In [ ]:

```
#!pip install wandb
#import wandb
#wandb.init()
```

In [ ]:

```
import os
import cv2
import math
import random
import numpy as np
import datetime as dt
import tensorflow as tf
from tensorflow import keras


import matplotlib.pyplot as plt
%matplotlib inline

from sklearn.model_selection import train_test_split

from tensorflow.keras.layers import *
from tensorflow.keras.models import Sequential
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.utils import plot_model

from keras.applications.vgg16 import VGG16
from keras.applications.vgg16 import preprocess_input
```

In [ ]:

```
from google.colab import drive
drive.mount('/gdrive')
%cd /gdrive
```

```
Drive already mounted at /gdrive; to attempt to forcibly remount, call drive.mount
("/gdrive", force_remount=True).
/gdrive
```

In [ ]:

```
%cd 'My Drive'

%cd 'Action Recognition'
```

```
/gdrive/My Drive
/gdrive/My Drive/Action Recognition
```

In [ ]:

```
image_height, image_width = 64, 64
images_per_class = 8000
dataset_directory = "hmdb51"
classes_list = ["pullup", "punch", "dive", "fencing", "ride_bike", "golf"]
model_output_size = len(classes_list)
```

In [ ]:

```
def frames_extraction(video_path):
    frames_list = []
    video_reader = cv2.VideoCapture(video_path)
```

In ...

```
def create_dataset():


    temp_features = []
    features = []
    labels = []


    for class_index, class_name in enumerate(classes_list):
        print(f'Extracting Data of Class: {class_name}')


        files_list = os.listdir(os.path.join(dataset_directory, class_name))


        for file_name in files_list:
            video_file_path = os.path.join(dataset_directory, class_name, file_n
            frames = frames_extraction(video_file_path)
            temp_features.extend(frames)
        features.extend(random.sample(temp_features, images_per_class))
        labels.extend([class_index] * images_per_class)
        temp_features.clear()

    features = np.asarray(features)
    labels = np.array(labels)

    return features, labels
```

In [ ]:

```
features, labels = create_dataset()
```

```
Extracting Data of Class: pullup
Defected frame
Defected frame
Defected frame
Defected frame
Defected frame
Defected frame
Defected frame
Defected frame
Defected frame
Defected frame
Defected frame
Defected frame
Defected frame
Defected frame
Defected frame
Defected frame
Defected frame
Defected frame
Defected frame
Defected frame
Defected frame
Defected frame
Defected frame
Defected frame
Defected frame
```

In [ ]:

```
seed_constant = 23
np.random.seed(seed_constant)
random.seed(seed_constant)
tf.random.set_seed(seed_constant)
```

I...

```
print (features.shape)
print (labels.shape)

one_hot_encoded_labels = to_categorical(labels)

features_train, features_test, labels_train, labels_test = train_test_split(featur

print (features_train.shape)
print (labels_train.shape)
```

```
(48000, 64, 64, 3)
(48000,)
(38400, 64, 64, 3)
(38400, 6)
```

In [ ]:

```
print (labels_train)
```

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

In...

```
# Load model
base_model = Sequential()
base_model.add(VGG16(input_shape=(64,64,3), weights='imagenet', include_top=Fals
base_model.add(Dense(288, activation = 'relu'))
base_model.add(Dense(288, activation = 'relu'))
base_model.add(Dense(6, activation='softmax'))
# summarize the model

base_model.layers[0].trainable = False
```

In ...

```
base_model.compile(optimizer='sgd',loss='categorical_crossentropy',metrics=['acc
```

In [ ]:

```
plot_model(base_model,show_shapes = True, show_layer_names = True)
```

Out[ ]:

I...

```
# Adding Early Stopping Callback
early_stopping_callback = EarlyStopping(monitor = 'val_loss', patience = 10, mode

# Start Training
model_training_history = base_model.fit(x = features_train, y = labels_train, epoc
Epoch 1/40
1920/1920 [==============================] - 35s 10ms/step - loss: 0.1594 - accurac
y: 0.9585 - val_loss: 0.0309 - val_accuracy: 0.9908
Epoch 2/40
1920/1920 [==============================] - 18s 9ms/step - loss: 0.0101 - accuracy:
0.9976 - val_loss: 0.0147 - val_accuracy: 0.9960
Epoch 3/40
1920/1920 [==============================] - 18s 10ms/step - loss: 0.0030 - accurac
y: 0.9998 - val_loss: 0.0113 - val_accuracy: 0.9966
Epoch 4/40
1920/1920 [==============================] - 18s 10ms/step - loss: 0.0015 - accurac
y: 1.0000 - val_loss: 0.0096 - val_accuracy: 0.9975
Epoch 5/40
1920/1920 [==============================] - 18s 10ms/step - loss: 9.9968e-04 - accu
racy: 1.0000 - val_loss: 0.0088 - val_accuracy: 0.9975
Epoch 6/40
1920/1920 [==============================] - 18s 9ms/step - loss: 7.3064e-04 - accur
acy: 1.0000 - val_loss: 0.0080 - val_accuracy: 0.9979
Epoch 7/40
1920/1920 [==============================] - 18s 9ms/step - loss: 5.8856e-04 - accur
acy: 1.0000 - val_loss: 0.0077 - val_accuracy: 0.9978
Epoch 8/40
1920/1920 [==============================] - 18s 10ms/step - loss: 4.8512e-04 - accu
racy: 1.0000 - val_loss: 0.0077 - val_accuracy: 0.9978
Epoch 9/40
1920/1920 [==============================] - 18s 9ms/step - loss: 4.1695e-04 - accur
acy: 1.0000 - val_loss: 0.0075 - val_accuracy: 0.9977
Epoch 10/40
1920/1920 [==============================] - 18s 9ms/step - loss: 3.6671e-04 - accur
acy: 1.0000 - val_loss: 0.0071 - val_accuracy: 0.9978
Epoch 11/40
1920/1920 [==============================] - 18s 9ms/step - loss: 3.2759e-04 - accur
acy: 1.0000 - val_loss: 0.0070 - val_accuracy: 0.9979
Epoch 12/40
1920/1920 [==============================] - 18s 9ms/step - loss: 2.9161e-04 - accur
acy: 1.0000 - val_loss: 0.0071 - val_accuracy: 0.9978
Epoch 13/40
1920/1920 [==============================] - 18s 9ms/step - loss: 2.6644e-04 - accur
acy: 1.0000 - val_loss: 0.0070 - val_accuracy: 0.9980
Epoch 14/40
1920/1920 [==============================] - 18s 10ms/step - loss: 2.4417e-04 - accu
racy: 1.0000 - val_loss: 0.0068 - val_accuracy: 0.9979
Epoch 15/40
1920/1920 [==============================] - 18s 9ms/step - loss: 2.2375e-04 - accur
acy: 1.0000 - val_loss: 0.0069 - val_accuracy: 0.9980
Epoch 16/40
1920/1920 [==============================] - 18s 9ms/step - loss: 2.0753e-04 - accur
acy: 1.0000 - val_loss: 0.0067 - val_accuracy: 0.9979
Epoch 17/40
1920/1920 [==============================] - 18s 9ms/step - loss: 1.9307e-04 - accur
acy: 1.0000 - val_loss: 0.0068 - val_accuracy: 0.9980
Epoch 18/40
1920/1920 [==============================] - 18s 10ms/step - loss: 1.8017e-04 - accu
racy: 1.0000 - val_loss: 0.0067 - val_accuracy: 0.9979
Epoch 19/40
1920/1920 [==============================] - 18s 9ms/step - loss: 1.6935e-04 - accur
acy: 1.0000 - val_loss: 0.0066 - val_accuracy: 0.9980
Epoch 20/40
1920/1920 [==============================] - 18s 9ms/step - loss: 1.5937e-04 - accur
```

In [ ]:

```
model_evaluation_history = base_model.evaluate(features_test, labels_test)

from sklearn.metrics import classification_report

y_pred = base_model.predict(features_test, batch_size=4, verbose=1)
y_pred_bool = np.argmax(y_pred, axis=1)

l_test=np.argmax(labels_test, axis=1)

print(classification_report(l_test, y_pred_bool))
```

```
300/300 [==============================] - 4s 11ms/step - loss: 0.0114 - accuracy:
0.9972
2400/2400 [==============================] - 9s 4ms/step
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      1604
           1       1.00      1.00      1.00      1615
           2       1.00      0.99      1.00      1593
           3       0.99      1.00      0.99      1585
           4       1.00      1.00      1.00      1602
           5       1.00      1.00      1.00      1601

    accuracy                           1.00      9600
   macro avg       1.00      1.00      1.00      9600
weighted avg       1.00      1.00      1.00      9600
```

In [ ]:

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(l_test, y_pred_bool)

print (cm)
```

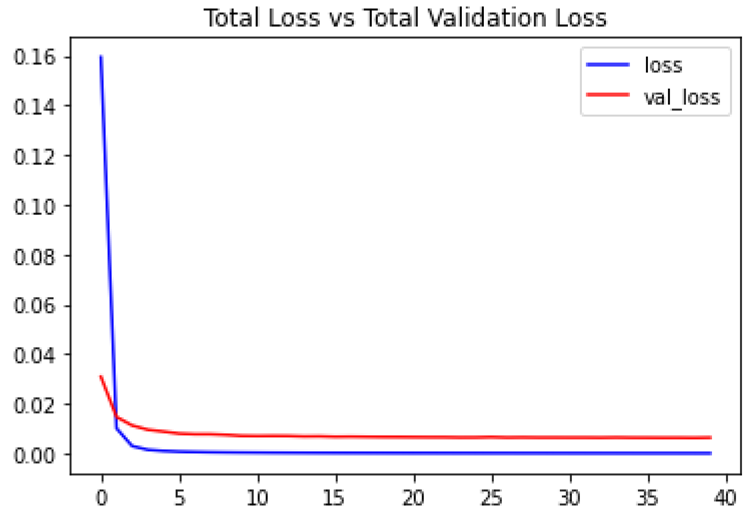
```
[[1604    0    0    0    0    0]
 [   1 1607    2    5    0    0]
 [   0    3 1585    2    1    2]
 [   0    5    2 1578    0    0]
 [   0    0    2    2 1598    0]
 [   0    0    0    0    0 1601]]
```

In [ ]:

```
def plot_metric(metric_name_1, metric_name_2, plot_name):
  # Get Metric values using metric names as identifiers
  metric_value_1 = model_training_history.history[metric_name_1]
  metric_value_2 = model_training_history.history[metric_name_2]

  # Constructing a range object which will be used as time
  epochs = range(len(metric_value_1))

  # Plotting the Graph
  plt.plot(epochs, metric_value_1, 'blue', label = metric_name_1)
  plt.plot(epochs, metric_value_2, 'red', label = metric_name_2)

  # Adding title to the plot
  plt.title(str(plot_name))

  # Adding legend to the plot
  plt.legend()
```

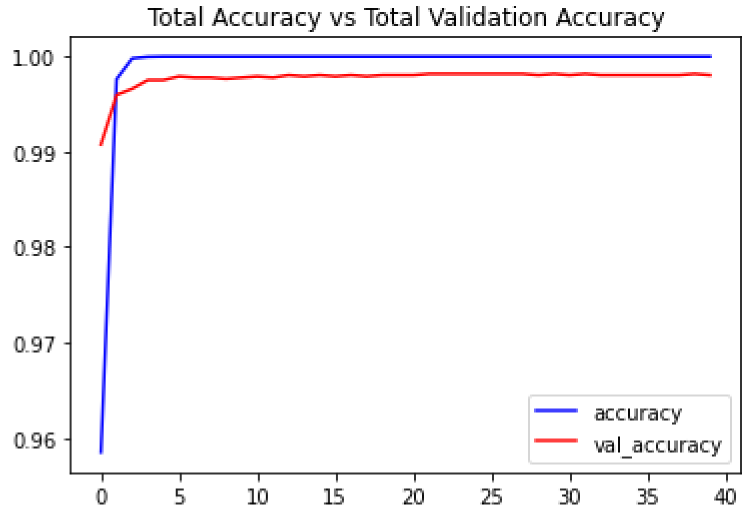
In [ ]:

```
plot_metric('loss', 'val_loss', 'Total Loss vs Total Validation Loss')
```



In ...

```
plot_metric('accuracy', 'val_accuracy', 'Total Accuracy vs Total Validation Accu
```



In [ ]:

```
from collections import Counter

def get_first_mode(a):
    c = Counter(a)
    mode_count = max(c.values())
    mode = {key for key, count in c.items() if count == mode_count}
    first_mode = next(x for x in a if x in mode)
    return first_mode
```

In [ ]:

```
def frames_extraction2(video_path):
    frames_list = []

    vidObj = cv2.VideoCapture(video_path)
    skip_frames=30



    # Used as counter variable
```

In ...

```
#Evaluating a different dataset

from tqdm import tqdm
from statistics import mode


predict = []
actual = []
dataset_directory2="UCF50"

temp_features = []
features = []
labels = []

cc=0

for class_index, class_name in enumerate(classes_list):
    print(f'Extracting Data of Class: {class_name}')

    files_list = os.listdir(os.path.join(dataset_directory2, class_name))

    for file_name in files_list:

        video_file_path = os.path.join(dataset_directory2, class_name, file_name

        frames = frames_extraction2(video_file_path)

        temppred=[]

        for i in frames:
          temppred.append(base_model.predict_classes(np.expand_dims(i, axis = 0)

        print (temppred)
        print ("mode", get_first_mode(temppred), cc)
        cc+=1
        predict.append(get_first_mode(temppred))
        actual.append(class_index)
```

```
Extracting Data of Class: pullup
Defected frame
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/engine/sequential.py:
455: UserWarning: `model.predict_classes()` is deprecated and will be removed after
2021-01-01. Please use instead:* `np.argmax(model.predict(x), axis=-1)`,   if your m
odel does multi-class classification   (e.g. if it uses a `softmax` last-layer activ
ation).* `(model.predict(x) > 0.5).astype("int32")`,   if your model does binary cla
ssification   (e.g. if it uses a `sigmoid` last-layer activation).
  warnings.warn('`model.predict_classes()` is deprecated and '
[0, 0, 0, 0, 0, 5, 0]
mode 0 0
Defected frame
[0, 3, 2, 3, 2, 2]
mode 2 1
Defected frame
[2, 2, 2, 2, 2, 2, 2, 1]
mode 2 2
Defected frame
```

In [26]:

```
print(classification_report(actual, predict))
```

```
              precision    recall  f1-score   support

           0       0.91      0.57      0.70       120
           1       0.62      0.28      0.39       160
           2       0.47      0.96      0.63       153
           3       0.31      0.21      0.25       111
           4       0.70      0.83      0.76       145
           5       0.76      0.66      0.71       142

    accuracy                           0.60       831
   macro avg       0.63      0.59      0.57       831
weighted avg       0.63      0.60      0.58       831
```

In [27]:

```
print(confusion_matrix(actual, predict))
```

```
[[ 68   9  16   9  11   7]
 [  5  45  60  34  14   2]
 [  0   2 147   0   4   0]
 [  1  12  44  23  15  16]
 [  0   1  18   1 121   4]
 [  1   3  29   8   7  94]]
```