A Project Report

On

CLEVERNATOR

By

Mohammed Jasam

CONTENTS

ABSTRACT

Crime rate has taken a hike in the past decade or two and the effort put by the security forces is not enough to hinder these actions. Sometimes it’s the guards’ fault other times it’s the response time that costs us. Due to the rapid development in technology we have live security footage available at our disposals, this project is intended to improve that response time between the crime and the event being recognized. The project uses Convolutional Neural Networks to predict what’s in the generated frames of the input security footage and an algorithm will decide whether or not the footage contains any violent ongoing action, if abnormalities are detected then an alert regarding the same is sent to authorities.

1. INTRODUCTION

This project has been implemented as a course project for Computational Intelligence SysEng 5211. I thank Dr. Corns for enlightening the field of Intelligence and teaching us important concepts in this revolutionary field such as Fuzzy systems, Neural Networks, Evolutionary Algorithms. I liked the concept of Neural Networks the most which has inspired me in creating this project. The perceptrons that we create mimic real brain neurons and can solve complex tasks if you provide sufficient data and architecture.

1.1 Motivation

We all know that crime has increased a lot in the past few decades and is increasing day by day, so we should contribute towards the betterment of the society. The amazing capabilities of neural networks, availability of huge amount of security video footage and great computation power together have motivated me in creating CleverNator which can impact the society and improve the lives of the people in it.

1.2 Problem Statement

The biggest problem that we face these days is to not respond to crimes in time, which generally leads to loss of property or life. If we can respond to these crimes in time, we can save lots of lives and avoid attacks on innocent people. So, the main goal is to reduce the response time between the crime and the action taken against the crime by the authority (Police, Security Force, etc.)

1.3 Objective of the Project:

This project has two main tasks:

i. Detect violent activity in security footage.

ii. Report it to authority.

1.4 Limitations of the Project:

i. CleverNator needs to be quick in detecting objects in video therefore I had to use an inferior but fast model named ‘Mobilenet v1’.

ii. Due to the lack of GPU, the project has been developed as a CPU-only, the drawbacks of this is that its approximately 500-1000 slow than its counterpart.

iii. Limited Dataset: It was hard to extract correct data for the training purposes as the accuracy of the model could drastically be affected by poor dataset.

2. ANALYSIS

2.1 Existing System

Currently all the inspection of video footage is done manually, it involves minimal to almost no usage of software. We are in a world where you can make the machines do the tasks for us in an efficient manner.

Disadvantages of the Existing System:

i. Manual inspection of the video footage.

ii. High latency between the crime and its identification.

iii. Lack of accuracy in predicting the crime in the footage.

iv. Humans are sometime lazy.

2.2 Proposed system:

In the proposed system, I have overcome the basic problem of the Existing system, that is, human inspection. This project is designed to detect abnormalities automatically and report the authorities about it, so they can take quick action and probably save a life in the process.

Advantages of Proposed System:

i. No human inspection of raw footage required.
ii. Improves the response time drastically and helps detect abnormalities with ease.

2.3 Requirement Specification:

Software Requirements:

i.	Operating system	:	Windows 10
ii.	Language	:	Python 3.5.3
iii.	IDE/Editor	:	Atom Text Editor

Dependencies:

i. TensorFlow – This is a module which provides tools for Deep Learning Algorithms.
ii. OpenCV – This module helps us interact with the webcam footage of our system.
iii. Twilio API – This API is utilized to send SMS.
iv. Numpy - This module is utilized to do large scale computations.
v. Glob – This module has been used to extract all filenames from the given directory.

Hardware Requirements:

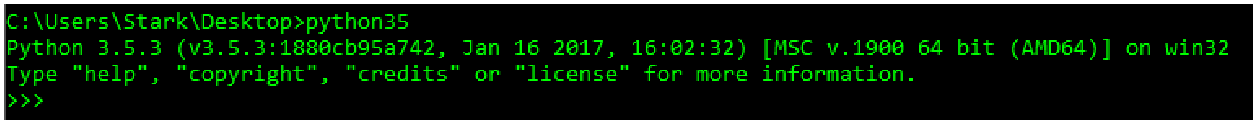
i.	Processor	:	Intel Core i5+
ii.	RAM	:	8+ Gigabytes

3. ENVIRONMENT SETUP

This section will provide step by step procedure for setting up your environment to run the project and also to tweak with it.

Python 3.5.3

i. To install python, navigate to python.org and search for python 3.5.3. Now click on Windows x86-64 web-based installer.
ii. During the installation check the "Add Python to PATH" checkbox which will add your python to environment variables.
iii. Once the setup is complete, open Command Prompt and type *python* to see the version information. It should look something like this. Type "*exit()*" to exit from python.



```
C:\Users\Stark\Desktop>python35
Python 3.5.3 (v3.5.3:1880cb95a742, Jan 16 2017, 16:02:32) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Atom Text Editor

This editor is very good for python programming as it supports many packages which will help you create a lightweight IDE. To download ATOM follow the steps:

i. Navigate to *atom.io* and click on Download Windows 64-bit installer.
ii. Follow the setup and it should install atom successfully.

TensorFlow

i. Open Command Prompt in admin mode.
ii. Type *pip install tensorflow*
iii. This command should download all the necessary packages to successfully install tensorflow on your system.

If that doesn't work, then install the library using Anaconda, follow the steps below:

i. Follow the instructions on the Anaconda download site to download and install Anaconda.
ii. Create a conda environment named tensorflow by invoking the following command:
C:> conda create -n tensorflow python=3.5

iii. Activate the conda environment by issuing the following command:

iv. C:> activate tensorflow

(tensorflow) C:> # Your prompt should change

v. Issue the appropriate command to install TensorFlow inside your conda environment. To install the CPU-only version of TensorFlow, enter the following command:

(tensorflow)C:> pip install --ignore-installed --upgrade tensorflow

To validate your installation:

i. Start a terminal. If you installed through Anaconda, activate your Anaconda environment.

ii. Invoke python from your shell as follows:

$ *python*

iii. Enter the following short program inside the python interactive shell:

```
>>> import tensorflow as tf
>>> hello = tf.constant('Hello, TensorFlow!')
>>> sess = tf.Session()
>>> print(sess.run(hello))
```

It should output *Hello, TensorFlow!*

OpenCV

i. Open Command Prompt in admin mode.

ii. Type *pip install opencv-python*

iii. This should install any necessary files needed by OpenCV for successful installation.

Twilio API

i. Open Command Prompt in admin mode.

ii. Type *pip install twilio*

iii. This should install any necessary files needed by Twilio for successful installation.

Glob

i. Open Command Prompt in admin mode.

ii. Type *pip install glob*

4. MODULES

This section describes the different modules in the project and also their contributions for successful execution of this project.

4.1 Image Classifier

This module is the main part of this project, it’s task is to classify the image passed to it as an input. The core API utilized for this module is the Google’s TensorFlow API. TensorFlow is an open source library for numerical computation, specializing in machine learning applications. It has some of the state-of-the-art algorithms built in, so its easy for others to utilize those algorithms easily. The Image Recognition Model that I’ve used in this project is *Mobilenet v1*.

4.2 Video Logger

This module is responsible to extract video from the camera source, in this case I’ve used my webcam as the video source. Other video cameras can also be connected by changing the parameter from 0 to 1, 2, 3 … so on.

4.3 Frame Generator

For generating frames from the raw video footage for testing purposes, this module can be utilized. This module can generate the required frames per seconds and save them in a folder for later image classification process.

4.4 Message Alert Delivery System

Once something violent has been detected by the classifier, the immediate task is to alert the authorities about the abnormality so that they can take care of the situation.

5. MODEL OVERVIEW

Mobilenet v1

MobileNets are light-weight, low-power, low-latency models configured to suit variety of computational needs. It can be utilized for image classification, object detection, image embeddings just like the other bulky accurate models like Google Inception, etc. I started the project by training the Inception model, which is huge with its weights and config files. The training time for inception was also more compared to the current model (Mobilenet). I chose to shift to Mobilenet as it is lightweight, trains faster and more importantly the prediction time is very less, which is one of the main goals of this project. Using Inception model my single image prediction took about 5-10 seconds, and assuming there were 10 frames per second, if we consider a one-minute footage it consists of approximately 600 frames and classifying them using Inception would take approximately 100 minutes which is ridiculous. While the Mobilenet does a good job in classifying approximately 30 frames per second which is an acceptable latency. The biggest trade-off between these two models is accuracy and latency, Google's Inception is amazingly accurate, giving out accurate predictions approximately 90% of the time but it is slow, while the Mobilenet is amazingly fast but the accuracy is approximately 70%.

Mobilenet Overview:

The main idea of Mobilenet is – Using *depthwise separable convolutions* to construct light weight deep neural networks. Generally, a Convolutional layer applies a convolution kernel (filter) to all of the channels in the input. The layer moves the filter across the image, here it computes the weighted sum of the input pixels covered by the filter along all the channels of the input image. This convolution task combines the values of all channels of the input, which is a really important part of Mobilenet. For example, if the input image had 3 input channels, then the output image after the convolution operation has only 1 channel per pixel.

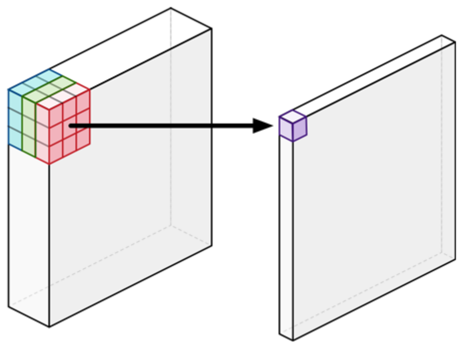


Figure 1: Convolution operation to generate 1 channel from multiple input channels per pixel.

Standard convolution is also used by the MobileNets only once in the first layer. All other layers perform the "depthwise separable" convolution. A Depthwise separable is a combination of two different convolution tasks; a depthwise convolution and a pointwise convolution.

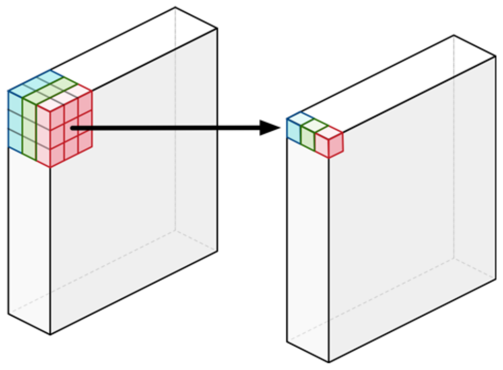


Figure 2: A depthwise convolution operation

It performs convolution on each channel separately unlike regular convolution layers. For an image with 3 channels, an output image will also have 3 channels. Each channel gets its own set of weights, which is created by a depthwise convolution. Its main task is to filter input channels for edge detection, color filtering, etc. The depthwise convolution is followed by a pointwise convolution. This really is the same as a regular convolution but with a kernel which takes input channels and produces one output channel in the output image as shown below:

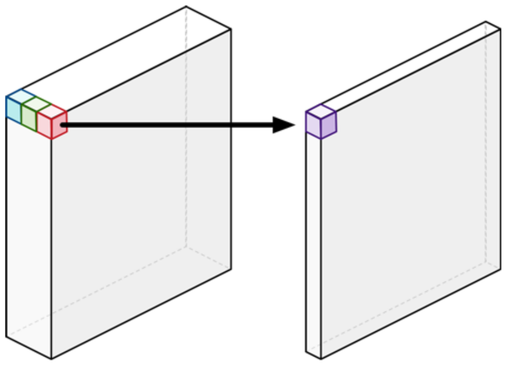


Figure 3: A pointwise convolution operation

It adds up all the channels (as a weighted sum). Just like general convolution; to create an output image with many channels we combine many of these pointwise kernels. The purpose of this pointwise convolution is to *combine* the output channels of the depthwise convolution to create new features.

By combining the depthwise convolution and pointwise convolution, we obtain a depthwise separable convolution. A depthwise separable convolution does the above two tasks separately, while a regular convolution does both filtering and combining in a single go. By separating the tasks, the depthwise separable has to do far less tasks than a regular convolution, thereby reducing huge computation power usage and also does it a lot quicker than the regular convolution. MobileNets uses up to 13 of these depthwise separable convolutions in a row. Hence, it is fast, with decent accuracy and is perfectly suitable for lightweight applications.

MobileNet Architecture

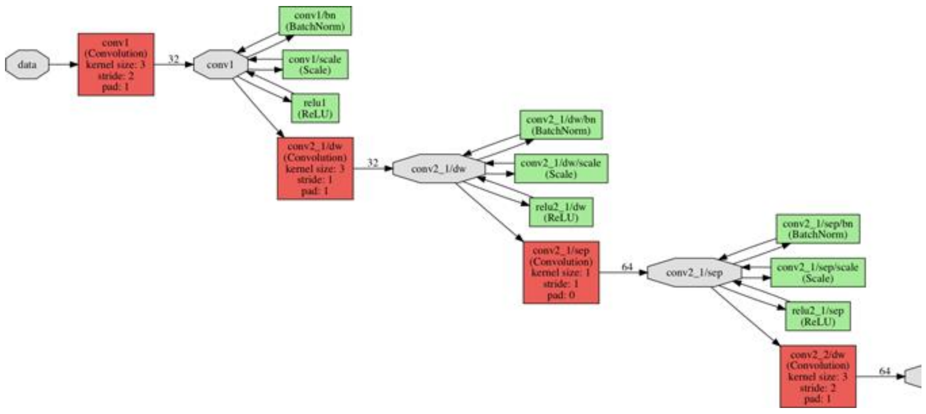

data
conv1 (Convolution) kernel size: 3 stride: 2 pad: 1
32
conv1
conv1/bn (BatchNorm)
conv1/scale (Scale)
relu1 (ReLU)
conv2_1/dw (Convolution) kernel size: 3 stride: 1 pad: 1
32
conv2_1/dw
conv2_1/dw/bn (BatchNorm)
conv2_1/dw/scale (Scale)
relu2_1/dw (ReLU)
conv2_1/sep (Convolution) kernel size: 1 stride: 1 pad: 0
64
conv2_1/sep
conv2_1/sep/bn (BatchNorm)
conv2_1/sep/scale (Scale)
relu2_1/sep (ReLU)
conv2_2/dw (Convolution) kernel size: 3 stride: 2 pad: 1
64

Figure 4: The MobileNet Architecture

The full MobileNets network has 30 layers. The design of the network is quite straightforward:

1. convolutional layer with stride 2
2. depthwise layer
3. pointwise layer that doubles the number of channels
4. depthwise layer with stride 2
5. pointwise layer that doubles the number of channels
6. depthwise layer
7. pointwise layer
8. depthwise layer with stride 2
9. pointwise layer that doubles the number of channels

and so on…

After the very first layer (a regular convolution), the depthwise and pointwise layers take turns. Sometimes the depthwise layer has a stride of 2, to reduce the width and height of the data as it flows through the network. Sometimes the pointwise layer doubles the number of channels in the data. All the convolutional layers are followed by a ReLU activation function. This goes on for a while until the original 224×224 image is shrunk down to 7×7 pixels but now has 1024 channels. After this there's an average-pooling layer that works on the entire image so that we end up with a 1×1×1024 image, which is really just a vector of 1024 elements.

If we're using MobileNets as a classifier, for example on ImageNet which has 1000 possible categories, then the final layer is a fully-connected layer with a softmax and 1000 outputs. If you wanted to use MobileNets on a different dataset, or as a feature extractor instead of classifier, you'd use some other final layer instead.

Even though MobileNets is designed to be pretty fast already, it's possible to use a reduced version of this network architecture. There are three hyperparameters you can set that determine the size of the network:

- **Width multiplier** (the paper calls this "alpha"): this shrinks the number of channels. If the width multiplier is 1, the network starts off with 32 channels and ends up with 1024.
- **Resolution multiplier** ("rho" in the paper): this shrinks the dimensions of the input image. The default input size is 224×224 pixels.
- **Shallow or deep**. The full network has a group of 5 layers in the middle that you can leave out.

These settings can be used to make the network smaller — and therefore faster — but at the cost of prediction accuracy. (If you’re curious, the paper shows the effects of changing these hyperparameters on the accuracy.)

For the full network the total number of learned parameters is **4,221,032** (after folding the batch normalization layers). That’s certainly a lot less than VGGNet, which has over 130 million!

6. IMPLEMENTATION

Label_image.py

```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function
import glob
import sys
from twilio.rest import TwilioRestClient
import argparse
import sys
import time

import numpy as np
import tensorflow as tf

# Loads the graph from the tf files
def load_graph(model_file):
   graph = tf.Graph()
   graph_def = tf.GraphDef()
   with open(model_file, "rb") as f:
      graph_def.ParseFromString(f.read())
   with graph.as_default():
      tf.import_graph_def(graph_def)
   return graph

def read_tensor_from_image_file(file_name, input_height=299, input_width=299,
input_mean=0, input_std=255):
   input_name = "file_reader"
   output_name = "normalized"
   file_reader = tf.read_file(file_name, input_name)

   if file_name.endswith(".png"):
      image_reader = tf.image.decode_png(file_reader, channels = 3, name='png_reader')
   elif file_name.endswith(".gif"):
      image_reader = tf.squeeze(tf.image.decode_gif(file_reader, name='gif_reader'))
```

```
    elif file_name.endswith(".bmp"):
        image_reader = tf.image.decode_bmp(file_reader, name='bmp_reader')
    else:
        image_reader = tf.image.decode_jpeg(file_reader, channels = 3, name='jpeg_reader')

    float_caster = tf.cast(image_reader, tf.float32)
    dims_expander = tf.expand_dims(float_caster, 0);
    resized = tf.image.resize_bilinear(dims_expander, [input_height, input_width])
    normalized = tf.divide(tf.subtract(resized, [input_mean]), [input_std])
    sess = tf.Session()
    result = sess.run(normalized)

    return result
countLabels = {'guns':0, 'fist fight':0, 'knife':0, 'vandalism':0, 'robbery':0, 'normal':0}

from textmagic.rest import TextmagicRestClient
import smtplib

# Sends message when situation is abnormal
def send_message():
    print("Sending message")
    server = smtplib.SMTP('smtp.gmail.com', 587)
    server.starttls()
    server.login("sender_email", "sender_pass")

    msg = "Situation is Abnormal"

    server.sendmail("sender_email", "receiver_email", msg)
    server.quit()

# Checks for abnormality
def check_Max():
    # print("In Check MAX")
    maxi = 0
    top = ""
    for k, v in countLabels.items():
        if maxi < v:
            top = k
    if not (top == "normal"):
        print("Situation is abnormal")
        send_message()

# Loads the different classes learnt from training data
def load_labels(label_file):
    label = []
    proto_as_ascii_lines = tf.gfile.GFile(label_file).readlines()
    for l in proto_as_ascii_lines:
        label.append(l.rstrip())
    return label

# excuted the whole program, classifies the image and sends message if situation is abnormal
if __name__ == "__main__":
    file_name = "tf_files/flower_photos/daisy/3475870145_685a19116d.jpg"
    model_file = "tf_files/retrained_graph.pb"
    label_file = "tf_files/retrained_labels.txt"
    input_height = 224
    input_width = 224
    input_mean = 128
    input_std = 128
    input_layer = "input"
    output_layer = "final_result"

    parser = argparse.ArgumentParser()
    parser.add_argument("--image", help="image to be processed")
    parser.add_argument("--graph", help="graph/model to be executed")
    parser.add_argument("--labels", help="name of file containing labels")
    parser.add_argument("--input_height", type=int, help="input height")
    parser.add_argument("--input_width", type=int, help="input width")
    parser.add_argument("--input_mean", type=int, help="input mean")
    parser.add_argument("--input_std", type=int, help="input std")
    parser.add_argument("--input_layer", help="name of input layer")
    parser.add_argument("--output_layer", help="name of output layer")
    args = parser.parse_args()

    if args.graph:
        model_file = args.graph
    if args.image:
        file_name = args.image
    if args.labels:
        label_file = args.labels
    if args.input_height:
        input_height = args.input_height
    if args.input_width:
        input_width = args.input_width
    if args.input_mean:
        input_mean = args.input_mean
    if args.input_std:
        input_std = args.input_std
    if args.input_layer:
        input_layer = args.input_layer
    if args.output_layer:
        output_layer = args.output_layer
    directory = glob.glob(file_name + "/*.jpg")
    graph = load_graph(model_file)

    c = 0
    for x in directory:
        c+=1
        if (countLabels['guns'] - countLabels['normal'] > 10 and countLabels['guns'] > 0 and
countLabels['fist fight'] > 0 and countLabels['knife'] > 0 and countLabels['vandalism'] > 0 and
countLabels['robbery'] > 0 ) or (countLabels['fist fight'] - countLabels['normal'] > 10  and
countLabels['guns'] > 0 and countLabels['fist fight'] > 0 and countLabels['knife'] > 0 and
countLabels['vandalism'] > 0 and countLabels['robbery'] > 0 ) or (countLabels['knife'] -
countLabels['normal'] > 10  and  countLabels['guns'] > 0 and countLabels['fist fight'] > 0 and
countLabels['knife'] > 0 and countLabels['vandalism'] > 0 and countLabels['robbery'] > 0 ) or
(countLabels['vandalism'] - countLabels['normal'] > 10  and countLabels['guns'] > 0 and
countLabels['fist fight'] > 0 and countLabels['knife'] > 0 and countLabels['vandalism'] > 0 and
countLabels['robbery'] > 0 ) or (countLabels['robbery'] - countLabels['normal'] > 10  and
countLabels['guns'] > 0 and countLabels['fist fight'] > 0 and countLabels['knife'] > 0 and
countLabels['vandalism'] > 0 and countLabels['robbery'] > 0 ):
            print("Situation is Abnormal")
            send_message()
            break

        if c % 10 == 0:
            print("Scanned" , c, "frames!")

        t = read_tensor_from_image_file(x, input_height=input_height, input_width=input_width,
input_mean=input_mean, input_std=input_std)

        input_name = "import/" + input_layer
        output_name = "import/" + output_layer
        input_operation = graph.get_operation_by_name(input_name);
        output_operation = graph.get_operation_by_name(output_name);

        with tf.Session(graph=graph) as sess:
            start = time.time()
            results = sess.run(output_operation.outputs[0], {input_operation.outputs[0]: t})
            end=time.time()
        results = np.squeeze(results)

        top_k = results.argsort()[-5:][::-1]
        labels = load_labels(label_file)


        countLabels[labels[top_k[0]]] +=1
    check_Max()
    print(countLabels)
```

Generate_frames_from_the_video.py

```
import cv2
import sys

vid_name = sys.argv[1]
import os
vidcap = cv2.VideoCapture(vid_name)
success,image = vidcap.read()
count = 0
success = True
os.chdir("path to output folder of the frames”)
# os.chdir(path)
while success:
  success, image = vidcap.read()
  if count % 100 == 0: # Captures 1 frame per second
    print('Read a new frame: ', count)
    cv2.imwrite("frame%d.jpg" % count, image)     # save frame as JPEG file
  count += 1
```

record_vid_from_webcam.py

```
import cv2
import time
import os
print(cv2.__version__)
import numpy as np
if not os.path.exists("generated_frames"):
    os.makedirs("generated _frames")

if not os.path.exists("flagged_clips"):
    os.makedirs("flagged_clips")

if not os.path.exists("video_log"):
    os.makedirs("video_log")

#number of frames being sent to clarifai
generated _fps = 6
log_time = 5
flagging = False

#select the capture device, 0 is the first, 1 is the second and so on
#Enter the name of a file to look at a already existing video file
cap = cv2.VideoCapture(0)
```

```
#initialize the codec
fourcc = cv2.VideoWriter_fourcc(*'XVID')
# fourcc = cv2.cv.CV_FOURCC(*'XVID')

#initialize output object
session_time = time.strftime("%d-%m-%Y")
out = cv2.VideoWriter("video_log/"+session_time+".avi",fourcc,20,(640,480))

clip_time = 0
frame_index = 0
clip_index = 0

clip_directory = ""

current_time = time.time()


#make directory for the frames from this session
if not os.path.exists("generated _frames/"+session_time):
    os.makedirs("generated _frames/"+session_time)

if not os.path.exists("flagged_clips/"+session_time):
    os.makedirs("flagged_clips/"+session_time)




while True:

    #ret is a boolean and frame is the frame
    ret,frame = cap.read()
    #write the frame to the file
    out.write(frame)
    #show the image (the frame)
    cv2.imshow("frame",frame)
    new_time = time.time()


    #save frames at the rate of the fps specified
    if(new_time-current_time >= 1.0/ generated _fps):
        new_time_str = str(new_time).replace(".","")
        cv2.imwrite("generated_frames/"+session_time+"/"+str(frame_index)+".jpg",frame)
        current_time = time.time()
        frame_index +=1
    if(new_time-clip_time <= log_time):
        flagging = True
        print(new_time-clip_time)
        print("writing frame")
        clip.write(frame)
    else:
        flagging = False



    #break loop if q pressed
    k = cv2.waitKey(1)&0xFF
    if k == ord('q'):
        break
    #REPLACE 'k == ord('f')' with model identifying abnormalities
    if k == ord('f') and not flagging:
        clip_time = time.time()
        clip_time_str = str(new_time).replace(".","")
        clip_directory = "flagged_clips/"+session_time+"/"+str(clip_index)+".avi"
        clip = cv2.VideoWriter(clip_directory,fourcc,20,(640,480))
        clip_index += 1

cap.release()
out.release()
cv2.destroyAllWindows()
```

7. TRAINING AND TESTING

The Mobilenet model was trained using 500 images on 6 different classes namely; *guns, knife, fist fight, vandalism, robbery and normal.* The training was done using a technique called transfer learning.

Transfer Learning is the process of training a pretrained model to recognize new features, this is generally efficient and save weeks of time in training. We only configure the final layers of the model to predict the classes that we want.

The training was done on the Mobilenet version 0.50.224 for 5000 steps, where each step would consist of Training, Validation and cross entropy. The testing was done using 5-fold cross validation; this method is widely popular as the data is divided into 5 parts where 1 part is used for testing and rest 4 parts are used for training, and these are permuted to cover the entire dataset.

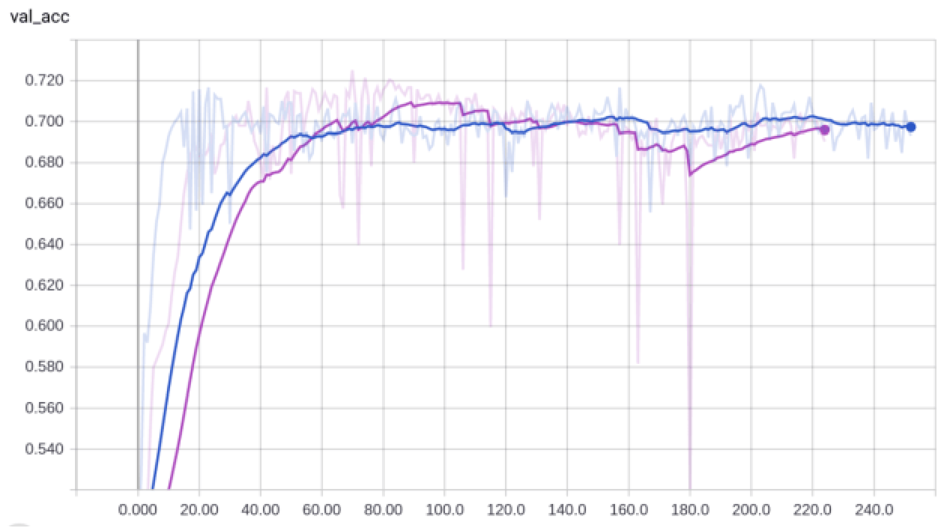


Figure 5: An overall validation accuracy achieved after 240 epochs

8. EXECUTION

Below is the command list used to successfully execute the project to obtain results, both webcam and video versions are included.

Training the Model:

i. The model can be custom trained with your own set of classes; to do this, first gather the image dataset and place them in a folder named trainData and place that folder in *ClevaNator-v2/violent_predictor/tf_files*

ii. After placing the dataset in the right folder, execute the following command

```
python35 -m tensorflow-for-poets-2.scripts.retrain
    --bottleneck_dir=tf_files/bottlenecks
    --how_many_training_steps=5000
    --model_dir=tf_files/models/
    --summaries_dir=tf_files/training_summaries/"mobilenet_0.50_224"
    --output_graph=tf_files/retrained_graph.pb
    --output_labels=tf_files/retrained_labels.txt
    --architecture="mobilenet_0.50_224"
    --image_dir=tf_files/trainData
```

Predicting output from webcam:

i. Navigate to ClevaNator-v2/violent_predictor/Video_Logger/

ii. Execute the command

```
python record_vid_from_webcam.py
```

iii. Record the necessary video and press q to quit the webcam recording.

iv. This will record the video and generate frames in the folder *generated_frames*

v. Now that you have the frames, execute the command

python -m tensorflow-for-poets-2.scripts.label_image
--graph=tf_files/retrained_graph.pb
--image=path to /ClevaNator-v2/violent_predictor/Video_Logger/generated_frames/

Predicting output from frames generated from a video:

i. Navigate to ClevaNator-v2/violent_predictor/TestData

ii. Place the video which you want to predict in that folder (video_filename is the name of the video that you've just placed in the folder TestData)

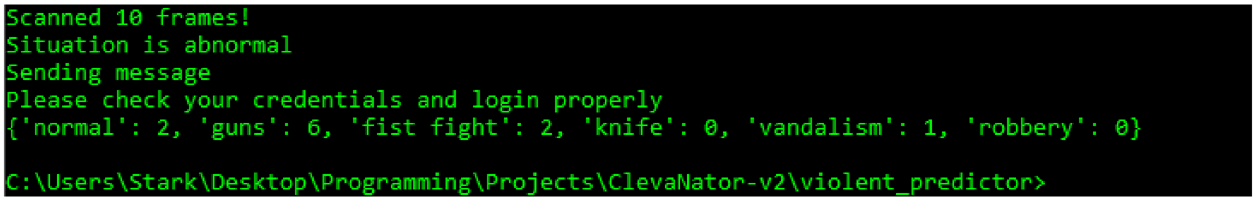
iii. Execute the command

python generate_frames_from_the_video.py video_filename

iv. This will generate the frames in the folder *test*

v. Now that you have the frames, execute the command

python -m tensorflow-for-poets-2.scripts.label_image
--graph=tf_files/retrained_graph.pb
--image=path to /ClevaNator-v2/violent_predictor/TestData/test/



```
Scanned 10 frames!
Situation is abnormal
Sending message
Please check your credentials and login properly
{'normal': 2, 'guns': 6, 'fist fight': 2, 'knife': 0, 'vandalism': 1, 'robbery': 0}

C:\Users\Stark\Desktop\Programming\Projects\ClevaNator-v2\violent_predictor>
```

Figure 6: Expected output for one of the test cases.

In the above example the predictor specifies the different classifications of images and since the count of 'normal' label is far less than the other labels, it considers the situation as abnormal and sends an alert message. (The error message is eliminated by placing your account credentials in label_image.py under the send_message())

9. CONCLUSION

The MobileNet architecture I've used has worked out pretty well for my application, which needs very less latency in prediction. Generating the frames in two different ways has helped me test the model on variety of data. I have observed that the model works pretty good and as per the validation accuracy the prediction accuracy is around 69-75%. The project that I've worked on has a huge potential and could possibly save many lives.

10. FUTURE WORK

I have developed a project which can automatically analyze frames from the given input and detect any abnormalities, but it is still limited in its computation power. And the most important thing is that this application has to work in Real-Time, I have planned on the next iteration of this project by implementing an Object Detection Model. This model can detect objects in Real-Time; using this model can improve my project's efficiency and also its accuracy, as everything happens in Real-Time. There are many state-of-the-art models like Yolo and Google's Object Detection API which suit my project, hence that would be my next milestone.

11. REFERENCES

[1] Http://machinethink.net/blog/googles-mobile-net-architecture-on-iphone/

[2] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Effi- cient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861, 2017. 1, 2, 3, 5, 6, 7

[3] Https://codelabs.developers.google.com/codelabs/tensorflow-for-poets/#0

[4] https://research.googleblog.com/2017/06/mobilenets-open-source-models-for.html