Abstract

The issue of mass surveillance in CCTV camera feed is very important. Surveillance can be of different forms like malicious activity detection, identification of a particular entity particular individual in a CCTV video) or in general keeping tracks of movements of human beings. In our project, the focus has been given to find the trajectory/path of human through the grid of CCTV cameras also known as tracking. Also, manually doing tracking can be very difficult and therefore we present to you our AI based solution that is capable to do this on its own. This is done with the help of face recognition plus video processing.

Current system in this field aims to search for an entity in video by extracting its face and matching (or running) it against a database of human faces that is in the interest. So, none of the systems solve the task if they do not have a predefined database against whom the matching is done. Our, Smart AI will do this in a smart way by first generating datasets from human faces taken from CCTV video and use it in a Face Recognition model we are using.

The use of deep learning libraries like Keras along with some image processing tools like openCV with a cloud based solution is done to achieve this task.

Keywords: Automated tracking, Convolutional-Neural-Network, face recognition.

Table of contents

List of Figures

1. Introduction

A *Grid OF CCTVs* installed in an area can give information about human activities happening in the area. The existing network of Government and public sector-installed Closed Circuit Television Cameras can be virtually networked to create a system that will monitor key public arenas, financial institutions, tourism sites and key infrastructure.

However, installing CCTV cameras and forgetting them until something happens, gives us a false sense of security. The truth is most CCTVs cameras go unmonitored and unmaintained. Some even stop functioning and we find out this only when there is a manual intervention to try to access some footage.

No Number of Human Being *will be able to monitor the terabytes* of information being generated on a second-to-second basis. Eyeballing petabytes of video is simply impossible for the eyeballs we employ in police. It also requires lots of patience.

Manual video tracking(CCTV video analysis) mostly relies on teams of specially trained officers watching thousands of hours of footage, waiting for that one crucial second of evidence. But this is the problem we are trying to solve.

Now, Advanced Computer Vision technology can monitor the footage (if not disapproved by law) for movement. We just need to bring these cameras online, and link them up in an encrypted grid using advanced Artificial Intelligence and Machine Learning technologies. The data from these live streams can be processed and analyzed in near-real time to produce a wealth of information, which will be a significant input when it comes to the topic of surveillance.

2. Proposed System

We propose an system of **“Mass Automated Tracking**”.

Mass

The area covered by the grid of CCTV cameras is very large. For example, a shopping Mall with a network of CCTV cameras, or a school/college/university or even to a small to mid-size town, etc. As number of cameras grows, the data generated also grows rapidly and hence there is a need of a smarter AI solution.

Automatic

The task of going through the terabytes of data generated by the Grid of CCTV cameras by an individual or a group of individuals is difficult. Also, for tasks like *tracking human activities from CCTV video feed* benefits greatly from an automated system, where records of tracking information will be generated by the AI system and given to the database, which can be used later. No manual input is needed other then Set up of the hardware components, and deploying the software.

Tracking

It is special form of surveillance where trajectory of person in an area under the CCTV view can be monitored and used for various tasks ranging from business analytics to security. For example in a Shopping Mall area, it will be beneficial to know that person *‘A’* moves from Position *‘X’* to Position *‘Y’* and at any instance of time. Therefore, our smart AI system which should automatically recognize human faces should be able to track it.

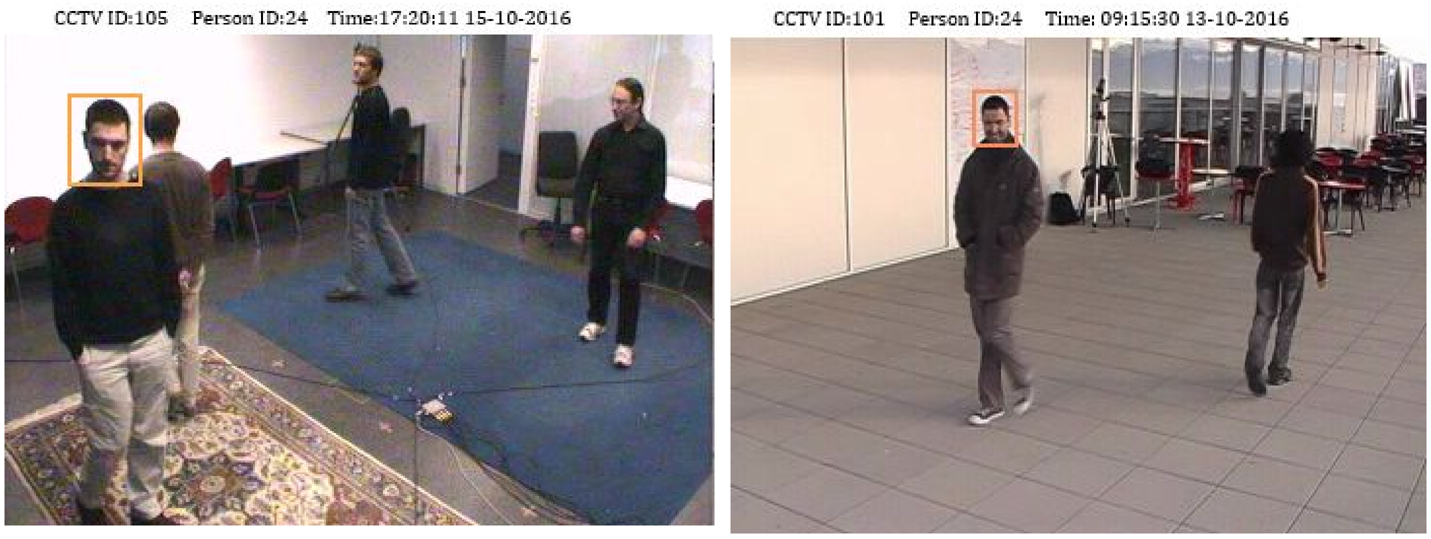
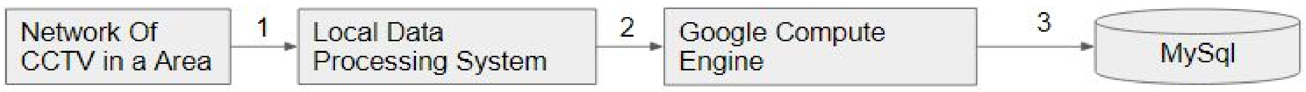

CCTV ID:105 Person ID:24 Time:17:20:11 15-10-2016
CCTV ID:101 Person ID:24 Time: 09:15:30 13-10-2016

Figure 2.1 Example of Tracking. Two cameras with Id 105 and 101 detect the same person with ID 25.

3. System Description



3.1. Grid of CCTV cameras

A collection of CCTV cameras connected to a local processing system that sends video Feed to the processing system.

3.2. Local data processing system

An optimal local processing system is a high performance computer that receives video feed from "Grid of CCTV cameras", process them and groups similar faces together.

The algorithm is discussed in section 4.

The hardware requirements of the system would be:

1. Minimum 32GB of Ram to handle of grid size of minimum 30 cameras.
2. Minimum 8 core CPU.

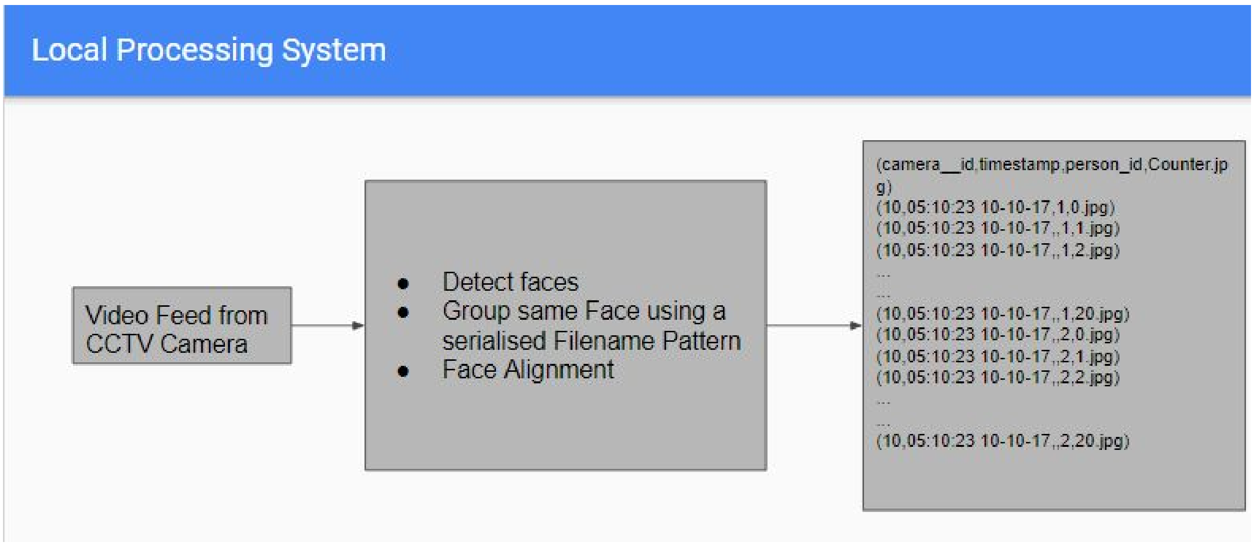

Local Processing System
Video Feed from CCTV Camera
Detect faces
Group same Face using a serialised Filename Pattern
Face Alignment
(camera__id,timestamp,person_id,Counter.jpg)
(10,05:10:23 10-10-17,1,0.jpg)
(10,05:10:23 10-10-17,,1,1.jpg)
(10,05:10:23 10-10-17,,1,2.jpg)
...
...
(10,05:10:23 10-10-17,,1,20.jpg)
(10,05:10:23 10-10-17,,2,0.jpg)
(10,05:10:23 10-10-17,,2,1.jpg)
(10,05:10:23 10-10-17,,2,2.jpg)
...
...
(10,05:10:23 10-10-17,,2,20.jpg)

Figure 3.1 Block diagram of Local Processing System

3.3. A Centralized server

Our smart AI that tracks the people has to recognize faces and assigns them a serial id. For example, in figure 2.1: serial id of 25 was assigned to the person. It is very important to note that we are not interested in the actual label (i.e, the name and details of the persons). We are just interested in assigning an serial id to that face which will be used later as the label for our training [section 2.6]. System such as Find Face by created by Ntech Labs ltd gets details about the detected faces by running them against Russian Social media site VK.com.

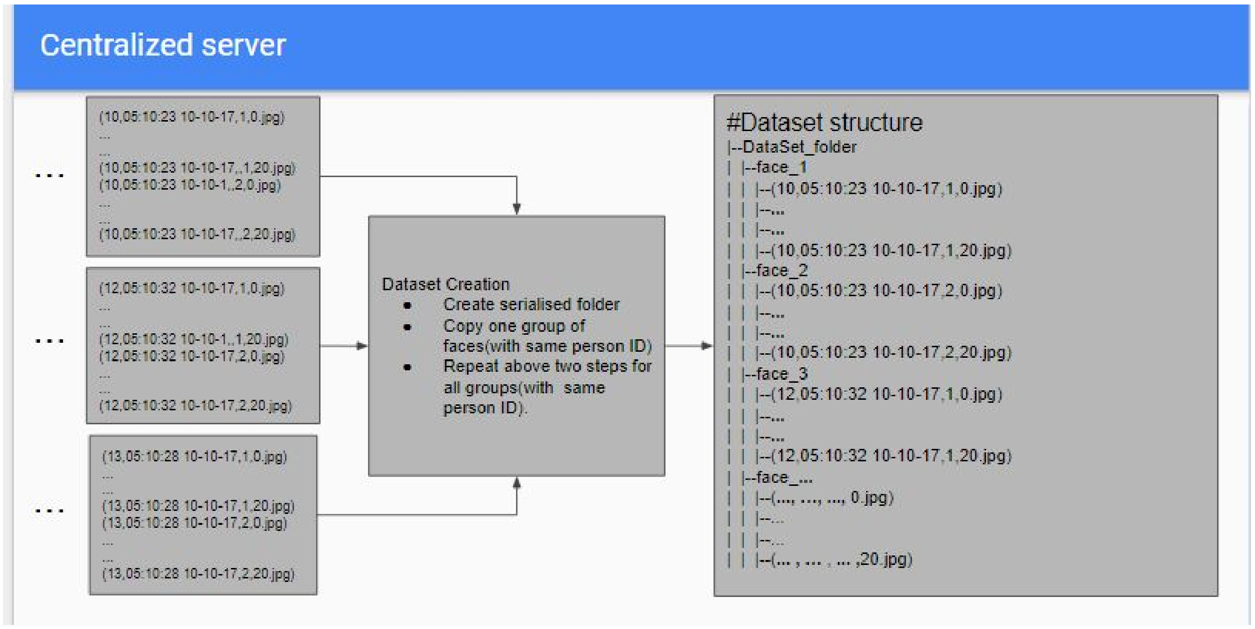

Centralized server
(10,05:10:23 10-10-17,1,0.jpg)
(10,05:10:23 10-10-17,,1,20.jpg)
(10,05:10:23 10-10-1,,2,0.jpg)
(10,05:10:23 10-10-17,,2,20.jpg)
(12,05:10:32 10-10-17,1,0.jpg)
(12,05:10:32 10-10-1,,1,20.jpg)
(12,05:10:32 10-10-17,2,0.jpg)
(12,05:10:32 10-10-17,2,20.jpg)
(13,05:10:28 10-10-17,1,0.jpg)
(13,05:10:28 10-10-17,1,20.jpg)
(13,05:10:28 10-10-17,2,0.jpg)
(13,05:10:28 10-10-17,2,20.jpg)
Dataset Creation
Create serialised folder
Copy one group of faces(with same person ID)
Repeat above two steps for all groups(with same person ID).
#Dataset structure
|--DataSet_folder
| |--face_1
| | |--(10,05:10:23 10-10-17,1,0.jpg)
| | |--(10,05:10:23 10-10-17,1,20.jpg)
| |--face_2
| | |--(10,05:10:23 10-10-17,2,0.jpg)
| | |--(10,05:10:23 10-10-17,2,20.jpg)
| |--face_3
| | |--(12,05:10:32 10-10-17,1,0.jpg)
| | |--(12,05:10:32 10-10-17,1,20.jpg)
| |--face_...
| | |--(..., ..., ..., 0.jpg)
| | |--(... , ... , ... ,20.jpg)

Figure 3.2 Block diagram of Centralized Server

The components inside the centralized server are:

3.3.1. A compute engine

A compute engine well capable supporting Deep Learning.The main task is to take a dataset and use it on the face recognition model [section x.y].

As Deep Learning is very expensive operation, and our problems requires real time performance, the use of distributed systems such as Hadoop with Spark is favored. Also, using a distributed neural network library like BIgDL is preferred [section 8].

3.3.2. Database

A database stores the faces of the images which will be later in our model. This database can grow to be very large for mass Surveillance.

The system uses MySql in our system but other Non-relational database can also be used.

4. Methodology

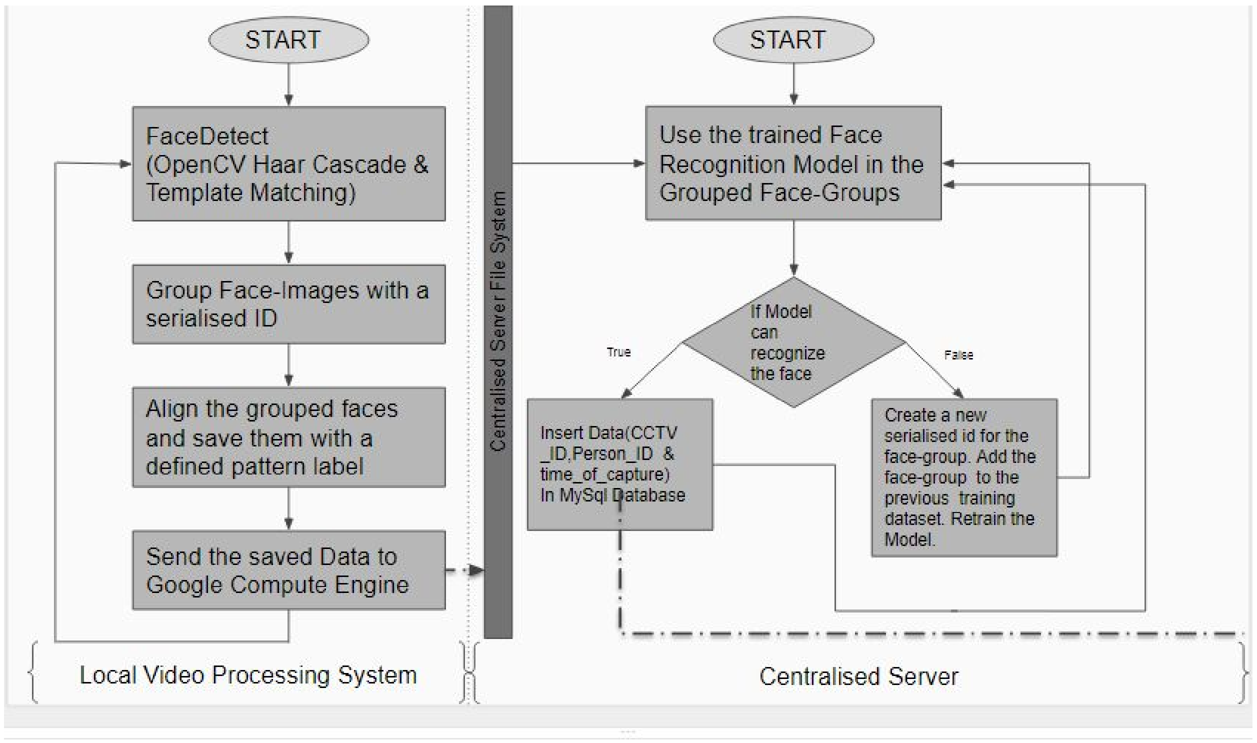

START
FaceDetect (OpenCV Haar Cascade & Template Matching)
Group Face-Images with a serialised ID
Align the grouped faces and save them with a defined pattern label
Send the saved Data to Google Compute Engine
Local Video Processing System
Centralised Server File System
START
Use the trained Face Recognition Model in the Grouped Face-Groups
If Model can recognize the face
True
False
Insert Data(CCTV _ID,Person_ID & time_of_capture) In MySql Database
Create a new serialised id for the face-group. Add the face-group to the previous training dataset. Retrain the Model.
Centralised Server

Figure 4.1: Flow Diagram of Whole System

4.1. Local Video processing system

The local video processing unit is mainly responsible for generating grouped images that is send to the server.

Steps involved:

4.1.1 .Faces detection in video

Detecting faces with high accuracy is a difficult task due to orientation and angle of the faces in the video. However, high accuracy systems such as DLIB provides high accuracy by using HOG features form images. But, as our system is based on real concept as video processing is continuous we have used Haar Cascade for our task which is a machine learning based approach where a cascade function is trained from a lot of positive and negative images. It is then used to detect objects in other images.

4.1.2 Haar Cascade

Haar-cascade is an object detection algorithm used to locate faces, pedestrians, objects and facial expressions in an image and is mainly being used for face detection. In Haar-cascade, the system is provided with several numbers of positive images (like faces of different persons at different backgrounds) and negative images (images that are not faces but can be anything else like chair, table, wall, etc.), and the feature selection is done along with the classifier training using Adaboost and Integral images.

The algorithm needs a lot of positive images (images of faces) and negative images (images without faces) to train the classifier. The algorithm then need to extract features from it by using haar features haar features shown in below image are used. They are just like our convolutional kernel. Each feature is a single value obtained by subtracting sum of pixels under white rectangle from sum of pixels under black rectangle.

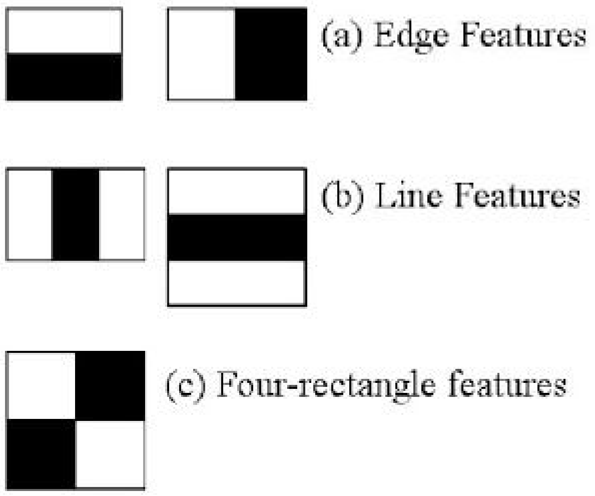


Figure 4.2: Convolution kernels in haar features.

OpenCV provides methods for performing Haar Cascade on a

```
face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')

eye_cascade = cv2.CascadeClassifier('haarcascade_eye.xml')
```

where haarcascade_frontalface_default.xml and haarcascade_eye.xml features files written by researchers.

4.1.3. Grouping images using Template matching

To gives an unique id to every new face in a video feed, we match it against all the other faces detected before it in a definite window. The matching is done by using Template matching where the current image is the source image and the target image is the faces preceding it.

Also, as this procedure is done only for one minute, after which the entire process is reset, therefore the time taken is not very high. The size of the definite window is 20, as most standard CCTV give approximately 20 FPS.

Template Matching

Template Matching is a method for searching and finding the location of a template image in a larger image. OpenCV comes with a function cv2.matchTemplate() for this purpose. It simply slides the template image over the input image (as in 2D convolution) and compares the template and patch of input image under the template image. Several comparison methods are implemented in OpenCV. It returns a grayscale image, where each pixel denotes how much does the neighbourhood of that pixel match with template.

4.2. Align the group faces and save them

We have to preprocess the image, so that they have optimal vertical alignment. Every faces has some description along with it such :

CCTV number	From which CCTV the image has come.
Timestamp	Time at which the image was taken
Group ID	the unique id given to all the different faces in the span of one minute
Counter	Order number of an image in a group

The image is saved in the local directory of the of the local processing system.

4.3. Sent the saved data to the server (Google compute engine)

Here we upload the dataset of detected faces that we have collected which is grouped with unique identifier to the server where it is used by our model. It is important to note that the local directory is cleared once we have uploaded to the server.

The workflow diagram is

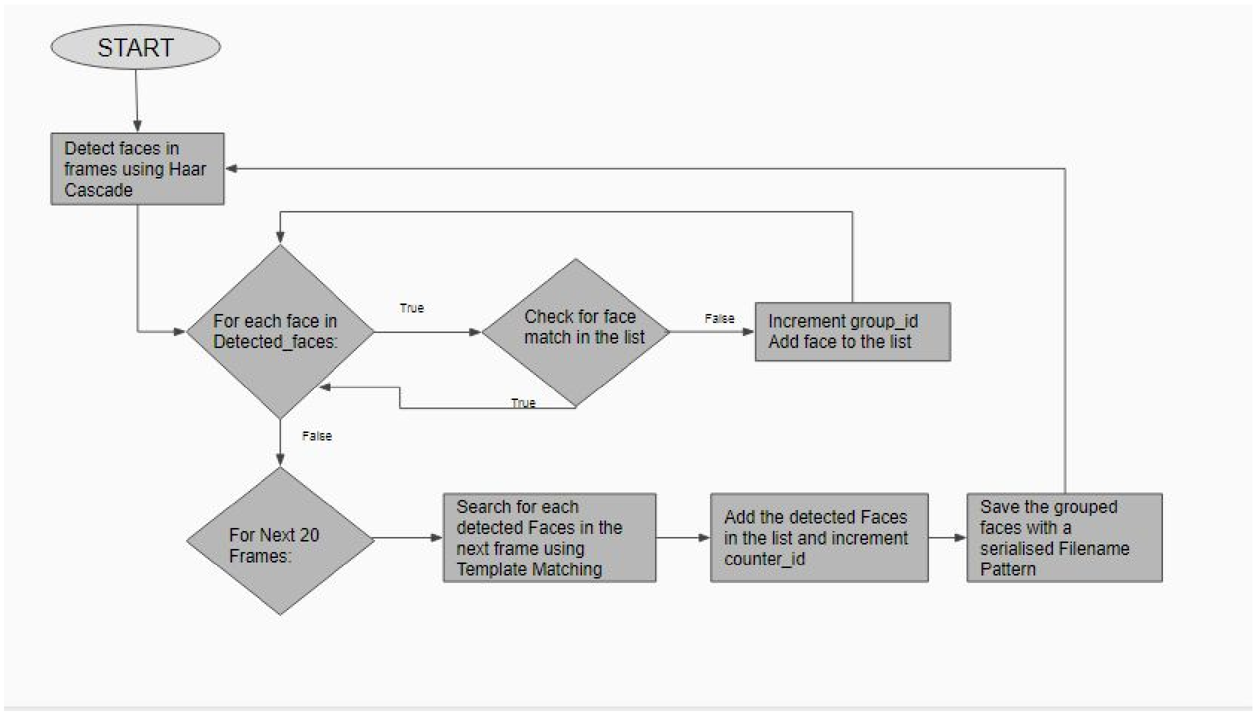

START
Detect faces in frames using Haar Cascade
For each face in Detected_faces:
True
Check for face match in the list
False
Increment group_id Add face to the list
True
False
For Next 20 Frames:
Search for each detected Faces in the next frame using Template Matching
Add the detected Faces in the list and increment counter_id
Save the grouped faces with a serialised Filename Pattern

Figure 4.3: Flow diagram in the local processing system

5. Centralised Server

5.1 Face recognition

We need a mechanism to map an incoming face to to a correct label that our system can identify.! This kind of mapping is difficult because the faces detected of a person later may be very different than the faces our system already knows of that person.

One of the ways to do this is by comparing selected facial features from the image and a compare with the faces our system already knows. Some face recognition algorithms identify facial features by extracting landmarks, or features, from an image of the subject's face. For example, an algorithm may analyze the relative position, size, and/or shape of the eyes, nose, cheekbones, and jaw. These features are then used to search for other images with matching feature.

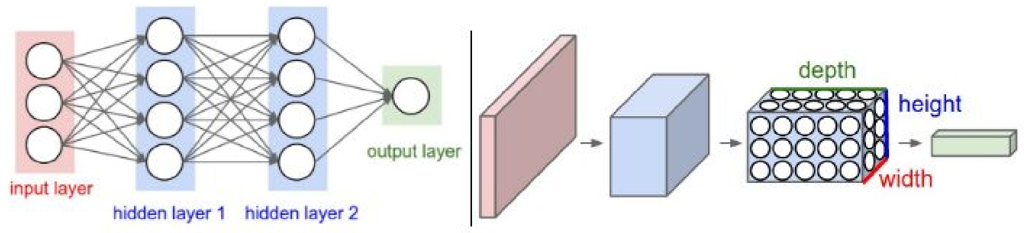
The state of the art technology for Face recognition is Convolutional Neural Networks.

5.1.1. Convolutional Neural Network

Convolutional Neural Networks are made up of neurons that have learnable weights and biases. Each neuron receives some inputs, performs a dot product and optionally follows it with a non-linearity. The whole network still expresses a single differentiable score function: from the raw image pixels on one end to class scores at the other. And they still have a loss function (e.g. SVM/Softmax) on the last (fully-connected) layer and all network learning phase is same as regular Neural Networks still apply.

Regular Neural Networks receive an input (a single vector), and transform it through a series of *hidden layers*. Each hidden layer is made up of a set of neurons, where each neuron is fully connected to all neurons in the previous layer, and where neurons in a single layer function completely independently and do not share any connections. The last fully-connected layer is called the "output layer" and in classification settings it represents the class scores.

Convolutional Neural Networks(ConvNet) take advantage of the fact that the input consists of images and they constrain the architecture in a more sensible way. In particular, unlike a regular Neural Network, the layers of a ConvNet have neurons arranged in 3 dimensions: width, height, depth. (Here the word *depth* here refers to the third dimension of an activation volume, not to the depth of a full Neural Network, which can refer to the total number of layers in a network.)



Left: A regular 3-layer Neural Network. Right: A ConvNet arranges its neurons in three dimensions (width, height, depth), as visualized in one of the layers. Every layer of a ConvNet transforms the 3D input volume to a 3D output volume of neuron activations. In this example, the red input layer holds the image, so its width and height would be the dimensions of the image, and the depth would be 3 (Red, Green, Blue channels).

Figure 4. 4: Comparison of Regular Neural Networks and Deep Neural Networks.

5.2. Face recognition Model Used

VGG-Face

The model is a part of the Deep face recognition project by M. Parkhi Omkar, Vedaldi Andrea and Zisserman Andrew at Visual Geometry Group Department of Engineering Science University of Oxford.

They have collected a very large scale dataset of 2.6M images, over 2.6K people for model training. The dataset collected were celebrities and public figures, such as actors or politicians, so that a sufficient number of distinct images are likely to be found on the web, and also to avoid any privacy issue in downloading their images. The images were downloaded from Internet Movie Database(IMDB) celebrity list and Freebase knowledge graph, which has information on about 500K different identities, resulting in a ranked lists of 2.5K males and 2.5K females. A candidate list of 5K names which are known to be popular (from IMDB).

The VGG-face mode is "very deep", in the sense that they comprise a long sequence of convolutional layers. Such CNNs have recently achieved state-of-the-art performance in some of the tasks of the ImageNet ILSVRC 2014 challenge.

The model architecture is

VGG-Face model details:

```
Layer (type)              Output Shape            Param #
=================================================================
permute_1 (Permute)        (None, 224, 224, 3)      0
_________________________________________________________________
conv1_1 (Conv2D)           (None, 224, 224, 64)      1792
_________________________________________________________________
conv1_2 (Conv2D)           (None, 224, 224, 64)      36928
_________________________________________________________________
max_pooling2d_1           (None, 112, 112, 64)      0
_________________________________________________________________
conv2_1 (Conv2D)           (None, 112, 112, 128)   73856
_________________________________________________________________
conv2_2 (Conv2D)           (None, 112, 112, 128)   147584
_________________________________________________________________
max_pooling2d_2           (None, 56, 56, 128)      0
_________________________________________________________________
conv3_1 (Conv2D)           (None, 56, 56, 256)      295168
_________________________________________________________________
conv3_2 (Conv2D)           (None, 56, 56, 256)      590080
_________________________________________________________________
conv3_3 (Conv2D)           (None, 56, 56, 256)      590080
_________________________________________________________________
max_pooling2d_3          (None, 28, 28, 256)      0
_________________________________________________________________
conv4_1 (Conv2D)           (None, 28, 28, 512)      1180160
_________________________________________________________________
conv4_2 (Conv2D)           (None, 28, 28, 512)      2359808
_________________________________________________________________
conv4_3 (Conv2D)           (None, 28, 28, 512)      2359808
_________________________________________________________________
max_pooling2d_4          (None, 14, 14, 512)      0
_________________________________________________________________
conv5_1 (Conv2D)           (None, 14, 14, 512)      2359808
_________________________________________________________________
conv5_2 (Conv2D)           (None, 14, 14, 512)      2359808
_________________________________________________________________
conv5_3 (Conv2D)           (None, 14, 14, 512)      2359808
_________________________________________________________________
max_pooling2d_5           (None, 7, 7, 512)         0
_________________________________________________________________
fc6 (Conv2D)              (None, 1, 1, 4096)        102764544
```

```
________________________________________________________________
dropout_1 (Dropout)       (None, 1, 1, 4096)        0
________________________________________________________________
fc7 (Conv2D)              (None, 1, 1, 4096)        16781312
________________________________________________________________
dropout_2 (Dropout)       (None, 1, 1, 4096)        0
________________________________________________________________
fc8 (Conv2D)              (None, 1, 1, 2622)        10742334
________________________________________________________________

flatten_1 (Flatten)       (None, 2622)              0

________________________________________________________________

activation_1 (Activation)    (None, 2622)           0
=================================================================
```

Total params: 145,002,878

Trainable params: 145,002,878

Figure 5.1: Model Summary of original VGG-Face

The model is evaluated on the Labeled Faces in the Wild and the YouTube Faces dataset.

As we can see that the number of parameters learned is very large and Training the ConvNet from scratch with less data is not real time and may lead to overfitting. Our Tracking application requires an more real time approach to this and thus it uses the model but only trains a part of the model.

5.3 Transfer Learning

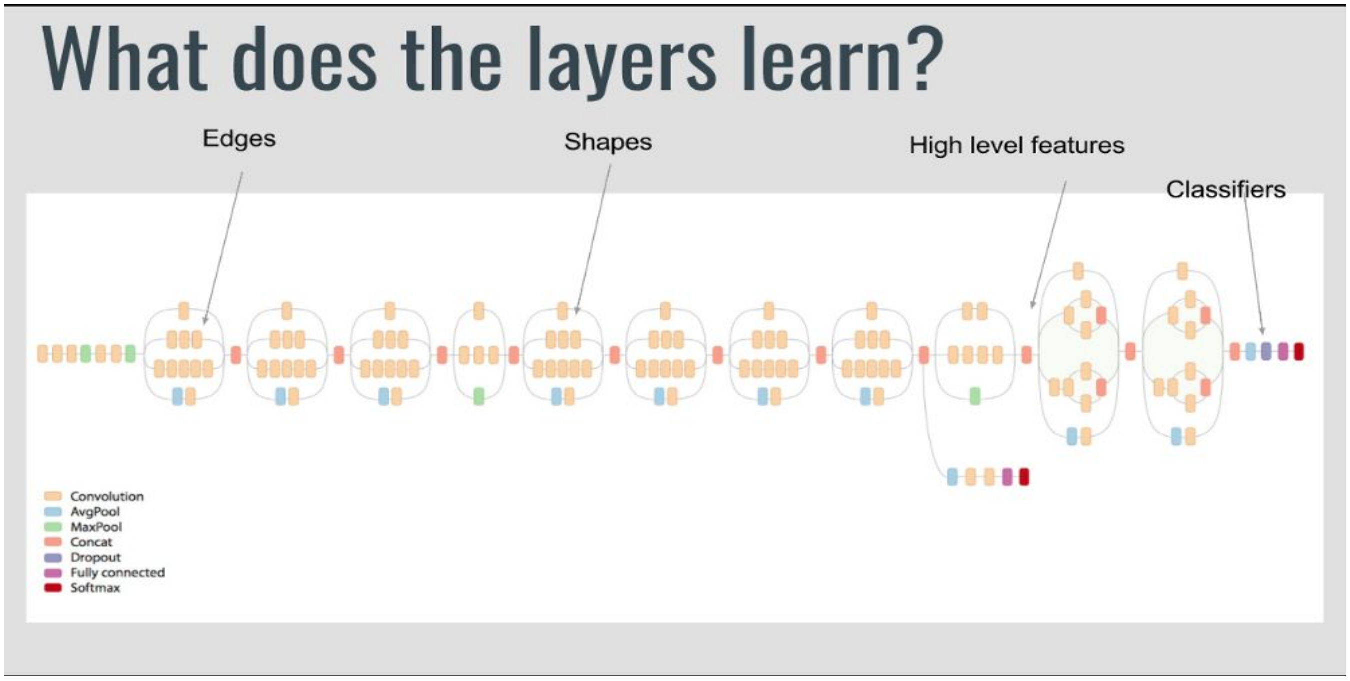


Figure 5.2 : Inception V3 Google Research

- In a trained ConvNet, the earlier features of the layers contain more generic features (e.g. edge detectors or color blob detectors), but later layers of the ConvNet becomes progressively more specific to the details of the classes contained in the original dataset.
- In practice, very few people train an entire Convolutional Network from scratch (with random initialization), because it is relatively rare to have a dataset of sufficient size. Instead, it is common to pre-train a ConvNet on a very large dataset (e.g. ImageNet, which contains 1.2 million images with 1000 categories), and then use the ConvNet either as an initialization or a fixed feature extractor for the task of interest. Since modern ConvNets take 2-3 weeks to train across multiple GPUs on ImageNet, it is common to see people release their final ConvNet checkpoints for the benefit of others who can use the networks for fine-tuning. For example, the Caffe library has Model Zoo where people share their network weights.

The major Transfer Learning scenarios look as follows:

- ConvNet as fixed feature extractor. We can take the ConvNet pre-trained on ImageNet or other deep neural Network, remove the last fully-connected layer, then treat the rest of the ConvNet as a fixed feature extractor for the new dataset. For example an AlexNet [2, this would compute a 4096-D vector for every image that contains the activations of the hidden layer immediately before the classifier. We call these features CNN codes. Once we extract the 4096-D codes for all images, we train a linear classifier (e.g. Linear SVM or Softmax classifier) for the new dataset.

- Fine-tuning the ConvNet. The second strategy is to not only replace and retrain the classifier on top of the ConvNet on the new dataset, but to also fine-tune the weights of the pre-trained network by continuing the backpropagation. It is possible to fine-tune all the layers of the ConvNet, or it's possible to keep some of the earlier layers fixed (due to overfitting concerns) and only fine-tune some higher-level portion of the network. This is motivated by the observation that the earlier features of a ConvNet contain more generic features (e.g. edge detectors or color blob detectors) that should be useful to many tasks, but later layers of the ConvNet becomes progressively more specific to the details of the classes contained in the original dataset.

When we have new dataset to fine tune the pre-trained CNN. almost similar to the original dataset used for pre-training and size new dataset is similar, the same weights can be used for extracting the features from the new dataset.

We have train only the final layers of the network to avoid overfitting, keeping all other layers fixed. So we removed the final layers of the pre-trained network and we Add new layers. Then we Retrain only the new layers.

```
Layer (type)             Output Shape            Param #
==================================================
permute_1 (Permute)       (None, 224, 224, 3)      0
________________________________________________________
conv1_1 (Conv2D)          (None, 224, 224, 64)     1792
________________________________________________________
conv1_2 (Conv2D)          (None, 224, 224, 64)     36928
________________________________________________________
max_pooling2d_1          (None, 112, 112, 64)     0
________________________________________________________
conv2_1 (Conv2D)          (None, 112, 112, 128)    73856
________________________________________________________
conv2_2 (Conv2D)          (None, 112, 112, 128)    147584
________________________________________________________
max_pooling2d_2          (None, 56, 56, 128)      0
________________________________________________________
conv3_1 (Conv2D)          (None, 56, 56, 256)      295168
________________________________________________________
conv3_2 (Conv2D)          (None, 56, 56, 256)      590080
________________________________________________________
conv3_3 (Conv2D)          (None, 56, 56, 256)      590080
________________________________________________________
```

```
max_pooling2d_3            (None, 28, 28, 256)       0
_________________________________________________________________
conv4_1 (Conv2D)           (None, 28, 28, 512)       1180160
_________________________________________________________________
conv4_2 (Conv2D)           (None, 28, 28, 512)       2359808
_________________________________________________________________
conv4_3 (Conv2D)           (None, 28, 28, 512)       2359808
_________________________________________________________________
max_pooling2d_4            (None, 14, 14, 512)       0
_________________________________________________________________
conv5_1 (Conv2D)           (None, 14, 14, 512)       2359808
_________________________________________________________________
conv5_2 (Conv2D)           (None, 14, 14, 512)       2359808
_________________________________________________________________
conv5_3 (Conv2D)           (None, 14, 14, 512)       2359808
_________________________________________________________________
max_pooling2d_5            (None, 7, 7, 512)         0
_________________________________________________________________
fc6 (Conv2D)               (None, 1, 1, 4096)        102764544
_________________________________________________________________
dropout_1 (Dropout)        (None, 1, 1, 4096)        0
_________________________________________________________________
fc7 (Conv2D)               (None, 1, 1, 4096)        16781312
_________________________________________________________________
dropout_2 (Dropout)        (None, 1, 1, 4096)        0
_________________________________________________________________
conv2d_1 (Conv2D)          (None, 1, 1, 22)          90134
_________________________________________________________________
flatten_2 (Flatten)        (None, 22)                0
_________________________________________________________________
activation_2 (Activation)  (None, 22)                0
=================================================================
Total params: 134,350,678
Trainable params: 90,134
Non-trainable params: 134,260,544
_________________________________________________________________
```

Figure 5.2: Model Summary of modified VGG-Face

6. Experimentation and Results

6.1 Technology Stack

6.1.1 Local processing technology stack

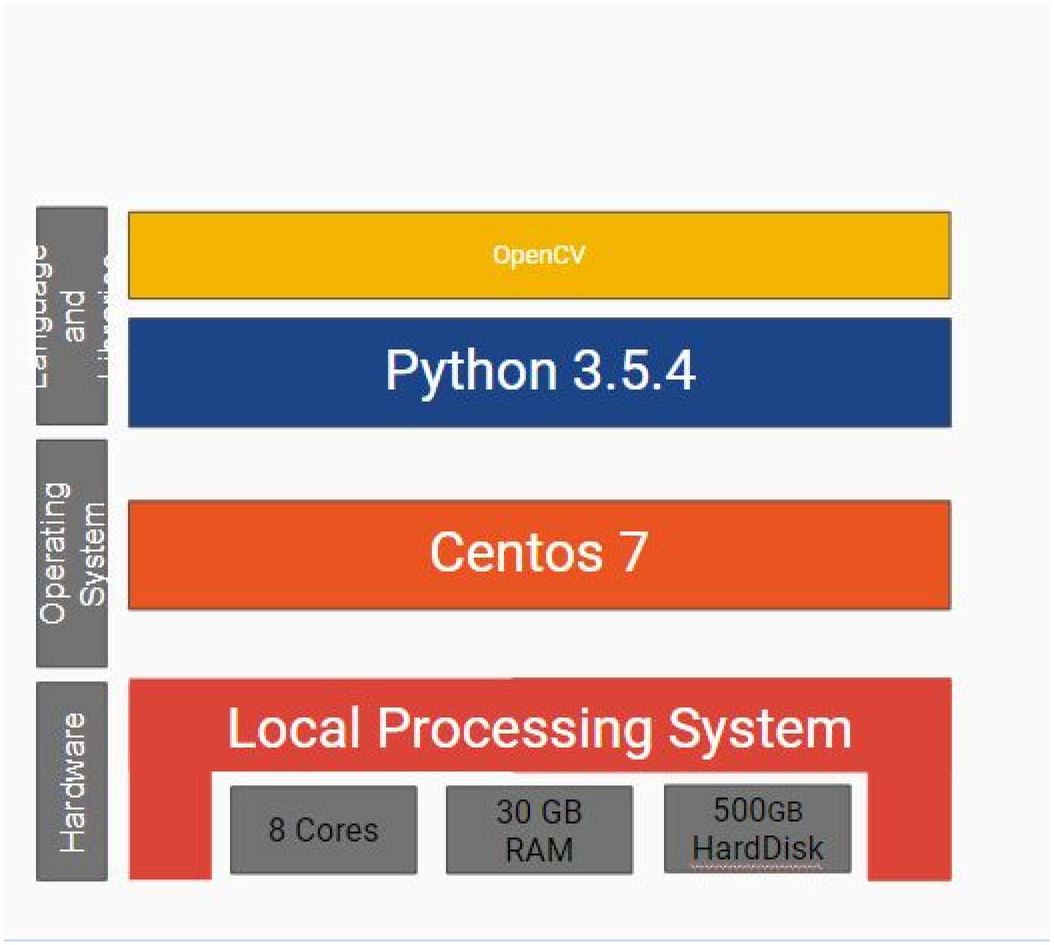


Figure 6.1: Stack Diagram of Local Processing System

6.1.2 Server side technology Stack

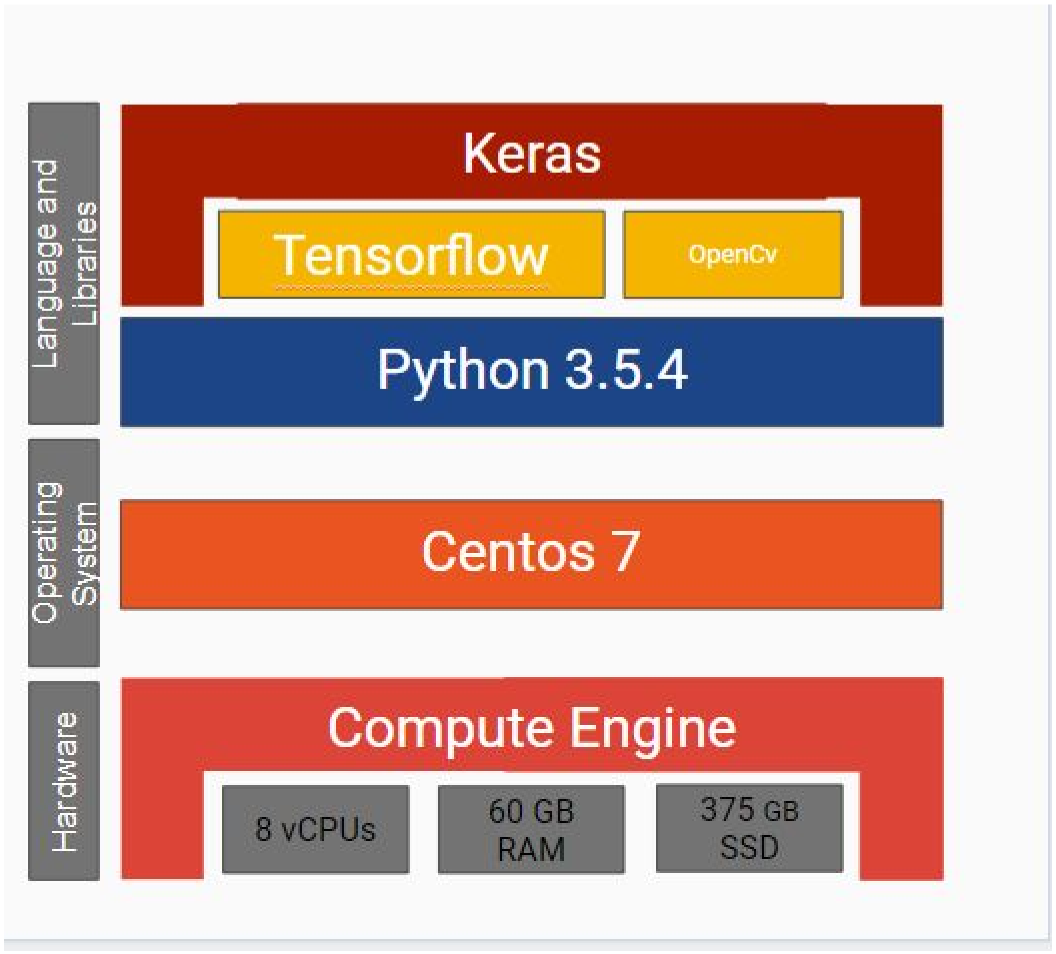


Figure 6.2: Stack Diagram of Centralized Server

7. Results:

As It can be seen that from figure 5.1 and 5.2, the number of trainale parametres have decreased form **145,002,878 to 90,134.** Therefore we have created a model that has fewer trainable parameters and thus take significantly lower time per epoch.

Also, we have developed a system that segregates the images extracted form CCTV camera feed and groups them by unique identifier based on the the different faces.

We have also created and evaluated an model that is at the heart of our system. The model VGG-face has been used for Transfer Learning. The model is giving accuracy of 62% when we used against our generated dataset, which is generated from web cams for debugging purposes. Higher level of accuracy can be achieved if we us use a HD CCTV camera such that images generated is more similar to the faces on the original model was trained on. .

8. Future Work

We have thought of a system which can can handle a large amount of training classes with huge amount of dataset while retaining the the model accuracy and training time by splitting the grouped dataset so that each model has a finite number of output classes.

We plan to deploy this technique and evaluate weather it will work in a real world system involving lot of training data.

The problem of Mass automated tracking can solved and this technique of splitting the grouped dataset can be applied here.

As we have developed an model that uses transfer learning and also developed an local processing system that continuously generates dataset, applying this new methodology will not be difficult and tracking logic can be written to build an end user application.

9. Conclusion

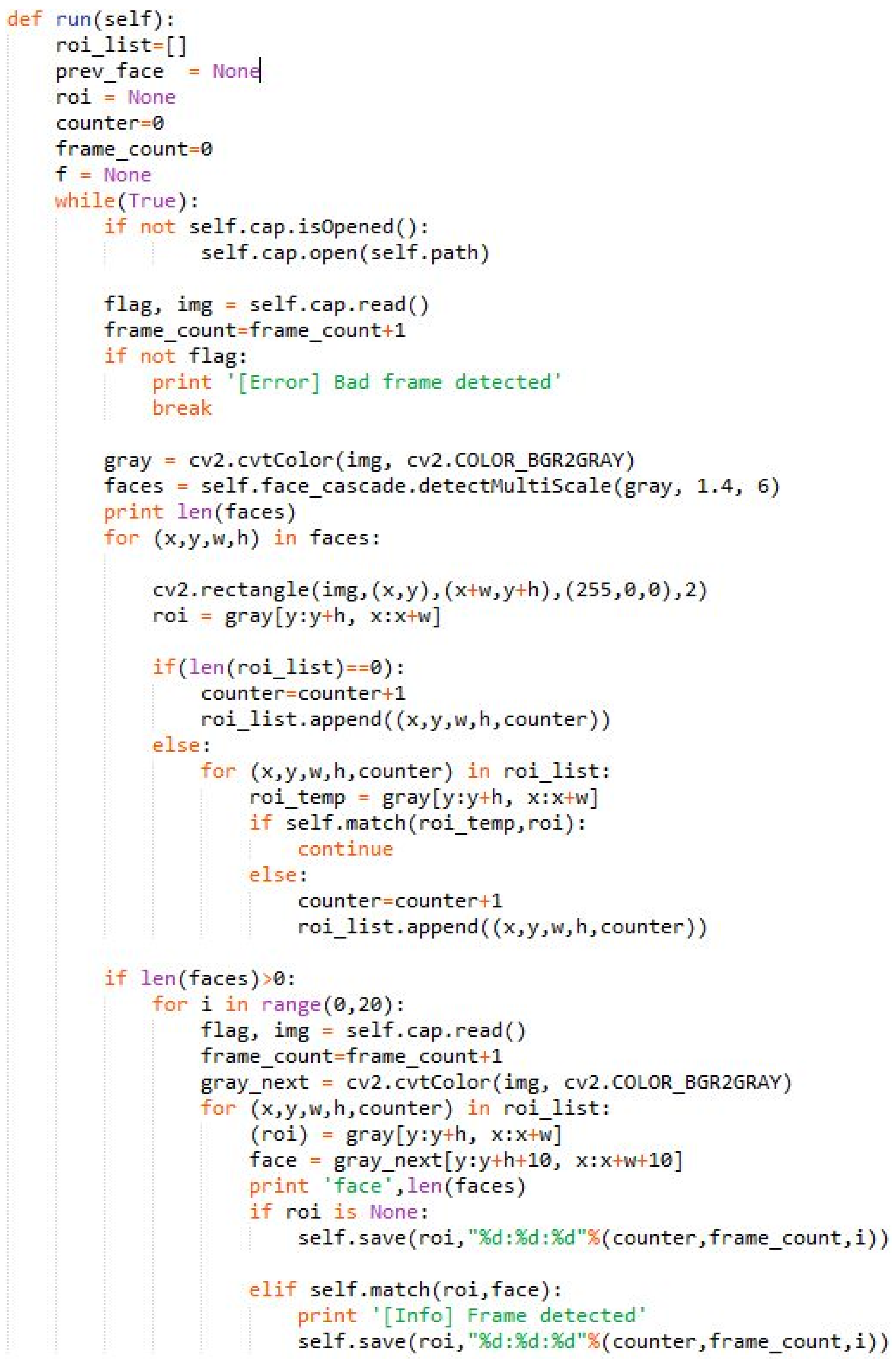
Automated mass tracking is a new tracking method and we are glad we have worked on this topic. A lot of applications can make use of such type of information like business analytics and security. Lot of work is being done on developing such applications. We have developed a base architecture that does the tracking and experimentally evaluates the concepts we are try to use in this project. The CCTV video feed is effectively processed to generate dataset of grouped faces. Used an appropriate face recognition model and modify it to use in our Automated tracking system, i.e, a highly accurate face recognition model, VGG-FACE is used for Fine Tuning.

10. Bibliography

1. https://findface.ru/, Findface
2. https://docs.opencv.org/3.3.0/d7/d8b/tutorial_py_face_detection.html , OpenCV haar Cascade
3. http://dlib.net/face_detector.py.html , Dlib Face Detection
4. https://docs.opencv.org/2.4/doc/tutorials/imgproc/histograms/template_matching/template_matching.html, Template Matching OpenCV
5. http://cs231n.github.io/convolutional-networks , CS211n Convolutional Neural Network
6. http://cs231n.github.io/transfer-learning/ , CS231n Convolutional Neural Networks for Visual Recognition
7. http://www.robots.ox.ac.uk/~vgg/software/vgg_face/ , VGG Face Descriptor
8. M.Parakhi Omkar, Vedaldi Andrea, Zisserman Andrew; PARKHI et al.: DEEP FACE RECOGNITION

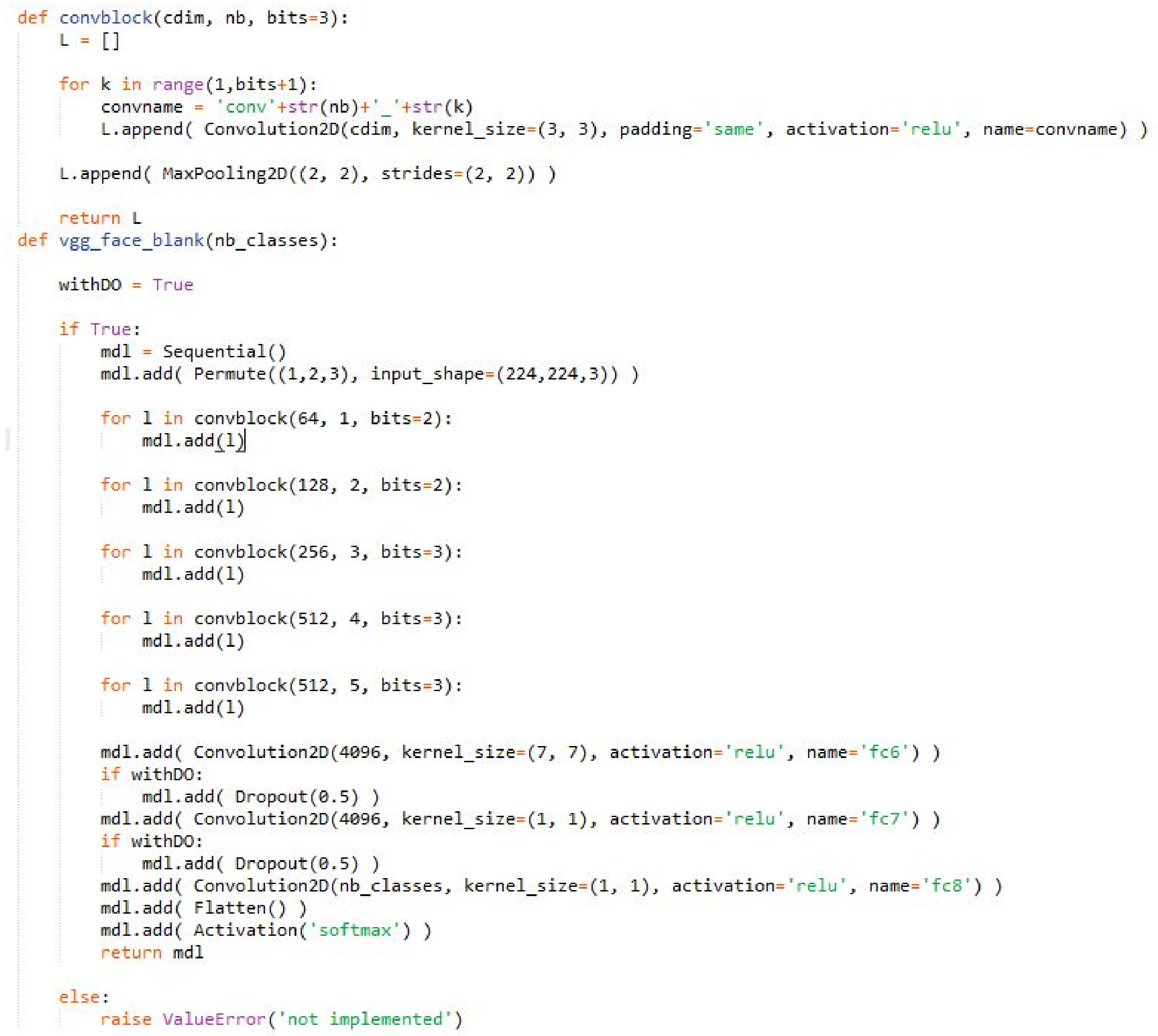
Appendix

Local Processing system Face detect and group code:



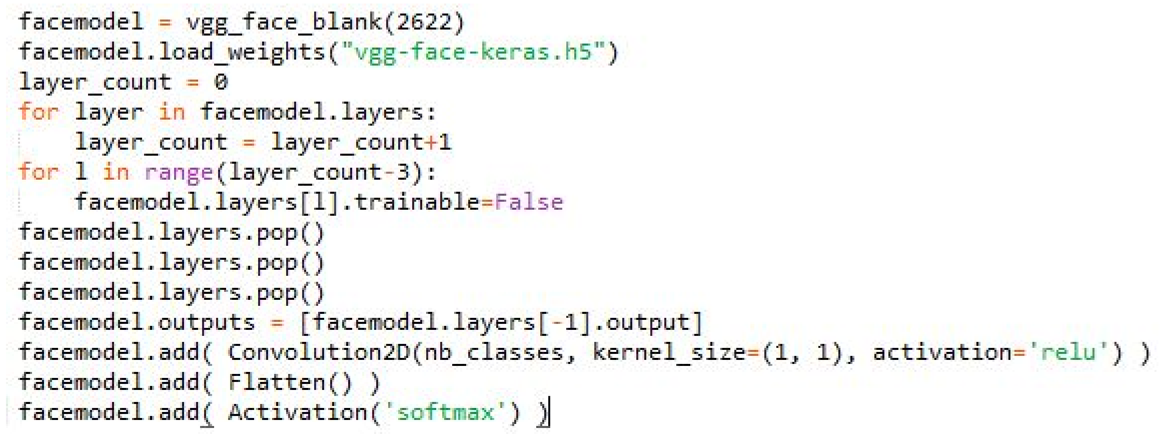
```
def run(self):
    roi_list=[]
    prev_face  = None
    roi = None
    counter=0
    frame_count=0
    f = None
    while(True):
        if not self.cap.isOpened():
                self.cap.open(self.path)

        flag, img = self.cap.read()
        frame_count=frame_count+1
        if not flag:
            print '[Error] Bad frame detected'
            break

        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        faces = self.face_cascade.detectMultiScale(gray, 1.4, 6)
        print len(faces)
        for (x,y,w,h) in faces:

            cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
            roi = gray[y:y+h, x:x+w]

            if(len(roi_list)==0):
                counter=counter+1
                roi_list.append((x,y,w,h,counter))
            else:
                for (x,y,w,h,counter) in roi_list:
                    roi_temp = gray[y:y+h, x:x+w]
                    if self.match(roi_temp,roi):
                        continue
                    else:
                        counter=counter+1
                        roi_list.append((x,y,w,h,counter))

        if len(faces)>0:
            for i in range(0,20):
                flag, img = self.cap.read()
                frame_count=frame_count+1
                gray_next = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
                for (x,y,w,h,counter) in roi_list:
                    (roi) = gray[y:y+h, x:x+w]
                    face = gray_next[y:y+h+10, x:x+w+10]
                    print 'face',len(faces)
                    if roi is None:
                        self.save(roi,"%d:%d:%d"%(counter,frame_count,i))

                    elif self.match(roi,face):
                        print '[Info] Frame detected'
                        self.save(roi,"%d:%d:%d"%(counter,frame_count,i))
```

Face Recognition Model code:



```python
def convblock(cdim, nb, bits=3):
    L = []

    for k in range(1,bits+1):
        convname = 'conv'+str(nb)+'_'+str(k)
        L.append( Convolution2D(cdim, kernel_size=(3, 3), padding='same', activation='relu', name=convname) )

    L.append( MaxPooling2D((2, 2), strides=(2, 2)) )

    return L
def vgg_face_blank(nb_classes):

    withDO = True

    if True:
        mdl = Sequential()
        mdl.add( Permute((1,2,3), input_shape=(224,224,3)) )

        for l in convblock(64, 1, bits=2):
            mdl.add(l)

        for l in convblock(128, 2, bits=2):
            mdl.add(l)

        for l in convblock(256, 3, bits=3):
            mdl.add(l)

        for l in convblock(512, 4, bits=3):
            mdl.add(l)

        for l in convblock(512, 5, bits=3):
            mdl.add(l)

        mdl.add( Convolution2D(4096, kernel_size=(7, 7), activation='relu', name='fc6') )
        if withDO:
            mdl.add( Dropout(0.5) )
        mdl.add( Convolution2D(4096, kernel_size=(1, 1), activation='relu', name='fc7') )
        if withDO:
            mdl.add( Dropout(0.5) )
        mdl.add( Convolution2D(nb_classes, kernel_size=(1, 1), activation='relu', name='fc8') )
        mdl.add( Flatten() )
        mdl.add( Activation('softmax') )
        return mdl

    else:
        raise ValueError('not implemented')
```

Transfer Learning Code:



```
facemodel = vgg_face_blank(2622)
facemodel.load_weights("vgg-face-keras.h5")
layer_count = 0
for layer in facemodel.layers:
    layer_count = layer_count+1
for l in range(layer_count-3):
    facemodel.layers[l].trainable=False
facemodel.layers.pop()
facemodel.layers.pop()
facemodel.layers.pop()
facemodel.outputs = [facemodel.layers[-1].output]
facemodel.add( Convolution2D(nb_classes, kernel_size=(1, 1), activation='relu') )
facemodel.add( Flatten() )
facemodel.add( Activation('softmax') )
```