COMPARATIVE ANALYSIS OF DETECTION OF TEXT FROM MORSE CODE IN HANDWRITTEN IMAGES USING CONVOLUTIONAL NEURAL NETWORKS

A
Project Report Submitted

in partial fulfillment of the requirements for the award of the Degree of

BACHELOR OF TECHNOLOGY

in
COMPUTER SCIENCE & ENGINEERING
By

Mangam Surya Prakash   **19761A0537**

Under the esteemed guidance of

Shaik Johny Basha

Sr. Assistant Professor

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

LAKIREDDY BALI REDDY COLLEGE OF ENGINEERING (AUTONOMOUS)
Accredited by NAAC with 'A' Grade & NBA (Under Tier - I), ISO 9001:2015 Certified Institution
Approved by AICTE, New Delhi and Affiliated to JNTUK, Kakinada
L.B. REDDY NAGAR, MYLAVARAM, NTR DIST., A.P.-521 230.
2019-2023

LAKIREDDY BALI REDDY COLLEGE OF ENGINEERING (AUTONOMOUS)

Accredited by NAAC with 'A' Grade & NBA (Under Tier - I), ISO 9001:2015 Certified Institution
Approved by AICTE, New Delhi and Affiliated to JNTUK, Kakinada
L.B. REDDY NAGAR, MYLAVARAM, NTR DIST., A.P.-521 230.
Department of COMPUTER SCIENCE & ENGINEERING

CERTIFICATE

This is to certify that the project entitled **"Comparative Analysis of Detection of Text from Morse Code in Handwritten Images using CNN"** is being submitted by

Mangam Surya Prakash   19761A0537

in partial fulfillment of the requirements for the award of degree of **B.Tech** in **Computer Science & Engineering** from **Jawaharlal Nehru Technological University Kakinada** is a record of bonafide work carried out by them at **Lakireddy Bali Reddy College of Engineering (A).**

The results embodied in this Project report have not been submitted to any other University or Institute for the award of any degree or diploma.

PROJECT GUIDE   **HEAD OF THE DEPARTMENT**

Sk. Johny Basha   **Dr. D. Veeraiah**

EXTERNAL EXAMINER

ACKNOWLEDGEMENT

Mangam Surya Prakash   **19761A0537**

DECLARATION

We are here by declaring that the project entitled **"Comparative Analysis of Detection of Text from Morse Code in Handwritten Images using CNN"** work done by us. We certify that the work contained in the report is original and has been done by me under the guidance of supervisor. The work has not been submitted to any other institute in preparing for any degree or diploma. We have followed the guidelines provided by the institute in preparing the report. We have conformed to the norms and guidelines given in the Ethical Code of Conduct of the institute. Whenever we have used materials (data, theoretical analysis, figures, and text) from other sources, we have given due credit to them by citing them in the text of the report and giving their details in the references. Further, we have taken permission from the copyright's owner of the sources, whenever necessary.

Signature(s) of the students(s)

Mangam Surya Prakash (19761A0537)

ABSTRACT

Morse code is a method used in telecommunication to encode text characters as standardized sequences of two different electronic pulses usually represented as short pulse(dot) and long pulse(dash). Detection of text from images of morse code is a complex process and there is no active research on this area. As these are morse code images, different images have different style of strokes. This project aims to develop an automated Morse code recognition system by training a convolutional neural network (CNN) model using a self-built dataset. The proposed approach involves collecting and preprocessing images of Morse code characters and creating a labeled dataset for training and testing the CNN model. The dataset creation process includes capturing images of different Morse code characters, augmenting the data to increase the dataset size, and annotating the images to label them correctly. The CNN model is then trained using the created dataset and evaluated for its accuracy in recognizing Morse code characters in images. The results demonstrate that the proposed approach achieves high accuracy in recognizing Morse code characters in images, making it a promising solution for automated Morse code recognition systems.

LIST OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVATIONS

1. **CNN:** Convolutional Neural Networks

2. **GPU:** Graphics Processing Unit

3. **BLSTM:** Bi-Directional Long Short-Term Memory

4. **RNN:** Recurrent Neural Networks

5. **CTC:** Connectionist Temporal Classification

6. **ML:** Machine Learning

7. **AI:** Artificial Intelligence

8. **ReLU:** Rectified Linear Unit

9. **TPU:** Tensor Processing Unit

10. **OpenCV:** Open-Source Computer Vision

11. **SSD:** Solid State Drive

12. **ILSVRC:** ImageNet Large Scale Visual Recognition Challenge

13. **HTML:** Hyper Text Markup Language

1. INTRODUCTION

1.1 OVERVIEW OF THE PROJECT

This project aims to use Convolutional Neural Networks (CNNs) to automatically detect and decode text encoded in Morse code from images. Morse code is a method of encoding text as a series of dots and dashes, which can be transmitted via radio waves, light signals, or other means. The project involves several steps, including image preprocessing to enhance the quality of the input images, Morse code detection using computer vision techniques, and text decoding using machine learning algorithms. The input images may contain Morse code signals embedded in a variety of backgrounds, which can pose a challenge for accurate detection. Therefore, the project will likely involve the use of various image processing techniques to improve the contrast, brightness, and sharpness of the input images to enhance the visibility of the Morse code signals. Once the Morse code signals are detected, the project will use CNNs to automatically decode the signals and convert them back into text. This will involve training the CNNs on a large dataset of Morse code signals and their corresponding text representations. Overall, this project has potential applications in various fields, such as military and emergency services, where Morse code signals may still be used for communication in some situations.

1.2 FEASIBILITY STUDY

All activities are available for use, provided with unlimited resources and time indefinite. However, software development is due to a lack of resources, as well as complex supply costs. It has been compulsory and sensible, to assess the feasibility of a project in a shortest time.

1.2.1 Economic Feasibility

The project's economic feasibility is moderate as it requires specific hardware and software resources, including image data sets, a GPU for training the CNN, and development tools such as Python and TensorFlow. These resources may require significant investments to acquire, and maintenance costs should also be taken into consideration.

1.2.2 Technical Feasibility

The project's technical feasibility is high as CNNs have proven to be highly effective in image processing tasks, including image recognition and object detection. Morse code, being a

simple binary code, can be easily recognized by the network. However, the performance of the model will depend on the quality of the images, noise present, and the complexity of the Morse code patterns.

1.2.3 Social Feasibility

The project's social feasibility is high as the proposed system can potentially be used in various fields, such as military, emergency services, and amateur radio. The system can be used to extract critical information from images captured by drones or satellites and provide real-time updates to the concerned authorities.

1.3 SCOPE

The scope of the project "Detection of text from Morse code in images using CNN" is to develop a system that can automatically detect and decode Morse code present in images using convolutional neural networks (CNNs). The project's primary objective is to provide a reliable and efficient solution to extract and decode Morse code information from images in real-time.

The system's scope includes acquiring and preprocessing image data sets containing Morse code patterns, designing, and training a CNN model to recognize and classify Morse code patterns in images, and developing a user interface for displaying the decoded text.

The system will be capable of detecting and decoding Morse code patterns with varying levels of complexity and noise in different image formats, including grayscale and color images. The system will also have the potential to handle multiple Morse code patterns present in a single image and provide the decoded text as output.

In summary, the scope of the project "Detection of text from Morse code in images using CNN" is to develop a reliable and efficient system that can detect and decode Morse code present in images with varying levels of complexity and noise, and have potential applications in various fields

2. <u>LITERATURE SURVEY</u>

To effectively accomplish this assignment, we read some source material before we started. For Detection of Text from Morse Code in Pictures Using CNN, several state-of-the-art methods have been created by researchers. In this part, a summary of their work is provided.

In the base paper **Research on Automatic Detection of Morse Code Based on Deep Learning,** they have taken morse signal time-frequency diagrams as input, then they have applied convolutional neural networks on the input which extracts the important features of the time-frequency diagrams including the position of dot and dash, the frequency drift and the strength of the signal. Therefore, the features extracted by the CNN can be arranged in time dimension, and the different feature values on the same time step are concatenated into one frame of a feature sequence. These frames are sequentially input into the RNN layer to predict the label of the current frame. For the prediction of each frame, a transcription layer is needed to integrate them into a Shorter final label. In this structure combining CNN and RNN is called CRNN. We build a neural network model based on the idea of CRNN, in which the Bidirectional Long Short-Term Memory (BLSTM) network is used to label feature frames from CNN, and CTC is used as the integration layer.

In the research paper **Morse Code Datasets for Machine Learning,** they have presented an algorithm to generate synthetic datasets of tunable difficulty on classification of Morse Code symbols for supervised machine learning problems, in particular, neural networks. The datasets are spatially one-dimensional and have a small number of input features, leading to high density of input information content. This makes them particularly challenging when implementing network complexity reduction methods. They have explored how network performance was affected by deliberately adding various forms of noise and expanding the feature set and dataset size. Finally, they have established several metrics to indicate the difficulty of a dataset and evaluate their merits.

2.1 EXISTING SYSTEM & DRAWBACKS

In the base paper, they have taken Morse Code frequency-diagrams as input. Then they have applied Convolutional Neural Networks to the frequency-diagrams in order to extract the important

features from the input data. Therefore, the features extracted by the CNN can be arranged in time dimension, and the different feature values on the same time step are concatenated into one frame of a feature sequence. These frames are sequentially input into the RNN layer to predict the label of the current frame. For the prediction of each frame, a transcription layer is needed to integrate them into a Shorter final label.
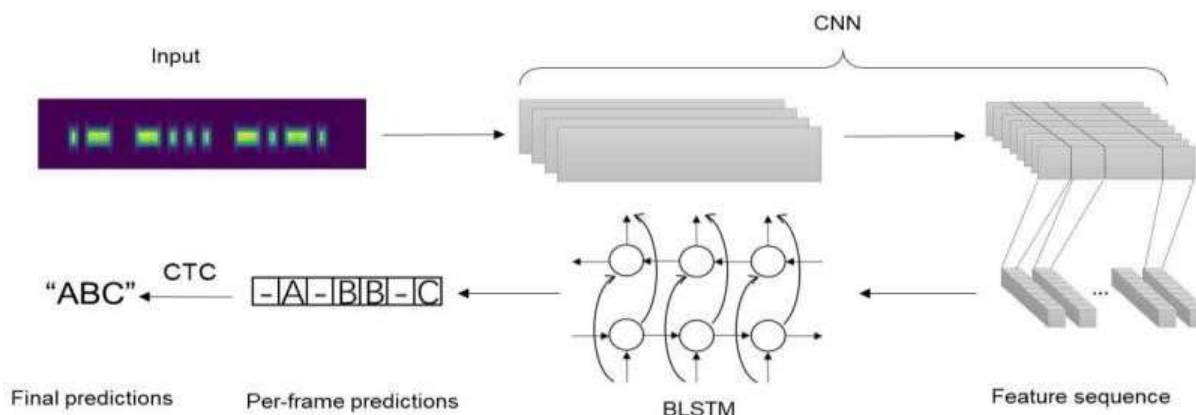
Figure 1: Flowchart of Existing System

In this structure combining CNN and RNN is called CRNN. We build a neural network model based on the idea of CRNN, in which the Bidirectional Long Short-Term Memory (BLSTM) network is used to label feature frames from CNN, and CTC is used as the integration layer. Hence CNN can sensitively capture the position of dot and dash. In the case of frequency drift, CNN can fully extract relevant information and transmit them to the subsequent BLSTM layer, which is beneficial for BLSTM to model timing information, thus improving the recognition performance of the whole network.

Limitations

- If the input signals have high frequency (>700HZ), the model will not perform well.

- If the length of the input signal is more than 4s the model will unable to identify input signal.

- The CTC process is very time consuming to annotate a dataset on character level.

- The BLSTM require more memory and take longer time to train.

- In the RNN, if we are using activation functions, then it becomes tedious to process long sequences.

2.2 PROPOSED SYSTEM

Detection of text from images of morse code is a complex process and there is no active research in this area. As the first step we need to build the dataset containing all the 36 characters including 26 alphabets and 10 digits. This dataset will include images with varying backgrounds, lighting conditions, and noise levels. The dataset consists a total of 7,200 images in which each character containing 200 images each. Then split the dataset into three parts as training, testing and validation. In the splitting of data, the training dataset must have 60% of the total data, validation dataset must have 20% of data and the testing dataset should have 10% of the data. Preprocess the images before feeding them into the CNN.

This includes normalization, resizing, and cropping. Then build a machine learning model Ex: Sequential. Choose an appropriate CNN architecture for the project. Train the CNN model on the training set and evaluate it on the validation set. Optimize the hyperparameters and the architecture of the model. After successfully detecting the individual characters of the morse code. Evaluate the performance of the model on the test set using appropriate metrics, such as accuracy, precision, recall, and F1 score. Now segment he input image based on word segmentation and character segmentation. After segmenting the image, there are individual character segments present, first predict individual characters and the combine these predicted outputs based on words. Hence after integrating all the results the English text is obtained, which is the required outcome.

The test images can have paragraphs and multiple sentences, so in order to show the difference between the characters and words, use the '/' symbol between the words to vary multiple words present in the sentence and use ' '(space) to vary multiple characters present in the words. Then the predictions of each character present in each word will be obtained, so as the words present in each sentence. After getting all the outputs combine the character outputs to form a word and combine the words sequentially to form a sentence.

In this way, for the input images which are having Morse Code present in them, can successfully detect the Morse Code present in them and by using technique image classification followed by sequential object detection converted the Morse Code encoded in images to English language. For this system the algorithms used are LeNet and AlexNet, the LeNet performed with an accuracy of

2%, but the AlexNet outperformed LeNet by giving an accuracy of 95%. AlexNet have additional layers such as Dropout layer and Batch Normalization layer which boost the performance of the model. After testing the model with various epochs like 5, 10, 15, 20 & 50 we concluded that AlexNet is performing much better when compared to LeNet.

Advantages:

- Irrespective of the size of the image, this model will detect the Morse Code present in the input image and convert the Morse Code into English Language.

- The Recurrent Neural Networks are used for the continuous detection of the characters present in the input images.

- The model will detect the Morse Code even if the input image have variety of backgrounds.

- The dataset have the capability to get adaptable for the various Convolutional Neural Network architectures.

- Even if the pixel rate of the images are low, the model will work fine and will convert the image of Morse Code into English Language.

- Once the model is trained, it can be easily scaled to process large amounts of data, which makes it suitable for applications that require processing of large datasets.

2.3 DATASET

We build the dataset on our own, as there is no active research in this area. This dataset consists of 36 characters of Morse Code, each character is having 200 images each. Hence a total of 7,200 images are present in the dataset. A Morse code dataset is a collection of examples of Morse code messages along with their corresponding text translations. Morse code is a system of transmitting messages using a series of dots and dashes to represent letters and numbers. Morse code was widely used in early communication systems, such as telegraphs and radios, and it is still used today in some contexts, such as amateur radio.

English Text	Morse Code	English Text	Morse Code
A	• —	S	• • •
B	— • • •	T	—
C	— • — •	U	• • —
D	— • •	V	• • • —
E	•	W	• — —
F	• • — •	X	— • • —
G	— — •	Y	— • — —
H	• • • •	Z	— — • •
I	• •	0	— — — — —
J	• — — —	1	• — — — —
K	— • —	2	• • — — —
L	• — • •	3	• • • — —
M	— —	4	• • • • —
N	— •	5	• • • • •
O	— — —	6	— • • • •
P	• — — •	7	— — • • •
Q	— — • —	8	— — — • •
R	• — •	9	— — — — •

Table 1: Morse Code Characters

These are the characters of the Morse Code, as you can observe the above figure, there are 36 characters present in the Morse Code. All these characters consist of dots and dashes. A Morse code dataset can be used for various applications, such as developing algorithms for decoding Morse code messages, developing machine learning models for recognizing Morse code signals, or creating Morse code training programs.

The dataset can be created by collecting Morse code messages from various sources, such as telegraph recordings or amateur radio transmissions, and manually transcribing them into text. Morse code dataset is a useful resource for researchers and developers working on various algorithms and models.
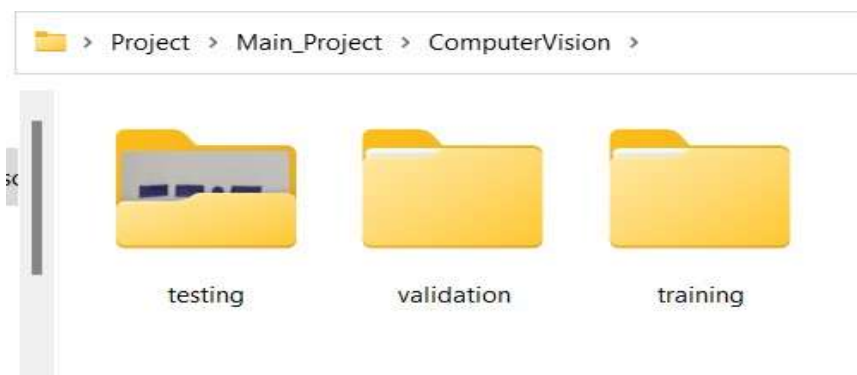
Figure 2: Spitting of Data

At first the entire dataset is divided into three parts those are : training, validation and testing. These splitting of data is performed as training dataset must consists of 70% of the data, validation dataset must consist of 20% of the data and testing dataset should consists of 10% of the entire dataset.
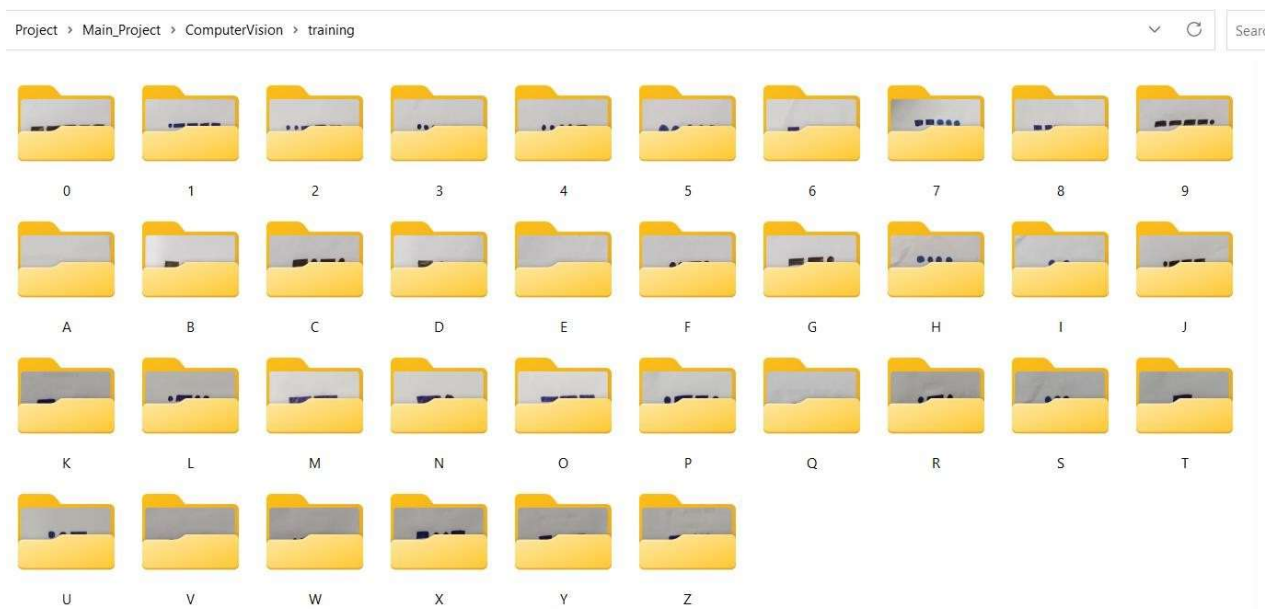
Figure 3: Training Dataset

These are all the characters of the training dataset. As you can observe in the above figure there are a total of 36 character of the morse code present in the dataset. Each character consists of 200 images, so a total of 7,200 images are present in the training dataset. The total size of the training dataset is 143 MB.

2.4 MACHINE LEARNING

Machine learning (ML) is a subset of artificial intelligence (AI) that focuses on developing algorithms and statistical models to enable computers to learn and improve from experience without being explicitly programmed. In other words, it is the process of teaching machines to recognize patterns in data and make decisions based on that knowledge. Machine learning has rapidly become one of the most important and rapidly evolving areas of computer science and technology in general, with applications in a wide range of fields, from healthcare and finance to transportation and entertainment.

One of the key advantages of machine learning is its ability to process and analyze large volumes of data much faster and more accurately than humans can. This enables machines to make predictions and decisions with a high level of accuracy, making them particularly useful for tasks such as image and speech recognition, natural language processing, and fraud detection. In addition, machine learning can be used to automate repetitive tasks and improve efficiency in various industries, leading to cost savings and increased productivity.

One of the most widely used techniques in machine learning is deep learning, which involves the use of neural networks to model complex relationships within data. Neural networks are essentially a series of algorithms that mimic the functioning of the human brain, enabling machines to learn from and make predictions based on vast amounts of data. Deep learning has been particularly successful in image and speech recognition, with applications in fields such as self-driving cars, medical diagnosis, and social media.

However, despite the many advantages of machine learning, there are also significant challenges and limitations. One of the biggest challenges is the need for high-quality data to train the algorithms, as the accuracy of the results depends heavily on the quality and quantity of the data used for training.

In addition, there is a risk of bias and discrimination in machine learning algorithms, particularly when the data used for training is not diverse or representative of the population. Furthermore, the complexity of machine learning models can make them difficult to understand and interpret, raising concerns about transparency and accountability.

2.5 DEEP LEARNING

Deep learning is a subset of machine learning that involves training artificial neural networks to recognize patterns and make predictions based on data. It is inspired by the structure and function of the human brain, which consists of interconnected neurons that process information in parallel. In deep learning, large amounts of data are fed into a neural network, which consists of multiple layers of interconnected nodes. Each layer extracts features from the data and passes them on to the next layer, until a final output is produced. The network learns by adjusting the weights and biases of the connections between nodes in response to feedback on its predictions.

One of the key advantages of deep learning is its ability to automatically learn hierarchical representations of data. This means that the network can learn to recognize complex patterns in the data, even if they are not explicitly defined by the programmer. For example, a deep learning algorithm can learn to recognize a face by automatically identifying the edges and contours of facial features at lower layers and combining them into more complex features such as eyes, nose, and mouth at higher layers. Deep learning has found a wide range of applications in fields such as computer vision, natural language processing, speech recognition, and robotics.

In computer vision, deep learning algorithms have achieved state-of-the-art performance in tasks such as image classification, object detection, and segmentation. In natural language processing, deep learning has been used to build language models that can generate coherent text, translate between languages, and perform question-answering tasks. In speech recognition, deep learning has been used to improve the accuracy of automatic speech recognition systems. In robotics, deep learning has been used to enable robots to learn from their environments and perform tasks such as object manipulation and navigation. Despite its many successes, deep learning also has some limitations.

One of the main challenges is the need for large amounts of labeled data to train the networks. This can be a bottleneck in many applications, particularly in domains where labeled data is scarce or expensive to obtain. Another challenge is the interpretability of deep learning models, which can be difficult to understand and debug due to their high complexity. Nevertheless, deep learning continues to be a rapidly growing field of research, with many exciting developments on the horizon. One of the key challenges in deep learning is the risk of overfitting, which occurs when

a model becomes too complex and starts to memorize the training data rather than learning generalizable patterns.

2.5.1 Convolutional Neural Network

A Convolutional Neural Network (CNN) is a type of deep neural network commonly used in image and video processing applications. CNNs are particularly well-suited to these tasks because they can automatically learn hierarchical representations of the data, starting with simple features such as edges and textures, and building up to more complex representations such as objects and scenes. A CNN consists of several layers, each of which performs a specific type of computation on the input data. The first layer is typically a convolutional layer, which applies a set of filters to the input image to extract features such as edges, corners, and textures. The output of the convolutional layer is then passed through a pooling layer, which reduces the spatial dimensions of the feature map by performing operations such as max pooling or average pooling. This helps to reduce the number of parameters in the model and makes it less susceptible to overfitting.
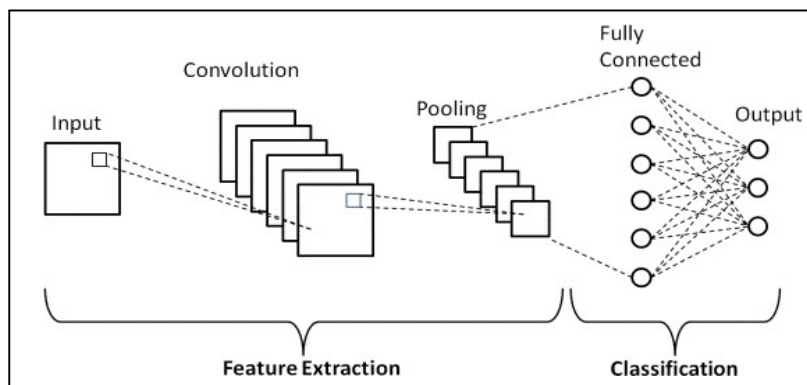
Figure 4: CNN Architecture

After several convolutional and pooling layers, the output is typically flattened and passed through one or more fully connected layers, which perform a classification or regression task based on the extracted features. The final layer of the network is often a softmax layer, which produces a probability distribution over the possible classes or outputs. One of the key advantages of CNNs is their ability to learn spatially invariant features, meaning that they can recognize objects regardless of their position, rotation, or scale in the input image. This is achieved by sharing the weights of the filters across the spatial dimensions of the input, which allows the network to learn

features that are applicable to different parts of the image. CNNs have been used for a wide range of image and video processing tasks, such as image classification, object detection, segmentation, and style transfer. They have achieved state-of-the-art performance on benchmark datasets such as ImageNet and have been deployed in real-world applications such as self-driving cars, medical imaging, and surveillance systems.

Despite their many successes, CNNs also have some limitations. One of the main challenges is the need for large amounts of labeled data to train the network, which can be a bottleneck in many applications. Another challenge is the interpretability of the learned features, which can be difficult to understand and visualize due to the high dimensionality of the feature maps. Nevertheless, CNNs continue to be a widely used and important tool in the field of deep learning. The layers of the CNN are:

- **Convolutional Layer:** A convolutional layer is one of the fundamental building blocks of a Convolutional Neural Network (CNN). It is designed to extract features from input data, such as images, by applying a set of learnable filters to the input data. With this layer we can extract the important data present in the input image.

- **Dropout Layer:** Dropout is a regularization technique commonly used in deep learning models, including Convolutional Neural Networks (CNNs), to prevent overfitting and improve generalization performance. The dropout layer is a type of layer that implements this technique.

- **Pooling Layer:** A pooling layer is a type of layer commonly used in Convolutional Neural Networks (CNNs) to reduce the spatial dimensions of the feature map generated by the convolutional layer. Pooling layers help to reduce the number of parameters in the network, making it less susceptible to overfitting, while also providing some degree of translation invariance. There are three types in pooling they are: Max Pooling, Average Pooling and Min Pooling.

- **Batch Normalization Layer:** The batch normalization layer operates on a mini batch of input data, typically of size 32, 64, or 128. During training, it normalizes the input activations of the previous layer by subtracting the batch mean and dividing by the batch standard deviation. The resulting activations are then scaled and shifted using learnable parameters (gamma and beta) to provide the network with the flexibility to adjust the normalized activations as needed.

- **Flatten Layer:** The flatten layer simply reshapes the output tensor of the previous layer into a

single dimension. For example, if the output tensor has dimensions 32 x 32 x 64, the flatten layer would reshape it into a vector of length 64 x 32 x 32 = 65,536. This flattened vector can then be fed into a fully connected layer for classification or regression.

- **Fully Connected Layer:** A fully connected layer, also known as a dense layer, is a type of layer commonly used in Convolutional Neural Networks (CNNs) that helps to map the features learned by the convolutional layers to the output classes.

2.5.2 LeNet-5

LeNet-5 is a convolutional neural network (CNN) architecture that was developed by Yann LeCun, Leon Bottou, Yoshua Bengio, and Patrick Haffner in 1998. It was one of the first successful convolutional neural networks and was designed specifically for handwritten digit recognition. The LeNet-5 architecture consists of seven layers: two convolutional layers, two subsampling layers, and three fully connected layers. The first convolutional layer uses 6 filters of size 5x5, and the second convolutional layer uses 16 filters of size 5x5. The subsampling layers use average pooling with a 2x2 filter size.

The fully connected layers have 120, 84, and 10 nodes, respectively, with the last layer being a softmax output layer for classification. The LeNet-5 algorithm has been widely used for digit recognition tasks and has achieved high accuracy rates. It has also served as a foundation for many modern CNN architectures, such as AlexNet and GoogleNet, and has paved the way for the current state-of-the-art deep learning models. Overall, LeNet-5 is a simple yet effective architecture that helped to demonstrate the power of convolutional neural networks for image recognition tasks.
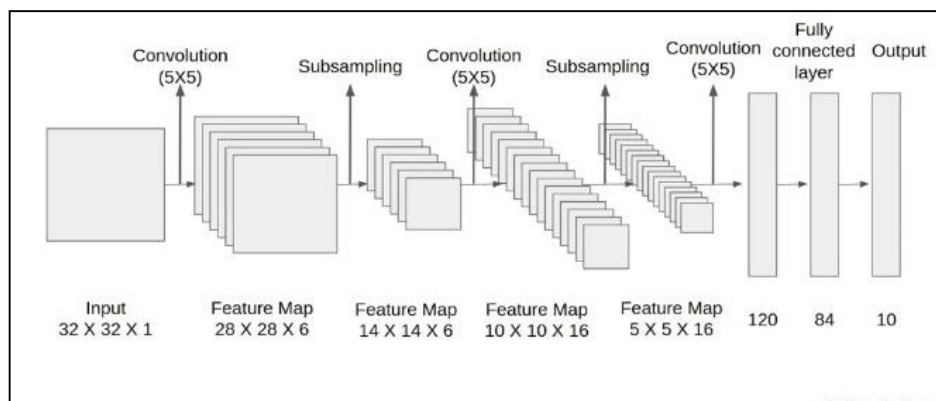
Figure 5: LeNet-5 Architecture

LeNet-5 achieved state-of-the-art performance on handwritten digit recognition tasks, surpassing previous methods such as Support Vector Machines (SVMs) and Multi-Layer Perceptrons (MLPs). On the MNIST dataset, which consists of 60,000 training and 10,000 testing grayscale images of size 28x28, LeNet-5 achieved a test error rate of 0.95%, which was a significant improvement over previous method. Since then, LeNet-5 has been used as a benchmark for many other image recognition tasks and has served as a foundation for many modern CNN architectures.

2.5.3 AlexNet

AlexNet is a convolutional neural network (CNN) architecture designed for image classification. The architecture of AlexNet consists of eight layers, including five convolutional layers, two fully connected layers, and a final output layer. The first layer has 96 filters with a size of 11x11 pixels and a stride of 4, followed by a rectified linear unit (ReLU) activation function and local response normalization.

The second and third layers have 256 filters each with a size of 5x5 pixels and a stride of 1, also followed by ReLU activation and local response normalization. The fourth and fifth layers have 384 and 256 filters, respectively, with a size of 3x3 pixels and a stride of 1. Between the convolutional layers, there are two max pooling layers with a filter size of 3x3 pixels and a stride of 2. These layers help to reduce the spatial dimensionality of the feature maps while preserving important features. The final three layers of AlexNet are fully connected layers.

Figure 6: AlexNet Architecture

The first fully connected layer has 4,096 neurons, followed by a dropout layer to prevent overfitting. The second fully connected layer has 4,096 neurons, followed by another dropout layer. Finally, there is an output layer with 1,000 neurons corresponding to the 1,000 categories in the ImageNet dataset. The output layer uses a softmax activation function to output a probability distribution over the categories.

It is important to note that AlexNet was trained on a sizable dataset of 1.2 million photos, and that the quantity and calibre of the training data have a significant impact on the network's performance. The selection of hyperparameters, including learning rate and batch size, can significantly affect the network's performance. Moreover, little research has been done on AlexNet's performance on tasks except picture categorization. Since its introduction in 2012, AlexNet's architecture and techniques have been refined and improved upon, leading to even better performance on the ILSVRC and other image classification challenges. However, AlexNet remains an important benchmark for image classification algorithms and its contributions to the field of deep learning cannot be overstated.

AlexNet uses data augmentation techniques to artificially increase the size of the training dataset. Specifically, the network applies random transformations to the input images during training, such as cropping, scaling, and horizontal flipping. This helps to prevent overfitting and improve the robustness of the network. AlexNet uses dropout regularization to prevent overfitting. Dropout randomly drops out a fraction of the neurons in the network during each training iteration, forcing the remaining neurons to learn more robust representations.

AlexNet uses rectified linear unit (ReLU) activation in CNN. Specifically, the network applies random transformations to the input images during training, such as cropping, scaling, and horizontal flipping. This helps to prevent overfitting and improve the robustness of the network. AlexNet uses dropout regularization to prevent overfitting. Dropout randomly drops out a fraction of the neurons in the network during each training iteration, forcing the remaining neurons to learn more robust representations. AlexNet uses rectified linear unit (ReLU) activation in CNN

3. <u>SYSTEM ANALYSIS</u>

3.1 OVERVIEW OF SYSTEM ANALYSIS

System analysis is an important step in any project to ensure that the system is designed and implemented to meet the requirements and achieve the desired performance. The problem addressed by the project is to detect and recognize text in Morse code from images using CNN. This involves several subtasks, such as preprocessing the images, segmenting the Morse code regions, decoding the Morse code signals, and recognizing the decoded text. The requirements for the system include the ability to accurately detect and recognize Morse code in images, handle different types of images with varying levels of noise, and process the images efficiently.

The system should also be able to handle different types of Morse code signals and recognize text in different languages. The system architecture for the project would include several components, such as image preprocessing, Morse code signal extraction, signal decoding, and text recognition. The image preprocessing component would be responsible for enhancing the quality of the input images, removing noise and artifacts, and segmenting the Morse code regions. The Morse code signal extraction component would extract the Morse code signals from the segmented regions and prepare them for decoding. The signal decoding component would decode the Morse code signals into text, and the text recognition component would recognize the text using CNN.

The project would require a dataset of images containing Morse code signals in different languages and different levels of noise. The dataset should be large enough to train and validate the CNN model effectively. The dataset would need to be preprocessed to segment the Morse code regions and extract the signals for training and testing. The CNN model would be trained and validated using the preprocessed dataset. The training process would involve optimizing the model parameters to minimize the loss function and improve the accuracy of the model.

The validation process would involve testing the trained model on a separate set of data to evaluate its performance and prevent overfitting. The system would be implemented using the trained CNN model and the various components described above. The system would be

tested on a set of images containing Morse code signals to evaluate its accuracy and performance. The system would need to be optimized for speed and efficiency to ensure that it can process images in real-time or near-real-time.

3.1.1 Requisites Accumulating and Analysis

The project would require a computer with a powerful CPU and GPU to train the CNN model and process images efficiently. The computer should have enough memory and storage to handle the large dataset of images and the trained model, and also require software tools for image processing, signal extraction, signal decoding, and text recognition. These tools could include Python libraries such as OpenCV, NumPy, and TensorFlow, as well as other software tools for data analysis and visualization.

The project would require a dataset of images containing Morse code signals in different languages and different levels of noise. The dataset should be large enough to train and validate the CNN model effectively. The dataset should be diverse and include various types of images with different resolutions, brightness, and contrast. The project would require the use of preprocessing techniques to enhance the quality of the input images, remove noise and artifacts, and segment the Morse code regions. The preprocessing techniques could include filters, thresholding, and morphological operations. The project would require text recognition techniques to recognize the decoded text using CNN. The text recognition techniques could include convolutional neural networks, recurrent neural networks, and deep learning algorithms.

3.1.2 System Design

System design involves the identification and specification of the system components, their interactions, and the design approach. The first component of the system design is the data collection and preparation. the system design for the project "Detection of text from Morse code in images using CNN" involves identifying and specifying the system components, their interactions, and the design approach. The system design includes data collection and preparation, signal extraction and decoding, CNN model architecture, training and validation, testing and evaluation, and user interface. By following this system design, the project can be

implemented to achieve the desired performance and meet the requirements of the project.

3.1.3 Implementation

These implementation steps are general guidelines and may vary depending on the specific requirements and specifications of the project. It is essential to follow a structured and organized approach to ensure that the system is developed efficiently and effectively and meets the project's requirements and specifications. Design the CNN model architecture for the detection of text from Morse code signals. The model should consist of convolutional layers, pooling layers, and fully connected layers, along with activation and normalization layers.

3.1.4 Testing

Testing involves evaluating the performance of the CNN model on a separate testing dataset to determine its accuracy and effectiveness in detecting text from Morse code signals. Prepare a separate dataset of Morse code images and corresponding text labels for testing. This dataset should be separate from the dataset used for training and validation to ensure an unbiased evaluation of the model's performance. Load the trained CNN model that was obtained in the training and validation stage. Use the trained CNN model to predict the text labels for the images in the testing dataset. This step involves feeding the testing images into the CNN model and obtaining the corresponding text predictions. Evaluate the performance of the CNN model using various metrics such as accuracy, precision, recall, and F1 score. These metrics provide a quantitative measure of the model's performance in detecting text from Morse code signals.

3.1.5 Deployment of System

The deployment stage involves making the system available for use in a production environment. Package the system, including the trained CNN model, any necessary libraries or dependencies, and the user interface, into a deployable format, such as a container or an executable file. Deploy the packaged system to the production environment. This can be done either on-premises or in the cloud, depending on the project's requirements and specifications. Configure the system to meet the specific needs of the production environment, including network connectivity, security settings, and any other necessary configurations. Test the

deployed system to ensure that it is functioning as expected and that it is providing accurate and reliable results.

3.1.6 Maintenance

The maintenance phase involves ensuring that the system continues to function properly after deployment. Monitor the system regularly to detect any issues that may arise. Address any issues that arise in the system. This may involve debugging the code, fixing any issues with the system configurations, or applying any necessary updates to the system. Apply any necessary updates to the system to ensure that it remains up to date with any new technologies, software, or security patches. Maintain proper documentation of the system, including any updates or modifications that are made, as well as any issues that are addressed. This will help ensure that future maintenance and updates can be made more efficiently and effectively.

3.2   SOFTWARE(S) USED IN THE PROJECT

3.2.1   Python

Python is a high-level, interpreted programming language that was created by Guido van Rossum in the late 1980s. It is a general-purpose language that is widely used in various fields, including web development, data analysis, scientific computing, artificial intelligence, and more. Python is also a cross-platform language, which means that it can be run on multiple operating systems, including Windows, macOS, and Linux. This makes it a popular choice for developers who want to develop applications that can run on different platforms without having to rewrite the code. Another key feature of Python is its extensive library support. There are numerous libraries available for Python that can be used to extend its capabilities and make it more powerful. These libraries can be used to perform tasks such as data analysis, web development, image processing, and more. Python can be used along with machine learning, here are some of the ways:

- Before applying machine learning algorithms, the data must be prepared and cleaned. Python offers powerful libraries such as NumPy, Pandas, and SciPy that can be used for data preparation tasks like cleaning, transformation, and feature selection.
- Python offers several libraries for implementing machine learning algorithms, such as

Scikit-learn, TensorFlow, and PyTorch. These libraries provide a wide range of machine learning algorithms, including regression, classification, clustering, and deep learning.

- Python has powerful visualization libraries like Matplotlib, Seaborn, and Plotly that can be used to visualize and analyze data. Data visualization helps to understand the data better and can be useful in identifying trends and patterns.

- Python can also be used to deploy machine learning models. For example, Flask and Django are popular web frameworks for deploying machine learning models as web applications. Python libraries like Keras and TensorFlow can also be used to deploy machine learning models on mobile and IoT devices.

- Python offers libraries such as Auto-sklearn and TPOT that can automate the process of selecting and tuning machine learning models. This can help to reduce the time and effort required for developing machine learning models.

3.2.2 Jupyter Notebook

Jupyter Notebook is an open-source web application that allows users to create and share documents that contain live code, equations, visualizations, and narrative text. It is widely used in data science, scientific research, and education. It provides an interactive computing environment that supports many programming languages, including Python, R, Julia, and more.

It allows users to write and execute code directly in the notebook, as well as create and edit text cells that can contain explanations, documentation, and other forms of narrative content. Jupyter Notebook also supports the creation of interactive visualizations and widgets, which can be used to create dynamic and engaging user interfaces. It also allows for the integration of external libraries and tools, such as NumPy, Pandas, Matplotlib, and more.
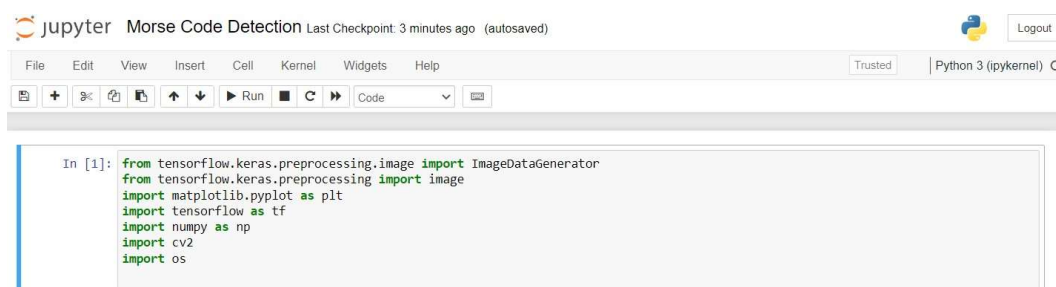
Figure 7: Jupyter Notebook environment

In addition to its use as a development environment, Jupyter Notebook also enables collaboration and sharing. Notebooks can be easily shared with others, allowing for collaboration and feedback on code and data analysis. The notebooks can also be exported in various formats, such as HTML, PDF, and Markdown. One of the main advantages of Jupyter Notebook is its ability to display the output of code cells in line with the code itself. This makes it easy to visualize and understand the results of code execution, which can be especially useful in data analysis and scientific research. Overall, Jupyter Notebook is a powerful tool for data analysis, scientific research, and education.

3.2.3 Google Colab

Google Colab, short for Google Collaboratory, is a cloud-based service provided by Google that enables users to write, run, and share Python code and machine learning models in an interactive notebook environment. Colab notebooks are Jupyter notebooks that are hosted on Google's servers and can be accessed and shared by anyone with a Google account. Google Colab provides many useful features for machine learning practitioners, including free access to powerful GPUs and TPUs for training models, pre-installed libraries for machine learning and data science, and seamless integration with other Google services like Google Drive and Google Sheets.
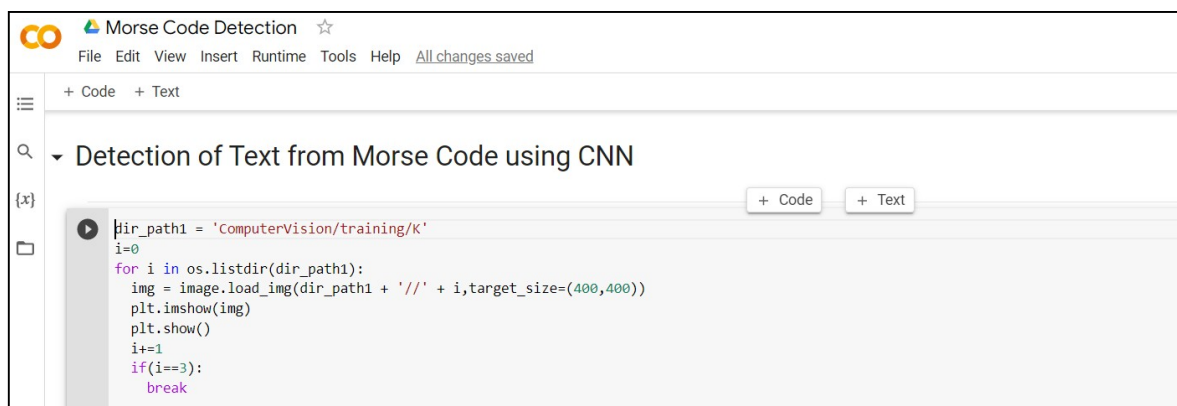
Figure 8: Google Colab Environment

One of the most significant advantages of using Google Colab is the ability to easily share and collaborate on notebooks. Users can share their notebooks with others and work on them simultaneously, making it easier to collaborate on projects and share ideas. Additionally,

Google Colab provides easy integration with other popular tools and services, such as GitHub and TensorBoard, making it easier for users to work with their existing workflows and tools. Overall, Google Colab is a useful tool for data scientists and machine learning practitioners, providing an easy-to-use, collaborative environment for developing and sharing code and models.

3.2.4 Packages

NumPy: NumPy is a Python package for numerical computation that provides a powerful array object and functions for working with arrays. It is widely used in scientific computing, data analysis, and machine learning. One of the key features of NumPy is its ndarray object, which is a multidimensional array that can hold elements of various data types. The ndarray object provides efficient operations for manipulating and performing mathematical operations on arrays, making it an essential tool for scientific computing and data analysis.

NumPy provides a wide range of functions for performing mathematical operations on arrays, including basic arithmetic operations, linear algebra, and statistical operations. These functions are optimized for speed and efficiency, making them suitable for large-scale data processing tasks. This makes it easy to work with data from various sources and integrate it into NumPy arrays for processing and analysis.

Matplotlib: Matplotlib is a Python library for creating static, animated, and interactive visualizations in Python. It is widely used for data visualization in scientific computing, data analysis, and machine learning. Matplotlib provides a wide range of customizable plots and charts, including line plots, scatter plots, bar plots, histograms, and more. It also provides tools for customizing the appearance of plots, including fonts, colors, and labels. One of the key features of Matplotlib is its integration with NumPy, a Python library for numerical computation.

Matplotlib can directly plot data stored in NumPy arrays, making it easy to visualize data from scientific computing and data analysis. Matplotlib also provides support for creating animations and interactive plots. These features are useful for visualizing dynamic data or exploring data interactively. Matplotlib supports several animations and interactive backends, including the HTML5 canvas, GTK, Qt, and more.

TensorFlow: TensorFlow is an open-source machine learning framework developed by Google. It is widely used for building and training machine learning models, including deep neural networks, for a wide range of applications, such as computer vision, natural language processing, and speech recognition. One of the key features of TensorFlow is its ability to create and manipulate tensors, which are multidimensional arrays used for storing and processing data in machine learning models. TensorFlow provides a powerful set of functions for manipulating tensors and performing mathematical operations on them, making it suitable for large-scale data processing tasks. TensorFlow also provides a wide range of pre-built machine learning models, such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs), which can be used as building blocks for more complex models. It also allows developers to build custom models using its high-level Keras API or its lower-level TensorFlow API.

TensorFlow also provides a suite of tools for visualizing and debugging machine learning models, including TensorBoard, which allows developers to visualize the training process and monitor the performance of models in real-time. Another important feature of TensorFlow is its ability to distribute computations across multiple devices, such as CPUs and GPUs, and multiple machines. This enables developers to scale up the training of machine learning models to handle large datasets and complex models. Overall, TensorFlow is a powerful machine learning framework that provides a wide range of tools for building and training machine learning models. Its ability to handle large datasets, distribute computations, and provide visualization and debugging tools make it an essential tool for machine learning researchers and developers.

CV2: OpenCV (Open Source Computer Vision) is a popular open-source library for computer vision, image processing, and machine learning applications. It is written in C++ and has bindings for Python and other languages. OpenCV provides a wide range of functions for image and video processing, including loading and saving images, image filtering, feature detection, object recognition, and camera calibration. It also provides support for working with video streams, including capturing video from cameras and processing video frames in real-time. One of the key features of OpenCV is its ability to work with multiple platforms and devices, including desktop computers, mobile devices, and embedded systems. It also provides a range of pre-trained models for object detection and recognition, such as the Haar cascades for detecting

faces and eyes. OpenCV also has a large and active community that has developed many add-on packages and tools. For example, OpenCV-Python is a popular Python wrapper for OpenCV that provides an easy-to-use interface for developing computer vision applications in Python.

OS: The 'os' module is a standard library in Python that provides a way to interact with the operating system. It provides a set of functions for working with files and directories, managing processes, and accessing system information. It is a powerful and versatile library that provides a wide range of functionality for working with the operating system.

3.3 SYSTEM REQUIREMENTS

3.3.1 Software Requirements

- **Operating System**   :   Windows 10th Gen
- **Languages**   :   Python
- **IDE**   :   Jupyter Notebook
- **Dataset**   :   Morse Code Dataset

3.3.2 Hardware Requirements

- **RAM**   :   Min. 8 GB
- **Processor**   :   Intel core i5
- **Memory**   :   512GB HDD/ SSD

3.3.3 Packages Required

- NumPy
- Matplotlib
- TensorFlow
- CV2
- OS

4. <u>SYSTEM DESIGN</u>

4.1 OVERVIEW OF SYSTEM DESIGN

There is no ongoing study in the domain of text detection from photographs of morse code since it is a challenging task. Building a dataset with all 36 characters, including 26 alphabets and 10 numerals, is the first stage. Images in this collection will have a range of backgrounds, lighting, and noise levels. The total number of photos in the collection is 7,200, with 200 images per character. The dataset was then divided into training, testing, and validation portions. The training dataset must contain 60% of the total data, the validation dataset should contain 20%, and the testing dataset should contain 10%. Before sending the photographs to CNN, preprocess them.
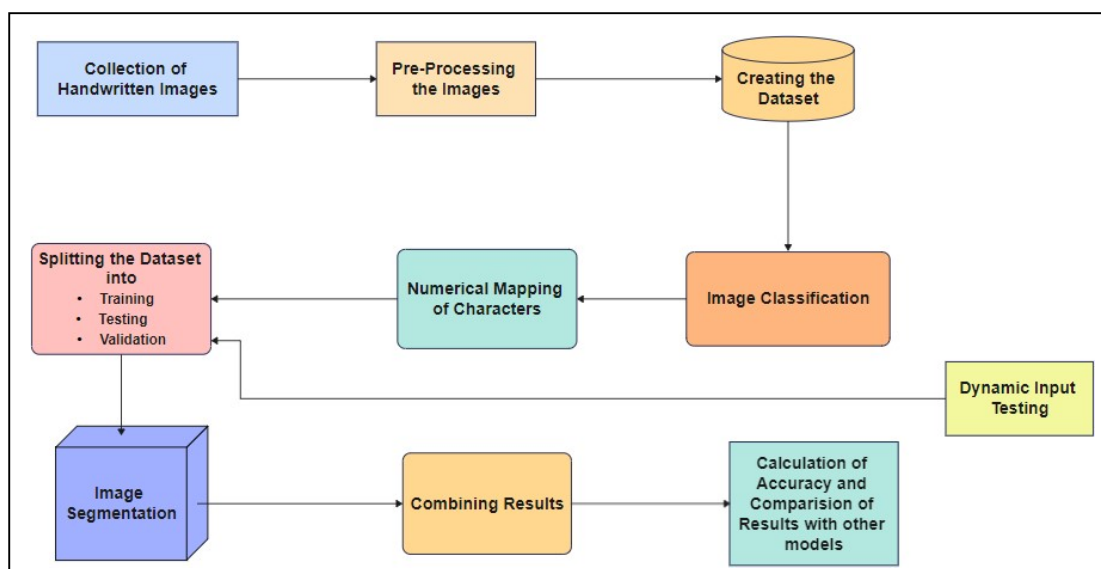
Figure 9: System Architecture

This covers cropping, resizing, and normalizing. Then create a Sequential machine learning model. Choose a CNN architecture that is suited for the purpose. Using the training set to build the CNN model, and the validation set to assess it. Improve the model's design and hyperparameters. once the morse code's individual characters have been identified. Use relevant metrics to assess the model's performance on the test set, such as accuracy, precision, recall, and F1 score. He will now divide the image he entered words and characters. Following picture segmentation, individual character segments are visible. Predict each character separately, then aggregate these anticipated outputs based on words. Thus, the English text, which is the final product of merging all the results.

The test images can have paragraphs and multiple sentences, so in order to show the difference between the characters and words, use the '/' symbol between the words to vary multiple words present in the sentence and use ' '(space) to vary multiple characters present in the words. Then the predictions of each character present in each word will be obtained, so as the words present in each sentence. After getting all the outputs combine the character outputs to form a word and combine the words sequentially to form a sentence.

In this way, for the input images which are having Morse Code present in them, can successfully detect the Morse Code present in them and by using technique image classification followed by sequential object detection converted the Morse Code encoded in images to English language. For this system the algorithms used are LeNet and AlexNet, the LeNet performed with an accuracy of 2%, but the AlexNet outperformed LeNet by giving an accuracy of 95%. AlexNet have additional layers such as Dropout layer and Batch Normalization layer which boost the performance of the model. After testing the model with various epochs like 5, 10, 15, 20 & 50 we concluded that AlexNet is performing much better when compared to LeNet.

4.2 SPLITTING THE DATASET

Due to the lack of ongoing study in this field, we created the dataset on our own. There are 200 photos for each of the 36 Morse Code characters in this collection. Hence, the collection has a total of 7,200 photos.
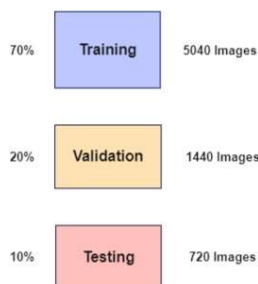
Figure 10: Splitting of Data

A collection of instances of Morse code messages and the text translations that go with them makes up a Morse code dataset. Using a succession of dots and dashes to represent letters and numbers, Morse code is a method of message transmission. Early communication technologies like telegraphs and radios made extensive use of morse code, and it is still utilized in some settings today, including amateur radio. We have spitted the entire dataset into three parts those are:

Training Dataset, Validation Dataset and Testing Dataset. Where the training dataset will have 60% of the data, validation with 20% and testing with 10% of the dataset.

4.3 CLASSIFICATION ALGORITHMS

4.3.1 LeNet-5

With a focus on handwritten digit identification, it was one of the first effective convolutional neural networks. Two convolutional, two subsampling, and three fully linked layers make up the seven-layer LeNet-5 architecture. Six filters of size 5x5 each are used in the first and second convolutional layers, respectively. With a 2x2 filter size, the subsampling layers employ average pooling. The fully connected layers have 120, 84, and 10 nodes, respectively, with the last layer being a SoftMax output layer for classification.

The LeNet-5 algorithm has been widely used for digit recognition tasks and has achieved high accuracy rates. It has also served as a foundation for many modern CNN architectures, such as AlexNet and GoogleNet, and has paved the way for the current state-of-the-art deep learning models. With a focus on handwritten digit identification, it was one of the first effective convolutional neural networks. LeNet convolutional neural networks are the first neural networks that was used for the image classification. LeNet-5 consists of 5 layers out of which there are two convolutional, two subsampling, and three fully linked layers make up the seven-layer LeNet-5 architecture.
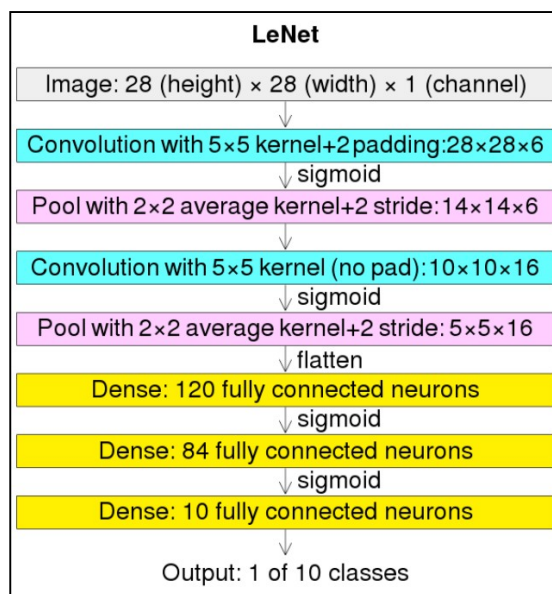
Figure 11: Layers in Working of LeNet-5 Architecture

The architecture of LeNet-5 consists of seven layers, including two convolutional layers, two subsampling layers, and three fully connected layers. Here's a breakdown of what each layer does: The input layer receives an input image of size 32x32 pixels. The first convolutional layer applies six filters to the input image, each with a size of 5x5 pixels. This results in six 28x28 feature maps. The first subsampling layer performs average p ooling with a filter size of 2x2 pixels, reducing the spatial dimensions of the feature maps by a factor of 2. This results in six 14x14 feature maps. The second convolutional layer applies 16 filters to the output of the first subsampling layer, each with a size of 5x5 pixels. This results in 16 10x10 feature maps. The second subsampling layer performs average pooling with a filter size of 2x2 pixels, reducing the spatial dimensions of the feature maps by a factor of 2. This results in 16 5x5 feature maps.

The output of the second subsampling layer is flattened into a 1D vector and passed through a fully connected layer with 120 nodes. This layer applies a non-linear activation function, typically a hyperbolic tangent (tanh) or a Rectified Linear Unit (ReLU). The output of the first fully connected layer is then passed through another fully connected layer with 84 nodes, which also applies a non-linear activation function. The final layer in the network is a fully connected layer with 10 nodes, one for each possible digit class (0-9). This layer uses a softmax activation function to produce a probability distribution over the possible classes.

Working of LeNet-5 Algorithm:

The following steps will help us understand about the LeNet-5 algorithm clearly.

Step 1: The algorithm takes an input image of size 32x32 pixels. The image is typically grayscale.

Step 2: The first layer in the network is a convolutional layer that applies six filters to the input image. Each filter has a size of 5x5 pixels and produces a feature map of the same size as the input image. The output of this layer is a set of six 28x28 feature maps.

Step 3: The output of the first convolutional layer is then passed through a subsampling layer that applies average pooling with a filter size of 2x2 pixels. This reduces the spatial dimensions of the feature maps by a factor of 2, resulting in six 14x14 feature maps.

Step 4: The second convolutional layer applies 16 filters to the output of the first subsampling layer. Each filter has a size of 5x5 pixels and produces a feature map of size 10x10 pixels. The output of this layer is a set of 16 10x10 feature maps.

Step 5: The output of the second convolutional layer is then passed through another subsampling layer that applies average pooling with a filter size of 2x2 pixels. This reduces the spatial dimensions of the feature maps by a factor of 2, resulting in 16 5x5 feature maps.

Step 6: The output of the second subsampling layer is flattened into a 1D vector and passed through a fully connected layer with 120 nodes. This layer applies a non-linear activation function, typically a hyperbolic tangent (tanh) or a Rectified Linear Unit (ReLU).

Step 7: The output of the first fully connected layer is then passed through another fully connected layer with 84 nodes, which also applies a non-linear activation function.

Step 8: The final layer in the network is a fully connected layer with 10 nodes, one for each possible digit class (0-9). This layer uses a softmax activation function to produce a probability distribution over the possible classes.

Step 9: The weights and biases of the network are trained using backpropagation with stochastic gradient descent (SGD) or another optimization algorithm. The objective is to minimize a loss function that measures the difference between the predicted probabilities and the true labels.

Step 10: During inference, an input image is passed through the trained network, and the output of the final layer is used to make a prediction about the class of the input image.

4.3.2 AlexNet

AlexNet is an image classification-focused convolutional neural network (CNN) architecture. It was created by Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton in 2012, and that year it won the ImageNet Large Scale Visual Recognition Competition, significantly raising the bar for picture classification accuracy. AlexNet's architecture comprises of eight layers, including a final output layer, two fully linked levels, and five convolutional layers. The first layer has 96 filters with a size of 11x11 pixels and a stride of 4, followed by a rectified linear unit (ReLU) activation function and local response normalization.

The second and third layers have 256 filters each with a size of 5x5 pixels and a stride of 1, also followed by ReLU activation and local response normalization. The fourth and fifth layers have 384 and 256 filters, respectively, with a size of 3x3 pixels and a stride of 1. Between the convolutional layers, there are two max pooling layers with a filter size of 3x3 pixels and a stride of 2. These layers help to reduce the spatial dimensionality of the feature

maps while preserving important features. The final three layers of AlexNet are fully connected layers.

The first fully connected layer has 4,096 neurons, followed by a dropout layer to prevent overfitting. The second fully connected layer has 4,096 neurons, followed by another dropout layer. Finally, there is an output layer with 1,000 neurons corresponding to the 1,000 categories in the ImageNet dataset. The output layer uses a SoftMax activation function to output a probability distribution over the categories. AlexNet introduced several innovations that helped to improve the performance of CNNs for image classification, including the use of ReLU activation functions, data augmentation techniques, dropout regularization, and parallelization on multiple GPUs.

The success of AlexNet sparked a renewed interest in deep learning and revolutionized the field of computer vision. AlexNet achieved state-of-the-art performance on the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2012, significantly outperforming all previous algorithms. It achieved a top-5 error rate of 15.3%, which was a significant improvement over the previous state-of-the-art error rate of 26.2%.

AlexNet

Image: 224 (height) × 224 (width) × 3 (channels)

Convolution with 11×11 kernel+4 stride:54×54×96

↓ ReLu

Pool with 3×3 max. kernel+2 stride: 26×26×96

Convolution with 5×5 kernel+2 pad:26×26×256

↓ ReLu

Pool with 3×3 max. kernel+2 stride:12×12×256

Convolution with 3×3 kernel+1 pad:12×12×384

↓ ReLu

Convolution with 3×3 kernel+1 pad:12×12×384

↓ ReLu

Convolution with 3×3 kernel+1 pad:12×12×256

↓ ReLu

Pool with 3×3 max. kernel+2 stride:5×5×256

↓ flatten

Dense: 4096 fully connected neurons

↓ ReLu, dropout p=0.5

Dense: 4096 fully connected neurons

↓ ReLu, dropout p=0.5

Dense: 1000 fully connected neurons

Output: 1 of 1000 classes

Figure 12: Layers in Working of AlexNet Architecture

It is important to note that AlexNet was trained on a sizable dataset of 1.2 million photos, and that the quantity and calibre of the training data have a significant impact on the network's performance. The selection of hyperparameters, including learning rate and batch size, can significantly affect the network's performance. Moreover, little research has been done on AlexNet's performance on tasks except picture categorization. Since its introduction in 2012, AlexNet's architecture and techniques have been refined and improved upon, leading to even better performance on the ILSVRC and other image classification challenges. However, AlexNet remains an important benchmark for image classification algorithms and its contributions to the field of deep learning cannot be overstated.

AlexNet uses data augmentation techniques to artificially increase the size of the training dataset. Specifically, the network applies random transformations to the input images during training, such as cropping, scaling, and horizontal flipping. This helps to prevent overfitting and improve the robustness of the network. AlexNet uses dropout regularization to prevent overfitting. Dropout randomly drops out a fraction of the neurons in the network during each training iteration, forcing the remaining neurons to learn more robust representations. AlexNet uses rectified linear unit (ReLU) activation in CNN.

Working of AlexNet Algorithm:

The following steps will help us understand about the LeNet-5 algorithm clearly.

Step 1: AlexNet takes an input image of size 227x227x3 pixels, where 3 represents the three-color channels: red, green, and blue.

Step 2: The input image is first passed through a series of five convolutional layers, each with a set of learned filters. These filters are applied to the input image to generate a set of feature maps, which capture different aspects of the image.

Step 3: After each convolutional layer, a non-linear activation function called ReLU is applied elementwise to the feature maps. ReLU sets all negative values in the feature maps to zero, allowing the network to learn more complex and discriminative features.

Step 4: After the first and second convolutional layers, a local response normalization step is performed to further improve the robustness of the network to variations in input data.

Step 5: Between the convolutional layers, there are two max pooling layers, each with a filter

size of 3x3 and a stride of 2. These layers help to reduce the spatial dimensionality of the feature maps while preserving important features.

Step 6: The feature maps generated by the convolutional and pooling layers are flattened and passed through two fully connected layers, each with a large number of neurons. These layers learn high-level representations of the input image.

Step 7: After each fully connected layer, a dropout regularization step is performed to prevent overfitting.

Step 8: The final layer of the network is a SoftMax output layer, which computes the probability distribution over the 1000 categories in the ImageNet dataset.

Step 9: The weights of the network are trained using backpropagation and stochastic gradient descent. The loss function used for training is the cross-entropy loss between the predicted and true labels.

Step 10: Once the network is trained, it can be used to classify new images by passing them through the network and computing the output probabilities.

4.4 SYSTEM DESIGN

4.4.1 Sigmoid Activation Function

The sigmoid function is also called a squashing function as its domain is the set of all real numbers, and its range is (0, 1). Hence, if the input to the function is either a very large negative number or a very large positive number, the output is always between 0 and 1. Same goes for any number between -∞ and +∞.
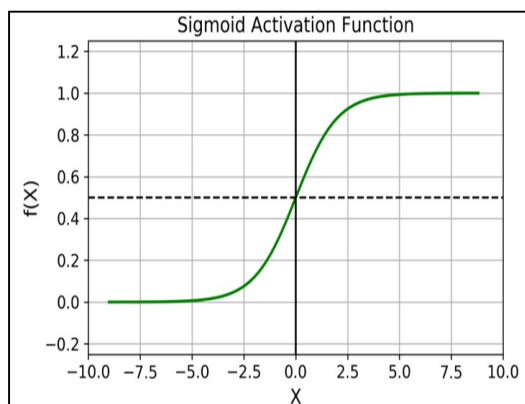
Figure 13: Sigmoid Activation Function

The sigmoid activation function is a common non-linear activation function used in artificial neural networks, including feedforward neural networks and recurrent neural networks. The sigmoid function has a characteristic S-shaped curve, which maps any input value to an output value between 0 and 1. The sigmoid function is defined as:

$$f(x) = 1 / (1 + exp(-x))$$

The sigmoid function is useful in neural networks because it allows neurons to model non-linear relationships between inputs and outputs. For example, a neuron with a sigmoid activation function might learn to output a high value when the input is greater than a certain threshold, but a low value when the input is less than that threshold. However, the sigmoid function has some drawbacks. One is that its gradient (derivative) becomes very small as the input becomes very large or very small, which can cause problems during the training of neural networks. This is known as the "vanishing gradient problem".

4.4.2 ReLU Activation Function

The rectified linear activation function or ReLU for short is a piecewise linear function that will output the input directly if it is positive, otherwise, it will output zero. It has become the default activation function for many types of neural networks because a model that uses it is easier to train and often achieves better performance.
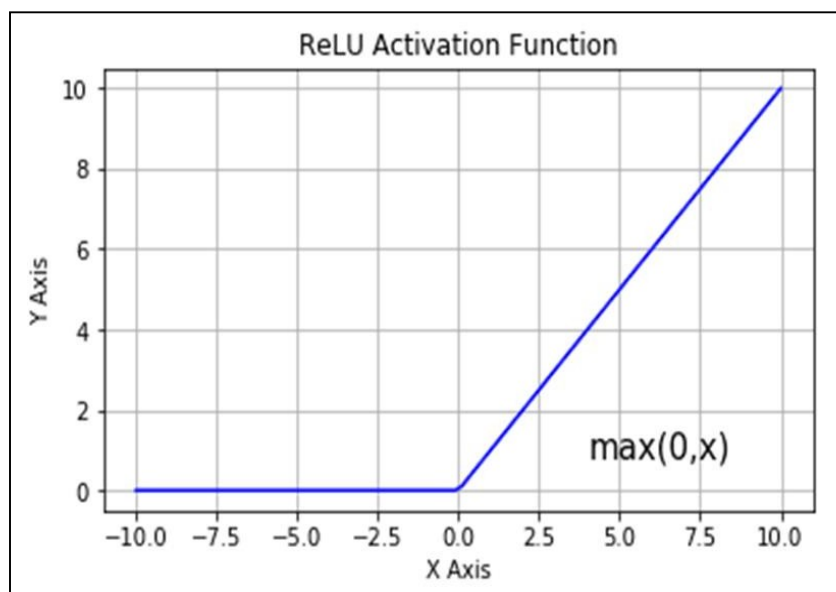
Figure 14: ReLU Activation Function

ReLU (Rectified Linear Unit) is an activation function commonly used in neural networks, including convolutional neural networks like AlexNet and ResNet. It is a simple and computationally efficient activation function that has been shown to improve the performance of deep neural networks compared to traditional activation functions such as sigmoid and tanh.

$$f(x) = max(0, x)$$

where x is the input to the neuron. The function takes the input x and returns the maximum of 0 and x, effectively setting all negative values to 0 and passing positive values through unchanged. This results in a simple, piecewise linear function that is easy to compute and differentiate. ReLU encourages sparsity in the activation of neurons, meaning that only a small fraction of neurons is activated for any given input. This can improve the efficiency of the network and help prevent overfitting. ReLU is a simple function that can be computed quickly and efficiently, making it well-suited for large-scale neural networks. Although ReLU is a linear function for positive values, it is a non-linear function overall.

5. <u>CODING AND IMPLEMENTATION</u>
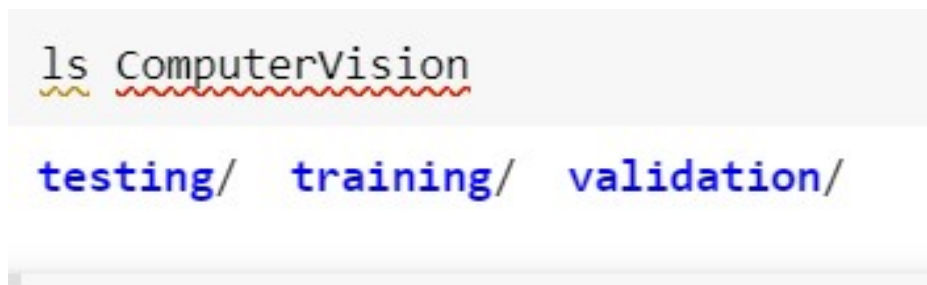
5.1 IMPORTING ESSENTIAL LIBRARIES

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.preprocessing import image

import matplotlib.pyplot as plt

import tensorflow as tf

import numpy as np

import cv2

import os
```

5.2 CONNECTING TO GOOGLE COLAB

```
from google.colab import drive

drive.mount('/content/drive')

cd /content/drive/MyDrive
```

5.3 DATASET SPLITS

Figure 15: Data Splitting

5.4 PRINTING IMAGES OF CHARACTERS

dir_path1 = 'ComputerVision/training/P'

n=0

for i in os.listdir(dir_path1):

    img = image.load_img(dir_path1 + '//' + i,target_size=(400,400))

    plt.imshow(img)

    plt.show()

    n+=1

    if(n==3):

        break
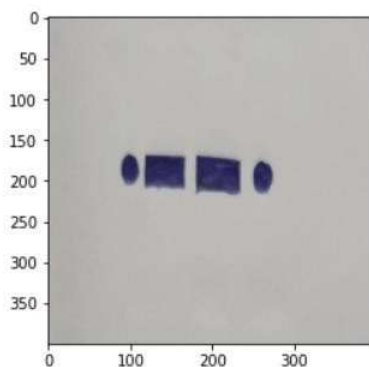
Figure 16: Recognition of Letter P
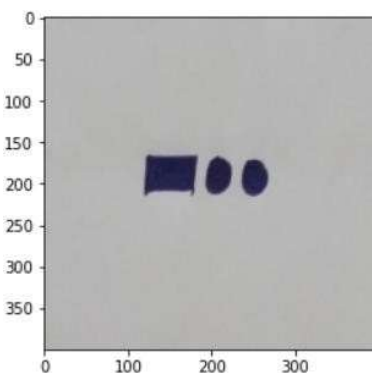
Figure 17: Recognition of Letter D

5.5 CONVERTING PIXEL RANGE FROM [0, 255] TO [0,1]

train = ImageDataGenerator(rescale = 1/255)

validation = ImageDataGenerator(rescale = 1/255)

5.6 READING THE DATASET DIRECTLY FROM THE DIRECTORY

train_dataset = train.flow_from_directory('ComputerVision/training',

target_size=(224,224),

batch_size=128,

class_mode='categorical')

validation_dataset = validation.flow_from_directory('ComputerVision/validation',

target_size = (224,224),

batch_size=128,

class_mode = 'categorical')

5.7 IMPORTING NECESSARY LAYERS FOR LENET-5 ARCHITECTURE

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense,Conv2D,MaxPool2D,Flatten,AveragePooling2D

5.8 WRITING LENET-5 FUNCTION

def LeNet_Architecture():

model = Sequential()

model.add(Conv2D(6, kernel_size=5, strides=1, activation='tanh', input_shape=(224,224,3), padding='same'))

```
model.add(AveragePooling2D())

model.add(Conv2D(16, kernel_size=5, strides=1, activation='tanh', padding='valid'))

model.add(AveragePooling2D())

model.add(Conv2D(120, kernel_size=5, strides=1, activation='tanh', padding='valid'))

model.add(Flatten())

model.add(Dense(84, activation='tanh'))

model.add(Dense(36, activation='softmax'))

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

return model
```

5.9 FITTING THE LENET-5 MODEL WITH 5 EPOCHS

```
LeNet = LeNet_Architecture()

LeNet.fit(train_dataset,

        batch_size=128,

        epochs = 5,

        validation_data = validation_dataset)
```

5.10 LeNet-5 MODEL TRAINING WITH 5 EPOCHS

```
Epoch 1/5
59/59 [==============================] - 763s 13s/step - loss: 3.9526 - accuracy: 0.0259 - val_loss: 3.6274 - val_accuracy: 0.0265
Epoch 2/5
59/59 [==============================] - 758s 13s/step - loss: 3.6004 - accuracy: 0.0238 - val_loss: 3.5854 - val_accuracy: 0.0354
Epoch 3/5
59/59 [==============================] - 737s 12s/step - loss: 3.5952 - accuracy: 0.0259 - val_loss: 3.5948 - val_accuracy: 0.0265
Epoch 4/5
59/59 [==============================] - 731s 12s/step - loss: 3.5931 - accuracy: 0.0242 - val_loss: 3.5911 - val_accuracy: 0.0265
Epoch 5/5
59/59 [==============================] - 729s 12s/step - loss: 3.5951 - accuracy: 0.0251 - val_loss: 3.5887 - val_accuracy: 0.0270
<keras.callbacks.History at 0x7fa2ed076550>
```

Figure 18: LeNet-5 Training with 5 Epochs

5.11 PLOTTING THE MODEL ACCURACY OF LeNet-5 WITH 5 EPOCHS

Accuracy

fig1 = plt.gcf()

plt.plot(LeNet.history.history['accuracy'])

plt.plot(LeNet.history.history['val_accuracy'])

plt.axis(ymin=0.01,ymax=0.05)

plt.grid()

plt.title('LeNet - Model Accuracy')

plt.ylabel('Accuracy')

plt.xlabel('Epochs')

plt.legend(['train', 'validation'])

plt.show()
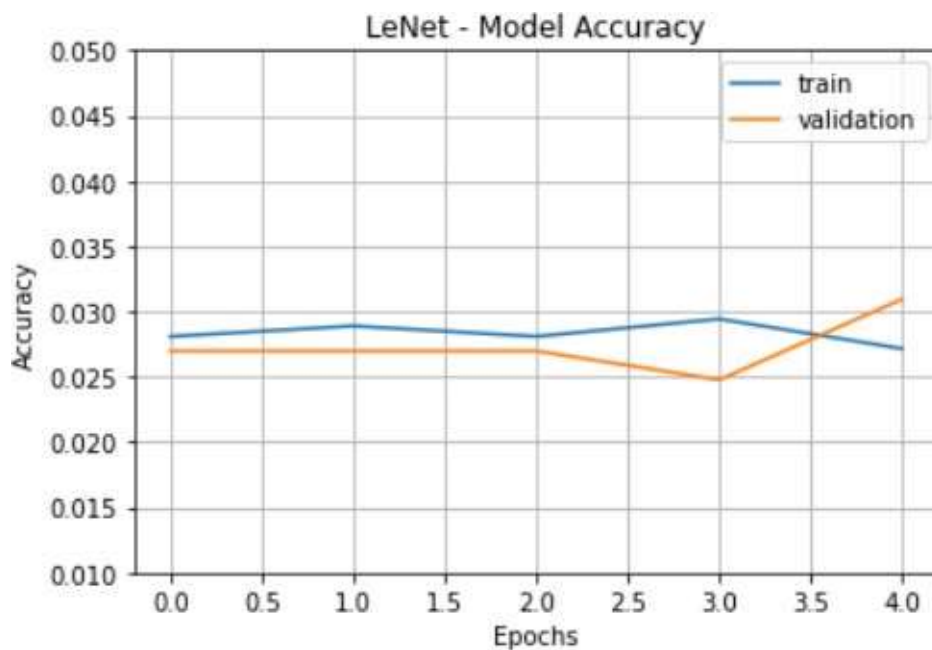
Figure 19: Accuracy of LeNet-5 with 5 Epochs

5.12 FITTING THE LENET-5 MODEL WITH 10 EPOCHS

model = LeNet_Architecture()

model.fit(train_dataset,

      batch_size=128,

      epochs = 10,

      validation_data = validation_dataset)

5.13 LeNet-5 MODEL TRAINING WITH 10 EPOCHS

```
Epoch 1/10
59/59 [==============================] - 3399s 57s/step - loss: 3.9955 - accuracy: 0.0274 - val_loss: 3.6153 - val_accuracy: 0.0310
Epoch 2/10
59/59 [==============================] - 744s 13s/step - loss: 3.5969 - accuracy: 0.0245 - val_loss: 3.5926 - val_accuracy: 0.0265
Epoch 3/10
59/59 [==============================] - 745s 13s/step - loss: 3.5934 - accuracy: 0.0255 - val_loss: 3.5883 - val_accuracy: 0.0265
Epoch 4/10
59/59 [==============================] - 746s 13s/step - loss: 3.5936 - accuracy: 0.0229 - val_loss: 3.5922 - val_accuracy: 0.0265
Epoch 5/10
59/59 [==============================] - 749s 13s/step - loss: 3.5965 - accuracy: 0.0231 - val_loss: 3.5982 - val_accuracy: 0.0287
Epoch 6/10
59/59 [==============================] - 742s 13s/step - loss: 3.5962 - accuracy: 0.0199 - val_loss: 3.5909 - val_accuracy: 0.0265
Epoch 7/10
59/59 [==============================] - 744s 13s/step - loss: 3.5956 - accuracy: 0.0259 - val_loss: 3.5938 - val_accuracy: 0.0265
Epoch 8/10
59/59 [==============================] - 745s 13s/step - loss: 3.5946 - accuracy: 0.0230 - val_loss: 3.5919 - val_accuracy: 0.0265
Epoch 9/10
59/59 [==============================] - 743s 13s/step - loss: 3.5970 - accuracy: 0.0255 - val_loss: 3.5933 - val_accuracy: 0.0248
Epoch 10/10
59/59 [==============================] - 745s 13s/step - loss: 3.5963 - accuracy: 0.0258 - val_loss: 3.5920 - val_accuracy: 0.0248
<keras.callbacks.History at 0x7f4609711370>
```

Figure 20: LeNet-5 Training with 10 Epochs

5.14 PLOTTING THE MODEL ACCURACY OF LeNet-5 WITH 10 EPOCHS

fig1 = plt.gcf()

plt.plot(model.history.history['accuracy'])

plt.plot(model.history.history['val_accuracy'])

plt.axis(ymin=0.01,ymax=0.03)

plt.grid()

plt.title('LeNet - Model Accuracy')

plt.ylabel('Accuracy')

plt.xlabel('Epochs')

plt.legend(['train', 'validation'])

plt.show()
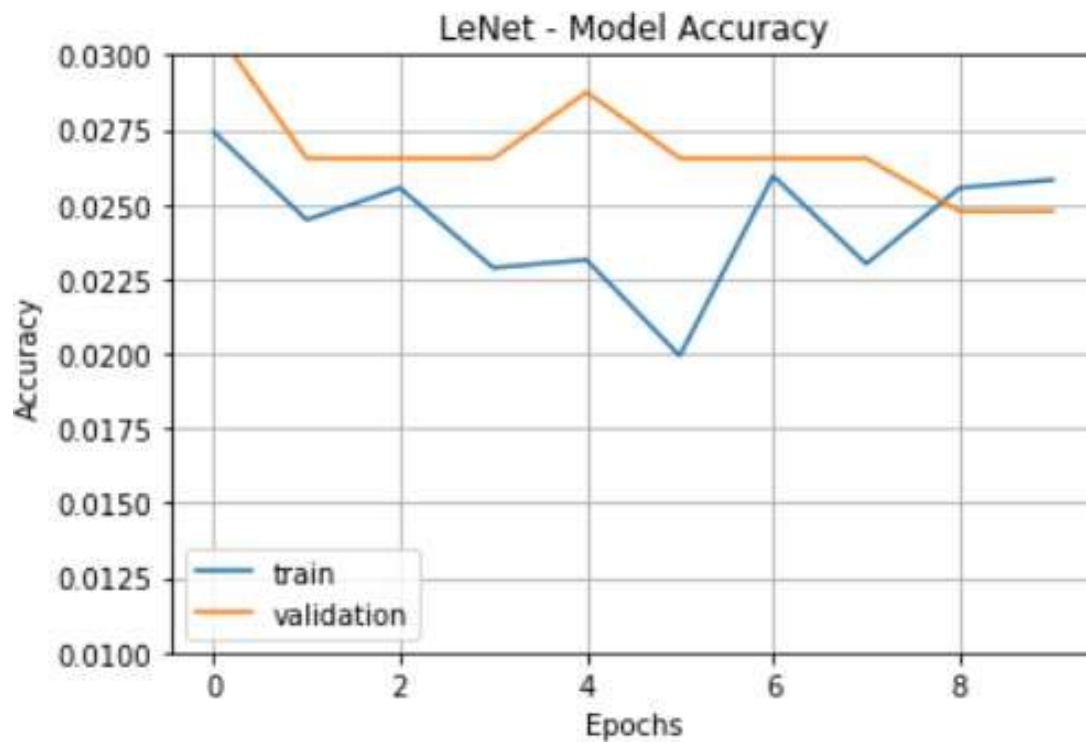
Figure 21: Accuracy of LeNet-5 with 10 Epochs

5.15 FITTING THE LENET-5 MODEL WITH 15 EPOCHS

model = LeNet_Architecture()

model.fit(train_dataset,

batch_size=128,

epochs = 15,

validation_data = validation_dataset)

5.16 LeNet-5 MODEL TRAINING WITH 15 EPOCHS

```
Epoch 1/15
59/59 [==============================] - 1436s 24s/step - loss: 3.9386 - accuracy: 0.0249 - val_loss: 3.5922 - val_accuracy: 0.0269
Epoch 2/15
59/59 [==============================] - 114s 2s/step - loss: 3.5918 - accuracy: 0.0267 - val_loss: 3.5895 - val_accuracy: 0.0265
Epoch 3/15
59/59 [==============================] - 112s 2s/step - loss: 3.5950 - accuracy: 0.0247 - val_loss: 3.5890 - val_accuracy: 0.0265
Epoch 4/15
59/59 [==============================] - 112s 2s/step - loss: 3.5946 - accuracy: 0.0243 - val_loss: 3.5890 - val_accuracy: 0.0357
Epoch 5/15
59/59 [==============================] - 108s 2s/step - loss: 3.5958 - accuracy: 0.0286 - val_loss: 3.5904 - val_accuracy: 0.0265
Epoch 6/15
59/59 [==============================] - 107s 2s/step - loss: 3.5929 - accuracy: 0.0225 - val_loss: 3.5955 - val_accuracy: 0.0282
Epoch 7/15
59/59 [==============================] - 109s 2s/step - loss: 3.5966 - accuracy: 0.0217 - val_loss: 3.5912 - val_accuracy: 0.0265
Epoch 8/15
59/59 [==============================] - 108s 2s/step - loss: 3.5968 - accuracy: 0.0257 - val_loss: 3.5886 - val_accuracy: 0.0265
Epoch 9/15
59/59 [==============================] - 109s 2s/step - loss: 3.5943 - accuracy: 0.0283 - val_loss: 3.5920 - val_accuracy: 0.0265
Epoch 10/15
59/59 [==============================] - 108s 2s/step - loss: 3.5956 - accuracy: 0.0277 - val_loss: 3.5903 - val_accuracy: 0.0309
Epoch 11/15
59/59 [==============================] - 109s 2s/step - loss: 3.5950 - accuracy: 0.0251 - val_loss: 3.5926 - val_accuracy: 0.0282
Epoch 12/15
59/59 [==============================] - 108s 2s/step - loss: 3.5962 - accuracy: 0.0265 - val_loss: 3.5897 - val_accuracy: 0.0265
Epoch 13/15
59/59 [==============================] - 110s 2s/step - loss: 3.5954 - accuracy: 0.0243 - val_loss: 3.5893 - val_accuracy: 0.0265
Epoch 14/15
59/59 [==============================] - 110s 2s/step - loss: 3.5967 - accuracy: 0.0262 - val_loss: 3.5918 - val_accuracy: 0.0265
Epoch 15/15
```

Figure 22: LeNet-5 Training with 15 Epochs

5.17 PLOTTING THE MODEL ACCURACY OF LENET-5 WITH 15 EPOCHS

```
# Accuracy

fig1 = plt.gcf()

plt.plot(model.history.history['accuracy'])

plt.plot(model.history.history['val_accuracy'])

plt.axis(ymin=0.01,ymax=0.05)

plt.grid()

plt.title('LeNet - Model Accuracy')

plt.ylabel('Accuracy')

plt.xlabel('Epochs')

plt.legend(['train', 'validation'])

plt.figure(figsize=(8,6))

plt.show()
```
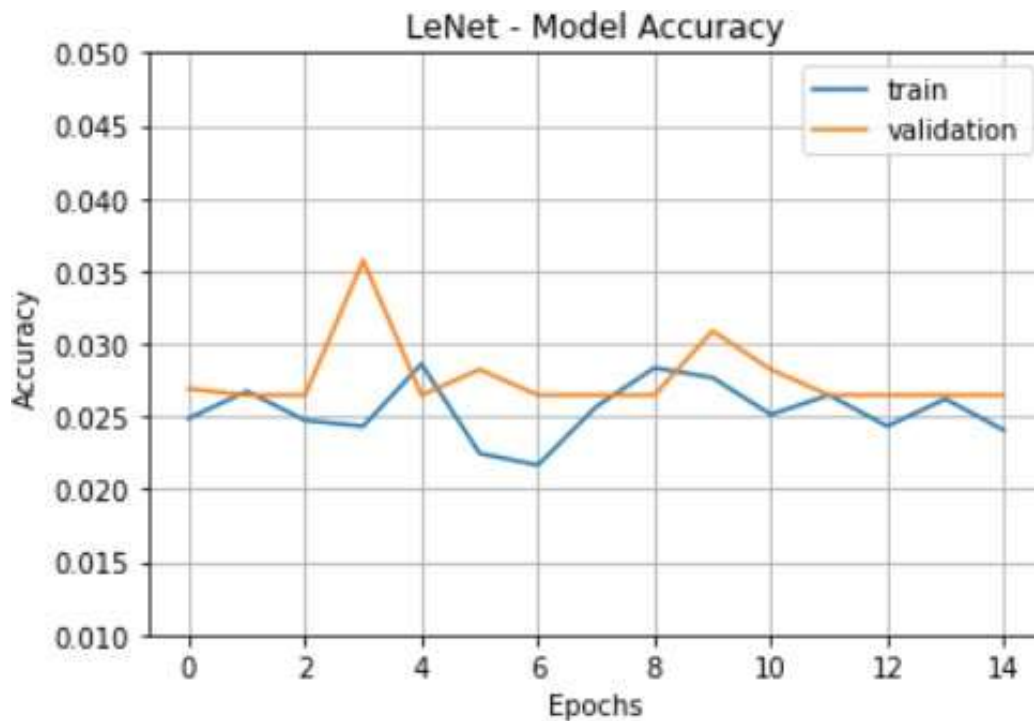
Figure 23: Accuracy of LeNet-5 with 15 Epochs

5.18 LeNet-5 MODEL ANALYSIS AT DIFFERENT EPOCHS

Epochs	Loss	Accuracy
5	3.9526	2.31
10	3.5934	2.59
15	3.5824	2.94
20	3.5967	2.58
50	3.5951	2.42

Table 2: LeNet-5 Analysis at different epochs

5.19 IMPORTING THE LAYER FOR AlexNet ARCHITECTURE

```
from tensorflow.keras.layers import BatchNormalization

from tensorflow.keras.layers import Dropout
```

5.20 WRITING AlexNet FUNCTION

```
def AlexNet_Architecture():

  model = Sequential()

  model.add(Conv2D(filters=128, kernel_size=(11,11), strides=(4,4), activation='relu', input
_shape=(224,224,3)))

  model.add(BatchNormalization())

  model.add(MaxPool2D(pool_size=(2,2)))

  model.add(Conv2D(filters=256, kernel_size=(5,5), strides=(1,1), activation='relu', padding
="same"))

  model.add(BatchNormalization())

  model.add(MaxPool2D(pool_size=(3,3)))

  model.add(Conv2D(filters=256, kernel_size=(3,3), strides=(1,1), activation='relu', padding
="same"))

  model.add(BatchNormalization())

  model.add(Conv2D(filters=256, kernel_size=(1,1), strides=(1,1), activation='relu', padding
="same"))

  model.add(BatchNormalization())

  model.add(Conv2D(filters=256, kernel_size=(1,1), strides=(1,1), activation='relu', padding
="same"))

  model.add(BatchNormalization())

  model.add(MaxPool2D(pool_size=(2,2)))
```

```
model.add(Flatten())

model.add(Dense(1024,activation='relu'))

model.add(Dropout(0.5))


model.add(Dense(1024,activation='relu'))

model.add(Dropout(0.5))

model.add(Dense(36,activation='softmax'))


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

return model
```

5.21 FITTING THE AlexNet MODEL WITH 5 EPOCHS

```
AlexNet = AlexNet_Architecture()

AlexNet.fit(train_dataset,

        batch_size=128,

        epochs = 5,

        validation_data = validation_dataset)
```

5.22 AlexNet MODEL TRAINING WITH 5 EPOCHS

```
Epoch 1/5
59/59 [==============================] - 1133s 19s/step - loss: 2.9509 - accuracy: 0.2520 - val_loss: 10.2521 - val_accuracy: 0.0283
Epoch 2/5
59/59 [==============================] - 1127s 19s/step - loss: 1.0199 - accuracy: 0.6689 - val_loss: 6.1009 - val_accuracy: 0.0823
Epoch 3/5
59/59 [==============================] - 1116s 19s/step - loss: 0.3642 - accuracy: 0.8793 - val_loss: 6.9745 - val_accuracy: 0.0973
Epoch 4/5
59/59 [==============================] - 1095s 19s/step - loss: 0.2223 - accuracy: 0.9319 - val_loss: 3.1460 - val_accuracy: 0.4069
Epoch 5/5
59/59 [==============================] - 1090s 18s/step - loss: 0.1905 - accuracy: 0.9375 - val_loss: 2.1802 - val_accuracy: 0.5993
<keras.callbacks.History at 0x7fa2d48c85b0>
```

Figure 24: AlexNet Training with 5 Epochs

5.23 PLOTTING THE AlexNet MODEL ACCURACY WITH 5 EPOCHS

```
# Accuracy

fig1 = plt.gcf()

plt.plot(AlexNet.history.history['accuracy'])

plt.plot(AlexNet.history.history['val_accuracy'])


plt.axis(ymin=0.1,ymax=1)

plt.grid()

plt.title('AlexNet - Model Accuracy')

plt.ylabel('Accuracy')

plt.xlabel('Epochs')

plt.legend(['train', 'validation'])

plt.show()
```
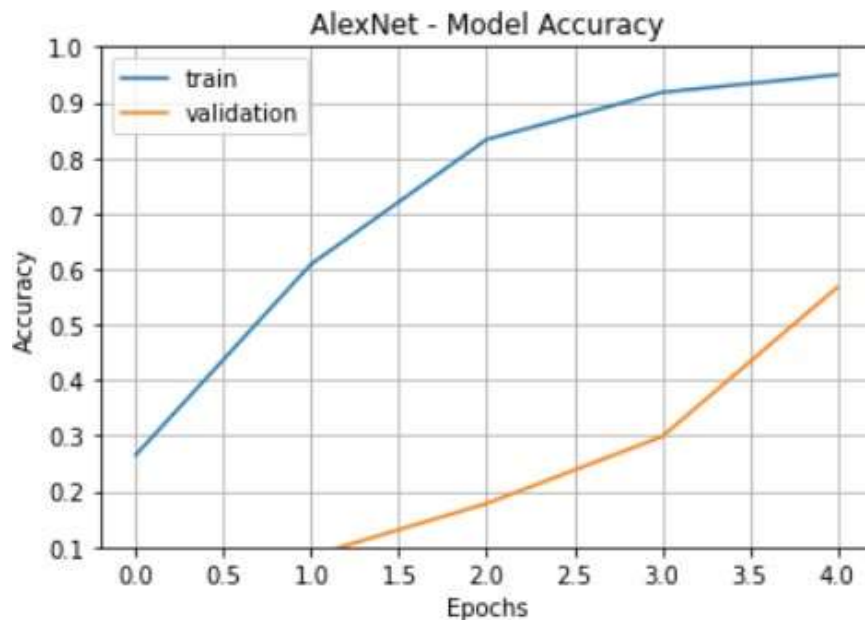

Figure 25: Accuracy of AlexNet with 5 Epochs

5.24 FITTING THE AlexNet MODEL WITH 10 EPOCHS

AlexNet = AlexNet_Architecture()

AlexNet.fit(train_dataset,

batch_size=128,

epochs = 10,

validation_data = validation_dataset)

5.25 AlexNet MODEL TRAINING WITH 10 EPOCHS

```
Epoch 1/10
59/59 [==============================] - 1815s 31s/step - loss: 2.9577 - accuracy: 0.2393 - val_loss: 18.1773 - val_accuracy: 0.0283
Epoch 2/10
59/59 [==============================] - 972s 16s/step - loss: 1.2462 - accuracy: 0.5964 - val_loss: 5.6650 - val_accuracy: 0.1017
Epoch 3/10
59/59 [==============================] - 988s 17s/step - loss: 0.5486 - accuracy: 0.8153 - val_loss: 5.7158 - val_accuracy: 0.0752
Epoch 4/10
59/59 [==============================] - 978s 17s/step - loss: 0.2336 - accuracy: 0.9207 - val_loss: 5.4423 - val_accuracy: 0.1234
Epoch 5/10
59/59 [==============================] - 963s 16s/step - loss: 0.1548 - accuracy: 0.9496 - val_loss: 4.9213 - val_accuracy: 0.3759
Epoch 6/10
59/59 [==============================] - 949s 16s/step - loss: 0.1296 - accuracy: 0.9607 - val_loss: 1.0399 - val_accuracy: 0.7435
Epoch 7/10
59/59 [==============================] - 955s 16s/step - loss: 0.0947 - accuracy: 0.9680 - val_loss: 0.3764 - val_accuracy: 0.8797
Epoch 8/10
59/59 [==============================] - 951s 16s/step - loss: 0.0870 - accuracy: 0.9707 - val_loss: 0.2630 - val_accuracy: 0.9310
Epoch 9/10
59/59 [==============================] - 961s 16s/step - loss: 0.0895 - accuracy: 0.9745 - val_loss: 3.7492 - val_accuracy: 0.5002
Epoch 10/10
59/59 [==============================] - 959s 16s/step - loss: 0.1342 - accuracy: 0.9638 - val_loss: 0.0638 - val_accuracy: 0.9713
<keras.callbacks.History at 0x7f49773a8d90>
```

Figure 26: AlexNet Training with 10 Epochs

5.26 PLOTTING THE AlexNet MODEL ACCURACY WITH 10 EPOCHS

Accuracy

fig1 = plt.gcf()

plt.plot(AlexNet.history.history['accuracy'])

plt.plot(AlexNet.history.history['val_accuracy'])

plt.axis(ymin=0.1,ymax=1)

plt.grid()

plt.title('AlexNet - Model Accuracy')

plt.ylabel('Accuracy')

plt.xlabel('Epochs')

plt.legend(['train', 'validation'])

plt.show()
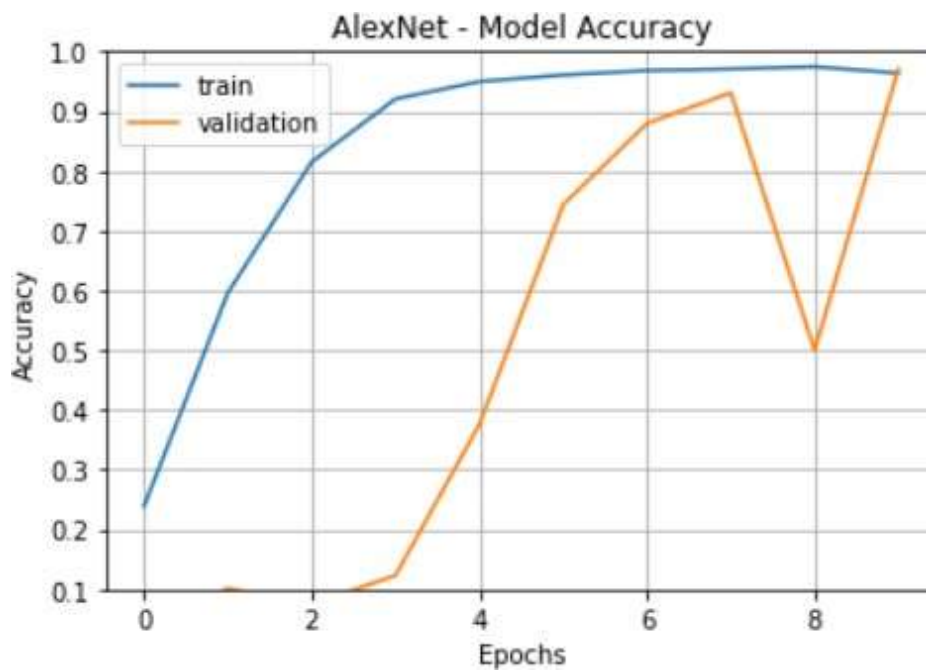
**Figure 27: Accuracy of AlexNet with 10 Epochs**

5.27 FITTING THE AlexNet MODEL WITH 15 EPOCHS

AlexNet = AlexNet_Architecture()

AlexNet.fit(train_dataset,

     batch_size=128,

     epochs = 15,

     validation_data = validation_dataset)

5.28 AlexNet MODEL TRAINING WITH 15 EPOCHS

```
Epoch 1/10
59/59 [==============================] - 1815s 31s/step - loss: 2.9577 - accuracy: 0.2393 - val_loss: 18.1773 - val_accuracy: 0.0283
Epoch 2/10
59/59 [==============================] - 972s 16s/step - loss: 1.2462 - accuracy: 0.5964 - val_loss: 5.6650 - val_accuracy: 0.1017
Epoch 3/10
59/59 [==============================] - 988s 17s/step - loss: 0.5486 - accuracy: 0.8153 - val_loss: 5.7158 - val_accuracy: 0.0752
Epoch 4/10
59/59 [==============================] - 978s 17s/step - loss: 0.2336 - accuracy: 0.9207 - val_loss: 5.4423 - val_accuracy: 0.1234
Epoch 5/10
59/59 [==============================] - 963s 16s/step - loss: 0.1548 - accuracy: 0.9496 - val_loss: 4.9213 - val_accuracy: 0.3759
Epoch 6/10
59/59 [==============================] - 949s 16s/step - loss: 0.1296 - accuracy: 0.9607 - val_loss: 1.0399 - val_accuracy: 0.7435
Epoch 7/10
59/59 [==============================] - 955s 16s/step - loss: 0.0947 - accuracy: 0.9680 - val_loss: 0.3764 - val_accuracy: 0.8797
Epoch 8/10
59/59 [==============================] - 951s 16s/step - loss: 0.0870 - accuracy: 0.9707 - val_loss: 0.2630 - val_accuracy: 0.9310
Epoch 9/10
59/59 [==============================] - 961s 16s/step - loss: 0.0895 - accuracy: 0.9745 - val_loss: 3.7492 - val_accuracy: 0.5002
Epoch 10/10
59/59 [==============================] - 959s 16s/step - loss: 0.1342 - accuracy: 0.9638 - val_loss: 0.0638 - val_accuracy: 0.9713
<keras.callbacks.History at 0x7f49773a8d90>
```

Figure 28: AlexNet Training with 15 Epochs

5.29 PLOTTING THE AlexNet MODEL ACCURACY WITH 15 EPOCHS

Accuracy

fig1 = plt.gcf()

plt.plot(AlexNet.history.history['accuracy'])

plt.plot(AlexNet.history.history['val_accuracy'])

plt.axis(ymin=0.1,ymax=1)

plt.grid()

plt.title('AlexNet - Model Accuracy')

plt.ylabel('Accuracy')

plt.xlabel('Epochs')

plt.legend(['train', 'validation'])

plt.show()
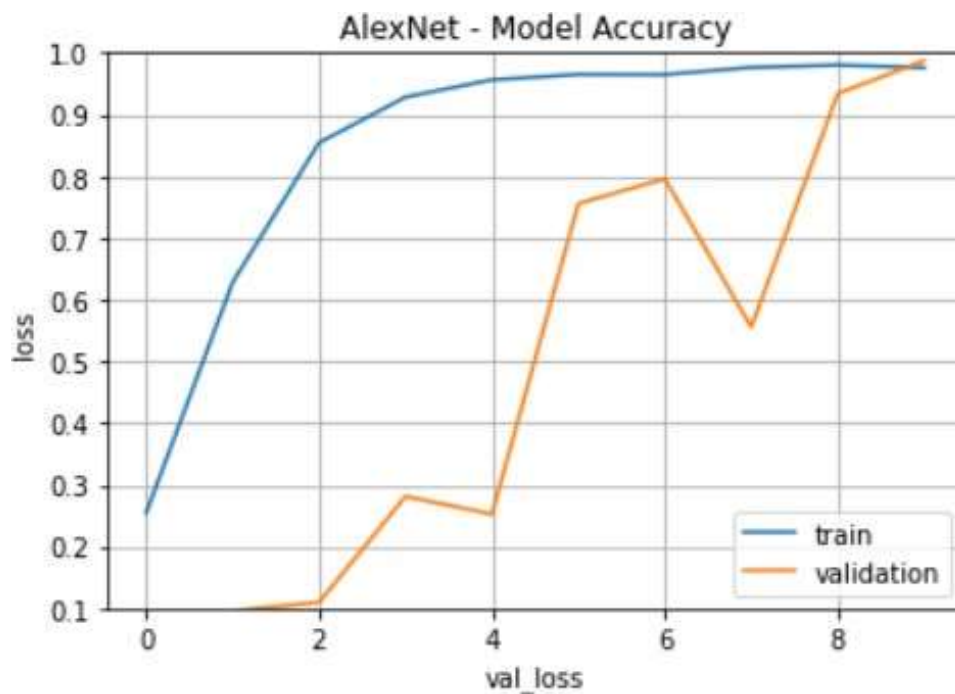
Figure 29: Accuracy of AlexNet with 15 Epochs

5.30 AlexNet MODEL ANALYSIS AT DIFFERENT EPOCHS

Epochs	Loss	Accuracy
5	1.2106	60.89
10	0.5486	81.53
15	0.1280	85.47
20	0.0748	93.06
50	0.0594	96.95

Table 3: AlexNet Model Analysis

6. SYSTEM TESTING

6.1 OVERVIEW OF TESTING

Testing involves a multi-step process that includes data preparation, model training, and testing. The goal of testing is to evaluate the performance of the trained model in detecting Morse code in real-world images. The testing phase typically involves splitting the dataset into two parts: a training set and a testing set. The training set is used to train the CNN model, while the testing set is used to evaluate the performance of the model. During testing, a set of real-world images containing Morse code is fed into the CNN model, which then outputs a predicted text string. The predicted string is then compared to the ground truth (i.e., the actual text string in the image) to determine the accuracy of the model. To evaluate the performance of the model, several metrics can be used, including accuracy, precision, recall, and F1 score. These metrics provide insight into how well the model is performing in detecting Morse code in images and can be used to fine-tune the model if necessary.

6.2 TYPES OF TESTING

6.2.1 Unit Testing

Unit testing is a type of testing method where individual units of code are tested in isolation to verify that they are functioning correctly. In the context of the project Detection of text from Morse code in images using CNN, unit testing can be carried out to verify that individual components of the code are functioning as expected.

The first step in unit testing is to identify the units of code that need to be tested. This could include individual functions, classes, or modules. For each unit of code, define the test cases that will be used to verify its functionality. Test cases should cover a variety of input values and edge cases and should test both expected and unexpected behavior. Write code to execute each test case and define the expected output for each input. The test code should be written in a way that isolates the unit being tested from other parts of the codebase. Run the test code to execute each test case.

Verify that the actual output matches the expected output for each test case. Analyze the results of the unit tests to identify any issues or unexpected behavior. If any tests fail, investigate the cause of the failure, and make the necessary changes to the code. Once a set of

unit tests has been created and executed manually, automate the testing process using a testing framework like pytest or unit test. This allows for automated testing in future development and integration phases.

6.2.2 Integration Testing

Integration testing is a type of testing method where multiple components of a system are tested together to verify that they are functioning correctly. In the context of the project Detection of text from Morse code in images using CNN, integration testing can be carried out to verify that the different components of the project are working together seamlessly. The first step in integration testing is to identify the components of the project that need to be tested together. This could include the CNN model, the pre-processing code, the Morse code detection code, and any other modules or components that are part of the project.

For each combination of components, define the integration test cases that will be used to verify their functionality. Test cases should cover a variety of input values and edge cases and should test both expected and unexpected behavior. Write code to execute each integration test case and define the expected output for each input. The test code should be written in a way that integrates the different components being tested. Run the integration test code to execute each test case. Verify that the actual output matches the expected output for each test case.

Analyze the results of the integration tests to identify any issues or unexpected behavior. If any tests fail, investigate the cause of the failure, and make the necessary changes to the code. Once a set of integration tests has been created and executed manually, automate the testing process using a testing framework like pytest or unit test. This allows for automated testing in future development and deployment phases. Analyze the results of the integration tests to identify any issues or unexpected behavior. If any tests fail, investigate the cause of the failure and make the necessary changes to the code. Verify that the actual output matches the expected output for each test case. Analyze the results of the integration tests to identify any issues or unexpected behavior.

6.2.3 System Testing

System testing is a type of testing method where the entire system is tested to verify that it meets the specified requirements and functions as expected. In the context of the project Detection of text from Morse code in images using CNN, system testing can be carried out to verify that the

entire system works as intended and meets the project objectives. Define the test cases that will be used to verify that the system functions as expected. These test cases should cover a variety of input values and scenarios, including both expected and unexpected behavior. Write code to execute each system test case and define the expected output for each input.

The test code should be written in a way that integrates all the different components of the system being tested. Run the system test code to execute each test case. Verify that the actual output matches the expected output for each test case. Analyze the results of the system tests to identify any issues or unexpected behavior. If any tests fail, investigate the cause of the failure and make the necessary changes to the code. In addition to testing the functional requirements of the system, test non-functional requirements such as performance, scalability, and security. Verify that the system meets all the specified requirements and objectives, and that it performs as expected under different scenarios.

Document the results of the system testing, including any issues that were identified and how they were resolved. If any tests fail, investigate the cause of the failure, and make the necessary changes to the code. In addition to testing the functional requirements of the system, test non-functional requirements such as performance, scalability, and security. Verify that the system meets all the specified requirements and objectives, and that it performs as expected under different scenarios.

7. <u>RESULTS</u>

7.1 PREDICTIONS OF LeNet-5 MODEL

Figure 30: Prediction of Letter I

Figure 31: Prediction of Letter N

```
1/1 [==============================] - 0s 106ms/step
[0.0220491  0.02971478 0.03101162 0.03030004 0.02287747 0.02506633
 0.02744599 0.03352806 0.02544937 0.02833373 0.02598547 0.02871836
 0.02280138 0.03078703 0.03171816 0.02866821 0.03190273 0.02418471
 0.02491642 0.02667914 0.03561369 0.02990204 0.02125951 0.02902283
 0.02751147 0.02853474 0.02452237 0.02587452 0.02317739 0.02981444
 0.02764411 0.02045309 0.02445991 0.02807963 0.03986328 0.03212883]
```
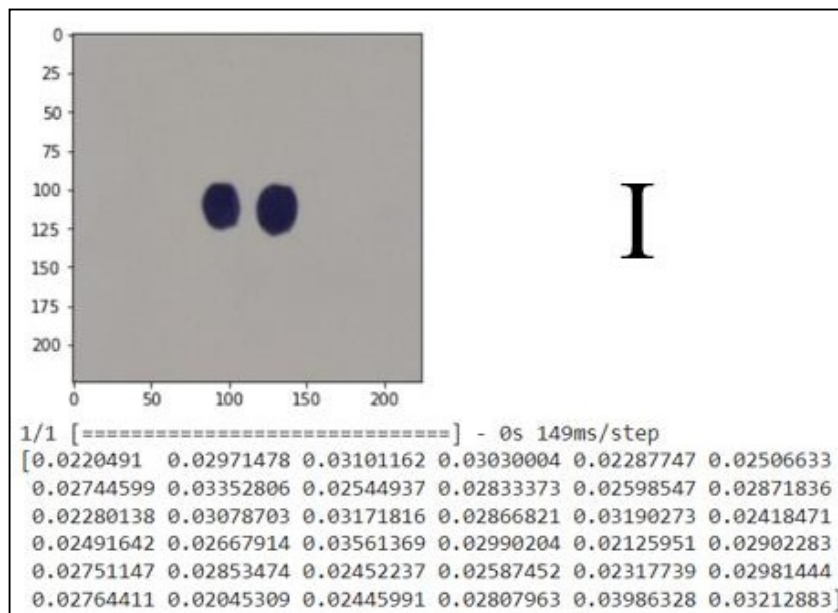
Figure 32: Prediction of Letter D


```
1/1 [==============================] - 0s 71ms/step
[0.0220491  0.02971478 0.03101162 0.03030004 0.02287747 0.02506633
 0.02744599 0.03352806 0.02544937 0.02833373 0.02598547 0.02871836
 0.02280138 0.03078703 0.03171816 0.02866821 0.03190273 0.02418471
 0.02491642 0.02667914 0.03561369 0.02990204 0.02125951 0.02902283
 0.02751147 0.02853474 0.02452237 0.02587452 0.02317739 0.02981444
 0.02764411 0.02045309 0.02445991 0.02807963 0.03986328 0.03212883]
```

Figure 33: Prediction of Letter M

```
1/1 [==============================] - 0s 98ms/step
[0.0220491   0.02971478 0.03101162 0.03030004 0.02287747 0.02506633
 0.02744599 0.03352806 0.02544937 0.02833373 0.02598547 0.02871836
 0.02280138 0.03078703 0.03171816 0.02866821 0.03190273 0.02418471
 0.02491642 0.02667914 0.03561369 0.02990204 0.02125951 0.02902283
 0.02751147 0.02853474 0.02452237 0.02587452 0.02317739 0.02981444
 0.02764411 0.02045309 0.02445991 0.02807963 0.03986328 0.03212883]
```

Figure 34: Prediction of Letter J


```
1/1 [==============================] - 0s 125ms/step
[0.0220491   0.02971478 0.03101162 0.03030004 0.02287747 0.02506633
 0.02744599 0.03352806 0.02544937 0.02833373 0.02598547 0.02871836
 0.02280138 0.03078703 0.03171816 0.02866821 0.03190273 0.02418471
 0.02491642 0.02667914 0.03561369 0.02990204 0.02125951 0.02902283
 0.02751147 0.02853474 0.02452237 0.02587452 0.02317739 0.02981444
 0.02764411 0.02045309 0.02445991 0.02807963 0.03986328 0.03212883]
```

Figure 35: Prediction of Letter U

7.2 PREDICTIONS OF AlexNet MODEL

Figure 36: Prediction of Letter S

Figure 37: Prediction of Letter K
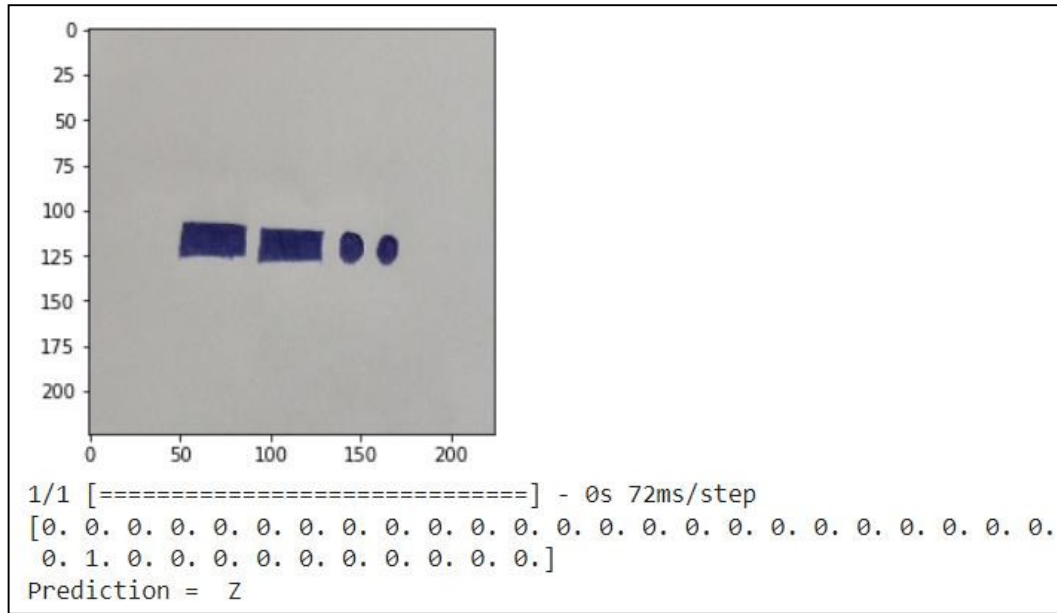
Figure 38: Prediction of Letter Z
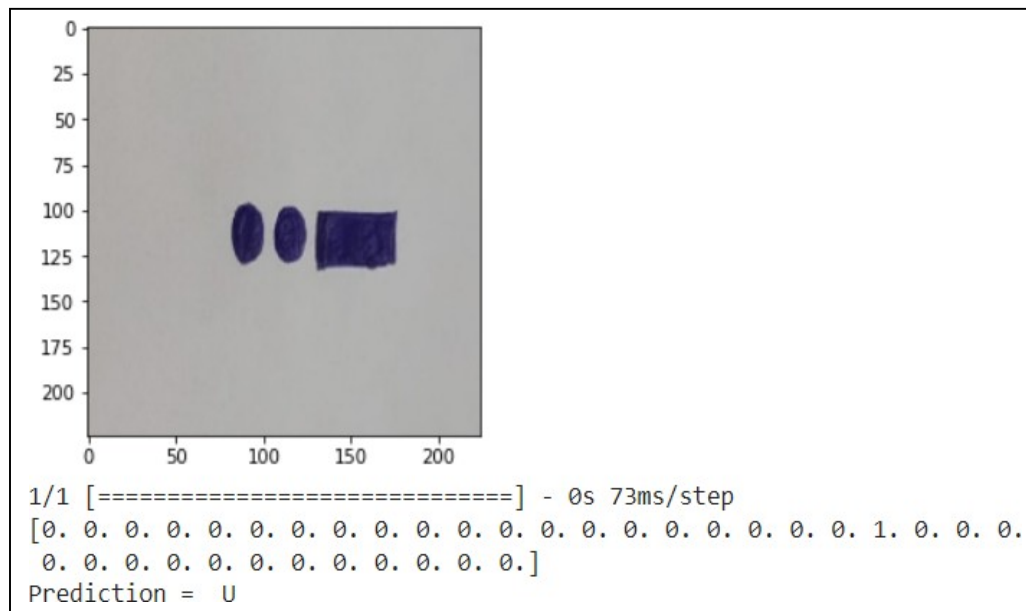
Figure 39: Prediction of Letter U

Figure 40: Prediction of Letter J

Figure 41: Prediction of Letter W

7.3 PREDICTION OF WORDS
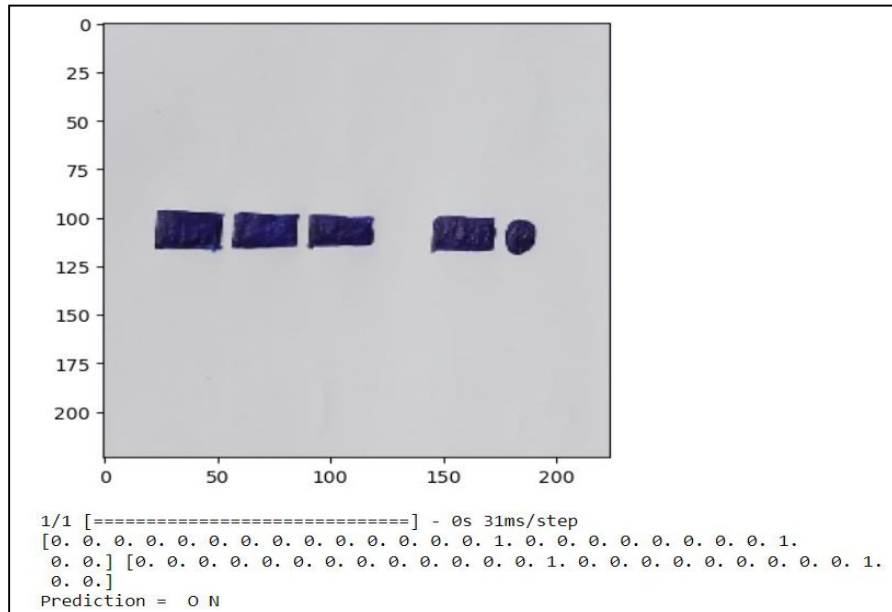
Figure 42: Prediction of Word ON
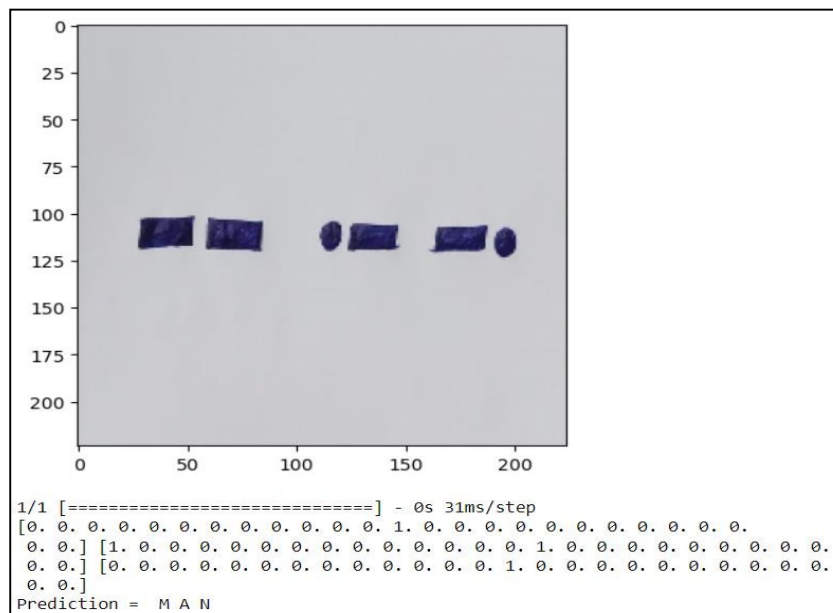
Figure 43: Prediction of Word MAN

8. <u>CONCLUSION</u>

The use of Convolutional Neural Networks (CNN) in detection of text from Morse code in images has shown promising results. The process involves converting the images containing Morse code into a format that can be fed into the CNN model for training and testing. The CNN model can successfully learn the patterns in the Morse code images and accurately predict the corresponding text for the individual characters of Morse Code. After training the model with single characters, then apply the image segmentation based on character segmentation and image segmentation. After segmenting the image, apply the image classification on each individual segmented image parts. After classifying each segment combine the results based on words and characters. Finally, the morse code is converted into English Language with better accuracy. However, the accuracy of the model may vary depending on the quality of the images and the complexity of the Morse code patterns. Overall, the use of CNN in text detection from Morse code in images can be a useful tool in various fields, including military and aviation communication, navigation systems, and emergency communication, among others. Further research can be done to improve the accuracy of the model and explore its potential applications in other areas.

9. REFERENCES

1) S. Sudha, S. Mythili and S. S. Balamurugan, "Detection of Text from Morse Code in Images using Convolutional Neural Network," 2021 International Conference on Electronics, Communication, and Aerospace Technology (ICECA), Coimbatore, India, 2021, pp. 1192-1197, doi: 10.1109/ICECA51584.2021.9485803.

2) R. Girshick, J. Donahue, T. Darrell and J. Malik, "Region-based convolutional networks for accurate object detection and segmentation," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 38, no. 1, pp. 142-158, Jan. 2016, doi: 10.1109/TPAMI.2015.2436995.

3) K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 2016, pp. 770-778, doi: 10.1109/CVPR.2016.90.

4) Y. LeCun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," Proceedings of the IEEE, vol. 86, no. 11, pp. 2278-2324, Nov. 1998, doi: 10.1109/5.726791.

5) J. Redmon, S. Divvala, R. Girshick and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 2016, pp. 779-788, doi: 10.1109/CVPR.2016.91.

6) K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," 2014 arXiv:1409.1556.

7) C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke and A. Rabinovich, "Going Deeper with Convolutions," 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Boston, MA, USA, 2015, pp. 1-9, doi: 10.1109/CVPR.2015.7298594.

8) Y. Taigman, M. Yang, M. Ranzato and L. Wolf, "DeepFace: Closing the Gap to Human-Level Performance in Face Verification," 2014 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Columbus, OH, USA, 2014, pp. 1701-1708, doi: 10.1109/CVPR.2014.220.

9) C. Yan, Z. He, H. Wang and X. Liu, "Morse code identification based on convolutional neural network," 2019 IEEE 5th International Conference on Computer and Communications (ICCC),

Chengdu, China, 2019, pp. 1531-1535, doi: 10.1109/CompComm49359.2019.8941357.

10) S. AlMajthoub, H. AlDmour, M. AlKabi and R. Alghazo, "Morse Code Decoder using Deep Learning," 2019 IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology (JEEIT), Amman, Jordan, 2019, pp. 449-454, doi: 10.1109/JEEIT.2019.8717433.

11) A. M. Abd El-Latif, A. M. Ibrahim and H. A. Hefny, "Morse code recognition using neural networks," 2015 8th International Conference on Human System Interaction (HSI), Warsaw, Poland, 2015, pp. 187-192, doi: 10.1109/HSI.2015.7170695.

12) X. Chen, H. Jia, Y. Yang, C. Li and H. Wang, "Morse code recognition based on convolutional neural network with improved structure," 2020 IEEE 3rd International Conference on Electronics Technology (ICET), Chengdu, China, 2020, pp. 1432-1436, doi: 10.1109/ICET48890.2020.00025.

13) G. Li, Y. Li, B. Li, X. Zhang and Q. Li, "Morse code recognition with convolutional neural networks," 2019 IEEE 14th International Conference on Industrial and Information Systems (ICIIS), Ansan, Korea (South), 2019, pp. 475-478, doi: 10.1109/ICIIS47389.2019.9026748.

14) A. Rizk and M. A. Sharaf, "Morse Code Recognition Using Convolutional Neural Network," 2018 International Conference on Innovative Trends in Computer Engineering (ITCE), Aswan, Egypt, 2018, pp. 371-375, doi: 10.1109/ITCE.2018.8354047.

15) N. Sharma and P. Singh, "Morse Code Decoding using Deep Learning," 2019 IEEE 9th International Conference on Advanced Computing (IACC), Tiruchengode, India, 2019, pp. 311-314.

Comparative Analysis of Detection of Text From Morse Code in Handwritten Images using Convolutional Neural Networks

Shaik Johny Basha
Project Guide, Department of CSE
Lakireddy Bali Reddy College of
Engineering (A), Mylavaram
NTR Dt., Andhra Pradesh, India
shaikhjanibasha@gmail.com

Duggineni Veeraiah
Project Guide, Department of CSE
Lakireddy Bali Reddy College of
Engineering (A), Mylavaram
NTR Dt., Andhra Pradesh, India
veeraiahdvc@gmail.com

Karavadi Raviteja
IV B.Tech Student, Department of CSE
Lakireddy Bali Reddy College of
Engineering (A), Mylavaram
NTR Dt., Andhra Pradesh, India
krtproperty72@gmail.com

Mangam Surya Prakash
IV B.Tech Student, Department of CSE
Lakireddy Bali Reddy College of
Engineering (A), Mylavaram
NTR Dt., Andhra Pradesh, India
suryaprakash9785@gmail.com

Kasturi Karthik
IV B.Tech Student, Department of CSE
Lakireddy Bali Reddy College of
Engineering (A), Mylavaram
NTR Dt., Andhra Pradesh, India
karthikkasturi007@gmail.com

Abstract — **Morse code is a method used in telecommunication to encode text characters as standardized sequences of two different electronic pulses usually represented as short pulse(dot) and long pulse(dash). Detection of text from images of morse code is a complex process and there is no active research on this area. As these are morse code images, different images have different style of strokes. Our work aims to develop an automated Morse code recognition system by training a convolutional neural network (CNN) model using a self-built dataset and involves in collecting and preprocessing images of Morse code characters and creating a labeled dataset for training and testing the CNN model. The dataset creation process includes capturing images of different Morse code characters, augmenting the data to increase the dataset size, and annotating the images to label them correctly. The CNN model is then trained using the created dataset and evaluated for its accuracy in recognizing Morse code characters in images. The results demonstrate that the proposed approach achieves high accuracy in recognizing Morse code characters in images, making it a promising solution for automated Morse code recognition systems.**

Keywords — Morse Code, OCR, AlexNet, Lenet-5, and ResNet-152, Convolutional Neural Network

I. INTRODUCTION

Morse Code is a method of telecommunication, used to transmit messages to longer distances across telegraph wires via electric pulses represented as short pulse(dot) and long pulse(dash). In the beginning of the 19th century, the messages were delivered either to horses or with ravens. For the first time in the history of mankind a group of scientists invented a telecommunication device called telegraph, which was used to transmit the electric pulses to the other device. They developed an electronic device which could send electric pulses to longer distances within less time. Then, Samuel F.B Morse invented a communication language which can be used with electric pulse signals for communication. These electronic pulses can be represented as long pulse(dash) and short pulse(dot). One short pulse and one long pulse can be represented as A and one long pulse, three short pulses can be represented as B. In this way they have successfully invented a communication language which can be associated with electronic pulses to transmit messages to distinct places.

The morse code consists of a total of 36 characters out of which 26 characters are alphabets and the remaining 10 characters are digits. There is no variation between lowercase and uppercase letters. Each morse code symbol is formed by a sequence of dots and dashes. If we consider the duration of dot as one unit then the duration of dash will be considered as three units. In the basic morse code transmission the letters present in the word are separated by a space which has a duration of three dots, and the words are separated by a space equal to seven dots.

TABLE I. CHARACTER SET OF MORSE CODE

English Text	Morse Code	English Text	Morse Code
A	· —	S	· · ·
B	— · · ·	T	—
C	— · — ·	U	· · —
D	— · ·	V	· · · —
E	·	W	· — —
F	· · — ·	X	— · · —
G	— — ·	Y	— · — —
H	· · · ·	Z	— — · ·
I	· ·	0	— — — — —
J	· — — —	1	· — — — —
K	— · —	2	· · — — —
L	· — · ·	3	· · · — —
M	— —	4	· · · · —
N	— ·	5	· · · · ·
O	— — —	6	— · · · ·
P	· — — ·	7	— — · · ·
Q	— — · —	8	— — — · ·
R	· — ·	9	— — — — ·

Optical Character Recognition – shortly known as OCR, in present scenario plays a key role in research for finding out the information available in the images by going through different patterns [4]. The primary focus of this OCR is to get the information from images, where the image consists of text that has been a handwritten text, or printed using a printer, text that has been typed, etc. OCR reduces the effort and time taken by the humans to understand the text in a picture without having adequate knowledge of the language.

One of the main challenges to humans is identifying the text of an unknown language which was handwritten or typed or printed. But there may be some chances that a person can understand the text which was typed or printed. But coming to the handwritten, it will be difficult and ambiguous to identify the text as the person-to-person writing is not going to be same. So, to identify the text in an image and convert them into English Language plays a key role. For that, first the character must be extracted from the image in different ways such as Diagonal based feature extraction [5], Deep Feature Extraction [6], etc. Like humans, machines also don't understand the information available in the images when they are not trained to a suitable language. To train a machine with different languages, we can use Convolution Neural Network Based frameworks like VGG-16, InceptionV3, ResNeXTt50, Xception, etc. These frameworks are really helping humans by giving a good prediction rate and high accuracy in results, when they are well trained and tested.

Convolutional Neural Network – shortly known as ConvNet/CNN, which is a part of Deep Neural Networks that reads the images or pictures as input and helps in image analysis such as feature extraction, layer pooling, etc. [7]. We know that Matrix Multiplication plays a major role in Neural Networks to perform operations over hidden layers and image analysis. In the same way, this CNN uses a most unique feature called Convolution that produces the new function based on the mathematical operation among two functions. That new function tells the changes that have been made on the images and how it has been modified. A generalized and standard view of how a Convolution Neural Network works has been shown in fig. 1, that has been taken from [8].
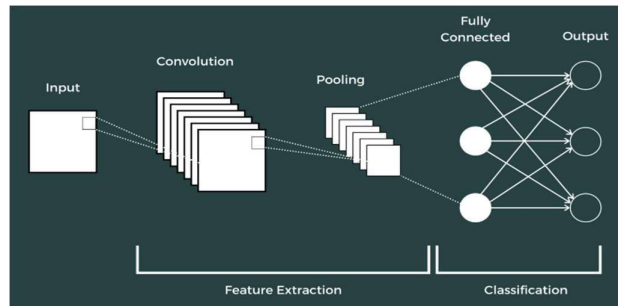
Fig. 1. Generalized view of working of CNN

LeNet-5 was a CNN-based architecture introduced by LeCun et. al. in 1998 [9]. This architecture is made up of 7-Levels which are 2 sets of fully connected as well as sub-sampling layers, and 3 sets of convolutional layers. LeNet-5 architecture is mainly used in classification of handwritten characters either may be on plain paper or bank cheques, or other items which helps for 32*32-pixel grayscale images [10]. As per the author of [11], LeNet-5 architecture consists of following functions for classifying MNIST digits: a tanh activation function, two fully connected layers along with a

SoftMax classifier. Overview of LeNet-5 architecture has been given in fig. 2, which is taken from [9].
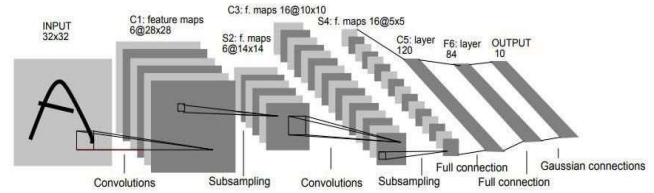
Fig. 2. Overview of LeNet-5 Architecture

ResNet-152, which is the latest version of Residual Neural Network (ResNet), which focuses primarily on the residual learning and skipping the connections [10]. These skip connections feature heavy batch normalization, which are also known as gated units (mostly similar to the elements applied in RNN). If our Neural Network has more than 152 layers, then ResNet-152 is more useful than VGGNet with lower complexity. The architecture of ResNet has been given in fig. 3, which was taken from [12].
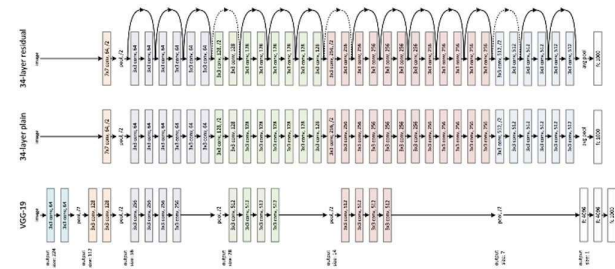
Fig. 3. ResNet-152 Architecture

AlexNet was designed with Eight Layers: 5 of them are convolutional layers (3 max-pooling layers and 2 normalization layers) and 3 are fully connected layers (2 fully connected layers and 1 softmax layer). These layers use ReLu activation except on the output layer [13]. The input size of this model is said to be 224*224*3, but when the padding happens sometimes, then it will be 227*227*3. Comparison of LeNet and AlexNet was given in Fig. 4, which was taken from [14].

Fig. 4. Comaprison of LeNet and AlexNet Layers

Our work aims to find the English meaning to the code in Amharic which was handwritten as shown in Fig. 5. For that purpose, we have used the existing CNN-Based algorithms named ResNet-152, LeNet-5, and AlexNet on our own dataset named LBRAMHARIC (created by our team from various age groups of our organization) and concatenated with existing datasets.
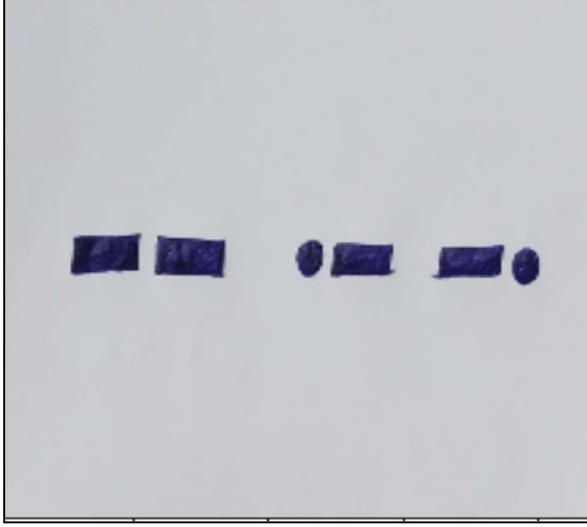
Fig. 5. Sample Word in Morse Code

Our paper has been organized in the following manner: As we discussed the introduction of our research in Section I, Section II deals with the work done by different authors on same area (aka literature survey or related work), Section III goes with our working methodology, Section IV with the results obtained for our work and analysis of different algorithms, and finally the conclusion was given in the ending of paper i.e., Section V.

II. RELATED WORK

Balamurugan et al. [10] have introduced the CNN-based method for Character Recognition of Morse Code Language. They have created a dataset of Morse Code Characters. Their work function with a max-pooling of 2*2. For their dataset and model, they obtained an average accuracy of 81% for characters. But they haven't performed their work on complex images.

Ren et al. [12] have generated a new dataset with more than 10,000 in Morse Code. In their proposed work, they have used Support Vector Machines (SVM) along with Connectionist Temporal. After that, they tested their model with the printed as well as synthesized datasets and obtained a good accuracy of more than 84% for printed datasets.

LeCun et al. [15] have performed the text recognition for Morse Code Sign Language. In this, they have considered the various types of handwritten images of Morse Code. For their work, they have obtained an accuracy of 91% for all the characters and numerics.

Redmon et al. [16] used various hybrid acoustic modeling units such as phonetics, syllables, and rounded phonetics. They developed a Deep Neural Network based on the above said units and they obtained a good outcome over the GMM (Gaussian Mixture Models). Their results are better with a WER of 11.28% and maximum of 17% for context-dependent and independent models with very limited dataset.

AlMajthoub et al. [19] have used their own dataset with more than 10,000 images. Their work primarily focused on identifying the characters in printed character images. They have done the comparison of various authors work on those type of images. The proposed work of them have achieved an accuracy of 92%, which is more than the other authors. But the problem is they have focused on printed characters, not on handwritten images.

TABLE II.   A SURVEY ON HANDWRITTEN MORSE CODE RECOGNITON SYSTEMS

Authors	Method	Accuracy
Balamurugan et al. [10]	Artificial Neural Networks	81%
Ren [12]	Support Vector Machines	84%
LeCun [13]	Artificial Neural Networks	91%
Redmon et al. [14]	CNN-Based Algorithm	94%
Our Method	AlexNet	96%

III. METHODOLOGY

In this section, we are providing the workflow of our work i.e., how we have gathered images, how the impurities were cleaned, how the images were trained and tested, classification of images into folders, applying the CNN based framework on our dataset, and finally testing on dynamic inputs to check whether we are getting correct result or not. The entire process was shown in fig. 6.
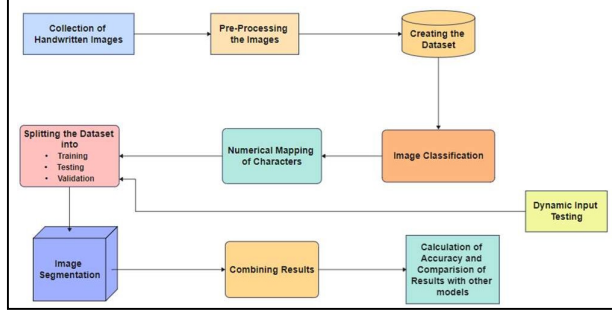
Fig. 6. Workflow of our Comparison

A. LBRMORSE

LBRMORSE is a dataset created by the authors from Lakireddy Bali Reddy College of Engg. We created this dataset by collecting the handwritten Morse Code Characters from different age groups ranging between 17 to 55 years i.e., students, teaching and supporting staff of our institute. For creating the dataset, we made a grid of 06*06 i.e., (06 rows of 06 columns each) as shown in fig. 7. Then that was taken out the print and handed over to the members to fill that. To get a better pixel quality, color ratio, smoothing of images, etc. we requested every member to fill the form using blue or black pen only (one of the filled forms by a member is given in fig. 8).

Fig. 7. Data Collection Form Template

Fig. 8. Sample Form from one of the Member

B. Preprocessing of Images

We have collected the samples from 460 members our organization. Now, we converted the handwritten forms into images by scanning them using a Digital Scanner with a pixel ration of 600 dpi. After that, every image will be divided into 36 small images which represent phonetics and syllables of Amharic Language with a pixel size of 32*32. A total of 36 folders have been created to store the images divided into single images. Then we have performed quality check on individual images to detect any impurities in them or any color issues. Those types of images have been removed from the folders. At the end, the normal handwritten images available in every folder are converted to Grayscale.

C. Mapping

To get a good understanding of images, we have given a symbol name for every folder for all 36 characters. The main intention behind mapping images with symbol name is for detecting the character very easily in the word and mapping the result to output.

D. Training and Testing

Now the complete dataset is ready for usage. Here we are categorizing the dataset into two sections: one is for training data and other is for testing data by considering in the ration of 70:30.

E. Implementation

After the model has trained with training and testing data, now the implementation has been done in Python using different libraries for different activities. Some of the libraries we used are TensorFlow, NumPy, PyPlot in matplotlib, etc. After that, our CNN-Based models have been passed using the Train and Test data. The weights of the samples will be adjusted automatically by CNN which we have used for training purpose. Epoch has been calculated for every phonetic and syllable till 100 epochs.

F. Accuracy Calculation and Analysis

In our work, we have considered the epoch number 100 as the performance metric. Different CNN-based frameworks were trained using our dataset and then we have considered the accuracy obtained from each framework as the Prediction Value (PV).

G. Dynamic Input Testing

At the end, we have passed a random image with Amharic Text (image which was written by a non-participant of dataset creation) and passed through the model to verify whether it is giving the accurate results or not. We passed nearly 250 dynamic images to test the accuracy and got good results.

IV. RESULTS AND ITS ANALYSIS

Section IV of our paper deals with the accuracy and prediction results obtained for various frameworks on our dataset. Our work has been performed on Windows 10 Operating System with a RAM of 64 GB and X64 based processor.

A. Model Training at various Epochs:

We trained our models (AlexNet, and LeNet-5) with 5 epochs, 10 epochs, 25 epochs and 50 epochs. From the results, we have observed that we get a better learning rate when the rate of epochs is increasing. In Table III, we have shown the loss and accuracy for various epochs for different models.

TABLE III. ACCURACY COMPARISON AT VARIOUS LEARNING RATES

Epochs	LeNet-5		AlexNet	
	Loss	Accuracy	Loss	Accuracy
5	3.9526	2.31	0.3911	0.6089
10	3.5934	2.59	0.5486	0.4514
15	3.5824	2.94	0.1280	0.872
20	3.5967	2.58	0.0748	0.9252
50	3.5951	2.42	0.0305	0.9695
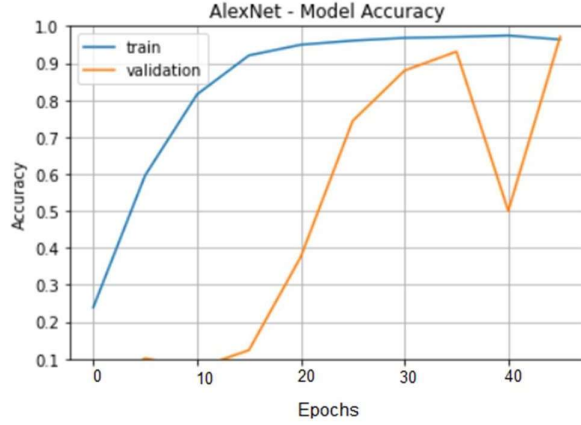
Fig. 9. Training and Validation Accuracy obtained at 50 Epochs for AlexNet

B. Training and Validation Accuracy

Accuracy was calculated for all the models after learning rate. We obtained different accuracies for different models using AlexNet, LeNet-5, ResNet, and VggNet frameworks and the summary of obtained results was shown in Table IV.

TABLE IV. ACCURACY OBTAINED FOR VARIOUS FRAMEWORKS

CNN-Based Framework	Loss	Accuracy
LeNet	3.5951	2.42
AlexNet	0.0305	0.9695
VggNet	2.9874	2.94
ResNet	0.2590	91.25

C. Testing the Model using Dynamic Inputs

We have passed the random inputs (a total of 250) taken from the normal people to our models. For dynamic inputs, we got an accuracy of more than 80%. We have given the sample output obtained from Dynamic Input in fig. 11, fig. 12, fig. 13, fig, 14 and considered the Dynamic Images Accuracy as:

Dynamic Input Accuracy = (Correct Result Obtained)/(Total Images Passed)

Character	Before Data-Processing	After Data-Processing	Accuracy
A	250 Images	212 Images	89.91
B	250 Images	223 Images	86.23
C	250 Images	201 Images	90.21
D	250 Images	198 Images	92.21
E	250 Images	240 Images	93.45
F	250 Images	189 Images	91.35
G	250 Images	178 Images	87.23
H	250 Images	189 Images	93.65
I	250 Images	178 Images	94.25
J	250 Images	200 Images	95.64
K	250 Images	203 Images	93.89
L	250 Images	219 Images	92.45
M	250 Images	229 Images	91.67
N	250 Images	232 Images	89.53
O	250 Images	211 Images	88.53
P	250 Images	204 Images	91.26
Q	250 Images	225 Images	92.86
R	250 Images	212 Images	93.46
S	250 Images	221 Images	81.45
T	250 Images	201 Images	91.67
U	250 Images	189 Images	85.26
V	250 Images	178 Images	86.59
W	250 Images	213 Images	91.86
X	250 Images	222 Images	92.75
Y	250 Images	201 Images	87.85
Z	250 Images	200 Images	92.56
0	250 Images	201 Images	93.76
1	250 Images	221 Images	94.89
2	250 Images	200 Images	95.64
3	250 Images	187 Images	94.52
4	250 Images	178 Images	89.25
5	250 Images	167 Images	88.46
6	250 Images	198 Images	94.78
7	250 Images	200 Images	93.78
8	250 Images	201 Images	92.85
9	250 Images	198 Images	91.56

Fig. 10. Dynamic Inputs Passed to the Model


```
1/1 [==============================] - 0s 31ms/step
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0.] [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
 0. 0.] [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0.]
Prediction =  M A N
```

Fig. 11. Sample Result Obtained for Dynamic Input 1

Fig. 12. Dynamic Input 2 Passed to the Model

V. CONCLUSION

The use of Convolutional Neural Networks (CNN) in detection of text from Morse code in images has shown promising results. The process involves converting the images containing Morse code into a format that can be fed into the CNN model for training and testing. The CNN model can successfully learn the patterns in the Morse code images and accurately predict the corresponding text for the individual characters of Morse Code. After training the model with single characters, then apply the image segmentation based on character segmentation and image segmentation. After segmenting the image, apply the image classification on each individual segmented image parts. After classifying each segment combine the results based on words and characters. Finally, the morse code is converted into English Language with better accuracy. However, the accuracy of the model may vary depending on the quality of the images and the complexity of the Morse code patterns. Overall, the use of CNN in text detection from Morse code in images can be a useful tool in various fields, including military and aviation communication, navigation systems, and emergency communication, among others. Further research can be done to improve the accuracy of the model and explore its potential applications in other areas.

REFERENCES

[1] Amharic, https://en.wikipedia.org/wiki/Amharic

[2] Ge'ez script, https://en.wikipedia.org/wiki/Ge%CA%BDez_script

[3] Optical character recognition, https://en.wikipedia.org/wiki/Optical_character_recognition

[4] Introduction to Convolutional Neural Networks (CNN), https://www.analyticsvidhya.com/blog/2021/05/convolutional-neural-networks-cnn/

[5] Introduction to Convolutional Neural Networks, https://www.theclickreader.com/introduction-to-convolutional-neural-networks/

[6] Siddharth Das, CNN Architectures: LeNet, AlexNet, VGG, GoogLeNet, ResNet and more…, CNN Architectures: LeNet, AlexNet, VGG, GoogLeNet, ResNet and more… | by Siddharth Das | Analytics Vidhya | Medium, 16/November/2017.

[7] Sean Benhur, CNN Architectures from Scratch: From Lenet to ResNet, https://medium.datadriveninvestor.com/cnn-architectures-from-scratch-c04d66ac20c2, 15/November/2020

[8] AlexNet: The First CNN to win Image Net, https://www.mygreatlearning.com/blog/alexnet-the-first-cnn-to-win-image-net/, Updated on 31/October/2022.

[9] AlexNet, https://en.wikipedia.org/wiki/AlexNet

[10] S. Sudha, S. Mythili and S. S. Balamurugan, "Detection of Text from Morse Code in Images using Convolutional Neural Network," 2021 International Conference on Electronics, Communication, and Aerospace Technology (ICECA), Coimbatore, India, 2021, pp. 1192-1197, doi: 10.1109/ICECA51584.2021.9485803.

[11] R. Girshick, J. Donahue, T. Darrell and J. Malik, "Region-based convolutional networks for accurate object detection and segmentation," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 38, no. 1, pp. 142-158, Jan. 2016, doi: 10.1109/TPAMI.2015.2436995.

[12] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 2016, pp. 770-778, doi: 10.1109/CVPR.2016.90.

[13] Y. LeCun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," Proceedings of the IEEE, vol. 86, no. 11, pp. 2278-2324, Nov. 1998, doi: 10.1109/5.726791.

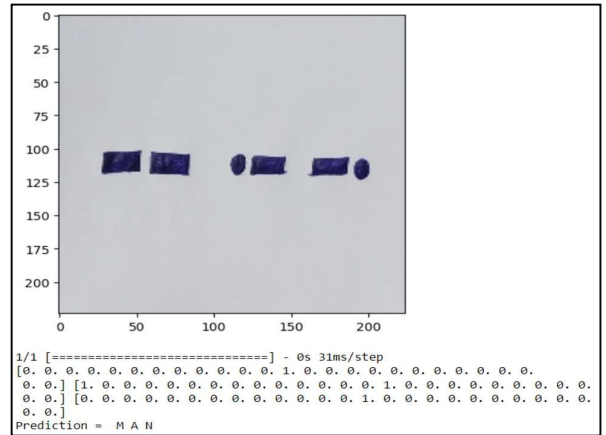[14] J. Redmon, S. Divvala, R. Girshick and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 2016, pp. 779-788, doi: 10.1109/CVPR.2016.91.

[15] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," 2014 arXiv:1409.1556.

[16] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke and A. Rabinovich, "Going Deeper with Convolutions," 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Boston, MA, USA, 2015, pp. 1-9, doi: 10.1109/CVPR.2015.7298594.

[17] Y. Taigman, M. Yang, M. Ranzato and L. Wolf, "DeepFace: Closing the Gap to Human-Level Performance in Face Verification," 2014 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Columbus, OH, USA, 2014, pp. 1701-1708, doi: 10.1109/CVPR.2014.220.

[18] C. Yan, Z. He, H. Wang and X. Liu, "Morse code identification based on convolutional neural network," 2019 IEEE 5th International Conference on Computer and Communications (ICCC), Chengdu, China, 2019, pp. 1531-1535, doi: 10.1109/CompComm49359.2019.8941357.

[19] S. AlMajthoub, H. AlDmour, M. AlKabi and R. Alghazo, "Morse Code Decoder using Deep Learning," 2019 IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology (JEEIT), Amman, Jordan, 2019, pp. 449-454, doi: 10.1109/JEEIT.2019.8717433.

[20] A. M. Abd El-Latif, A. M. Ibrahim and H. A. Hefny, "Morse code recognition using neural networks," 2015 8th International Conference on Human System Interaction (HSI), Warsaw, Poland, 2015, pp. 187-192, doi: 10.1109/HSI.2015.7170695.

[21] X. Chen, H. Jia, Y. Yang, C. Li and H. Wang, "Morse code recognition based on convolutional neural network with improved structure," 2020 IEEE 3rd International Conference on Electronics Technology (ICET), Chengdu, China, 2020, pp. 1432-1436, doi: 10.1109/ICET48890.2020.00025.

[22] G. Li, Y. Li, B. Li, X. Zhang and Q. Li, "Morse code recognition with convolutional neural networks," 2019 IEEE 14th International Conference on Industrial and Information Systems (ICIIS), Ansan, Korea (South), 2019, pp. 475-478, doi: 10.1109/ICIIS47389.2019.9026748.

[23] A. Rizk and M. A. Sharaf, "Morse Code Recognition Using Convolutional Neural Network," 2018 International Conference on Innovative Trends in Computer Engineering (ITCE), Aswan, Egypt, 2018, pp. 371-375, doi: 10.1109/ITCE.2018.8354047.

[24] N. Sharma and P. Singh, "Morse Code Decoding using Deep Learning," 2019 IEEE 9th International Conference on Advanced Computing (IACC), Tiruchengode, India, 2019, pp. 311-314.