Faculty of Engineering and Technology

Electrical and Computer Engineering Department

Computer Vision

ENCS5343

Arabic Handwritten Character Recognition (AHCR) using Convolutional Neural Networks (CNNs).

Prepared by: Majd Abubaha 1190069

Section #2

Instructor: Dr. Aziz Qaroush

Date: 26/1/2024

1 Contents

Table of Figures

1. Introduction

English text, in both its printed and handwritten forms, has been the subject of much research in the broad field of optical character recognition (OCR). The achievements of OCR for English are noteworthy, as they have shown impressive results. However, there is still a clear research void in the field of Arabic text recognition. The Arabic script is unique and each letter may take a variety of complex forms, which is the main reason for this discrepancy.

Throughout its development, Arabic Handwritten Character Recognition (AHCR) presents many complexities and obstacles. Finding a suitable, publicly available Arabic handwritten text dataset is the first step in this process. Difficulties persist throughout the segmentation and feature extraction phases and eventually arise in the recognition and classification phases. In contrast to English, where OCR systems have achieved a high degree of efficiency, AHCR systems have special complexities that arise from the different forms that an Arabic letter may take.

In addition to the technical difficulties, the rich complexity of Arabic calligraphy contributes to the scarcity of study in this field. The 28 letters that make up the Arabic alphabet are written in semi-cursive style, and each letter can have several different forms depending on where it appears in the word. To confront these obstacles, a sophisticated strategy that not only accommodates the nuances of Arabic handwriting but also overcomes the difficulties from data set acquisition to the final recognition stage is necessary.

The goal of this project is to fill this research gap by developing a convolutional neural network (CNN) architecture specifically designed for handwritten Arabic character recognition. This program aims to significantly advance AHCR systems by identifying the unique barriers presented by the Arabic script and leveraging knowledge gained from OCR research in other languages. Using an extensive investigation into segmentation, feature extraction and classification methods, the project seeks to open new ways to process and use handwritten Arabic text in the context of optical character recognition.

2. Datasets

The dataset consisting of training and test files was taken, the files with the CSV extension were chosen as a dataset. For instance, there are two files for training, one of which contains the read images, which are grayscale photos. Every row in the file is a picture, and the pixels of this image are contained in the row.

The photos in the other training file are divided into 28 classes. For the testing files, the same goes.

3. Experimental setup and results

1.1 Task 1:

Using the Arabic Handwritten Characters dataset, the attached code creates a Convolutional Neural Network (CNN) with Keras for picture categorization.

1.1.1 Model Architecture

There are several layers in the model architecture that the code defines:

1. **Layer of Input:**
   - The input shape, (32, 32, 1), represents a 32x32 pixel grayscale picture.
2. **Convolutional Layers:**
   - **Conv2D (32 filters):** Uses the ReLU activation function to apply 32 filters with a size of (3, 3).
   - **Batch Normalization**: accelerates convergence by normalizing the activations of the preceding layer.
   - **MaxPooling2D**: Reduces spatial dimensions by performing max-pooling with a pool size of (2, 2).
3. **Convolutional Layers (Second Block):**
   - **Conv2D (64 filters):** Uses the ReLU activation algorithm to apply 64 filters of size (3, 3).
   - **Batch Normalization**: Normalizes the activations.

- **MaxPooling2D**: Uses max-pooling to reduce the spatial dimensions by a given amount.

4. **Flatten Layer:**
   - Flattens the 2D output from the convolutional layers into a 1D vector for input to the fully connected layers.
5. **Fully Connected (Dense) Layers:**
   - **Dense (128 neurons):** 128 neurons in a fully linked layer with ReLU activation.
   - **Dropout:** To avoid overfitting, use a dropout layer with a rate of 0.5.
   - **Dense (Output Layer):** A fully connected layer for multi-class classification that uses SoftMax activation and neurons equal to the number of classes (output size).

1.1.2 Model Parameters (Weights and Biases)

- **Convolutional Layers**
  Filters (weights) and biases are among the trainable parameters for each convolutional layer.
  In the first Conv2D layer, for instance, there are 32 filters with sizes of (3, 3), for a total of 32x3x3 = 288 trainable parameters per filter plus 32 biases.

- **Dense Layers:**
  These layers contain biases and weights that link every neuron from the previous layer to the current layer.
  For example, the number of features (the number of weights of form) in the first dense layer (128 neurons) connected to the flattened output relies on the output size of the layers that come before it.

- **Total Parameters:**
  The trainable parameters in each layer, including the convolutional and dense layers, as well as biases, define the overall number of parameters in the model.

1.1.3 Training

The model's weights and biases are adjusted throughout training using backpropagation and optimization methods like the Adam optimizer.

For multi-class classification issues, categorical cross-entropy is the employed loss function.

Accuracy serves as the main performance indicator for the model, and a learning rate reduction callback is used to modify the learning rate in order to enhance convergence dynamically.

1.1.4 Evaluation Methodology

- **Training and Validation Split**: The train_test_split function from scikit-learn is used to divide the dataset into training and validation sets. 20% of the data is utilized for validation (X_val, y_val), while the remaining 80% is used for training (X_train, Y_train).
- **Model Training**: Using the training data (X_train, Y_train) and a predetermined batch size (42), the CNN model is trained over a period of 20 epochs. If the validation loss reaches a plateau during training, the learning rate is adjusted using a callback known as ReduceLROnPlateau.
- **Model Evaluation**: The test dataset (X_test, y_test) is used to evaluate the model following training. On the basis of the test results, predictions are generated and evaluation metrics are calculated.
- **Metrics for Evaluation (on Testing Data)**: The assessment metrics listed below are calculated:
  - **Accuracy**: Overall accuracy of the model on the test set.
  - **Precision**: Precision score averaged across all classes.
  - **Recall**: Recall score averaged across all classes.
  - **F1 Score**: F1 score averaged across all classes.
- **Confusion Matrix**: To see how well the model performs on the test set, a confusion matrix is plotted. It displays the numbers of accurate predictions—false negative, false positive, true negative, and true positive.

1.1.5 Plots

- **Training History Plots**: To track the progress of training, plots displaying training accuracy, validation accuracy, training loss, and training accuracy over epochs are displayed.
- **Confusion Matrix**: Use this heatmap-based visualization to see how well the model performs in various classes.

1.1.6 Experimental Setup

As seen in the figures below, the model is first trained for 10 epochs with a batch size of 42. Over time, the training accuracy rises and the training loss gradually declines. The accuracy of verification performance is decreased due to significant variations in verification accuracy and verification loss.

The actual labels of the data are represented by the rows of the matrix, while the anticipated labels are represented by the columns. All of the values in an ideal confusion matrix would be on the diagonal, signifying that every piece of data was properly categorized. The confusion matrix in this instance indicates that there are some errors in the network.

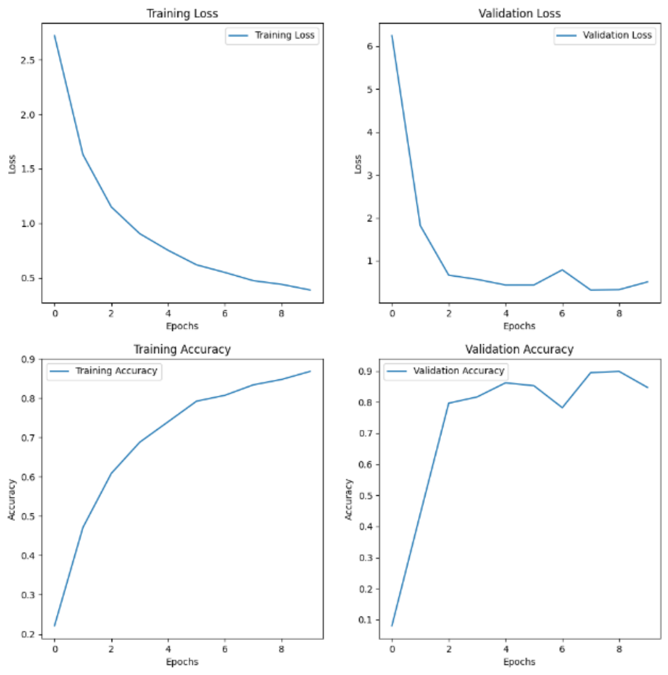


Figure 1. Training with Epoch = 10, and Batch size = 42

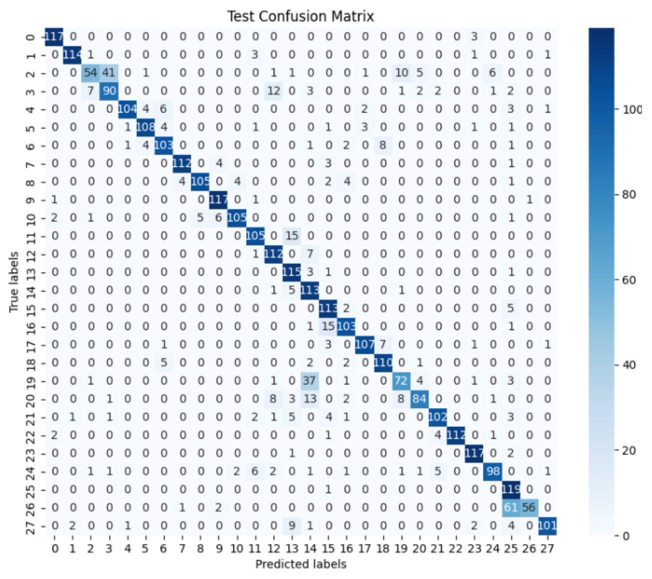
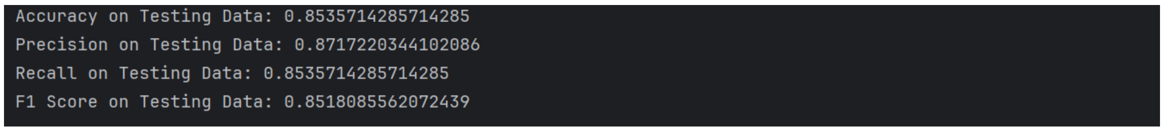


Figure 2. Confusion Matrix for Testing with Epoch = 10, and Batch size = 42



```
Accuracy on Testing Data: 0.8535714285714285
Precision on Testing Data: 0.8717220344102086
Recall on Testing Data: 0.8535714285714285
F1 Score on Testing Data: 0.8518085562072439
```

Figure 3. Testing Evaluation with Epoch = 10, and Batch size = 42

As seen in the figures below, the model is first trained for 20 epochs with a batch size of 42. As a result, both the training accuracy and training loss increased. Additionally, there were less noticeable variations in verification loss and accuracy.

Furthermore, the prediction appears to be more accurate than the preceding one in the confusion matrix.

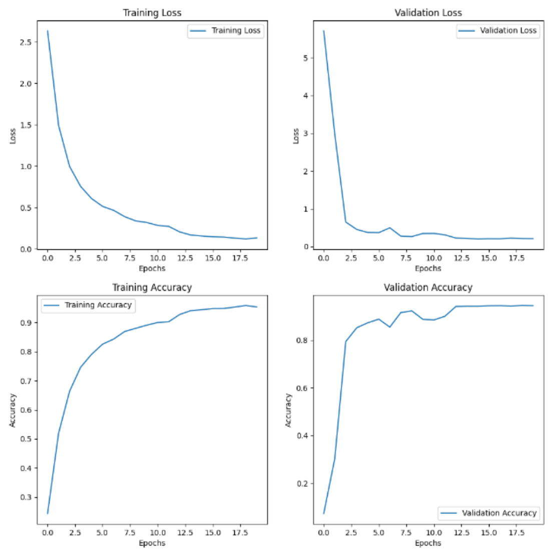


Figure 4. Epoch = 20, batch size = 42

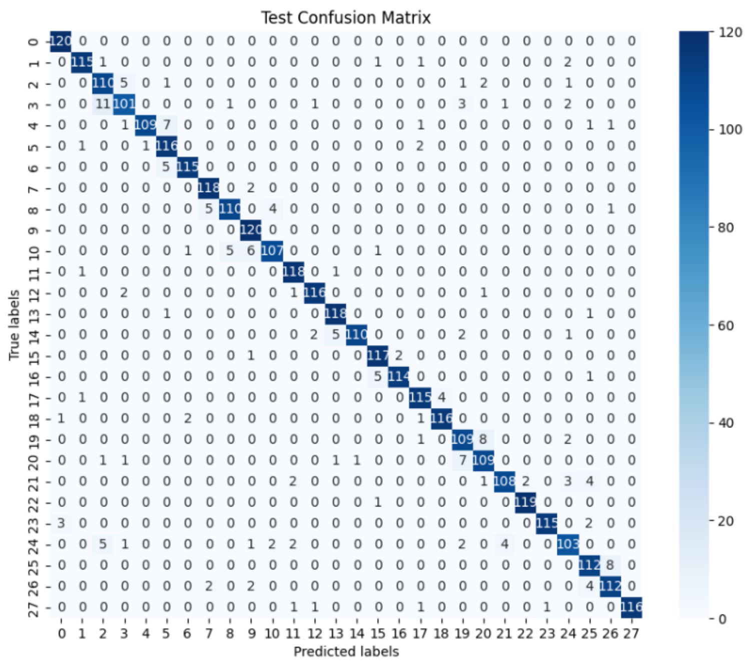
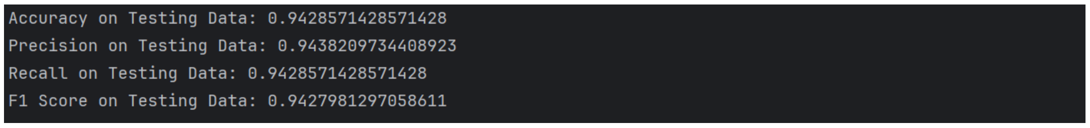


Figure 5. Epoch = 20, batch size = 42



```
Accuracy on Testing Data: 0.9428571428571428
Precision on Testing Data: 0.9438209734408923
Recall on Testing Data: 0.9428571428571428
F1 Score on Testing Data: 0.9427981297058611
```

Figure 6. Epoch = 20, batch size = 42

1.2 Task 2

The second task made use of the same model as task 1, but with 50 epochs, and data augmentation.

1.2.1 Data Augmentation

By applying different changes to the existing photos, a method known as data augmentation is utilized to artificially increase the variety of the training dataset. This enhances the trained model's resilience and generalization. The code supplied applies data augmentation to the training data (X_train) by utilizing the ImageDataGenerator class from Keras.

The augmentations applied:

- ✓ **Rotation Range:** rotation range=10: Rotates the photos up to ten degrees either clockwise or counterclockwise at random.

- ✓ **Width Shift Range:** width shift range = 0.1: This adjusts the photos horizontally at random, up to a maximum of 10% of the overall width.
- ✓ **Height Shift Range:** The photos are randomly shifted vertically by a percentage of the overall height, up to a maximum displacement of 10% of the total height, using the height_shift_range=0.1 parameter.
- ✓ **Shear Range:** shear_range=0.1: Transforms the pictures using a shear maximum intensity of 0.1 radians.
- ✓ **Zoom Range:** zoom_range=0.1: This setting arbitrarily enlarges or reduces the pictures by a factor of 0.1.
- ✓ **Horizontal Flip:** If set to False, the photos can be flipped horizontally. Since it's set to False in this instance, horizontal flipping is not used.
- ✓ **Vertical Flip:** optionally flips the photos vertically using the vertical_flip=False setting. Just as with horizontal flipping, this one is set to False.

Data augmentation helps the model to learn more robust features by exposing it to variations in the training data. It effectively increases the diversity of the training dataset without collecting additional labeled samples.

By introducing variations like rotations, shifts, shearing, and zooming, the model becomes more invariant to such transformations, leading to better generalization performance.

1.2.2 Experimental Results

As we mentioned before, increasing data leads to better generalization performance. The accuracy of the model's performance has increased, as can be seen in the pictures shown below. The f1 score, recall, and precision have all been raised.

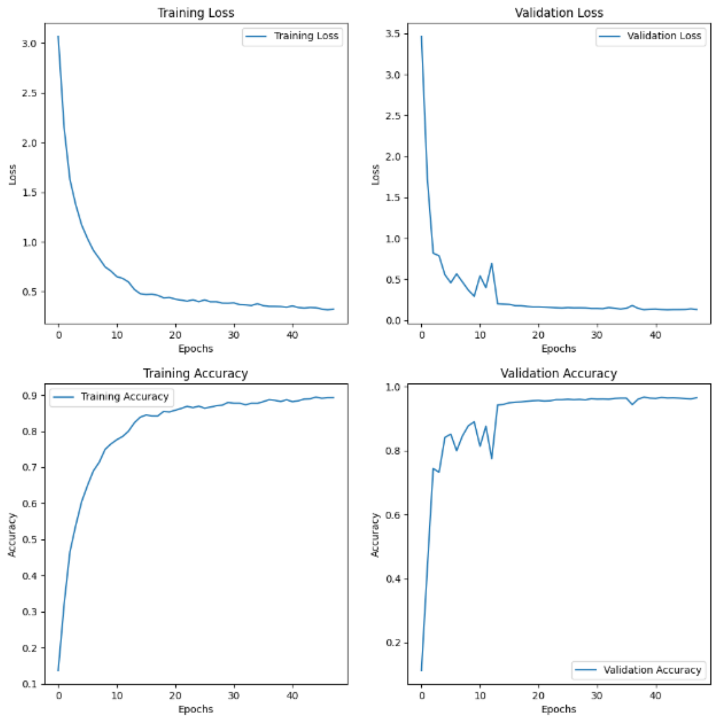


Figure 7. Training with Data Augmentation

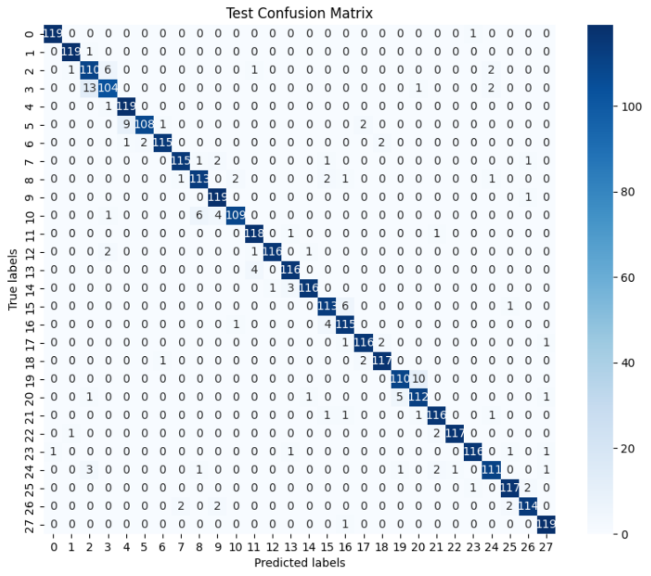
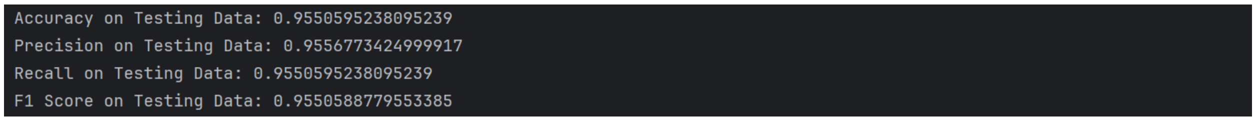


Figure 8. Confusion Matrix with Data Augmentation



```
Accuracy on Testing Data: 0.9550595238095239
Precision on Testing Data: 0.9556773424999917
Recall on Testing Data: 0.9550595238095239
F1 Score on Testing Data: 0.9550588779553385
```

Figure 9. Testing Evaluation with Data Augmentation

1.3 Task 3

1.3.1 Published CNN Networks

Utilizing published CNN designs can result in more efficient development cycles, enhanced model performance, and simpler integration of cutting-edge methods across a range of computer vision applications. To guarantee optimal performance, it is necessary to assess an architecture's appropriateness for a certain application and dataset.

1.3.2 Model Architecture (LeNet)

The model architecture is divided into many layers. Two convolutional layers are used in the beginning, and each is followed by a max-pooling layer to help with feature extraction and reduce spatial dimensions. The output of the convolutional layers is then transformed into a 1D vector by applying a flattening layer, which gets it ready for input into the fully connected layers. Repaired linear unit (ReLU) activation functions are used by these two thick layers to induce non-linearity and discover intricate patterns in the data. Lastly, by using SoftMax activation in the output layer, which provides probabilities for every class, multi-class classification is made easier. Effective feature extraction, nonlinear transformation, and classification are made possible by this organized layering, which makes the model a good choice for applications such as handwritten character recognition.

1.3.3 Model Parameters (Weights and Biases)

In this model, the convolutional and dense layers in particular, have parameters attached to them.

- **Parameters of Convolutional Layers:**
  A collection of learnable filters makes up each convolutional layer. For example, because of the input shape and the number of output channels (6), the first convolutional layer with a filter size of (5, 5) would contain filters of dimensions (5, 5, 1, 6).
  There is a total of 6 * (5 * 5 + 1) parameters for the first convolutional layer and 16 * (5 * 5 + 1) parameters for the second one because each filter also has a matching bias term.

Dense Layers Parameters:

Since every neuron in one layer is coupled to every other neuron in the following layer, the dense layers are fully connected.

The number of neurons in the current layer and the number of neurons in the preceding layer, plus a bias term for each neuron, define the number of parameters in a dense layer.

For example, the first dense layer has 120 neurons, and each of those neurons is linked to 400 neurons from the previous layer (16 * 5 * 5), therefore the total number of parameters is (120 * 400) + 120. In the same way, 84 neurons in the second dense layer are linked to 120 neurons, resulting in (84 * 120) + 84 parameters.

By iteratively adjusting the weights and biases to minimize the loss function, optimization algorithms such as Adam are used to learn these parameters during the training phase. The model has to be properly initialized and optimized for it to learn representations from the data and produce precise predictions.

1.3.4 Experimental Setup & Results

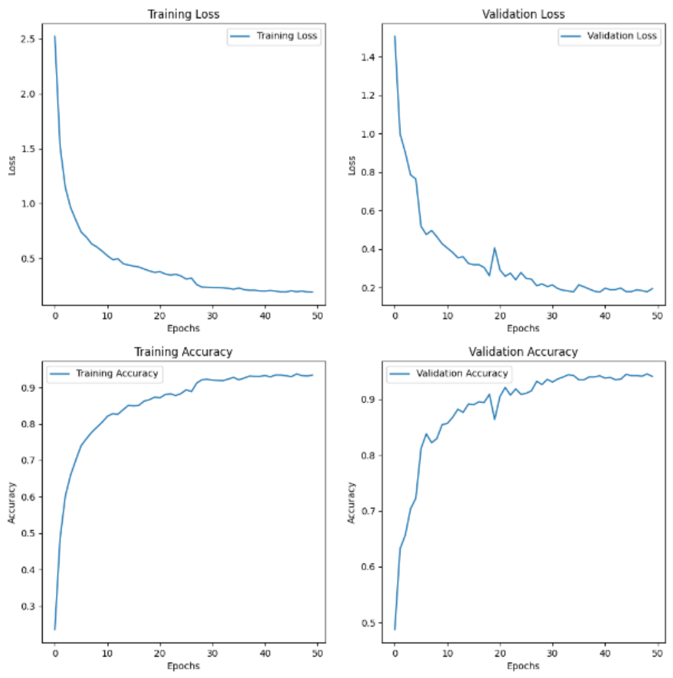


Figure 10. Training LeNet Model

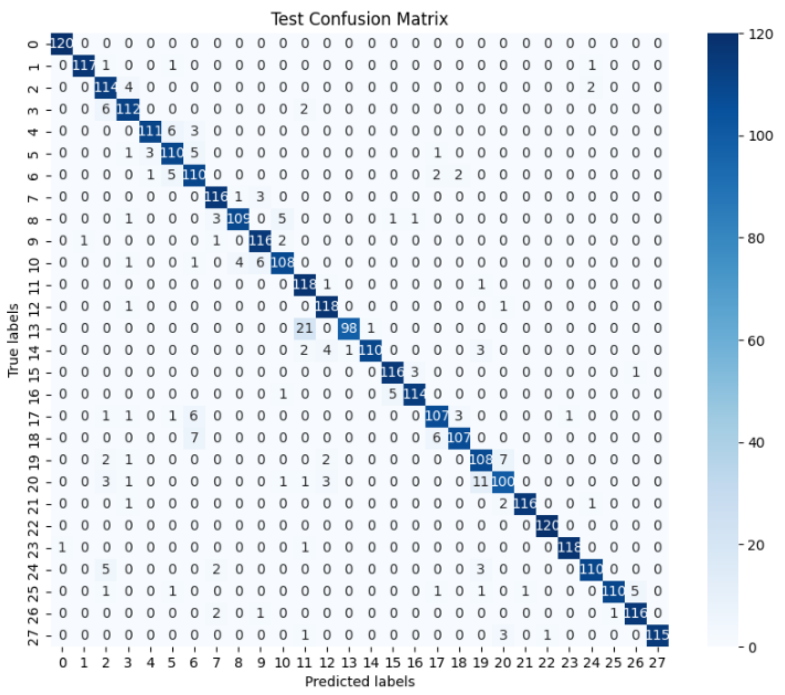
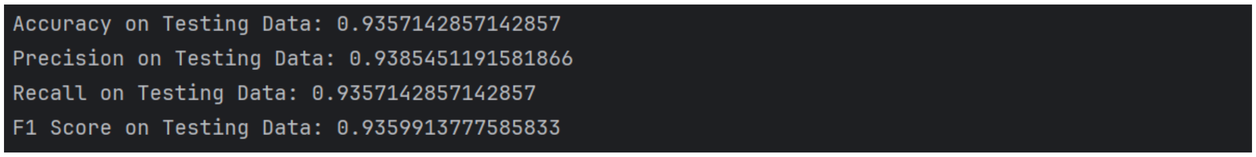


Figure 11. Confusion Matrix for Testing LeNet



```
Accuracy on Testing Data: 0.9357142857142857
Precision on Testing Data: 0.9385451191581866
Recall on Testing Data: 0.9357142857142857
F1 Score on Testing Data: 0.9359913777585833
```

Figure 12. LeNet Evaluation

The LeNet model's testing data findings show some very encouraging outcomes. The model shows a good capacity to accurately categorize handwritten characters from the test set, with an accuracy of around 93.57%. Additionally, the model does a good job of limiting false positives and false negatives, as seen by the excellent accuracy and recall scores, which hover around 0.94. At around 0.94, the F1 score—which combines precision and recall—is likewise strong, indicating that recall and precision were performed in a balanced manner. Overall, these findings show that the LeNet model is appropriate for the job of handwritten character identification since it has effectively acquired significant patterns from the training data and generalizes well to new data.

1.4 Task 4

1.4.1 Pre-Trained Models

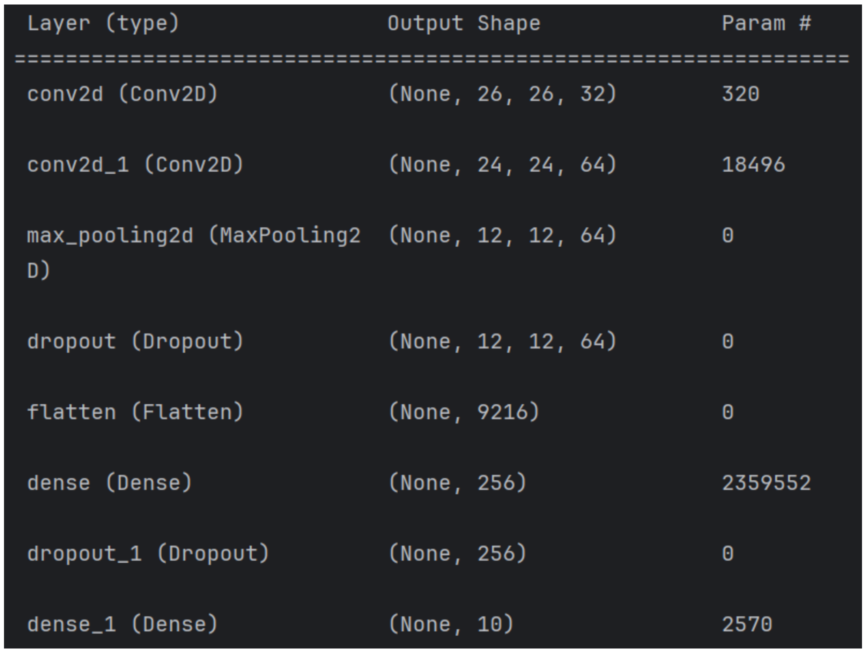
In many situations, using pre-trained networks—especially through transfer learning—is a highly efficient method. These networks are excellent at extracting useful information from photos since they were frequently trained on large and diverse datasets, such as ImageNet. By utilizing these pre-trained convolutional layers, practitioners may take use of the network's ability to acquire features at both low and high levels without having to start training from scratch. This not only creates a strong basis for future model adaption, but it also drastically cuts down on the time and computing resources needed for training.

Pre-trained networks provide a useful option when labeled data is scarce or expensive to collect. Data scarcity presents issues that practitioners can successfully solve by starting with a pre-trained model and fine-tuning it on the available data. Pre-trained networks also include information that has been gleaned from massive quantities of data and processing power, which enables academics and practitioners to build on this foundation and concentrate their efforts on resolving certain issues or improving performance in particular domains.

1.4.2 Experimental Setup & Results

To adapt a pre-trained model that was first trained on the MNIST dataset for digit recognition to a new task—recognizing Arabic handwritten characters—feature extraction transfer learning is used in the task code. First, the code loads "mnist.h5", a pre-trained model that was trained using the MNIST dataset. Because it can capture generic aspects of handwritten numbers, this model may be used as a starting point for identifying comparable patterns in Arabic letters.

Mnist.h5 model includes the following layers:



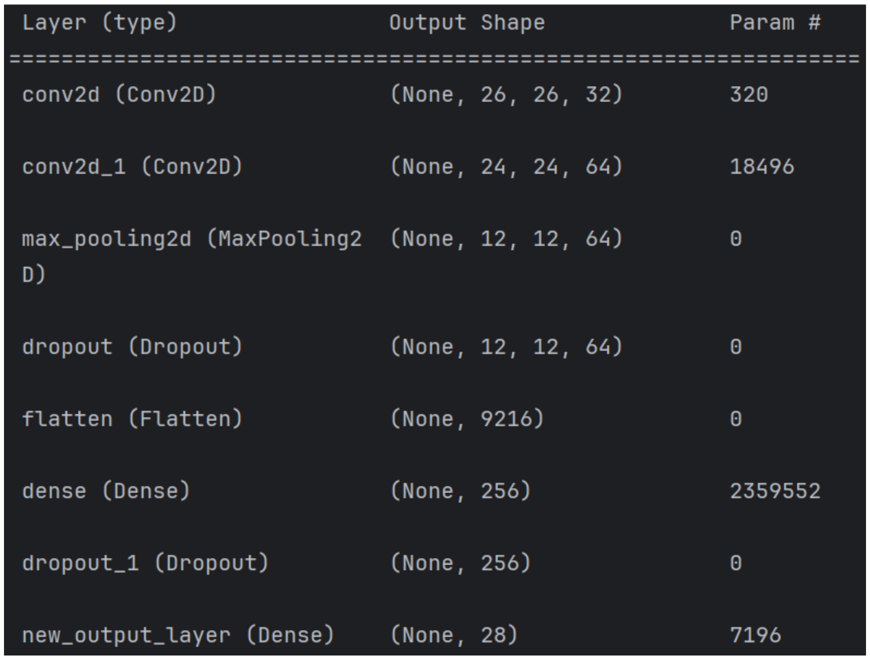
```
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 32)        320

 conv2d_1 (Conv2D)           (None, 24, 24, 64)        18496

 max_pooling2d (MaxPooling2  (None, 12, 12, 64)        0
 D)

 dropout (Dropout)           (None, 12, 12, 64)        0

 flatten (Flatten)           (None, 9216)              0

 dense (Dense)               (None, 256)               2359552

 dropout_1 (Dropout)         (None, 256)               0

 dense_1 (Dense)             (None, 10)                2570
```

Figure 13. MNIST model summary

- **The first convolutional layer:** called Conv2D, applies 32 filters with a size of (3, 3) to pick up subtle details like edges and textures. Shape of output: (26, 26, 32).
- **The second convolutional layer:** Applying 64 filters with a size of (3, 3) allows the second Conv2D layer to collect higher-level information. Shape of output: (24, 24, 64).
- **MaxPooling2D Layer (Pooling Layer):** Reduces spatial dimensions by half and concentrates on significant characteristics by performing max-pooling with a pool size of (2, 2).
- **Dropout Layer 1:** Added after the pooling layer to minimize overfitting by randomly setting a portion of input units to zero during training.
- **Dropout Layer 2:** Added after the initial dense layer to provide robust feature learning and further regularize the model.
- **Flatten Layer:** In order to prepare the 3D output from the convolutional layers for input into the fully connected layers, the flatten layer transforms it into a 1D vector.
- **Dense Layers:** The first dense layer learns complicated patterns and representations from the flattened input by activating 256 neurons using ReLU.

Ten neurons in the second dense layer represent the ten classes (numbers 0-9) in the output layer. It facilitates categorization by producing probability for each class using softmax activation.

The code loads the pre-trained model and then adapts it to the new task. The original output layer is eliminated in this change, and a new output layer specifically designed to categorize Arabic letters is added. In this way, the design of the model is tailored to the particular needs of the target task, guaranteeing that it can recognize and acquire Arabic letters.



```
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 32)        320

 conv2d_1 (Conv2D)           (None, 24, 24, 64)        18496

 max_pooling2d (MaxPooling2  (None, 12, 12, 64)        0
 D)

 dropout (Dropout)           (None, 12, 12, 64)        0

 flatten (Flatten)           (None, 9216)              0

 dense (Dense)               (None, 256)               2359552

 dropout_1 (Dropout)         (None, 256)               0

 new_output_layer (Dense)    (None, 28)                7196
```

Figure 14. MNIST Model Summary after Adding the New Layer

Moreover, the method freezes the pre-trained model's convolutional layers to make use of the useful characteristics that the model has learned while preventing overfitting the new dataset. By doing this, the learned representations of the features taken from the MNIST dataset are preserved and the weights of these layers are kept from changing throughout training.

The Arabic handwritten character dataset is then used to train the updated model. Only the weights of the recently inserted output layer are changed during training; the convolutional layers keep their previously acquired features. By using this method, the model can make use of both the unique properties of the Arabic handwritten characters dataset and the generic features discovered by the pre-trained model.

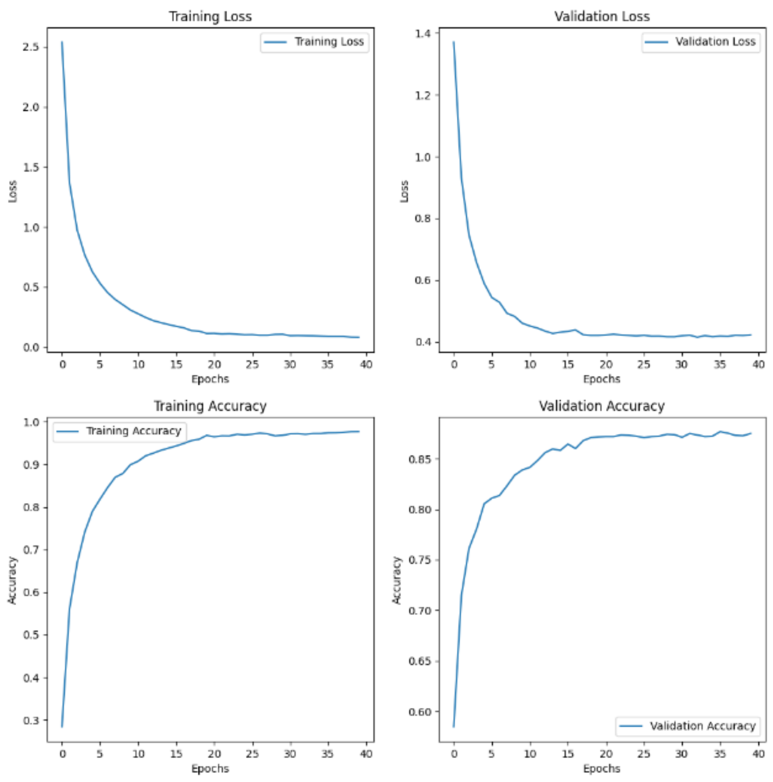


Figure 15. Training the Pre-Trained Model

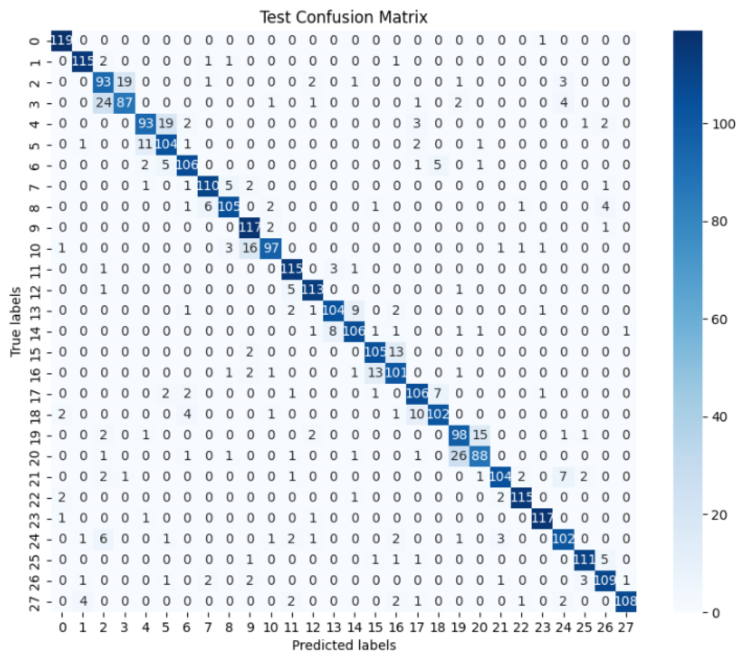
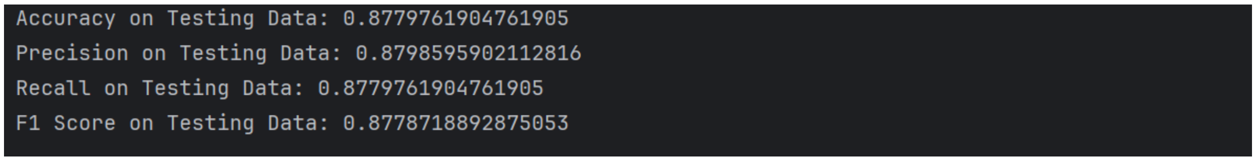


Figure 16. Confusion Matrix for Testing the Pre-Trained Model



```
Accuracy on Testing Data: 0.8779761904761905
Precision on Testing Data: 0.8798595902112816
Recall on Testing Data: 0.8779761904761905
F1 Score on Testing Data: 0.8778718892875053
```

Figure 17. Evaluation of the Pre-Trained Model

The results show that the modified pre-trained model performs well in recognizing Arabic handwritten characters, achieving a balance between precision, recall, and accuracy.

4. Conclusion

The development of a convolutional neural network architecture especially for Arabic Handwritten Character Recognition (AHCR) was the goal of this project. The effort effectively determined the distinct obstacles posed by Arabic script and utilized insights obtained from OCR investigations in other languages.

A modified pre-trained model that struck a compromise between precision, recall, and accuracy by thoroughly researching segmentation, feature extraction, and classification techniques was created. Accuracy, precision, recall, and F1 score were among the measures used to assess the model, and its performance was shown as a confusion matrix.

The project also explored data augmentation techniques and experimented with different CNN architectures, including LeNet. The results obtained showed promising improvements in the recognition and classification of Arabic handwritten characters.