DenseNet Models for Tiny ImageNet Classification

E4040.2021Fall.HHHH.report

Ruize Yu ry2404, Taotao Jiang tj2441, Xiaohang He xh2509
Columbia University

Abstract

In this paper, we reproduce the two image classification models on the Tiny ImageNet dataset by Zoheb Abai and Nishad Rajmalwar. Both networks are built from scratch. The architecture is designed based on the tiny-imagenet-200 data set. After calculating the receptive field of the convolutional layer to complete the model architecture, image augment and cyclical learning rate are also used to improve the accuracy of the model. The whole Networks are constructed and trained on Google Colab, which has limited computation resources and high constraints. We successfully constructed similar models and reached the accuracy close to the original paper.

1. Introduction

Image network classification is a common core task in computer vision, and it has been continuously improved. The challenge in image classification is to extract quantifiable features from three channels, namely red, blue and green. Before CNN, we needed to spend a lot of time on feature extraction algorithms.

Beginning in 2013, Convolutional Neural Networks (CNN) began to dominate the field of image classification. It covers local and global functions, and also learns different features from images. These advantages have enabled CNN to achieve amazing results in deep learning in the direction of image classification. In 2017, a new CNN, DenseNet, appeared. DenseNet is a new CNN architecture that uses fewer parameters to achieve the most advanced results on classification data sets.

The original paper "DenseNet Models for Tiny ImageNet Classification" by Zoheb Abai and Nishad Rajmalwar presents a method to build classification models Tiny ImageNet. Their goal is reaching 60% validation accuracy with the custom models inspired from DenseNet architecture [1] without any pre-trained network. We aim to reproduce the results of their work, construct similar networks and reach higher verification accuracy.

We encountered two big challenges during the whole procedure. The first challenge is that we use the same build environment as the original author, namely Google Colab. Its computing power and processing speed are very limited, which causes that the training process was repeatedly terminated. Therefore, we separate our training epochs into several groups to decrease the running time of each training. Also, in order to increase the computational speed of our convolutional network, we modified the original Network 1 by adding dense layers and dropout layers. Our second challenge is the new method of setting the cyclical learning rate in the original paper. To achieve the similar effect, we quoted Brad Kenstler's method on github [2].

2. Summary of the Original Paper

2.1 Methodology of the Original Paper

Our team reviewed the methodology of the original paper from 4 dimensions: basic infrastructure, determinants of layers, model architecture as well as optimization techniques.

2.1.1 Basic Infrastructure

For the original paper, the authors utilized DenseNet as the basic architecture. Traditional deep neural networks often face a knotty problem of gradient vanishing, which was luckily solved by the implementation of ResNet. ResNet won the championship of ILSVRC 2015 in image classification & detection for its superior prediction accuracy and unparalleled ability to allow deeper networks. As is shown below, the idea is to include a shortcut connection skipping several weight layers so that input X could be added to the model output.

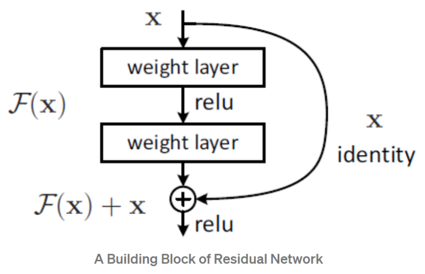

Figure 1: ResNet Block [3]

In Comparison to ResNet, DenseNet combines features by concatenation rather than aggregate them by summation. Additionally, the mechanism of DenseNet is to consider all results/feature maps from preceding layers as inputs for the current layer and pass the current output as input for every subsequent layer. Such design results in thinner and more compact networks, allowing fewer channels in configuration. Due to external limitations, the authors adopted DenseNet as the principle project infrastructure to make the computational procedure and memory utilization more efficient.

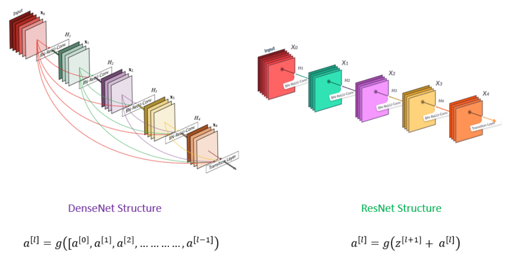

Figure 2: Comparison between DenseNet and ResNet

2.1.2 Determinants of Layers

The paper theoretically determines the optimal number of layers by firstly calculating the receptive fields. As illustrated by the following image, the receptive field is the region size on the initial matrix plot projected by the pixel points from each output feature map. It represents the extent to which a layer is influenced by the original image. The higher the receptive field, the more details the layer illustrates.

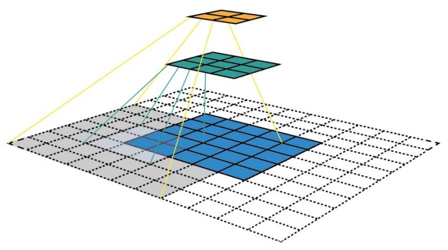

Figure 3: Projection for Receptive Field

The following formula shows how to calculate the receptive field for each layer. l_{k-1}is the receptive field for layer k-1, f_{k-1} is the kernel size for layer k-1, s_{k-1} is the stride for layer k-1.

$$l_k = \begin{cases} l_{k-1} + [(f_k - 1)] \dots\dots\dots\dots\dots (k = 1) \\ l_{k-1} + [(f_k - 1) \times \prod_{i=1}^{k-1} s_i] \dots\dots (k \geqslant 2) \end{cases}$$

Figure 4: Receptive Field Formula

Take the original paper for example, the authors chose kernel size = 3 * 3 with strides = (1, 1). For the first layer, the receptive field should be the same as filter size. Each receptive field for the following layer would then increase by 2 since the matrix dimension is decreasing by 2. It could be calculated by the kernel size and strides (spatial dimensions = 1+ (image size - kernel size)/stride). In particular, the max pooling layer shows a quite different picture, halving the spatial dimensions triggers the entire receptive field to double instead. Below are the receptive fields for the two networks. Worth to mention, the authors raised an interesting concern that many images are too confusing to detect without the consideration of backgrounds, which means that projecting the whole image size is insufficient. Consequently, they attempted to increase the network layers until the capture of over two times the original image size, which should be 128 * 128.

Network 1

- Convolution Layers 1 - 5 = 11x11 (11 = 3+2+2+2+2)
- MaxPooling Layer 6 = 22x22
- Convolution Layers 7 - 11 = 32x32 (32 = 22+2+2+2+2+2)
- MaxPooling Layer 12 = 64x64
- Concatenation Layer 13 = No Change
- Convolution Layers 14 -18 = 74x74 (74 = 64+2+2+2+2+2)
- MaxPooling Layer 19 = 148x148
- Concatenation Layer 20 = No Change

Network 2

- Convolution Layer 0 = 3x3
- Convolution Layers 1 - 4 = 11x11 (11 = 3+2+2+2+2)
- MaxPooling after Concatenation Layer 5 = 22x22
- Convolution Layers 6 - 9 = 30x30 (30 = 22+2+2+2+2)
- MaxPooling after Concatenation Layer 10 = 60x60
- Convolution Layers 11 - 14 = 68x68 (68 = 60+2+2+2+2)
- MaxPooling after Concatenation Layer 15 = 136x136

Figure 5: Receptive Fields for Networks [1]

2.1.3 Model Architecture

There were in total 4 models adopted by the authors in the original paper. Initially, they considered and implemented the vanilla ResNet-34 and ResNet-50 models. For ResNet-34, with a batch size of 500 images and epochs of 100, ~99% accuracy was achieved on the training dataset and ~34% on the validation dataset. In terms of ResNet-50, with a batch size of 512 images and epochs of 50, ~49% accuracy was reached on the training set versus ~26% on the validation set. Such results reflected the learning power of the ResNet model, but it indicated not only a poor generalization capability but also an early saturation in validation accuracy (26% close to 34% with a big gap of 50 in epochs). It's probably due to the negligible ability to learn for the marginal layers added in the case of 64*64 images. Thus, the authors transitioned to modified custom networks- DenseNet, where a shallow network was configured of ~15 convolution layers and ~ 1000 channels.

2.1.3.1 Network 1

The network flow chart is as follows:

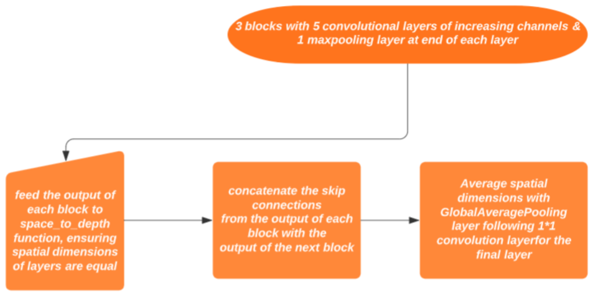
3 blocks with 5 convolutional layers of increasing channels & 1 maxpooling layer at end of each layer
feed the output of each block to space_to_depth function, ensuring spatial dimensions of layers are equal
concatenate the skip connections from the output of each block with the output of the next block
Average spatial dimensions with GlobalAveragePooling layer following 1*1 convolution layerfor the final layer

Figure 6: Network 1 Flowchart

2.1.3.2 Network 2

Network 2 is adjusted based on the ResNet-18 model, shown as follows:

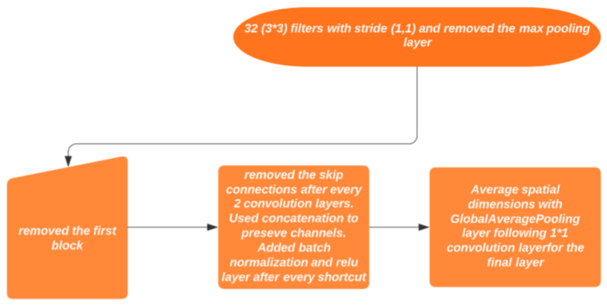
32 (3*3) filters with stride (1,1) and removed the max pooling layer
removed the first block
removed the skip connections after every 2 convolution layers. Used concatenation to preseve channels. Added batch normalization and relu layer after every shortcut
Average spatial dimensions with GlobalAveragePooling layer following 1*1 convolution layerfor the final layer

Figure 7: Network 2 Flowchart

2.1.4 Optimization Techniques

2.1.4.1 Image Augmentation

For network 1:

- Indirect methods: change image resolution to feed for various epoch stages.
- Direct methods: conducted image augmentation at 50% validation accuracy.
    - Scale
    - Rotate
    - CoarseDropout
    - Additive Gaussian Noise
    - Crop and Pad

For network 2:

- No change in image resolution
- Applied 11 types of image augmentation to half the dataset at the beginning of training. Augmentation techniques include Horizontal Flip, Vertical Flip, and etc.

2.1.4.2 Key Attributes Selection

For network 1:

- Batch Normalization: standardizes the inputs to a layer for each mini-batch to dramatically increase the computing speed and to stabilize the activations [6]
- Adam optimizer, 1e-3 learning rate and ReduceLRonPlateau callback function

For network 2:

- Batch Normalization
- L2 regularizer, variance scaling kernel initializer
- Adam optimizer, 1e-4 learning rate and epsilon of 1e-8
- Cyclical learning rate

2.2 Key Results of the Original Paper

2.2.1 Accuracy & Conclusion

In terms of network 1, first of all, with a batch size of 256, the 32*32 resolution images were trained for 15 epochs, achieving 25% validation accuracy. To reduce overfitting, 64*64 resolution images were trained for 30 epochs with an improved validation accuracy of 48%. Then 16*16 resolution images were fed into the model for 10 epochs, followed by 64*64 resolution images training for 30+ epochs, presenting a score of 48-49%. Finally, after image augmentation and running for another 150 epochs, the best model achieved 67% training accuracy and 59.5% validation accuracy.

As for network 2, the original paper evaluates 108 epochs with 128 batch size. The result shows a steep decline in loss in early training stages thanks to cyclical learning rate. The network loss landed on a plateau after 70~ epochs and finally it's great to see ~63% validation accuracy.

2.2.2 Errors & Limitation Analysis

Most correctly classified images have clear boundaries between the entity and the background. For incorrectly classified ones, the errors can be attributed to 3 reasons:

- low image resolution: models could only detect the type of image, such as cat or dog, but they aren't able to detect the more specific classification.
- entity misunderstanding: multiple entities appear in the image, making the model hard to determine the intent of the image.
- similar items confusion: some categories are too close to be set apart.

Although the authors tried to oversample the misclassified labels and soft weigh them, no improvement in validation accuracy happened.

3. Methodology (of the Students' Project)

3.1. Objectives and Technical Challenges

The objectives of our paper are to reproduce the DenseNet Models using TensorFlow 2.0 and improve the models with Tiny ImageNet dataset [4] in Google Colab environment.

The most challenging parts are:

1. Computation sources limitation: Google Colab does not have the abundant computing resources of a virtual machine. It strictly limits the time the script can run each time. Since the original article model is two neural network models composed almost entirely of convolutional layers, they require a very long calculation time and a lot of computation resources. We have encountered many times that the calculation of the model was forcibly terminated by Google Colab during half of the training.
2. Accuracy issue: At the beginning of our training process, the training accuracy kept at a pretty low level. We also encountered overfitting problem

3. Cyclical learning rate: The original paper mentions that they use a cyclical learning rate to train Network 2, which changes the learning rate within the value range of the specified number of iterations. But the method he used is only for Tensorflow 1.0, which is not available for Tensorflow 2.0.

3.2.Problem Formulation and Design Description

To solve the above problems, we design the following approaches.

First of all, because one of the highlights of the original paper's models is that they have achieved good accuracy in a highly restrictive environment which is Google Colab, we decided not to change the network training environment. However, the training time required by our models far exceeds the allowable running time of Colab. In order to solve the shortage of computing resources, we splitted our training procedure by separating 235 epoc-hs into several groups. Although the speed of training each epoch is still very slow, we solved the problem that Google Colab will disconnect during the training, enabling us to run the complete training process of the model.

For the cyclical learning rate of Network 2, we used the strategy provided by Brad Kenstler on Github [2] to achieve a similar effect.

In order to further improve the model, we have made improvements on the basis of the original model. We have added some dropout layers and dense layers to reduce computing time and required resources, and to improve accuracy.

4. Implementation

In our project, we implemented the Dense Convolutional Network on the Tiny ImageNet dataset [4]. When we reproduced the exact same models as the original paper, we found the model is overfitting with training acc 85%. Thus, we did image augmentation to avoid this problem. After reproducing the models and achieving the required accuracy, we also did some modification to Network 1 for a higher accuracy.

4.1 Data

4.1.1 Data Collection

Since the link to the dataset in the reference of the original paper has expired, we chose to search for the keyword "Tiny ImageNet dataset" on Google to find a suitable dataset. We found the dataset of Stanford University CS231n used in a competition on Kaggle. We downloaded it and used it as the data set for our paper.

4.1.2 Data Description

The Tiny ImageNet dataset [4] contains 110,000 images, of which 100,000 are training sets and 10,000 are the validation set. The images belong to 200 classes. The resolution of the image is only 64x64 pixels. Obtaining information with such a low pixel size is not that easy. In addition, it is difficult for the human eyes to detect objects in the images because most images are similar and blurry. These factors make it more challenging to extract information from it.

4.1.3 Data Preprocess

Before the training, we preprocessed the image data. We normalized the input image data. The formula for min-max normalization is:

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

For RGB images, this is equivalent to doing 1./255 since the pixel values of the image will be between 0 and 1.

4.1.4 Image Augmentation

After training Network 1 with 32×32, 64×64, and 16×16 resolutions for 85 epochs, we noticed that we had an overfitting problem due to the abnormal high training accuracy. So we did image augmentation to avoid this problem. We decided to use the similar methods as the original paper did. To achieve the similar augmentation, we used the ‘imgaug’ package which is a simple but powerful package for image augmentation.[4] Five kinds of augmentation methods are applied:

- scale: (0.5, 1.5)
- rotate: 20
- CoarseDropout: (0.0, 0.2), size_percent=(0.05, 0.07)
- AdditiveGaussianNoise: scale=0.05*255
- CropAndPad: percent=(-0.25, 0.25)

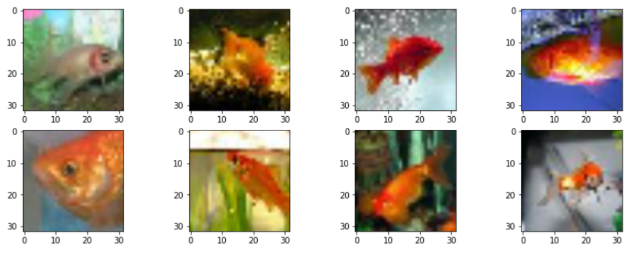
Figure 8: Images before augmentation

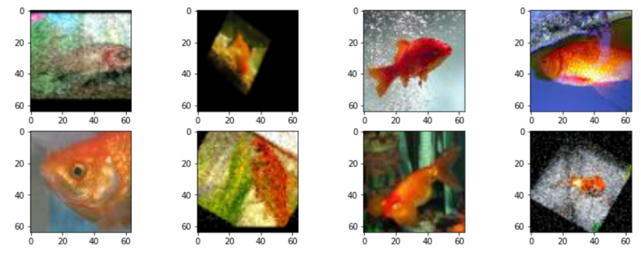
Figure 9: Images after augmentation

To avoid the overfitting also happens to Network 2, we did the image augmentation before training. The difference is that we only choose some of these methods for augmentation in each training:

- Fliplr: (0.5)
- Flipud: (0.2),
- GaussianBlur: sigma=(0, 2.0)
- CropAndPad:percent=(-0.1,0.2)
- scale: {"x": (0.8, 1.5), "y": (0.8, 1.5)}
- translate: {"x": (-0.2, 0.2), "y": (-0.2, 0.2)}
- rotate: (-45, 45)
- shear: (-16, 16)
- CoarseDropout: (0.03, 0.15), size_percent=(0.02, 0.05), per_channel=0.2
- Multiply: (0.8, 1.2), per_channel=0.2
- ContrastNormalization:(0.75,1.5), per_channel=0.5

4.2 Deep Learning Network
4.2.1 Architectural Block Diagrams

After reproducing the two Networks of the original paper, we added some Dense layers and Dropout layers to Network 1. The following are the architectural block diagrams of the two Networks.

Modified Network 1:

Layer	Output Shape
InputLayer	(None, None, None, 3)
Conv2D	(None, None, None, 32)
BatchNormalization	(None, None, None, 32)
Activation_relu	(None, None, None, 32)
Conv2D	(None, None, None, 64)
BatchNormalization	(None, None, None, 64)
Activation_relu	(None, None, None, 64)
Conv2D	(None, None, None, 128)
BatchNormalization	(None, None, None, 128)
Activation_relu	(None, None, None, 128)
Conv2D	(None, None, None, 256)
BatchNormalization	(None, None, None, 256)
Activation_relu	(None, None, None, 256)
Conv2D	(None, None, None, 512)
BatchNormalization	(None, None, None, 512)
Activation_relu	(None, None, None, 512)
MaxPooling2D	(None, None, None, 512)
Conv2D	(None, None, None, 64)
BatchNormalization	(None, None, None, 64)
Activation_relu	(None, None, None, 64)
Conv2D	(None, None, None, 128)
BatchNormalization	(None, None, None, 128)
Activation_relu	(None, None, None, 128)
Conv2D	(None, None, None, 256)
BatchNormalization	(None, None, None, 256)
Activation_relu	(None, None, None, 256)
Conv2D	(None, None, None, 512)
BatchNormalization	(None, None, None, 512)
Activation_relu	(None, None, None, 512)
Conv2D	(None, None, None, 1024)
BatchNormalization	(None, None, None, 1024)
Activation_relu	(None, None, None, 1024)
Lambda	(None, None, None, 2048)
MaxPooling2D	(None, None, None, 1024)
Concatenate	(None, None, None, 3072)
Conv2D	(None, None, None, 32)
BatchNormalization	(None, None, None, 32)
Activation_relu	(None, None, None, 32)

Conv2D	(None, None, None, 128)
BatchNormalization	(None, None, None, 128)
Activation_relu	(None, None, None, 128)
Conv2D	(None, None, None, 256)
BatchNormalization	(None, None, None, 256)
Activation_relu	(None, None, None, 256)
Conv2D	(None, None, None, 512)
BatchNormalization	(None, None, None, 512)
Activation_relu	(None, None, None, 512)
Conv2D	(None, None, None, 1024)
BatchNormalization	(None, None, None, 1024)
Activation_relu	(None, None, None, 1024)
Lambda	(None, None, None, 12288)
MaxPooling2D	(None, None, None, 1024)
Concatenate	(None, None, None, 13312)
Conv2D	(None, None, None, 200)
BatchNormalization	(None, None, None, 200)
GlobalAveragePooling2D	(None, 200)
Dense	(None, 1024)
BatchNormalization	(None, 1024)
Activation_relu	(None, 1024)
Dense	(None, 512)
BatchNormalization	(None, 512)
Activation_relu	(None, 512)
Dropout	(None, 512)
Dense_softmax	(None, 200)

Table 1: Network 1 Architectural Block Diagrams

Original Network 2:

Layer	Output Shape
InputLayer	(None, 64, 64, 3)
Conv2D	(None, 64, 64, 32)
BatchNormalization	(None, 64, 64, 32)
Activation_relu	(None, 64, 64, 32)
Conv2D	(None, 64, 64, 128)
BatchNormalization	(None, 64, 64, 128)
Activation_relu	(None, 64, 64, 128)
Conv2D	(None, 64, 64, 128)
BatchNormalization	(None, 64, 64, 128)
Activation_relu	(None, 64, 64, 128)
Conv2D	(None, 64, 64, 128)
BatchNormalization	(None, 64, 64, 128)
Activation_relu	(None, 64, 64, 128)
Conv2D	(None, 64, 64, 128)
BatchNormalization	(None, 64, 64, 128)
Activation_relu	(None, 64, 64, 128)
Concatenate	(None, 64, 64, 160)
MaxPooling2D	(None, 32, 32, 160)
Conv2D	(None, 32, 32, 256)
BatchNormalization	(None, 32, 32, 256)
Activation_relu	(None, 32, 32, 256)
Conv2D	(None, 32, 32, 256)
BatchNormalization	(None, 32, 32, 256)
Activation_relu	(None, 32, 32, 256)
Conv2D	(None, 32, 32, 256)
BatchNormalization	(None, 32, 32, 256)
Activation_relu	(None, 32, 32, 256)
Conv2D	(None, 32, 32, 256)
BatchNormalization	(None, 32, 32, 256)
Activation_relu	(None, 32, 32, 256)
Concatenate	(None, 32, 32, 416)

BatchNormalization	(None, 32, 32, 416)
Activation_relu	(None, 32, 32, 416)
MaxPooling2D	(None, 16, 16, 416)
Conv2D	(None, 16, 16, 512)
BatchNormalization	(None, 16, 16, 512)
Activation_relu	(None, 16, 16, 512)
Conv2D	(None, 16, 16, 512)
BatchNormalization	(None, 16, 16, 512)
Activation_relu	(None, 16, 16, 512)
Conv2D	(None, 16, 16, 512)
BatchNormalization	(None, 16, 16, 512)
Activation_relu	(None, 16, 16, 512)
Conv2D	(None, 16, 16, 512)
BatchNormalization	(None, 16, 16, 512)
Activation_relu	(None, 16, 16, 512)
Concatenate	(None, 16, 16, 928)
BatchNormalization	(None, 16, 16, 928)
Activation_relu	(None, 16, 16, 928)
MaxPooling2D	(None, 8, 8, 928)
Conv2D	(None, 8, 8, 200)
GlobalAveragePooling2D	(None, 200)
Activation_Softmax	(None, 200)

Table 2: Network 2 Architectural Block Diagrams

4.2.2 Training Algorithm Details

Original Network 1:

- epoch 1-15: Input images with 32 × 32 input resolutions, use ReduceLROnPlateau as the learning rate reducer, batch size 256
- epoch 15-45: Input 64 × 64 resolutions images, do not use learning rate reducer, batch size is 64
- epoch 45-55: The batch size of images with 16 × 16 input resolutions is 256
- epoch 55-85: The batch size of images with 64 × 64 input resolutions is 64
- Perform Image Augmentation
- epoch 85-235: The batch size of 64 × 64 input resolutions is 64

Original Network 2:

- Perform random Image Augmentation
- The pixels of the input image remain 64 × 64, and the batch size is 128.
- Use cyclical learning rate for training.

Modified Network 1:

- epoch 1-20: Input images with 32 × 32 input resolutions, use ReduceLROnPlateau as the learning rate reducer, batch size 256
- epoch 20-50: Input 64 × 64 resolutions images, do not use learning rate reducer, batch size is 64
- epoch 50-60: The batch size of images with 16 × 16 input resolutions is 256
- epoch 60-75: The batch size of images with 64 × 64 input resolutions is 64
- Perform Image Augmentation
- epoch 75-135: The batch size of 64 × 64 input resolutions is 64

4.2.3 Flowchart

Below is the flowchart of the two original Networks.

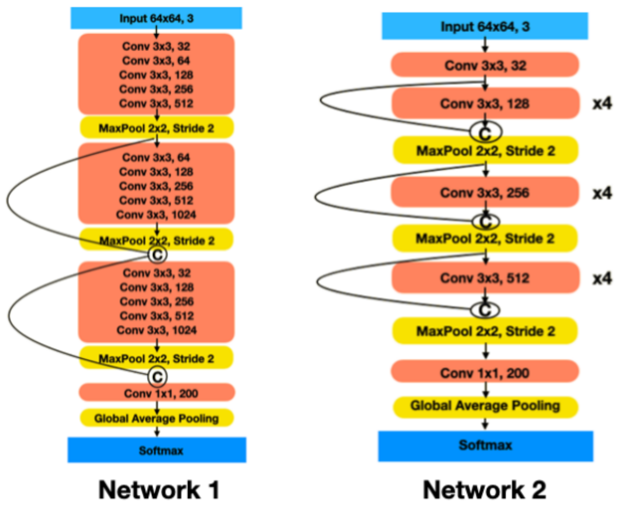

Figure 10: Network flowchart [1]

4.2.4 Data used

The training set we used involves 100,000 images belonging to 200 classes.

The validation set we used involvesd 10,000 validated image filenames belonging to 200 classes.

4.3 Software Design

Provide the description and discussion:

1. We implemented three models. Two exactly the same compared to paper and one modified model. The flowcharts are shown in Figure X.
2. In general, the models from the original paper consist of three similar blocks. Each block contains 4 to 5 convolution layers, a maxpool layer and a concatenation layer. The output is calculated by a global average pooling layer.
3. For our modified model, we added two dense layers containing 1024 and 512 units with batch normalization, ReLU activation and dropout to replace global average pooling in Network 1.

Github Link: **https://github.com/ecbme4040/e4040-2021Fall-Project-HHHH-ry2404-tj2441-xh2509**

5. Results

5.1 Project Results

Original Network 1:

For original Network 1, we first followed the procedure of the original paper and trained it with different resolutions for 85 epochs. We got a training accuracy of 84.54% and validation accuracy of 50.23%. It is indicated that the overfitting problem happened. So we did the image augmentation and continued the rest training. Finally, we reached 52.95% training accuracy and 47.22% validation accuracy. Our test accuracy is 53.76% and test loss is 2.156.

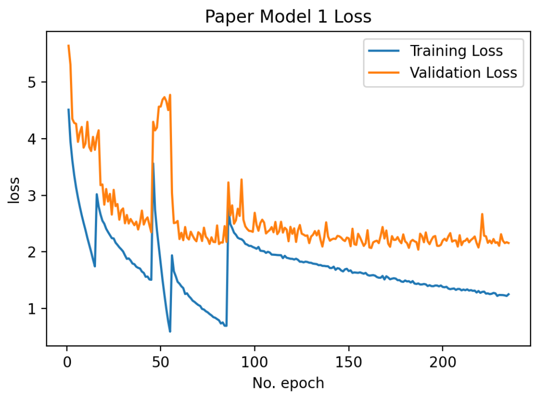

Figure 11: Original Network 1 Loss plot

Original Network 2:

For original Network 2, we finished the data augmentation and started to train the model. We got a training accuracy of 62.60% and validation accuracy of 62.15%. We reached 62.58% test accuracy and 4.163 test loss.

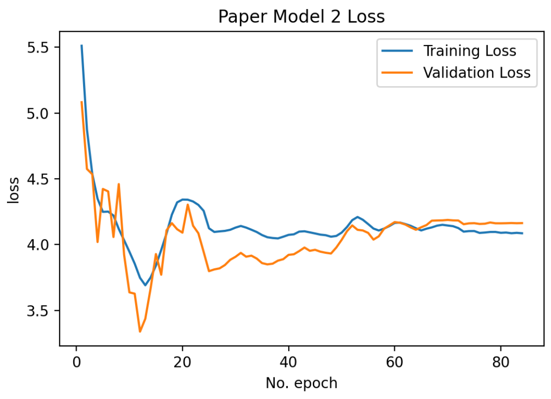

Figure 12: Original Network 2 Loss plot

Modified Network 1:

For the modified Network 1, we first fed the images with 32 × 32 resolutions for 20 training epochs. Then we fed 64 × 64 for 30 epochs and 16 × 16 for 10 epochs followed with 64 × 64 for 15 epochs. Then we performed image augmentation and fed images with 64 × 64 resolutions for 60 training epochs. Finally, we got a training accuracy of 44.09% and validation accuracy of 40.52%. Test accuracy reached 40.52% and test loss reached 2.578.

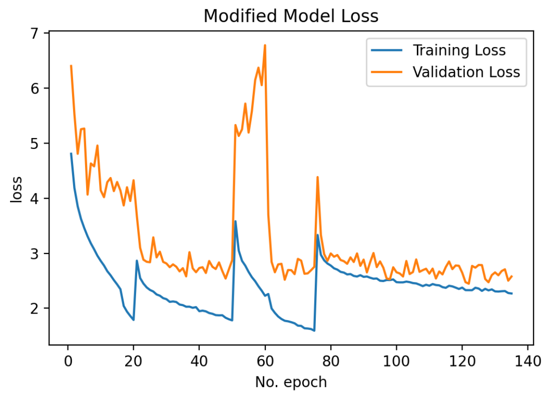

Figure 13: Original Network 2 Loss plot

5.2 Comparison of the Results Between the Original Paper and Students' Project

Network 1:

Network 1 in the original paper achieves 59% validation accuracy, 67% training accuracy, and 59.5% test accuracy. Obviously, their accuracy is higher than ours but the difference is still within an acceptable range.

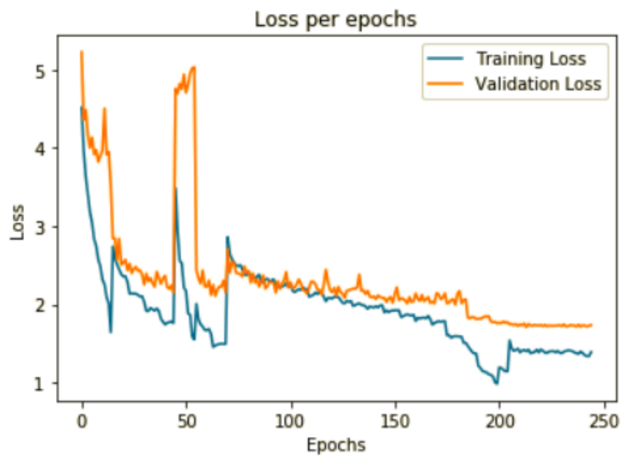

Figure 14: Loss curve of Network 1 in original paper [1]

Network 2:

For Network 2, the original paper reaches 62.73% validation accuracy of 62.73% and 68.11% training accuracy. Although our training accuracy is about six percent lower than theirs, we have achieved the same validation accuracy.

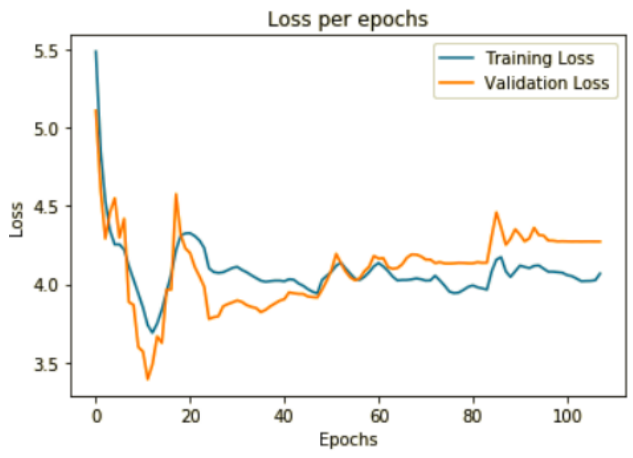

Figure 15: Loss curve of Network 2 in original paper [1]

We believed that the causes of our slightly worse results are:

1. Our data set contains lower image quality, which is not conducive to the model's acquisition of information.
2. Although the architecture of the models is almost the same. But the original author may use different parameters and other fine-tuning techniques that were not fully introduced in the original paper.
3. For the modified Network 1, we only updated 7 models, which may be the main reason for the low accuracy.

5.3 Discussion of Insights Gained

Most of our insights come from the process of training the model.

1. Variable image resolution and variable learning rate can effectively enhance the accuracy of the model. After we use CLR, our verification accuracy has increased by more than 10%.
2. For low-resolution images, image enhancement must be performed before starting to train the model to enhance the representativeness and diversity of the image, which can effectively avoid overfitting in the training process.
3. When building a neural network model, you should consider its computational efficiency to avoid work efficiency caused by too long training time.

6. Future Work

Since the original network model is composed of a large number of convolutional layers, this leads to shortcomings such as over-fitting and extremely long training time. We believe that in order to improve model performance and training efficiency, more types of neural network layers should be applied. In addition, we can also try different types of variable learning rates to improve the accuracy of the model. Also, to improve our modified Network 1, we should feed more images and update the model more times for better accuracy.

In terms of data sets, we can also use new image data sets to train the models we have obtained in the future, which will not only improve the accuracy of the model, but also improve its capacity.

7. Conclusion

The focus of this report is to summarize the key points of the original paper, to reproduce the DenseNet Network using TensorFlow 2.0 and to verify the effectiveness of the reproduced model on the tiny image dataset.

Although our results of the reproduced two Networks are slightly worse than the results achieved in the original paper, the difference is within an acceptable range. We also verified the effectiveness of data augmentation and cyclical learning rate contributed to accuracy improvement. There is still room for future performance optimization. For example, we still need to try more possible layers combinations of each Network to enhance the performance. More training epochs are also needed to improve our modified model.

8. Acknowledgement

We would like to thank Prof. Kostic and TAs for their teaching and helping in this project. During our

reproducing part, we appreciated the help of Brad Kenstler's CLR method published on Github.

9. References

[1] B. Kenstler, "Cyclical Learning Rate (CLR)," *GitHub*, Dec. 19, 2021. https://github.com/bckenstler/CLR (accessed Dec. 22, 2021).
[2] S.-H. Tsang, "Review: ResNet — Winner of ILSVRC 2015 (Image Classification, Localization, Detection)," *Medium*, Mar. 20, 2019. https://towardsdatascience.com/review-resnet-winner-of-ilsvrc-2015-image-classification-localization-detection-e39402bfa5d8.
[3] *Stanford.edu*,2020. http://cs231n.stanford.edu/tiny-imagenet-200.zip.
[4] "augmenters.arithmetic - imgaug0.4.0 documentation,"*imgaug.readthedocs.io*.https://imgaug.readthedocs.io/en/latest/source/overview/arithmetic.html (accessed Dec. 22, 2021).
[5] Z. Abai and N. Rajmalwar, "DenseNet Models for Tiny ImageNet Classification." Accessed: Dec. 22, 2021. [Online].Available:https://arxiv.org/ftp/arxiv/papers/1904/1904.10429.pdf.
[6] https://www.facebook.com/jason.brownlee.39, "Accelerate the Training of Deep Neural Networks with Batch Normalization," *Machine Learning Mastery*, Jan. 15,2019.https://machinelearningmastery.com/batch-normalization-for-training-of-deep-neural-networks/.

10. Appendix

10.1 Individual Student Contributions in Fractions

	ry2404	tj2441	xh2509
Last Name	Yu	Jiang	He
Fraction of (useful) total contribution	1/2	1/4	1/4
What I did 1	Network Design	Paper Review	Data Preparing
What I did 2	Coding	Methodology Summary	Network Design
What I did 3	Model Improvement	Report Writing	Report Writing