MSBD5016 Final Paper: Computer Vision for Facial Keypoint Detection

LAM Chun Ting, Jeff
ctjlam@connect.ust.hk

WONG Ka Wing
kwwongbv@connect.ust.hk

Abstract

This paper presents the approach we used in training computer vision models for facial keypoint detection. Although it is an old topic in the computer vision field, there is a huge demand since the COVID-19 outbreak. With the collaboration between medical, robotics, and computer vision, the requirement of facial keypoint detection was raised. Data augmentation is applied on the training data before conducting model training, so as to expand the training data and also generate additional sample images with different pose and scaling. This data preprocessing step aims at enabling the resulting model to generalize better to other datasets. Then, different models, such as CNN, ResNet and VGG are trained using the preprocessed training data, with their architecture and performance compared and contrasted. Instead of focusing on the training and testing loss, the ultimate goal is to find out a robust, scalable, and reliable model in detecting the facial keypoints in real-world application.

1. Introduction

1.1. Motivation

Since the outbreak of the COVID-19 pandemic, there is high pressure on the health care system. Doctors and nurses are in serious shortage. Training new staff takes lots of time and resources which cannot be a possible solution in order to soften the medical stress. Therefore, in order to figure out a quick and feasible solution to the issue, there is a huge potential incombining the computer vision research, and robotic research. The potential of robotics providing assistance in conducting basic diagnosis and initial screening of patients is actively explored, and therefore to reduce human to human interaction, and ease the medical staff workloads.

Facial keypoint detection would be the building block of medical application. However, as it comes to medical usage, reliable and diverse accurate results are more important than a training value, such as loss and accuracy, when it comes to usage, which is usually applied in camera streaming. Therefore, robust, scalable, and efficient models have to be found out to apply the tool in the real-world application.

During the pandemic, there are lots of medical checks related to the human face. There is medical research stating that sore eyes is a significant ocular symptom of COVID-19 [6]. Therefore, Identifying the position of eyes can help the robot to conduct basic checks for eye diagnosis, and identifying the position of nose and mouth could be helpful for possible collection of specimens for testing with the technology research of soft robotics [7] in recent years.

1.2. Problem Statement

Our objective is to identify the x-y coordinates of 15 facial features such as eyebrows, nose and lips for a given image. An example is provided below for reference.

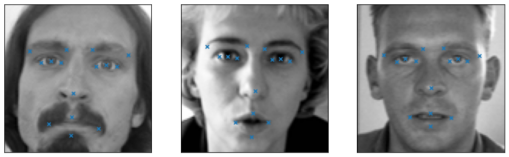


Figure 1. Sample facial keypoint extraction

Accuracy of the resulting model would be assessed by the root-mean-square error between the predicted x-y coordinates of the 15 facial features and the ground truth provided.

2. Dataset

The dataset that we used in our project is provided by the University of Montreal and is hosted in Kaggle. There are 7,049 single channel images in the training data represented by 96x96 pixels. There are also 1,783 single channel testing images by 96x96 pixels. The x-y coordinates of 15 facial characteristics, such as eyebrows, nose and lips for each training image are provided along with the training set. In other words, as each point contains x-y, then it will be an output size of 30.

The 15 feature names listed in Table 1.

Index	Feature Name
1	left eye center
2	right eye center
3	left eye inner corner
4	left eye outer corner
5	right eye inner corner
6	right eye outer corner
7	left eyebrow inner end
8	left eyebrow outer end
9	right eyebrow inner end
10	right eyebrow outer end
11	nose tip
12	mouth left corner
13	mouth right corner
14	mouth center top lip
15	mouth center bottom lip

Table 1. 15 Facial Keypoint Features

As we explore the training data, a number of problems and challenges are identified. Firstly, the size of the training data is quite limited which may undermine the ability to generalize the trained model to other datasets. Therefore, we would expand the training data through data augmentation. Secondly, for a subset of the training data, the ground truth of facial keypoints provided is not quite accurate which may undermine accuracy of the trained model on other datasets. An example is attached in Figure 2 for reference.

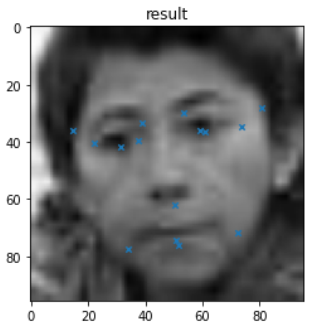


Figure 2. Inaccurate keypoint coordinates in training data

3. Methodology

In this project, we follow the flow shown in figure 3. We will first perform data augmentation (see section 3.2), in order to increase the training set quality. Then we will train different kinds of models, and in this project, a total of 8 models will be trained and tested (see section 3.3). After having all trained models, we will test it in a test set (see section 4.2), and also with real life examples (see Appendix A).

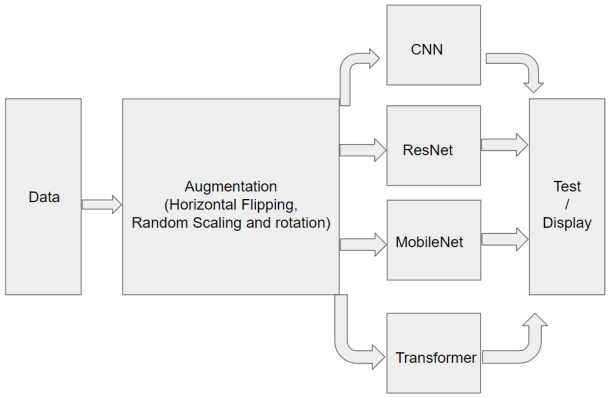


Figure 3. Training Flow

3.1. Training Setup

For all model training, we are using the following training setup.

Framework	Tensorflow (Except ViT), Pytorch (For ViT)
Optimizer	Adam
Epochs	400
Loss	Mean Squared Error
Metrics	Root Mean Squared Error
EarlyStopping	Based on validation loss (Patience 20)
Learning Rate Reduction	Based on validation Metrics (Patience 5)

Table 2. Training Setup

For all models, we start training with a learning rate 0.001, and 400 epochs, with 256 batch size.

3.2. Data Augmentation

As a data preprocessing step, we have applied data augmentation to the training data before doing the model training. The purpose of conducting data augmentation is twofold. On one hand, we would like to expand the size of the training data. And on the other hand, we would like the resulting model to generalize better to other images which could have different scaling or pose as compared to those images in the training data. The total training set after augmentation will be 21,144 data.

3.2.1 Horizontal Flipping

The first augmentation method we used is horizontal flipping. Given all the training images, we perform horizon-

tal flip, and also apply a flipping to the output keypoint for training by the formula below.

$$\forall x, flipped_x = 96 - x$$

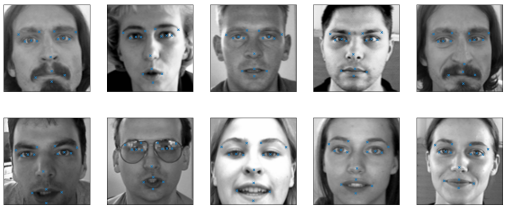


Figure 4. Horizontal Flipping

3.2.2 Rotation and Scaling

The second augmentation method we used is random scaling and rotation. Given all the training images, we perform a random scaling factor between (0.8,1.2) and a random rotation between angles between (-15, 15) degrees. The library we are using is "imgaug".

Figure 5. Rotation and Scaling

3.3. Models

Regarding the models used, we mainly get our inspiration in three ways. First is the research from Longpre et. al [5]. Second is the winning team for the Kaggle competition karanjakhar [4]. Third is curiosity trying out transfer learning in powerful models.

We have adopted two approaches as we train the model.

For ordinary CNN, LeNet and simple VGG, we have utilized the Keras API for building each layer and formulating the whole architecture.

For ResNet, MobileNet, and Vision Transformer(ViT), we have tried to adopt transfer learning and utilized the pretrained model provided either by Keras or Torch Vision. New convolutional layer is also trained on top of the pretrained model.

In short, we have tested out simple CNN, LeNet, LeNetDrop, SimpleVGG, advance CNN, ResNet, MobileNet, and ViT.

3.3.1 Simple CNN

Layer	Name	Size
1	Input	96x96x1
2	Hidden	500
3	Output	30

Table 3. Simple CNN Architecture

This model is from Longpre et. al [5] which serves as a baseline model. It is a very simple model, and should be under all models we built. There are a total of 4,623,530 parameters.

3.3.2 LeNet

The model is implemented with reference to the LeNet-like architecture proposed by karanjakhar [4]. It is a 5 layers version. There are a total of 2,310,854 parameters.

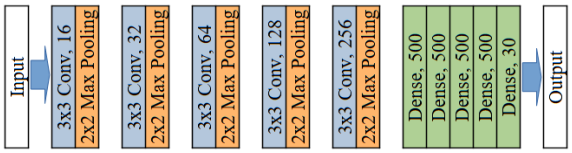


Figure 6. LeNet Architecture

3.3.3 LeNetDrop

As compared with the LeNet model based in section 3.3.2 , a dropout layer (p=0.1) is added after each of the convolutional layers for better generalization and to avoid overfitting. There are a total of 2,310,854 parameters.

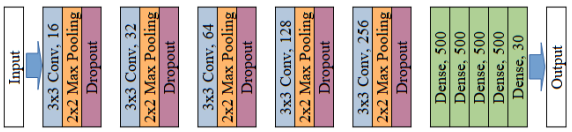


Figure 7. LeNet (with dropping) Architecture

3.3.4 SimpleVGG

It is VGG-style convolutional neural networks. There are a total of 20,821,598 parameters.

3.3.5 Advance CNN

It is a model from competition gold medal winner.There are a total of 7,268,670 parameters.

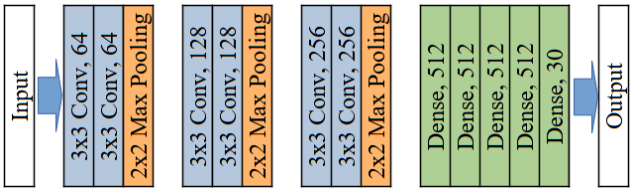


Figure 8. Simple VGG Architecture

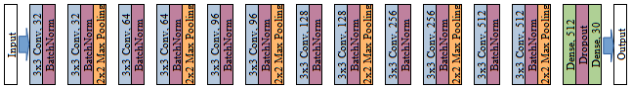


Figure 9. Advance CNN Architecture

3.3.6 ResNet

The ResNet model is trained by utilizing the pre-trained model ResNet50V2 provided by Keras. And in the classification head, there are 5 layers. The first coming after from the classification head is a global average pooling 2D, then with a fully hidden connection layer 512 neurons, and batch normalize with Leaky ReLu activation, and last is 30 connected neurons. There are a total of 24,631,332 parameters.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
		3×3 max pool, stride 2				
conv2_x	56×56	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 10. ResNet Architecture [2]

3.3.7 MobileNet

Aiming to apply the facial keypoint detection model on mobile devices with limited computing power, we have tried to train the MobileNet model for the facial keypoint detection task which is considered less computationally intensive. The MobileNet model is trained by utilizing the pre-trained model MobileNetV2 provided by Keras. In the classification head, it will be flatten and connected with 7 fully connected hidden layers (1024, 512, 256, Dropout, 128, 64, Dropout, 32, 30). There are a total of 14,755,844 parameters.

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

Figure 11. MobileNet Architecture [3]

3.3.8 Vision Transformer

As transformers gain increasing popularity in the field of computer vision, we have tried to train a vision transformer (ViT) for facial keypoint detection as well, trying to compare its performance with more traditional CNN architecture. There are a total of 86,567,656 parameters.

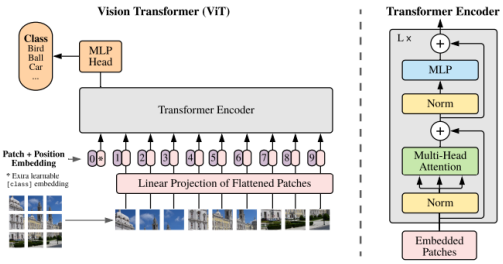


Figure 12. Vision Transformer Architecture [1]

3.4. Testing Methodology

By having any real-life image, we hope to detect and point out the facial keypoints from the image. And therefore, the process will be divided into 5 parts. The first is extracting faces from the image, and we are using a python library opencv Cascade Classifier. That function will return a list of faces coordinates in the image. For each returned face, we resize it to 96x96, which fits into the model of our project. The model will be specifically specified, and return the 15 facial keypoints coordinates. After having the result in 96x96 image, they will be resized back on the original image, and plot the result in the original image. The actual

result can be refer to Appendix A.

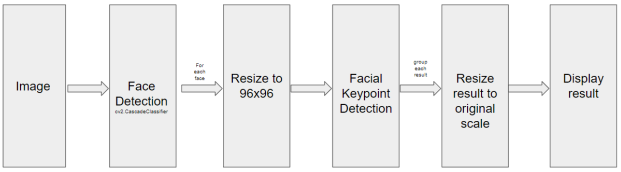


Figure 13. Flow diagram of testing procedure

4. Result

4.1. Training Score

Model	Epoch (Stopping)	Training Loss	Training RMSE	Validation Loss	Validation RMSE
CNN (LeNet Drop)	65	7.68	2.77	21.63	4.65
CNN (LeNet)	77	13.46	3.67	49.81	7.06
Vision Transformer	10	141.79	8.83	78.74	8.87
CNN (Advance)	66	165.48	13.50	89.86	9.48
Transfer Learning (ResNet)	92	198.50	14.21	101.74	10.09
Transfer Learning (MobileNet)	154	220.70	14.86	135.53	11.64
CNN (Simple VGG)	24	231.38	15.21	158.98	12.61
CNN (Baseline)	40	225.45	15.02	161.84	12.72

Figure 14. Training and validation result of each model

From figure 14, we can see that performance of LeNet, both with and without dropout, outperform the other model architectures, with validation RMSE of around 4.65 with dropout and 7.06 without dropout.

The performance of other models are similar to the task of facial keypoint detection, with validation RMSE greater than or roughly equal to 10.

With early stopping, model training mostly stops at around 70 epoch.

Instead of taking a look into the values itself, we would like to investigate real examples. (refer to appendix A)

4.2. Model Result

4.2.1 Simple CNN

In this model, we notice that the model is too simple and is far from enough to learn about the human face, tracking the facial keypoints. The computation time of this model is 0.126s.

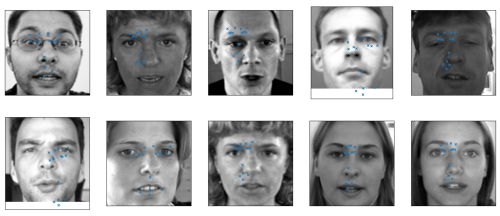


Figure 15. Test Set Result with Simple CNN

4.2.2 LeNet

In this model, we notice that it is able to detect the points for most of the faces. It learns pretty well except for the top left face with mouth and glasses. It is believed that the high frequency data in the tooth and shadow of his nose became overwhelmed then the original features. The computation time is 0.099s.

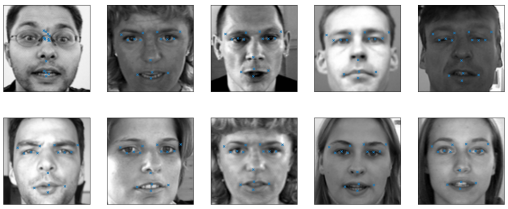


Figure 16. Test Set Result with LeNet

4.2.3 LeNetDrop

In this model, we notice that it is able to detect the points for most of the faces. It learns pretty well even than the model without drop. It is believed that the overfitting problem is solved by the drop layers.The computation time is 0.095s.

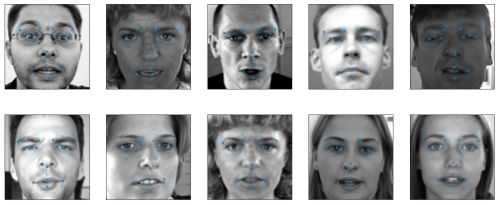


Figure 17. Test Set Result with LeNetDrop

4.2.4 SimpleVGG

In this model, we notice that it learns nothing about the human face, tracking the facial keypoints. The computation time is 0.144s.

4.2.5 Advance CNN

In this model, we notice that it is similar to the LeNet model, and able to detect the points for most of the faces. It learns pretty well except for the top left face with mouth and glasses. It is believed that the high frequency data in the tooth and shadow of his nose became overwhelmed then the original features. The computation time is 0.178s.

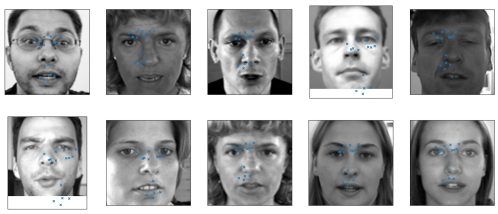


Figure 18. Test Set Result with Result with SimpleVGG

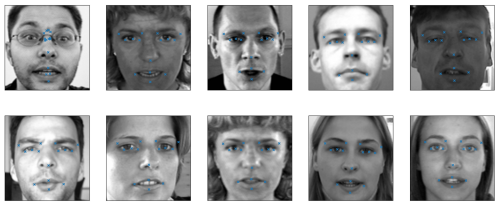


Figure 19. Test Set Result with Advance CNN

4.2.6 ResNet

In this model, we notice that it is similar to the LeNet model, and able to detect the points for most of the faces. It learns pretty well except for the top left face with mouth and glasses, but the result is a bit closer than the target points. However, refer to The result in appendix A, the generalization is weaker than Advance CNN and LeNet. The computation time is 0.477s.

Figure 20. Test Set Result with ResNet

4.2.7 MobileNet

In this model, we notice that it learns nothing about the human face, tracking the facial keypoints. However, the funny fact is that the RMSE loss is in the top 3. The computation time is 0.411s.

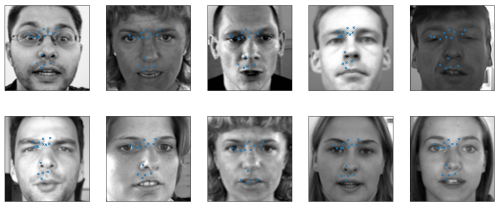


Figure 21. Test Set Result with MobileNet

4.2.8 Vision Transformer

Similar to the case of MobileNet, the model is not effective in extracting facial features but the validation RMSE is unexceptionally good.

Figure 22. Test Set Result with ViT

4.3. Discussion

There are two fun observations in this project. The first observation is that, when it comes to a more complicated model, it actually learns nothing. It is believed to be an overfitting problem, as the training loss in training increases when more epochs run. The second observation is that, we notice that in terms of performance, the value result does not have a positive correlation with the RMSE loss result. The most significant example comes to ViT, although it has a top 3 performance in terms of RMSE values, the actual result learns nothing. The other example is AdvanceCNN and LeNet. The actual result is similar, but the value result is far from each other. Therefore, when a computer vision problem comes to actual implementation, actual testing takes a more important proportion than only considering the loss values, with realising what kinds of data would be detected incorrectly.

Among all models, the best model is LeNetDrop. No matter the generalization result and the actual value result, it is nice. It is believed that the model fits the problem and it solves the overfitting issue compared to other models thanks to the drop layer. And the most important is that the computation time of computing results is the fastest among all models.

Figure 23. LeNet Drop Result on Girl Group (Collar)

Figure 24. LeNet Drop Result on Boy Group (Error)

Figure 25. LeNet Drop Result on Boy Group (Mirror)

Figure 26. LeNet Drop Result on Professor Face

5. Conclusion

Facial keypoints detection is an old problem, but due to the pandemic, it is an important foundation for combining the usage in medical and robotic fields. Once it comes to clinic usage with robotics technology, instead of focusing on the score result, it is important to notice the real world result. The shortage of machine learning computer vision models has to be realized, and it is important to ensure robustness, scalability, and efficiency.

In this project, we have tried different models, from building basic models to using knowledge in transfer learning. However, this problem seems to be not as difficult as expected. When more complicated models come in training, the result is much worse than a simpler model. The problem is believed to be an overfitting problem, as when more training goes on, the loss increases.

LeNetDrop outperformed the other models. In terms of values, it is the best with smallest loss, fastest computation, and the most generalized result from real-life examples, apart from the original training dataset. It fits with the requirements that would be achieved when starting with the project.

Finally, with this report, we hope that it provides a good foundation in the field combination of computer vision, robotics, and medical. We also hope that the pandemic can be over soon, and it can greatly reduce the stress in the medical area.

Acknowledgements

Thank you for Professor CK Tang from The Hong Kong University of Science and Technology for the support of the project

References

[1] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[3] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[4] Karanjakhar. Facial keypoint detection, Jun 2019. URL: `https://www.kaggle.com/code/karanjakhar/facial-keypoint-detection`.

[5] Shayne Longpre and Ajay Sohmshetty. Facial keypoint detection. *Facial Detection Kaggle competition*, 2016.

[6] Shahina Pardhan, Megan Vaughan, Jufen Zhang, Lee Smith, and Havovi Chichger. Sore eyes as the most significant ocular symptom experienced by people with covid-19: a comparison between pre-covid-19 and during covid-19 states. *BMJ open ophthalmology*, 5(1):e000632, 2020.

[7] Jianshu Zhou, Wei Chen, Shing Shin Cheng, Lingbin Xue, Michael C. Tong, and Yunhui Liu. Bio-inspired soft (bis) hand for tele-operated covid-19 oropharyngeal (op) swab sampling. *2021 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, 2021. `doi:10.1109/robio54168.2021.9739351`.

Appendix A: Model Performance on real-life examples

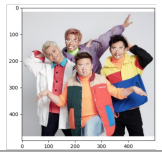
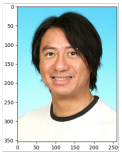
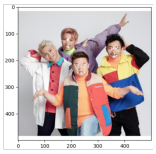
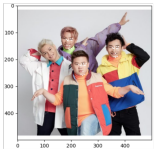
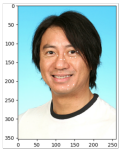
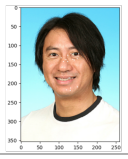
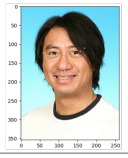
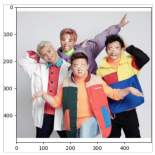
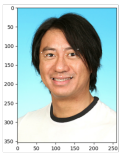
Model	Collar	Error	Mirror	Professor
CNN (LeNet Drop)				
CNN (LeNet)				
Transfer Learning (MobileNet)				
CNN (Simple VGG)				
CNN (Baseline)				
CNN (Advance)				
Transfer Learning (ResNet)				

Figure 27. Result summary on real-life examples