CSI 4130 Final Report

From Facial Landmarks Detection To 3D Animation

Tuan Pham
Oakland University
tuanpham@oakland.edu

Abstract

3D animation has numerous applications in the movie industry, education, entertainment, scientific visualization, creative arts, gaming, simulations, and more. Inspired by the results of CGMatter on his demonstration about facial tracking performance, I constructed my own convolutional neural network (CNN) and modified a CNN model from Yin Guobing to detect facial landmarks on a human face and generate a 3D animation in Blender. Two datasets are taken from Kaggle for this purpose, which consist of over 27,000 images along with facial key points. Initial results are promising with an error of 0.01%. I intended that my results benefit further work in generating 3D animation using machine learning and related tasks.

1. Introduction

3D animation is widely used for various needs in many fields. One can use animation to create his/her own movie, video game, or let the 3D character interacts with the audience when streaming if he or she does not want to show the face. Animation refers to the art of performing animated images, or 3D models, which frequently appear popular in animated movies, advertising programs, or video games. Animation, if understood in a more beautiful value, it represents an art with the flexibility of images moving vividly on the screen. It can simulate a story, a specific content to convey to those who receive a certain message [1]. Nevertheless, "Facial animation is a time-consuming and cumbersome task that requires years of experience and/or a complex and expensive set-up." [2].

Fortunately, due to the rapid developments of computer vision, machine learning systems can do the work that a human would typically do, or even potentially perform better than human in some cases. Thus, the work of this project focused on generating facial animation using machine learning, but the results can benefit further work in generating any type of 3D animation.

I propose a solution that uses CNNs to detect facial landmarks on a human face in real-time. And with the help of the OpenCV library [3], these landmarks or key points will be translated to keyframes in Blender which produce an animation accordingly. My CNN architecture is inspired by Yin Guobing's work and pre-trained VGG16 model on the public. The input of my algorithm are images of faces taken from a user's webcam in real-time, and I use a CNN to output predicted facial key points. The result taken from a video demo is shown in Figure 1.

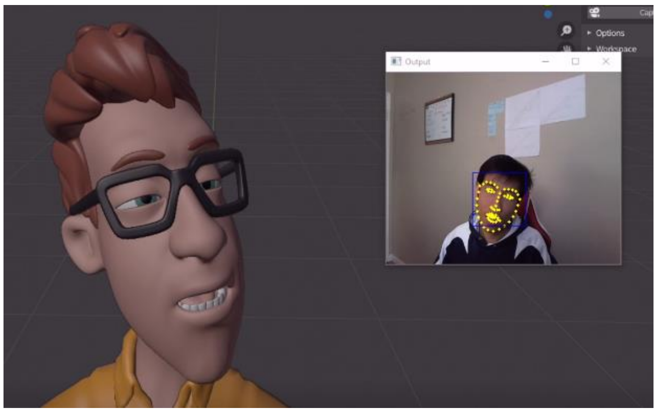


Fig. 1. An image of animation output from the model (Blender, OpenCV and CNN)

2. Related Work

The original idea was inspired form CGMatter when he illustrates how to generate a facial animation in Blender from facial landmarks that were manually marked on his face [4]. Therefore, I divided the related work into two sections as follows.

2.1 Generating Animation

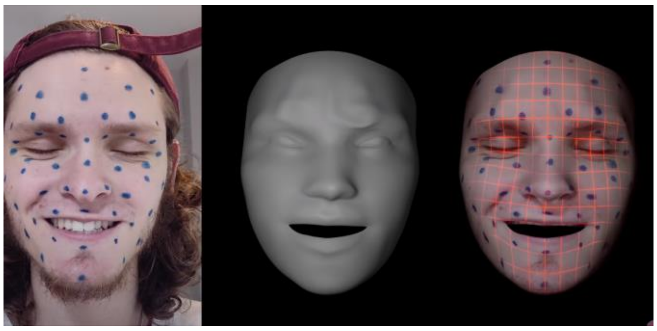


Fig. 2. A result image of CGMatter's work

First of all, the author put a sequence of dots with specific positions on his face as trackers. The input of his project was a pre-recorded video with the trackers. Then, the video footage was converted into an image sequence for Blender to process and render an animation clip.

2.2.1 Facial Landmark Detection

Instead of manually marking the key points as the author did, I would like to use a machine learning model, CNN in this case, to perform the same task automatically and visually. Facial landmark detection processes can be divided into two main steps, which are named as a template fitting process (which will be discussed in the next section) and cascaded regression algorithm [5]. The cascaded regression models iteratively update repeatedly pre-defined landmarks to detect landmarks. Valle et al adopt a heat map regressor approach in their model to regress the landmark coordinates by using a densely connected layer with shared weights among all heatmaps.

As will be discussed in the method sections, this project relies on the direct regression approach, which directly detects landmark coordinates represented by a vector of 136 elements (68x2). The initial testing CNN models were referenced from Yin Guobing Github [6] and the pre-trained VGG 16 model [7].

2.2.2 Face Recognition

The template fitting methods aim to learn the facial shape from the training set to fit input pictures during testing. The related works of template fitting algorithms are ASM [8] and AAM [9]. However, I would like to use the implementation of face recognition from OpenCV libraries due to the time constrain [10]. There are many face algorithms implemented by OpenCV including Haar Cascade, Eigen Faces, Fisherface, and Local Binary Pattern. Khan et al explained the OpenCV structure is organized loosely into five main elements which are shown in Figure 3.

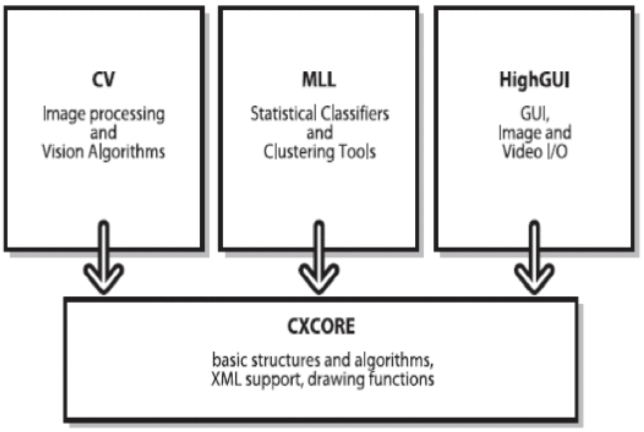

CV
Image processing and Vision Algorithms
MLL
Statistical Classifiers and Clustering Tools
HighGUI
GUI, Image and Video I/O
CXCORE
basic structures and algorithms, XML support, drawing functions

Fig 3. OpenCV Structure and Content

Now the question is how do OpenCV, CNN, and Blender work together? First of all, I would like to explain what facial landmarks are. They are a combination of key points that are illustrated in the first image in Figure 4. It consists of 68 key points indexing from 1 to 68. For example, the indexes from 18 to 22 and 23 to 27 represent eye browns. Then, a CNN model will predict these key points on several human face images which are taken continuously from a webcam thanks to OpenCV. Based on the movement of these key points in real-time, the Blender 3D character will animate accordingly.

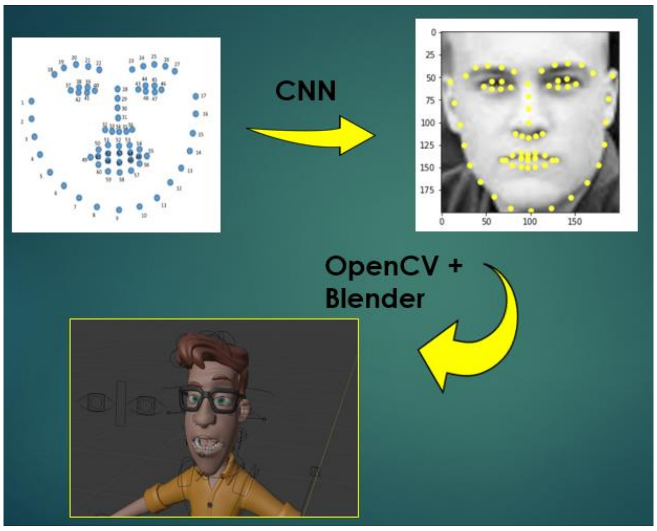


Fig 4. The workflow from facial landmark detection to animation.

3. Dataset

In order to train a convolutional neural network, a large dataset was needed to prevent potential overfitting. The first challenge to consider in this area was the scarcity of data that has facial landmarks annotation. One of the most popular datasets regarding this area is ibug 300W [11]. However, the dataset has only 600 images and some of them contain multiple faces which do not work for my model. Fortunately, I was able to find the first dataset provided by Dr. Yoshua Bengio of the University of Montreal that consists of over 7,000 images with 15 facial landmarks [12]. This dataset will be used to find a relevant CNN model.

3.1 Data cleaning

Not all images have an annotation or annotation that has complete 15 key points. Therefore, those images whose key points are not 15 would be dropped. The dataset's size reduced to about 3,000 images due to 57% missing values.

3.2 Data preparation

I found a better dataset consists over 20,000 cropped face images with 68 facial landmarks annotation. The dataset is provided by Yang Song and Zhifei Zhang [13] and would be used to train the final model. For the training time reduction, over 10,000 images with ages range from 18 to 35 were extracted from the original dataset. In addition, to prepare the dataset ready for the training phase, each image needed to be assigned with the right key points stored in the attached text file. The source code for data preparation can be found in appendix A.

4. Method

A CNN is a set of convolutional layers that are stacked on each other and use an activation function (ReLu was used in this project) to activate critical numbers in nodes. Each layer, after passing the trigger functions, will generate more abstract information for the next layers. The input of the next layer is the result of the previous convolutional layer so that we have local connections. Thus, each neuron in the next layer generated from the result of the filter is applied to a local image area of the previous neuron.

Given an input image, I designed two CNN models that are inspired by the architectures of Yin Guobing [6]. My next step is to find the best model that gives a highly accurate prediction.

4.1 Run on the First Dataset

The first dataset was used to test the performance of Yin Guobing's model [6] and the pre-trained VGG16 model [7]. The last fully connected layers of VGG16 were replaced by a fattened layer and two dense layers. The problem was that both models predicted the same key points for every image. I noticed the similarity between the two models that is described in Figure 5.

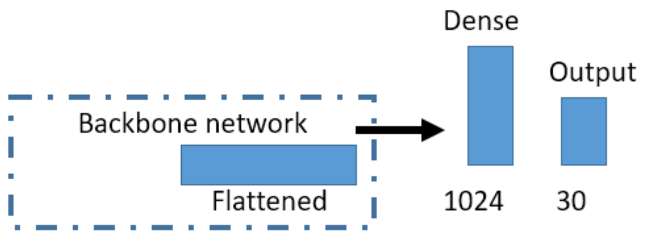
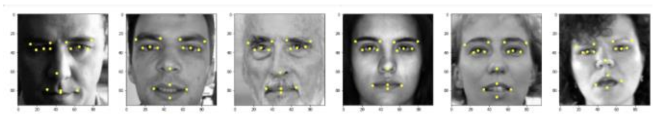


Fig 5. A general model architecture of modified VGG16 model and Yin Guobing's model.

I came up with a new model that did not have dense layers, which will be shown in the next section. The output of my model was from a Conv2d layer with the shape of (1, 1, 30), and below are the results with 15 key points.



The output images with 15 predicted key points of the new model.

4.2 Run on the Second Dataset

A portion of the second dataset (2,000 200x200 images) was used to confirm my observation. Since the key points increased from 15 to 68, the new model needs to be modified to fit the output shape of (1, 1, 136). The new model architecture can be seen in Figure 6.

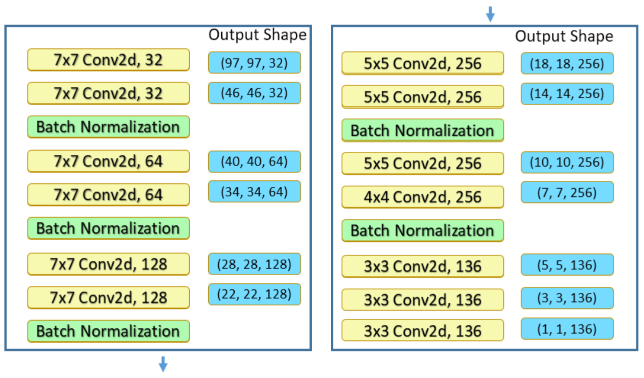


Fig 6. The new model's architecture with the input shape of (200, 200, 1)

Both models used the L2 loss function (also known as mean square error loss) and Adam Optimizer.

$$L = \sqrt{\sum i \, (y_{pred_i} - y_i)^2}$$

Where y_{pred_i} are the predicted key points from the model and y_i are the original key points. My goal is not to get the accuracy of 100% but to get the closest predicted coordinates compared to the true ones as much as possible. The followings are the prediction outputs of the two models.

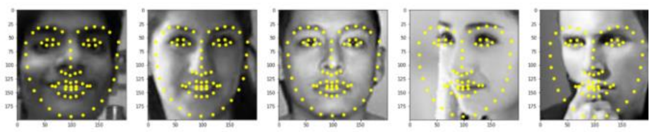


Fig 7. The output images with 68 predicted key points from the Yin Guobing's model. The model renders the same key points for every image.

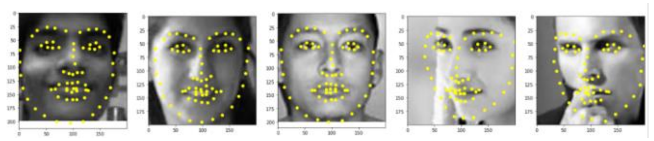


Fig 8. The output images with 68 predicted key points from my model give a decent accuracy.

Through various experiments, I've learned that Yin Guobing's model was too complicated against the given dataset. Thus, I removed a few convolutional layers and modified the kernel size. Finally, the modified model could give some good predictions. Four experiments will be carried out in the next section in order to conclude what is the best model and select it to train the entire dataset. Figure 9 includes the architecture of the modified model.

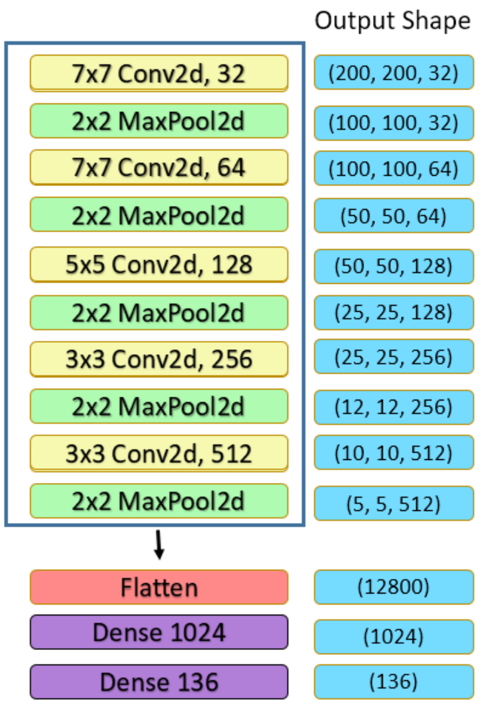


Fig 9. The modified model's architecture with the input shape of (200, 200, 1)

5. Experiments and Results

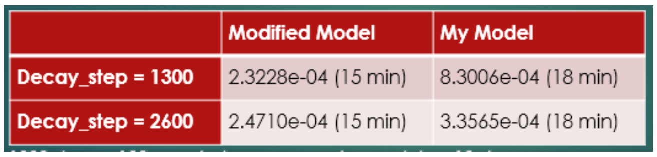
5.1 Models Comparison

I trained both the modified model and my model using Adam Optimizer with hyperparameters shown in Table 1.

TABLE 1 INITIAL HYPERPARAMETERS

Hyperparameter	Value
Epochs	250
Learning rate schedule	polynomial
Initial learning rate	0.0001
End learning rate	0.00001
Batch size	128
Num Training	1600

With the batch size of 128 and 1,600 training data, each epoch has 13 steps. Therefore, the model would reach the end learning rate after 100 epochs and 200 epochs with a decay step of 1300 and 2600 respectively. The mean square errors and training times were recorded in Figure 10.



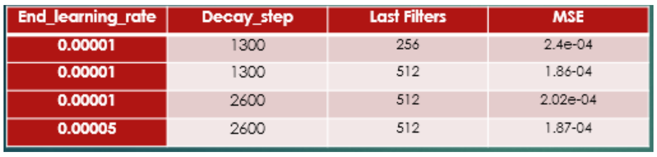
	Modified Model	My Model
Decay_step = 1300	2.3228e-04 (15 min)	8.3006e-04 (18 min)
Decay_step = 2600	2.4710e-04 (15 min)	3.3565e-04 (18 min)

Fig 10. The results of mean square errors and training time of each model.

It is obvious that the modified model has smaller errors as well as run faster than my model about three minutes. As a result, I decided to select it as the final model.

5.2 Training on 2,000 images dataset

With the same hyperparameters displayed in Table 1, I run four more experiments to learn the problem better.



End_learning_rate	Decay_step	Last Filters	MSE
0.00001	1300	256	2.4e-04
0.00001	1300	512	1.86-04
0.00001	2600	512	2.02e-04
0.00005	2600	512	1.87-04

Fig 11. The results of mean square errors when changing the decay step and the last filters.

I observed the improvement in the second experiment. And I noticed the model stop learning after 100 epochs when analyzing the graph in Figure 12. Thus, the learning rates were adjusted in the next two experiments. I decided to stop at the fourth experiment since there was no improvement and continue to train on the entire dataset.

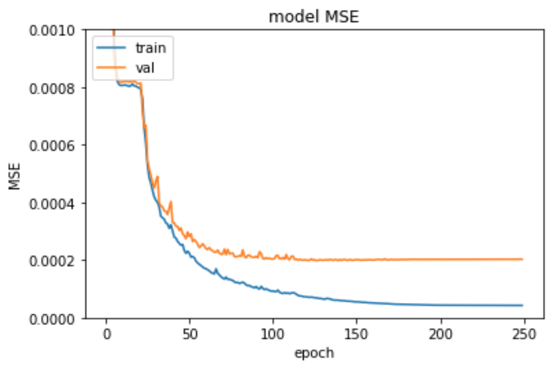
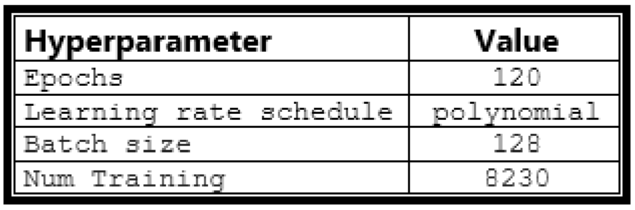


Fig 12. Results of train and validation learning curves in the 2nd experiment.

5.3 Training on 10,000 images dataset

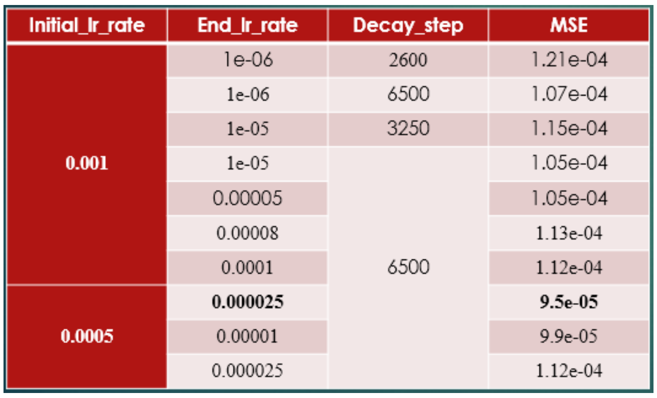
Various experiments were carried out on over 10,000 images dataset to find the best-fit learning rate. Due to the size of the dataset was increased significantly, the number of epochs was adjusted to 120 which shown in Table 2.

TABLE 2 INITIAL HYPERPARAMETERS



Hyperparameter	Value
Epochs	120
Learning rate schedule	polynomial
Batch size	128
Num Training	8230

The number of images in the training set increased to 8,230; hence each epoch now has 65 steps. Therefore, the model would reach the end learning rate after 50 epochs and 100 epochs with a decay step of 3250 and 6500 respectively. Figure 13 shows ten experiments to find the best range of learning rate.



Initial_lr_rate	End_lr_rate	Decay_step	MSE
0.001	1e-06	2600	1.21e-04
	1e-06	6500	1.07e-04
	1e-05	3250	1.15e-04
	1e-05	6500	1.05e-04
	0.00005		1.05e-04
	0.00008		1.13e-04
	0.0001		1.12e-04
0.0005	**0.000025**		**9.5e-05**
	0.00001		9.9e-05
	0.000025		1.12e-04

Fig 13. The results of mean square errors when changing the decay step and the learning rates.

The best hyperparameters among these experiments suggest that the learning rate should range from 0.0005 to 0.000025 in 100 epochs. The learning curves of the best experiment can be seen in Figure 14.

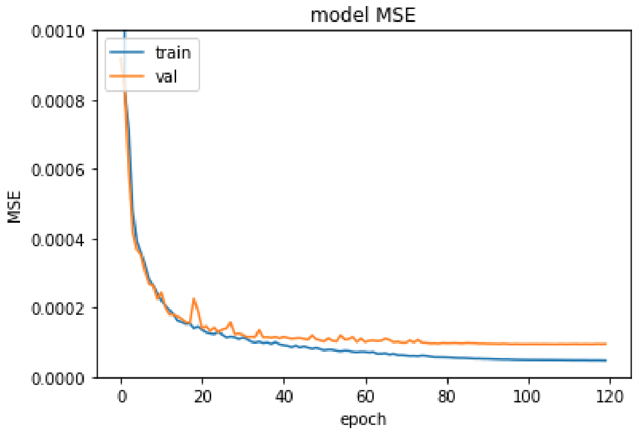


Fig 14. Results of train and validation learning curves in the 8th experiment.

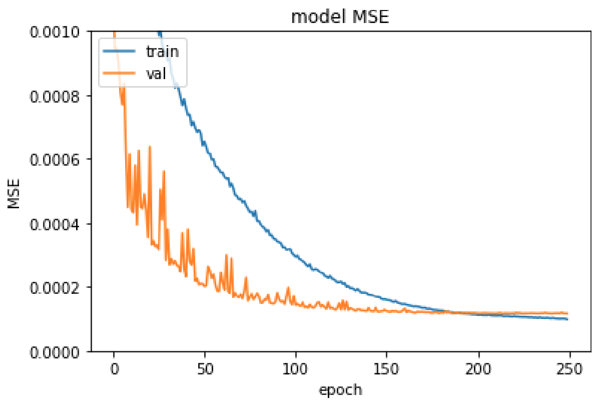


Fig 15. Results of train and validation learning curves in the last experiment.

With the given hyperparameter at the 8th experiment, I added a drop-out layer to the model and run it on 250 epochs to prevent overfitting, but the result did not improve which is shown in Figure 15.

6. Conclusions, Challenges, and Extensions

I find that I was able to implement a method to detect facial key points on a human face via a webcam output. Although I could not tune the hyperparameters to make the training and validation errors converged, the result was satisfactory with an average mean square error of 0.01%. I observe that the model is unable to predict the key point at some angles where the face moves too much to the left or right. Taking advantage of additional time and other resources, the immediate next step would be to find a more occluded angled faces dataset to develop a better model.

The final result can be seen in my Github repository, appendix A. The animation in the short demo is not very smooth. However, my purpose is to point out that the idea is applicable and can be expanded in many circumstances. Interested readers can develop a better model to apply to the entire human body instead of just the face in this project.

Another interesting extension would be to show the animation in real-time when streaming. The fact that many streamers are shy to show their face; hence, this idea is an incredible solution. Nonetheless, the result I have achieved is far from a final product due to the computational burden. Further research is needed to reach better implementations.

It may be many years until the first Toy Story was released in 1995, "Four years in the making, the 77-minute film required 800,000 machine-hours just to produce a final cut." [14]. However, based on the powerful combination of CNN models and existing libraries, the animation industry will explode sooner than we think.

Appendix A

The Jupyter Notebook with corresponding Python code used to complete this report can be found at my Github repository: https://github.com/tuan0101/Machine-Learning/tree/main/Facial%20Landmark%20Detection

References

[1] L. Pardew, "Character emotion 2D and 3D animation." Boston, Mass: Thomson Course Technology, 2008, pp. 1-8.

[2] J. serra, O. Cetinaslan, S. Ravikumar, V. Orvalho, D. Cosker, "Easy Generation of Facial Animation Using Motion Graphs." *Computer Graphics Forum,* vol. 37, no. 1, Feb. 2018, pp. 97-111.

[3] https://opencv.org/

[4] CGMatter. "Blender 2.8 Facial motion capture tutorial," Youtube, 2019. [Video file]. Available: https://www.youtube.com/watch?v=uNK8S19OSmA&ab_channel=CGMatter

[5] R. Valle, J. Buenaposada, L. Baumela, "Cascade of encoder-decoder CNNs with learned coordinates regressor for robust facial landmarks detection," *Pattern Recognition Letters,* vol. 136, Aug. 2020, pp. 326-332.

[6] https://github.com/yinguobing/cnn-facial-landmark

[7] https://www.tensorflow.org/api_docs/python/tf/keras/applications/VGG16

[8] T. F. Cootes, C. J. Taylor, D. H. Cooper and J. Graham, "Active shape models-their training and application", *Comput. Vis. Image Understand.*, vol. 61, no. 1, pp. 38-59, Jan. 1995.

[9] T. F. Cootes, G. J. Edwards and C. J. Taylor, "Active appearance models", *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 23, no. 6, pp. 681-685, Jun. 2001.

[10] M. Khan, S. Chakraborty, R. Astya, and S. Khepra, "Face Detection and Recognition Using OpenCV," presented at the 2019 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS), Oct. 2019, doi: 10.1109/icccis48478.2019.8974493.

[11] https://www.ecse.rpi.edu/~cvrl/database/Facial_Landmark_Databases.html

[12] https://www.kaggle.com/drgilermo/face-images-with-marked-landmark-points

[13] https://www.kaggle.com/jangedoo/utkface-new

[14] B. Snider, "The Toy Story Story", https://www.wired.com/1995/12/toy-story/