Web Front-End Realtime Face Recognition Based on TFJS

Chenyang Li
Faculty of Electronic and Information Engineering,
Nanyang College, Xi'an Jiaotong University
Xi'an, China

Chunfang Li
School of Computer Science and Cybersecurity
Communication University of China
Beijing, China

Abstract—**Web front-end face recognition enables developers to run models on browser-side to reduce latency and server load. In this paper, we first analyzed different algorithms, standard APIs and frameworks that can be used in web front-end face recognition. Then, we designed a workflow for this and implemented it in JavaScript. In the end, we tested it on different scenarios, and the result shows that front-end real-time face recognition is possible and user acceptable.**

Keywords-face recognition; tensorflow.js; video understanding

I. INTRODUCTION

For more than half a century, face recognition has drawn a lot of interests among researchers and companies. And much efforts currently have been devoted to creating better algorithms and shipping them on server, desktop or mobile devices. However, studies on web-based face recognition remains fresh and new. Also, with the boom of web-based one-click apps, face recognition running on browser-side is much needed, in order to reduce latency and server load.

The applications of face recognition are various, including payment, security, advertising, entertainment and etc. Payment companies like Alipay, WeChat Pay and MasterCard have developed algorithms for users to pay by taking a selfie. While Apple, Huawei, XiaoMi and other mobile phone manufacturers are focusing on unlocking devices using face recognition. Also, Face++ and polices are jointly using face recognition to identify criminals in public events. Douyin and kuaishou short video apps provide many face entertainment applications, such as face swap, make-up mode, and disfigure. In movie and TV shows, face recognition is used in video search such as only see him/her in iQiyi and Tencent video, and in film and television production industry, face recognition is used for face swapping or hunting for substitutes of a given actor/actress. Those applications show the mass usage of face recognition.

Face recognition is a biometrical technology to identify a person by his/her facial information from a digital image or a video frame from a video source. Generally, face recognition includes several tasks like face detection, alignment, identification and verification. Face detection is an object detection task, which is to locate faces on a given image and give a bounding box for each face detected. Face alignment is to automatically locate landmarks on a face image, such as eye corners, mouth corners, etc. Face identification is to find if the given face is shown in the probe set. Face verification is to check if the two faces given are from the same person. Fig. 1 shows the three different tasks of face detection, alignment, and identification implemented in this paper, in which face is the main character Lifei of a famous Chinese TV series *Ice Breaker* in 2019 spring.

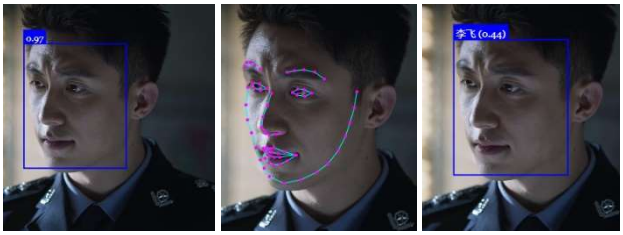

Figure 1. Face detection alignment and identification

This paper introduced a working implementation of web front-end real-time face recognition and it was used on detecting, identifying actors/actress in a playing video and webcam face tracking. Afterwards, we ran three tests on these two scenarios, and the results shows that front-end real-time face recognition is possible and user acceptable.

The remainder of the paper is organized as follows: Section II briefly reviews related works for face detection, alignment and identification. Then Section III we analyzed different face recognition frameworks and standard APIs. The proposed workflow for front-end face recognition is illustrated in Section IV. Experimental results are presented in Section V, leading to conclusions in Section VI.

II. RELATED WORK

A. CNN:Convolutional Neural Network

A Convolutional Neural Network, also known as CNN or ConvNet, is a class of neural networks that specializes in processing data that has a grid-like topology, such as an image. A CNN typically has three layers: a convolutional layer, pooling layer, and fully connected layer, as shown in Fig. 2.

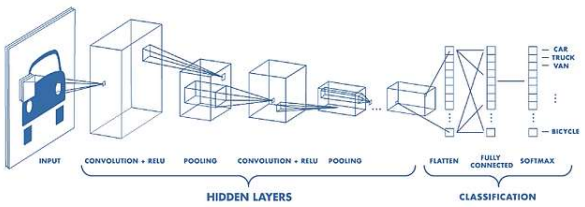

Figure 2. Typical ConvNet structure

978-1-7281-4852-6/19/$31.00 ©2019 IEEE

CNN is widely used in computer vision, and there are some classical implements including LeNet-5[1], AlexNet[2], VGG16[3], GoogLeNet and ResNet[4].

CNN has been successfully used in face recognition, object detection, semantic segmentation, image captioning, and sophisticated models like R-CNN, Fast R-CNN, and Faster R-CNN[5] are created for many object detection models deployed in autonomous vehicles, facial detection, and more.

B. Face Detection

Face detection includes traditional approaches and approaches based on deep learning. And traditional approaches consist template match[6-7], feature descriptor[8-9], wavelet[10] and neural network[11-12]. In order to achieve multi-scale face detection, a general structure called pyramid [13] is widely used in those approaches. Fig.3 shows the match template in reference 1 and 2. In fact, template match models the eyes, node, and mouth with fitted curves including their relative location.

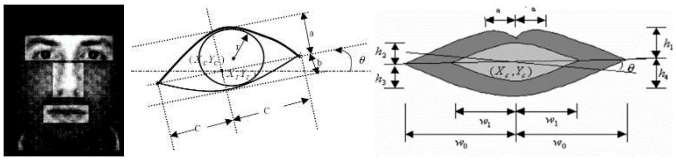

Figure 3. Template match in face detection

Recently, deep learning is used in face detection and it significantly improves performances. Those algorithms are categorized as two-stage or end-to-end approaches. Typical two-stage algorithms include R-CNN、Fast R-CNN、Faster R-CNN[5]. And famous end-to-end algorithms consist YOLO (You Only Look Once)[14], SSD(Single Shot MultiBox Detector)[15], and MTCNN.

YOLO regards object detection (including face detection) as a regression problem to spatially separated bounding boxes and associated class probabilities. A single neural network predicts bounding boxes and class probabilities directly from full images in one evaluation[14].

SSD is a single-shot object detector for multiple categories using multi-scale convolutional whose bounding box outputs attached to multiple feature maps at the top of the network[15].

Object Detection using Haar feature-based cascade classifiers is an effective object detection method proposed by Paul Viola and Michael in 2001[16]. It is a machine learning based approach where a cascade function is trained from a lot of positive and negative images. It is then used to detect objects in other images. OpenCV implements the Haar-cascade detection with many pre-trained classifiers for face, eyes, smile etc., and XML files are used to store the parameters in opencv/data/haarcascades/ folder.

C. Face Alignment

Face alignment, or facial landmark detection can be used in face morphing, face averaging and face swapping.

OpenCV's facial landmark API is called Facemark. It has three different implementations of landmark detection based on three different papers FacemarkKazemi[17], FacemarkAAM[18], FacemarkLBF[19].

Face alignment algorithms may output different number of landmarks. So those algorithms can be categorized by the size of their output, such as often used 5-point, 68-point, 1000-point models and etc.

The 5-point model is the simplest and commonly-used model, and it contains 2 points for the corners of both eye and 1 point for the nose. Usually 5-point face landmarks are used to detect the pose of a face or align a face by rectifying its rotation and size.

The 68-point model is another commonly-used model, it contains 5 points for each eye brow, 6 points for each eye, 9 points for nose, 20 points for mouth and 17 points for face edge. Also, points of 5-point model are contained in 68-point model. The difference of 5-point, 68-point, and 1000-point is illustrated in Fig. 4. The 1000-point model is developed by Face++. By adding more points, it can give precise shape of the face, so it can be used for face make-up or beauty filter.

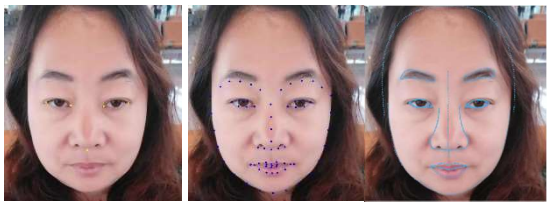

Figure 4. The difference between face alignment models

D. Face Identification

Typically face identification is done by embedding the face into a vector space, using metric learning technique. The vectors are called face descriptors. And usually a CNN is used to compute those descriptors. With face descriptors, we can identify or verify faces, using K-nearest or similar algorithms. Here we use dnn_metric_learning_on_images_ex.cpp, the example of dlib, to illustrate general ideas of face identification using metric learning.

Firstly, a CNN is constructed, with a backbone network to describe input faces and a few fully-connected layers behind to output the vector. And a large amount of faces is used for training, which usually contains more than one face image for each person. To distinguish one person from another in the vector space, face descriptors from different person are expected to be as far as possible, while descriptors from the same person are expected to be nearing. So, a special loss function, called metric loss, is applied to the CNN. After enough training, we can use the CNN to embed face image into the vector space.

E. Dataset of Face Recognition

MTFL(Multi-Task Facial Landmark) dataset contains 12,995 face images collected from the Internet. The images are annotated with five facial landmarks, and attributes of gender, smiling, wearing glasses, and head pose.

LFW(Labeled Faces in the Wild) dataset has more than 13,000 images of faces collected from the web. Each face has been labeled with the name of the person pictured. 1680 of the

people pictured have two or more distinct photos in the data set[20].

CelebFaces Attributes Dataset (CelebA) is a large-scale face attributes dataset with more than 200K celebrity images, each with 40 attribute annotations. The images in this dataset cover large pose variations and background clutter. CelebA has large diversities, large quantities, and rich annotations, including 10,177 number of identities, 202,599 number of face images, and 5 landmark locations, 40 binary attributes annotations per image. The dataset can be employed for face attribute recognition, face detection, landmark (or facial part) localization, and face editing and synthesis [21].

III. FACE RECGNITION FRAMEWORKS AND API

A. OpenCV and Dlib

OpenCV is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc.

Dlib is a toolkit containing machine learning algorithms and computer vision to solve real world problems. It is used in robotics, embedded devices, mobile phones, and large high performance computing environments. In face detection, Dlib provides CNN face detector, face alignment, face clustering, face detector, face jittering/augmentation, face landmark detection, face recognition.

B. Web Standard APIs

In this subsection, we listed possible web standard APIs that can be used in front-end face recognition.

1) HTML5 Web Worker

Web Worker is a set of APIs for multithreading on browser side. By utilizing Web Worker, we can speed up the inference of DNN(Deep Neural Network).

2) WebGL

WebGL is an OpenGL alternative on browser side. It provides fine-grained APIs to access GPU and render 3D scenes, so we can use Shader language for parallel computation. This boosts the inference of DNN significantly.

3) Emscripten and WebAssembly

Emscripten is a backend of the famous compiler toolchain LLVM(Low Level Virtual Machine). It compiles native code to vanilla JavaScript or WebAssembly. While WebAssembly is a binary format designed for cross-compiled native apps, which provides a nearly-native speed for running native apps. By using Emscripten and WebAssembly altogether, we can cross-compile Dlib or OpenCV and make use of them on browser side.

C. Web-based Frameworks

In this subsection, we analyzed current available browser-side face recognition frameworks. Table 1 shows their differences.

TABLE I. COMPARISON OF DIFFERENT FRONT-END FACE RECOGNITION FRAMEWORKS

Name	CPU Support	GPU Support	Is Continued	Usability (1 - 5)
ConvNet.js	Yes (Vanilla)			3
TF.js	Yes (Web Worker)	Yes (WebGL)	Yes	4
OpenCV.js	Yes (WebAssembly)		Yes	2
face-api.js	Yes (TF.js)	Yes (TF.js)	Yes	5
Keras.js	Yes (Vanilla or Web Worker)	Yes (WebGL)		3
WebDNN	Yes (Web Worker or WebAssembly)	Yes (WebGL)	Yes	4

1) ConvNet.js

ConvNet.JS is one of the most famous front-end neural network frameworks, as well as an early practice, which was started by Karpathy[22]. It implements browser-side the training, inference of MLP(Multi-Layer Perceptron) and CNN, using vanilla JavaScript. It is well documented and great to use. But with the rise of TF.js, ConvNet.js is discontinued.

2) TF.js (TensorFlow.js)

TF.js is the most famous and mature front-end DNN framework until 2019, which was originated by Google as an auxiliary of TensorFlow [23]. As a result, it is fully compatible of TensorFlow. Also, it can make use of Web Worker and WebGL, accelerating the inference process.

3) OpenCV.js

OpenCV.js is a browser-side port of OpenCV, utilizing Emscripten and WebAssembly. It provides almost all features of OpenCV, consequently, it has a large program size and it's too heavy for browser-side face recognition.

4) Face-api.js

Face-api.js is developed by Vincent Muhler. Based on TF.JS, it provides a lot of out-of-box models and APIs for face recognition[24]. It's optimized for web environment and easy to use.

5) Keras.js

Keras.js is developed by Leon Chen, which is partly compatible with Keras models. It provides WebGL and Web Worker support, but with the rise of TF.JS, Keras.js is discontinued because of better tensorflow.js emerge.

6) WebDNN

WebDNN is developed by MIL Lab of Tokyo University, it provides support of Web Worker, WebGL and WebAssembly. It is comprehensive and easy to use.

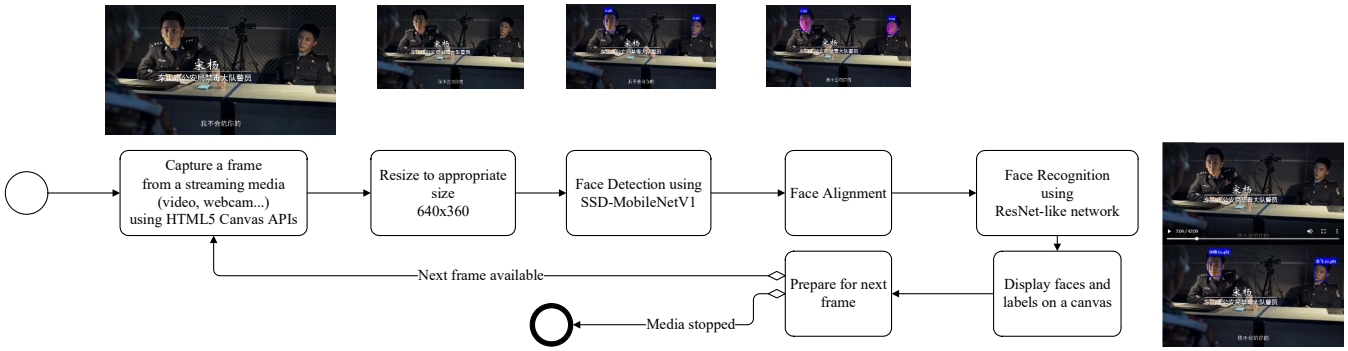

Figure 5. Video face recognition based on SSD-MobileNetV1

In conclusion, for academic demonstration, ConvNet.js suits best. For implementing new algorithms, TF.js and WebDNN fits for their comprehensive and fine-grained APIs. For developing apps, face-api.js is better, because of its out-of-box support of face recognition.

IV. Workflow Design

To implement a front-end real-time face recognition prototype system, a workflow is designed and described as follows. Fig. 5 visualizes the whole process.

Firstly, the workflow requires a video stream as an input. It can be a movie, TV show or a webcam stream. And it will be loaded via a HTML <video> tag. As the video starts to play, the workflow loops to process frames.

To capture a frame, we utilize HTML5 Canvas API. The canvas is a web-based equivalent to Java's java.awt.Graphics2D, which enables developers to render and manipulate 2D shapes, text and images on browser-side. Also, they have similar APIs, such as drawSometing() for drawing objects and fillSometing() for fill objects. And here we use drawImage(*obj*, *x*, *y*, *w*, *h*), where obj can be an image or a video. If a video is passed to drawImage, it will draw the immediate frame of the video. Also, we can resize the frame to a smaller size by specifying parameter *w* and *h* and here we choose 640x360. Compared to common video resolution 1920x1080, this significantly reduces pixels passed to CNNs, which ensures speed of the process.

Afterwards, we run SSD-MobileNetV1 model provided by face-api.js to detect faces on the captured frame, where SSD is the object detector described before. Originally SSD uses VGG16 as its backbone network, but here VGG16 is replaced by MobileNetV1, another CNN backbone network developed by Google. Compared to VGG16, MobileNetV1[25] is optimized for mobile devices by employing a trick called depthwise separable convolution, which reduces amount of computation.

Then the face alignment is applied to the detected faces. Here we use a tiny yet accurate 68-point face landmark predictor provided by face-api.js. The whole model is ~200kb, which is suitable for web environment. Afterwards, we align predicted 68-point landmarks and the standard landmarks. Then we have aligned faces for face recognition.

To do face recognition, a ResNet-34 like architecture is implemented to compute a face descriptor. The face descriptor is a vector of 128 values, representing a face in the 128-dementional vector space. Also, the euclidean metric of different vectors represents their similarity. Then we can match each face shown in the video frame with pre-calculated fingerprints of each person, using naive KNN(k-nearest neighbor) algorithm.

Last, we visualize the results on the screen, utilizing the Canvas API, again. We put the original video frame on the background and the labeled bounding box of each detected face on the foreground. Then, it loops back to capture the next frame if the video is still playing, or the workflow comes to an end.

V. Experiments

We implemented the workflow by using Vue.js and Bootstrap Framework. Then we evaluate the prototype system on a Windows 10 workstation with Ryzen 1700 CPU, 16GB DDR4 memory and GTX1080 graphics card. And the different test cases and their results are listed as follows.

A. Front-end face recogntion in TV series

Firstly, we assessed the effectiveness of our workflow by visualizing face descriptors from different actors. To simplify 128-dimensional vectors and show an intuitive image of the distribution of face descriptors, we chose PCA (Principal Components Analysis) to reduce dimensionality to 2. Fig. 6 shows the clusters of different actor's/actress' face descriptors on a 2D plane after PCA. In the figure, LifeiX, LiweiminX and ChenkeX represent faces from different actors/actresses in TV series *Ice Breaker*.

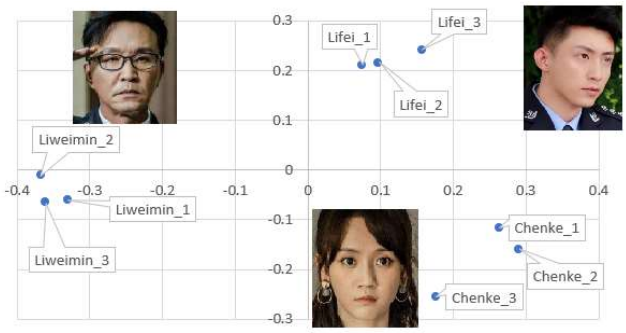

Figure 6. Distribution of face descriptors

Then, we measured the performance of our system by playing a clip of Episode 1 in TV series *Ice Breaker*, from 02:00 to 05:00. And the results show that our prototype system runs ~12 FPS (frames per second), which is fast enough for real-time face recognition. Fig. 7 shows the system in running.

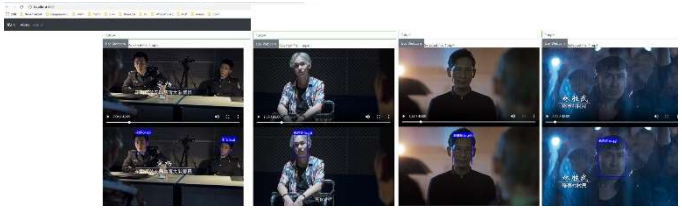

Figure 7. Face identification in *ice breaker* TV series

B. *Front-end face recognition with camera*

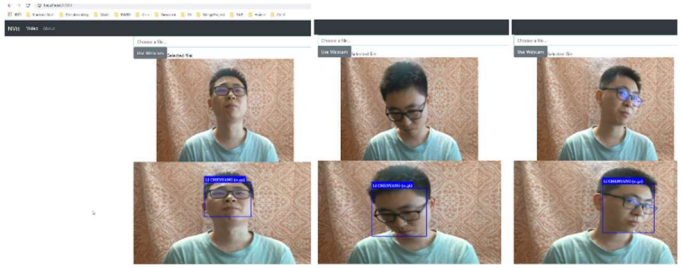

Figure 8. Camera face recognition

To evaluate the robustness of our prototype system, we also run it with a webcam. Fig. 8 shows the output of the system under different poses. And this shows the system is robust to pose changing.

VI. CONCLUSIONS

In this paper, we proposed a web front-end real-time face recognition system. By analyzing different approaches, standard APIs and frameworks, we came up to a design. Then we implemented it and tested it on different scenarios. The result shows that front-end real-time face recognition is possible and user acceptable. In the future, the system shall be integrated to a video/live streaming site, in order to realize face searching or face swapping.

ACKNOWLEDGMENT

This work was supported by the National Social Foundation of China(18BC034), Key Cultivation Engineering Project of Communication University of China (3132017XNG1606, 3132017XNG1719, CUC2019B023), and Double Class Project of CUC (CUC18JL065).

REFERENCES

[1] Lecun Y, Bottou L, and Bengio Y., "Gradient-based learning applied to document recognition," Proceedings of the IEEE, vol.86(11), pp. 2278-2324, 1998.

[2] Krizhevsky, A., Sutskever, I. and Hinton, G. E., "ImageNet classification with deep convolutional neural networks," Advances in Neural Information Processing, 25, MIT Press, Cambridge, MA, 2012.

[3] Simonyan K, Zisserman A., "Very deep convolutional networks for large-scale image recognition," International Conference On Learning Representations, 2015.

[4] He K, Zhang X, Ren S. and Sun J., "Deep residual learning for image recognition,". Computer Vision and Pattern Recognition, pp.770-778, 2016.

[5] Ren, S., He, K., Girshick, R., & Sun, J., "Faster R-CNN: Towards real-time object detection with region proposal networks," In Advances In Neural Information Processing Systems, pp. 91-99, 2015.

[6] Brunelli, R., & Poggio, T., "Face recognition: Features versus templates," IEEE Transactions On Pattern Analysis And Machine Intelligence, vol.15(10), pp.1042-1052,1993.

[7] Shan S.G., "Research and implementation of face feature detection and recognition," Dissertation for master degree of Harbin Institute of Technology, 1999.

[8] Dalal N., and Triggs B. "Histograms of oriented gradients for human detection," Computer Vision And Pattern Recognition, pp.886-893, 2005.

[9] Ojala T., Pietikainen M., and Maenpaa T., "Multiresolution gray-scale and rotation invariant texture classification with local binary patterns," IEEE Transactions on Pattern Analysis & Machine Intelligence, vol.7, pp.971-987, 2002.

[10] Papageorgiou, C. P., Oren, M., and Poggio, T., "A general framework for object detection," In Sixth International Conference on Computer Vision pp. 555-562, 1998.

[11] Rowley, H. A., Baluja, S., and Kanade, T., "Human face detection in visual scenes," In Advances in Neural Information Processing Systems, pp. 875-881, 1996.

[12] Rowley, H. A., Baluja, S., Kanade, T., "Rotation invariant neural network-based face detection," Carnegie-Mellon Univ Pittsburgh Pa School Of Computer Science, 1997.

[13] Tanimoto, S., & Pavlidis, T., "A hierarchical data structure for picture processing," Computer graphics and image processing, vol.4(2), pp.104-119, 1975.

[14] Redmon, J., Divvala, S., Girshick, R., and Farhadi, A., "You only look once: Unified, real-time object detection," In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 779-788, 2016.

[15] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C. Y., and Berg, A. C., "SSD: Single shot multibox detector," In European conference on computer vision, pp. 21-37, 2016.

[16] Viola P A, and Jones M J., "Rapid object detection using a boosted cascade of simple features," computer vision and pattern recognition, pp.511-518, 2001.

[17] Kazemi V, and Sullivan J., "One millisecond face alignment with an ensemble of regression trees," IEEE Conference on Computer Vision and Pattern Recognition, pp. 1867-1874, 2014.

[18] Tzimiropoulos G, and Pantic M., "Optimization problems for fast aam fitting in-the-wild," International Conference On Computer Vision, pp.593-600, 2013.

[19] Ren S, Cao X, and Wei Y, "Face alignment at 3000 fps via regressing local binary features," Computer Vision And Pattern Recognition, pp.1685-1692, 2014.

[20] Erik G. L., Huang G.B., Roychowdhury A., Li H.X., and Hua G., "Labeled Faces in the Wild: A Survey," In Advances in Face Detection and Facial Image Analysis, edited by Michal Kawulok, M. Emre Celebi, and Bogdan Smolka, Springer, pp.189-248, 2016.

[21] Liu Z, Luo P, Wang X., "Deep Learning Face Attributes in the Wild," International Conference On Computer Vision, pp.3730-3738, 2015.

[22] https://cs.stanford.edu/people/karpathy/convnetjs/started.html, 2019

[23] https://tensorflow.org/js/, 2019

[24] https://justadudewhohacks.github.io/face-api.js, 2019

[25] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., & Adam, H.. Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint.,2017