COMPARISON OF DEEP LEARNING ALGORITHMS FOR SIGN LANGUAGE RECOGNITION

A THESIS SUBMITTED TO

THE FACULTY OF ARCHITECTURE AND ENGINEERING

OF

EPOKA UNIVERSITY

BY

LEJDI LOCI

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR

THE DEGREE OF MASTER OF SCIENCE

IN

COMPUTER ENGINEERING

JUNE, 2024

Approval sheet of the Thesis

This is to certify that we have read this thesis entitled **“Comparison of Deep Learning Algorithms for Sign Language Recognition”** and that in our opinion it is fully adequate, in scope and quality, as a thesis forthe degree of Master of Science.

____

Assoc.Prof.Dr. Arban Uka
Head of Department
Date: June 28, 2024

Examining Committee Members:

Assoc. Prof. Dr. Dimitrios Karras	(Computer Engineering)	________________
Prof. Dr. Bekir Karlık	(Computer Engineering)	________________
Dr. Florenc Skuka	(Computer Engineering)	________________

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: Lejdi Loci

Signature: _______________

ABSTRACT

COMPARISON OF DEEP LEARNING ALGORITHMS FOR SIGN LANGUAGE RECOGNITION

Loci, Lejdi

M.Sc., Department of Computer Engineering

Supervisor: Prof.Dr. Bekir Karlık

Communication has an essential impact in facilitating interaction between individuals. It is a crucial and fundamental way of expressing feelings, thoughts, and opinions. The community of deaf people relies on visual communication of information which uses sign language and speechreading. The significant application of sign language is now a vital part of the hearing-impaired culture. Sign language recognition systems implement machine learning techniques to convey the hand pattern movement into an understandable message.

This thesis aims to make a comparative study between two deep learning models, more specifically, the Convolutional Neural Network (CNN) architecture used as feature extractor and classifier and the hybrid model CNN – Support Vector Machine (SVM), which uses the CNN model as feature extractor and SVM algorithm for the classification process. The paper is divided into two parts, the first one lays into a comprehensive study of both models' development, architecture, and design. The second part is about the practical comparison of methods using coding to observe their performance.

The methodology used to conduct this study is a combination of literature review and practical application of two used models in data classification and prediction tasks. The techniques used for this project include both the qualitative approach which is used in the first section and the quantitative approach employed in the other section.

The thesis dives deeply into the architecture of the models to ensure that each model will perform at maximum capacity so the comparison will be held under the same environment and restrictions. A real-world dataset is taken under consideration to validate the performance of each of the used models.

In conclusion, we emphasize the importance of using machine-learning techniques to enhance the interaction of deaf people within society, as well as the efficiency of the model that may be applied for other data classification and prediction tasks.

***Keywords**: sign language recognition system, feature extractor, classifier, deep learning, CNN model, CNN-SVM hybrid model*

ABSTRAKT

KRAHASIMI I ALGORITMEVE TË DEEP LEARNING PËR NJOHJEN E GJUHËS SË SHENJËVE

Loci, Lejdi

Master Shkencor, Departamenti i Inxhinierise Kompjuterike

Udhëheqësi: Prof.Dr. Bekir Karlık

Komunikimi ka një ndikim thelbësor në lehtësimin e ndërveprimit ndërmjet individëve. Është një mënyrë thelbësore dhe themelore për të shprehur ndjenjat, mendimet dhe opinionet. Komuniteti i njerëzve të shurdhër mbështetet në komunikimin vizual të informacionit që përdor gjuhën e shenjave dhe leximin e të folurit. Zbatimi i konsiderueshëm i gjuhës së shenjave është tani një pjesë jetike e kulturës me dëmtim të dëgjimit. Sistemet e njohjes së gjuhës së shenjave zbatojnë teknika të mësimit të makinës për të përcjellë lëvizjen e modelit të dorës në një mesazh të kuptueshëm.

Kjo tezë synon të bëjë një studim krahasues midis dy modeleve të të mësuarit të thellë, më konkretisht, arkitekturës së Rrjetit Neural Konvolucionist (CNN) të përdorur si nxjerrës dhe klasifikues i veçorive dhe modelit hibrid CNN – Mbështetje Vector Machine (SVM), i cili përdor modelin CNN si nxjerrës i veçorive dhe algoritmi SVM për procesin e klasifikimit. Punimi është i ndarë në dy pjesë, e para përfshin një studim gjithëpërfshirës të zhvillimit, arkitekturës dhe dizajnit të të dy modeleve. Pjesa e dytë ka të bëjë me krahasimin praktik të metodave që përdorin kodimin për të vëzhguar performancën e tyre.

Metodologjia e përdorur për të kryer këtë studim është një kombinim i rishikimit të literaturës dhe zbatimit praktik të dy modeleve të përdorura në detyrat e klasifikimit dhe parashikimit të të dhënave. Teknikat e përdorura për këtë projekt përfshijnë si qasjen cilësore që përdoret në seksionin e parë ashtu edhe atë sasiore të përdorur në seksionin tjetër.

Teza perqendrohet në arkitekturën e modeleve për të siguruar që secili model të performojë me kapacitetin maksimal, kështu që krahasimi do të mbahet nën të njëjtin mjedis dhe kufizime. Një grup të dhënash të botës reale është marrë në konsideratë për të vërtetuar performancën e secilit prej modeleve të përdorura.

Si përfundim, ne theksojmë rëndësinë e përdorimit të teknikave të mësimit të makinerive për të rritur ndërveprimin e njerëzve të shurdhër brenda shoqërisë, si dhe efikasitetin e modelit që mund të zbatohet për detyra të tjera të klasifikimit dhe parashikimit të të dhënave.

Fjalë kyçe: *sistem i njohjes së gjuhës së shenjave, feature extraction, klasifikues, deep learning, modeli CNN, model hibrid CNN-SVM*

ACKNOWLEDGEMENTS

I want to express my gratitude to Prof.Dr. Bekir Karlik, my supervisor, for his support. It was a pleasure for me to have the opportunity to work on my thesis, with such a dedicated and understandable professor like him.

Grateful!

TABLE OF CONTENT

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

In this chapter, a comprehensive description will be provided regarding the primary focus of the study, namely, sign language, and the significance of its exploration. The objectives to be attained by the study's conclusion will be enumerated. Furthermore, the chapter will focus on how machine learning techniques have a pivotal role in advancing sign language detection systems, thereby enhancing communication accessibility for individuals with hearing disabilities.

1.1 Problem Statement

Communication is the key component that enables people to share thoughts, beliefs, and perspectives in their day-to-day lives. However, a significant portion of the world's population suffers from a disability to directly communicate a message to its community. Studies indicate that there are people who face hearing loss and others who are orally incapacitated. Cases, when both these anomalies occur, are numerous and are called deaf-muteness, a disability that verbally isolates these people from the rest of the world. But with today's advancements, the development of machine learning and artificial intelligence has managed to break down this barrier.

According to the report of the World Health Organization (WHO), over 5% of the world's population, or 430 million (83%) people are affected by hearing loss. In line with this estimation, 34 million (17%) are children under the age of sixteen.[21] Some statistical reports estimate that by the year 2050 over 700 million people (1 in every 10 people) will require hearing rehabilitation. Records indicate that 80% of people with this disorder live in low-middle-income countries. In this rapid growth of deaf-mute individuals, it is vital to come up with solutions to help this community communicate with others.

Sign language is what deaf-mute people use to bridge the communication gap with the rest of the world. It aims to create a communication system that uses hand gestures and movements to express spoken words. Following the World Federation of Deaf (WFD) there are more than 300 sign languages used by these communities. There are two approaches for sign language recognition, vison-based systems and wearable sensing mechanisms (sensory gloves). Vision-based techniques for sign detection include machine learning, feature extraction, and pattern recognition which are used to analyze the images and recognize the signs. Meanwhile, sensory gloves are equipped with sensors to detect hand movement and translate them as signals to the computer.

Integrating machine learning techniques and models into systems that detect sign language holds a significant social impact for several reasons.

By enhancing the accuracy and efficiency of sign language detection systems through machine learning, deaf and hard-of-hearing individuals gain improved access to communication platforms. This facilitates their participation in various social, educational, and professional contexts, reducing barriers to interaction and inclusion.

Integrating machine learning into sign language detection systems fosters a more inclusive society by breaking down communication barriers between individuals with different abilities. It promotes understanding, empathy, and respect for diversity, ultimately contributing to a more cohesive and harmonious community.

Better sign language detection systems can improve education for deaf people by helping them get to teaching materials, lessons, talks or even communicate with teachers and others of their kind. Similarly, they promote an inclusive learning atmosphere where every learner can enjoy such resources and opportunities.

1.2 Thesis Objectives

The main goal of the project is to develop two models for sign language recognition and compare the performance of each (CNN and CNN-SVM model) for predicting the American Sign Language.

The thesis resolves around conducting a comparative analysis of two deep learning techniques utilized in sign language recognition. Precisely, the study will focus on the employment of a Convolutional Neural Network where CNN is used both as a feature extractor and classifier, and the hybrid CNN-Support Vector Machine (SVM) model where CNN architecture operates as a feature extractor and SVM as a classifier.

There are also multiple aims and goals, which are queued below:

- **Dataset exploration:**

One of the main focuses was the selection of an appropriate dataset that would be in alignment with the algorithms used and the scope of my thesis.

This paper draws upon two widely recognized sources for beginners in machine learning projects: Kaggle and GitHub. Following thorough research, a dataset was chosen from Kaggle, specifically the MNIST sign language dataset, comprising grayscale images. This dataset includes two distinct subsets for training and testing purposes. While there are numerous image-based datasets available for various languages worldwide, the MNIST sign language dataset was selected due to its simplicity and ease of maintenance.

- **Development of machine learning techniques:**

The field of machine learning continues to evolve rapidly, with advancements in development techniques occurring daily. This work aims to explore the various steps involved in a machine learning project, starting from dataset selection, through data preprocessing, feature extraction, model building and optimization, to training and testing. One of the primary objectives of this thesis is to gain familiarity with these techniques and implement them effectively to augment the models' accuracy and capability.

- **CNN architecture:**

Examine the architecture of CNN, optimizing its multiple-layered structure, and CNN performance as a feature extractor and classifier.

The CNN architecture, comprising essential layers such as convolutional, pooling, and fully connected layers, has been thoroughly investigated regarding both the technical aspects of data training and the classification process into respective classes.

- **CNN-SVM model integration:**

Evaluation and integration of CNN-SVM model, with a specific focus on adopting a new strategy: SVM employed as a classifier, replacing so the fully–connected layer of CNN architecture.

- **Performance evaluation**:

Evaluate and test the performance of each CNN and CNN-SVM model to find out which method has the most potential for improving sign language recognition technology. This incorporates evaluation of metrics such as precision, recall, and F1 score, to identify the efficiency of each algorithm.

By outlining and following these objectives, the thesis attempts to contribute valuable observations and advancements in the sign language recognition field, the motive to enhance communication for communities with hearing disabilities.

1.3 Scope of works

This thesis aims to implement machine-learning approaches to recognize and analyze sign language characters from a predefined dataset. The primary objective of my work is to compare the performance of two unique deep-learning models based on the metrics employed. This survey of the literature will give me a hand in identifying potential gaps in current studies that our work aims to resolve.

The following stage focuses on the acquisition and preparation of the deployed dataset. The dataset will contain characters from American Sign Language, each represented in an image format. To provide consistency and compatibility with the models, the dataset images will go through preprocessing steps including normalization, reshaping, and augmentation.

The next stage includes the design of the Convolutional Neural Network model, where CNN architecture is used simultaneously as a feature extractor and classifier. The focus will be on designing an architecture model where convolutional, pooling, and dense layers will optimize the performance of the algorithm. Furthermore, a hybrid CNN-SVM model will be implemented, where CNN is used as a feature extractor and SVM is employed as a classifier, replacing so the dense layer of CNN.

After training and testing both models with a special focus on metrics and hyperparameters to optimize the validation set, the performance of the used models then will be compared. This comprehensive comparison will be based on our predefined metrics, which include accuracy, precision, recall, and F1-score. Additionally, the performance of the models will be discussed and evaluated.

In the end, after the discussion regarding the results and analyses conducted, we will try to add a section about future work recommendations. This scope will try to suggest promising insights for further investigation and improvements in the area of Sign Language Recognition.

1.4 Organization of the Thesis

This thesis is divided into 5 chapters. Chapter 1 includes the problem statement, thesis objectives, and scope of work. In Chapter 2, a literature review together with a historical development of machine learning & machine learning background is presented. Chapter 3 describes the methodology used to conduct this study. In Chapter 4 we have the architecture and implementation of the models. Chapter 5 includes model testing and performance comparison. The last Chapter 6 presents conclusions and future recommendations for further research.

CHAPTER 2

LITERATURE REVIEW

In this chapter, we will delve deeply into the historical evolution of machine learning (ML), deep learning (DL), and neural networks, exploring their relation. We will also categorize the field of ML into supervised learning, unsupervised learning, and reinforcement learning, determining the category to which our multiclassification problem belongs, given our dataset's structure with several classes. Additionally, we will extensively review relevant literature and similar works to support our findings and enhance existing methodologies with more efficient solutions.

2.1 Historical Development of Machine Learning

Machine Learning (ML) is a subset of Artificial intelligence (AI) that allows software applications to make promising predictions without implementing explicit programming for a specific task.

Machine learning has a wide application in many real-world sectors and fields like healthcare, manufacturing, finance, autonomous vehicles, and Natural Language Processing (NPL), which we are most concerned about in this thesis. ML techniques implemented in NPL have significantly optimized the efficiency and accuracy of the models that are used to recognize and analyze such patterns. Below we will mention some milestones in the development of ML over the years.

In 1957 Frank Rosenblatt was the first to introduce the Perceptron, which is a fundamental concept in Machine Learning and Artificial Neural Network (ANN). He proposed this algorithm based on McCulloch-Pitts (MCP) neurons, to handle complex agorithms and data applied in today's recognition tasks.

Support Vector Machines (SVMs) were initially presented by Corinna Cortes and Vladimir Vapnik in 1995. It was based on a nonparametric method with kernel functions. SVM today is widely used in both linear and nonlinear classification problems by determining boundaries between data points on given datasets.

IBM’s Deep Blue supercomputer defeated the most popular chess champion, GM Garry Kasparov (1997). This was the first time a chess computer was able to defeat a champion player. This victory is considered a milestone in artificial intelligence since it emphasizes the fact that AI overcame the capacity of a human being on a specific task.

Another important achievement was the development of the AlexNet, a convolutional neural network model, that performed remarkable results compared to other models, on the ImageNet Challenge for image classification. This model was represented in 2012 by Alex Krizhevsky and contributed to further research in deep learning and convolutional neural networks.

In 2017 Vaswani et al, a computer scientist proposed the transformer model. This model contributed to the development of natural language processing. A great illustration of a transformer model in today's advancements is ChatGPT which uses a Decoder-Only Transformer.

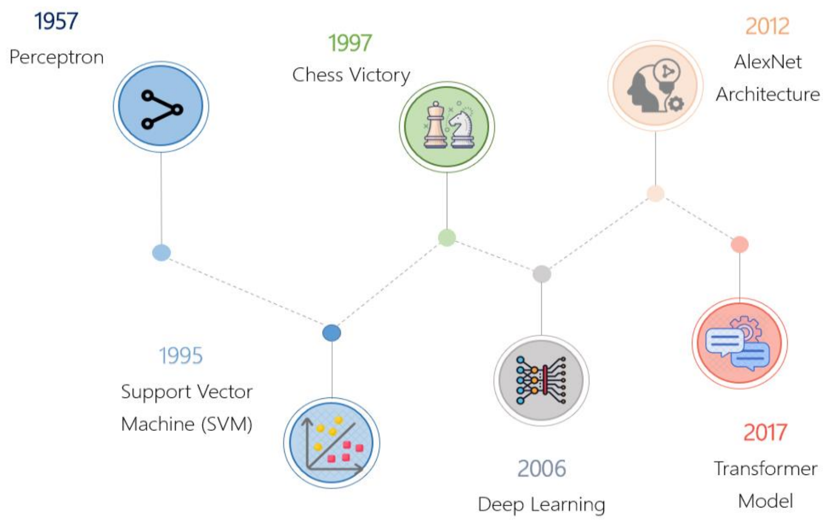


Figure 2.1 - ML development over the years, period 1957-2017

As a constantly growing field, the future of ML remains both promising and challenging for developers, researchers, and society.

2.2 Related Work

In this section, we will consider different paper research and articles published in recent years. Through a critical review of the literature, my goal is to explore the best-performing deep learning models used in sign language recognition. Below I will mention some papers of interest for deeper insights that will help to defend my thesis theory.

A review article based on a comparative study for sign language identification and recognition was published in Open Computer Science 2022. This paper utilizes static and dynamic datasets from different sign languages, to make a comparison of the effectiveness of different machine and deep learning techniques. The main focus is to identify the model that outperforms others in terms of accuracy by optimizing deep learning parameters such as the architecture of the model, number of epochs, classifier used, and so on. There are used different datasets and methods such as CNN, KNN, SVM, and ANN. To recognize Bhutanese Sign Language of numbers from 0-9, the authors used 20000 images of digits from 21 students. Images were captured from different conditions and angles. After all the models were implemented the CNN exceeded each of them by achieving an average accuracy of 97.62%, followed by KNN with 78.95% and SVM with 70.25% accuracy. In this case, SVM wasn't the best option to be used.[10]

In [14], is developed a comparison between end-to-end approaches for sign language recognition. The employed models are CNN, SVM, KNN, multilayered perceptrons (MLP), and linear discriminant analysis (LDA). The utilized dataset is Irish Sign Language (ISL) which contains 50000 images. The authors compose the architecture of CNN with 4 convolutional layers with ReLU non-linearity, 2 fully connected layers, and dropout layers to prevent overfitting, and the Adadelta optimizer is used as a loss function. The MLP is structured with a first layer, one hidden layer with 256 neurons, and an output layer with 23 neurons. The SVM approach uses a polynomial kernel, while KNN is set to k = 1, and for the LDA they used singular value decomposition. After the experimental result, the model that outperformed the other was CNN with an accuracy of 99.8%, followed by SVM at 97.86% and then the rest of the algorithms had lower performance metrics. As a future work, the authors promise to build an automatic transcript system for sign language detection and recognition.

Another paper named "Digit Recognition in Sign Language Based on Convolutional Neural Network and Support Vector Machine" proposed a hybrid model that combines a conventional neural network (CNN) and support vector machine (SVM) for image recognition in sign language. The CNN-SVM model is applied to two sustainable datasets respectively named MU_HandImages_ASL and standard databases-SDL. The paper aims to recognize the digits from American Sign Language (ASL) and the standard Sign Language Digits. In this model, CNN is used as a feature extractor while as a classifier SVM is used. A pre-trained Inception-v3 CNN architecture is utilized for the extraction of features. The initial layers are frozen since they extract only low-level features (edges, corners, and texture), while the remaining layers are then retrained to extract high-level features. Then the SVM uses the radial basis function to recognize the images. The CNN-SVM model reached a recognition accuracy of 98.2% in the ASL dataset and 98.3% in the SLD dataset. These results were superior compared to other CNN models. [9]

"Sign Language Recognition using Neural Networks" is another article dedicated to the discovery and implementation of a sign language recognition system. The proposed model by the authors is divided into three stages. During the first stage, the dataset of images is processed, and then in the next stage, the authors use the masking method to prepare the dataset for training. The final phase then applies the cross-validation method to validate and train the data. The image dataset consists of 90 images, 60 used for training and 30 for testing. The total number of images is prepared with 3 non-identical images using the Bosnian language alphabet. It consists of 15 input layers, 2 hidden layers, and one input layer. The network then is trained using a backpropagation algorithm and then to estimate the quality of the network a 3-fold cross-validation is applied. After the performance of ANN is evaluated the accuracy of the model is calculated to be 84%. In the end, the authors propose that the performance of the algorithm would be better if the images taken into consideration were improved, considering the brightness, contrasts, shadows, and background transparency. [11]

On the overall investigation and insights gained from all the papers considered for this thesis, I managed to choose among the deep learning and hybrid models that have a higher accuracy compared to others. CNN was the main approach which gives you space to experiment with different architecture models and hyperparameters, to optimize the performance of the algorithm, for sign language detection and recognition. Among the used classifiers within the papers, the SVM classifier tends to achieve better results in the classification of the images in their proper classes. A combination of CNN architecture as a feature extractor and SVM as a classifier is a promising hybrid model that uses the elements of each machine-learning technique to optimize the recognition tasks.

2.3 Machine Learning

Based on the characteristics of the data obtained and used we categorized the machine learning into three core types. The three main types are supervised learning, unsupervised learning, and reinforcement learning.

2.3.1 Supervised Learning

In supervised learning the algorithm learns from a dataset that is labeled, meaning that data comes with a prior description. After the termination of the training process, the model is passed to testing and then the output is predicted. The main goal of the model is to find a general rule that matches the input features to output labels.

There are two subtasks of supervised learning: regression and classification. Regression trains and predicts a quantity or continuous data such as predicting a real estate price, while classification attempts to locate data on the appropriate class label such as secure /insecure, spam or not spam email, male/female persons, and so on. We have many supervised learning algorithms which are Neural Networks (NNs), Logistic Regression, Support Vector Machines (SVMs), Linear Discriminant Analysis, and other models that can run on labeled data.

Supervised Learning (SL) comes with both advantages and disadvantages based on the nature of the dataset and the complexity of the model. The strengths of SL include prior knowledge of the category or class that provides information about the dataset used. On the other hand, SL requires a lot of training time and is not so suitable for handling complex tasks. Also, in cases where the test data is so different from the training data the model can not predict the correct output.

2.3.2 Unsupervised Learning

Unsupervised Learning trains on unlabeled data, meaning that we don’t have a mapping between input and corresponding output labels. The main aim is to discover underlying patterns and relationships within the data. The rule is “Learn by doing”, which means that the model aligns itself according to the input signals.

Clustering and Association are two common approaches used in unsupervised learning. Clustering aims to group similar objects into classes or clusters as we recognize them, however, association finds the relationship between data points while determining the set of items that occur in the same dataset. K-means Clustering, Hierarchal Clustering, and Fuzzy Clustering are some of the main algorithms used for this approach.

Unsupervised Learning also comes with strengths and limitations. This model is generally used for more complex tasks, it can identify hidden patterns or relationships within the variables without the need for labeled data. Also, the model can be useful since creating clusters for similar items reduces the dataset dimensionality, making it easier to extract information. On the other hand, unsupervised learning lacks corresponding outputs making it more challenging to evaluate the performance of the algorithm. Following this logic, unsupervised learning is more suitable for classification and data analysis rather than data prediction

2.3.3 Reinforcement Learning

Reinforcement Learning (RL) is a method utilized in machine-learning technique that learns through trial and error by receiving feedback in the form of rewards and penalties. This model uses an agent, that interacts with the environment to achieve an objective. RL is a learning paradigm, where the agent aquires the ability to make choices to maximize the total rewards gained over time.

RL is broadly used in robotics, where robots interact with the environment by learning different complex behaviors and performing tasks such as manipulation and control. Another application of RL is the development of autonomous vehicles, where the vehicles learn to navigate under certain conditions and follow traffic rules. This type of machine learning is also used in other industries including finance, healthcare, education, gaming, and resource management.

2.4 Deep Learning

Deep Learning (DL) is a subset of machine learning that utilizes Artificial Neural Networks (ANNs) with multiple complex layers to represent and interpret the data. Both ML and DL are subsets of Artificial Intelligence (AI) but they differ in terms of the techniques used and applications. Deep learning algorithms are used in a wide range of applications, such as extraction & segmentation, detection & recognition, classification tasks, and so on.

The primary objective of deep learning is to develop models by using techniques that can automatically learn the hierarchical representation of data from raw input avoiding so manual feature engineering. DL algorithms show strong generalization capabilities which means that can achieve satisfying predictions even if the unseen data was not part of the training set. Deep learning techniques can also be adapted and developed to suit specific requirements since they are highly flexible with data handling.

Besides the strong capabilities of deep learning, it also comes with some challenges related to computational complexity, overfitting, and hyperparameter tuning. Deep learning models require high-performance GPUs and TPUs when they deal with a large number of layers and parameters. This leads to additional costs and long time consumption of the model implementation.

Overfitting is an event that occurs when a model demonstrates high proficiency on training data but does not perform well in the validation data. The opposite of overfitting is underfitting, which means that the model performs poorly on a set of data for training and cannot fully learn from the data training. These two approaches are considered to be disadvantages of DL, but there exist different techniques for their adjustment.

Due to their multilayered structure, DL models can be challenging to interpret, which makes it difficult to understand how the model generates predictions to identify incorrect results. Such problems are known as “black-box” models.

2.5 Neural Networks

Neural Networks (NNs) serve as the underlying principle of deep learning and artificial intelligence, inspired by the functioning of the human brain.

Similarly to the cells in our brains analogously, the neurons compose the fundamental framework of a neural network. In this network, a neuron first receives input, then the input is processed and generated as an output.

As the input is received by the neuron it is multiplied by a numerical value known as weight. The weight itself represents the strength of the connection between two units. If the neuron has more than one input then each input will have a corresponding weight assigned to it.

The weights are randomly initialized and are consecutively updated throughout the model training process. After training, neural networks assign higher weights to inputs that are considered more important. A weight of 0 indicates that a given feature is not essential to effect the neuron's progress.

Besides the weights, another linear component known as bias is applied to the input. After adding the bias, the result would be a·W1+ bias. This is the last linear component of the input transformation. After a linear component is applied to the input, a non-linear function, called Activation Function f(a·W1+bias) is used.

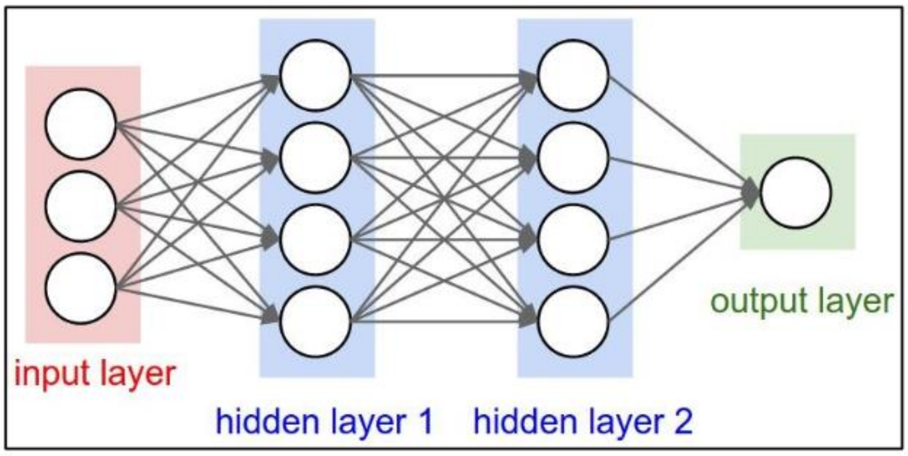


Figure 1.2 **-** Neural Networks Schema

A neural network is composed of layers of neurons. Initially, the input layer receives the input while the output layer generates the final output. The hidden layer, located between these two, performs the majority of the computations on the network. Neurons of one layer are connected to neurons of the other layer through channels. There can be more than one hidden layer.

Once the output is obtained in a single iteration, we compute the network error. The error is fed back to the network together with the gradient of the cost function (it measures how well an ML model performs by calculating the difference between the predicted output and the actual output) to update the weights. Following up the updated weights, the errors in the following iterations will be reduced.

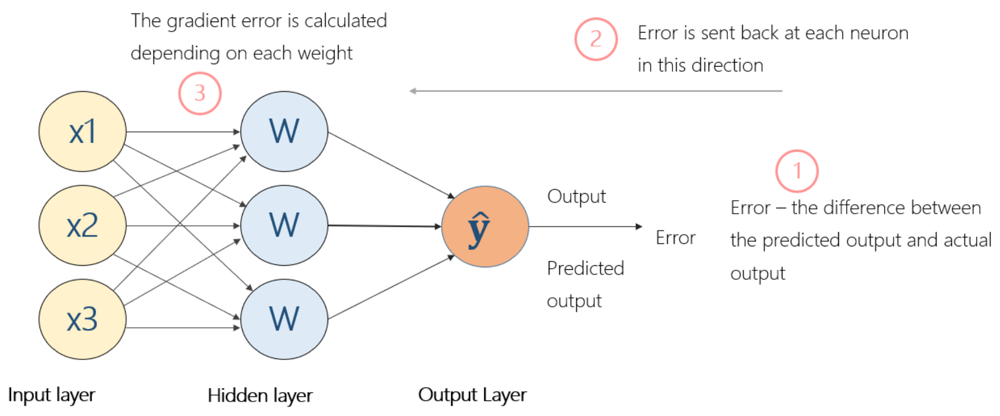

The gradient error is calculated depending on each weight
3
2
Error is sent back at each neuron in this direction
x1
x2
x3
W
W
W
ŷ
Output
Error
Predicted output
1
Error – the difference between the predicted output and actual output
Input layer
Hidden layer
Output Layer

Figure 2.3 **-** Graphical presentation of Backpropagation

This process of adjusting the weights while utilizing the gradient of the cost function is known as back-propagation.

When training a model, instead of sending all the input at once, we divide it into several equally sized parts. Training data in batches leads to a more structured model compared to the model built when the entire dataset is fed into the network simultaneously.

CHAPTER 3

METHODOLOGY

3.1 Data Collection

In this chapter, we will work step by step to solve the concrete problem which is classifying images with corresponding labels (letters). The focus will be on the preparation of the work environment, dataset selection, and data preprocessing to adapt them to a comprehensible format for the models.

Since the study aims to compare the performance of deep learning algorithms (CNN and hybrid CNN-SVM) we will deploy a quantitative approach. This means that the interpretation of the results we be made based on accuracy, precision, recall, F1 score, and other relevant used metrics.

3.1.1 Work Environment

The work environment where we have applied our project is Jupyter Notebook. Jupyter Notebook is an interactive computing environment that allows the creation and sharing of documents that include live code and visualizations. The advantage of Jupyter when processing a large set of data is that it allows us to divide the code into cells. These cells can be executed independently reducing so the execution time and making the coding process easier. Jupyter Notebook runs locally on our machine, meaning that it uses its computing resources. When we start the Jupyter Notebook server, it runs as a local web application on our computer. We can access it by determining the address http://localhost:8888/tree where the localhost represents the local address of our machine and 8888 is the number of the default gate of the server.

Taking into consideration the fact that Jupyter Notebook IDE is based on the computer capacity of our machine, some of its parameters are listed below:

Processor	11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz 2.42GHz

RAM	8.00 GB (7.77 GB usable)
OS	Microsoft Windows 11 Home, 64-bit operating system, x64-based processor

The programming language used in this paper is Python, as one of the most popular languages used in machine learning, due to its simplicity, readability, and the splendid support it provides for both beginner and experienced programmers. Python has a variety of libraries, which facilitate working with this language. Some of the main libraries that the language provides and are used in this task are:

TensorFlow is an end-to-end open-source machine learning framework developed to create and train deep learning models. With support for imperative and symbolic programming, it offers an adaptable and scalable neural network platform.

NumPy is one of the main libraries used for numerical operations which provides a multi-dimensional array. NumPy arrays are homogeneous, which means that every element has to be the same data type.

Pandas is a fundamental library for cleaning, manipulating, and analyzing data. This library provides a concept called "DataFrame". A DataFrame is a two-dimensional data structure similar to an Excel spreadsheet, organized in rows and columns.

MatPlotLib is a library that creates static, animated, and dynamic visualizations.

Seaborn library provides a high-level interface for creating statistical graphs.

Scikit–Learn is a widely used library for machine learning tasks such as classification and regression. This library provides algorithms for data processing, model selection, and evaluation.

Keras is a library designed primarily for building and training models of deep learning. It serves as a front-end API that allows users to define and configure neural networks, in a few lines of written code.

3.1.2 Dataset Selection

Kaggle is the most used data science community, that allows users to determine resources and utilities, to build and implement different ML models. The dataset employed in our work is sourced from Kaggle.com.

The original MNIST image dataset (a large database, containing handwritten numbers, used for training various images in processing systems) is a well-known reference point for image-based machine learning methods. The scientists constantly worked to update, develop, and improve sets of data that are more complex for the computer vision field and original for applications in the real world. The sign language MNIST used in this paper follows the same CSV format with labels and pixel values in single lines (rows).

More than 203 datasets, which are used in sign language, are available on the Kaggle platform. These datasets include both characters (letters) and numbers. The letters representing these data may be part of the alphabet of a particular country.

American Sign Language letters, is the dataset obtained for this thesis. It represents a multi-class problem (to classify instances into 1 of 3 or more classes) with 24 letter classes (excluding J and Z as they are characters that require movement as shown in Fig 4.

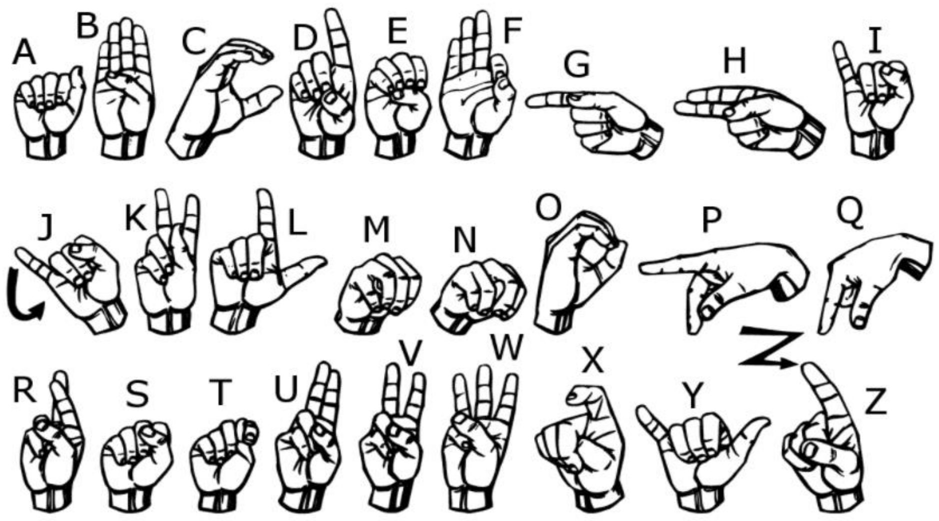


Figure 3.1 - American Sign Language, including J and Z

Among all the available datasets on Kaggle and GitHub, the MNIST sign language dataset was selected due to the reasons outlined below:

- It uses grayscale images thus reducing the complexity of the dataset compared to images in RGB format. Grayscale images have only a single channel (meaning a pixel has only one ranked representative numerical value from 0 (black) to 255(white)) representing the intensity of each pixel, meanwhile, color images (RGB images) have 3 channels which include red, green and blue. In sign language recognition, the primary focus is on capturing shapes and hand gesture structure rather than color information. The grayscale images effectively capture the changes in brightness and contrast, which are crucial for distinguishing between different sign gestures.

- The obtained grayscale images tend to avoid overfitting since they have fewer channels compared to RGB images, lowering the possibility that the model may retain noise or unimportant information from the data.
- The selected dataset contains a large amount of data, which is pre-classified into training and testing. As a result, the accuracy and performance of the algorithm increase while using more data to train the model and make predictions.
- This dataset provides a wide community of users who have built models similar for sign language recognition. These have served as reference points to enhance the design of our used models.

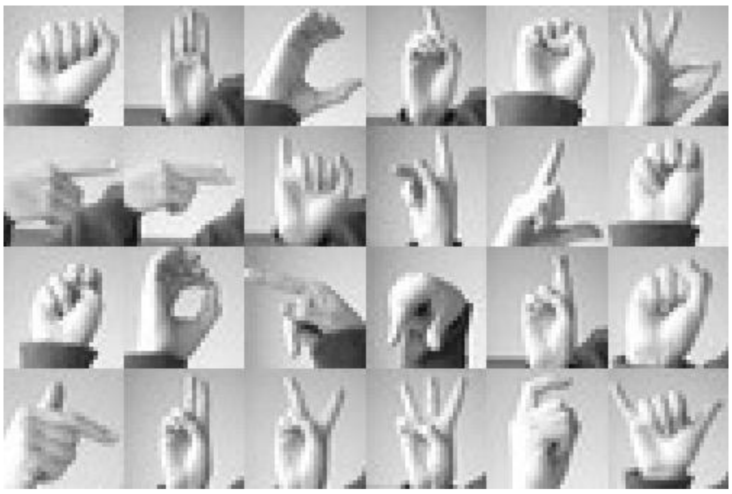


Figure 3.2 - Grayscale image dataset

Every label (0–25) in the training and test data corresponds to a single letter A–Y (0 being A, 1 to B, and so forth). Meanwhile, the letters J(9) and Z(25) are excluded since they require motion meaning they are not static actions.

The standard MNIST is almost half the size of the test data (7172 instances) and training data (27455 instances). Each line with one data contains the label (0-25) and pixel1, pixel2,pixel3, ..., pixel 784, which stands for a single grayscale 28 by 28 (28x28) pixel image with values ranging from 0-255. To fit this dataset, it was used ImageMagick (free, open-source software for adjusting and manipulating images digital).

ImageMagick tools are utilized for cropping the images (to include only the hands), grayscale, resizing, and creating at least 50+ variations to increase the quantity. Different filters such as ('Mitchell', 'Robidoux', 'Catrom', 'Spline', ‘Hermite') were used in the extension and modification method, coupled with 5% random pixelation, +/- 15% brightness/contrast, and then rotation by 3 degrees.

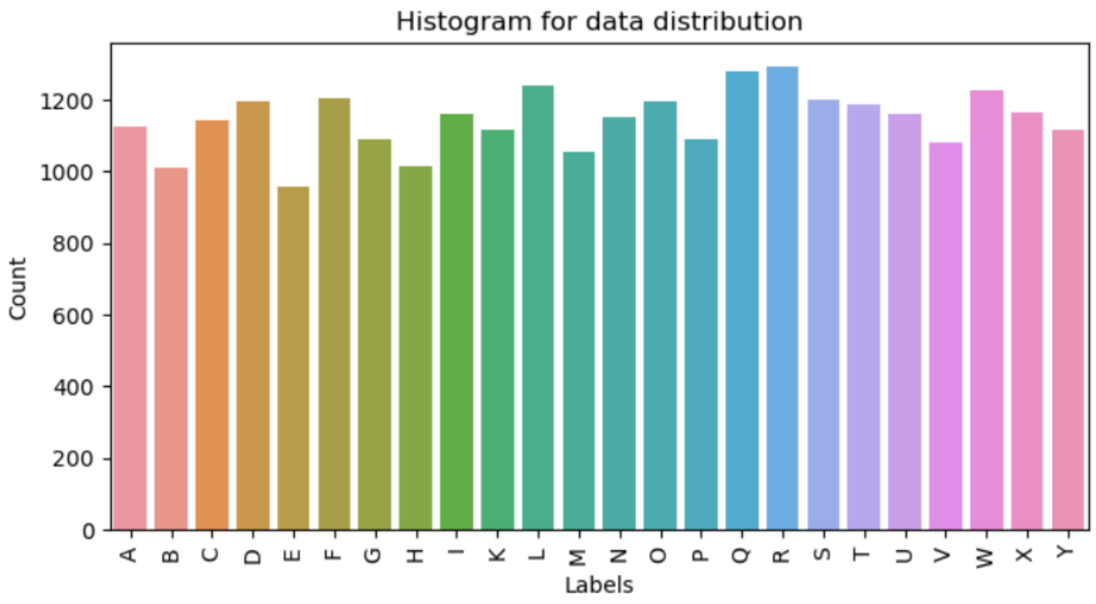


Figure 3.3 - Histogram for data distribution

From the chart above we notice that our dataset is generally balanced. A balanced dataset is essential because it prevents biased results in data analysis and machine learning models. On the other hand, an imbalanced dataset means that a certain class or category overwhelms the others, causing the model to perform poorly on the minority categories. There are several techniques we can handle imbalanced data, such as under-sampling majority class, oversampling minority class by duplication, oversampling minority class by SMOTE (generates synthetic examples using k nearest neighbors algo), ensemble method, and so on.

Also, a balanced dataset ensures that each class has a sufficient number of records, allowing the model to learn and generalize well across the classes. In this way, it enhances the model’s ability to make predictions and classifications with a high degree of accuracy.

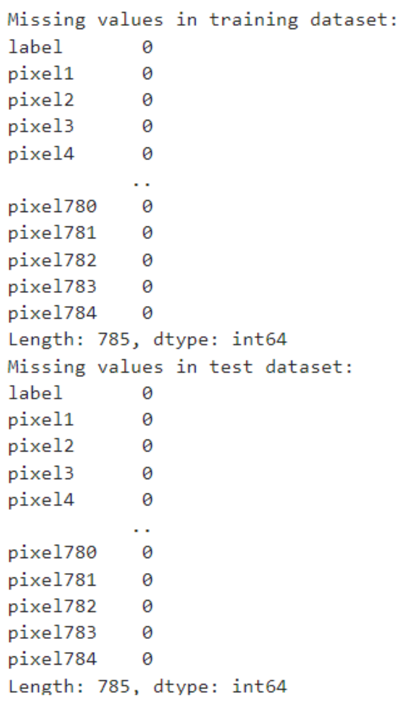
3.2 Data Preprocessing

Following the dataset selection, an important step applied in ML projects is data preprocessing. In this section, we will dive into some crucial steps to get our data ready for analysis and other implementation tasks. We'll go over the essential methods for cleaning, organizing, and improving our dataset, including handling missing values, scaling, encoding, normalization, and data augmentation.

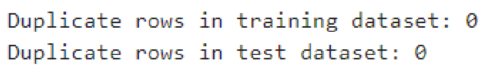
3.2.1 Handling Missing Values

It happens that there are missing values in the datasets, or even instances that are not in the right format to be used by ML models. Since we can not proceed within these conditions, we use some techniques that "clean" the data and make it suitable for these models, increasing thus the accuracy and efficiency of a model in ML.

It is common practice to check for data that are irrelevant, incorrect, duplicate, incomplete, or null values and find ways to handle them. By identifying these issues, the values must be modified or deleted as required by the model.



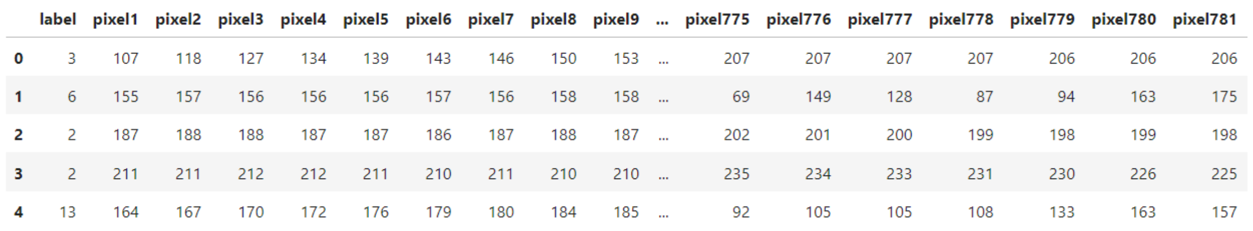
```
Missing values in training dataset:
label       0
pixel1      0
pixel2      0
pixel3      0
pixel4      0
           ..
pixel780    0
pixel781    0
pixel782    0
pixel783    0
pixel784    0
Length: 785, dtype: int64
Missing values in test dataset:
label       0
pixel1      0
pixel2      0
pixel3      0
pixel4      0
           ..
pixel780    0
pixel781    0
pixel782    0
pixel783    0
pixel784    0
Length: 785, dtype: int64
```



```
Duplicate rows in training dataset: 0
Duplicate rows in test dataset: 0
```

It is visible that our set of data does not have any missing or duplicated values in both training and testing datasets. Each row in the dataset contains a value and this value is unique, so extensive preprocessing steps will not be conducted since the data recording is all clean and complete at this point.

Below is a visual representation of the structure and content of the utilized dataset.



	label	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel775	pixel776	pixel777	pixel778	pixel779	pixel780	pixel781
0	3	107	118	127	134	139	143	146	150	153	...	207	207	207	207	206	206	206
1	6	155	157	156	156	156	157	156	158	158	...	69	149	128	87	94	163	175
2	2	187	188	188	187	187	186	187	188	187	...	202	201	200	199	198	199	198
3	2	211	211	212	212	211	210	211	210	210	...	235	234	233	231	230	226	225
4	13	164	167	170	172	176	179	180	184	185	...	92	105	105	108	133	163	157

Figure 3.4 - Dataset structure representation

3.2.2 Label Extraction

For this phase, we extract the target variable (columns that belong to labels) from the columns that belong to the input features which in our case are represented by images. Now the dataset is separated into two parts labels and input features.

```
del train_df['label']

del test_df['label']
```

During the training phase, this separation enables the model to discover the relationship between the images and the associated labels.

This step is carried out for the following reasons:

- Labels serve as target variables, which the model aims to predict based on the input data. By separating the labels, a clear distinction is made between the input data and the desired output data. This allows the model to learn to map between them.
- This separation ensures that the model makes predictions on unknown instances based on the learned relationship between input features and labels.

3.2.3 One-hot Encoding

One-hot encoding is a technique that is used to convert categorical data (categorical data – consists of discrete categories or labels that represent groups, different classes, or attributes) into binary vector representations or a numerical format, where each category is represented by a binary value (0 or 1).

In our case, one-hot encoding is applied to the labels (y_train & y_test) to transform them into a format suitable for the model training process. This transformation allows the model to treat each category independently and avoid serial (ordinal) relationships between them.

The importance of one-hot encoding in sign language recognition is essential if we want to increase the efficiency and accuracy of the model performance. Below we will mention some reasons for using this technique in our pre-processing steps:

- When labels are presented with numerical values as in our case (0-25), this leaves space in the model for misinterpretation. For example, if 0 is “A”, 1 is “B”, 2 is “C” and so on, it is possible that the model assumes that the characters (letters) have an ordinal relationship. In this scenario, the model detects labels 1(A) and 4 (E) and incorrectly concludes that label 4 is more important than label 1, leading to incorrect interpretations, meanwhile in reality the dataset has categorical variables with no sequence among the letters.
- The use of numeric labels without one-hot encoding can lead to the adjustment of bias (a type of error, that assigns more weight to some data points compared to other instances) or arbitrary weights during the model training process, based on the numerical values of these labels.
- "One-hot encoding" technique is especially necessary when using a certain loss function
  (method that evaluates the performance of a model, the more incorrect predictions the model makes, the loss function will generate a large number of errors or inaccuracies). In our task, we have used the "categorical cross-entropy" function, which we will talk about further. These functions expect labels to be in the one-hot encoded format, where each class has its binary column.

How does one-hot encoding work for our model?

In our dataset, we have 26 classes (letters from A-Z). We build a binary vector (a collection of 0s and 1s) with the same length as the number of classes (26). For each binary vector, we assign the value 1 to the letter corresponding to the class and 0 to other instances. Below is a representation of the vectors:

A [1 0 0 0 0 … 0 0 0]

B [0 1 0 0 0 … 0 0 0]

C [0 0 1 0 0 … 0 0 0] …

The conversion of (y_train & y_test) labels into one–hot encoded vector is done by using the LabelBinarizer class from scikit -learn.

3.2.4 Normalization

Normalization is a data preprocessing technique that transforms values and numerical input features with a similar scalability range, usually between 0 and 1 (as in our case) or -1 and 1. The purposes of using normalization in our data inputs are:

- In our dataset, the image's pixel values range from 0 to 255. Scaling these values between 0 and 1, brings the data into an approximate range, thus preventing certain parts (such as color intensity) from overwhelming the learning process of the model.
- Normalization ensures fast convergence during the training phase. The model convergence toward the most optimal solution is particularly beneficial in deep model learning such as CNN for image classification.
- Normalization acts as a regulator by determining limits on the size of input features. This prevents the effects of overfitting, as it encourages the model not to rely heavily on individual pixel intensity values to make predictions.

There are two approaches when working with normalization: Min- Max Scaling and Z- score. Since we have information regarding the range of data it is more favorable to use Min- Max Scaling for the dataset.

$$X_{new} = \frac{X - X_{min}}{X_{max} - X_{min}} \tag{1}$$

The minimum and maximum after scaling will take approximately values 0 and 1. This scaling is achieved by dividing each pixel value by 255.

```
x_train = x_train / 255
x_test = x_test / 255
```

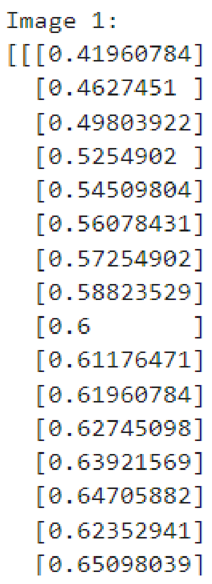
3.2.5 Reshaping

Another important pre-processing step when working with the CNN model is reshaping. Reshaping refers to the process of changing the input shapes and dimensions, without interfering with the original number of elements (images) within the dataset. This process consists of converting 2D images (with dimensions: length and width) into 4D format (with dimensions: batch_size, length, width, and channels) suitable for inputs to the CNN model.

```
x_train = x_train.reshape(-1,28,28,1)
x_test = x_test.reshape(-1,28,28,1)
```

Batch size represents the number of images in each batch. In our case “-1” serves as a placeholder, where the total size of the data (image’s number) is determined. The code also shows that there is only one channel (number 1 indicates that the dataset consists of grayscale images with a single channel).

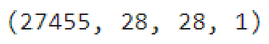
Dataset representation after reshaping:



```
Image 1:
[[[0.41960784]
  [0.4627451 ]
  [0.49803922]
  [0.5254902 ]
  [0.54509804]
  [0.56078431]
  [0.57254902]
  [0.58823529]
  [0.6       ]
  [0.61176471]
  [0.61960784]
  [0.62745098]
  [0.63921569]
  [0.64705882]
  [0.62352941]
  [0.65098039]
```

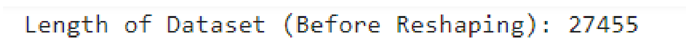
Results after reshaping:

- Each dataset image is reshaped into a 28x28 matrix format
- It generates a 4D array with a single grayscale channel

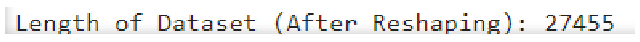


```
(27455, 28, 28, 1)
```

- The length of the dataset remains the same before and after reshaping



```
Length of Dataset (Before Reshaping): 27455
```

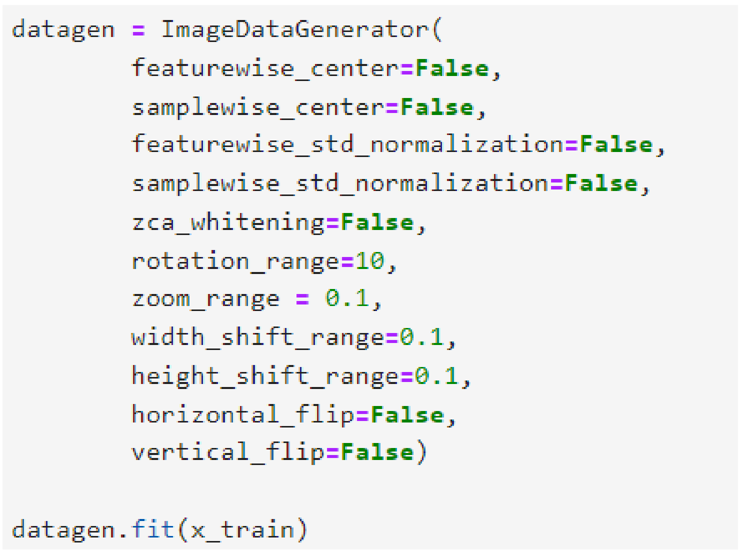


```
Length of Dataset (After Reshaping): 27455
```

3.2.6 Data Augmentation

CNN model is not scale or rotation invariant (if you rotate an image CNN might not achieve high accuracy), in this scenario, we implement a method called Data augmentation. Data augmentation is a technique that increases the size and diversity of the training dataset by applying different transformations to the data existing. By creating additional samples (images), which are variations of the original images, the model is exposed to a wider range of data. By incorporating variations into images in their lighting and orientation and many other processing components, the model generalizes better and improves his ability to recognize different images.

Some of the transform parameters that control the image extension process for our paper are rotation range (specifies the range of angles with which the images randomly move), zoom range, width/height shift range, and horizontal/vertical flip. "ImageDataGenerator" is implemented, which represents a class of the Keras library, designed for the generation of augmented data in tasks related to image data.



```
datagen = ImageDataGenerator(
        featurewise_center=False,
        samplewise_center=False,
        featurewise_std_normalization=False,
        samplewise_std_normalization=False,
        zca_whitening=False,
        rotation_range=10,
        zoom_range = 0.1,
        width_shift_range=0.1,
        height_shift_range=0.1,
        horizontal_flip=False,
        vertical_flip=False)


datagen.fit(x_train)
```

Figure 3.5 - Data Augmentation Technique

CHAPTER 4

ARCHITECTURE OF THE MODELS

In this chapter, we will study the Convolutional Neural Network (CNN) architecture, the characteristics of the system, its operation, and code implementation. We will see how this architecture serves as a feature extractor and further as classifications. For the CNN – SVM hybrid model we will employ CNN architecture as feature extractor and SVM as classifiers.

4.1 CNN Architecture

Convolutional neural networks (CNNs) are a type of deep learning architecture designed to learn directly from data. CNNs are particularly effective at identifying patterns in images, making them well-suited for object recognition and classification tasks. Below is an illustration of a CNN model used for image classification.

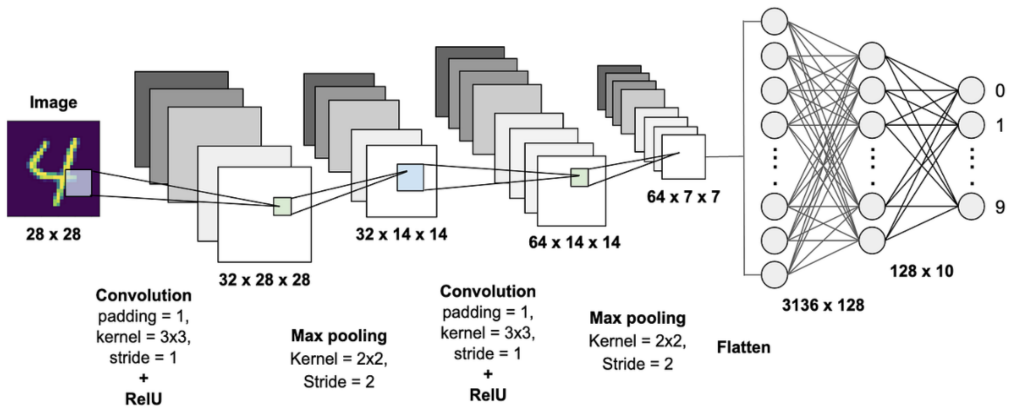


Figure 4.1 - CNN model for image classification

The above CNN takes a 28x28 image from the MNIST handwritten image dataset and builds the model for image classification.

From the descriptions on the image it is visible that four main processes take place:

- Convolution Layer
- Activation Function (ReLU)
- Pooling Layer
- Fully Connected Layer

The first three operations are part of feature extraction (which we will emphasize below) while the last operation introduces classification tasks.

4.1.1 Feature Extraction

Feature extraction is the dimensionality reduction process by which an initial raw data set is reduced into manageable sets of data for processing. The importance of this step for our model is due to:

- By reducing the dimensions of the data, the computational complexity is reduced, the risk of model overfit is prevented and the performance of the models is improved.
- Focusing more on features that carry important information, the feature extraction process removes redundant or useless information. In this way, a more accurate and concise representation of the data is obtained while preserving their essential characteristics.
- By presenting the data in a more interpretable format, feature extraction makes it easier for the model to understand and interpret the data and make decisions about it.
- CNN models trained on big datasets can be efficiently transferred to tasks requiring a smaller amount of labeled data through feature extraction. Developers can save time and computational resources by creating accurate and reliable models using pre-trained features without starting from scratch.

4.1.1.1 Convolutional Layer

Convolution is a mathematical operation that combines two functions to form a third one. It is commonly used in CNN models and other deep learning techniques to extract information from input data such as images, texts, audio, and so on. In our grayscale image dataset, what convolution does is that it applies a filter, known as a kernel to our input image.

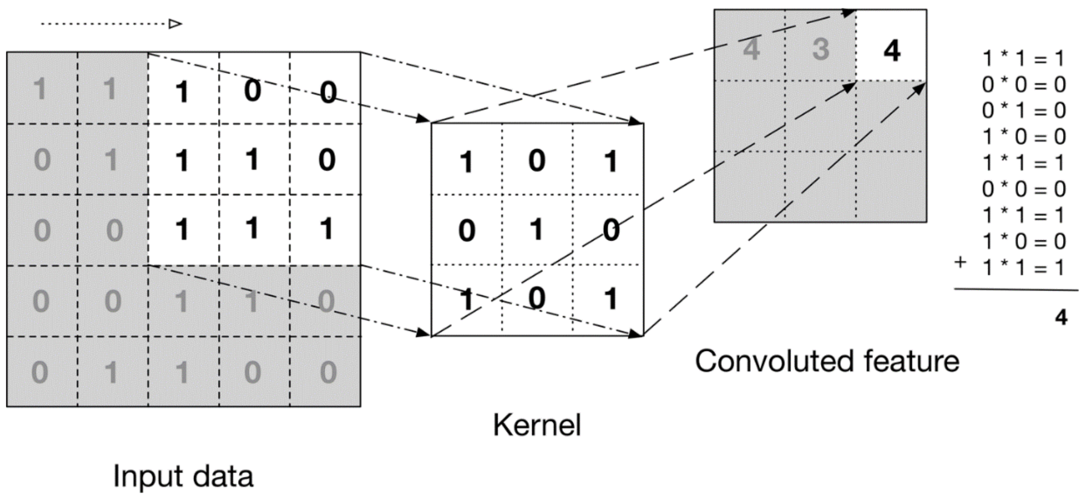


Figure 4.2 - Convolution operation on grayscale image

In the above figure, we represent the convolution operation. At first, a 3x3 filter (kernel) is applied to the image, to detect similar input portions with the filter applied. This filter is slid over the image, performing element-wise multiplication (between image pixels and filter) and then these results are summed up to give a final result.

After these operations, it is formed a new matrix called a “Feature Map” or as shown in the figure above “Convoluted feature”. Every value in the feature map captures patterns in the grayscale input image. We can employ different filters for an input image such as edge detection filters, sharpening, and blur filters.

The filter size or number is adjusted to achieve different results meanwhile the size of a feature map is made up of layers in the feature map (depth), stride (number of pixels by which a filter flows among the image), and padding (used to slid filter in the corners of the image).

Depth represents the number of filters (where each filter refers to a single feature map), that are employed for the convolution computations. If we have several N filters we will also have N feature maps.

Why is this important?

When you use multiple filters it means that we have more depth. The more filters you have the better you train the neurons to learn different patterns during the training process. Deeper layers are used to detect more complex features on the input image. For a better visualization of the depth concept, we can focus on the figure below.

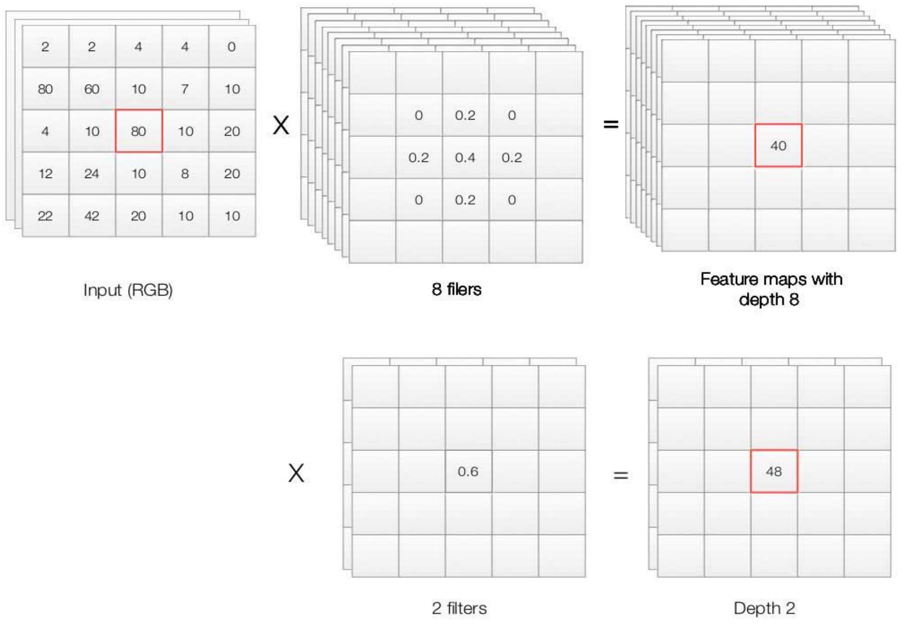


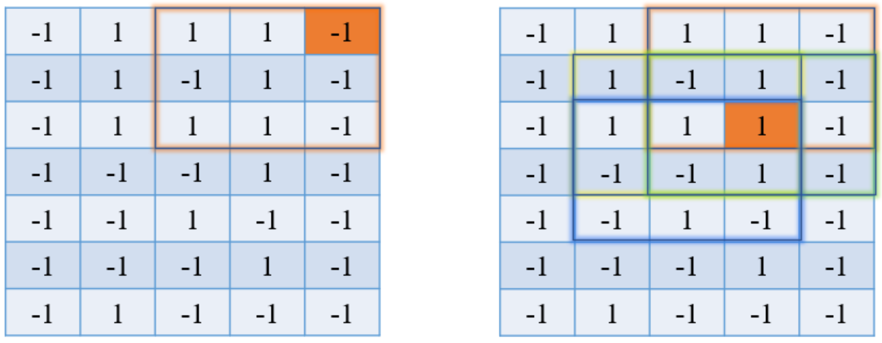
Figure 4.3 - Depth representation

After the convolution, the image dimensions are measured to be:

(Image size input – filter size +1) x (Image size input – filter size +1)

A disadvantage of the convolution is that the image becomes smaller and smaller losing too much information while doing the operations.

During convolution, the pixels in the corners and edges are less included in the convolution operations since the filter is applied only once to them. This means that the filters in the corner don’t play an important role in feature detection. Meanwhile, the pixels located in between (middle) contribute to multiple filters.



-1	1	1	1	-1
-1	1	-1	1	-1
-1	1	1	1	-1
-1	-1	-1	1	-1
-1	-1	1	-1	-1
-1	-1	-1	1	-1
-1	1	-1	-1	-1

-1	1	1	1	-1
-1	1	-1	1	-1
-1	1	1	1	-1
-1	-1	-1	1	-1
-1	-1	1	-1	-1
-1	-1	-1	1	-1
-1	1	-1	-1	-1

Figure 4.4 – Filters on the corner pixel and middle pixel

As a result, the pixels in the corners and edges contribute less to the final output, so the information in these parts may not be taken into consideration. To fix this problem, a technique is implemented called padding.

There are two types of padding: Valid Convolution or “Valid Padding” and Same Convolution. Valid Convolution has no padding, as explained above the problem with this approach is that the corner pixels do not contribute as much to image classification tasks. The dimensions (n x n) where the filter (f x f) is applied, will give the output dimensions to be (n – f + 1) x (n – f + 1).

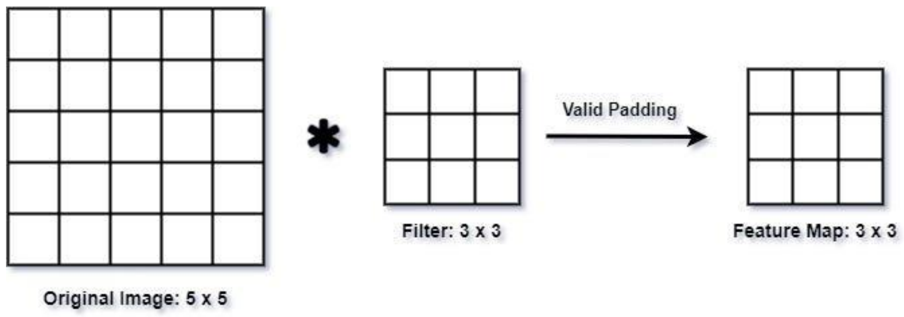


Figure 4.5 - Valid Convolution

To solve this problem we use the same convolution. In this technique, we add rows and columns of pixels from each side of the input image. Once the convolution operation starts sliding over the image, the corner pixels are included more than one time on different filters, giving these corners the same weight as the middle ones. After the convolution process, the output image has the same dimensions as the original input image. This is called the same convolution.

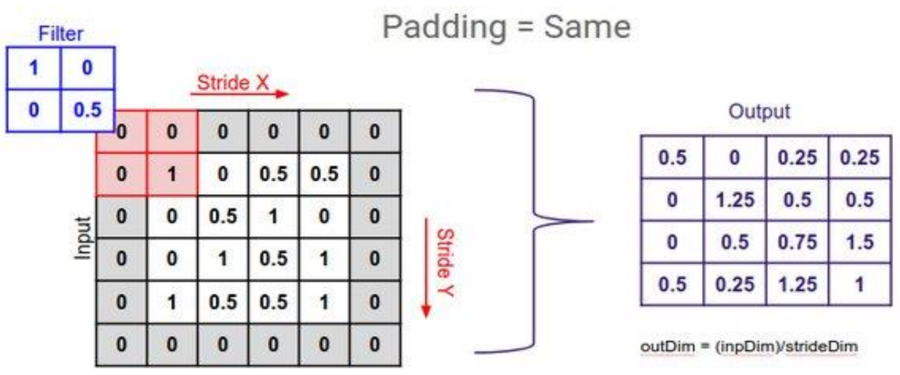


Figure 4.6 - Same Convolution

In this project, we apply a 3 x 3 size filter, as it requires less weight compared to other kernel sizes, it makes the model capture and recognize patterns in different positions effectively and helps to build more complex networks. The stride indicates the step with which the filter (kernel) moves horizontally and vertically over the pixels of the image.

The value determined for stride depends on what we expect in the output image. We prefer a small stride value if we expect some detailed features to be reflected in the output.

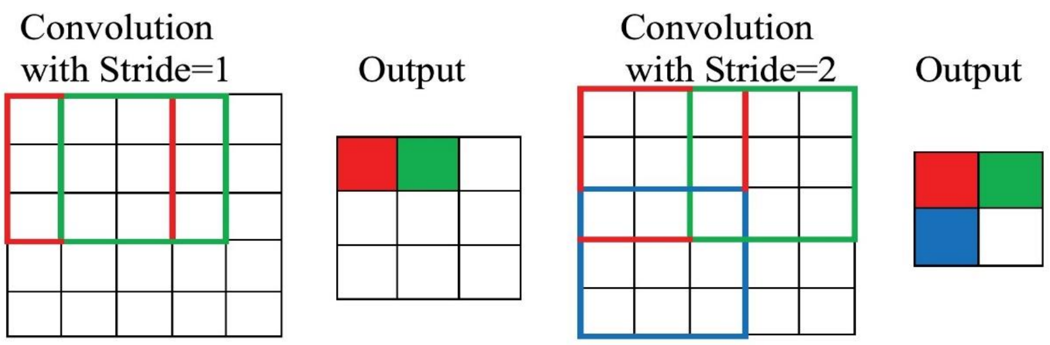


Figure 4.7 - Stride Visualization

4.1.1.2 ReLU Activation Function

The activation function is a mathematical operation that introduces non-linearity within the neural network. It helps the network to "learn" nonlinear relationships between inputs and outputs (these nonlinear relationships exist in tasks that require recognition of images since these objects have complex shapes that cannot be accurately represented by straight lines or simple combinations of pixels).

There are some activation functions such as ReLU (Rectified Linear Unit), Sigmoid, and Hyperbolic tangent. The most commonly used activation function in deep learning algorithms is the ReLU function. This choice is due to the efficiency in the calculation, the demand for less memory, and also its role as a regulator, which allows the network to focus on the informative features of the data.

The ReLU operation is applied after the convolutional operation

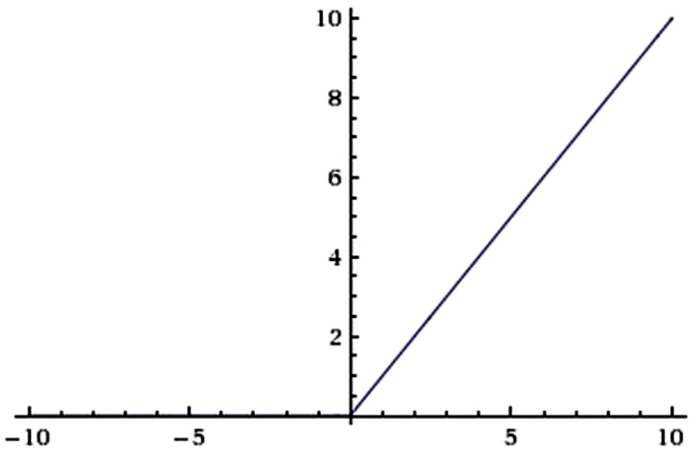


Figure 4.8 - ReLU operation

The ReLU function takes a feature map and whatever negative values are there it will replace them with 0. If the value is more than 0 it will keep it as it is. The mathematical operation is defined as below:

f(x) = max(0,x)

After we have computed the convolution operation, with the appropriate padding and stride as well as the activation function we build our convolution layers. In our work we have used 3 convolutional layers:

model.add(Conv2D(75 , (3,3) , strides = 1 , padding = 'same' , activation = 'relu' , input_shape = (28,28,1)))

model.add(Conv2D(50 , (3,3) , strides = 1 , padding = 'same' , activation = 'relu'))

```
model.add(Conv2D(25 , (3,3) , strides = 1 , padding = 'same' , activation = 'relu'))
```

The reason for applying several convolutional layers is to enable the model to learn complex and abstract patterns of input images. As the information passes through several layers, the model can capture hierarchical features that are progressively more necessary for the classification task.

When using multiple convolutional layers the model tends to incure overfitting. For this reason, we use some regularizations on the layers which will be discussed later in the paper.

4.1.1.3 Pooling Layer

Pooling is another layer in the CNN model, used also for feature extraction. The pooling layer is used to reduce the size or dimensions of the convolved feature that is obtained after computing the convolution calculations.

There are two types of pooling: max pooling and average pooling. The most commonly used approach is max pooling, which will be used in this work as well. In max pooling, you take a window (it can be the size of 2x2, the yellow window in Figure 4.9) and pick the maximum number. The maximum value in the yellow window is 30 so we insert it in the newly created feature map. We follow the same logic for other windows and create a feature map that requires less computation due to the reduced dimensions.

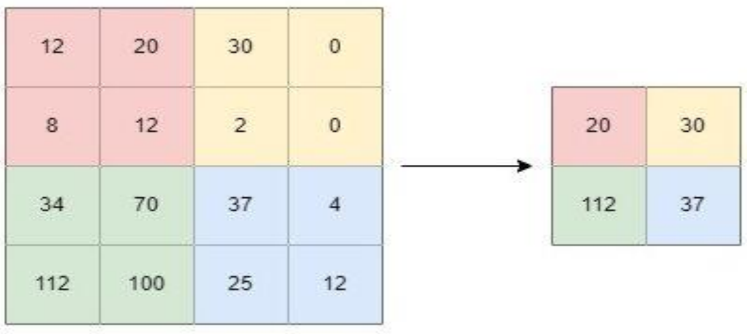


Figure 4.9 - Max pooling vizualization

Some benefits of the pooling layer are:

- It reduces the dimensions of the feature maps, leading to less complex computational operations. By both dimensions & computation reduction, the network can be trained faster and more efficiently.
- Pooling reduces overfitting since there are fewer parameters in the following layers of the network (it keeps the model more focused on the important filters rather than memorizing noise or unimportant details from the training set).
- Model is tolerant towards variations and distortion. Pooling aggregates local information becoming insensitive to small shifts, orientations of the object, and changes in contrast, shadow, or other lighting conditions.

4.1.2 CNN Classification

The CNN architecture is not completed yet. We have explained the feature extraction part, which includes the computations on the convolutional and pooling layers. The next step to be trained and implemented is the classification.

Classification is a supervised machine learning system, in which the data model is to generate the true label belonging to an input data. The model is trained with the particular data for its training and then evaluated with the particular data for testing. In this way, the model is prepared (trained) to perform one on other unknown data.

4.1.2.1 Fully Connected Layer

In this thesis, a layer of the CNN architecture called the Dense Layer or “Fully Connected Layer”, is used to classify the data. Up to the point where we compile the pooling layer we have a 2D array. Now we convert this 2D array (the feature map that is obtained) into a 1D array, in this way, the vector can be processed by the subsequent layers (Dense Layers).

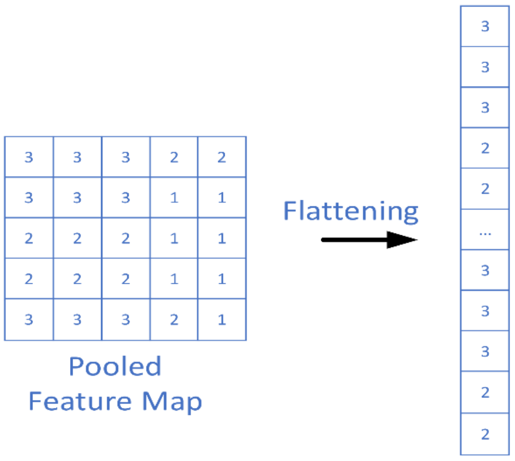


Figure 4.10 - Flattering visualization

This conversion process is called “flattening”. Flatten does not affect the batch size.

model.add(Flatten())

A layer utilized in the last phase of a neural network is called a dense layer. Each node of the dense layer is closely connected to the nodes of the subsequent layer, from which the dense layer receives the generated outputs. Dense layer nodes perform matrix-vector production. The condition for this action is that the number of rows of the output vector of the previous layer is equal to the number of columns of the dense layer vector. Dense Layer carries several hyperparameters, which are also used in our work.

A “unit” is a basic building element of a neural network that receives input data (for every input then are assigned weights), after that it is performed the weighted sum which is passed to the activation function to generate the final output.

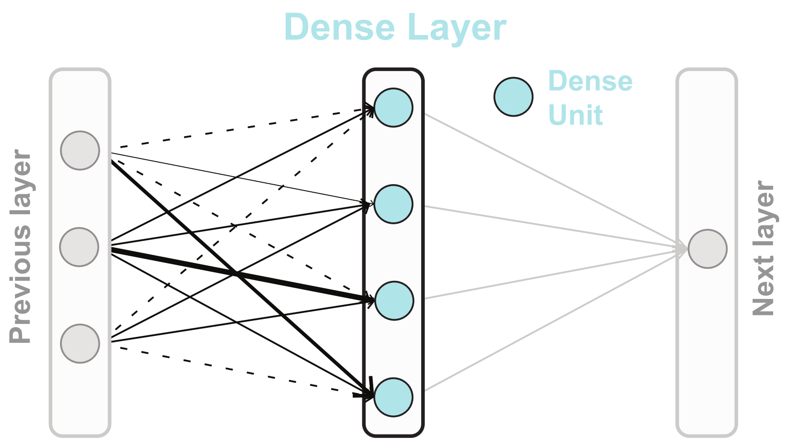


Figure 4.11 - Dense layer architecture

The activation function is an important step when dealing with multiple dense layers. In this study, we have implemented two types of activation functions: ReLU (which was discussed above) and softmax. Softmax is mostly used in the last layer of the network for solving multi-class classification tasks. It is defined as a normalized exponential function that converts the results of the previous layer into probabilities. It ensures that the sum of the generated probabilities is equal to one. The class that holds the greater probability it the one that is chosen as a predicted class from the model.

```
model.add(Dense(units = 512 , activation = 'relu'))

model.add(Dense(units = 24 , activation = 'softmax'))
```

The presence of several dense layers enables the model to learn more sophisticated relationships between input features and target classes. In this way, the model manages to make more accurate predictions for a classification task.

The “model.compile()” function is applied to set up the optimizer, loss function, and the accuracy metric for the training process.

For the configuration of the training process, Adam optimizer was used, which is a well-known optimization algorithm in the field of deep learning. The Adam optimizer adjusts the weights and biases of the model based on the gradients calculated by the loss function. These updates help the network learn and optimize its parameters.

For tasks that involve multi-class classification, the most commonly used loss function is "categorical_crossentropy". This function calculates the difference between the predicted class probabilities and the real labels. This difference serves as an estimator to show how well a model predicts the appropriate classes.

```
model.compile(optimizer = 'adam' , loss = 'categorical_crossentropy' , metrics = ['accuracy'])
```

To train the model, a parameter called "epochs" is set, which indicates the number of times, when the entire training dataset is passed through the layers of the model during the training phase. The configuration of the number of epochs depends on several factors:

- size of the dataset
- complexity of the problem
- progress during the training phase

- computer resources

```
model.fit(datagen.flow(x_train,y_train, batch_size = 128) ,epochs = 20 , validation_data = (x_test,
y_test))
```

In the case of a dataset with a significant number of data, a larger number of epochs is needed to train the model. When the accuracy decreases or stops improving after a certain number of epochs, this indicates that the model has converged and further training may not be necessary.

A balance between underfitting (few epochs) and overfitting (many epochs) is needed to find the point when the model achieves good generalization and performance.

4.1.3 Regularization Methods

To prevent overfitting and improve the efficiency of the model, CNN uses regularization techniques. In this work, we have used some approaches:

- Dropout regularization: It is a technique used to address overfitting problems in deep learning. Adding a dropout layer increases the performance of the neural network. What the technique does is that it randomly drops some neurons during each training sample. When we drop these neurons, the neurons in the subsequent layer cannot rely on one input as it might be dropped out at random. In the example below the neuron colored in yellow in the no-dropout neural network relies on 5 neurons, meanwhile when we use dropout the selected neuron relies on only 2 neurons.

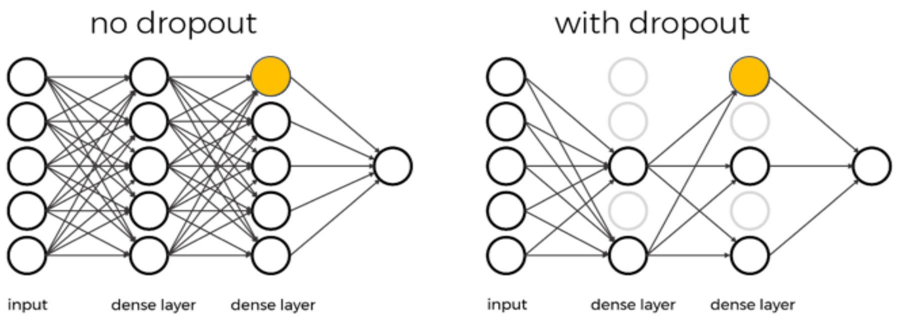


Figure 4.12 - Dropout regularization

In our model, we have used 2 dropout layers, with respectively a dropout rate of 0.2 and 0.3 (the layer randomly deactivates 20% and 30% of the input during each training sample).

```
model.add(Dropout(0.2))
model.add(Dropout(0.3))
```

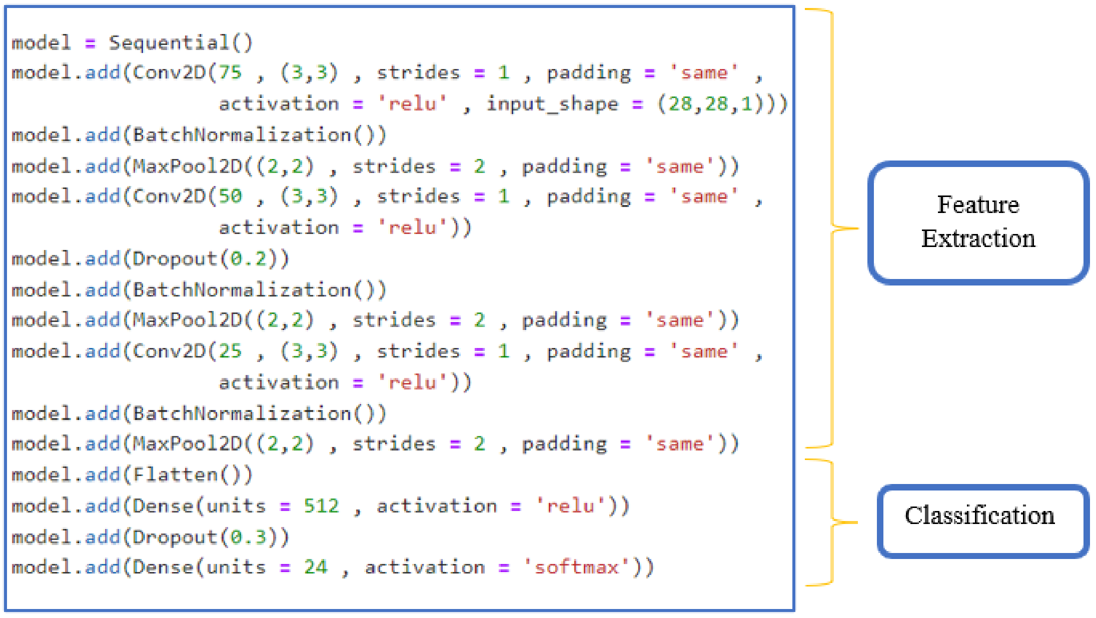
Batch Normalization is a technique that helps stabilize and speed up the model training process, which is realized by normalizing activations. Activations refer to the values produced by each neuron or node of a layer. They hold information that is necessary to make predictions or decisions. A small set of training data (batch) is taken and the activation values are adjusted in such a way that they have the same scalability. This technique avoids the divergence of the training process, which occurs due to the large variation of values. Batch Normalization also acts as a regulator by adding a small amount of noise during training to improve the model's capacity to generalize and reduce the effects of overfitting.

```
model.add(BatchNormalization())
```

In our model, we have used 3 layers of batch normalization. The choice was made by experimenting with the CNN architecture. Besides the advantages that it has on stabilization and generalization, multiple batch normalization layers come with additional computation costs affecting the overall training time of the evaluation.

4.1.4 Complete Architecture

The whole operations and steps that we have covered up to now are dedicated to the training process of the CNN model. The architecture initializes filters and parameters for convolution, sequentially finds an output, propagates it backward to train the network, and repeats steps until predicted outputs are near the actual output. Below we will give the code implemented of the architecture of CNN with the feature extraction part and classification layers applied.



```
model = Sequential()
model.add(Conv2D(75 , (3,3) , strides = 1 , padding = 'same' ,
                 activation = 'relu' , input_shape = (28,28,1)))
model.add(BatchNormalization())
model.add(MaxPool2D((2,2) , strides = 2 , padding = 'same'))
model.add(Conv2D(50 , (3,3) , strides = 1 , padding = 'same' ,
                 activation = 'relu'))
model.add(Dropout(0.2))
model.add(BatchNormalization())
model.add(MaxPool2D((2,2) , strides = 2 , padding = 'same'))
model.add(Conv2D(25 , (3,3) , strides = 1 , padding = 'same' ,
                 activation = 'relu'))
model.add(BatchNormalization())
model.add(MaxPool2D((2,2) , strides = 2 , padding = 'same'))
model.add(Flatten())
model.add(Dense(units = 512 , activation = 'relu'))
model.add(Dropout(0.3))
model.add(Dense(units = 24 , activation = 'softmax'))
```

Figure 4.13 - CNN architecture code

This architecture is the same for both models (CNN and hybrid CNN-SVM) used in this paper. It means that the feature extraction stage does not change for both algorithms. This is done because we want the model to be implemented under the same conditions and constraints so we can have a fair comparison between the two used approaches. The only stage that is different is the classification task where the second model employs SVM (Support Vector Machine) which will be discussed below.

4.2 CNN - SVM Hybrid Architecture

In the CNN – SVM hybrid model, SVM is used as a multi-class classifier replacing so, the last layer of the CNN model, the dense layer. CNN is responsible for the feature extractor while SVM plays the role of classifier.

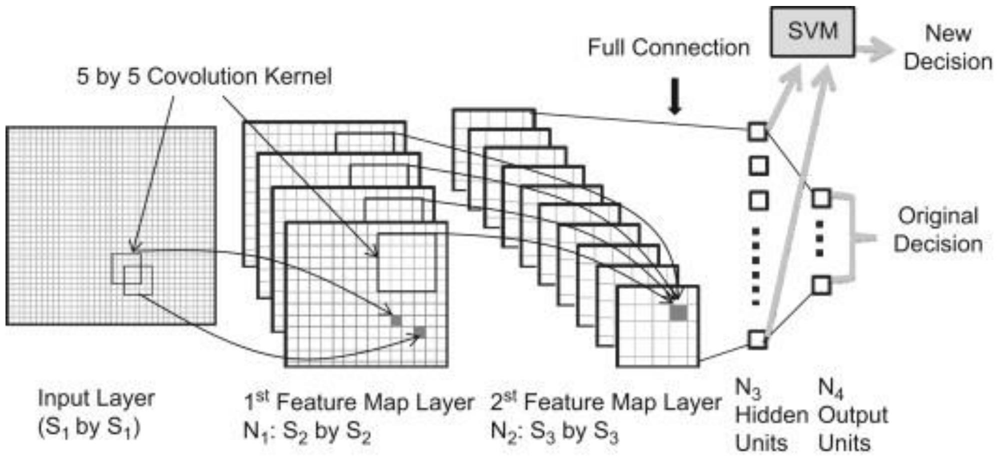


Figure 4.14 – CNN - SVM model architecture

The feature extraction and architecture of CNN are the same as the one that we used in the first model. The inputs that are passed to the N3 layer (the last layer, from the figure), in the hybrid model, will be extracted features that will be used by the SVM to train and continue with the tests and validation.

4.2.1 SVM Classifier

Support Vector Machine (SVM) is an algorithm that belongs to supervised machine learning. It is used for two purposes: regression & classification. Mainly SVM is used for classification tasks. The primary goal of this algorithm is to find an optimal decision boundary that best separates data points belonging to different categories. This decision boundary is called a hyperplane. To determine this hyperplane at first SVM needs a training set, which is already labeled with the respective category. The best hyperplane is chosen in such a way that the distance between it and the closest data points from each class is maximum (maximum margin). The closest data points to the hyperplane are called support vectors. They play an important role in defining the hyperplane and in classifying new data points.

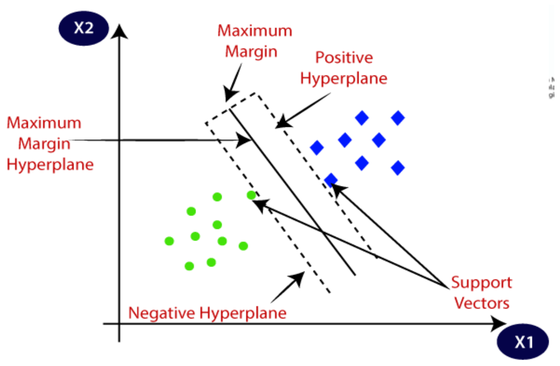


Figure 4.15 - SVM representation

The x_1 and x_2 are called features, based on which the SVM can decide where to classify the input image.

Based on the characteristics and complexity of the dataset we have two types of SVM: linear and non-linear SVM.

Linear SVM is used for data sets that are linearly separable. In this case, a straight line serves as a hyperplane to separate the classes. The algorithm finds the optimal hyperplane since there can be more than one line that can separate the classes. It does this by finding the closest data points for each of the classes (support vectors) and making sure that the distance of these vectors is maximal from the hyperplane. Maximizing distance, SVM seeks to create a greater separation between classes, which can improve the generalization of the model and make it more robust to noise or points being in the wrong class. Figure 4.15 demonstrates a linear SVM model.

Nonlinear SVM is used when the data are not linearly separable and require a more complex decision boundary. In the case where the data were separated linearly, we could use a straight line to separate the classes, here we can not. An example of non-linear SVM is shown in the figure below:

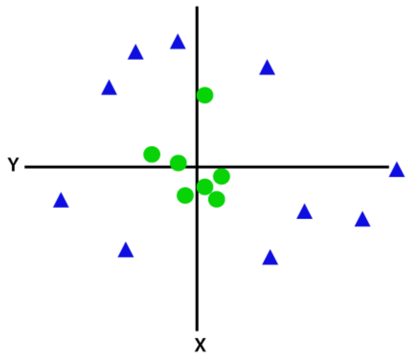


Figure 4.16 - Non-linear SVM

What do we do in this scenario?

A higher dimensional space is created so the data might be separatable by a hyperplane, even though in the original input they are not linearity separable. A kernel function then performs calculations on the high-dimensional space, finds the hyperplane, and projects back to the original space. This technique is differently called “Kernel Trick”. In the above example, we create a third dimension, (z dimension) using the formula:

$z = x^2 + y^2$

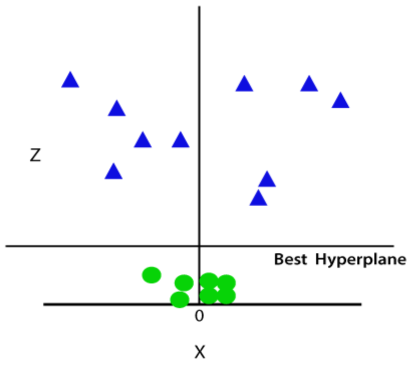


Figure 4.17 - SVM with z-dimension

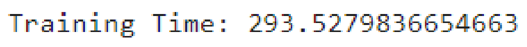
The transformation applied to the basic features is called a kernel. With this transformation, we can draw the decision boundary.

There are some types of kernels used in SVM classification tasks: RBF (Radial Basis Function), Sigmoid, Polynomial, and Linear kernel. In our model, we have implemented the RBF kernel to introduce non-linearity on the high dimensional space. The hyperplane generated by the RBF kernel is flexible and can take different shapes. This choice allows the SVM classifier to handle non-linear relationships and complex hyperplanes in the feature space.

```
svm_classifier = svm.SVC(kernel='rbf')

svm_classifier.fit(cnn_features_flatten, y_train)
```

We tried experimenting with different parameters for the SVM-CNN model, in this way we could find the optimal solution. When the Linear kernel was used the accuracy of the model decreased drastically compared to the above RBF kernel. Also, the training time when using the Linear kernel was longer compared to our optimal solution.



```
Training Time: 293.5279836654663
```

For the RBF kernel, we obtained an accuracy of 83% meanwhile for the alternative model the accuracy was 63%.

```
svm_classifier = svm.SVC(kernel='linear')

svm_classifier.fit(cnn_features_flatten, y_train)
```

By looking at these experimental results we decided to use the CNN-SVM model (which deploys a RBF kernel) for comparison with the CNN model since the accuracy and training time for these parameters, was more favorable for our case.

CHAPTER 5

MODEL TESTING & RESULTS COMPARISON

In this chapter, we will work on testing the models and comparing the results obtained from both models. We will interpret the conclusions drawn from the confusion matrixes and respective classification ratios. In the end, we evaluate the best-performing model.

5.1 Evaluation Metrics

Evaluation metrics are metrics used to evaluate performance and the effectiveness of a statistical or ML model. These metrics show how well the model performs and help compare different models or algorithms. Several metrics are taken into consideration in evaluating the results, some of which have been used in this paper as well.

A confusion matrix is a table that indicates the effectiveness of a model that classifies data. This table reflects the number of true positive, true negative, false positive, and false negative predictions. A confusion matrix, organized as a square, shows true classes in the rows and predicted classes in the columns. The matrix's main diagonal represents correctly classified instances, while the other part represents incorrect classifications.

The classification report is a summary of multiple evaluation metrics for each class in a classification task. Metrics such as precision, recall, F1 - score, and support for each class are determined and interpreted.

- Precision represents the value of the correctly predicted positive instances over the total predicted positive class. In other words, precision is the accuracy of the positive class predictions.

$$Precision = \frac{\text{TruePositive (TP)}}{TruePositive + FalsePositive\ (FP)} \tag{2}$$

- Recall is calculated as the ratio of the correctly predicted positive instances over the real (actual) positives. It measures the model's capacity to capture all positive observations.

$$Recall = \frac{\text{TruePositive (TP)}}{TruePositive + FalseNegative(FN)} \tag{3}$$

- F1 – Score provides a balance (harmonic mean) between precision and recall. The highest best value it can obtain is 1 while the worst takes the value of 0.

$$F1 = \frac{2 \times \text{Precision} \times \text{Recall}}{Precision + Recall} \tag{4}$$

- Support is the total number of observations in a specific class within the dataset used.

We also use other metrics to evaluate the model's performance such as accuracy, macro, and weighted average. Accuracy measures the total number of correctly predicted instances from all the classes of the dataset. Macro and weighted accuracy are useful for evaluating performance when the dataset is not balanced. They take into account the overall performance of all classes (categories).

A well-known technique in the field of ML for estimating the performance and generalization ability of a model is cross-validation. This approach involves dividing the available data into several subsets so that the model can be trained and tested with different data. There are several forms of cross-validation such as hold-out, k-fold cross-validation, etc., but in our paper, we used the simple case, which is hold–out cross-validation. It occurs when the dataset is divided into subsets for training and testing.

The dataset used is split into 80% of training data and 20% used for data testing.

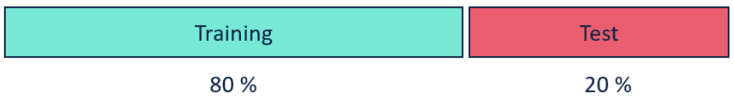
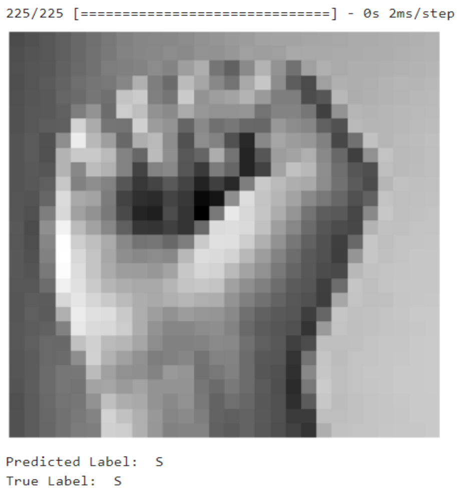


Figure 5.1 - Hold-out cross-validation

5.2 CNN Model Evaluation

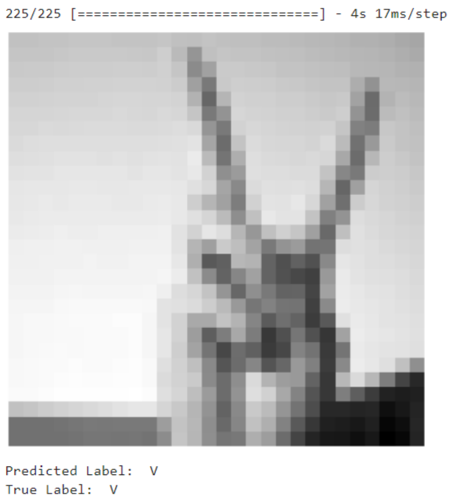
After we have trained the model, we will make some predictions to test the performance of the CNN model.



```
225/225 [==============================] - 0s 2ms/step
Predicted Label:  S
True Label:  S
```

Figure 5.2 – Predictions using CNN classifier

From the image, we can observe that our model performs well since the predicted label matches the real label. We run this part multiple times and it randomly prints the image together with the result of the predicted label and true label.



```
225/225 [==============================] - 4s 17ms/step
Predicted Label:  V
True Label:  V
```

Figure 5.3 - Random testing

Other metrics we need to observe when evaluating a model, are training and validation loss techniques.

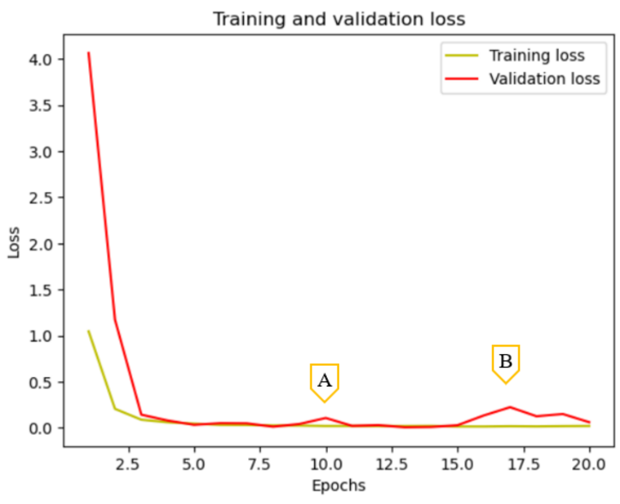


Figure 5.4 - Training and validation loss

Loss is a value that sums up the model's errors, so it measures how well or poorly our model is performing. Respectively the training loss (depicted in yellow) is used in the training set while the validation loss (depicted in red) is suited for the validation set. In the graph above we have visualized the performance of both lines (training and validation). We can observe that both loss-es tend to decrease and reach a steady state, which indicates an optimal balance within the dataset.

In the graph above we can observe two scenarios A and B, where the validation loss increases while the training loss remains the same. This demonstrates the effect of overfitting (the model performs less accurately on the testing set since it memorizes the training data very well).

Besides the training and validation loss, we can measure the performance of the model based on the accuracy. The training accuracy curve measures the model's fit to training data over subsequent epochs, with a gradual increase as training performs better. The validation accuracy curve measures the model's generalization to unseen data, indicating overfitting.

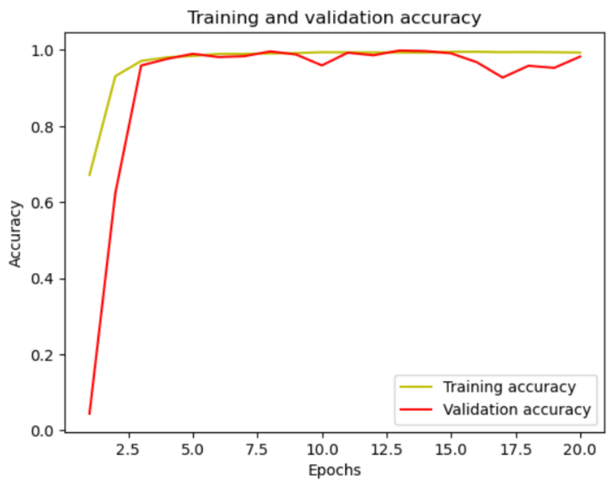


Figure 5.5 - Training and validation accuracy

The graph tends to increase meaning that the model is performing well (we can observe the tendency to overfit).

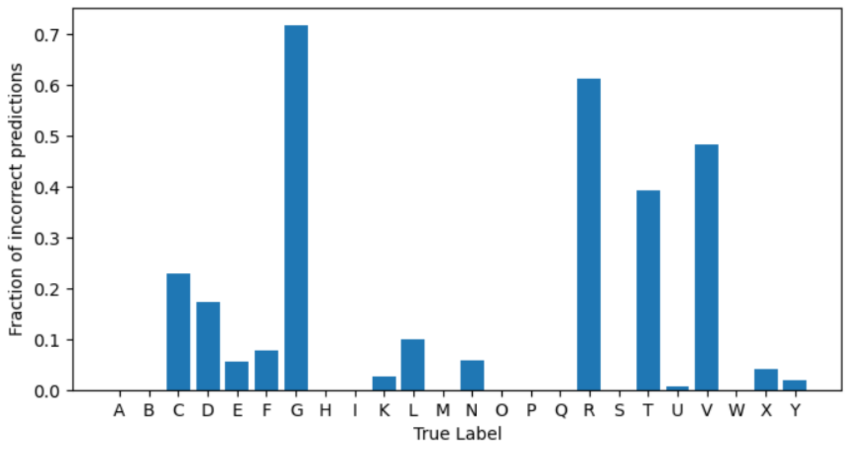


Figure 5.6 - Incorrect predictions from the CNN model

From the histogram, we can observe that mainly the letters G, R, T, and V are predicted incorrectly by the model.

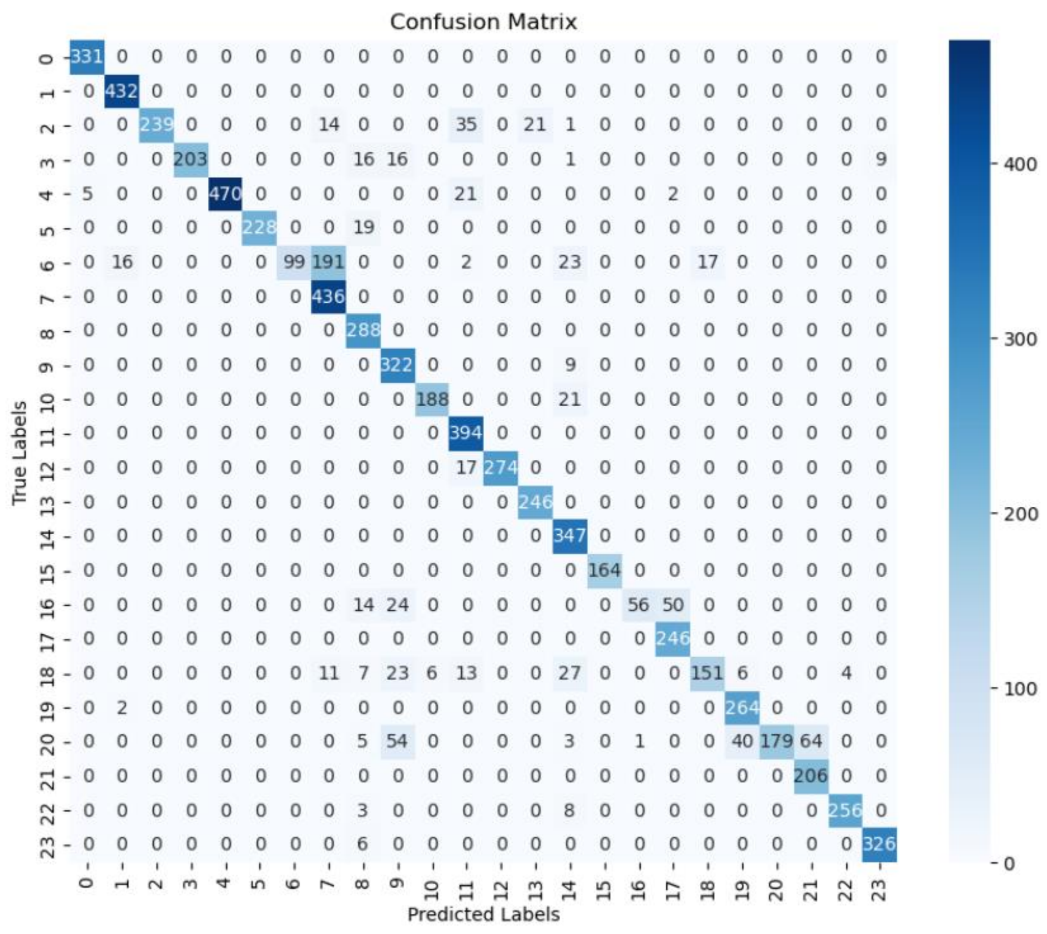


Figure 5.7 - Confusion matrix for CNN model

Figure 5.7 demonstrates the generated confusion matrix for the CNN model, which is used to evaluate the performance of the classification task. By looking at the true labels and predicted labels we can obtain the results of the total number of correct and incorrect predictions. The general rule is that anything that is not part of the main diagonal is calculated to be an error.

Table 5.1 - Confusion matrix results for the CNN model

Confusion Matrix Results	
Total number of correct predictions	6345
Total number of incorrect predictions	827

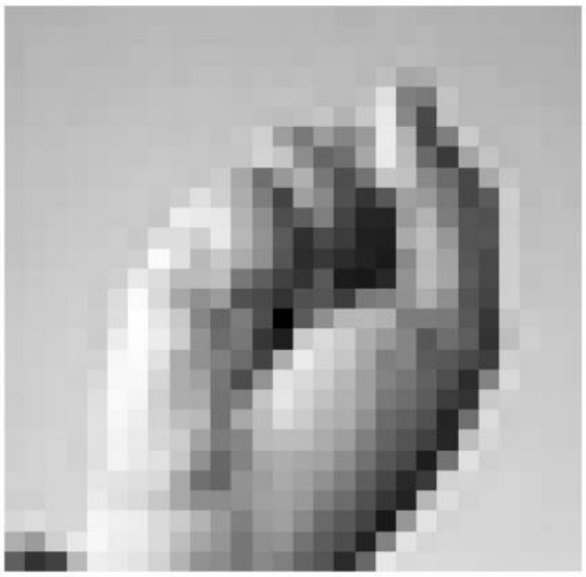
From the classification report, we will examine the results for the accuracy obtained for the CNN model.

```
accuracy                                0.88      7172
```

Since the standards of the industry are measured to be in the range of 70% and 90%, the model displays a valuable performance (88% in our case). The value 7172 indicates the number of the training samples.

5.3 CNN – SVM hybrid model evaluation

The accuracy of the CNN - SVM hybrid model will be evaluated using the test dataset data, the same as we did for the CNN model. Here are the prediction results:



```
Predicted Label:  S
True Label:  S
```

Figure 5.8 - Predictions using SVM classifier

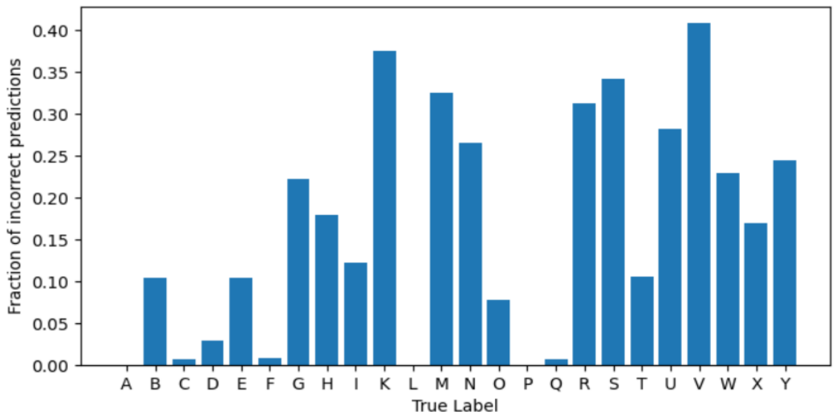
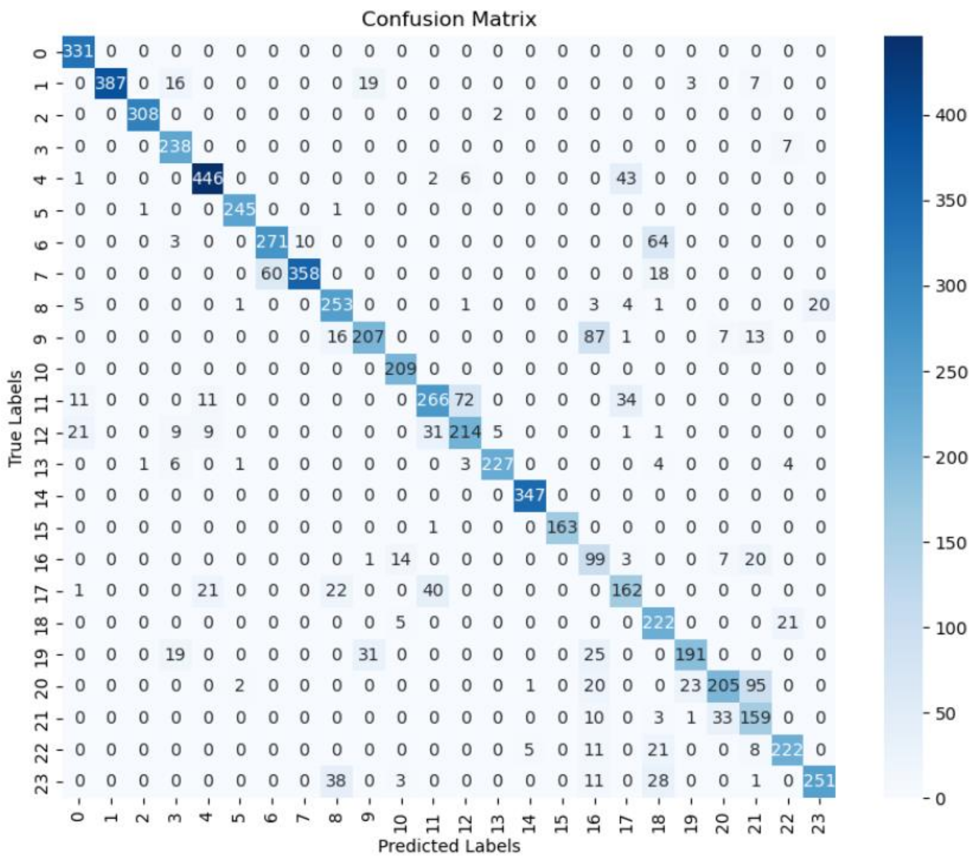


Figure 5.9 - Incorrect predictions from the CNN-SVM model

Comparec [illegible] licted more incorrect labels (letters K, N, R, S, etc), as can be seen from the histogram.

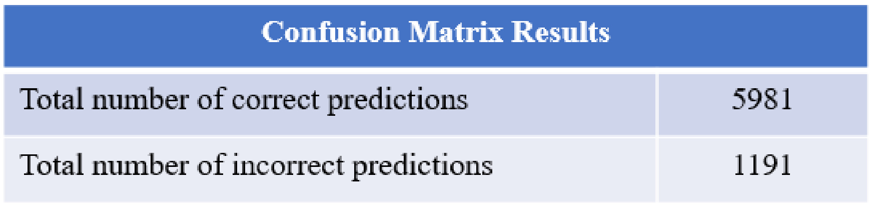


	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
0	331	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	387	0	16	0	0	0	0	0	19	0	0	0	0	0	0	0	0	0	3	0	7	0	0
2	0	0	308	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0
3	0	0	0	238	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	7	0
4	1	0	0	0	446	0	0	0	0	0	0	2	6	0	0	0	0	43	0	0	0	0	0	0
5	0	0	1	0	0	245	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	3	0	0	271	10	0	0	0	0	0	0	0	0	0	0	64	0	0	0	0	0
7	0	0	0	0	0	0	60	358	0	0	0	0	0	0	0	0	0	0	18	0	0	0	0	0
8	5	0	0	0	0	1	0	0	253	0	0	0	1	0	0	0	3	4	1	0	0	0	0	20
9	0	0	0	0	0	0	0	0	16	207	0	0	0	0	0	0	87	1	0	0	7	13	0	0
10	0	0	0	0	0	0	0	0	0	0	209	0	0	0	0	0	0	0	0	0	0	0	0	0
11	11	0	0	0	11	0	0	0	0	0	0	266	72	0	0	0	0	34	0	0	0	0	0	0
12	21	0	0	9	9	0	0	0	0	0	0	31	214	5	0	0	0	1	1	0	0	0	0	0
13	0	0	1	6	0	1	0	0	0	0	0	0	3	227	0	0	0	0	4	0	0	0	4	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	347	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	163	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	1	14	0	0	0	0	0	99	3	0	0	7	20	0	0
17	1	0	0	0	21	0	0	0	22	0	0	40	0	0	0	0	0	162	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	5	0	0	0	0	0	0	0	222	0	0	0	21	0
19	0	0	0	19	0	0	0	0	0	31	0	0	0	0	0	0	25	0	0	191	0	0	0	0
20	0	0	0	0	0	2	0	0	0	0	0	0	0	0	1	0	20	0	0	23	205	95	0	0
21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	3	1	33	159	0	0
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0	11	0	21	0	0	8	222	0
23	0	0	0	0	0	0	0	0	38	0	3	0	0	0	0	0	11	0	28	0	0	1	0	251

Figure 5.10 - Confusion matrix for the CNN- SVM hybrid model

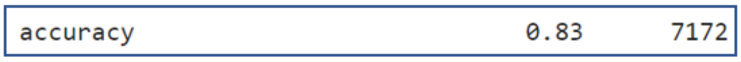
The results obtained from the confusion matrix are given in the table below:

Table 5.2- Confusion matrix results for CNN- SVM model



Confusion Matrix Results	
Total number of correct predictions	5981
Total number of incorrect predictions	1191

The accuracy of the hybrid CNN-SVM model seems to be slightly lower compared to the CNN model.



```
accuracy                                   0.83      7172
```

It is still within the accepted range of the model accuracy performance.

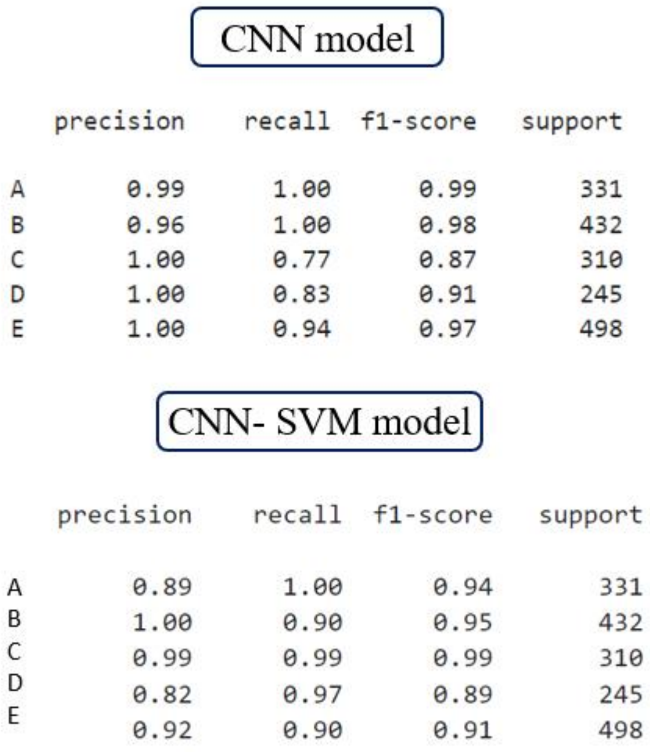
5.4 Model Comparison

To compare the performances and accuracies of each model, we will analyze all the estimation parameters generated between the respective algorithms.

Table 5.3 - Comparison of confusion matrices

	CNN model	**CNN-SVM hybrid model**
Total number of correct predictions	6345	5981
Total number of incorrect predictions	827	1191

The CNN model correctly predicted 6345 labels, while the hybrid CNN-SVM model predicted 5981. The CNN model outperformed the hybrid model in data classification due to its sophisticated learning architecture.


CNN model

```
     precision    recall  f1-score   support

A         0.99      1.00      0.99       331
B         0.96      1.00      0.98       432
C         1.00      0.77      0.87       310
D         1.00      0.83      0.91       245
E         1.00      0.94      0.97       498
```

CNN- SVM model

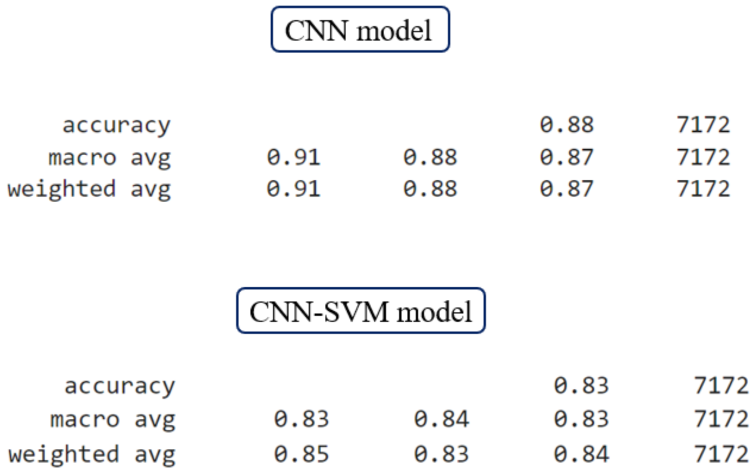
```
     precision    recall  f1-score   support

A         0.89      1.00      0.94       331
B         1.00      0.90      0.95       432
C         0.99      0.99      0.99       310
D         0.82      0.97      0.89       245
E         0.92      0.90      0.91       498
```

Figure 5.11 **-** Evaluation of metrics for both models

It is apparent that for the five labels, the values of precision, recall, and f1-score differ. In cases where the F1–score results in a larger value, it indicates a better balance between precision and recall, the largest achievable value of which is 1. If we were to consider label A and the others following the reasoning would be similar, the CNN model has achieved a higher precision value, showing us that the model has performed better in identifying cases for label A, compared to the other model.

Other parameters we have looked into are accuracy, macro average, and weighted average. The difference between the two algorithms (models) is not particularly large.



CNN model

```
    accuracy                           0.88      7172
   macro avg       0.91      0.88      0.87      7172
weighted avg       0.91      0.88      0.87      7172
```

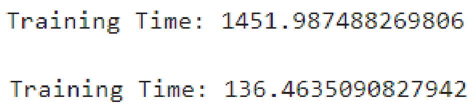
CNN-SVM model

```
    accuracy                           0.83      7172
   macro avg       0.83      0.84      0.83      7172
weighted avg       0.85      0.83      0.84      7172
```

Figure 5.12 - Accuracy evaluation

The CNN model has a higher accuracy compared to CNN-SVM indicating that the first model performs better on the sign language detection and classification tasks.

Another important parameter that we consider when evaluating and comparing two algorithms is the training time. The model's training time indicates its efficiency, resource allocation, and effectiveness in large datasets, particularly in real applications where time is a critical factor.



```
Training Time: 1451.987488269806

Training Time: 136.4635090827942
```

The first training time is generated from the training of the CNN model, while the second one is taken from the CNN-SVM model. We can observe a significant difference between the two algorithms due to the below-mentioned factors:

- The CNN model is structured with multiple layers (including the convolutional layers, pooling layers, and dense layers), compared to the CNN-SVM model which has a simpler architecture. This means that the complexity of the model directly affects the
- CNN model hyperparameters include the number of epochs, which has a huge influence on the training time. As the number of epochs increases the training time does so, since the number of operations performed is repeated for each epoch.

CHAPTER 6

CONCLUSIONS

6.1 Conclusion

This thesis presents a comparative study between two deep-learning models used for Sign language Detection and classification. The main objective was to distinguish which among the used models (CNN and CNN-SVM) outperforms the others. The first CNN model is used both as a feature extractor and classifier, meanwhile, the second approach CNN-SVM, uses SVM as a classifier. To maintain the models in equal conditions and conduct a fair comparison we have used the same CNN architecture for both algorithms, and the environment implemented is the same as well. From the results obtained after testing the dataset on multiple metrics, we can observe that the CNN model offers higher accuracy compared to the other model.

On the other hand, the SVM model requires less time to implement but lacks accuracy compared to CNN. The choice between using CNN and CNN-SVM depends on whether the application prioritizes the runtime or the level of accuracy. Real-life applications such as medical imaging require a higher rate of accuracy, justifying the long runtime, facial recognition systems and autonomous vehicles also require high accuracy for security reasons. While the CNN-SVM is suitable for applications that prioritize training time such as video systems or real-time image processing.

In conclusion, our study illustrates the trade-offs of CNN and CNN-SVM models for the identification of sign language and demonstrates how real-time sign language recognition technology might contribute to the development of a more diverse and connected society.

Table 6.1 - Comparison Table of Accuracy

Models	**Dataset**	**Image**	**Accuracy (%)**	**Training time (seconds)**
CNN (Used model with 20 epochs)	American Sign Language	Grayscale	88	1451.98
CNN-SVM (Used model)	American Sign Language	Grayscale	83	136.46
KNN- k-nearest neighbors [27]	American Sign Language	Grayscale	60	675
SVM with linear kernel [27]	American Sign Language	Grayscale	78	384
CNN (50 epochs) [27]	American Sign Language	Grayscale	88	Not mentioned
CNN- Convolution neural networks [9]	Bhutanese Sign Language	RGB	97	Not mentioned
KNN- k-nearest neighbors [9]	Bhutanese Sign Language	RGB	78	Not mentioned
SVM- Support vector machine [9]	Bhutanese Sign Language	RGB	70	Not mentioned

6.2 Future Recomandations

Within machine learning, there is frequently a trade-off between the accuracy and task-solving capabilities of the model as well as execution time. The CNN model has improved prediction accuracy, but it takes longer to train and more processing power to use. Conversely, the hybrid CNN-SVM approach improves speed at the expense of efficiency. Subsequent research endeavors should focus on optimizing hyperparameters for every model and integrating strategies for preparing data. We described a few other approaches, such k-fold cross-validation, for assessing model performances. Better feature selection for each model could result from experimenting with this evaluation technique.

Sign language recognition technologies are essential for reducing communication barriers between the hearing and the deaf communities. Although machine learning techniques for translating and recognizing sign language have advanced, more work has to be done to promote inclusivity.

Continuous research and development efforts can focus on improving the accuracy and robustness of sign language recognition systems. This includes addressing challenges such as language variations, individual differences, and handling complex sign gestures.

Efforts can be directed towards real-time sign language translation systems to provide access to communication for the deaf community. Integration of such systems into widely used video conferencing platforms can offer inclusivity. Sign language systems promote education and employment opportunities for individuals who are deaf. By providing access to information, these systems empower them to be educated, employed, and actively participate in various professional fields.

The overall study recommends developing a real-time sign language recognition system to promote inclusivity and communication for deaf individuals, thereby enabling their full participation in daily life.

REFERENCES

[1] "Segmentation of medical images using adaptive region growing," *IEEE,* p. 11, 2017.

[2] Mustafa Khan; Amita Gondi , "Interpreting Sign Language using Image Classification Techniques," *IEEE,* 2018.

[3] K. Bantupalli, "American Sign Language Recognition using Machine Learning and Computer Vision," *IEEE,* 2019.

[4] Ankita Wadhawan; Parteek Kumar, "Deep learning-based sign language recognition system for static signs," *IEEE,* 2020.

[5] MUNEER AL-HAMMADI; GHULAM MUHAMMAD; WADOOD ABDUL; MANSOUR ALSULAIMAN; MOHAMMED A. BENCHERIF; TAREQ S. ALRAYES; HASSAN MATHKOUR; MOHAMED AMINE MEKHTICHE, "Deep Learning-Based Approach for Sign Language Gesture Recognition With Efficient Hand Gesture Representation," *IEEE,* 2020.

[6] D. Bhavanaa ; K. Kishore Kumar ; Medasani Bipin Chandraa ; P.V. Sai Krishna Bhargava; D. Joy Sanjanaa ; G. Mohan Gopi, "Hand Sign Recognition using CNN," *IEEE,* 2021.

[7] Amitkumar Shinde; Ramesh Kagalkar, "Advanced Marathi Sign Language Recognition using Computer Vision," *IEEE,* 2015.

[8] ADEYANJU; I.A; BELLO; O.O. ;ADEGBOY; M.A, "Machine learning methods for sign language recognition: a critical review and analysis," *IEEE,* 2021.

[9] Md. Jahid Hasan; Md. Shahin Alom; Md. Ferdous Wahid, "Digit Recognition in Sign Language Based on Convolutional Neural Network and Support Vector Machine," *IEEE,* 2018.

[10] Ahmed Sultan; Walied Makram; Mohammed Kayed; Abdelmaged Amin Ali, "Sign language identification and recognition: A comparative study," *IEEE,* 2022.

[11] Sabaheta Đogić; Gunay Karli , "Sign Language Recognition using Neural Networks," *IEEE*, 2020.

[12] S. Raschka, "Model Evaluation, Model Selection, and Algorithm Selection in Machine Learning," *IEEE*, 2020.

[13] M. M. Saleem, "Deep Learning Application On American Sign Language Database For Video-Based Gesture Recognition," *IEEE*, 2020.

[14] Marlon Oliveir; Houssem Chatbri; Suzanne Little ;Noel E. O'Connor; Alistair Sutherland, "A comparison between end-to-end approaches and feature extraction-based approaches for Sign Language recognition.," *IEEE*, 2017.

[15] Amrutha K; Prabu P, "ML Based Sign Language Recognition System," *IEEE*, 2021.

[16] Rahib H.Abiyev; Murat Arslan; John Bush Idoko, "Sign Language Translation Using Deep Convolutional Neural Networks," *IEEE*, 2020.

[17] MUHAMMAD AL-QURISHI; THARIQ KHALID; RIAD SOUISSI, "Deep Learning for Sign Language Recognition: Current Techniques, Benchmarks, and Open Issues," *IEEE*, 2021.

[18] A. F. M. Agarap, "An Architecture Combining Convolutional Neural Network (CNN) and Support Vector Machine (SVM) for Image Classification," 2019.

[19] Yamashita, Rikiya, Mizuho Nishio, Richard Kinh Gian Do, and Kaori Togashi. "Convolutional neural networks: an overview and application in radiology.", *Insights into imaging* 9 ,(2018)

[20] Choudhary, Kamal, et al. "Recent advances and applications of deep learning methods in materials science.",*npj Computational Materials* 8.1, (2022)

[21] World Health Organization. (2024, February 2). *Deafness and hearing loss*. https://www.who.int/news-room/fact-sheets/detail/deafness-and-hearing-loss

[22] Kothadiya, Deep, Chintan Bhatt, Krenil Sapariya, Kevin Patel, Ana-Belén Gil-González, and Juan M. Corchado. "Deepsign: Sign language detection and recognition using deep learning." *Electronics* 11, no. 11, (2022)

[23] Adeyanju, Ibrahim Adepoju, Oluwaseyi Olawale Bello, and Mutiu Adesina Adegboye. "Machine learning methods for sign language recognition: A critical review and analysis." *Intelligent Systems with Applications* 12, (2021)

[24] Bantupalli, Kshitij, and Ying Xie. "American sign language recognition using machine learning and computer vision." (2019)

[25] Vakili, Meysam, Mohammad Ghamsari, and Masoumeh Rezaei. "Performance analysis and comparison of machine and deep learning algorithms for IoT data classification." *arXiv preprint arXiv:2001.09636*, (2020)

[26] Kamath, Cannannore Nidhi, Syed Saqib Bukhari, and Andreas Dengel. "Comparative study between traditional machine learning and deep learning approaches for text classification." In *Proceedings of the ACM Symposium on Document Engineering 2018*, (2018)

[27] Datamunge. https://www.kaggle.com/datasets/datamunge/sign-language-mnist. “Sign Language MNIST.”

[28] Salama, Wessam M., and Moustafa H. Aly. "Deep learning in mammography images segmentation and classification: Automated CNN approach."*Alexandria Engineering Journal* 60, no. 5, (2021)

[29] Kembuan, Olivia, Gladly Caren Rorimpandey, and Soenandar Milian Tompunu Tengker. "Convolutional neural network (CNN) for image classification of indonesia sign language using tensorflow." *2020 2nd International Conference on Cybernetics and Intelligent System (ICORIS)*. IEEE, 2020.

[30] Sharma, Sakshi, and Sukhwinder Singh. "Vision-based hand gesture recognition using deep learning for the interpretation of sign language." *Expert Systems with Applications* 182, (2021)