AI PROJECT

Saira Luqman (FA21-BCE-084)

Abdullah Javed (FA21-BCE-019)

Handwritten Letter Recognition with CNN

Introduction:

Handwritten letter recognition plays a pivotal role in various applications, from digitizing historical documents to automating postal services. In this project, we employ Convolutional Neural Networks (CNNs) to tackle the challenge of recognizing handwritten letters, utilizing the Extended Modified National Institute of Standards and Technology (EMNIST) dataset. The primary objective is to develop an effective model that can accurately classify and identify handwritten letters, showcasing the potential of deep learning in pattern recognition tasks.

Code Implementation:

```
pip install emnist
```

Import Libraries:

```
import numpy as np
from emnist import extract_training_samples, extract_test_samples
from keras.models import Sequential
from Keras.layers import Conv2D, MaxPooling2D, Flatten, and Dense
from Keras.utils import to_categorical
import matplotlib.pyplot as plt
```

- Import necessary libraries such as **NumPy** for numerical operations, **EMNIST** for accessing the dataset, **Keras** for building and training neural networks, and **Matplotlib** for visualizations.

Load EMNIST Dataset:

```
train_images, train_labels = extract_training_samples('letters')
test_images, test_labels = extract_test_samples('letters')
```

- Extract training and test samples from the EMNIST dataset, specifically for handwritten letters.

Data Preprocessing:

```
train_images = train_images.reshape((train_images.shape[0], 28, 28, 1)).astype('float32') / 255
test_images = test_images.reshape((test_images.shape[0], 28, 28, 1)).astype('float32') / 255
train_labels = to_categorical(train_labels - 1, 27)
test_labels = to_categorical(test_labels - 1, 27)
```

- In the code above, we are converting the pixels of the images into raw data that the neural network can use for training.
- We first add another dimension using the *reshape* and *shape* function. Considering this is a greyscale image we use '1' as the fourth dimension. The '28,28' tells us that the images in the dataset are 28*28 pixels.
- We are dividing each pixel by the value 255. This helps us get an integer value between 0 and 1.
- The to_categorical function is used for one hot encoding. We are subtracting the train and test labels by 1 because we want to start from 0.

Create CNN Model:

```
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dense(27, activation='softmax'))
```

- Sequential function creates a stack of neural network layers.
- We create a convolutional layer that takes the input of 28*28 pixels greyscale images, then applies 32 filters of size 3*3 using the Relu activation function. In Relu, the pixels with a negative value are replaced with a value of zero.
- Max pooling helps to reduce the spatial dimensions. We can call it downsampling. We are making 2*2 windows. From each window, the function takes in the most prominent feature represented by the maximum value.

- This additional convolutional layer with 64 filters of size 3*3 continues to extract higher-level features from the input data.
- The flatten function helps us to convert a multi-dimension vector into a single-dimension vector. This helps the model to look across the data more easily and helps to create and find relations in the data. Moreover, it prepares the data for the dense layer.
- Relu is used in the hidden layers for feature learning, and softmax is used in the output layer for multi-class classification. The dense function is used to connect all the neurons. We have used the dense function at the end because it looks at the data, and global patterns and makes predictions.

Compile the Model:

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

- Compile the model with the Adam(Adaptive moment optimizer) optimizer, categorical cross-entropy loss function, and accuracy as the metric.

Train the Model:

```
history = model.fit(train_images, train_labels, epochs=5, batch_size=64, validation_data=(test_images, test_labels))
```

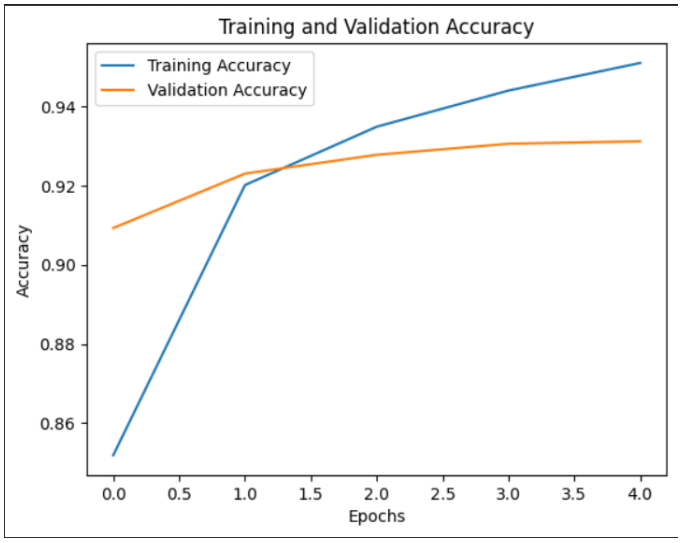
- Train the model on the training data for five epochs with a batch size of 64.
- Record training history for later visualization.

Visualize Training and Validation Accuracy:

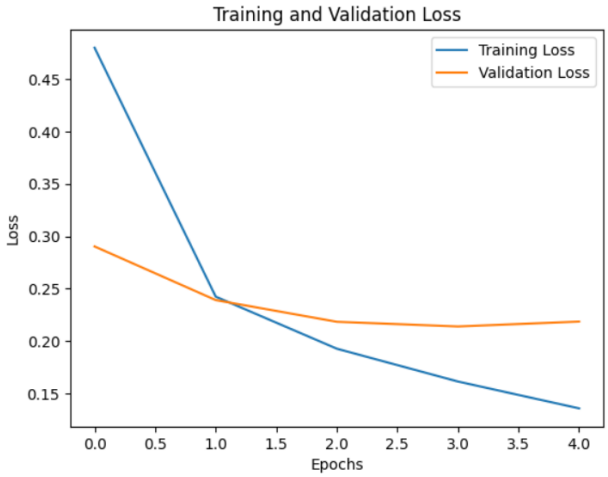
```
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

# Plot training and validation loss values
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
```

```
plt.ylabel('Loss')
plt.legend()
plt.show()
```

- Plot training and validation accuracy over epochs. Over time our goal is to achieve higher accuracy. Validation means that the model can make accurate predictions over time of unseen data. Both factors increase with the number of epochs.



- For training loss, it represents the error of the model in its judgment and the actual results. Moreover, Validation loss represents the number of misjudgments the model makes on unseen data. Both factors decrease by the number of epochs in a functioning model.



Model Evaluation:

```
test_loss, test_accuracy = model.evaluate(test_images, test_labels)
print(f"Test Accuracy: {test_accuracy * 100:.2f}%")
```

- The code evaluates the trained model on the test dataset, retrieves the test accuracy, and prints it as a percentage with two decimal places. The higher the accuracy, the higher the chance for the model to predict correctly.

Visualizing Model Predictions:

```
num_rows = 4
num_cols = 4
num_images = num_rows * num_cols
predictions = model.predict(test_images)
plt.figure(figsize=(10, 10))
for i in range(num_images):
    plt.subplot(num_rows, num_cols, i + 1)
    plt.imshow(test_images[i].reshape(28, 28), cmap='gray')
    plt.title(f'Actual: {np.argmax(test_labels[i])}\nPredicted: {np.argmax(predictions[i])}')
    plt.axis('off')
plt.show()
```

Conclusion:

This project focused on employing Convolutional Neural Networks (CNNs) for Handwritten Letter Recognition using the EMNIST dataset. The process involved importing necessary libraries, loading, and preprocessing the dataset, building and training a CNN model, and evaluating its performance. The project successfully recognized handwritten letters, achieving a notable test accuracy. Visualizations were utilized to provide insights into the model's predictions. While acknowledging achievements, future enhancements were suggested, including further optimization and real-world deployment. Overall, the project highlights the effectiveness of CNNs in addressing the challenges of handwritten character recognition.