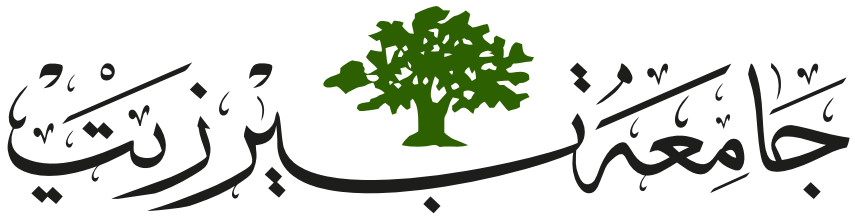


BIRZEIT UNIVERSITY

Department of Electrical & Computer Engineering

ENCS5343 - COMPUTER VISION

Arabic Handwriting Character Recognition

Prepared by:
Yazan Abd ALMutee 1190145
Mohamed Shkierat 1190702
Instructor: Dr.Aziz Qaroush & Ismail Khater
Date: January 31, 2024

Abstract

The main aim of this project is to develop a fully functioning Arabic Handwriting Character Recognition system using different convolution neural network architectures , and to comprehensive compare their scores and evaluation , and finally to transfer learning a CNN with similar task into the project's task.

Contents

1 Introduction **1**
1.1 Problem statement 1
1.2 Objectives 1

2 Experimental setup and results **2**
2.1 Data Set Selection 2
2.2 Data set processing 3
2.3 Parameters tuning 4
2.4 Creating Baseline CNN model (Task 1) 5
2.5 Training the model with augmented data (Task 2) 7
2.6 Training published CNN model with augmented data (Task3) 10
2.6.1 ALEX-NET 10
2.6.2 Res-net 50 12
2.7 Pre-trained model trained (Task 4) 14
2.7.1 Standard data set training 16
2.7.2 Augmented data set training 17

3 Conclusion **18**

List of Figures

2.1 Sample of the data set . . . 2
2.2 Data set split . . . 3
2.3 classes distribution . . . 3
2.4 Hyper-parameters tuning . . . 4
2.5 Baseline CNN Model . . . 5
2.6 Baseline Model Training/Testing results . . . 5
2.7 Baseline Model Training/Testing results Plot . . . 5
2.8 Test sample on Baseline model . . . 6
2.9 data augmented Model Training/Testing results . . . 8
2.10 data augmented Model Training/Testing results Plot . . . 8
2.11 Test sample on data augmented model . . . 9
2.12 Alex-net Model Training/Testing results . . . 10
2.13 Alex-net Model Training/Testing results Plot . . . 10
2.14 Alex-net model test sample . . . 11
2.15 res-net 50 model Training/Testing results . . . 12
2.16 res-net 50 model Training/Testing results plot . . . 12
2.17 res-net 50 model sample test . . . 13
2.18 model accuracy on Minst data set . . . 14
2.19 Pre-trained tuned model description . . . 15
2.20 Pre-trained model scoring . . . 16
2.21 Pre-trained model training/testing result plot . . . 16
2.22 Pre-trained model sample test . . . 16
2.23 Pre-trained model scoring result . . . 17
2.24 Pre-trained model training/testing plot . . . 17
2.25 Pre-trained model sample test . . . 17

1 Introduction

1.1 Problem statement

Handwriting recognition is a critical area in computer vision, involving the conversion of handwritten text into a digital format. This text can come from various sources, both physical (like paper) and digital (such as touchscreens). The challenge in handwriting recognition varies significantly across languages due to diverse handwriting styles influenced by individual characteristics. In recent times, several machine learning approaches, including K-nearest neighbors, Support Vector Machines, and advanced techniques like Neural Networks and Convolutional Neural Networks, have been explored for this purpose. The complexity increases with languages like Arabic, which has unique characteristics such as semi-cursive script, right-to-left writing, and variable character shapes depending on their position in words, making its automatic recognition a complex task.

1.2 Objectives

This project aims to acquaint students with the intricacies of Arabic Handwritten Character Recognition (AHCR) using Convolutional Neural Networks (CNNs). Key goals include:

- Developing a fundamental understanding of CNNs by constructing a basic network for AHCR. This includes defining network architecture, understanding loss functions and optimizers, conducting training and evaluation, and learning about performance visualization.

- Advancing knowledge through exploration of sophisticated techniques. This involves discussing data augmentation methods to enhance model adaptability, utilizing pre-trained CNN models for AHCR, and introducing students to more complex CNN architectures and their potential applications in AHCR.

2 Experimental setup and results

This Section Describes the evaluation methodology, including the datasets used, evaluation metrics, and experimental setup. also Presents , analyze, and discusses the results of the evaluation, including accuracy and loss curves. Analyze the effectiveness of using data augmentation, published CNN networks, and pre-trained networks.

2.1 Data Set Selection

The used dataset for this project contains 16800 gray-scaled 32x32 size images , where 600 image sample per class (Letter) , and the data set was collected from 60 participants the range of their ages between 19 to 40 years old , also 90% of the participants are right-handed [1]. The following figure displays a sample of the data attached with it's label.

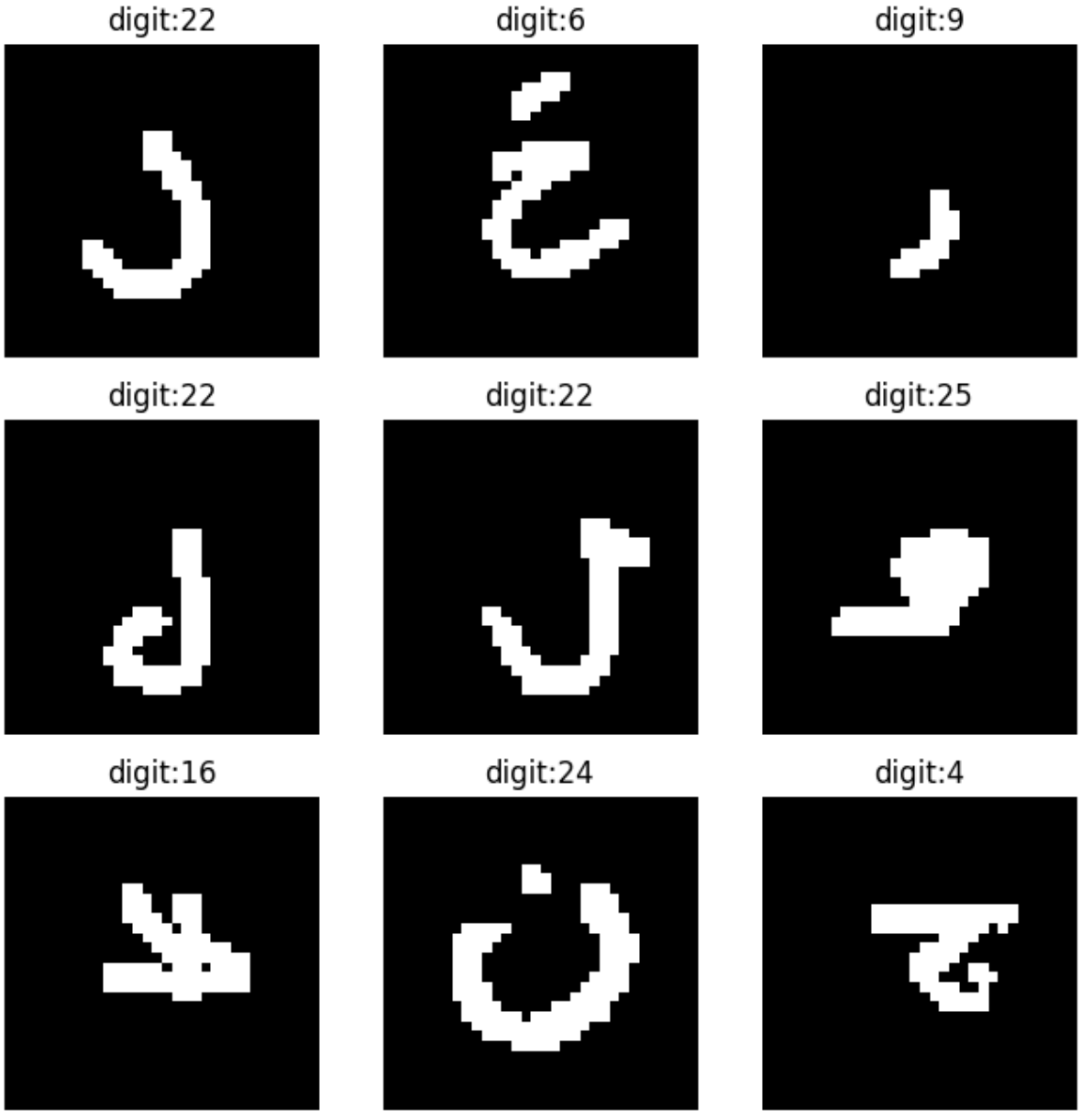


Figure 2.1: Sample of the data set

2.2 Data set processing

The used approach for training testing is splitting the data set using the (80/20) rules , where 80% of the data goes for the Training ,while the rest (20%) goes for the training, so the 16800 collected images will be split into (13440 training images , 3360 testing images) , so with this data split , there will be 480 image per class for training data split , and 120 image per class for testing data split . The following figure illustrates the distribution of the dataset .

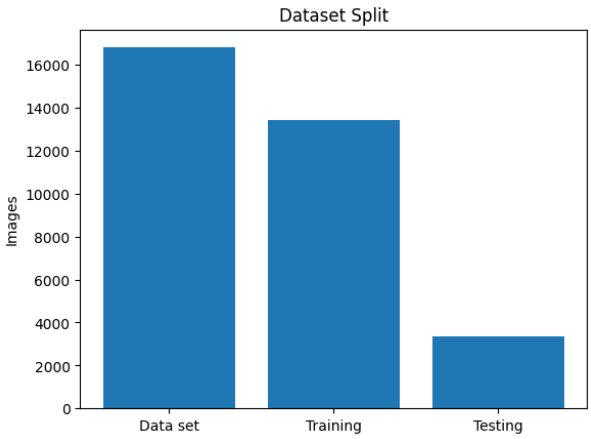


Figure 2.2: Data set split

And as mentioned before , the collected data set were already labeled where each label contains 480 image sample (600 image * 28 class label = 16800 total) , the following figure illustrates the distribution of images among the 28 class.

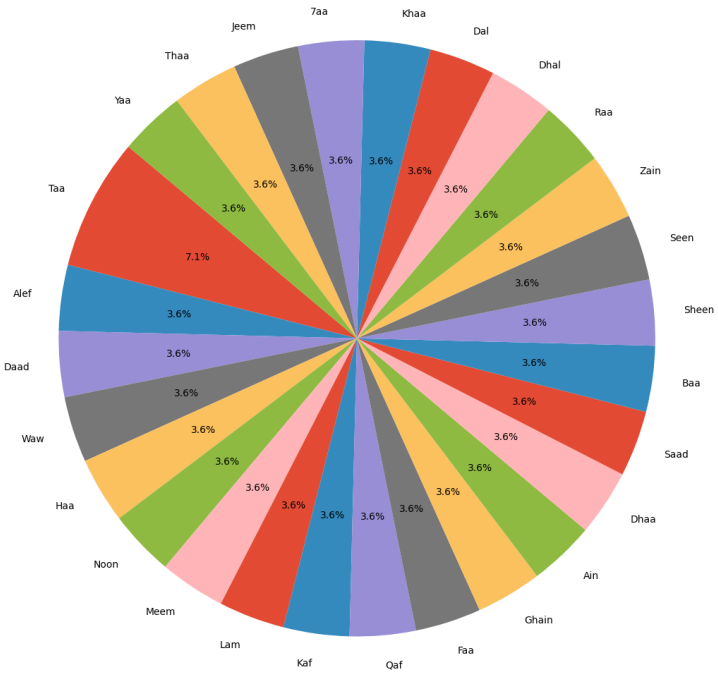


Figure 2.3: classes distribution

2.3 Parameters tuning

Based on the task requirements , which is classifying the handwriting of Arabic letters and recognize it , and with the chosen data set where the images with one channel (gray-scale single color) .The following code displays the choosing parameter tuning scenarios for output channels , kernel size , drop out rate and learning rate .

```
from sklearn.model_selection import ParameterGrid

param_grid = {
    'output_channels': [32, 64],
    'kernel_size': [3, 5],
    'dropout_rate': [0.2, 0.5],
    'learning_rate':[0.001,0.0001]
}
```

The convolution out evaluated based on the following

$$\boxed{\text{Con out} = \left(\frac{\text{Image size} - \text{Kernel size} + 2 \times \text{padding}}{\text{stride (default} = 1)}\right) + 1}$$

As for the padding size , it depends on the kernel size , based on the following equation :

$$\boxed{\text{Pool\_out} = \frac{\text{image size} - \text{filter size} + 2 \times \text{padding}}{\text{stride}} + 1}$$

```
    for params in param_combinations:

    if params['kernel_size'] == 3:
        params['padding'] = 1
    elif params['kernel_size'] == 5:
        params['padding'] = 2
```

and the Flatt parameters are evaluated based on the following equation :

$$\boxed{\text{Flatt} = \text{image size} \times \text{image size} \times \text{output channels}}$$

After the parameters tuning , the following parameters gained the best test Accuracy :

```
Best Test Accuracy: 0.9485 with hyperparameters: {'dropout_rate': 0.5, 'kernel_size': 5, 'learning_rate': 0.001, 'output_channels': 32, 'padding': 2}
```

Figure 2.4: Hyper-parameters tuning

2.4 Creating Baseline CNN model (Task 1)

in this Task , a base line model was created , the choice of making a baseline model comes after surfacing research papers for similar tasks. Figure below displays the architecture of the chosen model with the parameters which were evaluated from the tuning process.

40 epochs iteration were used to train the model

The Testing results for baseline model with normal Dataset = **94.08 %**

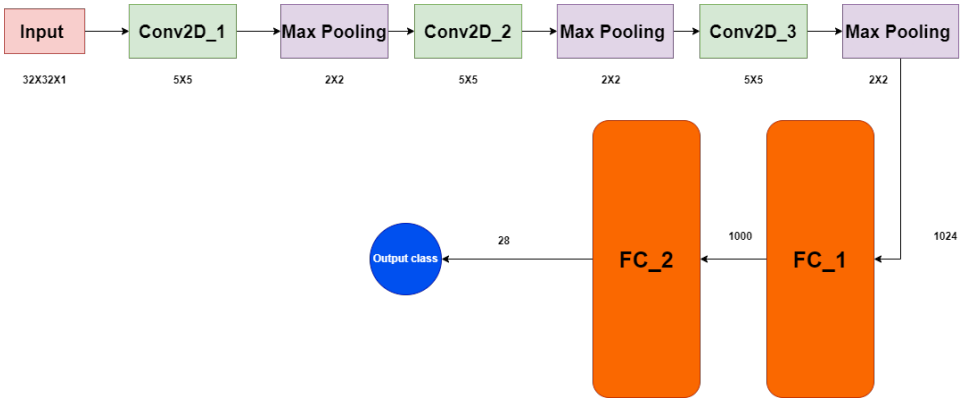


Figure 2.5: Baseline CNN Model

The following figure illustrates Training and testing accuarcy/loss over Epoch.

```
Weighted Precision: 0.9455
Weighted Recall: 0.9408
Weighted F1 Score: 0.9410

Test Accuracy of the model on the test set: 94.08 %
```

Figure 2.6: Baseline Model Training/Testing results

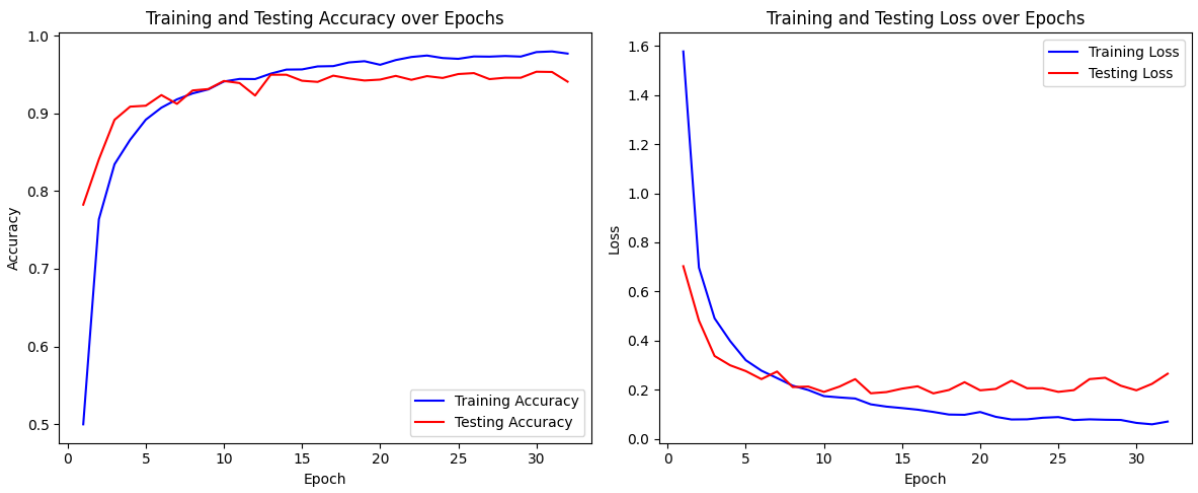


Figure 2.7: Baseline Model Training/Testing results Plot

The Following figure illustrates a random test samples tested on the model :

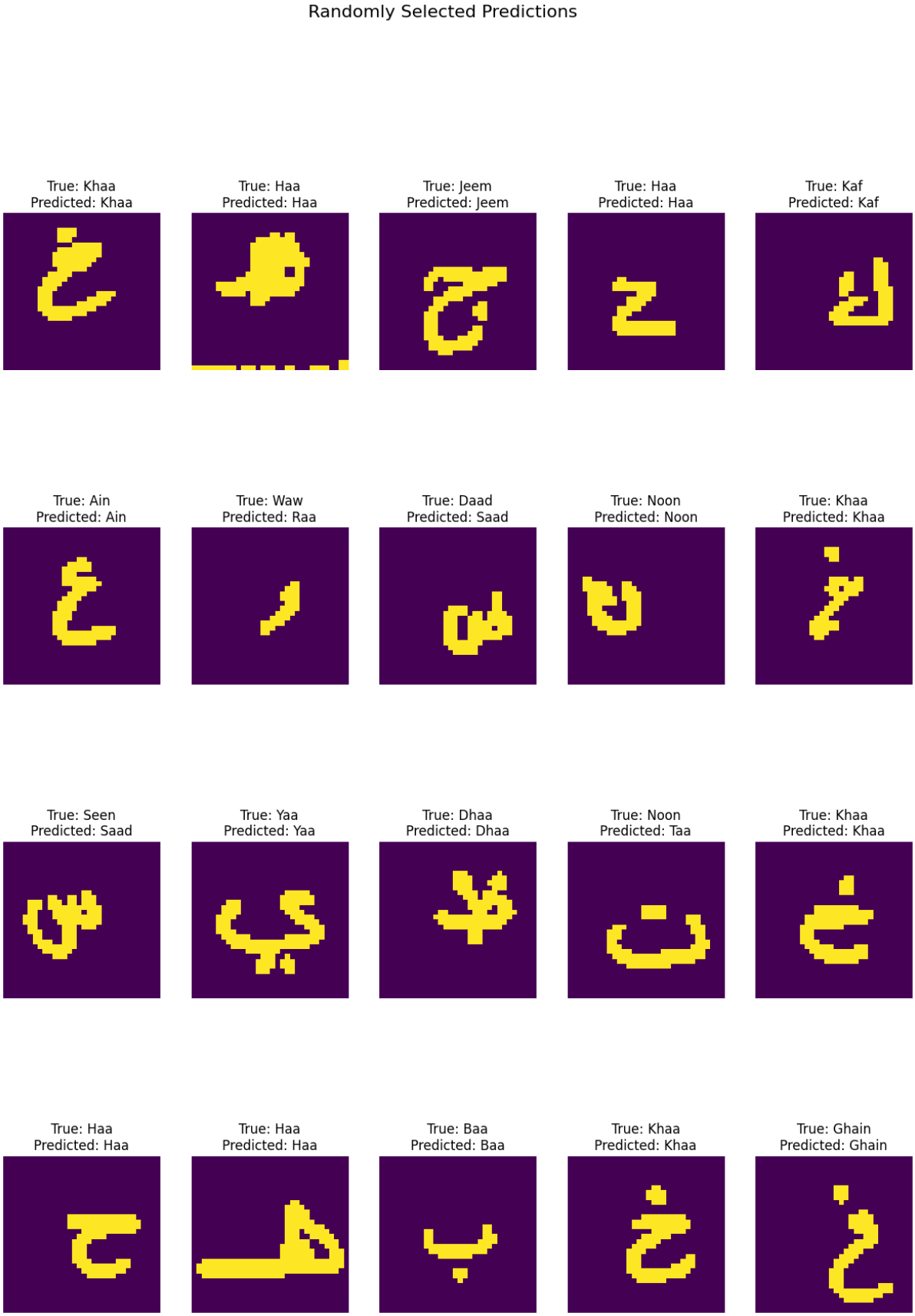


Figure 2.8: Test sample on Baseline model

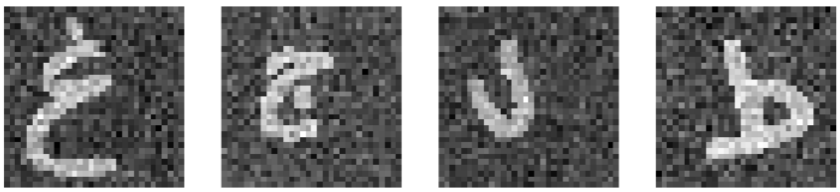
2.5 Training the model with augmented data (Task 2)

after building up the baseline CNN model and testing it with the data set , data augmentation methods were used for improving the model , 5 data augmentation techniques were used :

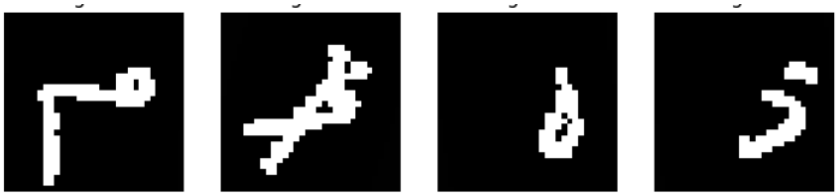
- Random Rotate (Range in 70 degrees)

- Random Blur with Gaussian Blur(kernel size=3)

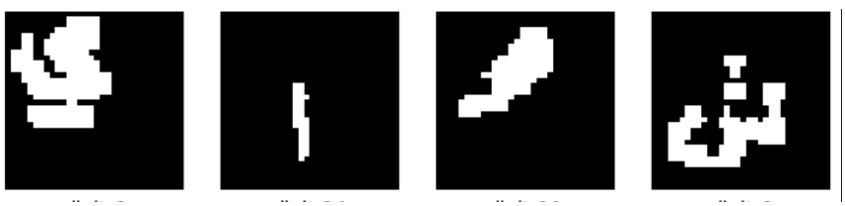
- Random Noise (mean=-0.1, std=0.2)



- Random shear (shear=20)



- Random shift with translate=(0.15, 0.15), scale=(0.67, 1.47)



Every time the data set gets augmented , the size increases , so with the 5 augmented data techniques used , the data set size get buffed 5 times , from 13440 image into 67,200 image , note that the augmentation only applied on the data set and excluded the testing set.

40 epochs iteration were used to train the model.

The result of using the data augmentation is a positive improve to the accuracy , the final accuracy : **96.31%** which is 2.23% improvement

```
Weighted Precision: 0.9640
Weighted Recall: 0.9631
Weighted F1 Score: 0.9631

Test Accuracy of the model on the test set: 96.31 %
```

Figure 2.9: data augmented Model Training/Testing results

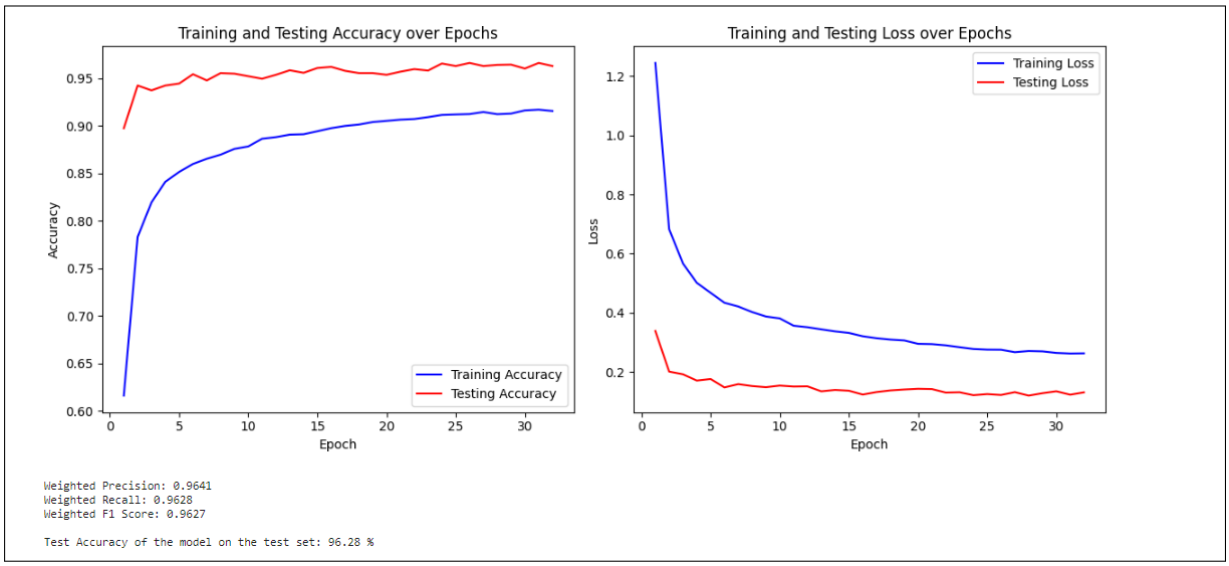


Figure 2.10: data augmented Model Training/Testing results Plot

The Following figure illustrates a random test samples tested on the model :

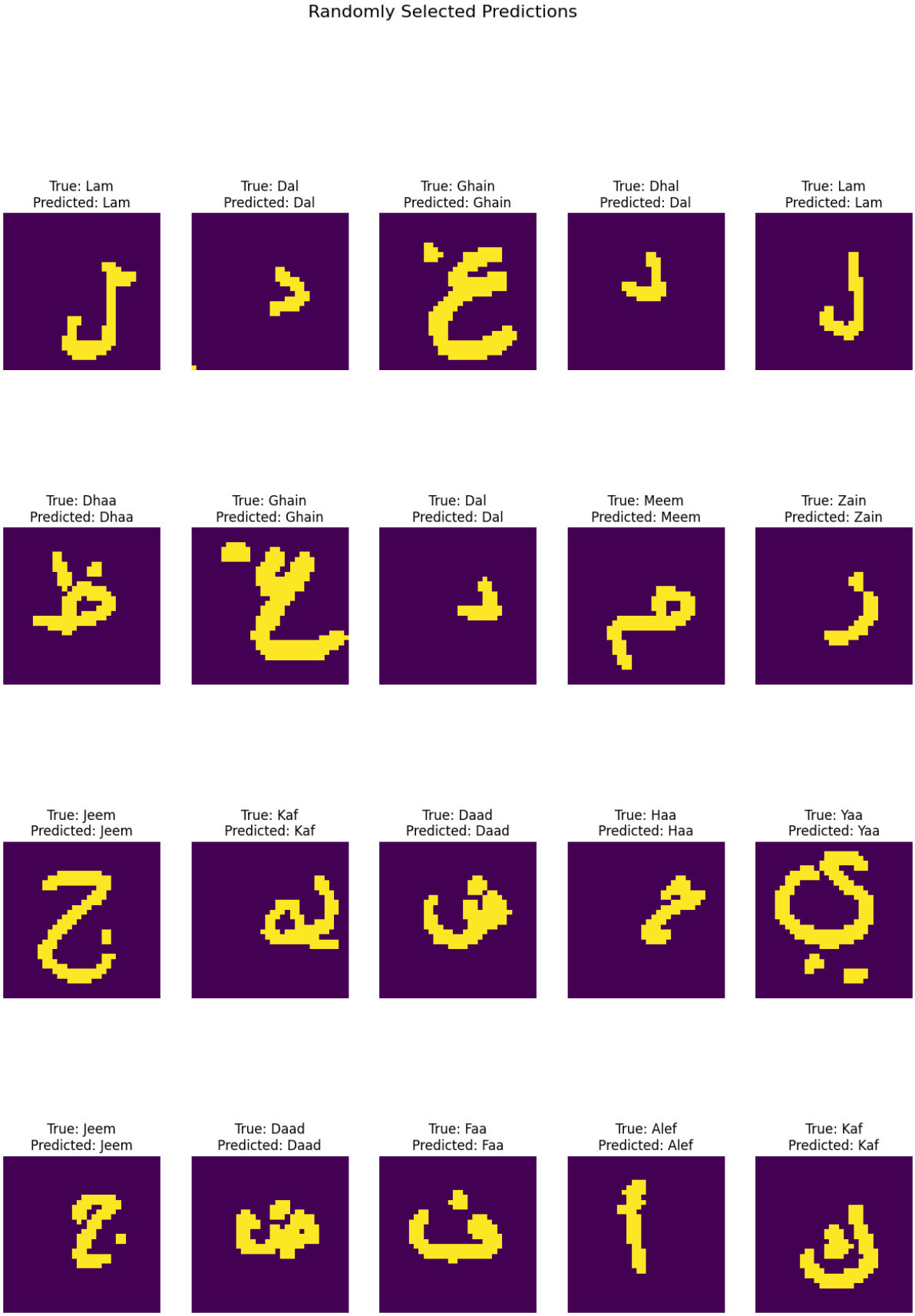


Figure 2.11: Test sample on data augmented model

2.6 Training published CNN model with augmented data (Task3)

This task is choosing a well-knows published CNN model and train it using the augmented data set that was used in task 2. two well-known were picked to be examined , **ALEX-NET** and **Res-Net50**

2.6.1 ALEX-NET

Alex-net CNN has 8 layers , the first and the last layers were tuned to suit the task.

The input images of the data were up-scaled from 32x32 to 224x224.

The first layer parameter channel number changed from 3 to 1 .

```
from torchvision import models

alexnet_model = models.alexnet(weights=None)
alexnet_model.features[0] = nn.Conv2d(1, 64, kernel_size=11, stride=4,
↪  padding=2)
```

And the last layer were tuned into 4096 input and 28 output

```
alexnet_model.classifier[6] = nn.Linear(4096, 28)
```

This model scored **96.07 %** accuracy.

```
Weighted Precision: 0.9613
Weighted Recall: 0.9607
Weighted F1 Score: 0.9607

Test Accuracy of the model on the test set: 96.07 %
```

Figure 2.12: Alex-net Model Training/Testing results

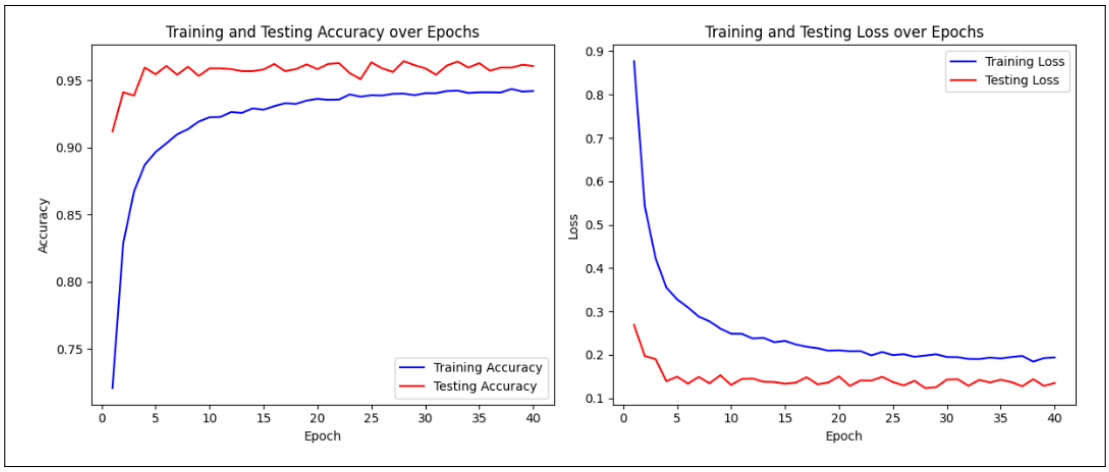


Figure 2.13: Alex-net Model Training/Testing results Plot

The Following figure illustrates a random test samples tested on the model :

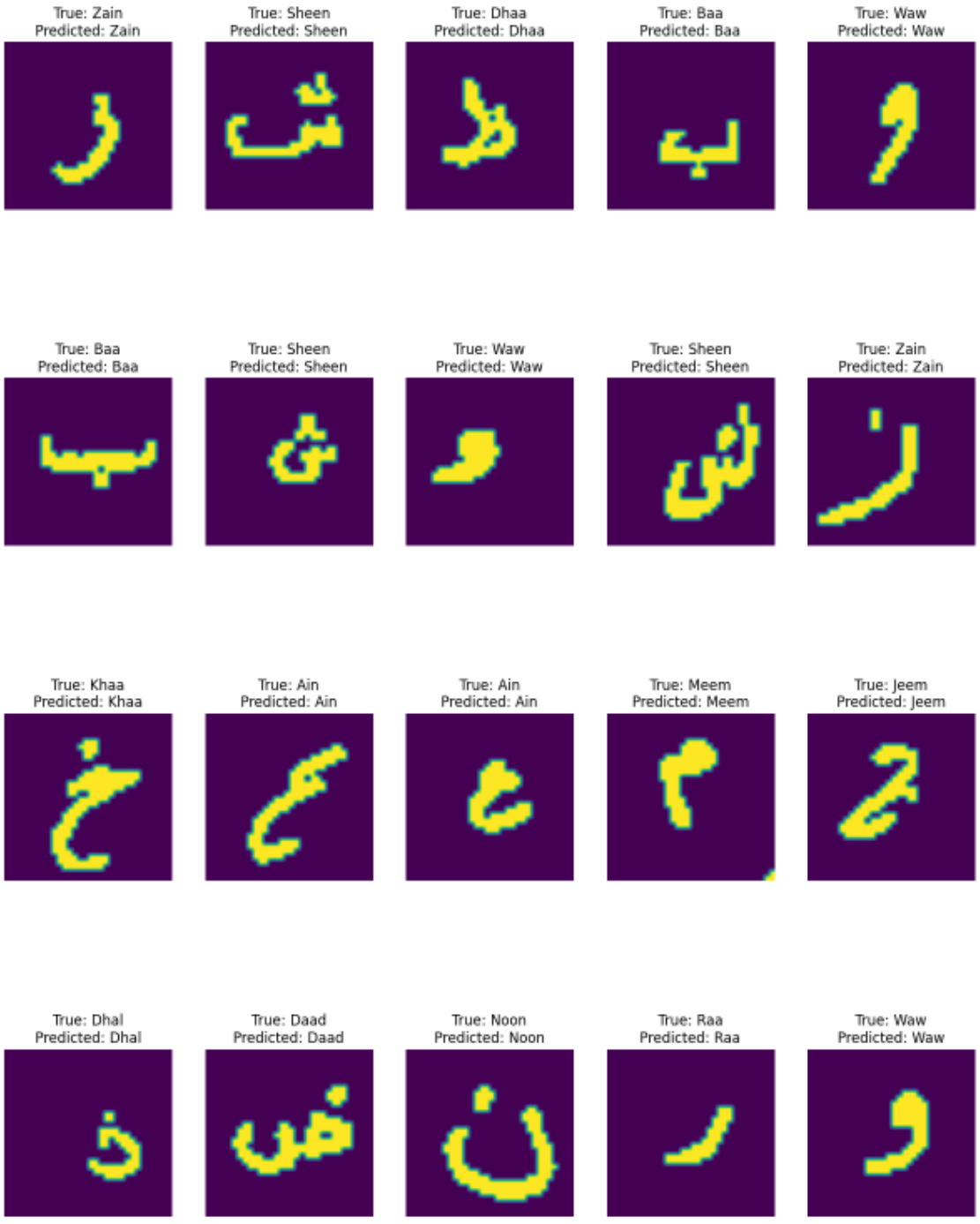


Figure 2.14: Alex-net model test sample

2.6.2 Res-net 50

The second used published CNN model which has 50 layers , this model was picked because of the huge number of layers (50 Layers) , thought the training process , the model took 150min to complete the training of 40 epoch .

The input images of the data were up-scaled from 32x32 to 224x224.

Same as Alex-net parameters tuning the res-net was turned.

```
resnet50_model = models.resnet50(weights=None)

resnet50_model.conv1 = nn.Conv2d(1, 64, kernel_size=7, stride=2, padding=3,
↪  bias=False)

resnet50_model.fc = nn.Linear(resnet50_model.fc.in_features, 28)
```

This model scored **96.19%** accuracy.

```
Weighted Precision: 0.9626
Weighted Recall: 0.9619
Weighted F1 Score: 0.9618

Test Accuracy of the model on the test set: 96.19 %
```

Figure 2.15: res-net 50 model Training/Testing results

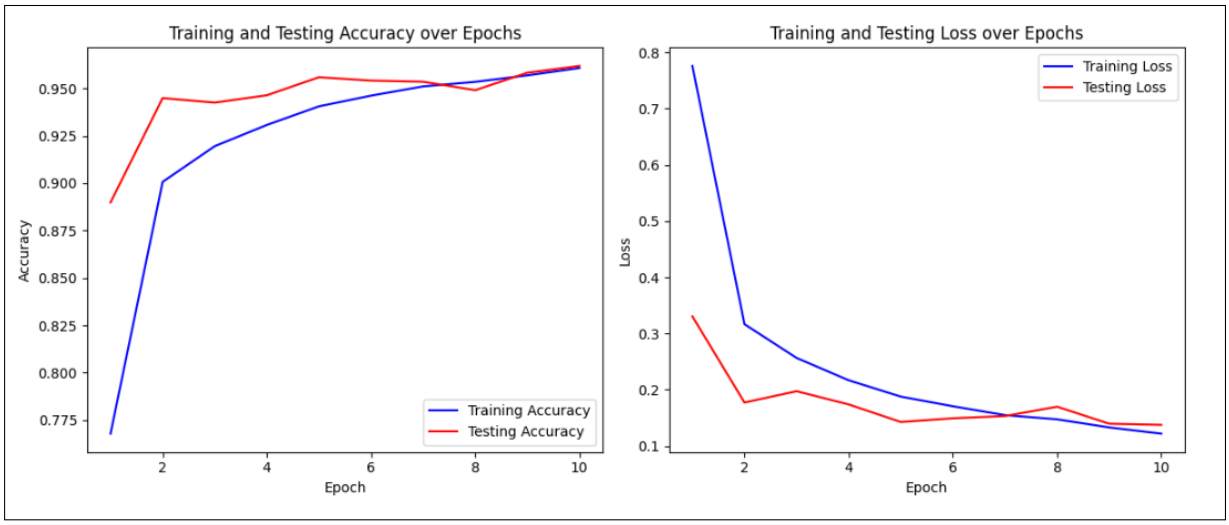


Figure 2.16: res-net 50 model Training/Testing results plot

The Following figure illustrates a random test samples tested on the model :

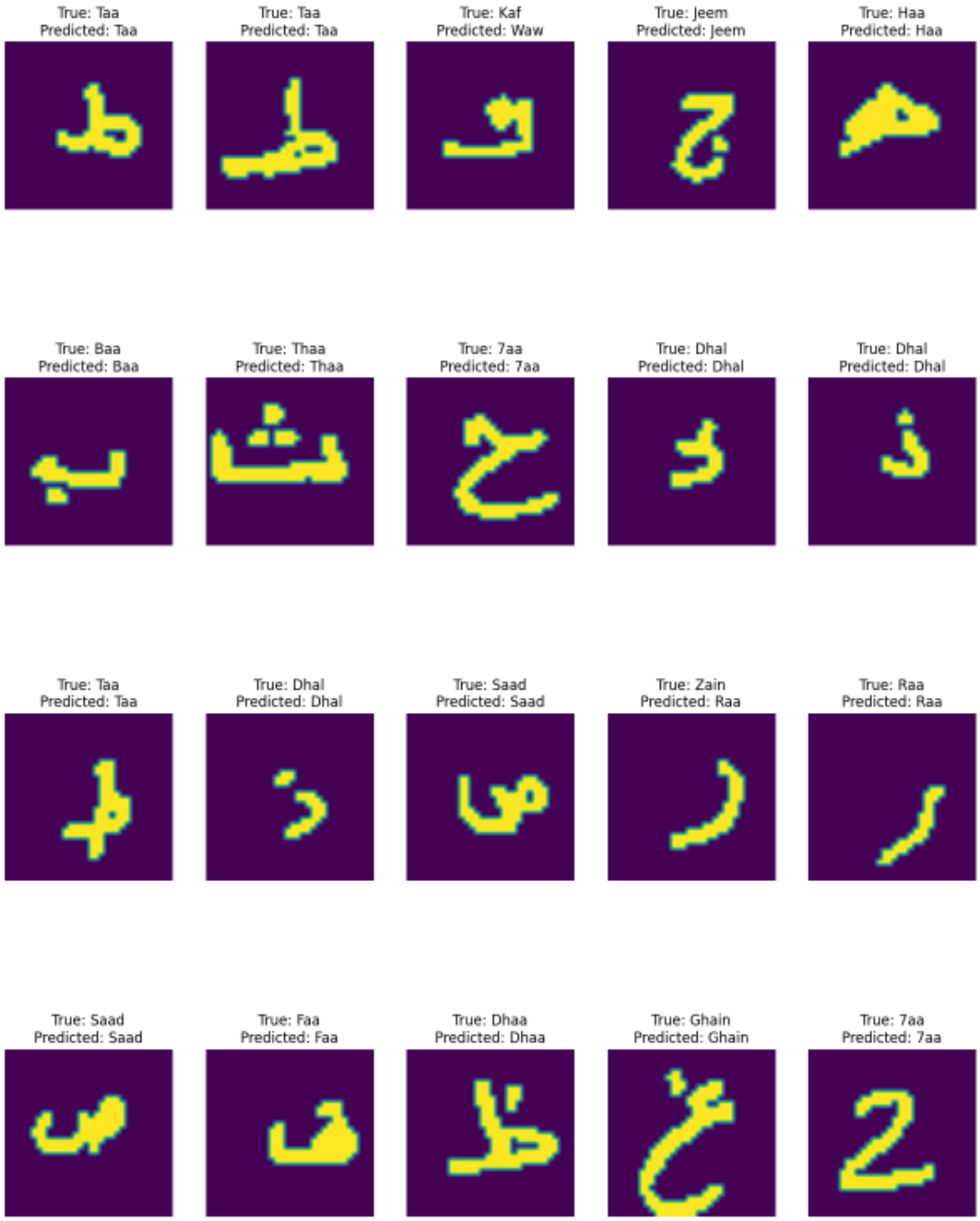


Figure 2.17: res-net 50 model sample test

The trade-of between the two models was in complexity and space and time , where the Alex-net scored slightly lower than Res-net but in a sensible time frame while the Res-net scored better but in a huge time difference and more complicity , so the better choice is the Alex-net.

2.7 Pre-trained model trained (Task 4)

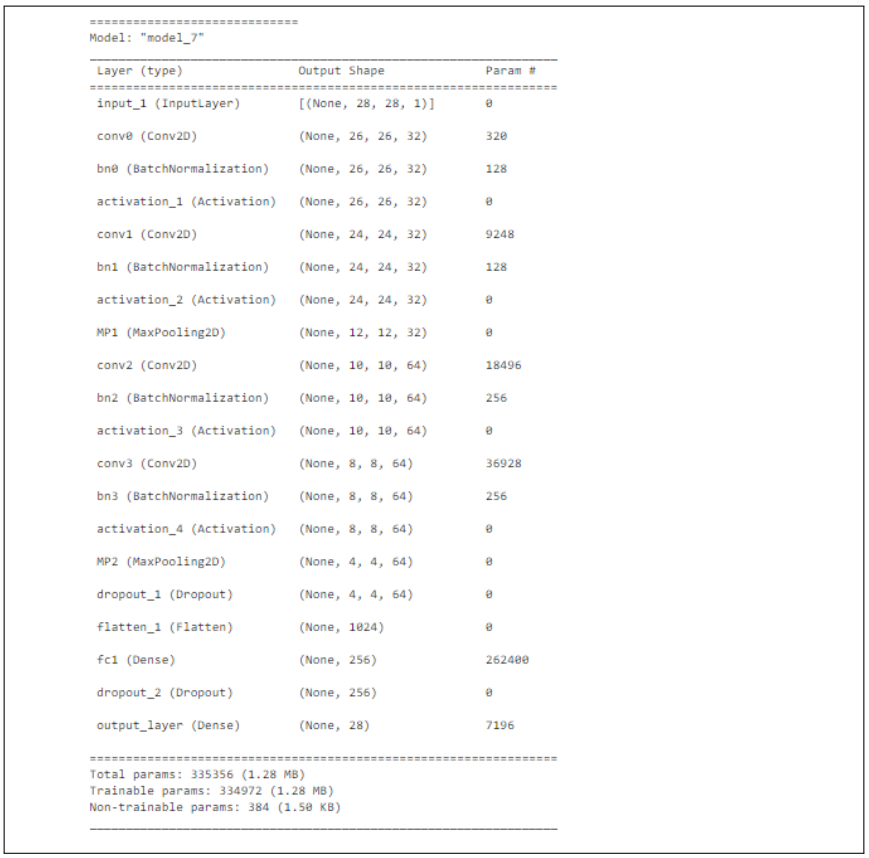
This task is using a pre-trained model that was used for similar tasks like digits recognition or English letter recognition . The picked pre-trained model is the same model that used for minst digits classification task.
The model was found from GitHub repo. from this citation [2].

```
model = load_model('/kaggle/input/datasetminist/MNIST_keras_CNN.h5')
(_, _), (test_images, test_labels) = mnist.load_data()
test_images = test_images / 255.0
test_labels = to_categorical(test_labels)

test_loss, test_acc = model.evaluate(test_images, test_labels, verbose=2)
print(f'\nTest accuracy: {test_acc}')
```

```
313/313 - 1s - loss: 0.0268 - accuracy: 0.9925 - 974ms/epoch - 3ms/step

Test accuracy: 0.9925000071525574
```

Figure 2.18: model accuracy on Minst data set

the following figure displays the pre-trained model after changing the parameters to fit with out task where we changed the output from 10 to 28.



```
============================
Model: "model_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 28, 28, 1)]       0

 conv0 (Conv2D)              (None, 26, 26, 32)        320

 bn0 (BatchNormalization)    (None, 26, 26, 32)        128

 activation_1 (Activation)   (None, 26, 26, 32)        0

 conv1 (Conv2D)              (None, 24, 24, 32)        9248

 bn1 (BatchNormalization)    (None, 24, 24, 32)        128

 activation_2 (Activation)   (None, 24, 24, 32)        0

 MP1 (MaxPooling2D)          (None, 12, 12, 32)        0

 conv2 (Conv2D)              (None, 10, 10, 64)        18496

 bn2 (BatchNormalization)    (None, 10, 10, 64)        256

 activation_3 (Activation)   (None, 10, 10, 64)        0

 conv3 (Conv2D)              (None, 8, 8, 64)          36928

 bn3 (BatchNormalization)    (None, 8, 8, 64)          256

 activation_4 (Activation)   (None, 8, 8, 64)          0

 MP2 (MaxPooling2D)          (None, 4, 4, 64)          0

 dropout_1 (Dropout)         (None, 4, 4, 64)          0

 flatten_1 (Flatten)         (None, 1024)              0

 fc1 (Dense)                 (None, 256)               262400

 dropout_2 (Dropout)         (None, 256)               0

 output_layer (Dense)        (None, 28)                7196

=================================================================
Total params: 335356 (1.28 MB)
Trainable params: 334972 (1.28 MB)
Non-trainable params: 384 (1.50 KB)
_________________________________________________________________
```

Figure 2.19: Pre-trained tuned model description

The approached transfer learning method is Freezing Layers method , where all pre-trained layers were freezed so their weight are preserved and not exposed to the new task training data , just the new layer of the task get exposed to the training data.

Since the used data set (minst) for pre-training the model was digits numbers of size 28x28 means that we scaled down the data set images from 32x32 into 28x28 to fit in the model.

The model was exposed to the the standard data set , and augmented data set.

2.7.1 Standard data set training

using standard data set (13440 train images) , the pre-trained model scored : **93.87%**

```
Epoch 13/32
...
420/420 [==============================] - 2s 6ms/step - loss: 0.0599 - accuracy: 0.9803 - val_loss: 0.2906 - val_accuracy: 0.9387
105/105 [==============================] - 0s 2ms/step - loss: 0.2906 - accuracy: 0.9387
Test Loss: 0.2906
Test Accuracy: 0.9387
```

Figure 2.20: Pre-trained model scoring

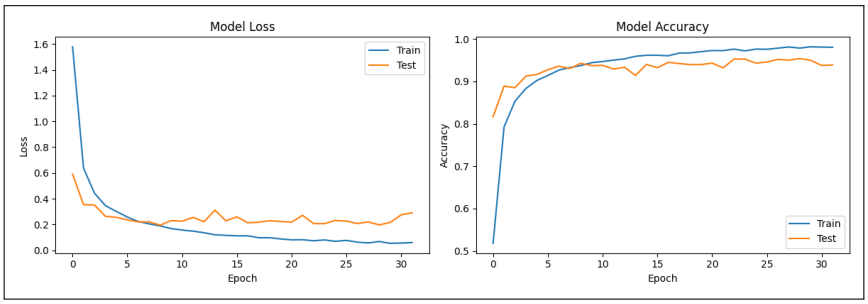


Figure 2.21: Pre-trained model training/testing result plot

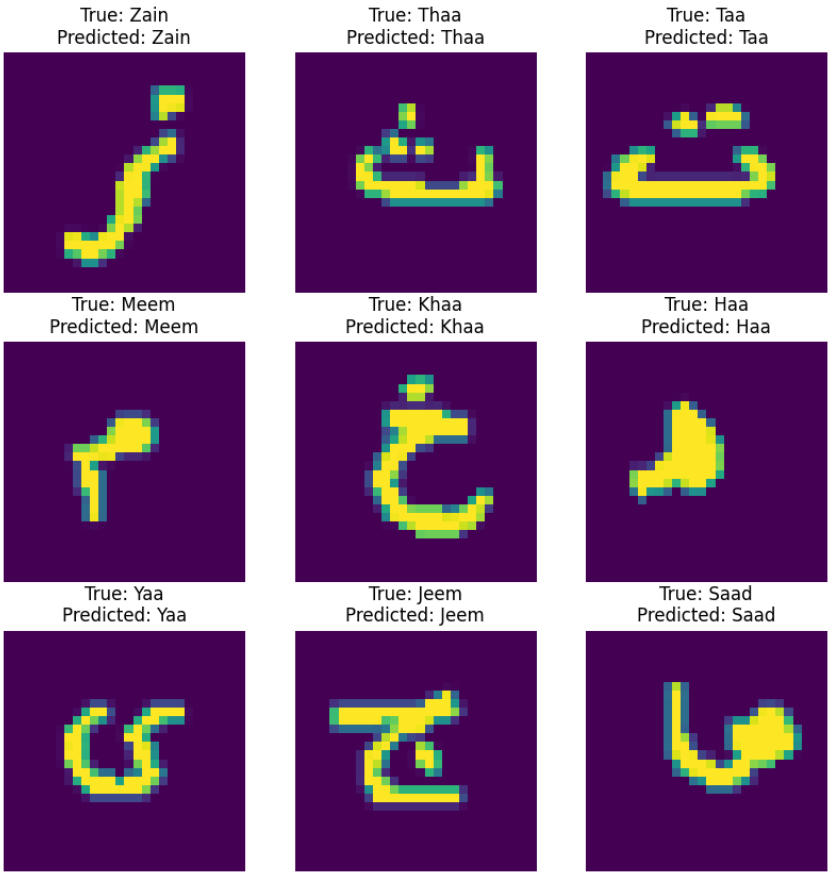


Figure 2.22: Pre-trained model sample test

2.7.2 Augmented data set training

using augmented data set (40,320 train images) with 3 data augmentation techniques (Rotation , shift , zoom) .
The model scored : **96.61%** which is 2.74 % improvement.

```
..
525/525 [==============================
105/105 [==============================
est Loss: 0.1936
est Accuracy: 0.9661
```

Figure 2.23: Pre-trained model scoring result

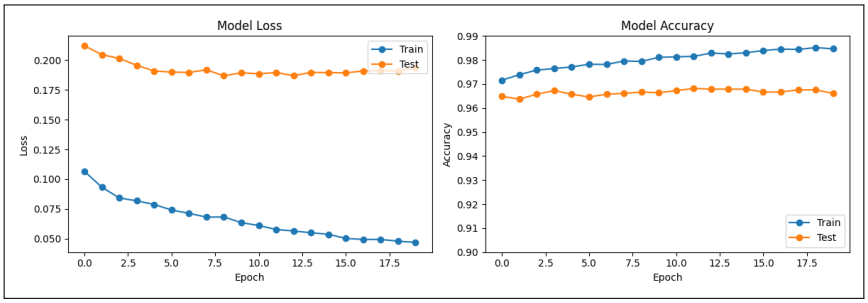


Figure 2.24: Pre-trained model training/testing plot

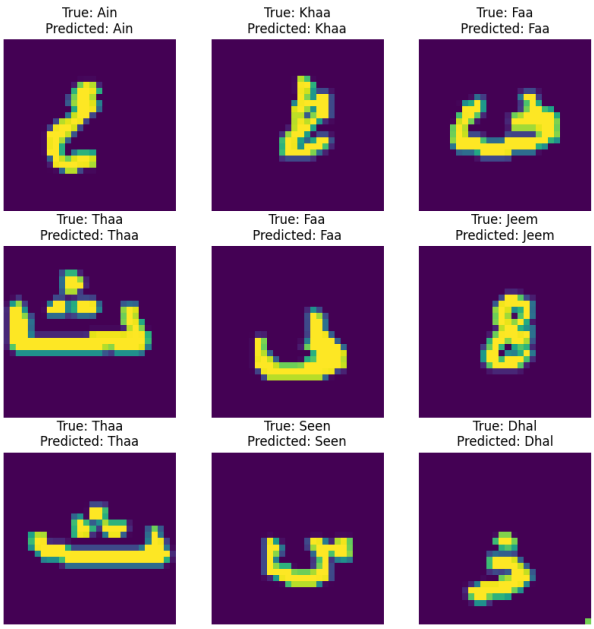


Figure 2.25: Pre-trained model sample test

Pre-trained model showed a swift training period comparing to the previous tasks , where there were an improvements in performance in training time , and efficiency in Resources.

3 Conclusion

In conclusion , Arabic handwriting character recognition showed a huge insights into CNN models in computer vision , and how to build the architectures of the kind of network , and tuning their parameters to fit based on the task requirements , thought the project , we successfully built a custom CNN model as a baseline with a help of research papers that conducted similar tasks , and trained/tested the model on a collected data set , and applied number of data augmentation techniques and analyzed the improvements on the model , also surfaced number of well-known published CNN models and tested them on our data set , and we applied transfer learning on pre-trained CNN models and analyzed the benefits comparing with other tasks.

Based on the analysis of the tasks thought the project , we found that the base line gave an initial score of 93% and how it improved by using data augmentation making the score increase to 96% , also we noticed how some published CNN models are better in term of training time like Alex-net , and some other models takes a tremendous amount of training time like res-net 50 , and Finlay we realised how transfer-learning on pre-trained models can save a huge amount of time and efforts on different tasks.

References

[1] A. El-Sawy, M. Loey, and H. El-Bakry, "Arabic handwritten characters recognition using convolutional neural network," *WSEAS Transactions on Computer Research*, vol. 5, no. 1, pp. 11–19, 2017.

[2] K. Kunal, "Mnist handwritten digit classification using keras," https://github.com/kj7kunal/MNIST-Keras, 2019.