The-Phy Lieng

Tensorflow Facial Recognition - OpenFace - jsTensorflow

ABSTRACTION

According to Andrew Ng, formerly head of Google Brain, machine learning has a ton of day-to-day life applications ranging from self-driving cars and practical speech to effective web search and face recognition. To investigate how machine learning works, We use Tensorflow as a mean and deploy the structure of Tensorflow itself. We use the orignal research Tensorflow models with two models configuration: Faster-RCNN and SSD-MobileNet for face recognition. The dataset is extracted from 8 members of Android Department to obtain 80 samples for the first expereriment and 498 samples for 2 members for the second experiment using. Although the results of the two process are completely failed due to lack of data and the data standard respectively, We find out a more simple yet efficient method is to use the Face Recogntion and dlib python library which specializes in facial recognition system. The dataset used in Tensorflow project is then made use of as a resource for two different experiemnent one with the orignal dataset and one with difference in image's contrast. The experiment shows that with the origial dataset , OpenFace cannot recognizes any faces in the test pictures but the adjusted intensity using sauna filter effect dataset give out 100% accuracy. Besides, the jsTensorflow project is made along with the time of training models with the result of a game using socket to communicate among the server and game client.

Index Term,, Tensorflow, face recognition, models,

I. INTRODUCTION

Tensorflow is an open source machine framework which originally developed by Google Brain team within Google's AI organization [1]. There are a variety of projects in the Tensorflow ecosystems such as Tensorflow Lite, Swift for Tensorflow, Magenta. However, this report is only talking about Tensorflow Models and Tensorflow.js.

Face recognition and detection has become a part of humans everyday life which is a special feature of several reknown social application . According to Jared Bennett, Center for Public Integrity, there are 2 billion monthly users who upload 350 million photos every day on Facebook which provides the company an infinite sources of face dataset[3].

There has been a rise in objection detection and recognition, eg face recognition until the year of 2018. The Taylor Swift detector [4] is an example. It also used the Tensorflow Object Detection API[1], with MobileNet pre-trained model as a checkpoint for transfer learning. The problem of this project is it used the dataset orignally from Google Images with high resolution which creates a barrier of taking dataset not from Google Images and what is the quality of the image can be used to train. //Therefore, there should be some methods to figure out how suitable the dataset to reduce the time and cost wastefor training?

This research tries several different strategies to have a bigger overview of how each methods affect the result of face recognition. The main contribution of our report is that the

quality and quantity of the dataset should be considered carefully before using for Tensoflow models. Furthermore, in term of face recognition and detection, we found out a more reliable yet simple and fast method is to use OpenFace[5] and dlib[6] open source library. In the end, the purpose of this research is to figure out how Tensorflow and OpenFace work on face recognition and their applications.

In the first experiment with Tensorflow, the authors collect all the dataset by using Samsung Galaxy A6 with camera 16mp. Next, the authors preprocess images by resizing, labeling, splitting them into training and test sets, and then convert those images into test and train record file for feeding into Object Detection API. In the second experiment with Tensorflow, the collecting dataset step is the same with the first one. However, now there are only two classes with 498 samples in total. Both the above experiments using MobileNet on NVIDIA Quadro K2000 to train the model.The result of all above experiments all completely failed after nearly 200,000 training steps.

In the first experiment with OpenFace , the authors use the orignal dataset from the experiment with Tensorflow. Using the k-nearest-neighbors (KNN) algorithm for face recognition, the result failed again with the error that the number of neighors are bigger than the number samples needed. However, after modifying the image intensity, the accuracy when using OpenFace can reach to 100% with adjusted distance_threshold equals 0.4.

The rest of this report is structured in the following way. In section II, we present about the applications and some related works of Tensorflow and OpenFace. The important procedures of our Face Recognition works are demonstrated in section III using Tensorflow Objection Detection API [6] and OpenFace [5]. Section IV displayed experiment results and the analysis. The conclusion and related works are presented in section V.

II. Application and Related Works

A. Tensorflow

There are a lot of Tensorflow projects which are applied in a variety of fields. A JavaScript library, Tensorflow.js [7], which is implemented for training and deploying ML models in the browser and on Node.js. One of the Tensorflow.js project is Teachable Machine[8] which uses deeplearn.js library for everyone to explore machine learning. The experiment can be used live in browser without any coding required.

Furthermore, Tensorflow also provides several pre-trained classifers as well as models which anyone can use to train their own convolutional neural network objection detection from scracth. Edje Electronics who uses Tensorflow Object Detection Classifier to train the Pinochle Deck [9], create a playing card detecotr that can accurately detec nines, tens, jacks, queens, kings, and aces. The tutorial instructs the step for setting up and installation on Windows 10 which the authors also use as a tutorial.

B. OpenFace

Florian Schroff and et al. proposed a unified embedding for FaceRecognition and Clustering which also known as FaceNet[11]. OpenFace is a Python and Torch implementation of face reccognition with deep neural networks based on the above paper. Face Recognition library Matt Kiser et al. at Algorithmia [12], created automated front desk A.I using CMU OpenFace library that uses facial recognition to detect and greet the workers when they arrive at the office. The Face Recognition library based on OpenFace which is made by Adam Geitgey [13]. It is built using dlib's state-of-the-art face recognition built with deep learning which has a remarkable accuracy of 99.38% on the Labeled Faces in the Wild [14]. The authors also use this library as a mean to implement face recognition.

II. METHOD

In order to complete the Face Recognition systems of the humans in Android department, we use two different methods: Tensorflow Object Detection API and Face Recognition Python library. More details of the two strategies are described below.

A. Tensorflow Object Detection API

The process of creating a Face Recognition model using Tensorflow is briefly illustrated in Figure 1. In the preparing dataset term, the authors first collects the dataset by capturing several images using Samsung Galaxy A6. Then all the collected images are resized suitably before being labeled using LabelImg. The LabelImg will generate the bounding boxes to identify where the object is in our image and what label associates with that bounding box. In the training dataset term, all the labeled samples are converted into proper record files to feed the pre-trained Tensorflow models. Finally, after serveral hours of training process, we create the trained models from the saved checkpoint.

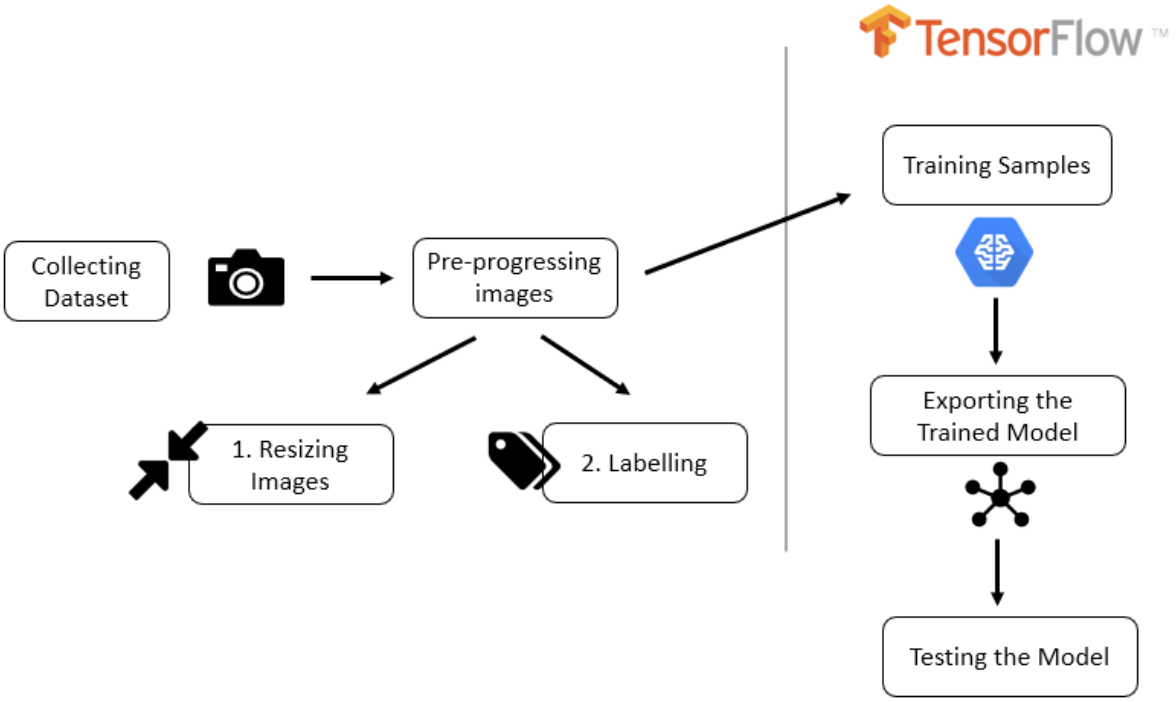

Fig. 1. Training model process

1. Object Dectection Model

1.1. Meta-architectures

In our report, we mainly forcus on two recent architectures: Faster-RCNN [15], and SSD (Single Shot Detector [16]).

1.1.1 Faster RCNN

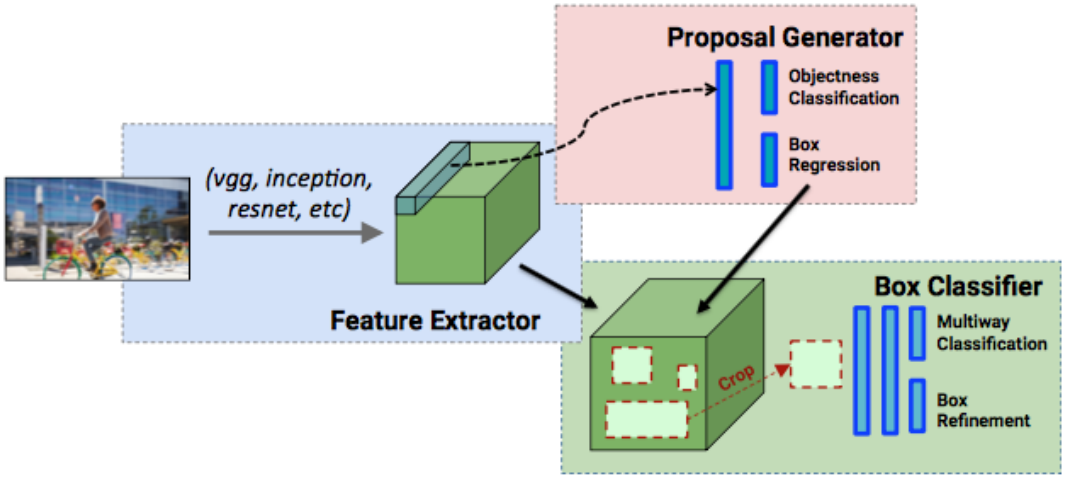

Fig. 2. Faster RCNN high level diagrams [17]

Faster R-CNN has two networks: region proposal network (RPN) for generating region proposals and a network using these proposals to detect objects[18] and the process are shown in Figure 3.

In the stage called Region Proposal Network, there are several feature extractors like inception, resnet, mobilenet. Images are processed through one of these features extractors and features from some selected intermediate stages are used for predicting class – agnostic box propsals. Box regression is necessary because it will significantly reduce mislocalization which are the dominant error mode. The result of Region Proposal Network are a ton of boxes that are predicted by the possiblity and its refinement.

In the second stage, box proposals are used to crop features from the same intermediate level feature map which are subsequently fed to the remainder of the feature extractor in order to predict a class and refine box for each specific class. The cropping algorithm must runs once per region to prevent duplicate computation and therefore, the runing time depends on the number of regions defined by the RPN.

1.1.2 Single Shot Detector (SSD)

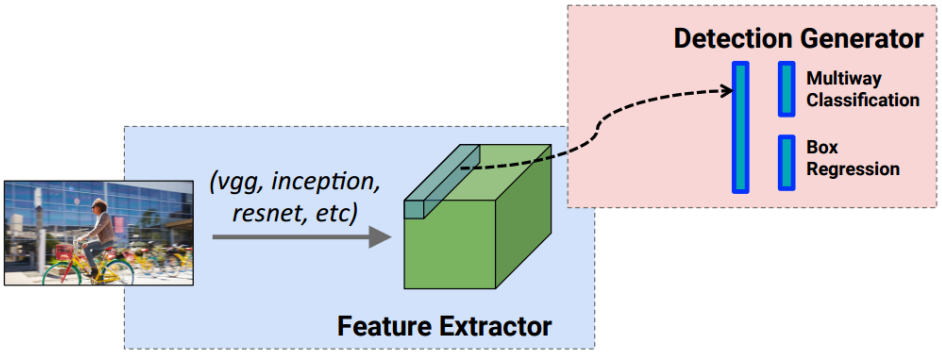

Fig. 3. SSD high level diagrams [17]

SSD uses a single feed-forward convolutional network to directly predict classes and object localization without requiring a second stage per-proposal classification operation. The network is an object detector but also classify those detected objects.

The bounding boss regression technique of SSD is inspired by Szegedy's work on Multibox [16], a method of fast class-agnostic bounding box coordinate proposals. The researchers created what we call priors as a starting point for bounding boss regression algorithm that closely match the distribution of the original ground truth boxes. MultiBox starts with the priors as predictions and attempt to regress closer to the ground truth bounding boxes. There are still more further maths and improvements about SSD as we do not go deep into how SSD works.

1.2. Model Selection

Tensorflow provides several object detection models (pre-trained models with different types of neural network architectures). However, the authors only consider two potential candidates of COCO-trained models which are Faster RCNN and SSD architectures with the feature extractor Inception_ V2 and MobileNet_V1 respectively.

Our purpose is to deployed the trained model on mobile since SSD-MobileNet offers a faster detection but with less accuracy according to a report by Google Research[15]. Therefore, this project will use Faster RCNN for a fast trial to see how everything works and SSD-MobileNet_V1 as a model for the whole training process.

B. Face Recognition and dlib

We use K-Nearest Neighbors algorithm (KNN) provided by Skikit-learn library and writtern in Python 3.6 [18] with the supported Face Recognition library built using dlib' state-of-the-art face recognition to train and get the face encondings of each image respectively.

1. K-Nearest Neighbors (KNN)

KNN algorithm is among one of the simplest algorithm for regression and classfication in supervised learning [19]. KNN is non-parametric which means it does not make any assumptions but bases on the model structure generated from the data.

An object is classsifed based on the majority votes of its neighbors (training set). The new example object will be assigned to the class with its most similar k nearest neighbors. Regression can also be applied for KNN and the result is the value of the average of the values in its k nearest neighbors.

2. Face Recognition

The following instructions of how Facial Recognition system works on OpenFace and dlib are extracted and adjusted by author understandings from the source [19]:

- **Step 1**

Using HOG algorithm to encode a picture by replacing each pixels by an arrow that show the direction of the brightness surrounding it getting darker. From a known generic HOG face pattern generated from lots of face images, it find the part of the simplified image that looks the most similar.

- **Step 2**

The face landmark estimation algorithm will be used to figure out 68 specific points that exists on every face. From the found landmarks, OpenCV 's affine transformation will use some basic image tranformations like rotation, scale and shear to try to make the eyes and lip always appear in the same location on each image that are centered as best as possible.

- **Step 3**

The centered face images are pass through a deep convolution neural network to obtain 128 measurements of each face which is also known as an embedding. The embedding is the generic representation of a human's face. They can be used to compare the same person in different pictures with the same face embeddings or two totally different people based on the distance between two face embeddings.

- **Step 4**

This final step, any basic machine learning classfication algorithm can be applied to learn the all the measurements from step 3. We will use KNN-classifier to train the embeddings of all the pictures. This classifier will be used to recognize the images that it has never seen before.

3. Features Training and Testing

The training dataset consists of 114 samples of 22 classes varying in the number of data samples of each person. After creating the dataset for each person, we create and train the KNN classifier with the support of Face Recognition library and dlib. The default algorithm to choose neighbors is Ball Tree, and the number of neighbors to be considered is 2.

III. Experiment and Results

1. Tensorflow Object Detection

a) FasterRCNN_Inception_V2

To have a better overview of how everything works, we first choose the Faster-RCNN model to train the "Pinochle Deck" as instructed from EdjeElectronics [10]. After 5 hours of training on CPU core i5 Ram 8GB, the models can detect three over six classes with a remarkable accuracy as shown in Figure 2.

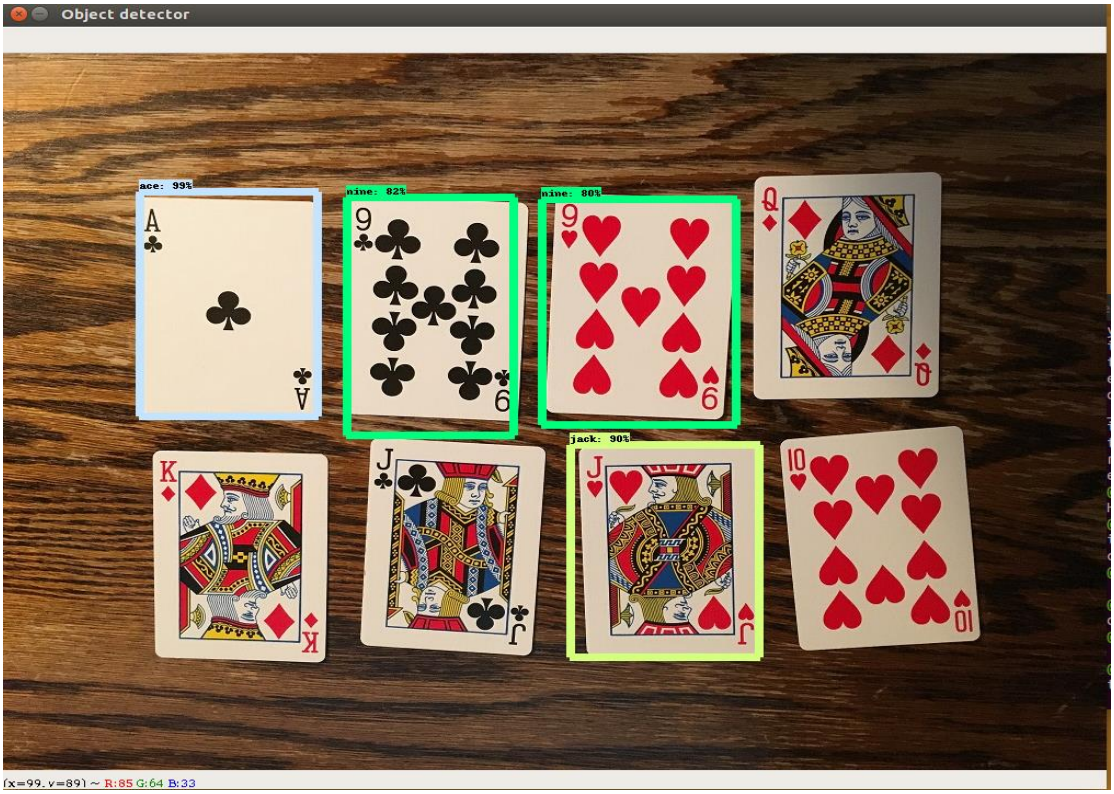

Fig. 4. “Pinochle Deck” card detector result

b) SSD-MobileNet

At first, we try the dataset with small number of samples, about 80 images with 80% train and 20% test on 8 classes and the training is processed on GPU NVIDIA Quadro K2000. Finally, after nearly 200,00 steps, we fail to recognize any faces even when the images used to detect is from the test or train dataset. There are a lot of problems with this approach. We face the problem of lacking dataset and we do not know whether our data samples does meet the standard requirement. The loss graph does not decrease over training time as it should be but vary dramatically.

Acknowledging the above problem, we create a more dense samples with around 500 images for 2 classes . However, after several testing process, there are no positive signal which continuously creates a models that cannot recognize any faces. The training process of 2 classes is shown in figure.

From the above failures, we infer that there are problems with the quality of data and we must figure out a more effiecent yet faster approach. Therefore, we investigate more into the Face Recognition library built with dlib.

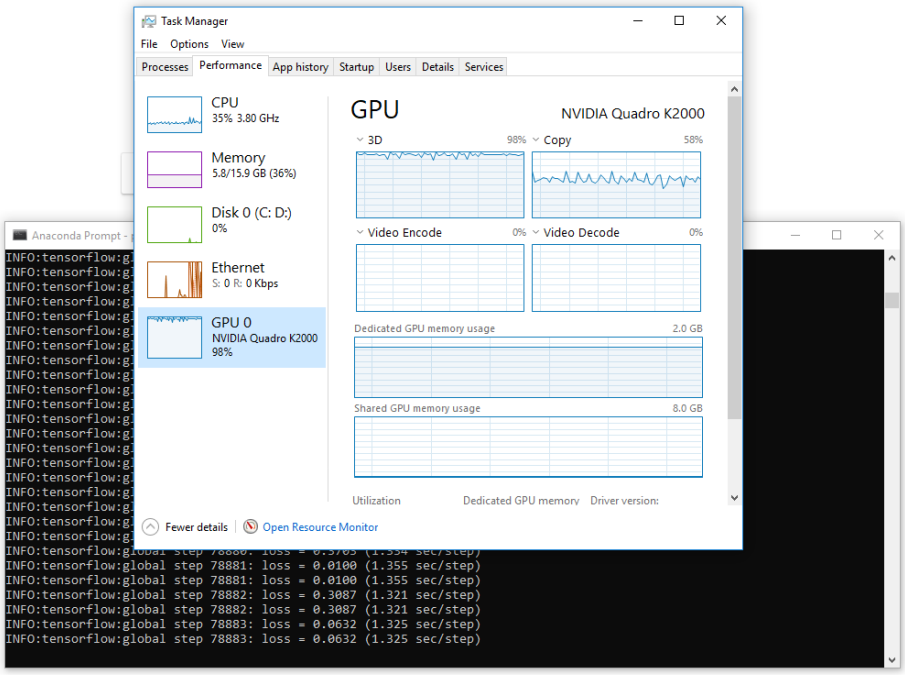

Fig. 5. Training process with 2 classes on NVIDA Quadro K2000

C. Face Recognition and dlib

For the first phase, we try to use original dataset with 80 images and separate them into 8 different classes. However, It raises the error “Expected n_neighbors <= n_samples, but n_samples = 1, n_neighbors = 2”. The error is shown in Figure [6].

This error means the NN algorithm cannot find any samples but returns itself as nearest neighbors which is something which we are not interested and the number of neighbors is augmented of one. The error leads us to the conclusion that there are problems with the dataset that the Face Recognition algorithm cannot extract any face encodings.

In the second phase, we adjust the effect of all trainning dataset by changing filter to sauna with intensity of 50. We also increases the number of classes to 22 varying in the number of samples in each class and the total data samples are increased to 114. The differences of before and after adjusting the effect of images is shown in the Figure [7]. The additional data samples that are added with the adjusted images to be used in the training set are shown in the Figure [8]. The representatives are chosen to be shown in Figure[7] and Figure[8].

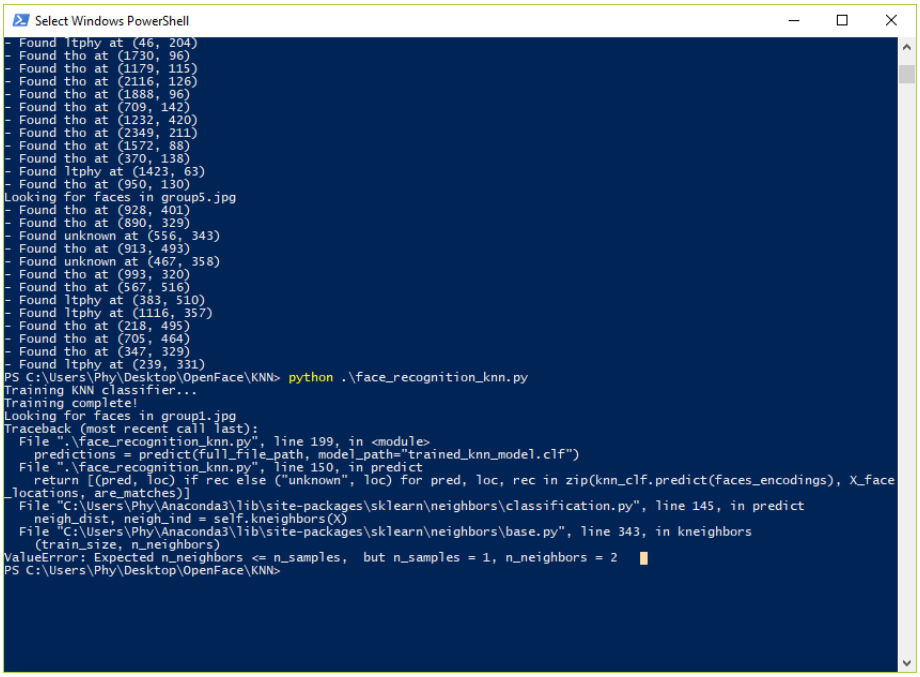

Fig. 6. Error when training with the orignal dataset

Fig. 7. Before and after adjusting intensity using sauna effect

Fig. 8. Additional Dataset

After reimplementing all the training process with the modified data, the result is shown in the Figure 9.

Fig. 9. Face recognition result of 22 classes.

TABLE I
RESULT SUMMARY

Training method	**Result**
Tensorflow with 8 classes and 80 samples	Fail
Tensorflow with 2 classes and 500 samples	Fail
Face Regconition with 8 classes and 80 samples	Error
Face Recognition with 22 classes and 114 samples	Success

Table I shows the final outcomes of 4 different experiments with the successful result of the final experiment using Face Recognition library with dlib. 8 classes used in the first and third experiment are reused in the fourth experiment. However, they are adjusted with their intensity.

D. jsTensorflow

The jsTensorflow project is made along during the training time of the above project. The jsTensorflow is based on the idea of Teachable Machine[ ] The game is made by using Unity Engine. In order to communicate among the jsTensorflow with Game created by Unity, we investigate into several different sockets to figure out which one is suitable for each platform. The comparision between sockets used on each platforms is shown in the Table II.

	Socketio	Simple WebSocket	WebSocketSharp
PC	Supported	Supported	Supported
WebGL	Unsupported	Supported	Unsupported

CONCLUSION

[1] https://www.tensorflow.org/

[2] https://github.com/tensorflow/models

[3] https://www.thedailybeast.com/how-facebook-fights-to-stop-laws-on-facial-recognition

[4] https://towardsdatascience.com/build-a-taylor-swift-detector-with-the-tensorflow-object-detection-api-ml-engine-and-swift-82707f5b4a56

[5] https://github.com/cmusatyalab/openface

[6] https://dlib.net

[7] https://github.com/tensorflow/models/tree/master/official

[8]https://js.tensorflow.org

[9] https://teachablemachine.withgoogle.com/

[10] https://github.com/EdjeElectronics/TensorFlow-Object-Detection-API-Tutorial-Train-Multiple-Objects-Windows-10

[11] https://cmusatyalab.github.io/openface/

[12] http://blog.algorithmia.com/2016/02/hey-zuck-we-built-your-facial-recognition-ai/

[13] https://github.com/ageitgey/face_recognition#face-recognition

[14] http://vis-www.cs.umass.edu/lfw/

[15] https://arxiv.org/abs/1506.01497

[16] https://arxiv.org/abs/1512.02325

[17] https://arxiv.org/pdf/1611.10012.pdf

[18] https://medium.com/@smallfishbigsea/faster-r-cnn-explained-864d4fb7e3f8

[18] https://www.researchgate.net/publication/51969319_Scikit-learn_Machine_Learning_in_Python

[19] https://www.quantinsti.com/blog/machine-learning-k-nearest-neighbors-knn-algorithm-python/

[19] https://medium.com/@ageitgey/machine-learning-is-fun-part-4-modern-face-recognition-with-deep-learning-c3cffc121d78