Sign Language Digit Recognition Using Machine Learning

1. Project Overview

- **Objective**: Develop a system that translates images of hand signs representing digits (0-9) into their corresponding numerical values.
- **Motivation**: Facilitate communication for individuals who are not proficient in sign language, thereby bridging the gap between the deaf community and the general population.

2. Background and Motivation

- **Significance**: Approximately 5% of the global population (about 430 million people) have disabling hearing loss, many of whom rely on sign language for communication. However, a significant portion of the general population does not understand sign language, leading to communication barriers. This project aims to mitigate these challenges by providing an automated translation system.

3. Dataset Description

- **Source**: Sign Language Digits Dataset https://github.com/ardamavi/Sign-Language-Digits-Dataset
- **Contents**: The dataset comprises 200 images for each digit (0-9), totaling 2,000 images. Each image is a 100x100 pixel colored representation of a hand sign.
- **Preprocessing**: Images were normalized by scaling pixel values to the range [0, 1]. Data was split into training and testing sets using an 80-20 ratio.

4. Technical Implementation

- **Libraries Used**:
  - NumPy
  - Pandas
  - Matplotlib
  - Seaborn
  - Scikit-learn
  - TensorFlow/Keras
- **Data Preparation**:
  - Loaded images and labels.

- Normalized pixel values to [0, 1].
- Reshaped data for compatibility with machine learning models.
- Converted labels to categorical format for neural network training.

- **Code Breakdown**

  Class Definition

  The core functionality is encapsulated in the ModelsClass. This class manages the workflow, from loading and preprocessing data to training, evaluating, and visualizing results.

  Setting up data to be ready for preprocessing:

```
from google.colab import drive
drive.mount('/content/drive')
```

  We linked google colab with our drive to get the data from our drive

```
mask_path = '/content/drive/MyDrive/nti folder/Dataset/0/'
with_mask_files = os.listdir(f'{mask_path}')
for img_file in with_mask_files:
  image = cv2.imread(mask_path + img_file)
  image = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
  image = cv2.resize(image,(100,100))
  image = np.array(image)
  data0.append(image)
```

  Inserted and loaded all the images from our drive and converted it to grayscale made sure that all images are of size (100x100)

  Used numpy to convert images to array of pixels

  Appended the data with its corresponding class (0-9)

```
data = data0 + data1 + data2 + data3 + data4 + data5 + data6 + data7 + data8 +data9
```

  appended the all the data in one list

```
label0 = [0]*len(data0)
label1 = [1]*len(data1)
label2 = [2]*len(data2)
label3 = [3]*len(data3)
label4 = [4]*len(data4)
label5 = [5]*len(data5)
label6 = [6]*len(data6)
label7 = [7]*len(data7)
label8 = [8]*len(data8)
label9 = [9]*len(data9)

All_labels = label0 + label1 + label2+ label3+ label4 + label5 + label6 + label7 + label8 + label9
```

set all data labels and append them to All_labels

Importing libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_recall_fscore_support
from sklearn.metrics import confusion_matrix
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.utils import to_categorical
```

All the libraries we gonna use

Data assignment and preprocessing

```
class ModelsClass:
    def __init__(self):
        # Load and prepare data
        self.load_data()
        self.class_names = [0,1,2,3,4,5,6,7,8,9]

    def load_data(self):
        "Sgin Language Numbers"
        # Load data
        self.x_train , self.x_test , self.y_train , self.y_test = train_test_split(np.array(data) , np.array(All_labels)


        # Normalize pixel values
        self.x_train = self.x_train.astype('float32') / 255.0
        self.x_test = self.x_test.astype('float32') / 255.0

        # Reshape for traditional classifiers
        self.x_train_flat = self.x_train.reshape(self.x_train.shape[0], -1)
        self.x_test_flat = self.x_test.reshape(self.x_test.shape[0], -1)

        # Prepare labels for neural network
        self.y_train_cat = to_categorical(self.y_train)
        self.y_test_cat = to_categorical(self.y_test)
```

Initializes the class by loading data and defining class labels (digits 0–9).

In load_data we split the data to train and test then,

We normalize the pixel values to be set between 0 and 1

Then we flatten the values for ML

Then we categories labels for CNN

Model Training

K-Nearest Neighbors

```
def train_knn(self, n_neighbors=5):
    """Train K-Nearest Neighbors classifier"""
    print("Training KNN classifier...")
    self.knn = KNeighborsClassifier(n_neighbors=n_neighbors)
    self.knn.fit(self.x_train_flat, self.y_train)

    # Make predictions
    y_pred = self.knn.predict(self.x_test_flat)

    # Calculate metrics
    accuracy = accuracy_score(self.y_test, y_pred)
    precision, recall, f1, _ = precision_recall_fscore_support(self.y_test, y_pred, average='weighted')

    print(f"KNN Metrics:")
    print(f"Accuracy: {accuracy:.4f}")
    print(f"Precision: {precision:.4f}")
    print(f"Recall: {recall:.4f}")
    print(f"F1-score: {f1:.4f}")

    return accuracy, precision, recall, f1
```

Trains and evaluates a KNN classifier.

As well as returns the evaluations for comparison with other models in the future

Logistic Regression

```
def train_knn(self, n_neighbors=5):
    """Train K-Nearest Neighbors classifier"""
    print("Training KNN classifier...")
    self.knn = KNeighborsClassifier(n_neighbors=n_neighbors)
    self.knn.fit(self.x_train_flat, self.y_train)

    # Make predictions
    y_pred = self.knn.predict(self.x_test_flat)

    # Calculate metrics
    accuracy = accuracy_score(self.y_test, y_pred)
    precision, recall, f1, _ = precision_recall_fscore_support(self.y_test, y_pred, average='weighted')

    print(f"KNN Metrics:")
    print(f"Accuracy: {accuracy:.4f}")
    print(f"Precision: {precision:.4f}")
    print(f"Recall: {recall:.4f}")
    print(f"F1-score: {f1:.4f}")

    return accuracy, precision, recall, f1
```

Uses logistic regression for multi-class classification.

Train the model on the data and evaluate it accordingly

As well as returns the evaluations for comparison with other models in the future

SVM

```
def train_svm(self):
    print("Training SVM...")
    self.svm = SVC(max_iter=100, kernel = 'rbf')
    self.svm.fit(self.x_train_flat , self.y_train)

    y_pred = self.svm.predict(self.x_test_flat)

    accuracy = accuracy_score(self.y_test, y_pred)
    precision, recall, f1, _ = precision_recall_fscore_support(self.y_test, y_pred, average='weighted')

    print(f"SVM Metrics:")
    print(f"Accuracy: {accuracy:.4f}")
    print(f"Precision: {precision:.4f}")
    print(f"Recall: {recall:.4f}")
    print(f"F1-score: {f1:.4f}")

    return accuracy, precision, recall, f1
```

Trains and evaluate SVM on our data

Returns for comparison with other models

Convolutional Neural Network (CNN)

```
def build_neural_network(self):
    """Build and train neural network"""
    print("Building and training neural network...")
    self.model = Sequential([
        Conv2D(32, (3, 3), activation='relu', input_shape=(100,100,1)),
        MaxPooling2D((2, 2)),

    # Convolutional Layer 2
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),

    # Convolutional Layer 3
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),

    # Flatten and Fully Connected Layers
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(10, activation='softmax')
    ])

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

    # Train the model
    history = self.model.fit(self.x_train, self.y_train_cat,
                             epochs=10,
                             batch_size=32,
                             validation_split=0.2,
                             verbose=1)

    # Evaluate the model
    test_loss, test_accuracy = self.model.evaluate(self.x_test, self.y_test_cat)
    y_pred = np.argmax(self.model.predict(self.x_test), axis=1)
    precision, recall, f1, _ = precision_recall_fscore_support(self.y_test, y_pred, average='weighted')

    print(f"\nNeural Network Metrics:")
    print(f"Test accuracy: {test_accuracy:.4f}")
    print(f"Precision: {precision:.4f}")
    print(f"Recall: {recall:.4f}")
    print(f"F1-score: {f1:.4f}")
```

Builds a CNN for image recognition.

Convolutional and pooling layers for feature extraction.

Fully connected layers for classification.

Visualization

```
def visualize_results(self, history=None):
    """Visualize training results and examples"""
    # Set up the figure
    plt.figure(figsize=(15, 10))

    # Plot sample images
    for i in range(10):
        plt.subplot(2, 5, i + 1)
        plt.imshow(self.x_train[i], cmap='gray')
        plt.title(self.class_names[self.y_train[i]])
        plt.axis('off')

    plt.tight_layout()
    plt.show()

    # If neural network history is provided, plot learning curves
    if history is not None:
        plt.figure(figsize=(12, 4))

        plt.subplot(1, 2, 1)
        plt.plot(history.history['accuracy'], label='Training')
        plt.plot(history.history['val_accuracy'], label='Validation')
        plt.title('Model Accuracy')
        plt.xlabel('Epoch')
        plt.ylabel('Accuracy')
        plt.legend()

        plt.subplot(1, 2, 2)
        plt.plot(history.history['loss'], label='Training')
        plt.plot(history.history['val_loss'], label='Validation')
        plt.title('Model Loss')
        plt.xlabel('Epoch')
        plt.ylabel('Loss')
        plt.legend()

        plt.tight_layout()
        plt.show()
```

Visualizes sample images, training progress, and confusion matrices.

Requirements

Install the necessary libraries using the following command:

```
pip install numpy pandas matplotlib seaborn scikit-learn tensorflow
```

- **Models Implemented**:
  - **K-Nearest Neighbors (KNN)**:

        - Trained with `n_neighbors=5`.
        - Achieved accuracy: 65.1%.
    - **Logistic Regression**:
        - Multinomial classification with `max_iter=1000`.
        - Achieved accuracy: 77.0%.
    - **Support Vector Machine (SVM)**:
        - Radial basis function kernel with `max_iter=100`.
        - Achieved accuracy: 83.5%.
    - **Convolutional Neural Network (CNN)**:
        - Architecture:
            - Conv2D → MaxPooling2D
            - Conv2D → MaxPooling2D
            - Conv2D → MaxPooling2D
            - Flatten → Dense → Dropout → Dense
        - Achieved accuracy: 88.1%.
- **Evaluation Metrics**:
    - Accuracy
    - Precision
    - Recall
    - F1-Score
    - Confusion Matrix

5. Results and Analysis

- **Model Performance**:
    - CNN outperformed other models with an accuracy of 88.1%.
    - SVM also showed strong performance, with accuracy of 83.5%.
- **Confusion Matrices**:
    - Provided for each model to visualize misclassifications.
- **Learning Curves**:
    - Displayed for CNN to illustrate training and validation accuracy/loss over epochs.

6. Timeline

- **December 17**:
    - Project initiation and dataset acquisition.
- **December 18**:
    - Data preprocessing and exploratory data analysis.

- **December 22**:
  - Model development (KNN, Logistic Regression, SVM).
  - CNN architecture design and training.
  - Model evaluation and comparison.
- **December 24**:
  - Documentation and final report preparation.

7. Future Work

- **Dataset Expansion**: Incorporate a more diverse set of hand images to improve model robustness.
- **Real-time Application**: Develop a live video recognition system for dynamic hand sign translation.
- **Multilingual Support**: Extend the system to recognize sign languages from different regions.

8. References

- Arda Mavi, "Sign Language Digits Dataset,".
- TensorFlow Documentation
- Scikit-learn Documentation