Large Scale Data Engineering

Group 14: Face-Join

Azim Afroozeh
Student number: 2639408
VunetID: aah770
a.afroozeh@student.vu.nl

Ali Reza Farid Amin
Student number: 2611785
VunetID: afn530
alireza@student.uva.nl

Paul Klaassens
Student number: 2655141
VunetID: pks215
p.klaassens@student.vu.nl

ABSTRACT

The well known website Flickr hosts a staggering amount of pictures. We were supplied with a dataset which contained links and meta data of 100 million of these pictures. By crawling through these pictures and using face recognition software we identified faces in these images and matched them to a variety of famous people. These results are displayed on a website showing a ranking of the matches made on Flickr sorted by their confidence level.

1. BACKGROUND

The art of facial recognition has been researched since the 1970s but has in the recent years seen great improvement in both accuracy and speed. Face recognition software used to be dominated by private industry and government large scale data sets. However recently open source facial recognition software has been showing improvements. In our case we looked at OpenFace [1]. OpenFace has achieved 92.9% accuracy on the widely used LFW benchmark. This is near human accuracy and comes close to the accuracy of the best private industry models.

In our case not only accuracy but speed is also very important because of the large scale of the task at hand. OpenFace 2.0 halves the execution time by using more efficient pre-processing, image alignment and smaller models. In earlier facial recognition software the focus was on deriving a low dimensional face representation by using the ratios of the distances, areas and angels of the face. However this approach proved not to be accurate. Later research focused on using statistics and AI to learn from the data and perform well from on the data set. Using the correlations a Principal Component Analysis [4] was performed to extract the 'eigenfaces' which were then used as landmarks.

Currently, the top performing facial recognition software is in the hands of private corporations and the government. Two prime examples are FaceNet [3] by Google and DeepFace [5] by Facebook. Both use convolutional neural networks which are trained using private data sets containing millions of pictures.

Artificial neural networks are systems that learn, in this case to recognize a face, without being programmed with specific rules. A neural network consists of neurons arranged in a series of layers. The input layer receives the initial data or information that the network will learn about.The output layer signals how the network responds to the data. Between the input and output layer there are one or multiple hidden layers which make up the core of the neural network.

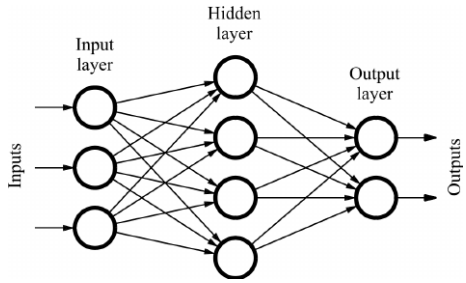

Figure 1: A simple neural network

An example of a neural network can be seen in figure 1.
In a feedforward neural network, the network consists of many function compositions and a loss function $\mathcal{L}$ which determines how well the network models the data. Training the model is then an optimization problem, where one seeks to find the specific set of parameters that minimize the loss function $\mathcal{L}$.

Before we can input the pictures into the neural network they will first have to be prepossessed. A face detector is run on each image and a bounding box for each face is generated. These faces are then resized to the specifics of a neural network and can be used as input data. This could give issues as faces of the same person can be looking in different directions or are exposed to different lighting settings. FaceNet doesn't use further transformation and can overcome these problems by using a large dataset. In contrast OpenFace uses affine transformation to align the pictures in such a way that the eyes, nose and mouth are at similar locations for each picture. Because the input images are normalized this shrinks the input space and improves the training time. Another modern technique which is employed is modelling the image to a 3D model so that it will seem it looks directly at the camera.

1.1 Triplet loss function

Originally face recognition neural networks used a classification layer trained over a set of known images and then took an intermediate layer as a representation to generalize the facial recognition beyond the set of known images. As this is both inefficient FaceNet introduced a way to directly train the output to a m-dimensional embedding by using a triplet loss function which was based on the work of Weinberger et al (2009)[6]. A triplet consists of two matching and

one not matching face image and the triplet loss function tries to separate the matching faces from the not matching face by using a distance margin. Determining which triplets are chosen have shown to have a large impact on the performance of the network. The triplet loss function is adopted by Openface and is used to classify the images on a 128 dimension unit hyper sphere.

2. DATASET

The dataset which we were supplied with consists of ten csv files compressed in BZIP2 format. Each of the files consists approximately 10 million records. Each record contains a link to a Flickr image and some descriptive data, see table 1.

Table 1: Textual information for each record

Identifier	Date taken	User NSID
Title	Date uploaded	User Tags
Description	Longitude / Latitude	User Nickname
Page Url	Photo/video marker	Capture device

As a result the entire dataset spanning all ten files gives us 100 million records to work with. With an average image size of approximately 112 kilobytes the complete set of downloaded images amounts to 11.200 gigabytes. However, a large portion of these images were corrupt, of invalid format or didn't contain a face. Approximately 18,8% of the images contained a face which could be aligned and inserted into our trained model.

3. RESEARCH QUESTIONS

To achieve the projected goal of crawling through the entire Flickr image archive and matching the faces to known identities the following research questions should be answered:

- Which facial recognition algorithm is sufficient in both speed and accuracy to get accurate results?
- Because of the large amount of unknown people in Flickr images should we also include a lot of unknowns in our training dataset?
- After a model is trained how do we efficiently parallelize the interference on the 100 million Flickr images.

3.1 Project setup

To process such a huge amount of pictures requires a lot of bandwidth, computing power and time. The general approach is training a model on a small set of known identities using a GPU accelerated machine to reduce training times. Testing the model on a sample of the Flickr images and evaluating results. Then improve the model, either by changing the known identities (training data) or the classifier. After we were satisfied with a trained model which was sufficient in both speed and accuracy we can deploy the model. This would have to be done in a way that the task can be parallelized on a cluster.

3.2 OpenFace

The library that we used for the face recognition is the aforementioned OpenFace. The preprocessing is performed using dlib's pretrained face detector. By using 68 landmarks

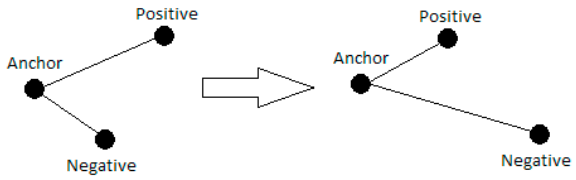

Figure 2: Triplet loss

of a face an affine transformation is performed which generates a face with the eyes and nose on similar locations for each face. The picture size that we used as input to the neural network is 96 x 96 pixels.
OpenFace uses a variation of FaceNet's nn4 network. FaceNet's approach is to create an embedding $f(x)$ from an image x onto a m-dimensional space: $\mathbb{R}^m$ which lives on an m-dimensional hypersphere: $\|f(x)_2\| = 1$. This is done in such a way that the euclidean distance between faces of the same identity is small and the distance between different identities is larger. We define x_i^a to be the anchor face of an identity, x_i^p be a face of the same identity and x_i^n be face of any other identity (negative). Now if we want the faces of the same identity closer to each other than any image by a margin of α we arrive at:

$$\|f(x_i^a)) - f(x_i^p)\|_2^2 + \alpha < \|f(x_i^a) - f(x_i^n)\|_2^2 \quad (1)$$

$$\forall (f(x_i^a)), f(x_i^p), f(x_i^n)) \in \mathbb{T} \quad (2)$$

where $\mathbb{T}$ is all possible triplets in our training data. This gives us the following loss function $\mathcal{L}$ which is minimized in the deep neural network:

$$\mathcal{L} = \sum_{\mathbb{T}} [\|f(x_i^a) - f(x_i^p)\|_2^2 - \|f(x_i^a)) - f(x_i^n)\|_2^2 + \alpha] \quad (3)$$

An illustration of this is shown in figure 2. Computing over all triplets would be computationally exhausting and would give us many triplets which are easily satisfied and do not contribute to the training. Therefore it is wise to select triplets which can improve the model. By selecting faces from the same identify which are furthest apart (max $\|f(x_i^a) - f(x_i^p)\|_2^2$) and a face from another identify which is closest to the anchor (min $\|f(x_i^a) - f(x_i^n)\|_2^2$) we can create triplets which violet the constraint and ensure faster convergence. As doing this for the whole training set is not feasible FaceNet proposes generating triplets every n steps. The minimum and maximum are computed on the latest checkpoint on a subset of the data. This is the approach that is used by OpenFace.

By selecting a number of F faces for each person and a total number of P persons a mini sample is taken. This sample consists of $N \approx FP$ and is send through the neural network. By selecting anchor pairs this results in N triplets, where $N = Q\binom{P}{2}$. Then the triplet loss is computed and the derivative is mapped back to the image by using a backward network pass. If a negative image is not found for the anchor pair it is not used. We used the following values for these parameters based on OpenFace and FaceNet:

$$\alpha = 0.2 \qquad m = 28$$
$$F = 20 \qquad P = 15$$

OpenFace uses Torch, Lua and lujajit for the training of the neural network and interference. Python is used for array operations, computer vision and classifying. In particular the libraries numpy, OpenCV and scikit-learn were used. Scikit-learn provides you with a number of different classifiers, were OpenFace favours the linear SVM we also consider the Radial SVM. By evaluating the squared euclidean distance of new unknown images to it's closest match in the training set we can evaluate the chance that it is actually the same person. If the squared euclidean distance is below a certain threshold the person is the same, otherwise it is different. OpenFace reports this as the confidence level that a pair is the same person.

3.3 Training data

The pictures which would go through or trained model would contain a lot images with unknown people. To not get too many false positives it would make sense to make our training data mimic the real dataset which is model was being trained for. We collected data from a number of sources:

- PubFig[1] is a large dataset consisting of 58,797 images of 200 famous persons
- Adience collection[2] is a dataset of 26,580 pictures of unknown identities in real-word imagining conditions
- GRAZ dataset[3] consists of a collection of persons, bikes and cars pictures split into GRAZ01 and GRAZ02, only the person images were used
- Labeled Faces in the Wild[4] (LFW) is a widely used dataset consisting of 13,000 images for 5749 persons
- Out of curiosity images we added images of two persons at the VU to training set to attempt to find them on Flickr

The PubFig dataset would constitute the famous people that we wanted to find in the Flickr images archive and the other datasets are used as unknowns to reduce the false positives. The LFW contained a large number of the persons which were present in PubFig, those were removed. After removing the corrupt pictures and aligning the faces properly we were left with the following number of classes and pictures: The

Table 2: Number of aligned pictures per dataset

Dataset	Identities	Total pictures	Picture/identity
PubFig	200	23765	118,8
Adience	2192	20127	9,2
GRAZ	216	432	2,0
LFW	5587	17152	3,1
VU	2	21	10,5
Total	**8197**	**61497**	**28,7**

total number of unknown identities are thus 7,997 identities and 37,732 pictures while there are 200 famous people with a total of 23,765 images.

[1]http://www.cs.columbia.edu/CAVE/databases/pubfig/
[2]https://talhassner.github.io/home/projects/Adience/Adience-data.html
[3]http://pascal.inrialpes.fr/data/human/
[4]http://vis-www.cs.umass.edu/lfw/

3.4 Training the model

To train the model we first tried to use SageMaker, an Amazon managed platform that enables you to train and deploy your machine learning model at any scale. It supplies you with a variety of machine learning algorithms and also allows you to add your own. Adding the OpenFace algorithm as a pre-build docker container we tried to import it into SageMaker to let SageMaker handle the scaling. By using the NVIDIA docker the intent was to improve the training speed by using GPU powered devices. However the NVIDIA docker with OpenFace proved to be not compatible and we abonded SageMaker for a single EC2-instance. Instead of using NVIDIA docker we installed OpenFace natively on a p2.8xlarge instance. Torch allows the network to be executed on a GPU by utilizing CUDA. As the p2.8xlarge instances comes with 8 Nvidia K80 GPUs this accelerated training of the network enough to make it feasible to train the model using the complete dataset discussed in section 3.3.

3.5 Amazon Web Services (AWS) for Big Data

After training the model it would have to be deployed on a system where it could run in parallel to make it possible to go through all 100 million pictures in time. As we were given access to the SURFsara cluster we tried to install OpenFace on it to deploy the model. However, we were met with many dependencies issues which we not able to fix. We found another solution by utilizing the numerous services that are being offered by AWS. The following services were used:

- EC2: Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides secure, re-sizable compute capacity in the cloud. It is designed to make web-scale cloud computing easier for developers. Amazon EC2 provides a wide selection of instance types optimized to fit different use cases. Instance types comprise varying combinations of CPU, memory, storage, and networking capacity. T2 instances are bur-stable Performance Instances that provide a baseline level of CPU performance with the ability to burst above the baseline. We used 100 t2.medium EC2s and on each of them we run 20 dockers. Each docker runs with the minimum of 128 MG memory and maximum of 700 MG byte memory. Based on our calculation we don't need more than 700Mg. So with the help of setting minimums and maximums we were able to run 20 dockers on each t2.medium instances.
- ECS: Amazon Elastic Container Service (Amazon ECS) is a highly scalable, high-performance container orchestration service that supports Docker containers and allows you to easily run and scale containerized applications on AWS. A Docker container is a standardized unit of software development, containing everything that a software application needs to run: code, runtime, system tools, system libraries, etc.
- Task Definition: The task definition is a text file, in JSON format, that describes one or more containers, up to a maximum of ten, that form one application. Examples of task definition parameters are which containers to use, which launch type to use, which ports should be opened for the application, and what data volumes should be used with the containers in the task.

- Task: A task is the instantiation of a task definition within a cluster.
- Auto Scaling: The service provides a simple, powerful user interface that lets you build scaling plans for resources including Amazon EC2 instances and Spot Fleets, Amazon ECS tasks, Amazon DynamoDB tables and indexes and Amazon Aurora Replicas.
- EFS: Amazon Elastic File System (Amazon EFS) provides simple, scalable, elastic file storage for use with AWS Cloud services and on-premises resources.

4. EXPERIMENTS AND RESULTS

Before deploying the model on the complete dataset and potentially wasting a lot of resources we first wanted to be sure that it performed well.

4.1 Creating a classification model

The first step was to determine a classifier as OpenFace supplies you with a range of different classifiers to pick from. We tested the Linear SVM, Radial SVM, Gaussian Naive Bayes, GMM and DecisionTree classifiers, the results are displayed in table 3. In a controlled environment we tested the classifiers to see their respective accuracies and average confidence for both known and unknown identities. This was done by testing the trained models on new images of identities it was trained on and by sampling in unknowns as well. As can be seen from the results in table 3 the DecisionTree classifiers performed worse than the rest, with the linear and Radial SVM showing very similar results, this concides with the results which were observed by the OpenFace research.

Table 3: Accuracy per different classifier

Classifier	Average Confidence	Accuracy
Radial SVM	0,448	92,9%
DecisionTree	0,693	48,5%
Gaussian	0,996	92,0%
Linear SVM	0,446	92,7%

To test the model discussed in section 3.4 we downloaded a sample of 10,000 Flickr images which were run through the trained model. The results were abysmal though so we had to tweak our initial approach.
Instead of adding a large number of unknowns to the training data we altered the approach and focused on a smaller number of known identities. We used the 60 identities of the development set of PubFig as training data. As an alternative we also used 50 and 100 identities of the LFW funneled dataset, where the identities with the most amount of pictures were chosen. Both linear SVM and radial SVM were considered. Radial SVM results have shown to be marginally better than the linear SVM, however this comes at the cost of extra computations.

As there were no more unknowns in the model we would need to determine a threshold confidence level γ for which a match to a famous person was credible. To compute baseline values for the confidence levels we ran 100 images of known identities through the models and calculated the average confidence level. We did the same for 100 images of unknown identities. All of these images were not used in training the models, the results are summarized in table 4.

Table 4: Average confidence levels for unknown and known identities

	Linear SVM		Radial SVM	
Training set	Unknown	Known	Unknown	Known
PubFig dev	0.252	0.557	0.248	0.542
LFW 100	0.133	0.446	0.104	0.448

Once again we evaluate the results of the trained models by testing them on a sample of 10.000 Flickr images. By taking $\gamma = 0.55$ for the PubFig model there were 103 matches with approximately 90% of them being matched to Miley Cyrus. Even though the confidence levels were very high for some of the matches (> 0.9) the people didn't resemble Miley Cyrus in the slightest to the human eye. This result was apparent both when using the linear and radial classifiers. The LFW-100 model performed a lot better. The matched Flickr images all had a striking resemblance to the famous identity. As the LFW-100 trainig set combined with the Radial SVM classifier gave us the best results it was selected to be used on the Flickr image archive.

4.2 Deploying the model

The next step is deploying the LFW-100 model on the EC2-cluster as discussed in section 3.5.

4.2.1 Implementation

We need to process 100 million number of records, where in each record the information about the image, including it's download URL, and location is specified. We divided the raw data into a set of data files, where each file includes 10,000 records. Next, we defined a Task as a set of processes which is applied to a data file to perform face recognition. These processes are as following:

1. For every record in dataset read the image URL: if the end of URL ends with 'png', 'jpg', or 'gif', then download the image from Flicker and store the image binary. Next, if the size of the image is less than the threshold of 30 Kb remove the image.
2. For every image in the local image folder, check if there are any faces in the image using OpenFace face detection method. Remove the image if no faces are found.
3. For every aligned image in a local image folder, apply the OpenFace feature detection method to get their feature vector. This result is stored in feature vectors in JSON-format.
4. For each feature-vector, apply open-face classifier method, with our trained model (radial SVM), and store the result if there are detected matches with a confidence value equal or higher than a certain threshold (γ).

In the end, the classification result will be stored in a JSON file with the the name of the data file as it's identifier. These steps are executed one after the other, and are called through a shell script. A global counter is used to select the next data file to process. Upon execution of the script, the counter would be incremented, and the corresponding data file would be selected for processing. This process proved to be very CPU-intensive, which was the bottleneck of processing the

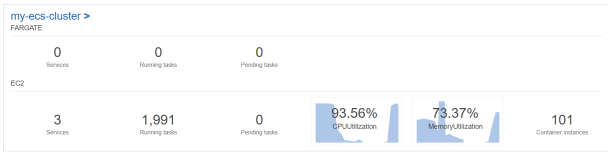

Figure 3: CPU-utilization on AWS

images. By utilizing 100 containers and running 2000 tasks we processed the dataset in batches of 20 million pictures. The AWS metrics are displayed in figure 3. Finding the faces in the pictures proved to be the most CPU intensive task which increased processing times significantly. Downloading the images and calculating the confidence level of each match was relatively fast. After using Openface docker on our created cluster, spark was used for analyzing JSON files. Our extracted JSON file has the following format:


```
▼ array [3]
  ▼ 0 {3}
      url : http://farm7.staticflickr.com/6125/5964923354_29871c629f.jpg
      person : Gloria_Macapagal_Arroyo
      conf : 0.439
  ▼ 1 {3}
      url : http://farm9.staticflickr.com/8275/8962524435_acfd53f593.jpg
      person : George_W_Bush
      conf : 0.362
  ▼ 2 {3}
      url : http://farm6.staticflickr.com/5142/5628526765_f5df565225.jpg
      person : Colin_Powell
      conf : 0.55
```


Figure 4: JSON structure

The created JSON files were moved to SURFsara. Our pipeline in SURFsara is as follows.

- Read all JSON files into one DataFrame
- Sort them with the confidence score and person
- Perform filter action on them to chose which rows have confidence score more than our threshold
- Join with raw dataset in order to find location
- create new JSON file that will be used in our visualization.


```
▼ array [50]
  ▼ 0 {3}
      name : George_W_Bush
      profilePic : http://farm1.staticflickr.com/80/244233302_576e4421c9.jpg
    ▼ foundPics [1]
      ▼ 0 {3}
          URL : http://farm1.staticflickr.com/862/41812017150_ccbafd7ff4_b.jpg
          location : Amsterdam
          rank : 0
```


Figure 5: JSON result structure

4.3 Results evaluation

In the final result we collected 2,748,159 matches which are displayed in a density histogram in figure 6. The average confidence is 0.084 with a standard deviation of 0.051. The distribution of the confidence seems to closely follow a Log-Normal distribution, which is was fitted to the data and is displayed in red in the figure. The average confidence is very low as is to be expected when cycling through random images from Flickr. The figure is cut off at a confidence of 0.5 even though there are matches with higher confidence, however this is such a small fraction that it was not visible in the histogram.

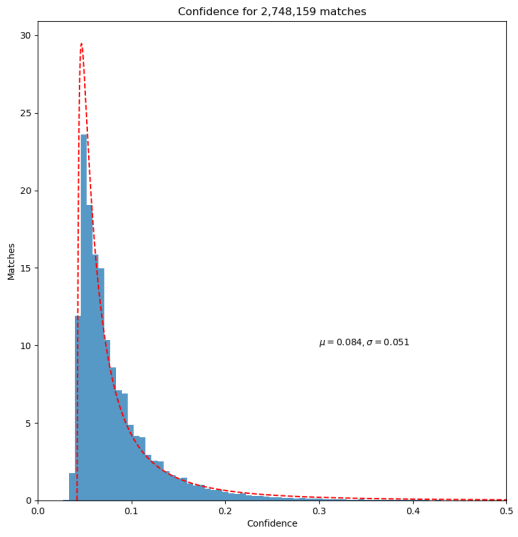

Figure 6: CPU-utilization on AWS

We are only interested in the matches with a higher confidence level, about 3,4% of the matches have a confidence level larger than 0.2. If we take the average confidence level of the known identities from table 4 and set this as the cut-off threshold γ we end up with a total of 5012 matches, or 0,18% of the total. These are the matches which are considered as feasible and are used in the final results and the visualization. The total number of training images varies per identity, with a minimum of 14 images and a maximum of 530 images. It was remarkable that the person with the most training images (George W. Bush) also had the most matches in the Flickr archive. To test if there is no bias to the identities with the most training images the average confidence and total number of matches per identities was computed. Then the identities were sorted based on the number of training images and given a percentile rank of 0 - 1, where George W. Bush is 1 as it is the identity with the most training images. The percentile rank is plotted against the average confidence and total number of matches in figure 7. As can be seen there is no decisive connection between the number of training images and the total number of matches or the average confidence level. When we go to the very top of the percentiles we do see a small increase in both though, but this proves to not be consistent across both number of matches and average confidene. As an example George W. Bush has by far the most matches but the average confidence level is below average.

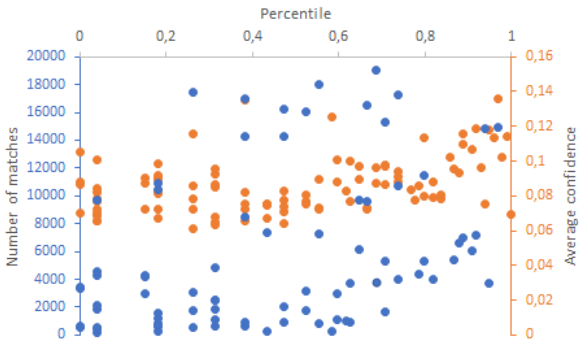

Figure 7: Scatter plot based on percentile rank

4.3.1 Visualization

The results of the classification is visualized through a web-based user interface. The names of the known persons which the model was trained on are placed on a scrollable sidebar. By clicking a particular person's name it will show the matches that were made for that person in the Flickr image archive ranked by their confidence level. For the matches that have location data available to them the location is also displayed which links to the pinpointed Flickr map. Only matches above the confidence threshold are considered. Upon selecting a particular image in the list, the original image will be opened in Flickr website.

5. DISCUSSION AND CONCLUSION

During the course of this project we were met with many issues to get OpenFace to run on a large scale. In the end a solution was found by training the model on a GPU powered instance and deploying the model on Amazon's Elastic Container Service. Open source facial recognition software is getting closer to the performance of the private facial recognition software and can also be deployed on a large scale as is seen in this project. However the extremely large datasets from social media which Google and Facebook can use to enhance their training are still unmatched. Big corporations have also started to develop image and video recognition as a service, for example Amazon Rekognition[5].

More research should still be done on the best way to train a model when dealing with facial recognition with a very large number of unknowns. In our case adding unknowns worsened the model and the best solution was only using known people and setting a threshold confidence level for matches, recent discussion on the OpenFace Github[6] also didn't come to a decisive conclusion.

Which facial recognition software is best fit is also still up for debate. Very recently a modified OpenFace[2] was released which incorporate pairs which were discarded to improve both accuracy and performance. Facial recognition has gone through a lot of developments recently but is far from solved, this results in a rapidly expanding field led by the biggest tech corporations.

[5]https://aws.amazon.com/rekognition/

[6]https://github.com/cmusatyalab/openface/issues/144

6. REFERENCES

[1] B. Amos, B. Ludwiczuk, and M. Satyanarayanan. Openface: A general-purpose face recognition library with mobile applications. 2016.

[2] K. Santoso and G. P. Kusuma. Face recognition using modified openface. *Procedia Computer Science*, 135:510 – 517, 2018. The 3rd International Conference on Computer Science and Computational Intelligence (ICCSCI 2018) : Empowering Smart Technology in Digital Era for a Better Life.

[3] F. Schroff, D. Kalenichenko, and J. Philbin. Facenet: A unified embedding for face recognition and clustering. *CoRR*, abs/1503.03832, 2015.

[4] A. S. Syed Navaz, T. Dhevi Sri, and M. Pratap. Face recognition using principal component analysis and neural networks. 3:245–256, 2013.

[5] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf. Deepface: Closing the gap to human-level performance in face verification. pages 1701–1708, 2014.

[6] K. Q. Weinberger and L. K. Saul. Distance metric learning for large margin nearest neighbor classification. *J. Mach. Learn. Res.*, 10:207–244, June 2009.