Q2 [P ‖ CO2 & CO3] Autoencoder (6 points)

Pick random 1000 images (Train- 900, Test-100) of 5 Superclasses of the CIFAR-100 dataset (https:// www.cs.toronto.edu/~kriz/cifar.html, https://www.cs.toronto.edu/~kriz/cifar-100-python.tar. gz) and train an autoencoder to regenerate the images. Apply batch normalization and plot the loss vs. epoch training curve. Print a 5x2 grid containing 1 test image of each class, in which the first column contains the original image and the second column contains the autoencoder output of the same image. Now, use the latent embeddings learned in the above autoencoder to build a five-class classifier. Show performance on train and test sets using accuracy as an evaluation metric.

```
!pip install tensorflow
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: tensorflow in /usr/local/lib/python3.9/dist-packages (2.11.0)
Requirement already satisfied: protobuf<3.20,>=3.9.2 in /usr/local/lib/python3.9/dist-packages (from tensorflow) (3.19.6)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.9/dist-packages (from tensorflow) (1.16.0)
Requirement already satisfied: setuptools in /usr/local/lib/python3.9/dist-packages (from tensorflow) (63.4.3)
Requirement already satisfied: gast<=0.4.0,>=0.2.1 in /usr/local/lib/python3.9/dist-packages (from tensorflow) (0.4.0)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.9/dist-packages (from tensorflow) (1.6.3)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.9/dist-packages (from tensorflow) (1.51.3)
Requirement already satisfied: h5py>=2.9.0 in /usr/local/lib/python3.9/dist-packages (from tensorflow) (3.8.0)
Requirement already satisfied: numpy>=1.20 in /usr/local/lib/python3.9/dist-packages (from tensorflow) (1.22.4)
Requirement already satisfied: tensorflow-estimator<2.12,>=2.11.0 in /usr/local/lib/python3.9/dist-packages (from tensorflow) (2.11.0
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.9/dist-packages (from tensorflow) (15.0.6.1)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.9/dist-packages (from tensorflow) (2.2.0)
Requirement already satisfied: tensorboard<2.12,>=2.11 in /usr/local/lib/python3.9/dist-packages (from tensorflow) (2.11.2)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/python3.9/dist-packages (from tensorflow) (0.31.
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.9/dist-packages (from tensorflow) (4.5.0)
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.9/dist-packages (from tensorflow) (3.3.0)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.9/dist-packages (from tensorflow) (1.4.0)
Requirement already satisfied: keras<2.12,>=2.11.0 in /usr/local/lib/python3.9/dist-packages (from tensorflow) (2.11.0)
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.9/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: flatbuffers>=2.0 in /usr/local/lib/python3.9/dist-packages (from tensorflow) (23.3.3)
Requirement already satisfied: packaging in /usr/local/lib/python3.9/dist-packages (from tensorflow) (23.0)
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.9/dist-packages (from tensorflow) (1.15.0)
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.9/dist-packages (from astunparse>=1.6.0->tensorflow) (0.4
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.9/dist-packages (from tensorboard<2.12,>=2.11->tensorflow) (
Requirement already satisfied: tensorboard-data-server<0.7.0,>=0.6.0 in /usr/local/lib/python3.9/dist-packages (from tensorboard<2.12,
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.9/dist-packages (from tensorboard<2.12,>=2.11->tensorflow) (
Requirement already satisfied: google-auth<3,>=1.6.3 in /usr/local/lib/python3.9/dist-packages (from tensorboard<2.12,>=2.11->tensorf
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.9/dist-packages (from tensorboard<2.12,>=2.11->tensorflow
Requirement already satisfied: tensorboard-plugin-wit>=1.6.0 in /usr/local/lib/python3.9/dist-packages (from tensorboard<2.12,>=2.11->
Requirement already satisfied: google-auth-oauthlib<0.5,>=0.4.1 in /usr/local/lib/python3.9/dist-packages (from tensorboard<2.12,>=2.
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.9/dist-packages (from google-auth<3,>=1.6.3->tensorboard<2.12,>
Requirement already satisfied: cachetools<6.0,>=2.0.0 in /usr/local/lib/python3.9/dist-packages (from google-auth<3,>=1.6.3->tensorboa
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.9/dist-packages (from google-auth<3,>=1.6.3->tensorboa
Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/python3.9/dist-packages (from google-auth-oauthlib<0.5,>=0.
Requirement already satisfied: importlib-metadata>=4.4 in /usr/local/lib/python3.9/dist-packages (from markdown>=2.6.8->tensorboard<2.
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.9/dist-packages (from requests<3,>=2.21.0->tensorboard<2.12,>=2.
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.9/dist-packages (from requests<3,>=2.21.0->tensorboard<2.
Requirement already satisfied: charset-normalizer~=2.0.0 in /usr/local/lib/python3.9/dist-packages (from requests<3,>=2.21.0->tensorb
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.9/dist-packages (from requests<3,>=2.21.0->tensorboard
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.9/dist-packages (from werkzeug>=1.0.1->tensorboard<2.12,>=2
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.9/dist-packages (from importlib-metadata>=4.4->markdown>=2.6.8->ten
Requirement already satisfied: pyasn1<0.5.0,>=0.4.6 in /usr/local/lib/python3.9/dist-packages (from pyasn1-modules>=0.2.1->google-auth
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.9/dist-packages (from requests-oauthlib>=0.7.0->google-auth-
```

Double-click (or enter) to edit

```
import seaborn as sns
import numpy as np
import pandas as pd
import matplotlib
from matplotlib import pyplot as plt
from sklearn import svm, datasets
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.model_selection import KFold
import pickle
import gzip
from google.colab import drive
import random
import keras
from keras.layers import Flatten,Dense,Conv2D,MaxPooling2D,UpSampling2D,Dropout
from keras.models import Model,load_model
from keras import Input,Sequential
from keras.layers import BatchNormalization
from keras.optimizers import RMSprop
from keras.utils import to_categorical
import tensorflow as tf
from keras.datasets import fashion_mnist
from keras.applications import VGG16
from keras.applications import VGG19
from keras.applications import ResNet50V2
from keras.applications import MobileNet
from keras.applications import EfficientNetB0
from keras.applications.mobilenet import preprocess_input
```

```
import cv2
import gc
from sklearn import preprocessing
```

```
tf.config.run_functions_eagerly(True)
```

```
drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

```
!tar -xvf "/content/gdrive/MyDrive/ML_A3/cifar-100-python.tar.gz"
```

```
cifar-100-python/
cifar-100-python/file.txt~
cifar-100-python/train
cifar-100-python/test
cifar-100-python/meta
```

```
with open("/content/cifar-100-python/train", 'rb') as data:
  train = pickle.load(data, encoding='bytes')

with open("/content/cifar-100-python/test", 'rb') as data:
  test = pickle.load(data, encoding='bytes')

with open("/content/cifar-100-python/meta", 'rb') as data:
  meta = pickle.load(data, encoding='bytes')
```

```
print(train.keys())
print(len(train[b'data']))
print(train[b'data'][0])
print(len(train[b'data'][0]))
```

```
dict_keys([b'filenames', b'batch_label', b'fine_labels', b'coarse_labels', b'data'])
50000
[255 255 255 ...  10  59  79]
3072
```

```
superclasses = np.unique(np.concatenate((np.array(train[b'coarse_labels']),np.array(test[b'coarse_labels'])),axis=0))
print(superclasses)
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19]
```

```
random.seed(12223)

selected_classes = random.sample(list(superclasses), 5)
print(selected_classes)
```

```
[2, 8, 10, 18, 0]
```

```
dict_of_indices = {}
for i in selected_classes:
  dict_of_indices[i]=[]
  for j in range(len(train[b'coarse_labels'])):
    if train[b'coarse_labels'][j]==i:
      dict_of_indices[i].append(j)
print(dict_of_indices)
```

```
{2: [12, 22, 70, 126, 141, 164, 167, 174, 221, 225, 246, 274, 314, 318, 324, 330, 336, 352, 362, 419, 445, 456, 460, 466, 472, 498, 5
```

```
x_train = []
y_train = np.array([])
x_indices_present = []
while len(x_train)!=900:
  classChosen=selected_classes[random.randrange(len(selected_classes))]
  elementChosen = dict_of_indices[classChosen][random.randrange(len(dict_of_indices[classChosen]))]
  if elementChosen not in x_indices_present:
    y_train=np.concatenate((y_train,np.array([classChosen])),axis=0)
    x_train.append(np.array(train[b'data'][elementChosen]))
    x_indices_present.append(elementChosen)
x_train = np.array(x_train)
dict_of_indices = {}
for i in selected_classes:
  dict_of_indices[i]=[]
  for j in range(len(test[b'coarse_labels'])):
```

```
    if test[b'coarse_labels'][j]==i:
      dict_of_indices[i].append(j)
x_test = []
y_test = np.array([])
x_indices_present = []
while len(x_test)!=100:
  classChosen=selected_classes[random.randrange(len(selected_classes))]
  elementChosen = dict_of_indices[classChosen][random.randrange(len(dict_of_indices[classChosen]))]
  if elementChosen not in x_indices_present:
    y_test=np.concatenate((y_test,np.array([classChosen])),axis=0)
    x_test.append(np.array(test[b'data'][elementChosen]))
    x_indices_present.append(elementChosen)
x_test = np.array(x_test)
```

```
print(type(x_train[0]))
print(len(x_train))
print(len(y_train))
print(len(x_test))
print(len(y_test))
```

```
<class 'numpy.ndarray'>
900
900
100
100
```

```
x_train = x_train / np.max(x_train)
x_test = x_test / np.max(x_test)
```

```
x_train = x_train.reshape(-1,3,32,32).transpose(0, 2, 3, 1)
x_test = x_test.reshape(-1,3,32,32).transpose(0, 2, 3, 1)
```

```
print(x_train.shape)
print(x_test.shape)
```

```
(900, 32, 32, 3)
(100, 32, 32, 3)
```

```
from tensorflow.keras.layers import Conv2D, BatchNormalization, MaxPooling2D

def encoder(input_tensor):
    # First convolutional layer
    conv1 = Conv2D(filters=32, # 32 output channels
                   kernel_size=(3,3), # 3x3 kernel size
                   activation='relu', # ReLU activation function
                   padding='same')(input_tensor) # Same padding
    # Batch normalization layer after conv1
    bn1 = BatchNormalization()(conv1)
    # Max pooling layer after bn1
    maxpool1 = MaxPooling2D(pool_size=(2,2), padding='same')(bn1)

    # Second convolutional layer
    conv2 = Conv2D(filters=64, kernel_size=(3,3), activation='relu', padding='same')(maxpool1)
    # Batch normalization layer after conv2
    bn2 = BatchNormalization()(conv2)

    # Third convolutional layer
    conv3 = Conv2D(filters=128, kernel_size=(3,3), activation='relu', padding='same')(bn2)
    # Batch normalization layer after conv3
    bn3 = BatchNormalization()(conv3)
    # Max pooling layer after bn3
    maxpool2 = MaxPooling2D(pool_size=(2,2), padding='same')(bn3)

    # Fourth convolutional layer
    conv4 = Conv2D(filters=256, kernel_size=(3,3), activation='relu', padding='same')(maxpool2)
    # Batch normalization layer after conv4
    bn4 = BatchNormalization()(conv4)

    # Final convolutional layer to encode the input image

    encoded_img = Conv2D(filters=16, kernel_size=(3,3), activation='sigmoid', padding='same')(conv4_layer)
    # Use sigmoid activation for the last layer to squash the pixel values between 0 and 1, making the output a binary image

    # Return the encoded image
    return encoded_img


def my_decoder(encoded_img):
    # Decoder network
    conv1_layer = Conv2D(filters=16, kernel_size=(3,3), activation='relu', padding='same')(encoded_img)
```

```
    conv1_layer = BatchNormalization()(conv1_layer)

    conv2_layer = Conv2D(filters=32, kernel_size=(3,3), activation='relu', padding='same')(conv1_layer)
    conv2_layer = BatchNormalization()(conv2_layer)

    # Upsampling layer 1
    upsampled_layer1 = UpSampling2D(size=(2,2))(conv2_layer)

    conv3_layer = Conv2D(filters=64, kernel_size=(3,3), activation='relu', padding='same')(upsampled_layer1)
    conv3_layer = BatchNormalization()(conv3_layer)

    conv4_layer = Conv2D(filters=128, kernel_size=(3,3), activation='relu', padding='same')(conv3_layer)
    conv4_layer = BatchNormalization()(conv4_layer)

    # Upsampling layer 2
    upsampled_layer2 = UpSampling2D(size=(2,2))(conv4_layer)

    decoded_img = Conv2D(filters=3, kernel_size=(3,3), activation='sigmoid', padding='same')(upsampled_layer2)
    # Use sigmoid activation for the last layer to squash the pixel values between 0 and 1, making the output a binary image

    # Return the decoded image
    return decoded_img


autoEnc = my_decoder(encoder())
print(autoEnc.summary())
autoEnc.compile(optimizer = RMSprop(),loss="mean_squared_error",metrics=['mean_squared_error'])
```

```
 max_pooling2d (MaxPooling2D  (None, 16, 16, 64)       0
 )

 conv2d_3 (Conv2D)           (None, 16, 16, 64)        36928

 batch_normalization_3 (Batc  (None, 16, 16, 64)       256
 hNormalization)

 conv2d_4 (Conv2D)           (None, 16, 16, 32)        18464

 batch_normalization_4 (Batc  (None, 16, 16, 32)       128
 hNormalization)

 conv2d_5 (Conv2D)           (None, 16, 16, 16)        4624

 batch_normalization_5 (Batc  (None, 16, 16, 16)       64
 hNormalization)

 conv2d_6 (Conv2D)           (None, 16, 16, 16)        2320

 batch_normalization_6 (Batc  (None, 16, 16, 16)       64
 hNormalization)

 conv2d_7 (Conv2D)           (None, 16, 16, 32)        4640

 batch_normalization_7 (Batc  (None, 16, 16, 32)       128
 hNormalization)

 conv2d_8 (Conv2D)           (None, 16, 16, 64)        18496

 up_sampling2d (UpSampling2D  (None, 32, 32, 64)       0
 )

 batch_normalization_8 (Batc  (None, 32, 32, 64)       256
 hNormalization)

 conv2d_9 (Conv2D)           (None, 32, 32, 32)        18464

 batch_normalization_9 (Batc  (None, 32, 32, 32)       128
 hNormalization)

 conv2d_10 (Conv2D)          (None, 32, 32, 16)        4624

 batch_normalization_10 (Bat  (None, 32, 32, 16)       64
 chNormalization)

 conv2d_11 (Conv2D)          (None, 32, 32, 3)         435

 batch_normalization_11 (Bat  (None, 32, 32, 3)        12
 chNormalization)

=================================================================
Total params: 134,127
Trainable params: 133,353
Non-trainable params: 774
_________________________________________________________________
None
```

```
trackLoss = autoEnc.fit(x_train,x_train,batch_size=100,epochs=50,validation_split=0.3,callbacks = [
    keras.callbacks.ModelCheckpoint("autoencoder.h5",save_best_only=True),
    keras.callbacks.EarlyStopping(monitor="val_loss", patience=25, verbose=1),
])
```

```
    Epoch 4/50
    7/7 [==============================] - 1s 159ms/step - loss: 0.7525 - mean_squared_error: 0.7525 - val_loss: 0.0564 - val_mean_squared
    Epoch 5/50
    7/7 [==============================] - 1s 151ms/step - loss: 0.7227 - mean_squared_error: 0.7227 - val_loss: 0.0560 - val_mean_squared
    Epoch 6/50
    7/7 [==============================] - 1s 131ms/step - loss: 0.7004 - mean_squared_error: 0.7004 - val_loss: 0.0574 - val_mean_squared
    Epoch 7/50
    7/7 [==============================] - 1s 156ms/step - loss: 0.6799 - mean_squared_error: 0.6799 - val_loss: 0.0530 - val_mean_squared
    Epoch 8/50
    7/7 [==============================] - 1s 133ms/step - loss: 0.6615 - mean_squared_error: 0.6615 - val_loss: 0.0605 - val_mean_squared
    Epoch 9/50
    7/7 [==============================] - 1s 131ms/step - loss: 0.6358 - mean_squared_error: 0.6358 - val_loss: 0.0573 - val_mean_squared
    Epoch 10/50
    7/7 [==============================] - 1s 143ms/step - loss: 0.6024 - mean_squared_error: 0.6024 - val_loss: 0.0594 - val_mean_squared
    Epoch 11/50
    7/7 [==============================] - 1s 167ms/step - loss: 0.5389 - mean_squared_error: 0.5389 - val_loss: 0.0605 - val_mean_squared
    Epoch 12/50
    7/7 [==============================] - 1s 184ms/step - loss: 0.4597 - mean_squared_error: 0.4597 - val_loss: 0.0606 - val_mean_squared
    Epoch 13/50
    7/7 [==============================] - 1s 183ms/step - loss: 0.4218 - mean_squared_error: 0.4218 - val_loss: 0.0618 - val_mean_squared
    Epoch 14/50
    7/7 [==============================] - 1s 130ms/step - loss: 0.3926 - mean_squared_error: 0.3926 - val_loss: 0.0605 - val_mean_squared
    Epoch 15/50
    7/7 [==============================] - 1s 130ms/step - loss: 0.3692 - mean_squared_error: 0.3692 - val_loss: 0.0675 - val_mean_squared
    Epoch 16/50
    7/7 [==============================] - 1s 132ms/step - loss: 0.3520 - mean_squared_error: 0.3520 - val_loss: 0.0664 - val_mean_squared
    Epoch 17/50
    7/7 [==============================] - 1s 147ms/step - loss: 0.3563 - mean_squared_error: 0.3563 - val_loss: 0.0603 - val_mean_squared
    Epoch 18/50
    7/7 [==============================] - 1s 142ms/step - loss: 0.3265 - mean_squared_error: 0.3265 - val_loss: 0.0637 - val_mean_squared
    Epoch 19/50
    7/7 [==============================] - 1s 142ms/step - loss: 0.3159 - mean_squared_error: 0.3159 - val_loss: 0.0612 - val_mean_squared
    Epoch 20/50
    7/7 [==============================] - 1s 131ms/step - loss: 0.3038 - mean_squared_error: 0.3038 - val_loss: 0.0658 - val_mean_squared
    Epoch 21/50
    7/7 [==============================] - 1s 130ms/step - loss: 0.2926 - mean_squared_error: 0.2926 - val_loss: 0.0658 - val_mean_squared
    Epoch 22/50
    7/7 [==============================] - 1s 132ms/step - loss: 0.2818 - mean_squared_error: 0.2818 - val_loss: 0.0621 - val_mean_squared
    Epoch 23/50
    7/7 [==============================] - 1s 131ms/step - loss: 0.2722 - mean_squared_error: 0.2722 - val_loss: 0.0785 - val_mean_squared
    Epoch 24/50
    7/7 [==============================] - 1s 155ms/step - loss: 0.2618 - mean_squared_error: 0.2618 - val_loss: 0.0687 - val_mean_squared
    Epoch 25/50
    7/7 [==============================] - 1s 174ms/step - loss: 0.2547 - mean_squared_error: 0.2547 - val_loss: 0.0691 - val_mean_squared
    Epoch 26/50
    7/7 [==============================] - 1s 190ms/step - loss: 0.2470 - mean_squared_error: 0.2470 - val_loss: 0.0634 - val_mean_squared
    Epoch 27/50
    7/7 [==============================] - 1s 136ms/step - loss: 0.2481 - mean_squared_error: 0.2481 - val_loss: 0.0625 - val_mean_squared
    Epoch 28/50
    7/7 [==============================] - 1s 143ms/step - loss: 0.2382 - mean_squared_error: 0.2382 - val_loss: 0.0624 - val_mean_squared
    Epoch 29/50
    7/7 [==============================] - 1s 142ms/step - loss: 0.2202 - mean_squared_error: 0.2202 - val_loss: 0.0641 - val_mean_squared
    Epoch 30/50
    7/7 [==============================] - 1s 142ms/step - loss: 0.2204 - mean_squared_error: 0.2204 - val_loss: 0.0709 - val_mean_squared
    Epoch 31/50
    7/7 [==============================] - 1s 130ms/step - loss: 0.2083 - mean_squared_error: 0.2083 - val_loss: 0.0709 - val_mean_squared
    Epoch 32/50
    7/7 [==============================] - 1s 143ms/step - loss: 0.2045 - mean_squared_error: 0.2045 - val_loss: 0.0666 - val_mean_squared
    Epoch 32: early stopping
```

```
with open('autoencoder.pkl','wb') as data:
  pickle.dump(trackLoss.history,data)
```

```
with open('autoencoder.pkl','rb') as data:
  trackLoss = pickle.load(data)
```

```
plt.plot(trackLoss['val_loss'])
plt.plot(trackLoss['loss'])
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend(['Validation Loss', 'Train Loss'])
plt.show()
```


```
autoEnc = load_model("autoencoder.h5")
```

```
len(np.unique(y_train))
```

    5

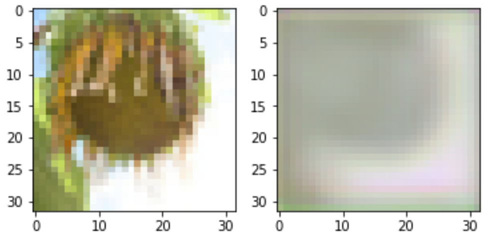
```
indexes_taken = {}
for i in selected_classes:
  indexes_taken[i]=[]
  for j in range(len(y_test)):
    if y_test[j]==i:
      indexes_taken[i].append(j)
print(indexes_taken)
```

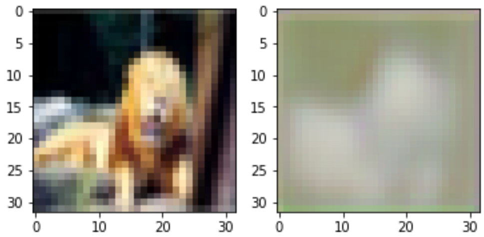
    {2: [0, 13, 19, 24, 26, 28, 32, 34, 43, 46, 48, 58, 60, 67, 69, 70, 71, 73, 82, 92, 97], 8: [10, 14, 15, 17, 18, 22, 31, 37, 44, 57, 5

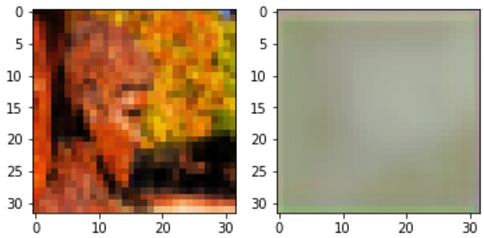
```
y_pred = autoEnc.predict(x_test)
```

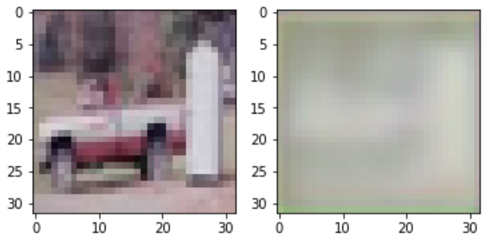
    4/4 [==============================] - 0s 64ms/step

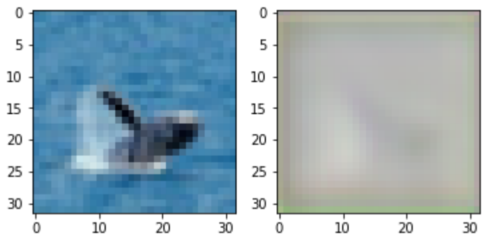
```
y_pred = y_pred/np.max(y_pred)
for j in selected_classes:
  fig, ax = plt.subplots(1,2)
  indexChosen = indexes_taken[j][random.randrange(len(indexes_taken[j]))]
  ax[0].imshow(x_test[indexChosen])
  ax[1].imshow(y_pred[indexChosen])
  plt.show()
```











```
print(np.min(y_pred))
```

```
0.41315737
```

```
dict_of_indices = {}
for i in range(len(selected_classes)):
  dict_of_indices[selected_classes[i]] = i
print(dict_of_indices)
```

```
{2: 0, 8: 1, 10: 2, 18: 3, 0: 4}
```

```
y_train_labelled = []
for i in y_train:
  y_train_labelled.append(dict_of_indices[i])
y_test_labelled = []
for i in y_test:
  y_test_labelled.append(dict_of_indices[i])
print(len(y_train_labelled))
print(len(y_test_labelled))
```

```
900
100
```

```
y_train_labelled = to_categorical(y_train_labelled)
y_test_labelled = to_categorical(y_test_labelled)
```

```
print(y_test_labelled)
```

```
 [0. 0. 1. 0. 0.]
 [1. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0.]
 [0. 0. 1. 0. 0.]
 [1. 0. 0. 0. 0.]
 [0. 0. 0. 1. 0.]
 [1. 0. 0. 0. 0.]
 [0. 0. 0. 1. 0.]
 [0. 0. 0. 1. 0.]
 [0. 0. 1. 0. 0.]
 [0. 0. 0. 0. 1.]
 [0. 0. 0. 0. 1.]
 [0. 0. 0. 1. 0.]
 [0. 0. 0. 1. 0.]
 [0. 0. 1. 0. 0.]
 [0. 1. 0. 0. 0.]
 [1. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0.]
 [1. 0. 0. 0. 0.]
 [0. 0. 1. 0. 0.]
 [0. 1. 0. 0. 0.]
 [0. 0. 1. 0. 0.]
 [0. 0. 0. 0. 1.]
 [0. 0. 1. 0. 0.]
 [0. 0. 0. 0. 1.]
 [1. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1.]
 [1. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0.]
 [0. 0. 1. 0. 0.]
 [1. 0. 0. 0. 0.]
 [0. 0. 1. 0. 0.]
 [0. 1. 0. 0. 0.]
 [0. 0. 1. 0. 0.]
 [0. 0. 0. 1. 0.]
 [0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 1.]
 [0. 0. 1. 0. 0.]
 [0. 1. 0. 0. 0.]
 [1. 0. 0. 0. 0.]
 [0. 0. 1. 0. 0.]
 [0. 0. 0. 1. 0.]
 [0. 0. 0. 1. 0.]
 [0. 0. 0. 1. 0.]
 [0. 1. 0. 0. 0.]
 [0. 0. 0. 0. 1.]
 [0. 0. 1. 0. 0.]
 [0. 1. 0. 0. 0.]
 [0. 0. 0. 1. 0.]
 [1. 0. 0. 0. 0.]
 [0. 0. 1. 0. 0.]
 [0. 1. 0. 0. 0.]
 [0. 0. 0. 0. 1.]
 [0. 0. 1. 0. 0.]
 [1. 0. 0. 0. 0.]
 [0. 0. 1. 0. 0.]
 [0. 1. 0. 0. 0.]]
```

```
def classifier(cl):
  cl.add(Flatten())
  cl.add(Dense(128, activation='relu'))
  cl.add(Dense(64, activation='relu'))
  cl.add(Dense(5, activation='softmax'))
  return cl
```

```
classifier = classifier(encoding())
```

```
print(classifier.summary())
```

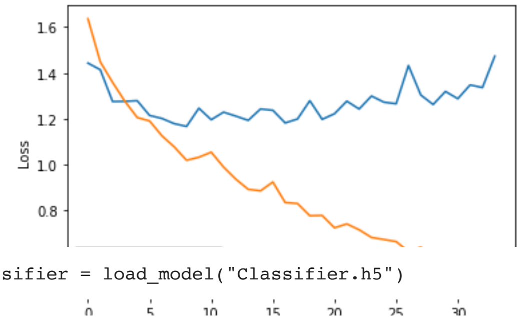
```
Model: "sequential_17"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_12 (Conv2D)          (None, 32, 32, 16)        448

 batch_normalization_12 (Bat  (None, 32, 32, 16)       64
 chNormalization)

 conv2d_13 (Conv2D)          (None, 32, 32, 32)        4640

 batch_normalization_13 (Bat  (None, 32, 32, 32)       128
 chNormalization)

 conv2d_14 (Conv2D)          (None, 32, 32, 64)        18496

 batch_normalization_14 (Bat  (None, 32, 32, 64)       256
 chNormalization)

 max_pooling2d_4 (MaxPooling  (None, 16, 16, 64)       0
 2D)

 conv2d_15 (Conv2D)          (None, 16, 16, 64)        36928

 batch_normalization_15 (Bat  (None, 16, 16, 64)       256
 chNormalization)

 conv2d_16 (Conv2D)          (None, 16, 16, 32)        18464

 batch_normalization_16 (Bat  (None, 16, 16, 32)       128
 chNormalization)

 conv2d_17 (Conv2D)          (None, 16, 16, 16)        4624

 batch_normalization_17 (Bat  (None, 16, 16, 16)       64
 chNormalization)

 flatten_5 (Flatten)         (None, 4096)              0

 dense_15 (Dense)            (None, 128)               524416

 dense_16 (Dense)            (None, 64)                8256

 dense_17 (Dense)            (None, 5)                 325

=================================================================
Total params: 617,493
Trainable params: 617,045
Non-trainable params: 448
_________________________________________________________________
None
```

```
for i in range(13):
  classifier.layers[i].set_weights(autoEnc.layers[i].get_weights())
  classifier.layers[i].trainable = False
```

```
classifier.compile(optimizer = 'adam',loss="categorical_crossentropy",metrics=['accuracy'])
```

```
with open('classifier.pkl','rb') as data:
  trackLossClasifier = pickle.load(data)
```

```
plt.plot(trackLossClasifier['val_loss'])
plt.plot(trackLossClasifier['loss'])
plt.legend(['Validation Loss', 'Train Loss'])
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.show()
```




```
classifier = load_model("Classifier.h5")
```

```
print("Training accuracy is "+str((classifier.evaluate(x_train,y_train_labelled))[1]*100))
print("Test accuracy is "+str((classifier.evaluate(x_test,y_test_labelled))[1]*100))
```

```
29/29 [==============================] - 2s 51ms/step - loss: 1.0244 - accuracy: 0.6056
Training accuracy is 60.5555534362793
4/4 [==============================] - 0s 62ms/step - loss: 1.1475 - accuracy: 0.6000
Test accuracy is 60.00000238418579
```

Q3 [P ‖ CO3 & C4] SVM (10 points)

Use the following SVM strategies to classify sapodillas from kiwi for the dataset available here1 . For each strategy, plot the learned decision boundary. Use an appropriate evaluation metric to present the test scores. (a) Linear SVM (b) Polynomial SVM (c) Kernel SVM Finally, write the individual and combined inferences of the obtained results.

```
df = pd.read_csv('/content/gdrive/MyDrive/ML_A3/Sapodillas_and_Kiwis_MLA3.csv')
```

```
x = df[['Weight','Size']]

y = df['Class']

x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.3,random_state=15)
x_of_df = x.copy()
y_of_df = y.copy()
```

```
# normalising the data
for col in x_train.columns:
  if col=='Class':
    continue
  # print(col)
  x_test[col]=(x_test[col] - x_train[col].min()) / (x_train[col].max()- x_train[col].min())
  x_of_df[col]=(x_of_df[col] - x_train[col].min()) / (x_of_df[col].max()- x_train[col].min())
  x_train[col]=(x_train[col] - x_train[col].min()) / (x_train[col].max()- x_train[col].min())

# normalising the data based on the training data
```

```
print(x_train)
# printing the training data
print(y_train)
```

```
    Weight      Size
1      0.4  0.112360
2      0.0  0.044944
39     0.8  0.573034
6      0.5  0.870787
26     0.2  0.000000
34     0.3  0.460674
38     0.5  0.887640
37     0.9  0.825843
33     0.8  0.651685
9      0.0  0.146067
4      0.2  0.387640
29     0.6  0.702247
10     0.8  1.000000
36     0.4  0.421348
23     0.3  0.039326
13     0.3  0.258427
17     1.0  0.696629
15     0.0  0.264045
22     0.4  0.337079
21     0.5  0.460674
11     0.5  0.820225
7      1.0  0.617978
27     0.5  0.117978
28     0.9  0.696629
0      0.4  0.213483
5      0.8  0.938202
```

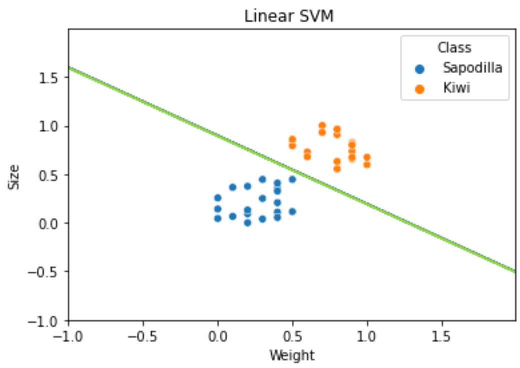
```
12      0.9   0.853933
8       0.9   0.758427
1     Sapodilla
2     Sapodilla
39         Kiwi
6          Kiwi
26    Sapodilla
34    Sapodilla
38         Kiwi
37         Kiwi
33         Kiwi
9     Sapodilla
4     Sapodilla
29         Kiwi
10         Kiwi
36    Sapodilla
23    Sapodilla
13    Sapodilla
17         Kiwi
15    Sapodilla
22    Sapodilla
21    Sapodilla
11         Kiwi
7          Kiwi
27    Sapodilla
28         Kiwi
0     Sapodilla
5          Kiwi
12         Kiwi
8          Kiwi
Name: Class, dtype: object
```

```
classifier = svm.SVC(kernel ='linear', C = 1).fit(x_train, y_train)

y_pred = classifier.predict(x_test)
linear_acc_score = accuracy_score(y_test, y_pred)
print("Accuracy score for linear SVM "+str(linear_acc_score))
```

```
Accuracy score for linear SVM 1.0
```

```
min_wt, max_wt = x_of_df['Weight'].min() - 1, x_of_df['Weight'].max() + 1
min_size, max_size = x_of_df['Size'].min() - 1, x_of_df['Size'].max() + 1
x_axis_val, y_axis_val = np.meshgrid(np.arange(min_wt, max_wt, 0.005), np.arange(min_size, max_size, 0.005))
boundary = classifier.predict(np.c_[x_axis_val.ravel(), y_axis_val.ravel()])

boundary = preprocessing.LabelEncoder().fit_transform(boundary)
boundary = boundary.reshape(x_axis_val.shape)
plt.contour(x_axis_val, y_axis_val, boundary)
plt.xlabel('Weight')
plt.ylabel('Size')
plt.title('Linear SVM ')
sns.scatterplot(x=x_of_df['Weight'], y=x_of_df['Size'], hue=y)

plt.show()
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but SVC was fitted with
  warnings.warn(
```




```
print(np.array(y_pred))
print(np.array(y_test))
```

```
['Kiwi' 'Sapodilla' 'Sapodilla' 'Sapodilla' 'Sapodilla' 'Kiwi' 'Kiwi'
 'Kiwi' 'Sapodilla' 'Kiwi' 'Kiwi' 'Sapodilla']
['Kiwi' 'Sapodilla' 'Sapodilla' 'Sapodilla' 'Sapodilla' 'Kiwi' 'Kiwi'
 'Kiwi' 'Sapodilla' 'Kiwi' 'Kiwi' 'Sapodilla']
```

```
from sklearn.model_selection import KFold
from sklearn import svm
from sklearn.metrics import accuracy_score
import numpy as np

# Set up a KFold cross-validation object with 5 splits
kFold = KFold(n_splits=5)
```

```
kFold = KFold(n_splits=5)

# Initialize an empty list to store the accuracy scores for each fold
accuracy_scores = []

# Iterate through each fold
for i, (train_index, test_index) in enumerate(kFold.split(x_of_df)):

  # Initialize empty lists to store the training and testing data for this fold
  x_train_k = []
  y_train_k = []
  x_test_k = []
  y_test_k = []

  # Retrieve the training and testing data for this fold from the input data
  for j in train_index:
    x_train_k.append(np.array(x_of_df.loc[j]))
    y_train_k.append(np.array(y_of_df.loc[j]))
  for j in test_index:
    x_test_k.append(np.array(x_of_df.loc[j]))
    y_test_k.append(np.array(y_of_df.loc[j]))

  # Convert the training and testing data into numpy arrays
  x_train_k = np.array(x_train_k)
  y_train_k = np.array(y_train_k)

  # Train a linear SVM model on the training data for this fold
  classifier = svm.SVC(kernel='linear', C=1).fit(x_train_k, y_train_k)

  # Use the trained model to predict the labels for the testing data for


  y_pred_k = classifier.predict(x_test_k)
  accuracy_scores.append(accuracy_score(y_test_k, y_pred_k))

print("5- fold cross validation accuracy score of linear SVM model is "+str(sum(accuracy_scores)/len(accuracy_scores)))
```

```
    5- fold cross validation accuracy score of linear SVM model is 1.0
```

```
classifier = svm.SVC(kernel ='poly', C = 1).fit(x_train, y_train)

y_pred = classifier.predict(x_test)
polynomial_acc_score = accuracy_score(y_test, y_pred)
print("Accuracy score for polynomial SVM "+str(polynomial_acc_score))
```

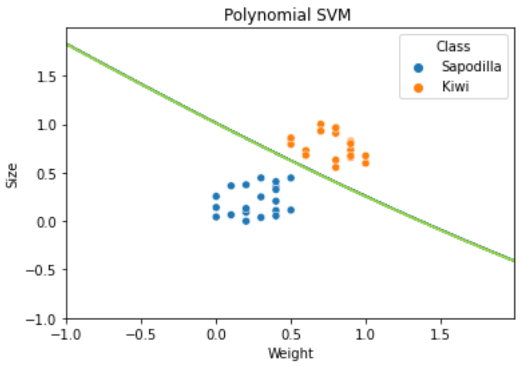
```
    Accuracy score for polynomial SVM 1.0
```

```
kFold = KFold(n_splits=5)

accuracy_scores=[]
for i, (train_index, test_index) in enumerate(kFold.split(x_of_df)):
  x_train_k= []
  y_train_k = []
  x_test_k= []
  y_test_k = []
  for j in train_index:

    x_train_k.append(np.array(x_of_df.loc[j]))

    y_train_k.append(np.array(y_of_df.loc[j]))
  for j in test_index:
    x_test_k.append(np.array(x_of_df.loc[j]))

    y_test_k.append(np.array(y_of_df.loc[j]))
  x_train_k = np.array(x_train_k)
  y_train_k = np.array(y_train_k)
  classifier = svm.SVC(kernel ='poly', C = 1).fit(x_train_k, y_train_k)

  y_pred_k = classifier.predict(x_test_k)
  accuracy_scores.append(accuracy_score(y_test_k, y_pred_k))

print("5- fold cross validation accuracy score of polynomial SVM model is "+str(sum(accuracy_scores)/len(accuracy_scores)))
```

```
    5- fold cross validation accuracy score of polynomial SVM model is 1.0
```

```
min_wt, max_wt = x_of_df['Weight'].min() - 1, x_of_df['Weight'].max() + 1
min_size, max_size = x_of_df['Size'].min() - 1, x_of_df['Size'].max() + 1
x_axis_val, y_axis_val = np.meshgrid(np.arange(min_wt, max_wt, 0.005), np.arange(min_size, max_size, 0.005))
boundary = classifier.predict(np.c_[x_axis_val.ravel(), y_axis_val.ravel()])

boundary = preprocessing.LabelEncoder().fit_transform(boundary)
```

```
boundary = boundary.reshape(x_axis_val.shape)
plt.contour(x_axis_val, y_axis_val, boundary)
plt.xlabel('Weight')
plt.ylabel('Size')
plt.title('Polynomial SVM ')
sns.scatterplot(x=x_of_df['Weight'], y=x_of_df['Size'], hue=y)

plt.show()
```



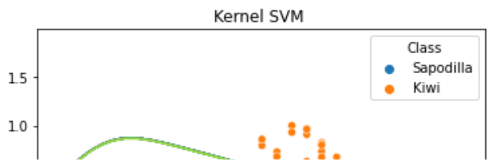

```
classifier = svm.SVC(kernel ='rbf', C = 1).fit(x_train, y_train)

y_pred = classifier.predict(x_test)
kernel_acc_score = accuracy_score(y_test, y_pred)
print("Accuracy score for ref kernel SVM "+str(kernel_acc_score))
```

```
Accuracy score for ref kernel SVM 1.0
```

```
kFold = KFold(n_splits=5)

accuracy_scores=[]
for i, (train_index, test_index) in enumerate(kFold.split(x_of_df)):
  x_train_k= []
  y_train_k = []
  x_test_k= []
  y_test_k = []
  for j in train_index:
    x_train_k.append(np.array(x_of_df.loc[j]))
    y_train_k.append(np.array(y_of_df.loc[j]))
  for j in test_index:
    x_test_k.append(np.array(x_of_df.loc[j]))

    y_test_k.append(np.array(y_of_df.loc[j]))
  x_train_k = np.array(x_train_k)
  y_train_k = np.array(y_train_k)
  classifier = svm.SVC(kernel ='rbf', C = 1).fit(x_train_k, y_train_k)

  y_pred_k = classifier.predict(x_test_k)
  accuracy_scores.append(accuracy_score(y_test_k, y_pred_k))

print("5- fold cross validation accuracy score of polynomial SVM model is "+str(sum(accuracy_scores)/len(accuracy_scores)))
```

```
5- fold cross validation accuracy score of polynomial SVM model is 1.0
```

```
min_wt, max_wt = x_of_df['Weight'].min() - 1, x_of_df['Weight'].max() + 1
min_size, max_size = x_of_df['Size'].min() - 1, x_of_df['Size'].max() + 1
x_axis_val, y_axis_val = np.meshgrid(np.arange(min_wt, max_wt, 0.005), np.arange(min_size, max_size, 0.005))
boundary = classifier.predict(np.c_[x_axis_val.ravel(), y_axis_val.ravel()])

boundary = preprocessing.LabelEncoder().fit_transform(boundary)
boundary = boundary.reshape(x_axis_val.shape)
plt.contour(x_axis_val, y_axis_val, boundary)
plt.xlabel('Weight')
plt.ylabel('Size')
plt.title('Kernel SVM ')
sns.scatterplot(x=x_of_df['Weight'], y=x_of_df['Size'], hue=y)

plt.show()
```




```
print("The accuracy for kernel SVM "+str(kernel_acc_score))
print("The accuracy for polynomial SVM "+str(polynomial_acc_score))
print("The accuracy for linear SVM is "+str(linear_acc_score))
```

```
The accuracy for kernel SVM 1.0
The accuracy for polynomial SVM 1.0
The accuracy for linear SVM is 1.0
```

Q4 [P ‖ CO3 & C4] Transfer Learning (20 points)

Use the FashionMNIST dataset (https://keras.io/api/datasets/fashion_mnist/) from Keras to build one classifier each based on the following pre-trained architectures trained on IMAGENET dataset (reference: https://keras.io/api/applications/). Remove the last layer, add your own few dense layers and the output layer. Train only these added layers from scratch on the data. Use an appropriate evaluation metric.

```
(x_train,y_train),(x_test,y_test)= fashion_mnist.load_data()
```

```
print(x_train.shape)
```

```
(60000, 28, 28)
```

```
print(x_train.shape)
```

```
(60000, 28, 28)
```

```
x_train_final=[]
x_test_final=[]
for i in x_train:
  x_train_final.append(cv2.resize(i, (56, 56)))
for i in x_test:
  x_test_final.append(cv2.resize(i, (56, 56)))
x_train_final = np.array(x_train_final)
x_test_final = np.array(x_test_final)
```

```
print(x_train_final.shape)
```

```
(60000, 56, 56)
```

```
x_train_final = np.dstack([x_train_final] * 3)
x_test_final = np.dstack([x_test_final] * 3)
```

```
x_train_final = x_train_final.reshape(-1,56,56,3)
x_test_final = x_test_final.reshape(-1,56,56,3)
```

```
print(x_train_final.shape)
```

```
(60000, 56, 56, 3)
```

```
print(np.unique(y_train))
print(np.unique(y_test))
```

```
[0 1 2 3 4 5 6 7 8 9]
[0 1 2 3 4 5 6 7 8 9]
```

```
y_train_labelled = to_categorical(y_train)
y_test_labelled = to_categorical(y_test)
```

VGG16

```
modVgg16 = Sequential()
modVgg16.add(VGG16(include_top=False,weights='imagenet',input_shape=(56,56, 3)))

for i in range(len(modVgg16.layers)):
  modVgg16.layers[i].trainable = False
print(modVgg16.summary())
```

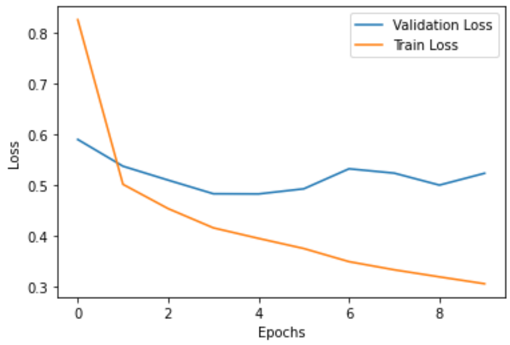
```
Model: "sequential_18"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 1, 1, 512)         14714688

=================================================================
Total params: 14,714,688
Trainable params: 0
Non-trainable params: 14,714,688
_________________________________________________________________
None
```

```
modVgg16.add(Flatten())
modVgg16.add(Dense(128, activation='relu'))
modVgg16.add(Dense(64, activation='relu'))
modVgg16.add(Dense(10, activation='softmax'))
print(modVgg16.summary())
```

```
Model: "sequential_18"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 1, 1, 512)         14714688

 flatten_6 (Flatten)         (None, 512)               0

 dense_18 (Dense)            (None, 128)               65664

 dense_19 (Dense)            (None, 64)                8256

 dense_20 (Dense)            (None, 10)                650

=================================================================
Total params: 14,789,258
Trainable params: 74,570
Non-trainable params: 14,714,688
_________________________________________________________________
None
```

```
modVgg16.compile(optimizer = 'adam',loss="categorical_crossentropy",metrics=['accuracy'])
```

```
with open('VGG16classifier.pkl','rb') as data:
  trackLossClasifierVgg16 = pickle.load(data)
plt.plot(trackLossClasifierVgg16['val_loss'])
plt.plot(trackLossClasifierVgg16['loss'])
plt.legend(['Validation Loss', 'Train Loss'])
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.show()
```




```
modVgg16 = load_model("VGG16Classifier.h5")
val_vgg16 = (modVgg16.evaluate(x_test_final,y_test_labelled))[1]*100
print("Training accuracy is "+str((modVgg16.evaluate(x_train_final,y_train_labelled))[1]*100))
print("Test accuracy is "+str((modVgg16.evaluate(x_test_final,y_test_labelled))[1]*100))
```

```
/usr/local/lib/python3.9/dist-packages/tensorflow/python/data/ops/structured_function.py:256: UserWarning: Even though the `tf.config.
  warnings.warn(
313/313 [==============================] - 15s 47ms/step - loss: 0.4988 - accuracy: 0.8281
1875/1875 [==============================] - 55s 29ms/step - loss: 0.3874 - accuracy: 0.8611
Training accuracy is 86.1050009727478
```

```
313/313 [==============================] - 9s 28ms/step - loss: 0.4988 - accuracy: 0.8281
Test accuracy is 82.81000256538391
```

```
gc.collect()
```

```
37843
```

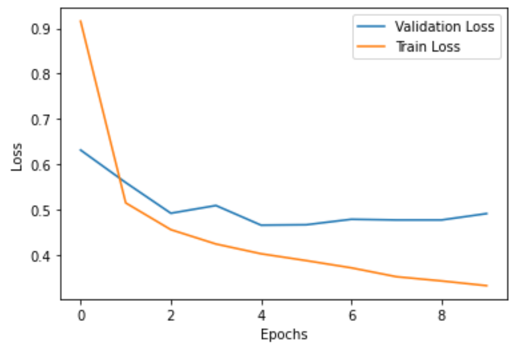
VGG19

```
# Initialize a sequential model in Keras
modVgg19 = Sequential()

# Add the VGG19 model with pretrained weights from ImageNet, and freeze all of its layers
modVgg19.add(VGG19(include_top=False,weights='imagenet',input_shape=(56,56, 3)))
for i in range(len(modVgg19.layers)):
    modVgg19.layers[i].trainable = False

# Print a summary of the model architecture
print(modVgg19.summary())

# Add a flattening layer to convert the output of the VGG19 model into a 1D array
modVgg19.add(Flatten())

# Add a fully connected layer with 128 neurons and ReLU activation
modVgg19.add(Dense(128, activation='relu'))

# Add a second fully connected layer with 64 neurons and ReLU activation
modVgg19.add(Dense(64, activation='relu'))

# Add a final fully connected layer with 10 neurons and softmax activation
modVgg19.add(Dense(10, activation='softmax'))

# Print a summary of the model architecture
print(modVgg19.summary())

# Compile the model using the Adam optimizer, categorical crossentropy loss, and accuracy as a metric
modVgg19.compile(optimizer = 'adam',loss="categorical_crossentropy",metrics=['accuracy'])
```

```
Model: "sequential_19"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 vgg19 (Functional)          (None, 1, 1, 512)         20024384  
                                                                 
=================================================================
Total params: 20,024,384
Trainable params: 0
Non-trainable params: 20,024,384
_________________________________________________________________
None
Model: "sequential_19"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 vgg19 (Functional)          (None, 1, 1, 512)         20024384  
                                                                 
 flatten_7 (Flatten)         (None, 512)               0         
                                                                 
 dense_21 (Dense)            (None, 128)               65664     
                                                                 
 dense_22 (Dense)            (None, 64)                8256      
                                                                 
 dense_23 (Dense)            (None, 10)                650       
                                                                 
=================================================================
Total params: 20,098,954
Trainable params: 74,570
Non-trainable params: 20,024,384
_________________________________________________________________
None
```

```
import pickle
import matplotlib.pyplot as plt

# Load the training history data from a pickle file
with open('VGG19classifier.pkl', 'rb') as f:
    trackLossClassifierVgg19 = pickle.load(f)

# Plot the validation and training loss curves over time
plt.plot(trackLossClassifierVgg19['val_loss'])
plt.plot(trackLossClassifierVgg19['loss'])
```

```
plt.legend(['Validation Loss', 'Train Loss'])
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.show()
```




```
modVgg19 = load_model("VGG19Classifier.h5")
val_vgg19 = (modVgg19.evaluate(x_test_final,y_test_labelled))[1]*100
print("Training accuracy is "+str((modVgg19.evaluate(x_train_final,y_train_labelled))[1]*100))
print("Test accuracy is "+str((modVgg19.evaluate(x_test_final,y_test_labelled))[1]*100))
```

```
313/313 [==============================] - 11s 36ms/step - loss: 0.4882 - accuracy: 0.8282
1875/1875 [==============================] - 60s 32ms/step - loss: 0.3922 - accuracy: 0.8569
Training accuracy is 85.69166660308838
313/313 [==============================] - 10s 33ms/step - loss: 0.4882 - accuracy: 0.8282
Test accuracy is 82.81999826431274
```

```
gc.collect()
```

```
7954
```

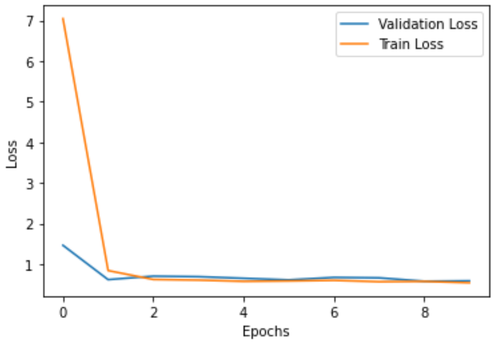
ResNet50V2

```
modResNet50V2 = Sequential()
modResNet50V2.add(ResNet50V2(include_top=False,weights='imagenet',input_shape=(56,56, 3)))
for i in range(len(modResNet50V2.layers)):
  modResNet50V2.layers[i].trainable = False
print(modResNet50V2.summary())
modResNet50V2.add(Flatten())
modResNet50V2.add(Dense(128, activation='relu'))
modResNet50V2.add(Dense(64, activation='relu'))
modResNet50V2.add(Dense(10, activation='softmax'))
print(modResNet50V2.summary())
modResNet50V2.compile(optimizer = 'adam',loss="categorical_crossentropy",metrics=['accuracy'])
```

```
Model: "sequential_20"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 resnet50v2 (Functional)     (None, 2, 2, 2048)        23564800

=================================================================
Total params: 23,564,800
Trainable params: 0
Non-trainable params: 23,564,800
_________________________________________________________________
None
Model: "sequential_20"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 resnet50v2 (Functional)     (None, 2, 2, 2048)        23564800

 flatten_8 (Flatten)         (None, 8192)              0

 dense_24 (Dense)            (None, 128)               1048704

 dense_25 (Dense)            (None, 64)                8256

 dense_26 (Dense)            (None, 10)                650

=================================================================
Total params: 24,622,410
Trainable params: 1,057,610
Non-trainable params: 23,564,800
```

```
None
```

```
with open('ResNet50V2Classifier.pkl','rb') as data:
  trackLossClasifierResNet50V2 = pickle.load(data)
plt.plot(trackLossClasifierResNet50V2['val_loss'])
plt.plot(trackLossClasifierResNet50V2['loss'])
plt.legend(['Validation Loss', 'Train Loss'])
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.show()
modResNet50V2 = load_model("ResNet50V2Classifier.h5")
val_ResNet50V2 = (modResNet50V2.evaluate(x_test_final,y_test_labelled))[1]*100
print("Training accuracy is "+str((modResNet50V2.evaluate(x_train_final,y_train_labelled))[1]*100))
print("Test accuracy is "+str(val_ResNet50V2))
```




```
313/313 [==============================] - 36s 115ms/step - loss: 0.6195 - accuracy: 0.7887
1875/1875 [==============================] - 205s 109ms/step - loss: 0.5268 - accuracy: 0.8132
Training accuracy is 81.32166862487793
Test accuracy is 78.86999845504761
```

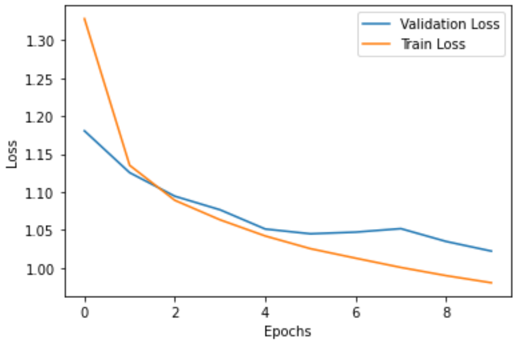
MobileNet

```
modMobileNet = Sequential()
modMobileNet.add(MobileNet(include_top=False,weights='imagenet',input_shape=(56,56, 3)))
for i in range(len(modMobileNet.layers)):
  modMobileNet.layers[i].trainable = False
print(modMobileNet.summary())
modMobileNet.add(Flatten())
modMobileNet.add(Dense(128, activation='relu'))
modMobileNet.add(Dense(64, activation='relu'))
modMobileNet.add(Dense(10, activation='softmax'))
print(modMobileNet.summary())
modMobileNet.compile(optimizer = 'adam',loss="categorical_crossentropy",metrics=['accuracy'])
```

```
WARNING:tensorflow:`input_shape` is undefined or non-square, or `rows` is not in [128, 160, 192, 224]. Weights for input shape (224,
Model: "sequential_21"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 mobilenet_1.00_224 (Functio  (None, 1, 1, 1024)       3228864
 nal)

=================================================================
Total params: 3,228,864
Trainable params: 0
Non-trainable params: 3,228,864
_________________________________________________________________
None
Model: "sequential_21"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 mobilenet_1.00_224 (Functio  (None, 1, 1, 1024)       3228864
 nal)

 flatten_9 (Flatten)         (None, 1024)              0

 dense_27 (Dense)            (None, 128)               131200

 dense_28 (Dense)            (None, 64)                8256

 dense_29 (Dense)            (None, 10)                650

=================================================================
Total params: 3,368,970
Trainable params: 140,106
Non-trainable params: 3,228,864
```

```
None
```

```
with open('MobileNetClassifier.pkl','rb') as data:
  trackLossClasifierMobileNet = pickle.load(data)
plt.plot(trackLossClasifierMobileNet['val_loss'])
plt.plot(trackLossClasifierMobileNet['loss'])
plt.legend(['Validation Loss', 'Train Loss'])
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.show()
modMobileNet = load_model("MobileNetClassifier.h5")
val_MobileNet = (modMobileNet.evaluate(preprocess_input(x_test_final),y_test_labelled))[1]*100
print("Training accuracy is "+str((modMobileNet.evaluate(preprocess_input(x_train_final),y_train_labelled))[1]*100))
print("Test accuracy is "+str(val_MobileNet))
```




```
313/313 [==============================] - 20s 65ms/step - loss: 1.0337 - accuracy: 0.6310
1875/1875 [==============================] - 120s 64ms/step - loss: 0.9794 - accuracy: 0.6532
Training accuracy is 65.3166651725769
Test accuracy is 63.099998235702515
```

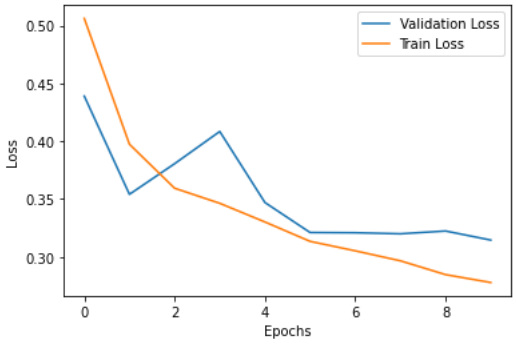
EfficientNetB0

```
from tensorflow.keras.applications import EfficientNetB0
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, Dense

# Initialize a sequential model in Keras
modEfficientNetB0 = Sequential()

# Add the EfficientNetB0 model with pretrained weights from ImageNet, and freeze all of its layers
modEfficientNetB0.add(EfficientNetB0(include_top=False,weights='imagenet',input_shape=(56,56, 3)))
for i in range(len(modEfficientNetB0.layers)):
    modEfficientNetB0.layers[i].trainable = False

# Print a summary of the model architecture
print(modEfficientNetB0.summary())

# Add a flattening layer to convert the output of the EfficientNetB0 model into a 1D array
modEfficientNetB0.add(Flatten())

# Add a fully connected layer with 128 neurons and ReLU activation
modEfficientNetB0.add(Dense(128, activation='relu'))

# Add a second fully connected layer with 64 neurons and ReLU activation
modEfficientNetB0.add(Dense(64, activation='relu'))

# Add a final fully connected layer with 10 neurons and softmax activation
modEfficientNetB0.add(Dense(10, activation='softmax'))

# Print a summary of the model architecture
print(modEfficientNetB0.summary())

# Compile the model using the Adam optimizer, categorical crossentropy loss, and accuracy as a metric
modEfficientNetB0.compile(optimizer = 'adam',loss="categorical_crossentropy",metrics=['accuracy'])
```

```
Model: "sequential_22"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 efficientnetb0 (Functional)  (None, 2, 2, 1280)       4049571

=================================================================
Total params: 4,049,571
```

```
Trainable params: 0
Non-trainable params: 4,049,571
_________________________________________________________________
None
Model: "sequential_22"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 efficientnetb0 (Functional)  (None, 2, 2, 1280)       4049571

 flatten_10 (Flatten)        (None, 5120)              0

 dense_30 (Dense)            (None, 128)               655488

 dense_31 (Dense)            (None, 64)                8256

 dense_32 (Dense)            (None, 10)                650

=================================================================
Total params: 4,713,965
Trainable params: 664,394
Non-trainable params: 4,049,571
_________________________________________________________________
None
```

```
with open('EfficientNetB0Classifier.pkl','rb') as data:
  trackLossClasifierEfficientNetB0 = pickle.load(data)

plt.plot(trackLossClasifierEfficientNetB0['val_loss'])
plt.plot(trackLossClasifierEfficientNetB0['loss'])
plt.legend(['Validation Loss', 'Train Loss'])
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.show()
modEfficientNetB0.load_weights("EfficientNetB0Classifier.h5")
val_EfficientNetB0 = (modEfficientNetB0.evaluate(x_test_final,y_test_labelled))[1]*100
print("Training accuracy is "+str((modEfficientNetB0.evaluate(x_train_final,y_train_labelled))[1]*100))
print("Test accuracy is "+str(val_EfficientNetB0))
```




```
313/313 [==============================] - 51s 162ms/step - loss: 0.3369 - accuracy: 0
1875/1875 [==============================] - 301s 161ms/step - loss: 0.2578 - accuracy
Training accuracy is 90.41333198547363
Test accuracy is 87.80999779701233
```

```
print("The accuracy for VGG16 "+str(val_vgg16))
print("The accuracy for VGG19 "+str(val_vgg19))
print("The accuracy for ResNet50V2 "+str(val_ResNet50V2))
print("The accuracy for MobileNet "+str(val_MobileNet))
print("The accuracy for EfficientNetB0 "+str(val_EfficientNetB0))
```

```
The accuracy for VGG16 82.81000256538391
The accuracy for VGG19 82.81999826431274
The accuracy for ResNet50V2 78.86999845504761
The accuracy for MobileNet 63.099998235702515
The accuracy for EfficientNetB0 87.80999779701233
```

+ Code + Text