ECE1512 Digital Image Processing and Application - Winter 2021
Project B

Zhe Su (1002113116)

Handed by: Zhe Su
Handed at: April 5, 2021
Due: April 5, 2021

Task 1: Facial Emotion Detection

1. Xception Implementation

After implementing the **init** and **forward** functions, we are able to acquire a complete Xception model architecture as specified in the handout. The architecture specifics are shown in Listing 1.

```
Model(
  (customConv2d1): CustomConv2dBlock(
    (customConv2d): Conv2d(1, 8, kernel_size=(3, 3), stride=(1, 1), bias=False)
    (bn): BatchNorm2d(8, eps=0.001, momentum=0.99, affine=True, track_running_stats=True)
    (relu): ReLU()
  )
                                  ... x2 ...
  (residualBlock1): ResidualBlock(
    (residual_conv): Conv2d(8, 16, kernel_size=(1, 1), stride=(2, 2), bias=False)
    (residual_bn): BatchNorm2d(16, eps=0.001, momentum=0.99, affine=True,
        track_running_stats=True)
    (sepConv1): SeparableConv2d(
      (depthwise): Conv2d(8, 8, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=8,
          bias=False)
      (pointwise): Conv2d(8, 16, kernel_size=(1, 1), stride=(1, 1), bias=False)
    )
    (bn1): BatchNorm2d(16, eps=0.001, momentum=0.99, affine=True, track_running_stats=True)
    (relu): ReLU()
    (sepConv2): SeparableConv2d(
      (depthwise): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=16,
          bias=False)
      (pointwise): Conv2d(16, 16, kernel_size=(1, 1), stride=(1, 1), bias=False)
    )
    (bn2): BatchNorm2d(16, eps=0.001, momentum=0.99, affine=True, track_running_stats=True)
    (maxp): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  )
                                  ... x4 ...
  (conv2d): Conv2d(128, 7, kernel_size=(3, 3), stride=(1, 1), bias=False)
  (adaptpool2d): AdaptiveMaxPool2d(output_size=(1, 1))
)
```

Listing 1: Xception Architecture

CustomConv2dBlock is a customized block containing convolution layer, batch normalization, and non-linear activation function ReLu, which is adapted in the first two blocks of model. There are four **ResidualBlocks** appended afterword, where we could see that the separable convolution is included. The benefit of using separable convolution is to reduce the number of parameters, save computational time, and output a light-weight model.

2. Loss and Accuracy Metrics Plots

In order to visualize the loss and accuracy over training, functions of plotting loss and accuracy are implemented as shown in Listing 2. Methods take a dictionary parameter to identify testing categories of train, validate (private), and validate (public) respectively.

```
def plot_acc(acc_dict):
   plt.figure()
   t = np.arange(len(acc_dict['train']))
   plt.plot(t, acc_dict['train'], t, acc_dict['public_validate'], t,
       acc_dict['private_validate'])
   plt.legend(['Train Accuracy', 'Validate Accuracy(public)', 'Validate Accuracy(private)'])
   plt.xlabel("Epochs")
   plt.ylabel("Accuracy Score")
   plt.title("Accuracy Plots")
   plt.savefig('task1-accuracy-plot')


def plot_loss(loss_dict):
   plt.figure()
   t = np.arange(len(loss_dict['train']))
   plt.plot(t, loss_dict['train'], t, loss_dict['public_validate'], t,
       loss_dict['private_validate'])
   plt.legend(['Train Loss', 'Validate Loss(public)', 'Validate Loss(private)'])
   plt.xlabel("Epochs")
   plt.ylabel("Loss")
   plt.title("Loss Plots")
   plt.savefig('task1-loss-plot')
```

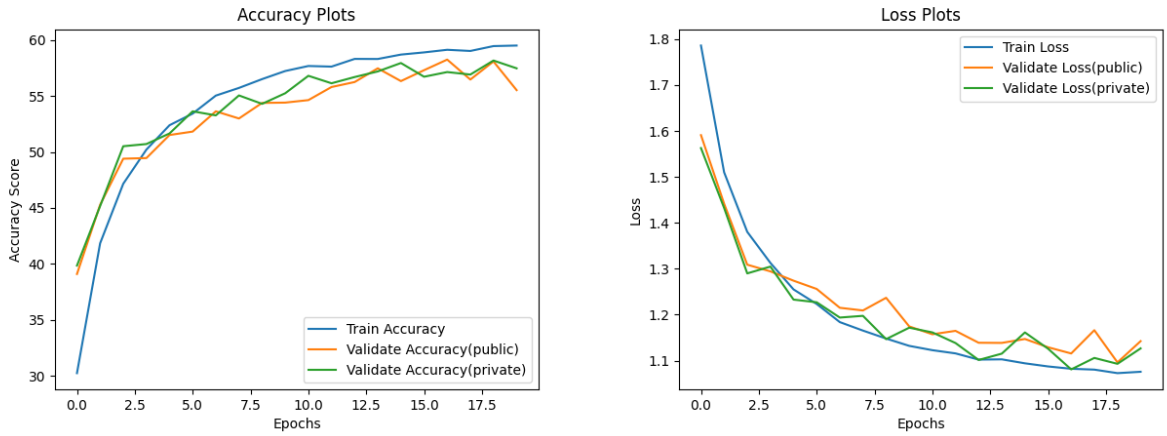
Listing 2: Plot loss and accuracy

3. Train

We trained the model for 20 epochs as instructed and the results are presented in Table 1. The total computation time is around 33 minutes, and the plots of accuracy and loss versus training epoch are shown in Figure 1. The accuracy here is defined by the total correctly predicted labels over the total number of samples. In order to view more insights about the model's performance, we also plot the confusion matrix to indicate how the model perform the prediction task on each class. The confusion matrix is shown in Figure 2. From there we could clearly observe that the model perform the task of predicting happy expression very well, and perform the worst on disgust expression. The performance difference could be due to various reasons, and one of the reason could be the unbalanced data sample used in the training, which we would address this limitation in the later section.

	Train	Test (private)	Test (public)
Accuracy	59.52	58.17	58.09
Loss	1.0758	1.0931	1.0962

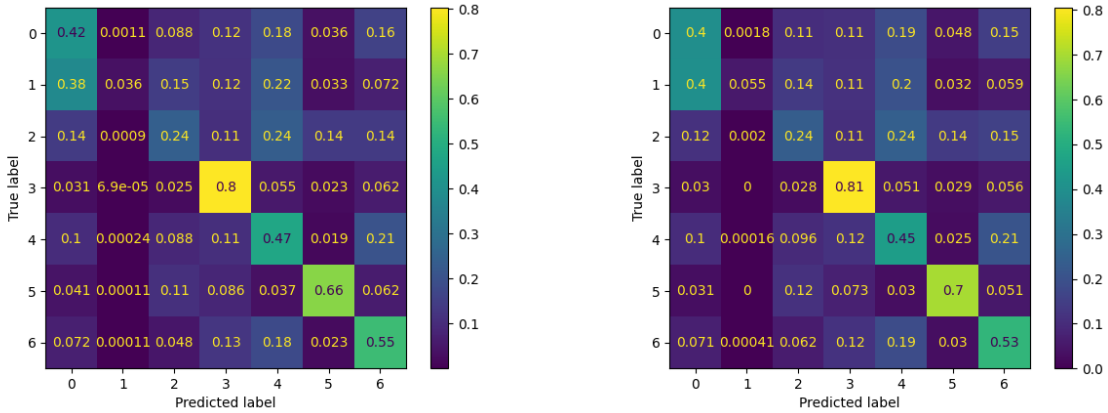
Table 1: Xception best results after 20 epochs training



(a) Accuracy Plot

(b) Loss Plot

Figure 1: Accuracy and loss plot over training 20 epochs



(a) Confusion Matrix on Training Data

(b) Confusion Matrix on Validation Data

Figure 2: Confusion matrix over training 20 epochs. Label 0 to 6 represent expression of angry, disgust, fear, happy, sad, surprise and neutral respectively

4. Performance Evaluation on Test Folder

Before commenting on the model performance, we would like to briefly explain the meaning of each column from the Figure 3. There are in total 7 columns with 6 data samples shown in the figure. The first column plots the original image input. The second and the third columns are the true label and predicted label respectively. The digits in the fourth column are the confidence score on the predicted label. The last three columns demonstrate the images viewed from the model's 'vision'. Specifically, the last column of images could explain which region of the image that the model weighted the most when make a prediction. We intentionally add an additional image with disgust expression in the testing folder to see how the model perform the task on this category.

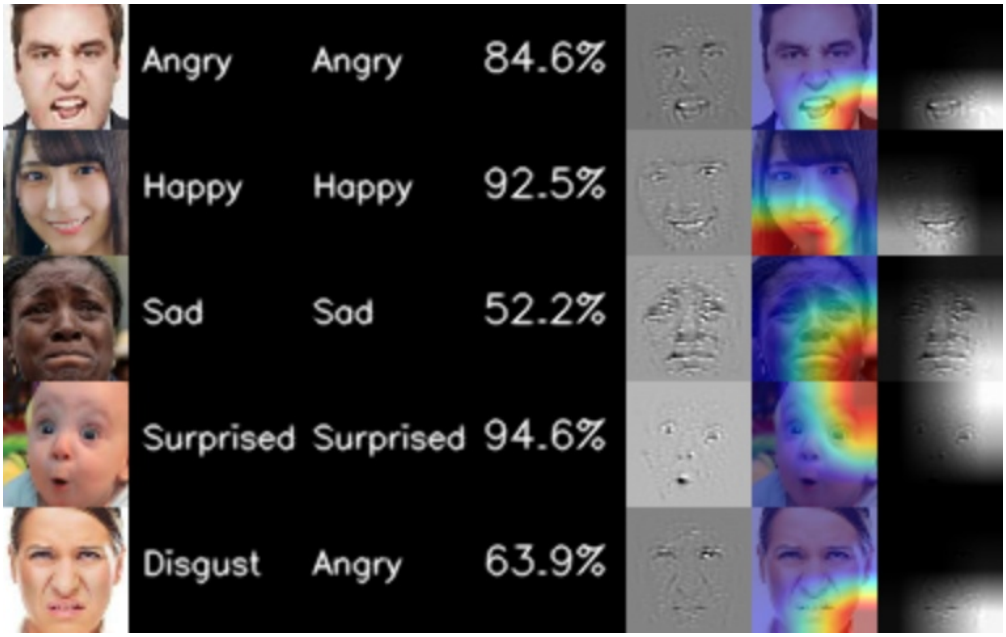


Figure 3: Guided GradCAM from test folder

From Figure 3, we could see that the model correctly categorize all the expressions except disgust. Specifically, both happy and surprised faces achieve a pretty high confidence score of 92.5% and 94.6% respectively. By observing the activation masks, those correctly predicted expressions are all showing with obvious or exaggerated facial components. For example, the happy expression has a obvious smiling mouse, the angry expression is shown with a big opening mouse, and the facial expression of surprises has rounded eyes. Those features are weighted importantly when the model categorizes the facial expressions, and all of them are intuitively related to their own categories. Therefore, we conclude that overall the model performs well, and the decisions that the model made are reasonable and explainable based on the presented activation masks. For the image that the model wrongly predicting on, we print out the prediction scores that the model made on each class as shown in the Table 2, where the model only shows a little confidence on the correct class (17.3%).

Class	Angry	Disgust	Fear	Happy	Sad	Surprise	Neutral
Disgust	63.9%	17.3%	7.4%	0.4%	4.0%	0.5%	6.6%

Table 2: The prediction scores on each class for the disgust expression image

5. Performance Evaluation on Test2 Folder

We apply the same analysis approach as the previous section to comment on the model's performance on the images in test2 folder. Similarly, the model performs extraordinarily well on predicting happy and surprising expressions, which we could see that their activation masks are all shown with their signatures facial components. However, the model predicts the angry expression correctly but with a relatively low confidence

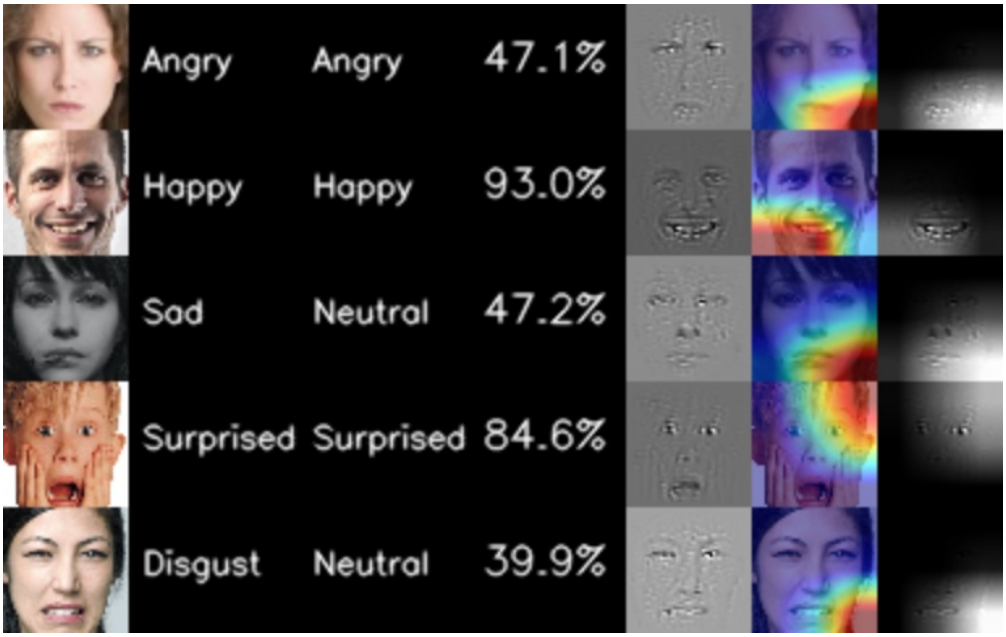


Figure 4: Guided GradCAM from test2 folder

score, and predict incorrectly on both sad and disgust expressions. By observing the activation masks, we could surprisingly find that the activation images from the incorrect or low confidence score expressions does not have any obvious or exaggerated facial components shown. For example, the sad expression image here shows a pretty neutral faces, which is even hard to recognize it is a expression of sadness by human perception. Therefore, we conclude that the model perform generally well on those expressions only shown with signature facial components, but would probably miss-classify the expressions without showing obvious facial components. We also present the prediction scores of the incorrectly predicted images in Table 3 to see how off the decisions are made.

Class	Angry	Disgust	Fear	Happy	Sad	Surprise	Neutral
Sad	4.9%	0.3%	13.8%	0.5%	32.4%	1.0%	47.2%
Disgust	23.8%	5.8%	10.4%	7.8%	11.0%	1.3%	39.9%

Table 3: The prediction scores on each class for the incorrectly predicted images

6. Small Experiment on Class Re-weighting

We choose to experiment on class re-weighting after examining the distribution of data sample. The data sample distribution and their normalized weights are presented in Table 4.

	Angry	Disgust	Fear	Happy	Sad	Surprise	Neutral
Sample Size	4953	547	5121	8989	6077	4002	6198
Sample Weight	0.862	0.985	0.857	0.750	0.831	0.888	0.827

Table 4: Data Distribution and Data Sample Weights

We see that the data is not distributing evenly, where expressions of happy and neutral appear the most (8989 and 6198 respectively), and expression of disgust has only 547. If we compute the loss with a same weight, it might cause the model to out-perform on those expressions with more data and to under-perform on the expressions with fewer data. We could observe this limitation from metrics in the previous two sections, where happy expression is easier to be classified correctly with a higher confidence score and disgust expression is merely correctly classified. This limitation could also be clearly observed from the confusion matrix in Figure 2, where we could see the accuracy values are almost linear to the sample size of each class. However, we surprisingly found that one exception is made by class of surprising, which it has the second least sample size (4002) but achieve the accuracy score of 0.67 and 0.71 on training and validation respectively. One potential reason is that the the facial expression of surprising has signature features such as big rounded eyes and rounded opening mouse, and those features are easier to be recognized by our model. Then the class of surprising performs well on the task even it has less data samples.

In order to improve on the limitation of unbalanced data sample, we would count the weighted factor of data sample while calculating the loss during training session by the approach of class re-weighting. The weights would be calculated linearly based on the sample size to address the limitation of data unbalance. We expect an experiment result as following: the model overall accuracy on prediction would be improved, and the accuracy of predicting the expression with least data size, such as expression of disgust, would be improved. The code snipe in Listing 3 shows how we calculate the weighted tensor linearly based on the sample count, and Listing 4 shows how we apply the weights to the loss function in pyTorch.

```
# Linearly compute the weighted tensor based on sample data count
# The more data, the less weight applied
emotion_name = ['Angry', 'Disgust', 'Fear', 'Happy', 'Sad', 'Surprise', 'Neutral']
emotion_count_dict = {}
for emotion_list in [emotions, private_emotions, public_emotions]:
   for index in emotion_list:
      if index in emotion_count_dict:
         cur = emotion_count_dict[index]
         emotion_count_dict[index] = cur+1
      else:
         emotion_count_dict[index] = 1
print("---------------------- DATA ATTRIBUTIONS ------------------------")
sample_count_list = []
for index in range(len(emotion_name)):
   sample_count_list.append(emotion_count_dict[index])
   print("{}: {}".format(emotion_name[index], emotion_count_dict[index]))
normed_weights = [1 - (x / sum(sample_count_list)) for x in sample_count_list]
print("Normed weights: {}".format(normed_weights))
print("-----------------------------------------------------------------")
```

Listing 3: Weight Tensor Calculation

```
weights_tensor = torch.FloatTensor(factory.weights).to(device)
criterion = nn.CrossEntropyLoss(weight=weights_tensor)
```

Listing 4: Apply weight to loss function

7. Small Experiment Results

	Train	Test (private)	Test (public)
Accuracy	59.49%	58.26%	58.81%
Loss	1.0936	1.1147	1.1199

Table 5: The best result of Xception with class re-weighting after 20 epochs training

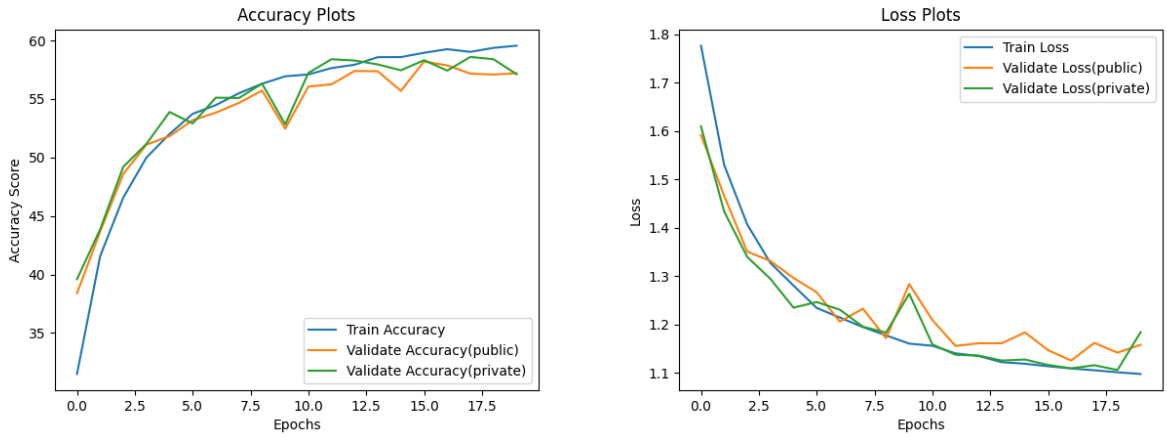
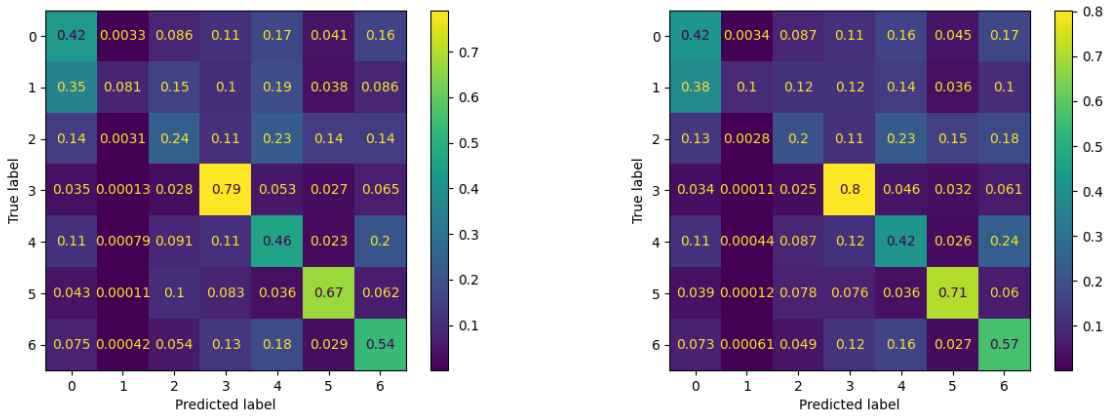


Figure 5: Accuracy and Loss Plot over training 20 epochs with class re-weighting

Table 5 presents how the model performs after applying the class re-weighting approach, and Figure 5 also shows the plot of overall accuracy and loss after training 20 epochs. We could see that the model's performance is merely improved regarding the metrics of overall accuracy and loss. Although the approach does not boost the performance of model, it certainly does not hinder the original model capability.

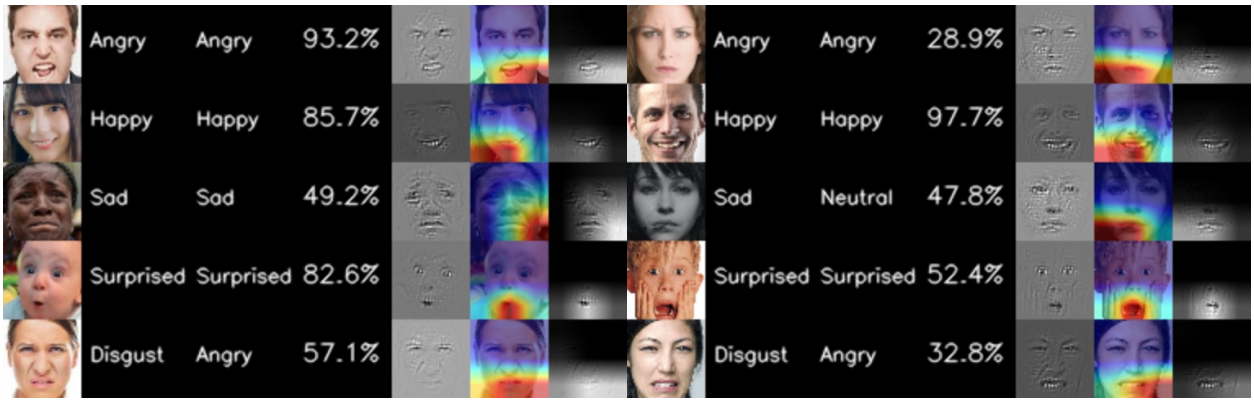
However, if we view the results from different perspectives, we are able to see that the approach of class re-weighting could actually address the limitation that caused by the unbalanced data. Figure 6 shows the confusion matrix on both training and validation dataset after class re-weighting. By observing the prediction score on class of disgust (label 1), we see that it achieves the confidence score of 0.081 and 0.1 on training and validation respectively, which shows a clear improvement comparing to the scores before class re-weighting (0.036 and 0.055). It means that the model now predicts the disgust expression better when we applied more weights to the class while training. Interestingly, we could see that it also affects the prediction on happy expression, which the scores decrease from 0.8 and 0.81 to 0.79 and 0.8 respectively. This is because we count less weights than before when we calculate the loss associated to this class, thus it downs the performance on the class a little. However, the class of happy expression still perform the best among all classes, we could probably conclude that the size of data sample still has more effeteness than the factor of weights in this task and model.



(a) Confusion Matrix on Training Data with Class Re-weighting

(b) Confusion Matrix on Validation Data with Class Re-weighting

Figure 6: Confusion matrix over training 20 epochs with class re-weighting. Label 0 to 6 represent expression of angry, disgust, fear, happy, sad, surprise and neutral respectively



(a) Test Folder

(b) Test2 Folder

Figure 7: Guided GradCAM on class re-weighted model

Class	Angry	Disgust	Fear	Happy	Sad	Surprise	Neutral
Disgust (Test)	57.1%	13.7%	18.3%	0.7%	7.4%	0.3%	2.5%
Sad (Test2)	12.4%	0.2%	6.1%	0.8%	31.5%	1.1%	47.8%
Disgust (Test2)	32.8%	5.3%	21.6%	9.9%	16.2%	1.2%	13.0%

Table 6: The prediction scores on each class for the incorrectly predicted images after class re-weighting

Figure 7 shows the guided GradCAM image on both test and test2 folders. For Figure 7a, model still predicts all expression correctly except the expression of disgust, however, the confidence score on the wrong predicted label are decreased from 63.9% to 57.1%. We could also see the confidence scores on both happy

and surprised expressions are also down-performed from 90+% to 80+%, which the re-weighted based on data sample not only slight improve the performance on the class with fewer data size, but it also affects the performance on the class with more data sample. Similarly, we could observe pretty much the same results from Figure 7b, where it is less confidence on the wrongly predicted label, but the performance on the class that perform well originally are also affected. Table 6 demonstrates after class re-weighting how much confidence the model predicts on each class when wrongly classify an image. Comparing to Table 2 and Table 3, even though the model shows less confidences on wrongly predicted classes, however, the confidence scores on the true label are not improved.

There is a certain limitation in the way we implement the class re-weighting approach. Simply computing the weighted tensor linearly based on sample size is not ideal for class re-weighting. We could see from Table 6 that the re-weighting model performs on tasks sometime even worse than the previous model. One potential defect of addressing the weight vectors linearly based on the sample size is that it only considers the size of each class, but other important features from the images are neglected. Instead, we could probably consider weight as a parameter and train it along with the model. In this way, the weight vectors learned from training process would be more informative, and would possibly give a better performance.

Task 2: Facial Action Unit Detection

1. Bottleneck Residual Block Implementation

After implementing the **init** and **forward** functions, we then have a class of bottleneck residual block. The architecture of block is shown in Listing 5. As we could see from Listing 5, **downsample** is the residual part that would bypass the middle layers and be directly added to the last.

```
Bottleneck(
  (downsample): Sequential(
    (0): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (conv1): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (bn1): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True, track_running_stats=True)
  (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (bn2): BatchNorm2d(64, eps=0.001, momentum=0.1, affine=True, track_running_stats=True)
  (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (bn3): BatchNorm2d(256, eps=0.001, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU()
)
```

Listing 5: Bottleneck Residual Block Architecture

2. Deconvolutional Block Implementation

The deconvolution block, or more precisely to be called as transposed convolution block, is constructed by **ConvTranspose2d**, and will be appended after the bottleneck blocks. The transposed convolution block architecture is shown in the Listing 6.

```
(deconv_layers_1): Sequential(
    (0): ConvTranspose2d(2048, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
)
```

Listing 6: Transposed Convolution Architecture

After concatenating all blocks together, we are able to acquire the complete modified ResNet architecture as shown in Listing 7.

```
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  (layer1): Sequential(
    (Bottleneck0)
    (Bottleneck1)
    (Bottleneck2)
    (Bottleneck3)
  )
```

```
  ... x4 ...
  (deconv_layers_1)
  ... x3 ...
  (final_layer): Conv2d(256, 10, kernel_size=(1, 1), stride=(1, 1))
)
```

Listing 7: Modified ResNet Architecture

3. Visualization from Test Folder

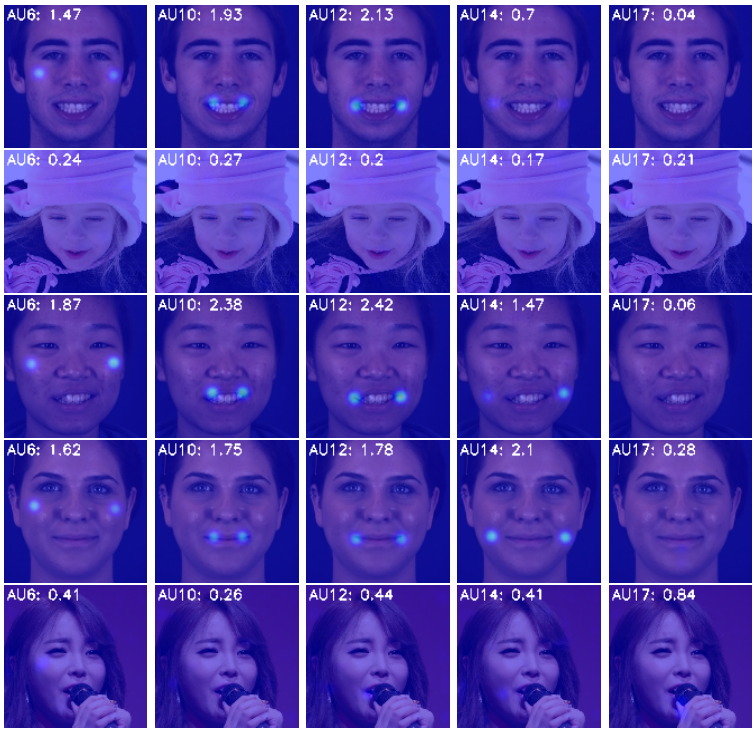


Figure 8: Visualization of FAUs from test folder

As shown in Figure 8, we output 25 images consisted by 5 subjects each with 5 Facial Action Units (FAUs). AU17 represents the face component of chin raiser according to the Facial Action Coding (FAC) system [1]. Comparing all AU17 intensity scores across 5 subjects from Figure 8, we could see that all images achieve a low intensity score on this FAU. The image on row 5 column 5 achieve the highest intensity value on AU17 across all subjects (0.84). However, it seems that the model wrongly recognizes the microphone and hands as the raiser chin. Horizontally comparing the FAUs for each subject, it is obvious to see that the intensity value of AU12 and AU10 are generally high, where AU12 and AU10 represent lip corner puller and upper lip raiser respectively. Therefore, one thing we could probably learn from this visualization is that the facial units around lip region play more significant role in facial expression interpretation compared to facial unit around edge region such as chin. We also realize that the image subject on row 1 and row 4 achieve the relatively higher intensity values compared to the others. The major difference between them is that the image subjects with better achievement show only the face without other components such as hair and hat.

Therefore, we could also conclude that the model perform well on the subjects where only have the facial components directly displaying in the image.

4. Visualization from Test2 Folder

Similarly, we apply the same analysis approach as previous section on Figure 9. We could tell that AU12 and AU10 are two of the FAUs that have the higher intensity values across all subjects. By observing only the AU17 across all subjects, we see that all images have a low intensity value except the one located at row 4 column 5. The only image with high intensity value on AU17 has the chin shown obviously, it then obtains a higher value compared to the other. The conclusion we made on previous section could also be verified by Figure 9, where image subjects at row 2, row 3, and row 5 have relatively lower intensity scores since they have included other components in the image such as hair and mask that might affect the model's performance.

Figure 9: Visualization of FAUs from test2 folder

5. Comparison of FAUs Approach and Task-1 Approach

In general, Task-2 FAUs approach explains the model decision better due to the ability of explicitly decompositing a facial expression to different components. We intentionally pick a happy emotion image from test2 folder as the comparison sample, and the output from Task-1 approach and Task-2 approach are displayed in Figure 10a and Figure 10b respectively. The model from Task-1 successfully predicts the emotion of happiness with a pretty high confidence score. In order to understand if the model make the prediction reasonably, we applied GradCAM to visualize the activation mask, which we know that the model makes the

decision majorly based on the features of smiling mouse. Differently, FAUs approach output the intensity score of each facial component. By adapting FAUs, the interpretation becomes easier since the weighted value or intensity score of each FAU that contribute to the facial expression is explicitly provided, rather than roughly picturing from an activation mask. Therefore, we are able to clearly tell that which FAUs are more important in certain facial expression. Moreover, with the helps of FAC system, we could evaluate the performance of the model on certain emotions given the intensity value of FAUs.

(a) Task-1 Approach

(b) Task-2 Approach

Figure 10: Comparison of two approaches on a happy emotion image

6. Fine Tuning the Pre-trained Model

We train the model with 60 FFHQ [2] images with 5 epochs based on the original pre-train model, which is also referred as few-shot tuning. The final loss after training is still high (approximately 1750) and the loss figure is shown in Figure 11. Notice that the few-shot tuning on 60 unlabelled images is adapted from the idea of contrastive learning, where the provided loss function is Mean Squared Error (MSE) between the model's output and the augmented version of the model's output. Therefore, in this case a relatively high loss after 5 epochs tuning is acceptable since we do not want the model unlearned the original features due to keep minimizing the loss on new data samples.

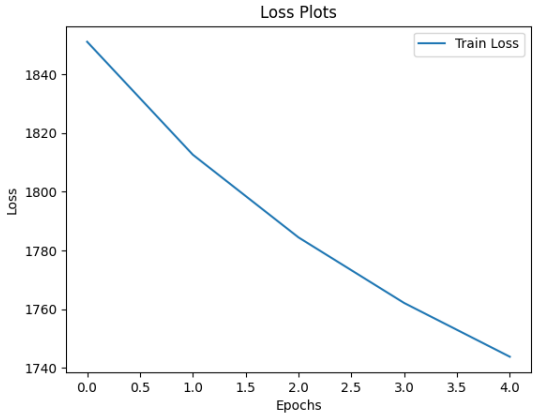


Figure 11: Loss Plot over training 5 epochs

7. Evaluation

The model after few-shot tuning slightly improves the performance on estimating intensities of FAUs on certain subjects but failed on the others. Again, from the FAC system we know that AU6 and AU12 are the two AUs that composite the facial emotion of happiness [1]. By looking at images at row 3 in Figure 12, the happy emotion image achieves AU6 and AU12 intensity values of 2.41 and 2.66 respectively, which are both higher than the values shown in the Figure 9 (2.39 and 2.64) before fine tuning. However, we could find that the model's performance is decreased on the other images, where all AUs intensities are lower than before. One potential reason is that the model is affected when learn toward the new image samples, which the model slightly unlearns the original features obtained in pre-train process. Therefore, we could know that the choice on epochs, learning rate, optimizer, and loss function are important while conducting the contrastive learning on new unlabelled data since it might have negative performance if we do not choose these parameters carefully.

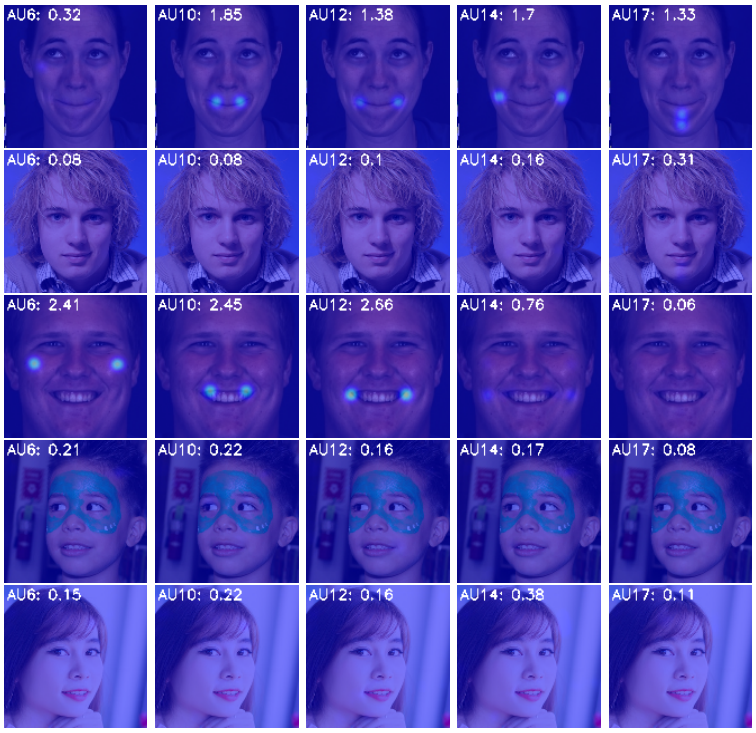


Figure 12: Visualization of FAUs from test2

8. Novel Idea of Paper and Comparison to ResNet

We choose Niinuma et al.[3] as the complementary study of our project. Niinuma et al. mainly examines how the design choices of models affect the performance of facial action intensity estimation, including image normalization, model pre-train, training size, and the choices on optimizer and learning rate. The paper states that even most of the state-of-art models perform the task pretty well, the reasoning behind their design choices remain unclear[3]. From my point of view, this is the approach more relating to empirical learning, which is to experiment on different parameters setting based on intuitive choices and to see how

much it affect the performance on tasks. Niinuma et al. concludes that among the experiments from all four aspects, making right choice on optimizer and learning rate, and pre-train the model are performance influenced, but tuning training size and image normalization merely improve the performance due to various reasons.

The modified ResNet model above are pre-trained by the approach of supervised learning with labelled data, and the output from the model are intensity values of five picked AUs. The above model improves the performance on intensity estimation majorly by fine tuning, which adapts the approach of contrastive learning with unlabelled data. Differently, Niinuma et al. focus more on the aspect of carefully choosing parameters and training setups in order to improve on the task of intensity estimation.

9. Limitations

The proposed method is highly empirical, which the suggested optimizer and learning rate combinations are learned by trial and error. The result could potentially be off by the differences in setting. Moreover, the proposed method only heavily investigates on the influences on parameters setting, but lack of considering how the model's architecture impacts on the performance.

10. Small Experiment Setup

Although the set up of Niinuma et al. proposed method is quite different than the set up of our modified ResNet model, we could still try to apply the ideas to improve our model's performance on intensity estimation, such as experimenting on how the choice of optimizer and learning rate affect the performance of our few-shot tuning. Therefore, we would like to set up a small experiment as following: we choose the optimizer of Adam and SGD with different learning rates while fine tuning our ResNet model 5 epochs. With different combinations of optimizer and learning rate, we would like to see which one improve our model the most.

In order to evaluate the experiment results, we need to carefully choose our evaluation metrics. Since the model outputs five AUs (6, 10, 12, 14, and 17), and AU6 + AU12 is considered as the intensity value of happy emotion, thus we intentionally pick four images with happy emotions from test and test2 folders to be our experiment testing samples, which are shown in Figure 13. We pass our testing samples into our ResNet model without the fine tuning in order to output the benchmark scores. As you could see that the benchmark scores we defined are presented in Table 7, and it would be used in the later section.

(a) 014.jpg (b) 016.jpg (c) 556.jpg (d) 1014.jpg

Figure 13: Testing samples on our small experiment

	014.jpg	016.jpg	556.jpg	1014.jpg
AU6	1.94	1.77	2.39	1.67
AU12	2.49	1.91	2.64	2.27
AU6+AU12	4.43	3.68	5.03	3.94

Table 7: Performance benchmarks of AU6 and AU12 on the testing samples with untuned modified ResNet model

11. Small Experiment Results

	014.jpg	016.jpg	556.jpg	1014.jpg
Benchmarks	**4.43**	**3.68**	**5.03**	**3.94**
Adam & lr=1e-6	4.29	3.39	**5.07**	3.60
Adam & lr=5e-6	4.17	3.24	4.97	3.38
Adam & lr=1e-5	3.68	2.68	4.61	2.72
Adam & lr=5e-5	1.20	0.55	2.58	0.59
Adam & lr=1e-4	0.46	0.30	1.29	0.32
Adam & lr=5e-4	0.26	0.26	0.26	0.26
SGD & lr=1e-6	4.22	3.40	**5.04**	3.56
SGD & lr=5e-6	4.20	3.47	**5.04**	3.63
SGD & lr=1e-5	4.23	3.55	**5.07**	3.80
SGD & lr=5e-5	4.16	3.32	**5.14**	2.94
SGD & lr=1e-4	2.08	0.54	3.13	0.74
SGD & lr=5e-4	0.40	0.42	0.41	0.40

Table 8: Intensity score of AU6+AU12 with different combinations of optimizer and learning rate on testing samples

Table 8 presents the results of our small experiment on four testing samples. We choose Adam and SGD optimizer with different learning rates to fine tune our model. The red texts indicate the intensity score of AU6 + AU12 that out-perform the benchmarks. And the green texts indicate the best combination of optimizer and learning rate among each choice of optimizer. Similar to the results presented in Niinuma et al., Adam with *1e-6* learning rate and SGD with *1e-5* learning rate and no momentum indeed achieve the best results[3] compared to other combination based on the intensity scores on all four testing samples. However, it seems like the performance improvement is only applied on sample *556.jpg*, and the rest of the samples are not shown with any improvements. Therefore, we would like to conclude that the model's performance is not improved by the fine tuning with all different combination of optimizer and learning rate.

We notice that after tuning with a large learning rate, or tuning with longer epochs, the loss will keep decreasing, and all testing samples will result to a similar intensity value. For example, Adam with *5e-4* finally leads to all samples with a same intensity score (0.26), and SGD with *5e-4* also lead the a same intensity value (around 0.40). Since we adapt the contrastive learning to fine tune our model, it is possible

that the model unlearns some of the original features thus leads to a worse performance. Therefore, in order to experiment more on the behind reasoning, it drives us to examine how the true label is defined while we fine tuning our model. Listing 8 shows how true label is defined originally, which is to apply a color map *COLORMAP_JET* to the model output to represent the augmented data.

```
...
# heatmap is the output from the model
true_map = heatmap.detach() / 256 #10x64x64
label = torch.zeros(true_map.shape) #10x64x64
for j in range(0, true_map.shape[0]):
   temp = cv2.applyColorMap((np.float32(true_map[j,:,:])).astype(np.uint8), cv2.COLORMAP_JET)
       #64x64
   label[j, :, :] = torch.FloatTensor(temp).mean(-1).unsqueeze(0) #1x64x64
# return true label
return label #10x64x64
```

Listing 8: Define true label of contrastive learning

As inspired by Tiu[4], now we would like to redefine true label, where we want to augment the original image first and pass it to the model later. In this way, the true label would be defined as the model output of an augmented image. The way we calculate the loss would still be the same as before in order to maintain the basic setup. The portion of code to define new true label is shown in Listing 9.

```
# prepare augumented image
image_grayscale = cv2.imread(PATH_TO_IMAGE, cv2.IMREAD_GRAYSCALE)
image_augmented = cv2.applyColorMap(image_grayscale, cv2.COLORMAP_JET)
image_np_augmented = torch.from_numpy((image_augmented/255.0).swapaxes(2,1).swapaxes(1,0))
sample['ImAug'] = image_np_augmented.type_as(torch.FloatTensor())

...

# return true label
img_augmented = sample['ImAug']
true_label = net(img_augmented).squeeze(0)
```

Listing 9: Redefine true label of contrastive learning

	014.jpg	016.jpg	556.jpg	1014.jpg
Benchmarks	**4.43**	**3.68**	**5.03**	**3.94**
Adam & lr=1e-6	3.54	4.27	**5.04**	3.54
SGD & lr=1e-5	2.81	3.65	**5.04**	2.9

Table 9: Intensity score of AU6+AU12 with new true label, the best two choice of optimizer and learning rate on testing samples

Table 9 shows the experiment results on the best two choices of optimizer and learning rate. It shows that our new approach reach to a similar result, which does not improve the model's performance on intensity estimation after fine tuning with contrastive learning. There are some potential reasons leading to this result: First, there might be some better augmented methods to define true label that might improve the model's

performance; Second, there are probably better loss functions other than MSE that is more compatible on this task; Third, our modified ResNet model is probably not fully suitable for fine tuning with contrastive learning, and the supervised learning with labelled data might better improve the model instead.

References

[1] Paul Ekman. Facial action coding system (facs). 2002.

[2] Samuli Laine Tero Karras and Timo Aila. A style-based generator architecture for generative adversarial networks. pages 4401–4410, 2019.

[3] Itir Onal Ertugrul Jeffrey F. Cohn Koichiro Niinuma, Laszlo A. Jeni. Unmasking the devil in the details: What works for deep facial action coding? *The British Machine Vision Conference (BMVC)*, page 4, 09 2019.

[4] Ekin Tiu. Understanding contrastive learning. 2021.