Machine Learning Analysis of Tirhuta Lipi

Sumit Yadav(076bct088)
Raju Kumar Yadav(076bct100)
Prashant Bhandari(076bct049)

INTRODUCTION

Tirhuta Lipi is a script that is used to write the Tirhuta language, which is primarily spoken in the Indian states of Bihar and the Terai Belt of Nepal

In this project, we propose a machine learning-based character recognition system for Tirhuta Lipi that aims to address this challenge

Our objective is to build a machine learning model that can accurately recognize individual characters in Tirhuta Lipi text with high precision and recall

Background

Tirhuta Lipi is a script used to write the Tirhuta language, which has roots in the ancient Sanskrit language

Tirhuta Lipi is primarily used in Mithila region of India and Nepal, where it has been used for religious and literary purposes for centuries

As a result, Tirhuta Lipi has received relatively little attention in the field of natural language processing and language technology

Objectives

The objective of this project is to develop a machine learning-based character recognition system for Tirhuta Lipi that can accurately recognize individual characters in Tirhuta Lipi text with a high level of precision and recall

We aim to achieve this by collecting a large and diverse dataset of Tirhuta Lipi text and using it to train a machine learning and neural network based model

Problem statement

Tirhuta Lipi is a low resource language that is not widely used in modern times, making it difficult to use modern computational techniques to study and analyze the language

The lack of resources for processing text in Tirhuta Lipi has limited its potential for use in modern contexts, and has hindered efforts to study and preserve the language

Scope of Project

The scope of this project is to develop a machine learning-based character recognition system for Tirhuta Lipi, which has the potential to contribute to the existing body of knowledge on the topic

This study aims to provide new insights, perspectives, and solutions to the research problem, with the ultimate goal of informing policy decisions, guiding future research, and improving practices related to Tirhuta Lipi language technology

LITERATURE REVIEW

Tirhuta Lipi is a unique script that is primarily used for writing the Tirhuta language, which is spoken in the Mithila region of India and Nepal

Researchers have explored the use of machine learning techniques for language identification, part-of- speech tagging, and sentiment analysis, among other tasks

Our literature review shows that our proposed project is timely and relevant, and has the potential to contribute to the growing body of research on natural language processing for low resource languages

THEORETICAL BACKGROUND

Tirhuta Lipi is a script used for writing the Tirhuta language, which is a member of the Indo- Aryan language family

The script has 14 vowels and 38 consonants, along with a number of diacritics and modifiers

The project will use a machine learning-based approach to recognize individual characters in Tirhuta vowels Lipi text

TOOLS AND TECHNOLOGIES

Python: Python is a widely-used programming language with a rich set of libraries and frameworks for machine learning and natural language processing

Scikit-learn : Sklearn is a popular open-source machine learning library for Python

TensorFlow: TensorFlow is an open-source machine learning framework developed by Google that can be used to build neural networks and other machine learning models

Keras: Keras is a high-level neural network API that is built on top of TensorFlow

OpenCV: OpenCV is an open-source computer vision library that provides a wide range of image processing and computer vision algorithms

Flask: Flask is a lightweight web framework for Python that can be used to build web- based applications

METHODOLOGY

To achieve our objective of building a machine learning-based character recognition system for Tirhuta Lipi, we will follow the methodology described below

Data Collection Process

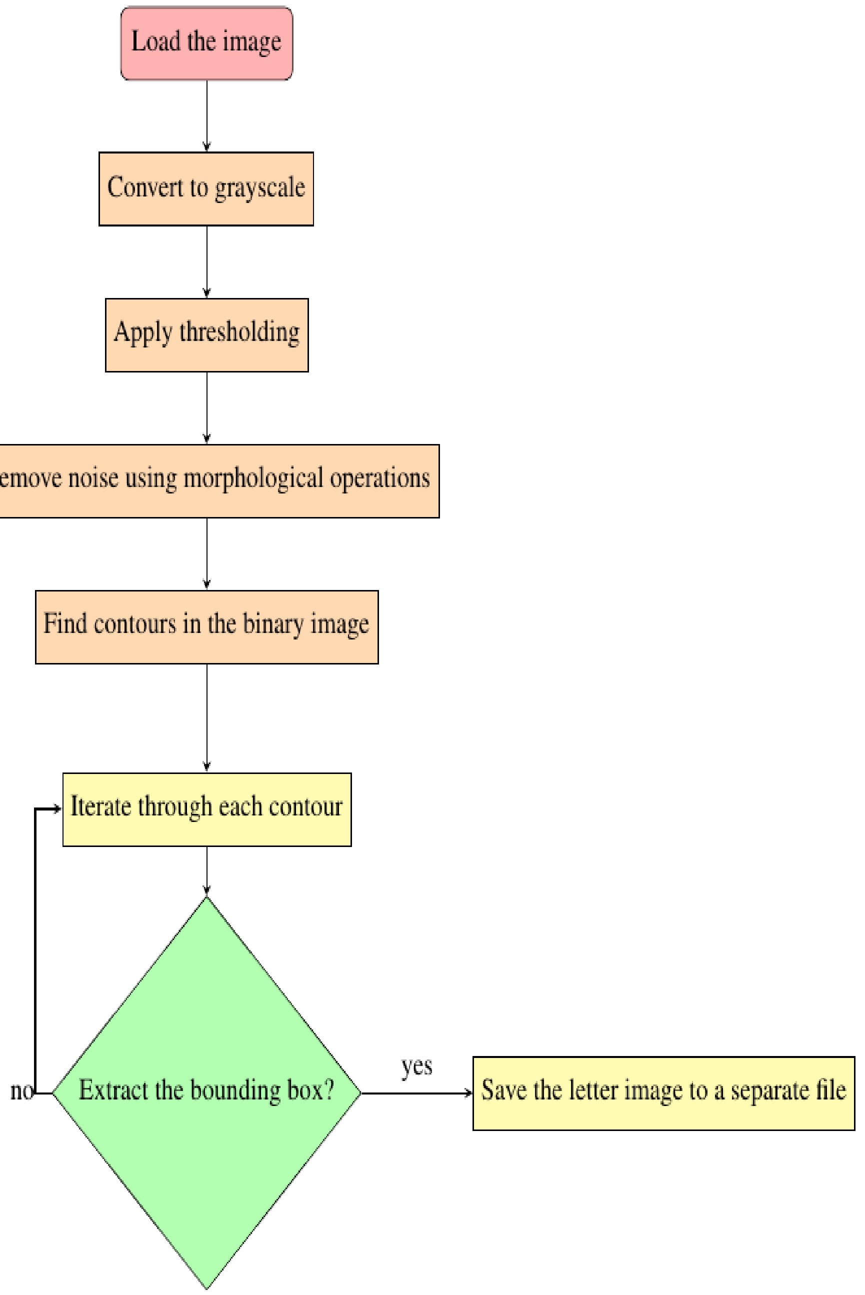
We will collect a large and diverse dataset of Tirhuta Lipi text from various sources, including books, manuscripts, and by writing by our self

The dataset will be carefully curated to ensure that it covers a broad range of topics and styles of writing, including handwritten and printed text

We will also ensure that the dataset includes a variety of fonts and styles of Tirhuta Lipi writing

The flowchart given below describes the steps involved in preprocessing raw images to ex- tract individual characters

Flow Chart


Load the image
Convert to grayscale
Apply thresholding
Remove noise using morphological operations
Find contours in the binary image
Iterate through each contour
Extract the bounding box?
no
yes
Save the letter image to a separate file

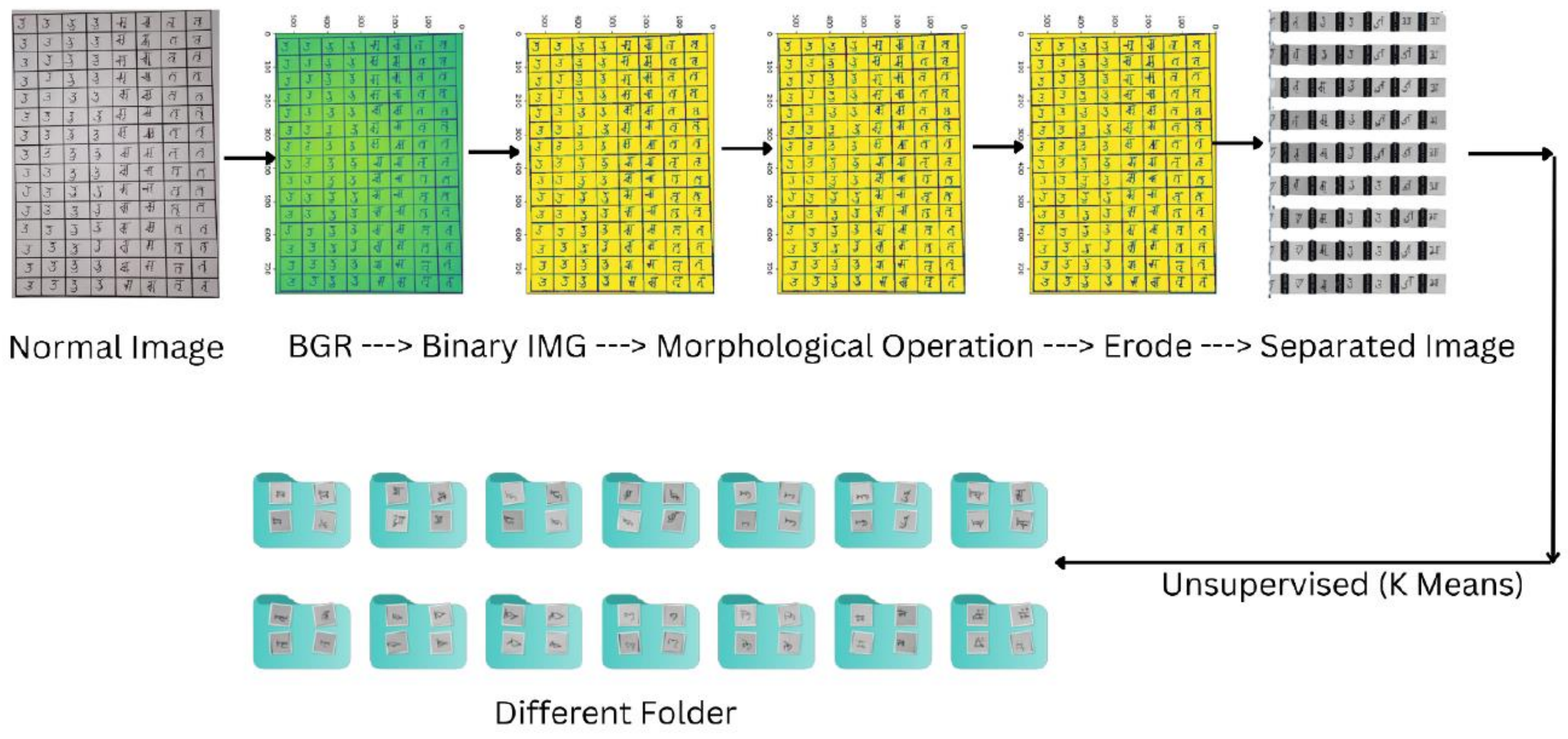

Normal Image
BGR ---> Binary IMG ---> Morphological Operation ---> Erode ---> Separated Image
Unsupervised (K Means)
Different Folder

Figure 10.2: Data Collection Process

Preprocessing and Cleaning of Data

In this step, we aim to preprocess and clean the collected Tirhuta Lipi dataset to make it suitable for machine learning

The primary objective of this step is to prepare a standardized dataset that can be used to train and test our machine learning models

The first step in this process is to remove any non-Tirhuta Lipi characters from the dataset

Data Analysis and Visualization Techniques

In this section, we have performed various data analysis and visualization techniques on our preprocessed dataset

These techniques are aimed at gaining a deeper understanding of the dataset and identifying any patterns or trends that may be present

Specifically, we have done the following

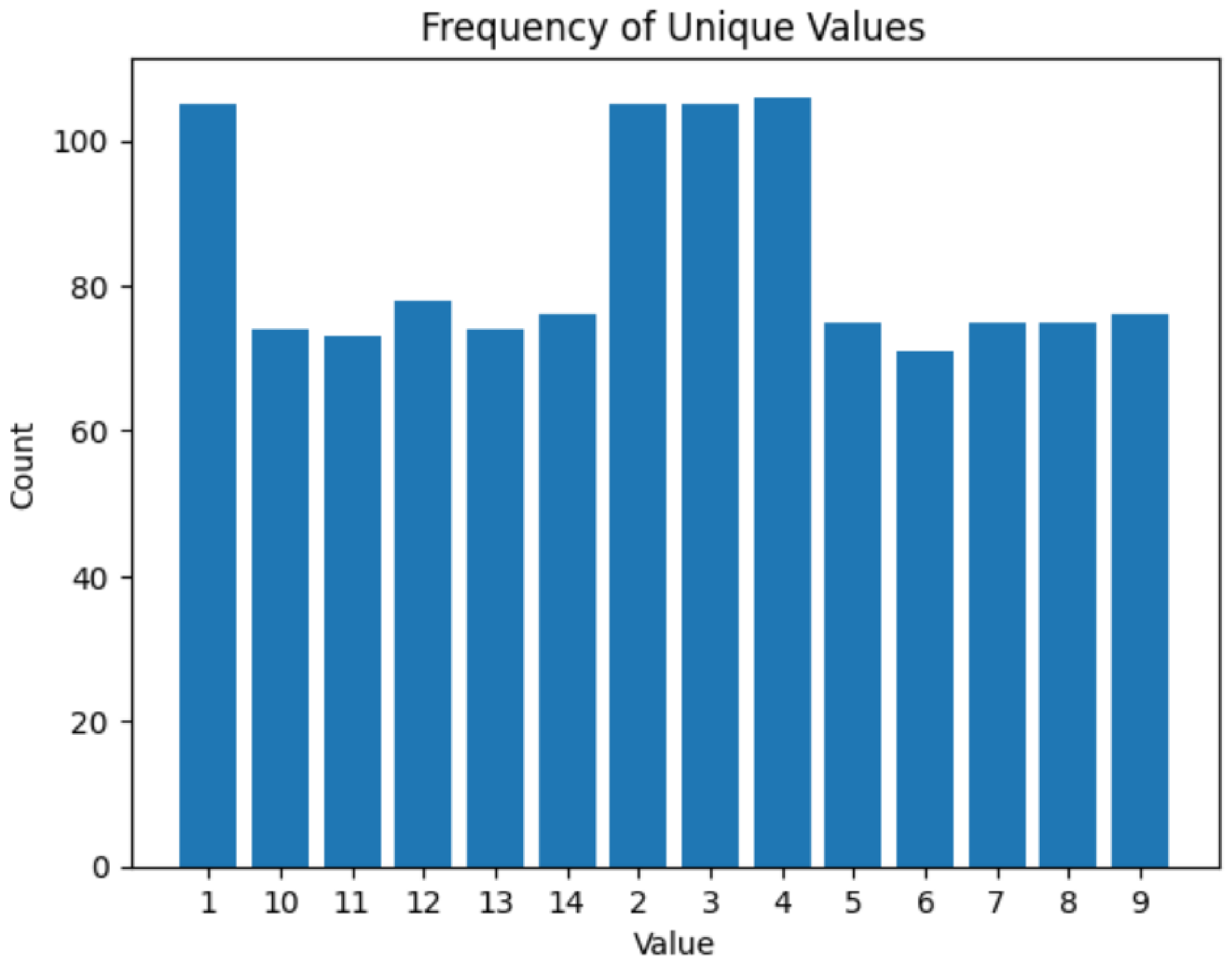


Figure 10.6: Images Counts.

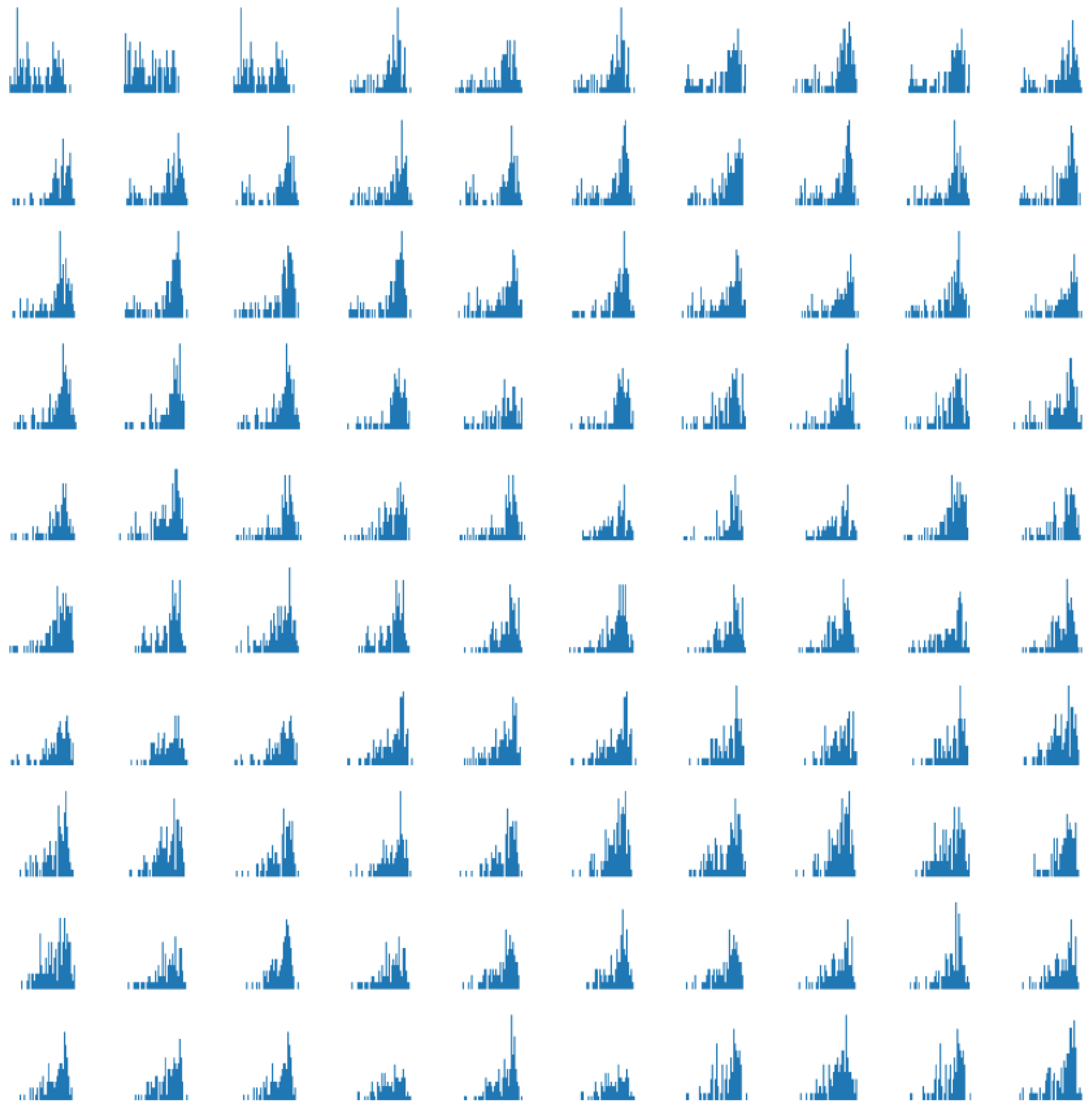


Figure 10.7: Pixel Distribution of Images (Histogram).

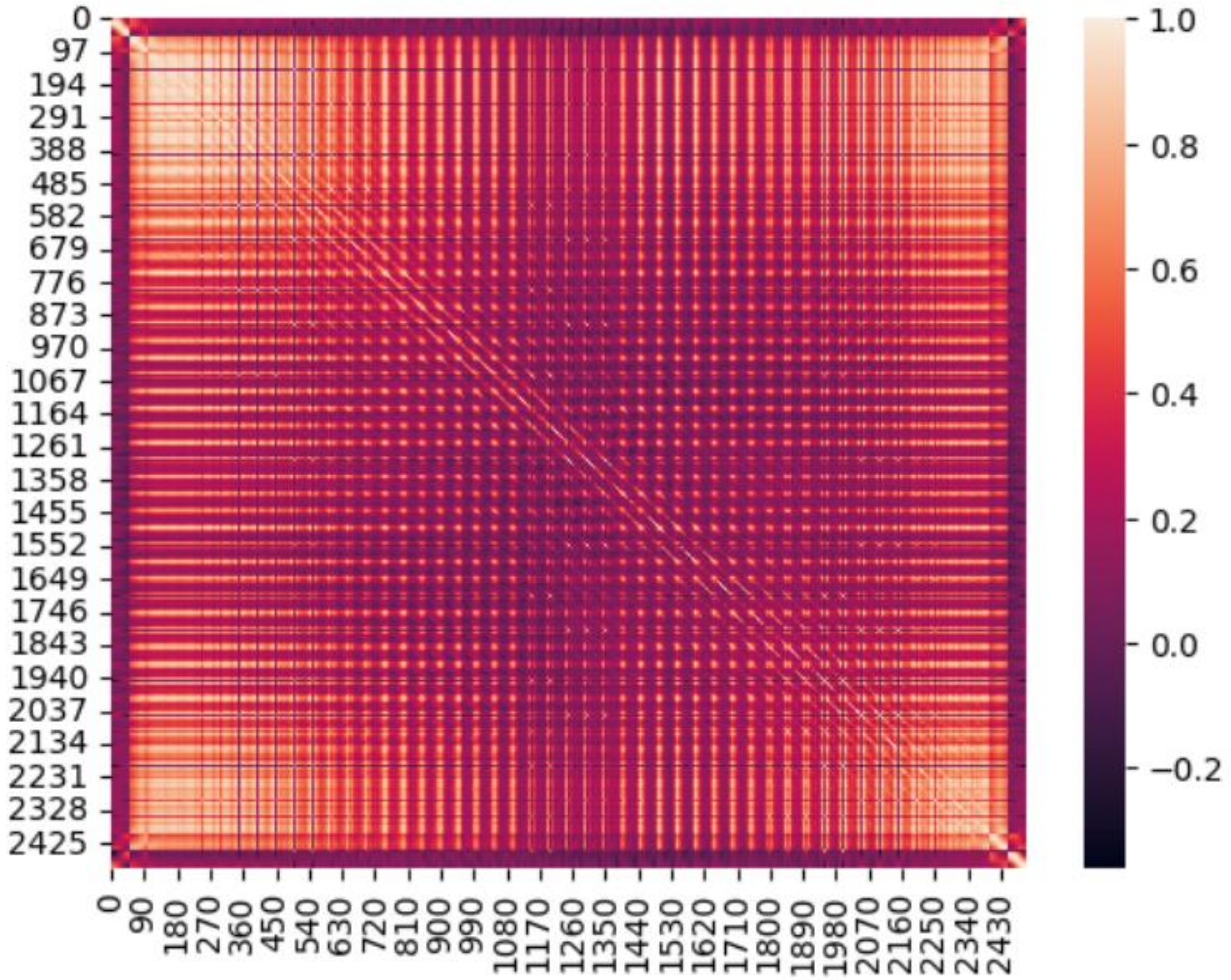


Figure 10.8: Correlation between features.

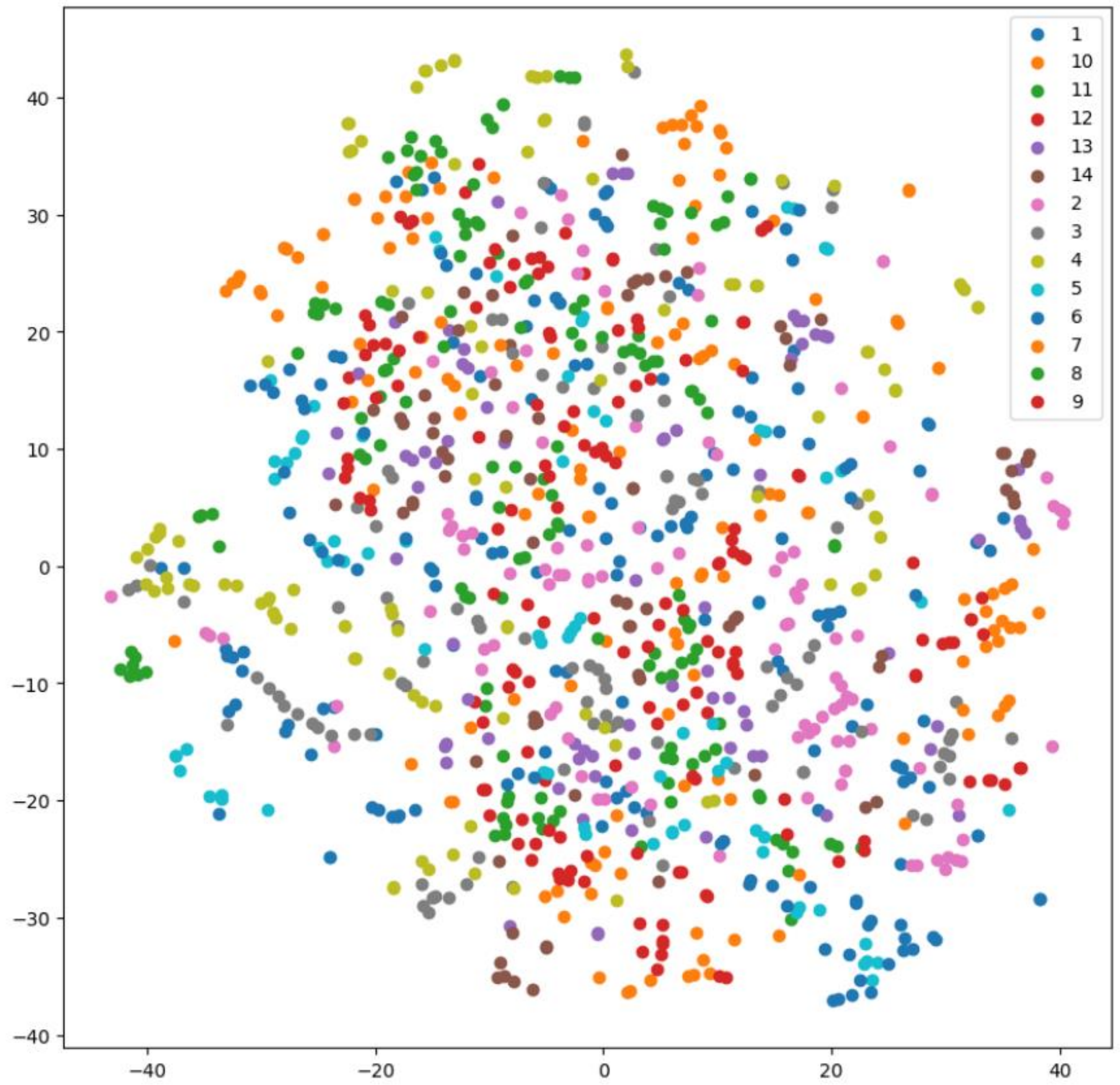


Figure 10.9: 2D TSNE.

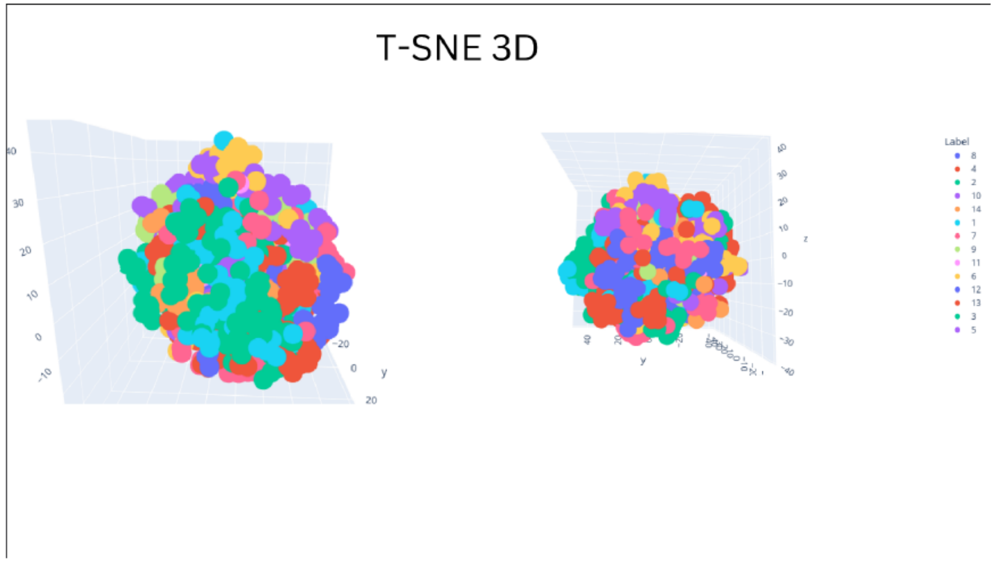


Figure 10.10: 3D T-SNE.

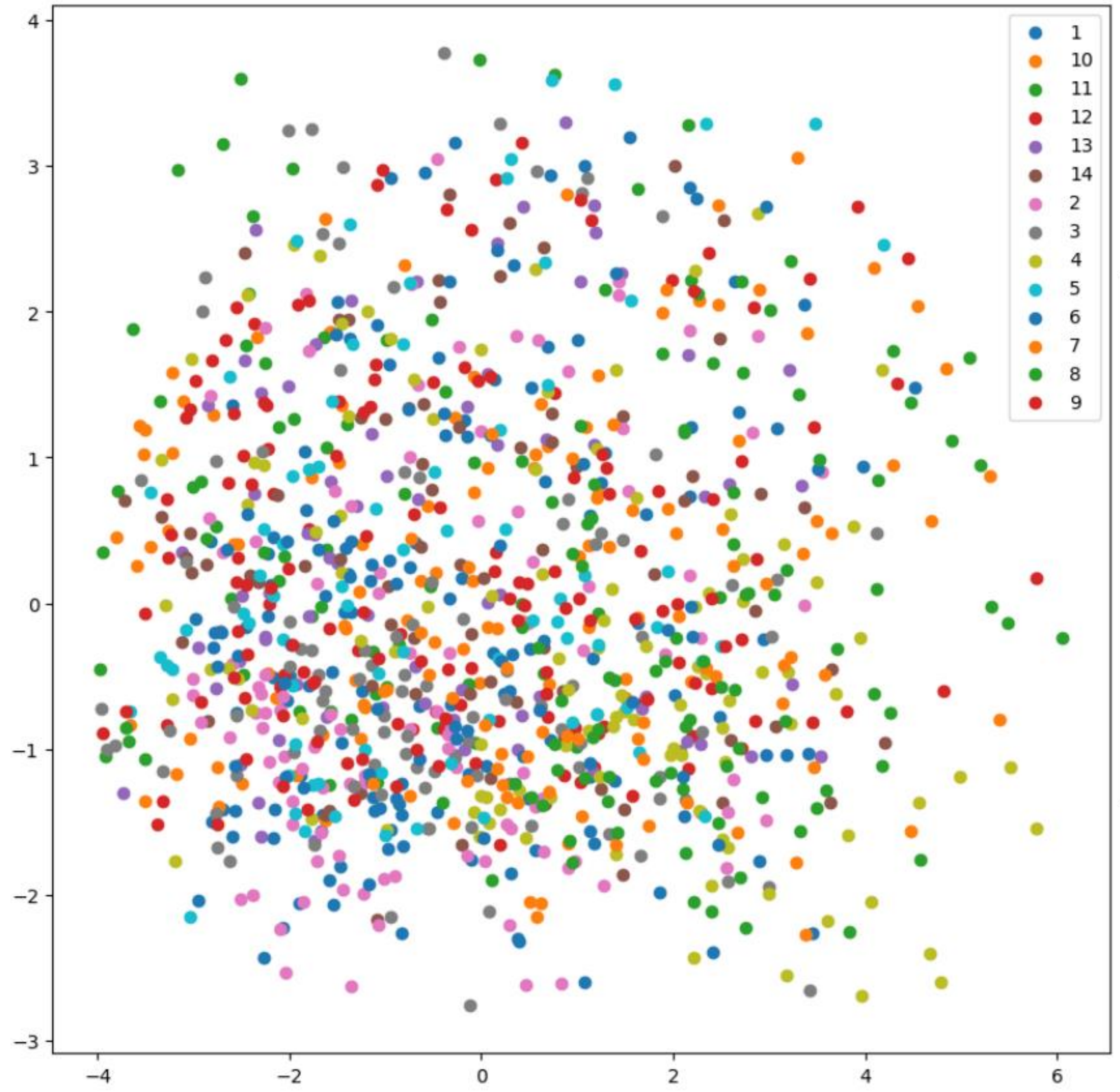


Figure 10.11: 2D PCA.

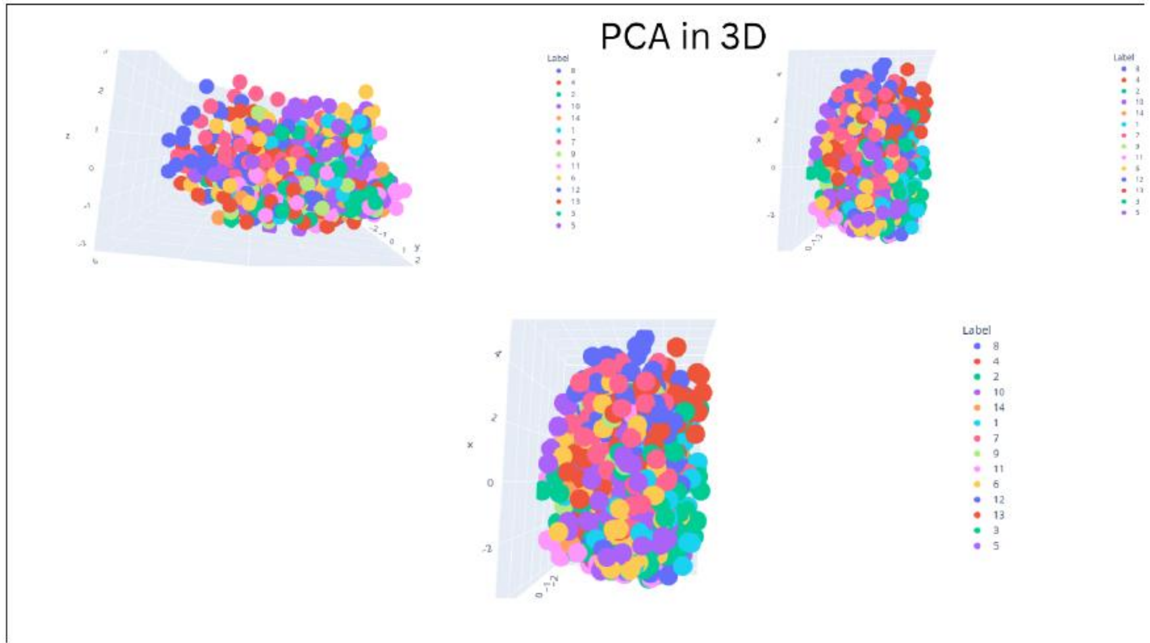


Figure 10.12: 3D PCA.

Machine Learning Algorithms Used

In this project, we have developed a machine learning algorithm from scratch for Tirhuta Lipi character recognition

To improve the accuracy, we performed dimensionality reduction on the images using PCA and passed 200 embeddings of the whole images into the Decision Tree Classifier

Overall, we have used decision tree classifier algorithm and implemented various techniques such as PCA and transfer learning to improve the accuracy of our algorithm

Decision Tree Classification

Decision tree is a popular machine learning algorithm used for classification and regression tasks

It is a tree-structured model where internal nodes represent features, branches repre- sent decisions based on the values of these features, and leaf nodes represent the output or class label

The DecisionTreeClassifier class provided in the code implements the decision tree algorithm for classification tasks

The decision tree is grown recursively using a top-down, greedy approach

The algorithm continues to split the data at each internal node until a stopping criterion is met, such as reaching a maximum depth or having a minimum number of samples in a leaf node

Decision Tree Classification

```
class Node:
    def __init__(self):
        self.feature = feature
        self.threshold = threshold
        self.left = left
        self.right = right
        self.value = value
        self.class_probs = class_probs
```

This code defines a class 'Node' that represents a node in a decision tree. A decision tree is a tree-like model used for classification and regression tasks. Each node in the tree represents a split in the data based on a feature and threshold value.

Decision Tree Classification

```
def _entropy(self, y):
        # calculate class probabilities
        y = y.astype(np.int)
        counts = np.bincount(y, minlength=self.n_classes)
        probs = counts / len(y)
        # calculate entropy
        entropy = -np.sum(probs * np.log2(probs + 1e-10))
        return entropy
```

This code defines a private method called **_entropy** inside the **DecisionTreeClassifier** class. The purpose of this method is to calculate the entropy of a set of labels **y**. Entropy is a measure of the impurity or disorder of a set of labels. In the context of decision trees, entropy is commonly used as a criterion to determine the best split for a given node.

Decision Tree Classification

```
def _information_gain(self, y, feature, threshold):
        parent_entropy = self._entropy(y)
        left_indices = feature < threshold
        right_indices = feature >= threshold
        left_entropy = self._entropy(y[left_indices])
        right_entropy = self._entropy(y[right_indices])
        n_left, n_right = len(y[left_indices]), len(y[right_indices])
        child_entropy = (n_left / len(y)) * left_entropy +
                        (n_right/ len(y)) * right_entropy
        gain = parent_entropy - child_entropy
        return gain
```

This code calculates the information gain of a split in a decision tree. Information gain measures how much a given feature and threshold split reduces the entropy (i.e., disorder or uncertainty) of the target variable (y) in comparison to the entropy of the original dataset.

Decision Tree Classification

```
def _traverse_tree(self, x, node):
        if node.class_probs is not None:
            return node.class_probs
        if x[node.feature] < node.threshold:
            return self._traverse_tree(x, node.left)
        else:
            return self._traverse_tree(x, node.right)
```

This code represents the implementation of the tree traversal algorithm. Given a new data point **x** and a node of the tree, the algorithm recursively traverses the tree to determine the leaf node that corresponds to the data point.

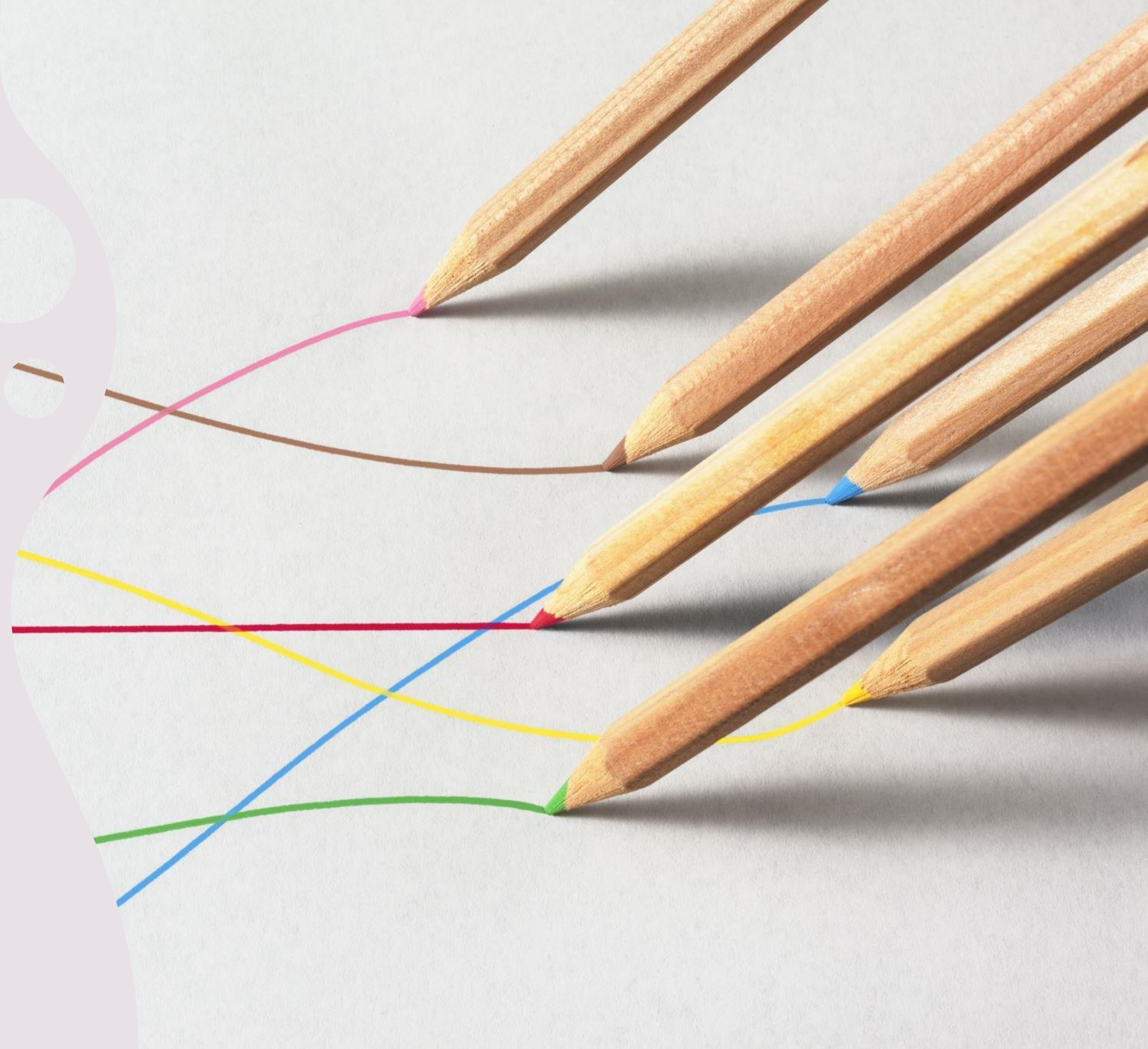
```
def _grow_tree(self, X, y, depth=0):

        n_samples, n_features = X.shape

        n_labels = len(np.unique(y))

        if depth == self.max_depth or n_labels == 1:

            y = y.astype(np.int)

            class_probs = np.bincount(y, minlength=self.n_classes) / len(y)

            return
Node(class_probs=class_probs)

        best_gain = -1

        for feature in range(n_features):

            thresholds = np.unique(X[:, feature])

            for threshold in thresholds:

                gain = self._information_gain(y, X[:, feature], threshold)

                if gain > best_gain:

                    best_gain = gain

                    best_feature = feature

                    best_threshold = threshold

        left_indices = X[:, best_feature] < best_threshold

        right_indices = X[:, best_feature] >= best_threshold

        left = self._grow_tree(X[left_indices, :], y[left_indices], depth+1)

        right = self._grow_tree(X[right_indices, :], y[right_indices], depth+1)

        return Node(feature=best_feature,
threshold=best_threshold,                           left=left,right=right)
```

This code represents the implementation of a decision tree algorithm for classification problems. Here is a brief overview of the code:

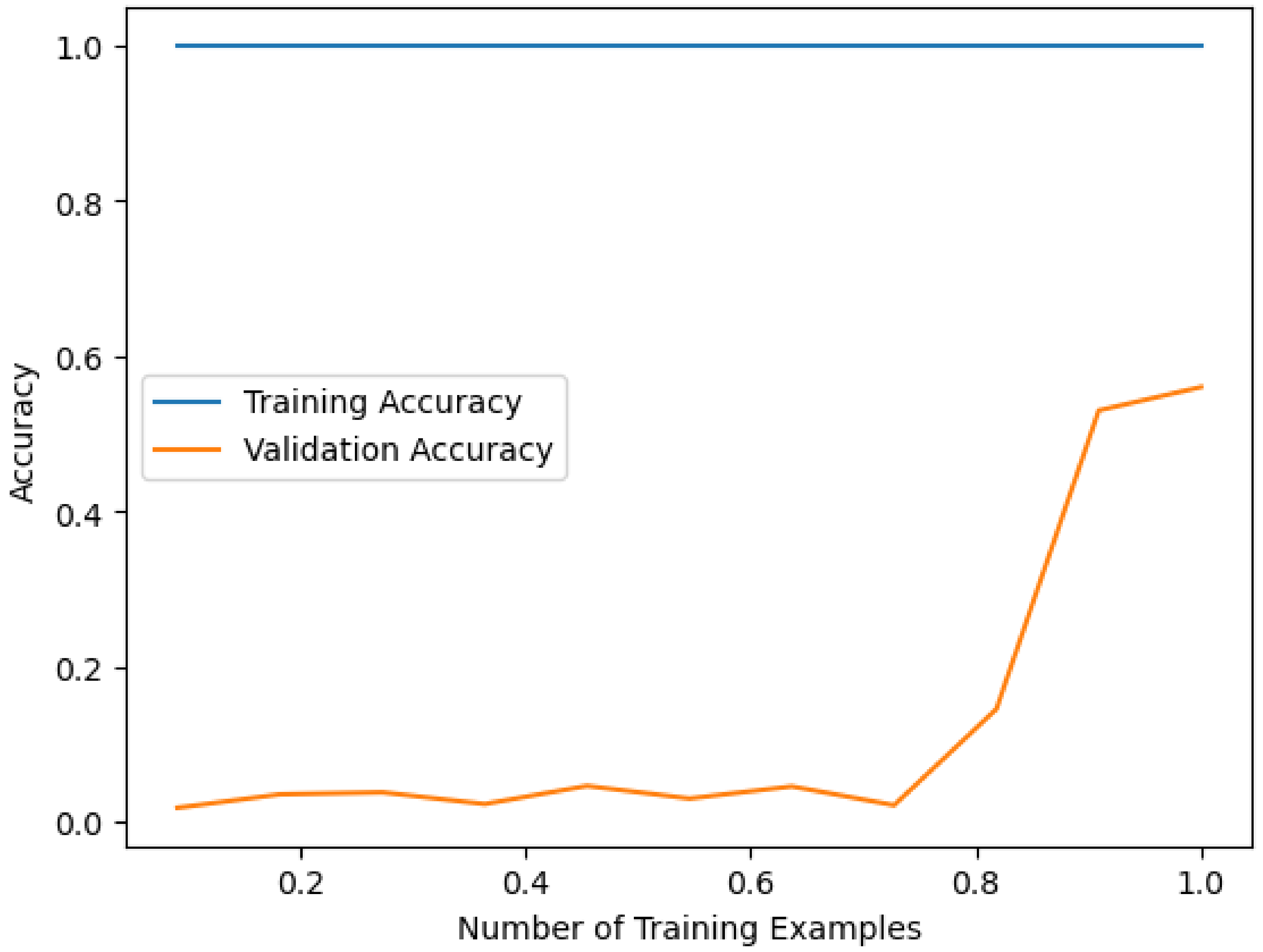
- The **_grow_tree** method takes three arguments: **X**, **y**, and **depth**. **X** is the matrix of features, **y** is the vector of target variables, and **depth** is the current depth of the tree.
- First, the method checks whether the maximum depth of the tree has been reached or whether there is only one unique label in the target variable. If either of these conditions is true, the method creates a leaf node with the class probabilities of the target variables and returns it.
- If the stopping criteria are not met, the method finds the best feature and threshold to split the data based on the information gain. It calculates the information gain for each feature and threshold combination and selects the one with the highest gain.
- The method splits the data into left and right branches based on the best feature and threshold. It then recursively calls itself to grow the left and right branches of the tree until the stopping criteria are met.
- Finally, the method creates a node with the best feature and threshold values and the left and right branches, and returns it.

RESULTS

For this project, we have tried many algorithms, which include the model written from scratch i.e

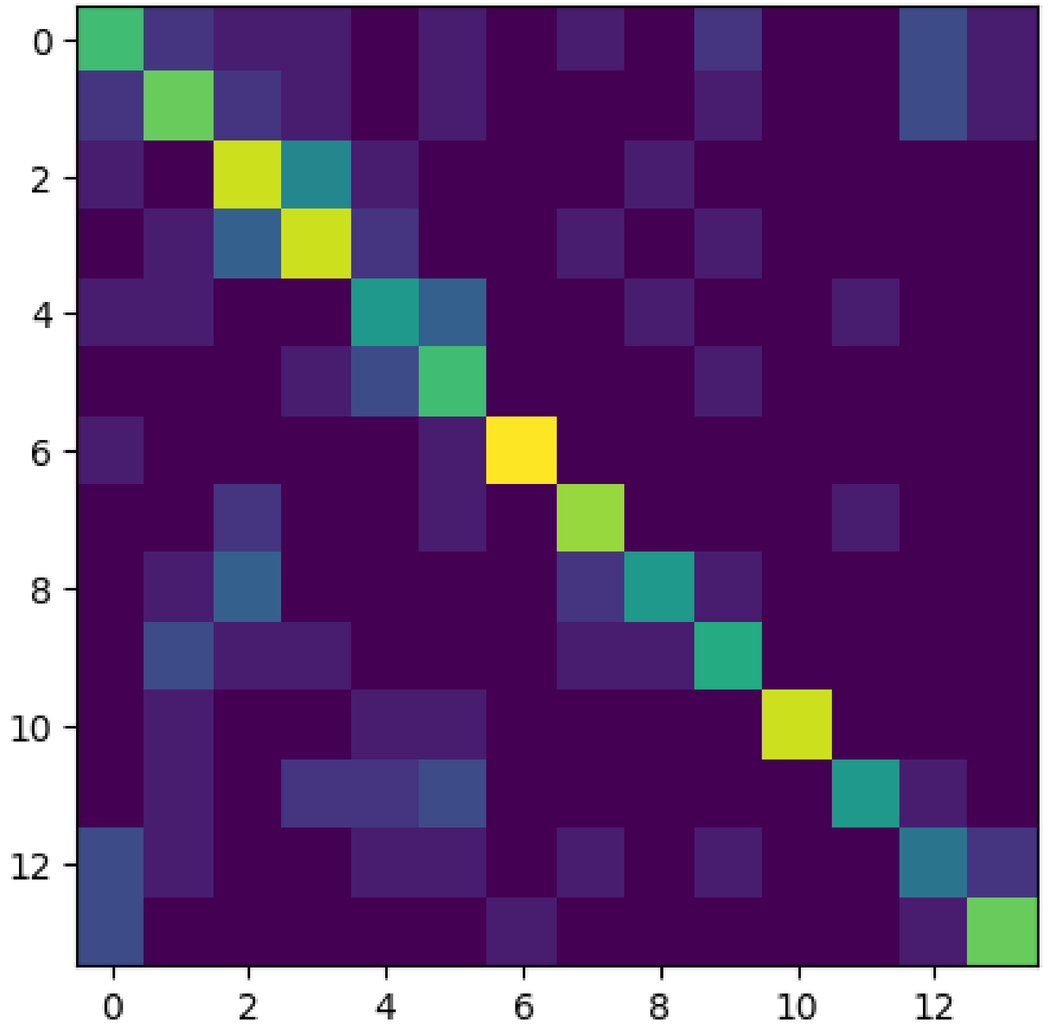


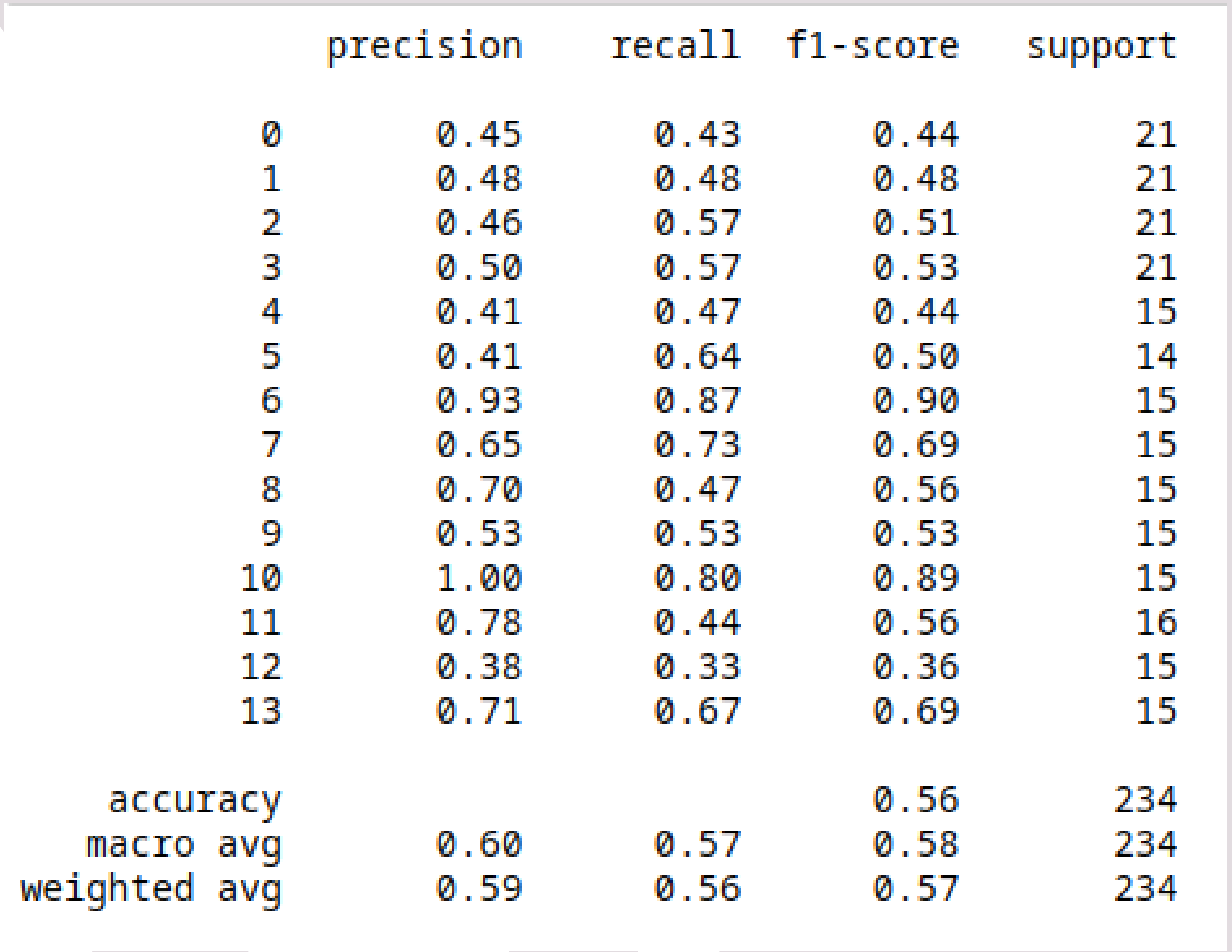
Results



1.0
0.8
0.6
0.4
0.2
0.0
Accuracy
Training Accuracy
Validation Accuracy
0.2
0.4
0.6
0.8
1.0
Number of Training Examples

Results



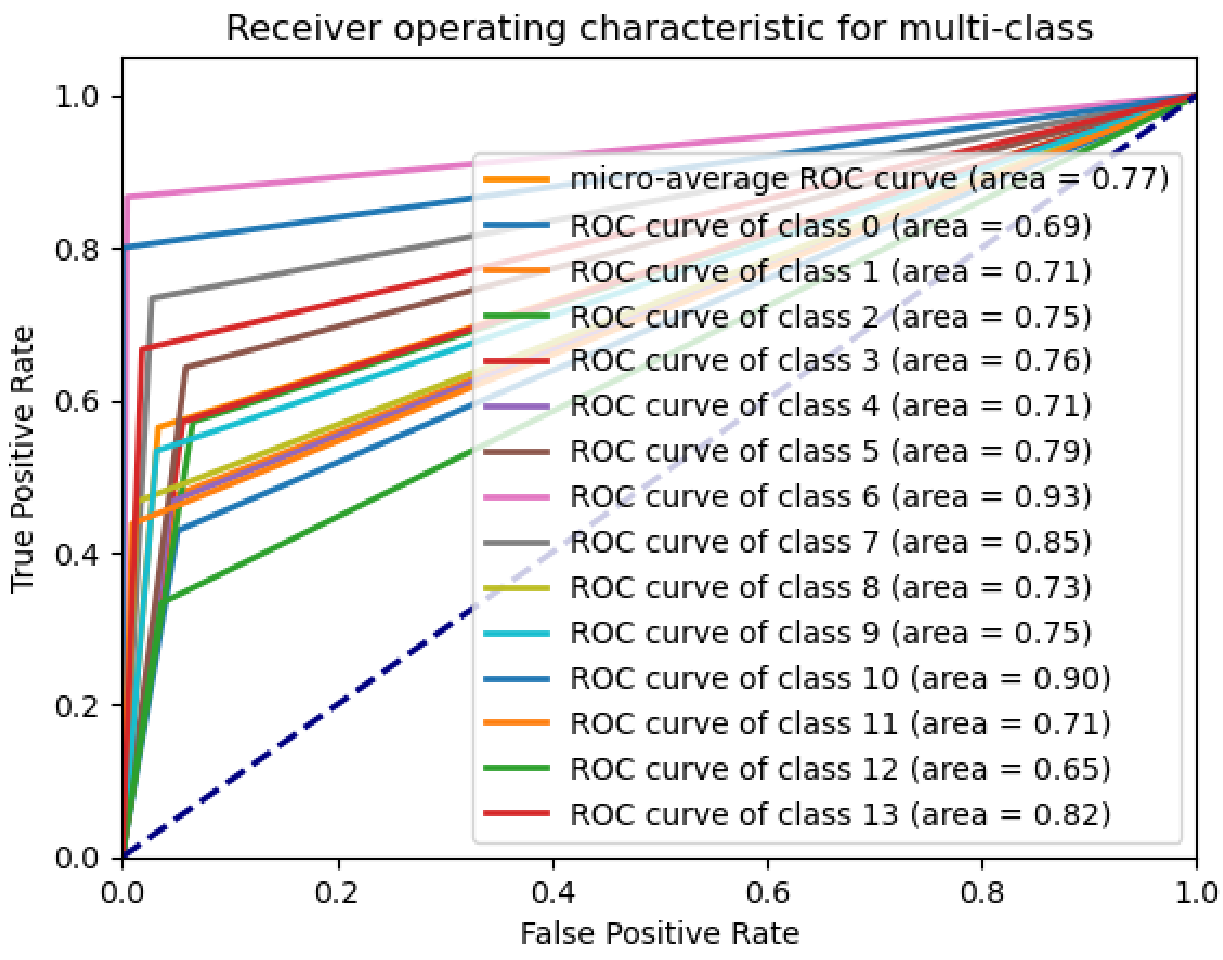


```
              precision    recall  f1-score   support

           0       0.45      0.43      0.44        21
           1       0.48      0.48      0.48        21
           2       0.46      0.57      0.51        21
           3       0.50      0.57      0.53        21
           4       0.41      0.47      0.44        15
           5       0.41      0.64      0.50        14
           6       0.93      0.87      0.90        15
           7       0.65      0.73      0.69        15
           8       0.70      0.47      0.56        15
           9       0.53      0.53      0.53        15
          10       1.00      0.80      0.89        15
          11       0.78      0.44      0.56        16
          12       0.38      0.33      0.36        15
          13       0.71      0.67      0.69        15

    accuracy                           0.56       234
   macro avg       0.60      0.57      0.58       234
weighted avg       0.59      0.56      0.57       234
```

Results

The output shows the area under the ROC curve (AUC) for each class, represented by a numerical value between 0 and 1. AUC measures the overall performance of the classifier and ranges from 0.5 (random guessing) to 1.0 (perfect classification).

ROC curves plot the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings. TPR is also known as sensitivity, recall, or hit rate and measures the proportion of actual positive samples that are correctly classified as positive. FPR, on the other hand, measures the proportion of actual negative samples that are incorrectly classified as positive.



Mobile Net Embedding

```
def dir_to_category(Categories, datadir):
    flat_data_arr=[] #input array
    target_arr=[] #output array
    for i in Categories:
        print(f'loading... category: {i}')

        path = os.path.join(datadir,i)

        for img in os.listdir(path):

            img_array =
load_img(os.path.join(path,img),target_size=(224,224))
            img_array = img_to_array(img_array)
            img_array = np.expand_dims(img_array, axis=0)
            img_array = preprocess_input(img_array)
            img_array = model.predict(img_array)
            flat_data_arr.append(img_array.flatten())
            target_arr.append(Categories.index(i))
        print(f'loaded categoies:{i} successfully')

    flat_data = np.array(flat_data_arr)
    target = np.array(target_arr)
    df = pd.DataFrame(flat_data)

    df['Target'] = target

    x = df.iloc[:,:-1]
    y = df.iloc[:,-1]

    return x, y
```

The code we used defines a function named dir to category that takes two arguments: Cate- gories and datadir

datadir parameter is the path to the directory where the images are located

The function then loops over each category in Categories and reads all the images in the corresponding subdirectory of datadir

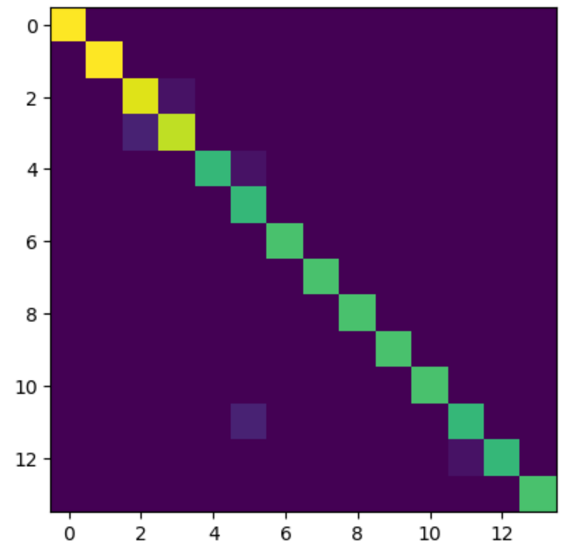
Fine Tuning MobileNet

The fine-tuning pre-trained MobileNet v2 model to classify images into 14 different cate- gories

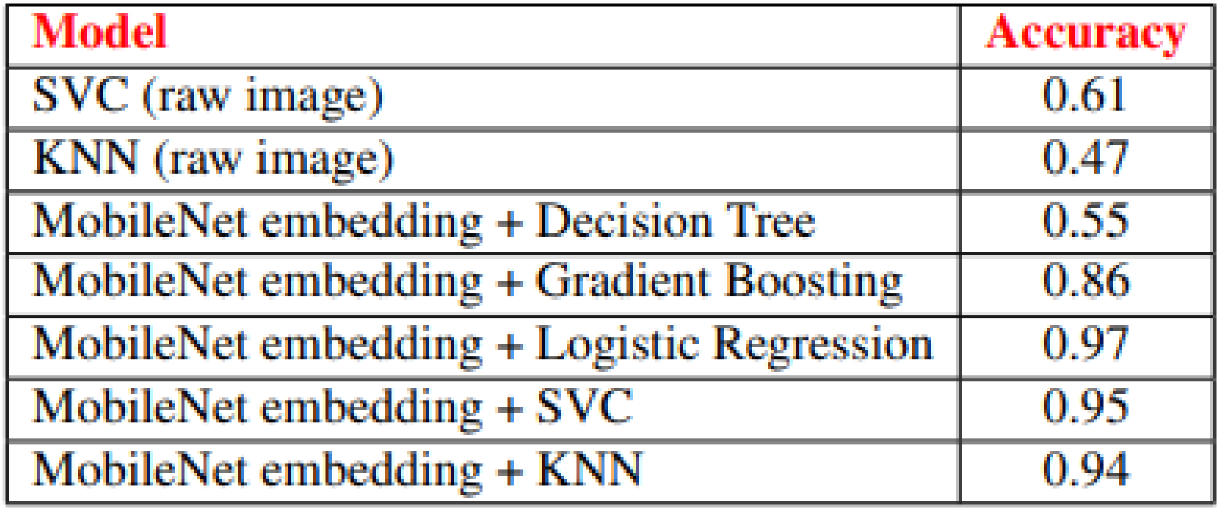
First, the pre-trained MobileNetv2 model is loaded and the last fully connected layer is replaced with a new layer with 14 output features to match the number of categories

The model is trained for 10 epochs, with each epoch iterating over the training set and updating the weights using backpropagation

```
Accuracy: 0.97
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        21
           1       1.00      1.00      1.00        21
           2       0.91      0.95      0.93        21
           3       0.95      0.90      0.93        21
           4       1.00      0.93      0.97        15
           5       0.82      1.00      0.90        14
           6       1.00      1.00      1.00        15
           7       1.00      1.00      1.00        15
           8       1.00      1.00      1.00        15
           9       1.00      1.00      1.00        15
          10       1.00      1.00      1.00        15
          11       0.93      0.88      0.90        16
          12       1.00      0.93      0.97        15
          13       1.00      1.00      1.00        15

    accuracy                           0.97       234
   macro avg       0.97      0.97      0.97       234
weighted avg       0.97      0.97      0.97       234
```



Analysis On Different Algorithms



Model	Accuracy
SVC (raw image)	0.61
KNN (raw image)	0.47
MobileNet embedding + Decision Tree	0.55
MobileNet embedding + Gradient Boosting	0.86
MobileNet embedding + Logistic Regression	0.97
MobileNet embedding + SVC	0.95
MobileNet embedding + KNN	0.94

LIMITATIONS AND FUTURE ENHANCEMENTS

Limitations of this project include the fact that the model may not generalize well to other types of characters or scripts, as it has been trained and tested only on Tirhuta vowels Lipi

In future work, it would be beneficial to add more data per class to increase the overall diver- sity and size of the dataset

Additionally, more characters from the Tirhuta Lipi script could be added to the dataset to enhance the model's ability to recognize a wider range of characters

Thank You!!