DenseNet Models for Tiny ImageNet Classification

E4040.2020Fall.SSSS.report

Mingfang Chang mc4795, Han Wang hw2761, Yingyu Cao yc3713
Columbia University

***Abstract*—In this report we built two image classification models following the networks structures in paper *DenseNet Models for Tiny ImageNet Classification* and trained them on the Tiny ImageNet dataset, which is a small resampling subset of ImageNet dataset. We aimed to reach a similar validation and training accuracy in the original paper. [1] There are many complicated models that have good performance on ImageNet data set. However, the aim of the project is not to achieve high accuracy, but to build small and shallow neural networks that only require low computation power and can effectively learn from the data information. We successfully built two small DenseNet that is learning data information effectively, but we did not achieve a result that is as good as the one in the original model given time limitation. We get around 45% validation accuracy while in the paper they get around 60% validation accuracy. The difference might be due to different hyperparameters which is explained further in the paper.**

I. INTRODUCTION

Image classification is a core task in computer vision and is widely used in media, transportation, security, etc. Convolutional Neural Network (CNN) has achieved huge success and thus popularizes in solving image classification problems. One of the most famous challenges is the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [2], which starts from 2010 and now acts as a benchmark of image classification. Tiny ImageNet Challenge is a similar challenge proposed by Standford University using a subset of ILSVRC dataset called Tiny ImageNet [3].

The paper *DenseNet Models for Tiny ImageNet Classification* [1] built its own DenseNet model with the Tiny ImageNet dataset, aiming to achieve the highest validation accuracy of 60% using models that only require low computation power. The goal of this project is to dive into the paper, and reproduce the result by recreating the models using Tensorflow and Python to get a better understanding of CNN models and data augmentation methods.

The report is organized as follows: In Section II, we go over the methodology and key results of the original paper [1]. Then, in Section III, we discuss the project objective, challenges, how we tackle those problems, and the difference between our and the original methods. Next, we show our network and training process in Section IV. we compare our results with the original results, analyze the reasons for the differences and some future research directions in Section V and Section VI. Finally, we include the acknowledgement, individual contribution, ethical concerns and GitHub link for this project in Section VII and Section VIII.

II. SUMMARY OF THE ORIGINAL PAPER

The original paper [1] built two CNN models from scratch and implemented some noval data augmentation and learning rate tuning methods, aiming to achieve 60% validation accuracy on the Tiny ImageNet dataset using low computation power.

A. Methodology of the Original Paper

The researchers in the original paper built a convolutional neural network with 16 layers and another one with 14 layers. Figure 1 shows the architecture of the models. Then, they tuned the models with regularization, customized learning rate and different optimizers. The process can be described as follows.

1) Model Structure: Deep neural networks can be hard to train due to vanishing gradient problems.To solve this problem, residual network, which add shortcut connections between blocks for neural network, was introduced. However, instead of using aggregate function like ResNet, the DenseNet in the paper concatenate the outputs of two blocks and feed it to the next layer. The residual blocks structure in the two models are very similar, except using different filter number, kernel initializer and kernerl regularizer. Both models contain three residual blocks, with model 1 containing five 2D Convolution layers per block and model 2 containing four Convolution layers per block.Batch Normalization function and ReLU activation function were used after every 2D Convolution layer. A MaxPooling layer was applied after each residual block and

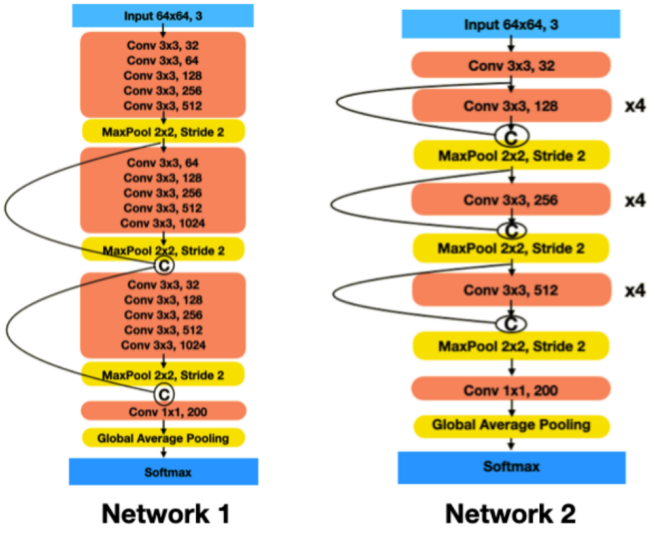

Fig. 1: CNN of Original Paper.

a final 2D Convolution layer and a global average pooling layer were applied after residual blocks for both models.The researchers used accuracy as metric and categorical cross-entropy as loss function for both models.

2) Training process: for model 1, the researchers in the original paper trained the model with images with $32x32$ resolutions for 15 epochs, followed by $64x64$ images for 30 epochs, $16x16$ images for 10 epochs, $64x64$ images resolutions for 15 epochs sequentially. After achieving 50% validation accuracy, they fed $64x64$ augmented data for another 175 epochs (which is shown in their code[1], though they stated 150 epochs in the paper). For model 2, they fed the model with default $64x64$ augmented images for 108 epochs. They used CyclicLR for learning rate adjustment for the training process.

3) Data augmentation: the original paper used imgaug library [2] to implement data augmentation. For model 1, the researchers applied a random number of the following augmentations on the whole dataset: Scale, CoarseDropout, Rotate, Addictive Gaussian Noise, Crop and Pad. For model 2, they applied the following augmentations on half of the dataset: Horizontal Flip, Vertical Flip, Gaussian Blur, Crop and Pad, Scale, Translate, Rotate, Shear, Coarse Dropout, Multiply, Contrast Normalisation.

4) Hyperparamter setting: For model 1, in the original paper the researchers used the default Adam optimizer with $1e-3$ learning rate. Kernel regularizer was not used in this model, and the default value of kernel initializer was not changed. For callback function, they used ReduceLROnPlateau, which can be used to reduce learning rate when the validation loss stagnates for 5 epochs and no training improvement is seen for efficient training. In the training process, researchers set different batch size for different size of input images. Specifically, for 32x32 and 16x16 resolution images, the batch size is 256, while for 64x64 resolution images, the batch size is 64.

For model 2, in the original paper the researchers used Adam optimizer with learning rate 1e-4 and epsilon 1e-8 (though in the Colab coding file they also tried RMSprop and SGD as optimizer). To prevent overfitting, they applied L2 regularization term with coefficient 2e-4. The batch size is 128 for all epochs. They manually tuned the learning rate range of the CyclicLR function every 12 or 24 epochs according to the validation accuracy.

B. Key Results of the Original Paper

Researchers of the original paper trained their first model with 17.9 million paramters for 235 epochs in total and changed feeding images when the model started overfitting. They trained the second model with 11.8 million parameters for 108 epochs with augmented images. Finally, they got 67% training accuracy and 59.5% validation accuracy on the first model, and 68.11% training accuracy and 62.73% validation accuracy on the second model [1].

III. Methodology

A. Objectives and Technical Challenges

This project aims to achieve the following objectives:

1) Build two CNN models with similar structures as those in the original paper [1] in functional structure using TensorFlow
2) Implemented similar data augmentation methods
3) Train the models and compare our result with that of the original paper

The biggest challenge for this project is low computation resources and limited time. The original paper built two CNN models and trained 235 epochs for 26 hours, 108 epochs for 36 hours separately using Google Colab. Similarly, we only have one NVIDIA Tesla K80 GPU and Google Colab for training models.

B. Difference with original paper

1) Training process: Because of the time limitation, we modified some training steps to shorten the training process. For model 1, we only trained half of the original epoch

[1]https://github.com/ZohebAbai/Tiny-ImageNet-Challenge
[2]https://imgaug.readthedocs.io/en/latest/

numbers for each stage. For different size of input image, instead of setting different batch size for training, we used same 128 batch size number for any size of training images. For model 2, in the original paper, for each epoch, the number of steps is (training images (100000) // batch size (128)) for each epoch. In our project the number of steps was shrunk to 200 for the first 80 epochs, and for the last 20 epochs, the number of steps is 312, calculated by (training images (80000) // batch size (256)).

2) Data augmentation: As described in Section II, the original paper applied various kinds of augmentation, whereas the process was implemented in random order and on a random segment of training samples. Also, the original paper did not specify the parameters they used in the paper. We choose our own parameter based on the examples in the imgaug library [3] and our dataset. Details of the parameter settings and explanations can be found in our GitHub[4]. Figure 2 shows an example of our orginal and augmented images. Given the two reasons, though we use the same kind of data augmentation methods as in the original paper, the result would be different.

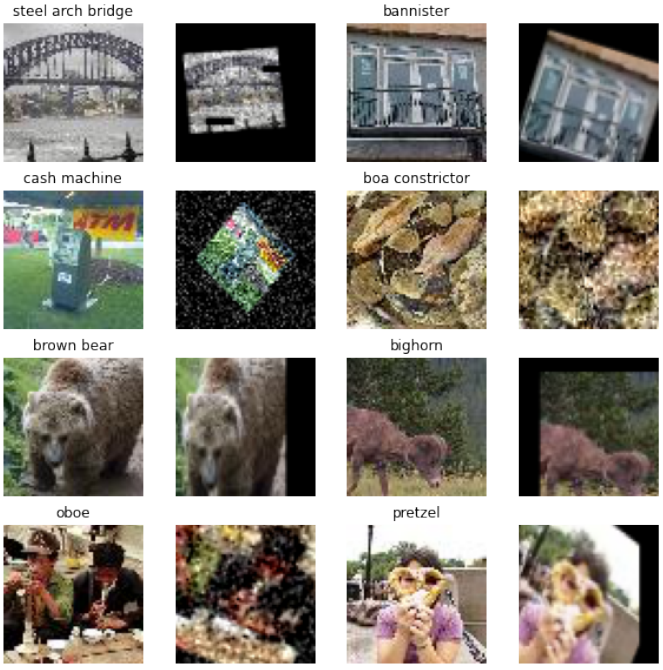

Fig. 2: Example of augmented data. First and third columns are original images, second fourth columns are augmented images.

3) Model evaluation: In the original paper, the author used 100,000 training data and 20,000 validation data to do the model training and hyperparameter tuning, and finally achieved the highest validation accuracy of 59.5% for model 62.7%. They tuned hyperparameter based on the validation accuracy in the training process, thus the final highest validation accuracy could be a biased evaluation for the model. For our project, we split the data set to training (80,000 images), validation (20,000 images) and test data set (20,000 images). We trained the models using training set and chose the best model based on the highest validation accuracy, and then use the best model to predict test data to get test accuracy.

[3]https://imgaug.readthedocs.io/en/latest/

[4]https://github.com/ecbme4040/e4040-2020fall-project-ssss-hw2761-mc4795-yc3713

IV. IMPLEMENTATION

A. Deep Learning Network

1) Data: The dataset we use for training and validation is the same as the dataset in the original paper [1], i.e. the Tiny ImageNet dataset [3]. The dataset contains 200 classes with 100,000 training samples, 10,000 validation samples, and 10,000 testing samples. All of the samples are images with $64x64$ pixels, and the training and validation samples are labeled.

For this project, we use 80% of 100,000 training samples for training and 20% of that for validation. The original 10,000 validation samples are used as testing set.

2) Model 1: For model 1, the structure is similar to the architecture of Network 1 in the original paper, which implemented the ResNet-34 with modifications. By following the Network 1 architecture shown in Figure 1 and the description of original paper, we set up 3 ”bottleneck” blocks, and each block contains 5 2D convolution layers with increasing number of filters and 1 MaxPooling layer. Moreover, after each convolution layer, a Batch Normalization and a ReLU activation layer were applied. For output of each block, we applied it to space_to_depth function before concatenating each blocks, because we want to ensure that the spatial dimensions of both the layers are equal. Then in order to keep the information of both two blocks, we add skip connections by concatenating the output of one block with the output of the next block. Finally we added a 2D convolution layer with kernel size 1x1 and a GlobalAveragePooling layer, which ensures that we can train the model better with input images of any size. In terms of coding, We build add_layer, add_block, and concate_blocks functions to put up the model.

3) Model 2: We build the second model using the same structure of network 2 in the paper. Because the structure of the two models are very similar, we apply the same functions add_layer, add_block, and concate_blocks in the models.py to put up the second models.

B. Training process

1) Model 1: For Model 1, the training process consists of 5 continuous subprocesses as follows:

1) 8-epoch training with 32x32 size data
2) 15-epoch training with 64x64 size data
3) 5-epoch training with 16x16 size data
4) 15-epoch training with 64x64 size data
5) 75-epoch training with 64x64 augmented data

For each of these subprocesses, the batch size was set to 128, and a callback function ReduceLROnPlateau was used to reduce learning rate if a relatively small change of validation loss between epochs is seen. And our parameter setting of function ReduceLROnPlateau was same as the setting in original paper.

2) Model 2: In the original paper, the learning rate range was manually tuned after seeing the validation accuracy and training accuracy for every 12 or 24 epochs. They started with learning rate range 1e-4 to 6e-4 and tried learning rate range between 1e-5 and 6e-5, 1e-6 and 6e-6, 1e-7 and 6e-7 according to the results. We also manually tune the learning rate range and also changed the batch size in the learning rate process.

Epoch	Base_lr	Max_lr	Batch_size	Max Val.acc
1-25	1e-4	6e-4	256	16.67
25-50	1e-4	6e-4	256	33.05
50-83	1e-4	6e-4	256	33.38
83-100	1e-4	6e-4	128	36.48
100-123	1e-5	6e-5	128	44.75

3) Tools: Our models are built with TensorFlow 2.2 in Python. The data loading, data augmentation, training and evaluation processes are all wrapped in functions. Due to limited computation resource, we train model 1 on Google Cloud Platform (GCP) and model 2 on Google Colab seperately, with model 1 takes about and the model 2 takes about 26 hours.

4) Saving strategy: Due to the low computation power provided by Google Colab and the limitation of computing resource provided by GCP, the session ended after 12 hour continuous computation and it had to be reconnected. But some parts of our training process required more than 12 hours to run, so we set up two folders, model and checkpoint, for manually saving the current models and for automatically saving models with the highest validation accuracy. Besides, we also manually saved the training history to document files in case the running session was interrupted accidentally.

V. Results

A. Project Results

1) Model 1: For each of 5 subprocess of training process, the validation accuracy is 18%, 37%, 26%, 45% and 41% respectively. After training, the training accuracy reached around 42%. The training time is around 23 minutes, 2.7 hours, 5 minutes, 2.7 hours and 13.6 hours respectively, so the total training time for model 1 is around 19.5 hours. For running on testing data, model 1 has reached accuracy of 55%. Based on the loss curve plot, it seems that that the overfitting problem occured around epoch 40, which is in the second training subprocess.

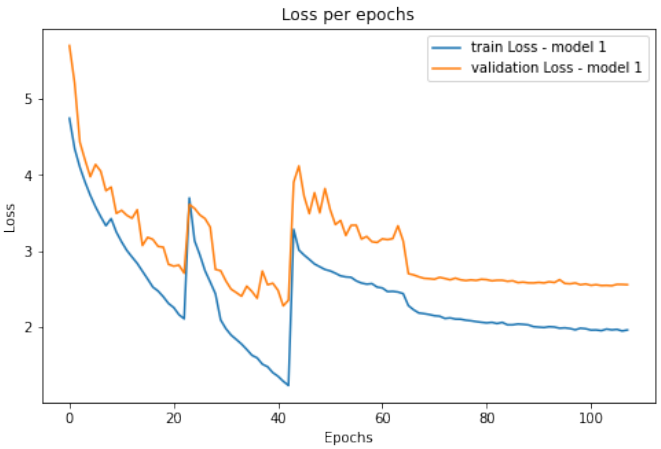

Fig. 3: Loss curve of Model 1, our result

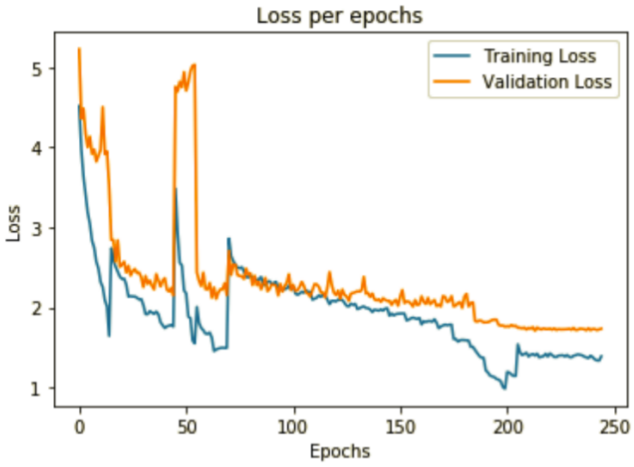

Fig. 4: Loss curve of Model 1, original paper

2) Model 2: The validation accuracy achieves 45% and the training accuracy achieves 57%after 123 epochs. The lowest validation loss is lower than the one in the original paper while the accuracy is not as well as the original one. The

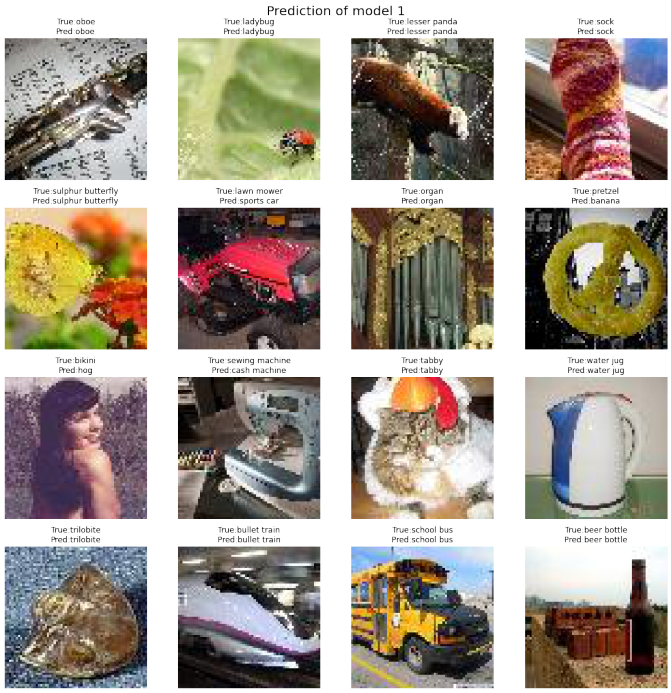

Fig. 5: Prediction of Model 1

training time before 83 epochs, with batch size 256 , is about 12 minutes per epoch, and the training time after 83 epochs, with batch size 128, is about 18 minutes per epochs. The total training time is about 26 hours.

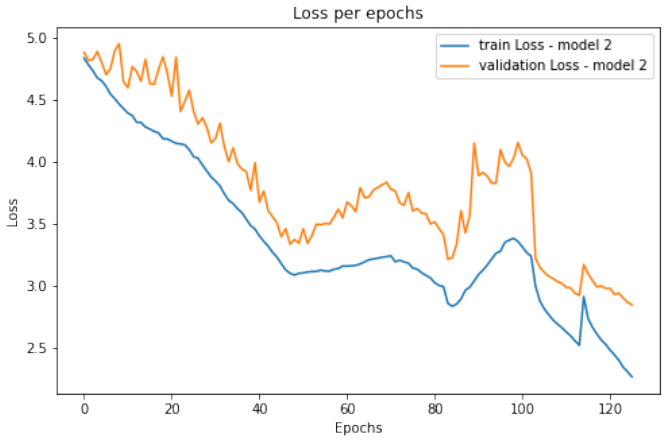

Fig. 6: Loss curve of Model 2, our result

B. Comparison of the Results Between the Original Paper and Students' Project

1) Data Augmentation: In the original paper, the data augmentation generator has a specific random seed, which ensures that for every training cycle, the data that was trained on is the same. However, we do not specify a certain seed for the data augmentation generator, which brings to the problem that every time we restart the runtime, the augmentation data set is different. The lower accuracy result in our project might be partially due to this problem. However, if trained appropriately and trained for a longer period of time, our models should generalize better on unseen data set.

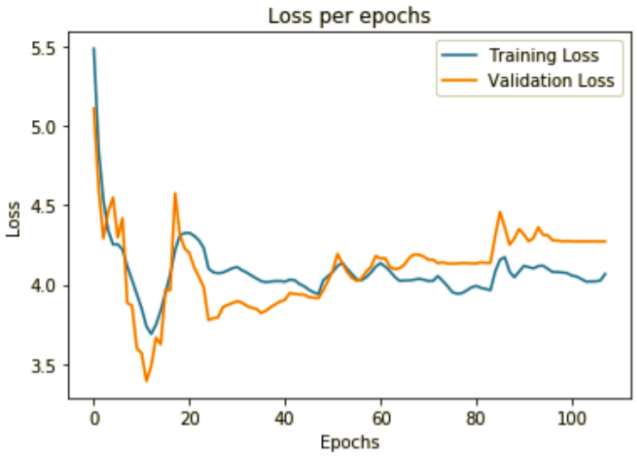

Fig. 7: Loss curve of Model 2, original paper

2) Model 1: In the original paper, the Network 1 reached validation accuracy of 25% after the first training subprocess, but the researchers noticed overfitting problem in their model. Then they made the second training subprocess and the Network 1 reached the validation accuracy of 48%, and it did not change drastically after the fourth training subprocess. They stopped training when the validation accuracy reached around 59%, and at this point, the training accuracy was 67%. Compare to the original paper, we did not reach the same or higher accuracy for both validation and training, and our training time was shorter, without counting the retraining time caused by the disconnection problem of GCP or Google Colab.

3) Model 2: The second network in the original paper reached to an highest accuracy of 53% after the first 24 epochs. When applying the same learning rate range and a larger batch size, the accuracy stuck at around 33% accuracy and stop increasing. We then shrink the batch size and decrease the learning rate range, and the accuracy starts to increase. For epochs number 100 to 110, the accuracy increases to 43.75% and fluctuates around 43% from the 110th to 123rd epochs. We should definitely try a smaller learning rate range for more epochs if time permits.

C. Discussion of Insights Gained

For both model 1 and model 2, we got different results comparing to the original paper. Given the comparison and

thinking about our model implementation, we think the main reasons of difference as follows:

1) We trained the models with smaller number of epochs, different batch sizes, and different parameter settings in data augmentation.
2) The data augmentation in our models may be different from the original paper, which is possible to cause different results. The random seed was set in augmentation in the original paper, while we did not specify a random seed in the data augmentation.
3) Learning rate is a very important hyperparameter that can affect overall accuracy. In addition to applying some learning rate scheduler or Cyclic learning rate, we can also manually tune learning rate in the training process based on the results.

D. Comparison with other paper(s)

The classification accuracy with Tiny image data set can definitely achieve a higher number with a more complicated model. In the paper *Deep Residual Learning for Image Recognition* [4], a 152 layers Residual Neural Network is built to classify the ImageNet dataset. It performs extremely well and has only a 3.57% error on the ImageNet test set. However, it is much more complicated. It has 11.3 billion FLOPs and trained for 64k iterations on two GPUs, while our models only have 16 layers and 14 layers. The DenseNet models are definitely much more smaller and easier to build compared to the 152 Residual Neural Network and thus are definitely useful and meaningful.

VI. Conclusion

In this project, we reproduce the result of [1], build two CNN models, and compare the result with the original paper. We get a 55% test accuracy on the first model and 57% on the second model.

The model accuracy is heavily influenced by the quality of training data, of which data augmentation plays an important part. There might be some corner cases that our dataset did not cover. For future improvements, we could try more ways of data augmentation, apply decaying learning rate in the training process, try different batch sizes and see how accuracy changes, and train for more epochs.

VII. Acknowledgement

Code for our project an be found in our GitHub [5], and models can be found on Google Drive through this link: https://drive.google.com/drive/folders/1oVAgrx2Ast4yQ_blt_eJqqsfzCGn99SA?usp=sharing.

References

[1] Z. Abai and N. Rajmalwar, "Densenet models for tiny imagenet classification," 2020.

[2] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[3] *Tiny ImageNet Dataset*. [Online]. Available: http://cs231n.stanford.edu/tiny-imagenet-200.zip

[4] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[5] Y. C. Mingfang Chang, Han Wang. (2020) E4040 2020fall final project (team: Ssss). [Online]. Available: https://github.com/ecbme4040/e4040-2020fall-project-ssss-hw2761-mc4795-yc3713

VIII. Appendix

A. Individual Student Contributions in Fractions

UNI	Name	Contribution(%)	What I did
mc4795	Mingfang Chang	$\frac{1}{3}$	Initial implementation and training of model 1, report writing.
hw2761	Han Wang	$\frac{1}{3}$	Initial implementation and training of model 2, report writing.
yc3713	Yingyu Cao	$\frac{1}{3}$	Organize code structure and write functions of loading data, data augmentation, evaluation, and visualization. Write report.

TABLE I: Workload of Team Members