UNIVERZA NA PRIMORSKEM
FAKULTETA ZA MATEMATIKO, NARAVOSLOVJE IN
INFORMACIJSKE TEHNOLOGIJE

Master's thesis
(Magistrsko delo)

Neural Network Based Classification of Brain Lesions

(Klasifikacija možganskih lezij z umetno nevronsko mrežo)

Ime in priimek: *Nedim ŠIŠIĆ*
Študijski program: *Računalništvo in informatika, 2. stopnja*
Mentor: *doc. dr. Peter Rogelj*

Koper, september 2022

Ključna dokumentacijska informacija

Ime in PRIIMEK: Nedim ŠIŠIĆ
Naslov magistrskega dela: Klasifikacija možganskih lezij z umetno nevronsko mrežo

Kraj: Koper
Leto: 2022

Število listov: 82 Število slik: 32 Število tabel: 4
Število referenc: 68
Mentor: doc. dr. Peter Rogelj

Ključne besede: računalniški vid, nevro-slikanje, konvolucijske nevronske mreže, multipla skleroza

Izvleček: Možganske lezije so področja poškodb tkiva ali bolezni v možganih. Tehnike slikanja možganov, kot je MRI, omogočajo diagnozo ali prognozo različnih takšnih poškodb ali stanj. Pomemben del teh analiz je razvrščanje možganskih lezij v vnaprej poznane tipe z različnimi lastnostmi. V tem delu se osredotočamo na razvrstitev lezij v dve skupini glede na status remielinizacije lezij, ki odraža napredek celjenja lezij pri multipli sklerozi. Nevronske mreže so se doslej izkazale kot uspešne na različnih področjih računalniškega vida, vključno z obdelavo možganskih slik. V tem magistrskem delu smo se posvetili razvoju konvolucijske nevronske mreže, ki samodejno razvršča MRI slike lezij pri bolnikih z multiplo sklerozo. Vhodni slikovni podatki sestojijo iz treh slikovnih kanalov tridimenzionalnih (volumetričnih) slik: MRI FLAIR, MRI QSM in kanala maske, ki označuje lokacijo lezije. V nalogi so predstavljene teoretične osnove konvolucijskih nevronskih mrež in opisano medicinsko ozadnje našega problema. Opisani so postopki priprave vhodnih podatkov za doseganje večjega uspeha pri učenju nevronskih mrež, arhitektura in parametri učenja nevronske mreže ter podani rezultati modela razvrščanja. Natančneje, predstavljamo natančnost razvrščevalnika pri uporabi različnih kombinacij slikovnih kanalov. Rezultati kažejo, da je pristop z uporabo konvolucijske nevronske mreže zelo obetaven za razvrščanje lezij multiple skleroze, ko se uporablja kanal QSM MRI. Za uporabo razvrščevalnika v klinični praksi je potreben nadaljnji razvoj postopka, da bo mogoče tudi prepoznavanje tistih lezij, ki jih radiologi ne uspejo razvrstiti.

Key document information

Name and SURNAME: Nedim ŠIŠIĆ
Title of the thesis: Neural Network Based Classification of Brain Lesions

Place: Koper
Year: 2022

Number of pages: 82 Number of figures: 32 Number of tables: 4
Number of references: 68
Mentor: Assist. Prof. Peter Rogelj, PhD

Keywords: computer vision, neuroimaging, convolutional neural networks, multiple sclerosis

Abstract: Brain lesions are areas of tissue injury or disease within the brain. Brain-imaging techniques, such as MRI, produce images of the brain used for diagnosis or prognosis of a variety of injuries or conditions. An important part of such analysis is classification of brain lesions into known lesion types. We focus on classification of lesions into two groups regarding their remyelination status, which reflects the lesions' healing progress in multiple sclerosis (MS). Recently, neural networks have been used to great success in different computer vision domains, including brain image processing. In this thesis, we design a convolutional neural network model that automatically classifies MRI images in multiple sclerosis patients. The image dataset consists of three-dimensional (volumetric) images with three channels: FLAIR MRI, QSM MRI, and a mask channel indicating lesion location. We present the theoretical foundations of convolutional neural networks, and motivate our problem by providing its medical background. We describe data manipulation techniques we use for making the dataset more suitable for network training, present the network architecture and learning parameters, and show the results of the model. Specifically, we present the accuracies of the network when trained on different combinations of the three image channels. The results demonstrate that the convolutional neural network approach is highly promising for the problem exactly when the QSM MRI channel is used. Further work, especially one concerning recognizing lesions that radiologists cannot classify, is necessary for applying the network in clinical practice.

List of Contents

List of Tables

List of Figures

List of Abbreviations

i.e.	that is
e.g.	for example
MS	multiple sclerosis
MRI	Magnetic Resonance Imaging
QSM	Quantitative Susceptibility Mapping
$FLAIR$	fluid attenuated inversion recovery
$MP2RAGE$	magnetization-prepared 2 rapid acquisition gradient echoe
MLP	multilayer perceptron
FC	fully connected
CNN	convolutional neural network

Acknowledgments

I would like to express my greatest gratitude to my mentor Peter Rogelj for his guidance, advice, time, and patience during the research and the writing of this thesis. I would like to thank the Translational Imaging in Neurology (ThINk) group, especially Muhamed Baraković, for help in defining the research goals and providing guidance, data, and resources.

I am very grateful to UP FAMNIT for caring for its students and providing them with great opportunities.

Finally, I am deeply indebted to my parents for their support and motivation throughout my education.

1 INTRODUCTION

In this chapter, we first give an introduction to our problem of classifying magnetic resonance imaging images in multiple sclerosis patients, and the methods we use throughout the work, in **Section 1.1**. We then provide an overview of previous work related to deep learning in MRI imaging in **Section 1.2**. In section **Section 1.3**, we present the structure of the thesis.

1.1 INTRODUCTION

Medical imaging produces images of the interior of the body in order to diagnose, monitor, and treat medical conditions. Magnetic resonance imaging (MRI) is a technique in medical imaging where strong magnetic fields and radio waves are used to generate images of the organs in the body. MRI is widely used in medical practice for imaging the central nervous system and assisting the diagnosis and treatment of neurological disorders. One such disorder is *multiple sclerosis* (MS). MS is a demyelinating disease, in which the insulating myelin covers of nerve cells in the brain and spinal cord are damaged. MS is rarely fatal but results in a range of symptoms. An estimated 2 million people worldwide currently have the disease [56]. Recently, Quantitative Susceptibility Mapping (QSM), a type of MRI, has been shown to allow classification of MS lesions – areas of tissue abnormality – into demyelinating and remyelinating types [57]. In remyelinating lesions, a repair process called remyelination attempts to create new myelin sheaths on the surface of demyelinating axons; in demyelinating lesions, the process does not occur. Remyelinating lesions indicate the state of the patient is expected to improve, while demyelinating lesions indicate that the state is expected to worsen. Classifying MS lesions into the two types based on QSM MRI may therefore be very beneficial in diagnosis of MS patients.

Medical images are typically manually processed by medical experts, which can prove to be a time consuming and laborious task. For that reason, computer-aided systems are frequently used to reduce the time and energy required for image processing and even improve diagnostic accuracy. Specifically, computer vision and image processing are used for processing, classification and segmentation of medical images; a review of such techniques can be found in works by Gao et al. [21] and Yanase et al. [66].

Neural networks have had a large impact on computer vision, image processing, and the related fields. Neural networks are machine learning models that can be applied to numerous problems, such as natural language processing, computer vision, protein folding, playing games etc. A specific type of neural network, the convolutional neural network (CNN), has revolutionized many subfields of computer vision and image processing by achieving state-of-the-art results in problems such as image classification, image segmentation, and object detection. While CNNs have achieved remarkable results in medical imaging as well, the field often presents challenges to their use. In particular, neural networks are machine learning models that require large amounts of standardized data to be successfully trained, and in medical imaging such datasets may frequently not be available. In medical imaging datasets, the number of sample images may be small, the labels associated with the images may be sparse, or the image categories may be heterogeneous and imbalanced. Medical imaging thus often requires additional efforts, commonly regarding data augmentation, when neural networks are to be used. A review of neural networks in medical imaging is given in [41].

In this work, we present a convolutional neural network for a novel problem in classification of multiple sclerosis MRI images. The dataset of images we work on is the one given by [57]. Each image is three-dimensional (volumetric), of size $35 \times 35 \times 35$, and consisting of three channels: FLAIR MRI, QSM MRI, and a mask channel indicating lesion location. The CNN classifies the lesions as either remyelinating or demyelinating. We train the network using different combinations of the three image channels, and compare the results for each combination. The results demonstrate that convolutional neural networks are a promising tool for this problem. Expanding the work may enable the use of a model such as ours in clinical practice, helping medical experts in MS diagnosis by easing the burdensome task of image processing and improving diagnosis accuracy.

1.2 PREVIOUS WORK

Computer vision and image processing can be used in numerous tasks in the medical imaging workflow. On one end are the tasks closer to signal processing and image acquisition, which include image reconstruction and restoration, multimodal image registration, and others. The tasks on the other end deal with extracting valuable medical information from images and include image segmentation, disease detection and prediction, etc. An overviews of computer vision and image processing in medical imaging is given by Gao et al. [21], while a systematic survey of computer-aided systems used for medical diagnosis is given by Yanase et al. [66].

In this work, we are primarily interested in using deep learning techniques for MRI

image classification. In the last decade, deep learning techniques have become the standard approach to a wide variety of computer vision and image processing problems, and have shown enormous potential for dealing with such problems in healthcare. While the use of the techniques in medical imaging is still in its early stages, a large number of works in the domain has been published; for example, a survey by Litjens et al. [41] on deep learning in medical image analysis gives a list of more than 200 larger contributions that appeared in 2016 alone. There are multiple thorough reviews of deep learning in both medicine and biology [11], in medical imaging in general [41], or in specific areas such as neuroimaging [60], brain segmentation [1], oncology [51], etc.

Specifically for MRI, a comprehensive overview is given by the work of Lundervold et al. [44], which surveys the use of deep learning in the tasks of MRI data acquisition and image reconstruction, QSM and MR fingerprinting, image restoration, image super-resolution, image synthesis, image registration, image segmentation, diagnosis and prediction, and content-based image retrieval. The work also lists valuable resources such as tutorials, code repositories, and public benchmarks and challenges.

A popular technique in deep learning is *transfer learning* (or *pre-training*), where weights from a neural network trained on a large dataset are imported to a neural network that is to be trained on another, typically much smaller, dataset. The technique allows the later network to use information that is learned by the former one and might prove to be applicable to both datasets. In our case, however, we did not succeed in finding a model that would provide us with weights applicable to our problem. This is likely due to the uncommonness of image dimensions in our dataset. More specifically, the images have three spatial dimensions and three channels, while convolutional neural networks more frequently operate on two-dimensional images or three-dimensional images with one channel.

1.3 THESIS STRUCTURE

The thesis is organized as follows. We give a theoretical background of the model we design, concerning machine learning, neural networks, and especially convolutional neural networks, in **Chapter 2**. In **Chapter 3**, we present the medical background and the definition of our problem, and give a summary of related works. We present the methods we use to solve the problem, which concern data manipulation, the choice of CNN architecture and the learning parameters, in **Chapter 4**. We present and discuss the results of our model in **Chapter 5**. Finally, we give a conclusion of the work in **Chapter 6**.

2 THEORETICAL BACKGROUND

In this chapter, we give a theoretical background of the concepts used in the work. Convolutional neural networks are machine learning models, we therefore first present the principles of machine learning in **Section 2.1**. In **Section 2.3**, we describe neural networks, one of the most powerful learning models. In **Section 2.4**, we present convolutional neural networks, a specific family of neural networks frequently used in computer vision and image processing.

There are numerous works that provide an introduction to and give an overview of machine learning, including the ones by Bishop [8] and Hastie [25]. Detailed introductions to neural networks can be found in the works by [68], [23], and [49]. An overview of the history of important ideas in neural networks is given by [61], while a recent survey of this fast-growing field is given by [3]. A detailed introduction to convolutional neural networks can be found in the CS231 Stanford course [13], while a review of the field is given by [55]

2.1 MACHINE LEARNING

Machine learning is broadly concerned with the study and construction of algorithms that may learn from and make predictions on data. As the field is broad, we only describe the terms relevant to the remainder of the work. We consider our classification problem as belonging to *supervised learning*, a subfield of machine learning where a function is inferred from labeled training data. Let $f : X \to Y$ be a function we want a model to learn. In supervised learning, the model learns a function $g : X \to Y$ from a set of pairs $(x, f(x))$ where $x \in X_1 \subset X$, called the *training set*, where g minimizes a *loss function* (or *cost function*) - the better g approximates f, the smaller the loss function. Ideally, the model will learn f. As we want the model to generalize beyond the training set and onto the entire set X, we test the model on a set of pairs $(x, f(x))$ where $x \in X_2 \subset X$ and $X_1 \cap X_2 = \emptyset$, called the *test set*.

The architecture of a model, specified by its *hyperparameters* is decided before learning. The model outputs values based on its input and on the values of its variables, called *parameters*. A learning algorithm is peformed on the model using the training set, so that the parameters are adjusted to best fit the training examples; this process is called *training*. Typically, a larger number of parameters increases the expresionall

power of the model. If the parameters are too few, the model cannot adequately capture the underlying structure of the data, a problem called *underfitting*. On the other hand, when the parameters are too numerous, the model may be too complex and correspond too closely to the training set, hence failing to generalize to the test set - a problem called *overfitting*. The simplification of a model in order to prevent overfitting is called *regularization*. Balancing the errors of underfitting and overfitting is often very important in constructing an appropriate model.

2.2 NEURAL NETWORKS

Artificial neural networks are one of the most powerful learning models. They are loosely inspired by biological neural networks and can be used for numerous tasks. Convolutional neural networks, the model we use for image classification in this thesis, are a special type of neural networks.

2.2.1 The Neuron

The basic building block of a neural network is the (artificial) *neuron*. The artificial neuron is inspired by biological neurons, the most important cells in the brain [32]. Biological neurons form connections with each other and communicate by "firing" electrical signals. The *rate* of firing specifies the extent of activation of a neuron. Depending on the rate of firing of its incoming signals and the "strength" of its connections, a neuron decides if and how strongly it fires. In the brain, learning happens by neurons forming connections to other neurons and the strengths of these connections adapting over time.

Artificial neurons, introduced first by McCulloch and Pitts [46] in 1943, are loosely inspired by biological ones. We refer to an artificial neuron simply as *neuron* in the remainder of the work. A neuron has the following structure. It takes a vector $\mathbf{x}$ of n scalar inputs $x_1, x_2, ..., x_n$. A vector $\mathbf{w}$ of n scalar weights $w_1, w_2, ..., w_n$ is assigned to the inputs, where weight w_i specifies the "importance" of input x_i for the neuron - simulating the strength of a connection between two biological neurons. The artificial neuron outputs the activation value y as in definition 2.1.

Definition 2.1. For inputs $x_1, x_2, ..., x_n$ and weights $w_1, w_2, ..., w_n$, a *neuron* outputs the *activation* value y:

$$y = \sigma(\sum_{i=1}^{n} x_i w_i + b)$$

where $\sum_{i=1}^{n} x_i w_i$ is the *weighted sum* of the inputs, b is the *bias*, and σ is an *activation function*.

A visual representation of the neuron is shown if **Figure 1**. The weights and bias of a neuron are its parameters, as they are modified when the neuron is used as part of a learning model. Note that the weighted sum can be expressed by the vector product **wx**. To obtain a simpler notation, the bias is sometimes in literature considered to be the $(n+1)$-th weight of the neuron and the vector of inputs x is expanded into $(x_1, x_2, ..., x_n, 1)$, so that the activation function function is applied directly on the product **wx**. In the remainder of the work, we do not use this simplified notation unless explicitly specified.

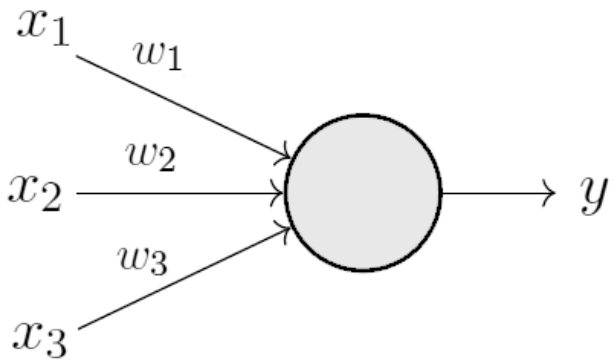

Figure 1: A neuron.

Different choices of the activation function may be used. The function is most often non-polynomial, for reasons we discuss in **Section 2.3**. One of the first activation functions suggested in literature is a simple binary step function similar to the Heaviside step function $\mathcal{H}$:

$$\mathcal{H} = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases}$$

The Heaviside step function is shown in **Figure 2**. In the neuron model, a *threshold* value θ such that $\theta = -b$ is sometimes used instead of the bias. Now, in the neuron model of McCulloch and Pitts and with the neuron using the Heaviside step function we have $y = \mathcal{H}(\mathbf{wx} - \theta)$. Such a neuron "fires" (outputs 1) if the weighted sum of inputs exceeds or equals the threshold value, and it otherwise outputs 0. We note that the artificial neuron simulates only some functionalities of the biological neuron, the biological one being much more complex [42].

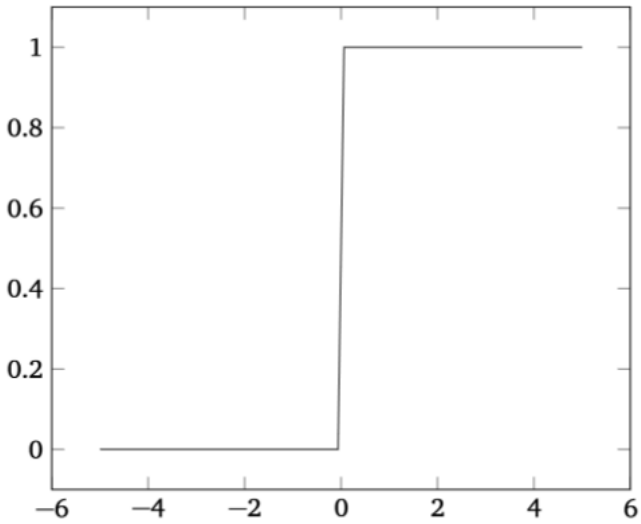

Figure 2: Heaviside step function $\mathcal{H}$.

We geometrically interpret the McCulloch and Pitts neuron using the Heaviside step function as follows. The input vector x is a point in space $\mathbb{R}^n$. The neuron splits the input space into two halves, where one is assigned an output value of 0 and the other an output value of 1. The boundary between the two halves is an $(n-1)$-dimensional hyperplane perpendicular to the weights vector $\mathbf{w}$ and with distance θ from the origin 0^n of the input space. As such, the McCulloch and Pitts neuron is a *linear separator* of its input space $\mathbb{R}^n$, because the hyperplane is a solution to the linear equation $\mathbf{wx} - \theta = 0$.

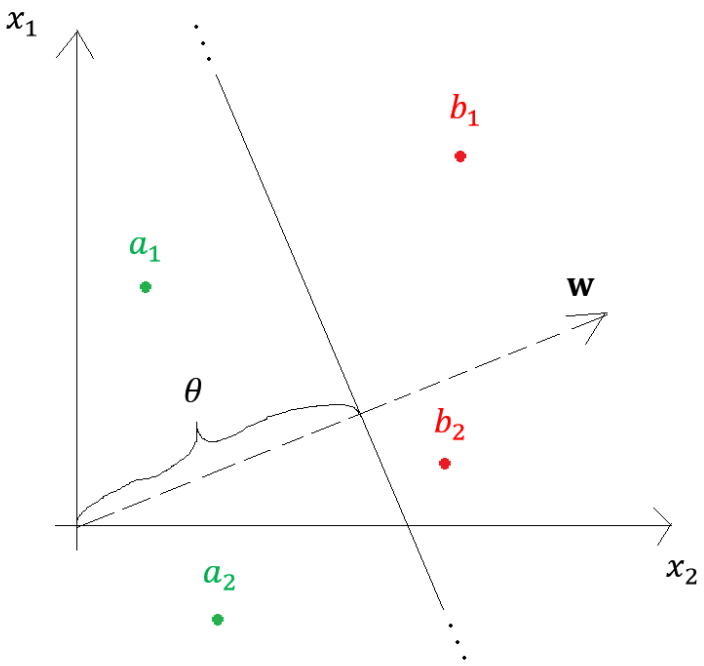

Figure 3: A neuron as a linear separator of sets A and B.

We say that two sets of points are *linearly separable* if they can be separated by a hyperplane, called a *separating hyperplane*. The usefulness of a neuron is in the following: for a set $X = A \cup B$ of points, where A and B are linearly separable, a neuron can express the separating hyperplane of A and B, allowing us to correctly classify each point into either A or B, depending on which side of the hyperplane the point lies. The hyperplane represents the "decision boundary" of the neuron. A geometric interpretation of a neuron with two inputs x_1 and x_2 that separates sets $A = \{a_1, a_2\}$ and $B = \{b_1, b_2\}$ is shown in **Figure 3**. In the remainder of this work, we will use the bias b instead of the threshold θ.

2.2.1.1 Neurons as Learners

By allowing a neuron's parameters (the weights and bias) to change, we may use it as a learning algorithm. Let $X = A \cup B$ be the set of input points, where A and B are linearly separable sets. We label each point in A by 1, and each point in B by -1; we denote the label of a point $\mathbf{x}$ by $l(\mathbf{x})$. We let the neuron classify a point $\mathbf{x}$ according to its parameters using the function $classify$:

$$classify_{\mathbf{w},b}(\mathbf{x}) = \begin{cases} 1, & \mathcal{H}(\mathbf{wx} + b) = 1 \\ -1, & \mathcal{H}(\mathbf{wx} + b) = 0 \end{cases},$$

to obtain $classify_{\mathbf{w},b}(\mathbf{x})$. Now, we want the neuron to learn the parameters so that for each point $\mathbf{x} \in X$ we have $classify_{\mathbf{w},b}(\mathbf{x}) = l(\mathbf{x})$, i.e., we want the neuron to correctly classify each point into set A or set B.

Algorithm 1 The Perceptron algorithm

```
Input: set of points X, label function l
Output: weights w, bias b
w ← 0
b ← 0
while some inputs are misclassified do
    for x in X do:
        if (classify(x) = l(x)) then
            w ← w + l(x)x
            b ← b + l(x)
        end if
    end for
end while
```

One way of achieving this is by using the classic *Perceptron* algorithm [59], proposed in 1958. The algorithm is shown by **Algorithm 1**, and works as follows. As long as

there exist inputs that the neuron classifies incorrectly, the algorithm will go through each input point $\mathbf{x}$ and calculate the activation of the neuron for $\mathbf{x}$. If the activation indicates the neuron misclassifies the point, an update of the neuron's parameters is performed. The update is such that it brings the neuron closer to classifying the point $\mathbf{x}$ correctly, as we show by the following. Assume the neuron has parameters $\mathbf{w}$ and b and misclassfies a point $\mathbf{x}$, where $l(\mathbf{x}) = 1$ (meaning $classify_{\mathbf{w},b}(\mathbf{x}) = -1$). We denote the value $\mathbf{wx} + b$ by a, so that the activation of the neuron is $y = \mathcal{H}(a)$. After the parameters are updated to $\mathbf{w}$' and b', the new activation is:

$$\begin{aligned} y' &= \mathcal{H}(\mathbf{w}'\mathbf{x} + b') \\ &= \mathcal{H}((\mathbf{w} + \mathbf{x})\mathbf{x} + b + 1) \\ &= \mathcal{H}(\mathbf{wx} + b + (\mathbf{x})^2 + 1) \\ &= \mathcal{H}(a + \|\mathbf{x}\|^2 + 1) \\ &= \mathcal{H}(a') \end{aligned}$$

Because $a' = a + \|\mathbf{x}\|^2 + 1 > a$, the value a' is closer to 0 than a is, so we are moving the parameters in the right direction. If the same update would be applied many times consecutively, a' would eventually become greater or equal to 0, and the neuron would correctly classify point $\mathbf{x}$. However, updating the parameters after one misclassified point may make the neuron misclassify some points that it has previously classified correctly. To correctly classify each point in the set, the algorithm keeps iterating through the points, updating the parameters only once each time it misclassifies a point. A proof that the algorithm does eventually converge to a separating hyperplane this way is given in [15].

Another way of learning a neuron's parameters in order to classify the points in set X is to treat the task as an optimization problem. In this approach, we first specify a search space, consisting of candidate solutions to the problem, and a cost function evaulating the suitability of the solutions. We then use an optimization algorithm to attempt to find an optimal solution. This way, we abstract away the concrete details of the neuron, and can use already existing knowledge of optimization problems instead. In our case, the search space consists of different choices of values for the neuron's parameters - the parameters are considered the *variables* of the problem. The cost function evaluates how suitable a choice of parameters is for classifying the points: the more points that are classified correctly, the lower the cost of a choice. An optimal solution is such that there exists no solution with lower cost. We first decide to define the cost C_X function as follows:

$$C_X(\mathbf{w}, b) = \frac{1}{2n} \sum_{\mathbf{x}} \|l(\mathbf{x}) - classify_{\mathbf{w},b}(\mathbf{x})\|^2.$$

where n is the total number of inputs. The value $\|l(\mathbf{x}) - classify_{\mathbf{w},b}(\mathbf{x})\|^2$ represents the error of classifying a point $\mathbf{x}$: the error is 0 if the neuron classifies the point correctly, and 1 otherwise. $C_X(\mathbf{w}, b)$ is the *mean squared error* (MSE) multiplied by a factor of $\frac{1}{2}$. MSE is a function that is often used as a cost function due to its suitable properties for evaluating solutions and applying optimization algorithms. We notice that the cost function is smaller the more points are classified correctly, hence an optimal solution would be the one which yields the lowest cost. The factor of $\frac{1}{2}$ does not effect the order of costs of different solutions, it is applied instead to simplify computing the derivative of the cost function, which is performed by the optimization algorithm we discuss soon. However, we first note that many optimization algorithms assume a differentiable function; because the $classify$ function is not differentiable, neither is our cost function. We make the cost function differentiable by changing the neuron's activation function to the sigmoid function s:

$$s(z) = \frac{1}{1 + e^{-z}},$$

and classifying points according to the activation value instead of using the $classify$ function. A graph of the sigmoid function is shown in **Figure 4**; we notice that s is a continuous approximation of the Heaviside function.

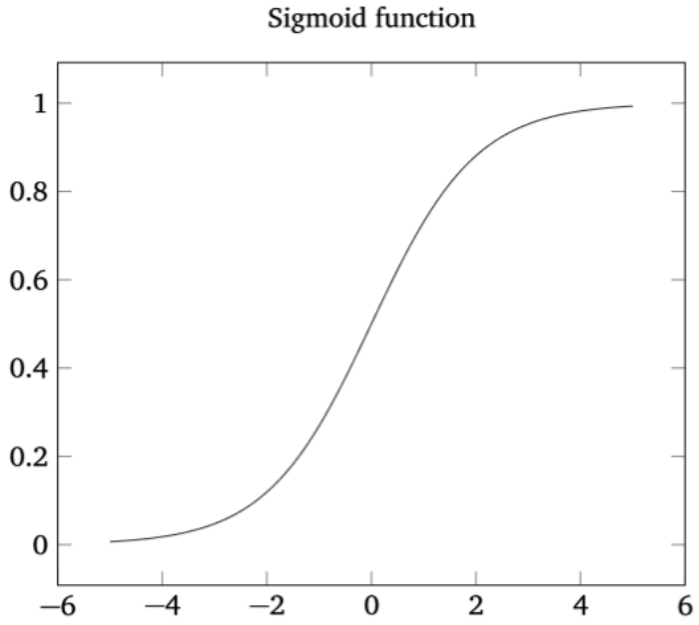

Figure 4: The sigmoid function s.

The new cost function is:

$$C_X(\mathbf{w}, b) = \frac{1}{2n} \sum_{\mathbf{x}} \|l(\mathbf{x}) - s(\mathbf{w}\mathbf{x} + b)\|^2.$$

Because of the continuous nature of the new cost function, an optimization algorithm is able to use the fact that small changes in the parameters result in only small changes to the cost; the differentiability of the cost function allows an algorithm to compute its derivatives. We can decide to interpret the output of a neuron to indicate a positive classification when $s(\mathbf{wx}+b) \geq 0.5$ and a negative classification otherwise. The sigmoid function has a few nice properties regardless of optimization this way, namely that it can represent intermediate values (the ones close to 0.5) and express different degrees of "confidence" for classification. For example, a value of 0.95 could indicate we are more confident in a positive classification than a value of 0.9.

Using the new cost function, we now present the gradient descent optimization algorithm. For variables $v_1, v_2, ..., v_m$ of the problem, the gradient descent algorithm attempts to find a solution (a choice of values of the variables) that minimizes the cost function. It is useful to view the algorithm from a geometric perspective. The cost function is an m-variable continuous function, and each solution is a point in m-dimensional vector space. The algorithm starts at some arbitrary solution $v_1 = (v_{11}, v_{21}, ..., v_{m1})$, with cost C_1. Then, it makes small changes to the solution so that the new solution $v_2 = (v_{12}, v_{22}, ..., v_{m2})$ has smaller cost C_2. From a geometric perspective, the algorithm moves from one point in the vector space to another, in such a way that the value of the function is smaller at the new point. The algorithm continues this process until it reaches a local minimum of the function. In general, the algorithm may then decide to terminate and return the local minimum value, or attempt to find other, possibly smaller, local minimums. In our specific example, the algorithm terminates as soon as it reaches a local minimum.

A non-obvious part of the algorithm is the choice of the new solution $(v_{12}, v_{22}, ..., v_{m2})$ based on the solution $v_1 = (v_{11}, v_{21}, ..., v_{m1})$. Because the cost function is continuous, by making only a small change $\Delta v = v_2 - v_1 = (\Delta v_1, \Delta v_2, ..., \Delta v_m)$, the change ΔC of the cost will be approximately:

$$\Delta C \approx \frac{\partial C}{\partial v_1}\Delta v_1 + \frac{\partial C}{\partial v_2}\Delta v_2 + ... + \frac{\partial C}{\partial v_m}\Delta v_m$$

We denote by Δv the vector of changes in v, i.e., $\Delta v = (\Delta v_1, \Delta v_2, ..., \Delta v_m)^T$, while the *gradient* ∇C of C is the vector of its partial derivatives, $\nabla C = (\frac{\partial C}{\partial v_1}, \frac{\partial C}{\partial v_2}, ..., \frac{\partial C}{\partial v_m})^T$, where T is the *transpose* operation. Note that he gradient specifies the direction and rate of fastest increase of the cost. We now write:

$$\Delta C \approx \nabla C \cdot \Delta v. \tag{2.1}$$

The direction specified by the gradient is opposite to the one of fastest decrease. So, a good choice of Δv is:

$$\Delta v = -\alpha \nabla C,$$

for a small $\alpha > 0$, where α is the *learning rate*. Now $\Delta C \approx -\alpha \|\nabla C\|^2 \leq 0$, so the cost decreases. Furthermore, the direction specified by $-\nabla C$ is the one with the largest decrease of C for a sufficiently small α. The algorithm's update rule is therefore:

$$v_2 = v_1 - \alpha \nabla C.$$

The value of α should be sufficiently small so that equation 2.1 holds, but large enough for the algorithm not to be too slow. In practice, the value of α is often varied throughout the runtime of the algorithm, so that both conditions may be fulfilled. A visual representation of gradient descent is given in **Figure 5**.

Because our cost function is non-negative, and thus bounded from below, a gradient descent algorithm that minimizes C will reach a local minimum after a finite number of updates. In non-convex optimization problems, there is no guarantee that the local minimum found by gradient descent is also a global minimum. However, our cost function is convex [31], so the local minimum is also the global minimum of the function, and therefore the algorithm finds the optimal value of C. In the next section, we describe neural networks, models that use multiple connected neurons. The cost function for multi-layer neural networks is non-convex, and the gradient descent algorithm may get stuck in local minima and not find the global minimum. We note that the problem of finding the parameters of neuron so that it classifies linearly separable sets is equivalent to multivariable logistic regression.

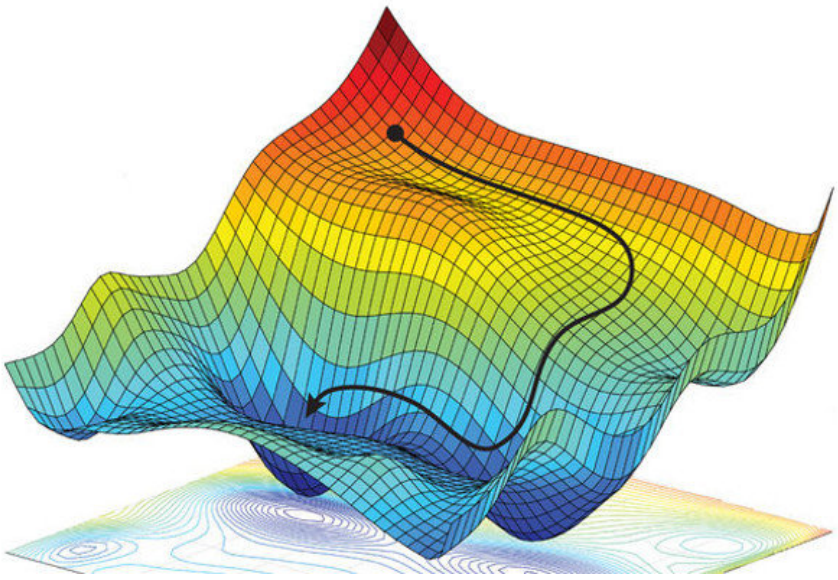

Figure 5: Finding a local minimum of a loss function by gradient descent (Source: [4]).

2.2.2 Artificial Neural Networks

The limitation of a single neuron is that its decision boundary is always a hyperplane, thus it may only classify linearly separable sets of inputs. A classic way of showing

this limitation is with the *exclusive disjunction* (XOR) logical operation. The XOR operation has two binary operands, taking values 0 or 1, and yields 1 when exactly one of the inputs has value 1, otherwise it yields 0. The graph of the operation is shown in **Figure 6**. Clearly, no line exists that can separate the inputs yielding value 0 from inputs yielding value 1, i.e., the two sets are not linearly separable. Therefore, a single neuron can not classify such inputs, and can not express the XOR operation. To construct models with larger expressive power, we connect multiple neurons together.

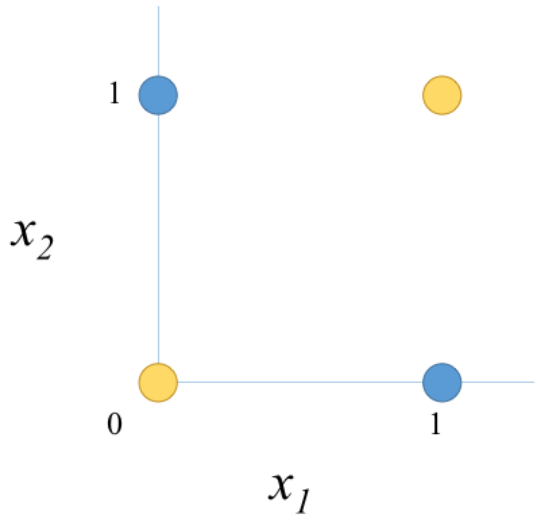

Figure 6: The XOR function. Blue and yellow points denote a result of 1 and 0 respectively.

A (artificial) neural network is a collection of neurons that are linked together by directed connections, so that the output of one neuron is an input to another neuron. A weight is assigned to every connection. The *architecture* of a neural network broadly describes how its neurons are connected and divided into components. While there are in principle no significant limitations to how we may connect the neurons, in this work we cover only feedforward neural networks. In such networks, there are no cycles (loops) formed by the neuron connections, therefore information moves only in one direction - from the inputs, forward through intermediate neurons (called *hidden* neurons), toward the output neurons which give the outputs of the network. More formally, feedforward networks are those that are modeled by directed acyclic graphs (DAGs).

While different neurons in a neural network may have different activation functions, in practice the majority of neurons will have the same activation function. It is important to note that if every neuron in a network has a linear activation function, then the output of the network is a composition of linear functions of the inputs, and the network itself is therefore a linear function of the inputs. Such a network is thus no more powerful than a single neuron with a linear activation function. Therefore, at least

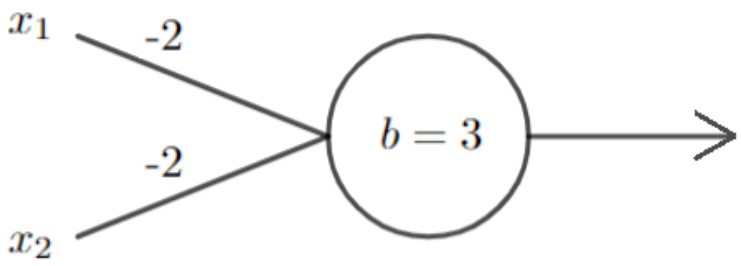

Figure 7: A neuron implementing the NAND operation.

some of the neurons in a neural network should have nonlinear activation functions. More generally, if every neuron in a network has a polynomial activation function, then the network itself is a polynomial function on the inputs. We describe an interesting and powerful result in **Section 2.3.1.1** that shows multilayer perceptrons of adequate structure may approximate any continuous function if and only if the activation functions of neurons are non-polynomial. Lastly, note that a neural network with only continuous activation functions outputs a continuous function.

We give an example of a small feedforward neural network that implements the NAND (NOT-AND) logical operation, shown in **Figure 7**. The NAND operation has two binary operands, and yields 1 when at least one of the inputs has value 1, otherwise it yields 0. We assume the network uses the Heaviside activation function and that the network's inputs x_1 and x_2 are binary values; in this case the network outputs exactly x_1 NAND x_2. Even if the sigmoid function is used or if the inputs can take any real values in $[0, 1]$, the network still behaves in a similar way, by expressing a continuous approximation of x_1 NAND x_2.

2.2.2.1 Functional Completeness of Neural Networks

The NAND logical operation is significant because it has the property of *functional completeness*, meaning that *any* Boolean function can be expressed using a composition of NAND operations [18]. Therefore, for any Boolean function f we can construct a neural network that implements f: we first implement f using a logical circuit consisting of NAND gates and then replace every NAND gate with its neural network implementation. The functional completeness of neural networks is an important result, as binary computations are often used as the basis of both theoretical computational models and their practical implementations (for example, the digital computer).

While the NAND operation is functionally complete, different Boolean functions are implemented using NAND gates circuits of different structures. Simply replacing each NAND gate with its neural network implementation would result in each func-

tion being implemented by a neural network with different architecture. It would be beneficial if there existed a single neural network architecture capable of expressing every Boolean function. For fixed numbers of inputs and outputs, there does exist such a functionally complete neural network architecture which needs only changes in its parameters, not structure, to implement different Boolean functions. We now show how to construct such a network.

Consider a Boolean function $f : \{0,1\}^n \to \{0,1\}^m$, $n, m \in \mathbb{N}$. The network takes n inputs $x_1, x_2, ..., x_n$, and has m outputs $y_1, y_2, ..., y_m$. We denote each possible input value $x_1x_2...x_n$ by a non-negative integer k, where $x_1x_2...x_n$ is the binary representation of k. The network we construct will have a hidden layer of 2^n neurons, $N_{h_1}, N_{h_2}, ..., N_{h_{2^n}}$, one for each possible n-bit integer value. Each neuron in this layer will take as input all of the network's n inputs. For neuron N_{h_i}, we set the weight assigned to the input x_j to 1 if the j-th bit of the binary representation of i has value 1, and to -1 if the bit has value 0. We set the neuron's bias to $b = -sum$, where sum is the sum of the digits of the binary representation of i. Now, each neuron N_{h_i} will output 1 precisely when the network's input is i. This way, each neuron in the hidden layer represents one possible value of the inputs, and only one neuron will fire for any input (a similar functionality can be seen in the demultiplexer digital circuit). Each of the m output neurons takes as input all of the outputs from the hidden layer. For each hidden neuron N_{h_i} and output neuron N_{o_j}, if y_j should be 1 for an input value i per function f, then the weight assigned to the output of N_{h_i} is 1, otherwise the weight is 0. The bias is -1 for each output neuron. Now, if the network is given input k, the output neuron N_{o_j} fires if and only if the j-th binary output is 1 for input k per function f; therefore, the network implements function f. As an example of this construction, the XOR function, which cannot be expressed by a single neuron, is implemented by a network shown in **Figure 8**.

We see that the architectures of networks constructed in this way does not depend on the function f for a fixed n and m. The only part of the network that depends on f are its parameters, namely the biases of the hidden neurons and the weights assigned to the outputs of those neurons. If n and m are not fixed, we can still achieve the same properties by constructing the same network but with infinitely many neurons in the hidden layer and in the output layer. Then, for a function f with n inputs and m outputs, we use only 2^n hidden neurons by setting the weights of the inputs to and outputs of the neurons of the form N_{h_i} such that $i \geq 2^n$, and we discard the outputs of output neurons that are not to be used. Of course, networks with an infinite number of neurons are only a theoretical construct. We also note that the number of hidden neurons in the finite architecture is 2^n - an exponential function of n - which is an infeasible number in practice due to time and space constraints. Therefore, these results are mostly of theoretical interest. On the other hand, an existence of functions

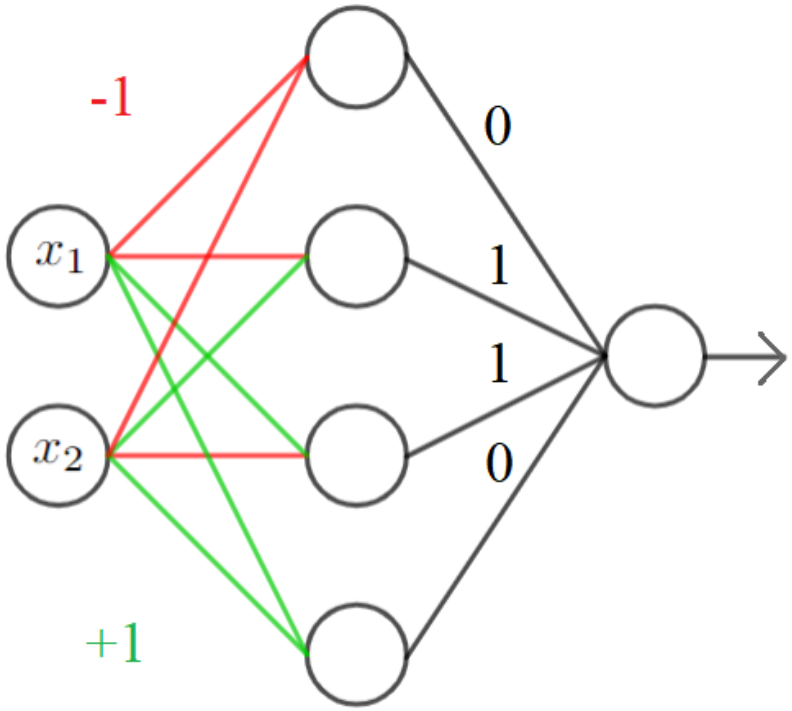

Figure 8: A neural network implementing the XOR operation.

not expressible by any neural network would potentially demotivate their use in some instances, so the results have limited value in regards to practice well.

2.2.3 Multilayer perceptrons

We present the structure of *multilayer perceptrons* (MLPs), a special class of feedforward neural networks that will be useful in studying the models used in the remainder of this work. An MLP is a sequence of at least three *layers* of *nodes*: an input layer, one or more hidden layers, and an output layer. We say that an MLP having $n-2$ hidden layers is an n-layer MLP. The nodes in the input layer simply represent inputs; the nodes in the hidden layers and the output layers are neurons. We say the input layer is the first layer of an n-layer MLP, the i-th hiddel layer is the $(i+1)$-th layer of the network, and the output layer is the last, or n-th, layer of the network.

Now, the MLP has the following structure: all nodes in the j-th layer are connected to all of the nodes in the $(j+1)$-th layer, so that the outputs of the nodes in the j-th layer are given as inputs to the nodes in the $(j+1)$-th layer. The outputs of the input layer are simply the input values of the MLP; the outputs of any other layer are the outputs of the layers' neurons. Because every neuron in one layer is connected to every neuron in the next layer, we say that the MLP is a *fully-connected* class of neural networks (note that the term does not imply that the neurons in non-neighboring layers

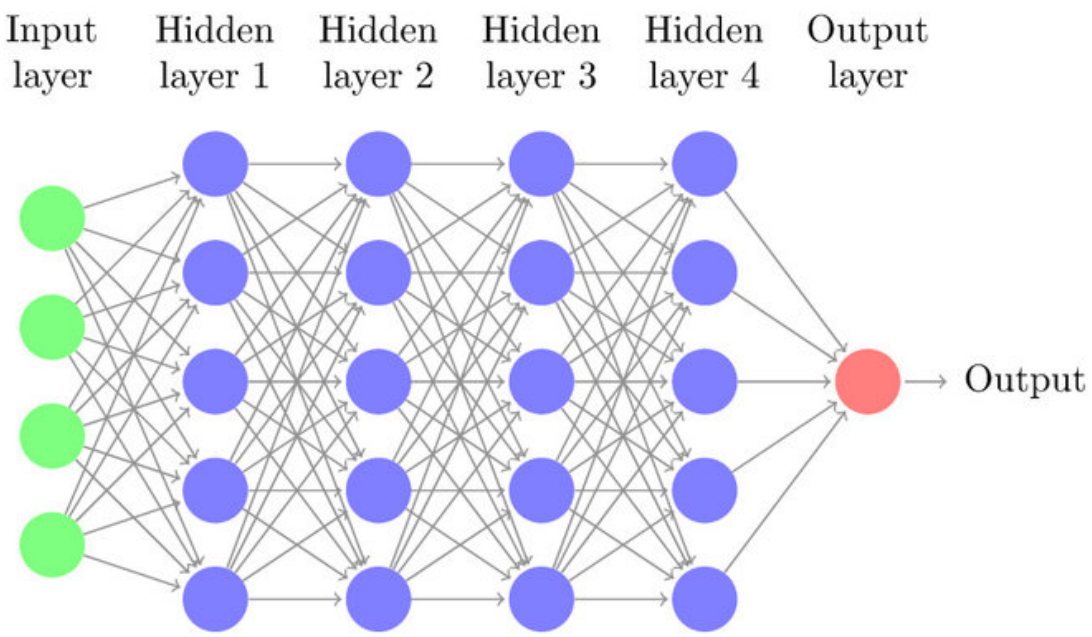

Figure 9: A multilayer perceptron (Source: [64]).

are connected to each other). An MLP is shown in **Figure 9**. The neural network architecture that can express any Boolean function, presented in **Section 2.3.1**, is a multilayer perceptron. To avoid confusion, we note that the "perceptron" in "multilayer perceptron" does not stand for the Perceptron algorithm.

2.2.3.1 Universality of Multilayer Percpetrons

A question of importance is which functions multilayer perceptrons can express. One of the most famous results in neural network theory is the following: under minor conditions on the activation function, an MLP with a single hidden layer can arbitrarily well approximate *any* continuous function on a compact set. Here, a compact set usually refers to a closed and bounded subset of the Euclidean space. Theorems concerning such approximation properties of MLPs are called *universal approximation theorems*. Similar variations of the stated universal approximation property were proven for networks with arbitrary squashing activation functions (functions that map the input to some small range, e.g., to [0, 1]) by [28] and for networks with sigmoid activation functions by [14] in 1989; [14] contains a discussion of the different results. Later, [40] and [54] showed that the universal approximation property is equivalent to having a nonpolynomial activation function. A detailed overview of various results in neural network theory in general is given in [53]. We state the universal approximation theorem of [54] more formally by **Theorem 1**:

Theorem 1 (Universal approximation theorem) Denote by $C(X, Y)$ the set of continuous functions from X to Y. Let $\sigma \in C(X, Y)$, and note that $\sigma \circ \mathbf{x}$ denotes σ applied to each component of $\mathbf{x}$. Then σ is not polynomial if and only if for every $n \in \mathbb{N}, m \in \mathbb{N}$, compact $K \subseteq \mathbb{R}^n, f \in C(K, \mathbb{R}^m), \epsilon > 0$ there exist $k \in \mathbb{N}, A \in \mathbb{R}^{k \times n}, \mathbf{b} \in \mathbb{R}^k, D \in \mathbb{R}^{m \times k}$ such that

$$\sup_{\mathbf{x} \in K} \|f(\mathbf{x}) - g(\mathbf{x})\| < \epsilon$$

where

$$g(\mathbf{x}) = D \cdot (\sigma \circ (A \cdot \mathbf{x} + \mathbf{b})).$$

In simpler terms, for every continuous function f on a compact set, there exists an MLP g taking inputs $\mathbf{x}$ such that $\|f(\mathbf{x}) - g(\mathbf{x})\| < \epsilon$ for all $\mathbf{x}$ and any $\epsilon > 0$. The proofs of most universal approximation theorems are quite technical, and we will not present them here. For a simple and much less mathematically oriented informal proof, we refer readers to [49], which informally proves the universality of MLPs with an intuitive visual argument.

Universality results are very important, because they show that there are no important limitations to which continuous functions the class of neural networks can approximate. However, the results often have limited implications to practice. They often assume that a network can have an unbounded (sometimes even infinite) number of neurons, which is infeasible in real world implementations. Second, sometimes the results only prove the existence of a network approximating a function, without providing the construction for the network or its weights. Finally, in practice we are mostly interested in neural networks as learning models, rather than as representation models. The existence of a network with weights such that it expresses some function does not imply that the network can feasibly learn the weights. On the other hand, a network can never learn a function it cannot express, thus universality results are still important for learning.

2.2.3.2 Multilayer Percpetrons as Learners

Like in a single neuron, learning in multilayer perceptrons can be treated as an optimization problem. In fact, this is the most common way of learning in MLPs, due to the complexity of the model. The solution space consists of all the weights and biases in the MLP, which we denote by $\mathbf{w}$ and $\mathbf{b}$ respectively. The cost function C_X under a set of inputs X is defined as follows:

$$C_X(\mathbf{w}, \mathbf{b}) = \frac{1}{2n} \sum_{\mathbf{x}} \|\mathbf{y}(\mathbf{x}) - \mathbf{o}_{\mathbf{w},\mathbf{b}}(\mathbf{x})\|^2,$$

where $\mathbf{y}(\mathbf{x})$ is the desired output vector for input $\mathbf{x}$ and $\mathbf{o}_{\mathbf{w},\mathbf{b}}(\mathbf{x})$ is the actual output of the network for input $\mathbf{x}$ for the weights $\mathbf{w}$ and biases $\mathbf{b}$ of the network. The gradient descent algorithm is again used to optimize the network by minimizing the cost function. The evaluation of the gradients of the function by the algorithm is more complicated than in the case of a single neuron, and is typically performed by the *backpropagation* algorithm; a formal yet intuitive description of the backpropagation algorithm can be found in the work by Nielsen [49].

2.3 CONVOLUTIONAL NEURAL NETWORKS

In this work, we use neural networks to perform image classification. In a digital computer, an image is a grid-like structure where every point in the grid contains C values - one value for each *channel* of the image. For example, most color photographs are two-dimensional RGB images, where "R", "G", and "B" denote the red, green, and blue channels, and grayscale images are two-dimensional images with one channel specifying the gray value of a point. The elements of a two-dimensional image are called *pixels* (from "pixel element") and elements of a three-dimensional image are called *voxels* (from "volume element"). Formally, an n-dimensional image with C channels is a tensor of rank $n+1$. For instance, an image I with dimensions $d_1 \times d_2 \times ... \times d_n$ and C channels, where every element of the image is a real value, is a point in space $\mathbb{R}^{d_1 \times d_2 \times ... \times d_n \times c}$, i.e., $I \in \mathbb{R}^{d_1 \times d_2 \times ... \times d_n \times C}$. Usually, the values in an image are discrete and lie in a bounded range.

In image classification, there is a set $\mathcal{I} = \{I_1, I_2, ..., I_n\}$ of images and a set $\mathcal{K} = \{K_1, K_2, ..., K_m\}$ of classes such that each image $I \in \mathcal{I}$ is assigned to exactly one class, denoted by $k(I) \in \mathcal{K}$. We note that the set $\{\{I : k(I) = K\} : K \in \mathcal{K}\}$ is a partition of the set of images $\mathcal{I}$. The task of a classification computer program is to correctly classify each image $I \in \mathcal{I}$ (of course, without knowing $k(I)$ beforehand). More formally, a classification program is a function $f : \mathbb{R}^{d_1 \times d_2 \times ... \times d_n \times C} \to \mathcal{K}$, and we optimally want $f(I) = k(I) \forall I \in \mathcal{I}$.

Because the set $\mathcal{K}$ of classes is finite, f is clearly discontinuous (we assume at least two images belong to a different class, otherwise the problem is trivial). However, we can formulate the problem in such a way that we use a continuous classification function whose value is later discretized to take only the values in $\mathcal{K}$. For example, given an image I, for every class $K \in \mathcal{K}$ function f might output a value indicating the probability of I belonging to K; then the class with highest probability is chosen as the classification result. Here, each probability is a continuous value in $[0, 1]$. This way we have $f : \mathbb{R}^{d_1 \times d_2 \times ... \times d_n \times C} \to [0, 1]^m$, so f is a continuous multivariate function. By the Universal Approximation Theorem, there is a multilayer perceptron that approximates

f. This means that there are theoretically no significant restrictions on using neural networks for image classification; indeed, deep MLPs have been successfully used on some image classification tasks [9].

While multilayer perceptrons have been shown to produce good results on some image classification problems, the problems were usually defined on small images or datasets and only a few classes. For larger images, datasets, and sets of classes (for example thousands of different classes), there are significant hurdles to using MLPs. These hurdles are caused by the fact that in most image classification tasks images are classified based on their spatial properties. On the other hand, the fully connected nature of MLPs means that their architecture does not assume anything about the spatial properties of their inputs. Rather, MLPs may potentially learn weights that reflect such properties of the data. However, this increases the time needed for learning, and also introduces a possibility of the network not learning the spatial properties but rather relying on other image features.

To overcome these problems, the *convolutional neural network* (CNN) architecture has been designed. Like an MLP, the neurons of a CNN are organized into layers, however most layers of a CNN are not fully connected. Rather, neurons are connected in such a way that the architecture of the network implicitly takes into account the spatial properties of images.

CNNs have been one of the dominant neural network architectures used in computer vision and image processing, and their success has revolutionized many subfields in these domains. This section is dedicated to convolutional neural networks. We first describe the problems in using fully connected networks on problems involving data with strong spatial structure, and then present the convolutional neural network architecture.

2.3.1 Spatial Properties of Images

In image classification, images are usually not assigned to arbitrary classes. Rather, image classification reflects real world vision problems, ones important to humans. For example, in everyday life, we are more interested in recognizing real world objects than we are in solving jigsaw puzzles. One of the most important aspects of making sense of a scene we see is being able to discern the different objects in the scene and how the objects are positioned relative to each other (the layout of the scene). The positioning of objects obviously includes a spatial component, but so does recognizing the objects itself.

In recognizing an object, one of its most important properties is its shape. Evidence is clear that shapes guide human attention [2] [65], and it is known that object shape is the most important cue for human object recognition [36]. Shapes can be thought

of as having a hierarchical and recursive structure: the lowest level shapes include, for instance, edges and corners, low-level shapes are combined to form more complex shapes, the more complex shapes are combined to form even more complex shapes, etc. For example, a standard shape of a human standing upright includes the head, torso, limbs, etc. The face of a head includes the contour of the face and the different parts: eyebrows, eyes, nose, mouth, hair, etc. Each eye then includes the lids, sclera, cornea, eyelashes, etc. Each of these components also has a distinct shape that can be further decomposed into smaller ones. The relative positions of lower level shapes is very important when they are combined into higher level ones - different arrangements of lower level shapes result in different higher level shapes. In computer images, the lowest level shapes themselves may be thought of as spatial arrangements of image elements (pixels, voxels) with different color values. Thus, we see that the spatial properties of an image are important in general, from recognizing its low level details, to recognizing more complex shapes, to understanding the layout of objects. Thus, approaches to image classification greatly benefit from taking account the spatial structure of data.

One of the problems with applying multilayer perceptrons to image classification is that MLPs, which are fully connected models, do not know the spatial properties of their inputs. We first demonstrate by the following example. Let there be a single neuron N_1 taking two inputs x_1 and x_2, and assume it has learned weights w_1 and w_2 for the two inputs respectively. Let the inputs be such that x_1 has binary representation $b_{1_1}b_{1_2}...b_{1_n}$ and x_2 has binary representation $b_{2_1}b_{2_2}...b_{2_n}$. Now, let N_2 be a neuron taking as inputs the individual bits of x_1 and x_2, i.e., N_2 has inputs $b_{1_1}, b_{1_2}, ...b_{1_n}, b_{2_1}, b_{2_2}...b_{2_n}$, and assume a weight w_{i_j} is assigned to each input b_{i_j}. Clearly, we can classify each input variable b_{i_j} to N_2 into two groups based on its index i: either as belonging to x_1 or as belonging to x_2. Furthermore, the index j of an input b_{i_j} signifies its "importance" to the value x_1: a small j indicates a large significance of b_{i_j} - for example, b_{i_1} is the most significant bit of x_i. These properties specify the spacial structure of the inputs to N_2. We illustrate the example in **Figure 10**.

Let us assume the inputs $b_{1_1}, b_{1_2}, ..., b_{1_n}, b_{2_1}b_{2_2}..., b_{2_n}$ are stored as a vector, and appear in that specific order. Now, the weighted input to neuron N_2 is:

$$\sum_{i=1}^{n} w_{1_j} b_{1_j} + \sum_{i=1}^{n} w_{2_j} w_{1_j}.$$

In this expression, there is nothing telling the neuron to treat the different inputs differently except the weights that are assigned to them. Assuming the weights do not somehow encode the spatial information of their input, the inputs are treated irrespective of their location. In this particular case, we could encode the information into the weights by having $w_{i_j} = 2^{n-j} w_i$. Now, the weighted input to N_2 would equal $w_1x_1 + w_2x_2$, which is the weighted input to N_1. It is theoretically possible

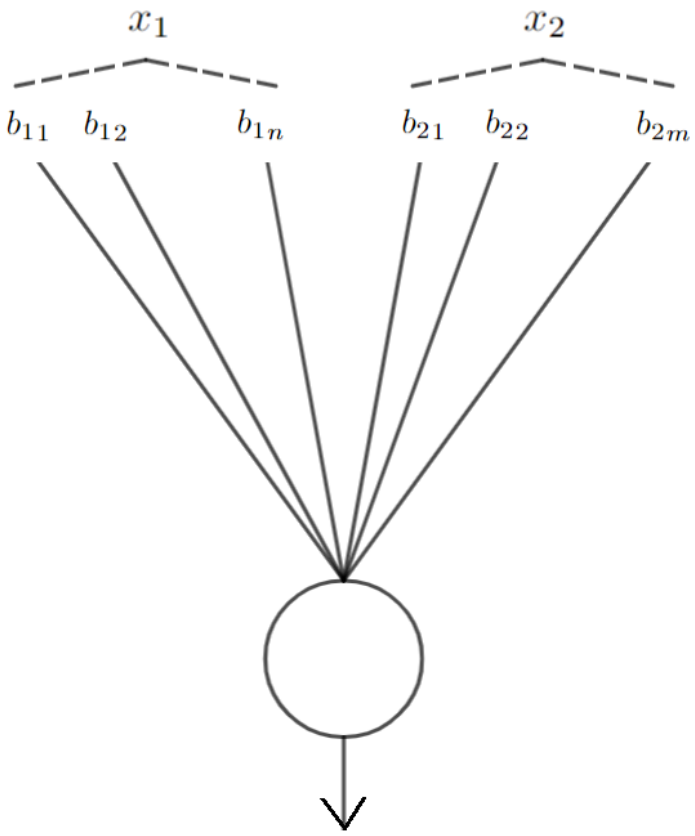

Figure 10: A neuron taking the binary representation of numbers x_1 and x_2 as inputs.

that N_2 could learn these weights. However, the expressions for updating the weights that appear in the learning algorithms we described do not assume spatial relations between different inputs. Therefore, neither the neuron nor the learning algorithm encode spatial information about the inputs, and the same is true for the multilayer perceptron architecture as a whole and for the learning algorithms for MLPs.

In fact, we do not want the general model of the neuron or the MLP to assume any spatial properties of data, as we want MLPs to work on arbitrary data, and different data may have different spatial properties, or no spatial properties at all. With data that does have spatial properties, they may potentially be learned from the data itself, but, as stated before, there are several major disadvantages to relying on learning without encoding the properties into the model itself.

We give an example that illustrates some implications of the above analysis for image classification. Assume a set $\mathcal{I}$ of two-dimensional images of size $H \times W$, H and W denoting the image height and width respectively, and having one channel. We denote the pixel in row i and column j of image I by $p_{i,j}$, and we can denote an image I by $\{p_{i,j} : i \in \{1, ..., H\}, j \in \{1, ..., W\}\}$. Let $g : \{1, ..., H\} \times \{1, ..., W\} \rightarrow \{1, ..., H\} \times \{1, ..., W\}$ be a bijection. Now, we "shuffle" each image $I \in \mathcal{I}$, to obtain a new image $I' = \{p_{g(i,j)} : i \in \{1, ..., H\}, j \in \{1, ..., W\}\}$, $\mathcal{I}'$ being the set of all such images. This rearranges the pixels of each image, so that the colors of the image are preserved but their arrangement is different. Each image is shuffled in the same way. We assign each image I' to the same class as image I, i.e., $k(I') = k(I)$.

Now, assume an MLP M correctly classifies the set $\mathcal{I}$, and that a neuron N_k in the first hidden layer has weight $w^k_{i,j}$ for input $x_{i,j}$. We can easily construct an MLP M' that correctly classifies the set $\mathcal{I}'$ in the following way. We let M' have the same architecture as M, but let every neuron N'_k in the first hidden layer of M' have weight $w^k_{i,j}{}' = w^k_{g(i,j)}$ for input $x_{i,j}$, and let the weights in other hidden layers equal the corresponding ones in M; this completes the construction. Essentially, we have shuffled the weights of the neurons in the first hidden layers the same way we shuffled the pixels in the image. Now M' classifies $\mathcal{I}'$ the same way M classifies $\mathcal{I}$. Moreover, due to the learning algorithms we described not knowing the spatial structure of image data, M would not learn its weights w any differently than M' would learn its weights w'. A network architecture that takes advantage of the structure of image data would allow for faster and more successful learning of the desired weights.

In short, before learning an image dataset, a fully connected neural network does not know where an input x_1 is positioned within the image, whether the input is close to or far away from another input x_2, the dimensions of the image, or even that the (two-dimensional) image is of rectangular shape. The flip side to fully connected networks is that they can learn datasets regardless of spatial properties of the data, while architectures that assume specific spatial properties of data are very likely to be

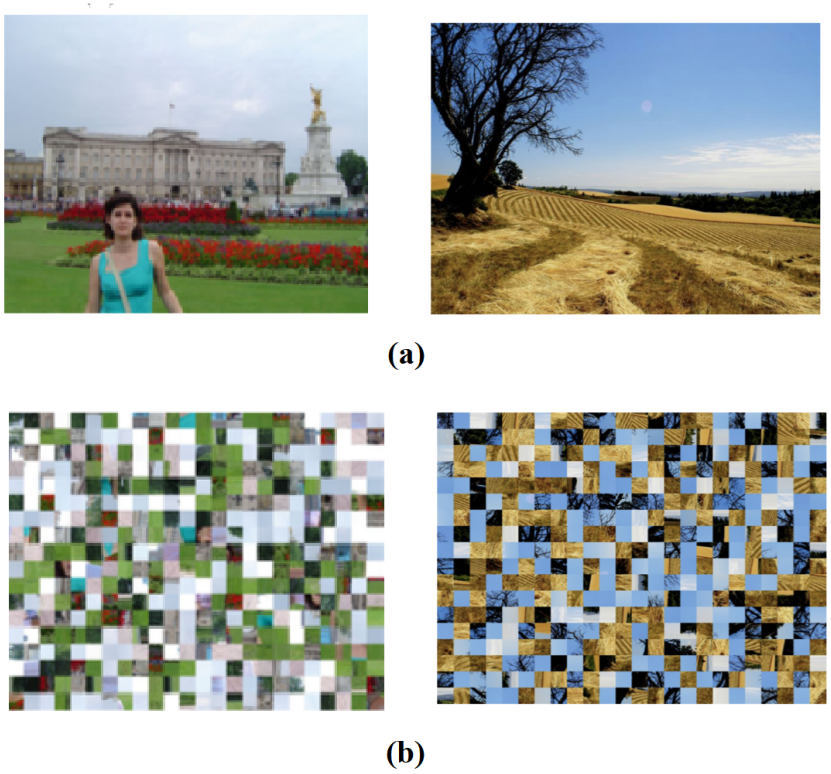

Figure 11: Two image datasets: dataset (b) is obtained by shuffling each image in dataset (a) the same way.

less flexible.

We illustrate this example by **Figure 11**. An MLP, being a fully connected architecture, learns the dataset of images (a) and the dataset of the shuffled images (b) with the same success and in the same amount of time. A learning model that assumes the images have their standard spatial properties, such as the convolutional neural network architecture, would very likely learn dataset A with more success or more quickly than would an MLP. However, such a model would probably learn dataset B with less success or more slowly than would an MLP. We notice that humans have considerable trouble with making sense of the shuffled images. This is because the human visual system itself assumes spatial properties of the image input it receives [63]. In fact, we will later see that convolutional neural networks are inspired by some structures in visual systems found in animals.

2.3.2 Two Preliminary Architectures

Among other properties, convolutional neural networks have an architecture that implicitly encodes the structure of images that they are to take as input. We first illustrate this property using a simplified example, one that does not relate to images but to audio (sound) data. Sounds propagate through a medium such as air, water and solids

as waves. An illustration of a sound recording, which displays some wave properties of a sound, is shown in **Figure 12**, where each vertical column in the recording specifies the intensity of the sound at a moment in time; the intensities appear chronologically. We note that, if each intensity in this sound wave was shown by a single pixel - the intensity of the pixel corresponding to the sound intensity - we may interpret such a sound wave as a one-dimensional image.

The recording in the figure can be stored by a computer as a vector $s = (s_3, s_2, ..., s_n)$, where s_i represents the sound intensity at moment i, thus the order of intensities is chronological. The spatial structure of the data is due to this chronological order, meaning that, for example, s_i is "closer" to s_{i+1} than to s_{i+2} and that s_i and s_{i+2} are on different "sides" of s_{i+1}. The shape of a sound wave carries implications to its analysis: an increase in a region of the graph represents an increase of loudness or an emergence of a new sound, a tall peak represents a short but loud sound, etc. For long sound recordings, detecting local features such as flat levels, peaks, and valleys, and how those features combine with neighboring ones into larger features, is of great importance. By the universality theorem, multilayer perceptrons can, at least in theory, be used for such sound analysis. However, being fully connected architectures, MLPs do not assume any spatial properties of data.

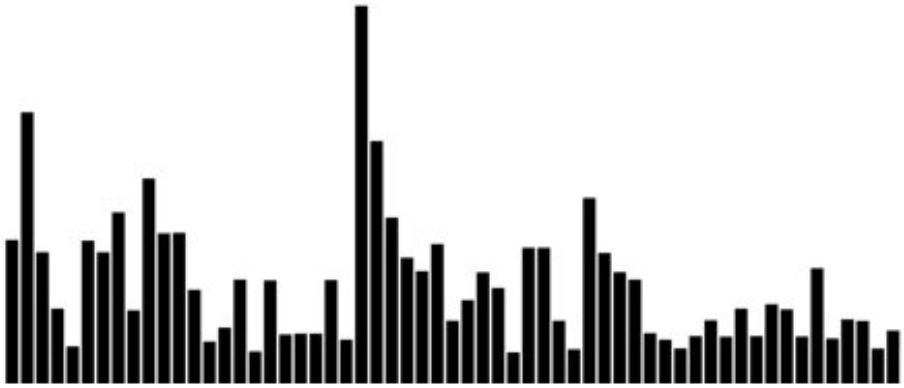

Figure 12: A sound recording.

We show an architecture that does assume a spatial structure of the data in **Figure 13**; this architecture contains one of the core ideas of convolutional neural networks, as we later see. In this specific example, there are 8 inputs $s_1, s_2, ...,$ and s_8, where s_{i+1} is the sound intensity that chronologically appears right after intensity s_i. The neural network has three non-input layers: the first layer contains neurons N_1^1, N_2^1, N_3^1, and N_4^1, the second layer contains neurons N_1^2 and N_2^2, and the output layer contains a neuron N_1^3. Each neuron N_i^1 takes inputs s_{2i-1} and s_{2i}, while each neuron N_j^i in the second and third non-input layer takes as inputs the outputs from neurons N_{2j-1}^{i-1} and

N_{2j}^{i-1}.

The main difference between an MLP and network A_1 is that in A_1 every neuron is connected to only 2 ("neighboring") nodes in the previous layer, whereas each neuron in an MLP is connected to all of the nodes in the previous layer. We say that the the neurons in A_1 have a *receptive field* of size 2, and that the receptive field of each neuron in an MLP is the entire previous layer.

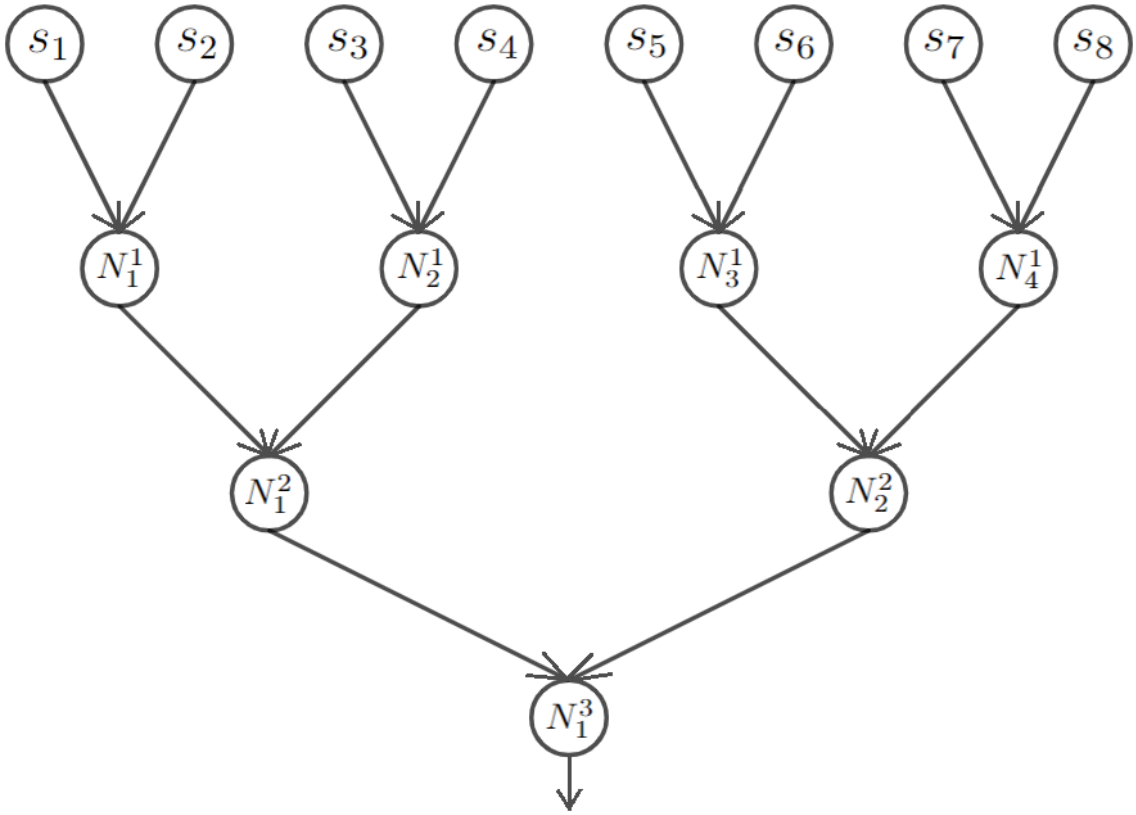

Figure 13: Architecture A_1.

Performance wise the network A_1, having very few neurons, is not nearly large enough to be used for analyzing sounds s, but we use it to illustrate the following idea. As each neuron N_i^1 in the first hidden layer takes as inputs only the neighboring sound intensities s_{2i-1} and s_{2i}, we may assume that N_i^1 has a considerably larger probability of learning local features found on these two inputs compared to a "fully connected" neuron that would take as inputs all of the sound intensities in s. Then, as each neuron N_i^2 of the second layer takes as inputs only the outputs of N_{2i-1}^1 and N_{2i}^1, it has a larger probability of combining the two local features recognized by N_{2i-1}^1 and N_{2i}^1 into larger local features that appear on the range $\{s_1, s_2, s_3, s_4\}$ of inputs, compared to a fully connected neuron that would take as inputs all of the outputs from the first hidden layer. This probability is especially larger when compared to that of a network where both the first hidden layer and the second hidden layer are fully connected to their inputs. These conclusions generalize to the third layer, and would also generalize to

lower layers of deeper architectures used on longer sounds.

We note that the size of the largest feature that may be learned by a neuron grows exponentially as we move down the layers. More specifically, a neuron in the i-th non-input layer can learn a feature defined on 2^i (consecutive) inputs to the network. The output layer can therefore learn features defined on the entire range of the input, while the depth of the network is logarithmic in n. The network contains $\mathcal{O}(n)$ neurons.

The architecture of network A_1 implicitly encodes some spatial properties of its data, namely how close some of the inputs are to each other. The architecture thus greatly simplifies the learning of local features and combining the features into larger features, which is of great importance to sound analysis. We will later see that the *convolutional layers* of CNNs generalize this architecture to image data, and we note that CNNs are used for analyzing audio signals [50].

There are spatial properties of the sound wave that are not encoded into the architecture of A_1. Specifically, consider a neuron N_j^i taking inputs N_{2j-1}^{i-1} and N_{2j}^{i-1}. While N_{2j-1}^{i-1} and N_{2j}^{i-1} are "linked" together, the neuron N_j^i does not know the spatial properties of its inputs, which in this case means it cannot determine that the input N_{2j-1}^{i-1} is to the "left" of input N_{2j}^{i-1}. This implies that we can shuffle (rearrange) each pair of inputs of the form (s_{2i-1}, s_{2i}) into (s_{2i}, s_{2i-1}) for every sound s in the dataset, and the network would learn its equivalent weights to the case when the pairs are not shuffled. Here, we say that weights in two networks are equivalent if the networks have the same values of weights but the arrangements of the weights within layers are different - they are shuffled the same way as the inputs. The conclusions generalize to lower layers, so, for example, splitting $s = (s_1, s_2, ..., s_n)$ into two halves and changing the order of the halves to get $s' = (s_5, s_6, s_7, s_8, s_1, s_2, s_3, s_8)$ for each s would also result in two networks that learn equivalent weights.

Moreover, consider a neuron N_j^i with inputs N_{2j-1}^{i-1} and N_{2j}^{i-1}, the first having inputs N_{4j-3}^{i-2} and N_{4j-2}^{i-2} and the second having inputs N_{4j-1}^{i-2} and N_{4j}^{i-2}. There is nothing that informs the architecture that N_{4j-2}^{i-2} is the neighbor of N_{4j-1}^{i-2} while neither N_{4j-3}^{i-2} nor N_{4j}^{i-2} are neighbors to any input of N_{2j}^{i-1} and N_{2j-1}^{i-1} respectively. This means that we do not have to shuffle both of the pairs of inputs to N_{2j-1}^{i-1} and N_{2j}^{i-1} in order to obtain two networks that learn equivalent weights, as they are obtained even when only one pair is shuffled. For example, we can shuffle any number of input pairs (s_{2i-1}, s_i) of the network (into (s_i, s_{2i-1})) and two networks would again learn equivalent weights. Similar conclusions apply to shuffling inputs to neurons in lower layers.

These two problems show that some spatial properties of the data are still missing from the architecture of network A_1. They exist because the neurons in the network have a *stride* of 2 while having a receptive field of size 2. A *stride* of 2 means that a neuron N_j^i is offset by 2 inputs from its predecessor, i.e., it takes inputs N_{2j-1}^{i-1} and N_{2j}^{i-1} (or s_{2j-1} and s_{2j}) while its predecessor N_{j-1}^i takes inputs N_{2j-3}^{i-1} and N_{2j-2}^{i-1} (or

s_{2j-3} and s_{2j-2}) - the difference in the indexes of the inputs is 2. Because the stride is not smaller than the receptive field, there is no overlap between the inputs of two neighboring neurons. The problems are solved by decreasing the stride to 1 and thus increasing the overlap between the inputs. We implement this in network A_2.

The network A_2 has the following architecture. We again assume the same inputs $s_1, s_2, ..., s_8$. A_2 has 7 non-input layers. The second layer of the network - the first hidden layer - contains 7 neurons N_i^1, where each N_i^1 takes inputs s_i and s_{i+1}. Each i-th layer for $i \in \{3, ..., 7\}$ contains neurons N_j^i where $j \in \{1, ..., 8-i\}$, and each neuron N_j^i takes inputs N_j^{i-1} and N_{j+1}^{i-1}. The neurons thus all have strides of 1. The network is shown in **Figure 14**.

Now, due to the structure of the first hidden layer of A_2, for every pair of consecutive intensities (s_i, s_{i+1}) there is a neuron in the first hidden layer taking only them as inputs. If we were to change the order of intensities (s_i, s_{i+1}) to (s_{i+1}, s_i) in the input vector s, the neuron N_{i-1}^1 would take non-neighboring intensities s_{i-1} and s_{i+1} as inputs instead of intensities s_{i-1} and s_i; an analogous problem would occur for neuron N_{i+1}^1. We propose that no shuffling of the inputs results in two networks with the architecture of A_2 learning equivalent weights - except for the single case where the input vector is reversed (i.e., when $s_i' = s_{8-i+1}$ for each i) - thus increasing the importance of the data having adequate spatial properties.

A second benefit in architecture A_2, brought by the structure of the lower non-input layers, is that for any two neighboring neurons N_j^i and N_{j+1}^i there is a neuron N_j^{i+1} in a lower layer that takes them as input, thus making it easier to combine features defined by the two neighboring neurons into larger ones. In the architecture of network A_1, only the neurons of the form N_{2j-1}^i and N_{2j}^i were "combined" by a neuron in the layer $i+1$. We note that in A_2 the size of the largest feature that may be learned by a neuron again increases as we move down the layers, but not exponentially like in network A_1. In A_2, a neuron in the i-th non-input layer can learn a feature defined on $i+1$ (consecutive) inputs to the network, thus this size increases linearly. The output neuron can again learn features on the entire range of the input, but the network is significantly deeper than A_1, as its depth is linear in n. Furthermore, network A_2 contains $\mathcal{O}(n^2)$ neurons, compared to $\mathcal{O}(n)$ of network A_1.

2.3.3 The Convolution Operation

One of the basic building blocks of convolutional neural network is the *convolutional* layer, which is named after the *convolution* operation. A convolution is a mathematical operation on two functions f and g, $f, g : \mathbb{R}^d \to \mathbb{R}$, that produces a third function

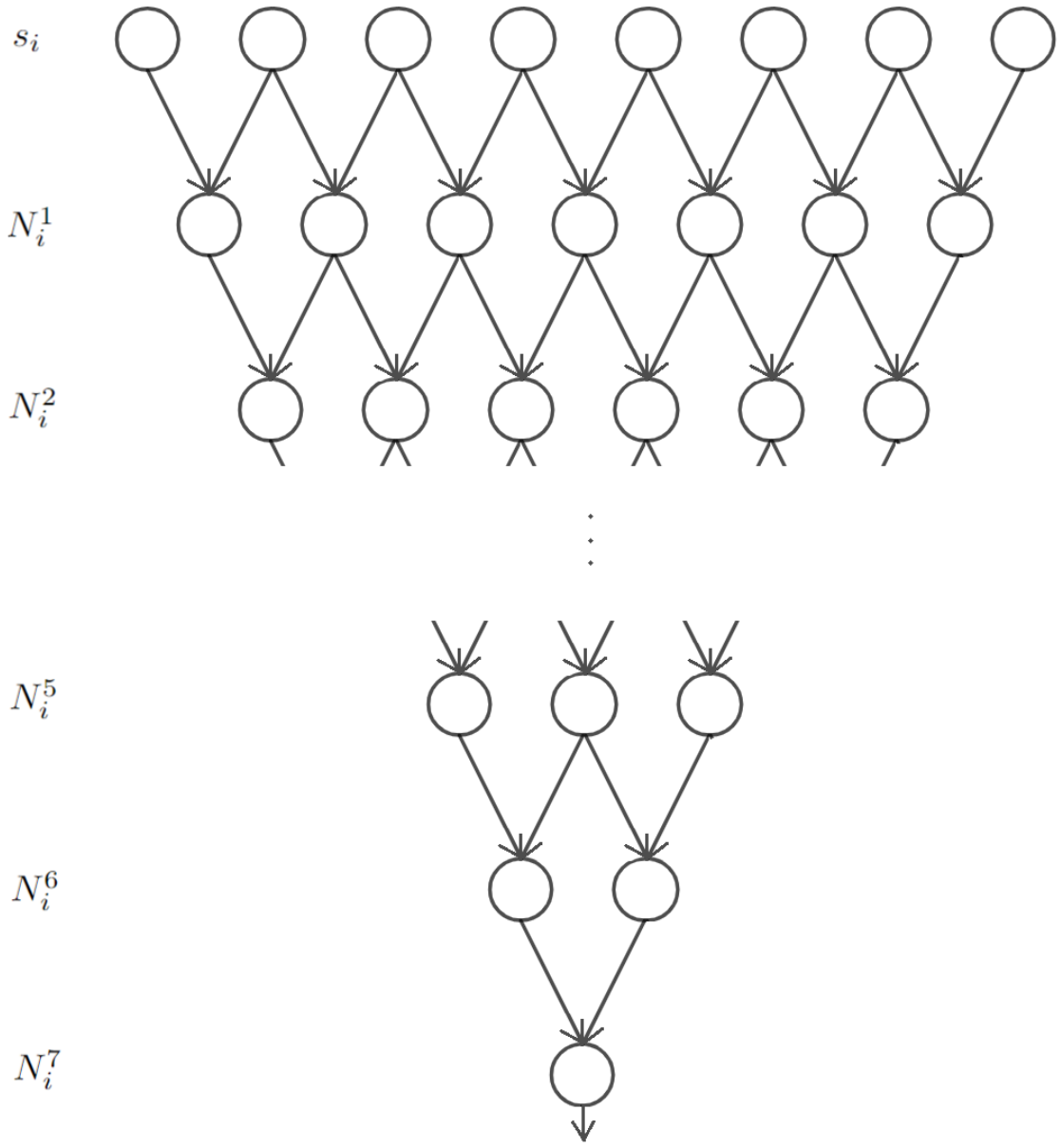

Figure 14: Architecture A_2.

$(f * g) : \mathbb{R}^d \to \mathbb{R}$ such that:

$$(f * g)(\mathbf{x}) = \int f(\mathbf{z})g(\mathbf{x} - \mathbf{z})d\mathbf{z}.$$

The convolution $(f * g)$ expresses the amount of "overlap" of functions f and g as one function is flipped and shifted by $\mathbf{x}$. For discrete functions, the integral turns into a sum. For example, for $f, g : \mathbb{Z} \to \mathbb{R}$ the convolution is:

$$(f * g)(x) = \sum_z f(z)g(x - z).$$

In general, for functions $f, g : \mathbb{Z}^d \to \mathbb{R}$ the convolution is:

$$(f * g)(x) = \sum_{z_1} \sum_{z_2} ... \sum_{z_d} f(z_1, z_2, ..., z_d)g(x_1 - z_1, x_2 - z_2, ..., x_d - z_d).$$

In the terminology of convolutional neural networks, the first argument (here f) to the convolution is often referred to as the *input*, the second argument (here g) as the *kernel*, and the result as the *feature map*. Convolution is commutative, meaning $(f * g) = (g * f)$, beacuse the kernel is "flipped" relative to the input, meaning that the argument of f increases as the argument of g decreases. If we do not flip the kernel, we obtain the *cross-correlation* operation $(f \star g)$ of functions f and g:

$$(f * g)(\mathbf{x}) = \int f(\mathbf{z})g(\mathbf{x} + \mathbf{z})d\mathbf{z}$$

for continuous f and g, and

$$(f \star g)(x) = \sum_{z_1} \sum_{z_2} ... \sum_{z_d} f(z_1, z_2, ..., z_d)g(x_1 + z_1, x_2 + z_2, ..., x_d + z_d)$$

for discrete f and g. Convolutional neural networks in fact use cross-relations instead of convolutions, so the term "convolutional" is a misnomer. However, the distinction is mostly cosmetic, and in machine learning these operations are most often used with other functions, so their combinations with the functions do not commute regardless whether the kernel is flipped or not. To keep in accordance with neural network literature, by "convolution" we refer to cross-correlation in the remainder of the work.

The convolution operation is frequently used in image processing and computer vision. In these fields, a *kernels* (also called *convolution matrix* or *mask*) is a small matrix that can be used for one of many purposes, such as blurring, sharpening, edge detection, etc. A convolution is performed between the kernel and the image to produce a new image. We give an example of using kernels in edge detection.

Edges are one of the most important features found in images. They represent discontinuities in image element color or brightness. Edges can be combined to form complex shapes, and are one of their defining properties. Furthermore, it can be

shown [5] that a rather general image formation model, edges are likely to correspond to discontinuities in depth, discontinuities in surface orientation, differences in scene illumination, and changes in material properties. Image kernels are one of the simplest but still powerful tools used for detecting edges.

For instance, in a two-dimensional image we expect a sharp vertical edge to correspond to a discontinuity in the values in (neighboring) columns of pixels. We can therefore perform a convolution of the vertical Prewitt operator:

$$G_x = \begin{bmatrix} +1 & 0 & -1 \\ +1 & 0 & -1 \\ +1 & 0 & -1 \end{bmatrix}$$

with an image I to obtain a new image I'. We perform the convolution in such a way that it assumes the operator has value 0 for any element appearing outside the matrix. Now, the intensity of each pixel $p'_{i,j}$ in I' is large if and only if there is a vertical edge in the close vicinity of the pixel $p_{i,j}$ in I. Formally, $I' = I \star G_x$. We show an example of convolving the vertical Prewitt operator with an image in **Figure 15**. We explain the role of convolutions in convolutional neural networks in the next section. Finally, we note that the computation of the convolution operation performed on a single pixel can be expressed by a scalar product - this allows the computations to be performed by neurons, as we will see.

(a) **(b)**

Figure 15: (a) Image I (b) Convolution of the vertical Prewitt operator and I.

2.3.4 The Architecture of Convolutional Neural Networks

Convolutional neural networks are a special family of neural networks designed for processing data with grid-like topology. CNNs are very similar to multilayer perceptrons in that they contain layers of neurons that have learnable weights and biases.

However, the fundamental layers of a CNN are *convolutional layers*, in which neurons are not fully connected to the layer preceding them. Furthermore, CNNs most often contain *pooling layers*, which are not composed of neurons. Pooling layers reduce the dimensions of data in the network, thus decreasing computational requirements. Both of these types of layers behave as regularization mechanisms. The layers take as input a k-dimensional tensor of data from the previous layer and transform it to a k-dimensional output tensor through a differentiable function, which may or may not have parameters. The network outputs a result for an input image; in image classification, this result may be a vector of scores (probabilities) for each class. The entire network expresses a differentiable function, therefore it may learn using the gradient descent algorithm.

Like artificial neural networks, CNNs are inspired by structures found in animal brains, specifically in visual cortices. Hubel et al. [26] have shown in the 1950s and 1960s that visual cortices of cats contain (biological) neurons that individually respond to small regions of the visual field. The region of visual space that affects a single neuron was termed the *receptive field* of the neuron, and neighboring neurons have similar and overlapping receptive fields. Hubel et al. [26] [27] also proposed a cascading model that includes such neurons for use in pattern recognition. Their work was an inspiration for the "neocognitron" model introduced in 1980 [19], which introduced convolutional layers and downsampling (pooling) layers and is the origin of the CNN architecture. Many of the early CNN implementations featured hand-designed weights. Zhang et al. in 1988 [67] and LeCun et al. in 1989 [39] used back-propagation to train CNNs for alphabet recognition and hand-written numbers recognition respectfully. The use of learning allowed faster construction of appropriate weights and was suited to a broader range of computer image problems.

The largest breakthrough in convolutional neural networks happened in the 21st century, when graphical processing units (GPUs) were used to train them. In 2011 Ciresan et al. [9] used GPUs to train CNNs which won multiple image recognition contests, even achieving superhuman performance for the first time [29]. One of the most influental works in the field of CNNs and deep learning in general [3] is AlexNet [34], a GPU-based CNN that won the ImageNet Large Scale Visual Recognition Challenge in 2012. Subsequently, CNNs became state of the art models in numerous computer vision and image processing tasks. We show the AlexNet architecture later in this section.

The typical architecture of a convolutional neural network is as follows. Like in an MLP, the first layer in a CNN is the input layer. The input layer is followed by one or more convolutional blocks. A convolutional block starts with one or more consecutive convolutional layers, with the sequence possibly followed by a pooling layer. These blocks are placed one after another, and they may or may not be of the same

structure. The sequence of convolutional blocks is possibly followed by one or more fully connected layers. Finally, the last two layers of a CNN are usually a softmax layer followed by a classification layer. An example of a convolutional neural network processing an MRI image is shown in **Figure 16**.

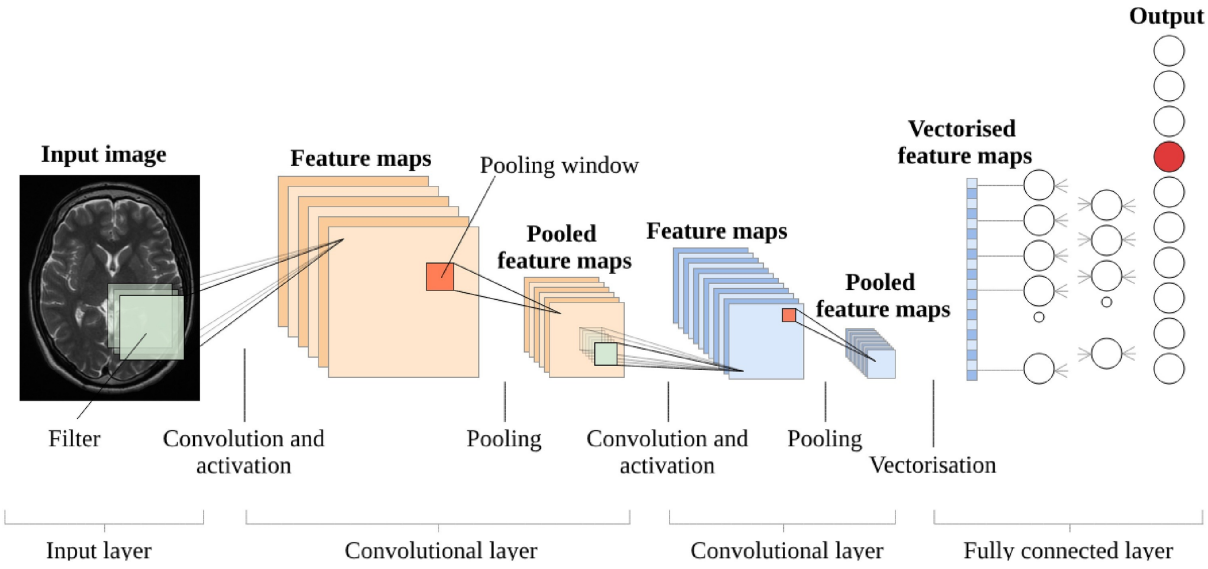

Figure 16: An example of a convolutional neural network. (Source: [44])

We now present each of the layers. For simplicity, we assume the CNN is designed for processing two-dimensional images of size $H \times W \times C$, where H, W, and C are the height, width, and number of channels respectively. We assume for simplicity that each element $I_{i,j,c}$ where $i \in \{1, ..., H\}, j \in \{1, ..., W\}, c \in \{1, ..., C\}$ of an image I is a real value, so each image is a point in space $\mathbb{R}^{H \times W \times C}$. $I_{i,j}$ is now the pixel with indices i and j, the pixel being a vector of all the channel values at that location, i.e., $I_{i,j} = (I_{i,j,1}, ..., I_{i,j,C})$.

2.3.4.1 The Input Layer

Like in multilayer perceptrons, the input layer simply represents the inputs to the network, so that every node in the input layer is an input value. As we have seen, the order of inputs in multilayer perceptrons does not matter as long as it is constant over all the instances in a dataset, and we represented the inputs as a vector of values. For simplicity in further discussion, we represent the input layer of a CNN as a tensor of dimensions $H \times W \times C$, where each node $N^1_{i,j,c}$ of the input layer is the element $I_{i,j,c}$ of the input image. Because CNNs assume spatial structure of data and are sensitive to the shuffling of the elements in an image, the correspondence between the image elements and the input nodes specified by the same indices must hold, otherwise CNNs do not work as intended.

2.3.4.2 The Convolutional Layer

A convolutional layer consists of neurons and takes as input a data volume (three-dimensional tensor of values) of size $H_1 \times W_1 \times D_1$, where H_1, W_1, and D_1 are the height, width, and depth of the input. We note that the first convolutional layer of the first convolutional block takes the input layer as input, so in that case $H_1 = H$, $W_1 = W$, and $D_1 = C$. Each convolutional layer transforms its input volume into an output data volume of size $H_2 \times W_2 \times D_2$, where H_2, W_2, and D_2 are the height, width, and depth of the output. We say that the height and width are the *spatial* dimensions of a volume. The input is transfomed into the output through a differentiable function. Goodfellow et al. [23] state that the transformation is such that it leverages the principles of *sparse interactions, parameter sharing*, and *equivariant representations*, while Zhang et al. [67] describe the principles of *translation invariance* and *locality*. We show that the two sets of principles significantly overlap as we describe them.

The locality principle states that, given a pixel $I_{i,j}$, we do not need look far away from $I_{i,j}$ in order to obtain information most relevant to interpreting the values in the pixel. In the first convolutional layer, this is achieved by the neurons taking pixel inputs only from a restricted and local area of the image. The spatial extent of the connectivity of a neuron is again called its receptive field, and is a hyperparameter of the neuron. Usually, neurons in the same convolutional layer all have receptive fields of the same size and structure. Due to their receptive fields being small and local, these neurons therefore produce responses to local input patterns, called *features*. The same principle is also applied to all of the deeper convolutional layers in the network, so that each layer uses neighboring local features from the previous layer to define larger and more complex features. The neurons in the convolutional layers thus learn patterns of increasing complexity as we move down the network. This is equivalent to the ideas shown through networks A_1 and A_2 in **Section 2.4.2**.

Sparse interactions are a consequence of the locality principle. Because neurons in convolutional layers are not fully connected to their previous layer, CNNs have less parameters than MLPs containing the same number of neurons. This reduction in parameters regularizes the network (thus improving its statistical efficiency), reduces the number of operations required in computing outputs (thus reducing learning time), and reduces the memory requirements of the model. These improvements in efficiency are usually quite large.

In the first convolutional layer, the locality principle is applied to only the spatial dimensions of the input image, and not to the channel dimension, because individual features very often depend on multiple channels. Therefore, the neurons in the first convolutional layer are connected to all channel values in their receptive field (which is defined over the spatial dimensions). The same apllies to the lower layers - we will

soon see that values at different depths of a volume correspond to different features - because individual (larger) features often depend on multiple different smaller features. Therefore, the extent of the connectivity along the depth axis is always equal to the depth of the input volume.

In theory, there are no significant restrictions to designing the receptive fields of neurons, as long as they satisfy the ideas behind locality. Here, we decide to specify the receptive fields algebraically, in the following way. We first assume that the first convolutional layer is of the same shape as the image regarding the spacial dimensions, i.e., a matrix with dimensions $H \times W$. Let $N_{i,j}$ be any neuron in the first convolutional layer at location (i, j) in the spatial dimensions of the layer. We want the neuron to learn features defined on a small area around the pixel $I_{i,j}$. Because images are rectangular, we may let the area be rectangular as well; here, we let the area be a square. Now, we discard the image information outside some range $|a| > \Delta$ and $|b| > \Delta$ from pixel $I_{i,j}$, by setting the weights for such pixels to 0. We obtain the following expression for the activation $y_{i,j}$ of neuron $N_{i,j}$:

$$y_{i,j} = \sigma(\sum_{c=1}^{C} \sum_{a=-\Delta}^{\Delta} \sum_{b=-\Delta}^{\Delta} u_{i,j,a,b,c} I_{i+a,j+b,c} + b),$$

where by $u_{i,j,a,b,c}$ we denote the weight assigned to input $I_{i+a,j+b,c}$ by neuron $N_{i,j}$. Analogous design is applied to the neurons in the deeper convolutional layers; there, the weighted inputs are just summed over different depths of the input volume instead of summing over the channels of the image. The tensor $u_{i,j}$ of all the weights $u_{i,j,a,b,c}$ determines the feature recognized by neuron $N_{i,j}$, so we call such tensors the *feature kernels*, just *kernels*, or *filters*. In the remainder of the work, we always assume that the receptive field of a neuron is rectangular. A visual representation of neurons in a convolutional layer can be seen in **Figure 17**.

We present an example of a feature that the neuron $N_{i,j}$ may learn. Assume that $\Delta = 3$, so the neuron has a receptive field of size 3×3. Then, assume the neuron learns the bias $b = 0$ and the kernel $u_{i,j}$ such that the matrix of weights $u_{i,j,c}$ is the vertical Prewitt operator (defined in **Section2.4.3**) for each $1 \leq c \leq C$. Now, N recognizes vertical edges! Such feature kernels may indeed be learned in practice, but the kernel values are often not as "discrete" and include more noise.

The vertical Prewitt operator is an example of what a neuron in a convolutional layer may learn. Usually, we want the network to be able to detect multiple features at each location, so we let each spatial location in a convolutional layer L contain D_L neurons, D_L specifying the depth of the output volume. We note that each spatial location in a layer contains the same number of neurons.

The principles of equivariant representations and translation invariance state that if a feature is learned to be recognized in one location, the convolutional neural network

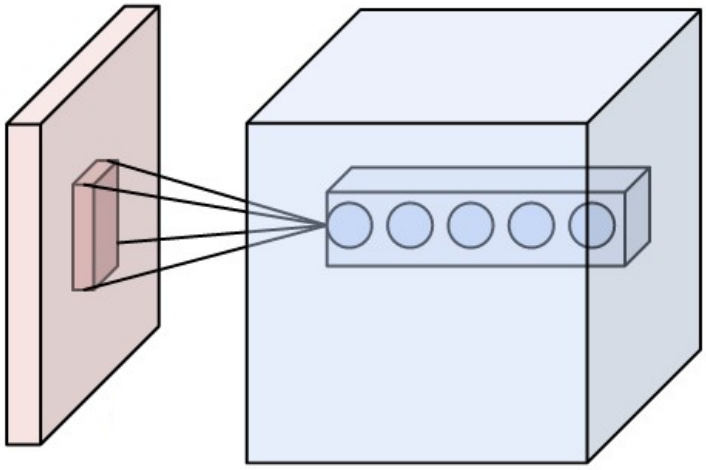

Figure 17: Neurons in a convolutional layer and their receptive field (Source: [13]).

should be able to recognize it no matter in what location it may appear. For example, if a neuron $N_{i,j}$ is activated by a feature at location (i_1, j_1) in the image, then for any location (i, j) there should be some neuron N_2 that is activated by the same feature appearing at location (i, j). This property is called translation invariance, because moving (translating) a feature from one location to another means the network still produces the same response, but now at another location in the convolutional layer.

CNNs implement the translation invariance principle by letting all neurons $N^L_{i,j,d}$ in every location (i, j) share the same weights and bias for each depth $d, 1 \leq d \leq D_L$ and each layer L. This way, all the neurons in a layer at the same depth have the same feature kernel, no matter their spatial location, and they all recognize the same feature. Because of that, the set of all neurons of a convolutional layer at the same depth is called a *feature map*. The output of a convolutional layer L is now a volume containing the activations in all spatial locations for each of the feature maps. Let us look at a single feature map in a single layer, and thus fix the indices d and L. The kernel $u_{i,j}$ of weights no longer depends on the spatial location indices i, j, so the output $y^L_{i,j}$ of each neuron $N^L_{i,j}$ in the map is now:

$$y^L_{i,j} = \sigma\Big(\sum_{d=1}^{D_L-1} \sum_{a=-\Delta}^{\Delta} \sum_{b=-\Delta}^{\Delta} u_{a,b,d} I_{i+a,j+b,d} + b\Big).$$

The set of weighted sums of inputs to all of the neurons is now a convolution of the kernel and the input volume along their spatial dimensions! More precisely, the operation is a cross correlation. As we move down the network, the feature maps represent features of increasing complexity. The shallow convolutional layers learn "low-level" features, while the deeper layers learn "high-level" features; this can be seen in **Figure 18**, which shows an example of features that may be learned by different convolutional layers in AlexNet [34].

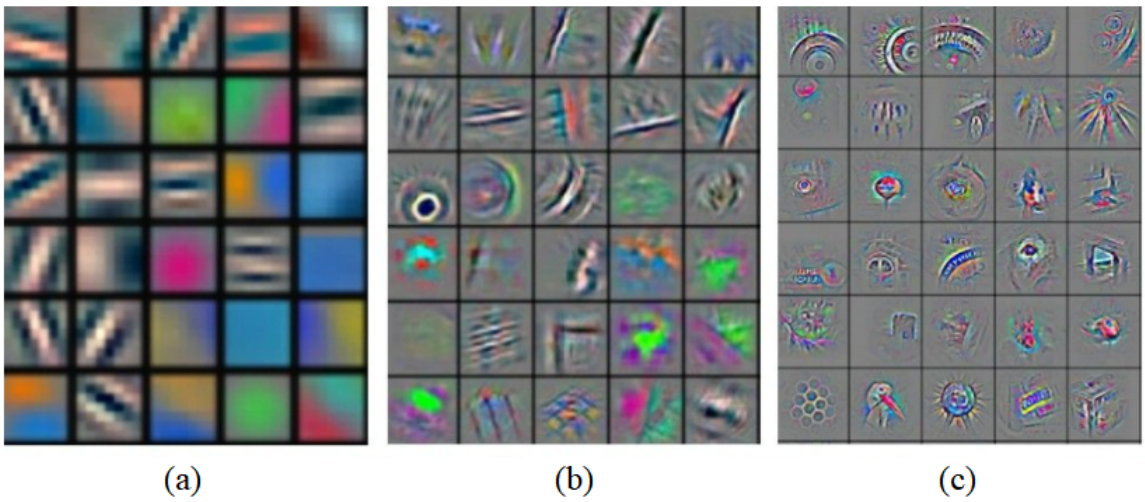

Figure 18: (a) low-level, (b) mid-level, and (c) high-level features in AlexNet.

We note that convolutional neural networks are technically not translation invariant, but rather translation *equivariant*, meaning that a translation transformation applied to the input is transferred to the output. Furthermore, multiple studies have shown that CNNs have highly limited translation equivariance; for a detailed discussion of the topic see the work by [7]. A single kernel of weights and the bias, which are shared among all of the neurons of a feature map, are the only parameters of the feature map. This is the idea of the parameter sharing principle. The sharing of parameters may be regarded as a regularization mechanism when compared to a layer with the same number of connections but with independent weights. Furthermore, the single kernel can be stored at a single place in memory. In the learning algorithm, we simply treat the feature map as contributing the single kernel of weights and the bias to the set of solution variables.

In most modern convolutional neural networks, the neurons in the convolutional layers use the Rectified Linear Unit (ReLU) function r as the activation function:

$$r(x) = \max(0, x).$$

The graph of the function is shown in **Figure 19**. The ReLU was used as the activation function in neural networks as early as 1969 by Fukushima [20]. It was later found that the function enables better training of networks [22], and argued that it has strong biological motivations [24]. Sometimes, the use of a ReLU function in a convolutional layer is indicated by placing a "ReLU layer" after the convolutional layer; we follow this convention in the thesis unless specified otherwise.

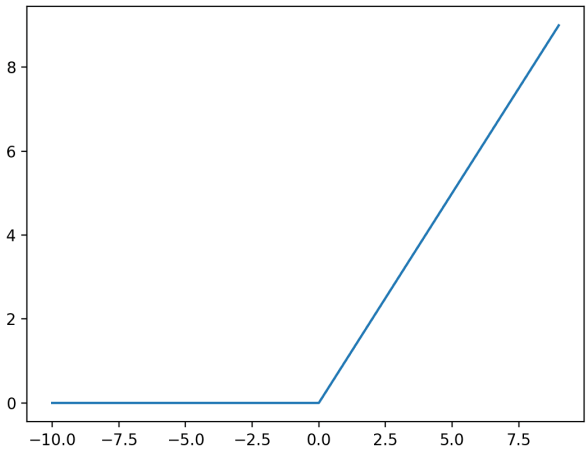

Figure 19: Rectified Linear Unit (ReLU).

ReLU activations have several advantages compared to the sigmoid ones. First, sigmoid activations have the problem of vanishing gradients, where their gradient becomes negligibly small, thus preventing the associated weights from changing their values during learning. Due to its derivative, the ReLU activation does not suffer from vanishing gradients as long as it "fires" (is positive) for some inputs. If the activation is zero for essentially all inputs, the weights of the neuron may get stuck in a perpetually inactive state. This is another form of the vanishing gradient problem, called the "Dying ReLU problem"; the problem an2.8 its possible solutions are decsribed in [43]. The second advantage of the ReLU activation function is that it allows for efficient computations due to its simple form, requiring only the comparison, addition and multiplication operations. Finally, the function is scale-invariant, as $\max(0, \alpha x) = \alpha \max(0, x)$ for all $\alpha \geq 0$. We note that r is not differentiable at 0, but the derivative there can arbitrarily chosen to be 0 or 1. In this work, we use convolutional layers that have ReLU activations.

2.3.4.3 The Pooling Layer

The function of pooling (also called *subsampling*, or *downsampling*) layers is to progressively reduce the spatial size of the data volumes in the network. For every depth "slice" of its input volume, a pooling layer splits the slice into rectangles - or *filters*) - and, for each rectangle, performs an operation on its values and outputs only the result of the operation. The most common pooling layer is the one with stride 2, filters of size 2×2, and that performs the max operation. Such a layer outputs only the maximum value for each of the filters, thus reducing the size of the input volume by 75%. In general, pooling using the max operation is called *max pooling*. An illustration of performing max pooling on the activations of a single feature map is shown in **Figure 20**.

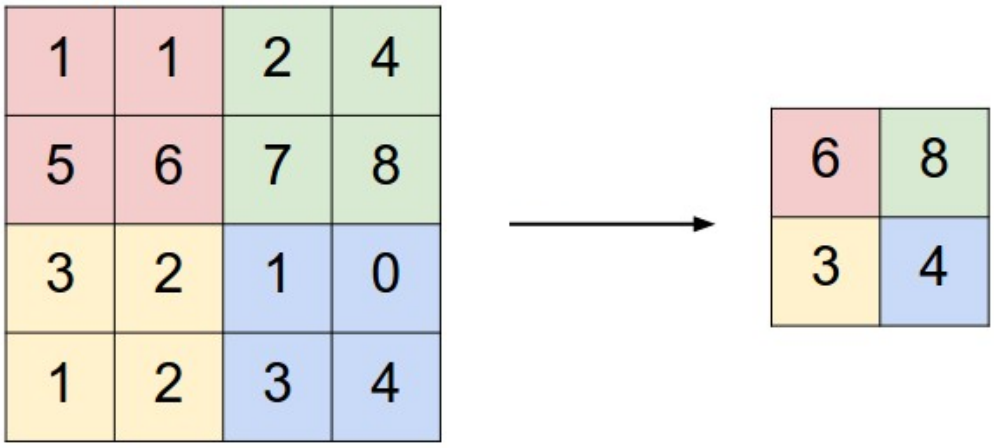

Figure 20: Input and output of a max pooling layer.

We describe the pooling layer more formally. The input volume V of the layer is a tensor of rank 3 and size $H_1 \times W_1 \times D_1$. Then, p-norm pooling with pooling size z and stride r applied to volume V is tensor $s(V)$ of rank 3 with the following values:

$$s_{i,j,u}(V) = (\sum_{h=-\lfloor k/2 \rfloor}^{\lfloor k/2 \rfloor} \sum_{w=-\lfloor k/2 \rfloor}^{\lfloor k/2 \rfloor} |V_{g(h,w,i,j,u)}|^p)^{1/p},$$

where $g(h, w, i, j, u) = (r \cdot i + h, r \cdot j + w, u)$ is the function mapping positions in s to positions in V respecting the stride, and p is the order of the p-norm. For $p \to \infty$, the pooling is max pooling. Pooling does not have parameters, and p, z, and r are the hyperparameters of layer.

The use of pooling layers in a CNN produces multiple benefits. First, they significantly reduce the amount of data that is passed through the network. For instance, the size of the output volume of a pooling layer with filters of size 2×2 and with stride 2 is only $\frac{1}{4}$ the size of its input volume. Therefore, the succeeding convolutional layer in the network has to perform only $\frac{1}{4}$ of the operations compared to the situation where the pooling layer is not used, and produce only $\frac{1}{4}$ of the activations to be stored in memory. The number of weights in the layer is also reduced, therefore regularizing the network. The features in the layer will also cover areas of the image that are four times as large, thus reducing the network depth needed to cover the entire image. If pooling is used frequently (for example, after every convolutional layer) then the needed depth is logarithmic in the height and width of the image. This reduction in depth further reduces the required memory and the number of computations performed, and further regularizes the network. Finally, as a pooling layers converts multiple neighboring values of its input V into a single output value, the succeeding layer in the CNN becomes more invariant to small deformations of features in V. For example, using a pooling

layer after the first convolutional layer in the network makes the network ignore some potential image deformations that are small enough to fit into a filter of the pooling layer.

We note that, while convolution layers may be implemented by fully connected layers (by setting the weights outside of the receptive field to 0 for each neuron in the layer), a max pooling layer cannot. This is due to the fact that neurons apply an activation function on the weighted sum of their inputs summed with the bias. On the other hand, the inputs to the max pooling layer are vectors of values in a filter. If the values were weighted and summed, the information of the maximum value would be lost, as vectors with different values could produce the same sum.

2.3.4.4 Putting the layers together

After the input layer and the sequence of convolutional blocks, a convolutional neural network may contain a sequence of fully connected layers. The fully connected layers are the same ones used in multilayer perceptrons. In a CNN, they increase the number of parameters, thus increasing its expressive power. In particular, they may be interpreted as enabling the network to process the activations in the output volume of the last convolutional block without assuming the locality principle on the features in the volume. However, we note that using fully connected layers does not improve results in many applications, so some modern CNN models do not include them.

A CNN uses the softmax layer followed by a classification layer to produce its output. For an input image I, we want the network to assign probabilites of I belonging to each of the classes. Because the classes are mutually exclusive, the network should output a probability distribution, i.e., output probability P_K for each class K such that the probabilites are non-negative and sum to 1. The network achieves this through a softmax layer, which applies the softmax functions to its inputs. The classification layer finally gives the output of the network. As an example of a complete convolutional neural network, we show the popular architecture of AlexNet in **Figure 21**, and its illustration in **Figure 22**.

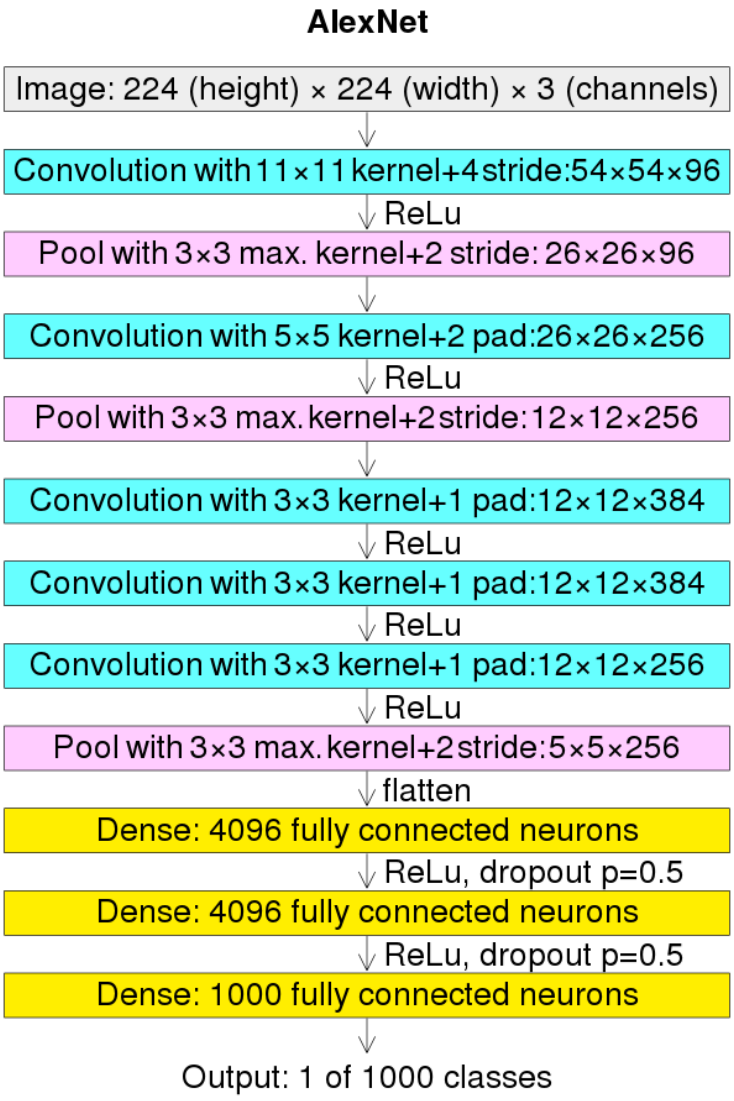

Figure 21: Architecture of AlexNet.

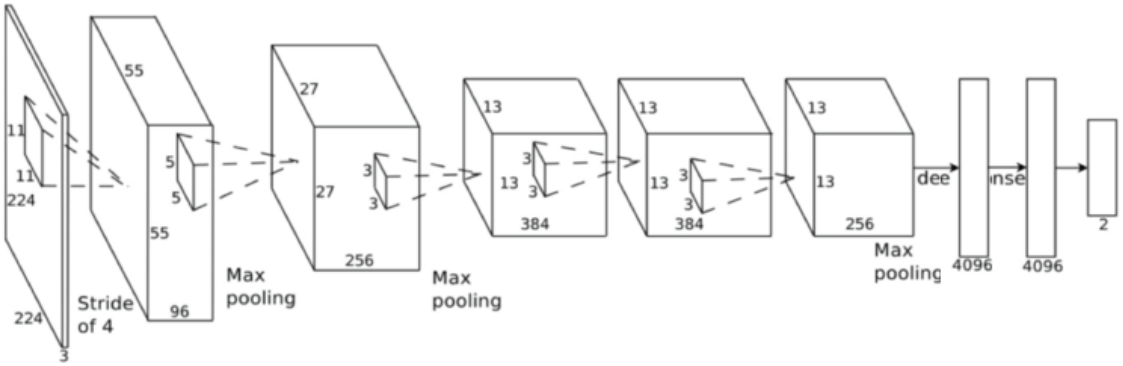

Figure 22: Illustration of AlexNet.

3 PROBLEM STATEMENT

In this chapter, we motivate and define our problem. Our task is to develop an automatic classifier of a dataset of MRI images in multiple sclerosis patients, potentially aiding medical experts in diagnosis. The image dataset consists of volumetric (three-dimensional) images with three channels: FLAIR MRI, QSM MRI, and a mask channel indicating the locations of the lesions. We classify the images into demyelinated and remyelinated types, indicating different expectations for patient state - worsening and improving, respectively. Automating the process of classification could help medical doctors in diagnosing patients more efficiently and accurately. We first motivate our problem by providing its medical background in **Section 3.1**, and then formally define our problem in **Section 3.2**.

3.1 MEDICAL BACKGROUND

Multiple sclerosis (MS), also known as encephalomyelitis disseminata, is a chronic, inflammatory disease that involves immune-mediated attacks on the central nervous system (CNS). It is the most common demyelinating disease [6], in which the insulating covers of nerve cells in the brain and spinal cord are damaged. An estimated 2 million people worldwide currently have the disease [56]. MS is not fatal, but results in a range of sings and symptoms. Interestingly, there is no known cure for the disease [56]. MS is usually diagnosed based on the present signs and symptoms, and the results of medical imaging and laboratory testing. A diagnosis can be difficult to confirm, especially in the early stages of MS, since the signs and symptoms can be similar to ones caused by other medical problems.

Multiple sclerosis has three main characteristics in regards to pathophysiology (study of the disordered physiological processes associated with a disease or injury): lesion formation in the central nervous system, destruction of myelin sheaths of neurons, and inflammation. Lesions are areas of tissue abnormality (injury, scars). The name *multiple sclerosis* is due to the scars (sclerae) that form in the nervous system. In MS, there is a loss of oligodendrocytes, the cells which create and maintain the myelin sheath, a fatty layer around nerve cell axons. The MS lesions refer to the tissue areas of damage to the myelin sheats.

Myelin sheaths insulate neuron axons, thus helping the axons carry electrical sig-

nals between neurons. MS lesions typically affect *white matter*, the areas of the CNS mostly made up of myelinated axons (distinguished from *gray matter*, which contains numerous nerve cell bodies and relatively few myelinated axons). An MS lesion is shown in **Figure 23**. In general, the presence of damage to the myelin sheath is called *demyelination*. If the myelin sheath is lost, a neuron can no longer effectively conduct electrical signals [12]. *Remyelination*, a repair process which forms new (but usually thinner) myelin sheaths, takes place in early phases of MS. In our work, we classify lesions into demyelinating and remyelinating ones. The two types of lesions carry different implications for patient state, indicating a potential worsening and potential improvement of the state, respectively.

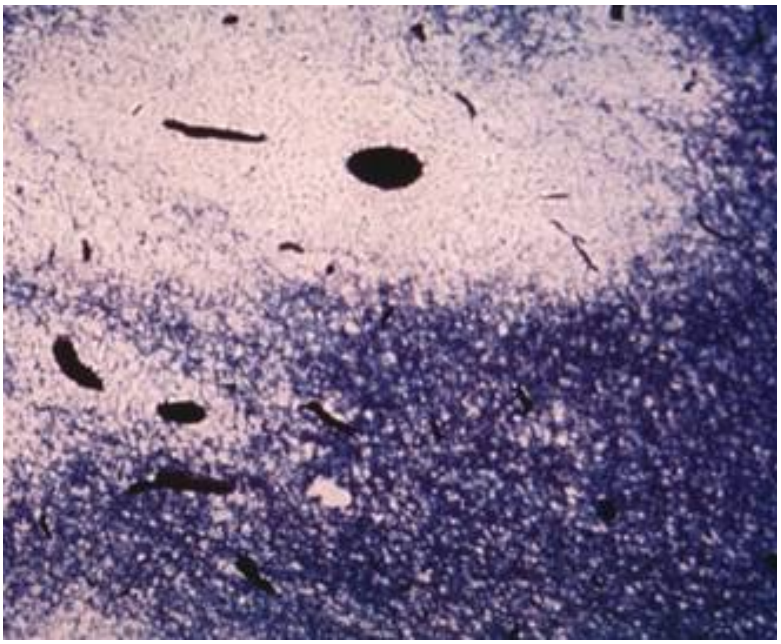

Figure 23: An MS lesion: myelin - blue, lesion - white, blood vessel - black (Source: [52]).

Inflammation is another significant sign of mutliple sclerosis. In MS, the body recognizes myelin as foreign and attacks it; the attack then starts inflammatory processes. As this represents a dysfunction of the immune system, MS is an immune-mediated disorder.

The damage done to myelin sheats in multiple sclerosis disrupts the ability of neurons to transmit signals, and thus may result in a range of symptoms. The symptoms include physical, mental, and psychiatric problems. MS is not fatal, but some of the symptoms may increase risk of injury or death. MS may cause [12]:

- *vision problems* such as blurred vision, unilateral vision loss, diplopia, oscillopsia, nystagmus,

- *sensory problems* such as numbness, dysethesias, paresthesias, Lhermitte's sign, squeezing around torso, proprioception deficits,
- textitmotor problems such as trunk or extremity weakness, hyperreflexia, spasticity, gait disturbance, imbalance,
- *Cerebellar problems* such as tremor, ataxia, incoordination,
- *Elimination dysfunction* such as urinary frequency, urgency or retention, incontinence, constipation,
- *Mood and cognition disorders* such as depression, anxiety, or impairment of memory, concentration, attention, or speed of information processing.

So far, the exact cause of multiple sclerosis is unknown. The disease is believed to occur as a result of a combination of environmental and genetic factors. The environmental factors include geographical factors, infectious agents, and risks factors such as smoking and obesity [48]. On average, MS occurs more frequently in areas farther from the equator [12]; the cause of this phenomenon is not clear [48]. Multiple microbes and viruses have been proposed as causes of MS, but no claim has yet been confirmed [12]. Regarding genetic factors, MS is not considered to be a hereditary disease, but it is known that multiple genetic variations do increase the risk of MS occurring.

Currently, there is no single test that can provide a definitive diagnosis of multiple sclerosis in living humans; in post-mortem autopsy, MS can be detected through histopathological techniques [47]. The two most significant components of MS diagnosis are recognizing sings and symptoms, and medical imaging and laboratory testing. The 2017 revision of McDonald criteria [47] is the most commonly used method for diagnosis. The McDonald criteria rely on clinical, radiologic, and laboratory methods for finding evidence of MS lesions in different areas and at different times. The criteria relies, among other evidence, on magnetic resonance imaging findings, which is the method of most interest in this work.

3.1.1 Magnetic Resonance Imaging and Multiple Sclerosis

Magnetic resonance imaging (MRI) is a non-invasive medical imaging technique that produces detailed three-dimensional images of the anatomy and physiological processes of the body. MRI was invented in 1971, when Lauterbur [38] used magnetic field gradients in three spatial dimensions and a back-projection technique to create images. Mansfield [45] further improved these techniques, and the two shared the Nobel Prize in Physiology or Medicine in 2003 for their contributions. MRI scanners are the devices that perform MRI, and they use strong magnetic fields, magnetic field gradients, and radio waves to generate images of the body. An MRI scanner is shown in **Figure 24**. MRI is distinguished from CT and PET, other commonly used imaging methods, by the fact that it does not use X-rays or ionizing radiation to perform imaging.

Figure 24: An MRI scanner.

In most medical applications of MRI, a strong magnetic field is used to line up a percentage of hydrogen protons in the direction of the field. Then, pulses of radio waves and additional weaker magnetic fields are used to excite the atoms. By varying the parameters of the pulses, different contrasts are obtained between tissues based on the relaxation properties of their hydrogen atoms. Hydrogen atoms are naturally abundant in biological organisms, particularly in water and fat. Because the myelin sheaths on nerve cells axons is fatty, it repels water. Multiple sclerosis lesions, which are damaged ares of the sheath, repel less water than the parts of the sheath surrounding them. Thus, MRI scans show MS lesions as either darkened or brightened areas, depending on the type of scan that is used; such a scan is shown in **Figure 25**. Due to the quality of generated images and its nonintrusive nature, MRI is one of the most important diagnostic tools for MS.

Studies have shown that multiple sclerosis lesions are heterogeneous in terms of myelin damage and repair, as well as iron content [37]. Recently, Quantitative Susceptibility Mapping (QSM) MRI has been shown to allow classification of MS lesions into demyelinated and remyelinated types [57]. In QSM MRI, concentrations of substances in the body are measured by quantifying the spatial distribution of magnetic susceptibility in the tissues. In this work, we will use a dataset of MS lesions classified using QSM. The dataset also includes fluid-attenuated inversion recovery (FLAIR) MRI images of the lesions.

Medical images, including all types of MRI, are typically manually processed by medical experts, which can prove to be a laborious and time consuming task. By creating an automatic MS lesion MRI image classifier, we attempt to potentially help medical experts in MS by reducing the time and energy needed for diagnosis and

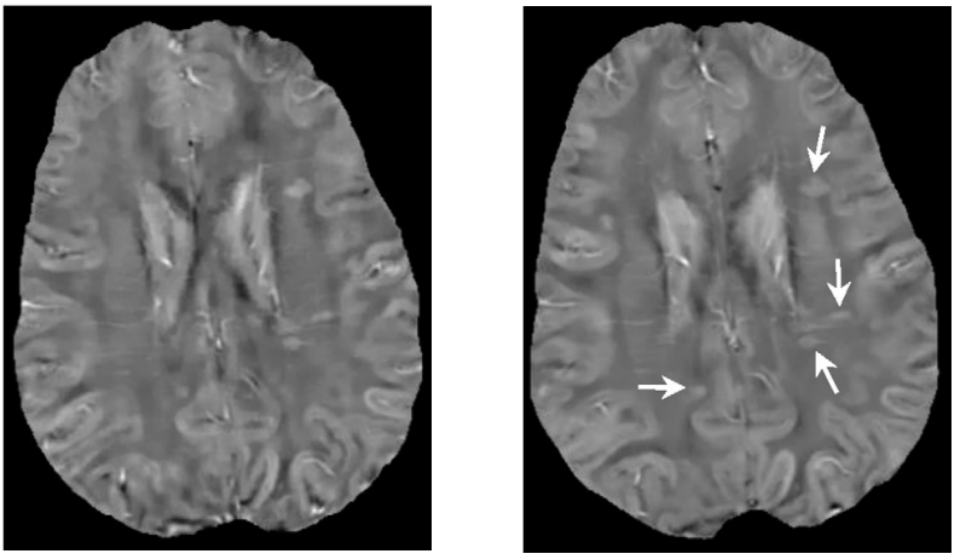

Figure 25: A QSM MRI scan of MS lesions - the lesions are indicated by white arrows (Source: [10]).

possibly improving diagnosis accuracy.

3.2 PROBLEM DEFINITION

The dataset we work with is the one described from the work of Reza et al. [57]. The data was originally obtained in the following way. 91 multiple sclerosis patients underwent 3T MRI (MRI with magnetic field strength of 3 Tesla) of different types. These were 3D FLAIR 1mm isotropic, MP2RAGE (magnetization-prepared 2 rapid acquisition gradient echoes) MRI 1mm isotropic, and 3D EPI 0.67mm isotropic for QSM.

The FLAIR and MP2RAGE MRI images were used to perform segmentation of MS lesions. Segmentation was first performed by an automatic segmentation procedure and then followed by manual correction. The automatic procedure is similar to the one described by La Rosa et al. [35]. In the work, a modification of the U-Net [58], a popular convolutional network for biomedical image segmentation, is used on FLAIR and MP2RAGE MS lesion images of size $88 \times 88 \times 88$ to produce binary mask images of size $44 \times 44 \times 44$. In each mask image, the location of the MS lesions is indicated by white voxels and the background is black. An illustration of the procedure used in the work by La Rosa et al. [35] is shown in **Figure 26**. In our dataset, similar automatic segmentation was performed on the FLAIR and MP2RAGE images in the dataset, and the results were then manually corrected as needed. The final results of the segmentation were mask images (masks) where lesion location is again indicated by white voxels and the background is black. The masks were co-registered to 3D EPI.

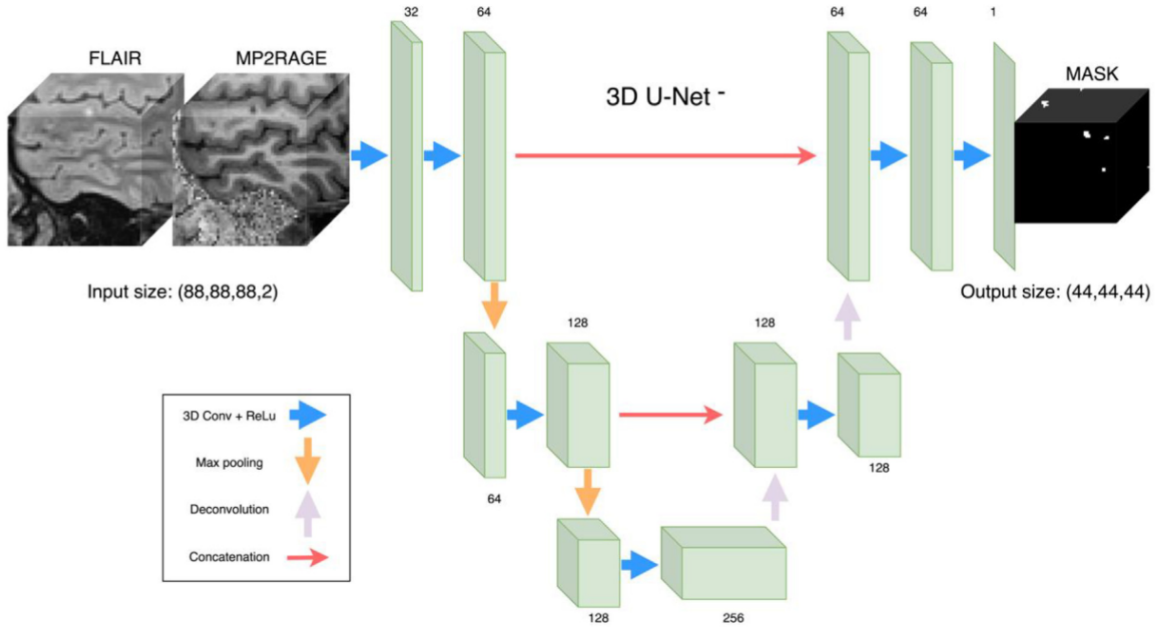

Figure 26: Segmentation of lesion images by U-Net (Source: [35]).

In total, 5250 lesions were registered (one patient may have numerous multiple sclerosis lesions). For each lesion, a patch of $35 \times 35 \times 35$ voxels was created such that the lesion of interest is centered in the patch. In the remainder of the paper, we use the term "image" when referring to these patches as well. Our dataset consists of these images, where each image is a $35 \times 35 \times 35$ voxel image with three channels: FLAIR, QSM, and mask, the mask specifying the location of the lesion within the image (patch).

Following the segmentation, the lesions were manually classified by medical experts based on their qualitative properties in QSM images. The first study on classifying MS lesions into the classes we present was given by [57] in 2021. Each lesion in our dataset belongs to one of 6 classes. We present the classes and their visual properties in QSM images:

- **Class 1**: *unclassified lesions*: lesions that could not be classified into one of the classes **2-6** by the medical experts. The class is very heterogeneous. There were multiple reasons preventing lesions from being classified; some of them are as follows. In some cases, the lesion were formed by multiple lesions from different classes **2-6** merging together into one larger lesion. In other cases, big vessels traverse the lesion area. In yet other cases, the images in this class include image artifacts which prevent them from being classified.
- **Class 2**: *isointense lesions*: lesions that show no difference in intensity in QSM compared to the surrounding tissue.
- **Class 3**: *PRL lesions*: lesions whose borders have larger values compared to the

inside and the outside of the lesion.

- **Class 4**: *hypointense RIM lesions*: lesions whose borders have values larger compater to the center of the lesion.
- **Class 5**: *hyperintense lesions*: lesions where the inside of the lesion has larger values compared to its surroundings, and no specific delineation of the border is present.
- **Class 6**: *hypointense lesions*: lesions where the value inside the lesion has smaller values compared to its surroundings, and no specific delineation of the border is present.

We show examples of images in each of the classes in **Figure 27**, each image containing the three channels. The number of lesions in each class is as follows: class 1 contains 3664 lesions, class 2 contains 460 lesions, class 3 contains 214 lesions, class 4 contains 19 lesions, class 5 contains 841 lesions, and class 6 contains 71 lesions. We notice that the majority of the images were not able to be classified.

Importantly for our work, the lesions in the dataset can be classified into *remyelinating* and *demyelinating* ones based on the presented classification: lesions in classes 2 and 6 are remyelinating, and lesions in classes 3 and 5 are demyelinating. This classification specifies the *remyelination status* of a lesion. Class 1 can be considered heterogeneous in this regard, while class 4 is kept out of the classification. We show the lesion classes, along with their remyelination status and the number of lesions in the class (i.e., the class size), in **Table 1**.

Table 1: Lesion classes.

Class	**Name**	**Size**	**Rem. status**
1	Unclassified	3664	/
2	Isointense	460	remyelinating
3	PRL	214	demyelinating
4	Hypointense RIM	19	/
5	Hyperintense	841	demyelinating
6	Hypointense	71	remyelinating

Importantly for diagnosis, the remyelination status of a lesion carries possible implications for patient states. Remyelinating lesions indicate that the remyelination repair process has occurred, indicating that the patients state may improve. On the other hand, demyelinating lesions are more destructive and indicate that the patients state may worsen. Therefore, a model that automatically classifies MS lesion into remyelinating or demyelinating ones would potentially improve the efficiency and the accuracy of diagnosis.

We define the goal of our research as follows. We want to build a classification model that, given an image of an MS lesion belonging to class 2, 3, 5, or 6, correctly predicts the remyelination status of the lesion. An input image has the same format as the ones in our dataset. The model should correctly classify the images in our dataset, but should also generalize to new images. Furthermore, we wish to analyze the effects of using different channels for classification. More specifically, we want to compare the effects of using each subset of the set of channels {FLAIR, QSM, mask} for classification, and compare the results of such classifiers.

In the next chapter, we design such a classification model. In **Chapter 5**, we present the results of running the model on our dataset. We evaluate the model by computing its *accuracy*, as well as its *sensitivity*, *specificity*, *precision*, and *negative predictive value* (NPR). Accuracy is simply the percentage of lesions that the model classifies correctly. We label the "remyelinating" type as positive, and "demyelinating" as negative; the assignment of labels to the types is arbitrary. Here, we let "R" and "D" stand for "remyelinating" and "demyelinating", respectively. The sensitivity, or *true positive rate*, and the specificity, or *true negative rate*, are now defined as follows:

$$\text{Sensitivity} = \frac{\text{num. of "R" lesions classified as "R"}}{\text{total num. of "R" lesions}},$$

$$\text{Specificity} = \frac{\text{num. of "D" lesions classified as "D"}}{\text{total num. of "D" lesions}}.$$

The precision and negative predictive value are:

$$\text{Precision} = \frac{\text{num. of "R" lesions classified as "R"}}{\text{total num. of lesions classified as "R"}},$$

$$\text{NPR} = \frac{\text{num. of "D" lesions classified as "D"}}{\text{total num. of lesions classified as "D"}}.$$

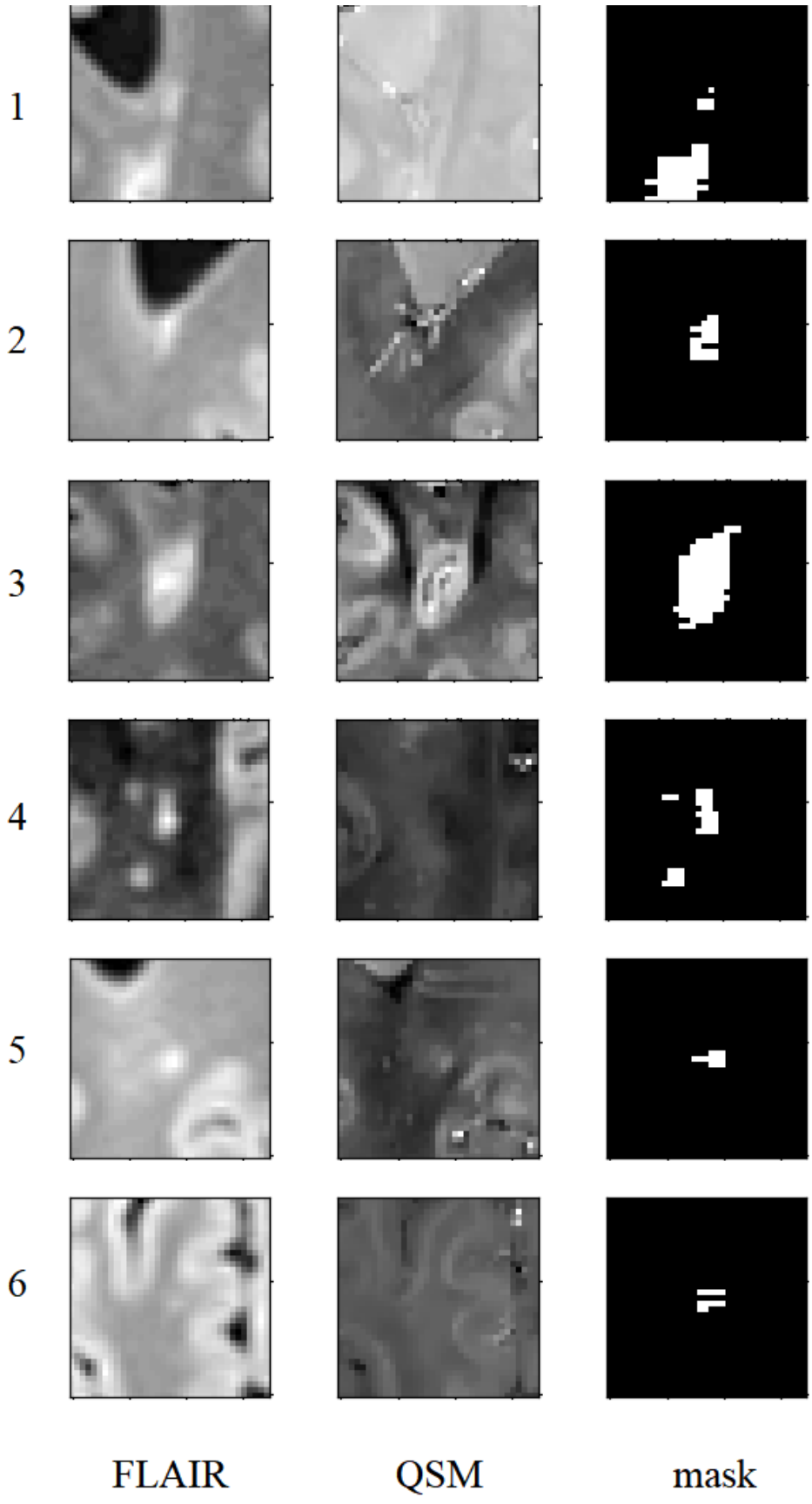

Figure 27: Image examples for each class in our dataset.

4 MODEL

In this chapter, we present our classification model for the problem defined in **Section 3.2**. Because we want the model to generalize beyond our dataset, we treat the problem as a machine learning one, and use a convolutional neural network architecture for classification. We split the dataset into a *training set*, used for training the network, and a *test set* - used for testing the network. Then, we modify the dataset in order to make it more suitable for learning. We design the architecture of the convolutional neural network and specify the parameters used in learning. In **Chapter 5**, we finally evaluate the model.

We discuss splitting the dataset for the purposes of training and testing in **Section 4.1**. We present the data manipulation techniques we use in **Section 4.2**. In **Section 4.3** and **Section 4.4**, we describe the convolutional neural network architecture and the learning parameters that achieved the best results on the dataset, respectively.

4.1 DATASET SPLIT

Because we want our model to generalize beyond our dataset, we train it on one part of the dataset, called the *training set*, and then evaluate on another part of the dataset, called the *test set*. The performance of the model on the test set indicates its generalization capability. We use a 70/30 split: the training set consists of 70% of the images and the test set consists of the remaining 30% of the images; the two sets are disjoint. More precisely, the training set consists of 70% of the images in each of the classes 2, 3, 5, and 6, while the test set consists of the remaining images in these classes. The 70/30 split is commonly used, as it is such that the training set has high probability of containing sufficient information for learning and the test set has high probability of having a probability distribution sufficiently similar to that of all of the data, including unseen one.

4.2 DATA MANIPULATION

Before training our model on the training set, we need to address two problems imposed by our dataset. The first problem is that the dataset is imbalanced: it contains approximately twice as many demyelinating lesions as remyelinating lesions. Optimally,

we would want the dataset to contain approximately the same number of images of each type. We solve this problem by *oversampling*. The second problem concerns the dataset size itself. The dataset contains only 1586 images, which is a small number compared to standard computer vision and image processing datasets. We increase its size significantly using *data augmentation*. We describe oversampling in **Section 4.2.1**, present the data augmentation techniques we use in **Section 4.2.2**, and finally discuss a numerical problem we encounter in **Section 4.2.3**.

4.2.1 Oversampling

The first problem in our dataset is that it is imbalanced. Specifically, there are approximately twice as many demyelinating (1055) lesions as there are remyelinating lesions (531). Beacause our model classifies each lesion into exactly one of the two types, it is optimal if the training set contains the same number of images belonging to each type. Otherwise, the convolutional neural network may learn to prioritize the more numerous type in classification, i.e., the network may incorporate a bias towards demyelinating lesions.

To see why this is so, observe the situation where the network is not able to learn lesion features that help it correctly classify the lesions, and hence may resort to a more "randomized" classification. Assume there are fixed numbers d and r of demyelinating lesions and remyelinating lesions respectfully, where $d > r$. Assume further that the network classifies each lesion as demyelinating or remyelinating with probabilities P_d and P_r respectfully, where P_d and P_r form a probability distribution. The expected number n of lesions in the dataset that are classified correctly is then:

$$n = dP_d + rP_r = dP_d + r(1 - P_d) = (d - r)P_d + r$$

The total cost of classifying the dataset is dependent on n, and is expected to decrease as n increases. From the expression we can see that, because $d > r$, n increases with P_d and decreases with P_r. We may (informally) write $\mathrm{d}n/\mathrm{d}P_d = (d-r)$, so the dependence of the cost on P_d and P_r is minimized by decreasing $|d - r|$.

A bias toward a more numerous type is undesirable, even if the test set is imbalanced the same way as the training set. This is because such a bias is a result of pure dataset statistics, rather then of the information and features contained in the images themselves. We solve this problem by letting the number of demyelinating lesions be approximately equal to the number of remyelinating ones. To do so, we simply *oversample* the less numerous type: we increase the number of remyelinating images by copying them. Because there are approximately twice as many demyelinating lesions as there are remyelinating ones, we copy each remyelinating lesion once. The dataset now contains 1055 demyelinating lesions and 1062 remyelinating lesions. The lesion

copies do not introduce new image-specific information into the training set. Rather, they prevent the network from learning a bias from data imbalance.

We perform oversampling only on the training set and leave the test set imbalanced. This is due to three reasons. First, the test set does not influence the network during training, so it does not introduce a bias. Second, the data imbalance is a property of the original dataset, so by evaluating the model using the imbalanced test set we are remaining faithful to original data. Third, when testing, we are able to see the performance of the network on the remyelinating and demyelinating types separately, in the form of *true positive* and *true negative* rates for each type, allowing us to evaluate the performance of the model regardless of the lesion distribution in the test set.

4.2.2 Data Augmentation

The second problem with the dataset concerns its size and requires a more complicated solution than does the data imbalance problem. After oversampling, the dataset contains 2117 lesions. Medical imaging datasets are usually small [60], and our dataset is large in that regard. However, it is small compared to popular 2-D image datasets such as MNIST [16] and ImageNet [17], which contain tens of thousands and tens of millions of images, respectively (but also include many more classes, making classification much more difficult).

A small dataset size is a problem for neural networks due to multiple reasons. First, the larger the training set, the more information is available for the network to learn. Second, a network that has the right structure and a large enough number of parameters may in theory be powerful enough to "memorize" large parts of the training set, instead of learning the important image features shared among all of the data. In that case, the network would perform well on the training set but poorly on the test set, i.e., it would overfit the training set. A larger training set is harder to memorize, and forces the model to learn the general features of the data, increasing its generalization power.

For these reasons, we increase the dataset significantly by performing data augmentation on the training set. Data augmentation means increasing the dataset by adding slightly modified copies of existing data, thus producing new data expected to be faithful to the original dataset. In our case, we generate new lesion images based on the existing ones in the dataset, using rotation and flip transformations. Because CNNs are not equivariant to rotations and flips of input images, the augmented images provide the network with more information in learning. First, we rotate the images by multiples of 90 degrees, i.e., 90, 180, and 270 degrees, around any combination of the three axes, including consecutively using different axes. Each of the image channels is rotated in the same way. The choice of angles is due to the fact that rotating by angles

that are not multiples of 90 degrees requires interpolation when computing new voxel values, potentially changing the fine-grained information contained in the images. An illustration of this fact can be seen in **Figure 28**, where a rectangular grid does not overlap a slightly rotated copy of itself.

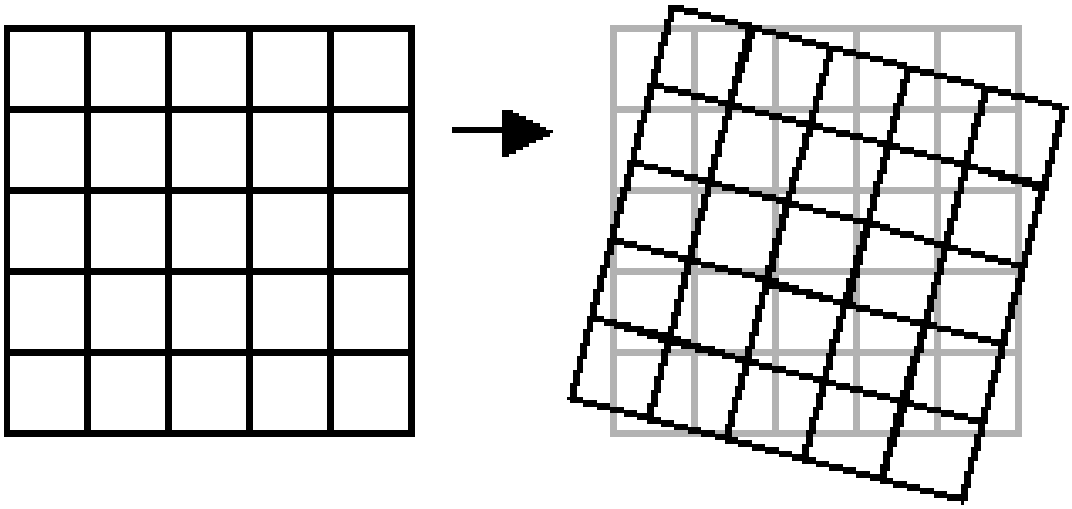

Figure 28: Rotation by angles that are not multiples of 90 degrees requires interpolation.

We can consider the lesion images to have the shape of a cube. The number of unique images that can be obtained by rotations by multiples of 90 degrees is equal to 24, i.e., to the number of rotational symmetries of a cube. The number is computed as follows. Observe a cube with adjacent vertices A and B. As a cube has 8 vertices, we can place the vertex A into a cube 8 different ways. As each vertex of a cube has three adjacent vertices, once A is fixed, we can place vertex B into a cube in three different ways. The locations of all the vertices are determined once both of the vertices A and B are fixed, therefore the number of rotational symmetries of a cube is $8 \times 3 = 24$. As the first step of augmentation, we generate unique 24 images for each image in the dataset by rotation transformations.

After rotation, we further augment the training set by flip transformations. As a single flip is never equivalent to a series of rotations, a flip transformation produces a new image. However, a single flip across one axis may be equivalent to a flip across another axis followed by a series of rotations, thus not producing a new image. We therefore only flip the images vertically, doubling the images in the training set. Now, after the augmentation by rotation transformations and flip transformations, our dataset increases 48 times and contains 101616 lesion images in total. We show an example of images generated from an original image in **Figure 29**.

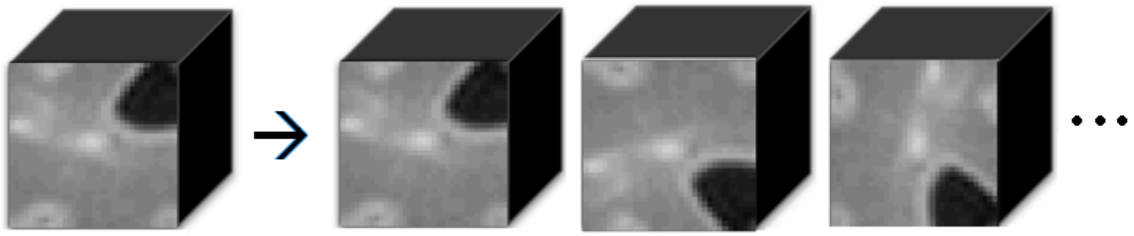

Figure 29: Producing augmented images by rotation and flipping.

4.2.3 Correcting Unsuitable Values

We describe a technical point that proved important in using the classification model. Upon manually inspecting the dataset, we encountered multiple images with QSM values that are unsuitable for using neural networks. Displaying the QSM channel of such images does not give the expected result. We show an example of the QSM channel of such an image in **Figure 30**; notice that the image appears binary (black and white) instead of appearing grayscale like the QSM images in **Figure 27**. We note that **Figure 30** is displayed using Python's "plt" function, which assigns color values to pixels linearly, relative to the maximum and minimum pixel value in the image. More specifically, let v_{min} denote the smallest value in an image and v_{max} denote the greatest value in the image. Then, a pixel with value v is assigned a gray color value of $(v - v_{min})/(v_{max} - v_{min})$.

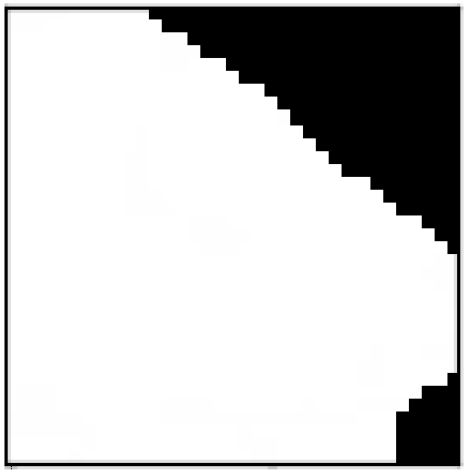

Figure 30: The QSM channel of an image with unsuitable values.

In the images, the black voxels have QSM values of -32768. The origin of the values is due to *skullstripping*, or *brain extraction*, where voxels that belong outside the brain are indicated by a special value, in this case $-(2^{15})$. This value is much smaller than the majority of the typical QSM values in our dataset, which fall in the range $[-200, 200]$.

Computing a histogram of the QSM values in the dataset shows that approximately 37% of the images contain the value -32768, often in large quantities.

We show why such a value may prove problematic for convolutional neural network use. Assume a neuron N in the first convolutional layer of the network assigns a kernel $\mathbf{w}$ of weights to its input $\mathbf{x}$, where we represent the kernel $\mathbf{w}$ as a vector of weights $\mathbf{w} = (w_1, w_2, ..., w_n)$ and the input as a vector of input values $\mathbf{x} = (x_1, x_2, ..., x_n)$, where each weight w_i is assigned to input x_i, for simplicity. Furthermore, the neuron has bias b. Observe two cases of inputs, x^1 and x^2, where $x_i^1 \in [-200, 200]\ \forall i$, while $x_1^1 = -(2^{15})$ and $x_i^2 = x_i^1\ \forall i \in 2, ..., n$. In other words, x^2 is obtained by replacing the first input in x^1 by a value of $-(2^{15})$. We have $|x_1^2| \gg |x_1^1|$. Assume $(|w_1| \not\ll |w_i|)\ \forall i \in \{1, ..., n\}$, meaning that the weight assigned to input x_1 is significant, and assume $b \not\gg \mathbf{w}\mathbf{x}_1$, meaning the weighted sum of inputs is at least approximately equally significant as the bias; it is reasonable to expect that these assumptions hold for at least some feature maps in the layer. Because receptive fields typically have small sizes (e.g., $3\times3\times3$), and the input image has three channels, we have $n \ll |x_1^2/x_1^1|$. Under these assumptions, we the following holds for the inputs $Z_1 = \mathbf{w}\mathbf{x}^1 + b$ and $Z_2 = \mathbf{w}\mathbf{x}^2 + b$ to the activation function of N:

$$\begin{aligned} |Z_2 - Z_1| &= |\mathbf{w}\mathbf{x}^2 + b - (\mathbf{w}\mathbf{x}^1 + b)| \\ &= |\mathbf{w}\mathbf{x}^2 - \mathbf{w}\mathbf{x}^1| \\ &\approx |\mathbf{w}\mathbf{x}^2| \\ &\gg |\mathbf{w}\mathbf{x}^1| \end{aligned}$$

Because $|\mathbf{w}\mathbf{x}^1| \approx |Z_a|$, we have $|Z_2 - Z_1| \gg |Z_a|$. Therefore, changing just the value x_1 to $-(2^{15})$ in the input results in a drastic change to the input value of the activation function. This is undesirable behaviour, because it makes the influence of inputs $x_i, i \neq 1$, negligible compared to that of x_1, while exactly the opposite should hold. The problem is likely to remain even when $w_1 = 0$ for neuron N because different neurons at the same depth share the same weight kernel, thus the input x_1 takes on the different weights in the kernel for different neurons. Furthermore, because we use ReLU activations, which are linear when their input is non-negative, the large changes in their input may propagate further to deeper layers.

To solve this numerical problem, we simply replace the values set by skullstripping by 0 in the entire dataset. The QSM channel of the lesion shown in **Figure 31** is shown after replacing the unsuitable values in **Figure 30**. We note that his analysis demonstrates the importance of data inspection before training a model.

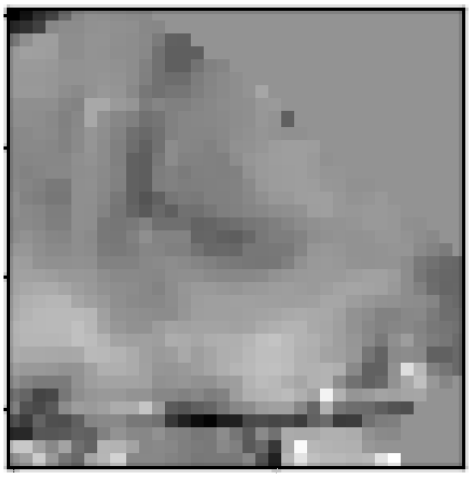

Figure 31: The QSM channel of an image after replacing unsuitable values.

4.3 NETWORK ARCHITECTURE

Once the oversampling, data augmentation, and replacement of unsuitable values have been performed, we train the convolutional neural network on the dataset. We evaluated different network architectures, as finding a suitable one is often an experimental process. Here, we present the network that yielded the best results and shortly discuss our other attempts. The network consists of an input layer, three convolutional blocks, and an "output block" consisting of a fully connected layer of two neurons, a softmax layer, and an output layer. We show the network in **Table 2**, and implement it in the MATLAB programming language using the Deep Learning Toolbox.

The input layer in the network is of size $35 \times 35 \times 35 \times c$, where $c \in \{1, 2, 3\}$ is the number of image channels used. Matlab automatically normalizes the inputs, meaning the images are modified to have a mean of 0 and a standard deviation of 1; in MATLAB, this is called "'zerocenter' normalization".

The input layer is followed by a sequence of three convolutional blocks. Each convolutional block starts with a convolutional layer without an activation function, meaning the layer only computes the weighted sum of its inputs. The convolutional layers in all of the blocks have a receptive field of size $3 \times 3 \times 3$. In the first two blocks, these layers consist of 64 feature maps, while the convolutional layer in the third block consists of 128 feature maps. Increasing the number of feature maps as we move to the deeper convolutional layers is a common practice – perhaps due to the assumption that there are more high-level features than low-level ones. The convolutional layer of each block is followed by a batch normalization layer. This layer performs batch normalization [30], which normalizes the succeeding layer's inputs by fixing their mean and variance. After batch normalization is performed, the ReLU activation function is applied to the normalized sums of weighted inputs. Each convolutional block then

ends with a max pooling layer with filters of size $2 \times 2 \times 2$ and stride 2, which reduces its input data volume by a factor of $\frac{7}{8}$.

The sequence of convolutional blocks is followed by an output block. The block begins with a fully connected layer consisting of two neurons. The intent behind the layer is not to increase the expressive power of the network; rather, MATLAB by default uses a fully connected layer as the first step in outputting classification scores. The layer is followed by a softmax layer and then by a classification layer which computes cross-entropy loss.

Table 2: Model architecture.

Layer	**Type**	**Description**
1	3-D Image Input	$35 \times 35 \times 35 \times c$, 'zerocenter' normalization
2	**Convolutional**	64 $3 \times 3 \times 3$ kernels, stride $1 \times 1 \times 1$
3	Batch Normalization	
4	ReLU	
5	3-D Max Pooling	filter size $2 \times 2 \times 2$, stride $2 \times 2 \times 2$
6	**Convolutional**	64 $3 \times 3 \times 3$ kernels, stride $1 \times 1 \times 1$
7	Batch Normalization	
8	ReLU	
9	3-D Max Pooling	filter size $2 \times 2 \times 2$, stride $2 \times 2 \times 2$
10	**Convolutional**	128 $3 \times 3 \times 3$ kernels, stride $1 \times 1 \times 1$
11	Batch Normalization	
12	ReLU	
13	3-D Max Pooling	filter size $2 \times 2 \times 2$, stride $2 \times 2 \times 2$
14	Fully Connected	Width 2
15	Softmax	
16	Classification Output	Cross Entropy Loss

The architecture we described is the one that achieved the best results on the dataset; we present the results in **Chapter 5**. We now discuss alternative architecture choices that we have evaluated. As for network depth, networks with only two convo-

lutional blocks gave noticeably worse results, even when larger receptive fields (up to a size of $7 \times 7 \times 7$) were used. Shallower architectures have fewer parameters and smaller sizes of the features in their deepest feature maps. Therefore, it would be interesting to see the performance of a network with two convolutional blocks with an even larger receptive field and a larger number of feature maps per layer. Next, networks with more than three convolutional blocks did not improve upon the best results given by the presented network. As these deeper networks have larger numbers of parameters, they require more resources for training and increase the risk of overfitting. Therefore, we decide on the "smallest" network that gives the optimal results. Finally, we note that we have attempted using dropout layers, which are layers that randomly set layer activations to 0 at each step during training, thus potentially reducing overfitting [62]. In our case, however, dropout layers did not have an impact on the results.

4.4 LEARNING PARAMETERS

The parameters used for training the network are as follows. We use Adam [33], a popular stochastic first-order gradient-based optimization algorithm, as the optimization algorithm. The algorithm is an extension to stochastic gradient descent. Empirical results have shown that Adam has advantages compared to many other popular optimization algorithms, due to its computational efficiency and numerical robustness [33].

We train the network for 15 epochs, shuffling the training set before every epoch. The learning rate α is initially set to 0.01, then decreased by a factor of $\sqrt[2]{0.1} \approx 0.316$ at epoch 6, and decreased again by the same factor (i.e., to 0.001) at epoch 11; the learning rate for each epoch can be seen in **Table 3**. Starting with a larger learning rate allows the network to make large modifications to its parameters, thus being able to change quickly at the beginning of training. In the later epochs, the network has already coarsely learned some important image features, and a lower learning rate allows it to then fine-tune those features.

Table 3: Learning rate throughout training.

Epoch	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\frac{Rate}{0.01}$	1	1	1	1	1	0.32	0.32	0.32	0.32	0.32	0.1	0.1	0.1	0.1	0.1

We train the network on all possibe combinations of the three image channels. We note that all of the training has been done on GPUs.

5 RESULTS

We present the results of training our network using each of the different combinations of the image channels in **Table 4**. We show the accuracies achieved by the model on the test set, as they demonstrate the generalization capability of the model; in all cases, the training accuracies are larger than the test accuracies. Because oversampling was not performed on the test set, the set is imbalanced and contains 160 remyelinating and 318 demyelinating lesions. Thus, the *baseline* accuracy for the test set is 66.66%, as a network that classifies all lesions as demyelinating trivially achieves the accuracy. Therefore, we compare the model to the baseline when evaluating its generalization capability.

Table 4: Test accuracies of the model.

Channels Used	**Test Accuracy**
None (baseline)	66.66%
Flair	70.09%
QSM	**89.31%**
Mask	69.22%
Flair, QSM	**88.59%**
Flair, Mask	71.01%
QSM, Mask	**89.45%**
FLAIR, QSM, Mask	**89.00%**

The best test accuracies obtained by the network are promising, being approximately 89%; they are highlighted in bold in **Table 4**. The true positive and true negative rates are similar (i.e., both larger than 86%) in such cases; we show the confusion matrix of the model when all three image channels are used in **Figure 32**. These results show that the model could potentially help medical experts in MS lesion diagnosis, especially if further improvements to the accuracy are made. We note that this is particularly true considering the fact that manual image classification by medical doctors may in general be error prone itself. The non-highlighted accuracies are not sufficiently better than the baseline to be interpreted as useful.

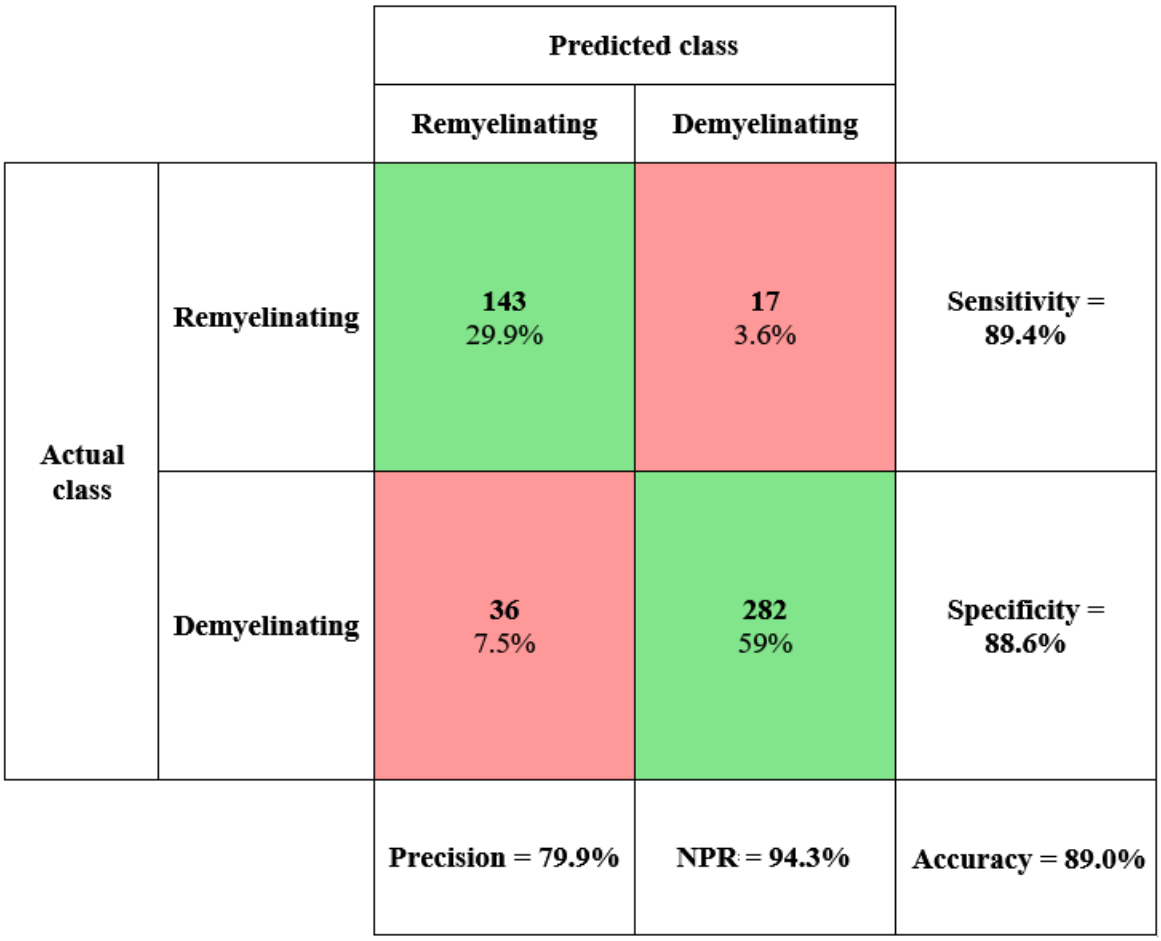

Figure 32: Confusion matrix of the test results when all three image channels are used.

Data augmentation and the choice learning rate had a large impact on the results of the model. When the network was trained on the non-augmented dataset using a fixed learning rate of 0.01, the best results obtained were $\sim 74\%$. By increasing the dataset 48 times by data augmentation, the results increased to $\sim 84\%$, and by also using the decreasing learning rate sequence the results increased to $\sim 89\%$.

We notice the following from the table. The accuracies are high if and only if the QSM channel is used, consistently being approximately 89%, irrespective of which other channels are used alongside QSM. When the FLAIR and mask channels are used without QSM, the test accuracies are much lower – not surpassing 72%, which is not a significant improvement compared to the baseline. We conclude that QSM is necessary for the network to successfully classify the dataset, and that the FLAIR and mask channels do not improve the accuracy when QSM is used. This concurs with how the dataset was originally manually classified, as it was the QSM channel that enabled differentiating between the different lesion classes [57].

While the FLAIR and mask channels do not contribute to the classification by the convolutional neural network itself, the two channels were important in originally ob-

taining the dataset. Each image in the dataset was centered around a lesion, and the centering was performed by using the mask of the original MRI images, which indicates the location of a lesion. The mask itself was obtained using a segmentation neural network on the FLAIR and MP2RAGE images. Thus, the FLAIR and mask channels were necessary in setting up the model. A promising research direction therefore is investigating if lesion segmentation can be successfully performed using the QSM channel instead of the FLAIR and MP2RAGE channels. This would allow the images to then be centered around lesions, and, if classified using only the QSM channel, remove the need for the FLAIR and MP2RAGE channels from the entire process.

A large obstacle to deploying a model such as ours in clinical practice would be the difficulty of discerning between images that belong to class 1 of our dataset – images medical doctors were unable to classify – from those belonging to classes 2, 3, 5, or 6. Our model assumes that input images belong to classes 2, 3, 5, or 6; however, most images in our dataset are unclassified ones. Our attempts at constructing a neural network model similar to the one in **Chapter 4**, with the goal of successfully discerning classified images from unclassified ones, produced no positive results whatsoever. Future work of addressing ways to discard lesions unable to be classified would be necessary for applying our model in clinical practice, especially due to the frequency of such lesions. Including class 4 lesions in the classification model would also be beneficial, even if such lesions appear infrequently and do not carry significant implications for patient state.

Finally, a model being able of classifying lesions into each of the classes 2, 3, 4, 5, and 6, rather than just the remyelinating and demyelinating types, would provide medical doctors with even more information.

6 CONCLUSION

In this work, we presented a classification problem on multi-modal MRI images of multiple sclerosis lesions, and designed a convolutional neural network model for the problem. We presented the theoretical background of the model and the medical background of the problem. We described the approach and methods we use to solve the problem, including data manipulation, network architecture design, and choice of learning parameters. Finally, we presented and discussed the results of the model.

The model we designed achieves a test accuracy of approximately 89% whenever the QSM image channel is used, indicating the model has potential for helping medical doctors in diagnosis, especially if further improvements to the accuracy are achieved. The FLAIR and mask channels do not contribute to a higher accuracy when used alongside the QSM channel, and do not significantly improve the baseline network accuracy when used without the QSM channel, showing the two channels do not help classification itself. We note that, alongside the choice of network architecture, data augmentation and the choice of learning parameters had a high impact on improving the results of the model.

Traditional image processing and computer vision methods that do not use neural networks seldom achieve significant results in image classification problems. Neural network approaches have drastically increased the state of the art accuracies on many such problems. However, such approaches at times still do not achieve adequate results on medical imaging datasets, mostly due to scarcity of data. By achieving a test accuracy of 89%, we show that a convolutional neural network approach is a very promising one for our problem. Further improvements to the accuracy would be beneficial to developing trust in the model and to its applicability in aiding medical doctors. Such a software solution may help doctors by automatically classifying images, thus reducing the time and energy needed for diagnosis.

A large obstacle to deploying our model in clinical practice is the difficulty of discerning lesions that belong to class 1 of our dataset from those belonging to classes 2, 3, 5, or 6. Our model assumes that input images belong to the later classes, but most images in our dataset belong to class 1, meaning they could not be properly classified by medical doctors. We were not successful in attempting to construct a neural network model that would properly discern between these two categories. It would be necessary to address ways of discarding class 1 lesions before for applying our model in clinical practice, especially due to their frequency. Furthermore, including

class 4 lesions in the classification model would also be beneficial.

The success of our model shows using convolutional neural networks is a very promising direction in classification of MRI images of multiple sclerosis lesions. However, future work is necessary to make the model applicable in practice, especially work in regard to discerning between unclassified and classified lesions.

7 POVZETEK V SLOVENSKEM JEZIKU

Medicinske slikovne tehnike omogočajo vpogled v notranjost telesa s čimer je omogočeno diagnosticiranje, spremjanje in zdravljenje bolezenskih stanj. Magnetnoresonančno slikanje (MRI) za vpogled v telo uporablja močna magnetna poljater vzbujanje z radijskimi valovi. V klinični praksi se uporablja za vpogled v mehka tkiva, med drugim pogosto za slikanje centralnega živčnega sistema ter diagnosticiranje nevroloških motenj. Ena takšnih motenj je multipla skleroza (MS), demielinizirajoča bolezen, pri kateri so poškodovani izolacijski mielinski ovoji živčnih celic v možganih in hrbtenjači. Pred kratkim je bilo identificiranih pet vrst MS lezij, to so območja tkivne nenormalnosti, ter da jih je mogoče razvrščati na osnovi kvantitativne preslikave občutljivosti (ang. Quantitative Susceptibility Mapping – QSM), kar je ena izmed modalnosti MRI slik. Najpomembnejša delitev lezij je na demielinizirajoče in remielinizirajoče vrste [57]. Pri remielinizirajočih lezijah proces popravljanja, imenovan remielinizacija, poskuša ustvariti nove mielinske ovojnice na površini demielinizirajočih aksonov; pri demielinizirajočih lezijah se proces ne pojavi. Remielinizirajoče lezije kažejo na pričakovano izboljšanje bolnikovega stanja, demielinizacijske lezije pa na pričakovano poslabšanje stanja. Razvrščanje lezij MS v dva tipa na podlagi QSM MRI je zato lahko zelo koristno pri diagnozi bolnikov z MS.

Medicinske slike običajno ročno obdelajo medicinski strokovnjaki, kar se lahko izkaže za dolgotrajno in naporno opravilo, ugotovitve pa so podvržene subjektivni presoji. Z računalniško podprtimi rešitvami želimo zmanjšati obremenitev stokovnega osebja, omogočiti hitrejše diagnosticiranje in izboljšati diagnostično natančnost. Govorimo o področju računalniškega vida, kjer se obdelava medicinskih slik uporablja za segmentacijo in klasifikacijo slik. Pregled področja podajajo Gao et al. [21] in Yanase et al. [66].

V tem magistrskem delu predstavljamo nov postopek za klasifikacijo lezij pri multipli sklerozi, ki temelji na uporabi konvolucijske nevronske mreže (CNN). Uprabljena je podatkovna zbirka slik, ki so jo v svojih raziskavah MS lezij podali Reza et al. [57]. Gre za tridimenzionalne (volumetrične) slike glave v modalnostih MRI FLAIR in MRI QSM ter segmentiranimi ročno klasificiranimi lezijami. Te slike smo predobdelali tako, da smo jih geometrijsko poravnali, združili in izrezali v področja velikosti $35 \times 35 \times 35$ slikovnih elementov, kjer vsako poročje predstavlja eno lezijo s centrom v središču

izreza ter vključuje tri slikovne kanale: MRI FLAIR, MRI QSM in kanal maske, ki kaže lokacijo lezije.

Predlagan CNN klasifikator avtomatsko razvršča lezije v razreda remielinizirajočih in demielinizirajočih lezij. V tem delu predstavljamo teoretično ozadje modela in medicinsko ozadje problema. Opisujemo pristop in metode, ki jih uporabljamo za reševanje problema, vključno z manipulacijo podatkov, načrtovanjem mrežne arhitekture in izbiro parametrov učenja. Na koncu so predstavljeni rezultati testiranja modela, ki so tudi ovrednoteni v končni diskusiji opravljenega dela.

Model doseže približno 89-odstotno natančnost, kadar koli se uporabi slikovni kanal QSM, kar kaže, da bi model lahko pripomogle k diagosticiranju MS v klinični praksi. Kanala FLAIR in maska ne prispevata k večji natančnosti, če se uporabljata skupaj s kanalom QSM, kar kaže, da oba kanala pri klasifikaciji nista ključna. Ne smemo pa pozabiti, na ključno vlogo FLAIR modalnosti pri pripravi podatkov. Ugotavljamo, da sto bili za doseganje navedenih razultatov pomembni izbira omrežne arhitekture, priprava podatkov s postopki povečanja obsega (ang. augmentation) ter izbira učnih parametrov.

Uporabo našega predstavljenega modela v klinični praksi omejuje nezmožnost prepoznavanja lezij, ki jih zdravniki ne morejo pravilno razvrstiti. V uporabljeni podatkovni zbirki je takšnih večina vseh lezij. Sklepamo, da je pogosto vzrok v tem, da se z razvojem blezni več lezij združi v eno. Obravnava takšnih primerov je med cilji našega prihodnjega dela.

Z doseganjem 89-odstotne testne natančnosti smo pokazali, da je pristop s konvolucijsko nevronsko mrežo za reševanje problema klasifikacije MS lezij zelo obetaven. Ocenjujemo, da bi nadaljnje izboljšave omogočile dodatno povečanje natančnosti in s tem povečanje zaupanja v model in njegovo aplikabilnost. Takšna programska rešitev bi s samodejnim razvrščanjem slik lahko pomagala pri spremljanju bolezni z vidika pomembnega zmanjšanja potebnega diagnostičnega časa in potrebnega storkovnega kadra.

8 REFERENCES

[1] Z. Akkus, J. Schmidt, A. Galimzianova, A. Hoogi, D.L. Rubin in B.J. Erickson, Deep learning for brain MRI segmentation: state of the art and future directions. *Journal of Digital Imaging* 30 (2017) 449–59. *(Cited on page 3.)*

[2] R.G. Alexander, J. Schmidt in G.Z. Zelinsky, Are summary statistics enough? Evidence for the importance of shape in guiding visual search. *Visual Cognition* 22 (2014) 595–609. *(Cited on page 20.)*

[3] M.Z. Alom, T.M. Taha, T.M. Taha, C. Yakopcic, S. Westberg, P. Sidike in M.S. Nasrin, The History Began from AlexNet: A Comprehensive Survey on Deep Learning Approaches. *ArXiv* (2018) . *(Cited on pages 4 in 32.)*

[4] A. Amini, A.P- Soleimany, S. Karaman, in D. Rus, Spatial Uncertainty Sampling for End-to-End Control. *ArXiv* (2018) . *(Cited on pages VII in 12.)*

[5] H,G. Barrow, in J.M. Tenenbaum, Interpreting line drawings as three-dimensional surfaces. *Artificial Intelligence* 17 (1981) 75–116. *(Cited on page 31.)*

[6] K. Berer in G. Krishnamoorthy, Microbial view of central nervous system autoimmunity. *FEBS Letters* 22 (2014) 588. *(Cited on page 42.)*

[7] V. Biscione in J.S. Bowers, Learning Translation Invariance in CNNs. *ArXiv* 2 (2020) 1237–1242. *(Cited on page 37.)*

[8] C.M. Bishop, *Pattern Recognition and Machine Learning*, Information Science and Statistics, 2006. *(Cited on page 4.)*

[9] D. Ciresan, U. Meier, J. Masci, L.M. Gambardella in S. Schmidhuber, Flexible, High Performance Convolutional Neural Networks for Image Classification. *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence* 2 (2011) 1237–1242. *(Cited on pages 20 in 32.)*

[10] W. Chen, S.A. Gauthier, A. Gupta, J. Comunale, T. Liu, S. Wang in Y. Wang, Quantitative susceptibility mapping of multiple sclerosis lesions at various ages. *Radiology* 271 (2014) . *(Cited on pages VII in 46.)*

[11] T. Ching, D.S. Himmelstein, B.K. Beaulieu-Jones, A. Kalinin in B.T. Do, Opportunities and obstacles for deep learning in biology and medicine. *Royal Society* 15 (2018) . *(Cited on page 3.)*

[12] A. Compston, A. Coles, B.K. Beaulieu-Jones, A. Kalinin in B.T. Do, Multiple sclerosis. *Lancet* 359 (2002) 1221–1231. *(Cited on pages 43 in 44.)*

[13] *CS231n: Convolutional Neural Networks for Visual Recognition*, Stanford University. `https://cs231n.github.io/`. (Datum ogleda: 20. 07. 2022.) *(Cited on pages VII, 4 in 36.)*

[14] G. Cybenko, Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems* 2 (1989) 303–314. *(Cited on page 17.)*

[15] H. Daume III, *A Course in Machine Learning, Ch. 4.5, The Perceptron*, Self-published, 2017. *(Cited on page 9.)*

[16] L. Deng, The MNIST Database of Handwritten Digit Images for Machine Learning Research [Best of the Web]. *IEEE Signal Processing Magazine* 29 (2012) 141–142. *(Cited on page 53.)*

[17] J. Deng, W. Dong, R. Socher, L. Li, K. Li in L. Fei-Fei, ImageNet: A large-scale hierarchical image database. *IEEE Conference on Computer Vision and Pattern Recognition* (2009) 248–255. *(Cited on page 53.)*

[18] H. Enderton, A mathematical introduction to logic (2nd ed.), "Complete set of logical connectives". *Boston, MA: Academic Press* (2001) 386–408. *(Cited on page 14.)*

[19] K. Fukushima, Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position. *Biological Cybernetics* 36 (1980) 193–202. *(Cited on page 32.)*

[20] K. Fukushima, Visual feature extraction by a multilayered network of analog threshold elements. *IEEE Transactions on Systems Science and Cybernetics* 5 (1969) 322–333. *(Cited on page 37.)*

[21] J. Gao, Y. Yang, P. Lin, in D.S. Park, Computer Vision in Healthcare Applications. Journal of Healthcare Engineering. *Journal of Healthcare Engineering* (2018) . *(Cited on pages 1, 2 in 65.)*

[22] X. Glorot, A. Bordes in Y. Bengio, Deep sparse rectifier neural networks. *AISTATS* 1 (2011) . *(Cited on page 37.)*

[23] I.J. Goodfellow, Y. Bengio in A.C. Courville, *Deep Learning*. Nature, 521. *(Cited on pages 4 in 34.)*

[24] R. Hahnloser in H.S. Seung, Permitted and Forbidden Sets in Symmetric Threshold-Linear Networks. *NIPS* 2001 (2001) . *(Cited on page 37.)*

[25] T.J. Hastie, R. Tibshirani in J.H. Friedman, *The Elements of Statistical Learning*. Stanford, 2001. *(Cited on page 4.)*

[26] D.H. Hubel in T.N. Wiesel, Receptive fields of single neurones in the cat's striate cortex. *The Journal of Physiology* 148 (1959) . *(Cited on page 32.)*

[27] D.H. Hubel in T.N. Wiesel, Brain and visual perception: the story of a 25-year collaboration.. *Oxford University Press US* (2005) 106. *(Cited on page 32.)*

[28] K. Hornik, M. Stinchcombe in W. Halbert, Multilayer Feedforward Networks are Universal Approximators. *Neural Networks, Pergamon Press* 2 (1989) 359–366. *(Cited on page 17.)*

[29] *IJCNN 2011 Competition result table*, `https://benchmark.ini.rub.de/gtsrb_results.html`. (Datum ogleda: 10. 12. 2011.) *(Cited on page 32.)*

[30] S. Ioffe in C. Szegedy, Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *ArXiv* (2015) . *(Cited on page 57.)*

[31] D. Jurafsky in J. H. Martin, Speech and Language Processing, Chapter 5. *Draft* (2021) 13. *(Cited on page 12.)*

[32] E. R. Kandel, J. H. Schwartz in T. M. Jessell, Principles of neural science. *New York, N.Y.: McGraw-Hill Education LLC.* (2000) . *(Cited on page 5.)*

[33] D.P. Kingma in J. Ba, Adam: A Method for Stochastic Optimization. *ArXiv* (2014) . *(Cited on page 59.)*

[34] A. Krizhevsky, I. Sutskever in G.E. Hinton, ImageNet classification with deep convolutional neural networks. *Communications of the ACM* 60 (2012) 84–90. *(Cited on pages 32 in 36.)*

[35] F. La Rosa, A. Abdulkadir in M.J. Fartaria, Multiple sclerosis cortical and WM lesion segmentation at 3T MRI: a deep learning method based on FLAIR and MP2RAGE. *NeuroImage: Clinical* 27 (2020) . *(Cited on pages VII, 46 in 47.)*

[36] B. Landau, L.B. Smith in S.S. Jones, The importance of shape in early lexical learning. *Cognitive Development* 3 (1988) 299–321. *(Cited on page 20.)*

[37] H. Lassmann, Multiple sclerosis pathology. *Cold Spring Harbor Perspectives in Medicine* (2018) . *(Cited on page 45.)*

[38] P.C. Lauterbur, Image Formation by Induced Local Interactions: Examples Employing Nuclear Magnetic Resonance. *Nature* 242 (1973) 190-191. *(Cited on page 44.)*

[39] Y. LeCun, B. Boser, J.S. Denker, D. Henderson, R.E. Howard, W. Hubbard in L.D. Jackel, Backpropagation Applied to Handwritten Zip Code Recognition. *AT&T Bell Laboratories* (1989) . *(Cited on page 32.)*

[40] M. Leshno, V.Y. Lin, A. Pinkus in S. Schocken, Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks* 6 (1993) 861–867. *(Cited on page 17.)*

[41] G.J. Litjens, T. Kooi, B.E. Bejnordi, A.A. Setio, F. Ciompi, M. Ghafoorian, J.V. Laak, B.V. Ginneken in C.I. Sánchez, A survey on deep learning in medical image analysis. *Medical image analysis* 42 (2017) 60-88. *(Cited on pages 2 in 3.)*

[42] M. London in M. Häusser, Dendritic computation. *Annual review of neuroscience* 28 (2005) . *(Cited on page 6.)*

[43] L. Lu, Y. Shin , Y. Su in G.E. Karniadakis, Dying ReLU and Initialization: Theory and Numerical Examples. *ArXiV* (2019) . *(Cited on page 38.)*

[44] A.S. Lundervold in A. Lundervold, An overview of deep learning in medical imaging focusing on MRI. *Zeitschrift fur medizinische Physik* 29 (2019) 102–127. *(Cited on pages VII, 3 in 33.)*

[45] P. Mansfield in P.K. Grannell, "Diffraction" and microscopy in solids and liquids by NMR. *Physical Review* 12 (1975) . *(Cited on page 44.)*

[46] W. McCulloch in W. Pitts, A Logical Calculus of Ideas Immanent in Nervous Activity. *Bulletin of Mathematical Biophysics* 5 (1943) 115–133. *(Cited on page 5.)*

[47] W.I. McDonald, A. Compston, G. Edan, D. Goodkin, H.P. Hartung in F.D. Lublin, Recommended diagnostic criteria for multiple sclerosis: guidelines from the International Panel on the diagnosis of multiple sclerosis. *Annals of Neurology* 50 (2001) 121–127. *(Cited on page 44.)*

[48] R. Milo in E. Kahan, Multiple sclerosis: geoepidemiology, genetics and the environment. *Autoimmunity Reviews* 9 (2010) A387-94. *(Cited on page 44.)*

[49] M.A. Nielsen, *Neural Networks and Deep Learning, Ch. 4, A visual proof that neural nets can compute any function*, Determination Press, 2015. *(Cited on pages 4, 18 in 19.)*

[50] T.P. Oikarinen, K. Srinivasan, O.C. Meisner, J. Hyman, S. Parmar, R. Desimone, R. Landman in G. Feng, Deep Convolutional Network for Animal Sound Classification and Source Attribution using Dual Audio Recordings. *bioRxiv* (2018) . *(Cited on page 27.)*

[51] C. Parmar, J.D. Barry, A. Hosny, J. Quackenbush in J.W.L. Aerts, Data analysis strategies in medical imaging. *Clinical Cancer Research* 24 (2018) . *(Cited on page 3.)*

[52] *Neurobiology of Disease*, `https://learn.chm.msu.edu/neuroed/neurobiology_disease/content/disorders/multiple_sclerosis/pathology.html`. (Datum ogleda: 29. 07. 2022.) *(Cited on pages VII in 43.)*

[53] P.C. Petersen, Neural Network Theory. (2020) . *(Cited on page 17.)*

[54] A. Pinkus, Approximation theory of the MLP model in neural networks. *Acta Numerica* 9 (1999) 143–195. *(Cited on page 17.)*

[55] W. Rawat in Z. Wang, Deep Convolutional Neural Networks for Image Classification: A Comprehensive Review. *Neural Computation* 29 (2017) 2352–2449. *(Cited on page 4.)*

[56] D. Reich, C.F. Lucchinetti in J.S. Calabresi, Multiple sclerosis. *The New England Journal of Medicine* 378 (2018) 169–180. *(Cited on pages 1 in 42.)*

[57] R. Rahmanzadeh, R. Galbusera, P. Lu, P. Bahn, E. Weigel, M. Barakovic, J. Franz, T.D. Nguyen, P. Spincemaille, S. Schiavi, A. Daducci, F. La Rosa, M. Absinta, M.B. Cuadra, E. Radue, D. Leppert, J. Kuhle, W. Brück, D.S. Reich, C. Stadelmann, Y. Wang in C. Granziera, A New Advanced MRI Biomarker for Remyelinated Lesions in Multiple Sclerosis. *Annals of Neurology* 92 (2022) . *(Cited on pages 1, 2, 45, 46, 47, 61 in 65.)*

[58] O. Ronneberger, P. Fischer in T. Brox, U-Net: Convolutional Networks for Biomedical Image Segmentation. *ArXiv* (2015) . *(Cited on page 46.)*

[59] F. Rosenblatt, The perceptron: a probabilistic model for information storage and organization in the brain.. *Psychological review* 65 (1958) 386–408. *(Cited on page 8.)*

[60] D. Rueckert, B. Glocker in B. Kainz, Learning clinically useful information from images: past, present and future. *Medical Image Analysis* 33 (2016) 13-18. *(Cited on pages 3 in 53.)*

[61] J. Schmidhuber, Deep Learning in Neural Networks: An Overview. *Neural Networks* 61 (2015) 85–117. *(Cited on page 4.)*

[62] N. Srivastava, G.E. Hinton, A. Krizhevsky, I. Sutskever in R. Salakhutdinov, Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research* 15 (2014) 1929–1958. *(Cited on page 59.)*

[63] M. Tovee, *An Introduction to the Visual System (2nd ed.)*, Cambridge: Cambridge University Press, 2008. *(Cited on page 24.)*

[64] *UC Business Analytics R Programming Guide*, University of Cincinnati. `http://uc-r.github.io/2018/04/09/feedforward-deep-models/`. (Datum ogleda: 05. 08. 2022.) *(Cited on pages VII in 17.)*

[65] J. Wolfe in T.S. Horowitz, Five factors that guide attention in visual search. *Nature Human Behaviour* 1 (2017) . *(Cited on page 20.)*

[66] J. Yanase in E. Triantaphyllou, A systematic survey of computer-aided diagnosis in medicine: Past and present developments. *Expert Systems Applications* 138 (2019) . *(Cited on pages 1, 2 in 65.)*

[67] W. Zhang, Shift-invariant pattern recognition neural network and its optical architecture. *Proceedings of Annual Conference of the Japan Society of Applied Physics* (1988) . *(Cited on pages 32 in 34.)*

[68] A. Zhang, Z.C. Lipton, M. Li in A. Smola, *Dive into Deep Learning*. ArXiv, 2021. *(Cited on page 4.)*