Project Report on

Crime Detection using Videos

Submitted in partial fulfillment for the award of

Post Graduate Diploma in

ARTIFICIAL INTELLIGENCE From C-DAC, ACTS (Pune)

Guided by:

Mr. Manish Gupta

Presented by:

John Ealias	**200240128011**
Lokendra Carpenter	**200240128014**
Prince Pal	**200240128020**
Shivam Saini	**200240128027**

Centre of Development of Advanced Computing (C-DAC), Pune

ACKNOWLEDGEMENT

This project "**Crime Detection using Videos**" was a great learning experience for us and we are submitting this work to Advanced Computing Training School (CDAC ACTS).

We all are very glad to mention the name of *Mr. Manish Gupta* for his valuable guidance to work on this project. His guidance and support helped us to overcome various obstacles and intricacies during the course of project work.

Our most heartfelt thanks goes to *Miss. Priyanka Ranade* (Course Coordinator, PG- *DAI*) who gave all the required support and kind coordination to provide all the necessities like required hardware and extra Lab hours to complete the project and throughout the course up to the last day here in C-DAC ACTS, Pune.

CHAPTER 1
Abstract

In recent years, surveillance cameras are widely deployed in public places, and the general crime rate has been reduced significantly due to these ubiquitous devices. Usually, these cameras provide cues and evidence after crimes are conducted, while they are rarely used to prevent or stop criminal activities in time. It is both time and labor consuming to manually monitor a large amount of video data from surveillance cameras. However, it is too expensive to monitor a large amount of video data in real-time manually. Thus, automatically recognizing criminal scenes from videos becomes essential and challenging.

We have summarized several existing video datasets for violence detection and used normal and anomalous videos captured by surveillance cameras in real-world scenes. We have tried various approaches and have mentioned the methodology and the results that were obtained from each method. Our best approach was by using the method that utilizes both the merits of 3D-CNNs and optical flow, namely Flow Gated Network.

CHAPTER 2

Methodology / Techniques

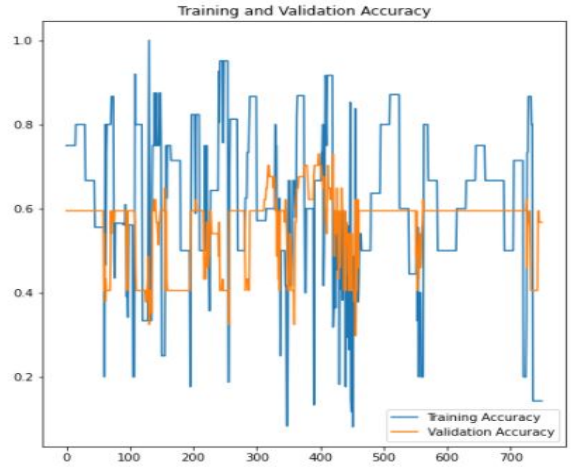
2.1 Numpy – LSTM approach

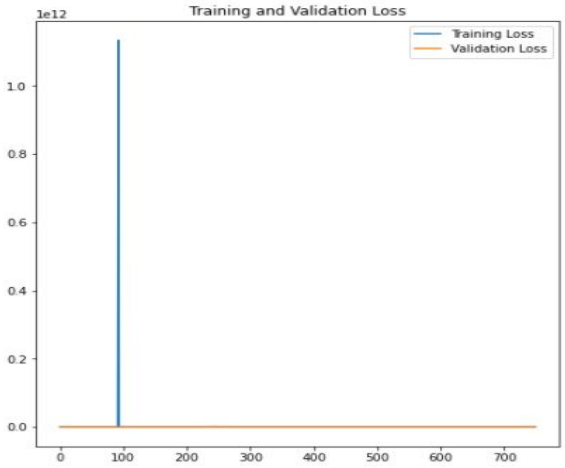
2.1.1 Relationship between Frames within bags

This approach involves extracting the frames from each video and converting them into numpy files as part of preprocessing. The preprocessed numpy files for each frame are bagged together into a bag of 60 frames. Time distributed convolution 2D networks along with max pooling and average polling are used to process the numpy files and extract image information of each bag. The output of time distributed CNNs are passed onto an LSTM network that will extract the sequence information within the frames inside a bag.

```
input_bag = Input(shape=(60,200,200,1))
time_d1 = TimeDistributed(Conv2D(64, (3,3), activation='relu', paddi
ng = "same",kernel_initializer='he_uniform'))(input_bag)
maxpool1 = TimeDistributed(MaxPooling2D())(time_d1)
time_d2 = TimeDistributed(Conv2D(128, (3,3), activation='relu', padd
ing = "same",kernel_initializer='he_uniform'))(maxpool1)
maxpool2 = TimeDistributed(MaxPooling2D())(time_d2)
time_d3 = TimeDistributed(Conv2D(256, (3,3), activation='relu', padd
ing = "same",kernel_initializer='he_uniform'))(maxpool2)
G_average = TimeDistributed(GlobalAveragePooling2D())(time_d3)
lstm = LSTM(1024, activation='relu', return_sequences=False)(G_avera
ge)
Dense1 = Dense(1024, activation = "relu")(lstm)
Dense2 = Dense(512, activation = "relu")(Dense1)
out = Dense(2, activation = "sigmoid")(Dense2)

model1 = Model(input_bag, out)

model1.compile(loss="binary_crossentropy", optimizer = "Adam", metri
cs = "accuracy")
```

Our Accuracy was shifting between very high values and very low values in this approach. Below is the screenshot of the accuracy per epoch plot that was obtained.



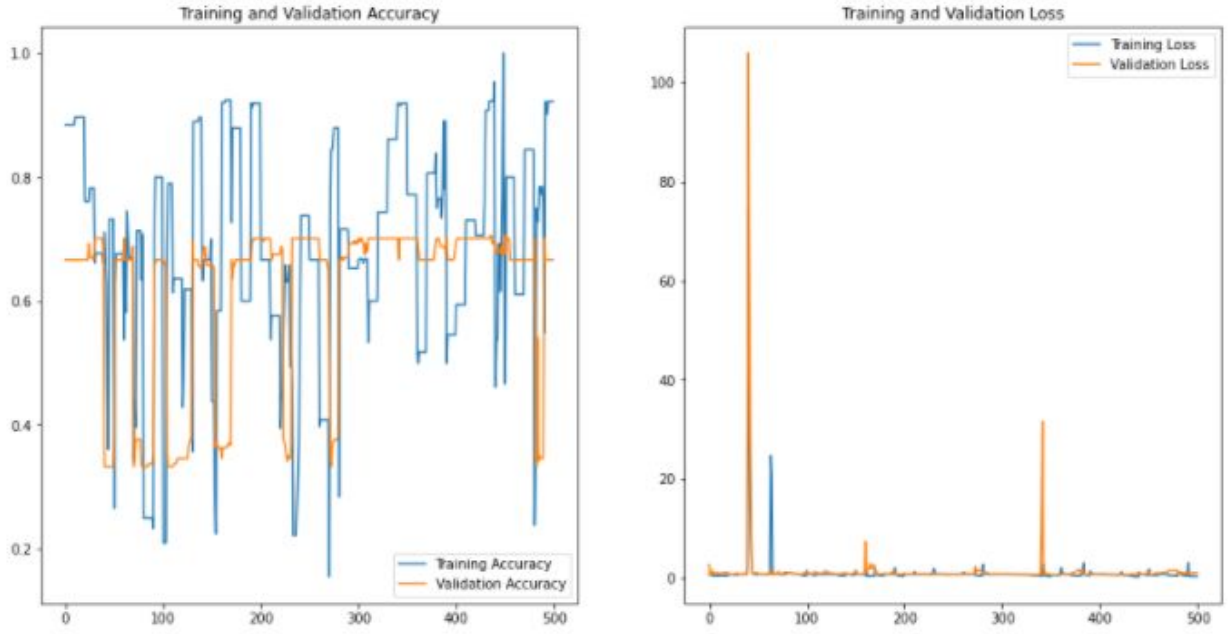


2.1.2 Relationship between bags

In our second approach we tried to extract the image information of frames within a bag using Convolution 3D and this information was fed into an LSTM that is used to extract the information between bags. Here each bag contained 10 frames and the sequential information between every 2 bags is extracted using LSTM

```
# Using Conv3d
input_bag = Input(shape=(10,200,200,1))
conv1 = Conv3D(64, (3,3,3), activation='relu', padding = "same", kernel_initializer='he_
uniform')(input_bag)
maxpool1 = MaxPooling3D()(conv1)
conv2 = Conv3D(128, (3,3,3), activation='relu', padding = "same", kernel_initializer='he
_uniform')(maxpool1)
maxpool2 = MaxPooling3D()(conv2)
conv3 = Conv3D(256, (3,3,3), activation='relu', padding = "same", kernel_initializer='he
_uniform')(maxpool2)

time_d1 = TimeDistributed(GlobalAveragePooling2D())(conv3)
lstm = LSTM(1024, activation='relu', return_sequences=False)(time_d1)
Dense1 = Dense(1024, activation = "relu")(lstm)
Dense2 = Dense(512, activation = "relu")(Dense1)

out = Dense(2, activation = "sigmoid")(Dense2)
model1 = Model(input_bag, out)

model1.compile(loss="binary_crossentropy", optimizer = "Adam", metrics = "accuracy")
```

Our Accuracy was shifting between very high values and very low values in this approach. Below is the screenshot of the accuracy per epoch plot that was obtained.



2.1.3 Main Challenges and drawbacks of these models :

The main challenges of these models include the highly unstable accuracy and loss curves at every epoch which was due to our training method (training one video at a time due to limited ram size)

Every time the new video was given for training, both loss and accuracy suffered a fluctuation.

To remove this we tried the approach using Keras Video Generator and instead of taking full video for normal and anomaly part, we put smaller chunks of video in different folders for normal and anomaly.

2.2 Keras Video Generator.

In this approach we tried using the Keras video generator to extract the frames from the videos. As part of initial pre-processing, we divided the entire video into anomalous and normal videos by extracting the frames using annotations and created normal videos and anomalous videos of similar sizes. Below are the condition applied:

- If the part extracted has less than 120 frames, random frames are added.

```
# generate random frame of n*m*3
def gen_frame(n,m):
    rand_list=[randint(0,255) for i in range(n*m*3)]
    rand_frame=np.array(rand_list,'uint8').reshape(n,m,3)
    return rand_frame
```

```
 if len(img_array)<120:
     r=randint(0,1)
     if r==0:
         for _ in range(120-len(img_array)):
             img_array.append(gen_frame(120,160))
     else:
         for _ in range(120-len(img_array)):
             img_array.insert(0,gen_frame(120,160))

     makeVideo(img_array,folder)
```

- The frames are kept the same if the extracted part is between 120 and 240.

```
if 120<=len(img_array)<=240:
    if folder=='N':
        os.chdir(N)
        file_name='Normal_'+str(n)+'_.mp4'
        n+=1
    else:
        os.chdir(A)
        file_name='Anomaly_'+str(a)+'_.mp4'
        a+=1
    height, width, layers = img_array[0].shape
    video = cv2.VideoWriter(file_name,cv2.VideoWriter_fourcc(*'mp4v'), 30, (width,height))

    for image in img_array:
        video.write(image)

    video.release()
```

- If the extracted part has more than 240 frames, it is converted into shorter videos with 240 frames each.

```
if len(img_array)>240:
    rang=(len(img_array)//240)+1
    temp=0
    for _ in range(rang):
        makeVideo(img_array[temp:temp+240],folder)
        temp+=240
```

By doing this preprocessing we were able generated videos with frame lengths between 120 and 240 frames.

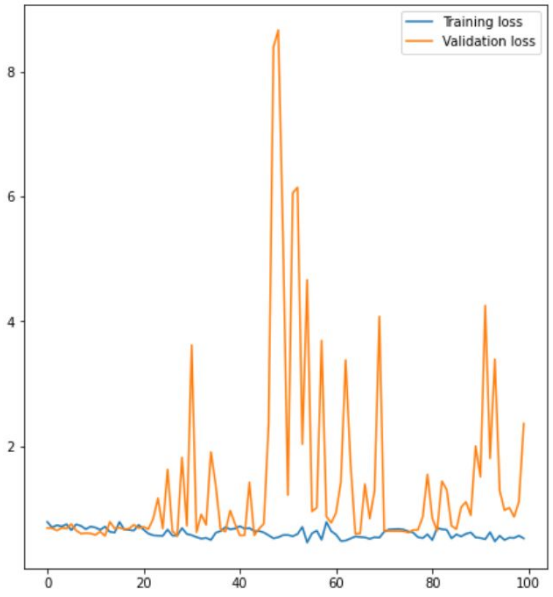
These preprocessed videos are then accessed using keras_video VideoFrameGenerator. Once the videos are in two folders(Normal, Anomalous) the VideoFrameGenerator extracts the videos and labels them as the name of the folder. 80 frames are extracted from each video and then fed into a model with time distributed convolution 2D networks.

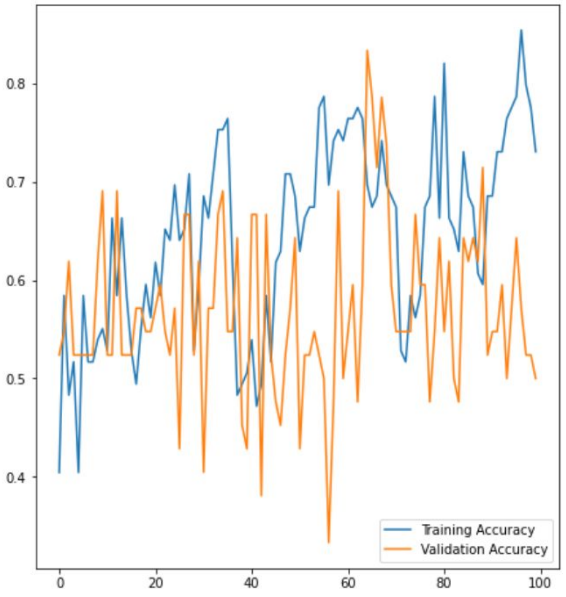
```
from keras.layers import Conv2D, BatchNormalization, \
    MaxPool2D, GlobalMaxPool2D

def build_convnet(shape=(120, 160, 3)):
    momentum = .9
    model = keras.Sequential()
    model.add(Conv2D(64, (3,3), input_shape=shape,
        padding='same', activation='relu'))
    model.add(Conv2D(64, (3,3), padding='same', activation='relu'))
    model.add(BatchNormalization(momentum=momentum))
    model.add(MaxPool2D())
    model.add(Conv2D(128, (3,3), padding='same', activation='relu'))
    model.add(Conv2D(128, (3,3), padding='same', activation='relu'))
    model.add(BatchNormalization(momentum=momentum))
    model.add(MaxPool2D())
    model.add(Conv2D(256, (3,3), padding='same', activation='relu'))
    model.add(Conv2D(256, (3,3), padding='same', activation='relu'))
    model.add(BatchNormalization(momentum=momentum))
    model.add(MaxPool2D())
    model.add(Conv2D(512, (3,3), padding='same', activation='relu'))
    model.add(Conv2D(512, (3,3), padding='same', activation='relu'))
    model.add(BatchNormalization(momentum=momentum))

    # flatten...
    model.add(GlobalMaxPool2D())
    return model
```

```
from keras.layers import TimeDistributed, GRU, Dense, Dropout,LSTM
def action_model(shape=(NBFRAME, 120, 160, 3), nbout=3):

    convnet = build_convnet(shape[1:])

    # then create our final model
    model = keras.Sequential()
    model.add(TimeDistributed(convnet, input_shape=shape))

    model.add(GRU(64))

    model.add(Dense(1024, activation='relu'))
    model.add(Dropout(.5))
    model.add(Dense(512, activation='relu'))
    model.add(Dropout(.5))
    model.add(Dense(128, activation='relu'))
    model.add(Dropout(.5))
    model.add(Dense(64, activation='relu'))
    model.add(Dense(nbout, activation='softmax'))
    return model
```

We got a final accuracy of 68% for the training set but the model was not stable.

Below are the plots for accuracy and loss obtained:





CHAPTER 3

Gated Flow Network

3.1 Flow Diagram

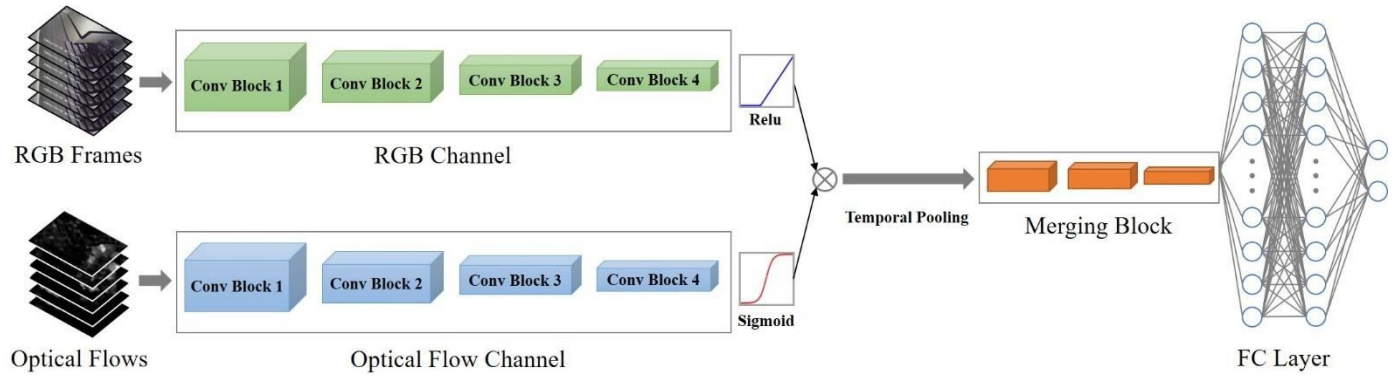


Figure . The structure of the Flow Gated Network

3.2 Cropping and Sampling

Since consecutive frames in a video are highly correlated, and the region of interest for recognizing human activity is usually a small area, we implement both cropping and sampling strategies to reduce the amount of input video data.

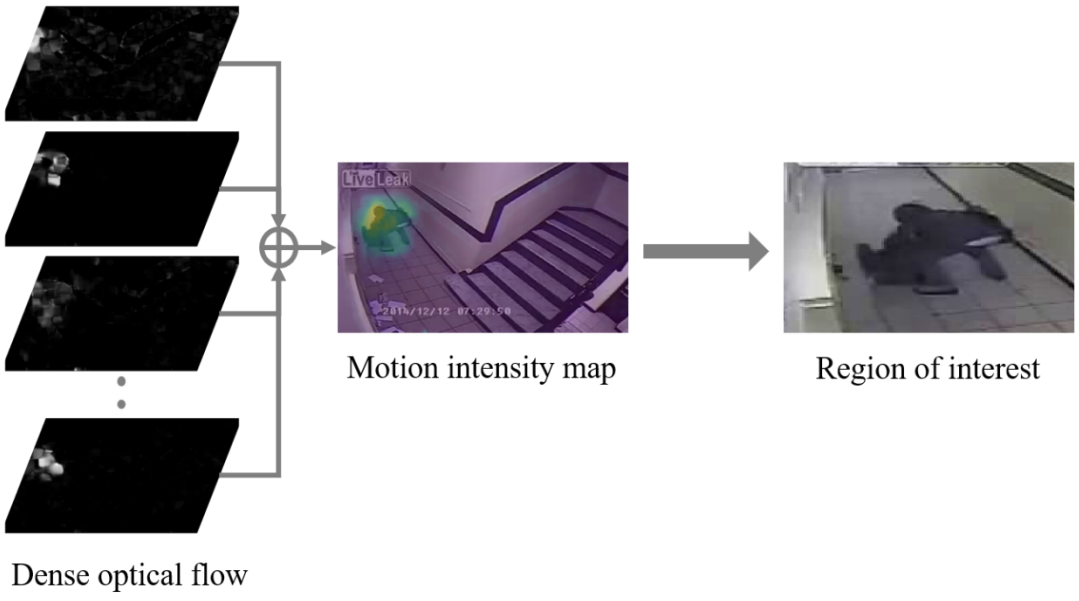
Initially we compute the dense optical flow between neighboring frames. The computed dense optical flow is a field of the 2-D displacement vector. Thus we calculate the norm of each vector to obtain a heat map for indicating the motion intensity. We use the sum of all the heat maps to be a final motion intensity map, then the region of interest is extracted from the location with the most significant motion intensity. Secondly, we implement a sparse sampling strategy. For each input video, we sparsely sample frames from the video by a uniform interval, then generate a fix-length video clip.

By adopting both cropping and sampling, the amount of input data is significantly reduced. In this project, the target length of the video clip is set to 64, and the size of the cropped region is 224 × 224.

3.3 Flow Network

Our model contains four parts: the RGB channel, the Optical Flow channel, the Merging Block, and the Fully Connected Layer. RGB channel and Optical Flow channel are made of cascading 3-D CNNs, and they have consistent structures so that their output could be fused. Merging Block is also composed of basic 3-D CNNs, which is used to process information after self-learned temporal pooling. And the FC layers generate the output.

The highlight of this model is to utilize a branch of the optical flow channel to help build a pooling mechanism. Relu activation is adopted at the end of the RGB channel, while the sigmoid function is placed at the end of the Optical Flow channel. Then, outputs from RGB and Optical Flow channels are multiplied together and processed by a temporal max-pooling. Since the output of the sigmoid function is between 0 and 1, it is a scaling factor to adjust the output of the RGB channel. Meanwhile, since max-pooling just reserve local maximum, the output of RGB channel multiplied by one will have a larger probability to be retained, and the value multiplied by zero is more natural to be dropped. This mechanism is a kind of self-learned pooling mechanism, which utilizes a branch of optical flow as a gate to determine what information the model should preserve or drop. The detailed parameters of the model structure are described in Table 1.



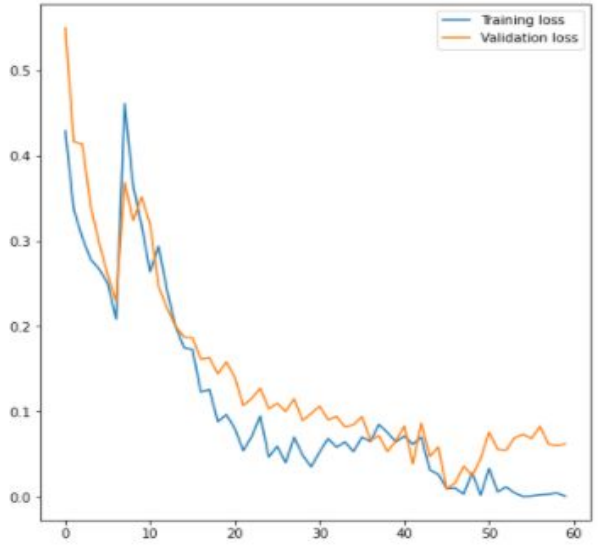
Cropping strategy using dense optical flow.

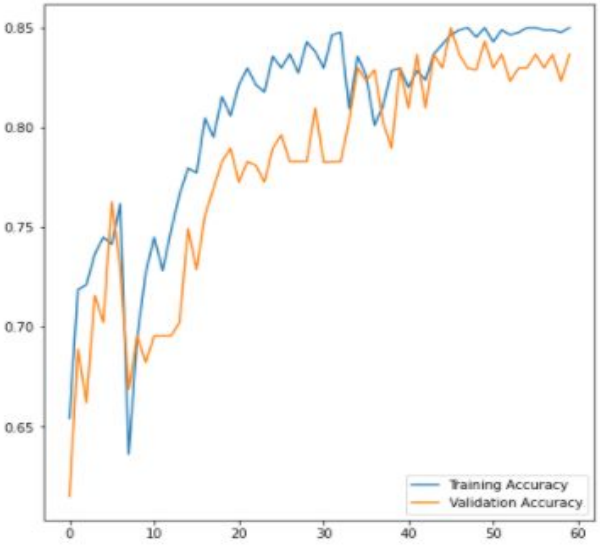
Block Name	Type	Filter Shape	t
	Conv3d	1×3×3@16	
	Conv3d	3×1×1@16	2
RGB/Flow	MaxPool3d	1×2×2	
channels	Conv3d	1×3×3@32	
	Conv3d	3×1×1@32	2
	MaxPool3d	1×2×2	
Fusion and	Multiply	None	1
Pooling	MaxPool3d	8×1×1	
	Conv3d	1×3×3@64	
	Conv3d	3×1×1@64	2
Merging	MaxPool3d	2×2×2	
Block	Conv3d	1×3×3@128	
	Conv3d	3×1×1@128	1
	MaxPool3d	2×2×2	
	FC layer	128	
FC layers	FC layer	128	1
	Softmax	2	

Table : Model parameters. The *t* represents the number of repeat.

3.4 Graph

We were able to attain a training accuracy of around 84 percent.





CHAPTER 4

4.1 Library Requirements

Below are the major libraries we tried out during the course of our project.

4.1.1 Keras Sequence Video generators

This package proposes some classes to work with Keras (included in TensorFlow) that generates batches of frames from video files.

Each of these generators accepts parameters:

- glob_pattern that must contain {classname}, e.g. './videos/{classname}/*.avi' - the "classname" in string is used to detect classes
- nb_frames that is the number of frame in the sequence
- batch_size that is the number of sequence in one batch
- transformation that can be None or or ImageDataGenerator to make data augmentation
- use_frame_cache to use with caution, if set to True, the class will keep frames in memory (without augmentation).

4.1.2 Convolution 2D and 3D

Conv2D layer:

2D convolution layer (e.g. spatial convolution over images)

This layer creates a convolution kernel that is convolved with the layer input to produce a tensor of outputs. If use_bias is True, a bias vector is created and added to the outputs. Finally, if activation is not None, it is applied to the outputs as well.

When using this layer as the first layer in a model, provide the keyword argument input_shape (tuple of integers, does not include the sample axis), e.g. input_shape=(128, 128, 3) for 128x128 RGB pictures in data_format="channels_last".

Conv3D layer

3D convolution layer (e.g. spatial convolution over volumes).

This layer creates a convolution kernel that is convolved with the layer input to produce a tensor of outputs. If use_bias is True, a bias vector is created and added to the outputs. Finally, if activation is not None, it is applied to the outputs as well.

When using this layer as the first layer in a model, provide the keyword argument input_shape (tuple of integers, does not include the sample axis), e.g. input_shape=(128, 128, 128, 1) for 128x128x128 volumes with a single channel, in data_format="channels_last".

4.1.3 LSTM and GRU

Long short-term memory (LSTM)

Long short-term memory (LSTM) units (or blocks) are building units for layers of a recurrent neural network (RNN). A common LSTM unit is composed of a cell, an input gate, an output gate and a forget gate. The cell is responsible for "remembering" values over arbitrary time intervals; hence the word "memory" in LSTM. Each of the three gates can be thought of as a "conventional" artificial neuron, as in a multi-layer (or feedforward) neural network: that is, they compute an activation (using an activation function) of a weighted sum. Intuitively, they can be thought of as regulators of the flow of values that goes through the connections of the LSTM; hence the denotation "gate". There are connections between these gates and the cell.

The expression long short-term refers to the fact that LSTM is a model for the short-term memory which can last for a long period of time. An LSTM is well-suited to classify, process, and predict time series given time lags of unknown size and duration between important events. LSTMs were developed to deal with the exploding and vanishing gradient problem when training traditional RNNs.

Gated Recurrent Units (GRU)

In simple words, the GRU unit does not have to use a memory unit to control the flow of information like the LSTM unit. It can directly make use of all hidden states

without any control. GRUs have fewer parameters and thus may train a bit faster or need fewer data to generalize. But, with large data, the LSTMs with higher expressiveness may lead to better results.

They are almost similar to LSTMs except that they have two gates: reset gate and update gate. The reset gate determines how to combine new input to the previous memory and the update gate determines how much of the previous state to keep. Update gate in GRU is what input gate and forget gate were in LSTM. We don't have the second non-linearity in GRU before calculating the output, nor do they have the output gate.

OPEN CV

OpenCV is released under a BSD license and hence it’s free for both academic and commercial use. It has C++, C, Python, and Java interfaces and supports Windows, Linux, Mac OS, iOS and Android. OpenCV was designed for computational efficiency and with a strong focus on real-time applications. Written in optimized C/C++, the library can take advantage of multi-core processing. Enabled with OpenCL, it can take advantage of the hardware acceleration of the underlying heterogeneous compute platform.

CHAPTER 5

5.1 Results:

Approaches	Results(Accuracy in percent)
Relationship between Frames within bags	unstable model with average accuracy of 52
Relationship between bags	unstable model with average accuracy of 55
Keras Video Generator	better model with slightly improved stability and final accuracy of 68
Gated Flow Network	best accuracy among all models 84%

CHAPTER 6

6.1 Conclusion:

After trying a few approaches and comparing the results, we came to the conclusion that flow gated approach works best in identifying anomaly in a video. Hence we have used this approach in our final model.

This system can be very helpful and can be used to detect the criminal activities in public places and to stop them in time. Further functionality can also be added such as triggering a message to the nearest police station in case of any crime.

Also this system is not limited to detect only criminal activities but can also be used to detect any other anomaly like road accidents, fire etc and report it to the nearest hospital, fire department or any other concerned authority in time and help in saving many lives .

References:

[1] DATA (Real-world Anomaly Detection in Surveillance Videos):

https://www.crcv.ucf.edu/projects/real-world/

[2] Time distributed LSTM -

https://medium.com/smileinnovation/how-to-work-with-time-distributed-data-in-a-neural-network-b8b39aa 4ce00

[3] GRU on image sequence -

https://medium.com/smileinnovation/training-neural-network-with-image-sequence-an-example-with-video -as-input-c3407f7a0b0f

[4] RWF_2000data_for_voilence_detection(Using Conv3d) -

https://github.com/mchengny/RWF2000-Video-Database-for-Violence-Detection

[5] Step-by-Step Deep Learning Tutorial to Build your own Video Classification Model - https://www.analyticsvidhya.com/blog/2019/09/step-by-step-deep-learning-tutorial-video-classification-py thon/

[6] Convolutional LSTM AutoEncoder for anomaly detection -

https://towardsdatascience.com/prototyping-an-anomaly-detection-system-for-videos-step-by-step-using-lstm-convolutional-4e06b7dcdd29