Drunk Analysis on Thermal Images

Knowledge Transfer Document

Table of Content:

Aim of the project

Privacy and Security go hand in hand. Having a camera everywhere is a good solution to many problems but it also brings about privacy concerns. This concern can be somewhat alleviated when you replace the camera with thermal imaging sensors.

The aim of this project is to use thermal imaging in a way that it can remove the need for RGB cameras.

Thermal Images have been around for quite some time now but they still lack the functionality that is observed in RGB images.

So, in this project we are trying to detect a person and then estimate whether a person is drunk or not given a thermal image.

Dependencies

Some of the basic concepts and terminologies used throughout this project are explained here.

1. Machine Learning
2. Openpose
3. Google teachable Machines
4. Transfer learning:

Machine Learning

Machine Learning is to make a machine act on a particular problem without being explicitly programmed to do so.

To make this happen we use statistical modeling and analysis. In ML we have a few modes of learning some of which are : 1. Supervised Learning 2.Unsupervise Learning 3. Reinforced Learning

In supervised Learning we give the machine some expected results and try to make it learn based on the expected outputs.

In unsupervised Learning we give the machine some data and allow it to form clusters in the data and learn on its own without us giving an expected output. This is generally used when we ourselves cannot make sense of the data so we use the machine to make sense of it.

Supervised Learning Algorithms:

1. Linear regression for regression problems: In this we find a line that best fits the data and use it to predict the next values.
2. Random forest for classification and regression problems
3. Logistic Regression
4. Support vector machines for classification problems

Unsupervised Learning Algorithms:

1. K-means for clustering problems
2. Apriori algorithm for association rule learning problems
3. Principal Component Analysis

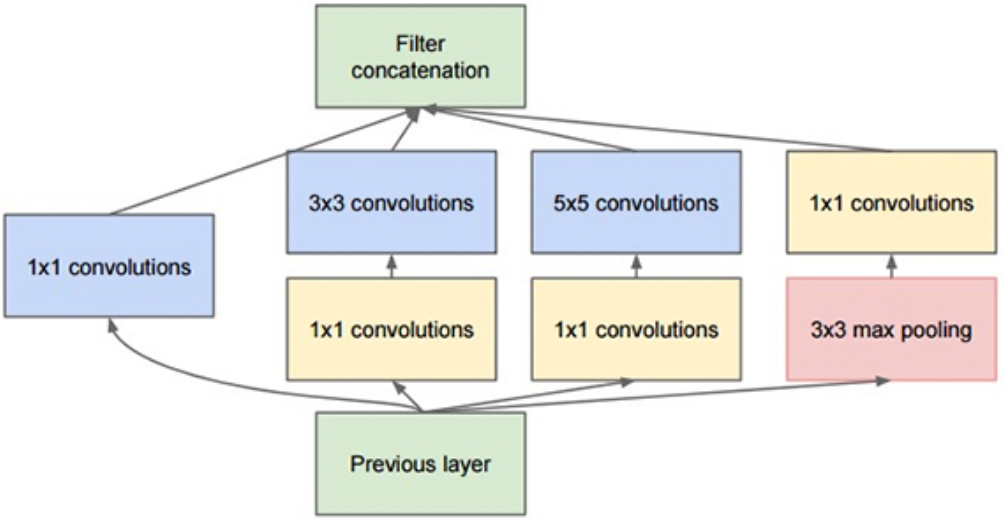
Reinforced Learning Algorithms:

1. Value-based
2. Policy-based and
3. Model based learning.

Transfer learning:

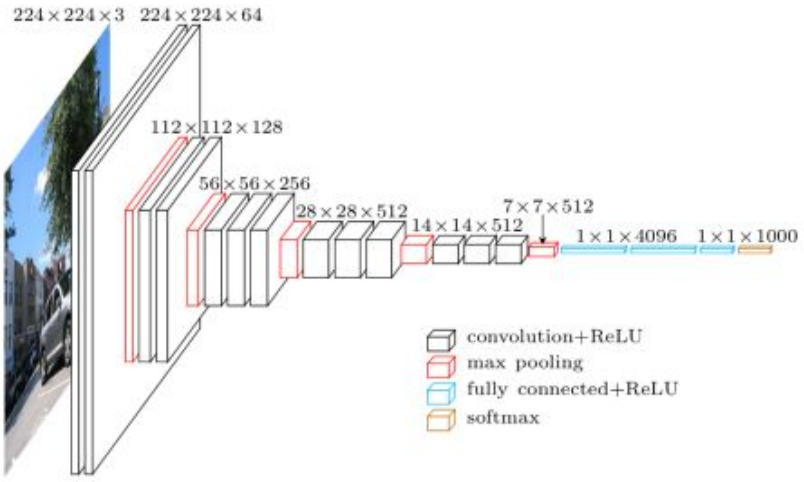
Transfer learning (TL) is a research problem in machine **learning** (ML) that focuses on storing knowledge gained while solving one problem and applying it to a different but related problem. For example, knowledge gained while **learning** to recognize cars could apply when trying to recognize trucks.
Here, we are using pre-trained models of Inception and VGG-16 from keras to predict whether a person is Drunk or Sober.

a. Inception



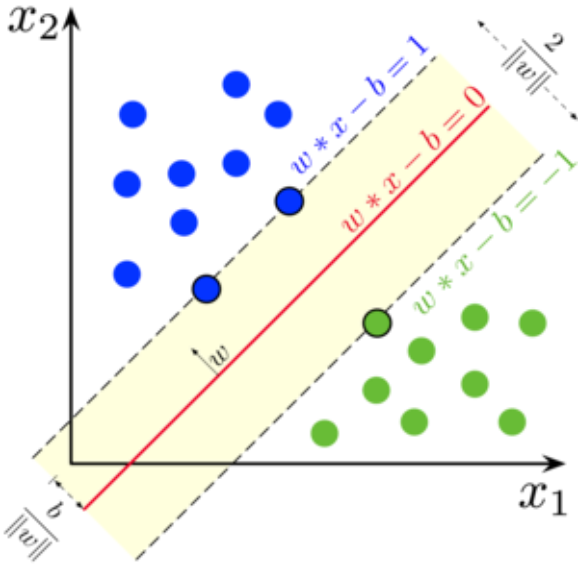
The goal of the inception module is to act as a “multi-level feature extractor” by computing *1×1*, *3×3*, and *5×5* convolutions within the *same* module of the network — the output of these filters are then stacked along the channel dimension and before being fed into the next layer in the network.

b. VGG-16



VGG16 is a convolution neural net (CNN) with a unique architecture where instead of having a large number of hyper-parameter it focuses on having convolution layers of 3x3 filter with a stride 1 and always used same padding and maxpool layer of 2x2 filter of stride 2. It follows this arrangement of convolution and max pool layers consistently throughout the whole architecture. In the end it has 2 FC(fully connected layers) followed by a softmax for output. The 16 in VGG16 refers to it has 16 layers that have weights. This network is a pretty large network and it has about 138 million (approx) parameters.

Apart from using Deep learning models for predictions, SVM models have also been used.



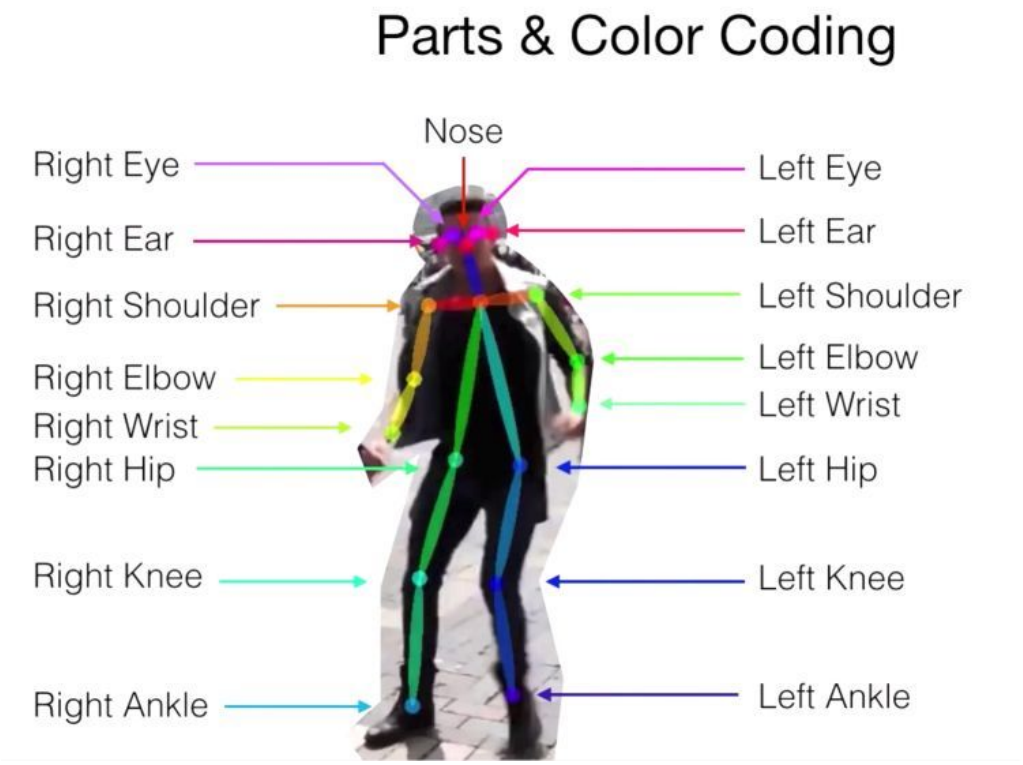
SVM is a supervised machine learning algorithm which can be used for classification or regression problems. It uses a technique called the kernel trick to transform your data and then based on these transformations it finds an optimal boundary between

the possible outputs. Simply put, it does some extremely complex data transformations, then figures out how to separate your data based on the labels or outputs you've defined.

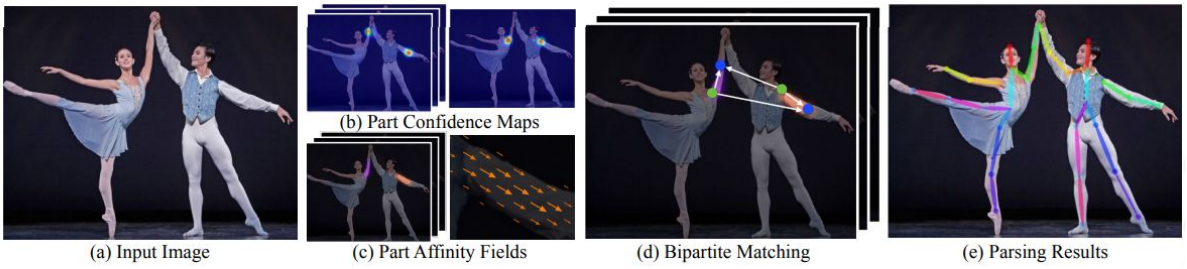
OpenPose:

Open pose is the current state of the art system for real time human pose estimation. It is also a multi-person system that gives you the position(key points) of the human body, hand, elbow, shoulder, neck, face, and feet.

Open pose was developed by researchers at Carnegie Mellon University and the code is open sourced at Github.

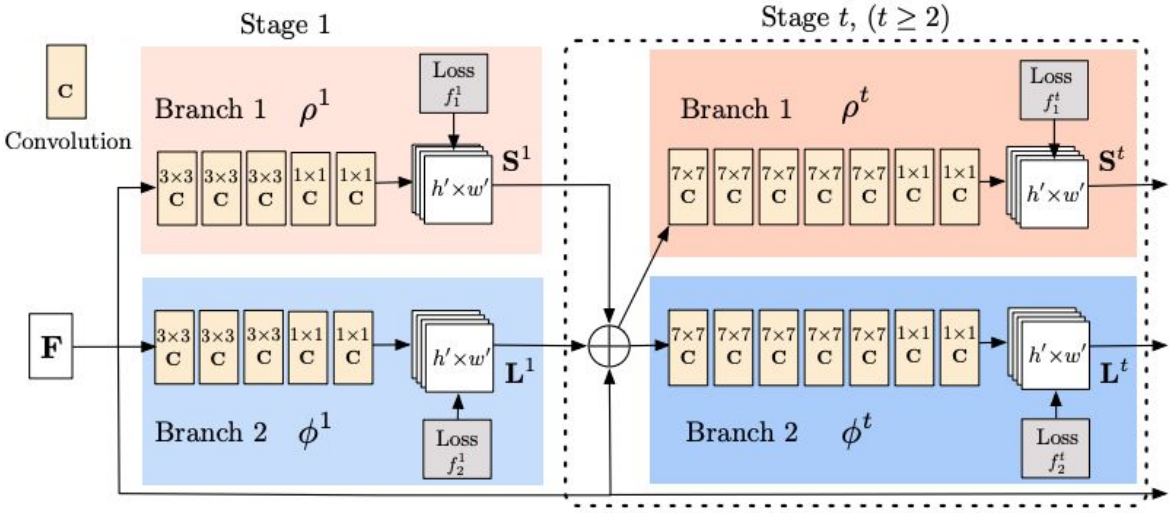


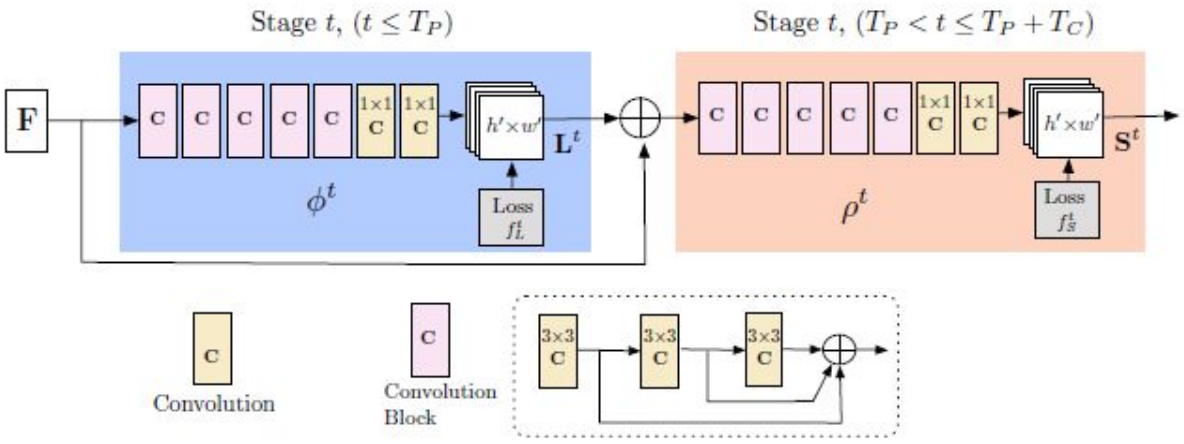
There are two versions of openpose, the first version was released on 24 Nov 2016 and the next version was released on 18 Dec 2018 which has some minor changes in the neural network but still has the same pipeline as the previous paper. You can find the 18 Dec 2018 released paper here.



You can find the original openpose github repo here.(Image credit original research paper.)

The above image shows the basic pipeline of open pose. Initially an image is fed into a two branch multistage CNN. Where two branches means that the CNN produces two different outputs and multistage means that many networks are stacked on top of each other at every stage. This multi stacking of networks increases the depth of the model which allows it to capture and analyze more data.





This two branch multi stage works in the following way: The top branch(represented by beige color) generates the confidence maps for different body parts like the left shoulder, right eye etc.

The confidence maps are 2D representations of the belief that a particular body part can be located in the given pixel. They are described by:

$$S = (S_1, S_2.., S_J) \ where \ S_j \epsilon R^{w*h}, j \epsilon 1...J$$

$$\mathbf{S}^*_{j,k}(\mathbf{p}) = \exp\left(-\frac{||\mathbf{p} - \mathbf{x}_{j,k}||_2^2}{\sigma^2}\right)$$

Where J is the number of body part locations.

The bottom branch(represented by blue color) is responsible for creating the Part Affinity Field which tells us about the associations between different body parts.

Part Affinity is a set of 2D vector fields that encodes location and orientation of limbs of different people in the image. It encodes the data in the form of pairwise connections between body parts. These are described by:

$$L = (L_1, L_2.., L_C) \ where \ L_c \epsilon R^{w*h*c}, c \epsilon 1...C$$

The multi Stacking works in the following way:

The left part(or the first stage) gives the output of confidence maps and association of different body parts and along with the original image **F** are concatenated (represented by **+**) and fed into the next stage.

There are t such subsequent stages and in each stage the confidence maps are enhanced and more features are recognised.In openpose we have 6 stages that is t = 6.

In the final stage the S(Confidence maps) and L(Part Affinity Field) are passed into greedy algorithms for further processing.

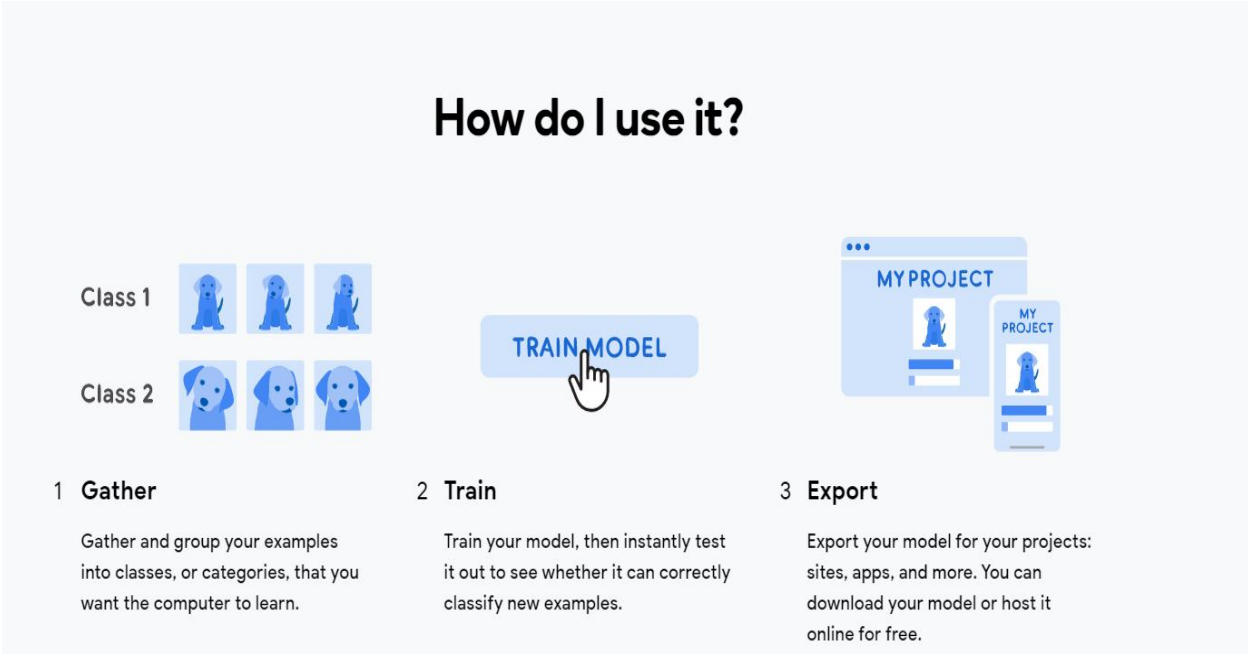
In this project we are using a pretrained Openpose model. This model will be giving us the locations of 18 key points and we are going to use these key points to extract the required ROI's.

Google teachable Machines:

Teachable Machine is a web-based tool that makes creating machine learning models fast, easy, and accessible to everyone.

Teachable Machine is flexible – use files or capture examples live. It's respectful of the way you work. You can even choose to use it entirely on-device, without any webcam or microphone data leaving your computer.

With this we can create a classifier that can validate the results of our Openpose ROI extraction.



The model has been trained on two classes.

- Class1 contains images of eyes.
- Class2 contains images of nose, mouth & forehead

We need to upload the training data for each class and then train the model. The trained model can be exported for using elsewhere. We will export the code to our main file and then stitch it to work with the rest of the code.

You can find the Teachable Machine website here.

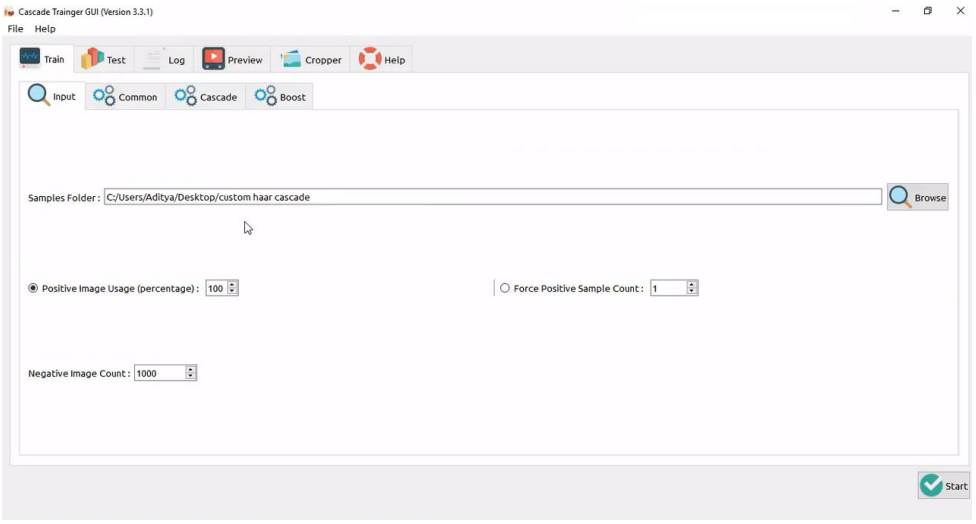
Haar Cascade Object Detection:

In respect to our aim, we are using Cascade Trainer for detecting the ROI of the eye.

Haar cascade classifiers are used for object detection. This method was proposed by Paul Viola and Michael Jones of the Viola-Jones method. This is a machine learning model where a lot of positive and negative images are used to train the model and classify the given images accordingly.

For training the Haar Cascade Object Detection we are going to use the Cascade Trainer GUI.

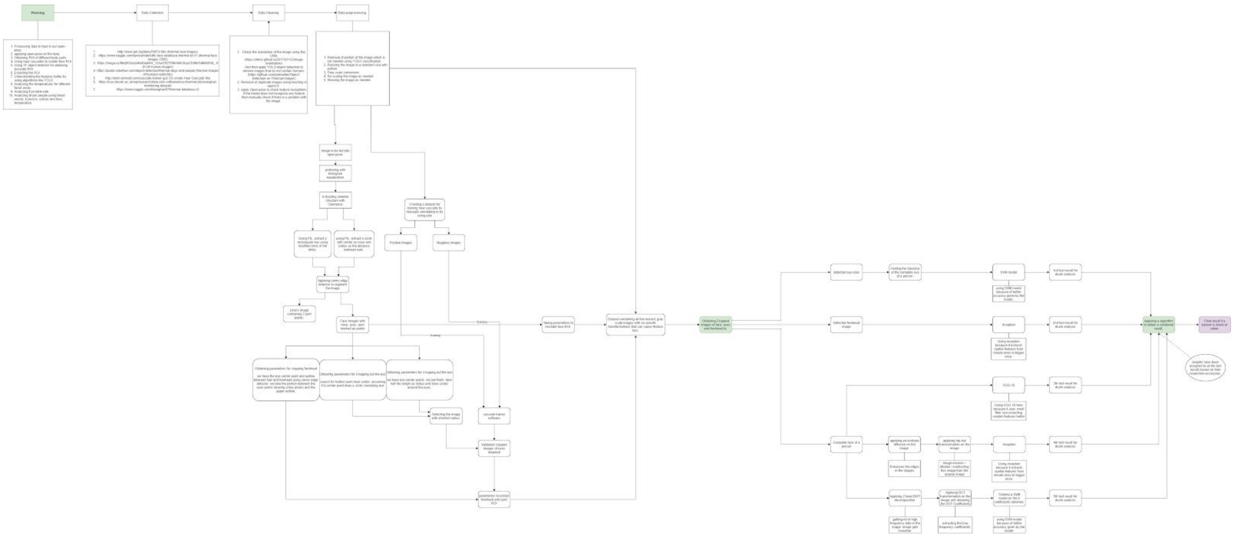
Using this GUI makes our task of training the model very easy.

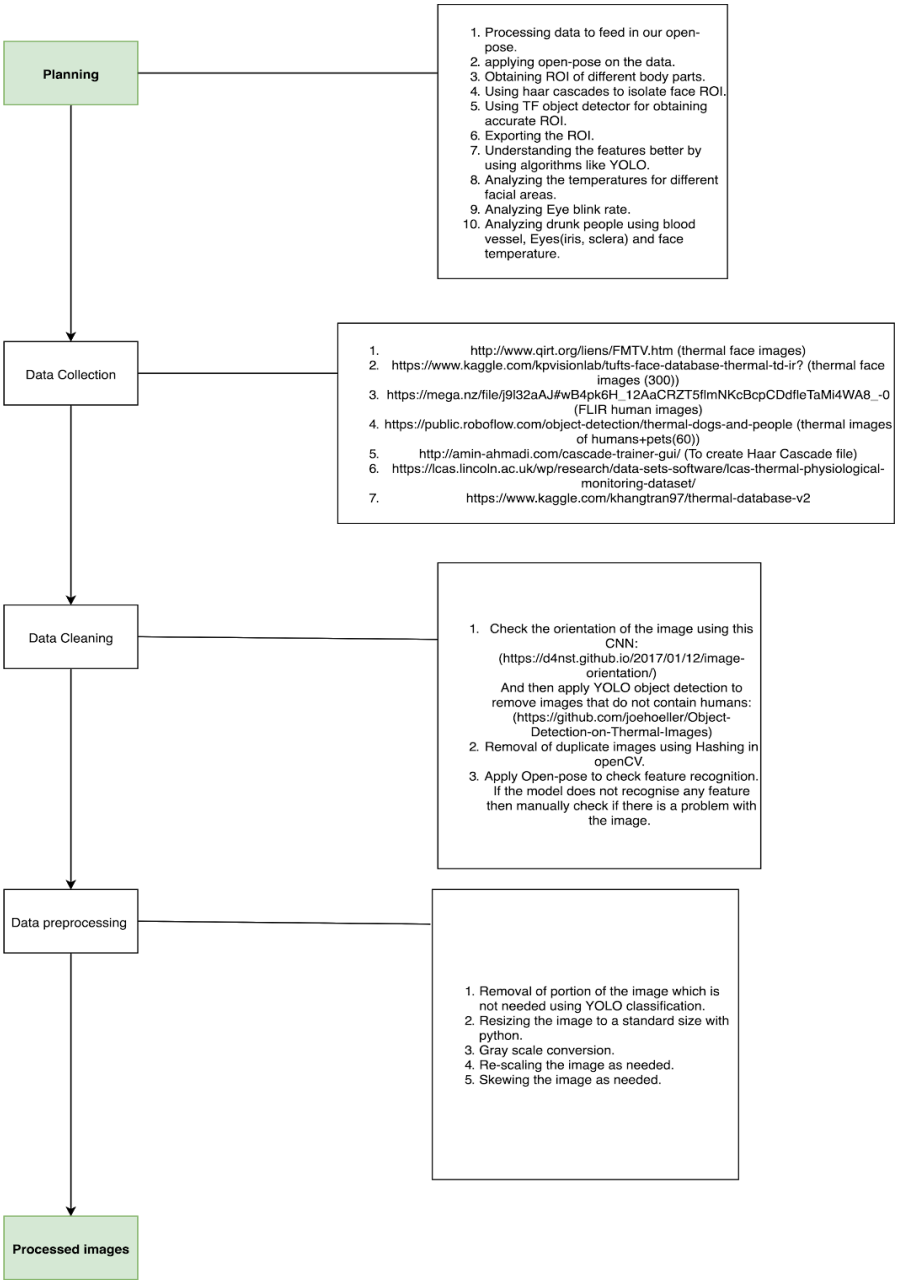


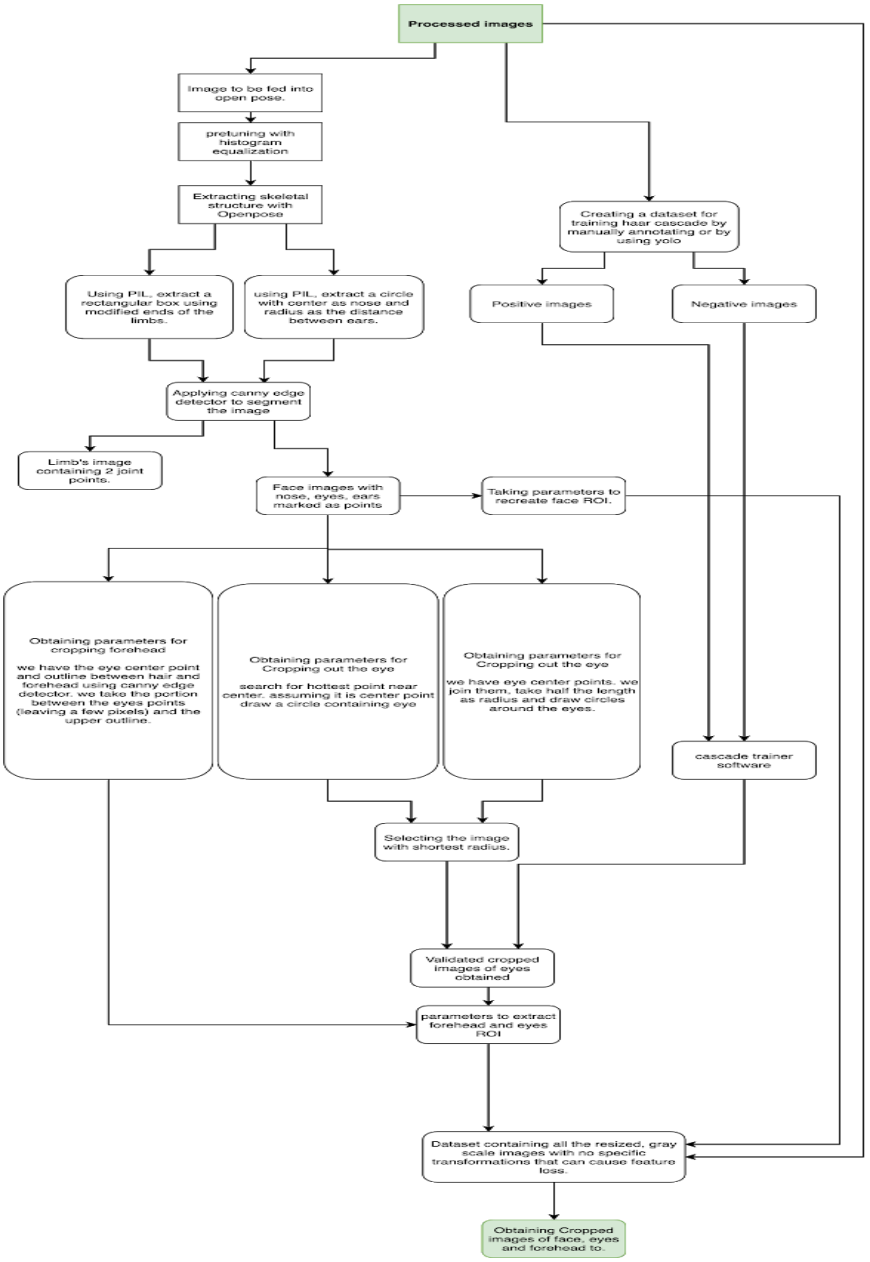
It just needs our data sets with positive and negative images. These data sets need to be in folders named “p” for positive images and “n” for negative images.

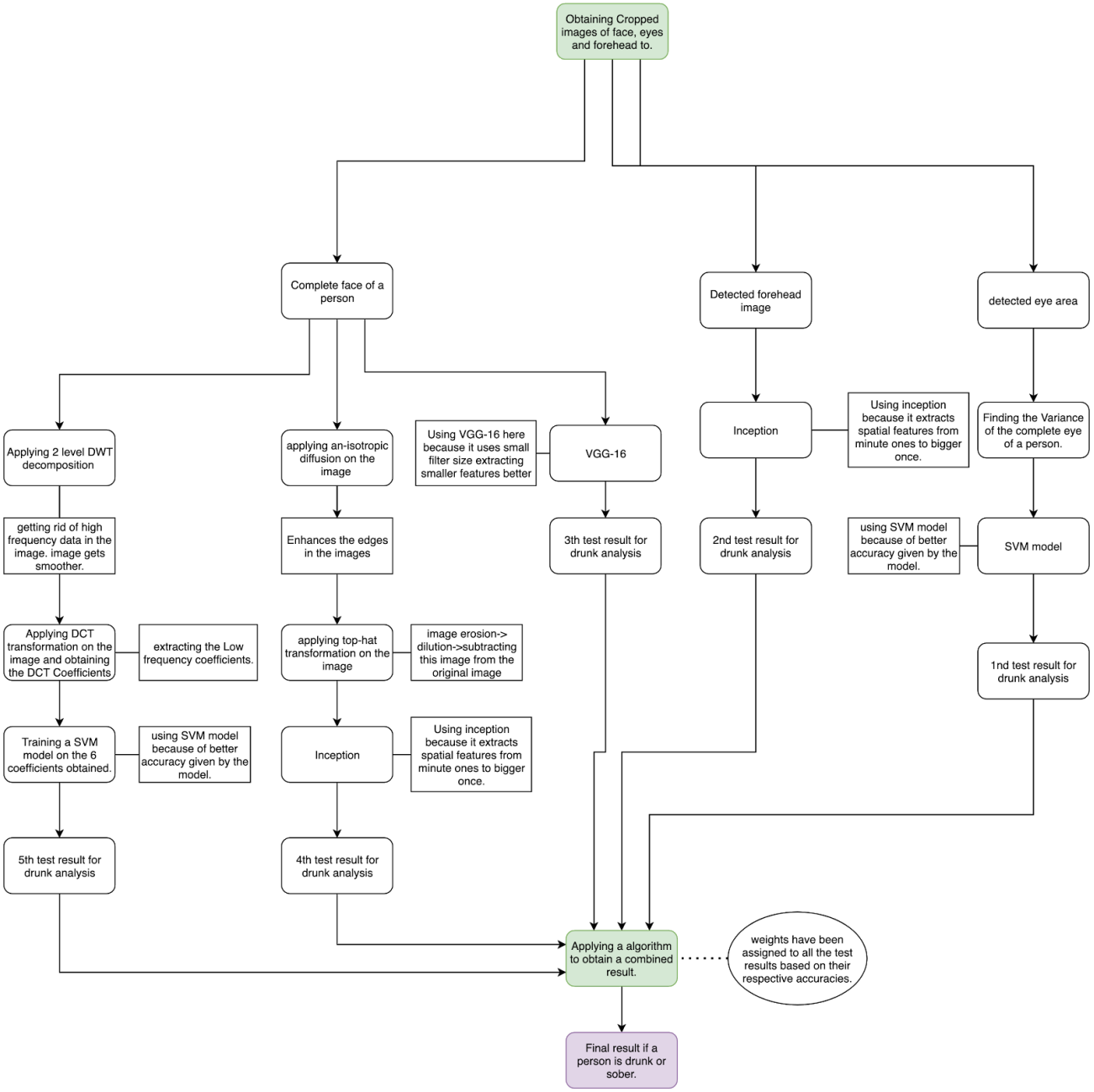
The p data set should contain all the images that we want to be recognised by the Haar Cascade object detection. Where as the n data set can be anything except the thing that you want to find. To make the model as accurate as possible we must choose the appropriate data sets.

Flowchart


Data Collection
Data Cleaning
Data preprocessing


Planning
1. Processing data to feed in our open-pose.
2. applying open-pose on the data.
3. Obtaining ROI of different body parts.
4. Using haar cascades to isolate face ROI.
5. Using TF object detector for obtaining accurate ROI.
6. Exporting the ROI.
7. Understanding the features better by using algorithms like YOLO.
8. Analyzing the temperatures for different facial areas.
9. Analyzing Eye blink rate.
10. Analyzing drunk people using blood vessel, Eyes(iris, sclera) and face temperature.
Data Collection
1. http://www.qirt.org/liens/FMTV.htm (thermal face images)
2. https://www.kaggle.com/kpvisionlab/tufts-face-database-thermal-td-ir? (thermal face images (300))
3. https://mega.nz/file/j9l32aAJ#wB4pk6H_12AaCRZT5flmNKcBcpCDdfleTaMi4WA8_-0 (FLIR human images)
4. https://public.roboflow.com/object-detection/thermal-dogs-and-people (thermal images of humans+pets(60))
5. http://amin-ahmadi.com/cascade-trainer-gui/ (To create Haar Cascade file)
6. https://lcas.lincoln.ac.uk/wp/research/data-sets-software/lcas-thermal-physiological-monitoring-dataset/
7. https://www.kaggle.com/khangtran97/thermal-database-v2
Data Cleaning
1. Check the orientation of the image using this CNN: (https://d4nst.github.io/2017/01/12/image-orientation/) And then apply YOLO object detection to remove images that do not contain humans: (https://github.com/joehoeller/Object-Detection-on-Thermal-Images)
2. Removal of duplicate images using Hashing in openCV.
3. Apply Open-pose to check feature recognition. If the model does not recognise any feature then manually check if there is a problem with the image.
Data preprocessing
1. Removal of portion of the image which is not needed using YOLO classification.
2. Resizing the image to a standard size with python.
3. Gray scale conversion.
4. Re-scaling the image as needed.
5. Skewing the image as needed.
Processed images


Processed images
Image to be fed into open pose.
prefuning with histogram equalization
Extracting skeletal structure with Openpose
Creating a dataset for training haar cascade by manually annotating or by using yolo
Using PIL, extract a rectangular box using modified ends of the limbs.
using PIL, extract a circle with center as nose and radius as the distance between ears.
Positive images
Negative images
Applying canny edge detector to segment the image
Limb's image containing 2 joint points.
Face images with nose, eyes, ears marked as points
Taking parameters to recreate face ROI.
Obtaining parameters for cropping forehead
we have the eye center point and outline between hair and forehead using canny edge detector. we take the portion between the eyes points (leaving a few pixels) and the upper outline.
Obtaining parameters for Cropping out the eye
search for hottest point near center. assuming it is center point draw a circle containing eye
Obtaining parameters for Cropping out the eye
we have eye center points. we join them, take half the length as radius and draw circles around the eyes.
cascade trainer software
Selecting the image with shortest radius.
Validated cropped images of eyes obtained
parameters to extract forehead and eyes ROI
Dataset containing all the resized, gray scale images with no specific transformations that can cause feature loss.
Obtaining Cropped images of face, eyes and forehead to.


Obtaining Cropped images of face, eyes and forehead to.
Complete face of a person
Detected forehead image
detected eye area
Applying 2 level DWT decomposition
applying an-isotropic diffusion on the image
Using VGG-16 here because it uses small filter size extracting smaller features better
VGG-16
Inception
Using inception because it extracts spatial features from minute ones to bigger once.
Finding the Variance of the complete eye of a person.
getting rid of high frequency data in the image. image gets smoother.
Enhances the edges in the images
3th test result for drunk analysis
2nd test result for drunk analysis
using SVM model because of better accuracy given by the model.
SVM model
Applying DCT transformation on the image and obtaining the DCT Coefficients
extracting the Low frequency coefficients.
applying top-hat transformation on the image
image erosion-> dilution->subtracting this image from the original image
1nd test result for drunk analysis
Training a SVM model on the 6 coefficients obtained.
using SVM model because of better accuracy given by the model.
Inception
Using inception because it extracts spatial features from minute ones to bigger once.
5th test result for drunk analysis
4th test result for drunk analysis
Applying a algorithm to obtain a combined result.
weights have been assigned to all the test results based on their respective accuracies.
Final result if a person is drunk or sober.

Methodology

OpenPose:

Collect all the required data from the following sources:

- http://www.qirt.org/liens/FMTV.htm (thermal face images)
- https://www.kaggle.com/kpvisionlab/tufts-face-database-thermal-td-ir? (thermal face images (300))
- https://mega.nz/file/j9l32aAJ#wB4pk6H_12AaCRZT5flmNKcBcpCDdfleTaMi4WA8_-0 (FLIR human images)
- https://public.roboflow.com/object-detection/thermal-dogs-and-people (thermal images of humans+pets(60))
- https://lcas.lincoln.ac.uk/wp/research/data-sets-software/lcas-thermal-physiological-monitoring-dataset/
- https://www.kaggle.com/khangtran97/thermal-database-v2

Open the main file in which we call Detection and Thermal Openpose.

We initiate the detection process through the __init__ dunder method.

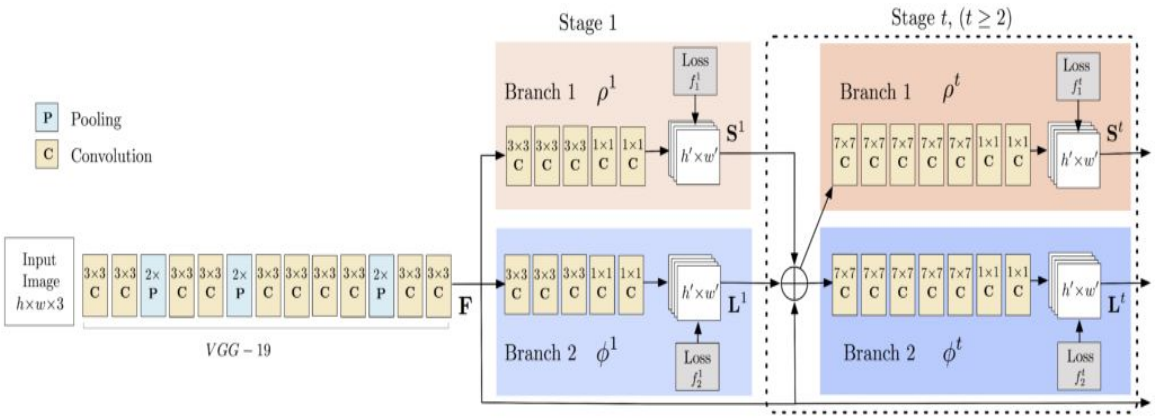
Inside the openpose model we have functions for individual body parts(Face,eyes,forehead etc).

Based on the use case we will be calling the required function.

We will be using the 18 point model trained on the COCO dataset for this project. The keypoints along with their numbering used by the COCO Dataset is given below:

COCO Output Format
Nose – 0, Neck – 1, Right Shoulder – 2, Right Elbow – 3, Right Wrist – 4, Left Shoulder – 5, Left Elbow – 6, Left Wrist – 7, Right Hip – 8, Right Knee – 9, Right Ankle – 10, Left Hip – 11, Left Knee – 12, LAnkle – 13, Right Eye – 14, Left Eye – 15, Right Ear – 16, Left Ear – 17, Background – 18

Network Architecture



The model takes as input a color image of size h x w and produces output, as an array of matrices which consists of the confidence maps of Keypoints and Part Affinity Heatmaps for each keypoint pair. The above network architecture consists of two stages

1. **Stage 0**: The first 10 layers of the VGGNet are used to create feature maps for the input image.
2. **Stage 1**: A two branch multistage CNN is used where
    a. The first branch predicts a set of 2D Confidence Maps (S) of body part locations (e.g. elbow, knee etc.). A Confidence Map is a grayscale image which has a high value at locations where the likelihood of a certain body part is high. For example, the Confidence Map for the Left Shoulder is shown in Figure below. It has high values at all locations where there is a left shoulder.

Fig - Confidence map for left shoulder

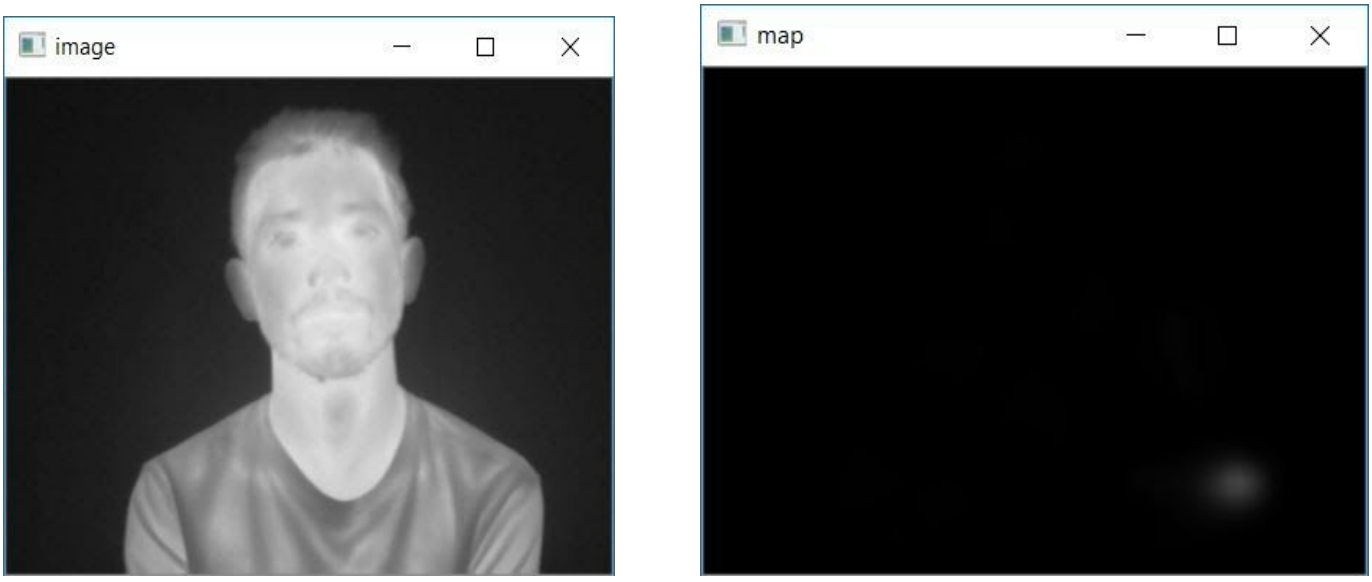


Fig - Thermal image(left) and Confidence map for Left-Shoulder(right).

b. The second branch predicts a set of 2D vector fields (L) of Part Affinities (PAF), which encode the degree of association between body parts (key points). In the figure below – part affinity between the Neck and Left shoulder is shown.

Fig - Paf between neck and L-shoulder

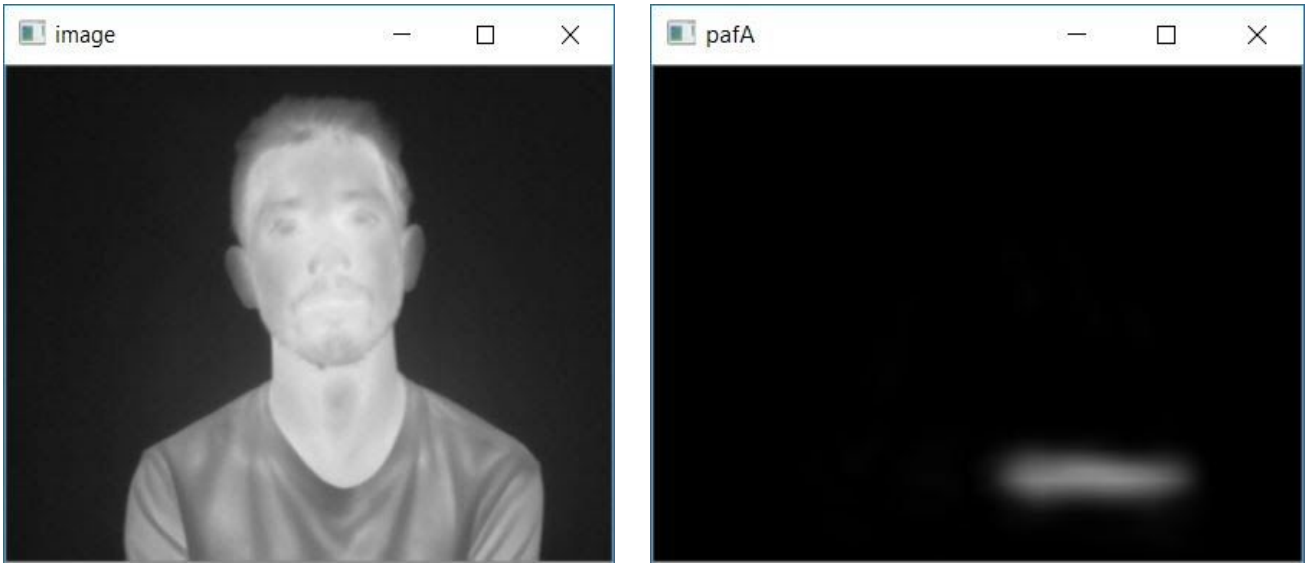


Fig - Thermal image(left) and Paf between Neck and L- sho

Generating Output from image:

1. Load Network:

```
#initialize openpose parameters
self.all_points = []
protoFile = ".//model_data//pose_deploy_linevec.prototxt"
weightsFile = ".//model_data//pose_iter_440000.caffemodel"
self.net = cv2.dnn.readNetFromCaffe(protoFile, weightsFile)
```

   This Code snippet is from the init method of the thermal pose class.

2. Load image and create input blob:

```
frameWidth = image1.shape[1]
frameHeight = image1.shape[0]

inHeight = 368
inWidth = int((inHeight/frameHeight)*frameWidth)

#blob contains all three channels of image in structure like [R,G,B]
inpBlob = cv2.dnn.blobFromImage(image1, 1.0 / 255, (inWidth, inHeight),
                          (0, 0, 0), swapRB=False, crop=False)
```

   This code snippet is from function start_openpose() where thermal image is passed as argument which is received by function in a variable named image1.

3. Forward pass through net

```
self.net.setInput(inpBlob)
t = time.time()
output = self.net.forward()
print("Time Taken in forward pass = {}".format(time.time() - t))
```

   This code snippet is also from the function start_openpose.

3.1 Sample Output

```
probMap = output[0,part,:,:]
probMap = cv2.resize(probMap, (image1.shape[1], image1.shape[0]))
cv2.imshow('map',probMap)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Here part = 0, which corresponds to the confidence map of the nose.

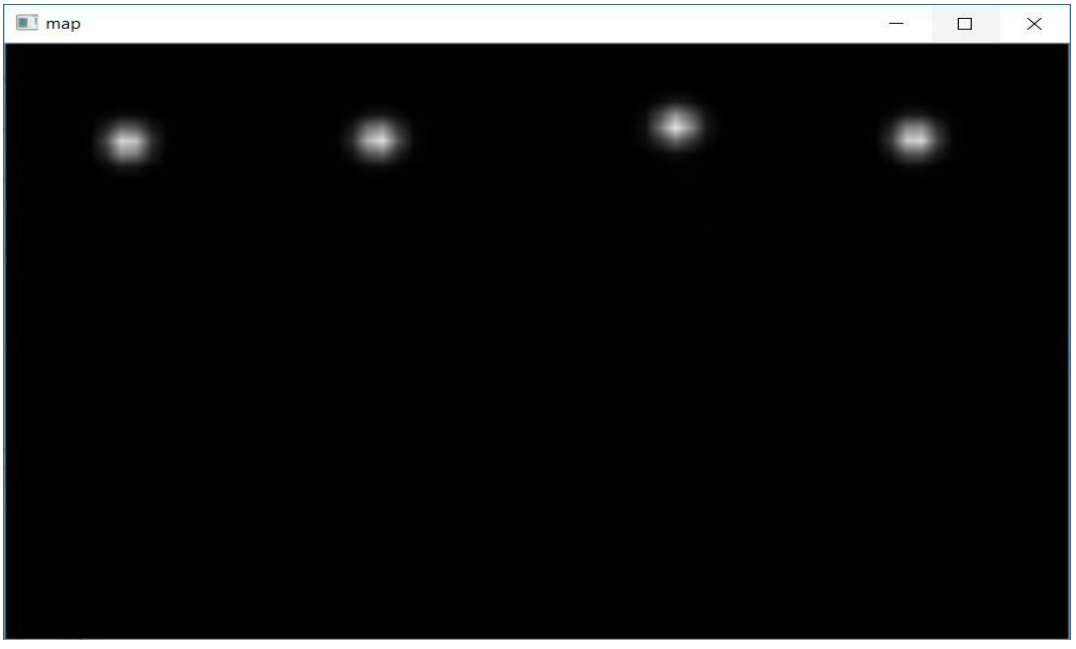


Fig - Original image with confidence map of nose in case of multiple persons

4. Detection of Keypoints

   As seen from the above figures, the zeroth matrix gives the confidence map for the nose. Similarly, the first Matrix corresponds to the neck and so on.

   All code snippets and explanation belong to getkeypoints() function

```
def getKeypoints(probMap, threshold=0.1):

    mapSmooth = cv2.GaussianBlur(probMap,(3,3),0,0)
    mapMask = np.uint8(mapSmooth>threshold)
    keypoints = []

    #find the blobs
```

   This gives a matrix containing blobs in the region corresponding to the keypoint as shown below.

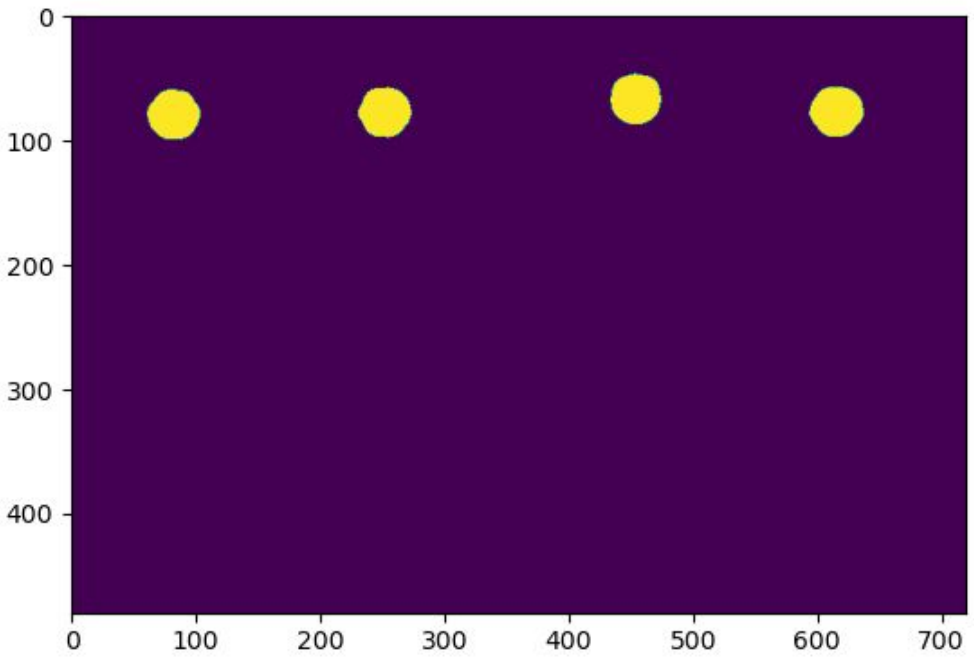


Fig - Confidence map after applying threshold

In order to find the exact location of the keypoints, we need to find the maxima for each blob. We do the following :

1. First find all the contours of the region corresponding to the keypoints.
2. Create a mask for this region.
3. Extract the probMap for this region by multiplying the probMap with this mask.
4. Find the local maxima for this region. This is done for each contour (keypoint region).

```
#find the blobs
contours, _ = cv2.findContours(mapMask, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

#for each blob find the maxima
for cnt in contours:
    blobMask = np.zeros(mapMask.shape)
    blobMask = cv2.fillConvexPoly(blobMask, cnt, 1)
    maskedProbMap = mapSmooth * blobMask
    _, maxVal, _, maxLoc = cv2.minMaxLoc(maskedProbMap)
    keypoints.append(maxLoc + (probMap[maxLoc[1], maxLoc[0]],))
```

We save the x, y coordinates and the probability score for each keypoint. We also assign an ID to each key point that we have found. This will be used later while joining the parts or connections between keypoint pairs.

Fig - detected keypoints overlaid on input image.

5. Find Valid Pairs

A valid pair is a body part joining two keypoints, belonging to the same person. One simple way of finding the valid pairs would be to find the minimum distance between one joint and all possible other joints. For example, in the figure given below, we can find the distance between the marked Nose and all other Necks. **The minimum distance pair should be the one corresponding to the same person**.

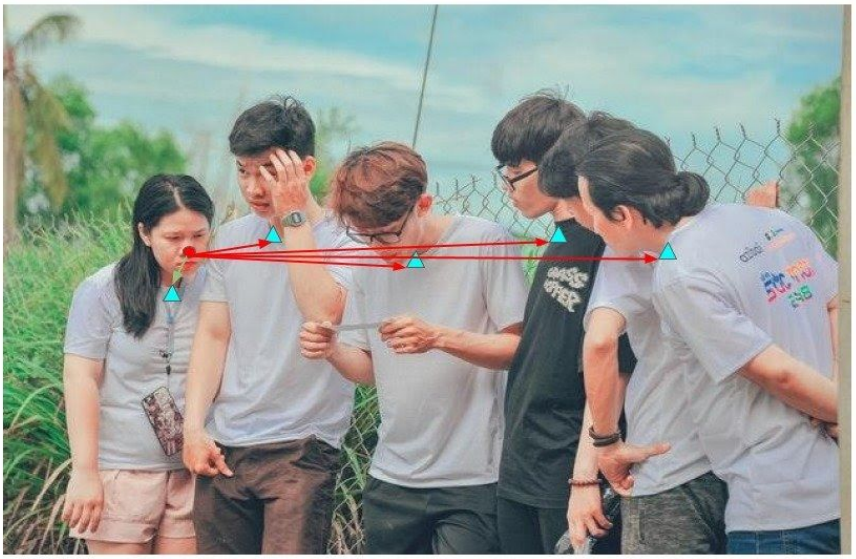


Fig - Getting the connection between keypoints by using a simple distance measure.

This approach might not work for all pairs, specially, when the image contains too many people or there is occlusion of parts. For example, for the pair, Left-Elbow -> Left Wrist – The wrist of the 3rd person is closer to the elbow of the 2nd person as compared to his own wrist. Thus, it will not result in a valid pair.

Fig - Only using distance between keypoints might fail in some cases.

This is where the Part Affinity Maps come into play. They give the direction along with the affinity between two joint pairs. So, the pair should not only have minimum distance, but their direction should also comply with the PAF Heatmaps direction.

Thus, in the above case, even though the distance measure wrongly identifies the pair, OpenPose will give the correct result since the PAF will comply only with the unit vector joining Elbow and Wrist of the 2nd person.

Note : All code snippets mentioned below are from function getValidPairs()

For each body part pair, we do the following :

1. Take the key points belonging to a pair. Put them in separate lists (candA and candB). Each point from candA will be connected to some point in candB. The figure given below shows the points in candA and candB for the pair Neck -> Right-Shoulder. Thus we try all possible connections that can be formed and choose the best one.

Fig - Showing the candidates for matching for the pair Neck -> Nose.

```
# A->B constitute a limb
pafA = output[0, self.mapIdx[k][0], :, :]
pafB = output[0, self.mapIdx[k][1], :, :]
pafA = cv2.resize(pafA, (frameWidth, frameHeight))
pafB = cv2.resize(pafB, (frameWidth, frameHeight))

# Find the keypoints for the first and second limb
candA = detected_keypoints[self.POSE_PAIRS[k][0]]
candB = detected_keypoints[self.POSE_PAIRS[k][1]]
```

Fig - Code snippet for above discussed process.

2. Find the unit vector joining the two points in consideration. This gives the direction of the line joining them.

```
# Find d_ij
d_ij = np.subtract(candB[j][:2], candA[i][:2])
norm = np.linalg.norm(d_ij)
if norm:
    d_ij = d_ij / norm
```

3. Create an array of 10 interpolated points on the line joining the two points.

```
# Find p(u)
interp_coord = list(zip(np.linspace(candA[i][0], candB[j][0], num=n_interp_samples),
                        np.linspace(candA[i][1], candB[j][1], num=n_interp_samples)))
# Find L(p(u))
paf_interp = []
for k in range(len(interp_coord)):
    paf_interp.append([pafA[int(round(interp_coord[k][1])), int(round(interp_coord[k][0]))],
                       pafB[int(round(interp_coord[k][1])), int(round(interp_coord[k][0]))] ])
```

Considering we know about two points, nose and neck, from the above code we try to find 10 points (n_interp_samples) on the line joining the two points. These points will be utilized to calculate if the vector lies in PAF or not.

4. Take the dot product between the PAF on these points and the unit vector d_ij

```
# Find E
paf_scores = np.dot(paf_interp, d_ij)
avg_paf_score = sum(paf_scores)/len(paf_scores)
```

5. Term the pair as valid if 70% of the points satisfy the criteria.

```
# Check if the connection is valid
# If the fraction of interpolated vectors aligned with PAF is higher then threshold -> Valid Pair
if ( len(np.where(paf_scores > paf_score_th)[0]) / n_interp_samples ) > conf_th :
    if avg_paf_score > maxScore:
        max_j = j
        maxScore = avg_paf_score
        found = 1
```

Detect forehead by calling get forehead():

In this function we apply openpose on the input image and extract the midpoint of the line joining eyes. Then we apply canny-edge detection for getting an image through which we can extract the boundaries of the forehead. Considering the obtained midpoint we traverse upwards by reducing the y coordinate by 1 and check if the pixel encountered is dark indicating an edge given by a canny-edge detector. Then by taking the rows and columns we extract a part of the image containing the forehead.

```
def getForehead(self,image):
    '''
    input - image
    return - cropped forehead image
    '''
    #clone image to retain original
    clone = image.copy()
    #applying openpose
    self.start_openpose(image,show=False)
    # extract eye coordinates
    l_point,r_point = self.all_points[14],self.all_points[15]
    #offset will be added if image is cropped during yolo procedure
    x1 = l_point[0][0] + self.offset_x
    y1 = l_point[0][1] + self.offset_y
    x2 = r_point[0][0] + self.offset_x
    y2 = r_point[0][1] + self.offset_y
    #mid point
    mid_point = (x1+x2)//2
    traverse=y2
    #applying canny filter
    canny_img = self.applyCanny(clone)
    #print('debug --------forehead_process--------')
    traverse-=10
    while canny_img[traverse][mid_point]==0:
        traverse-=1
    #extracting forehead roi
    forehead = image[traverse:y2,x1:x2]
    return forehead
```

Fig- code snippet of the function

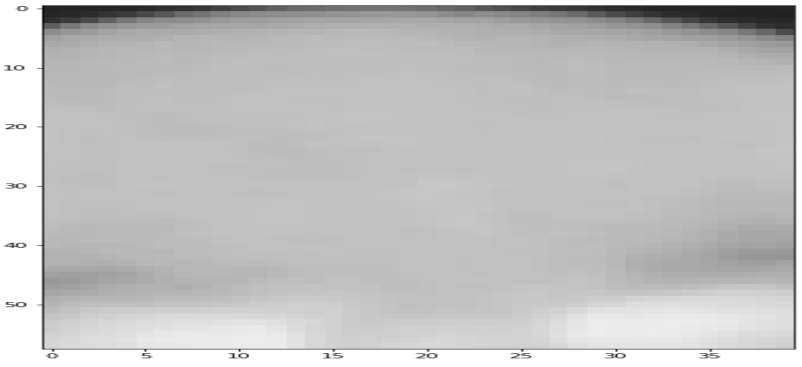


Fig - sample output of forehead

Next we extract the eyes by calling geteyes():

Here we call openpose on the image and then find the distance between the eyes. Taking half the distance as the radius we are cropping out the image.

```
def getEyes(self,image):
  '''
  input - image
  return - cropped images of both eyes as [leye,reye] list of two images
  '''
  #starting openpose
  self.start_openpose(image,show=False)
  #extracting eyes coordinates
  l_point,r_point = self.all_points[14],self.all_points[15]
  if not l_point or not r_point:
    print('eyes not detected :(')
    return None
  print(l_point,r_point)
  #assuming both eyes are now detected
  x1 = l_point[0][0] + self.offset_x
  y1 = l_point[0][1] + self.offset_y
  x2 = r_point[0][0] + self.offset_x
  y2 = r_point[0][1] + self.offset_y
  #print(x1,y1,x2,y2)
  #euclidian destance between two points
  dist = math.sqrt((x1-x2)**2 + (y1-y2)**2)
  #radius as half the distance
  margin = int(dist//2)
  #extracting roi
  left_eye = image[y1-margin:y1+margin,x1-margin:x1+margin]
  right_eye = image[y2-margin:y2+margin,x2-margin:x2+margin]
  # cv2.rectangle(image,(x1-margin,y1-margin),(x1+margin,y1+margin),(255,0,0),2) (debug process)
  return [left_eye,right_eye]
```

Fig - Code snippet of getEyes method.

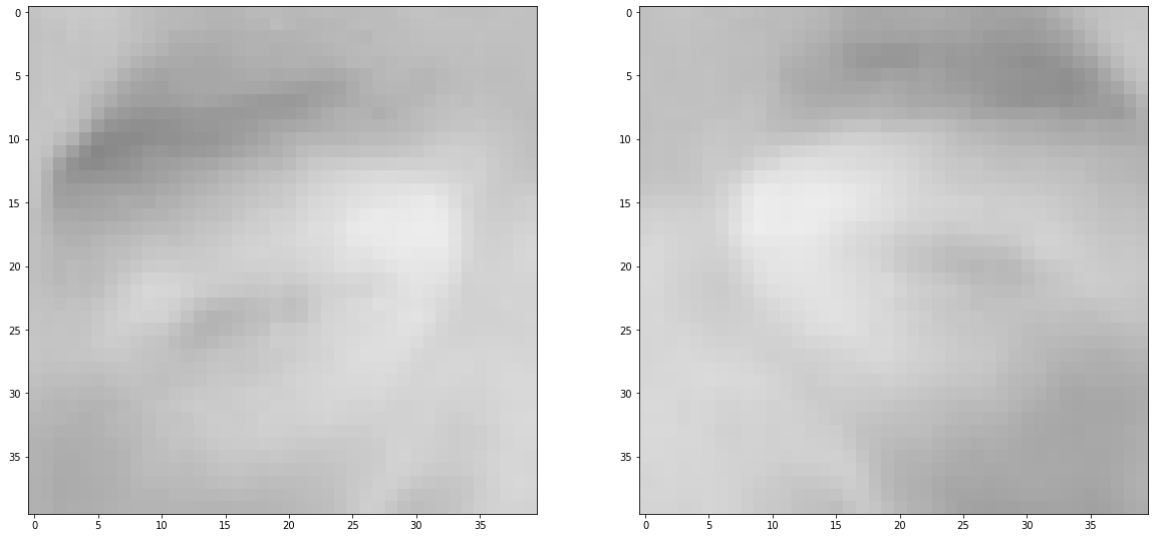


Fig - sample output of getEyes method [leye,reye]

After this we will be calling the getface() function:

In this function we will be calling the openpose model and save the nose point in a variable. Then we apply a canny-edge detector on the image, using the nose point we will travers vertically to find the top portion of the head. Using these two points we will crop a circle. This cropping is done by using a mask. This mask is fairly straightforward, here we will set all the values of the mask that lie inside the circle given by the two points as 1 and rest as 0. Then we perform an and operation on the mask and original image which gives us the required part of the image.

```
def getFace(self,image):
  '''
  input - image
  return - cropped roi of face from image
  '''
  #applying openpose
  self.start_openpose(image,show=False)
  #extracting nose point
  nose_point = self.all_points[0]
  #offest is added to balance out coordinated if yolo is applied on image to crop human roi
  x,y = nose_point[0][0]+self.offset_x,nose_point[0][1]+self.offset_y
  traverse = y
  #apply canny filter
  canny_img = self.applyCanny(image)
  #plt.subplot(1,3,1)      '''(debug)'''
  #plt.imshow(canny_img)
  res = 0
  #getting topmost point of head and assuming it as radius for cropping
  while traverse>0:
    if canny_img[traverse][x] == 255:
      res = traverse
    traverse-=1
  radius = abs(res-y)

  #croppig circle out of image
  # ref: https://stackoverflow.com/questions/36911877/cropping-circle-from-image-using-opencv-python
  height,width,_ = image.shape
  #create mask of zeros
  mask = np.zeros((height,width), np.uint8)
  #putting ones at points inside the circle
  circle_img = cv2.circle(mask,(x,y),radius,(255,255,255),thickness=-1)
  #bitwise and bw image and mask to get face
  masked_data = cv2.bitwise_and(image, image, mask=circle_img)
  grey = cv2.cvtColor(masked_data,cv2.COLOR_BGR2GRAY)
  _,thresh = cv2.threshold(grey,1,255,cv2.THRESH_BINARY)
  contours,_ = cv2.findContours(circle_img,cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)

  cnt = contours[0]
  approx = cv2.approxPolyDP(cnt,0.02*cv2.arcLength(cnt,True),True)
  x,y,w,h = cv2.boundingRect(approx)

  face_roi = image[y:y+h,x:x+w]
  #plt.subplot(1,3,2)
  #plt.imshow(face_roi)
  #plt.subplot(1,3,3)
  #plt.imshow(masked_data)
  return face_roi
```

Fig - Snippet of getFace method

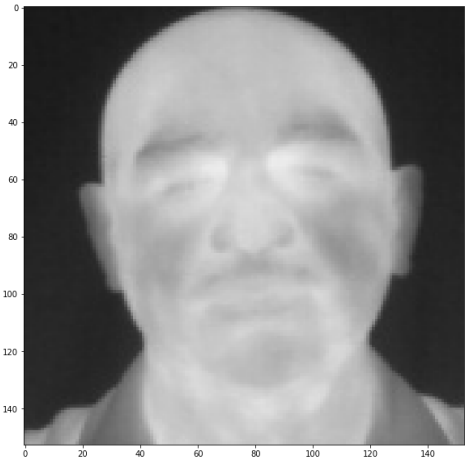


Fig - sample output of getFace method

Google Teachable Machine:

After collecting the ROI's we will feed the images to the classifier obtained from Google Teachable Machine. In the validation function we resize the image based on the size of the images used to train the classifier and we call predict to feed the images. These images are then passed onto the next models.

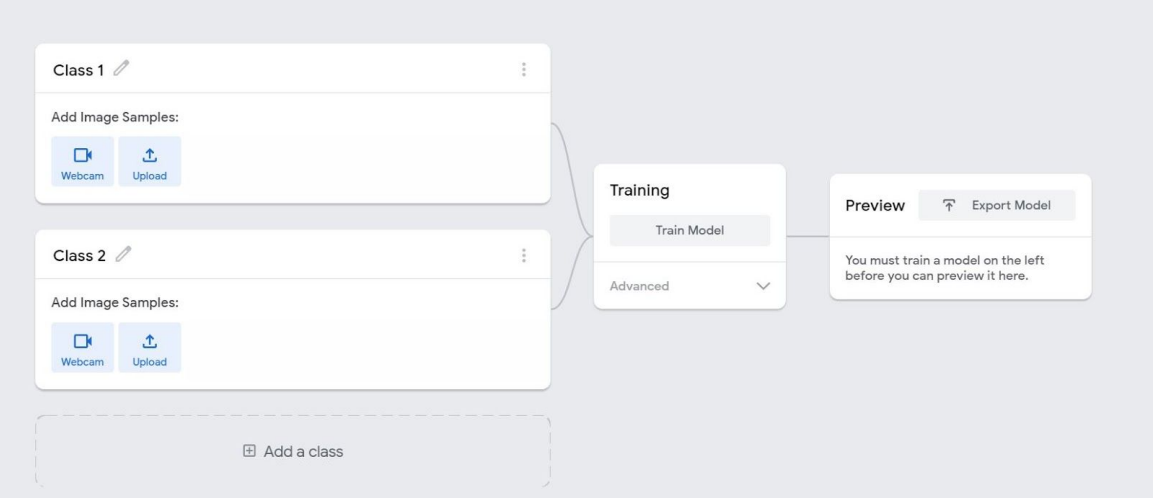


Fig- GUI of Teachable Machine

Drunk Analysis Models:

A total of 5 methods have been used to analyse if a person is Drunk or sober. The main idea behind using 5 different ways to analyse was to get a better prediction as a whole.

We already have all the 5 models trained and saved as h5 or sav files (CODE LINK HERE). We directly import the model in the code for prediction. Before predicting, if a person is Drunk or not, we pre-process the images as required for different models.

Then based on the requirement of each model we will call the associated getFace(), getEyes(), or the getForehead() function.

1st method: Training a model on raw face images.

We are using a VGG16 model for faces because faces have smaller details which need to be analysed and VGG16 model uses 3X3 filters to extract features making it

suitable for this training. Before feeding the image in the VGG network, it is normalized, resized and reshaped using the defined function preprocess(img).

```
def face_model(self,image):  #2
    def preprocess(img):
        img = img.astype('float32')/255
        resize_img = cv2.resize(img, (224, 224))
        reshape_img = resize_img.reshape(-1, 224, 224, 3)
        return reshape_img

    def prediction(img):
        img = preprocess(img)
        output = self.model2.predict(img)
        if(output<50):
            output = 0
        else:
            output = 1
        return output
    return prediction(image)
```

Fig - code snippet of face model

2nd method: Training a model on the forehead images.

A drunk person's forehead shows pattern difference, some parts tend to become light in color while others seem to be darker because of the change of rate of blood flow after alcohol consumption. An inception model helps us to study these patterns as it uses filters of various sizes (1x1, 3X3, 5X5) on the same level. Before feeding the image in the VGG network, it is normalized, resized and reshaped using the defined function preprocess(img)

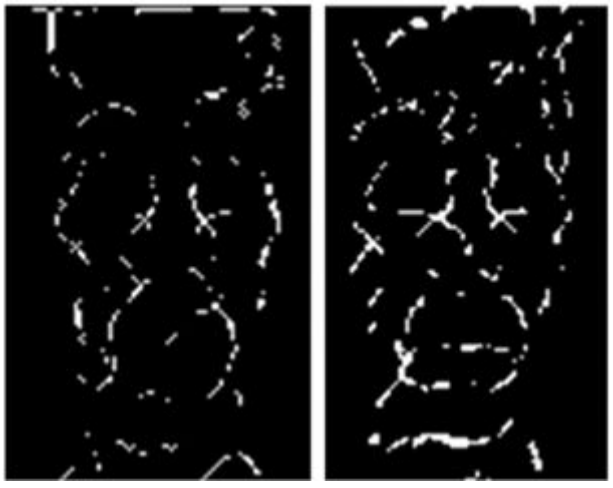
```
def forehead_model(self,image): #1
    def preprocess(img):
        img = img.astype('float32')/255
        resize_img = cv2.resize(img, (224, 224))
        reshape_img = resize_img.reshape(-1, 224, 224, 3)
        return reshape_img

    def prediction(img):
        img = preprocess(img)
        output = self.model1.predict(img)
        if(output<50):
            output = 0
        else:
            output = 1
        return output
    return prediction(image)
```

Fig - Code snippet of Forehead model

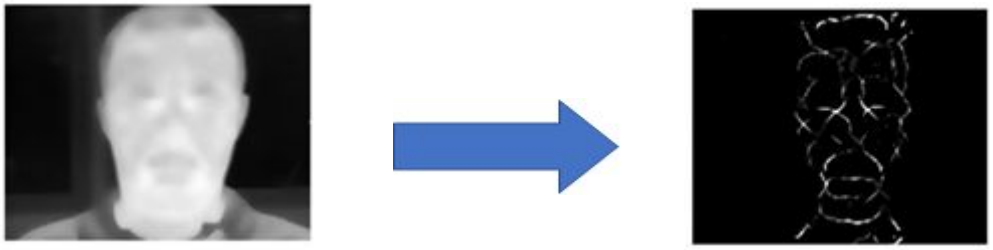
3rd method: Training a model image highlighting the blood vessels of the face.

When a person drinks, the blood flow in his body increases. Eyes, nose, forehead, mouth area of the face are full of vessels and hence it can be used to find if a person is drunk or sober.



For implementation, we apply anisotropic diffusion followed by top-hat transformation. Anisotropic diffusion enhances the image without making changes in the edges. Top-hat transformation extracts the image features which are readily visible. It extracts white (vessels here) features against a dark background. Process:

- Making an Opening -> Erosion of the image, followed by dilation.
- Hot THT -> Subtracting opening image from the original image.



After this, the image is fed in the inception model for prediction.

4th method: Training a model on the variance of the eye of drunk and sober people.

When a person consumes alcohol, his/her eyes shows a temperature difference (for the same reason that the blood flow rate changes and eyes are surrounded by many blood vessels). This difference in the eye can be used to detect if a person is drunk or sober. We take the variance of the eye image to spot the temperature difference (function defined as Variance(img)). Once the variance values are obtained, we will feed it to the SVM model for prediction.

```
def variance_method(self,image): # #5
    def Variance(img):
        """
        Finding the variance of the whole image.

        Arguments:
                img    - eye input image

        Returns:
                var    - Variance of the whole image

        """
        gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        var = ndimage.variance(gray_img)
        return var

    def prediction(img):
        variance = Variance(img)
        pred = self.model5.predict(np.array(variance).reshape(-1, 1))
        return pred

    return prediction(image)
```

Fig - Code snippet of Variance model

5th method: Training a model on the DCT coefficients of the face images.

Sometimes there are differences in two images that cannot be seen or visual identified. Through deep research, it was found that low frequency features of the image (removing the high frequency features) can make better drunk predictions. The DWT-DCT method helps to transform and extract the low frequency features from the image which we then used to predict if a person is Drunk or Sober. For DWT-DCT model, Discrete Wavelet Transform and Discrete cosine Transform are applied on the image before feeding to the SVM model. DWT is a decomposition removing some of the major high frequency features of the image. We apply 2 level decomposition for better extraction. After this, DCT is applied which gives us the 6 major cosine wave coefficients of the image. These coefficients are DCT coefficients that can be used to remake images similar to the original one. Now these 6 coefficients are fed into the SVM model for prediction.

After the proper training of all the models, a simple mathematical formula is used to get a joined prediction. Weights were assigned to all the model results biased

according to their implementation and theoretical accuracies. The last prediction is calculated as the sum of the product of predictions to their respective weights. For this, we have predefined functions to call for individual models. We call them to get their outputs which are then multiplied with their respective weights for the final prediction.

```
def DWT(img):
    """
    Applying Discrete wavelet transformation.
    Arguments:
            img    - face input image
    Returns:
            cA2    - transformed image.
    """
    #converting to gray_scale image
    gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    #transformation level1
    coeffs1 = pywt.dwt2(gray_img, 'db3', mode='periodization')
    cA1, (cH1, cV1, cD1) = coeffs1
    #transformation level2
    coeffs2 = pywt.dwt2(cA1, 'db3', mode='periodization')
    cA2, (cH2, cV2, cD2) = coeffs2
    return cA2
def DCT(img):
    """
    Applying DC transformation.
    Arguments:
            img    - input image
    Returns:
            c_list   - list of first 6 coefficients obtained by Z-scanning.
    """
    dct_cof = dct(dct(img.T, norm='ortho').T, norm='ortho')
    c_list=[]
    c_list.append(dct_cof[0][0])
    c_list.append(dct_cof[1][0])
    c_list.append(dct_cof[0][1])
    c_list.append(dct_cof[0][2])
    c_list.append(dct_cof[1][1])
    c_list.append(dct_cof[2][0])
    return c_list
```

Fig - Code snippet of dwt dct algorithms.

Results

After testing all the five models the following accuracies were obtained for the 5 drunk analysis model:

Percentage prediction accuracy: number of people out of a group of 100 people who will be correctly classified

Models	**Prediction Accuracy**
Variance model	58.3 %
DWT and DCT model	83.33%
Blood Vessel model	51%
Face model	50%
Forehead model	52%

```
#Estimation on the basis of extracted parts whether a person is drunk or not
out1 = B.forehead_model(forehead)  # 0.175
out2 = B.face_model(face)          #0.168
out3 = B.blood_vessel(face)         #0.1882
out4 = B.DWT_DCT(face)              #0.2799
if leye is not None:
    out5 = B.varience_method(leye)      #0.195
elif reye is not None:
    out5 = B.varience_method(reye)
else:
    print('Eyes not good')
    out5 = 0

#assigning weights to each of the outputs, to get higher accuraccy
final_out = out1*0.175 + out2*0.168 + 0.1882*out3 + 0.2799*out4 + 0.195*out5
thresh = 0.67
```

Fig - calling all five models.

Conclusion

Based on the research done on all the procedures the results obtained were better than expected. After applying openpose and getting all the ROI's as needed, when fed to the drunk and sober model(combining all the five models under one accuracy biased algorithm) makes the model more strong instead of just trusting one.