In [ ]:

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

In …

```
#Importing Modules

#Importing Keras and import sub-modules needed
import keras
from keras import applications
from keras.preprocessing.image import ImageDataGenerator
from keras import optimizers
from keras.models import Sequential, Model
from keras.layers import *
from keras.callbacks import ModelCheckpoint, LearningRateScheduler, TensorBoard,

#Importing miscallaneous modules
import os
import cv2
import numpy as np
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

 #Importing sklearn modules to calculate different metrics and create different
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import cohen_kappa_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import multilabel_confusion_matrix
from tensorflow.keras.utils import to_categorical
```

I…

```
from google.colab import drive    #Access Google Drive which is used as location
drive.mount('/gdrive')
#go to root of Google Drive
%cd /gdrive
```

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

In [ ]:

```
#Navigate to folder where all the datasets are
%cd 'My Drive'
%cd 'Action Recognition'
```

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

In…

```
data_dir = "hmdb51/" #Choose dataset by naming dataset folder name
img_height , img_width = 64, 64 #Set pixel values for frames
seq_len = 70 #Set number of frames/samples per video
classes = ["pullup", "punch", "dive", "fencing", "ride_bike", "golf"] #Select cla
```

I…

In ...

```
X, Y = create_data(data_dir)                                          #Fetch data for

#print (X.shape)
#print (Y.shape)
```

```
['dribble', 'draw_sword', 'chew', 'dive', 'catch', 'climb_stairs', 'climb', 'clap',
'cartwheel', 'brush_hair', 'drink', 'fall_floor', 'jump', 'hug', 'eat', 'hit', 'flic
_flac', 'handstand', 'golf', 'fencing', 'kick', 'kick_ball', 'kiss', 'punch', 'laug
h', 'pick', 'pour', 'pushup', 'pullup', 'push', 'ride_horse', 'ride_bike', 'run', 's
hake_hands', 'shoot_gun', 'somersault', 'situp', 'stand', 'shoot_bow', 'smoke', 'si
t', 'shoot_ball', 'smile', 'walk', 'sword', 'throw', 'turn', 'swing_baseball', 'swor
d_exercise', 'wave', 'talk']
pullup
Defected frame
Defected frame
Defected frame
punch
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
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
```

I…
```
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.20, shuffle=

print (X_train.shape)
print (y_train.shape)
print (X_test.shape)
print (y_test.shape)
```

```
(344, 70, 64, 64, 3)
(344, 6)
(86, 70, 64, 64, 3)
(86, 6)
```

I…
```
model = Sequential()  #Model initiated and layers added whilst specifying hyperpar

model.add(ConvLSTM2D(filters = 64, kernel_size = (3, 3), return_sequences = False,
model.add(Dropout(0.2))

model.add(Conv2D(filters = 128, kernel_size = (3, 3), activation = 'relu'))
model.add(Dropout(0.2))

model.add(Flatten())

model.add(Dense(256, activation="relu"))
model.add(Dense(256, activation="relu"))
model.add(Dropout(0.3))

model.add(Dense(6, activation = "softmax"))
model.summary()  #Print summary of model

opt = keras.optimizers.SGD(lr=0.001)    #Specify training algorithm and learning r
model.compile(loss='categorical_crossentropy', optimizer=opt, metrics=["accuracy"]
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv_lst_m2d (ConvLSTM2D)    (None, 62, 62, 64)        154624
_________________________________________________________________
dropout (Dropout)            (None, 62, 62, 64)        0
_________________________________________________________________
conv2d (Conv2D)              (None, 60, 60, 128)       73856
_________________________________________________________________
dropout_1 (Dropout)          (None, 60, 60, 128)       0
_________________________________________________________________
flatten (Flatten)            (None, 460800)            0
_________________________________________________________________
dense (Dense)                (None, 256)               117965056
_________________________________________________________________
dense_1 (Dense)              (None, 256)               65792
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 6)                 1542
=================================================================
Total params: 118,260,870
Trainable params: 118,260,870
```

l..

```
earlystop = EarlyStopping(monitor = 'val_loss', patience = 10, mode = 'min', resto
callbacks = [earlystop]

history = model.fit(x = X_train, y = y_train, epochs=40, batch_size = 8 , shuffle=
```

```
Epoch 1/40
35/35 [==============================] - 42s 618ms/step - loss: 1.7920 - accuracy:
0.2347 - val_loss: 1.8170 - val_accuracy: 0.0870
Epoch 2/40
35/35 [==============================] - 21s 589ms/step - loss: 1.6473 - accuracy:
0.3092 - val_loss: 1.4640 - val_accuracy: 0.3188
Epoch 3/40
35/35 [==============================] - 21s 589ms/step - loss: 1.3983 - accuracy:
0.4624 - val_loss: 1.7697 - val_accuracy: 0.3043
Epoch 4/40
35/35 [==============================] - 21s 589ms/step - loss: 1.1795 - accuracy:
0.6249 - val_loss: 1.4504 - val_accuracy: 0.4348
Epoch 5/40
35/35 [==============================] - 21s 589ms/step - loss: 1.1470 - accuracy:
0.5786 - val_loss: 1.1726 - val_accuracy: 0.5797
Epoch 6/40
35/35 [==============================] - 21s 588ms/step - loss: 0.9869 - accuracy:
0.6695 - val_loss: 1.2644 - val_accuracy: 0.4928
Epoch 7/40
35/35 [==============================] - 21s 588ms/step - loss: 0.8593 - accuracy:
0.7069 - val_loss: 1.0360 - val_accuracy: 0.6377
Epoch 8/40
35/35 [==============================] - 21s 588ms/step - loss: 0.6390 - accuracy:
0.8102 - val_loss: 1.2429 - val_accuracy: 0.5507
Epoch 9/40
35/35 [==============================] - 21s 588ms/step - loss: 0.6633 - accuracy:
0.7116 - val_loss: 0.8459 - val_accuracy: 0.7101
Epoch 10/40
35/35 [==============================] - 21s 589ms/step - loss: 0.4606 - accuracy:
0.8476 - val_loss: 3.1645 - val_accuracy: 0.3043
Epoch 11/40
35/35 [==============================] - 21s 589ms/step - loss: 0.8911 - accuracy:
0.7713 - val_loss: 0.9375 - val_accuracy: 0.6812
Epoch 12/40
35/35 [==============================] - 21s 589ms/step - loss: 0.3961 - accuracy:
0.8862 - val_loss: 0.8197 - val_accuracy: 0.7536
Epoch 13/40
35/35 [==============================] - 21s 588ms/step - loss: 0.2840 - accuracy:
0.9068 - val_loss: 1.1775 - val_accuracy: 0.5362
Epoch 14/40
35/35 [==============================] - 21s 588ms/step - loss: 0.3360 - accuracy:
0.8886 - val_loss: 0.9045 - val_accuracy: 0.7246
Epoch 15/40
35/35 [==============================] - 21s 589ms/step - loss: 0.2408 - accuracy:
0.9493 - val_loss: 0.8943 - val_accuracy: 0.7391
Epoch 16/40
35/35 [==============================] - 21s 589ms/step - loss: 0.2404 - accuracy:
0.9146 - val_loss: 1.9742 - val_accuracy: 0.4783
Epoch 17/40
35/35 [==============================] - 21s 589ms/step - loss: 0.2938 - accuracy:
0.9193 - val_loss: 0.8334 - val_accuracy: 0.7391
Epoch 18/40
35/35 [==============================] - 21s 589ms/step - loss: 0.1470 - accuracy:
```

In ...
```
model_evaluation_history = model.evaluate(X_test, y_test)  #Evaluate model on te


from sklearn.metrics import classification_report #Produce report with extra met

y_pred = model.predict(X_test, batch_size=4, verbose=1)

y_pred = np.argmax(y_pred, axis = 1)
y_test = np.argmax(y_test, axis = 1)

print(classification_report(y_test, y_pred))
```

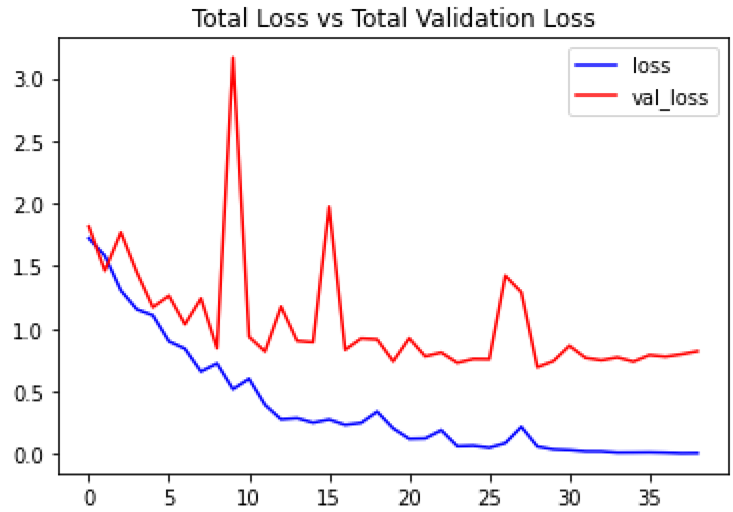
```
3/3 [==============================] - 2s 492ms/step - loss: 0.5912 - accuracy: 0.77
91
22/22 [==============================] - 3s 93ms/step
              precision    recall  f1-score   support

           0       0.74      0.91      0.82        22
           1       0.86      0.86      0.86         7
           2       0.33      0.21      0.26        14
           3       0.86      0.67      0.75         9
           4       0.80      0.94      0.86        17
           5       1.00      0.94      0.97        17

    accuracy                           0.78        86
   macro avg       0.76      0.75      0.75        86
weighted avg       0.76      0.78      0.76        86
```

In...
```
from sklearn.metrics import confusion_matrix  #Produce confusion matrix to show e
cm = confusion_matrix(y_test, y_pred)

print (cm)
```

```
[[20  0  0  0  2  0]
 [ 0  6  1  0  0  0]
 [ 7  1  3  1  2  0]
 [ 0  0  3  6  0  0]
 [ 0  0  1  0 16  0]
 [ 0  0  1  0  0 16]]
```

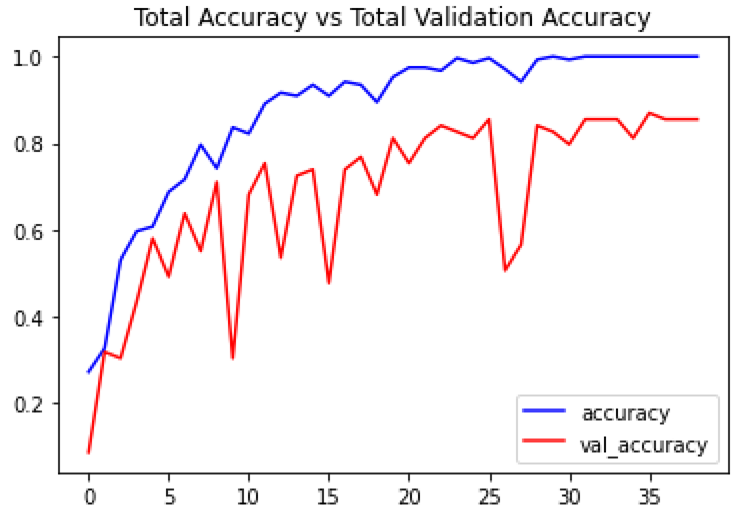
In...
```
def plot_metric(metric_name_1, metric_name_2, plot_name):
                                                                           # Fetch 
  metric_value_1 = history.history[metric_name_1]
  metric_value_2 = history.history[metric_name_2]

  epochs = range(len(metric_value_1))                                      # Get ep
                                                                           # Plot G
  plt.plot(epochs, metric_value_1, 'blue', label = metric_name_1)
  plt.plot(epochs, metric_value_2, 'red', label = metric_name_2)

  plt.title(str(plot_name))

  plt.legend()
```

In...
```
plot_metric('loss', 'val_loss', 'Total Loss vs Total Validation Loss')  #Plot Lo
```



I...

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



In ...

```
#evaluate on new different data set with similar classes

data_dir2 = "UCF50/"

X1, Y1 = create_data(data_dir2) #use previous helper functions to extract frames
```

```
['Billiards', 'BenchPress', 'BreastStroke', 'Drumming', 'CleanAndJerk', 'BaseballPit
ch', 'Basketball', 'HorseRiding', 'Kayaking', 'HighJump', 'JumpRope', 'JavelinThro
w', 'HulaHoop', 'JugglingBalls', 'HorseRace', 'JumpingJack', 'PoleVault', 'Lunges',
'MilitaryParade', 'PlayingViolin', 'PlayingGuitar', 'PizzaTossing', 'Mixing', 'Nunch
ucks', 'PlayingPiano', 'PlayingTabla', 'Skiing', 'SalsaSpin', 'PommelHorse', 'RockCl
imbingIndoor', 'SkateBoarding', 'Rowing', 'RopeClimbing', 'PushUps', 'Swing', 'Skije
t', 'SoccerJuggling', 'ThrowDiscus', 'WalkingWithDog', 'YoYo', 'TaiChi', 'TennisSwin
g', 'VolleyballSpiking', 'TrampolineJumping', 'golf', 'ride_bike', 'fencing', 'div
e', 'punch', 'pullup']
pullup
Defected frame
Defected frame
punch
Defected frame
Defected frame
Defected frame
```

In [ ]:

```
Eval_Hist = model.evaluate(X1, Y1)  #evaluate on whole set from new dataset
```

```
26/26 [==============================] - 14s 528ms/step - loss: 1.9524 - accuracy: 0.4386
```

In...

```
Y2 = model.predict(X1, batch_size=4, verbose=1)  #produce extra metrics for pred

Y2 = np.argmax(Y2, axis = 1)
Y1 = np.argmax(Y1, axis = 1)

print(classification_report(Y1, Y2))
```

```
206/206 [==============================] - 19s 92ms/step
              precision    recall  f1-score   support

           0       0.35      0.64      0.45       118
           1       0.33      0.06      0.11       156
           2       0.38      0.24      0.29       152
           3       0.07      0.03      0.04       111
           4       0.41      0.86      0.55       145
           5       0.82      0.79      0.81       141

    accuracy                           0.44       823
   macro avg       0.39      0.44      0.38       823
weighted avg       0.41      0.44      0.38       823
```

In ...

```
print(confusion_matrix(Y1, Y2))  #produce confusion matrix for prediction on new
```

```
[[ 76  11   2   2   8  19]
 [ 30  10   9  37  70   0]
 [ 71   2  36   1  42   0]
 [ 12   6  32   3  54   4]
 [  8   1  11   0 124   1]
 [ 20   0   4   0   5 112]]
```