



Pattern Recognition and Machine Learning

Digits-3D

GROUP # 15

Teacher
Lasse Lensu
Fedor Zolotarev

| <i>Students:</i> | <i>Role:</i> | <i>Student Number:</i> |
|-------------------------|----------------------------|-------------------------------|
| Fasie Haider | Coding, Data Preprocessing | 002648745 |
| Haider Ali | Coding, Reporting | 002386780 |
| Arman Golbidi | Coding, Testing | 002389114 |

9th December 2023

Table of Contents

| | |
|---|-----------|
| Pattern Recognition and Machine Learning..... | 1 |
| 1 Introduction..... | 2 |
| 2 Team Member | 3 |
| 3 Data Pre-Processing | 4 |
| 4 Training Model..... | 6 |
| 5 Results | 11 |
| 6 Conclusion | 15 |
| 7 Project Setup | 15 |
| Required Environment Setup..... | 15 |
| Example Commands..... | 16 |
| 8 References..... | 17 |

1 Introduction

This project is designed to make a digit identification model which will be able to identifying hand drawn digit from range 0 to 9. For this purpose, we use LeapMotion sensor data which has x, y and z coordinates. LeapMotion sensor is very powerful tool which records detailed information about the movements of the index finger. By utilizing this three-dimensional location information, the system aims to achieve accurate recognition of digits formed through hand gestures in air.

This report details the methodology and process of constructing a digit recognition model using the classifiers. We have many csv files in our zip file. Each dataset file corresponds to a specific digit 0 to 9 and contains data points representing the x, y, and z coordinates. The data in these files is already labeled we train our model on this data and when our model recognize the patterns in data very well than we move forward to testing and we test our train model on testing dataset.

To achieve these objective, we make project on the theoretical concepts and practical skills learned in our course. We will use specifically techniques like classification, confusion matrix, neural networks, model training, evaluation and some preprocessing techniques, ensuring a comprehensive understanding of the steps taken to achieve the final results. By integrating these methods, we aim to build a reliable model capable of accurately classifying 3D digits.

2 Team Member

In our project completion every member played a crucial role. We work as a team and finally we got results.

Fasih take responsibility of handling all aspects of data preprocessing and management to ensure the dataset is clean, consistent, and ready for model training. The work includes extracting data from zip file and then merge this data in an array then preprocess the data apply normalization, padding and truncate. And after all the preprocessing he split the dataset into training, testing and validation datasets. They also implement robust error handling to deal with any type of error during preprocessing.

Haider Take responsibility for building model and building training loop. He uses tensorflow for making a CNN model. He built the model and training loops from scratch and after this he also write a code to make some plots which shows training progress. His role also include experiment with different architectural configuration and training strategies.

Arman take responsibility of evaluating the trained CNN model performance and creating insightful visualizations to interpret the results effectively. The work includes making classification report and confusion metrics. He also writes Digit_classify function to classify the test data. Their role ensure that the project is clearly communicated.

3 Data Pre-Processing

In our project, our first task was to prepare dataset for training. We have start by putting the samples in the same directory as our script but there is also an option to provide the location of the training data to the script using command line. During the project we learn many things, now we will have processed with how we done preprocessing.

First of all, we extract csv files for our task and after extracting we merge these files into one dataframe. We traverse all the given csv files from the training data folder and append all data in data array. For labels we make separate array in which we are storing labels. So after all this process we have complete data for digits in one array. In all files we have three columns which have x, y and z coordinates and in file name we have digit which belong to these coordinates and this data we use for digit reorganization.

```
#function for load and preprocessing data
def unzip_and_load_data(zip_path,extract_path='digits_3d_data',max_length=200):
    """unzip dataset and preprocess csv files into dat and lables"""

    import zipfile
    with zipfile.ZipFile(zip_path, 'r') as zip_ref:
        zip_ref.extractall(extract_path)

    data, labels = [],[]
    for root, _,files in os.walk(extract_path):
        for file in files:
            if file.endswith('.csv'):
                try:
                    label = int(file.split("_")[1]) #extract labels from filename
                    sample = pd.read_csv(os.path.join( root, file), header=None).values # stroke_3_0001.csv
                    processed_samples =preprocess_sample(sample,max_length)
                    data.append(processed_samples)
                    labels.append(label)
                except Exception as e:
                    print(f"Error processing file {file}: {e}")
    return np.array(data), np.array(labels)
```

After data extraction we move forward with normalization, for normalization we decide to go with minmax normalization which normalize the value between minimum 0 and maximum 1 number. After normalization we apply padding for consistent length of data. If the data has fewer rows than zero added and if data have excessive row from a threshold, then these rows will be removed.

```
#function for preprocessing a single sample
def preprocess_sample(sample, max_length=200):
    """nomlize each column, pad or truncate to max_length"""
    for i in range(sample.shape[1]):
        sample[:, i] = (sample[:,i]-np.min(sample[:,i]))/(np.max(sample[:,i])-np.min(sample[:,i])+1e-6)
    if sample.shape[0]>max_length:
        sample = sample[:max_length]
    else:
        padding = np.zeros((max_length - sample.shape[0], sample.shape[1]))
        sample = np.vstack((sample, padding))
    return sample
```

After this we store the coordinates in a variable and labels in another variable. After this we split the dataset into training and testing. We use 70% data for training and 30% for testing purpose. After split we again split data into training and validation set. The purpose for this splitting is to validate learning of model during training. After all of this Now we will move forward with training. The code for splitting dataset is here:

```
x_train, x_temp, y_train, y_temp = train_test_split(x, y, test_size=0.3, random_state=42)
x_val, x_test, y_val, y_test = train_test_split(x_temp, y_temp, test_size=0.5, random_state=42)
```

4 Training Model

After preprocessing we now move forward with model training. For training purpose, we use CNN model. The goal of this model is to predict digits. The input layer of this model accept 3d coordinates with fixed length of 200 number of rows. We take length of 200 because we in dataset some files have data above 200 and some have below 200 so we take fixed length of 200. If the rows will be less than 200 it will be padded with zero and if it will be above 200 it will be truncate to 200.

In making our CNN model we use tensor flow. In first line we pass the input shape and in then we define the layer with 32 filters and kernel size of 3. Conv1D layers will extract spatial features from the data. We use RELU as activation function for this task. And after this we define max pooling with pooling size of 2. Which will take max value from the the sequence of window two. So it reduces the temporal dimension by half and make the model more computationally efficient. And Second layer have 64 filters with three kernels and we use RELU as activation function. And after this we define global average pooling.

```
# function for build cnn model
def build_model(input_shape, num_classes):
    """build a cnn for digital clasify model we are making layes here for the model we use tensorflow for making layers"""
    model =tf.keras.Sequential([
        tf.keras.layers.Input(shape=input_shape),
        tf.keras.layers.Conv1D(32, kernel_size = 3, activation='relu'),
        tf.keras.layers.MaxPooling1D(pool_size=2),
        tf.keras.layers.Conv1D(64, kernel_size= 3, activation='relu'),
        tf.keras.layers.GlobalAveragePooling1D(),
        tf.keras.layers.Dense(128, activation = 'relu'),|
        tf.keras.layers.Dense(num_classes, activation = 'softmax')
    ])
    return model
```

After this we define fully connected dense layers. First layer has 128 neurons to learn high level features from the data and Relu as activation function and second layer have 10 neurons with softmax activation which classify the digits from 0 to 9.

```
tf.keras.layers.Dense(128, activation = 'relu'),|
tf.keras.layers.Dense(num_classes, activation = 'softmax')
```

In optimizer we use Adam. Because It adjusts learning rate dynamically for each parameter, which helps improve convergence speed and model performance. We use Sparse Categorical Cross entropy loss function because this loss function is ideal for multi-class classification problems where the labels are integers. so that's why we use this.

```
# build model we initialize input_shape variable and number of classes and pass to the model
input_shape = (max_sequence_length, 3)
num_classes = 10
model = build_model(input_shape, num_classes)
model.compile(optimizer=tf.keras.optimizers.Adam(),
              loss=tf.keras.losses.SparseCategoricalCrossentropy(),
              metrics=['accuracy'])
```

After this we make a function for training our model in this function we define epoch loop and in epoch loop we define training loop and validation loop. We first define optimizer and loss function for training and then we start with epoch loop.

```
def train_model(model, X_train, y_train, X_val, y_val, epochs, batch_size):
    """here we are making custom training loop with accuracy and loss tracking we first make loop for number of epochs
    and validation and then we will show the results."""
    optimizer = tf.keras.optimizers.Adam()
    loss_fn = tf.keras.losses.SparseCategoricalCrossentropy()

    train_acc_metric = tf.keras.metrics.SparseCategoricalAccuracy()
    val_acc_metric = tf.keras.metrics.SparseCategoricalAccuracy()

    history = {"train_loss": [], "val_loss": [], "train_accuracy": [], "val_accuracy": []}
```

In Each epoch we shuffle the data. Because using this technique model can better generalize the features. More ever this thing also helped to avoid overfitting. In training loop, we take the data for according to batch size from the main data variable and same for labels and use them in training for this loop in next loop the next number of rows will be selected according to batch size. In training loop, we check also check gradient, accuracy and loss function which we show after training of each epoch.


```

for epoch in range(epochs):
    print(f"\nEpoch {epoch + 1}/{epochs}")
    indices = np.arange(X_train.shape[0])
    np.random.shuffle(indices)
    X_train, y_train = X_train[indices], y_train[indices]

    # Initialize loss accumulators
    train_loss = 0.0
    val_loss = 0.0

    # Training loop which will describe training process
    for i in range(0, X_train.shape[0], batch_size):
        X_batch, y_batch = X_train[i:i + batch_size], y_train[i:i + batch_size]
        with tf.GradientTape() as tape:
            logits = model(X_batch, training=True)
            loss = loss_fn(y_batch, logits)
            grads = tape.gradient(loss, model.trainable_weights)
            optimizer.apply_gradients(zip(grads, model.trainable_weights))
            train_loss += loss.numpy()
            train_acc_metric.update_state(y_batch, logits)

```

After training loop model is evaluated on validation dataset to check whether model is generalizing the text or not. The same things we do here which we do for training. Like Matrices like accuracy, loss is calculated on validation dataset.

```

# Validation loop
for i in range(0, X_val.shape[0], batch_size):
    X_val_batch, y_val_batch = X_val[i:i + batch_size], y_val[i:i + batch_size]
    val_logits = model(X_val_batch, training=False)
    val_loss += loss_fn(y_val_batch, val_logits).numpy()
    val_acc_metric.update_state(y_val_batch, val_logits)

```

After this we train our model using the train_model function which we described above. We send model, train data with inputs and labels, and number of epochs.

```

[ ] # train model
    history = train_model(model, X_train, y_train, X_val, y_val, epochs=250, batch_size=32)

```

After training we got the accuracy of 99% and validation accuracy of 94 % which is looking fine. Our model is train till 250 epochs after this we achieve this accuracy.

```

Epoch 244/250
Training Loss: 0.0567, Accuracy: 0.9900, Validation Loss: 0.5007, Accuracy: 0.9200

Epoch 245/250
Training Loss: 0.0550, Accuracy: 0.9900, Validation Loss: 0.4355, Accuracy: 0.9133

Epoch 246/250
Training Loss: 0.0565, Accuracy: 0.9900, Validation Loss: 0.4225, Accuracy: 0.9067

Epoch 247/250
Training Loss: 0.0513, Accuracy: 0.9914, Validation Loss: 0.4157, Accuracy: 0.9333

Epoch 248/250
Training Loss: 0.0572, Accuracy: 0.9900, Validation Loss: 0.4267, Accuracy: 0.9400

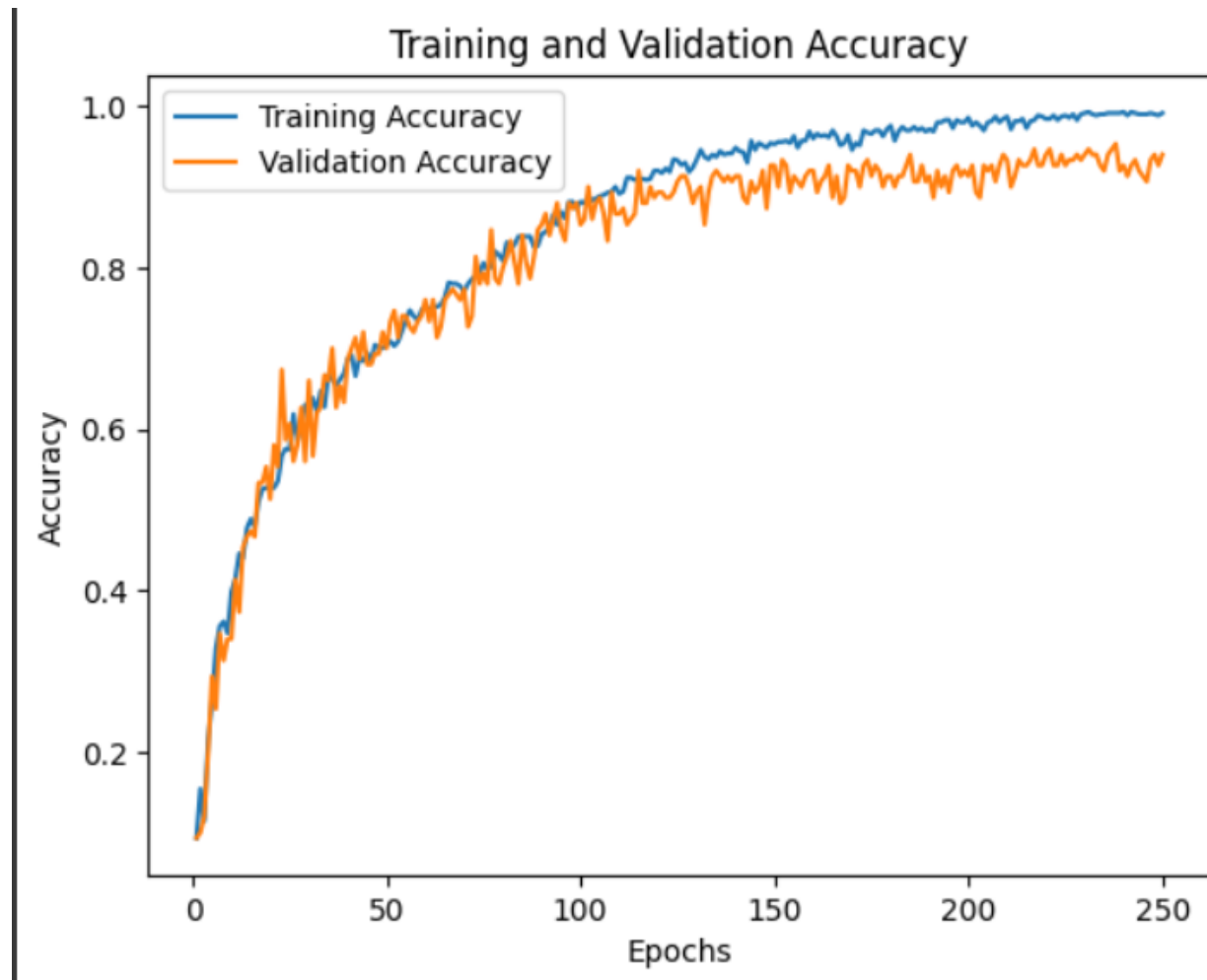
Epoch 249/250
Training Loss: 0.0558, Accuracy: 0.9886, Validation Loss: 0.4533, Accuracy: 0.9267

Epoch 250/250
Training Loss: 0.0489, Accuracy: 0.9914, Validation Loss: 0.4267, Accuracy: 0.9400
  
```

Now we will show the plot which show training and validation loss. We can see from the plot our model is well trained at the end we have very minimal loss for both training and validation.



the accuracy of model is increasing and at the end we can see we have good accuracy for both training and validation dataset.



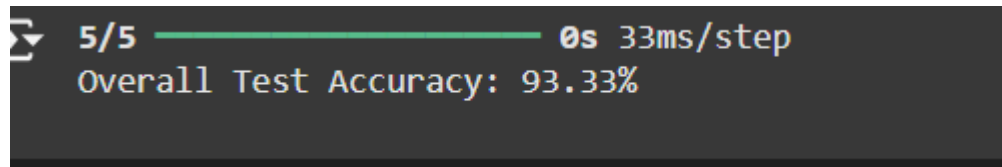
We can see the steady rise in both training and validation accuracy during the early epochs shows that the model is effectively learning to classify the data.

5 Results

After training of model we test our model on the test dataset. And we got the accuracy of 93%. This result is good for testing its mean our model is well trained.

```
y_pred = model.predict(X_test) # get model predictions for X_test
y_pred = np.argmax(y_pred, axis=1) # get class labels from predictions (if needed)

#calculating and displaying accuracy
accuracy = calculate_accuracy(y_test, y_pred)
print(f"Overall Test Accuracy: {accuracy:.2f}%")
```



```
5/5 ————— 0s 33ms/step
Overall Test Accuracy: 93.33%
```

After this we write a function to show classification report we use built in classification report function from Sklearn. The code for this is below:

```
from sklearn.metrics import classification_report # Import the correct classification_report

# Function: Print classification report
def print_classification_report(y_true, y_pred):
    """
    print the classification report.
    parameters:
    - y_true: true labels.
    - y_pred: predicted labels.
    """
    print("\nClassification Report:")
    print(classification_report(y_true, y_pred))

# print classification report
print_classification_report(y_test, y_pred)
```

The classification report provides detailed performance metrics for each class. In classification we see precision, recall, f1-score and support for each class. Overall we are getting good results

```

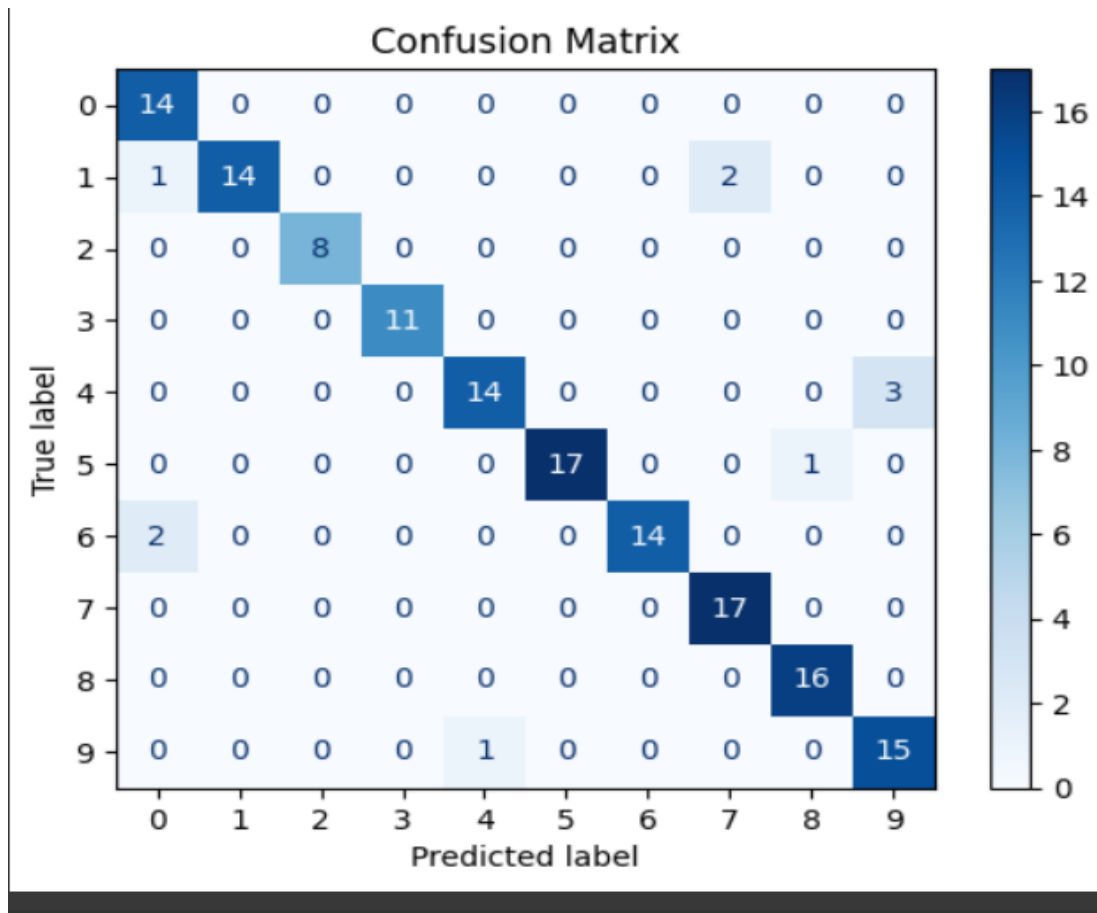
Classification Report:
              precision    recall  f1-score   support

    0           0.82         1.00         0.90         14
    1           1.00         0.82         0.90         17
    2           1.00         1.00         1.00          8
    3           1.00         1.00         1.00         11
    4           0.93         0.82         0.88         17
    5           1.00         0.94         0.97         18
    6           1.00         0.88         0.93         16
    7           0.89         1.00         0.94         17
    8           0.94         1.00         0.97         16
    9           0.83         0.94         0.88         16

 accuracy          0.93         150
 macro avg         0.94         0.94         0.94         150
 weighted avg      0.94         0.93         0.93         150

```

After this we make a confusion matrix for to see the performance of model on test data. The confusion matrix will visualize how good the model classified each digit by comparing true labels with predicted labels. The diagonal element is representing correct prediction.



Now we will write `digit_classify` function to classify the digits on the bases of model prediction.

```
# classify a single test sample
def digit_classify(model, test_data, max_length=200):
    """
    Classifies a single test sample.

    Parameters:
    - model: Trained model.
    - test_data (np.array): Input data as N x 3 matrix.
    - max_length (int): Maximum sequence length for padding.

    Returns:
    - int: Predicted class label.
    """
    # preprocess test data
    test_data = preprocess_sample(test_data, max_length)
    test_data = np.expand_dims(test_data, axis=0) # add batch dimensions

    # predict and return class with highest probability
    predictions = model.predict(test_data)
    return np.argmax(predictions)
```

And now we write `test_with_dataset_samples` function. In which we give data as parameter and also number of samples and in this function, we call `digit_classify` function to classify the digit. After this we will print our results.

```
# function: test the classifier with samples from the dataset
def test_with_dataset_samples(model, X_test, y_test, num_samples=5):
    """
    tests model with a few samples from the dataset and shows predictions.

    parameters:
    - model: Trained model.
    - X_test: Test data.
    - y_test: True labels for test data.
    - num_samples: Number of samples to test.
    """
    for i in range(num_samples):
        sample = X_test[i]
        true_label = y_test[i]
        predicted_label = digit_classify(model, sample, max_length = sample.shape[0])
        print(f"Sample {i+1}: true label = {true_label}, predicted label = {predicted_label}")

# test the classifier with real samples from the dataset
print("testing with real samples from the dataset:")
test_with_dataset_samples(model, X_test, y_test, num_samples=5)
```

```
Testing with real samples from the dataset:
1/1 _____ 0s 20ms/step
Sample 1: true label = 3, predicted label = 3
1/1 _____ 0s 20ms/step
Sample 2: true label = 5, predicted label = 5
1/1 _____ 0s 21ms/step
Sample 3: true label = 1, predicted label = 1
1/1 _____ 0s 19ms/step
Sample 4: true label = 9, predicted label = 9
1/1 _____ 0s 21ms/step
Sample 5: true label = 8, predicted label = 8
```

We can see all the predicted digit is correct. So that's mean our model is performing very well. We can extend this model in future if we want to classify the real time hand drawn digits.

6 Conclusion

In conclusion we achieve very good results. After preprocessing and model training our model give us very good results. We face many challenges like overfitting underfitting of model and we solve these problem time by time by applying different technique and changing of hyper parameter. we got a good training and validation accuracy and also test accuracy. we successfully developed and evaluated Convolutional Neural Network model to classify 3D handwritten stroke data into digits. Only minor misclassification occur which can also be removed by using advanced techniques. Overall, our project successfully shows effectiveness of CNNs in handling 3D sequential data for handwritten digit recognition we can improve this model more in future.

7 Project Setup

Required Environment Setup

1. **Python Version**
 - Ensure Python 3.11.1 is installed.
2. **Install Required Libraries**
 - Install the necessary Python libraries using below command:
pip3 install numpy tensorflow pandas scikit-learn matplotlib
3. **Prepare Project Files**
 - Ensure the following files and folders are in the same directory:
 - drecogniser.py: The main script.
 - training_data/: A folder containing the .csv files for training.

Commands for executing the script.

| Command | Variable | Description | Default Value |
|------------------|-----------------------|---|--|
| train | --epochs | Number of epochs for training | 250 |
| | --batch_size | Batch size for training | 32 |
| | --save_model | Path to save the trained model (no extension) | trained_model |
| | --max_sequence_length | Maximum sequence length for preprocessing | 200 |
| | --training_data_path | Path to the training data folder | training_data |
| test | --model_path | Path to load the trained model | trained_model |
| | --sample_path | Path to the test sample (CSV format) | Required |
| | --max_sequence_length | Maximum sequence length for preprocessing | 200 |
| | --model_path | Path to load the trained model | trained_model.keras |
| visualize | --model_name | Name of the model (no extension) | trained_model |
| predict | --model_path | Path to load the trained model | trained_model.keras |
| | --folder_path | Folder containing test sample files (CSV) | Required |
| | --max_sequence_length | Maximum sequence length for preprocessing | 200 |
| clean | --model_path | Base name of the model file to delete | Deletes all models if not specified |

Notes:

1. **Required Parameters:** Some commands have parameters marked as **Required**. These must be provided when running the command, or the script will throw an error.
2. **Default Behavior:** When no optional arguments are provided, the script will use the default values listed in the table.
3. **File Extensions:** For commands requiring `--model_path`, the script automatically appends `.keras` if not already included.

Example Commands

1. Train Command

- **Without Optional Parameters:**

```
python3 drecogniser.py train
```

- **With Optional Parameters (using default values):**

```
python3 drecogniser.py train --epochs 250 --batch_size 32 --save_model  
"trained_model" --max_sequence_length 200 --training_data_path  
"training_data"
```

2. Test Command

- **Without Optional Parameters (requires sample path):**

```
python3 drecogniser.py test --sample_path "training_data/sample.csv"
```

3. Evaluate Command

- **Without Optional Parameters:**

```
python3 drecogniser.py evaluate
```

- **With Optional Parameters (using default values):**

```
python3 drecogniser.py evaluate --model_path "trained_model"
```

4. Visualize Command

- **Without Optional Parameters:**

```
python3 drecogniser.py visualize
```

- **With Optional Parameters (using default values):**

```
python3 drecogniser.py visualize --model_name "trained_model"
```

5. Predict Command

- **Without Optional Parameters (requires folder path):**

```
python3 drecogniser.py predict --folder_path "data/samples"
```

- **With Optional Parameters (using default values):**

```
python3 drecogniser.py predict --folder_path "training_data" --  
max_sequence_length 200
```

6. Clean Command

- **Without Optional Parameters (deletes all .keras files):**

```
python3 drecogniser.py clean
```

- **With Optional Parameters (deletes a specific model file):**

```
python3 drecogniser.py clean --model_path "trained_model"
```

8 References

Chen, Z., Wang, Z., & Shen, X. (2015). Hand gesture recognition using a Leap Motion controller. In Proceedings of the International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery.

LeCun, Y., & Bengio, Y. (1995). Convolutional networks for images, speech, and time-series. In The Handbook of Brain Theory and Neural Networks (pp. 255-258). MIT Press.