j^l + b_k^{l+1} = \sum_j w_{kj}^{l+1} \sigma(z_j^l) + b_k^{l+1} \tag{3.19}$$

$$\frac{\partial z_k^{l+1}}{\partial z_j^l} = w_{kj}^{l+1} \sigma'(z_j^l) \tag{3.20}$$

Putting this result in (3.18), we finally have:

$$\delta_j^l = \sum_k w_{kj}^{l+1} \delta_k^{l+1} \sigma'(z_j^l) \tag{3.21}$$

This last equation gives the relation to calculate the error in layer l in terms of the error in layer $l+1$, the weights between these layers and the derivative of the activation function. Rewriting Equations 3.17 and 3.21 in terms of matrices, we find:

$$\delta^L = \nabla_a C \odot \sigma'(z^L) \tag{3.22}$$

and

$$\delta^l = \left(\left(w^{l+1} \right)^T \delta^{l+1} \right) \odot \sigma'(z^L) \tag{3.23}$$

where $\odot$ is the Hadamard product (Figure 3.6(c)). Using the chain rule as we did to obtain the two equations above, we also find:

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \tag{3.24}$$

and

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l \tag{3.25}$$

With these equations for the back-propagation algorithms, we can summarise the back-propagation algorithm in a few steps:

1. For a fixed sample, do the forward propagation to calculate the predictions.
2. Calculate the loss function with the predictions and the labels using (3.13).
3. Calculate the error for the last layer using (3.17). Then propagate the error to every other layer with Equation 3.21.
4. Calculate the derivatives of the loss with respect to the weights (3.25) and biases (3.24).

By using this procedure, the derivatives are computed in an efficient manner: First, the derivatives are not calculated using the definition of a derivative. As we said, proceeding like this would require propagating forward a small increment, which would require a lot of computations for every partial derivative. Second, there is no duplicate calculation, only a single pass on the network backward.

3.2.2 Optimiser

Doing back-propagation left us with the partial derivatives regarding the weights and the biases. To modify these parameters, we now need to follow the opposite direction of the gradient to reduce the loss. To choose the step length, we use an optimiser: Adam [26]. This optimiser showed the best results in 2015 on the MNIST database as we can see on Figure 3.7 and is now one of the most used optimiser for NNs.

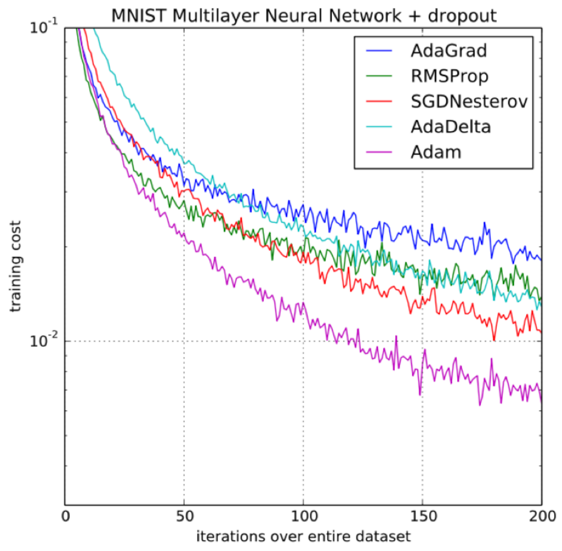


Figure 3.7: Adam optimiser compared to other optimisers on the MNIST dataset. Adam shows the best results on the MNIST dataset.

3.2.3 A word on hyper-parameters and data pre-treatment

Hyper-parameters are the parameters set before the training starts. There are the number of layers, their size, the activation functions, and two other parameters we haven't spoken about yet: Epochs and batch size. An epoch is the number of time the NN will see each sample of the database during training. The more epochs there are, the better the NN become on the training database. We saw above that learning consists in updating the weights to allow the NNs to learn and become more efficient on their tasks. This updating is time consuming. Still, it is mandatory. There are multiple options defining the updating moment of the weights:

- batch: the weights are updated once after each epoch
- mini-batch: weights are updated after a specific number of samples
- stochastic gradient descent : the weights are updated after each sample

The batch uses a lot of memory due to all of the information cached before the weight update and is slow but has a very stable gradient. The stochastic gradient descent causes a lot of noise due to unstable gradients computed for each sample but is fast and doesn't use a lot of memory. A compromise is to use mini-batch updating the weights after a few samples. This mitigates the advantages and disadvantages of the two other methods. This is the method used here.

The data pre-treatment was realised with TensorFlow. It consists in linking the output of Matjes to TensorFlow and possibly to make operations to increase the size of the database. This was done for the sake of convenience and efficiency. Proceeding like this allows using the NNs directly onto Matjes outputs without a separated script to organise the data correctly. Moreover, the data was fed to the NNs by realising a pipeline. This technique consists in loading the data only when they are needed. This allows less memory usage and bigger dataset to train on. To realise and improve this pre-treatment with a pipeline, a great amount of time was spent. It resulted in a time reduction of 80% for each loaded sample.

3.3 A deep NN for topological structure recognition

All of the explanations on the functioning about the created NN used in this chapter were given in the previous sections. Now, a look at the training and capacity of the networks can be taken.

3.3.1 NN structure

The structure of the NN created consists in five layers. The input layer has a size of 30.000 to match the size of generated samples ($100 \times 100 \times 3$). The three hidden layers are made of 1000, 256 and 64 neurons, all using the ReLU activation function. The final output is made of 3 neurons using the softmax activation function. The batch size is set to 32 to accelerate computation.

3.3.2 Databases

To train and evaluate the network, multiple databases were generated. They differ by their size and the parameters used. However, all the samples have a grid size of $100 \times 100 \times 3$.

A first database (LDB) is made of 8000 labelled samples. The centers of the structures are placed every 5 points along x and y, i.e. on 20 different positions along each direction. The size of the structure is set to an element $\in \{6, 16, 26, 36, 46\}$. Then, the helicity along x and y are set to -1 or 1. This results in a generation of 2 SK and 2 ASK at each position for each structure size.

After the first database, two other databases were generated (TDB), each containing a single structure type. The positions are set every 20 points, the size of the structure are $\in \{6, 10, 14, 18, 22, 26, 30, 34, 38, 42, 46\}$. The helicities are set to $(-1, -1)$ for the SK database and $(1, 1)$ for the ASK database. This results in 275 samples for each TDB with eight out of eleven different sizes from the LDB.

3.3.3 Results of the training

At first, the weights are randomly set and the database is divided in two parts ($80\% - 20\%$). Then, the training begins using back-propagation and adam gradient descent optimiser.

The results of the training are plotted in Figure 3.8 for 250 epochs. The value of the loss at the end of each epoch done with 80% of the database is reported in light orange: After the first epoch, the loss is above 1.2. One epoch further, the loss decreases quickly around 0.25 showing the efficiency of Adam optimiser. From there, the loss decreases almost linearly to 0.012. A linear fit is done in orange. One can observe that sometimes, the loss can increase or some losses are way above the general trend. This is interpreted as too high a learning rate applied by the optimiser. This kind of noise is also observed in Figure 3.7 whatever the optimiser used. Around epoch 220, the loss starts to get flatter, all values are under 0.020. This means the NN is stable, the weights and biases don't move much anymore.

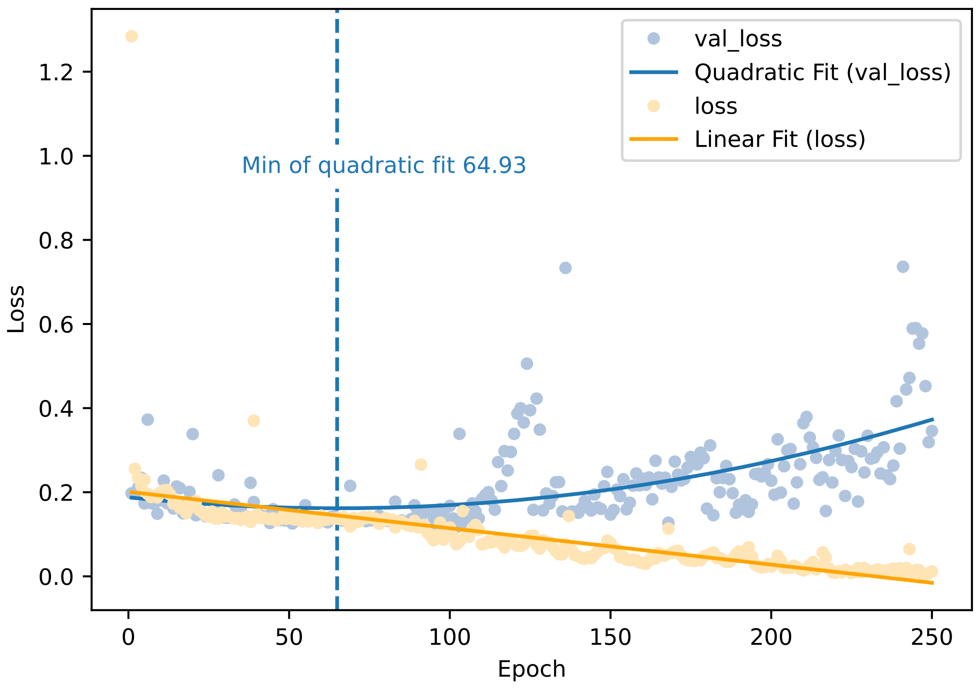


Figure 3.8: Plot of the losses during training over 250 epochs on LDB. The light orange dots are the loss on the training subset and the light blue ones represent the loss of the validation subset. The loss decreases almost linearly when the one on the validation sample decreases and then increases again. A quadratic fit allows to find the minimum of the validation loss.

The question is to know when to stop the training. The two extremes are after one epoch and when the weights and biases don't move anymore. This last limit was observed during the multiple training done during the project and fluctuates a lot[4]. A common habit is to keep a part of the dataset to do validation. We use 20% of the dataset for this purpose. The NN does not learn on this subset. It only evaluates the loss. This loss is reported in Figure 3.8 in light blue. At the beginning, this validation loss (VL) is below the loss. Through epochs, it tends to decrease and then moves on to increase. To evaluate the transition moment, a quadratic fit is realised. Using the derivative, the minimum is found around the sixty-fourth epoch. From that point, the VL is overall on the increase. When this happens, the NN starts to overfit the data. It is said that the NN begins to lose in generalisation capability.

We just saw that the loss on unseen data can be an answer to define a point where we can stop the training. We know that the loss function used is a metric to evaluate the difference between the predictions of the NN and the real probability of the correct predictions. And, the purpose of the loss is to have a metric on which the NN will be able to improve. But, this is not our ultimate goal. The structures must be correctly classified and this is evaluated with the accuracy (c.f. Equation 3.11). The accuracy evaluated during learning on the training and validation subsets are plotted in orange and in blue in Figure 3.9.

[4]Fluctuations were over 100 epochs.

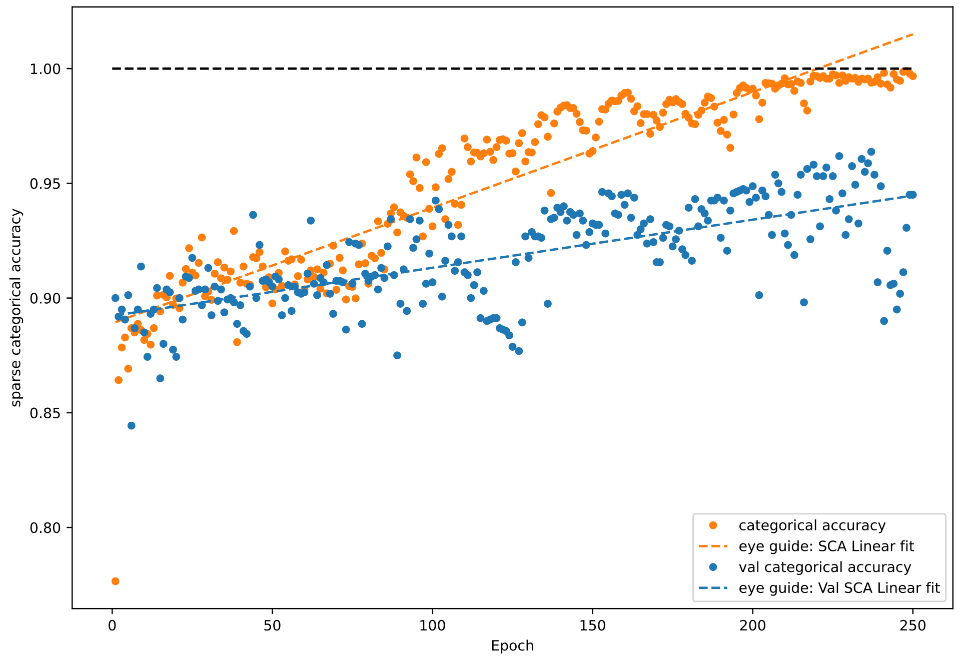


Figure 3.9: Plot of the accuracy evaluated during training over 250 epochs on LDB. The accuracy of the training subset increases and becomes gradually smoother through epochs. The accuracy on the validation subset is sparse and increases more slowly than the accuracy on the training subset.

Figure 3.9 shows an accuracy of 0.77 after a single epoch on the training subset and 0.90 on the validation one. It stays around 0.90 before increasing a lot from epoch 90. At that stage, the accuracy on the training subset starts to have smooth ups and downs each 10-20 epochs. A zoomed plot (Figure 11) can be found in the appendix. This highlights the differences between the two scatter plots. The general trend is an increase in the accuracy. However, the accuracy and validation accuracy do not grow at the same rate. The accuracy is higher and finishes at 0.996. The validation accuracy is always sparse and lower, as it finishes at 0.945. From epoch 120 to 250, the differences between the two accuracies are around 5%. Overall, the NN seems to classify better after 140 epochs. At the end, the network almost perfectly classifies the structure of the training subset. However, it does not on the validation one. Consequently, it is clear that a special attention must be given to the accuracy on unseen data.

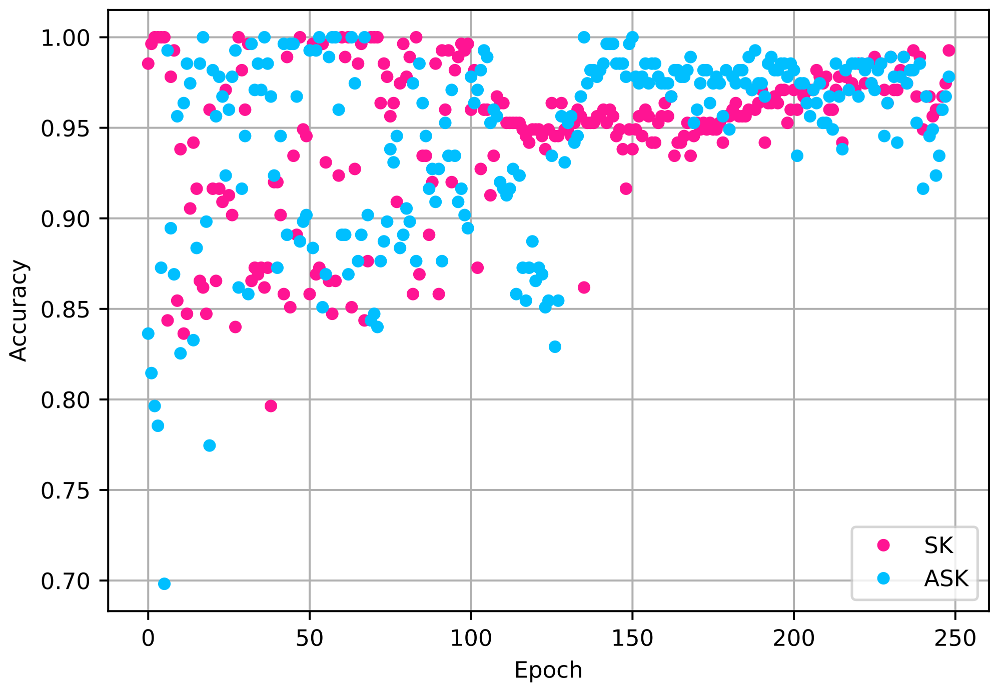


Figure 3.10: Evaluation of the network accuracy on unseen data. The evaluation is done on the two TDB containing either only SK or only ASK. At first, the data are scattered. From epoch 130, the accuracies tend to be more consistent. An overall accuracy of 97% is obtained at epoch 250.

Figure 3.10 shows evaluations of the network on unseen data. The data in question are the two databases TDB containing only SK or ASK. Each point on the figure corresponds to the accuracy of structures found in a database. To calculate the points, the weights of the NN at a fixed epoch are loaded. Then, the 275 samples of the database are propagated forward and the accuracy is computed. After this, the other database is evaluated in the same way. Finally, the weights of the next epoch are loaded. It is worth the computing time as this graph allows us to distinguish the types of structures recognised by the NN.

On Figure 3.10, there seems to be two regimes. One from epoch 0 to ≈130 and one from ≈130 to 250. In the first regime, the recognition is almost always above 82% . This recognition seems to be focused on one structure type to the detriment of the other. In the second regime, the recognition levels are closer and more consistent. They are above 90%. Choosing the weights carefully between epochs 200 and 230 seems to be the best choice regarding the results.

With this analysis, it appears that letting the NN overfit is not problematic. The ability of the network to recognise structures continues to improve even if the validation loss is increasing. From this observation, we conclude that great importance must be given to evaluating the model on unseen data. Additional evaluations were conducted (Figure 3.10). These evaluations for the NN created showed impressive results for the task at hand with an accuracy above 97% on the test databases.

Chapter 4

Convolutional Neural Network

The first convolutional neural network (CNN) was created in 1989 by Yann LeCun et al [27]. The original paper describes the network called LeNet-5. This network was created to recognise handwritten digit. It showed very low error rate and is now considered as pioneer work. Nowadays, the methodology is a standard component in modern NN architecture for image recognition. Even if our samples are not pictures and have negative values, the benefits of the assumptions of a picture encoded as a 3D matrix (height x width x colour channel) can be transferred to topological structure recognition.

4.1 Modifications of a RNN to obtain a CNN

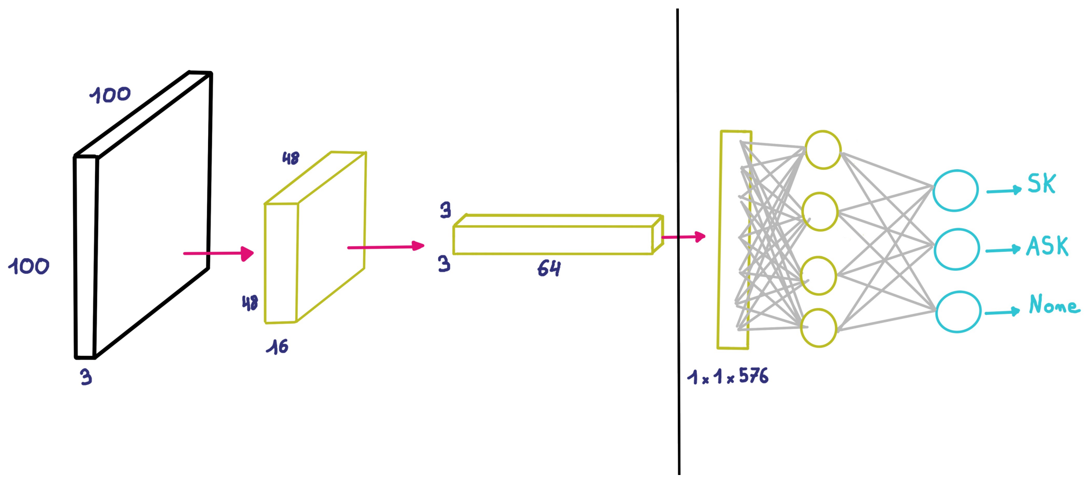


Figure 4.1: Example of CNN functioning. The network can be divided in two parts. The first part uses convolutional and pooling layers to reduce height and width while increasing the depth. Depth corresponds to features extracted by the convolutions. From part one to two, the data is flattened. In the second part, a RNN analyses the extracted features to classify the samples into categories.

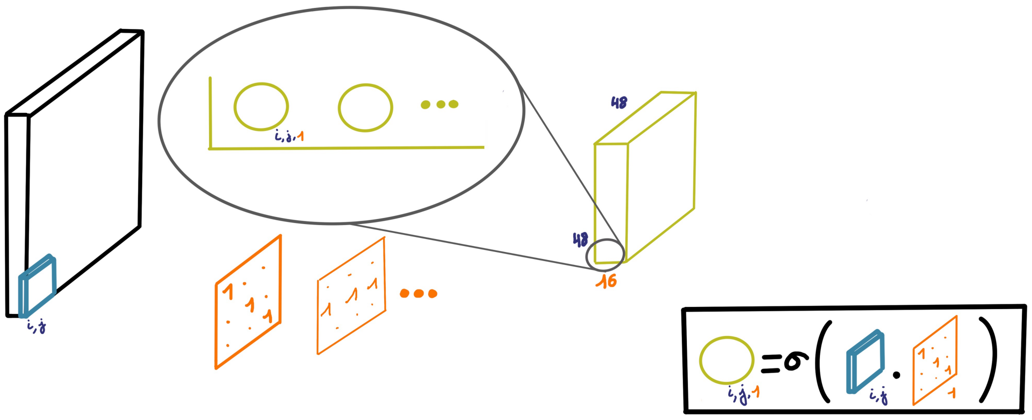
A CNN can be divided into two parts as represented in Figure 4.1. The first part contains new layer types: convolution layer, pooling layer and possibly a flatten layer. The first two layers types are presented with more details in the following subsections. The flatten layer is self-explaining: it arranges all the data in one dimension. The overall purpose of these layers is to extract spatial features. Doing so, the layers will reduce the height and width of the input while growing in the third direction, depth. To connect the first part of the network to the second one, the data may need to be flattened. The second part is a small RNN. It is made of a few layers fully connected to the features extracted by the convolutional and pooling layers on one side and to the ouput layer on the other. The output layer is constructed as before: the layer has a size corresponding to the class number and uses a softmax activation function to classify the samples. For the hidden layers, the ReLU function is chosen as in Chapter 3.

4.1.1 Convolution layers

The features are extracted by the convolutional layers (Figure 4.2) with filters (e.g. in orange). Let's explain how it works. The convolutional layer is made of neurons. Each neuron has access to a local 2D region formed by the results of the previous layer (e.g. in petrol blue) and a filter which is a matrix. It will compute the dot product of the region and the filter and optionally apply an activation function on the results. This will be the output of the neuron (e.g black frame). The filters are moved on the sample for each local region, hence the name convolutional layer. By applying multiple filters on the output of the previous layer, the depth is increased. Depending on the application, filters may be customised or learned. For example, if we're interested in diagonal lines, we can use the unitary matrix as a filter. The dot product will be sensible to value along the diagonal and therefore, the output of the neuron as well. For this CNN, we use the standard initialisation of filters included in TensorFlow and change their values like the weights in a RNN. This is standard. Using convolutional layers allows a drastic reduction of parameters. For example, in the RNN, the input layer has a size of 30000 +1 to include the biases and the second layer has 1000 neurons. This results in 30,001,000 parameters. The number of parameters for a convolutional layer is given by:

$$\text{parameter } \# = (\text{filter size} \times \text{input depth} + 1) \times \text{number of filter} \tag{4.1}$$

For the first convolutional layer, we used 32 filters of size 3x3 moved with a stride of 1. The input depth is equal to 3, the number of magnetic components. This results in only 896 parameters. Decreasing the number of parameters allows reduction of overfitting and computing time.



i,j,1
48
48
16
i,j
$\bigcirc_{i,j,1} = \sigma(\square_{i,j} \cdot \square_{1})$

Figure 4.2: Representation of the functioning of a neuron in a convolutional layer. Neurons are in pale olive green, the filters are in orange and a local region in petrol blue. To compute the output of a neuron $i, j, 1$, the activation of the dot product of the local region i, j and filter 1 is evaluated.

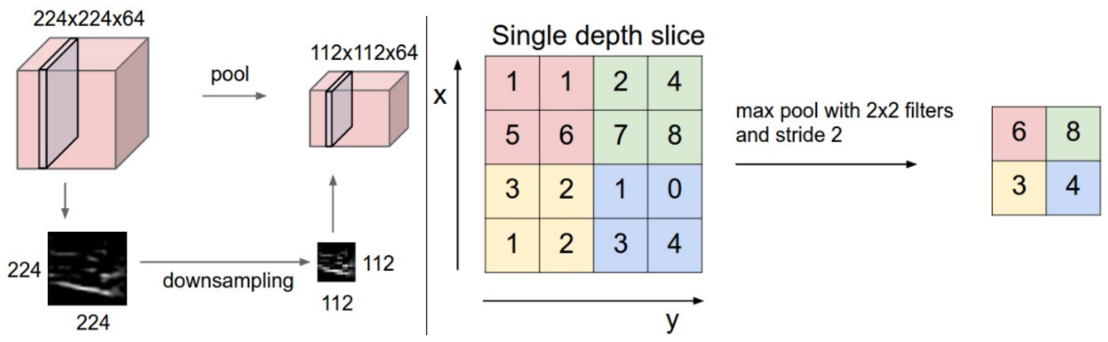


Figure 4.3: Functioning of MaxPooling layers. The figure comes from [28]. A filter of size 2x2 is applied with a stride of 2 to the input slice. This results in four windows (i.e. sub-matrices). From each window, the maximum value is taken as an output.

4.1.2 Pooling layers

The purpose of pooling layers is to reduce the width and height of their inputs. To do so, we use MaxPooling layers (Figure 4.3). These layers reduce the size of an input by slicing along the depth and dividing it in small windows (i.e in sub-matrices) along the height and width from which the maximum value is extracted. To define a window, a filter is applied. The filter size gives the size of the window and a stride defines how the filter is moved from one window to the next one. There is no learnable parameter for these layers.

4.2 A CNN for topological structure recognition

4.2.1 Structure of the CNN

The structure of the CNN created is summarised in Table 4.2.1. The structure contains two Maxpooling layers intercalled between Conv2D layers. Then, the resulting matrix is flattened before going to 2 fully connected layers. The total number of parameters is 1,862,915. In the case of the RNN, we had 30,273,899 parameters. This is an overall reduction from the previous network with a factor 16. The number of parameters between the first two layers of the CNN is 896. At the beginning of the RNN, it was equal to 30,001,000. From these numbers, we deduce that the main reduction in the number of parameters is between these first two layers. However, in the CNN, the first fully connected layer is in the end. Hence, more parameters are used in this part than in the RNN.

Layer type (parameters)	**Output Shape**	**Param #**
Conv2D - (32, (3, 3))	(None, 98, 98, 32)	896
MaxPooling2D - (2, 2)	(None, 49, 49, 32)	0
Conv2D - (64, (3, 3))	(None, 47, 47, 64)	18,496
MaxPooling2D - (2, 2)	(None, 23, 23, 64)	0
Conv2D - (64, (3, 3)	(None, 21, 21, 64)	36,928
Flatten	(None, 28224)	0
Dense (64)	(None, 64)	1,806,400
Dense (3)	(None, 3)	195

Table 4.1: Structure of the CNN created for the project. The structure is different from the one in Chapter 3: Convolutional layer with parameters as (Filter#, stride along (x, y)) and pooling layers with parameters as downscale factors along (x, y) are additions.

4.2.2 Databases

The databases used to train (LDB) and test (TDB) the CNN are the same as the ones used for the RNN. They are described in Subsection 3.3.2. The LDB is divided using the same ratio: 80% for training, 20% for the validation.

4.2.3 Results of the training

The hyper-parameters are set to 32 for the batch size and 250 for the number of epochs to be able to compare with the RNN. The optimiser is also the same as before: Adam.

Losses

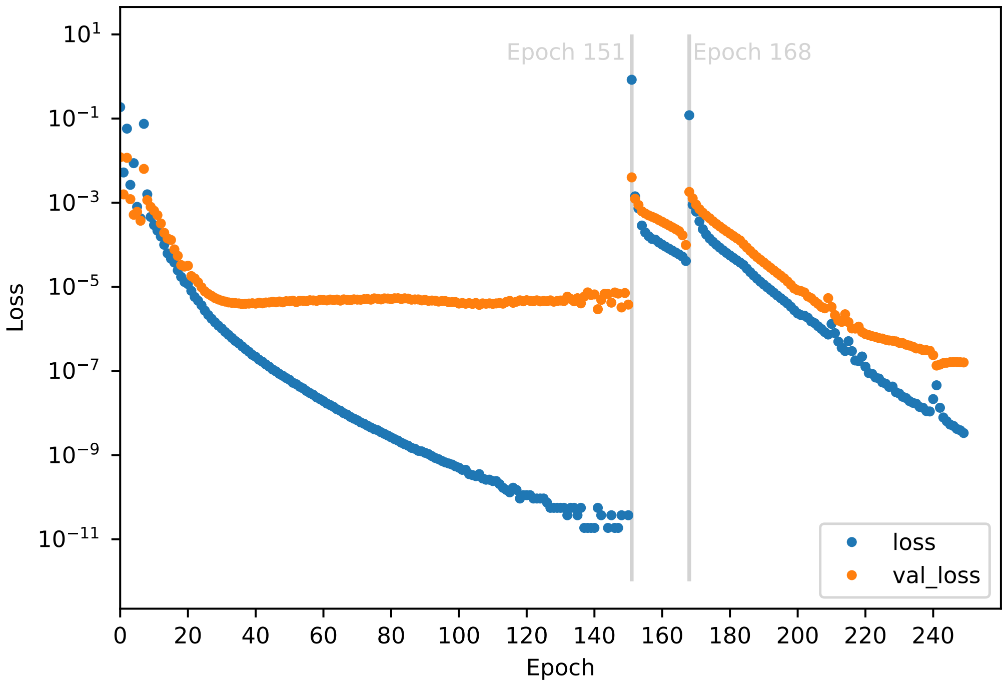


Figure 4.4: Plot of the losses during training over 250 epochs on LDB. The orange dots are the loss on the training subset and the blue ones represent the loss on the validation subset. Two drops are present at epochs 151 and 168 allowing the validation loss to decrease lower than 10^{-5}.

The losses (TL, in blue) and validation losses (VL, in orange) are reported in Figure 4.4. A zoomed plot for the 20 firsts epochs is in Appendix 12. Let's take a look at the first twenty epochs. After the first epoch, the TL is already down to 0.19. To compare with the RNN, it was 1.2 after one epoch and 0.25 after the second one. The second epoch with the CNN leads to a TL of 0.005. To obtain such a low loss with the RNN, 241 epochs were needed. Figure 4.4 shows a small instability in the loss for the first 7 epochs. Then, it becomes more stable and decreases down to epoch 151. However, the VL finds a limit around 10^{-5} after epoch 30. This indicates a first limit at which the CNN cannot generalise the data anymore. The weights have reached a minimum. We could stop the training and explanation there but an interesting phenomenon appears from epoch 105: the TL becomes unstable. After epoch 151, a huge increase in the loss is observed. This is due to the optimiser. From that point, the TL and VL start decreasing again before dropping once more at epoch 168. These drops allow the VL to decrease under the previous limit of 10^{-5}. This indicates that the previous minimum was a local one. The global minimum is not reached.

Accuracies on LDB

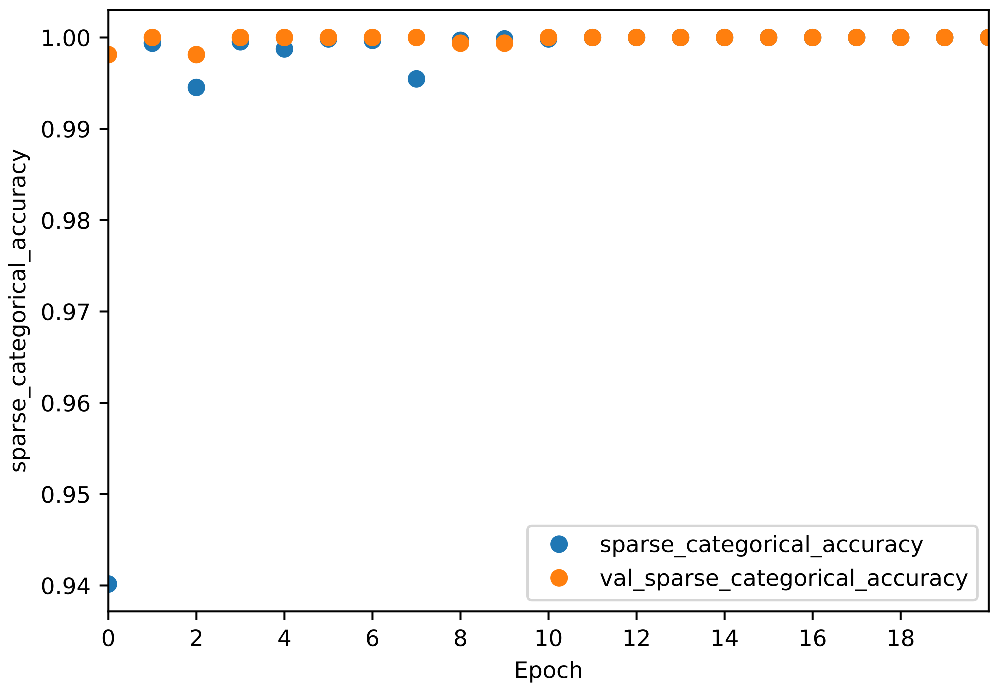


Figure 4.5: Plot of the accuracies calculated during training. The blue dots are the accuracy of the training subset and the orange dots correspond to the validation subset. After 12 epochs, both accuracies are merged. The CNN perfectly classify the LDB.

Reaching the global minimum does not seem necessary when we look at the accuracies in Figures 4.5 and 4.6. At epoch 1, the TA (training accuracy) is already above 94%. One iteration later on the LDB, it goes to 99%. At epoch 12, 100% of the structure are correctly classified. Hence, the CNN is better after 12 epochs than the RNN after 240 epochs, which saturated at 94.5%. From that point, the accuracies remain perfect except around the two loss drops at epoch 151 and 168. The decrease from the first local minimum to a lower one does not affect the results on the LDB.

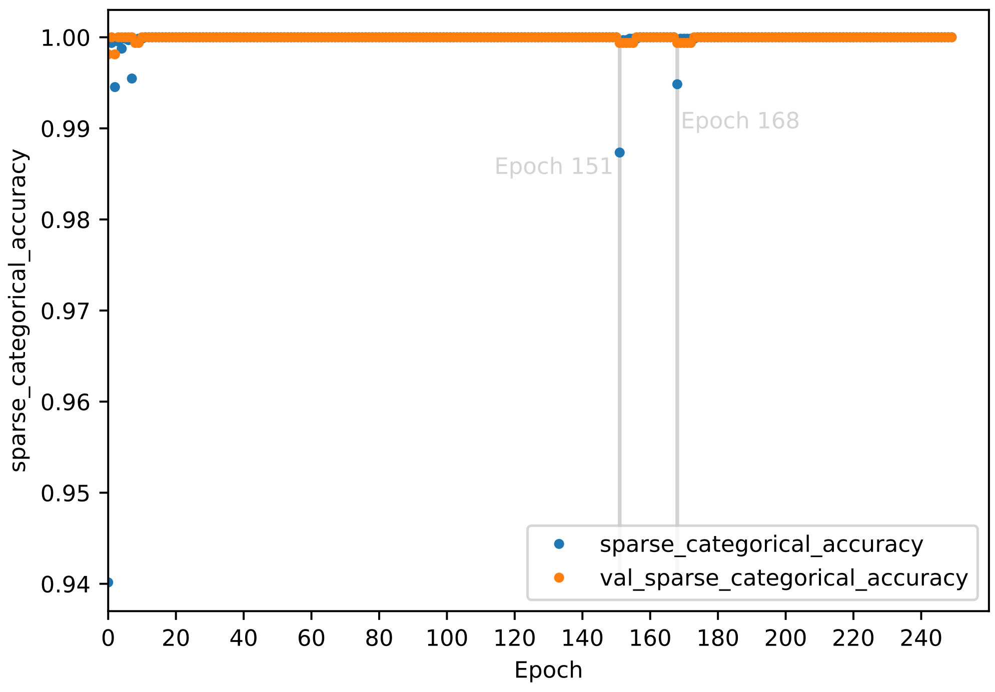


Figure 4.6: Plot of the accuracies calculated during training. The blue dots are the accuracies on the training subset and the orange dots on the validation subset. Accuracies are almost always perfect except at the beginning of the training and near the drops at epochs 151 and 168.

Accuracies on TDB

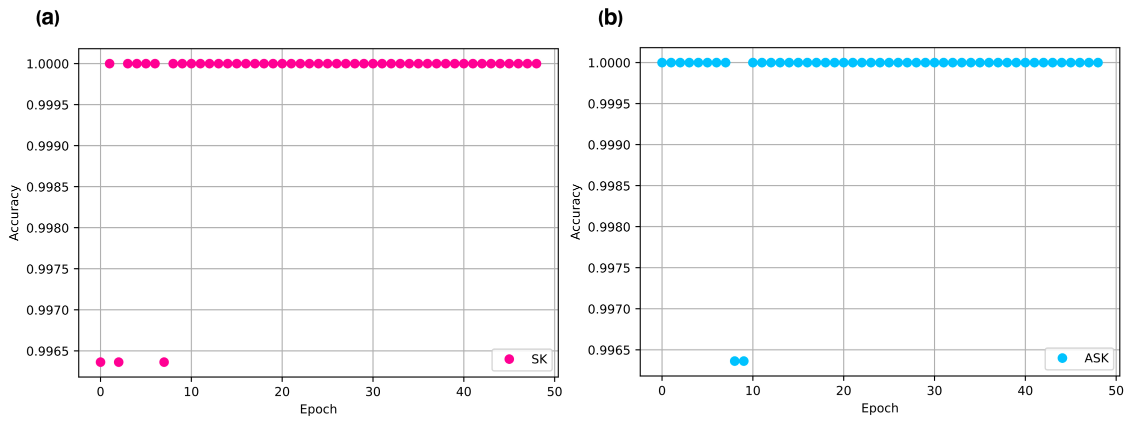


Figure 4.7: Plot of the accuracy calculated for: (a) the SK TDB, (b) the ASK TDB. Both TDBs are evaluated on 50 epochs. Both magnetic structures are almost perfectly recognise in the first ten epochs. After that, the accuracies is equal to 1.

Figure 4.7 (a) and (b) report the accuracies through 50 epochs with the TDB of SK and ASK. The skyrmions are perfectly classified after nine epochs. Ten are needed for the ASK. Unlike the accuracies of the RNN, the CNN perfectly classify the structures. Accuracies are not sparse and one is not above the other. These tests support the idea that twelve epochs are sufficient to perfectly classify the structures in the three databases generated.

With the CNN, a perfect classification of the structures generated is obtained after only twelve epochs. The loss on the validation subset continues to decrease after two drops. This indicates some improvement in generalisation can still be learned. To benefit and evaluate it, the databases should be complexified. Doing so, we would already use the training done to classify the new structures. Then, depending on the results, some diversity improvement of the LDB can be envisaged.

Conclusion

This master thesis explored the application of artificial intelligence, specifically deep learning models, for the recognition of topological structures, with a focus on magnetic skyrmions and anti-skyrmions. The work aimed to address the challenges associated with identifying these structures in dynamic conditions, which is crucial for advancing research in spintronics. By creating and training neural networks on synthetic magnetic data, this research demonstrated the potential of AI to generalise its knowledge to identify single topological phase on unseen samples with impressive accuracy and efficiency.

In the first chapter, the thesis provided an overview of the current state of magnetic topological structure with a focus on skyrmion and anti-skyrmion. The stakes of recognition are established as well as the limitations of traditional methods. It introduced the concepts of AI as a more adapted method, draw a research plan and set the foundation for the subsequent chapters.

Chapter 2 detailed the process of database generation and labelling in order to realise supervised learning. A library which automated both the generation and the labelling process is created. Comprehensive databases of synthetic topological structures can therefore be created using the generalised solution of the differential equation as given by Bogdanov et al [13, 14]. These databases served as the primary resource for training the neural networks, ensuring a robust and diverse set of examples for model learning.

Chapter 3 focused on the development and training of a Regular Feedforward Neural Network (FNN/RNN) for topological structure recognition. The results showed that the RNN was capable of classifying skyrmions, anti-skyrmions, and non-classified topological structures with a high degree of accuracy. The training process highlighted the importance of choosing carefully hyperparameters and applying systematic evaluation of the network. Even if the network overfitted the training data, the accuracy of the network was up to 97% on unseen one after 250 epochs.

In Chapter 4, the thesis expanded on the initial model by developing a Convolutional Neural Network (CNN). The CNN demonstrated superior performance in recognising topological structures due to its ability to extract spatial features from the data. The reduction in parameters compared to the RNN also helped mitigate overfitting, making the CNN a more efficient model for this task. In only 12 epochs, the CNN is able to perfectly classify the samples of the databases generated without being at the minimum of the loss function.

In summary, this thesis successfully demonstrated the use of AI for topological structure recognition, achieving impressive results with deep learning models. While challenges such as data bias and model generalisation remain, the research opens new avenues for the application of AI in the study of topological phases in matter, particularly in the rapidly evolving field of spintronics.

In light of the results obtained, several avenues for future research can be explored to build upon the work initiated in this thesis. Firstly, generating databases that incorporate temperature effects would be a significant advancement. This would allow for the simulation of more realistic conditions and a better understanding of the impact of thermal fluctuations on the recognition of topological structures. Additionally, increasing the number of classified topological structures, including more complex or less-studied configurations, would broaden the applicability of the developed models, making the neural networks more versatile and robust. Another important aspect to consider is the evaluation of the necessary number of samples required to effectively train the networks, which could optimise resource usage and training efficiency. Furthermore, training the networks multiple times to obtain error bars would provide a more reliable assessment of model performance and variability.

Building on the successes of the CNN, the next logical research step is the exploration of more advanced neural network architectures for segmenting topological structures. The UNet architecture, introduced in the next part of this work, represents a significant shift from classification to segmentation, offering a more detailed analysis for magnetic skyrmion dynamics. Although the initial implementation of the UNet did not reach the same accuracy levels as the CNN, it demonstrated the potential for capturing the spatial complexities of topological magnetic structures in greater detail. This advancement is not only crucial for the accurate mapping of magnetic skyrmions but also holds promise for broader applications in the dynamic analysis of topological phases. As such, the UNet architecture serves as a promising tool for future research, with the potential to significantly impact the development of spintronic technologies.

Perspective: UNet

This chapter describes the progress made through a UNet. It is not just an opening as a significant amount of time was spent in order to advance on the development plan. Even if the network does not show all the expected results yet, it is an achievement that will surely benefit the following research.

In Chapter 4, the created CNN is able to classify with an accuracy of 100% both training database and unseen data of the TDBs. Instead of pursuing in the way to classify more complex single topological structures, another way is chosen in the direction of skyrmion dynamics. The next goal is to develop an NN which would be able to convert moving magnetic structures into images containing coloured areas depending on the structures and their parameters. To deal with the moving property, a sequence of samples can be made. Hence, an NN capable of classification and localisation of multiple structures is sufficient. Then, from the image sequence obtained, post-treatment will be possible to analyse the dynamics.

To deal with the challenge of spatial separation of multiple structures in a sample, a network must be able to do segmentation. Image segmentation is a process able to simplify an image into meaningful regions. To do so, a UNet seems to be a good candidate. Unet was proposed in 2015 by Olaf Ronneberger, Philipp Fischer, and Thomas Brox [29] to deal with biomedical images. A few key points of the UNet can be pointed out:

- it is a convolutional NN,
- it requires only a small labelled database as it uses a lot of image augmentation,
- it performs segmentation by working at pixel scale. Each pixel is assigned to a class.

The structure of the original UNet is represented in Figure 5.8. With this figure, the origin of the UNet name is quite obvious. In this NN, the input image is a grey scale image. This image goes through convolution layers and MaxPooling layers on the left part of the U, the downsampler. The functioning of these layers are described in Chapter 4. On the right side of the U, the upsampler, the pooling layers are replaced by upsampling operators. Combined with the features extracted by the downsampler (the copy/crop represented in grey), the NN is able to reconstruct images with a great precision.

5.3 The assembled UNet

To implement a UNet, the fine tuning technique is chosen with transfer learning. This technique consists in taking parts of NN already trained on huge databases (millions of samples) and modify the inputs/outputs to correspond to the task at hand. Doing so, the weights of the network can be partly fixed. This reduces the computing time. For the downsampler, we take the MobileNetV2 [30] layers. These layers are trained for image sizes of multiples of 2, starting at 32. The weights of these layers are fixed. We use an input layer followed by five layers to get down to a size of 4×4. For the upsampler, we take the pix2pix upsampler [31] and let the weights change. We use five layers to go back to 128×128. There are connections between all the layers except for

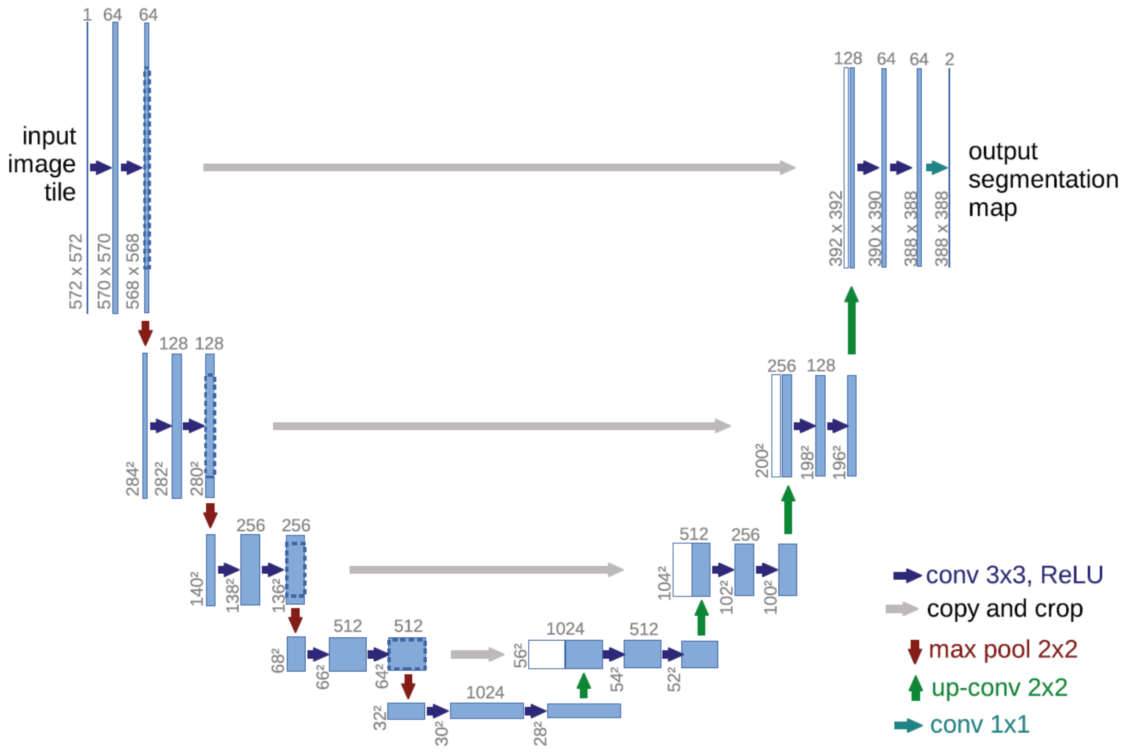


Figure 5.8: Structure of the original UNet [29]. The NN has a U shape. It consists in two parts: a downsampler and an upsampler. Both parts are connected through the grey arrows and the bottom of the U.

the input/output stage. Finally, the labels (which are represented for example by integers 0 for *nothing*, 1 for *skyrmions* and 2 for *anti-skyrmions*) must be changed to allow training. The NN works with the three labels for each pixel to compute the loss.

5.4 Modifications concerning the databases

The UNet needs two main changes in the database. First, the size of the samples must be a multiple of two. From 100×100, we change to 128×128. Second, the labels for each pixel must be created.

To do this labelling, we generate magnetic textures with a single topological magnetic texture and add masks. A mask is a matrix in which each value must be a label (so 0, 1 or 2 in our example). Each of these values is related to a pixel in the representation of the magnetic texture. After computing, we are left with a matrix of labels the position and values of which should correspond to the position and the type of the magnetic textures.

In practice, the mask matrices are initialised with 0. The number of matrix depends on the number of magnetic structures we want to classify. Then, depending on the structure size and position, some zeros are changed in the matrix. These changes depend on the topology of magnetic structure (here defined by the helicities along x and y). For example, if we have a sample of size (4×4) with an ASK in the position $(3, 2)$, the mask will have the value 2 in $(3, 2)$ and 0 elsewhere. The masks for the SK and None remain unchanged at value 0.

To train the network, a new database is generated. The centres of the structures are placed every 30 points along x and y, i.e. on 5 different positions along each direction. The size of the structure is set to an element $\in \{6, 16, 26, 36, 46\}$. Then, the helicities along x and y are set to -1, 0 or 1. This results in a generation of 2 SK, 2 ASK and 5 None at each position for each structure size. In total, 967 samples are generated with corresponding masks.

After the generation of the databases, changes are also made in the pre-treatment to include the masks. The masks are stacked as 3D matrices called tensors in the Tensorflow library. The inclusion of these in the pre-treatment are requiring a lot of operations and are therefore optimised via a procedure called pipeline, as explained in Subsection 3.2.3. The resulting procedure then only takes around 100ms for 32 samples (magnetic textures + the masks).

5.5 The first outputs of the UNet

The network is initialised with the weights already calibrated for both the downsampler and the upsampler. Hence, some predictions can already be made.

Figure 5.9 contains four samples with their predictions. The first column corresponds to the re-normalised magnetic moments. The magnetic moments have values between $[-1, 1]$ and will be referred to as pixels in the following. To plot the structure in an efficient way with Tensorflow, we re-normalise the values between $[0, 1]$. This allows easy representation and clear distinction between the background and the structure. The next two columns correspond to the masks of SK and ASK. The background masks always have 0 values. Thus, no representation is needed. The last column is the prediction of the network before training.

Let's start with panel (a), a texture that looks like a skyrmion, but is actually topologically trivial, is placed in the top of the simulation supercell. The masks of a SK and an ASK are containing only 0 (panel b and c, respectively). The predictions of Unet is then represented in panel (d). The image shows a series of green and red vertical lines. The green colour corresponds to label 1 (SK) and the red colour corresponds to label 2 (ASK). We would have expected only labels 0 (None) as the magnetic texture is not topologically protected. The discrepancy between the prediction and the input magnetic texture originates from the fact that we have not taught the NN yet. Looking a little more carefully, one can catch a glimpse of a sketch of structure. Location is indicated with the blue arrows on panel (d). However, the results should have been a black square.

Let's now look at the second sample. We increased the difficulty as we put a magnetic texture that resembles a skyrmion at the edge of the supercell. Both SK and ASK masks are set to 0 (panel f and g). The prediction of the Unet is then similar to the previous example showing green and red vertical lines. The prediction shows again features, this time in the corners of the image. The corners are slightly denser in green, except for the lower bottom corner, which is a bit denser in red.

The same kind of description can be made for the last two samples although they are showing skyrmion for sample 3 and antiskyrmions for sample 4.

These four predictions show that the shapes of the structures are emerging, but the classification is incorrect. This can simply be explained by the fact that the network has not been trained for the chosen categories yet.

Predictions of four samples after training on 50 epochs are represented in Figure 5.10. The figures shows only black pixels in the last columns (Predictions). Let's start with sample 1 comprising panel a, b, c and d. As for Figure 5.9 panel a, the magnetic texture resembles a topologically protected magnetic texture although it is trivial. In that case, both ASK and SK masks are set to 0. The result of the prediction shown in panel (d) is correctly determined with 'none' for each pixel. Therefore, the prediction in that case is a success and we did not manage to fool the NN.

The same holds for sample 2. The texture is now placed in the centre and is topologically trivial as shown panel (e). Both SK and ASK masks are set to 0 for none and the result of the prediction is indeed a 0 for each pixel as shown in panel (h).

The third sample is different from the first two. In this one, a topologically protected structure is placed on the top of the supercell and comes back due to periodic boundary conditions by the

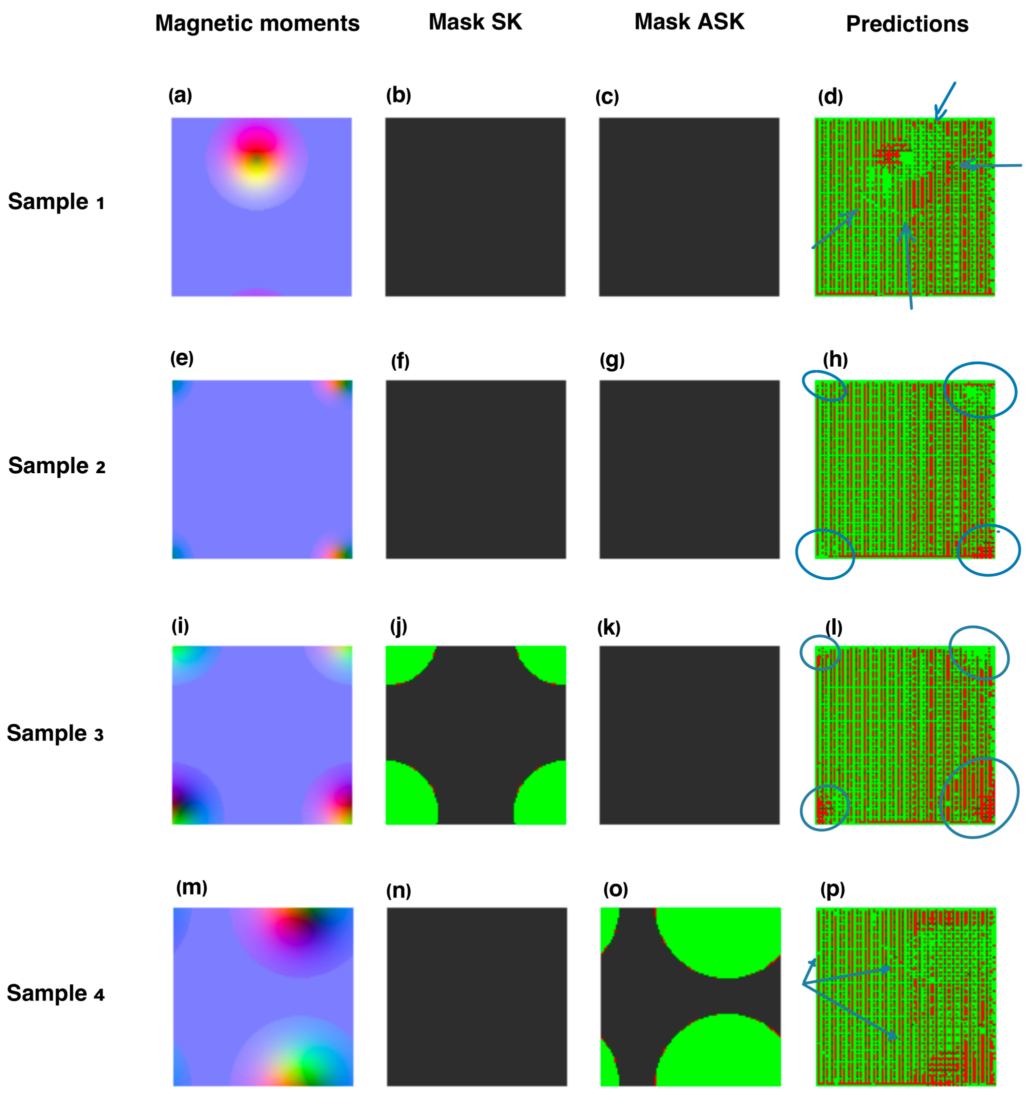


Figure 5.9: Pre-training UNet predictions - Representation of four samples: two without repertoried structure, one with a SK and one with an ASK. The first columns correspond to the re-normalised magnetic moments. The following two columns correspond to the masks of SK and ASK. The last column is the prediction of the network before training.

bottom. The magnetic structure is a SK. Therefore, panel (j) shows the mask that the NN should predict. And, as only one type of magnetic texture is present in a sample, panel (k) is black. The expected result (j) is not entirely the same as (l), the one predicted by the network. The ferromagnetic background of the structure is correctly predicted. Indeed, it is not a structure, we expect from the network a classification of these pixels as None. However, the network is not able to predict the pixels belonging to the SK. Overall, more than half of the pixels are correctly predicted.

The last sample (4) presented, contrarily to the third one, contains an ASK. Hence, the SK mask (n) is black and the ASK one (o) contains coloured pixels corresponding to the magnetic texture (m). As for the third sample, the ferromagnetic background is correctly predicted but not the pixels belonging to the topological structure. This time, the structure is smaller than the last one. Therefore, a large majority of the pixels are correctly predicted.

The results presented are the same as after a single epoch. We observed that the network correctly predicted the ferromagnetic background of all the samples. Unfortunately, the network does not correctly classify the pixels belonging to classified topological texture yet. This should not be the case. However, even if it may seem presumptuous, the network is quite good. Indeed, the network is learning to classify as well as possible. In each sample containing a SK or an ASK, the structure occupies a small area. Additionally, the database contains five-ninths of 'None'. By proceeding in this manner, the network correctly classifies a large majority of pixels (around 80%): all the samples without structures are perfectly classified, and those containing a SK or an ASK are mostly well classified. From these initial results, we suspect a bias problem in the database. The proportion of pixels in the LDB should be changed.

Throughout this chapter, we performed fine-tuning and transfer learning to create a UNet for the recognition of topological structures. We adapted the database generation to the appropriate size for MobileNetV2 layers and labelled all the pixels. Pre-training predictions had been made. These predictions show that features are already extracted by the network. Then, a first training was performed with the generated database. The results seem to indicate a bias in the LDB. Indeed, the network is focusing on the background rather than on the SK and ASK. Further investigations will be necessary. With an accuracy of 80% already reached, the development of this network is promising for the analysis of dynamics of topological magnetic texture.

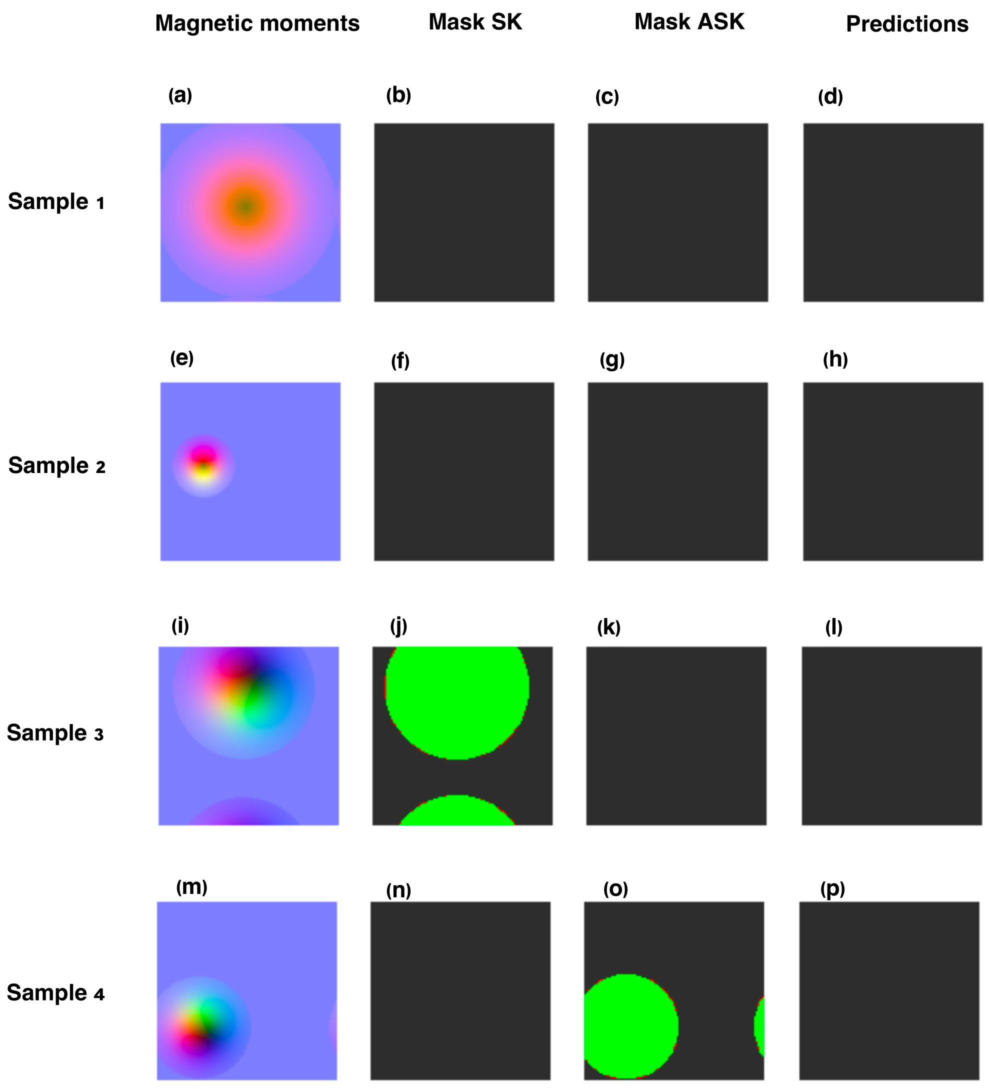


Figure 5.10: Post-training UNet predictions: Representation of four samples: two without repertoried structure, one with a SK and one with an ASK. The first columns correspond to the renormalised magnetic moments. The following two columns correspond to the masks of SK and ASK. The last column is the prediction of the network after training.

Additional information

Regarding memory space

For your information, here are some values for the required space for the databases:

- The databases of size 100×100:
  - LDB, 8000 samples = 62.5GB
  - TDB, 275 samples = 2.2GB
- The databases of size 128×128 with masks:
  - 960 samples = 12.5 GB
- The weights of a single training:
  - RNN: weights of 500 epochs= 182GB
  - CNN: weights of 250 epochs = 5.6GB

Regarding the computers

The generation of the databases were realised on a server of the RU dedicated for Matjes with GPU acceleration. It was chosen to work with TensorFlow for its ability to use GPU acceleration as well. However, the training sessions could not be conducted on the research unit workstation. We recommend installing Anaconda on Ubuntu with caution. Instead, the trainings were performed on a Lenovo ThinkBook 15 G2 ARE with AMD Ryzen 7 4700U. No GPU acceleration was possible on this computer.

Additional figures

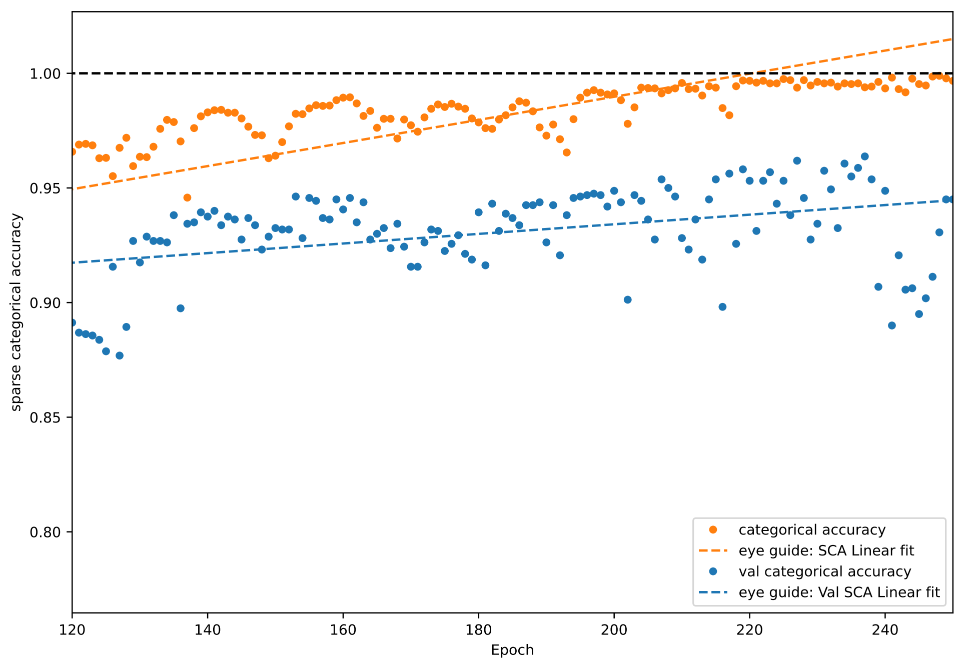


Figure 11: Zoom of Figure 3.9. The accuracy ends at 0.99. The validation accuracy is scattered but similarities between the two scatter plots are visible. The validation accuracy stays around 5% under the accuracy.

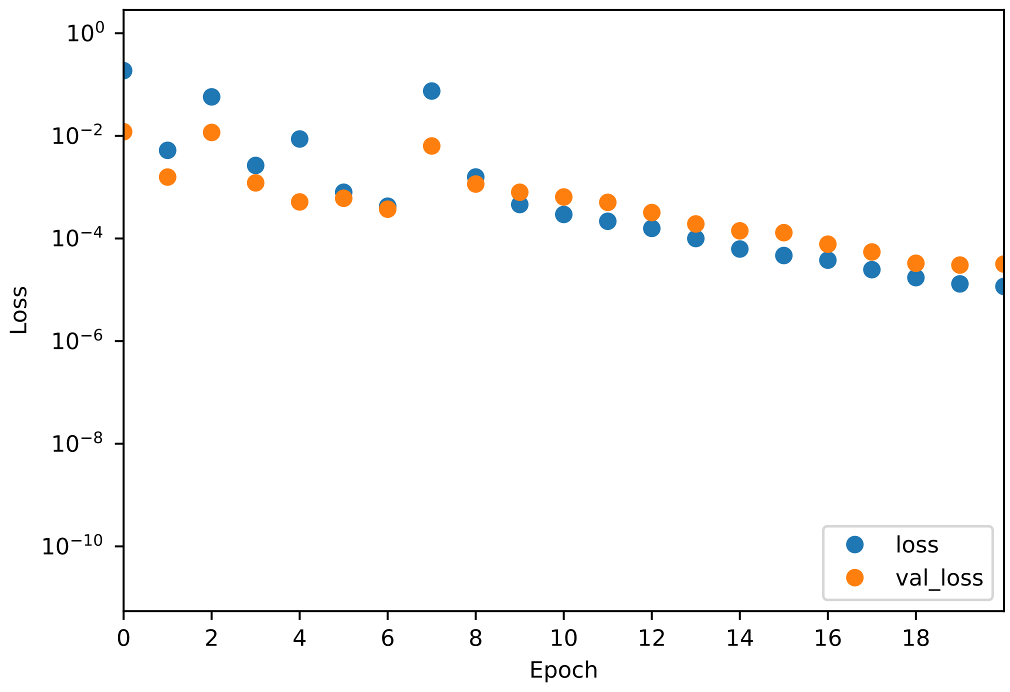


Figure 12: Plot of the losses on training and validation subsets from epochs 0 to 20 during training on the LDB. The orange dots are the loss on the training subset and the blue ones represent the loss calculated on the validation subset. The network shows very low losses, sparse at first. After epoch 8, the two losses continuously decrease.

Bibliography

[1] A. G. Grushin, *Introduction to topological phases in condensed matter* (n.d.), lecture notes.

[2] N. S. Kiselev, A. N. Bogdanov, R. Schäfer, and U. K. Rößler, Journal of Physics D: Applied Physics **44**, 392001 (2011), URL `https://dx.doi.org/10.1088/0022-3727/44/39/392001`.

[3] A. Fert, V. Cros, and J. Sampaio, Nature Nanotechnology **8**, 152 (2013).

[4] S. Meyer, Ph.D. thesis, Universität zu Kiel (2020), URL `https://macau.uni-kiel.de/receive/macau_mods_00000764?lang=en`.

[5] X. Zhang, G. P. Zhao, H. Fangohr, J. P. Liu, W. X. Xia, J. Xia, and F. J. Morvan, Scientific Reports **5**, 7643 (2015), ISSN 2045-2322, URL `https://doi.org/10.1038/srep07643`.

[6] C. Back, V. Cros, H. Ebert, K. Everschor-Sitte, A. Fert, M. Garst, T. Ma, S. Mankovsky, T. L. Monchesky, M. Mostovoy, et al., Journal of Physics D: Applied Physics **53** (2020), URL `https://dx.doi.org/10.1088/1361-6463/ab8418`.

[7] M. Mignolet, Master's thesis, Université de Liège, Liège, Belgique (2022), URL `https://matheo.uliege.be/handle/2268.2/16289`.

[8] I. A. Iakovlev, O. M. Sotnikov, and V. V. Mazurenko, Phys. Rev. B **98**, 174411 (2018), URL `https://link.aps.org/doi/10.1103/PhysRevB.98.174411`.

[9] I. Labrie-Boulay, T. B. Winkler, D. Franzen, A. Romanova, H. Fangohr, and M. Kläui, Phys. Rev. Appl. **21**, 014014 (2024), URL `https://link.aps.org/doi/10.1103/PhysRevApplied.21.014014`.

[10] D. Feng, Z. Guan, X. Wu, Y. Wu, and C. Song, Phys. Rev. Appl. **21**, 034009 (2024), URL `https://link.aps.org/doi/10.1103/PhysRevApplied.21.034009`.

[11] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach (4th Edition)* (Pearson, 2020), ISBN 9781292401133, URL `http://aima.cs.berkeley.edu/`.

[12] P. M. Buhl, L. Desplat, M. Boettcher, S. Meyer, and B. Dupe, *Matjes* (2024), URL `https://github.com/bertdupe/Matjes`.

[13] A. N. Bogdanov and D. A. Yablonskii, Zh. Eksp. Teor. Fiz. **95**, 178 (1989), ISSN 0044-4510, URL `http://www.jetp.ac.ru/cgi-bin/e/index/e/68/1/p101?a=list`.

[14] A. N. Bogdanov and U. K. Rössler, Physical review letters **87**, 037203 (2001), ISSN 0031-9007, URL `http://link.aps.org/doi/10.1103/PhysRevLett.87.037203`.

[15] N. Nagaosa and Y. Tokura, Nature Nanotechnology **8**, 899 (2013), URL `https://doi.org/10.1038/nnano.2013.243`.

[16] W. S. McCulloch and W. Pitts, The bulletin of mathematical biophysics **5**, 115 (1943), URL `https://doi.org/10.1007/BF02478259`.

[17] M. Sandeep, K. Tiprak, S. Kaewunruen, P. Pheinsusom, and W. Pansuk, Structures **47**, 1196 (2023), ISSN 2352-0124, URL `https://www.sciencedirect.com/science/article/pii/S2352012422011791`.

[18] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, et al., IEEE Signal Processing Magazine **29**, 82 (2012).

[19] A. Krizhevsky, I. Sutskever, and G. E. Hinton, Commun. ACM **60**, 84–90 (2017), ISSN 0001-0782, URL `https://doi.org/10.1145/3065386`.

[20] B. Xu, N. Wang, T. Chen, and M. Li, ArXiv **abs/1505.00853** (2015), URL `https://api.semanticscholar.org/CorpusID:14083350`.

[21] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning* (MIT Press, 2016), `http://www.deeplearningbook.org`.

[22] E. Gordon-Rodriguez, G. Loaiza-Ganem, and J. Cunningham, in *Proceedings of the 37th International Conference on Machine Learning*, edited by H. D. III and A. Singh (PMLR, 2020), vol. 119 of *Proceedings of Machine Learning Research*, pp. 3637–3647, URL `https://proceedings.mlr.press/v119/gordon-rodriguez20a.html`.

[23] E. Barsoum, C. Zhang, C. C. Ferrer, and Z. Zhang, in *Proceedings of the 18th ACM international conference on multimodal interaction* (2016), pp. 279–283.

[24] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, Nature **323**, 533 (1986), URL `https://doi.org/10.1038/323533a0`.

[25] M. A. Nielsen, *Neural Networks and Deep Learning* (Determination Press, 2015), URL `http://neuralnetworksanddeeplearning.com/chap2.html`.

[26] D. Kingma and J. Ba, International Conference on Learning Representations (2014).

[27] Y. LeCun, B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard, and L. Jackel, in *Advances in Neural Information Processing Systems*, edited by D. Touretzky (Morgan-Kaufmann, 1989), vol. 2, URL `https://proceedings.neurips.cc/paper_files/paper/1989/file/53c3bce66e43be4f209556518c2fcb54-Paper.pdf`.

[28] *Cs231n: Deep learning for computer vision* (2024), https://cs231n.github.io/convolutional-networks/, URL `https://cs231n.github.io/convolutional-networks/`.

[29] O. Ronneberger, P. Fischer, and T. Brox, in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, edited by N. Navab, J. Hornegger, W. M. Wells, and A. F. Frangi (Springer International Publishing, Cham, 2015), pp. 234–241, ISBN 978-3-319-24574-4.

[30] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, *Mobilenetv2: Inverted residuals and linear bottlenecks* (2019), `1801.04381`, URL `https://arxiv.org/abs/1801.04381`.

[31] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, *Image-to-image translation with conditional adversarial networks* (2018), `1611.07004`, URL `https://arxiv.org/abs/1611.07004`.