CSCE 636: Deep Learning

Submission 2

Topic

The goal of this project is to train models that can recognize a human action in videos, which can be used for "smart home" type of environments. In this submission, I am working on classifying of videos of people who drink at home.

Training and Test Data

First, I downloaded HMDB51 dataset, and I got 100 videos for drinking action and 80 other videos, especially the actions can be similar to drinking such as eating, chewing, waving and etc. the following code rename all the labels of drinking to “drink_ i” and other videos to “other_i”. Then I put All these videos to one folder for training. I prepared 40 drinking videos for test. Finally, I put Training_Drinking folder for training and Test_Drinking for test the model that I put in my google drive.

```
def rename_file (path , name):
    i=0
    for filename in os.listdir(path):
        os.rename(os.path.join(path,filename), os.path.join(path, name + str(i)+'.avi'))
        i = i +1

all_drinking = "C:\\Users\\shirani\\Desktop\\Action_recognition\\drinking"
all_other = "C:\\Users\\shirani\\Desktop\\Action_recognition\\other"
rename_file(all_drinking, 'drink_')
rename_file(all_other, 'other_')
```

After preparing training and test data folder I put their zip folder on my google drive and continued writing code in google Colab. **Data Preprocessing**

The first step is getting the frames of videos. Reading the video files, extracting frames from each corresponding video and putting them in the respective train/test folder. I used OpenCV library in python for this issue. The following function read 10 frames from each video for the sake of simplicity and save them in another folder to read them for later processing.

```python
#fuction gets the video and save the frames of videos to .jpg format
def get_frames(video_path, dest_path):
    file_name = video_path.split('.')[0].split('/')[-1]
    cap= cv2.VideoCapture(video_path)
    total_frames = int(cap.get(7))
    frame_rate = cap.get(5)
    # print(file_name,total_frames,frame_rate)
    frame_distance = total_frames//10

    count=0
    frame_count=0
    for i in range (total_frames): #iterating over all the frames of the video.
        ret, frame = cap.read()
        if ret ==False:  # Checking if the frame is missing, and if missing, we take the previous frame
            frame = f_prev
        elif ret==True:
            f_prev =frame
        frame_count +=1
        if ((frame_count ==(frame_distance)) and (count<10)):
            frame_count = 0
            cv2.imwrite(dest_path+'/'+file_name+'_'+ str(count)+".jpg",frame)
            count +=1
    cap.release()
    cv2.destroyAllWindows()
```

Then all the frames of videos save as .jpg format. As it is clear in the following the frames are in different size.

The following code resize the images to the size of (224,224) and the normalize them.

```python
[ ]  def get_train_data(file_name):
        training = []
        frames_to_select=[]
        for i , filename in enumerate(glob.glob(file_name)):

            image = cv2.imread(filename)
            img = cv2.resize(image, dsize=(img_high,img_width), interpolation=cv2.INTER_CUBIC)
            # normalizing the pixel value
            img = img/255
            training.append(img)
        return training
```

Regarding to the labels, all the frames with the name of drink got the label of 1 and others got the label of 0.

```
# put label 1 for drinking and 0 for others
train_label =[]
for file in os.listdir(frame_dir):
    label = file.split('_')[0]
    if label == 'drinking':
        train_label.append(1)
    else:
        train_label.append(0)
print(len(train_label))
```

Model Training and Performance

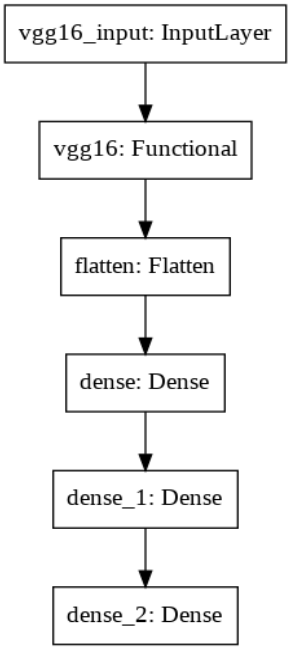
I used CNN using VGG-16 that is one of the most popular pre-trained models for image classification. Since I have small dataset of images, I made up for it by augmenting this data and increasing the dataset size.

```
[ ] datagen = ImageDataGenerator(
        featurewise_center=True,
        featurewise_std_normalization=True,
        rotation_range=20,
        width_shift_range=0.2,
        height_shift_range=0.2,
        horizontal_flip=True)

    datagen.fit(x_train)
```

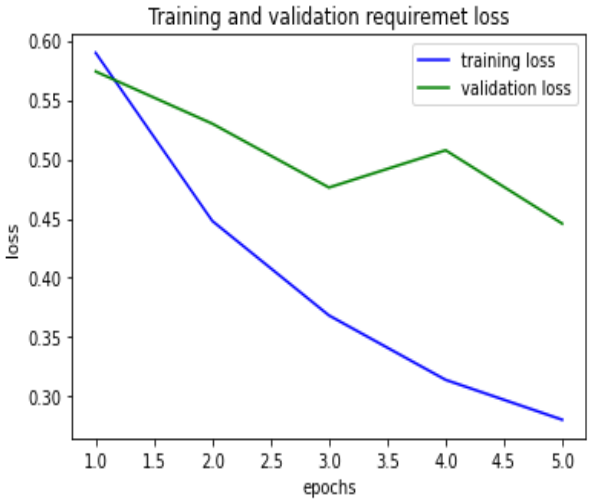
I used all convolution layers from VGG and only replace the last classification layer (Dense layer). The structure of the model is:

```
model = models.Sequential()
model.add(conv_base)
model.add(layers.Flatten())
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))

model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Functional)           (None, 7, 7, 512)         14714688
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
dense (Dense)                (None, 512)               12845568
_________________________________________________________________
dense_1 (Dense)              (None, 256)               131328
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 257
=================================================================
Total params: 27,691,841
Trainable params: 12,977,153
Non-trainable params: 14,714,688
_________________________________________________________________
```



The following plot shows the training and validation loss.



I evaluated the model using test data. The accuracy is 70% on 40 drinking videos that I considered for testing.

```
scores = model.evaluate(X_test, y_test)
print("%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
```

```
13/13 [==============================] - 2s 119ms/step - loss: 0.5914 - acc: 0.7025
acc: 70.25%
```

Performance on Youtube Videos

I tried to find the performance of model on YouTube Videos. So, I downloaded 10 videos from YouTube that includes just drinking action and preprocess the data. First, I split the video to video clips that their length is 2 seconds. Then for each video clips, all the preprocessing had done the same as training part. The model of training loaded and applied to the YouTube videos. Finally, the video clips that have probability of more than 50% are printed as video clips which include drinking action. The following code shows the process of applying the model on these videos.

```
img_high=224
img_width=224
url="https://www.youtube.com/watch?v=Q4GtVj7av-g"
yt= pytube.YouTube(url)
vid_id=yt.video_id
yt= yt.streams.first()
video=yt.download()

# capturing the video from the given path
cap = cv2.VideoCapture(video)
total_frames = int(cap.get(7))


#split youtube video to video clips of 2s
clip_len =2
frames=[]
frame_rate = cap.get(5) # frames per second
vid_len = int(total_frames/frame_rate) # video time  in seconds
intervals = int(vid_len/clip_len) #how many video clips
print(intervals,vid_id)
frame_per_interval = total_frames //intervals #how many frames are in each clips
frame_distance = frame_per_interval//10  # skip some frames
```

```
for i in range (total_frames): #iterating over all the frames of the video.
      ret, frame = cap.read()
      if ret ==False:  # Checking if the frame is missing, and if missing, we take the previous frame
          frame = f_prev
      elif ret==True:
          f_prev =frame

      frame_count_all +=1
      frame_count +=1

      if frame_count ==frame_distance:
          frame_count = 0
          frame=cv2.resize(frame,(img_high,img_width),interpolation=cv2.INTER_AREA)
          frames.append(frame)
          frame_time.append(frame_count_all*(1/frame_rate))

        # when we reach to video clip length save the frames
      if i == ((clip_count*frame_per_interval)-1):
          inpt = np.array(frames)
          inpt = inpt.astype('float32')
          inpt -= np.mean(inpt)
          inpt /=np.max(inpt)
```

```
            frames = []
            p = model.predict(inpt)
            prob = p[:][0][0]

            # find the time of interval
            min_s, sec_s = divmod(frame_time[0], 60)
            start_time=str("%d:%02d" % (min_s, sec_s) ) #Starting time of the interval
            min_e, sec_e = divmod(frame_time[-1],60)
            end_time= str("%d:%02d" % (min_e, sec_e) )  # Ending time of the interval
            frame_time=[]

            if prob>=0.50:
                print(prob,start_time,end_time)

            clip_count += 1 #next clip

            if cv2.waitKey(1) & 0xFF == ord('q'):
                break
    cap.release()
    cv2.destroyAllWindows()
```

Following is the result of one YouTube video (which includes 65 clips of 2s) which have probability of more than 50%. These time are predicted for drinking action in this video. I run this code for each of 10 Youtube videos and entered the video clips that the model could correctly predict in Ilab manually.

```
65 zDh8CJ8mhcI
0.5651156 0:08 0:10
0.53433836 0:10 0:12
0.81335115 0:12 0:14
0.58903134 0:16 0:18
0.54001427 0:18 0:20
0.52162254 0:20 0:22
0.6034077 0:28 0:30
0.5471341 0:38 0:40
0.6014416 0:52 0:54
0.5806018 0:56 0:58
0.5721907 1:00 1:02
0.64827275 1:04 1:06
0.8110906 1:06 1:08
0.5014159 1:12 1:14
0.6744274 1:18 1:20
0.8289679 1:20 1:22
0.6086868 1:24 1:26
0.62675583 1:26 1:28
0.7232847 1:34 1:36
0.561571 1:36 1:38
0.6421597 1:40 1:42
0.506693 1:44 1:46
0.5742778 1:52 1:54
0.5831579 1:58 2:00
0.65616286 2:00 2:02
```

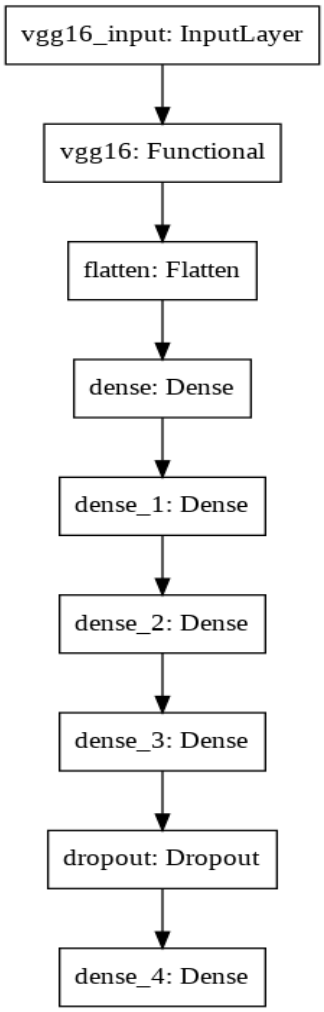
Submission 3 **3/11/2021**

In this submission, I used the same data training and data test for training part, but I applied the model on the new YouTube videos.

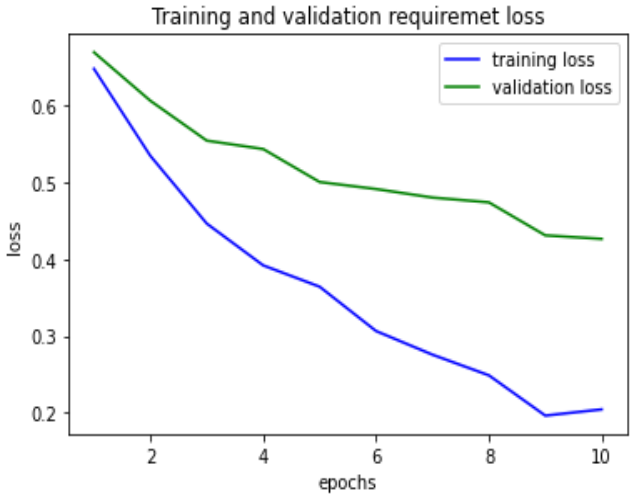
The VGG-16 is a CNN with 16 convolutional layers with a couple of max pooling layers in between and some dense fully-connected layers at the end. In this submission I added more layers to the final layers.

```
[ ]  model = models.Sequential()
     model.add(conv_base)
     model.add(layers.Flatten())
     model.add(layers.Dense(512, activation='relu'))
     model.add(layers.Dense(256, activation='relu'))
     model.add(layers.Dense(128, activation='relu'))
     model.add(layers.Dense(64, activation='relu'))
     model.add(Dropout(0.5))
     model.add(layers.Dense(1, activation='sigmoid'))

     model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Functional)           (None, 7, 7, 512)         14714688
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
dense (Dense)                (None, 512)               12845568
_________________________________________________________________
dense_1 (Dense)              (None, 256)               131328
_________________________________________________________________
dense_2 (Dense)              (None, 128)               32896
_________________________________________________________________
dense_3 (Dense)              (None, 64)                8256
_________________________________________________________________
dropout (Dropout)            (None, 64)                0
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 65
=================================================================
Total params: 27,732,801
```



I changed the epoch to 10 in this submission and the following plot shows the training and validation loss.



I evaluated the model using test data. The accuracy on the same test videos has increase 5 percent in this submission.

```
scores = model.evaluate(X_test, y_test)
print("%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
```

```
13/13 [==============================] - 2s 117ms/step - loss: 0.7586 - acc: 0.7500
acc: 75.00%
```

Submission 5 **3/25/2021**

In this submission I am working on smoking action. I downloaded 176 videos from HMDB51 dataset for training part which include 88 videos of smoking and 88 videos of other actions. The preprocessing of data is the same as before.

In this submission I change the method and use CNN+LSTM. Convolutional Neural Networks are a powerful tool for extracting features from images and are widely used in image processing. Videos are just a set of multiple images recording at different time steps. Thus, a video has both spatial and temporal dimensions. Thus, use of convolutional layers only might not be suffice in dealing with video processing. So adding Time distributed layer helps to have better performance in action recognition in video. I considered 10 frames for each video for simplicity, so time distributed layer would consider 10 frames as 10 time steps and apply the convolutional layer onto those 10 frames keeping the temporal aspect attached.

The challenge of using time distributed layer is the input shape of the dataset. So The training data and labels are reshaped for this method.

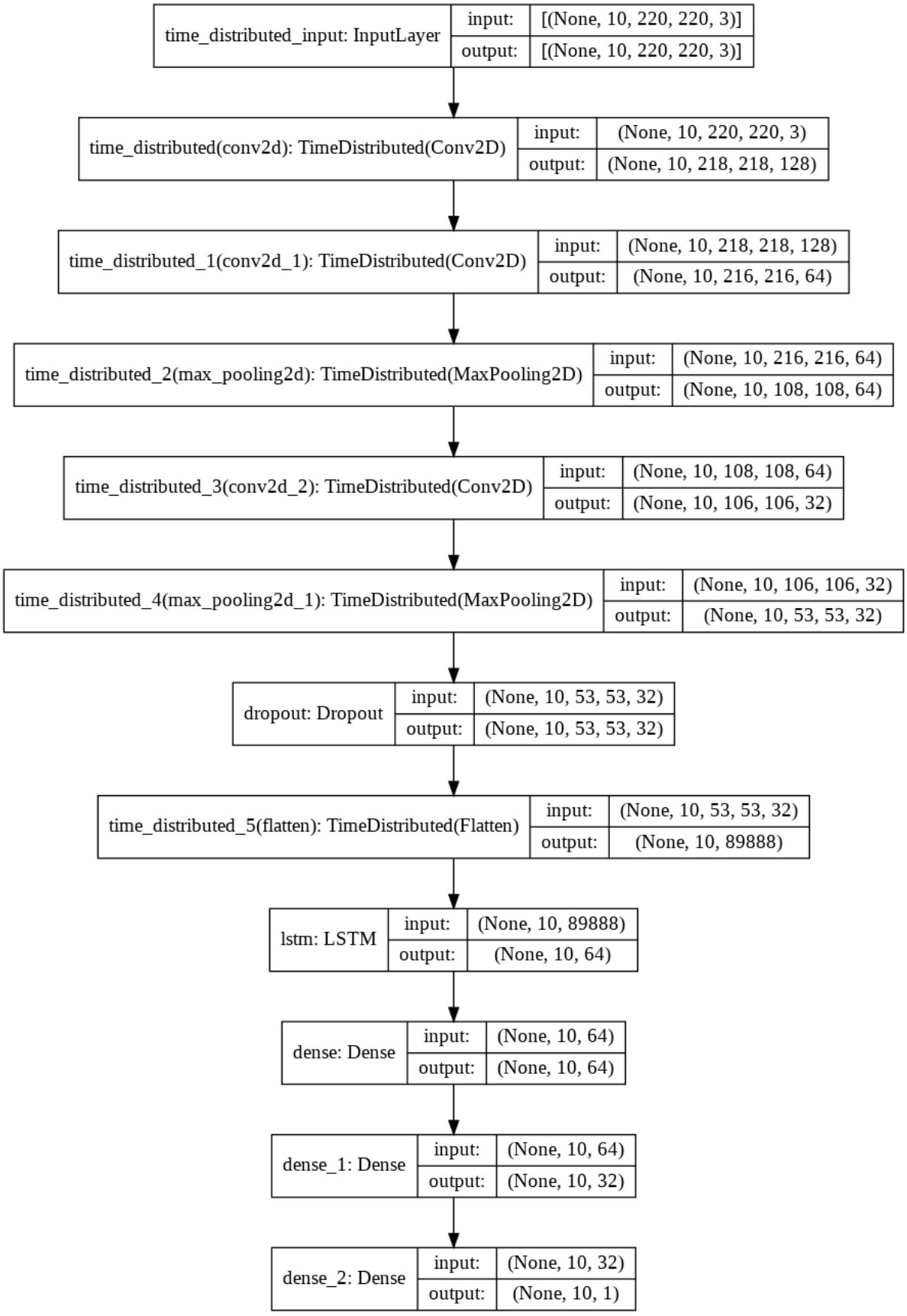
```
vid_num=len(train_label)//frame_no_video
X = np.asarray(training)
y = np.asarray(train_label)
X=X.reshape(vid_num,frame_no_video, img_high, img_width, 3)
print(X.shape)
y=y.reshape(vid_num,frame_no_video,1)
print(y.shape)
```

```
(176, 10, 220, 220, 3)
(176, 10, 1)
```

Then I built a very basic model. I contained 3 Time Distributed Convolutional Layers, 2 Max Pooling Layers followed by an LSTM Layer for Sequence Learning and 3 Fully Connected Dense Layers with 3rd layer as the Output Layer.

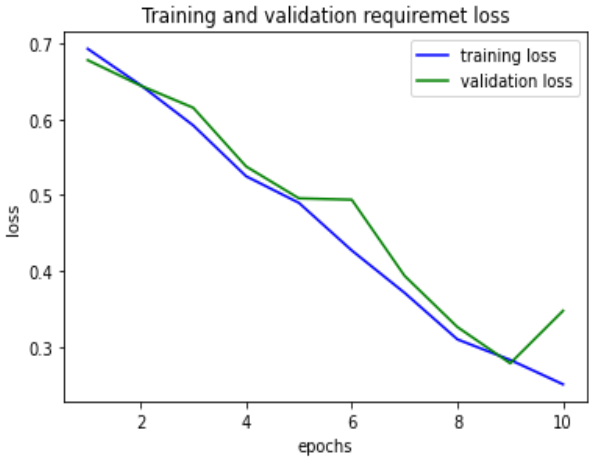
```
timestep=10
model = models.Sequential()
model.add(TimeDistributed(Conv2D(128, (3, 3), strides=(1,1),activation='relu'),input_shape=(timestep,img_high,img_width,3)))
model.add(TimeDistributed(Conv2D(64, (3, 3), strides=(1,1),activation='relu')))
model.add(TimeDistributed(MaxPooling2D(2,2)))
model.add(TimeDistributed(Conv2D(32, (3, 3), strides=(1,1),activation='relu')))
model.add(TimeDistributed(MaxPooling2D(2,2)))
model.add(Dropout(0.5))
model.add(TimeDistributed(Flatten()))

model.add(LSTM(64, return_sequences=True, dropout=0.5))
model.add(Dense(64, activation= 'relu'))
model.add(Dense(32, activation= 'relu'))
model.add(Dense(1, activation='sigmoid'))
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
time_distributed (TimeDistri (None, 10, 218, 218, 128) 3584
_________________________________________________________________
time_distributed_1 (TimeDist (None, 10, 216, 216, 64)  73792
_________________________________________________________________
time_distributed_2 (TimeDist (None, 10, 108, 108, 64)  0
_________________________________________________________________
time_distributed_3 (TimeDist (None, 10, 106, 106, 32)  18464
_________________________________________________________________
time_distributed_4 (TimeDist (None, 10, 53, 53, 32)    0
_________________________________________________________________
dropout (Dropout)            (None, 10, 53, 53, 32)    0
_________________________________________________________________
time_distributed_5 (TimeDist (None, 10, 89888)         0
_________________________________________________________________
lstm (LSTM)                  (None, 10, 64)            23027968
_________________________________________________________________
dense (Dense)                (None, 10, 64)            4160
_________________________________________________________________
dense_1 (Dense)              (None, 10, 32)            2080
_________________________________________________________________
dense_2 (Dense)              (None, 10, 1)             33
=================================================================
Total params: 23,130,081
Trainable params: 23,130,081
Non-trainable params: 0
```


time_distributed_input: InputLayer
input: [(None, 10, 220, 220, 3)]
output: [(None, 10, 220, 220, 3)]
time_distributed(conv2d): TimeDistributed(Conv2D)
input: (None, 10, 220, 220, 3)
output: (None, 10, 218, 218, 128)
time_distributed_1(conv2d_1): TimeDistributed(Conv2D)
input: (None, 10, 218, 218, 128)
output: (None, 10, 216, 216, 64)
time_distributed_2(max_pooling2d): TimeDistributed(MaxPooling2D)
input: (None, 10, 216, 216, 64)
output: (None, 10, 108, 108, 64)
time_distributed_3(conv2d_2): TimeDistributed(Conv2D)
input: (None, 10, 108, 108, 64)
output: (None, 10, 106, 106, 32)
time_distributed_4(max_pooling2d_1): TimeDistributed(MaxPooling2D)
input: (None, 10, 106, 106, 32)
output: (None, 10, 53, 53, 32)
dropout: Dropout
input: (None, 10, 53, 53, 32)
output: (None, 10, 53, 53, 32)
time_distributed_5(flatten): TimeDistributed(Flatten)
input: (None, 10, 53, 53, 32)
output: (None, 10, 89888)
lstm: LSTM
input: (None, 10, 89888)
output: (None, 10, 64)
dense: Dense
input: (None, 10, 64)
output: (None, 10, 64)
dense_1: Dense
input: (None, 10, 64)
output: (None, 10, 32)
dense_2: Dense
input: (None, 10, 32)
output: (None, 10, 1)

I used callback to saves when the model is considered the "best" and the latest best model according to the quantity monitored will not be overwritten.

I achieved a test accuracy of 73% from this architecture. Trained the model for 10 epochs, with the batch size of 5.



It seems the loss increases in validation part after epoch 9 but the accuracy in test data is not high.
In the next submission, I will work on this model to improve it. The
prediction of YouTube videos is satisfied.

In this submission, I added the format of JSON file in my code and after that save the data to JSON file.

(I also added time with the format of second and minute for comparing with the video but saved with time in second)

```
start_time=round(frame_time[0],2)
end_time=round(frame_time[-1],2)
min_s, sec_s = divmod(frame_time[0], 60)
start_time2=str("%d:%02d" % (min_s, sec_s) ) #Starting time of the interval
min_e, sec_e = divmod(frame_time[-1],60)
end_time2 = str("%d:%02d" % (min_e, sec_e) )  # Ending time of the interval
probs.append(prob)
times.append(start_time)
frame_time=[]

if prob>=0.60:
    print(prob,start_time,end_time,start_time2,end_time2)
    data= {"videoId":vid_id,
           "type":"segment",
           "startTime":float(start_time),
           "endTime":float(end_time),
           "observer":nick_name,
           "isHuman":False,
           "confirmedBySomeone": False,
           "rejectedBySomeone": False,"observation":{"label": label,"labelConfidence": 0.85}}
    df.append(data)
```

```
import os
json_dir='/content/gdrive/MyDrive'
filename="CSCE636Spring2021-Farry-6.json"
with open(os.path.join(json_dir,filename), 'w',encoding='utf-8') as f:
      json.dump(df, f,ensure_ascii=False, indent=4)
```

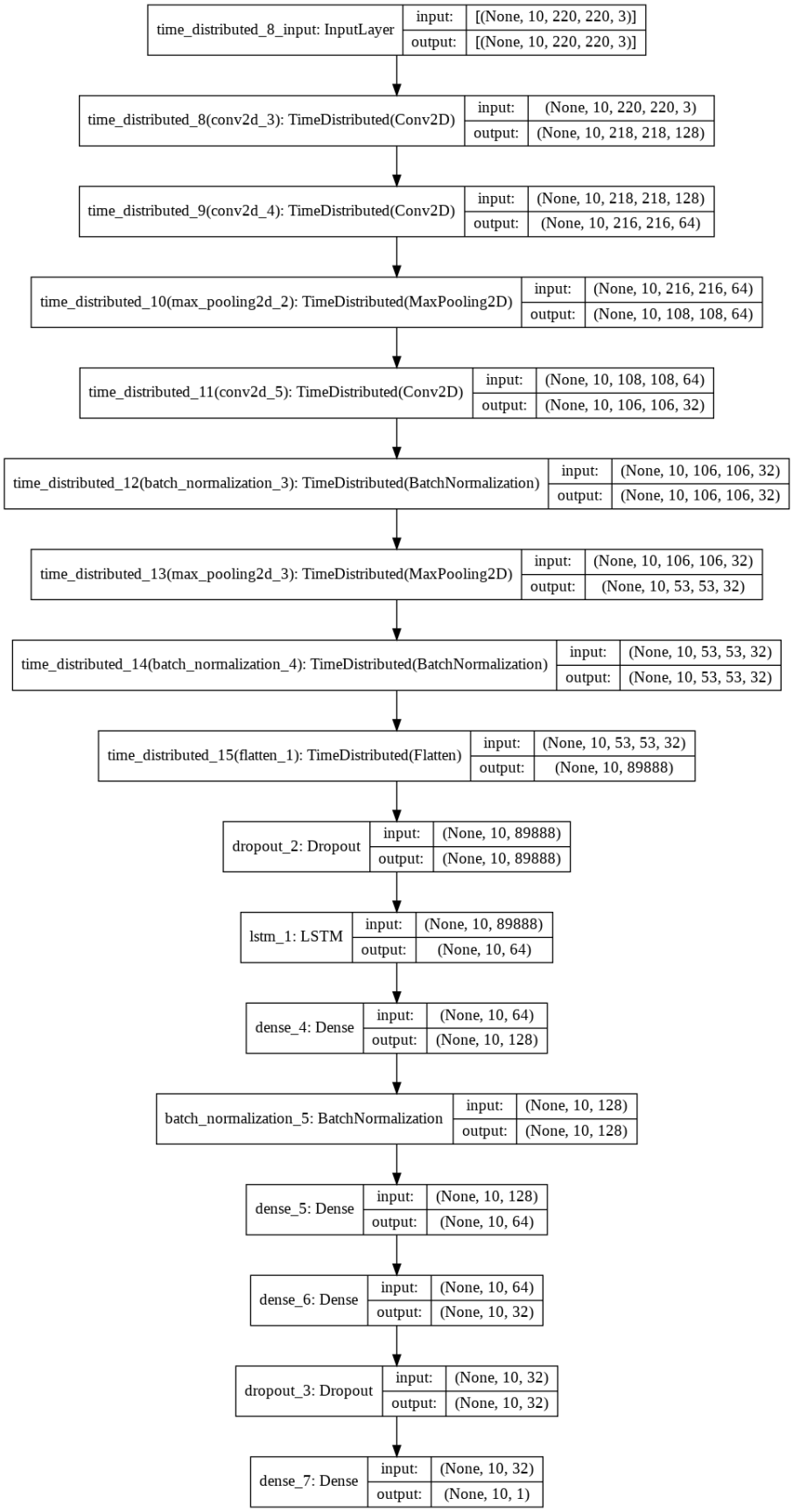
Submission 6 **4/1/2021**

In this submisson the dataset is the same as submission 5.

I tried to improve the model of CNN-LSTM that I built in the last submission. I made it more complex by adding layers. I added batch normalization layer that standardizes the inputs to a layer for each mini-batch.

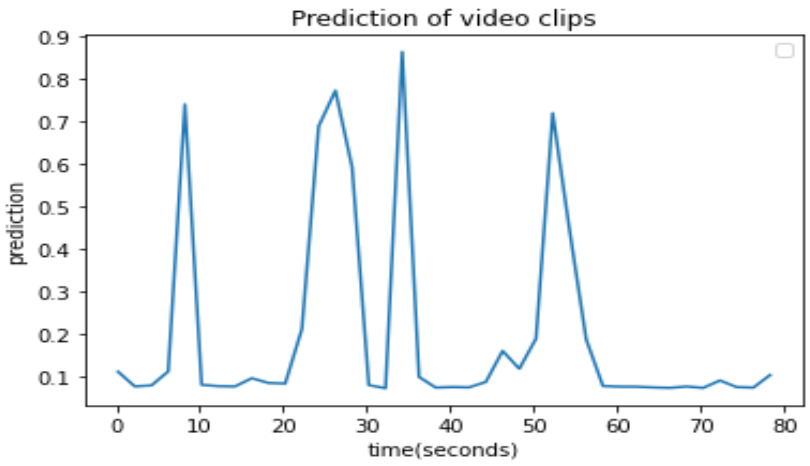
```
timestep=10
model= models.Sequential()
model.add(TimeDistributed(Conv2D(128, (3, 3), strides=(1,1),activation='relu'),input_shape=(timestep,img_high,img_width,3)))
model.add(TimeDistributed(Conv2D(64, (3, 3), strides=(1,1),activation='relu')))
model.add(TimeDistributed(MaxPooling2D(2,2)))
model.add(TimeDistributed(Conv2D(32, (3, 3), strides=(1,1),activation='relu')))
model.add(TimeDistributed(BatchNormalization()))
model.add(TimeDistributed(MaxPooling2D(2,2)))

model.add(TimeDistributed(BatchNormalization()))
model.add(TimeDistributed(Flatten()))
model.add(Dropout(0.5))

model.add(LSTM(64, return_sequences=True, dropout=0.5))
model.add(Dense(128, activation= 'relu'))
model.add(BatchNormalization())
model.add(Dense(64, activation= 'relu'))
model.add(Dense(32, activation= 'relu'))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
time_distributed (TimeDistri (None, 10, 218, 218, 128) 3584
_________________________________________________________________
time_distributed_1 (TimeDist (None, 10, 216, 216, 64)  73792
_________________________________________________________________
time_distributed_2 (TimeDist (None, 10, 108, 108, 64)  0
_________________________________________________________________
time_distributed_3 (TimeDist (None, 10, 106, 106, 32)  18464
_________________________________________________________________
time_distributed_4 (TimeDist (None, 10, 106, 106, 32)  128
_________________________________________________________________
time_distributed_5 (TimeDist (None, 10, 53, 53, 32)    0
_________________________________________________________________
time_distributed_6 (TimeDist (None, 10, 53, 53, 32)    128
_________________________________________________________________
time_distributed_7 (TimeDist (None, 10, 89888)         0
_________________________________________________________________
dropout (Dropout)            (None, 10, 89888)         0
_________________________________________________________________
lstm (LSTM)                  (None, 10, 64)            23027968
_________________________________________________________________
dense (Dense)                (None, 10, 128)           8320
_________________________________________________________________
batch_normalization_2 (Batch (None, 10, 128)           512
_________________________________________________________________
dense_1 (Dense)              (None, 10, 64)            8256
_________________________________________________________________
dense_2 (Dense)              (None, 10, 32)            2080
_________________________________________________________________
dropout_1 (Dropout)          (None, 10, 32)            0
_________________________________________________________________
dense_3 (Dense)              (None, 10, 1)             33
=================================================================
Total params: 23,143,265
Trainable params: 23,142,881
Non-trainable params: 384
_________________________________________________________________
```


time_distributed_8_input: InputLayer | input: | [(None, 10, 220, 220, 3)] | output: | [(None, 10, 220, 220, 3)]
time_distributed_8(conv2d_3): TimeDistributed(Conv2D) | input: | (None, 10, 220, 220, 3) | output: | (None, 10, 218, 218, 128)
time_distributed_9(conv2d_4): TimeDistributed(Conv2D) | input: | (None, 10, 218, 218, 128) | output: | (None, 10, 216, 216, 64)
time_distributed_10(max_pooling2d_2): TimeDistributed(MaxPooling2D) | input: | (None, 10, 216, 216, 64) | output: | (None, 10, 108, 108, 64)
time_distributed_11(conv2d_5): TimeDistributed(Conv2D) | input: | (None, 10, 108, 108, 64) | output: | (None, 10, 106, 106, 32)
time_distributed_12(batch_normalization_3): TimeDistributed(BatchNormalization) | input: | (None, 10, 106, 106, 32) | output: | (None, 10, 106, 106, 32)
time_distributed_13(max_pooling2d_3): TimeDistributed(MaxPooling2D) | input: | (None, 10, 106, 106, 32) | output: | (None, 10, 53, 53, 32)
time_distributed_14(batch_normalization_4): TimeDistributed(BatchNormalization) | input: | (None, 10, 53, 53, 32) | output: | (None, 10, 53, 53, 32)
time_distributed_15(flatten_1): TimeDistributed(Flatten) | input: | (None, 10, 53, 53, 32) | output: | (None, 10, 89888)
dropout_2: Dropout | input: | (None, 10, 89888) | output: | (None, 10, 89888)
lstm_1: LSTM | input: | (None, 10, 89888) | output: | (None, 10, 64)
dense_4: Dense | input: | (None, 10, 64) | output: | (None, 10, 128)
batch_normalization_5: BatchNormalization | input: | (None, 10, 128) | output: | (None, 10, 128)
dense_5: Dense | input: | (None, 10, 128) | output: | (None, 10, 64)
dense_6: Dense | input: | (None, 10, 64) | output: | (None, 10, 32)
dropout_3: Dropout | input: | (None, 10, 32) | output: | (None, 10, 32)
dense_7: Dense | input: | (None, 10, 32) | output: | (None, 10, 1)

The basic architecture gave us the accuracy of 73% on the testing data. But when I made it more complex by adding layers the accuracy is around 61%. So, the adding more layers could not help me to improve the result.

I applied the model to some YouTube videos and the prediction is more than 60% in some intervals of video. The following plot shows the prediction for one video. I compared with the video and I found smoking actions are predicted successfully.



Submission 8 **4/15/2021**

In this submission, I worked on hand waving action that is important action in smart homes. I downloaded 100 videos from HMDB51 dataset for training part which include 60 videos of smoking and 40 videos of other actions. The preprocessing of data is the same as before. I used 3D CNN in this submission, so I changed the input as a way that was compatible with the model.

I split data to training and validation part. Training is 60% of data that include waving action and other actions and 40% for validation. I changed a bit the code and make new folder for saving frames of training videos and another folder for saving validation frames.

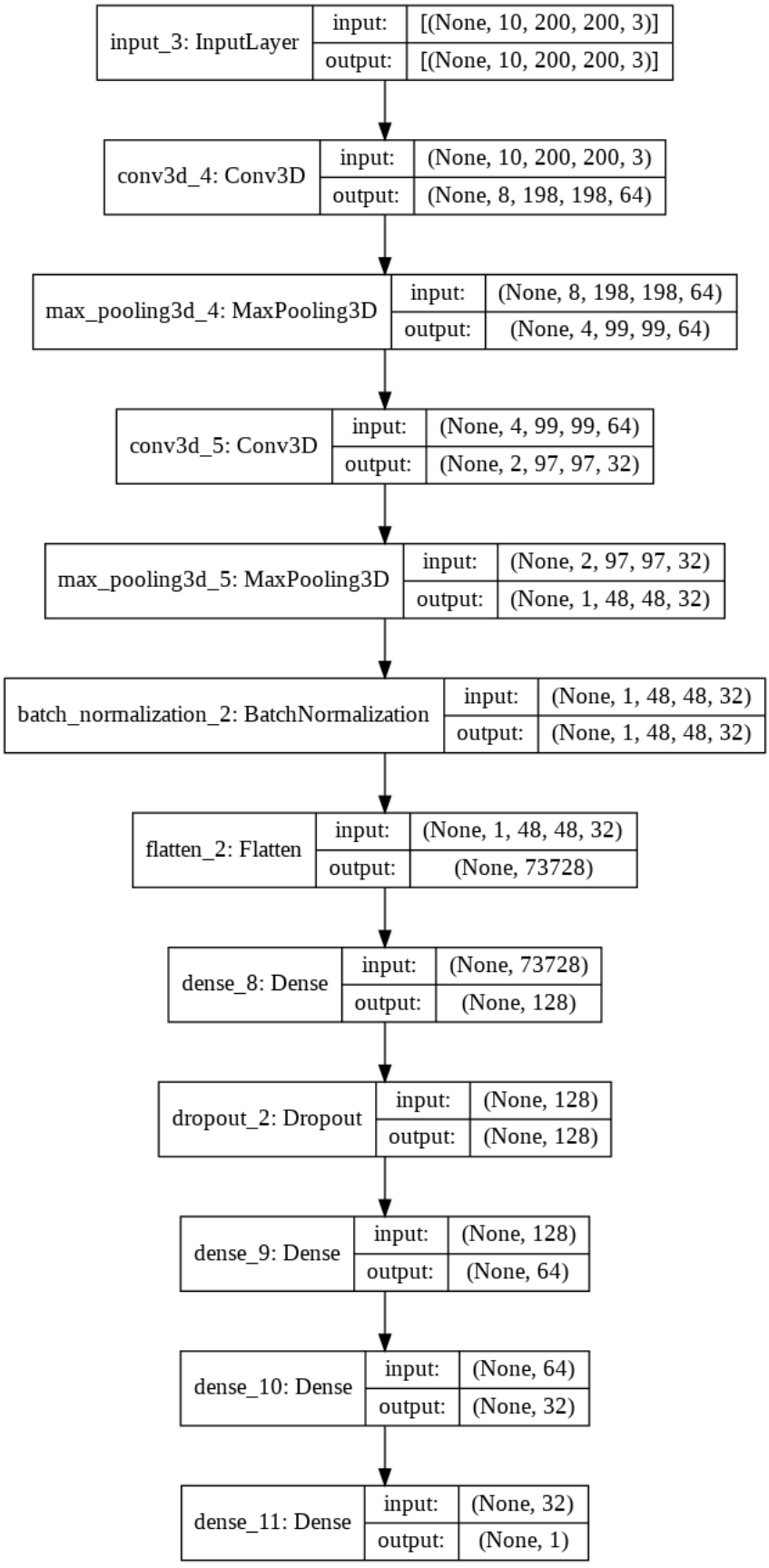
```
video_directory="/content/train_dir/Training"
videos=os.listdir(video_directory)
label=[]
for i in videos:
    if "other_" in i:
        label.append(0)
    else:
        label.append(1)

videos=pd.DataFrame(videos,label).reset_index()
videos.columns=["labels","video_name"]

# 40% videos for validation and 60% for training
other=videos.loc[videos["labels"]==0,]
waiving=videos.loc[videos["labels"]==1,]

other_range=np.arange(len(other))
waiving_range=np.arange(len(waiving))
np.random.seed(10)
np.random.shuffle(other_range)
np.random.shuffle(waiving_range)

waiving=waiving.iloc[waiving_range,]
other=other.iloc[other_range,]

train_waiving=waiving.iloc[:36,]
train_other=other.iloc[:24,]
valid_waiving=waiving.iloc[36:,]
valid_other=other.iloc[24:,]

train_set=train_waiving.append(train_other)
valid_set=valid_waiving.append(valid_other)

train_set=train_set.reset_index().drop("index",axis=1)
valid_set=valid_set.reset_index().drop("index",axis=1)
```

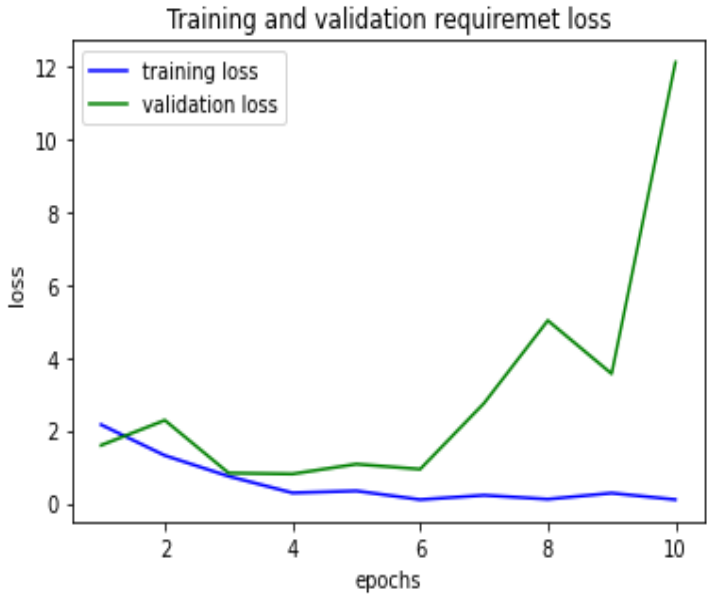
In this submission I built new model, I used 3D convolutional neural network (CNN) to predict the action on the video. 2D CNNs are commonly used to process RGB images (3 channels). A 3D CNN takes as input a 3D volume or a sequence of 2D frames.

```
# 3D Convolutional Model:
input_model=Input(shape=(10,200,200,3))
layer=Conv3D(64,(3,3,3),activation='relu')(input_model)
layer=MaxPooling3D((2,2,2))(layer)
layer=Conv3D(32,(3,3,3),activation='relu')(layer)
layer=MaxPooling3D((2,2,2))(layer)
layer=BatchNormalization()(layer)
layer=Flatten()(layer)
layer=Dense(128,activation='relu')(layer)
layer=Dropout(0.3)(layer)
layer=Dense(64,activation='relu')(layer)
layer=Dense(32,activation='relu')(layer)
layer_output=Dense(1,activation='sigmoid')(layer)

model=Model(input_model,layer_output)

model.summary()
```

Layer (type)	Output Shape	Param #
input_3 (InputLayer)	[(None, 10, 200, 200, 3)]	0
conv3d_4 (Conv3D)	(None, 8, 198, 198, 64)	5248
max_pooling3d_4 (MaxPooling3	(None, 4, 99, 99, 64)	0
conv3d_5 (Conv3D)	(None, 2, 97, 97, 32)	55328
max_pooling3d_5 (MaxPooling3	(None, 1, 48, 48, 32)	0
batch_normalization_2 (Batch	(None, 1, 48, 48, 32)	128
flatten_2 (Flatten)	(None, 73728)	0
dense_8 (Dense)	(None, 128)	9437312
dropout_2 (Dropout)	(None, 128)	0
dense_9 (Dense)	(None, 64)	8256
dense_10 (Dense)	(None, 32)	2080
dense_11 (Dense)	(None, 1)	33

Total params: 9,508,385
Trainable params: 9,508,321
Non-trainable params: 64


input_3: InputLayer
input: [(None, 10, 200, 200, 3)]
output: [(None, 10, 200, 200, 3)]
conv3d_4: Conv3D
input: (None, 10, 200, 200, 3)
output: (None, 8, 198, 198, 64)
max_pooling3d_4: MaxPooling3D
input: (None, 8, 198, 198, 64)
output: (None, 4, 99, 99, 64)
conv3d_5: Conv3D
input: (None, 4, 99, 99, 64)
output: (None, 2, 97, 97, 32)
max_pooling3d_5: MaxPooling3D
input: (None, 2, 97, 97, 32)
output: (None, 1, 48, 48, 32)
batch_normalization_2: BatchNormalization
input: (None, 1, 48, 48, 32)
output: (None, 1, 48, 48, 32)
flatten_2: Flatten
input: (None, 1, 48, 48, 32)
output: (None, 73728)
dense_8: Dense
input: (None, 73728)
output: (None, 128)
dropout_2: Dropout
input: (None, 128)
output: (None, 128)
dense_9: Dense
input: (None, 128)
output: (None, 64)
dense_10: Dense
input: (None, 64)
output: (None, 32)
dense_11: Dense
input: (None, 32)
output: (None, 1)



I evaluated the model on the test data and the accuracy is 66%. For the next submission I am going to improve this model to see if there is way to have better result.

Submission 10 **4/29/2021**

In this submission I am working on kick action. I downloaded 180 videos from HMDB51 dataset for training part which include 100 videos of kicking and 80 videos of other actions. The preprocessing of data is the same as before.

In this submission and previous one, I split data to training and validation and extract the frames of data in different folders. In 3D CNN model, a 3 dimensional filter to the dataset and the filter moves 3-direction (x, y, z) to calculate the low level feature representations. In the submission 8 I built a basic 3D CNN, I got the accuracy of 66% on testing data and the performance of model on YouTube videos was great. In this submission I try to add more layers and change the parameters of model to find if I can improve the model.

```
#3D CNN
input_model=Input(shape=(10,200,200,3))
layer=Conv3D(64,(3,3,3),activation='relu')(input_model)
layer=MaxPooling3D((2,2,2))(layer)
layer=BatchNormalization()(layer)

layer=Conv3D(64,(3,3,3),activation='relu')(layer)
layer=MaxPooling3D((2,2,2))(layer)
layer=BatchNormalization()(layer)
layer=Dropout(0.1)(layer)

layer=Flatten()(layer)
layer=Dense(128,activation='relu')(layer)
layer=Dropout(0.1)(layer)
layer=Dense(64,activation='relu')(layer)
layer_output=Dense(1,activation='sigmoid')(layer)

model=Model(input_model,layer_output)

model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 10, 200, 200, 3)] 0
_________________________________________________________________
conv3d (Conv3D)              (None, 8, 198, 198, 64)   5248
_________________________________________________________________
max_pooling3d (MaxPooling3D) (None, 4, 99, 99, 64)     0
_________________________________________________________________
batch_normalization (BatchNo (None, 4, 99, 99, 64)     256
_________________________________________________________________
conv3d_1 (Conv3D)            (None, 2, 97, 97, 64)     110656
_________________________________________________________________
max_pooling3d_1 (MaxPooling3 (None, 1, 48, 48, 64)     0
_________________________________________________________________
batch_normalization_1 (Batch (None, 1, 48, 48, 64)     256
_________________________________________________________________
dropout (Dropout)            (None, 1, 48, 48, 64)     0
_________________________________________________________________
flatten (Flatten)            (None, 147456)            0
_________________________________________________________________
dense (Dense)                (None, 128)               18874496
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 65
=================================================================
Total params: 18,999,233
Trainable params: 18,998,977
Non-trainable params: 256
```

```
callbacks_model=[keras.callbacks.EarlyStopping(
                monitor='acc',patience=10),
                keras.callbacks.ModelCheckpoint(
                filepath= "/content/gdrive/MyDrive/model10.h5" ,
                monitor='val_loss',
                save_best_only=True),]

optimizer_1=optimizers.RMSprop(lr=0.01)
optimizer_2=optimizers.Adam(lr=0.001)

model.compile(optimizer=optimizer_2,loss='binary_crossentropy',metrics=['acc'])

history=model.fit(x_train,y_train,batch_size=5,epochs=10,
                validation_data=(x_valid,y_valid),callbacks=callbacks_model)
```

The performance of this model on YouTube videos is not so satisfied, because last previous models (CNN-LSTM, CNN) predicted all target actions in video but this model, cannot predict all target actions during video.

Conclusion:

I tried to build different models and different actions to find best model that works better in different actions. The best accuracy that I got on testing data was 75%. To my understanding, I need more data of different actions for training the model and then apply model on YouTube videos because in real videos, there are different actions that similar to the target action. The CNN and LSTM model could predict all target actions in YouTube videos better than other models.

Future Work:

Based on the articles that I studied during this semester, I found, it is better to integrate features of frames of video and other features like pose of body and joints of body. Moreover, I plan to expand my dataset and try to test my models on different datasets.