Denoising_And_Encoding_Networks

August 12, 2022

1 Grapevine leaves classification, denoising and encoding networks parts

Reza Arabpour,
University of Tehran,
Final version, Aug 2022.

1.1 Environment Initialization

[ ]:
```
#Loading General Tools and Libraries
import os
import random
import shutil
import sklearn
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import tensorflow as tf
import tensorflow_datasets as tfds
from tensorflow import keras
from keras import layers
plt.style.use('ggplot')
```

1.2 Data Loading

1.2.1 Data Downloading

[ ]:
```
#Loading Dataset
!rm -rf Grapevine_Leaves_Image_Dataset*
!rm -rf Test_Data
!wget https://www.muratkoklu.com/datasets/Grapevine_Leaves_Image_Dataset.zip
!unzip -q Grapevine_Leaves_Image_Dataset.zip
!rm Grapevine_Leaves_Image_Dataset.zip
!ls -lh
```

```
--2022-08-06 06:26:31--
https://www.muratkoklu.com/datasets/Grapevine_Leaves_Image_Dataset.zip
Resolving www.muratkoklu.com (www.muratkoklu.com)… 185.179.25.150
```

```
Connecting to www.muratkoklu.com (www.muratkoklu.com)|185.179.25.150|:443…
connected.
HTTP request sent, awaiting response… 200 OK
Length: 113862245 (109M) [application/zip]
Saving to: ‘Grapevine_Leaves_Image_Dataset.zip’

Grapevine_Leaves_Im 100%[===================>] 108.59M  20.7MB/s    in 6.3s

2022-08-06 06:26:38 (17.4 MB/s) - ‘Grapevine_Leaves_Image_Dataset.zip’ saved
[113862245/113862245]

total 8.0K
drwxr-xr-x 7 root root 4.0K Feb 11 17:53 Grapevine_Leaves_Image_Dataset
drwxr-xr-x 1 root root 4.0K Aug  3 20:21 sample_data
```

1.2.2 Select and Move 20% of Data for OOS (Out Of Sample) Testing

[ ]:
```
Species = ["Ak", "Ala_Idris", "Buzgulu", "Dimnit", "Nazli"]
OriginalDataPath = "Grapevine_Leaves_Image_Dataset"
TestDataPath = "Test_Data"
os.mkdir(os.path.join(TestDataPath,""))

for folder_name in Species:
  Percentage = 20/100
  Test_location = os.path.join(TestDataPath,folder_name)
  os.mkdir(Test_location)
  Original_location = os.path.join(OriginalDataPath, folder_name)
  AllImages = os.listdir(Original_location)
  Count = int(Percentage * len(AllImages))
  Selecteds = random.sample(AllImages, Count)
  for file in Selecteds:
    shutil.move(os.path.join(Original_location,file), Test_location)
```

1.2.3 Datasets Loading (Train, Validation, and Test)

[ ]:
```
image_size = (256,256)
batch_size = 32
validation_split = 0.2
seed = 199

train_data = tf.keras.utils.image_dataset_from_directory(
    'Grapevine_Leaves_Image_Dataset',
    class_names = ["Ak", "Ala_Idris", "Buzgulu", "Dimnit", "Nazli"],
    batch_size=batch_size,
    image_size=image_size,
    seed=seed,
    validation_split=validation_split,
```

```
    subset="training"
)

validation_data = tf.keras.utils.image_dataset_from_directory(
    'Grapevine_Leaves_Image_Dataset',
    class_names = ["Ak", "Ala_Idris", "Buzgulu", "Dimnit", "Nazli"],
    batch_size=batch_size,
    image_size=image_size,
    seed=seed,
    validation_split=validation_split,
    subset="validation"
)

test_data = tf.keras.utils.image_dataset_from_directory(
    'Test_Data',
    class_names = ["Ak", "Ala_Idris", "Buzgulu", "Dimnit", "Nazli"],
    image_size=image_size,
    batch_size=100,
    shuffle=False)
```

```
Found 400 files belonging to 5 classes.
Using 320 files for training.
Found 400 files belonging to 5 classes.
Using 80 files for validation.
Found 100 files belonging to 5 classes.
```

1.3 Data Augmentation

1.3.1 Color Transformation

[ ]:
```
train_data2 = train_data.map(lambda x, y: (255-x, y))
validation_data2 = validation_data.map(lambda x, y: (255-x, y))
test_data2 = test_data.map(lambda x, y: (255-x, y))
```

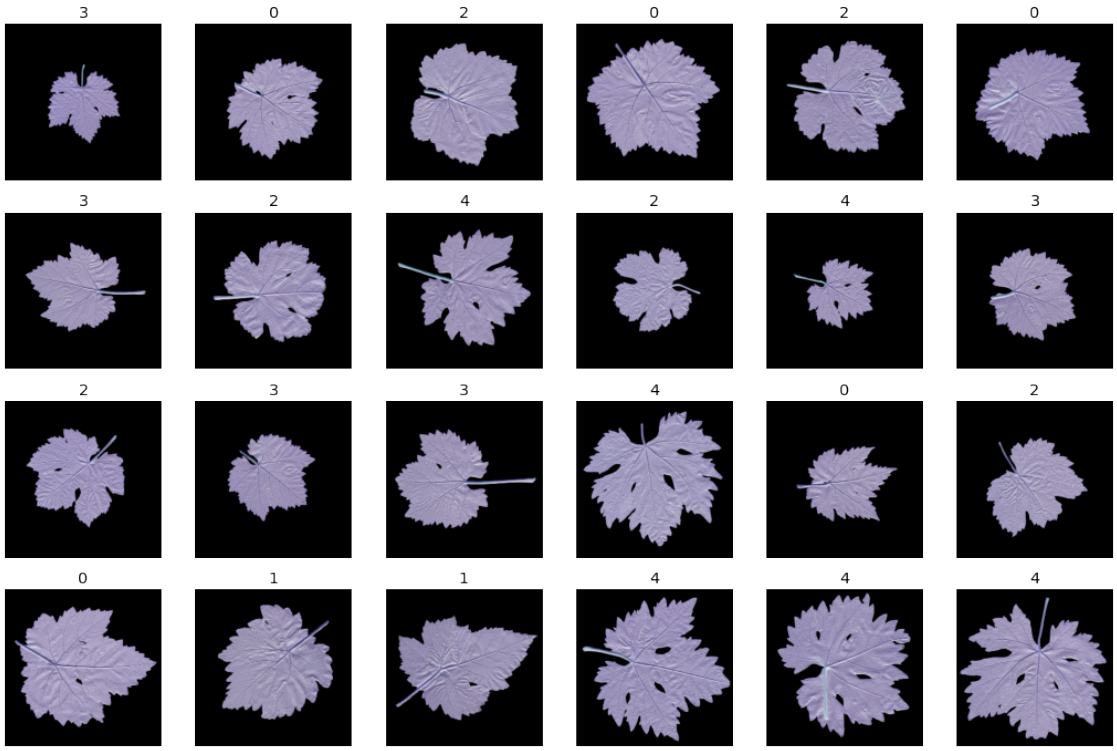
1.3.2 Data Augmentation

[ ]:
```
data_augmentation = keras.Sequential([
    layers.RandomFlip("horizontal"),
    layers.RandomFlip("vertical"),
    layers.RandomZoom(height_factor=(-0.2,0.2), width_factor=(-0.2,0.
  ↪2),fill_mode='constant', fill_value=0),
    layers.RandomRotation(0.3, fill_mode='constant', fill_value=0)
])

augmented_train_data = train_data2
for i in range(4):
  augmented_train_data = augmented_train_data.concatenate(train_data2.
  ↪map(lambda x, y: (data_augmentation(x, training=True), y)))
```

1.3.3 Showing a random sample

[ ]:
```
plt.figure(figsize=(18, 12))
for images, labels in augmented_train_data.take(1):
    for i in range(24):
        ax = plt.subplot(4, 6, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))
        plt.title(int(labels[i]))
        plt.axis("off")
```



1.4 Exploiting Autoencoder Networks

1.4.1 Denoising Autoencoder

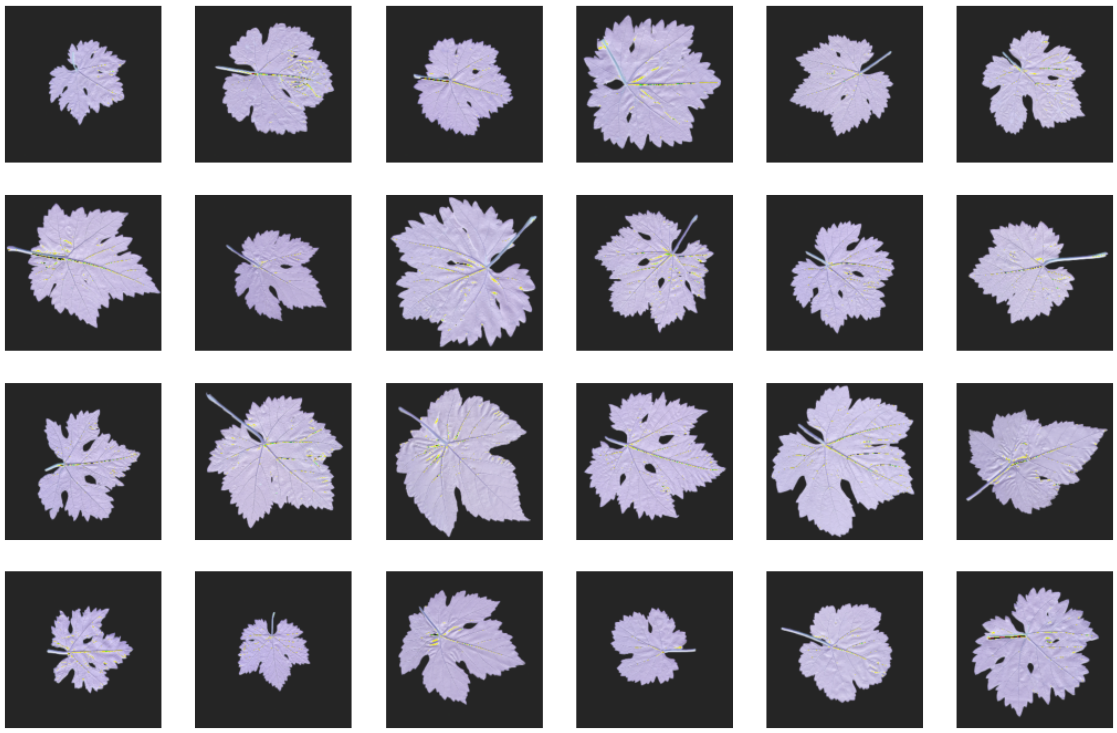
Noisy Data Generation

[ ]:
```
Noise_Factor = 0.3
Noise_Mean = 0.5 * 255

train_data_denoising = augmented_train_data.map(lambda x, y: (x + Noise_Factor␣
 ↪* np.random.normal(loc=Noise_Mean, scale=1.0, size=image_size + (3,)), x))
validation_data_denoising = validation_data2.map(lambda x, y: (x + Noise_Factor␣
 ↪* np.random.normal(loc=Noise_Mean, scale=1.0, size=image_size + (3,)), x))
test_data_denoising = test_data2.map(lambda x, y: (x + Noise_Factor * np.random.
 ↪normal(loc=Noise_Mean, scale=1.0, size=image_size + (3,)), x))
```

Random Sample

```
[ ]: plt.figure(figsize=(18, 12))
for images, labels in train_data_denoising.take(1):
    for i in range(24):
        ax = plt.subplot(4, 6, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))
        plt.axis("off")
```



Model Architecture Define

```
[ ]: Denoising_Model = keras.models.Sequential([
    layers.Input(shape = image_size + (3,)),

    layers.Conv2D(16, (3, 3), activation='relu', padding="same"),
    layers.MaxPooling2D((2, 2), strides = (2, 2), padding="same"),

    layers.Conv2D(32, (3, 3), activation='relu', padding="same"),
    layers.MaxPooling2D((2, 2), strides = (2, 2), padding="same"),

    layers.Conv2D(64, (3, 3), activation='relu', padding="same"),
    layers.MaxPooling2D((2, 2), strides = (2, 2), padding="same"),

    layers.Conv2D(128, (3, 3), activation='relu', padding="same"),
    layers.MaxPooling2D((2, 2), strides = (2, 2), padding="same"),
```

```
    layers.Conv2D(256, (3, 3), activation='relu', padding="same"),
    layers.MaxPooling2D((2, 2), strides = (2, 2), padding="same"),

    layers.Conv2DTranspose(256, (3, 3), strides = 2, activation="relu",␣
 ↪padding="same"),

    layers.Conv2DTranspose(128, (3, 3), strides = 2, activation="relu",␣
 ↪padding="same"),

    layers.Conv2DTranspose(64, (3, 3), strides = 2, activation="relu",␣
 ↪padding="same"),

    layers.Conv2DTranspose(32, (3, 3), strides = 2, activation="relu",␣
 ↪padding="same"),

    layers.Conv2DTranspose(16, (3, 3), strides = 2, activation="relu",␣
 ↪padding="same"),

    layers.Conv2D(3, (3, 3), activation="sigmoid", padding="same"),
    layers.Rescaling(scale = 255.0/1, offset = 0.0)
    ])
Denoising_Model.summary()
```

```
Model: "sequential_10"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_307 (Conv2D)         (None, 256, 256, 16)      448

 max_pooling2d_14 (MaxPoolin  (None, 128, 128, 16)     0
 g2D)

 conv2d_308 (Conv2D)         (None, 128, 128, 32)      4640

 max_pooling2d_15 (MaxPoolin  (None, 64, 64, 32)       0
 g2D)

 conv2d_309 (Conv2D)         (None, 64, 64, 64)        18496

 max_pooling2d_16 (MaxPoolin  (None, 32, 32, 64)       0
 g2D)

 conv2d_310 (Conv2D)         (None, 32, 32, 128)       73856

 max_pooling2d_17 (MaxPoolin  (None, 16, 16, 128)      0
 g2D)
```

```
 conv2d_311 (Conv2D)         (None, 16, 16, 256)       295168

 max_pooling2d_18 (MaxPoolin  (None, 8, 8, 256)        0
 g2D)

 conv2d_transpose (Conv2DTra  (None, 16, 16, 256)      590080
 nspose)

 conv2d_transpose_1 (Conv2DT  (None, 32, 32, 128)      295040
 ranspose)

 conv2d_transpose_2 (Conv2DT  (None, 64, 64, 64)       73792
 ranspose)

 conv2d_transpose_3 (Conv2DT  (None, 128, 128, 32)     18464
 ranspose)

 conv2d_transpose_4 (Conv2DT  (None, 256, 256, 16)     4624
 ranspose)

 conv2d_312 (Conv2D)         (None, 256, 256, 3)       435

 rescaling (Rescaling)       (None, 256, 256, 3)       0

=================================================================
Total params: 1,375,043
Trainable params: 1,375,043
Non-trainable params: 0
_________________________________________________________________
```

Model Creation and Training

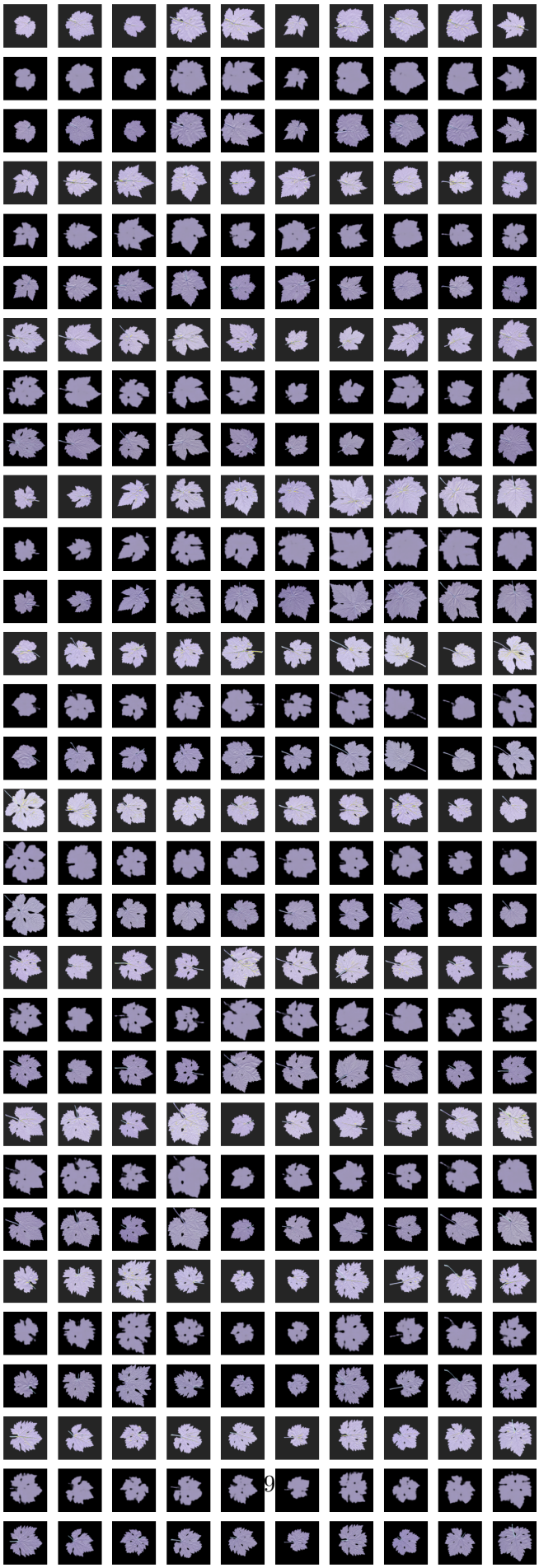
[ ]:
```
Denoising_Model.compile(
    optimizer=keras.optimizers.Adam(1e-3),
    loss=keras.losses.MeanSquaredError(),
    metrics=["accuracy"],
)
history_Denosing = Denoising_Model.fit(
    train_data_denoising,
    epochs=30,
    validation_data=validation_data_denoising)
Denoising_Model.save('Denoising_Model.h5')
```

Model Performance Evaluation on Test Data

[ ]:
```
Denoising_Output = Denoising_Model.predict(test_data_denoising).round()

plt.figure(figsize=(20, 60))
```

```
for images, labels in test_data_denoising:
  for rows in range(10):
    for i in range(10):
      ax = plt.subplot(30, 10, rows*30 + i + 1)
      plt.imshow(images[rows*10+i].numpy().astype("uint8"))
      plt.axis("off")
      ax = plt.subplot(30, 10, rows*30 + 10 + i + 1)
      plt.imshow(Denoising_Output[rows*10+i].astype("uint8"))
      plt.axis("off")
      ax = plt.subplot(30, 10, rows*30 + 20 + i + 1)
      plt.imshow(labels[rows*10+i].numpy().astype("uint8"))
      plt.axis("off")
```



Denosing and Classifier Models Merging

Model Architecture Define

[ ]:
```
pretrained_model = tf.keras.applications.ResNet152(
    include_top=False,
    weights="imagenet",
    input_shape=image_size + (3,),
    classes=1000,
)

Denoising_Model.trainable = False
pretrained_model.trainable = False

Denoised_ResNet152_Model = keras.models.Sequential([
    layers.Input(shape = image_size + (3,)),
    layers.RandomRotation(0.3, fill_mode='constant', fill_value=0),
    layers.RandomZoom(height_factor=(-0.2,0.2), width_factor=(-0.2,0.
 ↪2),fill_mode='constant', fill_value=0),
    layers.RandomFlip("horizontal"),
    layers.RandomFlip("vertical"),
    Denoising_Model,
    pretrained_model,
    layers.Flatten(),
    layers.Dense(512, activation='relu'),
    layers.Dense(256, activation='relu'),
    layers.Dense(256, activation='relu'),
    layers.Dense(5, activation='softmax')])

Denoised_ResNet152_Model.summary()
```

```
Model: "sequential_27"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 random_rotation_23 (RandomR  (None, 256, 256, 3)      0
 otation)

 random_zoom_22 (RandomZoom)  (None, 256, 256, 3)      0

 random_flip_46 (RandomFlip)  (None, 256, 256, 3)      0

 random_flip_47 (RandomFlip)  (None, 256, 256, 3)      0

 sequential_10 (Sequential)  (None, 256, 256, 3)       1375043
```

```
resnet152 (Functional)      (None, 8, 8, 2048)        58370944

flatten_22 (Flatten)        (None, 131072)            0

dense_82 (Dense)            (None, 512)               67109376

dense_83 (Dense)            (None, 256)               131328

dense_84 (Dense)            (None, 256)               65792

dense_85 (Dense)            (None, 5)                 1285

=================================================================
Total params: 127,053,768
Trainable params: 67,307,781
Non-trainable params: 59,745,987
_________________________________________________________________
```

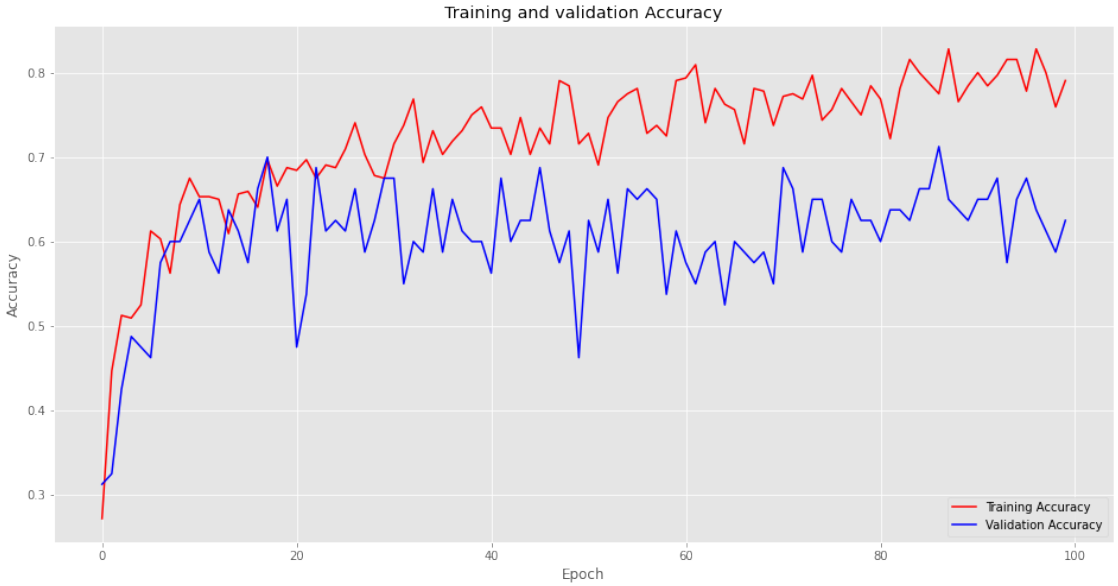
Model Creation and Training

[ ]:
```
Denoised_ResNet152_Model.compile(
    optimizer=keras.optimizers.Adam(1e-4),
    loss=tf.keras.losses.SparseCategoricalCrossentropy(),
    metrics=["accuracy"])
Denoised_ResNet152_Model_History = Denoised_ResNet152_Model.fit(
    train_data2,
    epochs = 100,
    validation_data = validation_data2)
Denoised_ResNet152_Model.save('Denoised_ResNet152_Model.h5')
```

Accuracy and Loss Curves

[ ]:
```
Denoised_ResNet152_Model_acc = Denoised_ResNet152_Model_History.
 ↪history['accuracy']
Denoised_ResNet152_Model_val_acc = Denoised_ResNet152_Model_History.
 ↪history['val_accuracy']
Denoised_ResNet152_Model_loss = Denoised_ResNet152_Model_History.history['loss']
Denoised_ResNet152_Model_val_loss = Denoised_ResNet152_Model_History.
 ↪history['val_loss']
epochs = range(len(Denoised_ResNet152_Model_acc))
fig = plt.figure(figsize=(16,8))
plt.plot(epochs, Denoised_ResNet152_Model_acc, 'r', label="Training Accuracy")
plt.plot(epochs, Denoised_ResNet152_Model_val_acc, 'b', label="Validation␣
 ↪Accuracy")
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Training and validation Accuracy')
plt.legend(loc='lower right')
fig2 = plt.figure(figsize=(16,8))
```

```
plt.plot(epochs, Denoised_ResNet152_Model_loss, 'r', label="Training Loss")
plt.plot(epochs, Denoised_ResNet152_Model_val_loss, 'b', label="Validation␣
 ↪Loss")
plt.legend(loc='upper right')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training and validation Loss')
plt.show()
```



Model Performance Evaluation on Test Data

```
[ ]: result = Denoised_ResNet152_Model.predict(test_data2)
Denoised_ResNet152_Model_y_predict = np.array([i.argmax() for i in result])

Denoised_ResNet152_Model_cm = confusion_matrix(y_test,␣
  ↪Denoised_ResNet152_Model_y_predict)
Denoised_ResNet152_Model_ac =␣
  ↪accuracy_score(y_test,Denoised_ResNet152_Model_y_predict)

print("confusion matrix on test data :\n",Denoised_ResNet152_Model_cm)
print("accuracy score on test data:\n",Denoised_ResNet152_Model_ac)

print(classification_report(y_test, Denoised_ResNet152_Model_y_predict))
```

```
confusion matrix on test data :
 [[ 9  1  2  3  5]
 [ 3  9  1  3  4]
 [ 0  2 11  6  1]
 [ 0  0  1 19  0]
 [ 0  0  0  4 16]]
accuracy score on test data:
 0.64
              precision    recall  f1-score   support

           0       0.75      0.45      0.56        20
           1       0.75      0.45      0.56        20
           2       0.73      0.55      0.63        20
           3       0.54      0.95      0.69        20
           4       0.62      0.80      0.70        20

    accuracy                           0.64       100
   macro avg       0.68      0.64      0.63       100
weighted avg       0.68      0.64      0.63       100
```

1.4.2 Dimension Reduction Autoencoder

Input Dataset Generation

```
[ ]: train_data_dimension_reduction = augmented_train_data.map(lambda x, y: (x, x))
validation_data_dimension_reduction = validation_data2.map(lambda x, y: (x, x))
test_data_dimension_reduction = test_data2.map(lambda x, y: (x, x))
```

Model Architecture Define

```
[ ]: Encoder_Model = keras.models.Sequential([
    layers.Resizing(128, 128),
    layers.Flatten(),
    layers.Dense(512, activation='relu'),
    layers.Dense(512, activation='relu'),
```

```
    layers.Dense(512, activation='relu'),
    layers.Dense(256, activation='relu'),
])
Decoder_Model = keras.models.Sequential([
    layers.Dense(512, activation='relu'),
    layers.Dense(512, activation='relu'),
    layers.Dense(512, activation='relu'),
    layers.Dense(128*128*3, activation='relu'),
    layers.Reshape((128,128,3)),
    layers.Resizing(256, 256),
    ])
AutoEncoder_Model = keras.models.Sequential([
    layers.Input(shape = image_size + (3,)),
    Encoder_Model,
    Decoder_Model,
    ])
AutoEncoder_Model.summary()
```

```
Model: "sequential_14"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 sequential_12 (Sequential)  (None, 256)               25822976

 sequential_13 (Sequential)  (None, 256, 256, 3)       25871872

=================================================================
Total params: 51,694,848
Trainable params: 51,694,848
Non-trainable params: 0
_________________________________________________________________
```

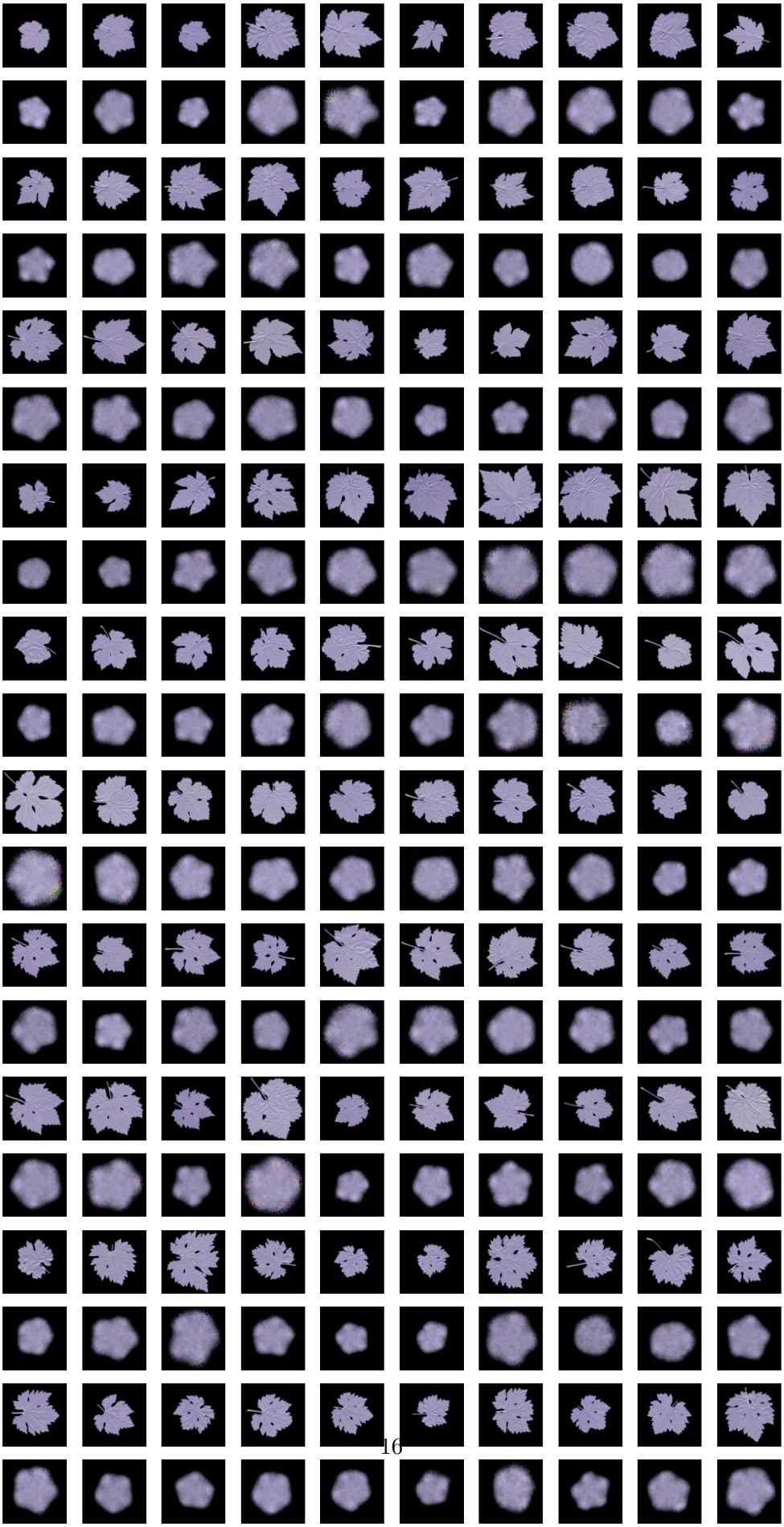
Model Creation and Training

[ ]:
```
AutoEncoder_Model.compile(
    optimizer=keras.optimizers.Adam(1e-3),
    loss=tf.keras.losses.MeanSquaredError(),
    metrics=["accuracy"])
AutoEncoder_Model_History = AutoEncoder_Model.fit(
    train_data_dimension_reduction,
    epochs = 30,
    validation_data = validation_data_dimension_reduction)
AutoEncoder_Model.save('AutoEncoder_Model.h5')
```

Model Performance Evaluation on Test Data

[ ]:
```
AutoEncoder_Output = AutoEncoder_Model.predict(test_data_dimension_reduction).
  ↪round()
```

```
plt.figure(figsize=(20, 40))
for images, labels in test_data_dimension_reduction:
  for rows in range(10):
    for i in range(10):
      ax = plt.subplot(20, 10, rows*20 + i + 1)
      plt.imshow(images[rows*10+i].numpy().astype("uint8"))
      plt.axis("off")
      ax = plt.subplot(20, 10, rows*20 + 10 + i + 1)
      plt.imshow(AutoEncoder_Output[rows*10+i].astype("uint8"))
      plt.axis("off")
```



Dimension Reduction and Classifier Models Merging

Model Architecture Define

```
[ ]: Encoder_Model.trainable = False

Dimension_Reduction_Model = keras.models.Sequential([
    layers.Input(shape = image_size + (3,)),
    layers.RandomRotation(0.3, fill_mode='constant', fill_value=0),
    layers.RandomZoom(height_factor=(-0.2,0.2), width_factor=(-0.2,0.
 ↪2),fill_mode='constant', fill_value=0),
    layers.RandomFlip("horizontal"),
    layers.RandomFlip("vertical"),
    Encoder_Model,
    layers.Dense(512, activation='relu'),
    layers.Dense(1024, activation='relu'),
    layers.Dense(2048, activation='relu'),
    layers.Dense(1024, activation='relu'),
    layers.Dense(512, activation='relu'),
    layers.Dense(256, activation='relu'),
    layers.Dense(128, activation='relu'),
    layers.Dense(5, activation='softmax')
    ])

Dimension_Reduction_Model.summary()
```

```
Model: "sequential_15"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 random_rotation_11 (RandomR  (None, 256, 256, 3)      0
 otation)

 random_zoom_10 (RandomZoom)  (None, 256, 256, 3)      0

 random_flip_22 (RandomFlip)  (None, 256, 256, 3)      0

 random_flip_23 (RandomFlip)  (None, 256, 256, 3)      0

 sequential_12 (Sequential)  (None, 256)               25822976

 dense_41 (Dense)            (None, 512)               131584

 dense_42 (Dense)            (None, 1024)              525312

 dense_43 (Dense)            (None, 2048)              2099200
```

```
 dense_44 (Dense)            (None, 1024)              2098176

 dense_45 (Dense)            (None, 512)               524800

 dense_46 (Dense)            (None, 256)               131328

 dense_47 (Dense)            (None, 128)               32896

 dense_48 (Dense)            (None, 5)                 645

=================================================================
Total params: 31,366,917
Trainable params: 5,543,941
Non-trainable params: 25,822,976
_________________________________________________________________
```

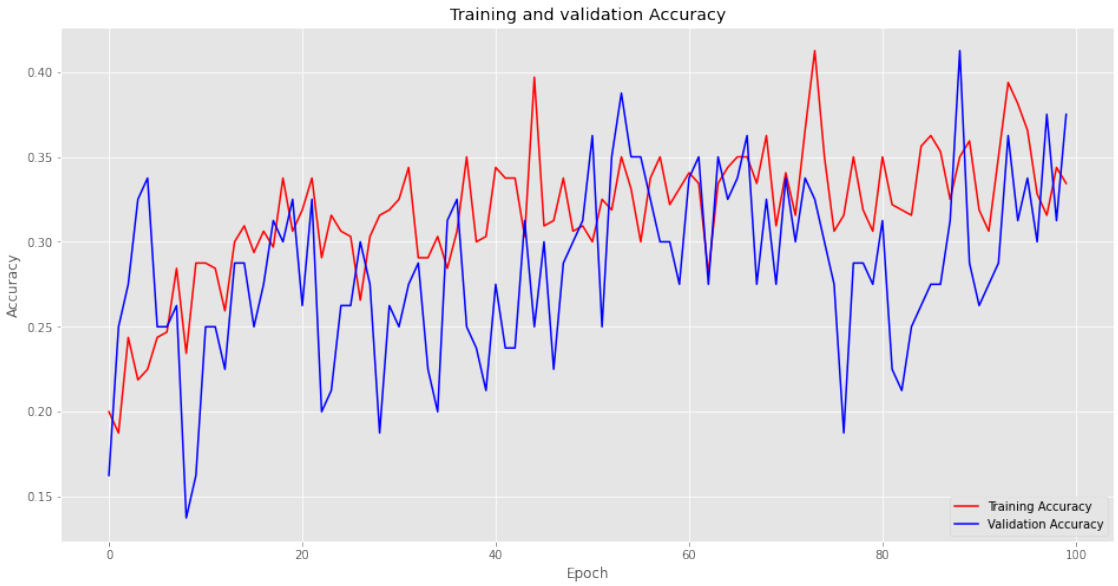
Model Creation and Training

[ ]:
```
Dimension_Reduction_Model.compile(
    optimizer=keras.optimizers.Adam(1e-3),
    loss=tf.keras.losses.SparseCategoricalCrossentropy(),
    metrics=["accuracy"])
Dimension_Reduction_Model_History = Dimension_Reduction_Model.fit(
    train_data2,
    epochs = 100,
    validation_data = validation_data2)
Dimension_Reduction_Model.save('Dimension_Reduction_Model.h5')
```

Accuracy and Loss Curves

[ ]:
```
Dimension_Reduction_Model_acc = Dimension_Reduction_Model_History.
 ↪history['accuracy']
Dimension_Reduction_Model_val_acc = Dimension_Reduction_Model_History.
 ↪history['val_accuracy']
Dimension_Reduction_Model_loss = Dimension_Reduction_Model_History.
 ↪history['loss']
Dimension_Reduction_Model_val_loss = Dimension_Reduction_Model_History.
 ↪history['val_loss']
epochs = range(len(Dimension_Reduction_Model_acc))
fig = plt.figure(figsize=(16,8))
plt.plot(epochs, Dimension_Reduction_Model_acc, 'r', label="Training Accuracy")
plt.plot(epochs, Dimension_Reduction_Model_val_acc, 'b', label="Validation␣
 ↪Accuracy")
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Training and validation Accuracy')
plt.legend(loc='lower right')
```

```
fig2 = plt.figure(figsize=(16,8))
plt.plot(epochs, Dimension_Reduction_Model_loss, 'r', label="Training Loss")
plt.plot(epochs, Dimension_Reduction_Model_val_loss, 'b', label="Validation␣
 ↪Loss")
plt.legend(loc='upper right')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training and validation Loss')
plt.show()
```



Model Performance Evaluation on Test Data

[ ]:
```
result = Dimension_Reduction_Model.predict(test_data2)
Dimension_Reduction_Model_y_predict = np.array([i.argmax() for i in result])

Dimension_Reduction_Model_cm = confusion_matrix(y_test,␣
 ↪Dimension_Reduction_Model_y_predict)
Dimension_Reduction_Model_ac =␣
 ↪accuracy_score(y_test,Dimension_Reduction_Model_y_predict)

print("confusion matrix on test data :\n",Dimension_Reduction_Model_cm)
print("accuracy score on test data:\n",Dimension_Reduction_Model_ac)

print(classification_report(y_test, Dimension_Reduction_Model_y_predict))
```

```
confusion matrix on test data :
 [[11  1  3  4  1]
 [ 3  6  1  4  6]
 [ 7  1  4  3  5]
 [ 5  5  1  2  7]
 [ 3  1  1  8  7]]
accuracy score on test data:
 0.3
              precision    recall  f1-score   support

           0       0.38      0.55      0.45        20
           1       0.43      0.30      0.35        20
           2       0.40      0.20      0.27        20
           3       0.10      0.10      0.10        20
           4       0.27      0.35      0.30        20

    accuracy                           0.30       100
   macro avg       0.31      0.30      0.29       100
weighted avg       0.31      0.30      0.29       100
```