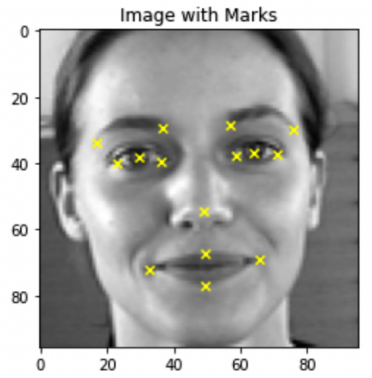
Facial Keypoints Detection Project Report

Wu Yi, Guo Zhiyang

1. Project Description

The objective of this kaggle competition is to predict 15 keypoint positions on face images under a supervised learning approach.



1.1 Datasets

Training set contains 7049 gray images, each of which has 30 positional labels, namely the 'left_eye_center_x', the 'left_eye_center_y', the 'right_eye_center_x', …, etc.

Test set contains xx gray images.

1.2 Objective Function

Submissions are scored on the root mean squared error, RMSE.

2. Related Work

Facial landmark detection algorithms aim to automatically identify the locations of the facial key landmark points on facial images or videos. It is challenging because facial appearance changes significantly across subjects under different facial expressions and head poses and unfavorable environmental conditions such as the illumination[1].

Modern in-the-wild face landmark detection algorithms are based on neural networks. They are divided into 2 main categories: direct (or coordinate) regression methods, when the model predicts x, y coordinates directly for each landmark; heatmap-based regression methods, where a 2D heatmap is built for each landmark.The values in the heatmap can be interpreted as probabilities of landmark location at a certain image location. Fig. 1 illustrates the two approaches.

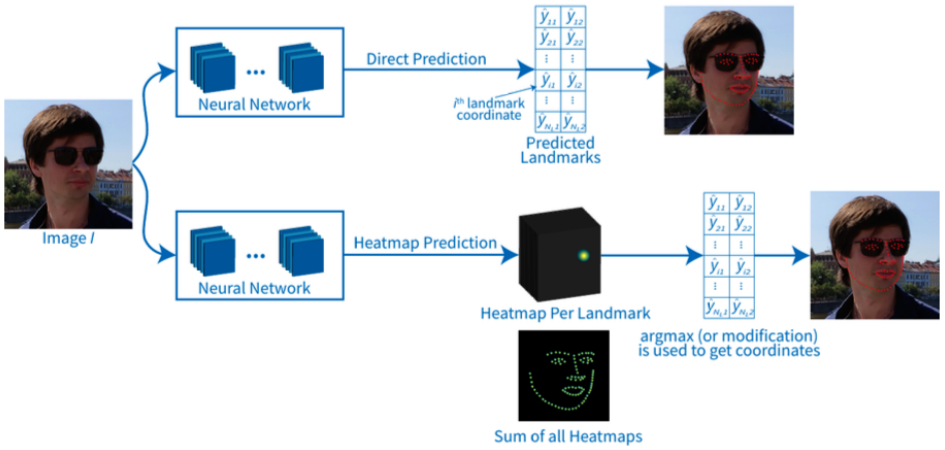


Figure 1. Direct landmark regression (upper row). Heatmap-based (bottom row).

Khabarlak et al. (2022) has compared modern facial landmark detection algorithms published from 2018 to 2021. Fig. 2 illustrates algorithm performance under dataset AFLW and dataset COFW.

Model	Full	Frontal
ERT	4.35 [40]	2.75 [40]
SAN	1.91	1.85
LAB	1.85	1.62
LAB (extra data)	1.25	1.14
Wing (CNN-6)	1.83	-
Wing (CNN-6/7)	1.65	-
Wing (ResNet-50)	1.47	-
PFLD 0.25X	2.07	-
PFLD 1X	1.88	-
AWing	1.53	1.38
GEAN (extra data)	1.59	1.34
HRNetV2	1.57	1.46
LUVLi	1.39	1.19
AnchorFace	1.56	1.38
PIPNet (MobileNetV2)	1.52	-
PIPNet (MobileNetV3)	1.52	-
PIPNet (ResNet-18)	1.48	-
PIPNet (ResNet-101)	1.42	-
SubpixelHeatmap	1.31	1.12

Model	NME (%)	FR (%)
Inter-pupil normalization		
Wing	5.44 [7]	3.75 [7]
AWing	4.94	0.99
PropagationNet	3.71	0.20
ADNet	4.68	0.59
Inter-ocular normalization		
LAB	5.58	2.76
LAB (extra data)	3.92	0.39
Wing (ResNet-50)	5.07 [60]	-
MobileFAN (0.5)	3.68	0.59
MobileFAN	3.66	0.59
HRNetV2	3.45	0.19
PIPNet (MobileNetV2)	3.43	-
PIPNet (MobileNetV3)	3.40	-
PIPNet (ResNet-18)	3.31	-
PIPNet (ResNet-101)	3.08	-
HIH_C	3.29	0.0
HIH_T	3.28	0.0
SubpixelHeatmap	3.02	0.0

Figure 2. Face landmark detection Normalized Mean Error on AFLW (left) and COFW (right).

Since many above state-of-art methods have used the ResNet and MobileNet as backbones, in our group project, we will fine tune these two models, and compare with a manually-designed CNN method. So in total we will show three methods' results.

What's more, we have tried the transformer method (ViT) as well, but due to the huge parameters, it's not plausible to fit into the memory provided by the kaggle notebook. More implementation details can be found in the next chapter.

3. Implementation

3.1 Part 1 - Feature Engineering

Fill nulls with method *df.fillna(method = 'ffill',inplace = True)* and plot the correlation heatmap, as shown in Fig. 3.

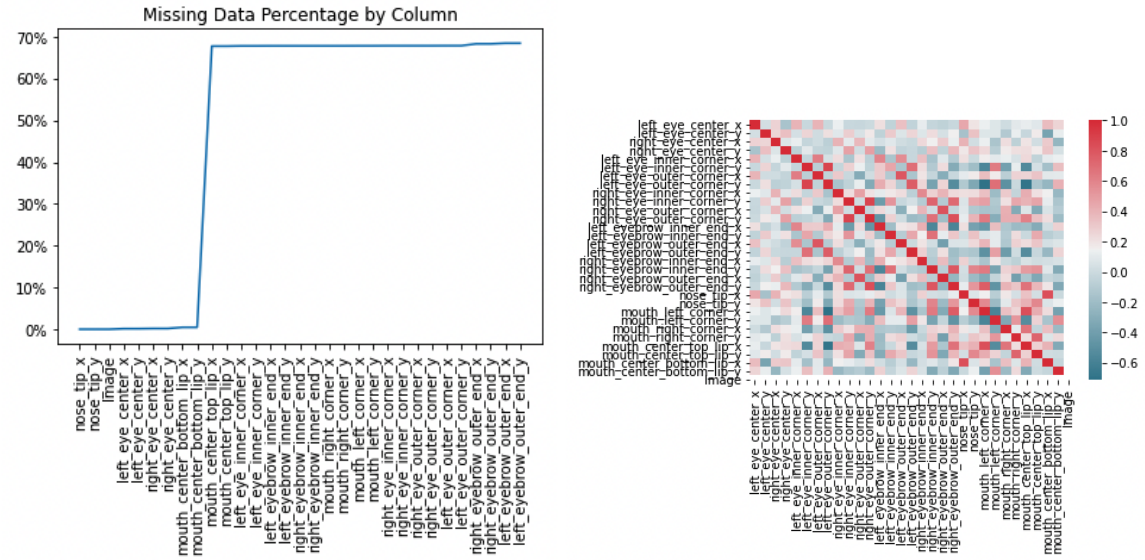


Figure 3. Missing Data Percentage (left) and Correlation Heatmap (right).

3.2 Part 2 - Image Augmentations

First we build a parameter pool of possible augmentations: rotation degree varying between (-10,10), zoom factor (0.9,1.1), shifting x axis (-10,10), and shifting y (-10,10).

For example, Fig. 4 shows an image got its augmentation parameter [10.0, 1.0, 10.0, -10.0]. Then iterate each image from the training set with a configuration from the parameter pool. Finally, the original training set has doubled its size from 7049 to 7049*2.

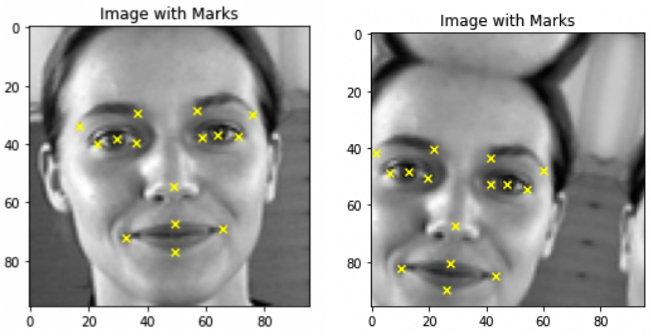


Figure 4. Image augmentation.

3.3 Part 3 - Split the Dataset

The training set is shuffled and splitted as 80% for train_loader, 20% for valid_loader, with batch size 32.

3.4 Part 4 - Build the Models

3.4.1 Model 1. CNN

The architecture has 4 convolutional layers and 3 fc layers.
Dropout is implemented to avoid overfitting. BN is used to speed up learning.

nn.Conv2d(in_channels=1, out_channels=4, kernel_size=[5,5], stride=1, padding=2), nn.BatchNorm2d(4), nn.ReLU(), nn.MaxPool2d(kernel_size=2), nn.Dropout(0.2)	nn.Conv2d(in_channels=4, out_channels=64, kernel_size=[3,3], stride=1, padding=2), nn.BatchNorm2d(4), nn.ReLU(), nn.MaxPool2d(kernel_size=2), nn.Dropout(0.2)	nn.Conv2d(in_channels=64, out_channels=128, kernel_size=[3,3], stride=1, padding=2), nn.BatchNorm2d(4), nn.ReLU(), nn.MaxPool2d(kernel_size=2), nn.Dropout(0.2)	nn.Conv2d(in_channels=128, out_channels=256, kernel_size=[3,3], stride=1, padding=2), nn.BatchNorm2d(4), nn.ReLU(), nn.MaxPool2d(kernel_size=2), nn.Dropout(0.2)

nn.Flatten(), nn.Linear(256*8*8,1024), nn.ReLU(), nn.Dropout(0.2)	nn.Linear(1024,256, nn.ReLU(), nn.Dropout(0.2)	nn.Linear(256,30)

3.4.2 Model 2. resnet18

Directly fine tune the model from *torchvision.models.resnet18(pretrained).*
We change the input channels to 1, and modify the last layer output to 30 dimensions.

3.4.3 Model 3. mobilenet_v3_large

Directly fine tune the model from *torchvision.models.mobilenet_v3_large(pretrained, progress=True).*
We change the input channels to 1, and modify the last layer output to 30 dimensions.

3.4.4 Model 4. vit-base-patch16-224-in21k

Directly fine tune the model from hugging face transformers
ViT('B_16_imagenet1k', pretrained=True)
We change the input channels to 1, and modify the last layer output to 30 dimensions.
In order to fit in this model, we resize the image from 96 to 384.

3.5 Part 5 - Train, Test and Evaluate

3.5.1 Configurations

80% for train_loader, 20% for valid_loader, with batch size 32. For criterion we use RMSE, and for optimizer we use Adam with lr=0.001. The number of epochs is 500,

3.5.2 Challenges

During training CNN Baseline, the loss becomes NaN after some batches. We made the learning rate lower, and the problem is solved. During training the ViT, kaggle RAM is not enough to implement the transformer model, we tried to release the memory, and train on 50 images only, finally, we changed the batch size from 32 to 8, and it works. But the training process takes forever. So we did not get the result of model 4 ViT although the codes work.

3.5.3 Result

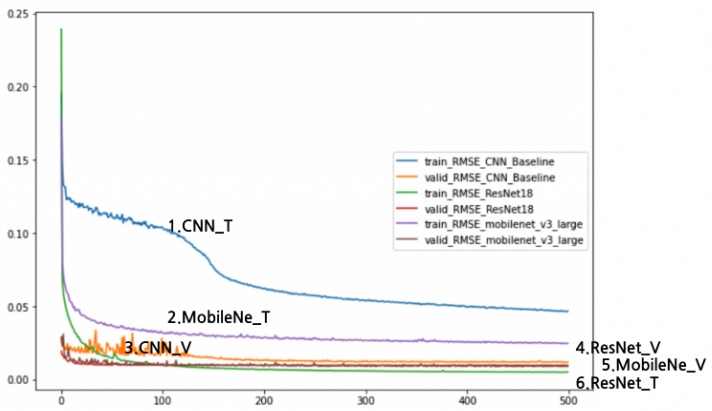


Figure 5. Results

As shown in Fig. 5, CNN has the worst result. MobileNet's training loss is higher than validation loss, but it is the opposite for ResNet. It could be that ResNet is overfitting. We submitted 3 times to the kaggle competition, and Fig. 6 shows our ranking in public score.

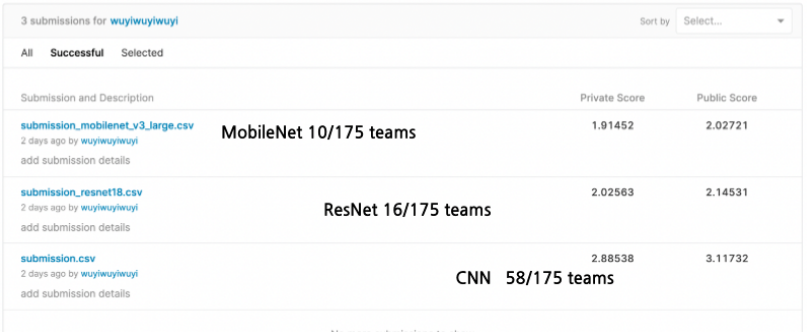


Figure 6. Kaggle Rankings

4. Conclusions

In a nutshell, we tried 4 models to implement the supervised learning on facial landmarks detection, and finally generated 3 results with favorable ranking in kaggle submissions. Unfortunately, the Vision transformer is incomplete. We shall continue trying. Moreover, we want to build a pipeline, from facial object detection to predicting facial landmarks in the future work.

5. References

[1] Wu, Y., & Ji, Q. (2019). Facial landmark detection: A literature survey. International Journal of Computer Vision, 127(2), 115-142.

[2] Khabarlak, K., & Koriashkina, L. (2021). Fast facial landmark detection and applications: A survey. arXiv preprint arXiv:2101.10808.