HANDWRITTEN RECOGNITION OF CHARACTERS USING CNN.

A Project Report

Submitted by

V S SRI SAI PRANEETH BULUSU -AP22110010049

P VENKATA SRINAG-AP22110010050

V MOHAN BALU-AP22110010057

VOONA SRI RAJ-AP22110010617

Under the Supervision of

Mr. Ajay Dilip Kumar Marapatla
Assistant Professor
Department of Computer Science and Engineering
SRM University-AP

In partial fulfilment for the requirements of the IDP-100 Project

BACHELOR OF TECHNOLOGY
IN
COMPUTER SCIENCE AND ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SRM UNIVERSITY-AP

NEERUKONDA

MANAGALAGIRI - 522503

ANDHRA PRADESH, INDIA

NOVEMBER-2024

CERTIFICATE

This is to certify that the project work entitled **"HANDWRITTEN RECOGNITION OF CHARACTERS USING CNN."** is a Bonafide record of project work carried out by the following students:

- **Mr. V S SRI SAI PRANEETH BULUSU** (Roll No.: AP22110010049)
- **Mr. P VENKATA SRINAG** (Roll No.: AP22110010050)
- **Mr. V MOHAN BALU** (Roll No.: AP22110010057)
- **MR. VOONA SRI RAJ(**Roll No:AP22110010617**)**

from the **Department of Computer Science and Engineering, SRM University-AP.** The students conducted this project work under my supervision during the period **August 2024 to November 2024**.It is further certified that, to the best of my knowledge, this project has not previously formed the basis for the award of any degree or any similar title to this or any other candidate.

This is also to certify that the project work represents the **teamwork** of the candidates.

Station: Mangalagiri
Date: 25 - 11 - 2024

Mr. Ajay Dilip Kumar Marapatla
Assistant Professor
Department of Computer Science & Engineering
SRM University-AP
Andhra Pradesh.

TABLE OF CONTENTS

ABSTRACT

Handwritten character recognition is a critical area of research in the field of computer vision and artificial intelligence, with numerous applications in document digitization, assistive technologies, and automated data entry. This project focuses on developing a robust and efficient system for recognizing handwritten characters using Convolutional Neural Networks (CNNs). The system aims to automatically identify individual characters from handwritten images, offering an accurate and scalable solution to address the challenges posed by varying handwriting styles, sizes, and orientations.

The project begins with the collection and preprocessing of a diverse dataset consisting of handwritten character images. Data augmentation techniques such as random rotation, scaling, and noise addition are employed to enhance the model's generalization ability. A CNN-based architecture is then designed, consisting of convolutional, pooling, and fully connected layers, optimized for accurate recognition of handwritten characters.

The model is trained using a labeled dataset and evaluated on key performance metrics such as accuracy, precision, recall, and F1-score. The system is deployed with a simple web interface that allows users to upload handwritten images for real-time recognition. Through this approach, the project demonstrates the potential of deep learning in solving real-world challenges in handwritten character recognition.

Future research directions include extending the system to support multilingual recognition, improving robustness to noisy and low-quality input, and optimizing the model for real-time applications on resource-constrained devices.

INTRODUCTION

One of the most important areas in computer vision and pattern recognition is handwritten character recognition, which interprets and classifies characters from handwritten input. This project focuses on developing a system with CNNs to achieve accurate recognition of individual handwritten characters from images. In order to accomplish this task, CNNs are particularly appropriate because they can extract spatial and structural features from images, and thus are robust against variability in the handwriting styles, shapes, and even orientations. The project, therefore, aims at automating the recognition process to classify handwritten characters efficiently and accurately.

This project will set up a labeled dataset by producing handwritten character images, preprocess them to be uniform, and formulate a CNN architecture designed specifically for this recognition problem. The model will learn specific patterns for each character through training and optimal fit will then be achieved using several techniques including hyperparameter tuning and data augmentation, among others. The goal should be a really robust system dealing with several noisy inputs and different types of handwriting. Such a system can be applied in Optical Character Recognition, digitization of documents by handwriting, and real-time analysis of the text for assistive technologies.

Building upon the foundation of using Convolutional Neural Networks (CNNs) for handwritten character recognition, the project proceeds to implement a well-defined workflow, starting with the creation of a comprehensive and diverse dataset. This dataset will encompass various handwriting styles, ensuring the model learns to generalize effectively across different individuals. Preprocessing will standardize the images by resizing, normalizing pixel values, and applying noise reduction techniques to improve consistency and eliminate extraneous variations.

Once the data is prepared, a custom CNN architecture will be designed, incorporating multiple convolutional layers for feature extraction, pooling layers to reduce spatial dimensions, and fully connected layers for classification. Techniques such as dropout will be used to prevent overfitting, while data augmentation will expand the dataset with variations such as rotations, scaling, and translations to enhance the model's robustness. Training will involve experimenting with different hyperparameters, such as learning rate, batch size, and number of epochs, to optimize performance.

PROBLEM DEFINITION

Handwritten character recognition has been a great area of continuous research in the domain of computer vision. However, it still has many problems to be solved, mainly due to variations in penmanship, uneven pressure, overlapping strokes, and other environmental factors like lighting and noise in scanned images. Most of the existing methods rely on handcrafted features or rule-based approaches and face difficulties in generalization across divergent styles or learning new datasets. Such restrictions make such systems less efficient and scalable in real-world applications like document digitization and automatic data entry.

Convolutional Neural Networks provide a robust solution to overcome that challenge: the ability to automatically learn spatial features directly from images. Unlike traditional techniques, CNNs eliminate the need for user feature engineering and adaptively capture handwriting patterns, which include stroke direction, spacing, and character shape. This work takes advantage of the features of CNNs and designs a character recognition system that can be very accurate in identifying individual handwritten characters. The system will look at preprocessing input images, normalizing their dimensions, and applying data augmentation techniques to achieve robust performance across different conditions and datasets.

The proposed solution also focuses on scalability and applicability in the real world. Ensuring that the model can work with noisy inputs and various handwriting styles makes it able to be applied in diverse use cases, such as educational tools, automated form processing, and assistive technologies for visually impaired people. Implementation was made light yet effective, adding space in deploying the model as needed in real-time systems and ensuring accessibility by users from different domains.

To address these challenges effectively, the proposed system emphasizes not only achieving high accuracy but also ensuring adaptability and robustness. By employing a structured approach to dataset preparation and model design, the system is tailored to learn and generalize across a wide spectrum of handwriting styles. The preprocessing stage involves standardizing image inputs to a uniform format while eliminating noise and inconsistencies, ensuring that the data fed into the CNN is clean and representative. The inclusion of data augmentation techniques enhances the model's ability to handle variations in scale, orientation, and lighting conditions, which are common in real-world scenarios.

PROBLEM STATEMENT

The recognition of the handwritten character still remains challenging as each person's handwriting style is unique. A person is supposed to write characters of different styles, shapes, and sizes. Sometimes, uneven stroke pressures, overlapping characters, noise present in the scanned images, and inconsistent orientations complicate the task. In such systems, traditional character recognition systems including manual feature extraction as well as rule-based methods are not very generalizing upon diverse handwriting samples and make bad performances in realistic scenarios.

The existing solutions also lack robustness. They cannot cope with noisy or distorted inputs, particularly in conditions like poor scanning or handwritten text on unusual surfaces. Therefore, automated character recognition has not spread in the way it could have had wide applications such as digitization of archival documents, real-time transcription systems, and assistive technologies. The lack of a scalable, yet reliable, solution heightens inefficiency in those processes which necessarily entail the interpretation by hand of handwritten content.

This project addresses these limitations through the development of a model on CNNs for handwritten character recognition. This is suitable since image classification can learn spatial features directly from images, and thus obviate the need for any form of manual feature engineering. The proposed system attempts to classify characters from input images with high accuracy, notwithstanding the noise, irregularities, and variations in handwriting styles of the latter.

Develop a powerful model whose scalability can be used in real applications, like OCR, form processing, and teaching tools for handwriting. This is a project aimed at developing a solution that can handle the diversified and challenging handwriting datasets by making use of different techniques such as data augmentation, image preprocessing, and model optimization. This will work for higher efficiency, lesser manual effort, and progression in automation and accessibility.

DATASETS OVERVIEW

Data Preprocessing

Data preprocessing involves reshaping the input data into a format suitable for training the CNN model, normalizing pixel values, and converting labels into one-hot encoded format. In this project, it prepares the handwritten alphabet dataset to ensure compatibility with the model architecture and improve learning efficiency.

Loading the Dataset

The dataset, "A-Z Handwritten Data," is loaded from a CSV file using `pd.read_csv`. This dataset contains pixel values of handwritten characters from A to Z. Each row corresponds to a flattened 28x28 pixel image, and the first column represents the label, i.e., the character it corresponds to (ranging from 0 for 'A' to 25 for 'Z').

The `astype('float32')` is used to ensure that the pixel values are of type `float32`, which helps during the model training by ensuring consistent precision.

Splitting Data into Features and Labels:

The dataset is divided into two parts:

1. `X` (features): Contains the pixel values of the images, which are obtained by dropping the first column (which contains the labels).
2. `y` (labels): Contains the corresponding labels of the images, representing the characters.

Splitting the Dataset into Training and Testing Sets:

The dataset is then split into a training set and a testing set using the `train_test_split` function from `sklearn.model_selection`. This function ensures that 80% of the data is used for training, and 20% is reserved for testing the model's performance. This split is important to evaluate the model's ability to generalize on unseen data.

Reshaping the Data for Image Representation:

Since the model expects images to be in a 2D format (28x28 pixels), the pixel values are reshaped into 28x28 arrays for both the training and testing sets. Each image is represented by a 28x28 matrix of pixel values.

This reshaping is done to convert the data into the correct format that the neural network can process. The data is represented as a 3D array where each image has a shape of (28, 28), and the first dimension represents the number of images.

Visualizing the Distribution of Alphabet Classes:

To check the balance of the dataset, the frequency of each character label (from A to Z) is plotted. This gives insights into the distribution of characters and helps identify if there is any imbalance in the dataset that may affect the model's performance.

This part of the code counts how many times each character appears in the training data, and then the frequency distribution is plotted using a horizontal bar graph.

Shuffling the Data:

Shuffling the data helps ensure that the training process is not biased by the order in which the images appear in the dataset. It is important to shuffle the data to avoid the model learning any unintended patterns based on the order of data.

This step shuffles the first 100 images from the training set and displays them to visualize the variability in the handwriting styles.

Reshaping Data for CNN Input:

Before feeding the data into the Convolutional Neural Network (CNN), the images must be reshaped to include an additional dimension for the color channels. Since the images are grayscale, this dimension will have a size of 1.

Now, the shape of `train_X` and `test_X` will be `(number_of_images, 28, 28, 1)`, where 1 represents the grayscale channel.

Converting Labels to One-Hot Encoding:

Since this is a classification problem with 26 possible classes (one for each letter), the labels are converted into one-hot encoded vectors. This means that for each label, a vector is created where the index corresponding to the character is set to 1, and all other indices are set to 0.

For example, if the label is 'A', the one-hot encoded vector would be `[1, 0, 0, ..., 0]` where the first element is 1, indicating that the character is 'A'. This encoding helps the neural network in classification tasks by converting the problem into a multi-class classification task.

OBJECTIVES

This project will build a reliable and scalable handwritten character recognition system using Convolutional Neural Networks. The problem of handwriting character recognition is one of the key applications in computer vision, and the applications range from the digitization of preprinted and handwritten documents to real-time transcription and assistive technologies. The project takes into account traditional recognition techniques through the problem of handwriting variability, noise within image data, and a lack of generalizability by exploiting CNN-based deep learning techniques.

Derive an accurate character recognition model:

The first goal will then be to create a CNN-based model capable, with acceptable accuracy, of identifying the particular handwritten characters from their image inputs. It would train on a diversified dataset in which the model will then be tested to perform well over various handwriting types, sizes, and orientations. Accuracy will be one of the important indicators of success in this case, and standard performance metrics for the model would be accuracy, precision, recall, and F1-score.

Automate Feature Extraction:

Since the traditional techniques frequently used in handwritten character recognition depend on human feature extraction, they are very lengthy and error-prone. It has been proposed to use the power of CNNs, so as to bypass the feature extraction process that can lead to direct learning of the model from raw spatial and structural patterns available in the image data. Therefore, the handcrafted features are thus avoided, and it becomes more adaptive and efficient.

Improve robustness against variations:

This is one of the characteristic problems encountered while doing handwriting styles, strokes, and orientations. The model will include augmentation techniques such as random rotation, change in brightness, adding various kinds of noise into the sets to make robustness. The system will be equipped with the capacity to distinguish from varied, noisy inputs after training the augmented data sets.

Preprocess Images for Consistency:

From previous research, preprocessing is one of the critical components in improving a model's performance. In this project, techniques used included resizing the images to a fixed dimension, conversion of all images into grayscale and normalization of pixel values. These steps ensure input to the CNN is consistent with reduced complexity in computations, making training faster.

Optimize model architecture:

Another related goal is building an optimized CNN architecture specifically to be used for character recognition. The outcome of this results in trying on the number of convolutional layers, filter sizes, activation functions, and pooling strategies in such a way that balances accuracy with computational efficiency. Again, the full set of layers or classification heads also maps learned features to character labels.

Finally train and validate the model:

Training the model entails making use of labeled datasets of handwritten characters and optimizing the parameters of the network so as to minimize recognition errors. With regard to the project, splitting the dataset into training, validation, and testing sets will be focused on to evaluate the model objectively. Validation will be critical during implementation to ensure that the model does not overfit and generalizes well to unseen data.

Incorporate data augmentation techniques:

To make the model adaptable to different datasets, the project will explore adding multiple kinds of data augmentation techniques. Transformations introduced during the training process-including rotation, scaling, flipping, and noise addition-make a character recognizable in other orientations and under various conditions.

Enhance Scalability for Diverse Character Sets:

A critical extension of the project is ensuring the system's ability to scale seamlessly to support multiple character sets, such as different languages, symbols, or numerals. By designing the model to be flexible, the recognition system can be fine-tuned or retrained with minimal effort to accommodate characters from various writing systems, including cursive scripts or specialized domains like mathematical notations. This objective ensures the system remains versatile and applicable across global use cases.

Deploy the Model for Real-Time Applications:

The final objective is to adapt the trained model for deployment in real-time systems, ensuring efficient inference without compromising accuracy. This includes optimizing the model for deployment on resource-constrained devices, such as mobile phones or embedded systems, by leveraging techniques like quantization or model pruning. The system will also integrate user-friendly interfaces, enabling applications such as real-time transcription, document digitization, and handwriting-based input systems for assistive technology, thus broadening the scope of its practical impact.

METHODOLOGY

The systematic approach applied in this project is to crack the challenges of the handwritten character recognition problems. This clearly starts off with understanding the problem domain, preparing the dataset, designing the CNN architecture, training the model, followed by testing and then deploying the system. All these steps are mandatory to secure the final system to be very robust and efficient for real-world applications.

The dataset collection and organization is the first step. A diverse dataset with hand-written characters, which covers various styles, sizes, and orientations in writing. Illustrative examples would be IAM, MNIST, other datasets, or even ad-hoc created ones. The dataset is formed into labeled folders in which each directory is named after a character. This would indeed mean that the data will be accordingly structured for training. Data augmentation techniques such as random rotation, scaling, brightness adjustment, and adding noise etc; to artificially increase dataset variability for improving the model to generalize.

Next, data preprocessing is essential in making sure the input images are uniform and clean. The processing involves resizing to a uniform dimension, such as 32x32 pixels, to maintain compatibility with the CNN input layer. It significantly removes unimportant color information but retains the most important information, that is, computational complexity. Normalization scales pixel values in such a way that they fall in between 0 and 1; further, this makes faster and more stable model training possible. Moreover, the application of noise reduction methods like Gaussian filters diminishes distortions from input images, increasing robustness against real-world challenges.

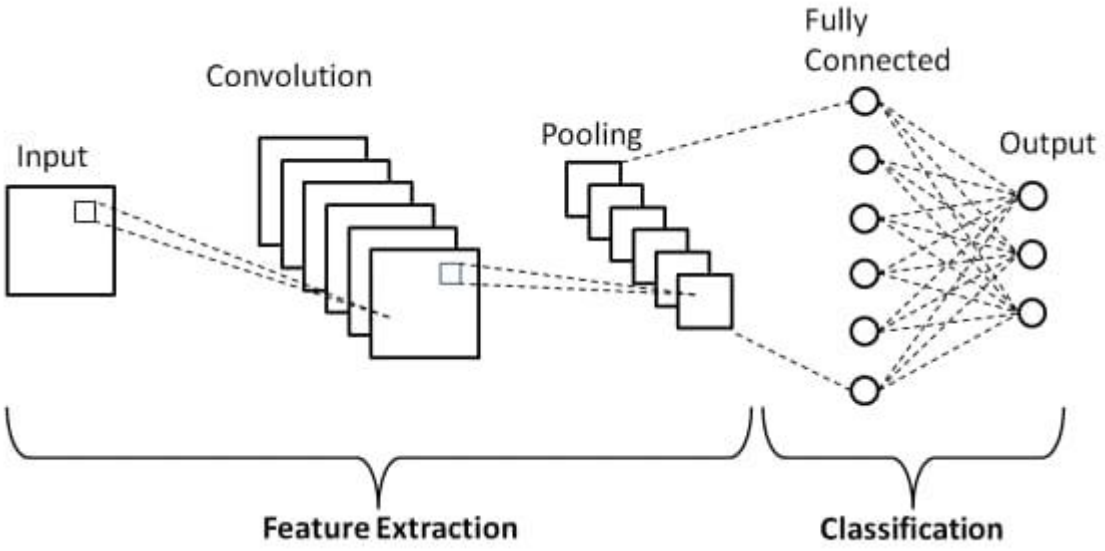
In this stage, the data is prepared, after which the project proceeds with the architecture design of CNN. The CNN is designed to extract descriptive features from the input images. Convolutional layers capture spatial features including edges and curves. Pooling layers downsample feature maps in order to balance the computational load. Fully connected layers interpret the extracted features to specific character classes. Dropout layers have also been incorporated in the architecture in order to prevent overfitting.

Output layer contains the softmax activation function, which assigns a probability to the class of every character it is trying to find, making this model appropriate for multi-class classification.

Error predictions in use With a categorical cross-entropy loss function, this model uses Adam optimizations to adjust the parameters. This model trains iteratively but focuses on measuring both the accuracy and the loss, avoiding overfitting and underfitting. The evaluation and deployment phase finally checks how well the models work through metrics of accuracy, precision, recall, and confusion matrices.

Once verified, it is optimized for deployment, making the model light and powerful enough for real-time applications, and it might apply model pruning or quantization to make the system suitable for resource-constrained environments. It would then be integrated into applications such as document digitization, education tools, or assistive technologies, thereby accomplishing its goals in the pragmatic world. This approach by the project ensures that the recognition system for handwritten characters is accurate, scalable, and adaptable to the many real-world challenges.

To further enhance the system's robustness and real-world applicability, the model undergoes iterative refinement during the deployment phase. Feedback from practical applications, such as real-time transcription or document processing, is used to fine-tune the model. This iterative loop allows the system to adapt to unforeseen challenges, such as handling exceptionally messy handwriting or recognizing characters in low-quality images. By continuously learning from deployment data, the system becomes more intelligent and adaptable over time, ensuring sustained performance in diverse scenarios.



MODEL ARCHITECTURE

The core of the handwritten character recognition system relies on a carefully designed Convolutional Neural Network (CNN) architecture. The choice of this architecture ensures efficient feature extraction and classification. This section outlines the layers, components, and rationale behind the CNN design used in this project.

Convolutional Layers

The architecture begins with multiple convolutional layers, which are responsible for extracting spatial features from input images. These layers detect low-level patterns such as edges and curves in the initial stages and progress to identifying higher-level patterns like character shapes in the later stages. Filters of size 3×33 \times 33×3 or 5×55 \times 55×5 are commonly employed, with strides of 1 and padding to preserve the spatial dimensions of the input.

Activation Function

Rectified Linear Unit (ReLU) activation is applied after each convolutional layer to introduce non-linearity. This function effectively helps in handling complex patterns by allowing the model to learn intricate features of handwritten characters.

Pooling Layers

Pooling layers, such as Max Pooling, are interleaved between convolutional layers to reduce the spatial dimensions of feature maps while retaining critical information. This reduces computational complexity and prevents overfitting. A pooling size of 2×22 \times 22×2 with a stride of 2 ensures a balanced reduction in dimensions.

Dropout Layers

To further mitigate overfitting, dropout layers are included, especially in deeper parts of the network. These layers randomly deactivate a fraction of neurons during training, forcing the model to generalize better.

Fully Connected Layers

The extracted features are flattened and passed through one or more fully connected layers. These layers map the features to character classes using learned weights. The final fully connected layer has as many neurons as the number of unique characters in the dataset.

Output Layer

A softmax activation function is used in the output layer to assign a probability to each class, enabling the system to classify input images into distinct character labels. The character with the highest probability is chosen as the prediction.

Optimization Techniques

The model employs the Adam optimizer, which combines the benefits of momentum and adaptive learning rates to achieve faster convergence. Categorical cross-entropy is used as the loss function to measure the discrepancy between predicted and true labels during training.

Hyperparameter Tuning

The model's performance was optimized by tuning hyperparameters such as the number of filters, kernel sizes, dropout rates, learning rate, and batch size. Grid search and random search techniques were applied to identify the best combination of these parameters.

Visualization of Architecture

A visualization of the CNN model’s architecture, showcasing the input, convolutional, pooling, and fully connected layers, is provided to help understand the flow of data through the network.This robust architecture is designed to balance accuracy and computational efficiency, making it suitable for real-world applications of handwritten character recognition

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}}$$

$$\text{Loss} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C} y_{ij} \cdot \log(\hat{y}_{ij})$$

RESULTS AND ANALYSIS

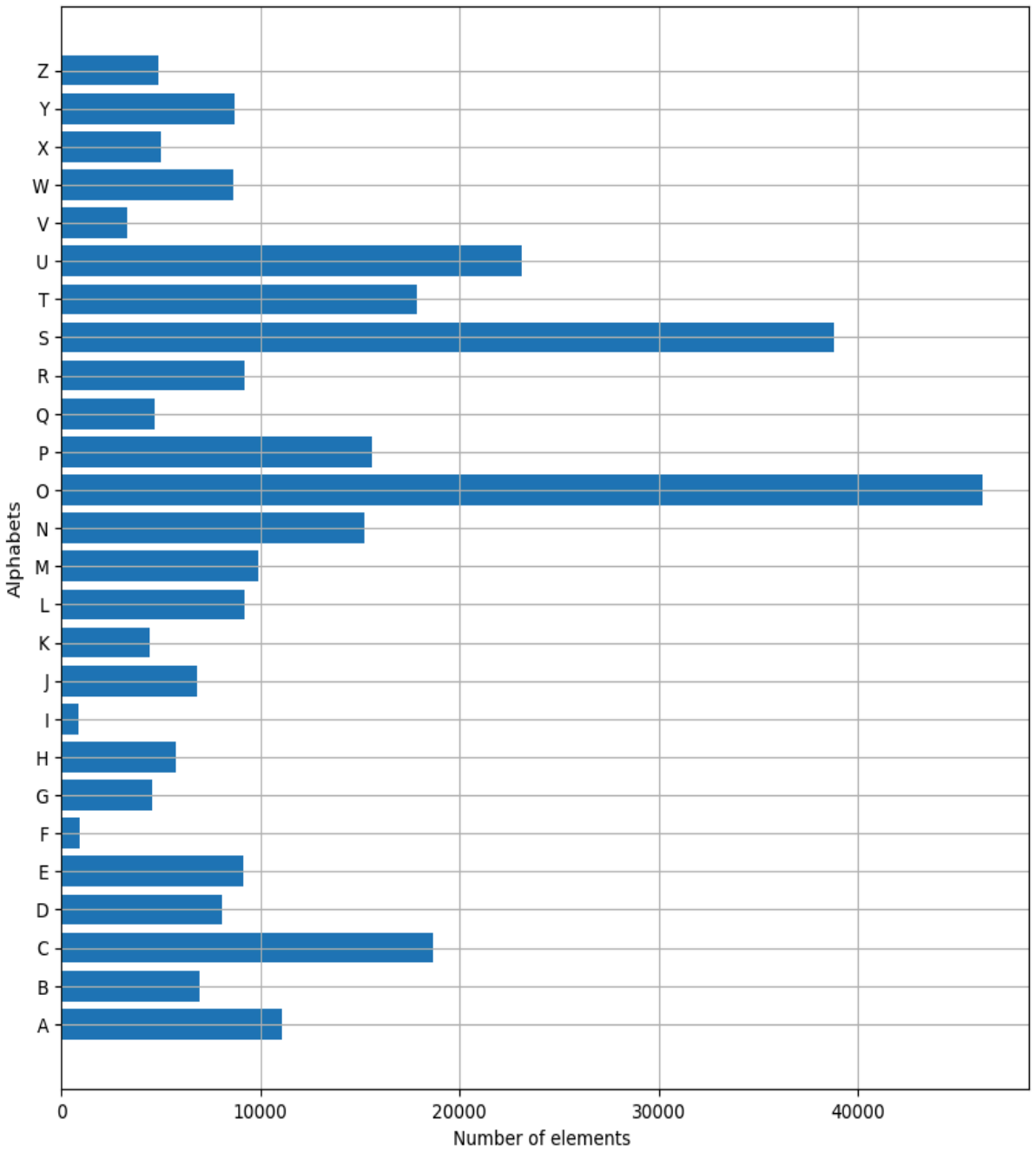


Fig.1. Horizontal Bar Chart

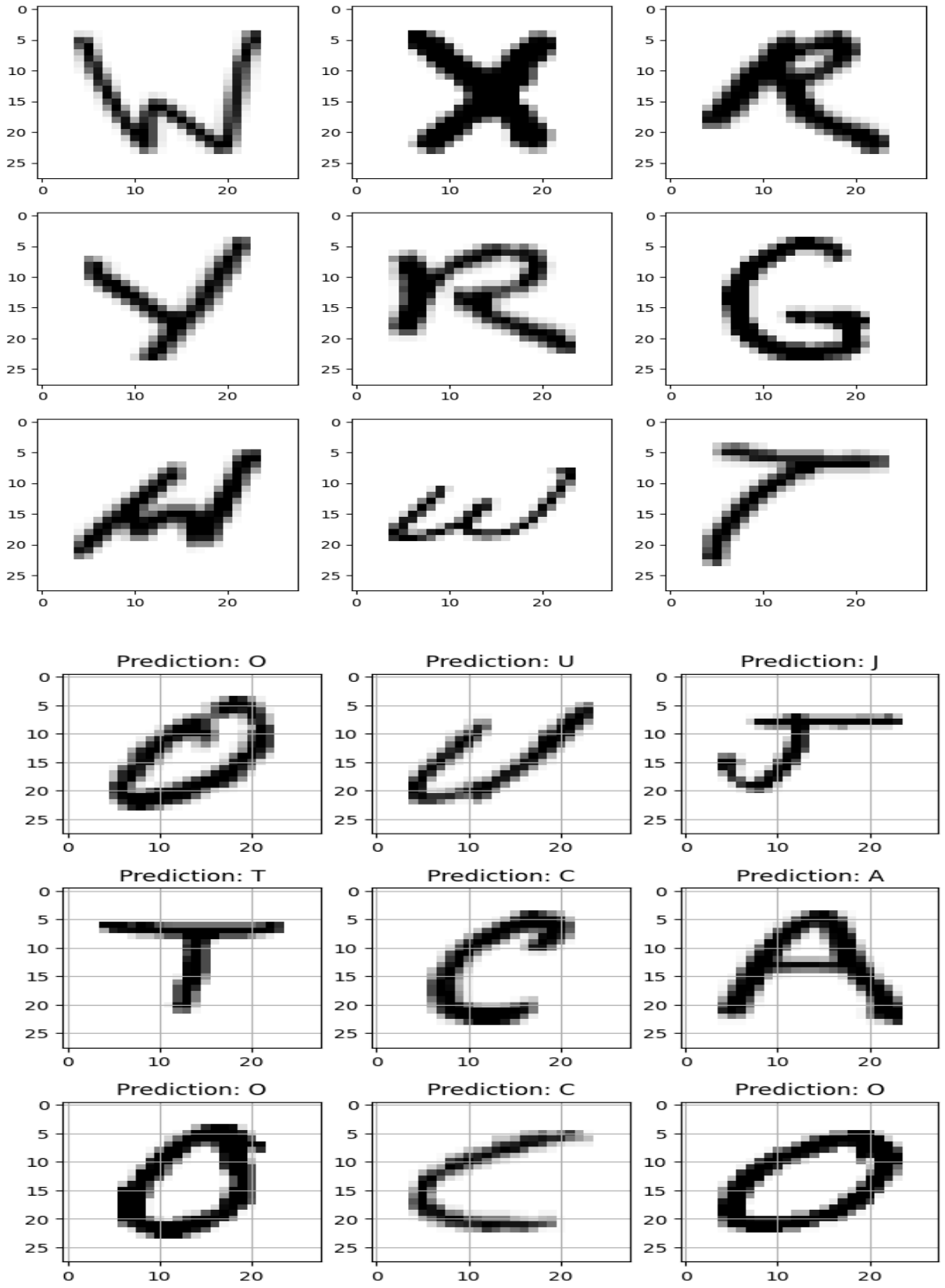


Fig.2. Predicted samples

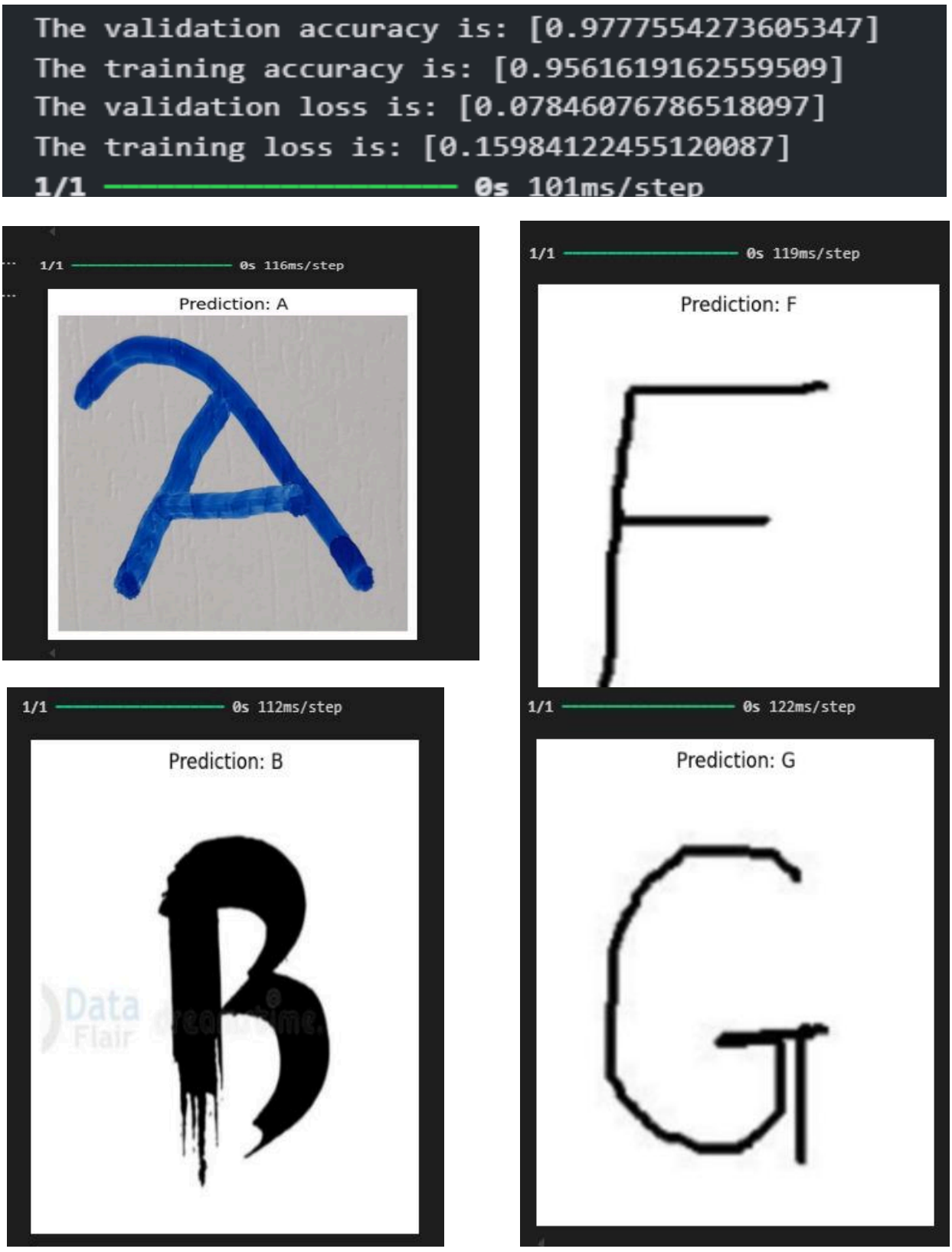


Fig.3. Predicted Outputs

FUTURE RESEARCH

Character recognition through handwritten sources is an exciting new field that constantly poses challenges to computer vision and machine learning. Though this project provides a solid foundation for how CNNs can be used, additional research avenues will fine-tune the capacity of the system, expand its usage, and bridge over the limitations.

One of the important directions for further research is multi-lingual and script recognition. The extension of the model to recognize characters more than one language and script-such as Arabic, Chinese, Devanagari-will enable the model to be much more versatile in real-world use. This would necessitate datasets that are larger and more diverse but crucial architecture changes within the model, so that it learns how to deal with unique features of different writing systems. Hybrid approaches like Transfer Learning can speed up the process while using pre-trained models for several scripts.

Another promising direction is that of improving the robustness to challenging inputs. Handwriting recognition systems, like many applications, must confront poor-quality inputs-blue and faded characters; overlapped characters. Future work may involve ways GANs may be brought in to enhance the quality of images used or attention mechanisms could concentrate on specific regions of the characters in images full of cluttered data. Such improvements would drastically improve real-world applications.

Another key area is contextual understanding. In the present system, characters are categorized in terms of isolation without any regard to word or sentence context. With sequence models such as Recurrent Neural Networks or Transformers, however, this system will allow for the ability to recognize characters within entire words or sentences. This approach also emulates the way in which humans read and enhances recognition over ambiguous characters.

These may also include real-time recognition systems running on resource-constrained devices. Techniques of model quantization, pruning, or distillation will be used on resource-constrained device use cases regarding CNN models: optimizing it to run edge computing platforms like mobile phones or embedded devices. These optimizations will contribute toward efficient, low-latency recognition for applications in education, healthcare, or even accessibility technologies

.

The development in unsupervised and semi-supervised learning opens up the possibility to reduce dependency on labeled data. The efforts of collecting and annotating datasets of handwriting are high cost and labor intensive. Unsupervised learning strategies can capitalize on voluminous amounts of unannotated data and semi-supervised methods use a few samples that are much more labeled to train high-performance models. This may democratize opportunities for low-resource languages and communities to handwritten recognition technologies.

In short, handwritten character recognition has much promise for the future. From dealing with different scripts and challenging inputs to optimizing performance in real time and understanding of context, lots of research can be done to take this technology forward. All these promise to produce more accurate, accessible, and integral recognition systems for applications in many industries.

Another exciting avenue for future research is the integration of multimodal learning techniques to enhance recognition accuracy. Combining handwriting recognition with other data modalities, such as voice input or context from surrounding text, can create a more robust and comprehensive system. For instance, pairing handwriting recognition with speech-to-text systems could assist in applications like live transcription, where errors in one modality can be corrected using data from another. This approach could further improve the model's ability to handle ambiguous or incomplete inputs.

Additionally, the incorporation of explainability and interpretability in character recognition systems is a critical area of research. While CNNs are highly effective, their "black-box" nature often makes it difficult to understand why a particular prediction was made. Developing techniques to visualize and interpret the learned features, such as heatmaps or activation maps, can provide valuable insights into the decision-making process. This not only improves trust in the system but also aids in debugging and refining the model for specific use cases.

Lastly, exploring cross-domain applications of handwriting recognition could open up novel research opportunities. Beyond traditional uses like document digitization, such systems could be employed in creative fields, such as analyzing historical manuscripts or aiding artists in converting sketches into digital formats. Research into domain adaptation methods would allow models trained on one type of handwriting to generalize effectively to entirely different datasets, making handwriting recognition systems versatile and impactful across a wide range of industries.

CODE IMPLEMENTATION

1) The code resolves `google.protobuf` errors by upgrading and adjusting the `protobuf` version based on the TensorFlow version, and attempts reinstallation if necessary.

```
# Step-by-step resolution for the `google.protobuf` error in one cell

try:

    # Upgrade protobuf

    !pip install --upgrade protobuf



    # Check TensorFlow version

    import tensorflow as tf

    print(f"TensorFlow version: {tf.__version__}")



    # Adjust protobuf version if needed (example: change
<compatible_version> based on TensorFlow version)

    tf_version = tf.__version__

    if tf_version.startswith("2.13"):

        !pip install protobuf<4.0.0

    elif tf_version.startswith("2.14") or
tf_version.startswith("2.15"):

        !pip install protobuf>=4.0.0



    # Verify protobuf installation

    from google.protobuf import message

    print("Protobuf installed and working successfully!")



except ModuleNotFoundError as e:
```

```
    print(f"Error: {e}. Trying to reinstall TensorFlow and Keras...")



    # Clear cache

    !pip cache purge



    # Reinstall TensorFlow and Keras

    !pip uninstall tensorflow keras -y

    !pip install tensorflow keras



    # Verify protobuf again

    try:

        from google.protobuf import message

        print("Protobuf installed and working successfully after
reinstallation!")

    except ModuleNotFoundError as e:

        print(f"Error persists: {e}.")
```

2) The code fixes `google.protobuf` issues by uninstalling, clearing the cache, reinstalling `protobuf`, upgrading TensorFlow and Keras, and verifying the installation.

```
# 1. Uninstall any corrupted or existing protobuf installation

!pip uninstall protobuf -y
```

```
# 2. Clear pip cache to avoid using cached corrupted files

!pip cache purge



# 3. Reinstall protobuf

!pip install protobuf



!pip install --upgrade keras tensorflow







# 4. Verify installation

try:

    from google.protobuf import message

    print("Protobuf installed and working successfully!")

except ModuleNotFoundError as e:

    print(f"Error: {e}")
```

3) This code implements a Convolutional Neural Network (CNN) to classify handwritten English alphabet characters from images, including training the model on a dataset, evaluating its performance, and predicting characters from external images.

```
import cv2

import numpy as np

from keras.models import load_model

import matplotlib.pyplot as plt

from tensorflow.keras.utils import to_categorical

import pandas as pd
```

```
from sklearn.model_selection import train_test_split

from sklearn.utils import shuffle

from tqdm.notebook import tqdm  # Use tqdm.notebook in Jupyter
Notebooks

from keras.datasets import mnist

import matplotlib.pyplot as plt

import cv2

import numpy as np

from keras.models import Sequential  # Ensure Sequential is imported
here

from keras.layers import Dense, Flatten, Conv2D, MaxPool2D, Dropout

from keras.optimizers import SGD, Adam

from keras.callbacks import ReduceLROnPlateau, EarlyStopping

from tensorflow.keras.utils import to_categorical  # Updated import

import pandas as pd

from sklearn.model_selection import train_test_split

from tqdm.notebook import tqdm  # Use tqdm.notebook in Jupyter
Notebooks

from sklearn.utils import shuffle

import os




data = pd.read_csv(r"C:\Users\prane\Downloads\IDP\A_Z Handwritten
Data.csv").astype('float32')




X = data.drop('0', axis=1)
```

```
y = data['0']



train_x, test_x, train_y, test_y = train_test_split(X, y,
test_size=0.2)

train_x = np.reshape(train_x.values, (train_x.shape[0], 28, 28))

test_x = np.reshape(test_x.values, (test_x.shape[0], 28, 28))



print("Train data shape: ", train_x.shape)

print("Test data shape: ", test_x.shape)



# Dictionary for getting characters from index values...

word_dict = {0: 'A', 1: 'B', 2: 'C', 3: 'D', 4: 'E', 5: 'F', 6: 'G', 7:
'H', 8: 'I', 9: 'J',

             10: 'K', 11: 'L', 12: 'M', 13: 'N', 14: 'O', 15: 'P', 16:
'Q', 17: 'R', 18: 'S',

             19: 'T', 20: 'U', 21: 'V', 22: 'W', 23: 'X', 24: 'Y', 25:
'Z'}



train_yint = train_y.astype(int)

count = np.zeros(26, dtype='int')

for i in train_yint:

    count[i] += 1



alphabets = [word_dict[i] for i in range(26)]



fig, ax = plt.subplots(1, 1, figsize=(10, 10))
```

```
ax.barh(alphabets, count)

plt.xlabel("Number of elements ")

plt.ylabel("Alphabets")

plt.grid()

plt.show()



shuff = shuffle(train_x[:100])



fig, ax = plt.subplots(3, 3, figsize=(10, 10))

axes = ax.flatten()



for i in range(9):

    axes[i].imshow(np.reshape(shuff[i], (28, 28)), cmap="Greys")

plt.show()



train_X = train_x.reshape(train_x.shape[0], train_x.shape[1],
train_x.shape[2], 1)

print("New shape of train data: ", train_X.shape)



test_X = test_x.reshape(test_x.shape[0], test_x.shape[1],
test_x.shape[2], 1)

print("New shape of test data: ", test_X.shape)



train_yOHE = to_categorical(train_y, num_classes=26)  # Removed
dtype='int'

print("New shape of train labels: ", train_yOHE.shape)
```

```
test_yOHE = to_categorical(test_y, num_classes=26)  # Removed
dtype='int'

print("New shape of test labels: ", test_yOHE.shape)


model = Sequential()


model.add(Conv2D(filters=32, kernel_size=(3, 3), activation='relu',
input_shape=(28, 28, 1)))

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


model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu',
padding='same'))

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


model.add(Conv2D(filters=128, kernel_size=(3, 3), activation='relu',
padding='valid'))

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


model.add(Flatten())


model.add(Dense(64, activation="relu"))

model.add(Dense(128, activation="relu"))


model.add(Dense(26, activation="softmax"))


model.compile(optimizer=Adam(learning_rate=0.001),
```

```
loss='categorical_crossentropy', metrics=['accuracy'])


history = model.fit(train_X, train_yOHE, epochs=1,
validation_data=(test_X, test_yOHE))


# Save model

model.save(r'model_hand.h5')


# Displaying the accuracies & losses for train & validation set...

print("The validation accuracy is:", history.history['val_accuracy'])

print("The training accuracy is:", history.history['accuracy'])

print("The validation loss is:", history.history['val_loss'])

print("The training loss is:", history.history['loss'])


# Making model predictions...

pred = model.predict(test_X[:9])


# Displaying some of the test images & their predicted labels...

fig, axes = plt.subplots(3, 3, figsize=(8, 9))

axes = axes.flatten()


for i, ax in enumerate(axes):

    img = np.reshape(test_X[i], (28, 28))

    ax.imshow(img, cmap="Greys")

    pred_label = word_dict[np.argmax(pred[i])]
```

```
    ax.set_title(f"Prediction: {pred_label}")
    ax.grid()

plt.show()

# Image prediction for external image...
img_path = r"C:\Users\prane\Downloads\A image.jpg"
# Read the image using OpenCV
img = cv2.imread(img_path)

if img is None:
    print("Error: Unable to load image.")
else:
    # Convert the image to RGB (OpenCV loads it in BGR by default)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img = cv2.resize(img, (400, 440))  # Resize the image

    # Preprocess the image for prediction
    img_copy = img.copy()
    img_copy = cv2.GaussianBlur(img_copy, (7, 7), 0)
    img_gray = cv2.cvtColor(img_copy, cv2.COLOR_BGR2GRAY)
    _, img_thresh = cv2.threshold(img_gray, 100, 255,
cv2.THRESH_BINARY_INV)
    img_final = cv2.resize(img_thresh, (28, 28))
    img_final = np.reshape(img_final, (1, 28, 28, 1))
```

```
# Make prediction
img_pred = word_dict[np.argmax(model.predict(img_final))]


# Display the image and prediction using Matplotlib
plt.imshow(img)
plt.title(f"Prediction: {img_pred}")
plt.axis('off')  # Hide axes for better image display
plt.show()


#
```

REFERENCES

1.Patel, Sachin. *AZ Handwritten Alphabets in CSV Format*. Kaggle, 2021. https://www.kaggle.com/datasets/sachinpatel21/az-handwritten-alphabets-in-csv-format

2."Convolutional Neural Network (CNN) in Machine Learning." GeeksforGeeks, 29 Sept. 2020, https://www.geeksforgeeks.org/convolutional-neural-network-cnn-in-machine-learning/.

3."Train and Test Datasets in Machine Learning." Javatpoint, https://www.javatpoint.com/train-and-test-datasets-in-machine-learning.

4."Neural Networks: A Beginner's Guide." GeeksforGeeks, 18 Nov. 2020, https://www.geeksforgeeks.org/neural-networks-a-beginners-guide/.

5.”Convolutional neural network” Wiki

https://en.wikipedia.org/wiki/Convolutional_neural_network

6.”How CNN Works” Youtube

https://www.youtube.com/playlist?list=PLuhqtP7jdD8CD6rOWy20INGM44kULvrHu

7. “Book about CNN” Google

https://www.deeplearningbook.org/contents/convnets.html

This book provides the understanding of Operations ,Pooling ,layers and How CNN works

8.Géron, A. (2022). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems* (3rd ed.). O'Reilly Media.

This book provides practical insights into building machine learning systems, including techniques for model evaluation and optimization used in this project.