CALABI-YAU VARIETIES AND MACHINE LEARNING

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

TUTKU DORUK BAYRAMOĞLU

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
MATHEMATICS

JULY 2022

Approval of the thesis:

CALABI-YAU VARIETIES AND MACHINE LEARNING

submitted by **TUTKU DORUK BAYRAMOĞLU** in partial fulfillment of the requirements for the degree of **Master of Science in Mathematics Department, Middle East Technical University** by,

Prof. Dr. Halil Kalıpçılar
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. Yıldıray Ozan
Head of Department, **Mathematics** _____

Assoc. Prof. Dr. Emre Coşkun
Supervisor, **Mathematics, METU** _____

Examining Committee Members:

Prof. Dr. Ali Sinan Sertöz
Mathematics, Bilkent University _____

Assoc. Prof. Dr. Emre Coşkun
Mathematics, METU _____

Assoc. Prof. Dr. Ali Ulaş Özgür Kişisel
Mathematics, METU _____

Date:

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Surname: Tutku Doruk Bayramoğlu

Signature :

ABSTRACT

CALABI-YAU VARIETIES AND MACHINE LEARNING

Bayramoğlu, Tutku Doruk
M.S., Department of Mathematics
Supervisor: Assoc. Prof. Dr. Emre Coşkun

July 2022, 57 pages

Techniques from Machine Learning (ML) have been applied to various mathematical problems in recent years. One such problem is the determination of Hodge numbers of Calabi-Yau varieties. In this thesis, a neural network model to estimate the Hodge numbers of complete intersection Calabi-Yau varieties is built and evaluated.

Keywords: Calabi-Yau variety, Hodge number, Machine Learning, neural network

ÖZ

CALABİ-YAU VARYETELERİ VE MAKİNE ÖĞRENMESİ

Bayramoğlu, Tutku Doruk
Yüksek Lisans, Matematik Bölümü
Tez Yöneticisi:

Temmuz 2022 , 57 sayfa

Makine Öğrenmesi (MÖ) teknikleri son yıllarda çeşitli matematiksel problemlere uygulanmıştır. Bu problemlerden biri, Calabi-Yau varyetelerinin Hodge sayılarının tespitidir. Bu tezde, tam kesişim Calabi-Yau varyetelerinin Hodge sayılarının tahmini için bir nöral ağ modeli inşa edilmiş ve değerlendirilmiştir.

Anahtar Kelimeler: Calabi-Yau varyetesi, Hodge sayısı, Makine Öğrenmesi, nöral ağ

ACKNOWLEDGMENTS

I thank my advisor Emre Coşkun for his help in supervising me not only about my thesis but also about trying to make me a better academic writer. I thank Serdar Çelebi from Istanbul Technical University to clarify my ideas about some materials I used in my thesis. He recommended me to express my discoveries through hyperparameter tables. I thank Ozan Yetkin for the support he gave me on loading the necessary packages in Python on Jupyter notebook before I put our code in Kaggle. Finally, I thank the members of the thesis committee for their helpful remarks and suggestions.

TABLE OF CONTENTS

ABSTRACT v

ÖZ vi

ACKNOWLEDGMENTS viii

TABLE OF CONTENTS ix

LIST OF TABLES xi

LIST OF FIGURES xii

LIST OF ABBREVIATIONS xiii

CHAPTERS

1 INTRODUCTION 1

2 PRELIMINARIES 5

2.1 Manifolds and Calabi-Yau Varieties 5

2.1.1 Complex Manifolds and Algebraic Varieties 5

2.1.2 Homology, Cohomology, and Betti Numbers 7

2.1.3 Hodge Theory 9

2.1.4 Riemannian Metrics, Complexification, and Kähler Manifolds 12

2.2 A Brief Introduction to Machine Learning 16

2.2.1 Neural Networks 19

2.2.1.1 Some important concepts about neural networks 19

2.2.1.2 Some parameters and hyperparameters of a neural network . . . 22

2.2.2 Some machine learning algorithms . . . 24

2.2.2.1 Forward propagation algorithm . . . 24

2.2.2.2 Gradient descent algorithm . . . 25

2.2.2.3 Backpropagation algorithm . . . 25

2.2.3 Measuring the Performance of the Model . . . 27

2.2.3.1 Training, Validation, Testing Accuracy and Loss Curves 27

2.2.3.2 Confusion Matrix . . . 27

2.2.3.3 Overfitting and underfitting . . . 28

2.2.4 The Calabi-Yau Dataset . . . 29

3 THE CODE, RESULTS, AND DISCUSSION . . . 31

3.1 The Code: Presentation and Discussion . . . 31

3.2 Results of interval tables . . . 39

3.3 Discussion of the results of interval tables . . . 42

3.4 Results of main tables . . . 46

3.4.1 Some alternatives for the constructed model and accuracies with lower limit 50 and upper limit 500 for the hodge number h^{12} . . . 46

3.5 Discussion of the results of main tables . . . 50

REFERENCES . . . 55

LIST OF TABLES

TABLES

Table 2.1 Confusion matrix 28

Table 3.1 $h^{12} \in [N$,500), inc.=25 40

Table 3.2 $h^{12} \in [N, N + 25)$, inc.=25 40

Table 3.3 $h^{12} \in [N, 500)$, inc.=50 41

Table 3.4 $h^{12} \in [N, N + 50)$, inc.=50 41

Table 3.5 $h^{12} \in [N, 500)$,inc.=100 41

Table 3.6 $h^{12} \in [N, N + 100)$,inc.=100 41

Table 3.7 Hyperparameters of main model 46

Table 3.8 Activation function varied 47

Table 3.9 Batch size varied 47

Table 3.10 Number of epochs varied 48

Table 3.11 Loss function varied 48

Table 3.12 Number of neurons varied 49

Table 3.13 Number of hidden layers varied 49

LIST OF FIGURES

FIGURES

Figure 2.1 Densely connected sequential network 20

Figure 2.2 Examples of non-sequential models 21

Figure 3.2 h^{12}=True, with interval $[350, 500)$ 44

Figure 3.3 Confusion matrix for h^{12}=True, with interval $[350, 500)$ 44

Figure 3.4 h^{12}=False, with interval $[350, 500)$ 45

Figure 3.5 Confusion matrix for h^{12}=False, with interval $[350, 500)$ 45

Figure 3.6 Confusion matrix of the main model for h^{12} 51

Figure 3.7 Learning curves for the main model h^{12} 52

Figure 3.8 Learning curves for h^{12} for 7500 epochs 52

Figure 3.9 Learning curves for h^{12}, for 1000 epochs. 53

LIST OF ABBREVIATIONS

T.A.E.F: Training accuracy at the end of first epoch

T.A.E.L: Training accuracy at the end of last epoch

T.L.E.F: Training loss at the end of first epoch

T.L.E.L: Training loss at the end of last epoch

V.A.E.F: Validation accuracy at the end of first epoch

V.A.E.L: Validation accuracy at the end of last epoch

V.L.E.F: Validation loss at the end of first epoch

V.L.E.L: Validation loss at the end of last epoch

T.A: Test accuracy

M.A.E: Mean absolute error

M.S.E: Mean squared error

S.C.C: Sparse Categorical Crossentropy

N.N.H.L: Number of neurons in hidden layers

inc.: increment size

CHAPTER 1

INTRODUCTION

Machine learning has a long history and it has been developed as a subfield of artificial intelligence in the 20th century. The advances of technology and computer science enabled the development of machines that can do very complex calculations, including optimizations in many problems in physics and technology. For example, today, image processing and speech recognition use algorithms in machine learning. However, more relevant to the purpose of this thesis, machine learning is also used to approach many problems in pure mathematics.

We briefly describe three problems in pure mathematics that were approached using machine learning techniques and then we go on to describe the problem considered in this thesis. The first problem is related to classification of facets of polytopes. In order to solve this problem, a neural network is given inputs as features of a facet and the output is the number of fine regular triangulations for the corresponding facet. (See [1] for more details.) The second problem is related to the classification of line bundles as stable or unstable vector bundles. Again, for this classification problem a neural network is constructed and an output set, which consists of 1 or 0 depending on the vector bundle is stable or not, is constructed. 1 is assigned to stable bundles and 0 is assigned to unstable vector bundles. The network is then trained using Chern classes and represented as $k_i \in [-50, 50]$. (See [24] for more details.) Finally, the third problem is related to dessins d'enfants which is related to a classification problem in algebra. Alexandre Grothendieck studied this structure and represented dessins as bipartite graphs. He conceived of dessins d'enfants as the representation of the Galois group of the base field over the rationals. The dessins are given as inputs in matrix format to a densely connected neural network and the output is the degree

of the field extension over $\mathbb{Q}$. (See [14] for more details.)

The problem considered in this thesis is also related to classifying mathematical entities. It is the classification of Hodge numbers of Calabi-Yau 3-fold over $\mathbb{C}$. Hodge numbers are algebraic geometric properties of a given projective variety that can be used to determine some topological and geometric structures of the variety. The ultimate aim of Hodge theory is to study homology and cohomology groups of smooth varieties. Also, Hodge theory has many applications in physics.

This problem has been studied by Yang-Hui He ([13]) by constructing a neural network. In this thesis, we construct our own neural network and train and test it with various values of the parameters involved. The data divisions we use are $\%80$ of the data as training set, $\%10$ of the data as test set, and the remaining $\%10$ as validation set. These ratios differ from those used by He, and are in fact more commonly used for training neural network models. As to our results, we obtain the same accuracy for the classification problem of Hodge numbers as He.

Our aim is to build a neural network to estimate whether the Hodge numbers of complete intersection Calabi-Yau varieties fall in given intervals. We use the Hold-out method, which consists of splitting the whole dataset as a training set, a validation set and a testing set. Our goal is to explore how training our neural network with different distributions of numbers in training, testing, and validation datasets affects the performance of the model and whether test accuracy is always a good measure to evaluate the performance of the model. In addition, we aim to explore how some hyperparameter changes like epoch, batch size, and activation function may affect the performance of the model. We investigate underfitting and overfitting problems in different cases. To conclude, we find that test accuracy is a good decision tool to evaluate the performance of the model only if we work with a balanced training, validation and test dataset and the optimal loss function for our problem is sparse-categorical cross-entropy. Optimal epochs and batch sizes are both 100, with sigmoid activation function. Two hidden layers with less than 100 neurons at each one is optimal and 'Adam' is a good choice as an optimizer for this problem.

The outline of the thesis is as follows. In Chapter 2, we review some preliminary material related to manifolds, Calabi-Yau varieties, and Hodge numbers, as well as

basic techniques and ideas in machine learning. In Chapter 3, we present the code and explain it, together with the results of the code and a discussion of these results.

CHAPTER 2

PRELIMINARIES

2.1 Manifolds and Calabi-Yau Varieties

In this section, we review the mathematical preliminaries on smooth manifolds, algebraic varieties including Calabi-Yau varieties, and Hodge numbers. For a more detailed exposition of various topics in 2.1.1 see [15], [11], [10], [9], and [16].

2.1.1 Complex Manifolds and Algebraic Varieties

Definition 1. *An* n-dimensional chart *on a topological space* X *is a homeomorphism* $\varphi : U \to V$*, where* $U \subset X$ *and* $V \subset \mathbb{C}^n$ *are open subsets. Such a chart will be denoted* (U, φ) *or sometimes just* φ*. Two charts* $\varphi : U \to V$ *and* $\psi : U' \to V'$ *are* holomorphically compatible *if the map* $\psi \circ \varphi^{-1} : \varphi(U \cap U') \to \psi(U \cap U')$ *is biholomorphic.*

Definition 2. *Let* X *be a topological space. A* complex atlas *on a set* X *is a set* $\{\psi_\alpha : U_\alpha \subset X \to V_\alpha \subset \mathbb{C}^n\}_{\alpha \in A}$ *of charts such that for all* $\alpha, \beta \in A$*, the charts* ψ_α *and* ψ_β *are holomorphically compatible, and* $X = \bigcup_{\alpha \in A} U_\alpha$*. A complex atlas* A *is called* maximal *if, for every atlas* B *such that all charts of* B *are holomorphically compatible with all charts of* A*, we have* $B \subset A$*.*

Definition 3. *If a topological space* X *is Hausdorff, second countable and admits a maximal complex atlas* A *as above, then* (X, A) *is called an* n-dimensional complex manifold*. The atlas* A *is called a* complex structure *on* X*.*

It is possible to define holomorphic functions on complex manifolds. The definition is analogous to the definition of smooth functions on a smooth manifold.

Definition 4. *Let X be a complex manifold. A function $f : X \to \mathbb{C}$ is* holomorphic *if given a chart $(U_\alpha, \varphi_\alpha)$ such that $U \cap U_\alpha \neq \varnothing$, the function $f \circ \varphi_\alpha^{-1} : \varphi_\alpha(U_\alpha \cap U) \to \mathbb{C}$ is holomorphic.*

Remark 1. *The definition above is independent of the chart $(U_\alpha, \varphi_\alpha)$ used.*

Definition 5. *A continuous map $f : X \to Y$ between complex manifolds is* holomorphic *if $\psi_\beta \circ f \circ \phi_\alpha^{-1} : \phi_\alpha(U_\alpha \cap f^{-1}(V_\beta)) \to \mathbb{C}^m$ is holomorphic for all charts (U_α, ϕ_α) of X and (V_β, ψ_β) of Y.*

One of the most important types of complex manifold is given by projective varieties, which are defined as the vanishing loci of a set of homogeneous polynomials in projective space, which we define first.

Definition 6. *The n-*dimensional complex projective space *is the complex manifold defined as the quotient $\mathbb{CP}^n := \mathbb{C}^{n+1} - \{0\}/\sim$, where the equivalence relation is defined as $x \sim y$ for $x, y \in \mathbb{C}^{n+1} - \{0\}$ iff $y = kx$ for some $k \in \mathbb{C} - \{0\}$.*

Theorem 1. *The complex projective space $\mathbb{CP}^n$ is a compact complex manifold.*

Proof. See [15, 56–57]. □

Affine algebraic varieties form, in an informal sense, the building blocks of projective algebraic varieties.

Definition 7. *An* affine algebraic variety *defined over $\mathbb{C}$ is a set $V = \{(a_1, \ldots, a_n) \mid f_i(a_1, \ldots a_n) = 0, i = 1, \ldots, m\}$, where $f_1, \ldots, f_m \in \mathbb{C}[x_1, \ldots, x_n]$ are a given collection of polynomials.*

Definition 8. *An* affine algebraic hypersurface *is a variety given by the vanishing of only one polynomial $f(x_1, x_2, \ldots x_n) \in \mathbb{C}[x_1, \ldots, x_n]$.*

Definition 9. *Let $f \in \mathbb{C}[x_0, \ldots, x_n]$ be a polynomial. Then f is* homogeneous of degree *d if $f(\lambda x_0, \ldots \lambda x_n) = \lambda^d f(x_0, \ldots, x_n)$ for $\lambda \in k$.*

In general, a polynomial $f \in \mathbb{C}[x_0, \ldots, x_n]$ cannot be evaluated at a point $P = [a_0 : \cdots : a_n] \in \mathbb{P}^n$ since in general $f(a_0, \ldots, a_n) \neq f(ta_0, \ldots, ta_n)$ for $t \in \mathbb{C} \setminus \{0\}$. However, if f is homogeneous of degree d, then one has $f(a_0, \ldots, a_n) = 0$ iff

$f(ta_0, \ldots, ta_n) = 0$ for $t \in \mathbb{C} \setminus \{0\}$. This implies that zero sets of homogeneous polynomials in $\mathbb{P}^n$ are well-defined.

Definition 10. *A* projective hypersurface *is the zero set*

$$Z(f) = \{[a_0 : \cdots : a_n] \in \mathbb{P}^n \mid f(a_0, \ldots, a_n) = 0\}$$

of a homogeneous polynomial $f \in \mathbb{C}[x_0, \ldots, x_n]$.

Example 1. *The degree 5 projective hypersurface* $X = Z(f) \subset \mathbb{P}^4$*, where* $f(x_0, \ldots, x_4) = x_0^5 + \cdots + x_4^5$*, is called a* Fermat quintic threefold *in* $\mathbb{P}^4$*. Quintic threefolds provide examples of* Calabi-Yau manifolds*, which we shall define later.*

Remark 2. *The example above is a special case of the statement that a homogeneous degree* $n+1$ *polynomial in* $n+1$ *variables defines a hypersurface in* $\mathbb{P}^n$ *which is a Calabi-Yau* $(n-1)$*-fold. See [13, Proposition 3, p. 40].*

2.1.2 Homology, Cohomology, and Betti Numbers

For this subsection, some general references are [12], [22], [23], and [25].

Definition 11. *Given* $n+1$ *points* $v_0, \ldots, v_n \in \mathbb{R}^m$*, with* $m \geq n$*, such that* $v_1 - v_0, \ldots, v_n - v_0$ *are linearly independent as vectors, the set*

$$[v_0, \ldots, v_n] = \{\sum_{k=0}^{n} a_k v_k \in \mathbb{R}^m \mid \sum_{k=0}^{n} a_k = 1, a_k \in \mathbb{R}_{\geq 0}\}$$

is called the n-simplex spanned by $v_0, \ldots, v_n$*. If one takes* $v_i = e_i$ *for* $i = 0, \ldots, n$*, where* $e_i \in \mathbb{R}^{n+1}$ *is the standard basis vector, the resulting* n*-simplex* $[e_0, \ldots, e_n]$ *is called the* standard n-simplex*. The standard* n*-simplex is denoted* $\triangle^n$*.*

Example 2. *In* $\mathbb{R}^n$*, a 0-simplex is a point, a* 1-simplex *is a line segment, a* 2-simplex *is a triangle.*

Definition 12. *Given a topological space* X*, a* singular n-simplex *in* X *is a continuous map* $\sigma : \triangle^n \to X$*.*

Definition 13. *Let* X *be a topological space, and* $n \geq 0$*.* $S_n(X)$ *is defined to be the free abelian group with basis the set of all singular* n*-simplexes in* X*. The elements of* $S_n(X)$ *are called* singular n-chains *in* X*.*

Definition 14. *The* faces *of the standard n-simplex are the images of the maps $\epsilon_i^n : \triangle^{n-1} \to \triangle^n$ defined by $\epsilon_i^n(t_0, \ldots, t_{n-1}) = (t_0, \ldots, t_{i-1}, 0, t_i \ldots, t_{n-1})$. The maps ϵ_i^n are called* face maps.

Example 3. *There are two* face maps $\triangle^0 \to \triangle^1$. *These are $\epsilon_0^1 : t_0 \mapsto (0, t_0)$; $\epsilon_1^1 : t_0 \mapsto (t_0, 0)$. There are three* face maps $\triangle^1 \to \triangle^2$. *These are denoted $\epsilon_i^2 : \triangle^1 \to \triangle^2$, $i = 0, 1, 2$, $\epsilon_0^2 : (t_0, t_1) \mapsto (0, t_0, t_1)$, $\epsilon_1^2 : (t_0, t_1) \mapsto (t_0, 0, t_1)$, $\epsilon_2^2 : (t_0, t_1) \mapsto (t_0, t_1, 0)$.*

Definition 15. *Let $\sigma : \triangle^n \to X$ be an $n - simplex$ considered as a basis element of $S_n(X)$. Then its* boundary *is defined as $\partial_n(\sigma) = \sum_{i=0}^{n} (-1)^i \sigma \circ \epsilon_i^n \in S_{n-1}(X)$. By linear extension, this defines a map $\partial_n : S_n(X) \to S_{n-1}(X)$.*

Since $\partial_n \circ \partial_{n+1} = 0$, holds we can define a chain complex as below.

Definition 16. *Let X be a topological space. The* singular complex of X *is the chain complex of the free abelian groups defined as:*

$$\cdots \longrightarrow S_{n+1}(X) \xrightarrow{\partial_{n+1}} S_n(X) \xrightarrow{\partial_n} S_{n-1}(X) \longrightarrow \cdots$$

and the *n-th homology group* of X is defined as

$$H_n(X) := ker(\partial_n)/im(\partial_{n+1}).$$

Example 4. *Let $\mathbb{RP}^2$ be the 2-dimensional real projective space. Then,*

$$H_n(\mathbb{RP}^2) = \begin{cases} \mathbb{Z} & \text{for } n = 0, \\ \mathbb{Z}/2\mathbb{Z} & \text{for } n = 1, \\ 0 & \text{otherwise.} \end{cases}$$

Example 5. *Let K be the Klein bottle. Then,*

$$H_n(K) = \begin{cases} \mathbb{Z} & \text{for } n = 0, \\ \mathbb{Z}/2\mathbb{Z} \bigoplus \mathbb{Z} & \text{for } n = 1, \\ 0 & \text{otherwise.} \end{cases}$$

Example 6.

$$H_k(\mathbb{CP}^n) = \begin{cases} \mathbb{Z} & \text{for k=0,2,...,2n} \\ 0 & \text{otherwise} \end{cases}$$

Definition 17. *For a topological space X, the* nth Betti number of X *is defined as the rank of* $H_n(X)$.

Example 7.

$$b_n(\mathbb{RP}^2) = \begin{cases} 1 & \textit{for } n = 0, \\ 0 & \textit{for } n = 1, \\ 0 & \textit{otherwise.} \end{cases}$$

Example 8.

$$b_n(\mathbb{K}) = \begin{cases} 1 & \textit{for } n = 0, \\ 1 & \textit{for } n = 1, \\ 0 & \textit{otherwise.} \end{cases}$$

Given a topological space X, let

$$\cdots \longrightarrow S_{n+1}(X) \xrightarrow{\partial_{n+1}} S_n(X) \xrightarrow{\partial_n} S_{n-1}(X) \longrightarrow \cdots$$

be the singular complex of X. Let G be an arbitrary abelian group. Denote C_n^* $= Hom(S_n(X), G)$. Then

$$\cdots \longrightarrow C_{n-1}^* \xrightarrow{\partial_n^*} C_n^* \xrightarrow{\partial_{n+1}^*} C_{n+1}^* \longrightarrow \cdots$$

is the *cochain complex of* X. This allows us to define the nth cohomology group of X with coefficients in G as

$$ker(\partial_{n+1}^*)/im(\partial_n^*).$$

Example 9.

$$H^n(\mathbb{RP}^2) = \begin{cases} \mathbb{Z} & \textit{for n = 0,} \\ \mathbb{Z}/2\mathbb{Z} & \textit{for n=2,} \\ 0 & \textit{otherwise} \end{cases}$$

Example 10.

$$H^k(\mathbb{CP}^n) = \begin{cases} \mathbb{Z} & \textit{for k=0,2,...,2n} \\ 0 & \textit{otherwise} \end{cases}$$

2.1.3 Hodge Theory

For more details on the topics treated in this subsection, see [18], [29], and [15].

We start with the definition of vector bundles, which play a key role in what follows.

Definition 18. *A vector bundle over a topological space X of rank r is a topological space E, called the* total space of the vector bundle*, with a continuous map $\pi : E \to X$, such that the fibers of π have the structure of vector spaces of dimension r and such that π "looks locally" like the projection $X \times k^r \to X$. Here k is either $\mathbb{R}$ or $\mathbb{C}$, and the vector bundles are called* real *or* complex *accordingly.*

Functions $\sigma : U \to E$, where $U \subset X$ is an open set and $\pi \circ \sigma = id_U$ are called sections of the vector bundle E over U. *Sections of E over X are called* global sections of E.

The most important vector bundles in the study of smooth or complex manifolds are the tangent and the cotangent bundles.

Definition 19. *Let X be a smooth manifold and $p \in X$. A* tangent vector at p *is defined to be a map $v : \mathbb{C}^\infty(X, \mathbb{R}) \to \mathbb{R}$ with the following properties:*

- *v is $\mathbb{R}$-linear,*
- *$v(fg) = f(p)v(g) + g(p)v(f)$ for all $f, g \in \mathbb{C}^\infty(X, \mathbb{R})$.*

Definition 20. *The* tangent space *to X at p is the set of all tangent vectors at p. This set, denoted $T_p(X)$, can be seen to have the structure of a real vector space.*

Let X be n-dimensional manifold. The tangent space $T_p(X)$ as an n-dimensional real vector space has the basis

$$\frac{\partial}{\partial x_1}|_p \dots \frac{\partial}{\partial x_n}|_p,$$

where $x_1, \dots, x_n$ are a local system of coordinates and the operators are partial derivatives in the sense of multivariable calculus. Define $TX = \sqcup_{p \in X} T_p X$. TX is called the *tangent bundle of X*.

One can also consider the dual of TX, denoted T^*X, which is called the *cotangent bundle of X*. At a point $p \in X$, T_p^*X has the basis

$$dx^1|_p, \dots, dx^n|_p,$$

where

$$dx^i|_p \left(\frac{\partial}{\partial x_j}|_p \right) = \delta_{ij}.$$

Given two vector bundles E and F over a manifold X, one can form the *tensor* and *exterior* products $E \otimes F$ and $E \bigwedge F$ respectively, using a fiberwise tensor or exterior product operation. One can then easily show that the resulting spaces can be given the structure of a vector bundle over X.

Given $k \geq 0$, one can form the vector bundle of k-forms $\Omega_X^k = \bigwedge^k (T^*X)$. Sections of Ω_X^k over an open subset $U \subset X$ are called *k-forms over U*. If there is a coordinate system $x_1, \ldots, x_n$ on U, then k-forms over U can be given by expressions of the form

$$\sum_I f_I dx_I,$$

where the index set I runs over the subsets of $\{1, \ldots, n\}$ containing k elements, the f_I are smooth functions over U, and for $I = \{i_1, \ldots, i_k\}$ with $i_1 < \cdots < i_k$, $dx_I = dx_{i_1} \wedge \cdots \wedge dx_{i_k}$. Given two coordinate systems over the same open set $U \in X$, transition from one way of writing the same k-form to another can be made by the ordinary methods of calculus.

Definition 21. *The* exterior derivative $d^k : \Omega_X^k \to \Omega_X^{k+1}$ *can be defined locally as follows. Consider local coordinates $x_1, \ldots, x_n$ on an open subset $U \subset X$. For a k-form ω on U given by*

$$\omega = \sum_I f_I dx_I,$$

we define

$$d\omega = \sum_i \sum_I \frac{\partial f_I}{\partial x_i} dx_i \wedge dx_I,$$

where the wedge product is understood to be zero if $i \in I$.

Definition 22. *A k-form $a \in \Omega_X^k$ that can be written as $a = d\omega$ for some $\omega \in \Omega_X^{k-1}$ is called* exact. *If $da = 0$, a is called* closed.

Since for any k-form ω on X, $d(d\omega) = 0$, if a form is exact, then it is closed. However, the converse is not always true. For example, let

$$\omega = \frac{xdy - ydx}{x^2 + y^2}$$

be a 2-form on $X = \mathbb{R}^2 \setminus \{(0,0)\}$. Then ω is closed, as can be seen by a straightforward calculation. However, it is not exact. To see this, we can use the integral formula for manifolds which states that $\int_{\partial X} \omega = \int_X d\omega$. Let $\mathbb{D}$ be the unit disk. Then

$\partial\mathbb{D}$ is the unit circle and $\int_{S^1} \omega = 2\pi$ as can again be seen from a calculation. Since $d\omega = 0$, the equality of integrals stated above does not hold. Hence, ω is not exact.

The de Rham cohomology of X can be considered as a measure of how far closed forms are from being exact.

Definition 23. *Consider the chain complex*

$$0 \to \Omega_X^0 \xrightarrow{d^0} \Omega_X^1 \xrightarrow{d^1} \ldots \xrightarrow{d^{n-1}} \Omega_X^n \to 0.$$

Define the de Rham cohomology groups of X *as the cohomology of this complex as*

$$H_{dR}^k(X) = \frac{ker(d^k)}{im(d^{k-1})}.$$

2.1.4 Riemannian Metrics, Complexification, and Kähler Manifolds

Throughout this subsection, X denotes a differentiable manifold. General references for this subsection are [29] and [15].

Definition 24. *An* almost complex structure of X *is a vector bundle endomorphism* $I : TX \to TX$ *such that* $I \circ I = -id$. I *is* integrable *if* $[T_X^{0,1}, T_X^{0,1}] \subset T_X^{0,1}$.

Definition 25. *A* Riemannian metric *on* X *is a smooth assignment of a symmetric and positive definite inner product to every tangent space* $T_p(X)$. *A* Riemannian manifold *is a real smooth manifold equipped with a Riemannian metric.*

Example 11. *Let* $X = \mathbb{R}^n$. *Then every tangent space* $T_p(X)$ *can be canonically identified with* $\mathbb{R}^n$ *and hence it can be equipped with the standard inner product.*

Remark 3. *A submanifold of a Riemannian manifold inherits a Riemannian metric from the ambient manifold by restriction of the inner product.*

Definition 26. *Let* V *be a real vector space. The* complexification *of* V *is* $V \bigotimes_{\mathbb{R}} \mathbb{C}$. *This is a complex vector space with complex multiplication given by* $w(v \otimes z) = v \otimes (wz)$ *for* $v \in V$ *and* $z, w \in \mathbb{C}$.

Remark 4. *The concept of complexification can be extended to vector bundles over* X.

Remark 5. *Now let X be a complex manifold of complex dimension n, and let $z^j = x^j + iy^j$ for $j = 1, \ldots, n$ be a complex coordinate system over an open subset $U \subset X$. Then the cotangent bundle T^*X can be complexified, and complex-valued 1-forms over U, which are sections of the complexification of T^*X over U, can be written as*

$$\sum f_j dz^j + g_j \overline{dz}^j,$$

where $dz^j = dx^j + idy^j$, $\overline{dz}^j = dx^j - idy^j$, and f_j and g_j are complex-valued smooth functions over U.

Definition 27. *A* symplectic structure on X *is a closed, symmetric, nondegenerate 2-form ω on X. X together with a symplectic structure on it is then called a* symplectic manifold.

Definition 28. *A* Kähler manifold *is a manifold X with a Riemannian metric g, an integrable almost complex structure J, and a symplectic form ω such that*

i) $g(v, w) = g(Jv, Jw)$,

ii) $\omega(v, w) = g(Jv, w)$.

where $p \in X$ and $v, w \in T_p(X)$.

If X is a complex manifold, then there is a basis for $T^*_X \otimes \mathbb{C}$ consisting of elements of the form dz^j and $d\overline{z}^j$; therefore, any 1-form can be written in local coordinates as $f_i dz^i + g_j d\overline{z}^j$, where $f_i, g_j \in C^\infty(X, \mathbb{C})$. Thus there exists a decomposition $\Omega^1_X \otimes \mathbb{C} = \Omega^{1,0}_X \bigoplus \Omega^{0,1}_X$, where $\Omega^{1,0}_X$ is generated by elements of the form dz^i and $\Omega^{0,1}_X$ is generated by elements of the form $d\overline{z}^j$. Moreover, we can define the vector bundle of (p, q)-type forms $\Omega^{p,q}_X = \bigwedge^p \Omega^{1,0}_X \bigotimes \bigwedge^q \Omega^{0,1}_X$. We then have a decomposition $\bigwedge^k \Omega_X \otimes \mathbb{C} = \sum_{p+q=k} \Omega^{p,q}_X$, and isomorphisms $\overline{\Omega^{p,q}_X} \cong \Omega^{q,p}_X$. See [29] for more details.

The Hodge numbers $h^{p,q}$ for a complex manifold are defined using Dolbeault cohomology. From now on, we consider a compact complex manifold X.

Definition 29. *The* Dolbeault derivatives *are maps $\partial^{p,q} : \Omega^{p,q}_X \to \Omega^{p+1,q}_X$ and $\overline{\partial}^{p,q} : \Omega^{p,q}_X \to \Omega^{p,q+1}_X$ that can be described in local coordinates $z_1, \ldots, z_n$ as follows. Consider a (p, q)-form*

$$\omega = \sum_{|I|=p, |J|=q} f_{I,J} dz^I \wedge d\overline{z}^J.$$

Then we can define

$$\partial^{p,q}\omega = \sum_{I,J}\sum_{k}\frac{\partial f_{I,J}}{\partial z_k}dz^k \wedge dz^I \wedge d\overline{z}^J,$$
$$\overline{\partial}^{p,q}\omega = \sum_{I,J}\sum_{k}\frac{\partial f_{I,J}}{\partial \overline{z}_k}d\overline{z}^k \wedge dz^I \wedge d\overline{z}^J.$$

Theorem 2. *The operators ∂ and $\overline{\partial}$ satisfy $\partial^2 = 0$, $\overline{\partial}^2 = 0$, and $\partial\overline{\partial} + \overline{\partial}\partial = 0$.*

Proof. See [29, Lemma 2.29]. □

Definition 30. *The* Dolbeault cohomology group $H^{p,q}(X)$ *is defined as*

$$H^{p,q}(X) = \frac{ker(\overline{\partial}^{p,q})}{im(\overline{\partial}^{p,q-1})}.$$

Theorem 3. *For a compact Kähler manifold X, we have $H^k(X,\mathbb{C}) = \bigoplus_{p+q=k} H^{p,q}(X)$.*

Proof. See [29, p. 142]. □

Definition 31. *A* Calabi-Yau manifold *is a Kähler manifold that has a global, holomorphic, nowhere-vanishing $(n,0)$-form.*

Some properties of Hodge numbers of a compact Kähler manifold of complex dimension n are as follows:

i) $h^{p,q}(X) = h^{n-p,n-q}(X)$,

ii) $h^{p,q}(X) = h^{q,p}(X)$,

iii) $b_k(X) = \sum_{p+q=k} h^{p,q}(X)$, where b_k is the k th Betti number.

See [15, Chapter 3] for details and proofs. We shall often drop X from the notation where there is no possibility of confusion.

By using these properties, we can draw the *Hodge diamond* of a compact Kähler

manifold X of complex dimension 3 as:

$$
\begin{array}{ccccccc}
 & & & h^{0,0} & & & \\
 & & h^{0,1} & & h^{0,1} & & \\
 & h^{0,2} & & h^{1,1} & & h^{0,2} & \\
h^{0,3} & & h^{2,1} & & h^{2,1} & & h^{0,3} \\
 & h^{0,2} & & h^{1,1} & & h^{0,2} & \\
 & & h^{0,1} & & h^{0,1} & & \\
 & & & h^{0,0} & & &
\end{array}
$$

Let us assume that X is connected for simplicity. Then $b_0(X) = 1$. This implies that $h^{0,0} = 1$. Moreover, if X is simply connected, $H_1(X) = 0$, which implies that $h^{1,0} = h^{0,1} = 0$. If we also assume that X is Calabi-Yau, since there exists a unique (up to scalar) nowhere-vanishing holomorphic $(3,0)$-form on X, we have $h^{3,0} = h^{0,3} = 1$.

As a consequence, the Hodge diamond of a compact, simply connected Calabi-Yau manifold X of complex dimension 3 is:

$$
\begin{array}{ccccccc}
 & & & 1 & & & \\
 & & 0 & & 0 & & \\
 & 0 & & h^{1,1} & & 0 & \\
1 & & h^{2,1} & & h^{2,1} & & 1 \\
 & 0 & & h^{1,1} & & 0 & \\
 & & 0 & & 0 & & \\
 & & & 1 & & &
\end{array}
$$

So it is clear that, in order to determine the Hodge diamond of a Calabi-Yau 3-fold completely, it suffices to give $h^{1,1}$ and $h^{2,1}$.

As a special case, the Hodge diamond of a quintic 3-fold, which has $h^{1,1} = 1$ and

$h^{2,1} = 101$ is:

$$
\begin{array}{ccccccccccc}
 & & & & & 1 & & & & & \\
 & & & & 0 & & 0 & & & & \\
 & & & 0 & & 1 & & 0 & & & \\
 & & 1 & & 101 & & 101 & & 1 & & \\
 & & & 0 & & 1 & & 0 & & & \\
 & & & & 0 & & 0 & & & & \\
 & & & & & 1 & & & & &
\end{array}
$$

2.2 A Brief Introduction to Machine Learning

In 1943, the neurophysiologist Warren McCulloch and mathematician Walter Pitts wrote a paper titled "A Logical Calculus of Ideas Immanent in Nervous Activity" [19], which explains how neurons might work. Moreover, in the 20th century, the development of computer science enabled making simulations about the human brain and this contributed to the development of neural networks. The concept of neural network in Artificial Intelligence was inspired by the neurons in human brain. Researchers tried to abstract how neural networks in the human brain work and wanted to use them as models in Artificial Intelligence. The concept of perceptron was invented in the 1950's by Frank Rosenblatt [21].

Toward the end of the 20th century, machines that could be educated were invented, such as machines who could evaluate positions better than world champions in chess and go were constructed. Of such machines, Deep Blue in chess and Alpha Go in Go are the most famous. Today, machine learning is used in various fields like visualization, image information extraction, pattern recognition, classification, and segmentation. The important point is that machines learn not by qualitative methods or senses; they learn by using numbers and geometry. In other words, the learning of machines is based on quantization of the data in the real world. For example, in order to build a classifier that can understand hand-written digits, one assigns the numerical value of each image, from 0 to 9.

Problems in Machine Learning (ML) can be separated into two types: classification and regression.

Classification is the process of assigning each element in a given set of data into a finite number of discrete classes. In more mathematical terms, classification can be considered as a function $g : X \to Y$ from a set of inputs X to a finite set of outputs Y. The set Y can be considered as a set of labels containing the categories the items will fall into. For example, the problem of categorizing images of animals can be considered. In this example, the set X would contain images of various animals, considered as a rectangular array of numbers, and the set Y would contain the names for these animals. Then the function $g : X \to Y$ maps each image to its corresponding label. Once the training is completed with this data, it becomes possible to present new images to the system and it will predict which animal is on the image. The goal is to train the system in such a way that the predictions of the system on images it has not seen are reasonably accurate. The classification might be *binary* if there are only two possible classes, or *multi-class* if there are more than two classes. The many applications of classification include recognition of handwriting, images, and speech, document classification, search engines, and biometric identification.For more details see[2]

Regression is a technique used to estimate the value of a dependent variable from the values of one or more independent variables. For example, in order to train a simple linear regression algorithm involving one independent variable x and one dependent variable y, one uses a dataset consisting of pairs of numbers (x_i, y_i) for $i \in I$, where I indexes the dataset. The system fits a linear function $y = mx + b$ to this data so as to minimize an error function, which depends on the parameters m and b as well as the dataset. This linear function then can be used to estimate, or predict, values of y for given values of x that are not in the dataset. More complicated examples include linear regression involving more than one independent variable, and polynomial regression using polynomials of any degree $n \geq 1$. For more details see[28]

The main difference between classification and regression is that, while classification problems predict *discrete* outputs, regression problems predict *continuous* outputs.

The three approaches to Machine Learning (ML) are supervised learning, unsupervised learning, and reinforcement learning. Since this thesis will make exclusive use of supervised learning, we will treat it in more detail below. Briefly put, in supervised

learning, the input data set comes attached with the labels for their output; these labels are then used to train the system. However, in unsupervised learning, the data is not always labeled and usually the system needs to cluster or group the dataset regarding the similarities in the dataset. For more details, see [3].

Supervised learning is an approach to Machine Learning where the system is trained using a labeled dataset, that is, a dataset where each entry contains both the inputs and the correct outputs corresponding to those inputs. The neural network method, which is a very commonly used submethod of supervised learning, is used in this thesis. For this method, the system is initialized using arbitrary weights and biases, which are inner parameters of the system. Then, as entries from the dataset are fed into the system, the internal parameters of the model such as weights and biases are adjusted so that the output of the system corresponds more closely to the correct outputs contained in the data set. The machine always optimizes itself to learn better for the next epochs.

A simple machine learning algorithm with neural networks consists of four main phases, namely data collection, data preprocessing, model training, and model evaluation.

Data collection is the process of collecting necessary data to train and evaluate the model. The main goal of this process is to collect the training data of the model and make future predictions.

Data preprocessing is the stage where the raw data is converted into an efficient form for the training phase. For example, if a string data type is converted into numbers or numpy arrays, this is an example of data preprocessing. Also scaling a dataset to the interval $[0, 1]$ is another method of data preprocessing called normalization.

Model training consists of seperating the data set into training data, validation data and the model is trained using the training dataset. Training accuracy must increase, training loss or error must decrease, depending on the increasing number of epochs or ongoing time. The validation stage comes after the training stage with very few data compared to training data. It is the last step before testing the model. The idea is to specify the validation dataset as a dataset, which is different compared

to the training dataset and to check that whether the model is performing well on the validation dataset. If there are problems at the validation stage, the idea is to change the hyperparameters of the model, which are parameters that are used while constructing the model such as the number of epochs, batch size, and number of layers. Afterwards the model is trained again for better validation.

Model evaluation consists of evaluating the performance of the model regarding how well the model makes predictions about the real life data by using the test dataset. In this stage the test accuracy is measured. In homogeneously distributed data, test accuracy is informative about the performance of the model, but if the data is not distributed homogeneously, high test accuracy does not necessarily mean that the model is good. The model may be overfitting. In other words, it may simply be memorizing some patterns. This in deeper sense means that the model is learning irrelevant information within the dataset, which is called noise in machine learning language. For more details, see [27] and [6].

2.2.1 Neural Networks

2.2.1.1 Some important concepts about neural networks

For the references of this subsection see [6],[20]

Definition 32. *A* neural network *is a collection of algorithms, which was produced by inspiration from the way the human brain works. It consists of nodes and layers. The most important part is that neural networks take inputs and produce an output. In other words, they make predictions after some specific inputs are given.*

Definition 33. Neurons *are simple structures which take input and produce output using various algorithms.*

Remark 6. *Every neuron is located in a unique* layer. *There are three types of layers, which are input layers, output layers and hidden layers, and for deep learning existence of hidden layers is necessary.*

Definition 34. *A* densely connected sequential neural network *consists of neurons and layers, and from one layer to the next, all the neurons are connected to each*

other. In other words, every neuron is connected to every neuron in the preceding layer.

Remark 7. *A sequential model can neither take multiple inputs simultaneously nor can it produce multiple outputs simultaneously. Furthermore, the data flow must be in sequential order from layer to layer as illustrated in 2.1. See [8] for more details.*

Remark 8. *This thesis will only include densely connected sequential neural networks.*

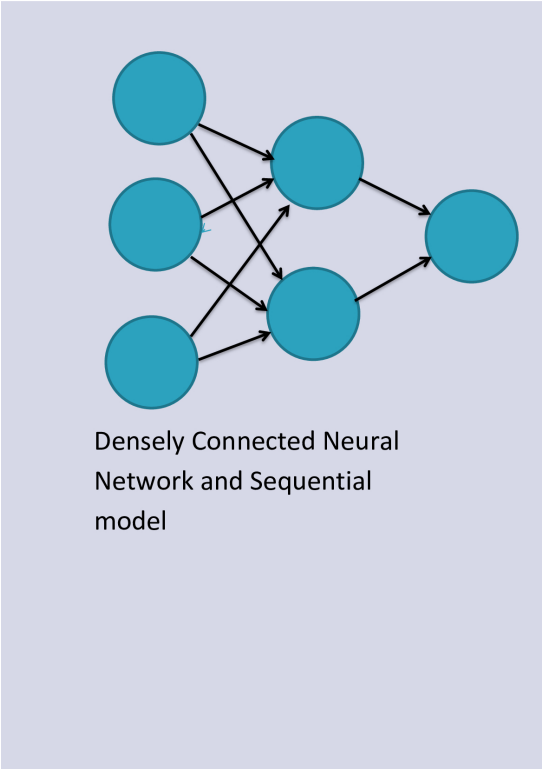
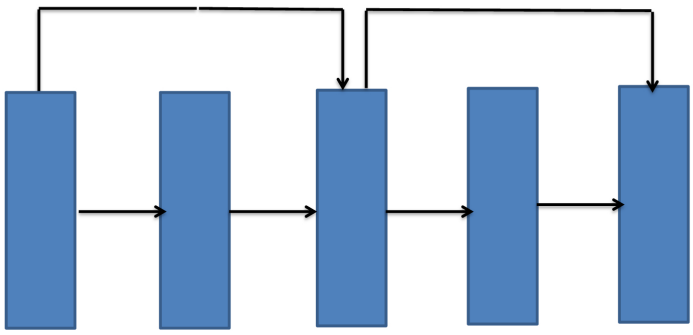


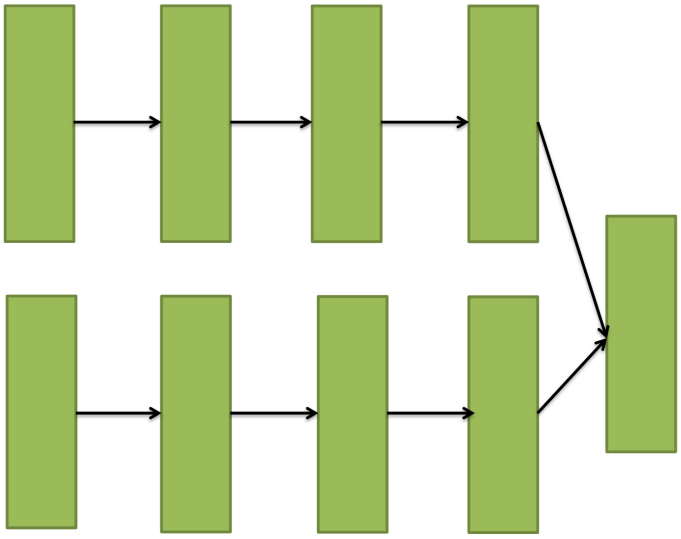
Figure 2.1: Densely connected sequential network

Definition 35. *A* functional model *is a model in which multiple inputs and multiple outputs can occur at the same time. Moreover, layers can connect not only in sequential order, but they can also connect to other forward or backward layers unlike in a sequential model. See [7] for more details.*

Remark 9. *Figure 2.2 shows two examples of functional models with non-sequential data flow and multiple inputs respectively.*



Non sequential data flow and each box represent one of the layers



Example of a multiple input non sequential model and each box represents one of the layers

Figure 2.2: Examples of non-sequential models

2.2.1.2 Some parameters and hyperparameters of a neural network

Definition 36. *The* parameters *of a neural network are some numbers, which are optimized by the neural network itself. Weights and biases are the basic parameters of the neural network.*

Definition 37. *The* hyperparameters *are the parts of the neural network, which the designer of the neural network optimizes. For example, batch size, number of epochs, and number of layers are the main hyperparameters of a sequential neural network.*

Definition 38. *The* batch size *is the number of minimum training samples for the model to update its parameters. The idea is to divide the whole dataset into chunks and make those chunks pass through the model and obtain an output. Then the model calculates its training error for the data in that batch and updates some of its parameters. Optimizing the batch size is most of the time necessary to guarantee the decrease of training error over time.*

Definition 39. *The* number of epochs *is defined as the number of times for the whole dataset to pass through the neural network. After each epoch, the neural network is optimized considering all samples in the dataset, so the training of a neural network is done by considering each epoch. The values like accuracy and loss are calculated for each epoch.*

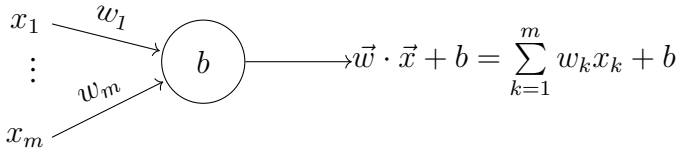
Definition 40. *The* weights *between neurons are a measure of the impact of change in the output of one neuron to the input of the next neuron that it is connected to. For example, if we have very low weights between neurons, this means that increasing the output of one neuron will not affect too much the input of the other one which it is connected to.*

Definition 41. *The* bias *is a parameter inside each neuron, which is used to shift the output value of each neuron. For example, when $w.x$, which is the dot product of the weight vector and the input vector is negative or it is zero, which most of the time is not useful, by using the bias value it is possible to make this value positive or make it inside the required interval range. Thus bias is a value that is inside each neuron and that is used to regulate the output of the neurons. This is important for the optimization process after each epoch.*

Remark 10. *The following definitions will clarify the inputs and outputs of each neuron when specific inputs are given to the neural network.*

Definition 42. *A* perceptron *is a function that takes an input and produces an output by the below formula, where $\vec{w}$ denotes the vector weights b the bias number of the neuron and $\vec{x}$ is the input vector of the neuron.*

$$output = \begin{cases} 1 & \text{if } \vec{w} \cdot \vec{x} + b > 0 \\ 0 & \text{if } \vec{w} \cdot \vec{x} + b \leq 0 \end{cases}$$



Definition 43. *An* activation function f *is a function that takes the weighted sum of inputs $\sum_{k=1}^{d} w_k x_k + b$ and maps it to the new value $f(\sum_{k=1}^{d} w_k x_k + b)$.*

Remark 11. *The reason for using such a function is to minimize the error in the neural network. For example, some activation functions convert the output number to a number between 0 and 1. Regarding the classification problems whose output is 0 or 1, using an activation function minimizes the gap between the predicted output and the desired output, which is defined as the* error. *Moreover, some activation functions add non-linearity to the neural network. This is crucial in some cases because linear structures do not always work in complex situations. Details can be found in [17]. For example, in the case of an image classification problems, the non-linearity of the activation function is crucial for the effectiveness of the models.*

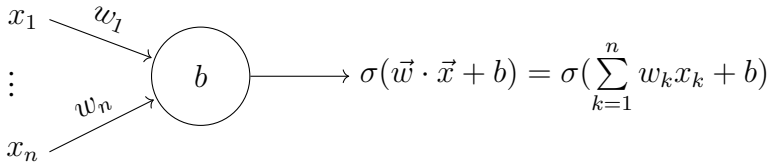
To sum up, activation functions are functions that are used to increase the performance of the model with regards to the distribution of the data set by minimizing the gap between predictions and desired outputs, and in many cases non-linear activation functions are needed to improve the performance of the model.

Example 12. *Some examples of activation functions are* relu, sigmoid *and* tanh *func-*

tions, defined as

$$relu(x) = max(x, 0),$$
$$sigmoid(x) = \frac{1}{1 + e^{-x}},$$
$$tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1}.$$

Definition 44. *A* sigmoid neuron *is a function that takes an input vector $\vec{x}$ and produces an output number $\sigma(\vec{w}\cdot\vec{x}+b)$, where σ is an activation function, $\vec{w}$ is the weight vector of the connection and b is the bias of the neuron.*



Definition 45. *The* loss function *is a function denoted by L, that measures the error between the desired output and the actual output. Through the loss function one can measure how far the actual output is from the predicted output. There are many kinds of loss functions. The simplest type of loss function is the* mean squared loss *denoted by* MSE *and it can be written as $L(\vec{y}, \vec{\overline{y}}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \overline{y_i})^2$ where $\vec{y} = (y_1, \ldots, y_n)$ is the vector of desired values and $\vec{\overline{y}} = (\overline{y}_1, \ldots, \overline{y}_n)$ is the vector of the predicted values of the neural network, which are values that neural network predicts. Also, there are losses that include logarithms. One of them, which is used commonly in classification problems is* crossentropy *loss and its formula is given by $L(y_i, \overline{y}_i) = -(\sum_{i=1}^{n}(y_i \log(\overline{y}_i))$.*

2.2.2 Some machine learning algorithms

For more details on this subsection, see [6], [20].

2.2.2.1 Forward propagation algorithm

Forward propagation is the flow of the given input forward through the network and obtaining an output using the activation functions, weights, and biases of the neurons.

For sequential models, the output of one neuron is input of another one in the next layer. For example, assume that neuron i in layer $l-1$ is connected to neuron j in layer l. If the output of neuron i in layer $l-1$ is $\sigma(\vec{w}\cdot\vec{x}+b)$, then this becomes one of the input numbers of neuron j in layer l and so on until the last layer, which gives the output of the neural network. This process is called forward propagation.

2.2.2.2 Gradient descent algorithm

One of the main issues in the training stage is to minimize the loss function for each epoch. The neural network uses some algorithms to optimize its weights and biases. One of the main optimization algorithms is *gradient descent*. This is based on reducing the loss function by using its gradient vector, which consists of its partial derivatives with respect to the weights and biases. The formula for gradient descent is given as $(\overline{b},\overline{w}) = (b,w) - \lambda\nabla L(b,w)$. Here w is one of the particular weights of the connection and b is the bias of the neuron to which the arrow with weight w is connected. Moreover, $\overline{b}$, $\overline{w}$ are the optimized bias and weight of the same neuron and connection respectively. Since L, the loss function, depends on all the biases and weights in the neural network, this algorithm is applied for each layer.

2.2.2.3 Backpropagation algorithm

Assume that we have a loss function

$$C = \frac{1}{2n}\sum_{x}\left\|y(\vec{x}) - a^{L}(\vec{x})\right\|^{2},$$

where n is the number of training samples, $\vec{x}$ is the input vector, $y(\vec{x})$ is the desired output vector corresponding to that input vector, and $a^{L}(\vec{x})$ is the predicted output vector.

Our aim is to find the partial derivatives of the loss function with respect to all biases and weights in the neural network, to implement the backpropagation algorithm. First, some notation on the weights and biases. Let $w^{l}_{j,k}$ be the weight for the connection from the kth neuron in the $l-1$th layer to the jth neuron in the lth layer. Let b^{l}_{j} be the bias for the jth neuron in the lth layer. Also define a^{l-1}_{k} as the activation output

of the kth neuron in $l-1$th layer, which means $a_k^{l-1} = \sigma(w \cdot x + b)$ where σ is the activation function and $\vec{w}$ is the weight vector, which consists of the weights of the nodes connected to the kth neuron in the $l-1$th layer and $\vec{x}$ is the input vector of the kth neuron in the $l-1$th layer. We then calculate the dot product of them and add the bias of the kth neuron in the $l-1$th layer which can be written as b. This convention allows us to write the output of the jth neuron in the lth layer,as $\sigma(z_j^l) = a_j^l$, where $z_j^l = \sum_k w_{j,k}^l a_k^{l-1} + b_j^l$.

We now describe the steps of the backpropagation algorithm using this notation.

- Assume that there exist L layers which are $0, 1, \ldots, L$. Compute the partial derivatives
$$\frac{\partial C}{\partial z_j^l} = \frac{\partial C}{\partial a_j^l} \times \frac{\partial a_j^l}{\partial z_j^l}$$
for all l in $\{0, 1, \ldots L\}$.

- We write
$$\frac{\partial C}{\partial z_j^l} = \sum_k \frac{\partial C}{\partial z_k^{l+1}} \times \frac{\partial z_k^{l+1}}{\partial z_j^l}.$$
This holds because $z_k^{l+1} = \sum_j w_{k,j}^{l+1} a_j^l + b_k^{l+1}$.

- Write
$$\frac{\partial C}{\partial w_{j,k}^l} = \frac{\partial C}{\partial z_j^l} \times \frac{\partial z_j^l}{\partial w_{j,k}^l},$$
$$\frac{\partial C}{\partial b_j^l} = \frac{\partial C}{\partial z_j^l} \times \frac{\partial z_j^l}{\partial b_j^l}.$$

The backpropagation algorithm ends above but if we want to combine it with gradient descent algorithm we optimize the weights and biases as below.

For all $l \in \{0, 1, \ldots L\}, j, k$ write
$$\bar{w}_{j,k}^l = w_{j,k}^l - \lambda \times \frac{\partial C}{\partial w_{j,k}^l},$$
$$\bar{b}_j^l = b_j^l - \lambda \times \frac{\partial C}{\partial b_j^l}.$$

The above algorithm optimizes the weights and biases of the whole network at the end of each batch where λ is a constant that is subject to change. See [20] for more details.

2.2.3 Measuring the Performance of the Model

2.2.3.1 Training, Validation, Testing Accuracy and Loss Curves

To measure the performance of the model, training accuracy, training loss, validation accuracy, validation loss, and test accuracy are taken into consideration. While training loss measures the error of the model on the training dataset, validation loss measures the error of the model on the validation dataset. How those parameters change with the increase in number of epochs is crucial. Firstly, training and validation accuracy and training and validation loss must be checked together. For a good model, there should not be a big gap between the training and validation curves and training accuracy must increase while training loss must decrease over time. While training accuracy and loss are behaving in such a way, validation loss must also decrease and must be close to training loss. It is necessary for test accuracy to be high for a good model but having high test accuracy is not enough to conclude that the model is learning well.This also depends on training and validation curves as mentioned above. While the loss function may change depending on the model, accuracy is the most common metric used in classification problems and is formulated as

$$\frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

In deep learning models accuracy is written after each epoch. For more details see [4],[6]

2.2.3.2 Confusion Matrix

Confusion matrix is an extremely useful tool to evaluate the performance of a machine learning model for classification problems in two classes. It gives information about the correct and incorrect predictions of the model. We can assume those are 0 and 1 without loss of generality. The confusion matrix gives information about the correct and incorrect predictions of the model. Moreover, this matrix gives an idea about whether the model has learned well or whether it is overfitting. See more details from [26]

Table 2.1: Confusion matrix

	Actual positive	Actual negative
Predicted positive	Number of correct predictions when the actual value is positive	Number of positive predictions when the actual value is negative
Predicted negative	Number of negative predictions when the actual value is positive	Number of correct predictions when the actual value is negative

2.2.3.3 Overfitting and underfitting

Underfitting of a model means that the model does not learn training data well. In this case, at the end of the training stage, the model will have still low training accuracies and high training losses, whereas overfitting is the converse of this in terms of learning the training data. In the broadest sense, overfitting can be defined as learning from the training dataset so well that the model can not go beyond the scope of that dataset.

There are two important concepts about overfitting and underfitting. The first one is *optimization* and the second one is *generalization*. The optimization process for the whole dataset occurs after each epoch in the training phase of the model. The model optimizes its weights and biases to obtain better predictions. This optimization for each epoch must be performed well by the model itself to prevent underfitting. Generalization is about how well the model can generalize its learning on the training samples to the test samples. At the beginning of the training stage, optimization and generalization are most of the time positively correlated, so when the model optimizes its weights, biases, it tends to generalize better. However, this situation may change when the number of epochs increases. While the neural network still optimizes well, generalization may not increase. This problem is called the overfitting problem. If the model can not generalize well, it means that it memorizes some patterns and tries to apply those to the new dataset. See [6] for more details.

The decision of whether the model overfits or underfits or neither may be made using the training and validation curves, confusion matrix and test accuracy. If there is no

increase in training accuracy and no decrease in training loss from the first epoch to the last one, this is a sign of overfitting or underfitting depending on the training accuracy and loss. Moreover, if the validation loss increases while training loss decreases and a gap occurs between them after some epochs with high training accuracy, this is a sign of overfitting. See [4] for details.

Lastly, we can check the confusion matrix to determine overfitting. For example, in a binary classification problem, if the confusion matrix assigns almost everything to one class, this is another sign of overfitting regardless of test accuracy. See [26].

2.2.4 The Calabi-Yau Dataset

We briefly describe the dataset we use in this thesis. The dataset was obtained from the website

`http://hep.itp.tuwien.ac.at/~kreuzer/CY/`

titled "Calabi-Yau Data" by Maximilian Kreuzer and Harald Skarke. Specifically, on the tab "CY / 4d" the dataset involving 184026 weights under the heading "K3 fibered toric CY hypersurfaces" was downloaded. This produced a text file whose lines have the format

1 1 1 1 1 5=d TS H: 1 101 M:126 5 N: 6 5 P:0 F:0

Here the first five entries are the weights of the homogeneous coordinates in the weighted projective 4-space. The sixth entry simply gives the degree of the polynomial defining the Calabi-Yau 3-fold, which must be equal to the sum of the degrees. The two entries "H: 1 101" indicate the Hodge numbers $h^{11} = 1$ and $h^{12} = 101$ for this particular Calabi-Yau 3-fold. The letter "T" indicates that the polynomial defining this Calabi-Yau 3-fold is transverse. In this thesis, we consider only the subset of the dataset containing "T" for simplicity, since this restricts to a subset of 7555 entries.

CHAPTER 3

THE CODE, RESULTS, AND DISCUSSION

In this chapter, we present the code, the results obtained by the code, and the discussion of the results. Section 3.1label presents the code in small chunks and discusses their functions briefly. Section 3.2 presents the results of the running of the code in table format. Section 3.3 is a brief discussion of the results obtained.Section 3.4 is the results of some possible changes in hyper parameters of the model in table format.Section 3.5 is a discussion of results written in section 3.4

3.1 The Code: Presentation and Discussion

We advise the reader that some lines in the code presented below have been split into two for typesetting purposes.

```
pip install -U scikit-learn
import os
for dirname,_,filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

This is code added to the beginning of the project automatically by Kaggle.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Activation, Dense
from tensorflow.keras.utils  import to_categorical
```

```
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.model_selection import train_test_split
import seaborn as sns
from numpy import argmax, asarray
import matplotlib.pyplot as plt
```

At the beginning of the code, we import a number of packages, classes, and functions. Tensorflow is one of the main libraries for machine learning purposes in Python. A *tensor* can be considered as a multidimensional array of numbers.[1] For example, $[[1,2,3],[4,5,6],[7,8,9]]$ is a tensor. The Tensorflow library is used not only to do operations such as multiplication and addition on tensors, but also it is used to train neural networks. In addition, Tensorflow includes Keras, which is one of the main libraries used to construct machine learning models. Sequential is a class used to model layers in sequential order. In sequential models, the previous layer gives its outputs as inputs to the next layer.Not only the layers do not cross each other but also every layer possesses a unique input and an output tensor at a time. Another library is Seaborn, which is one of the main libraries for data visualization. We use Seaborn to draw histograms of the Calabi-Yau dataset. While the `numpy.argmax()` function returns the index of the maximum element in a list, we use `numpy.asarray()` to convert the input data to a Numpy array. This is necessary for training the neural network. We use `matplotlib.pyplot` to plot lines or figures. Lastly, the `sklearn` library is used in many problems in machine learning such as classification and regression. Here we import the function `train_test_split()` and the module `ConfusionMatrixDisplay` from the `sklearn` library.

`ConfusionmatrixDisplay` displays a confusion matrix which measures the performance of the model by showing its true and false predictions in the form of a 2×2 matrix. Moreover, we use `train_test_split` to split the data into training and testing and validation sets to construct a machine learning model.

[1] This must not be confused with the tensor product in linear algebra.

```
def read_file(filename, key="T"):
    lines = open(filename).readlines()
    print("Initial file has", len(lines), "items.")
    result = []
    for l in lines:
        if key in l:
            result.append(l)
    print("Resulting list of entries has",
        len(result), "items.")
    return(result)
```

This code reads the file containing the dataset and produces a list, named `result`, containing the lines of the file in which the key, whose default value "T", appears. First, the file is opened and it is read line by line using the `readlines()` function. After this operation, an empty list is created. In a `for` loop, the code goes through all the lines in the file, and each line is added to the `result` list if the key appears in it.Consequently a list called `result` consisting of strings is created Then, the function returns the result.

```
calabi_yau_entries =
    read_file("../input/hodge-data/hodge data.gz")
```

Here we apply the function `read_file()` defined above to the dataset, which then produces its result, containing those lines in the dataset with the key "T".

```
def prepare_training_test(input_list,lower_limit=50,
    upper_limit=500,h12=True):
    data_x=[]
    data_y=[]
    data_z=[]
    for l in input_list:
        y=l.split()
        s=[float(y[0]),float(y[1]),
            float(y[2]),float(y[3]),float(y[4])]
```

```
        data_x.append(s)
        hodge_number=0
        if h12:
            hodge_number=float(y[8])
        else:
            hodge_number=float(y[7][2:])
        data_z.append(hodge_number)
        if(hodge_number>=lower_limit
        and hodge_number<upper_limit):
            data_y.append(1)
        else:
            data_y.append(0)
    return data_x,data_y,data_z
```

[2] This function takes the list `input _list` as well as two numbers `lower_limit` and `upper_limit` with default values which are 50 and 500 respectively. It also takes a boolean argument `h12` that determines whether the function will consider the Hodge numbers h^{12} or h^{11} (when `h12=False`). Then, it forms three empty lists, which are `data_x` ,`data_y`, `data_z`.

The `for` loop takes each string in the `input_list` and splits it across whitespace using `y=l.split()`. As a result we have a new list `y`. The first five entries of this list, which give the parameters of the Calabi-Yau manifold in question, are converted into floating-point numbers, assembled into a new list `s`, and appended to `data_x`. Next, there are two Hodge numbers in the string `input_list`, and the `if` statement in the code decides which one to choose. Finally, that Hodge number is added to the list `data_z`, and 0 or 1 is added to `data_y` depending on whether the Hodge numbers lies in the interval defined by `lower_limit` and `upper_limit`. These three lists, namely `data_x` ,`data_y`, `data_z`, are then returned.

```
data_x,data_y,data_z=
prepare_training_test(calabi_yau_entries)
data_x=asarray(data_x)
```

[2] The for loop in the code contains all the subsequent lines except for the return line.

```
data_y=asarray(data_y)
```

The `numpy` function `asarray` converts the lists `data_x` and `data_y` into a form that can be used by the machine learning packages.

```
train_x,rem_x=train_test_split
(data_x,train_size=0.8,random_state=42)
train_y,rem_y=train_test_split
(data_y,train_size=0.8,random_state=42)
test_x,val_x=train_test_split
(rem_x,test_size=0.5,random_state=42)
test_y,val_y=train_test_split
(rem_y,test_size=0.5,random_state=42)
```

The above code splits the training and testing data with a ratio of %80 for training and %10 for testing and %10 for validation.

```
network=Sequential()
network.add(Dense(50,activation="sigmoid",
input_shape=(5,)))
network.add(Dense(10,activation="sigmoid"))
network.add(Dense(2,activation="sigmoid"))
network.compile(optimizer='Adam',
loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
```

The above code constructs a sequential network and adds four layers to the model, including the input layer. The layers of the neural network are all dense. That means that two consecutive layers form a bipartite graph. The input layer has five neurons and is connected to 50 neurons in the second layer with an activation function called `relu`. The third layer has 10 neurons with `relu` activation function as well. The fourth layer has 2 neurons with `sigmoid` activation function. After adding these layers, the code compiles the neural network, which is necessary before training it, since

the compile command enables the network to define a loss function, optimizer, and metrics to get prepared for the training stage.

```
def train_and_evaluate_network(number_of_epochs,
    given_batch_size):
    history=network.fit(train_x,train_y,
    validation_data=(val_x,val_y),
    epochs=number_of_epochs,batch_size=given_batch_size)
    test_loss,test_acc=network.evaluate(test_x,test_y)
    print('test_acc:',test_acc)
    predict_y=network.predict(test_x,verbose=0)
    rounded_predictions=argmax(predict_y,axis=1)
    plt.plot(history.history['loss'],label=
    'Training Loss')
    plt.plot(history.history['accuracy'],
    label='Training Accuracy')
    plt.plot(history.history['val_ accuracy'],
    label='Validation accuracy')
    plt.plot(history.history['val_loss'],
    label='Validation loss')
    plt.xlabel('Number of epochs')
    plt.legend(loc="center")
    print(predict_y)
    plt.show

    return rounded_predictions
```

The function train_and_evaluate network takes two parameters, namely number_of_epochs and given_batch_size. The function network.fit is used to train the network using the training and validation datasets train_x and train_y,(val_x,val_y) which were produced by the train_test_split function before respectively. The number of epochs is the number of times the given

dataset will pass through the whole neural network. The batch size is used to split the dataset into parts and to enable the dataset to pass through the neural network in these small parts. This results in more effective training. After training the neural network and putting the results in the variable `history`, the `network.evaluate` function evaluates the performance of the neural network by using the test dataset. Moreover, `predict_y` is the prediction matrix on the test dataset. Then the above code displays the loss and the accuracy of the neural network as a function of the number of epochs. The loss and the accuracy graphs help to evaluate the performance of the model. If there is a healthy increase in accuracy and similarly decrease in loss, this means that the neural network is learning well without overfitting.

```
epochs=100
batches=100
rounded_predictions=
train_and_evaluate_network(epochs,batches)
ConfusionMatrixDisplay.from_predictions
(test_y,rounded_predictions)
print(rounded_predictions)
plt.show()
```

The above code runs the function `train_and_evaluate_network` with number of epochs equal to 100 and the batch size equal to 100. It also creates the confusion matrix of the model, which is a very helpful tool in evaluating the performance of the model, as the confusion matrix enables us to see the number of true and false predictions of the model.

```
sns.histplot(data_z)

def interval(a,b):
    counter=0:
    for i in data_z:
        if(i>a and i<b):
            counter=counter+1
```

```
    print(len(data_z)-counter)
    return(counter)
```

The above code creates a histogram of `data_z`, which shows the distribution of `data_z`. Also the interval function gives information on how many elements of `data_z` is in the given interval. By considering the returning value of this function we can know the distribution of `data_y` more exactly compared to histogram which gives intuitive information about distribution of `data_y`.

3.2 Results of interval tables

We present the test accuracies for various intervals for h^{12} below. In each row, the entry on the left specifies the interval, which determines the values of 0 and 1 that will be added to `data_y` and the entry on the right gives the test accuracy as a result of training and testing the neural network for that specific interval. For example, the third row in Table 3.2, which reads $50 \leq h^{12} < 75$ on the left and 0.88 on the right, indicates that the code is executed and the function `prepare_training_test` is run with parameters `lower_limit = 50` and `upper_limit = 75`, as well as `h_12 = True`. At the end of this process the `train_and_evaluate_network` function is run, and the resulting test accuracy is 0.88.

Table 3.1: $h^{12} \in [N,500)$, inc.=25

$0 \leq h^{12}$	1.00
$25 \leq h^{12}$	0.94
$50 \leq h^{12}$	0.95
$75 \leq h^{12}$	0.96
$100 \leq h^{12}$	0.96
$125 \leq h^{12}$	0.97
$150 \leq h^{12}$	0.99
$175 \leq h^{12}$	0.99
$200 \leq h^{12}$	0.99
$225 \leq h^{12}$	0.99
$250 \leq h^{12}$	1.00
$275 \leq h^{12}$	1.00
$300 \leq h^{12}$	1.00
$325 \leq h^{12}$	1.00
$350 \leq h^{12}$	1.00
$375 \leq h^{12}$	1.00
$400 \leq h^{12}$	1.00
$425 \leq h^{12}$	1.00
$450 \leq h^{12}$	1.00
$475 \leq h^{12}$	1.00

Table 3.2: $h^{12} \in [N, N+25)$, inc.=25

$0 \leq h^{12} < 25$	0.91
$25 \leq h^{12} < 50$	0.89
$50 \leq h^{12} < 75$	0.88
$75 \leq h^{12} < 100$	0.93
$100 \leq h^{12} < 125$	0.96
$125 \leq h^{12} < 150$	0.96
$150 \leq h^{12} < 175$	0.98
$175 \leq h^{12} < 200$	0.99
$200 \leq h^{12} < 225$	0.99
$225 \leq h^{12} < 250$	0.99
$250 \leq h^{12} < 275$	1.00
$275 \leq h^{12} < 300$	1.00
$300 \leq h^{12} < 325$	1.00
$325 \leq h^{12} < 350$	1.00
$350 \leq h^{12} < 375$	1.00
$375 \leq h^{12} < 400$	1.00
$400 \leq h^{12} < 425$	1.00
$425 \leq h^{12} < 450$	1.00
$450 \leq h^{12} < 475$	1.00
$475 \leq h^{12} < 500$	1.00

Table 3.3: $h^{12} \in [N, 500)$, inc.=50

$0 \leq h^{12}$	1.00
$50 \leq h^{12}$	0.95
$100 \leq h^{12}$	0.96
$150 \leq h^{12}$	0.99
$200 \leq h^{12}$	0.99
$250 \leq h^{12}$	1.00
$300 \leq h^{12}$	1.00
$350 \leq h^{12}$	1.00
$400 \leq h^{12}$	1.00
$450 \leq h^{12}$	1.00

Table 3.4: $h^{12} \in [N, N + 50)$, inc.=50

$0 \leq h^{12} < 50$	0.96
$50 \leq h^{12} < 100$	0.91
$100 \leq h^{12} < 150$	0.95
$150 \leq h^{12} < 200$	0.97
$200 \leq h^{12} < 250$	0.98
$250 \leq h^{12} < 300$	0.99
$300 \leq h^{12} < 350$	1.00
$350 \leq h^{12} < 400$	1.00
$400 \leq h^{12} < 450$	1.00
$450 \leq h^{12} < 500$	1.00

Table 3.5: $h^{12} \in [N, 500)$,inc.=100

$0 \leq h^{12}$	1.00
$100 \leq h^{12}$	0.96
$200 \leq h^{12}$	0.99
$300 \leq h^{12}$	1.00
$400 \leq h^{12}$	1.00

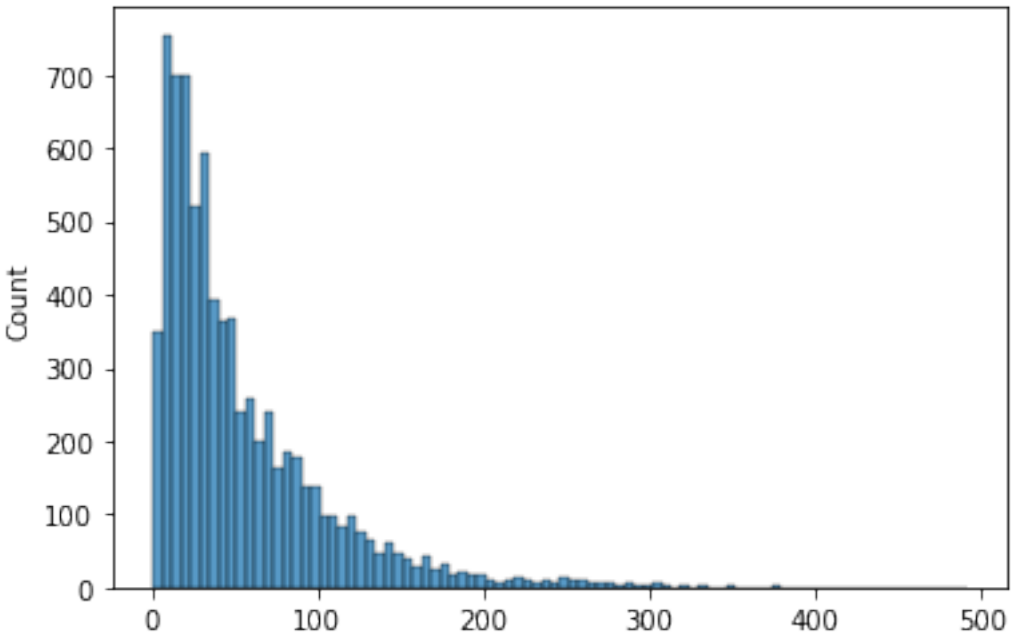
Table 3.6: $h^{12} \in [N, N + 100)$,inc.=100

$0 \leq h^{12} < 100$	0.96
$100 \leq h^{12} < 200$	0.94
$200 \leq h^{12} < 300$	0.99
$300 \leq h^{12} < 400$	1.00
$400 \leq h^{12} < 500$	1.00

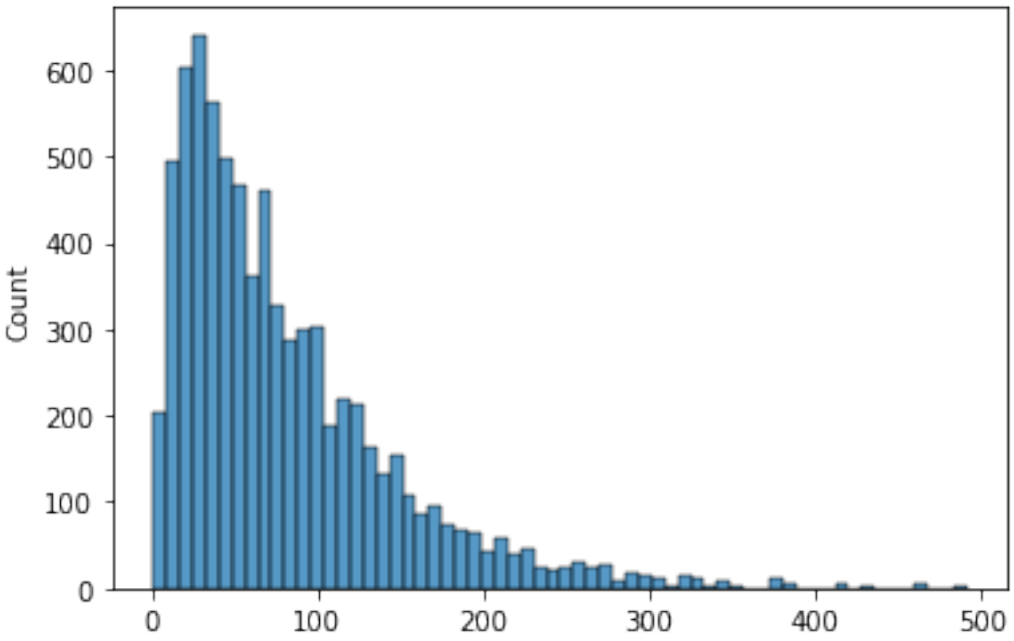
3.3 Discussion of the results of interval tables

The main reason for giving the tables above is to emphasize that, although test accuracy is one of the measures we can use to conclude that the model is performing well, sometimes it can be quite misleading to evaluate the performance of the model by considering the test accuracy exclusively. This is a problem for classification problems in particular. It is possible to see more details from [5]. For many intervals in the tables given above, the distribution of h^{12} and h^{11} are not homogeneous. Therefore, the distribution of the dataset of y values is not homogeneous including `train_y` and `test_y`. For example, if the cases $350 \leq h^{12}$ and $350 \leq h^{11}$ are considered, there are much more h^{12} and h^{11} values less than 350, and considering the `prepare_training_test` function of our code, there are much more 0's for the dataset y, compared to 1's. As a result, there are much more 0's in `train_y` and `test_y` than 1's. It is possible to see this by looking at the histograms of h^{12} in Figure 3.1a and h^{11} in Figure 3.1b. Consequently, the model does not really learn. It instead learns a pattern, which classifies approximately everything as 0 and it makes correct predictions. But the main reason for the correctness of these predictions is that in the test dataset almost everything is 0 as well. This leads to a high test accuracy by overfitting. This fact is an example of training of the neural network with an imbalanced dataset. Overfitting in this case can better be understood by learning curves and confusion matrices as depicted below in Figures 3.2, 3.3, 3.4, and 3.5.

If the interval [350, 500) for h^{12} and h^{11} is considered in the `prepare_train_test` function, then the learning curves and confusion matrices below represent non-learning models. For example, regarding Figure 3.2 the training accuracy, validation accuracy and validation loss curves are constant. Consequently, there is no improvement in the model over time. Regarding the confusion matrix of the same model in Figure 3.3, almost everything is 0. This is a clear sign of overfitting. Thus we can conclude that the model is not performing well although it has high accuracy. This situation is similar for the graphs in Figures 3.4, 3.5 belonging to h^{11} and for many other intervals.



(a) Histogram for h^{12}



(b) Histogram for h^{11}

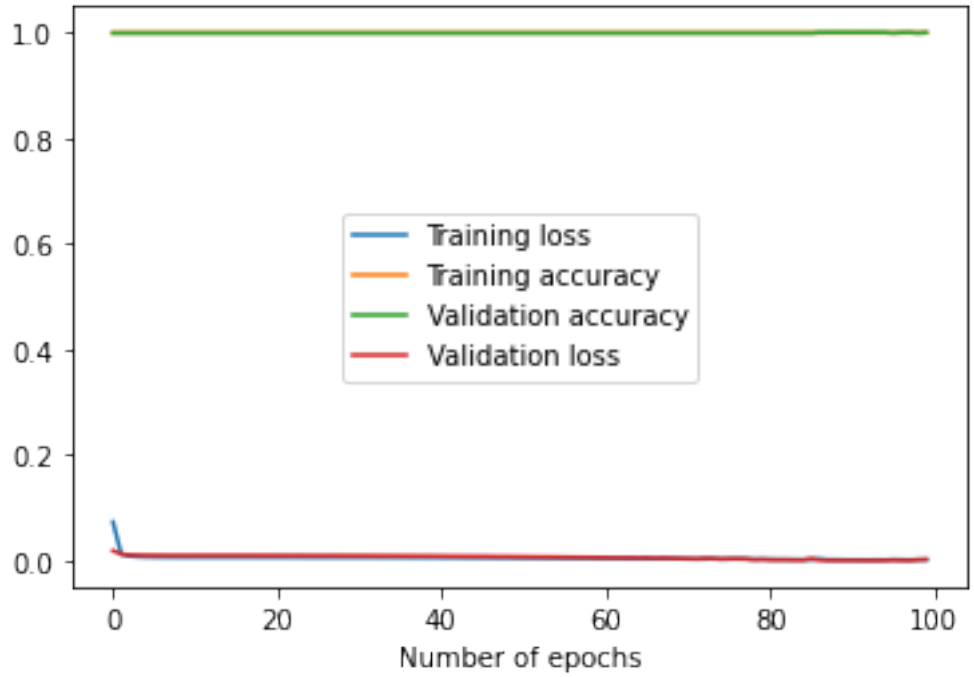


Figure 3.2: h^{12}=True, with interval $[350, 500)$

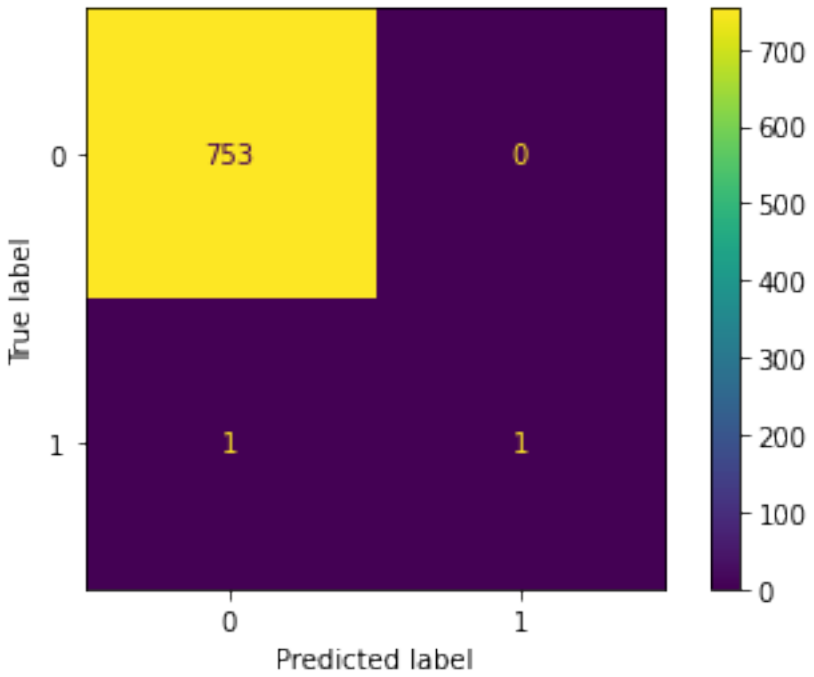


Figure 3.3: Confusion matrix for h^{12}=True, with interval $[350, 500)$

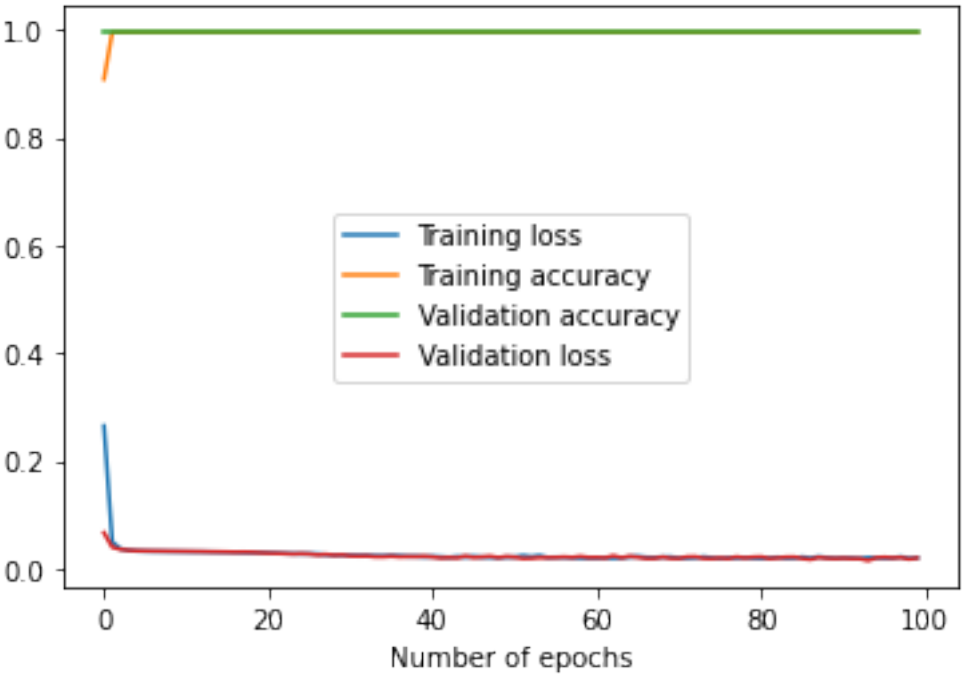


Figure 3.4: h^{12}=False, with interval $[350, 500)$

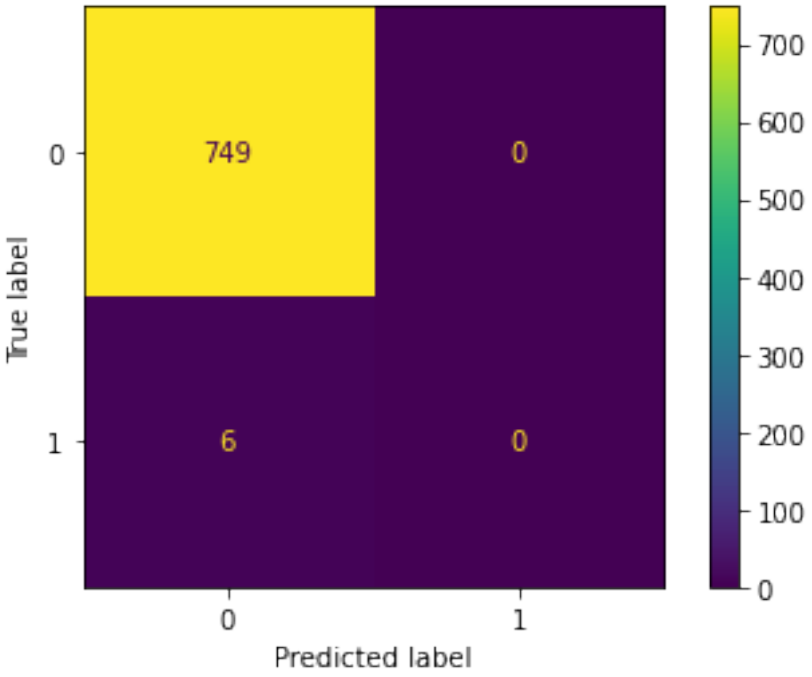


Figure 3.5: Confusion matrix for h^{12}=False, with interval $[350, 500)$

3.4 Results of main tables

3.4.1 Some alternatives for the constructed model and accuracies with lower limit 50 and upper limit 500 for the hodge number h^{12}

Remark 12. *Table 3.7 represents the hyperparameters of our model, which was mentioned in Chapter 2. In the next pages we change only one of the hyperparameters of this model and compare it with the models with exactly one different hyperparameter compared to the model mentioned in Chapter 2.*

Table 3.7: Hyperparameters of main model

Number of hidden layers	2
Number of neurons in hidden layers	$(50, 10)$
Activation function	sigmoid
loss	sparse categorical crossentropy
optimizer	Adam
metrics	Accuracy
number of epochs	100
batch size	100
Training accuracy at the end of first epoch	0.53
Training accuracy at the end of last epoch	0.95
Training loss at the end of first epoch	0.71
Training loss at the end of last epoch	0.11
Validation accuracy at the end of first epoch	0.63
Validation accuracy at the end of last epoch	0.96
Validation loss at the end of first epoch	0.32
Validation loss at the end of last epoch	0.06
Test accuracy	0.95
Good fit model	Yes

Remark 13. *In Table 3.8 we vary only the activation function of the main model, which is the model discussed in Chapter 2. The activation functions of the new models*

are tanh *and* relu *respectively. The data at the rightmost column represents the data of our main model. The data of the columns beginning with* tanh *and* relu *represent the accuracies and losses of the two models for which only the activation function differs from the main model.*

Table 3.8: Activation function varied

tanh	relu	sigmoid
T.A.E.F=0.59	T.A.E.F=0.36	T.A.E.F= 0.53
T.A.E.L=0.46	T.A.E.L.=0.91	T.A.E.L= 0.95
T.L.E.F=1.24	T.L.E.F=0.94	T.L.E.F= 0.71
T.L.E.L=0.69	T.L.E.L=1.44	T.L.E.L=0.11
V.A.E.F=0.63	V.A.E.F=0.91	V.A.E.F=0.63
V.A.E.L=0.44	V.A.E.L=0.91	V.A.E.L= 0.96
V.L.E.F =0.68	V.L.E.F=1.43	V.L.E.F= 0.32
V.L.E.L=0.69	V.L.E.L=1.43	V.L.E.L=0.06
T.A=0.46	T.A=0.91	T.A=0.95

Remark 14. *In Table 3.9 we vary only the batch size of the main model. The new batch sizes are* 6000 *and* 1000 *respectively.*

Table 3.9: Batch size varied

batch size=6000	batch size=1000	batch size=100
T.A.E.F=0.61	T.A.E.F=0.39	T.A.E.F=0.53
T.A.E.L=0.61	T.A.E.L=0.39	T.A.E.L=0.95
T.L.E.F=0.50	T.L.E.F=0.49	T.L.E.F=0.71
T.L.E.L=0.41	T.L.E.L=0.39	T.L.E.L=0.11
V.A.E.F= 0.63	V.A.E.F=0.37	V.A.E.F=0.63
V.A.E.L=0.63	V.A.E.L=0.37	V.A.E.L=0.96
V.L.E.F=0.50	V.L.E.F=0.47	V.L.E.F=0.32
V.L.E.L=0.40	V.L.E.L=0.37	V.L.E.L=0.06
T.A=0.61	T.A=0.39	T.A=0.95

Remark 15. *In Table 3.10 we vary only the number of epochs of the main model. The epoch numbers, which are different from the main model are* 1000 *and* 7500 *respectively.*

Table 3.10: Number of epochs varied

number of epochs=1000	number of epochs=7500	number of epochs=100
T.A.E.F=0.61	T.A.E.F=0.52	T.A.E.F=0.53
T.A.E.L=0.96	T.A.E.L=0.99	T.A.E.L=0.95
T.L.E.F=0.86	T.L.E.F=0.71	T.L.E.F=0.71
T.L.E.L =0.09	T.L.E.L=0.04	T.L.E.L=0.11
V.A.E.F=0.63	V.A.E.F=0.63	V.A.E.F=0.63
V.A.E.L=0.96	V.A.E.L=0.96	V.A.E.L=0.96
V.L.E.F=0.66	V.L.E.F=0.65	V.L.E.F=0.32
V.L.E.L=0.09	V.L.E.L=0.2	V.L.E.L=0.06
T.A=0.95	T.A=0.95	T.A=0.95

Remark 16. *In Table 3.11 we vary only the loss function of the main model. The new loss functions are 'mean absolute error' and 'mean squared error' respectively*

Table 3.11: Loss function varied

loss=M.A.E	loss=M.S.E.	loss=S.C.C
T.A.E.F=0.61	T.A.E.F=0.57	T.A.E.F=0.53
T.A.E.L=0.63	T.A.E.L=0.62	T.A.E.L=0.95
T.L.E.F=0.48	T.L.E.F=0.25	T.L.E.F=0.71
T.L.E.L=0.07	T.L.E.L=0.04	T.L.E.L=0.11
V.A.E.F=0.63	V.A.E.F=0.63	V.A.E.F=0.63
V.A.E.L=0.64	V.A.E.L=0.62	V.A.E.L=0.96
V.L.E.F=0.45	V.L.E.F=0.22	V.L.E.F=0.32
V.L.E.L=0.05	V.L.E.L=0.03	V.L.E.L=0.06
T.A=0.61	T.A=0.58	T.A=0.95

Remark 17. *In Table 3.12 we vary only the number of neurons in each hidden layer of the main model. The modified numbers are* 100 *and* 1000 *for each layer respectively.*

Table 3.12: Number of neurons varied

N.N.H.L=$(100, 100)$	N.N.H.L=$(1000, 1000)$	N.N.H.L=$(50, 10)$
T.A.E.F=0.7	T.A.E.F=0.64	T.A.E.F=0.53
T.A.E.L=0.95	T.A.E.L=0.95	T.A.E.L=0.95
T.L.E.F=0.58	T.L.E.F=0.83	T.L.E.F=0.71
T.L.E.L=0.12	T.L.E.L=0.12	T.L.E.L=0.11
V.A.E.F=0.83	V.A.E.F=0.82	V.A.E.F=0.63
V.A.E.L=0.96	V.A.E.L=0.96	V.A.E.L=0.96
V.L.E.F=0.52	V.L.E.F=0.44	V.L.E.F= 0.32
V.L.E.L=0.12	V.L.E.L=0.11	V.L.E.L=0.06
T.A.=0.95	T.A=0.96	T.A=0.95

The tuple $(50, 10)$ *means that we have* 50 *neurons in the first layer and* 10 *neurons in the second layer, and for the other layers this notation is similar.*

Remark 18. *In Table 3.13, we vary the number of hidden layers and the number of neurons in each layer of the main model. The modified numbers for hidden layers are* 4 *and* 7 *respectively. Taking the modified numbers for hidden layers into consideration, each hidden layer has* 50 *neurons.*

Table 3.13: Number of hidden layers varied

n.h.l=4	n.h.l=7	n.h.l=2
T.A.E.F=0.61	T.A.E.F=0.61	T.A.E.F=0.53
T.A.E.L=0.95	T.A.E.L=0.95	T.A.E.L=0.95
T.L.E.F=0.68	T.L.E.F=0.67	T.L.E.F=0.71
T.L.E.L=0.13	T.L.E.L=0.14	T.L.E.L=0.11
V.A.E.F=0.63	V.A.E.F=0.63	V.A.E.F=0.63
V.A.E.L=0.96	V.A.E.L=0.95	V.A.E.L=0.96
V.L.E.F=0.66	V.L.E.F=0.66	V.L.E.F=0.32
V.L.E.L=0.12	V.L.E.L=0.13	V.L.E.L=0.06
T.A=0.95	T.A=0.94	T.A=0.95

3.5 Discussion of the results of main tables

The aim of the tables above is to emphasize the importance of hyperparameters, like loss function, activation function, batch size, number of epochs, number of layers and neurons to find the best model. Table 3.7 is the model, which was obtained after many optimization trials. Not only all the parameters but also the learning curves behave well for that model. This is the main model of our code, and the learning curves and confusion matrix for it are given below as Figures 3.7 and 3.6 respectively. With regards to Table 3.8, only the activation functions are different and they are *relu*, which is a linear activation function, and *tanh* respectively. Making the model linear lowers the performance. It causes underfitting with high constant validation loss and an increase in training loss. However, the *tanh* activation function would be a better choice if the output were restricted to an interval like $[-1, 1]$, but for this model it does not work because the range of output is narrower. There occurs an underfitting problem when the *tanh* and *relu* activation functions are used. The most logical activation function for this model is already *sigmoid* because it is not only nonlinear but also it restricts the output in the interval $[0, 1]$, which is a reliable way to minimize the error, considering that the desired outputs for the current model are 0 and 1. Regarding Table 3.9, we change the batch size to a very large number such as 6000 and all the data pass through the neural network in two batches. Making the batch size too large affects the learning performance of the neural network negatively. Lower training accuracies and testing accuracies are obtained, which means the neural network has difficulties in learning with training data. This fact is a sign of underfitting. Regarding Table 3.10, the number of epochs increases too much and this causes the model to overfit. Although it may be difficult to understand this through only the information given in Table 3.10, if the learning curves of Figure 3.8 are examined, it can be clearly seen that after some time the validation loss increases and a gap occurs between training and validation loss, so this is very good sign of overfitting of the model after a number epochs. Regarding Figure3.9 Table 3.10, it can be concluded that the overfitting problems of the network start after 1000 epochs. The neural network works well for 1000 epochs because the validation loss and training loss are very close at the end of 1000 epoch, so it can be concluded that for 1000 epochs there exists a neural network that neither overfits nor underfits. Regarding Table 3.11, the loss function is

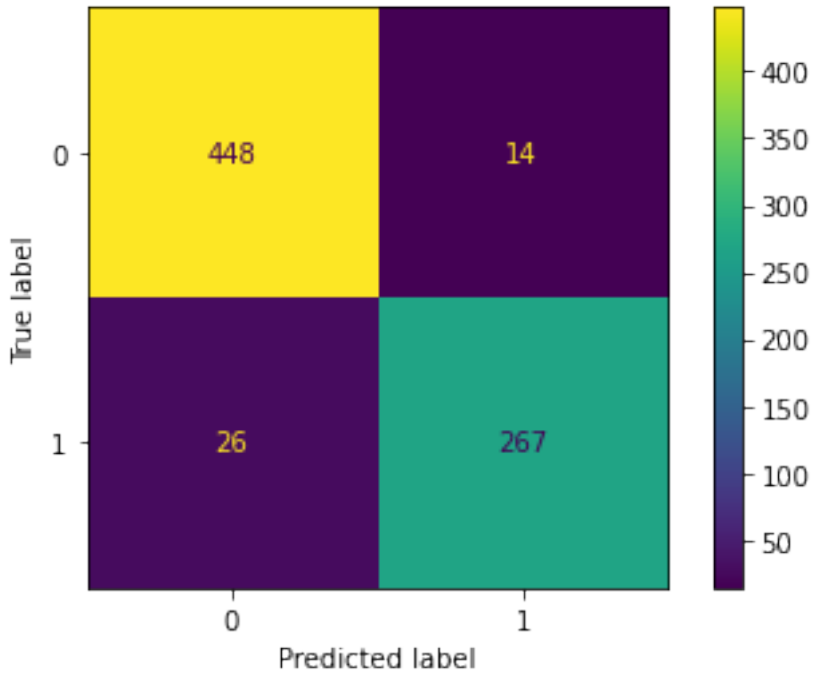


Figure 3.6: Confusion matrix of the main model for h^{12}

changed to *mean squared error loss* and *mean absolute error loss* respectively. As a result of this, the model performs insufficiently because very low training accuracies at the end of the last epochs are obtained, which is a sign of underfitting. The model can not even learn the training data with those loss functions. The reason is that *mean squared error* and *mean absolute error* are loss functions that are more convenient for regression problems rather than classification problems. Lastly, the number of hidden layers and neuron numbers are changed in Tables 3.12 and 3.13. Too many neurons and layers increase the complexity and lower the performance of the model. For example, we obtain a higher validation loss as a result of increasing neuron numbers in hidden layers excessively in Table 3.12 compared to the main model in Table 3.7. Also, we obtain higher training and validation losses as a result of increasing number of layers excessively as in Table 3.13 compared to the main model in Table 3.7.

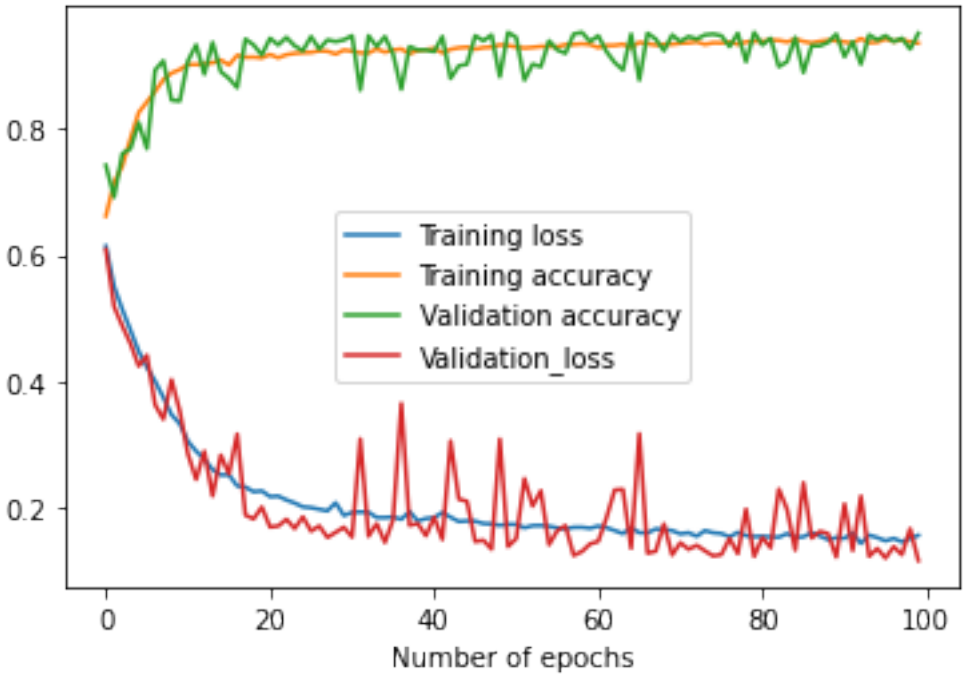


Figure 3.7: Learning curves for the main model h^{12}

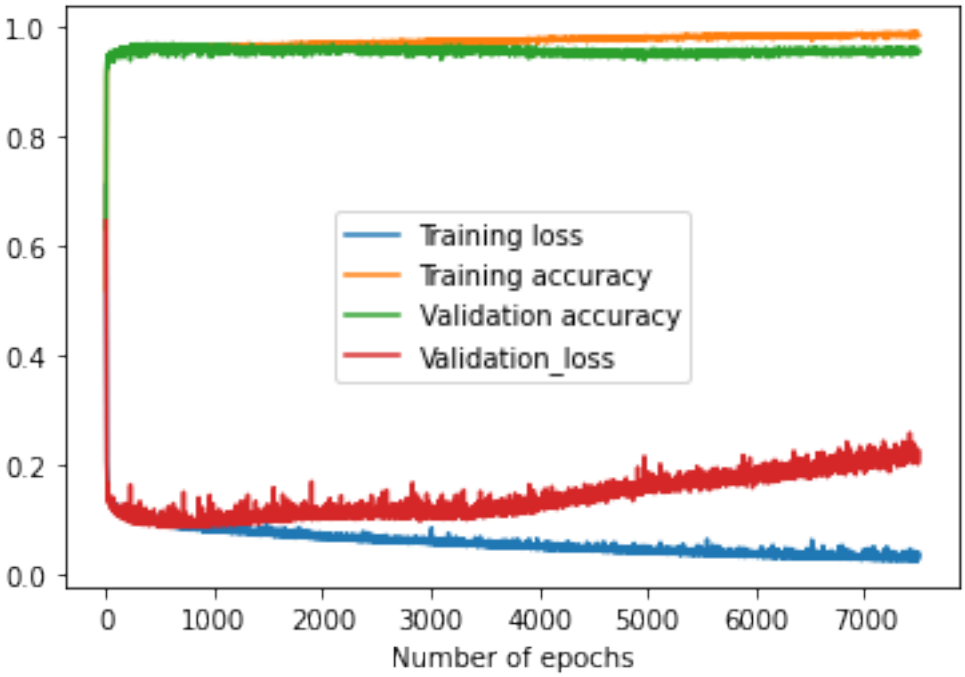


Figure 3.8: Learning curves for h^{12} for 7500 epochs

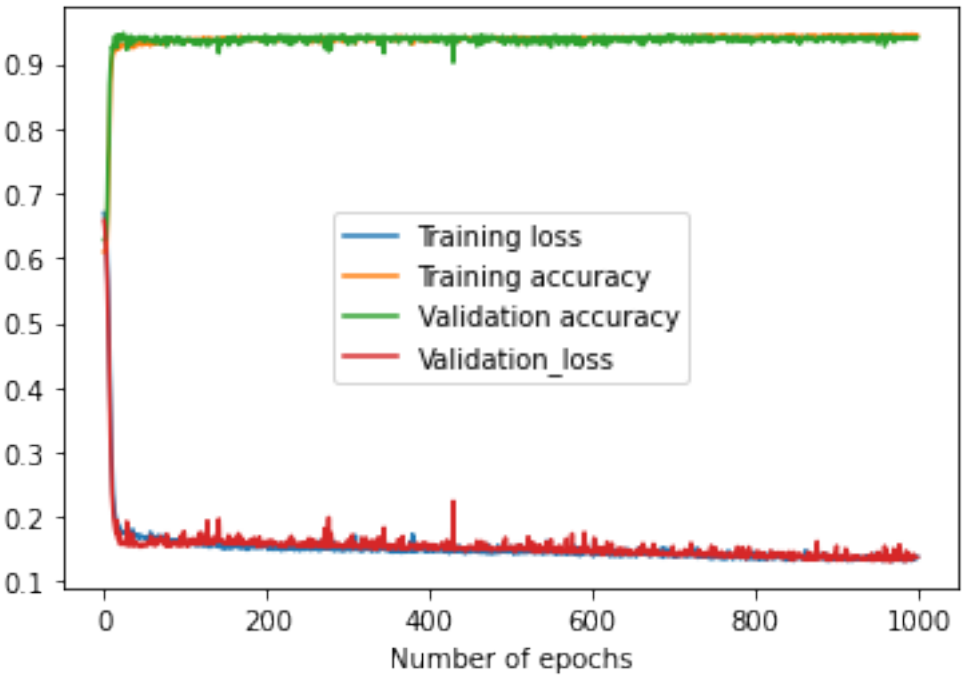


Figure 3.9: Learning curves for h^{12}, for 1000 epochs.

REFERENCES

[1] R. Altman, J. Carifio, J. Halverson, and B. D. Estimating Calabi-Yau hypersurface and triangulation counts with equation learners. *J. High Energy Phys.*, (3):186, 2019.

[2] S. Asiri. Machine learning classifiers. https://towardsdatascience.com/machine-learning-classifiers-a5cc4e1b0623, 2018. Accessed=2022-05-30.

[3] J. Brownlee. Supervised and unsupervised machine learning algorithms. https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/, 2016. Accessed=2022-05-30.

[4] J. Brownlee. How to use learning curves to diagnose machine learning model performance. https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/, 2019. Accessed:2022-05-22.

[5] J. Brownlee. Failure of classification accuracy for imbalanced class distributions. https://machinelearningmastery.com/failure-of-accuracy-for-imbalanced-class-distributions/, 2020. Accessed:2022-05-22.

[6] F. Chollet. *Deep Learning with Python*. Manning Publications, 2018.

[7] F. Chollet. The functional api. https://keras.io/guides/functional_api/, 2019. Accessed=2022-05-30.

[8] F. Chollet. The sequential model. https://keras.io/guides/sequential_model/, 2020. Accessed 2022-05-30.

[9] D. A. Cox, J. Little, and D. O'Shea. *Ideals,Varieties, and Algorithms*. SPRINGER, 2015.

[10] M. Ferrari. Complex manifolds. lecture notes based on the course by lambertus van geemen, 2013.

[11] O. Forster. *Lectures on Riemann Surfaces*. SPRINGER, 1981.

[12] A. Hatcher. *Algebraic Topology*. Cambridge University Press, 2002.

[13] Y.-H. He. *The Calabi-Yau landscape: from Geometry, to Physics, to Machine Learning*. Springer, 2020.

[14] Y.-H. He, E. Hirst, and T. Peterken. Machine-learning dessins d'enfants: explorations via modular and Seiberg–Witten curves. *J. Phys. A*, 54(7):075401, 2021.

[15] D. Huybrechts. *Complex Geometry. An Introduction*. Springer, 2005.

[16] J. Kollár. Algebraic hypersurfaces. *Bull. Amer. Math. Soc. (N.S.)*, 56(4):543–568, 2019.

[17] S. Kılıçarslan, K. Adem, and M. Çelik. An overview of the activation functions used in deep learning algorithms. *Journal of New Results in Science*, 10(3):75–88, 2021.

[18] J. M. Lee. *Introduction to Smooth Manifolds*. SPRINGER, 2012.

[19] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bull. Math. Biophys.*, 5:115–133, 1943.

[20] M. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015.

[21] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

[22] J. J. Rotman. *An Introduction to Algebraic Topology*. SPRINGER, 1998.

[23] J. J. Rotman. *An Introduction to Homological Algebra*. SPRINGER, 2009.

[24] F. Ruehle. Evolving neural networks with genetic algorithms to study the string landscape. *J. High Energy Phys.*, (8):038, front matter+19, 2017.

[25] U. Schreiber. Betti number. https://ncatlab.org/nlab/show/Betti+number, 2022. Accessed:2022-05-29.

[26] P. Shivaprasad. Understanding confusion matrix, precision-recall, and f1-score. https://towardsdatascience.com/understanding-confusion-matrix-precision-recall-and-f1-score-8061c9270011, 2020. Accessed:2022-05-22.

[27] Y. Sun, A. K. Wong, and M. S. Kamel. Classification of imbalanced data:a review. *International Journal of Pattern Recognition and Artificial Intelligence*, 23(4):687–719, 2009.

[28] S. Swaminathan. Linear regression — detailed view. https://towardsdatascience.com/linear-regression-detailed-view-ea73175f6e86, 2018. Accessed=2022-05-30.

[29] C. Voisin. *Hodge Theory and Complex Algebraic Geometry,*1. CAMBRIDGE UNIVERSITY PRESS, 2002.