CSCE636 – PROJECT REPORT – 1

NAME: Darpit Dave

UIN – 126004742

Abstract

In Submission 1 of my Project Report, I am working on detecting 'dizziness' using a neural network work. This can be an important addition in a smart home. Detecting dizziness among people with medical conditions can be used to send out alerts to the concerned personnel to take necessary actions. The model used in this submission has linearly stacked LSTM layers.

Data Description and Processing

The dataset consists of 160 self-made videos of roughly 5-6 seconds each. The training-validation split was 70-30 that is 112 for training and 48 videos for testing. The following steps were taken for processing the videos as input to the neural network model:

Step 1: OpenPose script was used to extract body landmarks from the videos for each frame. These were stored in a json format with x and y coordinates.

Step 2: The JSON files were further combined and stored as a .txt file for reading it as a matrix format.

```
def convert_data(start, end, output_json, output_json_key):
  '''
  start- Starting video number to be added to text file
  end - Last video number to be added
  output_json - Path of json files for videos
  output_json_key -  path of text file where the key-point values are to stored
  '''
  with open(output_json_key,"w") as x:
    for i in range(start, end+1):
          json_files = glob.glob(output_json + '/'+ str(i)+"/*")
          j=0
          for jfile in json_files:
            if j< 140:
              j = j + 1
              with open(jfile) as f:
                  data = json.load(f)
                  if data['people']:
                      kp= data['people'][0]['pose_keypoints_2d']
                      kp = [ str(v) for v in kp]
                      r =','.join(kp)
                      x.write('{}\n'.format(r))
```

Step 3: Once the files are read, separate training and validation data are created in a 70-30 split for training.

```
                                    ###DATA CONVERSION (only required once for generating text files)####
#INITIALIZATION:
#TRAINING:
output_json = '/content/drive/My Drive/CSCE636/JSON_Output'
output_json_key_train = "/content/drive/My Drive/CSCE636/Train_Val_Test/X_train.txt"
start = 1                   #training is from video 1 to video 70 --CHANGE
end = 70
Xtrain = convert_data(start, end, output_json, output_json_key_train)

#VALIDATION:
output_json = '/content/drive/My Drive/CSCE636/JSON_Output'
output_json_key_val = "/content/drive/My Drive/CSCE636/Train_Val_Test/X_val.txt"
start = 71                  #validation is from video 70 to video 100 --CHANGE
end = 100
Xtrain = convert_data(start, end, output_json, output_json_key_val)

                                          ##EXTENSION OF DATASET##
#ADDING MORE TRAINING DATA:
output_json = '/content/drive/My Drive/CSCE636/JSON_Output'
output_json_key_train = "/content/drive/My Drive/CSCE636/Train_Val_Test/X_train_new.txt"
start = 119                   #training is from video 119 to video 160 --CHANGE
end = 160
Xtrain_new = convert_data(start, end, output_json, output_json_key_train)

#ADDING MORE VALIDATION DATA:
output_json = '/content/drive/My Drive/CSCE636/JSON_Output'
output_json_key_val = "/content/drive/My Drive/CSCE636/Train_Val_Test/X_Val_new.txt"
start = 101                 #validation is from video 101 to video 118 --CHANGE
end = 118
Xval_new = convert_data(start, end, output_json, output_json_key_val)
```

Step 4: The read files are reshaped into appropriate tensor dimensions to be used as the final input for the model training.

```
#GENERATING TRAINING AND VALIDATION ARRAYS:

#FOR X-file:
train = "/content/drive/My Drive/CSCE636/Train_Val_Test/X_train.txt"    ##
X_train = read_data(train)

train2 = "/content/drive/My Drive/CSCE636/Train_Val_Test/X_train_new.txt"
X_train_new = read_data(train2)

X_train = np.concatenate((X_train, X_train_new), axis=0)
X_train = X_train.reshape(1568,10,50)

val = "/content/drive/My Drive/CSCE636/Train_Val_Test/X_val.txt"    ##CHAN
X_val = read_data(val)
val2 = "/content/drive/My Drive/CSCE636/Train_Val_Test/X_Val_new.txt"    #
X_val_new = read_data(val2)
X_val = np.concatenate((X_val, X_val_new), axis=0)
X_val = X_val.reshape(672,10,50)

#FOR Y-file:
Y_train,Y_val = generate_label()
Y_train = Y_train.reshape(1568,10,1)
Y_val = Y_val.reshape(672,10,1)

print('X_training = ', X_train.shape)
print('X_validation = ',X_val.shape)
print('Y_training = ', Y_train.shape)
print('Y_validation = ',Y_val.shape)
```

```
X_training =  (1568, 10, 50)
X_validation =  (672, 10, 50)
Y_training =  (1568, 10, 1)
Y_validation =  (672, 10, 1)
```

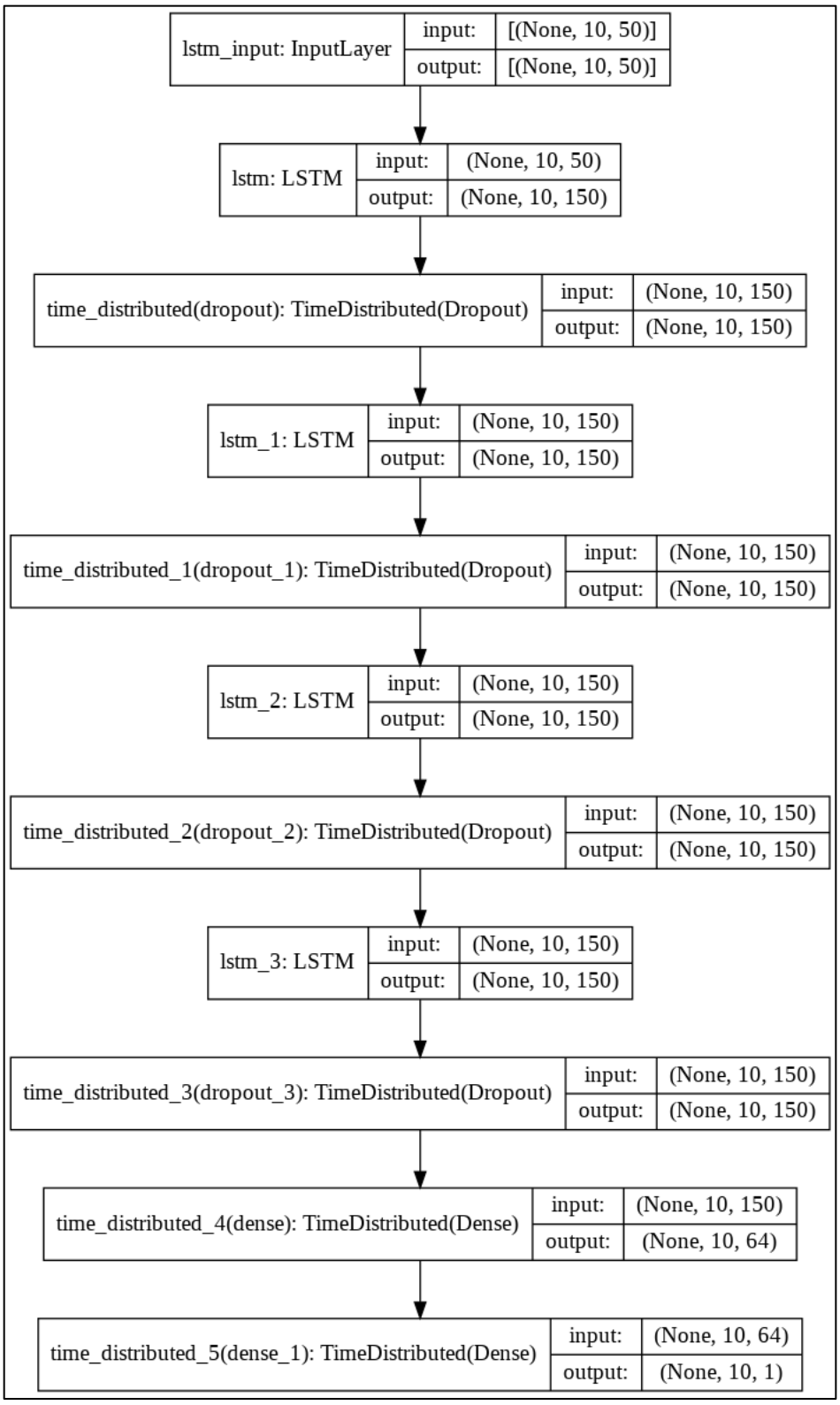
Model Architecture and Design

The model used in this submission is a linear stack of sequential layers. It is a 9-layer unidirectional model. The hidden layer has 4 LSTM layers and 5 time-distributed layers. The output is also a time-distributed layer.

LSTM architecture

```
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.LSTM(150, input_shape=(10,50), return_sequences=True))
model.add(tf.keras.layers.TimeDistributed(tf.keras.layers.Dropout(0.2)))
model.add(tf.keras.layers.LSTM(150, input_shape=(10,50), return_sequences=True))
model.add(tf.keras.layers.TimeDistributed(tf.keras.layers.Dropout(0.2)))
model.add(tf.keras.layers.LSTM(150, input_shape=(10,50), return_sequences=True))
model.add(tf.keras.layers.TimeDistributed(tf.keras.layers.Dropout(0.2)))
model.add(tf.keras.layers.LSTM(150, input_shape=(10,50), return_sequences=True))
model.add(tf.keras.layers.TimeDistributed(tf.keras.layers.Dropout(0.5)))
model.add(tf.keras.layers.TimeDistributed(tf.keras.layers.Dense(64, activation='relu', kernel_regula
model.add(tf.keras.layers.TimeDistributed(tf.keras.layers.Dense(1, activation='sigmoid')))

adam = tf.keras.optimizers.Adam(lr = 0.00001)
model.compile(loss = 'binary_crossentropy', optimizer=adam ,metrics = ['accuracy'])
```


lstm_input: InputLayer
input: [(None, 10, 50)]
output: [(None, 10, 50)]
lstm: LSTM
input: (None, 10, 50)
output: (None, 10, 150)
time_distributed(dropout): TimeDistributed(Dropout)
input: (None, 10, 150)
output: (None, 10, 150)
lstm_1: LSTM
input: (None, 10, 150)
output: (None, 10, 150)
time_distributed_1(dropout_1): TimeDistributed(Dropout)
input: (None, 10, 150)
output: (None, 10, 150)
lstm_2: LSTM
input: (None, 10, 150)
output: (None, 10, 150)
time_distributed_2(dropout_2): TimeDistributed(Dropout)
input: (None, 10, 150)
output: (None, 10, 150)
lstm_3: LSTM
input: (None, 10, 150)
output: (None, 10, 150)
time_distributed_3(dropout_3): TimeDistributed(Dropout)
input: (None, 10, 150)
output: (None, 10, 150)
time_distributed_4(dense): TimeDistributed(Dense)
input: (None, 10, 150)
output: (None, 10, 64)
time_distributed_5(dense_1): TimeDistributed(Dense)
input: (None, 10, 64)
output: (None, 10, 1)

HYPERPARAMETERS

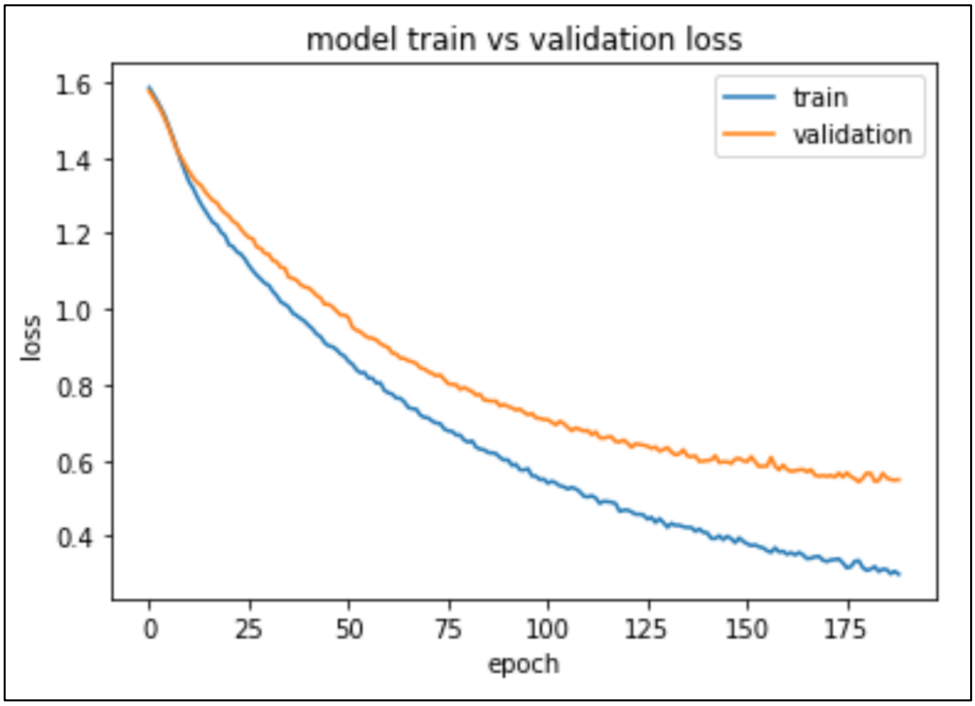
The following hyperparameters were tuned: units, batch_size, epochs, learning rate. The below table summarizes the different values used and the optimal values found.

Hyperparamter	**Values Tried**	**Optimal**
Batch Size	10, 20, 50, 100	20
Units	100, 150, 200, 250, 500	150
Learning rate	0.001, 0.0001, 0.01	0.0001

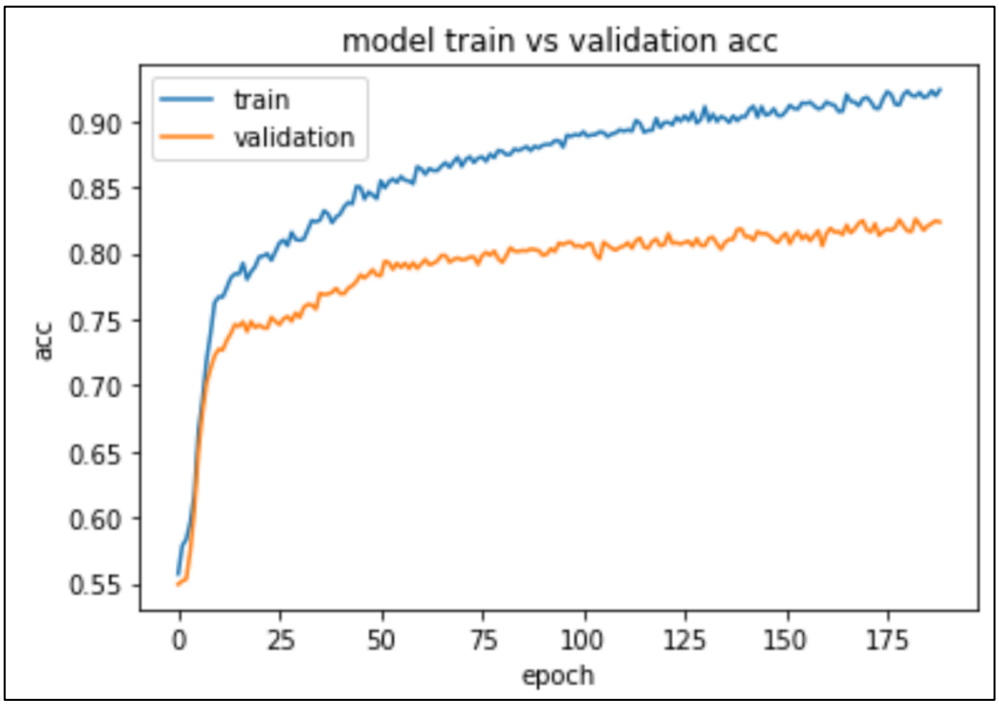
The above 2 hyperparameters were manually tried for various combinations to select the best output. For choosing the optimal epochs, keras features like early stopping and model checkpoint were utilized with 'validation_loss' to be the monitoring metric.

Training and Validation Performance

1. Validation Loss



2. Validation Accuracy

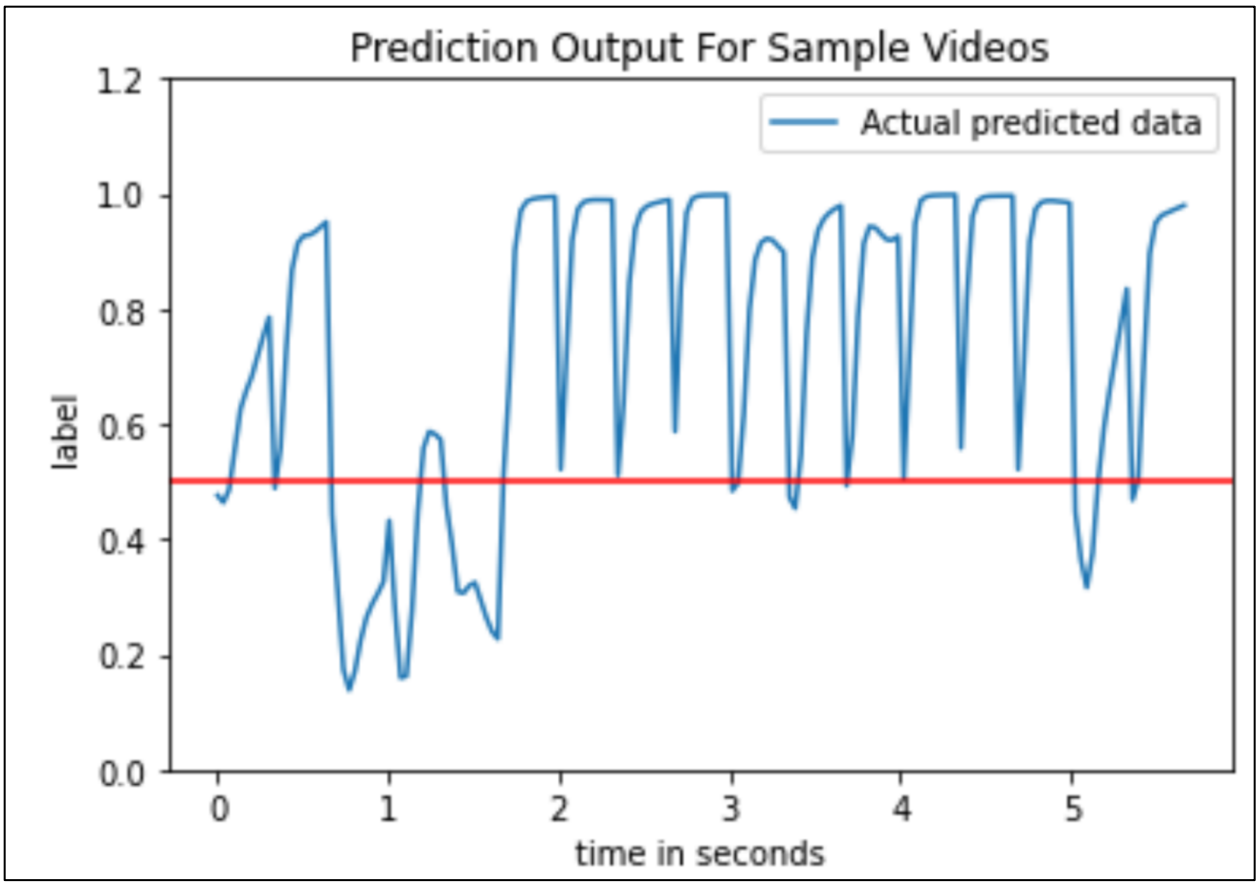


Prediction Results

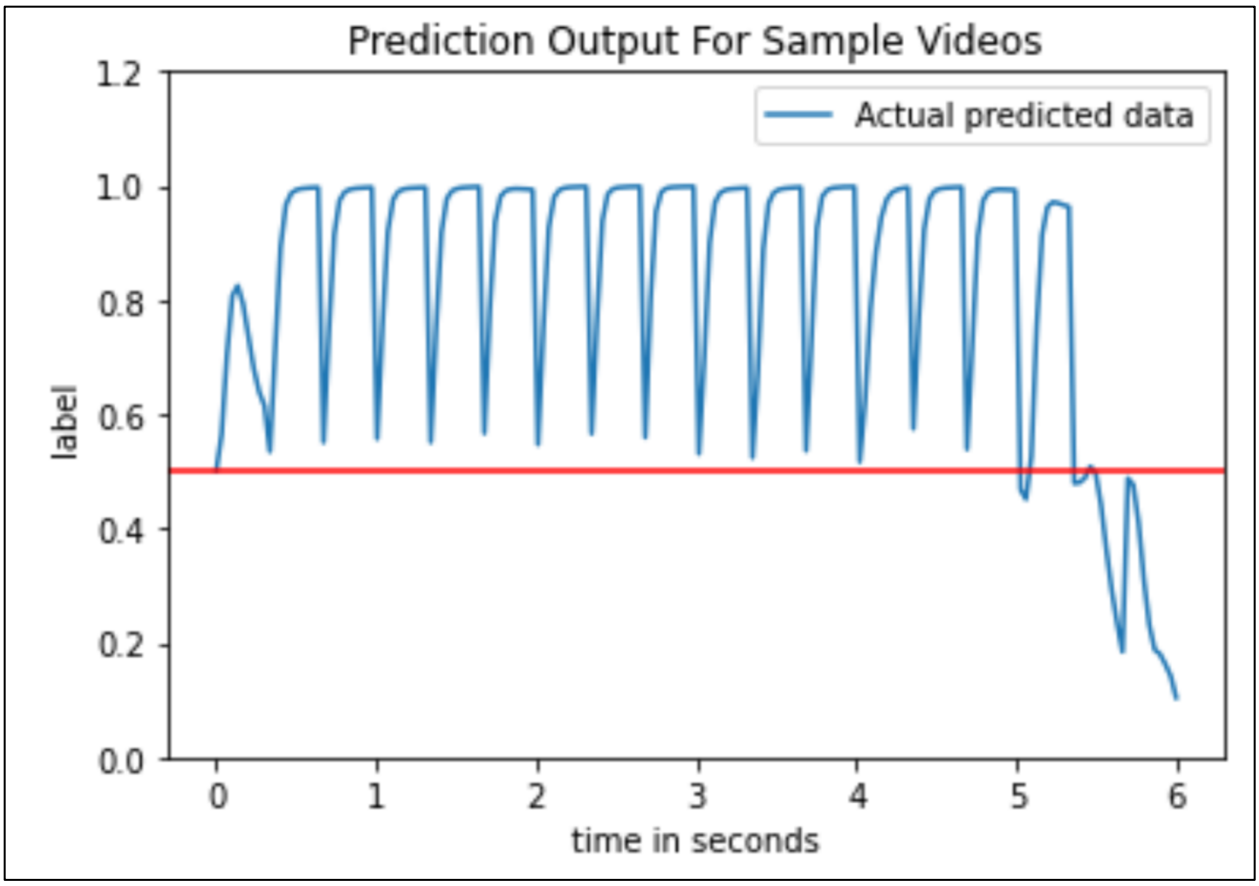
The following are a few sample results for positive (contains target action) and negative classes (does not have target action).

Positive Class

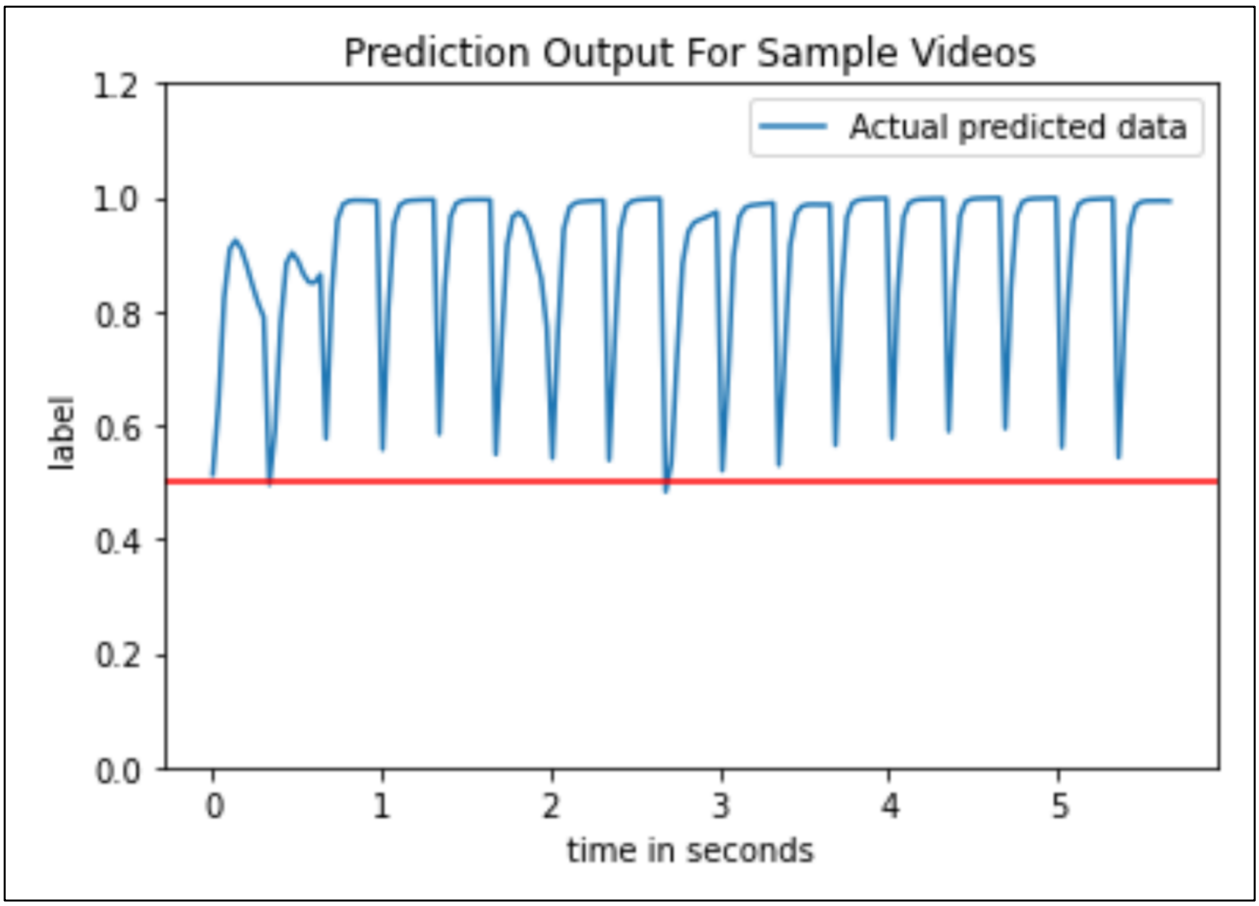
1. **Positive Sample - 1**



2. **Positive Sample - 2**

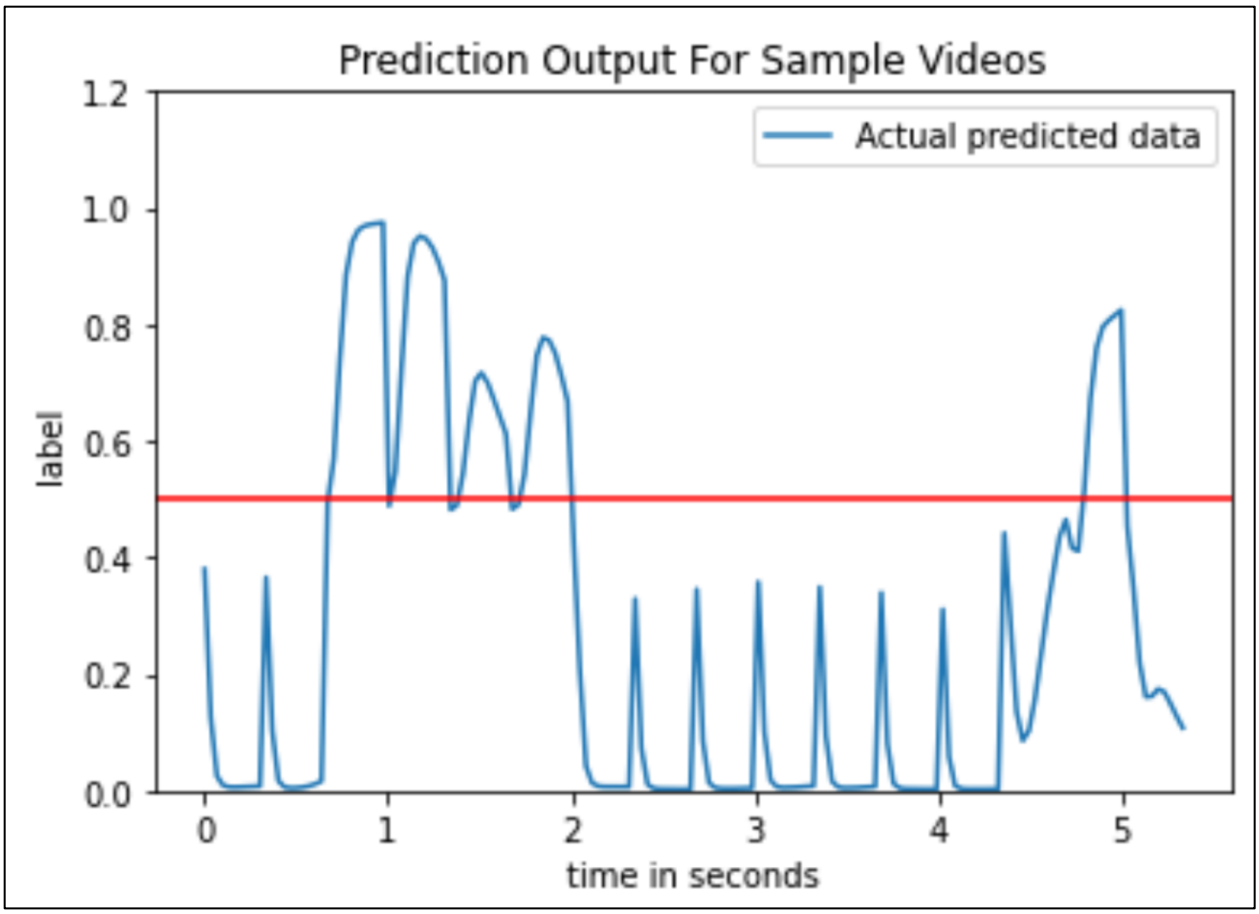


3. **Positive Sample - 3**

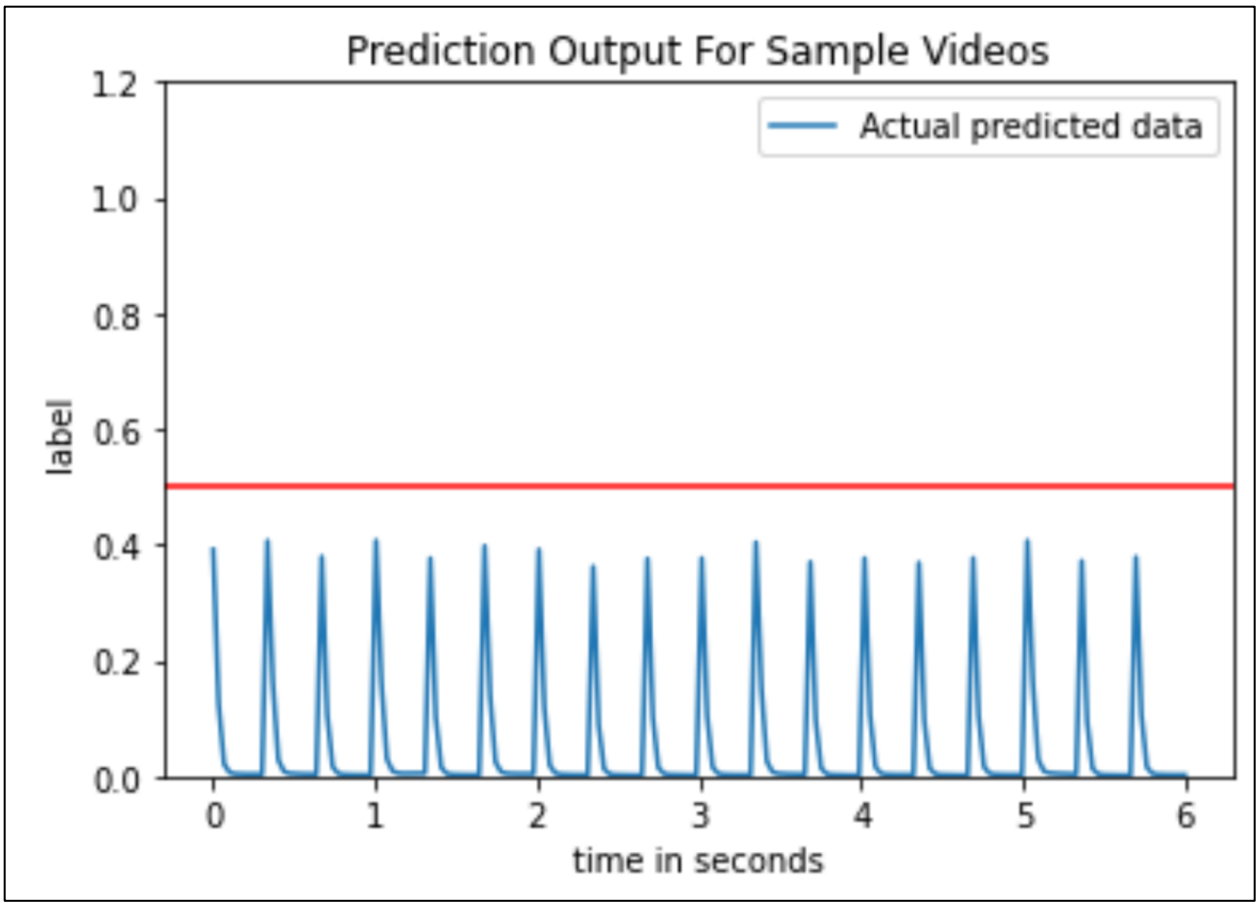


Negative Class

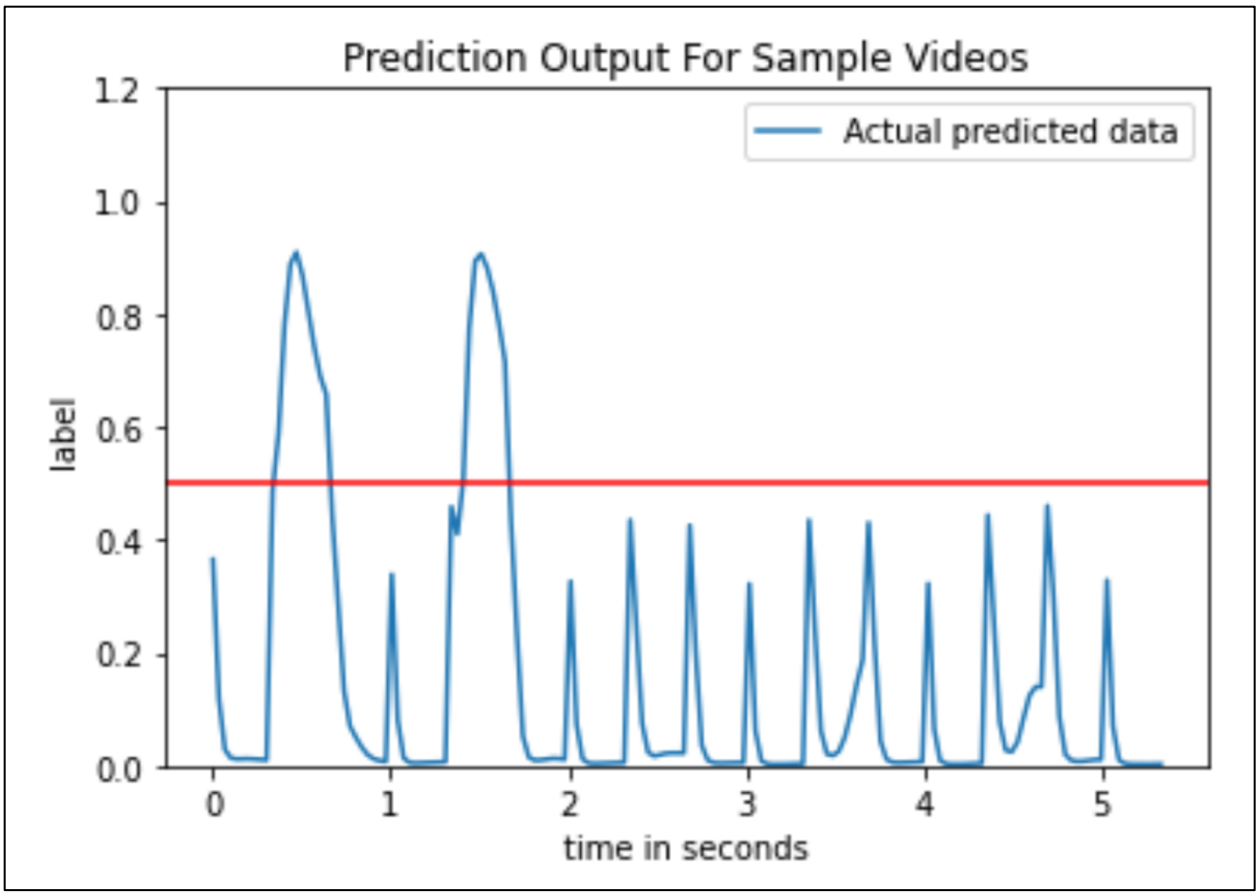
1. **Negative Sample – 1**



2. **Negative Sample – 2**



3. **Negative Sample – 3**



INSTRUCTION ON HOW TO RUN THE PREDICTION MODEL:

1. Download CSCE636_Submission1_Prediction.ipynb from shared github repository (https://github.com/darpitdavetamu/CSCE636-DeepLearning).
2. Upload the file to google colab.
3. JSON files and text file required to run the code can also be accessed at https://drive.google.com/drive/folders/1Ua297X--JqXMMfB33V4TeSG0rxCbzRZP?usp=sharing
4. The path written in the code is same as the google drive link shared.

LINK TO DEMO VIDEO:

https://www.youtube.com/watch?v=GF2JfeCy0RQ&feature=youtu.be