BIRZEIT UNIVERSITY

COMPUTER VISION ENCS5343.

COURSE PROJECT.

PREPARED BY:

SALEH KHATIB – 1200991.

SECTION 2.

DR: AZIZ QAROUSH.

DATE: 31/1/2024.

Table of Contents

Table of figures 2
Introduction 3
Arabic handwritten character recognition (AHCR) 3
Convolutional Neural Network (CNN) 3
Experimental setup and results 5
Evaluation methodology 5
Experiment and Results 6
Task 1: Custom CNN 7
Task 2: Custom CNN with data augmentation 10
Task 3: Well-Known CNN 12
.Task 4: Transfer learning for pre-trained 15
Conclusion 18
References 20

Table of figures

Figure 1: convolutional neural networks architecture (example). 4
Figure 2: some of training images. 5
Figure 3: Custom CNN Summary. 7
Figure 4: training and testing results for custom CNN. 8
Figure 5: evaluation figures for custom CNN. 9
Figure 6: data augmentation. 10
Figure 7: training and testing results for custom CNN with data augmentation. 10
Figure 8: evaluation figures for custom CNN with data augmentation. 11
Figure 9: Comparing Well-Known Archs. 12
Figure 10: Well-known CNN Summary. 13
Figure 11: training and testing results for ResNet-50 CNN. 13
Figure 12: evaluation figures for ResNet-50 CNN with data augmentation. 14
Figure 13: pre-trained CNN Summary. 15
Figure 14: training and testing results for pre-trained CNN after TL. 16
Figure 15: evaluation figures for pre-trained CNN with data augmentation. 17

Introduction

Arabic handwritten character recognition (AHCR)

Whether handwritten or printed, optical character recognition for English texts is arguably one of the most important areas of research. Excellent results have been obtained in the English text; however, the Arabic text lacks this kind of research. This is a result of the Arabic alphabet's structure and the variety of ways one letter can appear. Arabic handwritten character recognition (AHCR) systems involve a number of problems and difficulties, ranging from the phase of identifying an appropriate and publicly available Arabic handwritten text dataset to the phases of recognition and classification, which include feature extraction and segmentation.

Convolutional Neural Network (CNN)

One type of neural network that is frequently used for tasks involving object recognition and image classification is a convolutional neural network (CNN). CNNs outperform other neural networks when it comes to processing inputs such as speech, image, or audio signals. There are three basic classes of layers: fully connected layer (FC), pooling layer, and convolutional layer. The first layer of a convolutional network is called the convolutional layer. The fully connected layer is the last layer, although convolutional layers can be followed by pooling or other convolutional layers. The CNN becomes more complex with each layer, as it learns a larger area of the image. The previous layers emphasize key elements like edges and colors.

The image data begins to identify the larger components or shapes of the object as it moves through the CNN layers, eventually recognizing the desired object.

But how CNN works?

Convolutional neural networks perform better with image, speech, or audio signal inputs than other types of neural networks. There are three primary categories of layers in them:

- Convolutional layer.
- Pooling layer.
- Fully-connected (FC) layer.

A convolutional network's first layer is called the convolutional layer. The fully-connected layer is the last layer, even though convolutional layers can be followed by pooling layers or other convolutional layers. The CNN becomes more complex with each layer, recognizing a larger area of the image. Previous layers emphasize basic elements like edges and colors. The image data begins to identify larger components or shapes of the object as it moves through the layers of the CNN, and eventually it recognizes the desired object.

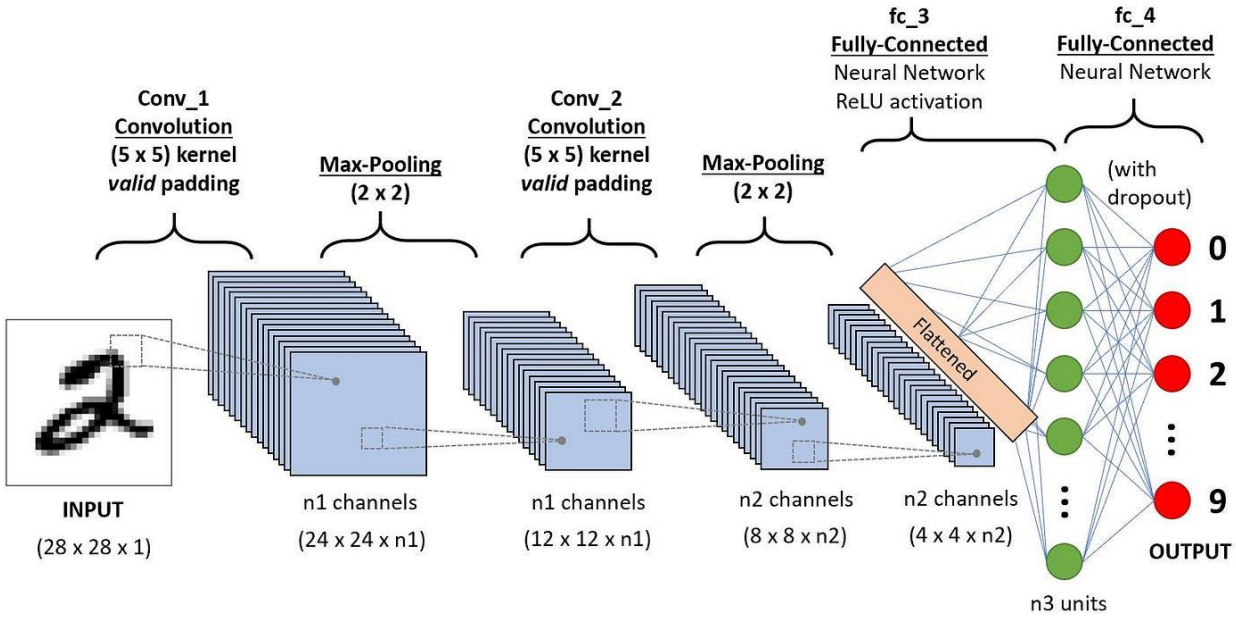

Figure 1: convolutional neural networks architecture (example).

Experimental setup and results

Evaluation methodology

As we know, our project is to build an Arabic handwritten character recognition (AHCR) model by CNN, so first of all let's talk about the Dataset.

I used the dataset provided by the project, which is a dataset contain of 13440 training images, and 3360 testing images, which are 32x32 grayscale images, in addition to their labels (28 label), for example:

Figure 2: some of training images.

As for evaluation methods, as we mentioned previously, this is a classification problem, which is the main metric is Accuracy.

But for CNN it's different, Because the weights are updated every epoch, so we measure the how good our model by these four figures:

1. Training loss vs. epoch.
2. Validation loss vs. epoch.
3. Training accuracy vs. epoch.
4. Testing accuracy vs. epoch.

For the experimental setup, it will be like this:

1- Load the dataset from csv files and reshape it.
2- Build our custom CNN based on some papers.
3- Train our CNN.
4- Form a suitable data augmentation.
5- Train our CNN with data augmentation.
6- Load a well-known CNN (ResNet 50) and train it with data augmentation.
7- Load a pre-trained which used for Devanagari Hand Written Character Recognition and perform transfer learning method to fine tune the pretrained network.

Experiment and Results

We will have four experiments, which are:

1- Custom CNN.
2- Custom CNN with data augmentation.
3- Well-Known CNN with data augmentation
4- Transfer learning for pre-trained.

Let’s start with the first one:

Task 1: Custom CNN

Returning to this research (Alsayed, Alhag & Li, Chunlin & Ahamed, & Hazim, Mohammed & Obied, Zainab. (2023).) and by doing some experiments, we get this CNN:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d1 (Conv2D)            (None, 32, 32, 64)        640

 conv2d2 (Conv2D)            (None, 32, 32, 64)        36928

 maxpooling2d1 (MaxPooling2  (None, 16, 16, 64)        0
 D)

 conv2d3 (Conv2D)            (None, 16, 16, 64)        36928

 conv2d4 (Conv2D)            (None, 16, 16, 64)        36928

 maxpooling2d2 (MaxPooling2  (None, 8, 8, 64)          0
 D)

 conv2d5 (Conv2D)            (None, 8, 8, 64)          36928

 conv2d6 (Conv2D)            (None, 8, 8, 64)          36928

 maxpooling2d3 (MaxPooling2  (None, 4, 4, 64)          0
 D)

 dropout1 (Dropout)          (None, 4, 4, 64)          0

 conv2d7 (Conv2D)            (None, 4, 4, 64)          36928

 conv2d8 (Conv2D)            (None, 4, 4, 64)          36928

 maxpooling2d4 (MaxPooling2  (None, 2, 2, 64)          0
 D)

 dropout2 (Dropout)          (None, 2, 2, 64)          0

 flatten1 (Flatten)          (None, 256)               0

 dense1 (Dense)              (None, 32)                8224

 dense2 (Dense)              (None, 64)                2112

 dropout3 (Dropout)          (None, 64)                0

 dense3 (Dense)              (None, 28)                1820

=================================================================
Total params: 271292 (1.03 MB)
Trainable params: 271292 (1.03 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Figure 3: Custom CNN Summary.

As we see, our CNN consisting of 19 layer, 8 convolutional layers with 64 filters, each filter with size 3x3 and the activation function is relu. 4 max pooling layers, with 2x2 pooling window. 3 Dropout layers to prevent overfitting. 1 flatten layer, and fully connected network consist of 3 layers with 32, 64, 28 neurons respectively, and the activation function is relu. So, the input is image with size 32x32 grayscale, and the output is 28 labels.

Now let's test our CNN:

```
336/336 [==============================] - 3s 9ms/step - loss: 0.2589 - accuracy: 0.9341 - val_loss: 0.2146 - val_accuracy: 0.9528
Epoch 23/50
336/336 [==============================] - 3s 10ms/step - loss: 0.2259 - accuracy: 0.9424 - val_loss: 0.2146 - val_accuracy: 0.9568
Epoch 24/50
336/336 [==============================] - 3s 9ms/step - loss: 0.2110 - accuracy: 0.9434 - val_loss: 0.2039 - val_accuracy: 0.9594
Epoch 25/50
336/336 [==============================] - 3s 9ms/step - loss: 0.1941 - accuracy: 0.9488 - val_loss: 0.2783 - val_accuracy: 0.9505
Epoch 26/50
336/336 [==============================] - 3s 9ms/step - loss: 0.2093 - accuracy: 0.9458 - val_loss: 0.1590 - val_accuracy: 0.9665
Epoch 27/50
336/336 [==============================] - 3s 10ms/step - loss: 0.1674 - accuracy: 0.9553 - val_loss: 0.1545 - val_accuracy: 0.9688
Epoch 28/50
336/336 [==============================] - 3s 9ms/step - loss: 0.2261 - accuracy: 0.9435 - val_loss: 0.2121 - val_accuracy: 0.9594
Epoch 29/50
336/336 [==============================] - 3s 9ms/step - loss: 0.1856 - accuracy: 0.9500 - val_loss: 0.2001 - val_accuracy: 0.9535
Epoch 30/50
336/336 [==============================] - 3s 10ms/step - loss: 0.2054 - accuracy: 0.9502 - val_loss: 0.1779 - val_accuracy: 0.9598
Epoch 31/50
336/336 [==============================] - 3s 10ms/step - loss: 0.1538 - accuracy: 0.9602 - val_loss: 0.2148 - val_accuracy: 0.9583
Epoch 32/50
336/336 [==============================] - 3s 10ms/step - loss: 0.1804 - accuracy: 0.9561 - val_loss: 0.2471 - val_accuracy: 0.9528
Epoch 33/50
336/336 [==============================] - 3s 10ms/step - loss: 0.1889 - accuracy: 0.9507 - val_loss: 0.1899 - val_accuracy: 0.9661
Epoch 34/50
336/336 [==============================] - 4s 10ms/step - loss: 0.1940 - accuracy: 0.9517 - val_loss: 0.2057 - val_accuracy: 0.9606
Epoch 35/50
336/336 [==============================] - 3s 9ms/step - loss: 0.1618 - accuracy: 0.9560 - val_loss: 0.2602 - val_accuracy: 0.9554
Epoch 36/50
336/336 [==============================] - 4s 11ms/step - loss: 0.2022 - accuracy: 0.9511 - val_loss: 0.2196 - val_accuracy: 0.9568
Epoch 37/50
336/336 [==============================] - 3s 10ms/step - loss: 0.1979 - accuracy: 0.9506 - val_loss: 0.1578 - val_accuracy: 0.9680
Epoch 38/50
336/336 [==============================] - 3s 10ms/step - loss: 0.1698 - accuracy: 0.9573 - val_loss: 0.1752 - val_accuracy: 0.9691
Epoch 39/50
336/336 [==============================] - 4s 10ms/step - loss: 0.1588 - accuracy: 0.9599 - val_loss: 0.1856 - val_accuracy: 0.9702
Epoch 40/50
336/336 [==============================] - 3s 9ms/step - loss: 0.1951 - accuracy: 0.9503 - val_loss: 0.1885 - val_accuracy: 0.9613
Epoch 41/50
336/336 [==============================] - 3s 10ms/step - loss: 0.1593 - accuracy: 0.9577 - val_loss: 0.1678 - val_accuracy: 0.9706
Epoch 42/50
336/336 [==============================] - 3s 9ms/step - loss: 0.1714 - accuracy: 0.9559 - val_loss: 0.1668 - val_accuracy: 0.9669
Epoch 43/50
336/336 [==============================] - 3s 9ms/step - loss: 0.1628 - accuracy: 0.9620 - val_loss: 0.3885 - val_accuracy: 0.9338
Epoch 44/50
336/336 [==============================] - 3s 9ms/step - loss: 0.1506 - accuracy: 0.9621 - val_loss: 0.2792 - val_accuracy: 0.9565
Epoch 45/50
336/336 [==============================] - 3s 10ms/step - loss: 0.1736 - accuracy: 0.9587 - val_loss: 0.1729 - val_accuracy: 0.9676
Epoch 46/50
336/336 [==============================] - 3s 9ms/step - loss: 0.1572 - accuracy: 0.9609 - val_loss: 0.2405 - val_accuracy: 0.9669
Epoch 47/50
336/336 [==============================] - 3s 9ms/step - loss: 0.1649 - accuracy: 0.9581 - val_loss: 0.2201 - val_accuracy: 0.9647
Epoch 48/50
336/336 [==============================] - 3s 10ms/step - loss: 0.1582 - accuracy: 0.9603 - val_loss: 0.2190 - val_accuracy: 0.9650
Epoch 49/50
336/336 [==============================] - 3s 10ms/step - loss: 0.1921 - accuracy: 0.9550 - val_loss: 0.2506 - val_accuracy: 0.9632
Epoch 50/50
336/336 [==============================] - 3s 9ms/step - loss: 0.1673 - accuracy: 0.9622 - val_loss: 0.2075 - val_accuracy: 0.9628
105/105 [==============================] - 1s 5ms/step - loss: 0.2172 - accuracy: 0.9571
Test accuracy: 0.9571300745010376
```

Figure 4: training and testing results for custom CNN.

As we see, I train the network for 50 epochs, and get testing accuracy equal 95.7% (sometimes may be 96%, due to the training an validation sets division) which is a very good accuracy.

Now let's see the figures:

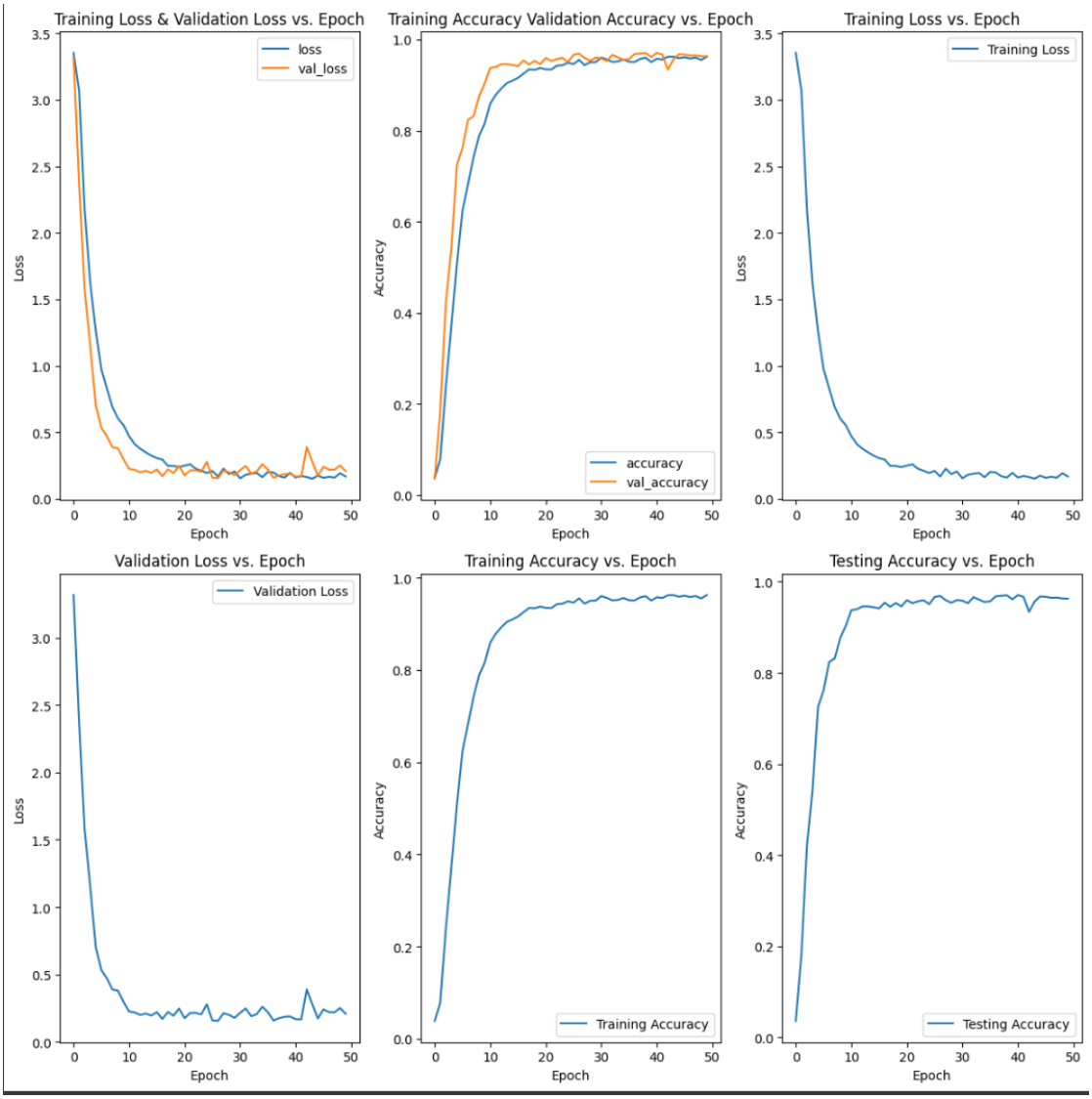

Figure 5: evaluation figures for custom CNN.

As can be seen, it peaked at 20 epochs and then barely increased after that. Furthermore, we observe that the graphs have little noise and comparable figures.

Task 2: Custom CNN with data augmentation

In this task we will train our CNN but with this data augmentation:

```
datagen = ImageDataGenerator(
rotation_range=10,
    width_shift_range=0.1,
    height_shift_range=0.1,
    shear_range=0.1,
    zoom_range=0.1,
    horizontal_flip=False,
    vertical_flip=False,
    fill_mode='wrap'
)
```

Figure 6: data augmentation.

As we see, we do rotation in range -10 to 10, same for width shift, height shift, shear, zoom, all in range -0.1 to 0.1, and fill_mode will be wrap.

Why we do this? To increase the diversity of the training dataset by applying various transformations to the existing images.

Now let's see the results:

```
Epoch 42/50
336/336 [==============================] - 7s 21ms/step - loss: 0.2378 - accuracy: 0.9402 - val_loss: 0.1921 - val_accuracy: 0.9516
Epoch 43/50
336/336 [==============================] - 7s 21ms/step - loss: 0.2274 - accuracy: 0.9445 - val_loss: 0.1714 - val_accuracy: 0.9673
Epoch 44/50
336/336 [==============================] - 6s 19ms/step - loss: 0.2195 - accuracy: 0.9445 - val_loss: 0.1776 - val_accuracy: 0.9647
Epoch 45/50
336/336 [==============================] - 7s 21ms/step - loss: 0.2405 - accuracy: 0.9371 - val_loss: 0.2385 - val_accuracy: 0.9472
Epoch 46/50
336/336 [==============================] - 9s 26ms/step - loss: 0.2518 - accuracy: 0.9382 - val_loss: 0.2490 - val_accuracy: 0.9591
Epoch 47/50
336/336 [==============================] - 6s 19ms/step - loss: 0.2667 - accuracy: 0.9374 - val_loss: 0.1648 - val_accuracy: 0.9661
Epoch 48/50
336/336 [==============================] - 7s 21ms/step - loss: 0.2405 - accuracy: 0.9418 - val_loss: 0.1650 - val_accuracy: 0.9639
Epoch 49/50
336/336 [==============================] - 7s 22ms/step - loss: 0.2172 - accuracy: 0.9487 - val_loss: 0.1914 - val_accuracy: 0.9587
Epoch 50/50
336/336 [==============================] - 7s 20ms/step - loss: 0.2058 - accuracy: 0.9492 - val_loss: 0.1700 - val_accuracy: 0.9673
105/105 [==============================] - 1s 13ms/step - loss: 0.2039 - accuracy: 0.9643
Test accuracy: 0.964275062084198
```

Figure 7: training and testing results for custom CNN with data augmentation.

As we see, it has a little bit improvement from task 1, with test accuracy equal 96.42%, but the disadvantage is it take more time than task 1.

Now let’s see the figures:

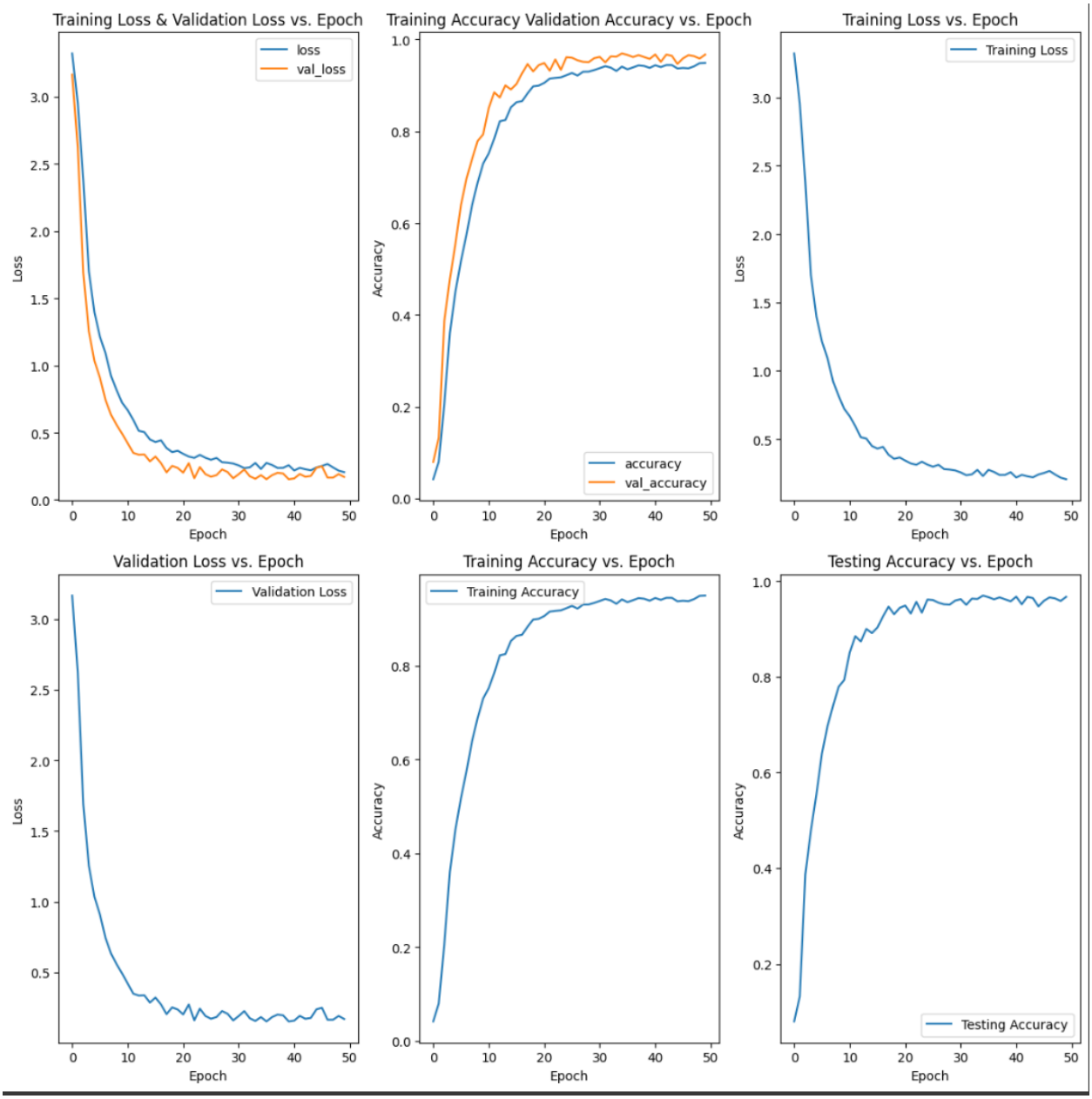

Figure 8: evaluation figures for custom CNN with data augmentation.

As can be seen, it peaked at 22 epochs and then barely increased after that. Furthermore, we observe that the graphs have little noise more than task 1, but here the validation and testing graphs are barely better than training.

Task 3: Well-Known CNN

In this task we will import the ResNet-50 CNN, due this:

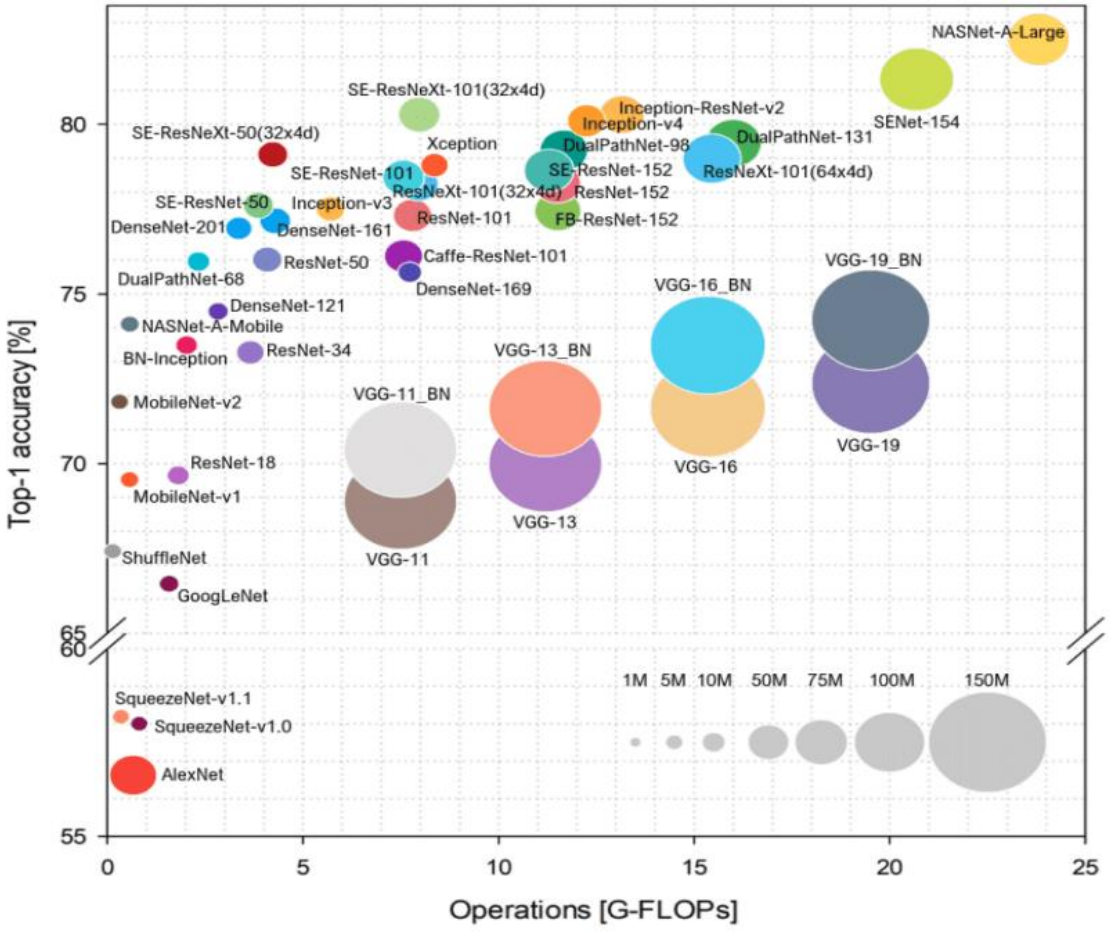

Figure 9: Comparing Well-Known Archs.

As we note, the ResNet-50 have good accuracy, above 75%, it takes a little memory size and little number of parameters, which in terms of trade of is very good.

But before we use it, we will convert the images to RGB because ResNet-50 takes only RGB (3 channels).

Now we will see it summary:

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 resnet50 (Functional)       (None, 1, 1, 2048)        23587712

 flatten (Flatten)           (None, 2048)              0

 dense (Dense)               (None, 256)               524544

 dropout (Dropout)           (None, 256)               0

 dense_1 (Dense)             (None, 28)                7196

=================================================================
Total params: 24119452 (92.01 MB)
Trainable params: 24066332 (91.81 MB)
Non-trainable params: 53120 (207.50 KB)
```

Figure 10: Well-known CNN Summary.

as we see it has a lot of parameters compere to our CNN.

Now let's see the results:

```
Epoch 41/50
336/336 [==============================] - 19s 57ms/step - loss: 0.8215 - accuracy: 0.7591 - val_loss: 0.6647 - val_accuracy: 0.7902
Epoch 42/50
336/336 [==============================] - 19s 56ms/step - loss: 0.6969 - accuracy: 0.7844 - val_loss: 2.1966 - val_accuracy: 0.5324
Epoch 43/50
336/336 [==============================] - 21s 62ms/step - loss: 0.7169 - accuracy: 0.7726 - val_loss: 0.8123 - val_accuracy: 0.7578
Epoch 44/50
336/336 [==============================] - 20s 59ms/step - loss: 0.5307 - accuracy: 0.8344 - val_loss: 0.3964 - val_accuracy: 0.8813
Epoch 45/50
336/336 [==============================] - 19s 58ms/step - loss: 0.4881 - accuracy: 0.8504 - val_loss: 0.3454 - val_accuracy: 0.8914
Epoch 46/50
336/336 [==============================] - 19s 58ms/step - loss: 0.5855 - accuracy: 0.8256 - val_loss: 0.6350 - val_accuracy: 0.8251
Epoch 47/50
336/336 [==============================] - 19s 57ms/step - loss: 0.5836 - accuracy: 0.8311 - val_loss: 0.3165 - val_accuracy: 0.8988
Epoch 48/50
336/336 [==============================] - 20s 59ms/step - loss: 0.8076 - accuracy: 0.7503 - val_loss: 0.5770 - val_accuracy: 0.7965
Epoch 49/50
336/336 [==============================] - 20s 58ms/step - loss: 0.6093 - accuracy: 0.8092 - val_loss: 0.2656 - val_accuracy: 0.9156
Epoch 50/50
336/336 [==============================] - 19s 58ms/step - loss: 0.4780 - accuracy: 0.8566 - val_loss: 0.2751 - val_accuracy: 0.9193
105/105 [==============================] - 3s 11ms/step - loss: 0.2771 - accuracy: 0.9196
Test accuracy: 0.9196189045906067
```

Figure 11: training and testing results for ResNet-50 CNN.

As we see and contrary to expectations, the testing accuracy equal 91.96%, and take very long time, but why this?

May due to huge number of parameters compere to my CNN, and data augmentation, make it need more epochs to train it. And may due that ResNet-50 is not specific to my problem.

Now let’s see the figures:

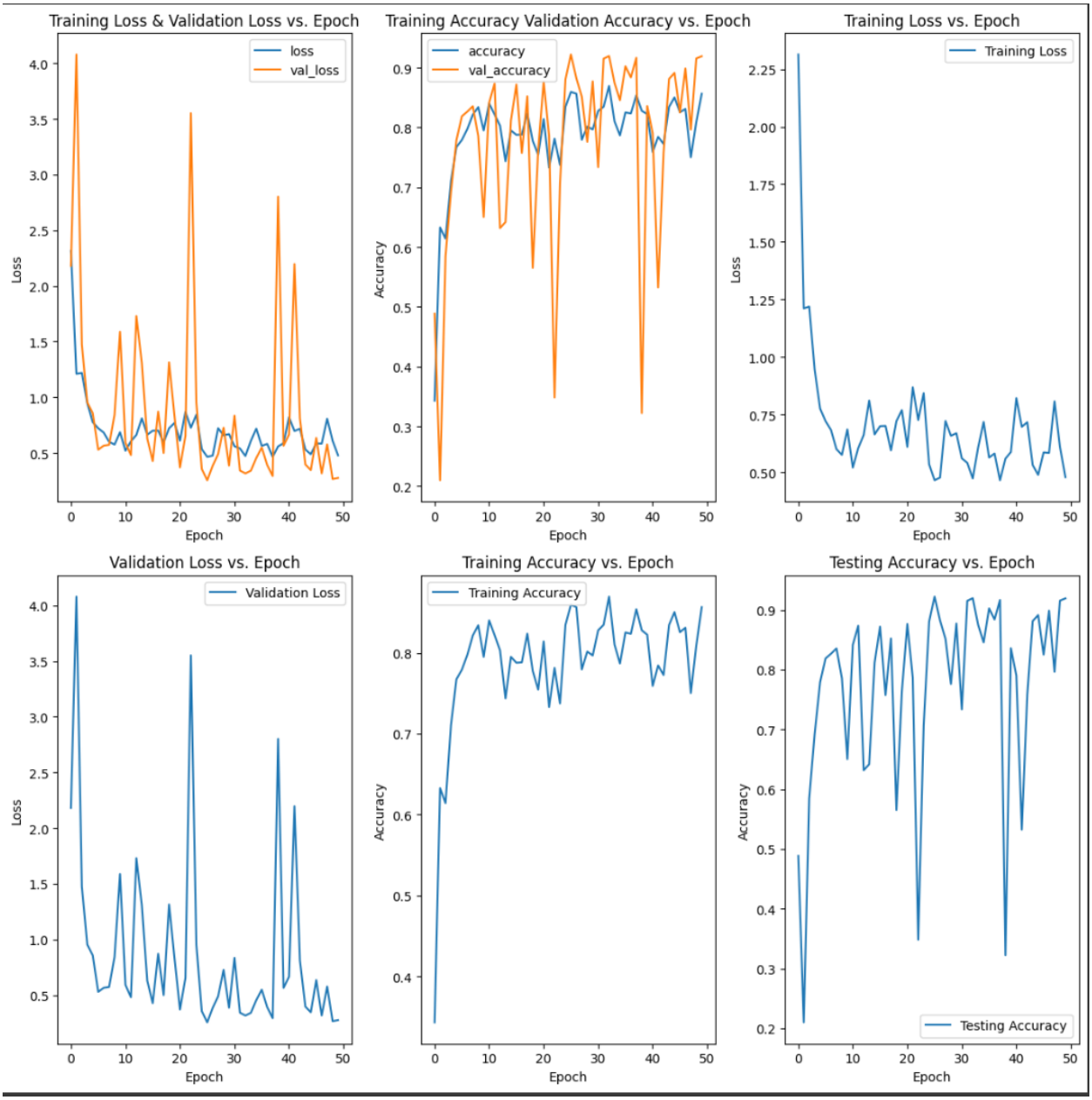

Figure 12: evaluation figures for ResNet-50 CNN with data augmentation.

As we see, it has a lot of noise, so we can’t decide the peak, but why?

Because the complexity of the CNN, it may struggle to generalize well to the validation set. Also the data augmentation make the noise worse.

Task 4: Transfer learning for pre-trained.

In this task we will do Transfer learning for Devanagari Hand Written Character Recognition, which is similar to my problem.

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_1 (Conv2D)           (None, 30, 30, 32)        896

 conv2d_2 (Conv2D)           (None, 28, 28, 64)        18496

 max_pooling2d_1 (MaxPoolin  (None, 14, 14, 64)        0
 g2D)

 conv2d_3 (Conv2D)           (None, 12, 12, 64)        36928

 conv2d_4 (Conv2D)           (None, 10, 10, 64)        36928

 max_pooling2d_2 (MaxPoolin  (None, 5, 5, 64)          0
 g2D)

 dropout_1 (Dropout)         (None, 5, 5, 64)          0

 flatten_1 (Flatten)         (None, 1600)              0

 dense_1 (Dense)             (None, 128)               204928

 dropout_2 (Dropout)         (None, 128)               0

 dense_2 (Dense)             (None, 46)                5934

=================================================================
Total params: 304110 (1.16 MB)
Trainable params: 304110 (1.16 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Figure 13: pre-trained CNN Summary.

As we see, pre-trained CNN consisting of 11 layer, 4 convolutional layers with 64 filters, except the first one 32 each filter with size 3x3 and the activation function is relu. 2 max pooling layers, with 2x2 pooling window. 2 Dropout layers to prevent overfitting. 1 flatten layer, and fully connected network consist of 2 layers with 128, 46 neurons respectively, and the activation function is relu. So, the input is image with size 32x32 RGB, and the output is 46 labels.

So I change the output to be 28 label.

Due that my dataset is not small and the data augmentation make it bigger, so I used Fine-Tuning the whole network.

Let's see our results:

```
Epoch 24/40
336/336 [==============================] - 9s 26ms/step - loss: 0.1452 - accuracy: 0.9593 - val_loss: 0.1148 - val_accuracy: 0.9743
Epoch 25/40
336/336 [==============================] - 9s 25ms/step - loss: 0.1400 - accuracy: 0.9597 - val_loss: 0.0993 - val_accuracy: 0.9743
Epoch 26/40
336/336 [==============================] - 9s 27ms/step - loss: 0.1291 - accuracy: 0.9647 - val_loss: 0.1121 - val_accuracy: 0.9732
Epoch 27/40
336/336 [==============================] - 10s 31ms/step - loss: 0.1419 - accuracy: 0.9594 - val_loss: 0.1109 - val_accuracy: 0.9762
Epoch 28/40
336/336 [==============================] - 9s 27ms/step - loss: 0.1424 - accuracy: 0.9607 - val_loss: 0.1225 - val_accuracy: 0.9673
Epoch 29/40
336/336 [==============================] - 8s 25ms/step - loss: 0.1311 - accuracy: 0.9617 - val_loss: 0.1353 - val_accuracy: 0.9676
Epoch 30/40
336/336 [==============================] - 8s 25ms/step - loss: 0.1402 - accuracy: 0.9609 - val_loss: 0.1318 - val_accuracy: 0.9684
Epoch 31/40
336/336 [==============================] - 9s 26ms/step - loss: 0.1351 - accuracy: 0.9629 - val_loss: 0.1274 - val_accuracy: 0.9699
Epoch 32/40
336/336 [==============================] - 9s 28ms/step - loss: 0.1211 - accuracy: 0.9675 - val_loss: 0.1109 - val_accuracy: 0.9728
Epoch 33/40
336/336 [==============================] - 9s 27ms/step - loss: 0.1139 - accuracy: 0.9661 - val_loss: 0.0947 - val_accuracy: 0.9743
Epoch 34/40
336/336 [==============================] - 10s 30ms/step - loss: 0.1325 - accuracy: 0.9639 - val_loss: 0.1321 - val_accuracy: 0.9673
Epoch 35/40
336/336 [==============================] - 9s 25ms/step - loss: 0.1185 - accuracy: 0.9670 - val_loss: 0.1120 - val_accuracy: 0.9721
Epoch 36/40
336/336 [==============================] - 9s 28ms/step - loss: 0.1314 - accuracy: 0.9643 - val_loss: 0.1187 - val_accuracy: 0.9717
Epoch 37/40
336/336 [==============================] - 9s 28ms/step - loss: 0.1213 - accuracy: 0.9665 - val_loss: 0.1108 - val_accuracy: 0.9740
Epoch 38/40
336/336 [==============================] - 9s 27ms/step - loss: 0.1237 - accuracy: 0.9667 - val_loss: 0.1005 - val_accuracy: 0.9754
Epoch 39/40
336/336 [==============================] - 9s 26ms/step - loss: 0.1083 - accuracy: 0.9711 - val_loss: 0.1234 - val_accuracy: 0.9699
Epoch 40/40
336/336 [==============================] - 9s 27ms/step - loss: 0.1075 - accuracy: 0.9705 - val_loss: 0.1048 - val_accuracy: 0.9751
105/105 [==============================] - 0s 4ms/step - loss: 0.1160 - accuracy: 0.9738
Final Test accuracy: 0.9738017320632935
Model: "sequential_2"
```

Figure 14: training and testing results for pre-trained CNN after TL.

As we see, is has the best testing accuracy equal 97.38% which is a good improvement, and just take 40 epochs to Fine-Tuning the whole network with time similar to task 2.

That's indicate that we choose a good CNN and good Transfer learning method.

Now let’s see the figures:

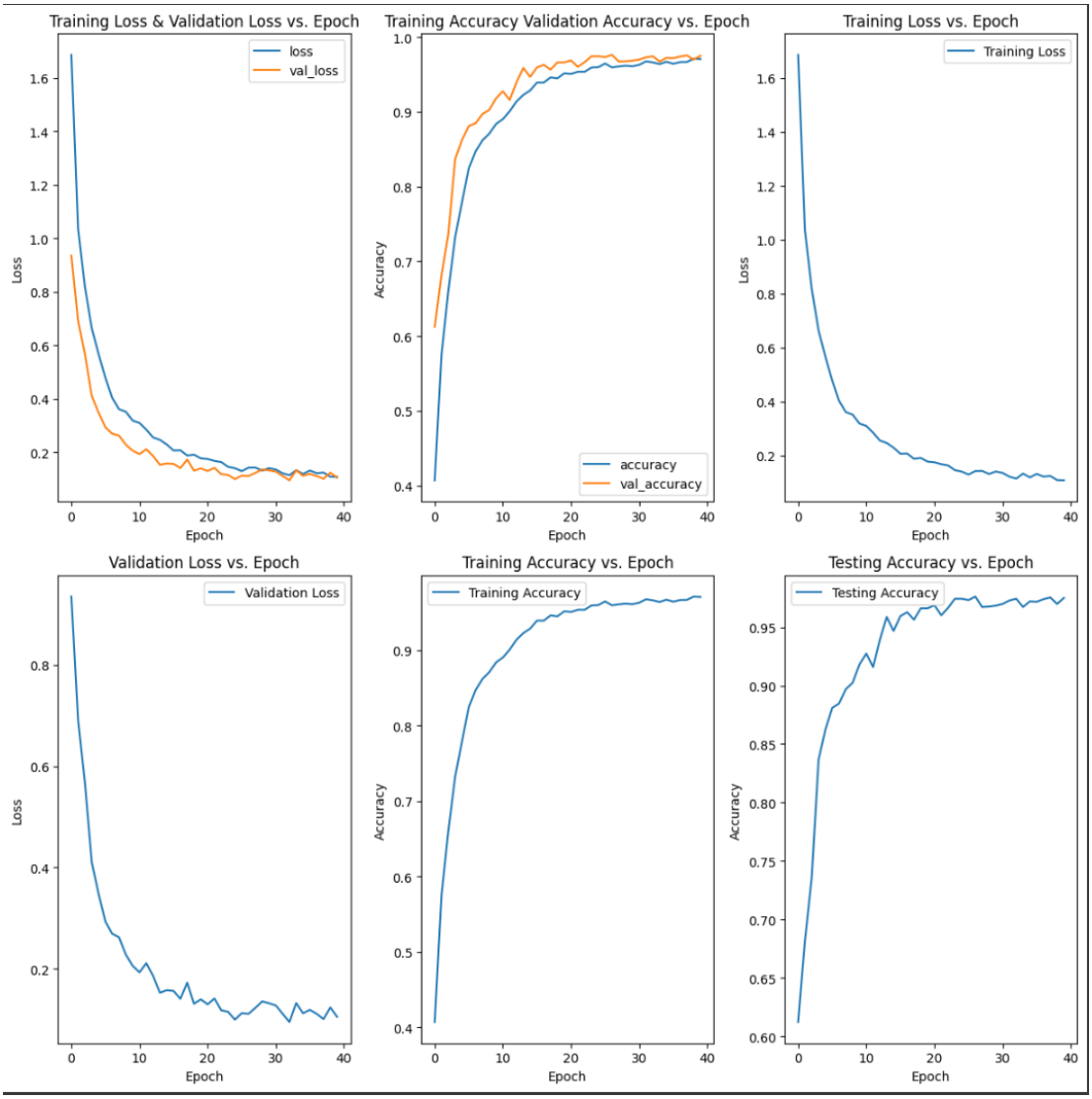

Figure 15: evaluation figures for pre-trained CNN with data augmentation.

As we see, it similar to task 2, with a more noise due the number of parameters.

So, data augmentation increases the accuracy, but it increases time and noise. For using published CNN networks, it may increase the time and noise due its complexity and need more epochs. Finally, the best thing to do is find a similar pre-trained CNN and perform Transfer learning, so you will get less time, epochs, and better accuracy.

Conclusion

To conclude, the project used Convolutional Neural Networks (CNNs) to address the issue of Arabic Handwritten Character Recognition (AHCR). The research has shown that the structure of the Arabic alphabet and its different looks for each character bring up unique challenges, calling for specific solutions in this sphere.

Four major steps were performed during an experimental stage; which include developing a custom CNN architecture; augmenting data in order to make it more diverse; employing ResNet-50 – a well-known CNN and performing transfer learning using a pre-trained network. There were some common trends observed in these tasks.

The developed custom CNN showed excellent results with a test accuracy of 95.7% at 50 epochs. Further experimentation with data augmentation brought about a small improvement with testing accuracy of 96.42%, however, training time was increased and there was additional noise in the results.

Integration of ResNet-50, a popularly known CNN proved challenging. With its inherent intricacy and considerable number of parameters, this model still achieved an accuracy rate during testing equal to 91.96%. Nevertheless, long-lasting training process together with much noisy outcomes suggests that generalization might be difficult.

With a pre-trained network customized for an analogous task, transfer learning yielded the most promising results. This method performed

better than the others in terms of testing accuracy, training efficiency, and noise reduction, with a testing accuracy of 97.38% after 40 epochs.

In conclusion, the project emphasizes how crucial it is to take the complexities of the problem domain into account when choosing and creating CNN architectures. The effectiveness of transfer learning highlights the possible advantages of using pre-trained models, providing a promising path toward effective and precise AHCR models.

References

Convolutional neural network - Wikipedia

What are Convolutional Neural Networks? | IBM

Pre trained CNN: Devanagari-Handwritten-Character-Recognition-CNN/CNN_DevanagariHandWrittenCharacterRecognition.ipynb at master · hariperisetla/Devanagari-Handwritten-Character-Recognition-CNN (github.com)