FINAL EVALUATION PROJECT REPORT

On

Fight Detection in Public Places

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR THE AWARD OF THE DEGREE OF

BACHELOR OF ENGINEERING
(INFORMATION TECHNOLOGY)
7th Semester

Supervisor:
Dr. Neelam Goel
Assistant Professor

Submitted By:
Ekaant Garg (UE198039)
Hitesh Chawla (UE198046)

To

Department of Information Technology
University Institute of Engineering and Technology
Panjab University, Chandigarh

December, 2022

Acknowledgment

We would like to thank the Department of Information Technology, UIET, Panjab University, Chandigarh for allowing us to undertake this project during the 7th Semester.

It gives us a great sense of pleasure to present the report of the Project undertaken during B.E. Fourth Year. We have taken sincere efforts in this project. However, it would not have been possible without the kind support and help of many individuals. We would like to extend my sincere thanks to all of them.

We are highly thankful to Dr. Neelam Goel, Assistant Professor, UIET, for her guidance and constant supervision as well as for providing necessary information regarding the project.

Finally, we would like to thank Dr. Neelam Goel, Assistant Professor, UIET for evaluating us.

Declaration

We here by declare that the project report entitled “Fight Detection in Public Places” submitted by us to University Institute of Engineering and Technology, Panjab University, Chandigarh in partial fulfillment of the requirement for the award of the degree of B.E. in INFORMATION TECHNOLOGY is a record of project work carried out by us, under the guidance of Dr Neelam Goel. We further declare that the work reported in this project has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Date: 7 December 2022

Ekaant Garg UE198039
Hitesh Chawla UE198046

Table of Contents

Introduction of the problem

- India, being a developing country, encounters many day-to-day problems affecting the life of civilians to a great extent.
- As a result of rapid population growth, streets, gardens, malls and other public places are getting overcrowded.
- This causes mass rush, riots and other fights, which ultimately results in total loss of control.
- Fights often occur at large gatherings, events and other public places like concerts, fairs, stadiums, rallies and so on.
- In 2018 there were 76,851 rioting-related offenses across India, which when adjusted for a population, works out to be 5.7 riot offenses per 100,000 people. At some public places, police and security guards are appointed to take care of such fighting situations but they are not enough.
- There are high chances of mass panic, fights and accidents which require continuous surveillance and highly efficient fight detection.
- This type of fight prevention is not possible by traditional methods.
- Day by day modern technology is advancing and helping people to make their life easier. In recent years a large amount of surveillance cameras have been installed at various public places like traffic signals, offices, schools and so on. They are also being used during various public events like concerts, marriages, rallies, etc.
- The main focus of “FIGHT DETECTION IN PUBLIC PLACES” is on monitoring such fight incidents and automatically and effectively determine whether any fighting occurs or not.

- It aims at giving an early intimation to the authorized person with minimum human intervention.
- Using Machine Learning, the system is smart enough to detect fights properly and handle such affairs efficiently.

Technology Stacks Used

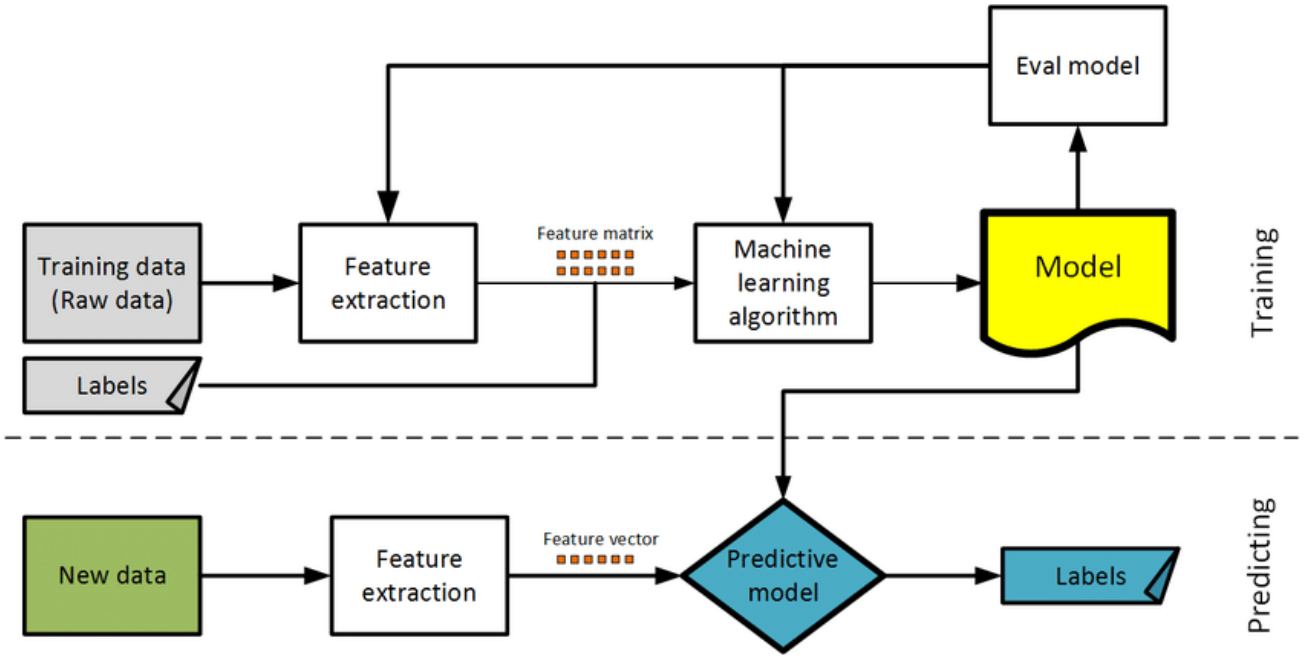
- Python
- Machine Learning
- Deep Learning
- Computer Vision
- Google collab
- Fast Api

Software/ System Analysis

- ## Requirement Gathering

- To propose a model that will automatically and effectively determine whether any fight is occurring or not, in real-time with focus on speed and accuracy.
- The primary objective is to help law enforcement agencies prevent breaking out of violence through real-time detection of fights and automatic security alerts.

- ## Design/use case diagrams/ Flowcharts



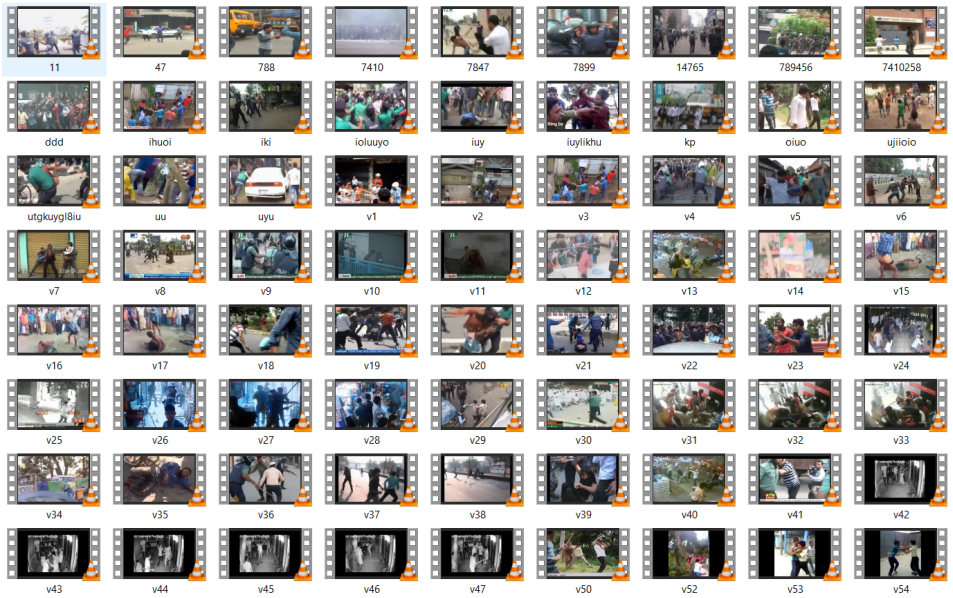
- Implementation

- About Dataset

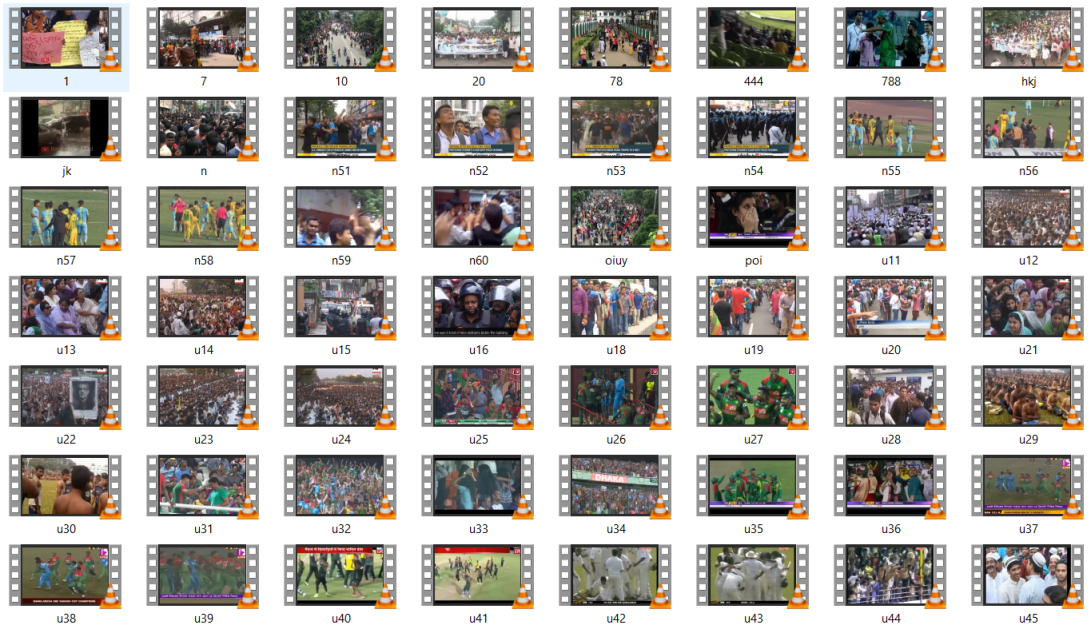
SOURCES	GOOGLE, YOUTUBE, NEWS
DATASET SIZE	4 GB (Approx.)
CATEGORIES	Fight, NoFight
NO. OF VIDEOS	2000 (Approx.)
NO. OF IMAGES	2000*15 = 30000(Approx.)
WIDTH	Random
HEIGHT	Random
EXTENSION	JPG, AVI
IMAGE SIZE	0 - 5 MB

Categories:

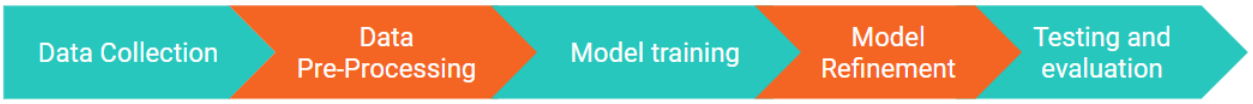
1. Fight (contains videos with severe fighting in various situations)



2. NoFight (which contains real life situations videos like eating, sports activity, singing, etc)



Proposed Methodology



1. Data Collection :

a. Collection of various videos of fighting and non-fighting scenes in public places.
b. Videos are manually classified into fight and Nofight categories.

2. Data Pre-Processing

a. Multiple Frames are extracted from videos.
b. Converting images into pixels from which the algorithm will learn.
c. The entire images dataset is divided into training set and test set in the ratio 8:2.

3. Model training

a. Exploring useful models.
b. Building a model, training it, and assessing its performance.
c. Select the final model and move ahead.

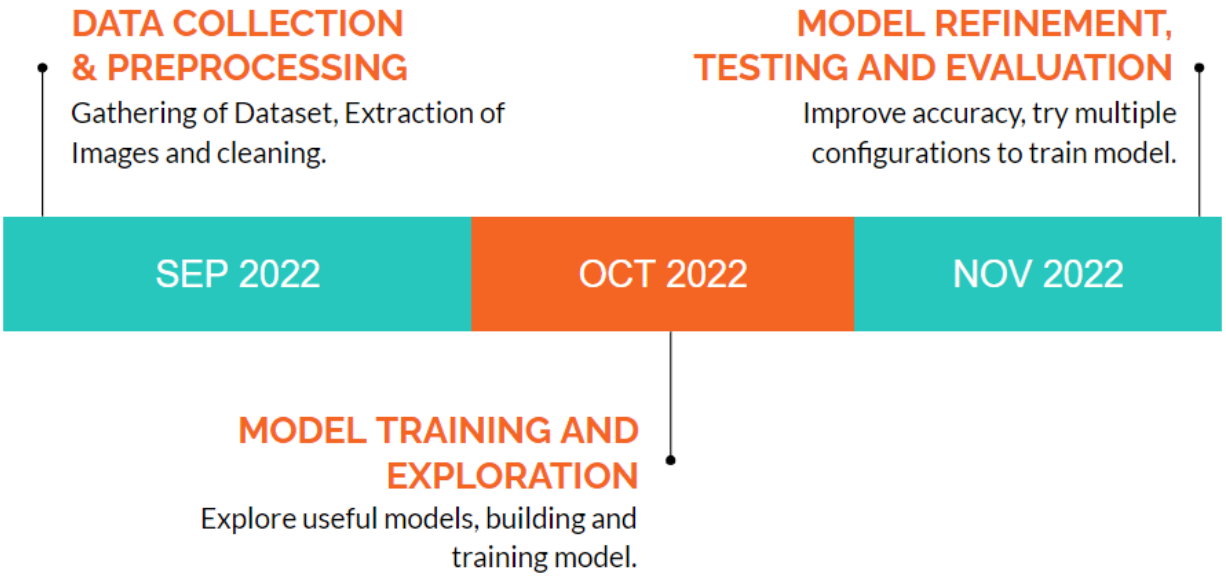
4. Model Refinement

a. Increase the accuracy of the model, and reduce overfitting.
b. Model parameter tuning.

5. Testing and evaluation

a. Testing to make sure that the algorithm performs well on unseen data.
b. Compute model accuracy.

Work Plan



MODEL TRAINING & EXPLORATION

a. Data Exploration:

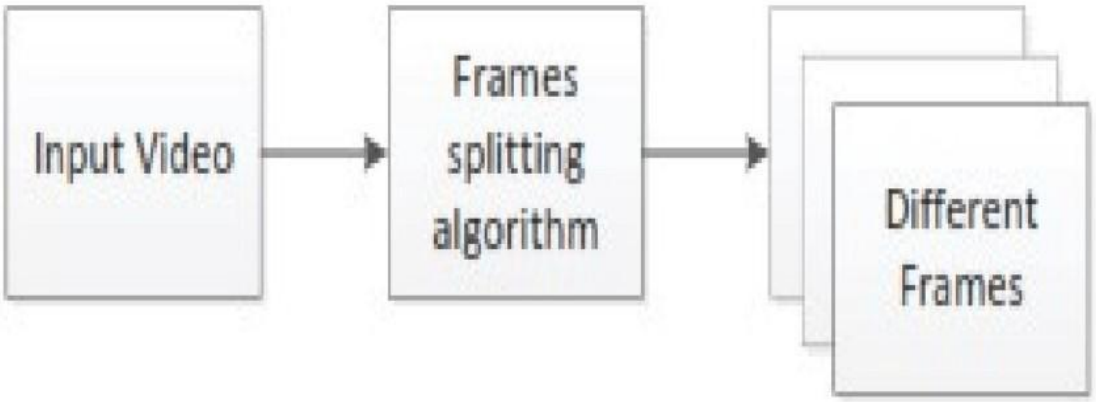
Data exploration is used to explore and visualize data to uncover insights from the start or identify areas or patterns to dig into more. Here, we used the code below to display the videos in the google colab itself making it easier to explore the dataset videos.

```
from IPython.display import
HTML from base64 import
b64encode
folderlocation =

'/content/gdrive/MyDrive/Fight_Dataset/' # Function to

Display Video

def DispVid(vidsrc):
    html = ''
    video = open(vidsrc,'rb').read()
    src = 'data:video/mp4;base64,' + b64encode(video).decode()
    html += '<video width=640 muted controls autoplay
    loop><source
src="%s" type="video/mp4"></video>' %
    src return HTML(html)
```

Output:

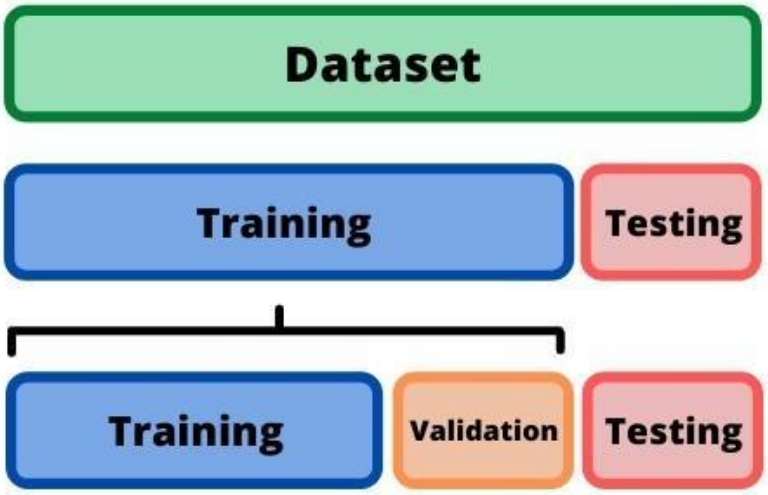
```
DispVid(folderlocation+'NoFight/NV_202.mp4')
```

b.Frame Extraction & Data Formation:



- 15 frames are extracted from videos in both categories i.e. NoFight And Fight. Any video with less than 15 frames is discarded.
- Each Frame is resized to 65x65.
- These pixels are stored in an Numpy array Pixels and their corresponding category in Numpy array category.

c. Splitting Dataset:



- The Dataset is splitted into training and testing with ratio 8:2.
- Training Dataset:
    - Pixels Dimensions: (1606 videos, 15 frames, 65 height, 65 width, 3 color channels)
    - Category Dimensions: (1606 videos, 2 categories)

- Testing Dataset:
    - Pixels Dimensions: (402 videos, 15 frames, 65 height, 65 width, 3 color channels)
    - Category Dimensions: (402 videos, 2 categories)

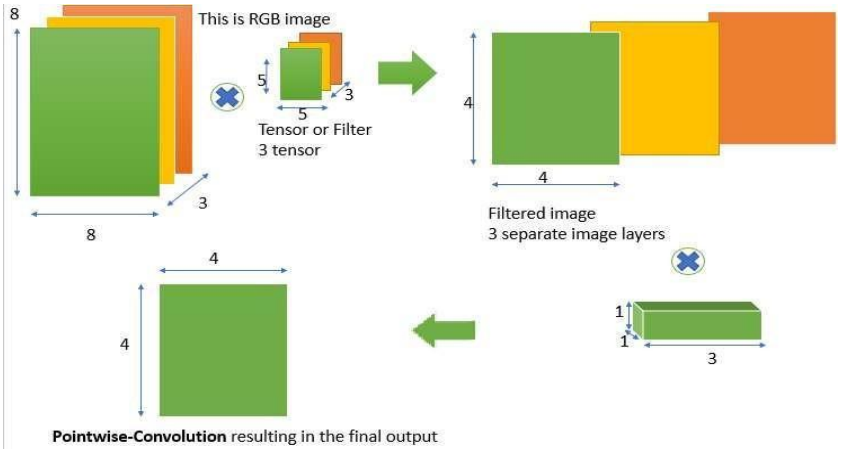
Output:

```
[ ]  print("Training Dataset :")
     print("Pixels Dimensions: ",pixels_train.shape, "Category Dimensions: ", category_train.shape)
     print()
     print("Testing Dataset :")
     print("Pixels Dimensions: ",pixels_test.shape,"Category Dimensions: ", category_test.shape)
```

```
Training Dataset :
Pixels Dimensions:  (1606, 15, 65, 65, 3) Category Dimensions:  (1606, 2)

Testing Dataset :
Pixels Dimensions:  (402, 15, 65, 65, 3) Category Dimensions:  (402, 2)
```

d.Model Creation:

- MobileNet is used to classify categories, and LSTM is used to enhance the performance of the model by maintaining state information of features encountered in previous generation image classification.
- Mobilenet is a model which does the same convolution as done by CNN to filter images but in a different way than those done by the previous CNN. It uses the idea of Depth convolution and point convolution which is different from the normal convolution as done by normal CNNs. This increases the efficiency of CNN to predict images.
- Since these ways of convolution reduce the comparison and recognition time a lot, so it provides a better response in a very short time and hence we are using them as our image recognition model.



- A long short-term memory network is a type of recurrent neural network (RNN). LSTMs are predominately used to learn, process, and classify sequential data because these networks can learn long- term dependencies between time steps of data.

e.Model Fitting:

Defining Callbacks:

1. **Early Stopping Callback** - Minimize the loss by stopping training when metric has stopped improving.

2. **ReduceLROnPlateau Callback** - To reduce overfitting by reducing learning rate when a metric has stopped improving.

The Training Dataset is splitted into training and validation in ratio 8:2.

The Model is run for 25 epochs.

While fitting, callbacks are provided as parameters to the Mobnet_LSTM_model.

```
# Model Compiling
Mobnet_LSTM_model.compile(loss = 'categorical_crossentropy', optimizer = 'sgd', metrics = ["accuracy"])

# Model Fitting
Mobnet_LSTM_model_history = Mobnet_LSTM_model.fit (x = pixels_train, y = category_train, epochs = 25,
                                                   batch_size = 8 , shuffle = True, validation_split = 0.2,
                                                   callbacks = [early_stopping,Reduce_LROnPlateau] )
Mobnet_LSTM_model_history = Mobnet_LSTM_model_history.history
```

Output:

```
Epoch 1/25
161/161 [==============================] - 239s 1s/step - loss: 0.6943 - accuracy: 0.5016 - val_loss: 0.6864 - val_accuracy: 0.5652 - lr: 0.0100
Epoch 2/25
161/161 [==============================] - 224s 1s/step - loss: 0.6874 - accuracy: 0.5514 - val_loss: 0.6753 - val_accuracy: 0.6646 - lr: 0.0100
Epoch 3/25
161/161 [==============================] - 216s 1s/step - loss: 0.6740 - accuracy: 0.5802 - val_loss: 0.6415 - val_accuracy: 0.7236 - lr: 0.0100
Epoch 4/25
161/161 [==============================] - 217s 1s/step - loss: 0.6157 - accuracy: 0.6822 - val_loss: 0.5135 - val_accuracy: 0.7578 - lr: 0.0100
Epoch 5/25
161/161 [==============================] - 226s 1s/step - loss: 0.4944 - accuracy: 0.7960 - val_loss: 0.5030 - val_accuracy: 0.7609 - lr: 0.0100
Epoch 6/25
161/161 [==============================] - 223s 1s/step - loss: 0.4405 - accuracy: 0.8178 - val_loss: 0.3506 - val_accuracy: 0.8758 - lr: 0.0100
Epoch 7/25
161/161 [==============================] - 221s 1s/step - loss: 0.3622 - accuracy: 0.8505 - val_loss: 0.2799 - val_accuracy: 0.8975 - lr: 0.0100
Epoch 8/25
161/161 [==============================] - 222s 1s/step - loss: 0.3313 - accuracy: 0.8731 - val_loss: 0.3657 - val_accuracy: 0.8882 - lr: 0.0100
Epoch 9/25
161/161 [==============================] - 222s 1s/step - loss: 0.3098 - accuracy: 0.8886 - val_loss: 0.2266 - val_accuracy: 0.9193 - lr: 0.0100
Epoch 10/25
161/161 [==============================] - 221s 1s/step - loss: 0.2213 - accuracy: 0.9174 - val_loss: 0.2396 - val_accuracy: 0.9037 - lr: 0.0100
Epoch 11/25
161/161 [==============================] - 221s 1s/step - loss: 0.1976 - accuracy: 0.9385 - val_loss: 0.2971 - val_accuracy: 0.9006 - lr: 0.0100
Epoch 12/25
161/161 [==============================] - 220s 1s/step - loss: 0.1709 - accuracy: 0.9361 - val_loss: 0.4414 - val_accuracy: 0.8665 - lr: 0.0100
Epoch 13/25
161/161 [==============================] - 221s 1s/step - loss: 0.1764 - accuracy: 0.9330 - val_loss: 0.5245 - val_accuracy: 0.8261 - lr: 0.0100
Epoch 14/25
161/161 [==============================] - 223s 1s/step - loss: 0.1292 - accuracy: 0.9572 - val_loss: 0.2050 - val_accuracy: 0.9410 - lr: 0.0100
Epoch 15/25
161/161 [==============================] - 222s 1s/step - loss: 0.1499 - accuracy: 0.9478 - val_loss: 0.2808 - val_accuracy: 0.9130 - lr: 0.0100
Epoch 16/25
161/161 [==============================] - 221s 1s/step - loss: 0.1372 - accuracy: 0.9470 - val_loss: 0.2014 - val_accuracy: 0.9410 - lr: 0.0100
Epoch 17/25
161/161 [==============================] - 220s 1s/step - loss: 0.1088 - accuracy: 0.9650 - val_loss: 0.3021 - val_accuracy: 0.9224 - lr: 0.0100
Epoch 18/25
161/161 [==============================] - 219s 1s/step - loss: 0.0905 - accuracy: 0.9673 - val_loss: 0.4646 - val_accuracy: 0.8665 - lr: 0.0100
Epoch 19/25
161/161 [==============================] - 220s 1s/step - loss: 0.1374 - accuracy: 0.9540 - val_loss: 0.3611 - val_accuracy: 0.8913 - lr: 0.0100
Epoch 20/25
161/161 [==============================] - 220s 1s/step - loss: 0.1112 - accuracy: 0.9618 - val_loss: 0.4470 - val_accuracy: 0.8820 - lr: 0.0100
Epoch 21/25
161/161 [==============================] - ETA: 0s - loss: 0.0624 - accuracy: 0.9798
Epoch 21: ReduceLROnPlateau reducing learning rate to 0.005999999865889549.
161/161 [==============================] - 220s 1s/step - loss: 0.0624 - accuracy: 0.9798 - val_loss: 0.3164 - val_accuracy: 0.9099 - lr: 0.0100
Epoch 22/25
161/161 [==============================] - 220s 1s/step - loss: 0.0745 - accuracy: 0.9743 - val_loss: 0.4462 - val_accuracy: 0.8851 - lr: 0.0060
Epoch 23/25
161/161 [==============================] - 221s 1s/step - loss: 0.0793 - accuracy: 0.9805 - val_loss: 0.3839 - val_accuracy: 0.8944 - lr: 0.0060
```

CHANGING PARAMETERS

1. **Epochs** - An epoch in machine learning means one complete pass of the training dataset through the algorithm. This epoch's number is an important hyperparameter for the algorithm. It specifies the number of epochs or complete passes of the entire training dataset passing through the training or learning process of the algorithm. With each epoch, the dataset's internal model parameters are updated. For a model with good accuracy, the number of epochs must be optimum.
2. **Early Stopping Callback** - Minimize the loss by stopping training when metric has stopped improving. The callback of Early Stopping is used to stop the learning process if there is no accuracy improvement.
3. **ReduceLROnPlateau Callback** - To reduce overfitting by reduce learning rate when a metric has stopped improving.

Final Production: Designing API for practical use

1. Installing fastapi:

```
[ ] !pip install fastapi

Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting fastapi
  Downloading fastapi-0.88.0-py3-none-any.whl (55 kB)
     |████████████████████████████████| 55 kB 1.2 MB/s
Requirement already satisfied: pydantic!=1.7,!=1.7.1,!=1.7.2,!=1.7.3,!=1.8,!=1.8.1,<2.0.0,>=1.6.2 in /usr/local/lib/python3.8/dist-packages (from fastapi) (1.10.2)
Collecting starlette==0.22.0
  Downloading starlette-0.22.0-py3-none-any.whl (64 kB)
     |████████████████████████████████| 64 kB 1.5 MB/s
Requirement already satisfied: typing-extensions>=3.10.0 in /usr/local/lib/python3.8/dist-packages (from starlette==0.22.0->fastapi) (4.1.1)
Requirement already satisfied: anyio<5,>=3.4.0 in /usr/local/lib/python3.8/dist-packages (from starlette==0.22.0->fastapi) (3.6.2)
Requirement already satisfied: idna>=2.8 in /usr/local/lib/python3.8/dist-packages (from anyio<5,>=3.4.0->starlette==0.22.0->fastapi) (2.10)
Requirement already satisfied: sniffio>=1.1 in /usr/local/lib/python3.8/dist-packages (from anyio<5,>=3.4.0->starlette==0.22.0->fastapi) (1.3.0)
Installing collected packages: starlette, fastapi
Successfully installed fastapi-0.88.0 starlette-0.22.0
```

2. Creating Data Model:

```
[ ]  from pydantic import BaseModel

     class URL(BaseModel):
         src: str
         frames : int
         class Config:
             schema_extra = {
                 "example": {
                     "src": folderlocation + "NoFight/NV_32.mp4",
                     "frames" : 15
                 }
             }
```

3. Sample Input JSON format:

SAMPLE JSON:

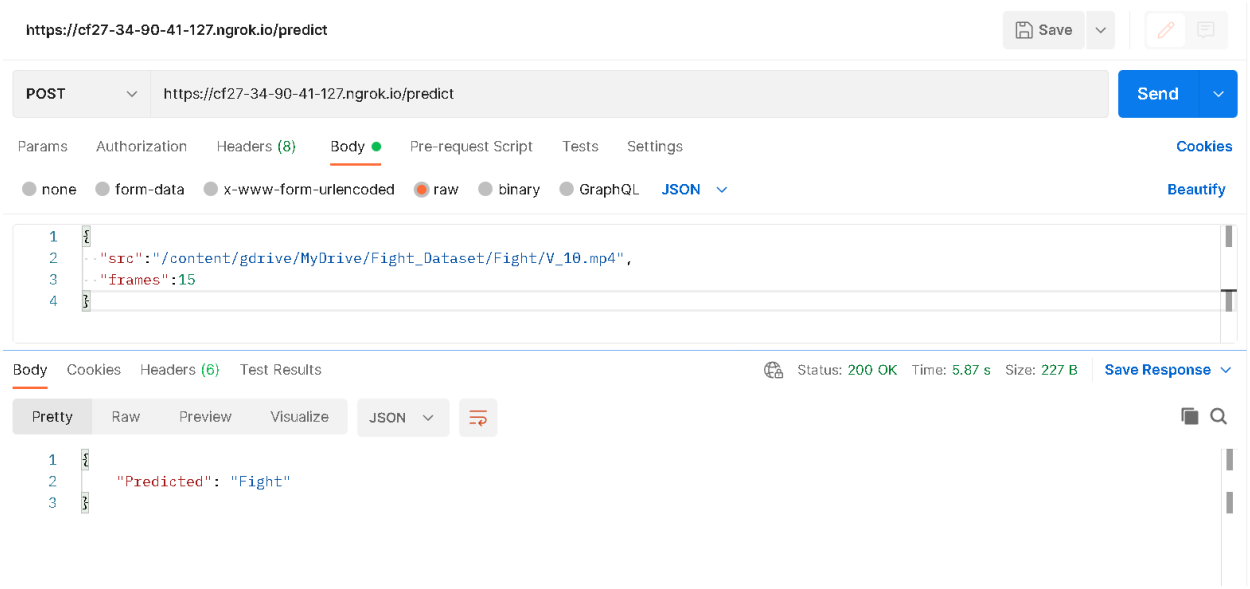
```
{
  "src":"/content/gdrive/MyDrive/Fight_Dataset/Fight/V_10.mp4",
  "frames":15
}
```

```
[ ]  from colabcode import ColabCode
     server = ColabCode(port=10000, code=False)
```

```
     server.run_app(app=app)
```

```
Public URL: NgrokTunnel: "https://cf27-34-90-41-127.ngrok.io" -> "http://localhost:10000"
INFO:     Started server process [75]
INFO:uvicorn.error:Started server process [75]
INFO:     Waiting for application startup.
INFO:uvicorn.error:Waiting for application startup.
WARNING:pyngrok.process.ngrok:t=2022-12-07T01:44:47+0000 lvl=warn msg="failed to open private leg"
INFO:     Application startup complete.
INFO:uvicorn.error:Application startup complete.
INFO:     Uvicorn running on http://127.0.0.1:10000 (Press CTRL+C to quit)
INFO:uvicorn.error:Uvicorn running on http://127.0.0.1:10000 (Press CTRL+C to quit)
1/1 [==============================] - 2s 2s/step
INFO:     117.206.212.175:0 - "POST /predict HTTP/1.1" 200 OK
INFO:     Shutting down
INFO:uvicorn.error:Shutting down
INFO:     Waiting for application shutdown.
INFO:uvicorn.error:Waiting for application shutdown.
INFO:     Application shutdown complete.
INFO:uvicorn.error:Application shutdown complete.
INFO:     Finished server process [75]
INFO:uvicorn.error:Finished server process [75]
```

4. Output using Postman:



- Testing

MODEL EVALUATION

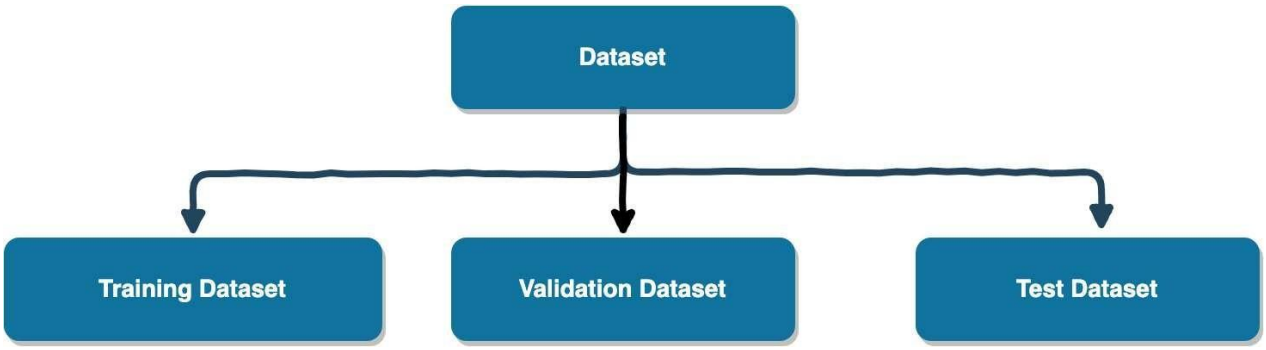
Validation loss is a metric used to assess the performance of a deep learning model on the validation set. The validation set is a portion of the dataset set aside to validate the performance of the model. The validation loss is similar to the training loss and is calculated from a sum of the errors for each example in the validation set.

```
draw_graph(Mobnet_LSTM_model_history, 'loss', 'val_loss',
'Total Loss vs Total Validation Loss')
```

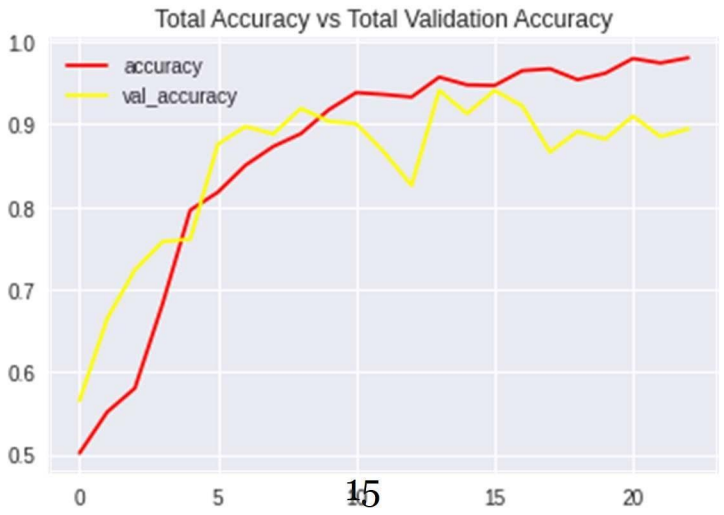
When we are training the model in keras, accuracy and loss in keras model for validation data could be variating with different cases. Usually with every epoch increasing, loss should be going lower and accuracy should be going higher.

But with **val_loss**(keras validation loss) and **val_acc**(keras validation accuracy), many cases can be possible like below:

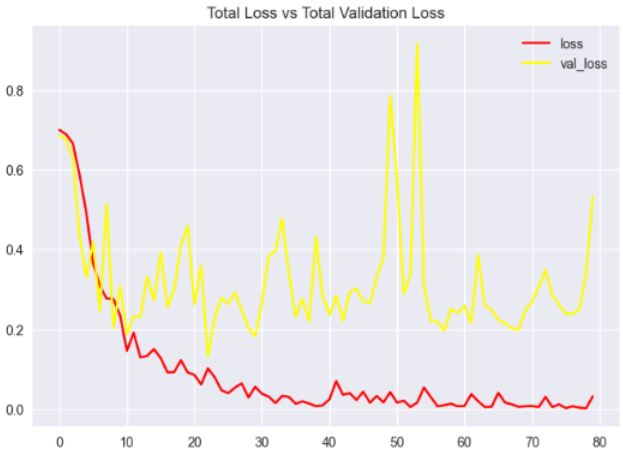
1. val_loss starts increasing, val_acc starts decreasing. This means model is cramming values not learning
2. val_loss starts increasing, val_acc also increases.This could be case of overfitting or diverse probability values in cases where softmax is being used in output layer
3. val_loss starts decreasing, val_acc starts increasing. This is also fine as that means model built is learning and working fine.

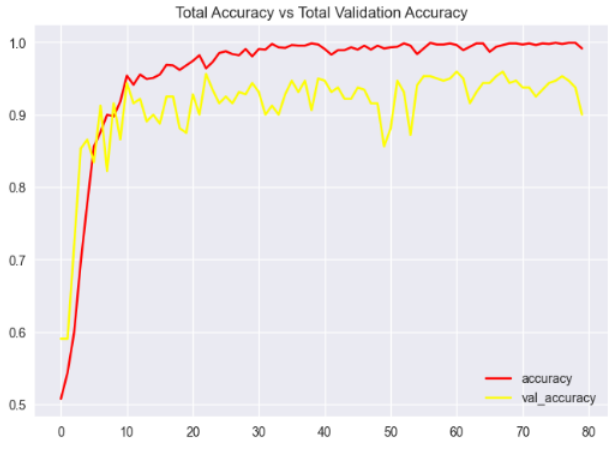


Validation accuracy will be usually less than training accuracy because training data is something with which the model is already familiar with and validation data is a collection of new data points which is new to the model.



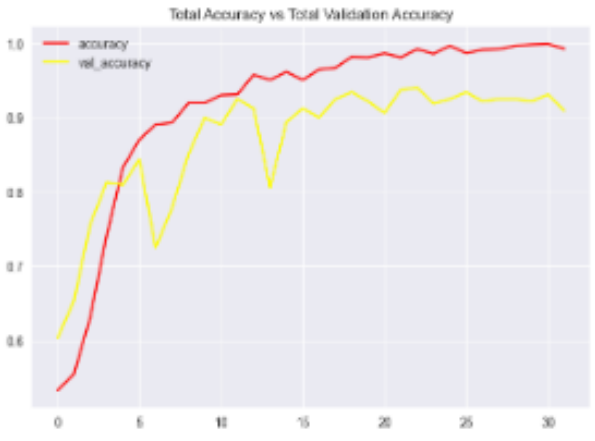
1. No. of epochs = 80
   Callbacks = None





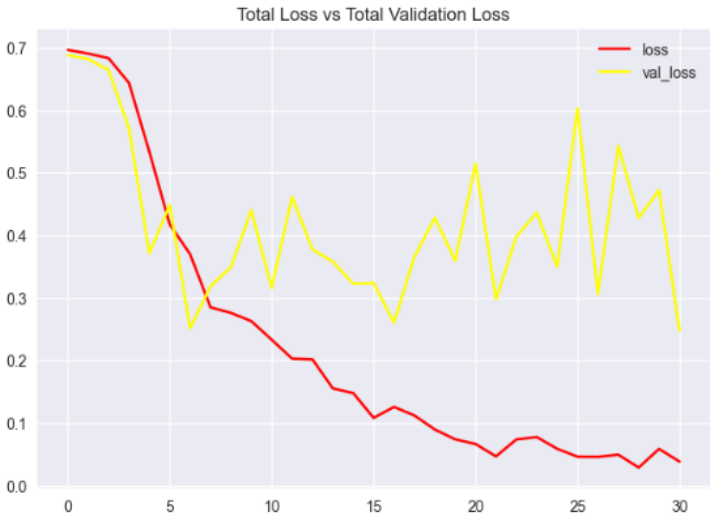

```
Accuracy on Testing Data :  89.5  %
```

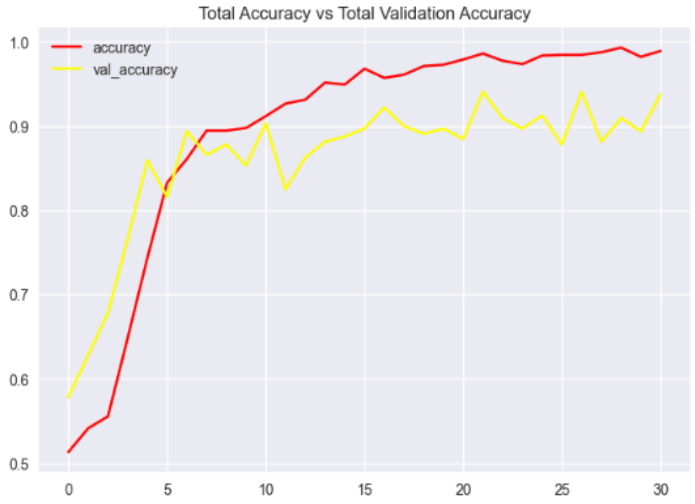
2. No. of epochs = 60
   Callbacks = [early_stopping,Reduce_LROnPlateau]




```
Accuracy on Testing Data :  89.5  %
```

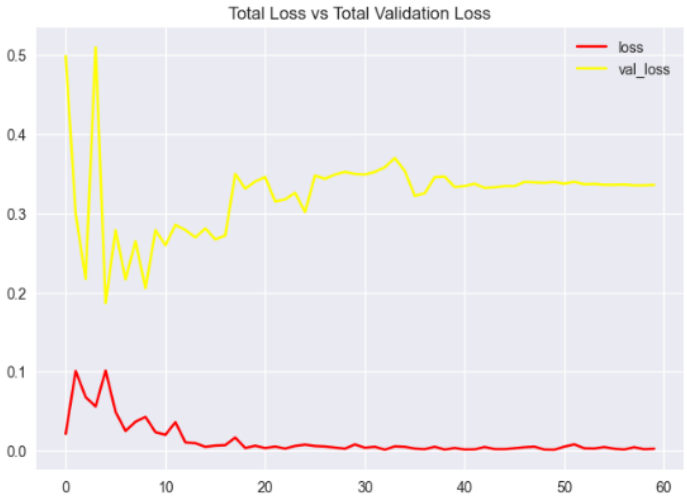
3. No. of epochs = 70
Callbacks = [early_stopping]

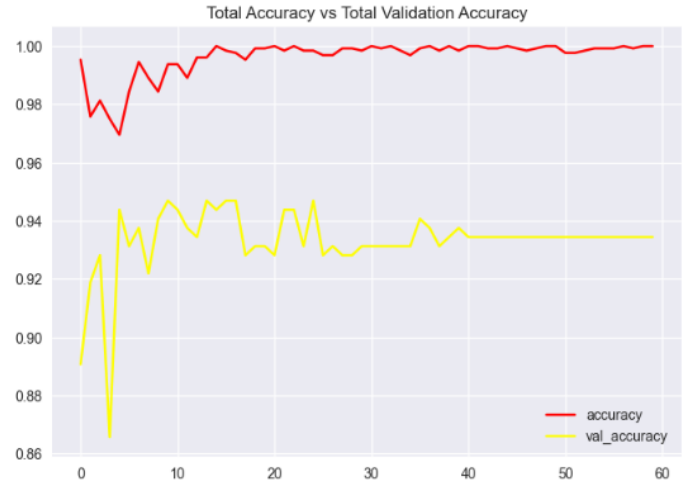





```
Accuracy on Testing Data :  91.0  %
```

4. No. of epochs = 80
Callbacks = [Reduce_LROnPlateau]






```
Accuracy on Testing Data :  93.0  %
```

PREDICTING THE TEST SET & ACCURACY SCORE

Model accuracy is a machine learning classification model performance metric that is defined as the ratio of true positives and true negatives to all positive and negative observations.

Mathematically, it represents the ratio of the sum of true positive and true negatives out of all the predictions.

Accuracy Score = (TP + TN)/ (TP + FN + TN + FP)

```
[ ]  predicted_cat = Mobnet_LSTM_model.predict(pixels_test)
     predicted_cat = np.argmax(predicted_cat , axis=1)
     original_cat = np.argmax(category_test , axis=1)
     print("Original Category dimensions: ",original_cat.shape , " Predicted Category dimensions: ", predicted_cat.shape)

     13/13 [==============================] - 18s 1s/step
     Original Category dimensions:  (402,)  Predicted Category dimensions:  (402,)
```

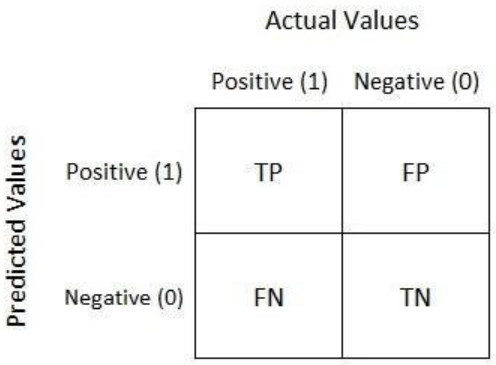
```
[ ]  from sklearn.metrics import accuracy_score
     AccuracyScore = accuracy_score(predicted_cat, original_cat)
     print('Accuracy on Testing Data : ', AccuracyScore*100 , " %")

     Accuracy on Testing Data :  92.28855721393035  %
```

CONFUSION MATRIX

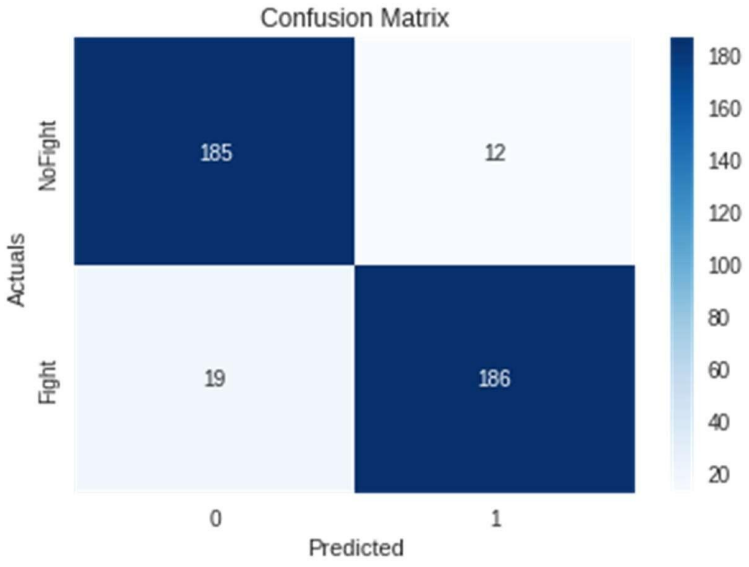
Confusion matrices represent counts from predicted and actual values.

The output “TN” stands for True Negative which shows the number of negative examples classified accurately. Similarly, “TP” stands for True Positive which indicates the number of positive examples classified accurately. The term “FP” shows False Positive value, i.e., the number of actual negative examples classified as positive.

The most frequently used performance metrics for classification according to these values are accuracy (ACC), precision (P), sensitivity (Sn), specificity (Sp), and F-score values.




```
#CONFUSION MATRIX FOR OUR MODEL
import seaborn as sns
from sklearn.metrics import confusion_matrix

ax = plt.subplot()
cm = confusion_matrix(original_cat, predicted_cat)
sns.heatmap(cm, annot=True, fmt='g',
ax=ax,cmap="Blues");
ax.set_xlabel('Predicted');ax.set_ylabel('Actuals'
); ax.set_title('Confusion Matrix');
ax.yaxis.set_ticklabels(['NoFight', 'Fight']);
```



CLASSIFICATION REPORT

A classification report is a performance evaluation metric in machine learning. It is used to show the precision, recall, F1 Score, and support of your trained classification model

$$P = \frac{TP}{TP+FP}$$

$$Sn = \frac{TP}{TP+FN}$$

$$Sp = \frac{TN}{TN+FP}$$

$$F - score = 2 \times \frac{P \times Sn}{P+Sn}$$

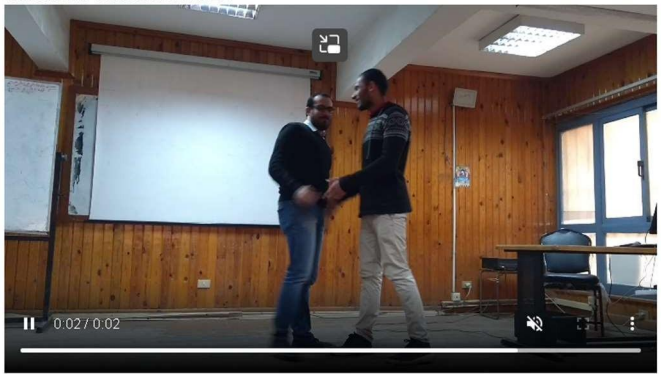
```
#CLASSIFICATION REPORT OF OUR MODEL
from sklearn.metrics import classification_report

Report = classification_report(original_cat, predicted_cat,
target_names=['NoFight','Fight'])
print('Classification Report:
\n',Report) #Precision = TP/TP+FP
#Recall = TP/TP+FN
#F1 Score = 2*(Recall * Precision) / (Recall + Precision)
```

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

     NoFight       0.91      0.94      0.92       197
       Fight       0.94      0.91      0.92       205

    accuracy                           0.92       402
   macro avg       0.92      0.92      0.92       402
weighted avg       0.92      0.92      0.92       402
```

PREDICTION OF TEST VIDEOS

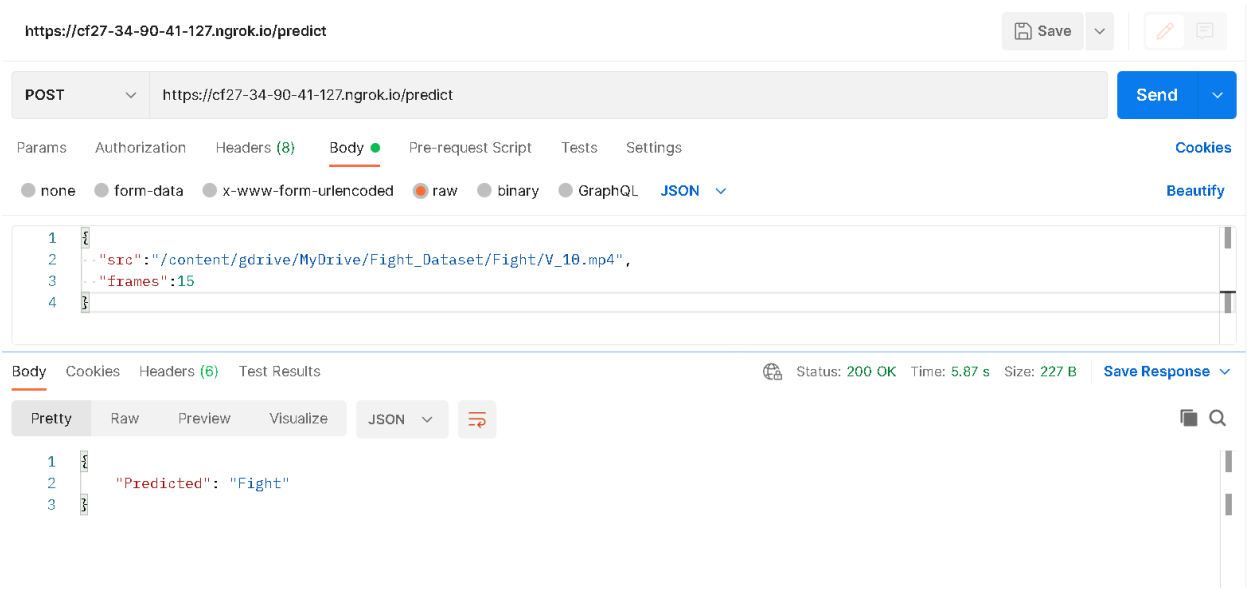
```
# Perform Prediction on the Test Video.
predict_video(input_video_file_path,
SEQUENCE_LENGTH) # Play the actual video
Play_Video(input_video_file_path)
```

```
1/1 [==============================] - 0s 25ms/step
Predicted: NoFight
Confidence: 0.9986905455589294
```




```
1/1 [==============================] - 0s 364ms/step
Predicted: Fight
Confidence: 0.9972391128540039
```

Project Snapshots



FUTURE SCOPE

1. Attaching Hardware Components:

   Attaching hardware components i.e. Cameras and integrating it with the project.

2. Specialization:

   Collecting data for a specific place & extending the model to improve accuracy for a specific place. E.g. University Campus.

BIBLIOGRAPHY

1. https://docs.opencv.org/3.4/d7/dbd/group__imgproc.html
2. https://docs.python.org/3/tutorial/index.html
3. https://keras.io/api/applications/mobilenet/
4. https://keras.io/api/layers/recurrent_layers/lstm/
5. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html
6. https://keras.io/api/losses/
7. https://www.baeldung.com/cs/training-validation-loss-deep-learning
8. https://keras.io/api/callbacks/
9. https://stackoverflow.com/