Data Mining Course Project

Grapevine Leaves Classification

Zahra Rabbany
610398124

Prof. Hedieh Sajedi
Department of Computer Science, Tehran University
July 2022

[Link to Google Colaboratory]

1. Loading The Dataset and Pre-processing

1.1. Load the Dataset

Attribute Information:

This dataset is a collection of png pictures of five class of grapevine leaves. Namely **Ak**, **Ala Idris**, **Nazli**, **Buzgulu**, and **Dimnit**.

We first mount the google colab to google drive and use the dataset uploaded there.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   images  500 non-null    object
 1   label   500 non-null    object
 2   path    500 non-null    object
dtypes: object(3)
memory usage: 11.8+ KB
```

We store the dataset in a dataframe where each row is consist of the label and the path to the picture.

	images	label	path
0	Ak (51).png	Ak	/content/Grapevine_Leaves_Image_Dataset/Ak/Ak ...
1	Ak (21).png	Ak	/content/Grapevine_Leaves_Image_Dataset/Ak/Ak ...
2	Ak (4).png	Ak	/content/Grapevine_Leaves_Image_Dataset/Ak/Ak ...
3	Ak (10).png	Ak	/content/Grapevine_Leaves_Image_Dataset/Ak/Ak ...
4	Ak (69).png	Ak	/content/Grapevine_Leaves_Image_Dataset/Ak/Ak ...

```
Number of Leaves Classes:  5
--------------------
Number of Leaves in Each Class:
Ak           100
Ala_Idris    100
Buzgulu      100
Dimnit       100
Nazli        100
Name: label, dtype: int64
--------------------
```

As we can see the dataset is completely balanced and we have 100 pictures for each class of leaves.

Below are some instances of our dataset.

1.2. Split Train, Test, and Validation Data

We split our dataset into test and train sets using *train_ test_ split* of scikit-learn.

```
Num train images: 320
Num validation images: 80
Num test images: 100
Num classes: 5
Num iterations per epoch: 10
```

1.3. Data Augmentation

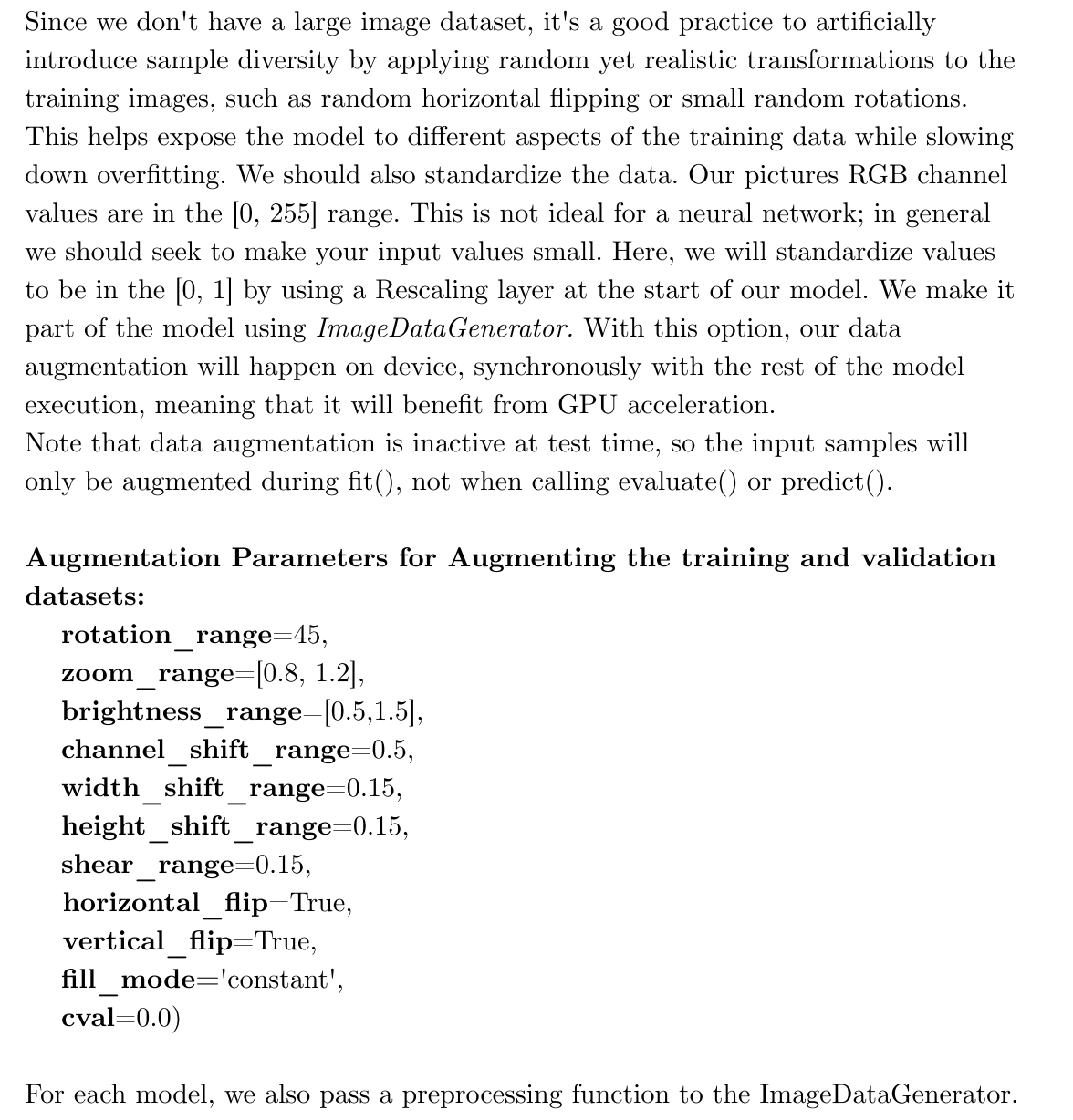
Since we don't have a large image dataset, it's a good practice to artificially introduce sample diversity by applying random yet realistic transformations to the training images, such as random horizontal flipping or small random rotations. This helps expose the model to different aspects of the training data while slowing down overfitting. We should also standardize the data. Our pictures RGB channel values are in the [0, 255] range. This is not ideal for a neural network; in general we should seek to make your input values small. Here, we will standardize values to be in the [0, 1] by using a Rescaling layer at the start of our model. We make it part of the model using *ImageDataGenerator.* With this option, our data augmentation will happen on device, synchronously with the rest of the model execution, meaning that it will benefit from GPU acceleration.
Note that data augmentation is inactive at test time, so the input samples will only be augmented during fit(), not when calling evaluate() or predict().

Augmentation Parameters for Augmenting the training and validation datasets:

rotation_range=45,
zoom_range=[0.8, 1.2],
brightness_range=[0.5,1.5],
channel_shift_range=0.5,
width_shift_range=0.15,
height_shift_range=0.15,
shear_range=0.15,
horizontal_flip=True,
vertical_flip=True,
fill_mode='constant',
cval=0.0)

For each model, we also pass a preprocessing function to the ImageDataGenerator. This function would perform the appropriate scaling on the data. We call *flow_from_ dataframe* on the generators to get and argument the data.
Note that we won't augment the test dataset but we still use the preprocessing function on them.

Using augmentation we get:



The left-most column is the real data and the others are the augmented.

2. Pre-trained Convolutional Neural Network -Transfer Learning-

2.1. Introduction

Transfer learning is a technique in machine learning where we can take a model developed on one task and use it as a starting point in some other similar but different tasks. Transfer learning is extremely popular in Deep Learning since the "transfer" of knowledge from one "parent" model to a "child" model means that the "child" model can be trained to high accuracies with a much smaller dataset compared to the "parent" model.

Most Image Classification Deep Learning tasks today will start by downloading one of these 18 pre-trained models, modify the model slightly to suit the task on hand, and train only the custom modifications while freezing the layers in the pre-trained model. This approach gives very high accuracy on real-world image classification tasks since "ImageNet" collects many real-world images.

However, choosing which of the 18 pre-trained models to use is not always an exact science. Many developers stick to the models they are familiar with and gave them good results in the past.

Pre-trained model available on *Keras:*

```
DenseNet121
DenseNet169
DenseNet201
EfficientNetB0
EfficientNetB1
EfficientNetB2
EfficientNetB3
EfficientNetB4
EfficientNetB5
EfficientNetB6
EfficientNetB7
EfficientNetV2B0
EfficientNetV2B1
EfficientNetV2B2
EfficientNetV2B3
EfficientNetV2L
EfficientNetV2M
EfficientNetV2S
InceptionResNetV2
InceptionV3
MobileNet
MobileNetV2
MobileNetV3Large
MobileNetV3Small
NASNetLarge
NASNetMobile
ResNet101
ResNet101V2
ResNet152
ResNet152V2
ResNet50
ResNet50V2
VGG16
VGG19
Xception
```

In general, there are two competing criteria while doing any machine learning task in the industry:

1. **Accuracy** of the model: Higher Better
2. **Speed** of Model Training and Predictions: Faster Better

The two criteria are pretty straightforward. We want the model that gives us the highest accuracy on validation data since it can make useful predictions. We also want the model to train and predict as fast as possible because, in production, we might need to serve hundreds or thousands of predictions every second.

Unfortunately, to get higher accuracy, we need to use a "deeper" or larger model. But a larger model has many more parameters that make it slower to execute. So there is a **tradeoff** between the accuracy and size/number of operations of the model.

Training on the *cats_ vs_ dogs* dataset we have:

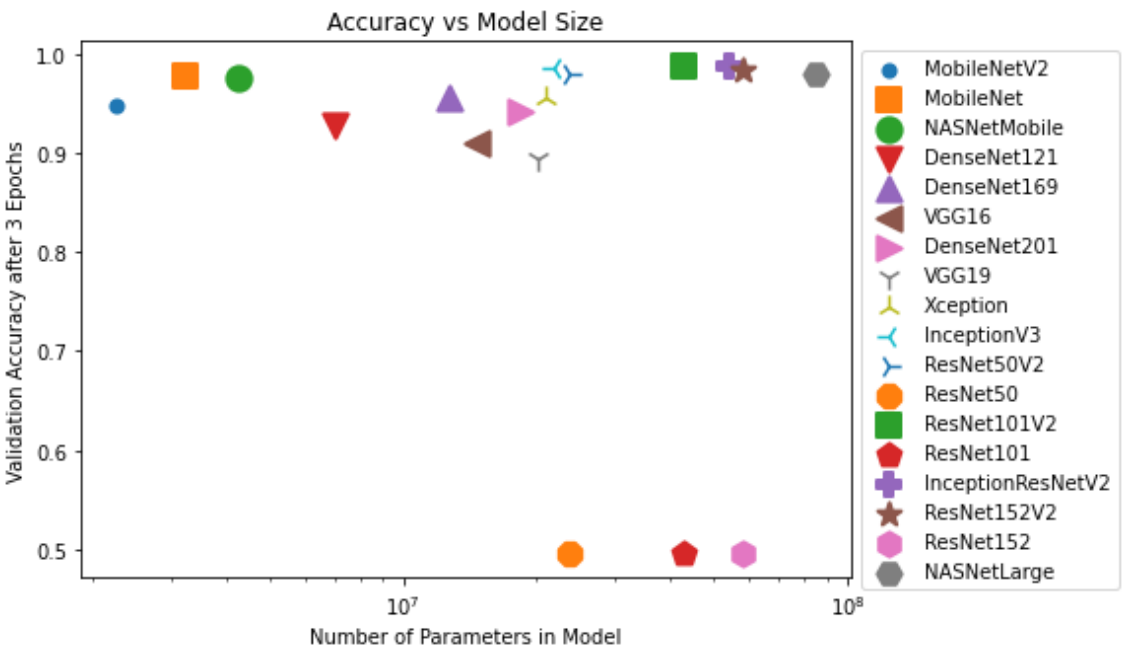


Figure 1: reference

Due to hardware limitations I cannot try all the possible pre-trained models. Therefore I'll try 4 popular models namely:

1. MobileNetV2
2. ResNet50
3. EfficientNetB3
4. InceptionV3

For each model, I try three attempts to get the maximum test accuracy possible. You can see these attempts in parts 2.1 to 2.4.

In parts 3 and 4 I use autoencoder and 10 fold cross validation only on the best performing model with the highest test accuracy (again due to time and GPU limits.)

In part 5 I try to use some techniques to improve accuracy and further fine tune the models.

2. Pre-trained Convolutional Neural Network -Transfer Learning-

2.1. MobileNetV2

◬First Attempt◬

Validation Accuracy = 93.75%
Test Accuracy = 77.00%

◉ **Parameters:**

Optimization Algorithm: adam
Learning Rate: NA
Momentum: NA
Epoch: 150
Pooling: avg
Number of trainable layers of the base model: 5
Activation: softmax

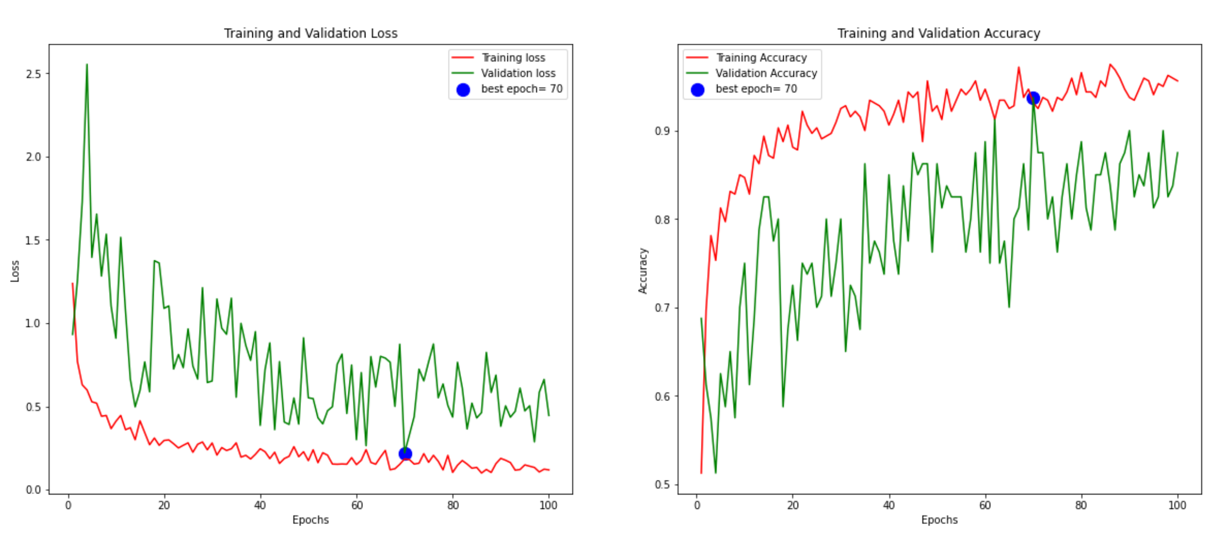
◉ **Model Summery:**

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 mobilenetv2_1.00_224 (Funct  (None, 1280)             2257984
 ional)

 dense_7 (Dense)             (None, 5)                 6405

=================================================================
Total params: 2,264,389
Trainable params: 419,205
Non-trainable params: 1,845,184
_________________________________________________________________
```

◉ **Last 10 Epochs (with early stopping, patience=30):**

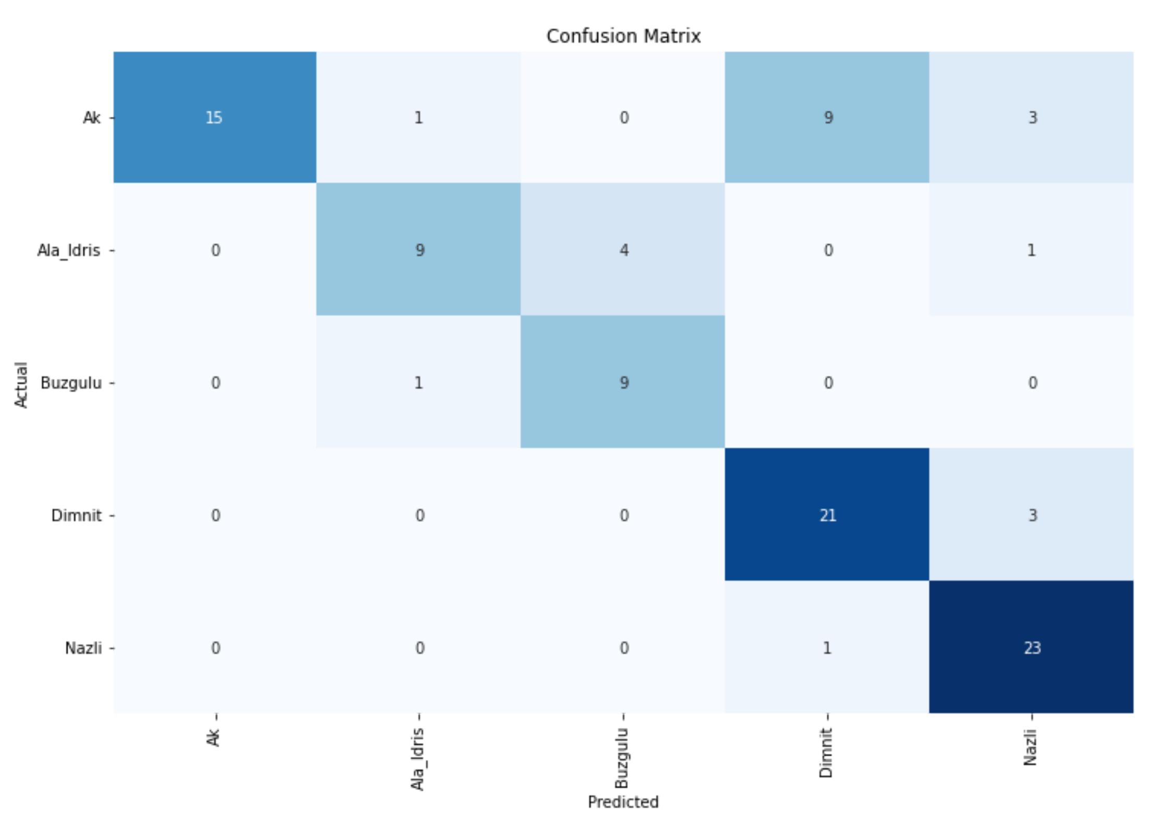
```
Epoch 90/150
10/10 [==============================] - 6s 656ms/step - loss: 0.1874 - accuracy: 0.9375 - val_loss: 0.3802 - val_accuracy: 0.9000
Epoch 91/150
10/10 [==============================] - 6s 660ms/step - loss: 0.1763 - accuracy: 0.9344 - val_loss: 0.5030 - val_accuracy: 0.8250
Epoch 92/150
10/10 [==============================] - 7s 669ms/step - loss: 0.1622 - accuracy: 0.9469 - val_loss: 0.4349 - val_accuracy: 0.8500
Epoch 93/150
10/10 [==============================] - 6s 657ms/step - loss: 0.1167 - accuracy: 0.9594 - val_loss: 0.4703 - val_accuracy: 0.8375
Epoch 94/150
10/10 [==============================] - 7s 662ms/step - loss: 0.1205 - accuracy: 0.9563 - val_loss: 0.6098 - val_accuracy: 0.8750
Epoch 95/150
10/10 [==============================] - 6s 654ms/step - loss: 0.1483 - accuracy: 0.9406 - val_loss: 0.4727 - val_accuracy: 0.8125
Epoch 96/150
10/10 [==============================] - 6s 655ms/step - loss: 0.1404 - accuracy: 0.9531 - val_loss: 0.5034 - val_accuracy: 0.8250
Epoch 97/150
10/10 [==============================] - 6s 652ms/step - loss: 0.1329 - accuracy: 0.9500 - val_loss: 0.2867 - val_accuracy: 0.9000
Epoch 98/150
10/10 [==============================] - 7s 663ms/step - loss: 0.1060 - accuracy: 0.9625 - val_loss: 0.5839 - val_accuracy: 0.8250
Epoch 99/150
10/10 [==============================] - 7s 664ms/step - loss: 0.1227 - accuracy: 0.9594 - val_loss: 0.6619 - val_accuracy: 0.8375
Epoch 100/150
10/10 [==============================] - ETA: 0s - loss: 0.1183 - accuracy: 0.9563Restoring model weights from the end of the best epoch:
10/10 [==============================] - 7s 674ms/step - loss: 0.1183 - accuracy: 0.9563 - val_loss: 0.4450 - val_accuracy: 0.8750
Epoch 100: early stopping
------------------
Best accuracy achieved: 93.75 %
```

◉ **Training And Validation Loss/Accuracy:**



The plot above is showing definite signs of overfitting.

◉ **Confusion Matrix:**



◉ **Classification Report:**

```
Classification Report:
----------------------
              precision    recall  f1-score   support

          Ak     1.0000    0.5357    0.6977        28
   Ala_Idris     0.8182    0.6429    0.7200        14
     Buzgulu     0.6923    0.9000    0.7826        10
      Dimnit     0.6774    0.8750    0.7636        24
       Nazli     0.7667    0.9583    0.8519        24

    accuracy                         0.7700       100
   macro avg     0.7909    0.7824    0.7632       100
weighted avg     0.8104    0.7700    0.7621       100
```

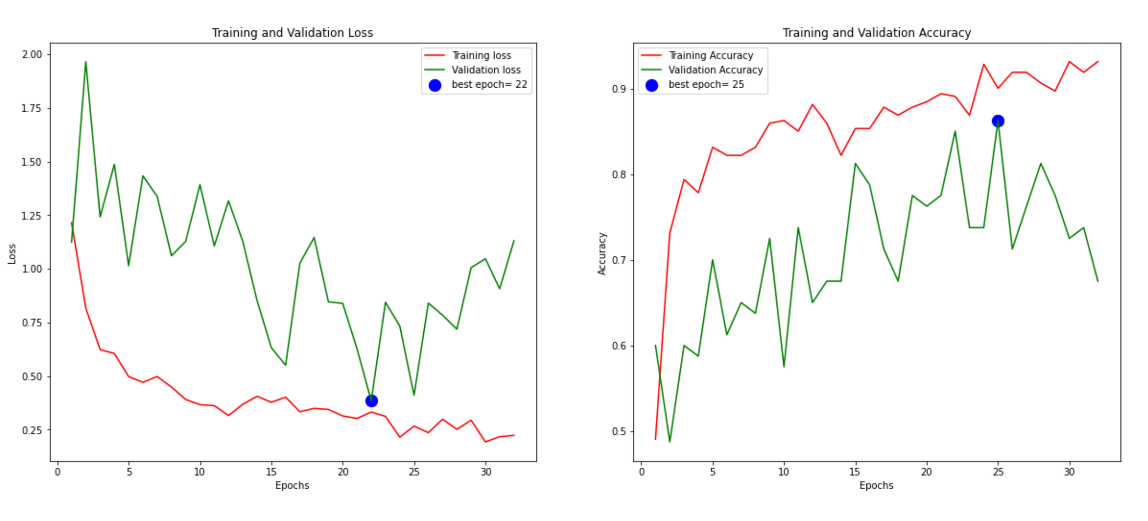
◬Second Attempt◬

Decreasing the patience to 10
Validation Accuracy = 86.25%
Test Accuracy = 81%

◉ **Last 10 Epochs (with early stopping, patience=10):**

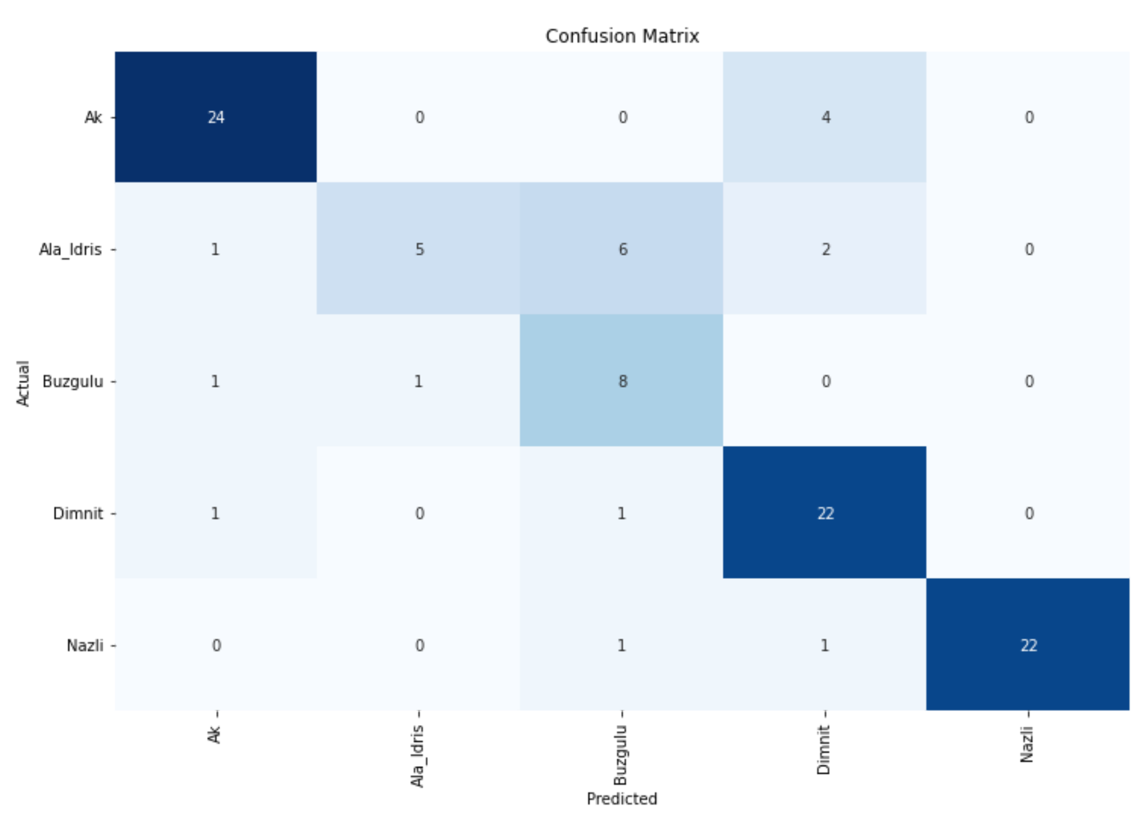
```
Epoch 22/150
10/10 [==============================] - 7s 665ms/step - loss: 0.3317 - accuracy: 0.8906 - val_loss: 0.3844 - val_accuracy: 0.8500
Epoch 23/150
10/10 [==============================] - 7s 663ms/step - loss: 0.3119 - accuracy: 0.8687 - val_loss: 0.8439 - val_accuracy: 0.7375
Epoch 24/150
10/10 [==============================] - 7s 665ms/step - loss: 0.2145 - accuracy: 0.9281 - val_loss: 0.7328 - val_accuracy: 0.7375
Epoch 25/150
10/10 [==============================] - 7s 725ms/step - loss: 0.2662 - accuracy: 0.9000 - val_loss: 0.4104 - val_accuracy: 0.8625
Epoch 26/150
10/10 [==============================] - 7s 706ms/step - loss: 0.2355 - accuracy: 0.9187 - val_loss: 0.8399 - val_accuracy: 0.7125
Epoch 27/150
10/10 [==============================] - 6s 657ms/step - loss: 0.2986 - accuracy: 0.9187 - val_loss: 0.7834 - val_accuracy: 0.7625
Epoch 28/150
10/10 [==============================] - 6s 660ms/step - loss: 0.2512 - accuracy: 0.9062 - val_loss: 0.7181 - val_accuracy: 0.8125
Epoch 29/150
10/10 [==============================] - 7s 672ms/step - loss: 0.2937 - accuracy: 0.8969 - val_loss: 1.0050 - val_accuracy: 0.7750
Epoch 30/150
10/10 [==============================] - 6s 658ms/step - loss: 0.1928 - accuracy: 0.9312 - val_loss: 1.0467 - val_accuracy: 0.7250
Epoch 31/150
10/10 [==============================] - 6s 657ms/step - loss: 0.2169 - accuracy: 0.9187 - val_loss: 0.9061 - val_accuracy: 0.7375
Epoch 32/150
10/10 [==============================] - ETA: 0s - loss: 0.2231 - accuracy: 0.9312Restoring model weights from the end of the best epoch:
10/10 [==============================] - 7s 665ms/step - loss: 0.2231 - accuracy: 0.9312 - val_loss: 1.1306 - val_accuracy: 0.6750
Epoch 32: early stopping
------------------
Best accuracy achieved: 86.25 %
```

◉ **Training And Validation Loss/Accuracy:**



Again we can see clear signs of overfitting.

◉ **Confusion Matrix :**



◉ **Classification Report:**

```
Classification Report:
----------------------
              precision    recall  f1-score   support

          Ak     0.8889    0.8571    0.8727        28
   Ala_Idris     0.8333    0.3571    0.5000        14
     Buzgulu     0.5000    0.8000    0.6154        10
      Dimnit     0.7586    0.9167    0.8302        24
       Nazli     1.0000    0.9167    0.9565        24

    accuracy                         0.8100       100
   macro avg     0.7962    0.7695    0.7550       100
weighted avg     0.8376    0.8100    0.8047       100
```

◬Third Attempt◬

Adding a dropout layers
Validation Accuracy = 90%
Test Accuracy = 80%

Possible solutions for the overfitting:
1. We add **Dropout** layers before the classification layer, for regularization.
2. Also freeze all the layers of the pre-trained model to prevent overfitting.

◉ **Parameters:**

Optimization Algorithm: adam
Learning Rate: NA
Momentum: NA
Epoch: 150
Pooling: avg
Number of trainable layers of the base model: 0
Activation: softmax

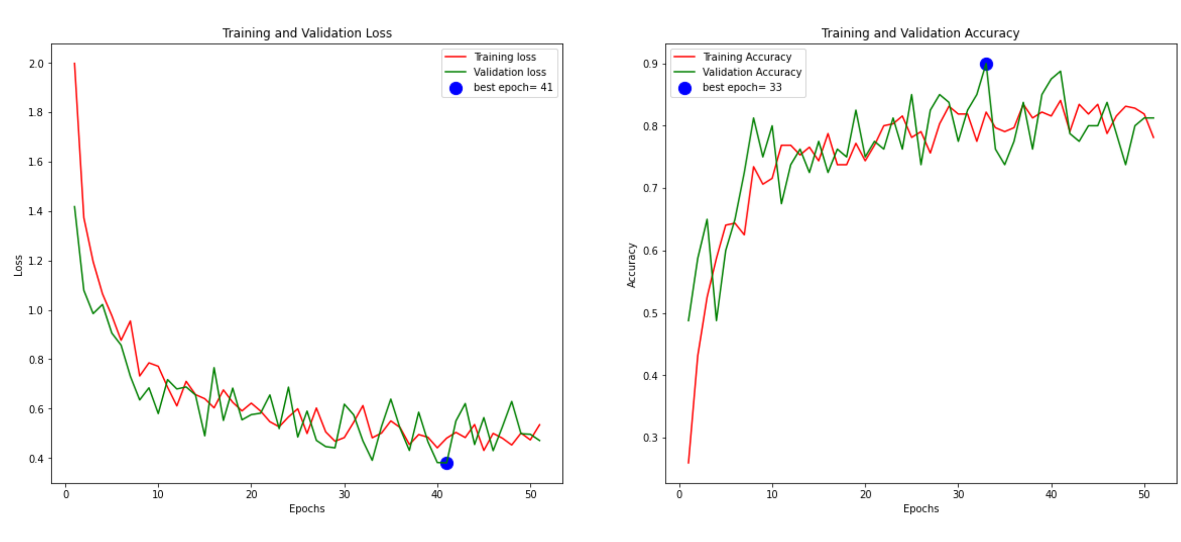
◉ **Model Summery:**

```
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 mobilenetv2_1.00_224 (Funct  (None, 1280)             2257984
 ional)

 dense_3 (Dense)             (None, 512)               655872

 dropout_3 (Dropout)         (None, 512)               0

 dense_4 (Dense)             (None, 512)               262656

 dropout_4 (Dropout)         (None, 512)               0

 dense_5 (Dense)             (None, 5)                 2565

=================================================================
Total params: 3,179,077
Trainable params: 921,093
Non-trainable params: 2,257,984
_________________________________________________________________
```

◉ **Last 10 Epochs (with early stopping, patience=10):**

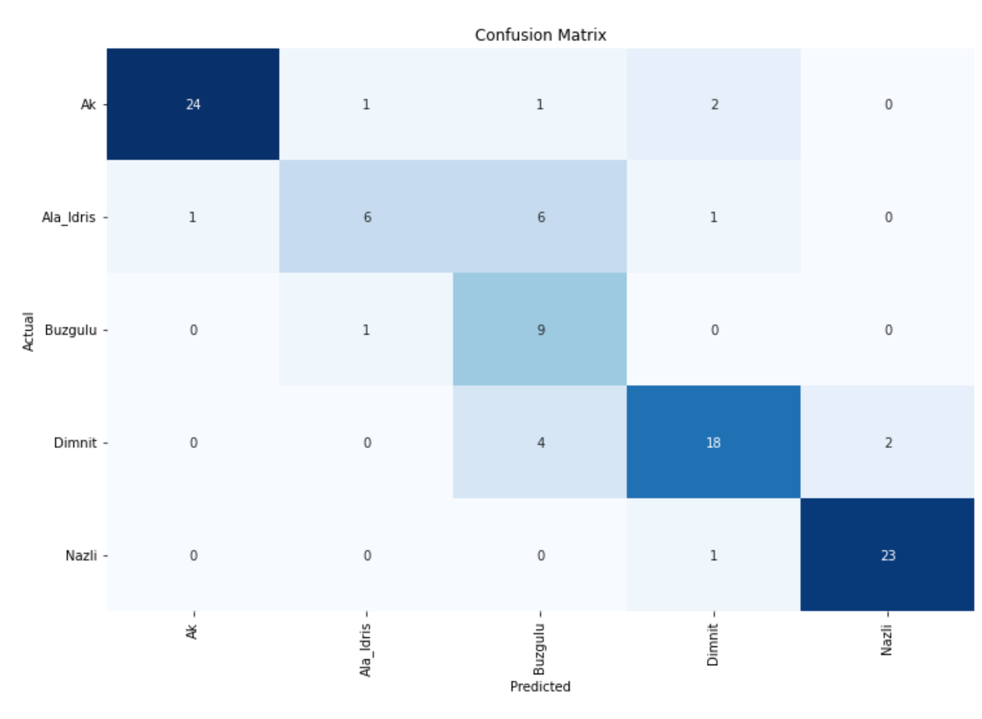
```
Epoch 41/150
10/10 [==============================] - 7s 669ms/step - loss: 0.4805 - accuracy: 0.8406 - val_loss: 0.3804 - val_accuracy: 0.8875
Epoch 42/150
10/10 [==============================] - 6s 656ms/step - loss: 0.5039 - accuracy: 0.7906 - val_loss: 0.5503 - val_accuracy: 0.7875
Epoch 43/150
10/10 [==============================] - 6s 654ms/step - loss: 0.4830 - accuracy: 0.8344 - val_loss: 0.6209 - val_accuracy: 0.7750
Epoch 44/150
10/10 [==============================] - 7s 669ms/step - loss: 0.5358 - accuracy: 0.8188 - val_loss: 0.4548 - val_accuracy: 0.8000
Epoch 45/150
10/10 [==============================] - 6s 653ms/step - loss: 0.4307 - accuracy: 0.8344 - val_loss: 0.5638 - val_accuracy: 0.8000
Epoch 46/150
10/10 [==============================] - 7s 661ms/step - loss: 0.4999 - accuracy: 0.7875 - val_loss: 0.4298 - val_accuracy: 0.8375
Epoch 47/150
10/10 [==============================] - 6s 652ms/step - loss: 0.4804 - accuracy: 0.8156 - val_loss: 0.5270 - val_accuracy: 0.7875
Epoch 48/150
10/10 [==============================] - 6s 654ms/step - loss: 0.4528 - accuracy: 0.8313 - val_loss: 0.6294 - val_accuracy: 0.7375
Epoch 49/150
10/10 [==============================] - 6s 658ms/step - loss: 0.5011 - accuracy: 0.8281 - val_loss: 0.4984 - val_accuracy: 0.8000
Epoch 50/150
10/10 [==============================] - 7s 664ms/step - loss: 0.4735 - accuracy: 0.8188 - val_loss: 0.4963 - val_accuracy: 0.8125
Epoch 51/150
10/10 [==============================] - ETA: 0s - loss: 0.5349 - accuracy: 0.7812Restoring model weights from the end of the best epoch: 41.
10/10 [==============================] - 7s 670ms/step - loss: 0.5349 - accuracy: 0.7812 - val_loss: 0.4710 - val_accuracy: 0.8125
Epoch 51: early stopping
------------------
Best accuracy achieved: 90.0 %
```

◉ **Training And Validation Loss/Accuracy:**



The plot above is showing that we are neither overfitting nor underfitting! So this is an ideal fit!

◉ **Confusion Matrix:**



◉ **Classification Report:**

```
Classification Report:
----------------------
              precision    recall  f1-score   support

          Ak     0.9600    0.8571    0.9057        28
   Ala_Idris     0.7500    0.4286    0.5455        14
     Buzgulu     0.4500    0.9000    0.6000        10
      Dimnit     0.8182    0.7500    0.7826        24
       Nazli     0.9200    0.9583    0.9388        24

    accuracy                         0.8000       100
   macro avg     0.7796    0.7788    0.7545       100
weighted avg     0.8360    0.8000    0.8031       100
```

◬Summary of MobileNetV2◬

Best Parameters:

Optimization Algorithm: adam
Learning Rate: NA
Momentum: NA
Epoch: 150
Pooling: avg
Number of trainable layers of the base model: 0
Activation: softmax

Best Validation Accuracy: 90%

Best Test Accuracy: 80%

Best CNN Architecture:

```
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 mobilenetv2_1.00_224 (Funct  (None, 1280)             2257984
 ional)

 dense_3 (Dense)             (None, 512)               655872

 dropout_3 (Dropout)         (None, 512)               0

 dense_4 (Dense)             (None, 512)               262656

 dropout_4 (Dropout)         (None, 512)               0

 dense_5 (Dense)             (None, 5)                 2565

=================================================================
Total params: 3,179,077
Trainable params: 921,093
Non-trainable params: 2,257,984
_________________________________________________________________
```

Notes:

Dropout layers would increase the test accuracy and prevent overfitting.
More parameter tuning is done in part 5.

---

2. Pre-trained Convolutional Neural Network -Transfer Learning-

2.2. ResNet50

⚠First Attempt⚠

Validation Accuracy = 81.25%

Test Accuracy = 75%

- **Parameters:**
  - **Optimization Algorithm:** adam
  - **Learning Rate:** NA
  - **Momentum:** NA
  - **Epoch:** 150
  - **Pooling:** avg
  - **Number of trainable layers of the base model:** 5
  - **Activation:** softmax

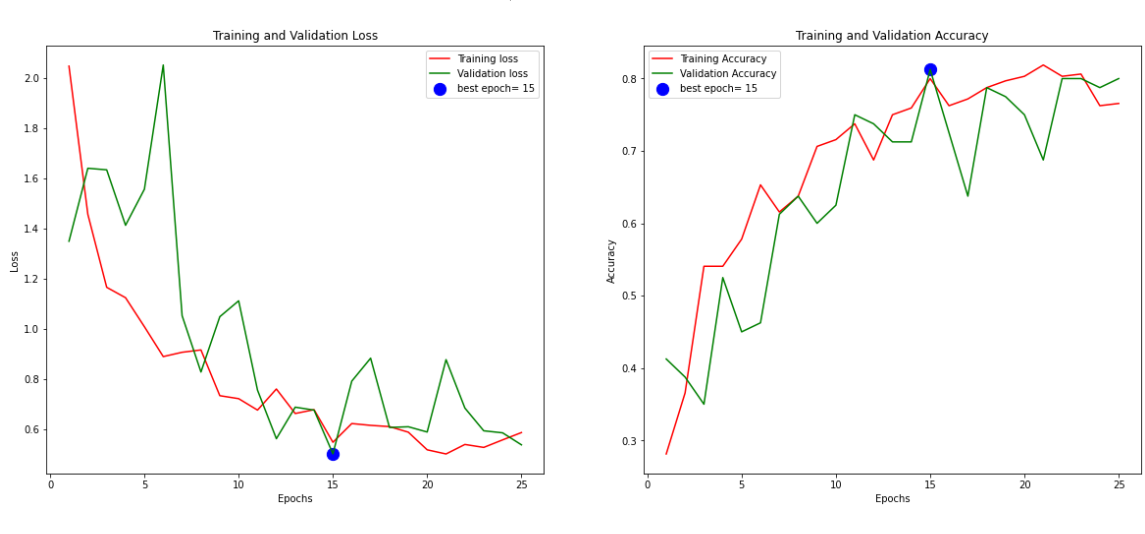
- **Model Summery:**

```
Model: "sequential_10"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 resnet50 (Functional)       (None, 2048)              23587712

 dense_12 (Dense)            (None, 512)               1049088

 dropout_9 (Dropout)         (None, 512)               0

 dense_13 (Dense)            (None, 512)               262656

 dropout_10 (Dropout)        (None, 512)               0

 dense_14 (Dense)            (None, 5)                 2565

=================================================================
Total params: 24,902,021
Trainable params: 2,369,029
Non-trainable params: 22,532,992
_________________________________________________________________
```

- **Last 10 Epochs (with early stopping, patience=10):**

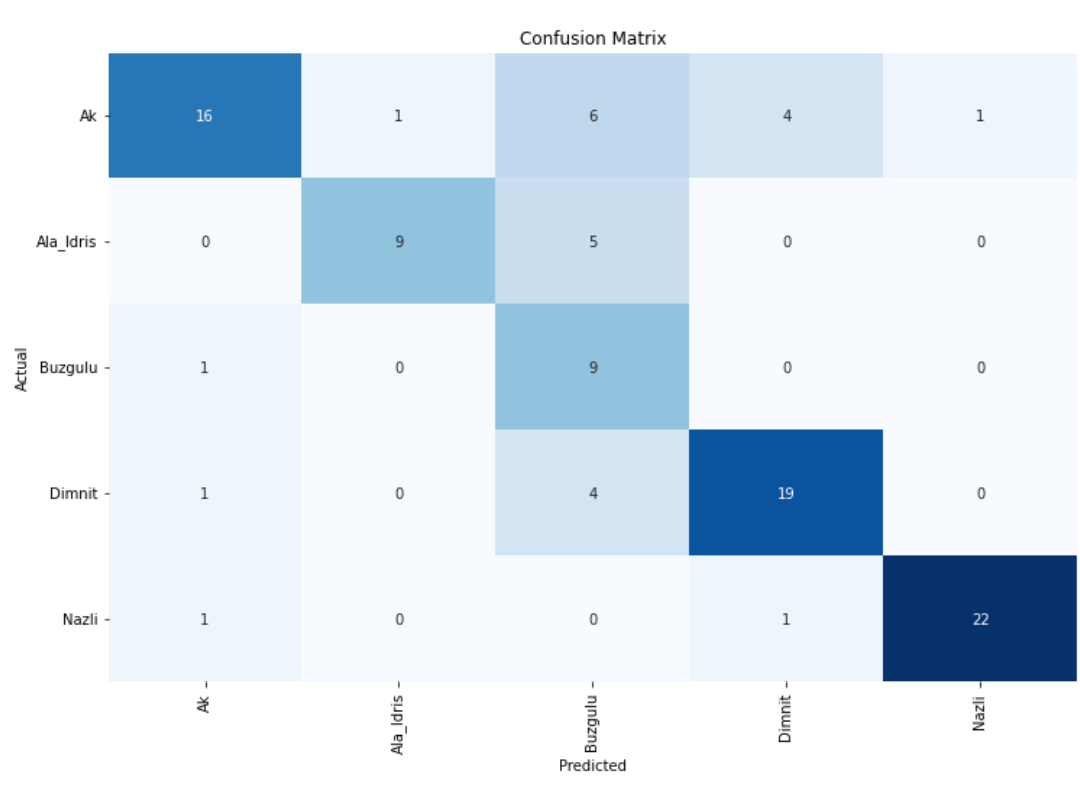
```
10/10 [==============================] - 7s 698ms/step - loss: 0.6782 - accuracy: 0.7594 - val_loss: 0.6765 - val_accuracy: 0.7125
Epoch 15/150
10/10 [==============================] - 7s 713ms/step - loss: 0.5485 - accuracy: 0.8000 - val_loss: 0.5025 - val_accuracy: 0.8125
Epoch 16/150
10/10 [==============================] - 7s 696ms/step - loss: 0.6230 - accuracy: 0.7625 - val_loss: 0.7919 - val_accuracy: 0.7250
Epoch 17/150
10/10 [==============================] - 7s 697ms/step - loss: 0.6158 - accuracy: 0.7719 - val_loss: 0.8832 - val_accuracy: 0.6375
Epoch 18/150
10/10 [==============================] - 7s 697ms/step - loss: 0.6109 - accuracy: 0.7875 - val_loss: 0.6074 - val_accuracy: 0.7875
Epoch 19/150
10/10 [==============================] - 7s 701ms/step - loss: 0.5886 - accuracy: 0.7969 - val_loss: 0.6102 - val_accuracy: 0.7750
Epoch 20/150
10/10 [==============================] - 7s 701ms/step - loss: 0.5184 - accuracy: 0.8031 - val_loss: 0.5893 - val_accuracy: 0.7500
Epoch 21/150
10/10 [==============================] - 7s 697ms/step - loss: 0.5019 - accuracy: 0.8188 - val_loss: 0.8774 - val_accuracy: 0.6875
Epoch 22/150
10/10 [==============================] - 7s 692ms/step - loss: 0.5398 - accuracy: 0.8031 - val_loss: 0.6846 - val_accuracy: 0.8000
Epoch 23/150
10/10 [==============================] - 7s 696ms/step - loss: 0.5280 - accuracy: 0.8062 - val_loss: 0.5941 - val_accuracy: 0.8000
Epoch 24/150
10/10 [==============================] - 7s 693ms/step - loss: 0.5580 - accuracy: 0.7625 - val_loss: 0.5861 - val_accuracy: 0.7875
Epoch 25/150
10/10 [==============================] - ETA: 0s - loss: 0.5871 - accuracy: 0.7656Restoring model weights from the end of the best epoch: 15.
10/10 [==============================] - 7s 710ms/step - loss: 0.5871 - accuracy: 0.7656 - val_loss: 0.5383 - val_accuracy: 0.8000
Epoch 25: early stopping
------------------
Best accuracy achieved: 81.25 %
```

- **Training And Validation Loss/Accuracy:**



Slightly overfitting.

- **Confusion Matrix:**



- **Classification Report:**

```
Classification Report:
----------------------
              precision    recall  f1-score   support

          Ak     0.8421    0.5714    0.6809        28
   Ala_Idris     0.9000    0.6429    0.7500        14
     Buzgulu     0.3750    0.9000    0.5294        10
      Dimnit     0.7917    0.7917    0.7917        24
       Nazli     0.9565    0.9167    0.9362        24

    accuracy                         0.7500       100
   macro avg     0.7731    0.7645    0.7376       100
weighted avg     0.8189    0.7500    0.7633       100
```

⚠Second Attempt⚠

Changing the CNN architecture in hope of better accuracy.

Validation Accuracy = 78%

Test Accuracy = 62%

- **Parameters:**
  - **Optimization Algorithm:** adam
  - **Learning Rate:** NA
  - **Momentum:** NA
  - **Epoch:** 150
  - **Pooling:** max
  - **Number of trainable layers of the base model:** 0
  - **Activation:** softmax

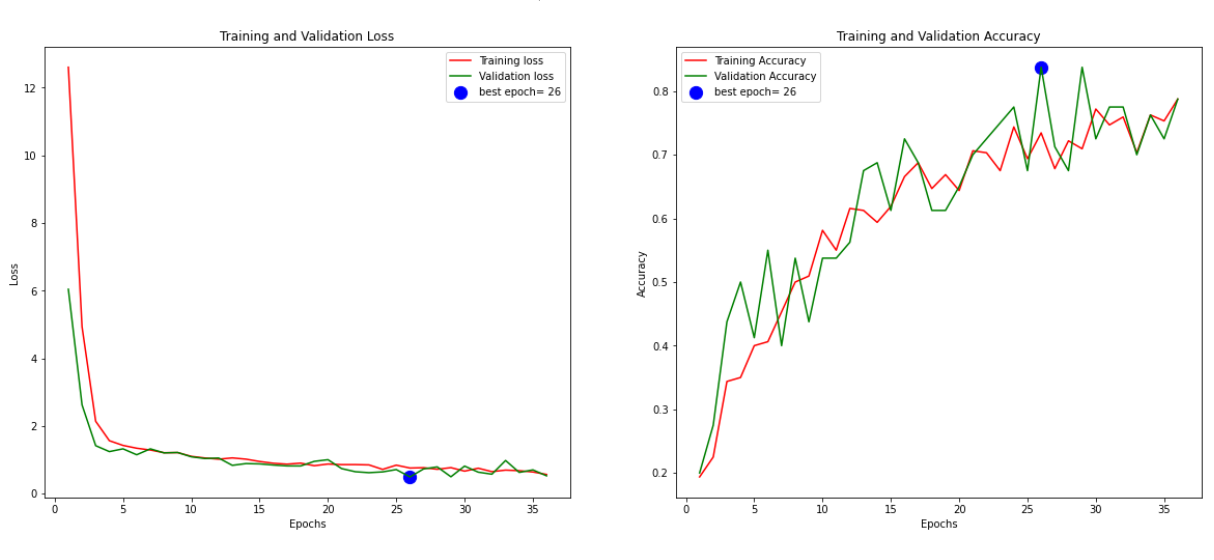
- **Model Summery:**

```
 conv5_block3_2_conv (Conv2D)   (None, 8, 8, 512)    2359808     ['conv5_block3_1_relu[0][0]']

 conv5_block3_2_bn (BatchNormal  (None, 8, 8, 512)   2048        ['conv5_block3_2_conv[0][0]']
 ization)

 conv5_block3_2_relu (Activatio  (None, 8, 8, 512)   0           ['conv5_block3_2_bn[0][0]']
 n)

 conv5_block3_3_conv (Conv2D)   (None, 8, 8, 2048)   1050624     ['conv5_block3_2_relu[0][0]']

 conv5_block3_3_bn (BatchNormal  (None, 8, 8, 2048)  8192        ['conv5_block3_3_conv[0][0]']
 ization)

 conv5_block3_add (Add)         (None, 8, 8, 2048)   0           ['conv5_block2_out[0][0]',
                                                                  'conv5_block3_3_bn[0][0]']

 conv5_block3_out (Activation)  (None, 8, 8, 2048)   0           ['conv5_block3_add[0][0]']

 global_average_pooling2d_14 (G  (None, 2048)        0           ['conv5_block3_out[0][0]']
 lobalAveragePooling2D)

 dense_21 (Dense)               (None, 5)            10245       ['global_average_pooling2d_14[0][
                                                                 0]']

==================================================================================================
Total params: 23,597,957
Trainable params: 10,245
Non-trainable params: 23,587,712
__________________________________________________________________________________________________
```

- **Last 10 Epochs (with early stopping, patience=10):**

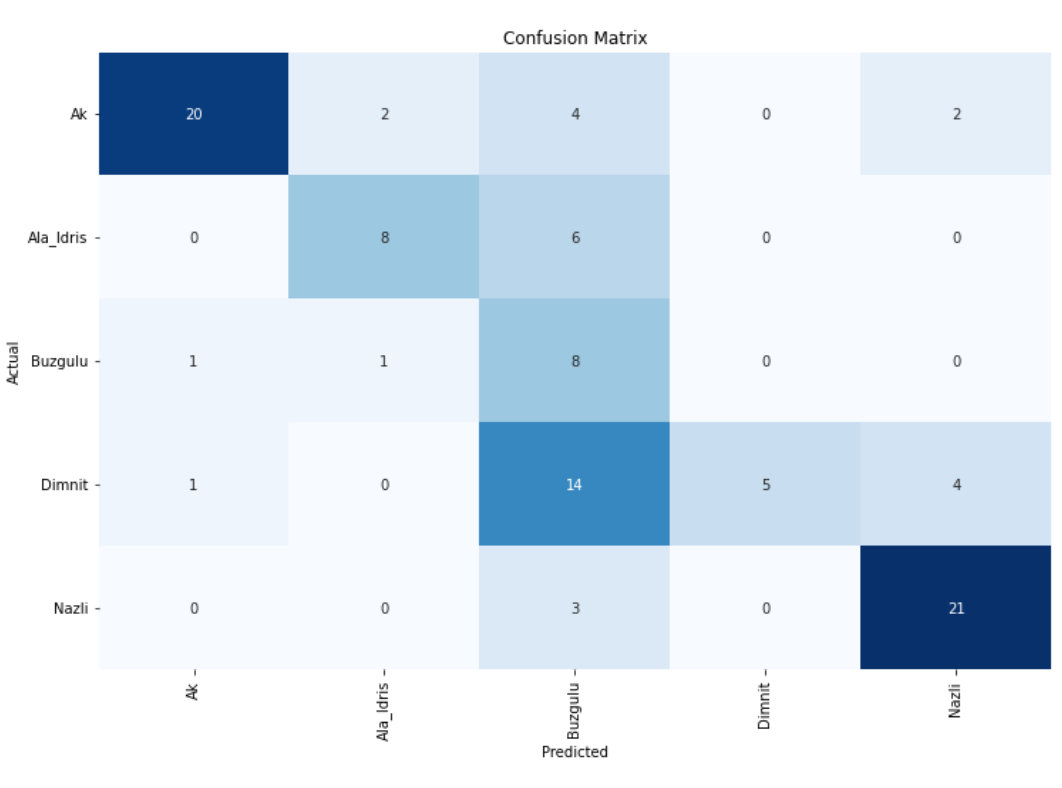
```
Epoch 40/50
10/10 - 7s - loss: 0.7256 - accuracy: 0.7500 - val_loss: 0.5910 - val_accuracy: 0.8500 - 7s/epoch - 709ms/step
Epoch 41/50
10/10 - 7s - loss: 0.7609 - accuracy: 0.7469 - val_loss: 0.7312 - val_accuracy: 0.7250 - 7s/epoch - 690ms/step
Epoch 42/50
10/10 - 7s - loss: 0.6678 - accuracy: 0.7969 - val_loss: 0.6365 - val_accuracy: 0.8125 - 7s/epoch - 694ms/step
Epoch 43/50
10/10 - 7s - loss: 0.6699 - accuracy: 0.7750 - val_loss: 0.6827 - val_accuracy: 0.7250 - 7s/epoch - 695ms/step
Epoch 44/50
10/10 - 7s - loss: 0.6648 - accuracy: 0.7969 - val_loss: 0.6489 - val_accuracy: 0.8000 - 7s/epoch - 693ms/step
Epoch 45/50
10/10 - 7s - loss: 0.6594 - accuracy: 0.7750 - val_loss: 0.7196 - val_accuracy: 0.7000 - 7s/epoch - 698ms/step
Epoch 46/50
10/10 - 7s - loss: 0.6652 - accuracy: 0.8000 - val_loss: 0.6371 - val_accuracy: 0.8125 - 7s/epoch - 704ms/step
Epoch 47/50
10/10 - 7s - loss: 0.6767 - accuracy: 0.7719 - val_loss: 0.6567 - val_accuracy: 0.8000 - 7s/epoch - 698ms/step
Epoch 48/50
10/10 - 7s - loss: 0.6947 - accuracy: 0.7969 - val_loss: 0.6084 - val_accuracy: 0.8125 - 7s/epoch - 688ms/step
Epoch 49/50
10/10 - 7s - loss: 0.6499 - accuracy: 0.7906 - val_loss: 0.6835 - val_accuracy: 0.7250 - 7s/epoch - 695ms/step
Epoch 50/50
Restoring model weights from the end of the best epoch: 40.
10/10 - 7s - loss: 0.6213 - accuracy: 0.8062 - val_loss: 0.6599 - val_accuracy: 0.7875 - 7s/epoch - 708ms/step
Epoch 50: early stopping
```

- **Training And Validation Loss/Accuracy:**



No sign of overfitting but the accuracy is very bad.

- **Confusion Matrix :**



- **Classification Report:**

```
Classification Report:
----------------------
              precision    recall  f1-score   support

          Ak     0.9091    0.7143    0.8000        28
   Ala_Idris     0.7273    0.5714    0.6400        14
     Buzgulu     0.2286    0.8000    0.3556        10
      Dimnit     1.0000    0.2083    0.3448        24
       Nazli     0.7778    0.8750    0.8235        24

    accuracy                         0.6200       100
   macro avg     0.7285    0.6338    0.5928       100
weighted avg     0.8059    0.6200    0.6296       100
```

⚠Third Attempt⚠

Unfreeze the last 2 layers of ResNet

Add two dense layers

Validation Accuracy = 83.75%

Test Accuracy = 65%

- **Parameters:**
  - **Optimization Algorithm:** adam
  - **Learning Rate:** NA
  - **Momentum:** NA
  - **Epoch:** 150
  - **Pooling:** max
  - **Number of trainable layers of the base model:** 2
  - **Activation:** softmax

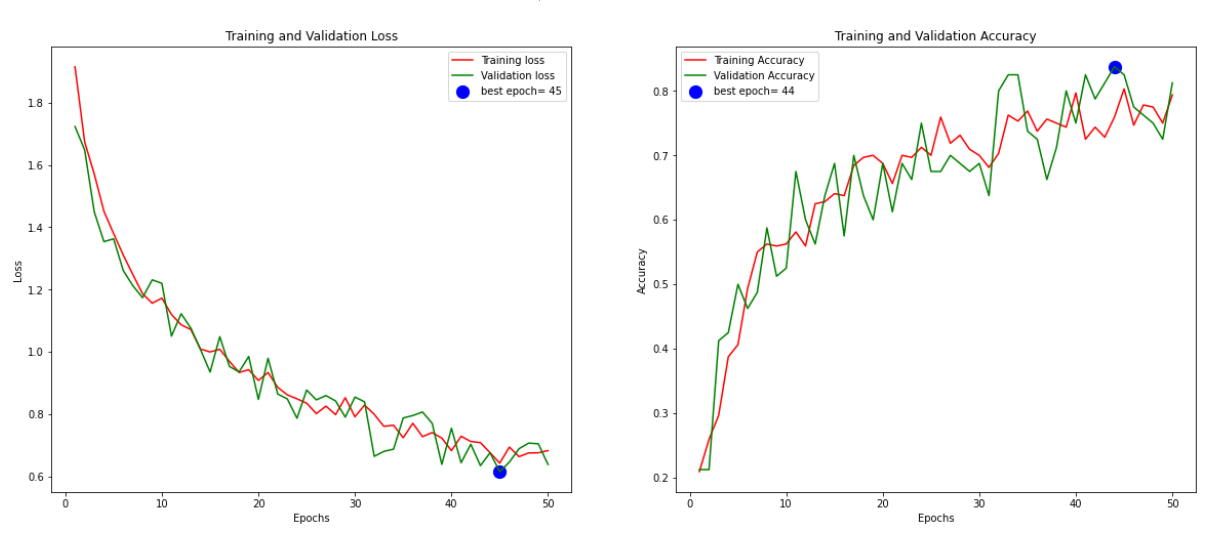
- **Model Summery:**

```
 conv5_block3_2_conv (Conv2D)   (None, 8, 8, 512)    2359808     ['conv5_block3_1_relu[0][0]']

 conv5_block3_2_bn (BatchNormal  (None, 8, 8, 512)   2048        ['conv5_block3_2_conv[0][0]']
 ization)

 conv5_block3_2_relu (Activatio  (None, 8, 8, 512)   0           ['conv5_block3_2_bn[0][0]']
 n)

 conv5_block3_3_conv (Conv2D)   (None, 8, 8, 2048)   1050624     ['conv5_block3_2_relu[0][0]']

 conv5_block3_3_bn (BatchNormal  (None, 8, 8, 2048)  8192        ['conv5_block3_3_conv[0][0]']
 ization)

 conv5_block3_add (Add)         (None, 8, 8, 2048)   0           ['conv5_block2_out[0][0]',
                                                                  'conv5_block3_3_bn[0][0]']

 conv5_block3_out (Activation)  (None, 8, 8, 2048)   0           ['conv5_block3_add[0][0]']

 avg_pool (GlobalAveragePooling  (None, 2048)        0           ['conv5_block3_out[0][0]']
 2D)

 dense_27 (Dense)               (None, 5)            10245       ['avg_pool[0][0]']

==================================================================================================
Total params: 23,597,957
Trainable params: 10,245
Non-trainable params: 23,587,712
__________________________________________________________________________________________________
```

- **Last 10 Epochs (with early stopping, patience=10):**

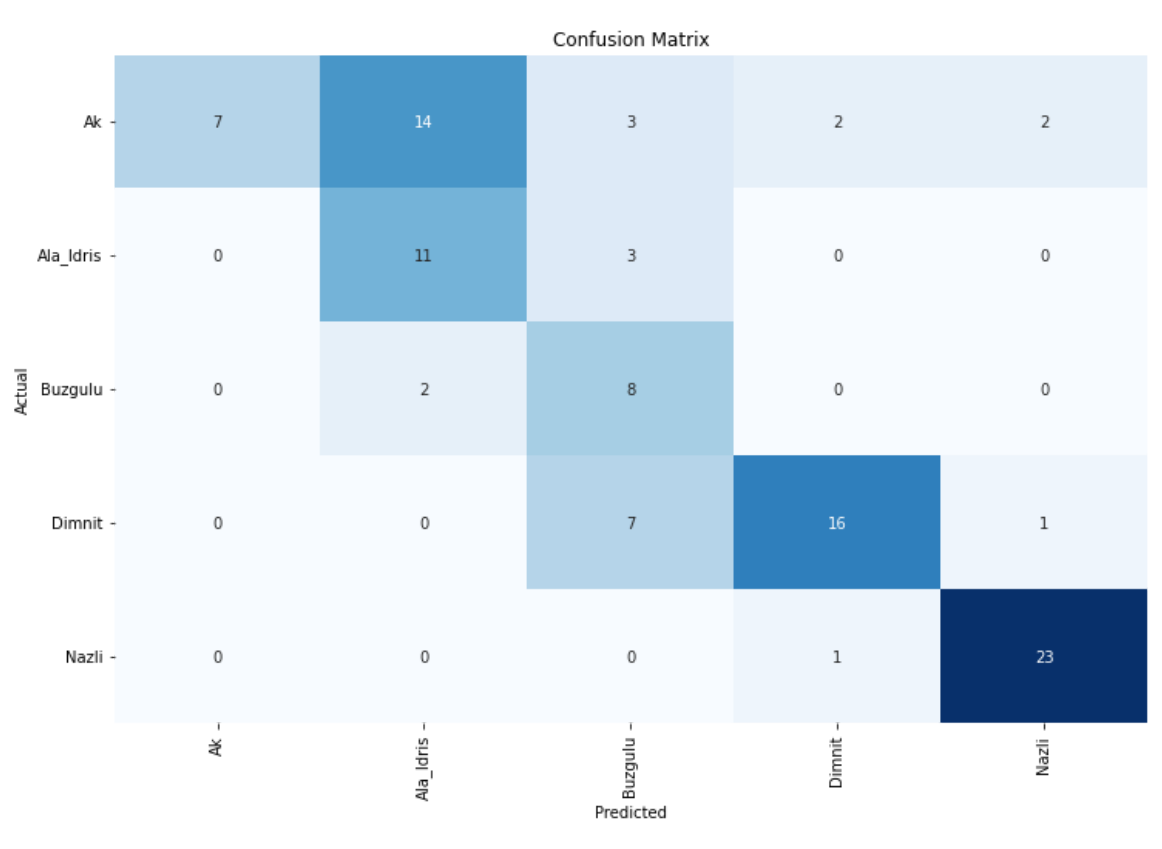
```
Epoch 40/50
10/10 - 7s - loss: 0.6829 - accuracy: 0.7969 - val_loss: 0.7549 - val_accuracy: 0.7500 - 7s/epoch - 696ms/step
Epoch 41/50
10/10 - 7s - loss: 0.7292 - accuracy: 0.7250 - val_loss: 0.6441 - val_accuracy: 0.8250 - 7s/epoch - 691ms/step
Epoch 42/50
10/10 - 7s - loss: 0.7124 - accuracy: 0.7437 - val_loss: 0.7034 - val_accuracy: 0.7875 - 7s/epoch - 694ms/step
Epoch 43/50
10/10 - 7s - loss: 0.7088 - accuracy: 0.7281 - val_loss: 0.6348 - val_accuracy: 0.8125 - 7s/epoch - 707ms/step
Epoch 44/50
10/10 - 7s - loss: 0.6763 - accuracy: 0.7594 - val_loss: 0.6762 - val_accuracy: 0.8375 - 7s/epoch - 696ms/step
Epoch 45/50
10/10 - 7s - loss: 0.6428 - accuracy: 0.8031 - val_loss: 0.6156 - val_accuracy: 0.8250 - 7s/epoch - 707ms/step
Epoch 46/50
10/10 - 7s - loss: 0.6945 - accuracy: 0.7469 - val_loss: 0.6463 - val_accuracy: 0.7750 - 7s/epoch - 696ms/step
Epoch 47/50
10/10 - 7s - loss: 0.6638 - accuracy: 0.7781 - val_loss: 0.6891 - val_accuracy: 0.7625 - 7s/epoch - 696ms/step
Epoch 48/50
10/10 - 7s - loss: 0.6757 - accuracy: 0.7750 - val_loss: 0.7072 - val_accuracy: 0.7500 - 7s/epoch - 696ms/step
Epoch 49/50
10/10 - 7s - loss: 0.6761 - accuracy: 0.7500 - val_loss: 0.7050 - val_accuracy: 0.7250 - 7s/epoch - 697ms/step
Epoch 50/50
10/10 - 7s - loss: 0.6832 - accuracy: 0.7937 - val_loss: 0.6385 - val_accuracy: 0.8125 - 7s/epoch - 702ms/step
------------------
Best accuracy achieved: 83.75 %
```

- **Training And Validation Loss/Accuracy:**



No sign of overfitting but the accuracy is still not good.

- **Confusion Matrix :**



- **Classification Report:**

```
Classification Report:
----------------------
              precision    recall  f1-score   support

          Ak     1.0000    0.2500    0.4000        28
   Ala_Idris     0.4074    0.7857    0.5366        14
     Buzgulu     0.3810    0.8000    0.5161        10
      Dimnit     0.8421    0.6667    0.7442        24
       Nazli     0.8846    0.9583    0.9200        24

    accuracy                         0.6500       100
   macro avg     0.7030    0.6921    0.6234       100
weighted avg     0.7895    0.6500    0.6381       100
```

⚠Summary of ResNet50⚠

Best Parameters:
- **Optimization Algorithm:** adam
- **Learning Rate:** NA
- **Momentum:** NA
- **Epoch:** 150
- **Pooling:** avg
- **Number of trainable layers of the base model:** 5
- **Activation:** softmax

Best Validation Accuracy: 81.25%

Best Test Accuracy: 75%

Best CNN Architecture:

```
Model: "sequential_10"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 resnet50 (Functional)       (None, 2048)              23587712

 dense_12 (Dense)            (None, 512)               1049088

 dropout_9 (Dropout)         (None, 512)               0

 dense_13 (Dense)            (None, 512)               262656

 dropout_10 (Dropout)        (None, 512)               0

 dense_14 (Dense)            (None, 5)                 2565

=================================================================
Total params: 24,902,021
Trainable params: 2,369,029
Non-trainable params: 22,532,992
_________________________________________________________________
```

Notes:

We didn't have much overfitting issue with ResNet50 but the accuracy is also not desirable.

No more fine tuning for ResNet. Evidently this CNN architecture does not work well on our dataset.

2. Pre-trained Convolutional Neural Network -Transfer Learning-

2.3. EfficientNetB3

◬First Attempt◬

Validation Accuracy = 72.5%

Test Accuracy = 72.00%

◉ **Parameters:**

Optimization Algorithm: adam
Learning Rate: NA
Momentum: NA
Epoch: 150
Pooling: max
Number of trainable layers of the base model: 2
Activation: softmax

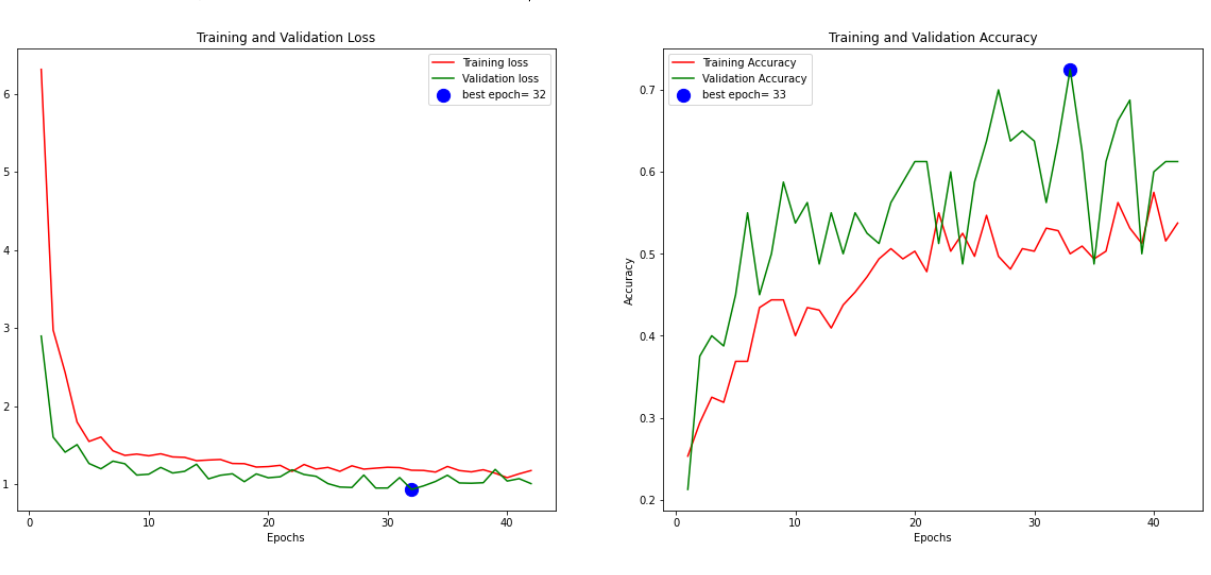
◉ **Model Summery:**

```
Model: "sequential_15"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 efficientnetb3 (Functional)  (None, 1536)             10783535

 dense_31 (Dense)            (None, 512)               786944

 dropout_15 (Dropout)        (None, 512)               0

 dense_32 (Dense)            (None, 512)               262656

 dropout_16 (Dropout)        (None, 512)               0

 dense_33 (Dense)            (None, 5)                 2565

=================================================================
Total params: 11,835,700
Trainable params: 1,052,165
Non-trainable params: 10,783,535
_________________________________________________________________
```

◉ **Last 10 Epochs (with early stopping, patience=10):**

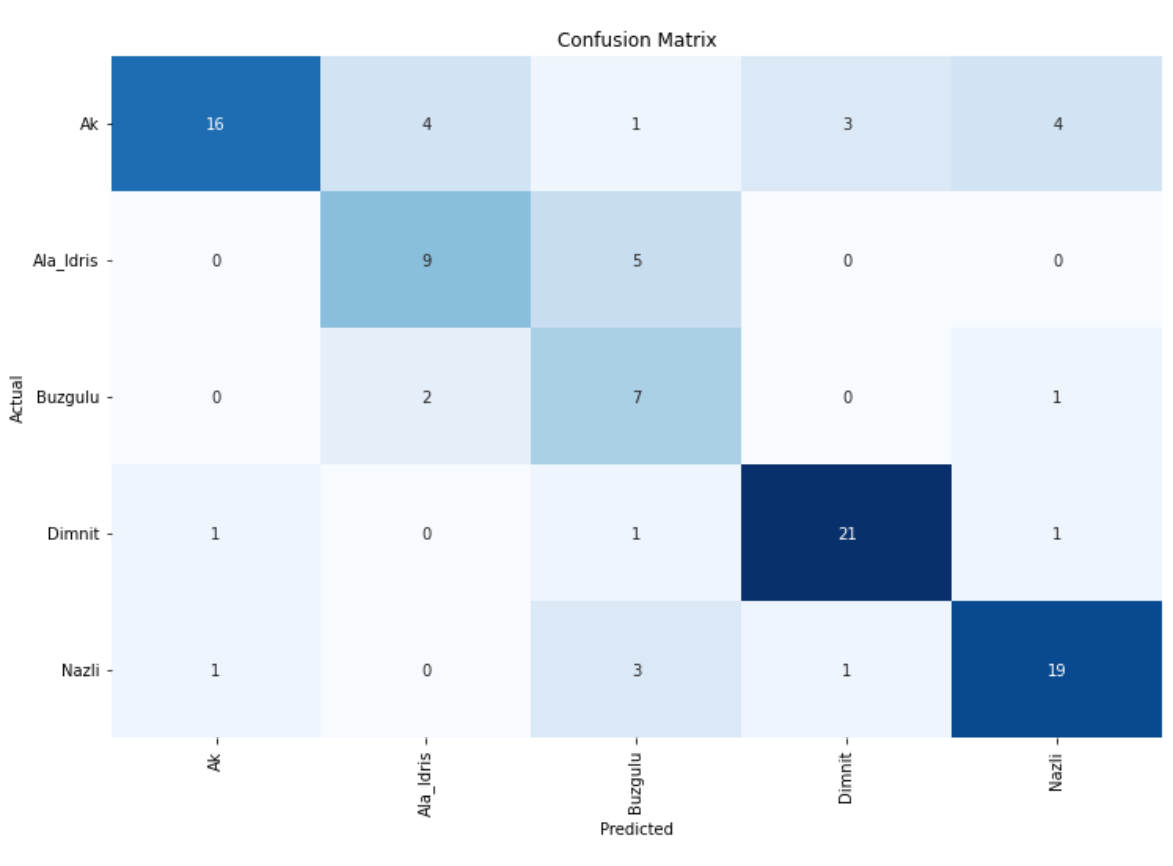
```
Epoch 32/50
10/10 - 7s - loss: 1.1771 - accuracy: 0.5281 - val_loss: 0.9315 - val_accuracy: 0.6375 - 7s/epoch - 695ms/step
Epoch 33/50
10/10 - 7s - loss: 1.1746 - accuracy: 0.5000 - val_loss: 0.9766 - val_accuracy: 0.7250 - 7s/epoch - 681ms/step
Epoch 34/50
10/10 - 7s - loss: 1.1535 - accuracy: 0.5094 - val_loss: 1.0333 - val_accuracy: 0.6250 - 7s/epoch - 680ms/step
Epoch 35/50
10/10 - 7s - loss: 1.2255 - accuracy: 0.4938 - val_loss: 1.1132 - val_accuracy: 0.4875 - 7s/epoch - 683ms/step
Epoch 36/50
10/10 - 7s - loss: 1.1731 - accuracy: 0.5031 - val_loss: 1.0142 - val_accuracy: 0.6125 - 7s/epoch - 681ms/step
Epoch 37/50
10/10 - 7s - loss: 1.1563 - accuracy: 0.5625 - val_loss: 1.0107 - val_accuracy: 0.6625 - 7s/epoch - 686ms/step
Epoch 38/50
10/10 - 7s - loss: 1.1834 - accuracy: 0.5312 - val_loss: 1.0171 - val_accuracy: 0.6875 - 7s/epoch - 684ms/step
Epoch 39/50
10/10 - 7s - loss: 1.1388 - accuracy: 0.5125 - val_loss: 1.1877 - val_accuracy: 0.5000 - 7s/epoch - 684ms/step
Epoch 40/50
10/10 - 7s - loss: 1.0822 - accuracy: 0.5750 - val_loss: 1.0385 - val_accuracy: 0.6000 - 7s/epoch - 689ms/step
Epoch 41/50
10/10 - 7s - loss: 1.1311 - accuracy: 0.5156 - val_loss: 1.0691 - val_accuracy: 0.6125 - 7s/epoch - 687ms/step
Epoch 42/50
Restoring model weights from the end of the best epoch: 32.
10/10 - 7s - loss: 1.1733 - accuracy: 0.5375 - val_loss: 1.0056 - val_accuracy: 0.6125 - 7s/epoch - 698ms/step
Epoch 42: early stopping
-------------------
Best accuracy achieved: 72.5 %
```

◉ **Training And Validation Loss/Accuracy:**



The plot above is showing definite signs of underfitting.

◉ **Confusion Matrix:**



◉ **Classification Report:**

```
Classification Report:
----------------------
              precision    recall  f1-score   support

          Ak     0.8889    0.5714    0.6957        28
   Ala_Idris     0.6000    0.6429    0.6207        14
     Buzgulu     0.4118    0.7000    0.5185        10
      Dimnit     0.8400    0.8750    0.8571        24
       Nazli     0.7600    0.7917    0.7755        24

    accuracy                         0.7200       100
   macro avg     0.7001    0.7162    0.6935       100
weighted avg     0.7581    0.7200    0.7254       100
```

◬Second Attempt◬

Removing a dropout layer
unfreezing more pre-trained layer.

Validation Accuracy = 83.75%

Test Accuracy = 85.00%

◉ **Parameters:**

Optimization Algorithm: adam
Learning Rate: NA
Momentum: NA
Epoch: 150
Pooling: max
Number of trainable layers of the base model: 10
Activation: softmax

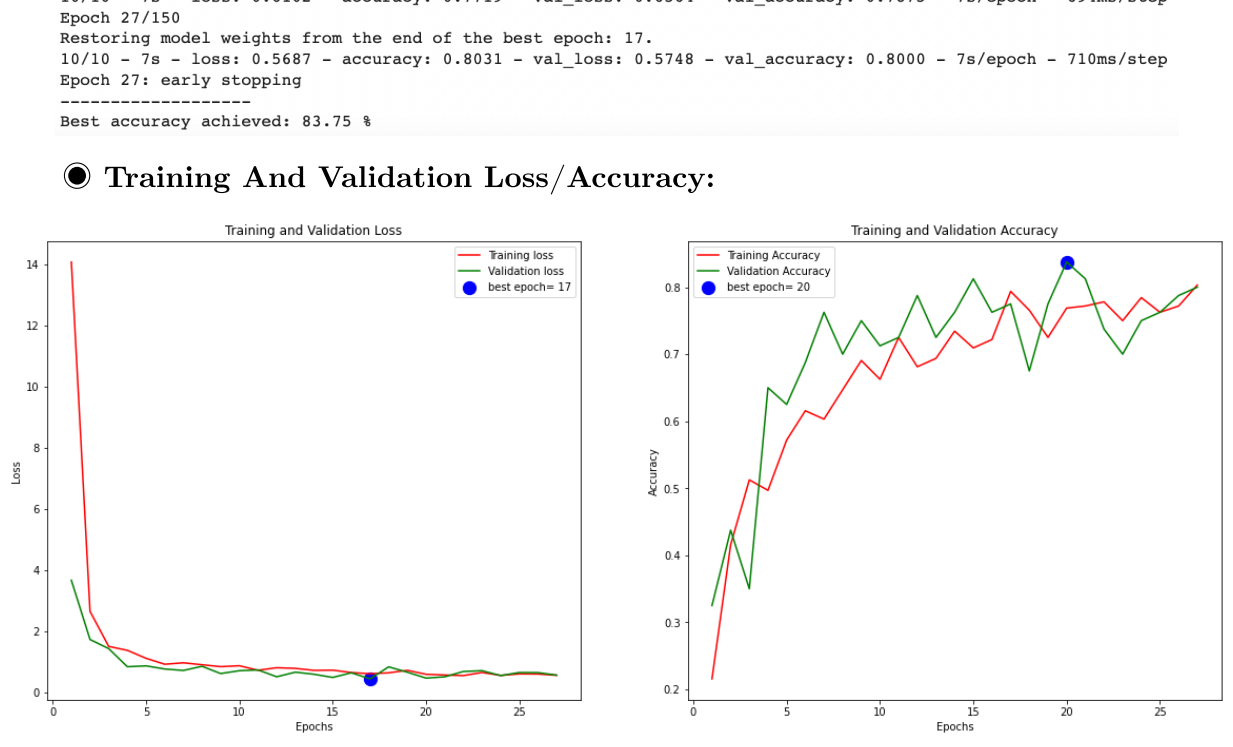
◉ **Model Summery:**

```
Model: "sequential_17"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 efficientnetb3 (Functional)  (None, 1536)             10783535

 dense_36 (Dense)            (None, 512)               786944

 dropout_18 (Dropout)        (None, 512)               0

 dense_37 (Dense)            (None, 5)                 2565

=================================================================
Total params: 11,573,044
Trainable params: 2,491,397
Non-trainable params: 9,081,647
_________________________________________________________________
```

◉ **Last 10 Epochs (with early stopping, patience=10):**

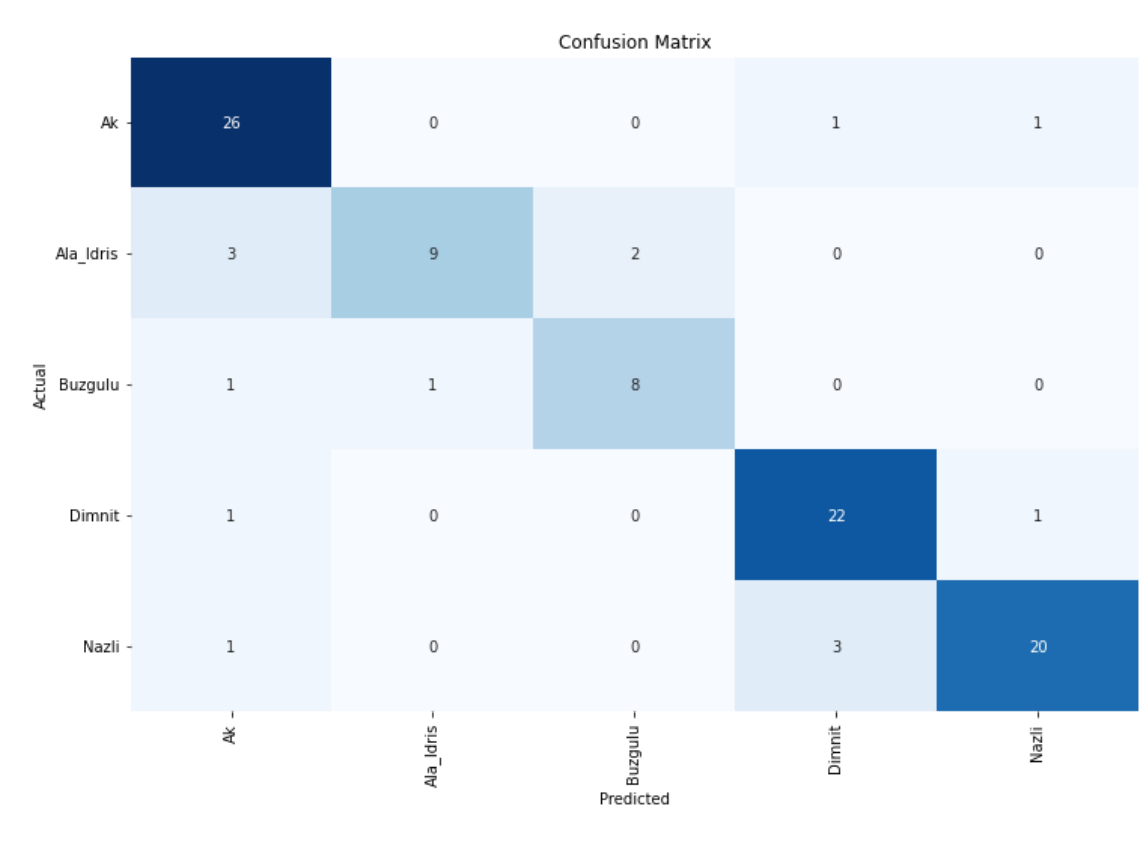
```
Epoch 17/150
10/10 - 7s - loss: 0.6197 - accuracy: 0.7937 - val_loss: 0.4523 - val_accuracy: 0.7750 - 7s/epoch - 700ms/step
Epoch 18/150
10/10 - 7s - loss: 0.6500 - accuracy: 0.7656 - val_loss: 0.8458 - val_accuracy: 0.6750 - 7s/epoch - 689ms/step
Epoch 19/150
10/10 - 7s - loss: 0.7314 - accuracy: 0.7250 - val_loss: 0.6698 - val_accuracy: 0.7750 - 7s/epoch - 688ms/step
Epoch 20/150
10/10 - 7s - loss: 0.6023 - accuracy: 0.7688 - val_loss: 0.4776 - val_accuracy: 0.8375 - 7s/epoch - 686ms/step
Epoch 21/150
10/10 - 7s - loss: 0.5786 - accuracy: 0.7719 - val_loss: 0.5189 - val_accuracy: 0.8125 - 7s/epoch - 686ms/step
Epoch 22/150
10/10 - 7s - loss: 0.5599 - accuracy: 0.7781 - val_loss: 0.6936 - val_accuracy: 0.7375 - 7s/epoch - 694ms/step
Epoch 23/150
10/10 - 7s - loss: 0.6600 - accuracy: 0.7500 - val_loss: 0.7200 - val_accuracy: 0.7000 - 7s/epoch - 690ms/step
Epoch 24/150
10/10 - 7s - loss: 0.5661 - accuracy: 0.7844 - val_loss: 0.5588 - val_accuracy: 0.7500 - 7s/epoch - 690ms/step
Epoch 25/150
10/10 - 7s - loss: 0.6146 - accuracy: 0.7625 - val_loss: 0.6604 - val_accuracy: 0.7625 - 7s/epoch - 691ms/step
Epoch 26/150
10/10 - 7s - loss: 0.6102 - accuracy: 0.7719 - val_loss: 0.6564 - val_accuracy: 0.7875 - 7s/epoch - 694ms/step
Epoch 27/150
Restoring model weights from the end of the best epoch: 17.
10/10 - 7s - loss: 0.5687 - accuracy: 0.8031 - val_loss: 0.5748 - val_accuracy: 0.8000 - 7s/epoch - 710ms/step
Epoch 27: early stopping
-------------------
Best accuracy achieved: 83.75 %
```

◉ **Training And Validation Loss/Accuracy:**



Slightly underfitting.

◉ **Confusion Matrix :**



◉ **Classification Report:**

```
Classification Report:
----------------------
              precision    recall  f1-score   support

          Ak     0.8125    0.9286    0.8667        28
   Ala_Idris     0.9000    0.6429    0.7500        14
     Buzgulu     0.8000    0.8000    0.8000        10
      Dimnit     0.8462    0.9167    0.8800        24
       Nazli     0.9091    0.8333    0.8696        24

    accuracy                         0.8500       100
   macro avg     0.8535    0.8243    0.8332       100
weighted avg     0.8548    0.8500    0.8476       100
```

◬Third Attempt◬

Add Batch Normalization
Add Regularizes
Use Adamax for optimization

Validation Accuracy = 95%

Test Accuracy = 89.00%

Since there is hope for EfficientNetB3, we try to change the CNN architecture to achieve better accuracy.

◉ **Parameters:**

Optimization Algorithm: adamax
Learning Rate: 0.001
Momentum: 0.99
Epoch: 150
Pooling: max
Number of trainable layers of the base model: 10
Activation: softmax

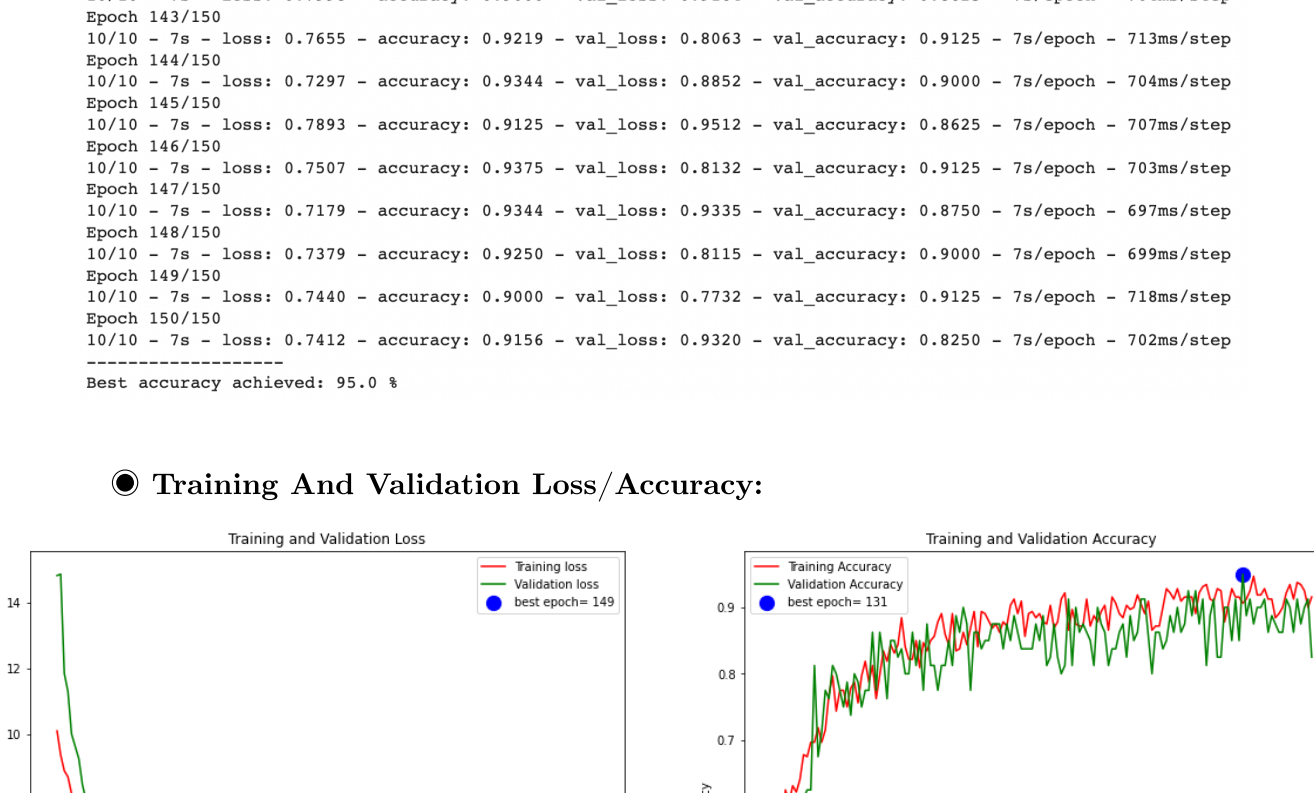
◉ **Model Summery:**

```
 block7b_project_conv (Conv2D)  (None, 8, 8, 384)    884736      ['block7b_se_excite[0][0]']

 block7b_project_bn (BatchNorma  (None, 8, 8, 384)   1536        ['block7b_project_conv[0][0]']
 lization)

 block7b_drop (Dropout)         (None, 8, 8, 384)    0           ['block7b_project_bn[0][0]']

 block7b_add (Add)              (None, 8, 8, 384)    0           ['block7b_drop[0][0]',
                                                                  'block7a_project_bn[0][0]']

 top_conv (Conv2D)              (None, 8, 8, 1536)   589824      ['block7b_add[0][0]']

 top_bn (BatchNormalization)    (None, 8, 8, 1536)   6144        ['top_conv[0][0]']

 top_activation (Activation)    (None, 8, 8, 1536)   0           ['top_bn[0][0]']

 max_pool (GlobalMaxPooling2D)  (None, 1536)         0           ['top_activation[0][0]']

 batch_normalization_2 (BatchNo  (None, 1536)        6144        ['max_pool[0][0]']
 rmalization)

 dense_42 (Dense)               (None, 256)          393472      ['batch_normalization_2[0][0]']

 dropout_21 (Dropout)           (None, 256)          0           ['dense_42[0][0]']

 dense_43 (Dense)               (None, 5)            1285        ['dropout_21[0][0]']

==================================================================================================
Total params: 11,184,436
Trainable params: 2,099,717
Non-trainable params: 9,084,719
__________________________________________________________________________________________________
```

◉ **Last 10 Epochs (with early stopping, patience=10):**

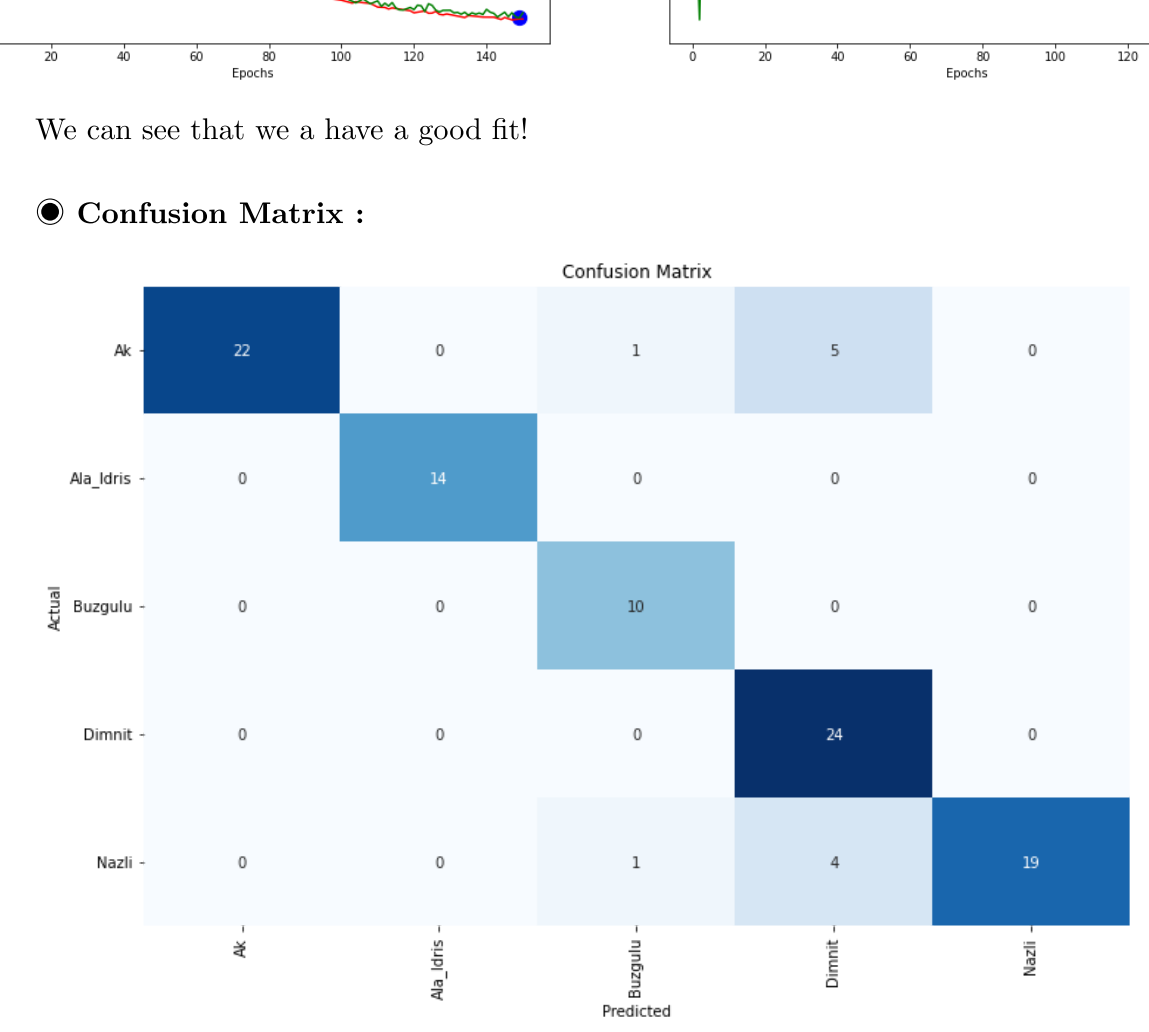
```
Epoch 140/150
10/10 - 7s - loss: 0.8032 - accuracy: 0.8844 - val_loss: 1.0388 - val_accuracy: 0.8750 - 7s/epoch - 703ms/step
Epoch 141/150
10/10 - 7s - loss: 0.8332 - accuracy: 0.8906 - val_loss: 0.9523 - val_accuracy: 0.8625 - 7s/epoch - 710ms/step
Epoch 142/150
10/10 - 7s - loss: 0.7998 - accuracy: 0.9000 - val_loss: 0.9164 - val_accuracy: 0.8625 - 7s/epoch - 704ms/step
Epoch 143/150
10/10 - 7s - loss: 0.7655 - accuracy: 0.9219 - val_loss: 0.8063 - val_accuracy: 0.9125 - 7s/epoch - 713ms/step
Epoch 144/150
10/10 - 7s - loss: 0.7297 - accuracy: 0.9344 - val_loss: 0.8852 - val_accuracy: 0.9000 - 7s/epoch - 704ms/step
Epoch 145/150
10/10 - 7s - loss: 0.7893 - accuracy: 0.9125 - val_loss: 0.9512 - val_accuracy: 0.8625 - 7s/epoch - 707ms/step
Epoch 146/150
10/10 - 7s - loss: 0.7507 - accuracy: 0.9375 - val_loss: 0.8132 - val_accuracy: 0.9125 - 7s/epoch - 703ms/step
Epoch 147/150
10/10 - 7s - loss: 0.7179 - accuracy: 0.9344 - val_loss: 0.9335 - val_accuracy: 0.8750 - 7s/epoch - 697ms/step
Epoch 148/150
10/10 - 7s - loss: 0.7379 - accuracy: 0.9250 - val_loss: 0.8115 - val_accuracy: 0.9000 - 7s/epoch - 699ms/step
Epoch 149/150
10/10 - 7s - loss: 0.7440 - accuracy: 0.9000 - val_loss: 0.7732 - val_accuracy: 0.9125 - 7s/epoch - 718ms/step
Epoch 150/150
10/10 - 7s - loss: 0.7412 - accuracy: 0.9156 - val_loss: 0.9320 - val_accuracy: 0.8250 - 7s/epoch - 702ms/step
-------------------
Best accuracy achieved: 95.0 %
```

◉ **Training And Validation Loss/Accuracy:**



We can see that we a have a good fit!

◉ **Confusion Matrix :**



◉ **Classification Report:**

```
Classification Report:
----------------------
              precision    recall  f1-score   support

          Ak     1.0000    0.7857    0.8800        28
   Ala_Idris     1.0000    1.0000    1.0000        14
     Buzgulu     0.8333    1.0000    0.9091        10
      Dimnit     0.7273    1.0000    0.8421        24
       Nazli     1.0000    0.7917    0.8837        24

    accuracy                         0.8900       100
   macro avg     0.9121    0.9155    0.9030       100
weighted avg     0.9179    0.8900    0.8915       100
```

◬Summary of EfficientNetB3◬

Best Parameters:

Optimization Algorithm: adamax
Learning Rate: 0.001
Momentum: 0.99
Epoch: 150
Pooling: max
Number of trainable layers of the base model: 10
Activation: softmax

Best Validation Accuracy: 95%

Best Test Accuracy: 89.00%

Best CNN Architecture:

```
 block7b_project_conv (Conv2D)  (None, 8, 8, 384)    884736      ['block7b_se_excite[0][0]']

 block7b_project_bn (BatchNorma  (None, 8, 8, 384)   1536        ['block7b_project_conv[0][0]']
 lization)

 block7b_drop (Dropout)         (None, 8, 8, 384)    0           ['block7b_project_bn[0][0]']

 block7b_add (Add)              (None, 8, 8, 384)    0           ['block7b_drop[0][0]',
                                                                  'block7a_project_bn[0][0]']

 top_conv (Conv2D)              (None, 8, 8, 1536)   589824      ['block7b_add[0][0]']

 top_bn (BatchNormalization)    (None, 8, 8, 1536)   6144        ['top_conv[0][0]']

 top_activation (Activation)    (None, 8, 8, 1536)   0           ['top_bn[0][0]']

 max_pool (GlobalMaxPooling2D)  (None, 1536)         0           ['top_activation[0][0]']

 batch_normalization_2 (BatchNo  (None, 1536)        6144        ['max_pool[0][0]']
 rmalization)

 dense_42 (Dense)               (None, 256)          393472      ['batch_normalization_2[0][0]']

 dropout_21 (Dropout)           (None, 256)          0           ['dense_42[0][0]']

 dense_43 (Dense)               (None, 5)            1285        ['dropout_21[0][0]']

==================================================================================================
Total params: 11,184,436
Trainable params: 2,099,717
Non-trainable params: 9,084,719
__________________________________________________________________________________________________
```

Notes:

As we can see, EfficientNetB3 performed really well and we would further fine tune this model to achieve the maximum accuracy possible in Part 5.

2. Pre-trained Convolutional Neural Network -Transfer Learning-

2.4. InceptionV3

◬First Attempt◬

Validation Accuracy = 83.75%

Test Accuracy = 79.00%

◉ **Parameters:**

Optimization Algorithm: adamax
Learning Rate: 0.001
Momentum: 0.99
Epoch: 150
Pooling: max
Number of trainable layers of the base model: 0
Activation: softmax

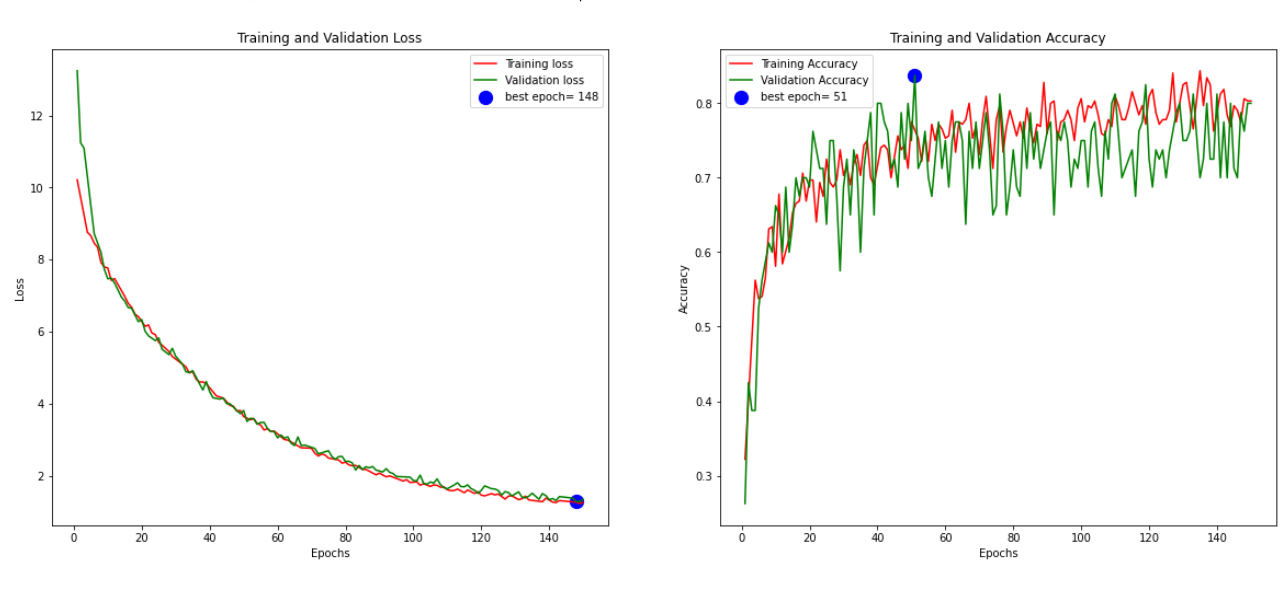
◉ **Model Summery:**

```
activation_273 (Activation)    (None, 5, 5, 320)    0          ['batch_normalization_277[0][0]']

mixed9_1 (Concatenate)         (None, 5, 5, 768)    0          ['activation_275[0][0]',
                                                                'activation_276[0][0]']

concatenate_5 (Concatenate)    (None, 5, 5, 768)    0          ['activation_279[0][0]',
                                                                'activation_280[0][0]']

activation_281 (Activation)    (None, 5, 5, 192)    0          ['batch_normalization_285[0][0]']

mixed10 (Concatenate)          (None, 5, 5, 2048)   0          ['activation_273[0][0]',
                                                                'mixed9_1[0][0]',
                                                                'concatenate_5[0][0]',
                                                                'activation_281[0][0]']

global_max_pooling2d_2 (Global  (None, 2048)        0          ['mixed10[0][0]']
MaxPooling2D)

batch_normalization_286 (Batch  (None, 2048)        8192       ['global_max_pooling2d_2[0][0]']
Normalization)

dense_46 (Dense)               (None, 256)          524544     ['batch_normalization_286[0][0]']

dropout_23 (Dropout)           (None, 256)          0          ['dense_46[0][0]']

dense_47 (Dense)               (None, 5)            1285       ['dropout_23[0][0]']

==================================================================================================
Total params: 22,336,805
Trainable params: 529,925
Non-trainable params: 21,806,880
```

◉ **Last 10 Epochs (with early stopping, patience=10):**

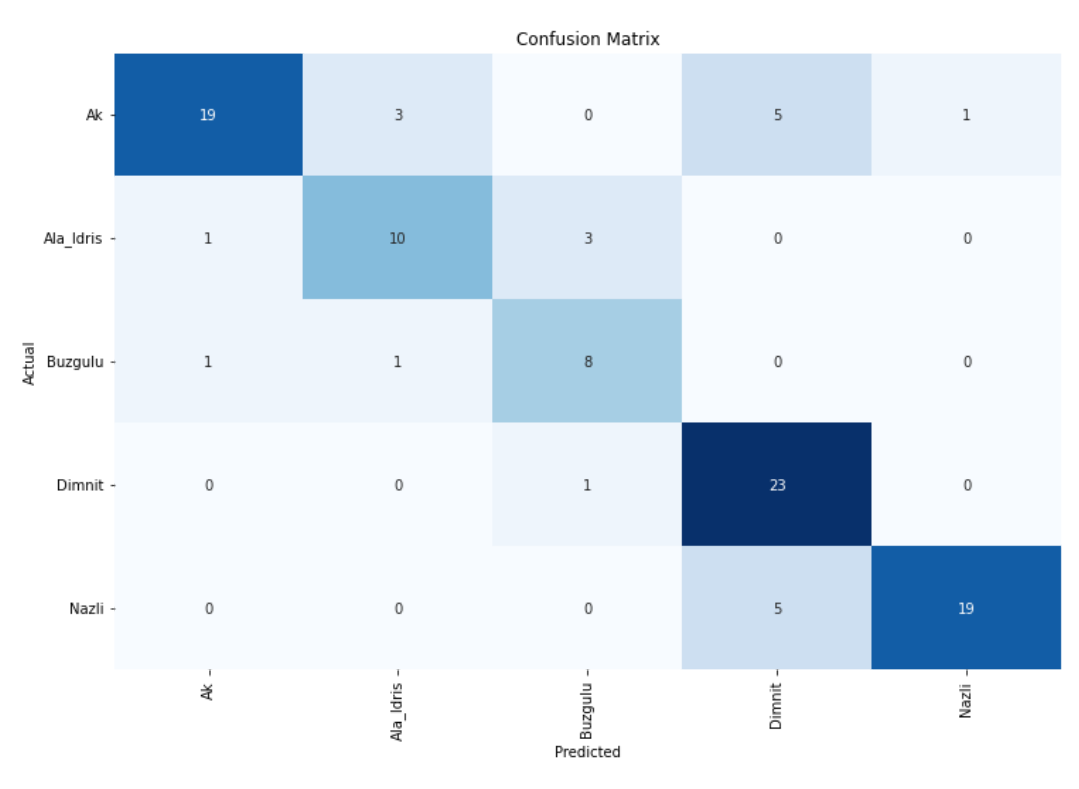
```
Epoch 140/150
10/10 - 7s - loss: 1.3218 - accuracy: 0.7875 - val_loss: 1.3364 - val_accuracy: 0.8125 - 7s/epoch - 688ms/step
Epoch 141/150
10/10 - 7s - loss: 1.2697 - accuracy: 0.8125 - val_loss: 1.3673 - val_accuracy: 0.7000 - 7s/epoch - 676ms/step
Epoch 142/150
10/10 - 7s - loss: 1.2539 - accuracy: 0.8188 - val_loss: 1.3119 - val_accuracy: 0.7750 - 7s/epoch - 693ms/step
Epoch 143/150
10/10 - 7s - loss: 1.3093 - accuracy: 0.7844 - val_loss: 1.4207 - val_accuracy: 0.7000 - 7s/epoch - 679ms/step
Epoch 144/150
10/10 - 7s - loss: 1.3023 - accuracy: 0.7688 - val_loss: 1.4110 - val_accuracy: 0.8000 - 7s/epoch - 679ms/step
Epoch 145/150
10/10 - 7s - loss: 1.2952 - accuracy: 0.7969 - val_loss: 1.4207 - val_accuracy: 0.7125 - 7s/epoch - 674ms/step
Epoch 146/150
10/10 - 7s - loss: 1.3064 - accuracy: 0.7906 - val_loss: 1.4101 - val_accuracy: 0.7000 - 7s/epoch - 674ms/step
Epoch 147/150
10/10 - 7s - loss: 1.2828 - accuracy: 0.7750 - val_loss: 1.3782 - val_accuracy: 0.7875 - 7s/epoch - 677ms/step
Epoch 148/150
10/10 - 7s - loss: 1.2646 - accuracy: 0.8062 - val_loss: 1.2809 - val_accuracy: 0.7625 - 7s/epoch - 690ms/step
Epoch 149/150
10/10 - 7s - loss: 1.2279 - accuracy: 0.8031 - val_loss: 1.2831 - val_accuracy: 0.8000 - 7s/epoch - 676ms/step
Epoch 150/150
10/10 - 7s - loss: 1.2570 - accuracy: 0.8031 - val_loss: 1.3409 - val_accuracy: 0.8000 - 7s/epoch - 672ms/step
-------------------
Best accuracy achieved: 83.75 %
```

◉ **Training And Validation Loss/Accuracy:**



The plot above is showing signs of overfitting.

◉ **Confusion Matrix:**



◉ **Classification Report:**

```
Classification Report:
----------------------
              precision    recall  f1-score   support

          Ak     0.9048    0.6786    0.7755        28
   Ala_Idris     0.7143    0.7143    0.7143        14
     Buzgulu     0.6667    0.8000    0.7273        10
      Dimnit     0.6970    0.9583    0.8070        24
       Nazli     0.9500    0.7917    0.8636        24

    accuracy                         0.7900       100
   macro avg     0.7865    0.7886    0.7775       100
weighted avg     0.8153    0.7900    0.7908       100
```

◬Second Attempt◬

Try a different CNN architecture

Increase the patience

Unfreeze more layers

Validation Accuracy = 88.75%

Test Accuracy = 84%

◉ **Parameters:**

Optimization Algorithm: adamax
Learning Rate: 0.001
Momentum: 0.99
Epoch: 150
Pooling: max
Number of trainable layers of the base model: 15
Activation: softmax

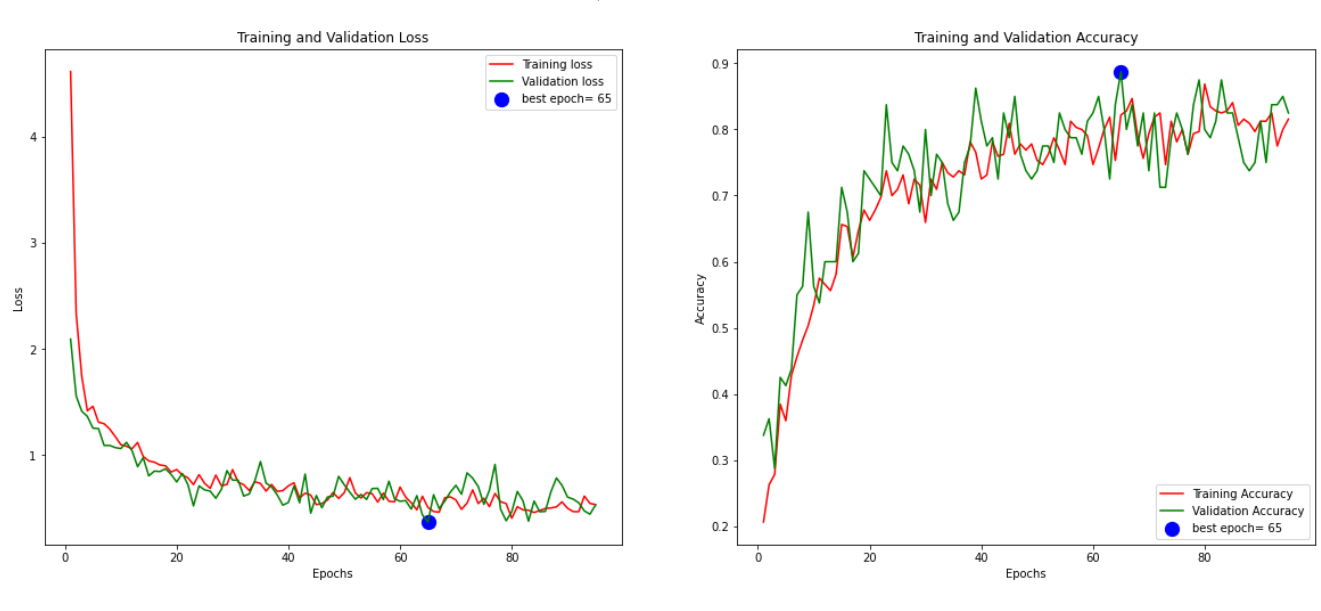
◉ **Model Summery:**

```
activation_837 (Activation)    (None, 5, 5, 320)    0          ['batch_normalization_842[0][0]']

mixed9_1 (Concatenate)         (None, 5, 5, 768)    0          ['activation_839[0][0]',
                                                                'activation_840[0][0]']

concatenate_17 (Concatenate)   (None, 5, 5, 768)    0          ['activation_843[0][0]',
                                                                'activation_844[0][0]']

activation_845 (Activation)    (None, 5, 5, 192)    0          ['batch_normalization_850[0][0]']

mixed10 (Concatenate)          (None, 5, 5, 2048)   0          ['activation_837[0][0]',
                                                                'mixed9_1[0][0]',
                                                                'concatenate_17[0][0]',
                                                                'activation_845[0][0]']

global_max_pooling2d_8 (Global  (None, 2048)        0          ['mixed10[0][0]']
MaxPooling2D)

dense_54 (Dense)               (None, 512)          1049088    ['global_max_pooling2d_8[0][0]']

dropout_28 (Dropout)           (None, 512)          0          ['dense_54[0][0]']

dense_55 (Dense)               (None, 512)          262656     ['dropout_28[0][0]']

dropout_29 (Dropout)           (None, 512)          0          ['dense_55[0][0]']

dense_56 (Dense)               (None, 5)            2565       ['dropout_29[0][0]']

==================================================================================================
Total params: 23,117,093
Trainable params: 1,708,805
Non-trainable params: 21,408,288
```

◉ **Last 10 Epochs (with early stopping, patience=30):**

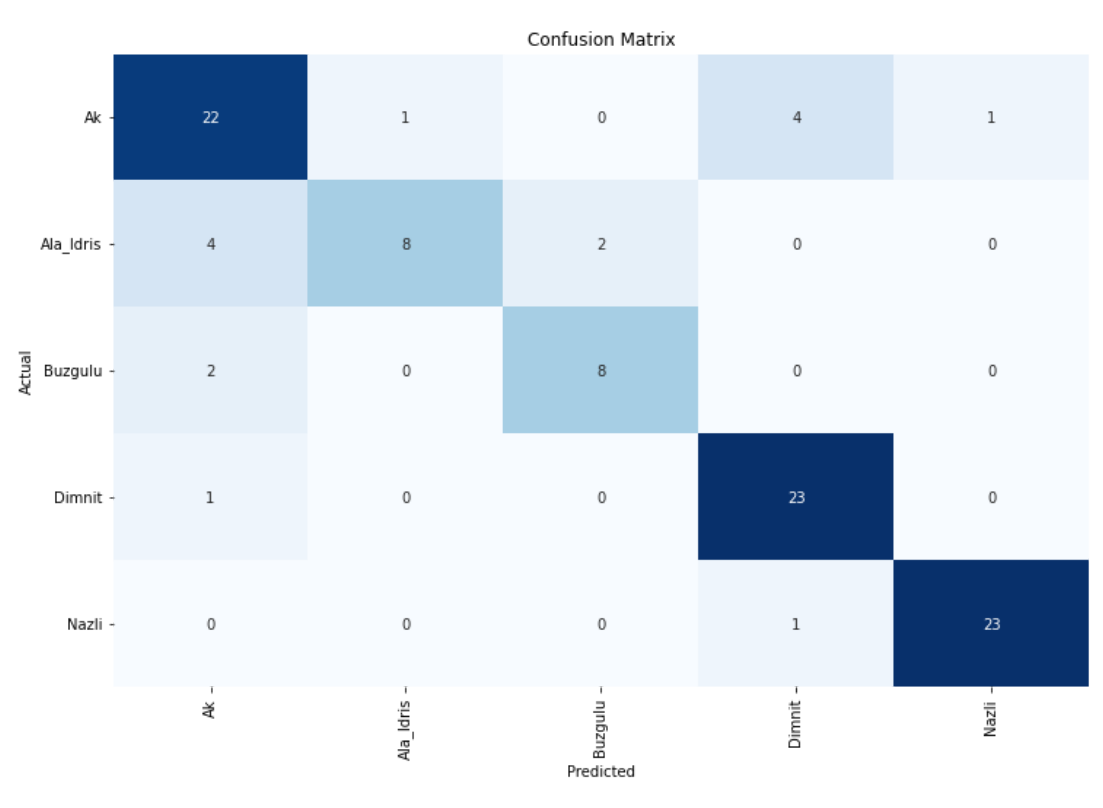
```
Epoch 85/150
10/10 - 7s - loss: 0.4801 - accuracy: 0.8406 - val_loss: 0.4688 - val_accuracy: 0.8250 - 7s/epoch - 668ms/step
Epoch 86/150
10/10 - 7s - loss: 0.4992 - accuracy: 0.8062 - val_loss: 0.4716 - val_accuracy: 0.7875 - 7s/epoch - 670ms/step
Epoch 87/150
10/10 - 7s - loss: 0.5037 - accuracy: 0.8156 - val_loss: 0.6544 - val_accuracy: 0.7500 - 7s/epoch - 671ms/step
Epoch 88/150
10/10 - 7s - loss: 0.5142 - accuracy: 0.8094 - val_loss: 0.7862 - val_accuracy: 0.7375 - 7s/epoch - 675ms/step
Epoch 89/150
10/10 - 7s - loss: 0.5613 - accuracy: 0.7969 - val_loss: 0.7166 - val_accuracy: 0.7500 - 7s/epoch - 680ms/step
Epoch 90/150
10/10 - 7s - loss: 0.5006 - accuracy: 0.8125 - val_loss: 0.6057 - val_accuracy: 0.8125 - 7s/epoch - 674ms/step
Epoch 91/150
10/10 - 7s - loss: 0.4699 - accuracy: 0.8125 - val_loss: 0.5873 - val_accuracy: 0.7500 - 7s/epoch - 672ms/step
Epoch 92/150
10/10 - 7s - loss: 0.4690 - accuracy: 0.8250 - val_loss: 0.5526 - val_accuracy: 0.8375 - 7s/epoch - 672ms/step
Epoch 93/150
10/10 - 7s - loss: 0.6160 - accuracy: 0.7750 - val_loss: 0.4766 - val_accuracy: 0.8375 - 7s/epoch - 671ms/step
Epoch 94/150
10/10 - 7s - loss: 0.5469 - accuracy: 0.8000 - val_loss: 0.4454 - val_accuracy: 0.8500 - 7s/epoch - 669ms/step
Epoch 95/150
Restoring model weights from the end of the best epoch: 65.
10/10 - 7s - loss: 0.5358 - accuracy: 0.8156 - val_loss: 0.5318 - val_accuracy: 0.8250 - 7s/epoch - 687ms/step
Epoch 95: early stopping
-------------------
Best accuracy achieved: 88.75 %
```

◉ **Training And Validation Loss/Accuracy:**



The fit is good. No under/over fitting.

◉ **Confusion Matrix :**



◉ **Classification Report:**

```
Classification Report:
----------------------
              precision    recall  f1-score   support

          Ak     0.7586    0.7857    0.7719        28
   Ala_Idris     0.8889    0.5714    0.6957        14
     Buzgulu     0.8000    0.8000    0.8000        10
      Dimnit     0.8214    0.9583    0.8846        24
       Nazli     0.9583    0.9583    0.9583        24

    accuracy                         0.8400       100
   macro avg     0.8455    0.8148    0.8221       100
weighted avg     0.8440    0.8400    0.8358       100
```

◬Third Attempt◬

Add dropouts

Add regularization

Add BatchNormalization

Validation Accuracy = 91.25%

Test Accuracy = 88%

◉ **Parameters:**

Optimization Algorithm: adamax
Learning Rate: 0.001
Momentum: 0.99
Epoch: 150
Pooling: max
Number of trainable layers of the base model: 15
Activation: softmax

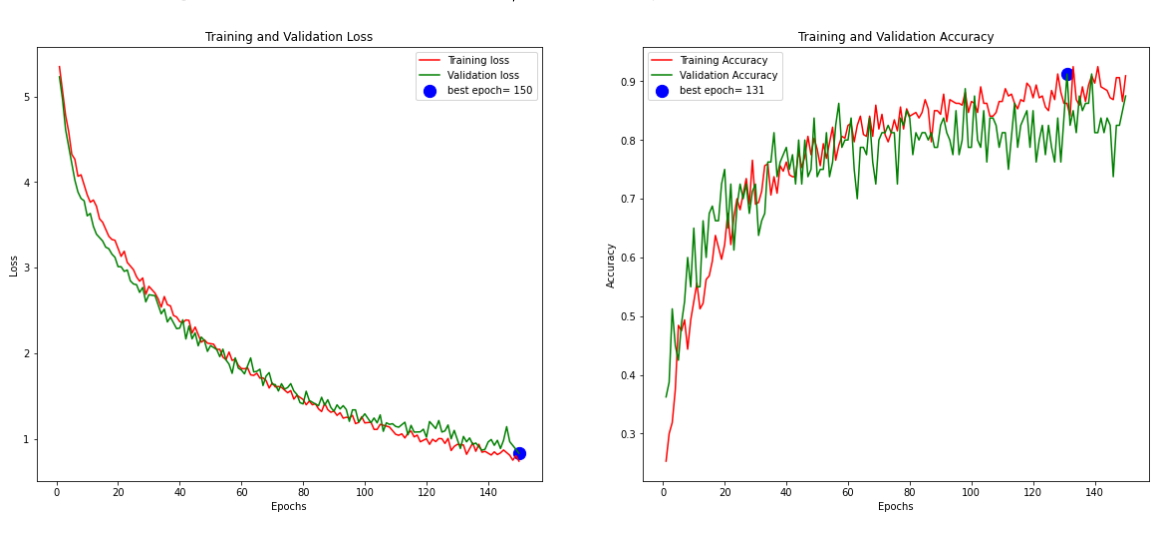
◉ **Model Summery:**

```
Model: "sequential_26"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 inception_v3 (Functional)   (None, 2048)              21802784

 batch_normalization_1699 (B  (None, 2048)             8192
 atchNormalization)

 flatten_7 (Flatten)         (None, 2048)              0

 dropout_45 (Dropout)        (None, 2048)              0

 dense_73 (Dense)            (None, 128)               262272

 dropout_46 (Dropout)        (None, 128)               0

 dense_74 (Dense)            (None, 256)               33024

 dropout_47 (Dropout)        (None, 256)               0

 dense_75 (Dense)            (None, 5)                 1285

=================================================================
Total params: 22,107,557
Trainable params: 695,173
Non-trainable params: 21,412,384
_________________________________________________________________
```

◉ **Last 10 Epochs (with early stopping, patience=30):**

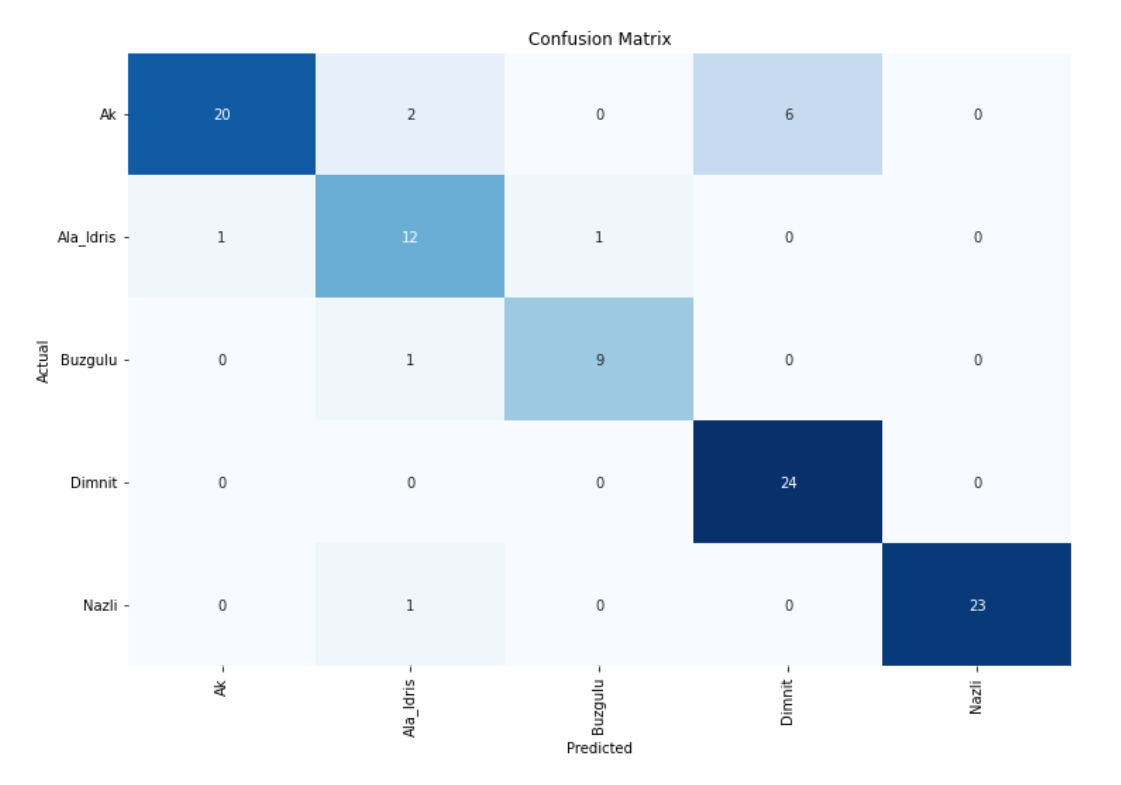
```
Epoch 140/150
10/10 - 7s - loss: 0.8259 - accuracy: 0.8969 - val_loss: 0.9617 - val_accuracy: 0.8125 - 7s/epoch - 673ms/step
Epoch 141/150
10/10 - 7s - loss: 0.8093 - accuracy: 0.9250 - val_loss: 0.9908 - val_accuracy: 0.8125 - 7s/epoch - 670ms/step
Epoch 142/150
10/10 - 7s - loss: 0.8463 - accuracy: 0.8906 - val_loss: 0.9226 - val_accuracy: 0.8375 - 7s/epoch - 669ms/step
Epoch 143/150
10/10 - 7s - loss: 0.8144 - accuracy: 0.8875 - val_loss: 0.9813 - val_accuracy: 0.8125 - 7s/epoch - 669ms/step
Epoch 144/150
10/10 - 7s - loss: 0.8345 - accuracy: 0.8844 - val_loss: 0.8856 - val_accuracy: 0.8375 - 7s/epoch - 672ms/step
Epoch 145/150
10/10 - 7s - loss: 0.8683 - accuracy: 0.8719 - val_loss: 0.9819 - val_accuracy: 0.8250 - 7s/epoch - 673ms/step
Epoch 146/150
10/10 - 7s - loss: 0.8399 - accuracy: 0.8687 - val_loss: 1.1414 - val_accuracy: 0.7375 - 7s/epoch - 677ms/step
Epoch 147/150
10/10 - 7s - loss: 0.8067 - accuracy: 0.9062 - val_loss: 0.9654 - val_accuracy: 0.8250 - 7s/epoch - 674ms/step
Epoch 148/150
10/10 - 7s - loss: 0.7491 - accuracy: 0.9062 - val_loss: 0.9258 - val_accuracy: 0.8250 - 7s/epoch - 673ms/step
Epoch 149/150
10/10 - 7s - loss: 0.8016 - accuracy: 0.8656 - val_loss: 0.8761 - val_accuracy: 0.8500 - 7s/epoch - 675ms/step
Epoch 150/150
10/10 - 7s - loss: 0.7378 - accuracy: 0.9094 - val_loss: 0.8290 - val_accuracy: 0.8750 - 7s/epoch - 685ms/step
-------------------
Best accuracy achieved: 91.25 %
```

◉ **Training And Validation Loss/Accuracy:**



We can see the model begin to overfit.

◉ **Confusion Matrix :**



◉ **Classification Report:**

```
Classification Report:
----------------------
              precision    recall  f1-score   support

          Ak     0.9524    0.7143    0.8163        28
   Ala_Idris     0.7500    0.8571    0.8000        14
     Buzgulu     0.9000    0.9000    0.9000        10
      Dimnit     0.8000    1.0000    0.8889        24
       Nazli     1.0000    0.9583    0.9787        24

    accuracy                         0.8800       100
   macro avg     0.8805    0.8860    0.8768       100
weighted avg     0.8937    0.8800    0.8788       100
```

◬Summary of InceptionV3◬

Best Parameters:

Optimization Algorithm: adamax
Learning Rate: 0.001
Momentum: 0.99
Epoch: 150
Pooling: max
Number of trainable layers of the base model: 15
Activation: softmax

Best Validation Accuracy: 91.25%

Best Test Accuracy: 88%

Best CNN Architecture:

```
Model: "sequential_26"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 inception_v3 (Functional)   (None, 2048)              21802784

 batch_normalization_1699 (B  (None, 2048)             8192
 atchNormalization)

 flatten_7 (Flatten)         (None, 2048)              0

 dropout_45 (Dropout)        (None, 2048)              0

 dense_73 (Dense)            (None, 128)               262272

 dropout_46 (Dropout)        (None, 128)               0

 dense_74 (Dense)            (None, 256)               33024

 dropout_47 (Dropout)        (None, 256)               0

 dense_75 (Dense)            (None, 5)                 1285

=================================================================
Total params: 22,107,557
Trainable params: 695,173
Non-trainable params: 21,412,384
_________________________________________________________________
```

Notes:

InceptionV3 is our second best model so I will try to fine tune it in part 5.

3. Autoencoders

Autoencoder is a type of neural network that can be used to learn a **compressed** representation of raw data. The model will take all of the input columns, then output the same values. It will learn to recreate the input pattern exactly.
An autoencoder is composed of an **encoder** and a **decoder** sub-models.
The encoder learns how to interpret the input and compress it to an internal representation defined by the **bottleneck layer**. The decoder takes the output of the encoder (the bottleneck layer) and attempts to recreate the input.
Once the autoencoder is trained, the decoder is discarded and we only keep the encoder and use it to compress examples of input to vectors output by the bottleneck layer.

The encoder can then be used as a data preparation technique to perform **feature extraction** on raw data that can be used to train a different machine learning model.

3.1. Autoencoder For Dimensionality Reduction

Here we first train the autoencoder on the train dataset then we discard the decoder part and feed the encoder to our best performing model so far which is **EfficientNetB3.** Not only it has a high accuracy, but also it has a high number of trainable parameters.

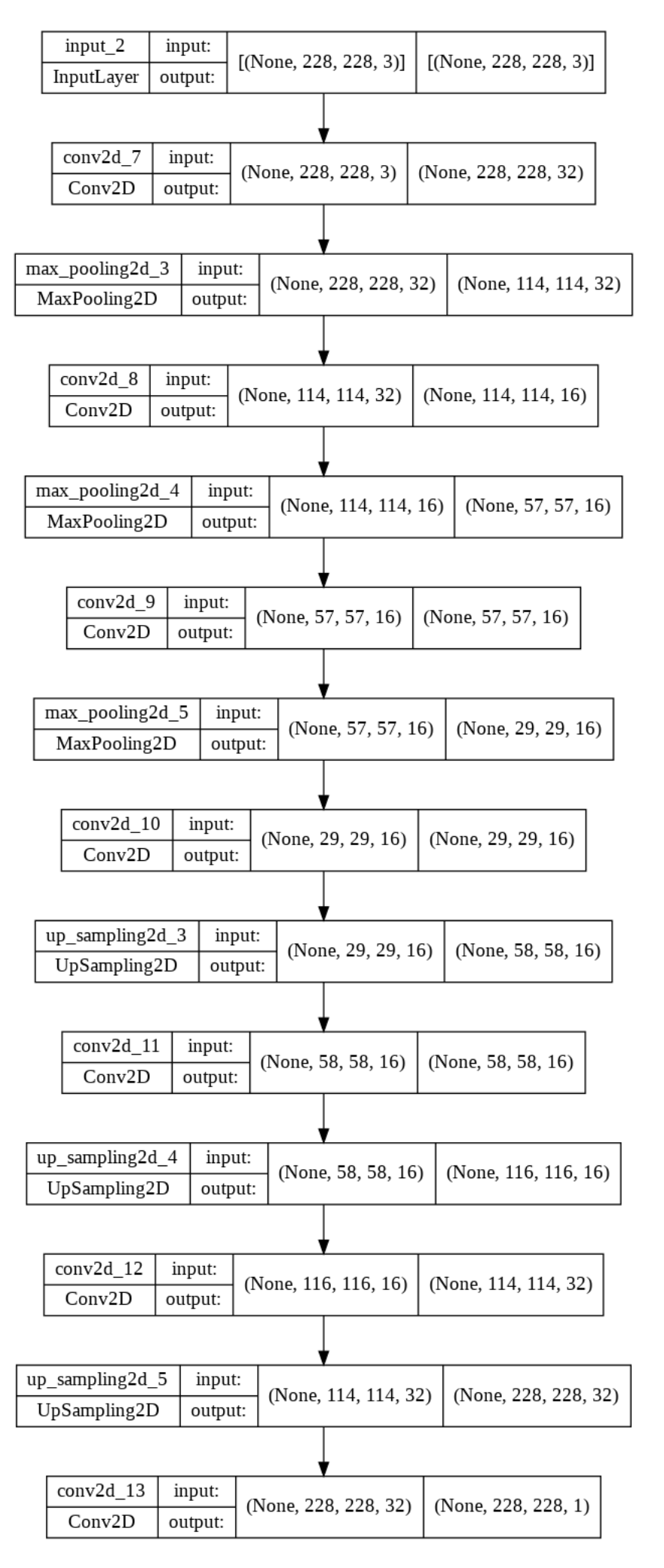
For the **encoder** we use **convolution** and **max pooling** layers and for the **decoder** part, we use **convolution** and **up sampling** layers. The aim is to train the autoencoder with minimum loss.

◬Training The Autoencoder◬

Validation Loss = 22.54%

◉ **Model Summery:**

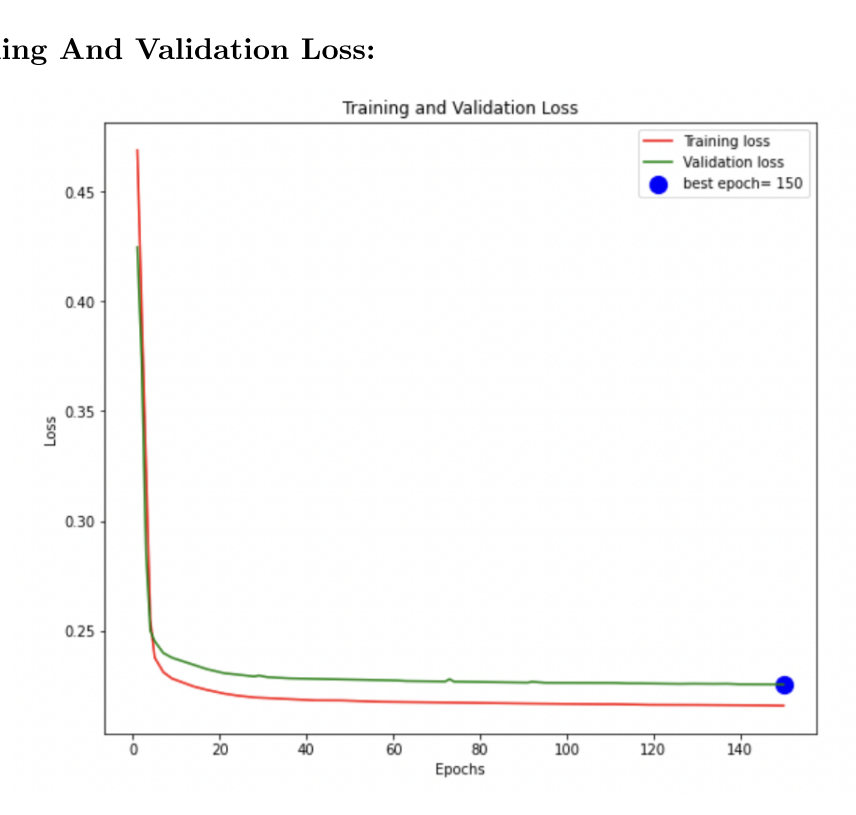
```
Model: "model_13"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_25 (InputLayer)       [(None, 228, 228, 3)]     0

 conv2d_562 (Conv2D)         (None, 228, 228, 64)      1792

 max_pooling2d_60 (MaxPoolin  (None, 114, 114, 64)     0
 g2D)

 conv2d_563 (Conv2D)         (None, 114, 114, 32)      18464

 max_pooling2d_61 (MaxPoolin  (None, 57, 57, 32)       0
 g2D)

 conv2d_564 (Conv2D)         (None, 57, 57, 16)        4624

 max_pooling2d_62 (MaxPoolin  (None, 29, 29, 16)       0
 g2D)

 conv2d_565 (Conv2D)         (None, 29, 29, 16)        2320

 up_sampling2d_40 (UpSamplin  (None, 58, 58, 16)       0
 g2D)

 conv2d_566 (Conv2D)         (None, 58, 58, 32)        4640

 up_sampling2d_41 (UpSamplin  (None, 116, 116, 32)     0
 g2D)

 conv2d_567 (Conv2D)         (None, 114, 114, 64)      18496

 up_sampling2d_42 (UpSamplin  (None, 228, 228, 64)     0
 g2D)

 conv2d_568 (Conv2D)         (None, 228, 228, 1)       577

=================================================================
Total params: 50,913
Trainable params: 50,913
Non-trainable params: 0
_________________________________________________________________
```



◉ **Last 10 Epochs (with early stopping, patience=30):**

```
Epoch 140/150
10/10 [==============================] - 3s 278ms/step - loss: 0.2160 - accuracy: 0.6584 - val_loss: 0.2255 - val_accuracy: 0.6433
Epoch 141/150
10/10 [==============================] - 3s 276ms/step - loss: 0.2160 - accuracy: 0.6584 - val_loss: 0.2258 - val_accuracy: 0.6434
Epoch 142/150
10/10 [==============================] - 3s 281ms/step - loss: 0.2160 - accuracy: 0.6584 - val_loss: 0.2255 - val_accuracy: 0.6433
Epoch 143/150
10/10 [==============================] - 3s 279ms/step - loss: 0.2160 - accuracy: 0.6584 - val_loss: 0.2256 - val_accuracy: 0.6432
Epoch 144/150
10/10 [==============================] - 3s 283ms/step - loss: 0.2159 - accuracy: 0.6584 - val_loss: 0.2258 - val_accuracy: 0.6434
Epoch 145/150
10/10 [==============================] - 3s 279ms/step - loss: 0.2159 - accuracy: 0.6584 - val_loss: 0.2255 - val_accuracy: 0.6434
Epoch 146/150
10/10 [==============================] - 3s 279ms/step - loss: 0.2159 - accuracy: 0.6584 - val_loss: 0.2255 - val_accuracy: 0.6433
Epoch 147/150
10/10 [==============================] - 3s 278ms/step - loss: 0.2159 - accuracy: 0.6584 - val_loss: 0.2255 - val_accuracy: 0.6433
Epoch 148/150
10/10 [==============================] - 3s 279ms/step - loss: 0.2158 - accuracy: 0.6584 - val_loss: 0.2255 - val_accuracy: 0.6434
Epoch 149/150
10/10 [==============================] - 3s 278ms/step - loss: 0.2158 - accuracy: 0.6584 - val_loss: 0.2255 - val_accuracy: 0.6433
Epoch 150/150
10/10 [==============================] - 3s 281ms/step - loss: 0.2158 - accuracy: 0.6584 - val_loss: 0.2254 - val_accuracy: 0.6433
```

◉ **Training And Validation Loss:**



We have slightly overfitting, more fine tuning is needed for our autoencodere to perform its best.

◬Training The Encoder+EfficientNetB3◬

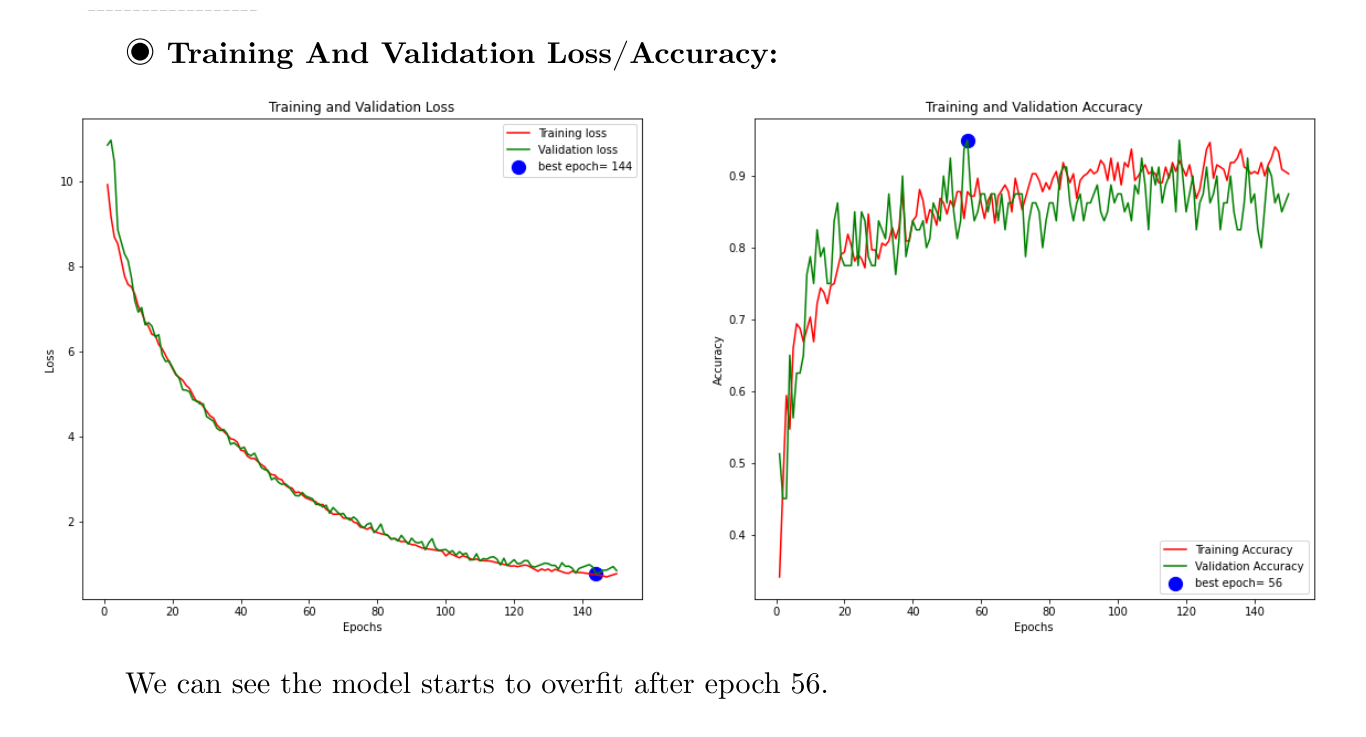
Validation Accuracy = 95%

Test Accuracy = 88%

◉ **Last 10 Epochs (with early stopping, patience=30):**

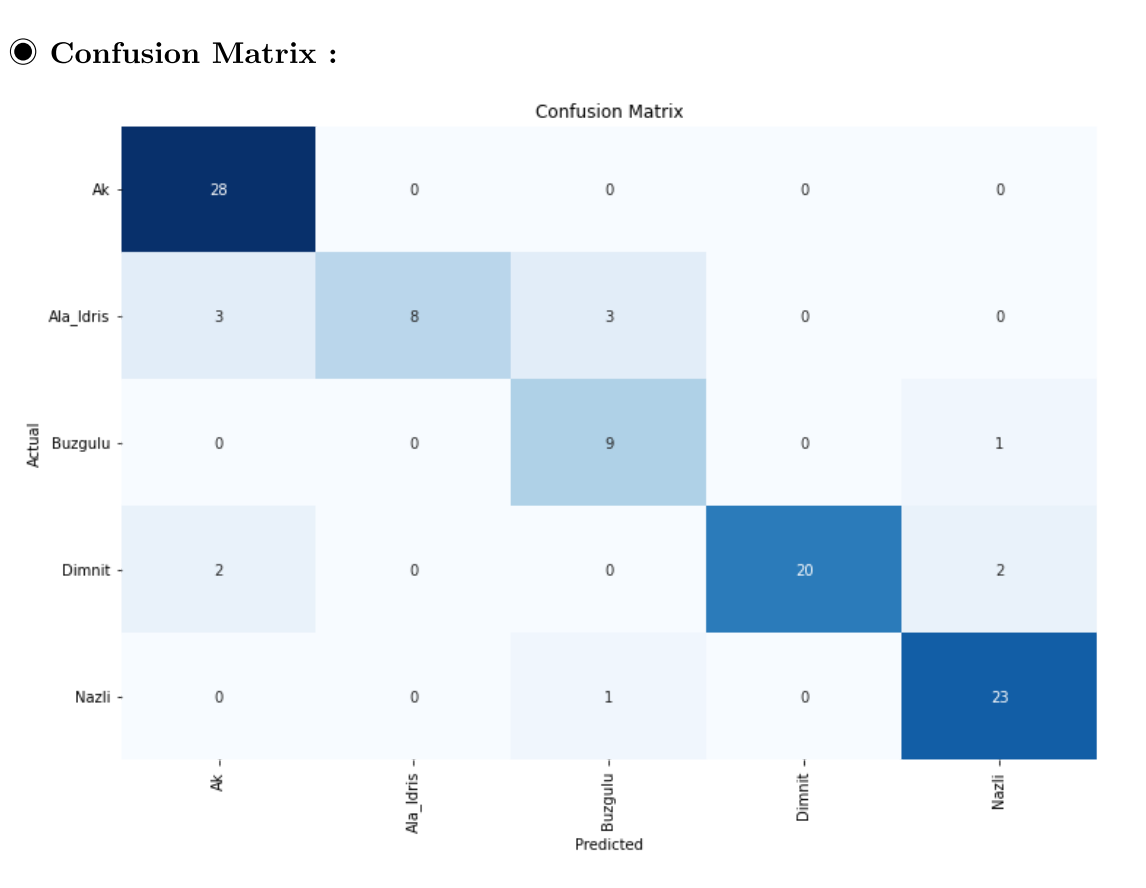
```
Epoch 140/150
10/10 [==============================] - 8s 814ms/step - loss: 0.8039 - accuracy: 0.9062 - val_loss: 0.9426 - val_accuracy: 0.8750
Epoch 141/150
10/10 [==============================] - 7s 692ms/step - loss: 0.7941 - accuracy: 0.9031 - val_loss: 0.9895 - val_accuracy: 0.8250
Epoch 142/150
10/10 [==============================] - 7s 690ms/step - loss: 0.7712 - accuracy: 0.9187 - val_loss: 0.9941 - val_accuracy: 0.8000
Epoch 143/150
10/10 [==============================] - 7s 696ms/step - loss: 0.7702 - accuracy: 0.9000 - val_loss: 0.9377 - val_accuracy: 0.8500
Epoch 144/150
10/10 [==============================] - 7s 703ms/step - loss: 0.7837 - accuracy: 0.9156 - val_loss: 0.7786 - val_accuracy: 0.9125
Epoch 145/150
10/10 [==============================] - 7s 689ms/step - loss: 0.7412 - accuracy: 0.9250 - val_loss: 0.7985 - val_accuracy: 0.9000
Epoch 146/150
10/10 [==============================] - 7s 684ms/step - loss: 0.7292 - accuracy: 0.9406 - val_loss: 0.8619 - val_accuracy: 0.8625
Epoch 147/150
10/10 [==============================] - 7s 695ms/step - loss: 0.7039 - accuracy: 0.9344 - val_loss: 0.8629 - val_accuracy: 0.8750
Epoch 148/150
10/10 [==============================] - 8s 832ms/step - loss: 0.7211 - accuracy: 0.9094 - val_loss: 0.9200 - val_accuracy: 0.8500
Epoch 149/150
10/10 [==============================] - 7s 690ms/step - loss: 0.7386 - accuracy: 0.9062 - val_loss: 0.9478 - val_accuracy: 0.8625
Epoch 150/150
10/10 [==============================] - 7s 696ms/step - loss: 0.7811 - accuracy: 0.9031 - val_loss: 0.8534 - val_accuracy: 0.8750
-------------------
Best accuracy achieved: 95.0 %
```

◉ **Training And Validation Loss/Accuracy:**



We can see the model starts to overfit after epoch 56.

◉ **Confusion Matrix :**



◉ **Classification Report:**

```
Classification Report:
----------------------
              precision    recall  f1-score   support

          Ak     0.8485    1.0000    0.9180        28
   Ala_Idris     1.0000    0.5714    0.7273        14
     Buzgulu     0.6923    0.9000    0.7826        10
      Dimnit     1.0000    0.8333    0.9091        24
       Nazli     0.8846    0.9583    0.9200        24

    accuracy                         0.8800       100
   macro avg     0.8851    0.8526    0.8514       100
weighted avg     0.8991    0.8800    0.8761       100
```

Before using autoencoder:

Validation Accuracy = 95%

Test Accuracy = 89%

After using autoencoder:

Validation Accuracy = 95%

Test Accuracy = 88%

As we can see using an autoencoder did not improve our result. This could be due to our autoencoder needing more fine tuning or the nature of our problem.

3.2. Denoising Autoencoder For Feature Extraction

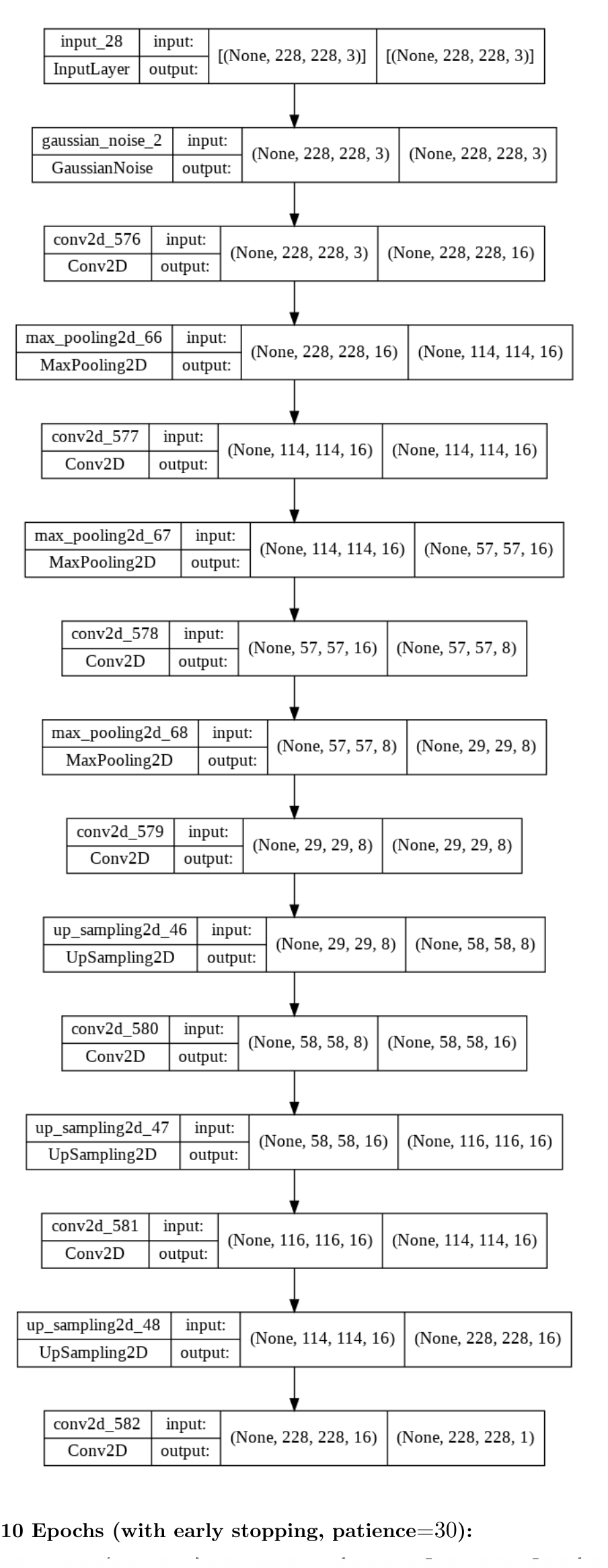
Denoising autoencoders are an extension of simple autoencoders. The hidden layers of the autoencoder learn more robust filters and reduce the risk of overfitting in the autoencoder. They also prevent the autoencoder from learning a simple identify function.
So we first add a layer of **guassian noise** to our data, train the denoising autoencoder and then attach the encoder part to top of our two best performing model so far which are **EfficientNetB3** and **InceptionV3.**

◬Training The Denoising Autoencoder◬

Validation Loss = 23.17%

◉ **Model Summery:**

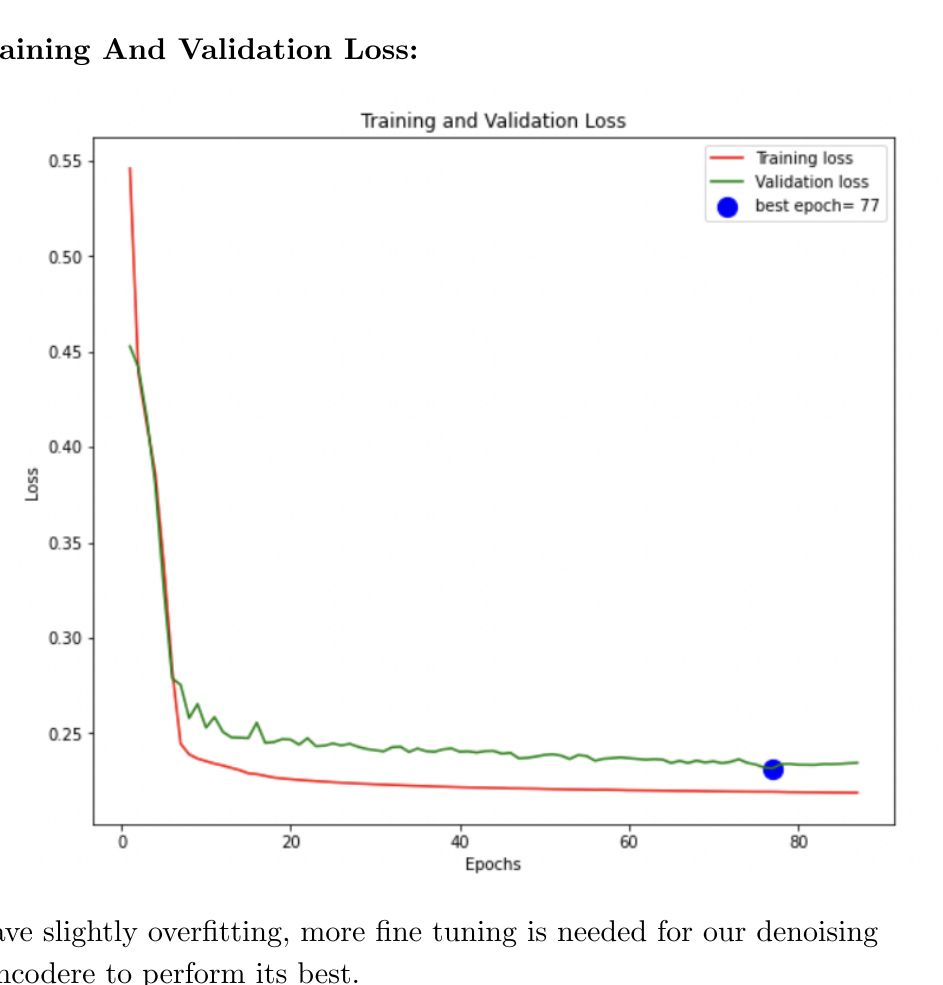
```
Model: "model_17"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_28 (InputLayer)       [(None, 228, 228, 3)]     0

 gaussian_noise_2 (GaussianN  (None, 228, 228, 3)      0
 oise)

 conv2d_576 (Conv2D)         (None, 228, 228, 16)      448

 max_pooling2d_66 (MaxPoolin  (None, 114, 114, 16)     0
 g2D)

 conv2d_577 (Conv2D)         (None, 114, 114, 16)      2320

 max_pooling2d_67 (MaxPoolin  (None, 57, 57, 16)       0
 g2D)

 conv2d_578 (Conv2D)         (None, 57, 57, 8)         1160

 max_pooling2d_68 (MaxPoolin  (None, 29, 29, 8)        0
 g2D)

 conv2d_579 (Conv2D)         (None, 29, 29, 8)         584

 up_sampling2d_46 (UpSamplin  (None, 58, 58, 8)        0
 g2D)

 conv2d_580 (Conv2D)         (None, 58, 58, 16)        1168

 up_sampling2d_47 (UpSamplin  (None, 116, 116, 16)     0
 g2D)

 conv2d_581 (Conv2D)         (None, 114, 114, 16)      2320

 up_sampling2d_48 (UpSamplin  (None, 228, 228, 16)     0
 g2D)

 conv2d_582 (Conv2D)         (None, 228, 228, 1)       145

=================================================================
Total params: 8,145
Trainable params: 8,145
Non-trainable params: 0
```



◉ **Last 10 Epochs (with early stopping, patience=30):**

```
Epoch 77/200
10/10 [==============================] - 4s 372ms/step - loss: 0.2194 - accuracy: 0.6582 - val_loss: 0.2317 - val_accuracy: 0.6433
Epoch 78/200
10/10 [==============================] - 3s 264ms/step - loss: 0.2193 - accuracy: 0.6582 - val_loss: 0.2339 - val_accuracy: 0.6434
Epoch 79/200
10/10 [==============================] - 3s 267ms/step - loss: 0.2192 - accuracy: 0.6582 - val_loss: 0.2341 - val_accuracy: 0.6434
Epoch 80/200
10/10 [==============================] - 4s 363ms/step - loss: 0.2192 - accuracy: 0.6582 - val_loss: 0.2336 - val_accuracy: 0.6434
Epoch 81/200
10/10 [==============================] - 3s 277ms/step - loss: 0.2191 - accuracy: 0.6582 - val_loss: 0.2335 - val_accuracy: 0.6434
Epoch 82/200
10/10 [==============================] - 5s 438ms/step - loss: 0.2191 - accuracy: 0.6582 - val_loss: 0.2335 - val_accuracy: 0.6434
Epoch 83/200
10/10 [==============================] - 3s 270ms/step - loss: 0.2191 - accuracy: 0.6582 - val_loss: 0.2339 - val_accuracy: 0.6434
Epoch 84/200
10/10 [==============================] - 3s 266ms/step - loss: 0.2190 - accuracy: 0.6582 - val_loss: 0.2339 - val_accuracy: 0.6434
Epoch 85/200
10/10 [==============================] - 3s 271ms/step - loss: 0.2190 - accuracy: 0.6582 - val_loss: 0.2341 - val_accuracy: 0.6434
Epoch 86/200
10/10 [==============================] - 3s 267ms/step - loss: 0.2189 - accuracy: 0.6582 - val_loss: 0.2343 - val_accuracy: 0.6434
Epoch 87/200
10/10 [==============================] - ETA: 0s - loss: 0.2189 - accuracy: 0.6582Restoring model weights from the end of the best epoch: 77.
10/10 [==============================] - 3s 268ms/step - loss: 0.2189 - accuracy: 0.6582 - val_loss: 0.2345 - val_accuracy: 0.6435
Epoch 87: early stopping
-------------------
Minimum Loss Achieved: 23.1725 %
```

◉ **Training And Validation Loss:**



We have slightly overfitting, more fine tuning is needed for our denoising autoencodere to perform its best.

◬Training The Denoising Encoder+EfficientNetB3◬

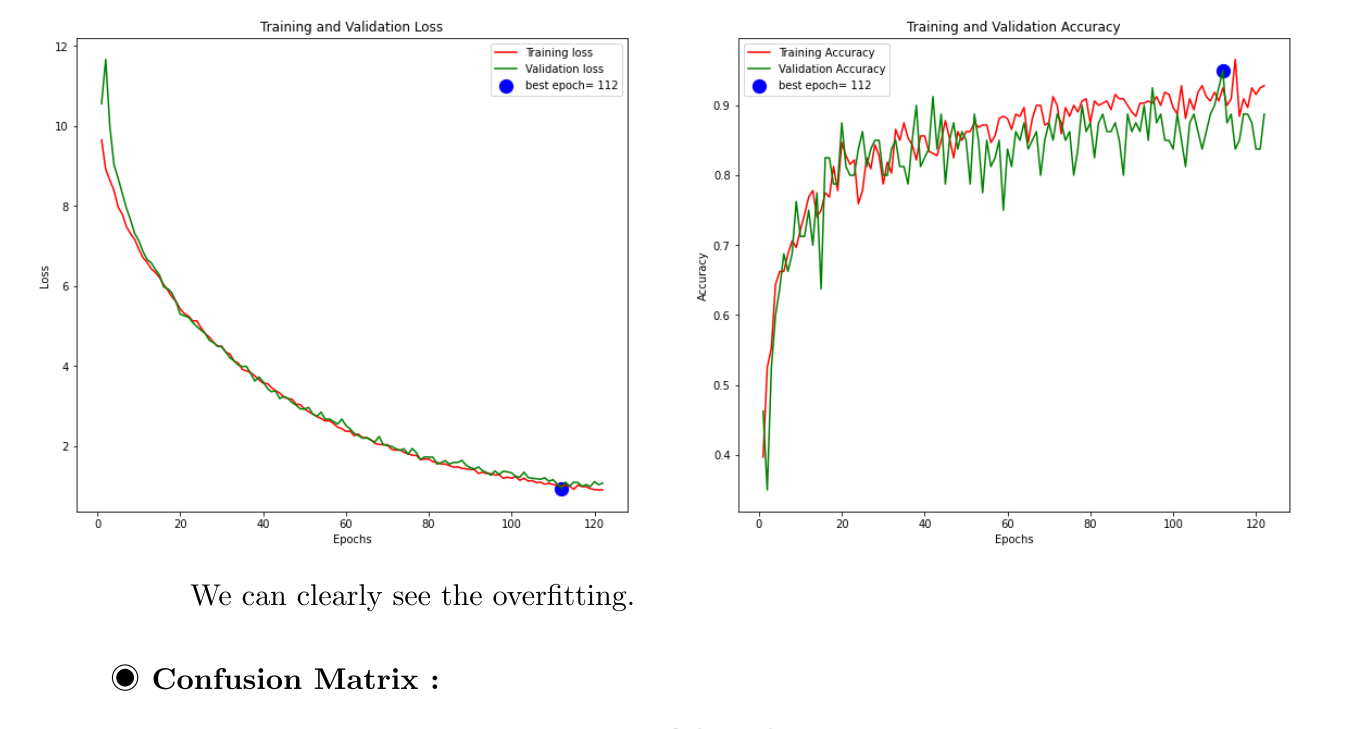
Validation Accuracy = 95%

Test Accuracy = 84%

◉ **Last 10 Epochs (with early stopping, patience=30):**

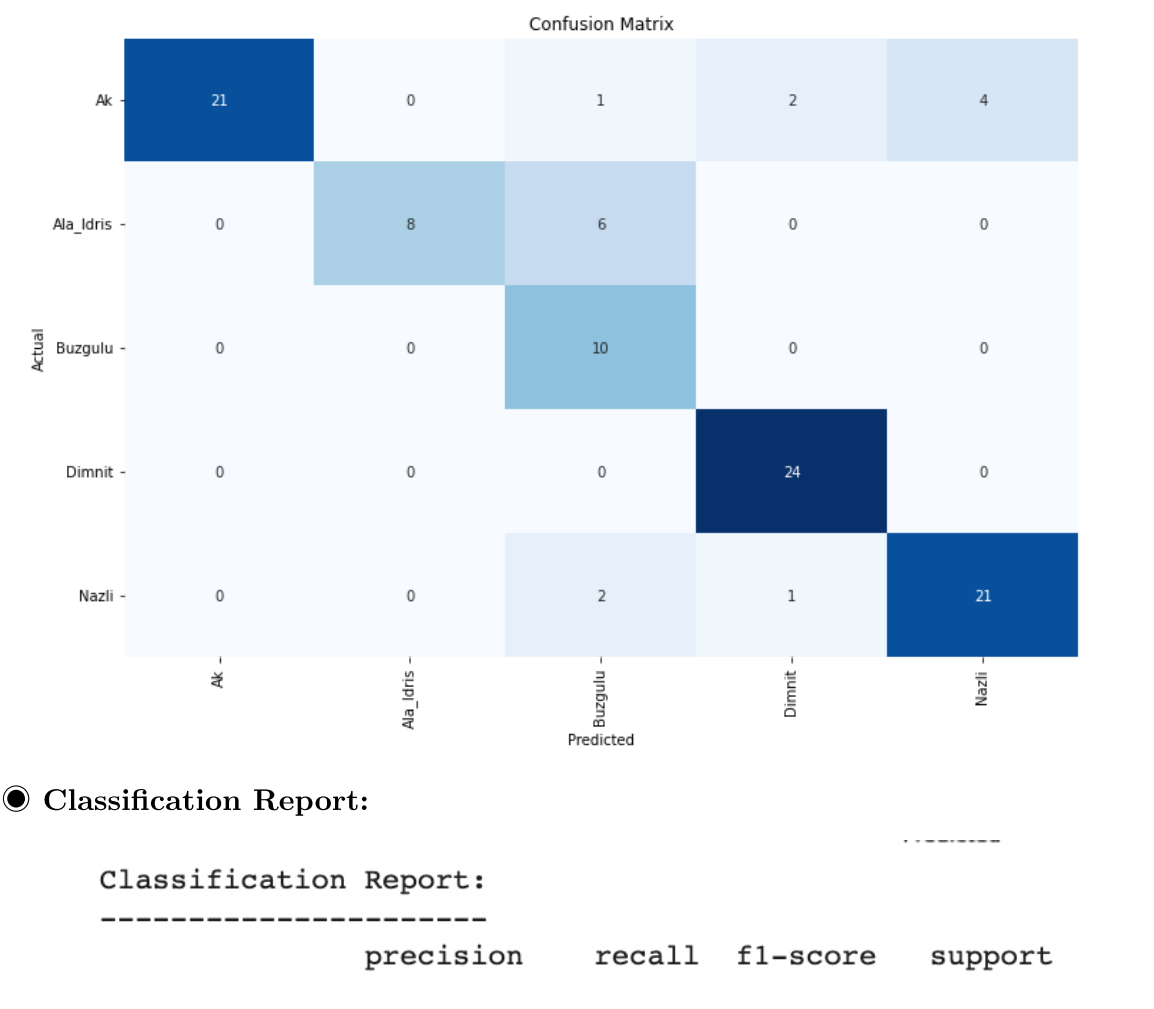
```
Epoch 112/150
10/10 [==============================] - 7s 714ms/step - loss: 1.0052 - accuracy: 0.9250 - val_loss: 0.9405 - val_accuracy: 0.9500
Epoch 113/150
10/10 [==============================] - 7s 694ms/step - loss: 1.0088 - accuracy: 0.9000 - val_loss: 1.0972 - val_accuracy: 0.8750
Epoch 114/150
10/10 [==============================] - 7s 690ms/step - loss: 0.9980 - accuracy: 0.9094 - val_loss: 0.9987 - val_accuracy: 0.8875
Epoch 115/150
10/10 [==============================] - 7s 698ms/step - loss: 0.9167 - accuracy: 0.9656 - val_loss: 1.0966 - val_accuracy: 0.8375
Epoch 116/150
10/10 [==============================] - 7s 695ms/step - loss: 1.0211 - accuracy: 0.8844 - val_loss: 1.0873 - val_accuracy: 0.8500
Epoch 117/150
10/10 [==============================] - 7s 696ms/step - loss: 0.9862 - accuracy: 0.9094 - val_loss: 0.9950 - val_accuracy: 0.8875
Epoch 118/150
10/10 [==============================] - 7s 686ms/step - loss: 0.9834 - accuracy: 0.8969 - val_loss: 1.0348 - val_accuracy: 0.8875
Epoch 119/150
10/10 [==============================] - 7s 695ms/step - loss: 0.9346 - accuracy: 0.9250 - val_loss: 0.9872 - val_accuracy: 0.8750
Epoch 120/150
10/10 [==============================] - 8s 798ms/step - loss: 0.9089 - accuracy: 0.9156 - val_loss: 1.1090 - val_accuracy: 0.8375
Epoch 121/150
10/10 [==============================] - 7s 692ms/step - loss: 0.9032 - accuracy: 0.9250 - val_loss: 1.0320 - val_accuracy: 0.8375
Epoch 122/150
10/10 [==============================] - ETA: 0s - loss: 0.9034 - accuracy: 0.9281Restoring model weights from the end of the best epoch: 112.
10/10 [==============================] - 7s 704ms/step - loss: 0.9034 - accuracy: 0.9281 - val_loss: 1.0718 - val_accuracy: 0.8875
Epoch 122: early stopping
-------------------
Best accuracy achieved: 95.0 %
```

◉ **Training And Validation Loss/Accuracy:**



We can clearly see the overfitting.

◉ **Confusion Matrix :**



◉ **Classification Report:**

```
Classification Report:
----------------------
              precision    recall  f1-score   support

          Ak     1.0000    0.7500    0.8571        28
   Ala_Idris     1.0000    0.5714    0.7273        14
     Buzgulu     0.5263    1.0000    0.6897        10
      Dimnit     0.8889    1.0000    0.9412        24
       Nazli     0.8400    0.8750    0.8571        24

    accuracy                         0.8400       100
   macro avg     0.8510    0.8393    0.8145       100
weighted avg     0.8876    0.8400    0.8424       100
```

Before using denoising autoencoder:

Validation Accuracy = 95%

Test Accuracy = 89%

After using denoising autoencoder

Validation Accuracy = 95%

Test Accuracy = 84%

Again using a denoising autoencoder did not improve our result. This could be due to our autoencoder needing more fine tuning or the nature of our problem. Actully this process caused our model to overfit.

4. 10 Fold Cross Validation

Cross-validation is a statistical method used to estimate the skill of machine learning models.
In 10 fold cross validation, we shuffle and split the training set into 10 parts and take one of those parts for validation and the rest for training. Thus we train our model 10 times on 10 different sets of train and validation datasets. Finally we announce the average of the validation accuracies as the final accuracy of the model.

We use the **stratified cross validation** to ensure that each fold of dataset has the same proportion of observations with a given label. Simply *StratifiedKFold* is the improved version of *KFold*

We use our second to best performing models so far which is **InceptionV3** to calculate accuracy using the 10 fold cross validation. The reason we don't do this for **EfficientNetB3** is that it has a very large number of trainable parameters and doing 10 fold cross validation would probably ban me from colab :)

4.1. InceptionV3

Since cross validation is very time consuming, specially for deep layers with high epochs, I had to limit the model to only 20 epochs. Thus we don't get the 91% accuracy we had before.

Result of the 10 fold cross validation on 20 epochs are as follows:

	1st Fold	2nd Fold	3rd Fold	4th Fold	5th Fold	6th Fold	7th Fold	8th Fold	9th Fold	10th Fold	Average
InceptionV3	62.5	67.5	65.0	67.5	70.0	57.5	72.5	80.0	65.0	60.0	66.75

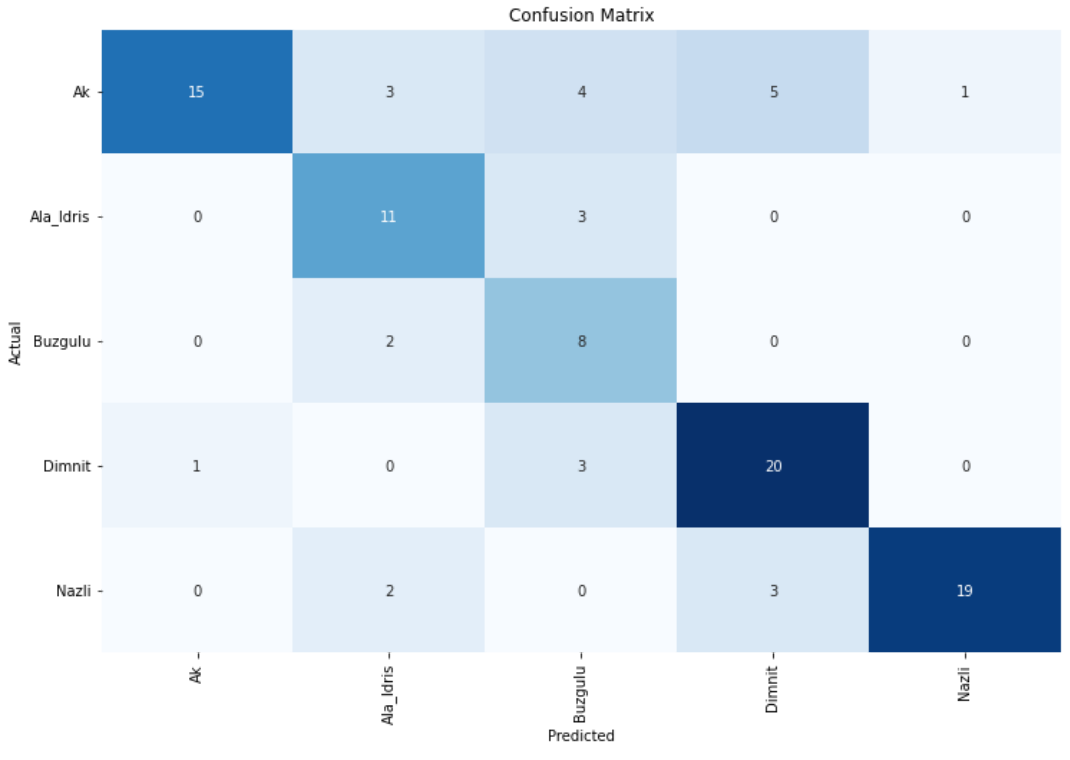
```
Validation Accuracy:
66.75% (+/- 5.84%)
Validation Loss:
3.19% (+/- 0.26%)
```

```
Best model number is 8 with validation accuracy of 80 %.
```

Running the best model on our test data set we get a test accuracy of 76%

$$\textbf{Validation Accuracy} = 80\%$$
$$\textbf{Test Accuracy} = 76\%$$

◉ **Confusion Matrix:**



◉ **Classification Report:**

```
Classification Report:
----------------------
              precision    recall  f1-score   support

          Ak     0.9375    0.5357    0.6818        28
   Ala_Idris     0.6111    0.7857    0.6875        14
     Buzgulu     0.4444    0.8000    0.5714        10
      Dimnit     0.7143    0.8333    0.7692        24
       Nazli     0.9500    0.7917    0.8636        24

    accuracy                         0.7300       100
   macro avg     0.7315    0.7493    0.7147       100
weighted avg     0.7919    0.7300    0.7362       100
```

5. Improving The Accuracy (Extra Credit)

In this section I try various techniques in hope of reaching a better test accuracy.

5.1. Unfreezing All the Pre-trained Layers

One idea is to unfreeze all layers of a pre-trianed model so it can fully learn our dataset and find the best weight for our purpose and not just the ImageNet.
We try this method with **EfficientNetB3**. To avoid overfitting we use BatchNormalization.

⚠Extra Attempt⚠

Add Batch Normalization
Add Regularizes
Use Adamax for optimization
Unfreeze all layers
Validation Accuracy = 90%
Test Accuracy = 97%

◉ **Parameters:**

Optimization Algorithm: adamax
Learning Rate: 0.001
Momentum: 0.99
Epoch: 150
Pooling: max
Number of trainable layers of the base model: all layers of EfficientNetB3
Activation: softmax

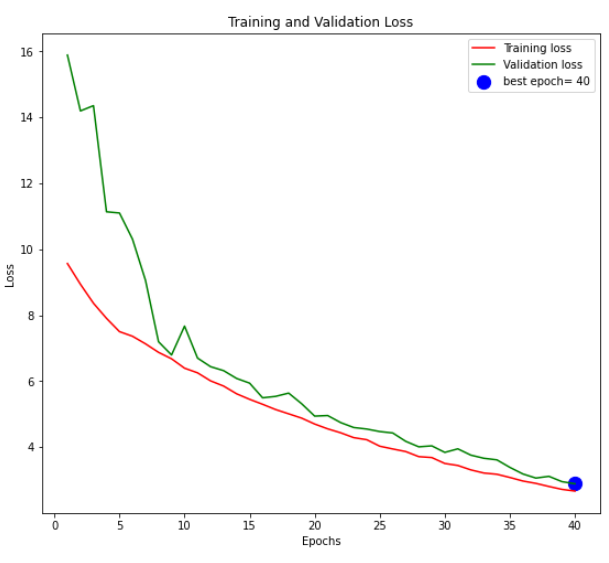
◉ **Model Summery:**

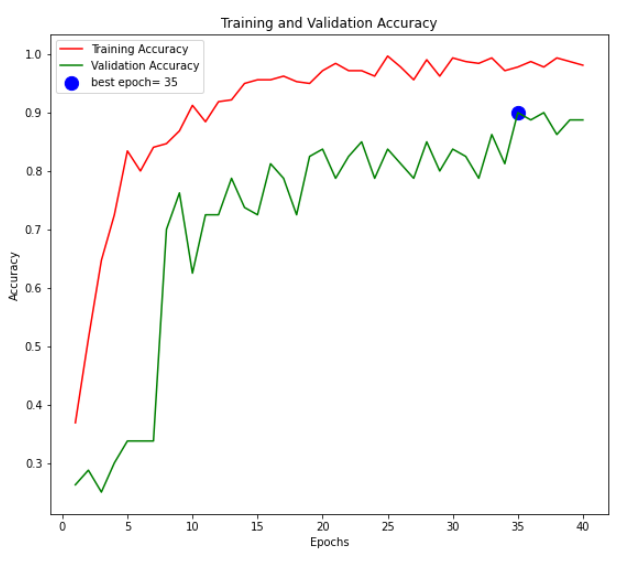
```
block7b_drop (Dropout)         (None, 8, 8, 384)    0          ['block7b_project_bn[0][0]']
block7b_add (Add)              (None, 8, 8, 384)    0          ['block7b_drop[0][0]',
                                                                'block7a_project_bn[0][0]']
top_conv (Conv2D)              (None, 8, 8, 1536)   589824     ['block7b_add[0][0]']
top_bn (BatchNormalization)    (None, 8, 8, 1536)   6144       ['top_conv[0][0]']
top_activation (Activation)    (None, 8, 8, 1536)   0          ['top_bn[0][0]']
max_pool (GlobalMaxPooling2D)  (None, 1536)         0          ['top_activation[0][0]']
batch_normalization_195 (Batch  (None, 1536)        6144       ['max_pool[0][0]']
Normalization)
dense_12 (Dense)               (None, 256)          393472     ['batch_normalization_195[0][0]']
dropout_8 (Dropout)            (None, 256)          0          ['dense_12[0][0]']
dense_13 (Dense)               (None, 5)            1285       ['dropout_8[0][0]']
==================================================================================================
Total params: 11,184,436
Trainable params: 11,094,061
Non-trainable params: 90,375
__________________________________________________________________________________________________
```

◉ **Last 10 Epochs (with early stopping, patience=10):**

```
Epoch 30/40
10/10 - 9s - loss: 3.5053 - accuracy: 0.9937 - val_loss: 3.8413 - val_accuracy: 0.8375 - 9s/epoch - 945ms/step
Epoch 31/40
10/10 - 9s - loss: 3.4445 - accuracy: 0.9875 - val_loss: 3.9489 - val_accuracy: 0.8250 - 9s/epoch - 926ms/step
Epoch 32/40
10/10 - 9s - loss: 3.3108 - accuracy: 0.9844 - val_loss: 3.7579 - val_accuracy: 0.7875 - 9s/epoch - 939ms/step
Epoch 33/40
10/10 - 9s - loss: 3.2156 - accuracy: 0.9937 - val_loss: 3.6626 - val_accuracy: 0.8625 - 9s/epoch - 936ms/step
Epoch 34/40
10/10 - 10s - loss: 3.1784 - accuracy: 0.9719 - val_loss: 3.6152 - val_accuracy: 0.8125 - 10s/epoch - 971ms/step
Epoch 35/40
10/10 - 10s - loss: 3.0806 - accuracy: 0.9781 - val_loss: 3.3866 - val_accuracy: 0.9000 - 10s/epoch - 1s/step
Epoch 36/40
10/10 - 9s - loss: 2.9736 - accuracy: 0.9875 - val_loss: 3.1875 - val_accuracy: 0.8875 - 9s/epoch - 938ms/step
Epoch 37/40
10/10 - 9s - loss: 2.9017 - accuracy: 0.9781 - val_loss: 3.0624 - val_accuracy: 0.9000 - 9s/epoch - 937ms/step
Epoch 38/40
10/10 - 9s - loss: 2.8080 - accuracy: 0.9937 - val_loss: 3.1155 - val_accuracy: 0.8625 - 9s/epoch - 928ms/step
Epoch 39/40
10/10 - 10s - loss: 2.7203 - accuracy: 0.9875 - val_loss: 2.9546 - val_accuracy: 0.8875 - 10s/epoch - 991ms/step
Epoch 40/40
10/10 - 11s - loss: 2.6686 - accuracy: 0.9812 - val_loss: 2.8894 - val_accuracy: 0.8875 - 11s/epoch - 1s/step
--------------------
Best accuracy achieved: 90.0 %
```

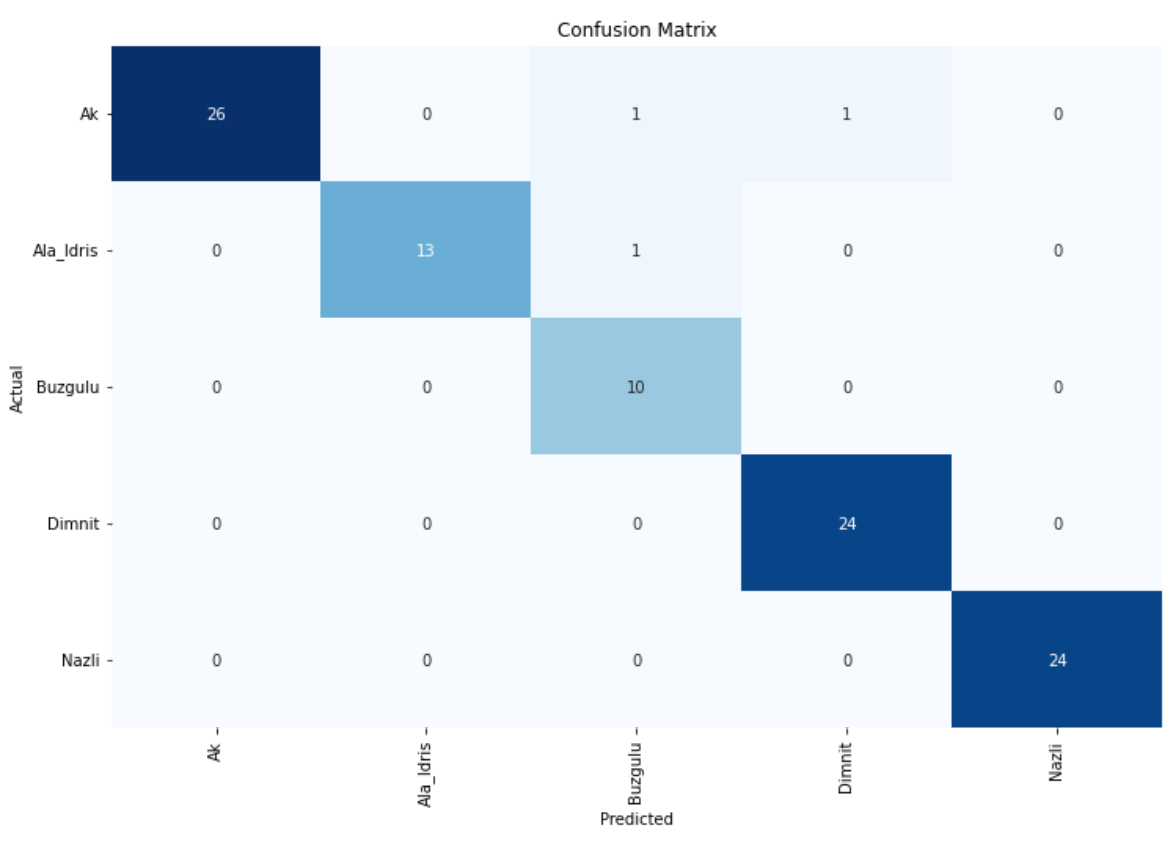
◉ **Training And Validation Loss/Accuracy:**





Although this looks like a bad case of overfit but it has a wonderful result on the test dataset!

◉ **Confusion Matrix :**



◉ **Classification Report:**

```
Classification Report:
----------------------
              precision    recall  f1-score   support

          Ak     1.0000    0.9286    0.9630        28
   Ala_Idris     1.0000    0.9286    0.9630        14
     Buzgulu     0.8333    1.0000    0.9091        10
      Dimnit     0.9600    1.0000    0.9796        24
       Nazli     1.0000    1.0000    1.0000        24

    accuracy                         0.9700       100
   macro avg     0.9587    0.9714    0.9629       100
weighted avg     0.9737    0.9700    0.9705       100
```

Now let's see how this model works with a 10 fold cross validation on epoch 40.

	1st Fold	2nd Fold	3rd Fold	4th Fold	5th Fold	6th Fold	7th Fold	8th Fold	9th Fold	10th Fold	Average
EfficientNetB3	80.0	57.5	75.0	90.0	80.0	77.5	87.5	90.0	87.5	72.5	79.75

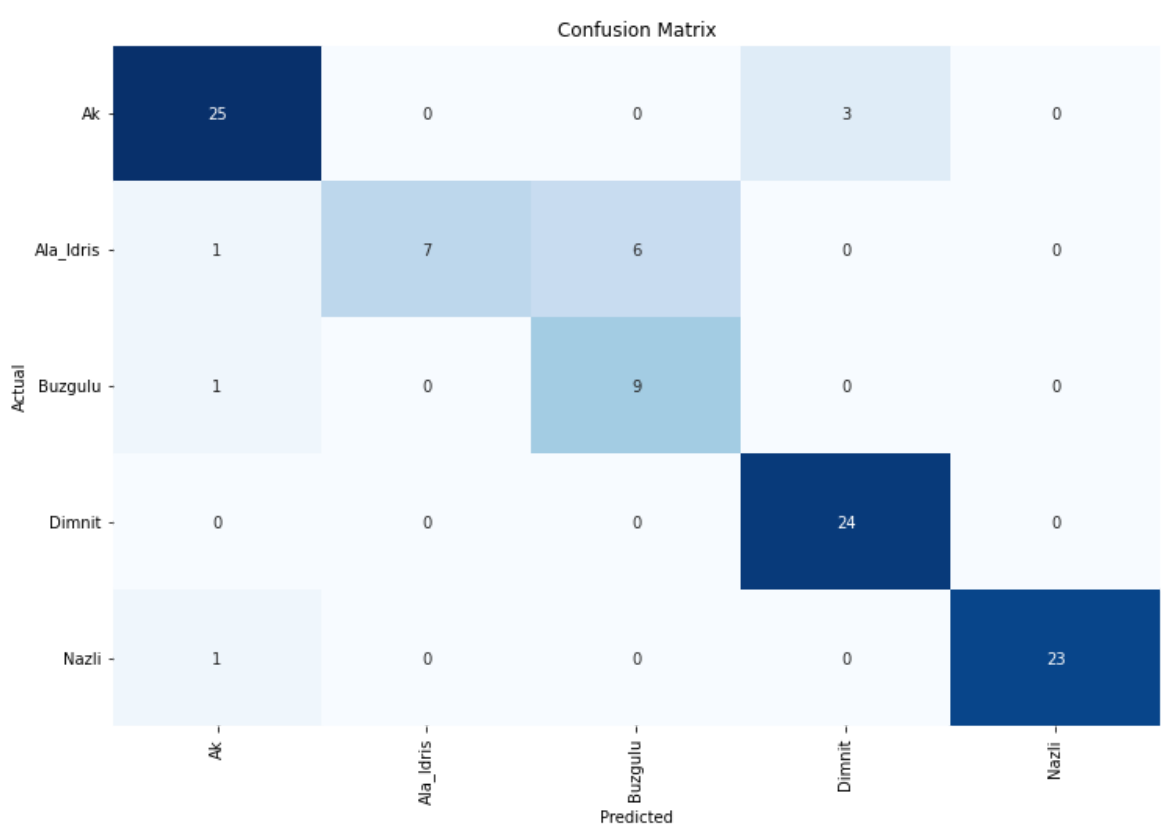
```
Validation Accuracy:
79.75% (+/- 9.07%)
Validation Loss:
6.25% (+/- 1.04%)
```

```
Best model number is 4 with validation accuracy of 90 %.
```

Running the best model on our test data set we get a test accuracy of 88%

Validation Accuracy = 90%
Test Accuracy = 88%

◉ **Confusion Matrix:**



◉ **Classification Report:**

```
Classification Report:
----------------------
              precision    recall  f1-score   support

          Ak     0.8929    0.8929    0.8929        28
   Ala_Idris     1.0000    0.5000    0.6667        14
     Buzgulu     0.6000    0.9000    0.7200        10
      Dimnit     0.8889    1.0000    0.9412        24
       Nazli     1.0000    0.9583    0.9787        24

    accuracy                         0.8800       100
   macro avg     0.8763    0.8502    0.8399       100
weighted avg     0.9033    0.8800    0.8761       100
```

6. Conclusion

In this section I summarize my work on this project.

6.1. Transform Learning

We used transform learning to train on top of 4 pre-trained models. Namely, MobileNetV2, ResNet50, EfficientNetB3, and InceptionV3.
For each model we attempted 3 times to increase the test accuracy.

Notes from the attempts:

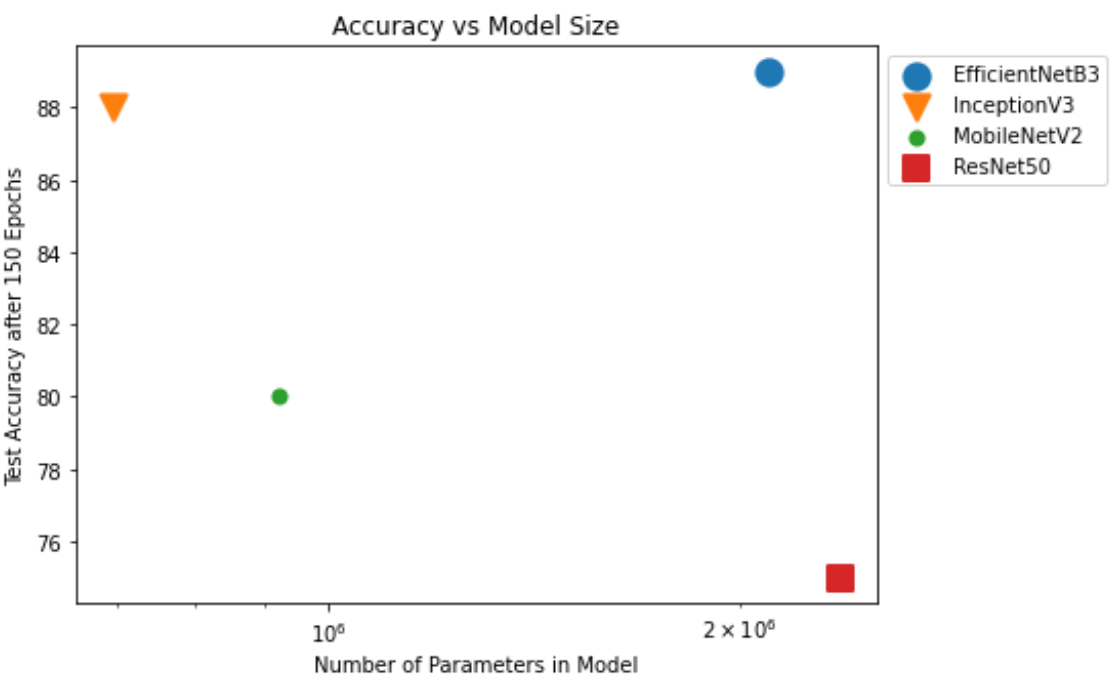
1. Adding dropout layers can reduce overfitting.
2. Adding more dense layers or unfreezing the last layers of out base model can help with underfitting.
3. Using a call back function with a low patience can also prevent overfitting.
4. Adding batch normalization and regularizing layers will help with overfitting.
5. Finding a good optimization algorithms can help the accuracy of our model.

In the table below we can see the comparison of their performance.

	Model_Name	Num_Trainable_Params	Validation_Accuracy	Test_Accuracy
2	EfficientNetB3	2099717	96	89
3	InceptionV3	695173	91	88
0	MobileNetV2	921093	90	80
1	ResNet50	2369029	81	75

We can see that without any advanced hyperparameter tuning, **EfficientNetB3** has a quite good test accuracy.

In the picture below we can see the test accuracy compared to the number of trainable parameters of each model.



Notes we can take from this plot:

1. High number of trainable parameters does not necessarily lead to higher accuracy.
2. There are no one network that fits all, some network would perform better on some datasets and we can only find our optimal network by practice.
3. Considering both accuracy and training time, one would choose the **InceptionV3** network.

6.2. Autoencoders

I trained two autoencoders for denoising and feature extraction purposes on the training dataset. Then I discarded the decoder and used the predicted images from the encoder to feed to EfficientNetB3, but it did not change the accuracy in any significant value.
My guess is that this could be due to our autoencoder needing more fine tuning or the nature of our problem.

Below we have the structure of autoencoderes usesd.

```
Model: "model_13"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_25 (InputLayer)        [(None, 228, 228, 3)]     0

conv2d_562 (Conv2D)          (None, 228, 228, 64)      1792

max_pooling2d_60 (MaxPoolin  (None, 114, 114, 64)      0
g2D)

conv2d_563 (Conv2D)          (None, 114, 114, 32)      18464

max_pooling2d_61 (MaxPoolin  (None, 57, 57, 32)        0
g2D)

conv2d_564 (Conv2D)          (None, 57, 57, 16)        4624

max_pooling2d_62 (MaxPoolin  (None, 29, 29, 16)        0
g2D)

conv2d_565 (Conv2D)          (None, 29, 29, 16)        2320

up_sampling2d_40 (UpSamplin  (None, 58, 58, 16)        0
g2D)

conv2d_566 (Conv2D)          (None, 58, 58, 32)        4640

up_sampling2d_41 (UpSamplin  (None, 116, 116, 32)      0
g2D)

conv2d_567 (Conv2D)          (None, 114, 114, 64)      18496

up_sampling2d_42 (UpSamplin  (None, 228, 228, 64)      0
g2D)

conv2d_568 (Conv2D)          (None, 228, 228, 1)       577

=================================================================
Total params: 50,913
Trainable params: 50,913
Non-trainable params: 0
_________________________________________________________________
```

Figure 1: Autoencoder

```
Model: "model_17"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_28 (InputLayer)        [(None, 228, 228, 3)]     0

gaussian_noise_2 (GaussianN  (None, 228, 228, 3)       0
oise)

conv2d_576 (Conv2D)          (None, 228, 228, 16)      448

max_pooling2d_66 (MaxPoolin  (None, 114, 114, 16)      0
g2D)

conv2d_577 (Conv2D)          (None, 114, 114, 16)      2320

max_pooling2d_67 (MaxPoolin  (None, 57, 57, 16)        0
g2D)

conv2d_578 (Conv2D)          (None, 57, 57, 8)         1160

max_pooling2d_68 (MaxPoolin  (None, 29, 29, 8)         0
g2D)

conv2d_579 (Conv2D)          (None, 29, 29, 8)         584

up_sampling2d_46 (UpSamplin  (None, 58, 58, 8)         0
g2D)

conv2d_580 (Conv2D)          (None, 58, 58, 16)        1168

up_sampling2d_47 (UpSamplin  (None, 116, 116, 16)      0
g2D)

conv2d_581 (Conv2D)          (None, 114, 114, 16)      2320

up_sampling2d_48 (UpSamplin  (None, 228, 228, 16)      0
g2D)

conv2d_582 (Conv2D)          (None, 228, 228, 1)       145

=================================================================
Total params: 8,145
Trainable params: 8,145
Non-trainable params: 0
_________________________________________________________________
```

Figure 2: Denoising Autoencoder

6.3. Cross Validation

Since cross validation is very time consuming, specially for deep layers with high epochs, I had to limit the model to only 20 epochs. Thus we don't get the 91% accuracy we had before.

	1st Fold	2nd Fold	3rd Fold	4th Fold	5th Fold	6th Fold	7th Fold	8th Fold	9th Fold	10th Fold	Average
InceptionV3	62.5	67.5	65.0	67.5	70.0	57.5	72.5	80.0	65.0	60.0	66.75

```
Validation Accuracy:
66.75% (+/- 5.84%)
Validation Loss:
3.19% (+/- 0.26%)
```

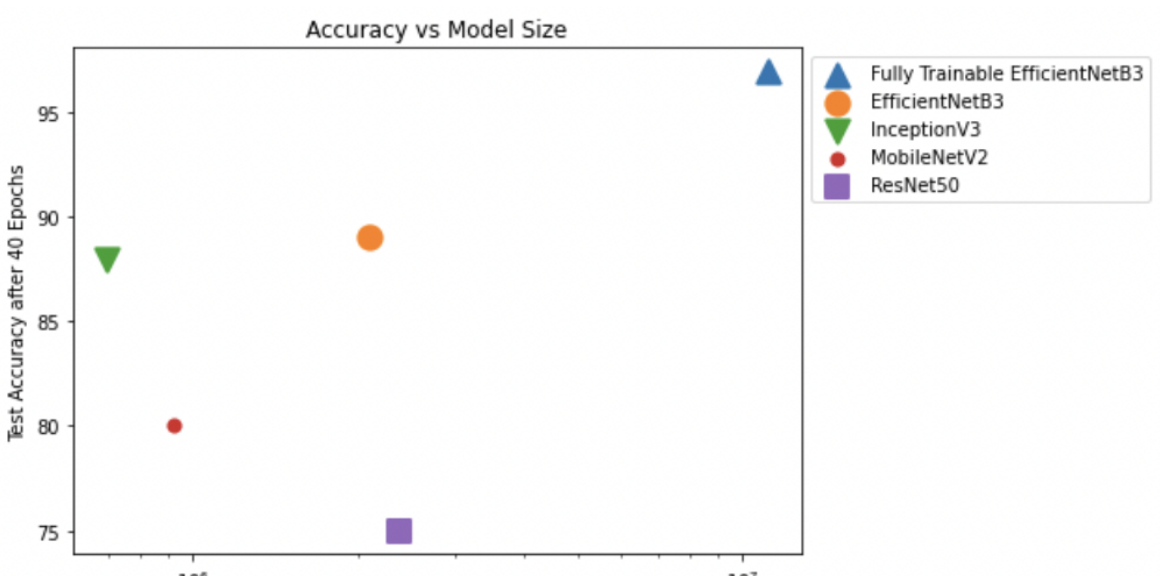
```
Best model number is 8 with validation accuracy of 80 %.
```

$$\textbf{Maximum Validation Accuracy} = 80\%$$
$$\textbf{Test Accuracy} = 76\%$$

6.4. Improving The Accuracy

In this part I unfroze all layers of pre-trianed **EfficientNetB3**, so it can fully learn our dataset and find the best weight for our purpose and not just the ImageNet.
To avoid overfitting I used BatchNormalization.
The result was amazing!
Now we have:

	Model_Name	Num_Trainable_Params	Validation_Accuracy	Test_Accuracy
4	Fully Trainable EfficientNetB3	11094061	90	97
2	EfficientNetB3	2099717	96	89
3	InceptionV3	695173	91	88
0	MobileNetV2	921093	90	80
1	ResNet50	2369029	81	75



The paper achieved an accuracy of 97.5% and I got a test accuracy of 97% so pretty close! Performing more hyperparameter tuning and doing more experiments would probably result in higher accuracy.

This is a very interesting fact! Although transfer learning can save us time and resource by providing us with the pre-trained weights, using the state of the art architecture of CNN models can lead us to very high accuracies because neurons would learn exactly our dataset and not ImageNet.

To make sure the accuracy was reliable, I performed 10 fold cross validation on this model. The result:

	1st Fold	2nd Fold	3rd Fold	4th Fold	5th Fold	6th Fold	7th Fold	8th Fold	9th Fold	10th Fold	Average
EfficientNetB3	80.0	57.5	75.0	90.0	80.0	77.5	87.5	90.0	87.5	72.5	79.75

```
Validation Accuracy:
79.75% (+/- 9.07%)
Validation Loss:
6.25% (+/- 1.04%)
```

And it turned out that the model is good most of the times but fails terribly at some folds! So we should always perform cross validation to make sure our accuracy is reliable.