'ScreenTime': Calculating On-Screen Time of Actors

Abdulqader Saafan & Vincent Nafrada
Dr. Sanja Fidler
Sayyed Nezhadi

April 13, 2019

Contents

1 Introduction

This section will discuss our project, ScreenTime and its objective. Firstly, the objective of our system is to learn the on-screen time of actors in a video clip. From the screen-time of each actor, we can calculate the percentage of how much an actor appeared. Furthermore, we are able to predict the gender of each actor. This gender classification allows us to discover the ratio of female to male actors. The system is ultimately able to give a percentage of on-screen time for each actor and each gender.

This report will first discuss the system design of ScreenTime. The system design will be discussed at a high-level to give an overview, and then in depth for each component of the system. Discussion of each component will include motivation, technical details, advantages and disadvantages, as well as available alternatives. The report will also discuss the performance of our design. This discussion is focused on what our design does well, and what we can improve. Improvements and future research is one of the final sections for our product. At the end of the report, we have also included a task breakdown to show how each member contributed, and all the works we referenced.

Please also note that the project is available publicly on Github. However, the repository is still under development and is not cleaned up yet for a release.

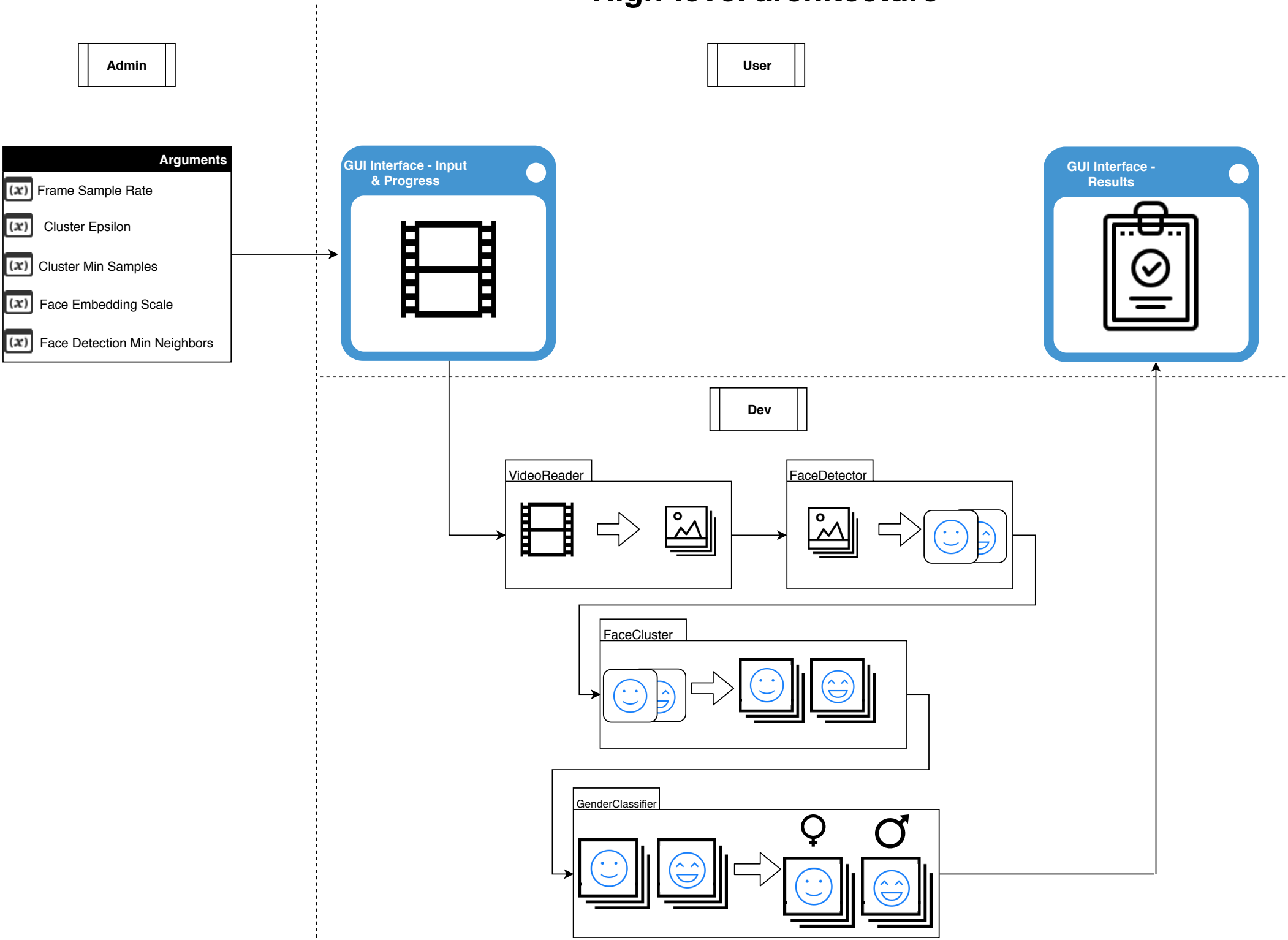
2 System Design

2.1 High-level architecture

The figure on the following page titled, "High-level architecture" shows the main work-flow of our system. Explanation of the figure follows in the page after the figure.

The two dotted lines on the diagram represent division of the system. The two division lines make up three regions, Admin, User, and Dev. Each region represents the intended target audience for that portion of the system. We will explain each region in terms of the components inside the region, as well as its use case.

High-level architecture



2.1.1 Admin

The admin region holds one component on the diagram, Arguments. These arguments are used on the startup of the system (GUI Interface). An admin of our system is responsible for setting these arguments to best compute the desired results.

Frame Sample Rate: This argument should be an integer between 1 and the frame rate of the video (Most usually 30). The sample rate will affect how many frames per second we analyze in the video. A higher sample rate will lead to greater accuracy of the system but it will also lead to a heavier computational workload. More specifically, a higher sample rate leads to more frames that run through the face detector, which leads to more faces in the face cluster and more faces to classify in the gender classifier.

Face Detection Min Neighbors: This argument is used in the FaceDetector class as an argument to the detector. This value checks the minimum number of neighbouring facial features that need to be found to indicate the detection of a face by the classifier. This value affects the amount of false positives detected. More specifically, relatively, the lower the value causes an increase in false positives and vice-versa.

Face Embedding Scale: This argument is used in the FaceCluster class while converting 3D (RGB) images of faces to 128-d vectors. This process is called embedding and is discussed later in this report. The argument here is used when converting an image to a blob object. The scale argument will scale that image with that factor as it converts into the new blob object which can then be used as an input to a network.

Cluster Epsilon: This argument is used in the FaceCluster class on the DBSCAN algorithm. The epsilon tells DBSCAN how far large the radius (Euclidean distance) is for a point's neighbourhood. This is important since it changes how we cluster our points. A larger epsilon means fewer and larger clusters since each point has a large neighbourhood. Conversely, a smaller epsilon leads to smaller and more frequent clusters. Epsilon also affects our noise levels since a large epsilon might incorrectly capture noise as part of a cluster and a small epsilon might incorrectly classify points as noise.

Cluster Min Samples: This argument is used in the FaceCluster class on the DBSCAN algorithm. The min samples variable tells DBSCAN how many points should be in the neighbourhood of a point p, for p to be a core point. DBSCAN is explained more in depth later in this report. A larger min sample means that we will have fewer core points and possibly more smaller clusters since they will not connect as much. A smaller min sample can lead to more core points, fewer and larger clusters as the clusters will connect together.

The responsibility of the admin in this layer is to properly set up the system for a user. The arguments are mainly interconnected and affect each other. For instance, the min sample argument should be set with the sample rate and epsilon in mind. If the sample rate is small, there might not be a great number of points which will lead to no clusters being formed. Additionally, a small epsilon coupled with a large min sample will also lead to no clusters being formed. The admin should be able to set these arguments so that the user is not concerned with them at all. This leads to the second region in our system design, the top right region, user.

2.1.2 User

The user region represents the components that the user interacts with. In this case, there is only one component, the GUI interface. The motivation here is to not burden the user with setting arguments and knowing about the technical details of the design. To make this possible, we have spent effort on creating a simple GUI that takes the user through the steps required to run the system.

The GUI is built in python with the TkInter library in python. This is one of the standard libraries used for building a UI. The GUI runs on the main thread to stay interactive for the user. During the computation portion of the system, we launch a thread to execute the computations. This thread communicates with the UI main thread to update the progress. After the computation is done, the thread is destroyed and only the main thread exists for interacting with the results.

The first thing the user is presented with is the main screen seen in figure 1 below. The screen that welcomes the user has only two things. An instructions label that directs the user to upload a video and a "browse" button that prompts their OS's file selection utility.

After the user has selected a video, the system will begin its computations. A screen with a label that reads, "Setting up", might show up for the user. This happens as the program is organizing the input and arguments and

Figure 1: First screen in ScreenTime Application

before it begins execution and only appears shortly. After the setup, the progress screen is shown. There is a progress bar that allows the user to keep track of the program's execution. There is also an indication of how many frames the program is running on. Brackets show how many frames have been processed and how many in total there is. This view is shown in figure 2.

Figure 2: Progress screen in ScreenTime Application. This is shown when the program is detecting faces in the video.

After the computation is finished, the user is shown a new screen. This screen informs the user that the program has finished running and that the results are ready. The user simply clicks the "Show Results" button to see the results of the program. This is seen in figure 3.

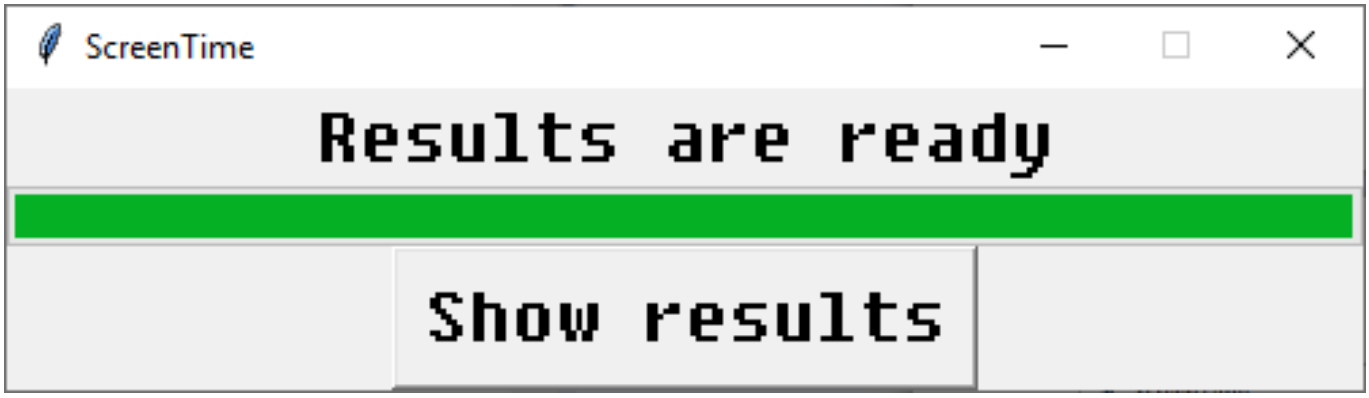


Figure 3: Results ready screen. This is shown when the program has finished computing and has the results.

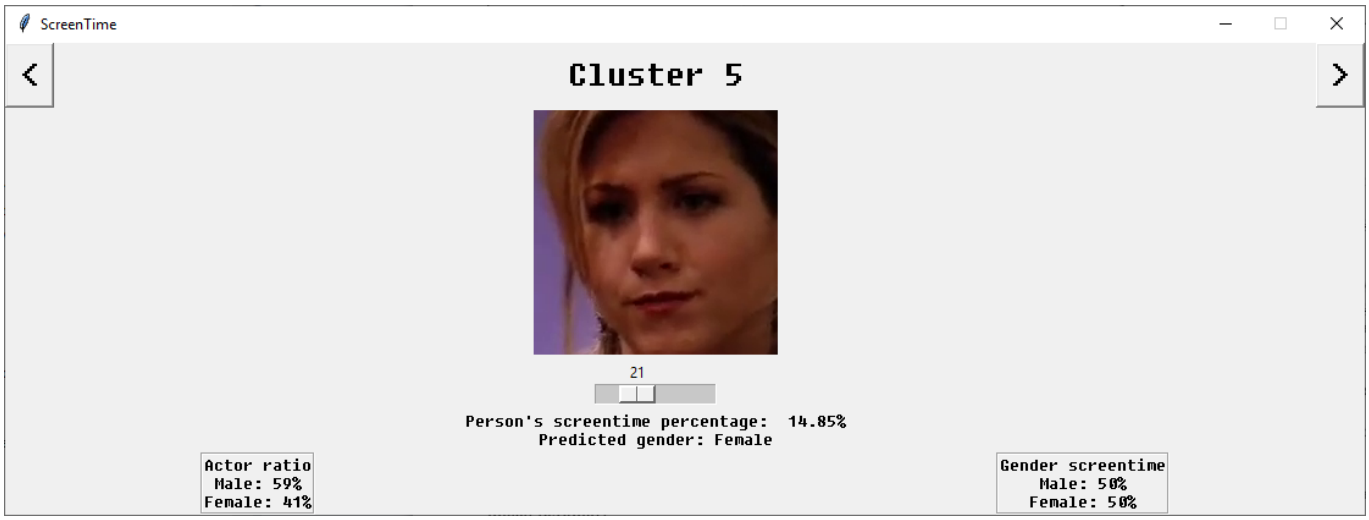


Figure 4: Results screen. This screen allows the user to interact with the results.

The next screen, as shown in figure 4, has multiple interactive elements for the user. The results screen shows the results of the program and allows the user to understand the results. The window has two buttons at the top corners. These are used to navigate through the clusters (or persons) that the program detected. The label in the top middle displays the current cluster the user is on. In the middle of the screen, there is an image and a slider.

The slider is used to navigate through the detected faces in that cluster. Sliding the slider updates the image in the middle. Just below the slider, there is information for that cluster which includes the predicted gender of the person and their screen-time in the video (in percentage). At the bottom of the window is two boxes that contain video-wide information. The left box contains the male to female actor ratio, i.e the percentage of male and female actors in the video. The right box contains information about how much screen-time each gender has in the video.

The GUI allows the user to keep track of the program execution. This is important since larger videos or/and higher sample rates can lead to a long running time of the program. The other important component of the GUI is the results screen. This screen neatly organizes the results and presents the user an easy way to read the results. The results screen presents cluster information as well as video information.

2.1.3 Dev

The region in the bottom right of the system design diagram illustrates the major components used in the backend of the project. These components are crucial to understanding the technical implementation of the system. The developers' responsibility here includes writing the code for the components and allowing them to work together. This includes a mixture of built-in or pre-trained algorithms and networks as well as new code created by the developers. The developers also have a responsibility to setting lower level variables that define how the system operates. The next section will go further into the technical architecture of each component.

2.2 In-Depth architecture: Video Reader

The VideoReader class is the first component that interacts with the user inputted video. The class uses OpenCV python libraries to open the video file and read it. The class extracts meta-data such as the frame rate of the video and the total frames in the video. The class also takes the sample rate argument and based on this, chooses which frames to extract. The class has a method that returns the next frame (if there is one). The function getNextFrame depends on the sample rate inputted and uses the cv2 capture object to skip past frames we don't want to look at.

```
1  def getNextFrame(self):
2          # Skip to next frame we need
3          for i in range(int(self.frameRate/self.sampleRate) - 1):
4              self.cap.read()
5
6          ret, frame = self.cap.read()
7
8          if(ret):
9              return frame
10
11         else:
12             return []
```

2.3 In-Depth architecture: Face detection

The method used for face detection is the Viola-Jones detection algorithm. The architecture utilized for the algorithm was OpenCV's built-in library coupled with their Haar cascade classifiers. It is important to note that a Haar cascade classifier has been pre-trained which consists of many stages of filters that classify a portion of an image is a face.

Essentially, the Viola-Jones algorithm accepts an image and creates a multi-scale representation of that one image, called an image pyramid. This is because the face detection needs to be adaptive to scale (capable of capturing large and small faces in a window). Similarly to convolution windows, the algorithm utilizes a detection window which slides from the top-left of an image to the bottom-right, covering all pixels of an image. During each shift of the sliding window, the sums of the pixels are computed (integral image). And that image within the window will undergo a cascade classifier. As stated before a cascade classifier has many stages; the image within the window has to pass every single stage consisting of Haar filters. If the image passes a threshold for a stage, it proceeds to the next one. If it passes all stages, the image in the window is classified as a face, and if it fails one stage, it is not classified as a face (Paul Viola and Michael Jones, 2001).

The inputs of the Face Detector class are cascade classifiers, scale factor and minimum neighbours (manipulates image pyramid).

```
def detectFaces(self):
        img = cv2.cvtColor(self.frame, cv2.COLOR_BGR2GRAY)
        faces = self.faceCascade.detectMultiScale(img,
                                scaleFactor=1.1, minNeighbors= 4)
        return faces
```

The given Haar cascade classifiers are for frontal faces and single eye angles. The reason is that most camera angles of faces in film are frontal. The single eye classifier was used to eliminate noise. Initially, profile face (side angle) classifiers were used however that detected more noise. Additionally, with the use of only a frontal face classifier, some noise was detected even with fine-tuned scale and minimum neighbors parameters. Thus, a face was only a candidate if it contained a frontal face and/or a single eye match.

The Viola-Jones algorithm was the best candidate for this project due to its efficiency and accuracy. The essential step of calculating the integral image stated above is efficient. For instance, for an image of P pixels and N regions of interest each covers W pixels, the naive algorithm has a complexity of (NxW), while the integral image-based approach has a complexity of $(P + 4N)$. In the case of face detection, a sliding window shifts around the image, which needs to sum up pixels for each shifted window. Therefore, N is approximately equal to P. The integral image approach reduces the complexity from (PxW) to $(P + 4P)$, which is two orders of magnitude reduction for a sliding window of size $10x10$ (Viola-Jones Face Detection 5KK73, 2012). Additionally, OpenCV had a diverse collection of Haar cascade classifiers available for use.

2.4 In-Depth architecture: DBSCAN Clustering

The FaceCluster component is responsible for clustering the faces detected using the detection class. The class first takes in all the faces as 3D (RGB) images and converts them to a 128-d vectors. The conversion happens through a pre-trained network. The network is trained using the triplets loss method. (Schroff, 2015) The method takes as input 3 images, 2 of the images are of a matching face/person and the third is of a different person. The network is then trained on creating 128-d vector for each image with the goal of maximizing the distance between the vectors of the non-matching photos while minimizing the distance between the vectors of the matching photos. This means that when we use this trained network, we can expect 2 images of similar faces to have a small distance between their respective 128-d vectors. The actual content in the 128-d vector does not matter to us. We only care about the relation of the vectors respective to the faces they represent. The addFaces method in the cluster class takes in faces (as 3D images), converts them into a blob that the network can take in, and then saves the output of the network (128-d vector) in the class.

```
def addFaces(self, faces):
        # Add to cluster
        for face in faces:
            # Creating a blob to pass into the network
            faceBlob = cv2.dnn.blobFromImage(cv2.resize(face, (96,96)),
                                             1.0/255, # Scale factor
                                             (96, 96)) # Spatial size

            self.embedder.setInput(faceBlob)

            self.actualFaces.append(face)
            self.faceVectors.append(self.embedder.forward()[0])
```

The clustering method used is DBSCAN (Ester, 1996). The reason for choosing this method is that we don't want to use other clustering methods which require an indication of how many clusters there will be. Choosing something like K-means forces us to come up with a K. Since we want the program to be as flexible as possible in terms of input, we can not choose K, unless we manually do so. The other advantage of DBSCAN is that it uses the distance between points to identify the clusters. This works well in our case since the 128-d vectors that we will cluster are optimized to have smaller distances when matching. This means we can pass in the vectors we have from the previous step directly into the DBSCAN method. DBSCAN was implemented in our code and not used from the OpenCV library.

The DBSCAN algorithm is shown below and explained with function descriptions and code comments. The class takes in a set of points that we will perform the clustering on, epsilon, and min samples. Epsilon describes how far the neighbourhood radius of each point extends. The min samples variable tells us how many points must be in a point's p neighbourhood to consider p a core point. The algorithm is as such:

1. Pick a random point in the cluster p
2. Check the number of points in the neighbourhood of p

(a) For every other point in the cluster

(b) Calculate the Euclidean distance between p and that other point

- If the distance is <= epsilon: count the point as a neighbour
- Else, move on to the next point

3. If the amount of points in the neighbourhood of p is $<$ minSamples: mark p as $NOISE$ and go to the next random unclassified point p
4. If the amount of points in the neighbourhood of p is >= minSamples: mark p as $CORE$ by assigning it a cluster number and expand the cluster. Then move on to the next random unclassified point p
   - To expand the cluster, visit all the neighbours of p and classify them
   - If the neighbour is a core point, then add the neighbours of that neighbour to the cluster and go through them to classify them. Core points are always included in the cluster and their neighbours visited to see if they are also core points.
   - If the neighbour is already marked as a $NOISE$ point then remark it as part of the cluster. This constitutes a border point.
   - Continue until there are more neighbours to visit
5. When all points are marked (either $NOISE$ or as part of a cluster) DBSCAN has finished

The idea of DBSCAN is to cluster points based on their spatial density (DBSCAN = Density Based Spatial Clustering of Applications with Noise). The arguments minSamples and epsilon affect the clustering as described earlier. After we have clustered based on the vectors we have, we can use the labels variable in the DBSCAN class to identify which faces (and their respective image) belong to what cluster. This information is passed back to the FaceCluster class which holds it. The GUI interacts with the FaceCluster class to show the results. The GenderClassifier class also interacts with the FaceCluster class as we will describe in the next section.

```
import numpy as np


'''
This class is used to run the DBSCAN algorithm
This is based on the following paper:
https://www.aaai.org/Papers/KDD/1996/KDD96-037.pdf
'''
class Dbscan():
    def __init__(self, eps, minSamples):
        self.__eps = eps
        self.__minSamples = minSamples
        self.labels = []
        self.__cId = 0


    '''
    This is the main function that is used in this class.
    This clusters the points provided based on the class globals such as
    eps and minSamples
    '''
    def fit(self, points):
        # Labels that track class of each point and a moving class id
        # that increments for new clusters (and is -1 for noise)
        self.labels = [None] * len(points)

        # Visit all points
        for pt in range(len(points)):

            if(self.labels[pt] != None):
                # Already classified
                continue

            nbrs = self.__getNeighbourPoints(points, pt)

            # Check if we found a core, border or noise point
            if(len(nbrs) < self.__minSamples):
                # Noise point, or possibly a border point
                self.labels[pt] = -1
            elif(len(nbrs) >= self.__minSamples):
                # p is a core point and the cluster is valid
                # we need to know the extent of the cluster
                self.__expandCluster(points, pt, nbrs)
                self.__cId += 1

        self.labels = np.array(self.labels)

        return self.labels

    '''
    This function is used to get the full extent of a cluster
    and is called on a point we know to be a core point
    '''
    def __expandCluster(self, points, pCoreIndex, nbrs):
        # Core is part of cluster
        self.labels[pCoreIndex] = self.__cId


        while(nbrs != []):
            # We visit whatever is left
            visitPointIndex = nbrs[0]

```

```
63              # If it's noise, it is now a border point
64              if(self.labels[visitPointIndex] == -1):
65                  self.labels[visitPointIndex] = self.__cId
66
67              # If we haven't checked it before, we can check if it's core
68              elif(self.labels[visitPointIndex] == None):
69                  # If or if it is not core, it is still part of cluster so we add it
70                  self.labels[visitPointIndex] = self.__cId
71
72                  visitPointNbrs = self.__getNeighbourPoints(points, visitPointIndex)
73
74                  # If it has enough neighbours, it is core
75                  if(len(visitPointNbrs) >= self.__minSamples):
76                      # So we visit all its neigbhours
77                      nbrs += visitPointNbrs
78
79              # We're done with this point now so take it out of loop
80              nbrs = nbrs[1:]
81
82
83      '''
84      Calculates the distance between two points based on euclidean function
85      '''
86      def __calculateDistance(self, p1, p2):
87          return np.linalg.norm(p1-p2)
88
89      '''
90      This function gets all the (indexes of) neighbours of a point, pCore
91      A point is a neighbour of pCore if it is
92      1. In allP (all points to consider) and
93      2. Has a euclidean distance less than eps
94      '''
95      def __getNeighbourPoints(self, allP, pCoreIndex):
96          c = []
97          for n in range(len(allP)):
98              dist = self.__calculateDistance(allP[pCoreIndex], allP[n])
99
100             # If point is a neighbour (based on eps) we add index to cluster
101             if(dist <= self.__eps):
102                 c.append(n)
103         return c
```

2.5 In-Depth architecture: Gender classification

2.5.1 Training the Convolutional Neural Network

The Convolutional Neural Network was trained for image classification based on gender. The network trained has 6 layers. Starting with a convolution 2d input layer, a (5,5) kernel, with ReLU applied to the output of every convolutional and fully-connected layer. The reason ReLu is chosen as opposed to tanh or sigmoid is that it is more simple and efficient, however, should only be used in hidden layers. Also, deep convolutional neural networks with ReLUs train several times faster than their equivalents with tanh units(Alex Krizhevsky et.al., 2012). Additionally, Since the data set has images dimensions of (250,250) a kernel layer of (5x5) was used to help learn larger spatial filters and to help reduce volume size (Adrian Rosebrock, 2018). And max pooling is added to help with overfitting. It reduces dimensions of the layer and the number of things to learn since it is downsized, hence extract the most prominent features. Below are the first three layers.

```
model.add(Conv2D(32, (3, 3), input_shape=input_shape))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
```

Dropout is a regularization technique, which aims to reduce the complexity of the model with the goal to prevent overfitting. A dropout layer removes a certain amount of nodes in the network. The nodes become more insensitive to the weights of the other nodes, and therefore the model is more robust. Essentially, the model is not reliant on particular nodes that have more valuable learning but instead the overall model is more resilient to noise. A rate of 0.5 was chosen for the dropout, hence randomly removing half of the neurons. We chose the loss function based on the number of classification categories. Since we want to do a two category classification (male and female), we used Dense(1, activation='sigmoid', name='output') as the output layer, and will compile the model using binary-cross entropy loss function. Binary-cross entropy loss sets up a binary classification problem between two classes, which suits the project objective. For the output layer, a sigmoid activation was used. It takes a real-valued number and puts it into a range between 0 and 1. In particular, large negative numbers become 0 and large positive numbers become 1, thus sigmoid is suitable for gender classification. (Walia, 2017)

```
model.add(Dense(64))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(1))
model.add(Activation('sigmoid'))
```

This is the augmentation configuration we will use for training. We need to make minor alterations to our existing dataset. Data augmentation is a way we can reduce overfitting on models, where we increase the amount of training data using the information only in our training data (Chollet, 2016).

```
train_datagen = ImageDataGenerator(
    rescale=1. / 255,
    shear_range=0.1,
    zoom_range=0.2,
    horizontal_flip=True)
```

Finally, the optimal trained model was done in 50 epochs with a batch size of 10. The training data provided to us was used, with 475 images being the training sample, 136 for the validation sample and the rest was the test sample (70%, 20%, 10% split). The run-time was relatively fast, therefore no major algorithm or input changes were necessary to improve it.

With these parameters, the training accuracy was 88% and a validation accuracy of 84%. Using the test samples, the classifier classified the correct gender with a 80% testing accuracy. We did trials with 10-20 epochs with the same batch size, and it was underfitting. Additionally, decreasing the batch size and doing 50 or more epochs overfit the model by a big margin.

2.5.2 Classifying faces

Using the trained CNN, the image clusters that were computed will be verified. This step takes n random sample images from the clusters and with the CNN model, it predicts the classification of those images. The amount of random samples, n, is based on a ratio that we set in the class. For example, a ratio of 0.5 (50%) will choose half the cluster randomly to classify. Since the classification is binary, the average of the results is taken and rounded down or up, specifying if the cluster whole cluster is more likely to be male or female. This verifies the accuracy of the cluster but also exposes the noise and miscalculations too, which was used to revise the hyperparameters in order to fine-tune the results.

3 Performance

3.1 Ideal cases

In terms of input, ideally, videos with most of the frames having faces positioned in a frontal angle are optimal. Based on our current configuration, the Haar cascade classifier detects more faces with a higher degree of accuracy with frontal faces in comparison to profile faces. In turn, the resulting clusters become more accurate since the majority of the detected faces are similar. The side profile of some actors does not get clustered with the rest of the actor's face since the image is different and the vector has a large distance difference.

Furthermore, minimal movement in the video allows for more detection. With movement comes with motion blur which reduces detected faces between frames, since the Haar cascade classifiers are not trained on the capturing features altered due to motion blur. This could be improved as well with face tracking as is discussed in the future research section below.

For our neural net, ideally, it should be trained around 50 epochs with a batch size of 10. Since the training sample size is fairly small, training with more epochs would risk overfitting the model, and training less would risk underfitting.

3.2 Mediocre cases

Videos with frames having faces slightly covered by something decreases the accuracy of face detection. Simply put, having a small feature being covered is significantly detrimental to our face detection approach. For instance, a frame of an actor drinking from a glass and covering their mouth is enough to deter that face from being detected. Hence, reducing the overall screen time of that actor, meaning less accurate results.

Gender classification accuracy suffers when a cluster has numerous frames with a profile angle. This is due to mostly the training data, there needs to be an extensive amount of profile samples that are appropriate enough to be trained on with our neural network. The features of a specific gender are not prominent enough to classify the detected faces. Hence, the inaccuracy of classifying the whole cluster.

4 Future research and improvements

This section will discuss future improvements we can make to the system, as well as areas we need to investigate further. Due to the time constraint of the project, many areas still were not visited or researched thoroughly. We focused on the areas that would create our main product and that will motivate further work.

4.1 Face tracking

Since we were constraint by time, we could only use face detection and did not have the resources to implement face tracking. As discussed in the performance section, our system does not work well when actors in the video turn or face away from the camera even for small periods of time since we will sometimes (depending on the sample rate) miss their presence. Implementing face tracking could improve the accuracy of our face detection and yield better results.

4.2 Server support for face detection

For larger videos and/or higher sample rates, we will need to process a great number of frames. An improvement to this process would be a better CPU on the user machine or conversion to GPU code. However, even the GPU code will require the user to have a good GPU (most likely a Nvidia GPU due to library support). Since we don't want to expect this out of our users, having a cloud-connected backend would greatly improve the performance and user experience. Face detection (and possibly tracking) can be offloaded to a dedicated server that can compute the results. The server would also take care of the rest of the flow and report back the results to the user.

4.3 Targeted training

Training on a more extensive and particular data set with actors would improve the results of this project. Since this project is designed towards the screen time of actors, it is more suitable to train the neural net using a data set from IMDB for instance. Therefore, the range of features learned will be more appropriate for the objective of the project. This applies for both the clustering face-to-vector network and the gender network.

We will also want to research the possibility of training the vector network using the triplet loss method to recognize side profiles of the same actor. This would increase the accuracy of our clustering algorithm and better cluster actors. Our initial thoughts are that this is a challenging feat since we would most likely need to create our own training set and train the model. This adds resource challenges and would likely cost a lot of time.

4.4 Pixel categorization/segmentation

A possible improvement for clustering and gender classification is to expand the bounding box of a detected face. More specifically, once a face has been detected, the relevant surrounding pixels such as their hair or ears. But the bounding box has to be particular with the background and noise that might be included, thus the expansion of the box has to be dynamic. Possibly labelling pixels initially as background and person. This would improve the clustering of actors since there are more features (hair, accessories, whole head) that can be used to match and organize the clusters. This, in turn, will improve gender classification for the same reason.

4.5 UI improvements

We spend a lot of effort building out the GUI knowing that it would be much easier to do a python script. The rationale is that we want a product that a normal everyday user can interact with. However, due to the resource and time constraints, we were not able to fully realize our vision for a friendly UI. In the future, we would want to work on a more appealing UI and an effortless UX. This would require conducting user studies and further researching python UI libraries. This research could even lead to creating a different front end (such as Angular) and using python in the back end for the computation.

5 Conclusion

Overall, this project has caused us to explore many unknown areas for us as developers. This system covers multiple areas of computer vision from face detection to clustering and classification. Our initial design was clarified as we further researched the areas of interest. We are proud of the system we have ended up with but, we know there is a lot of room for improvement. We believe that our focus on usability, as well as our combination of technologies, will set us apart from other systems and projects. Furthermore, we believe our attention to code cleanliness and structure will make our work accessible to other developers and researchers. In the future, we would be excited to explore multiple areas of improvement to better the system.

6 Task Breakdown

Both developers have learned and worked on every area in the project. However, different areas were led by different developers to better focus our resources. Both developers worked on this report equally, focusing on writing up the sections that they worked with the most.

Abdulqader (Abode) Saafan focused on and led the design of the GUI, the vector network, DBSCAN clustering, and aggregating results. Vincent Nafrada focused on and led Viola-Jones face detection, training and use of the gender classifier and fine-tuning built-in and pre-trained models.

7 Citations

REFERENCES

[1] About Keras layers. (n.d.). Retrieved from https://keras.io/layers/about-keras-layers/

[2] An Introduction to Tkinter (Work in Progress). (n.d.). Retrieved from http://effbot.org/tkinterbook/

[3] Chollet, F. (2016, June). The Keras Blog. Retrieved from https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html

[4] Cmusatyalab. (2018, June 17). Cmusatyalab/openface. Retrieved from https://github.com/cmusatyalab/openface

[5] Ester, M., Kriegel, H., Sander, J., & Xu, X. (n.d.). A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. Retrieved from https://www.aaai.org/Papers/KDD/1996/KDD96-037.pdf

[6] Face clustering with Python. (2019, February 04). Retrieved from https://www.pyimagesearch.com/2018/07/09/face-clustering-with-python/

[7] Goodfellow, I., Bengio, Y., & Courville, A. (2016). Regularization For Deep Learning. Retrieved from http://www.deeplearningbook.org/contents/regularization.html

[8] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (n.d.). ImageNet Classification with Deep Convolutional Neural Networks. Retrieved from http://www.cs.toronto.edu/ fritz/absps/imagenet.pdf

[9] NumPy v1.16 Manual. (n.d.). Retrieved from https://docs.scipy.org/doc/numpy/

[10] OpenCV Face Recognition. (2019, February 04). Retrieved from https://www.pyimagesearch.com/2018/09/24/opencv-face-recognition/

[11] Rosebrock, A. (2018, December 31). Keras Conv2D and Convolutional Layers. Retrieved from https://www.pyimagesearch.com/2018/12/31/keras-conv2d-and-convolutional-layers/

[12] Schroff, F., Kalenichenko, D., & Philbin, J. (2015). FaceNet: A unified embedding for face recognition and clustering. 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). doi:10.1109/cvpr.2015.7298682

[13] The Python Standard Library. (n.d.). Retrieved from https://docs.python.org/3/library/

[14] Viola, P., & Jones, M. (2001). Rapid Object Detection using a Boosted Cascade of Simple Features. Retrieved from https://www.cs.cmu.edu/ efros/courses/LBMV07/Papers/viola-cvpr-01.pdf.

[15] Viola-Jones Face Detection - 5KK73 GPU Assignment. (2012). Retrieved from https://sites.google.com/site/5kk73gpu2012/assignment/viola-jones-face-detection

[16] Walia, A. S. (2017, May 29). Activation functions and it's types-Which is better? Retrieved from https://towardsdatascience.com/activation-functions-and-its-types-which-is-better-a9a5310cc8f

[17] Welcome to OpenCV-Python Tutorials's documentation!. (n.d.). Retrieved from https://opencv-python-tutroals.readthedocs.io/en/latest/