DenseNet Models for Tiny ImageNet Classification

E4040.2021Fall.CDNM.report
Zijun Wei zw2781
Dawei He dh3027
Zhihan Yang zy2447
Columbia University

Abstract

In this report, we present two image classification models in the paper: DenseNet Models for Tiny ImageNet Classification by Zoheb Abai and Nishad Rajmalwar. The design of these two networks was based on the idea of Densely Connected Convolutional Networks, and the whole system is set up on the Stanford Tiny ImageNet dataset. The architecture of the networks is built on the image resolution of this dataset and by computation of the Receptive Field of the convolution layers. Some non-conventional techniques are used to image augmentation. We also used the Cyclical Learning Rate method to improve the accuracy of models just like the original paper. We tried to improve the network accuracy by adjusting a few methods, but we get the results that the methods of the original paper are optimized to the best. We aimed to achieve the best validation accuracy in the original paper for both networks, and compared our results with the original paper, and did error analysis.

1. Introduction

In the original paper, they gave their approach for building classification models for the Tiny ImageNet which is a subset of ImageNet. They built two different networks which are inspired by the DenseNet architecture. [3]The solution for vanishing gradients is the core of very deep neural networking. The solution was first given by residual networks. However, DenseNet does not aggregate features through summation like ResNet; instead, they are combined by concatenating. As a result, information passed from one layer to all subsequent layers. The flow of information and gradient can be protected effectively by using this method. We replicated these two models and tried to reach the top validation accuracy in the original paper for both networks. The result is very surprising because the best validation accuracy for both networks is better than the result from the original paper. Our biggest challenge was the limited budget in Google Cloud Computing. Due to the limited budget in GCP, we need to limit the amount of 1x1 filters, fully connected layers, and dropout layers. We also tried several innovative techniques such as using different learning rate schedulers and image augmentation to improve accuracy and reduce computation time.

2. Summary of the Original Paper

2.1 Methodology of the Original Paper

The paper, DenseNet Models for Tiny ImageNet Classification authored by Zoheb Abai and Nishad Rajmalwar, proposed two custom DenseNet models with design elements of ResNet-18 for a subset of ImageNet dataset called the Tiny ImageNet. The comparison of ResNet and DenseNet is shown in Figure1.

The resolution of the images in the Tiny ImageNet dataset is only 64x64 pixels, which makes it hard to extract information from them. For this reason, the authors designed two relatively shallow neural networks with only 15 convolution layers instead of a deep network like ResNet-50. Because if the network has too many convolution layers in it, the deeper layers will act as scaler layers that map the previous layer of size 1x1 and would learn negligible information with every additional layer. Another feature of these two networks is they do not contain any fully connected layer, dropout layer and 1x1 filters.

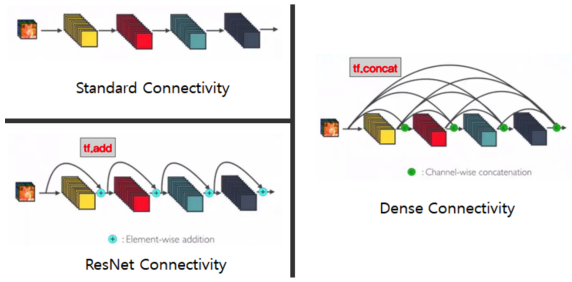

Figure1. Standard vs ResNet vs DenseNet structures

To decide how many layers are needed and what kind of layer should be deployed in the network, the authors calculated the receptive field after each layer. For the Tiny ImageNet dataset, once the receptive field of the model reaches 64x64, the model can detect the object in each image distinctly and is capable of classifying them. But the author wants to further expand the receptive field to 148x148 for the first DenseNet model and 136x136 for the second model so that models can also learn background in addition to the object in the image.

For the first custom DenseNet models, the author built 3 ‘bottleneck’ blocks containing 5 convolution layers with increasing channels. There are concatenations between blocks 2 & 3 and block 3 & the final convolution layer. Before the concatenation, the output before the skipped block reshaped the spatial dimensions and increased the depth, so two layers can be correctly connected. The output of the final convolution layer is of depth 200 and is followed by a global average pooling layer, as they act like a fully connected layer that outputs the probability for each class.

The second network is mainly based on the ResNet-18 architecture with some modifications. All filters are fixed to size 3x3 and some layers are removed to make the model only contain 15 convolution layers. Besides, replacing the skip connection between every 2 layers to every 4 layers. Similarly, the fully connected layer is replaced by a convolution layer with 200 channels and a global average pooling layer.

The authors also deployed image augmentation to increase the robustness and avoid the overfitting issue. For the first DenseNet model, images with even lower resolution 32x32 were fed to the model before the original 64x64 images were fed. These techniques improve the validation accuracy by 3-4%. After original images were fed for dozens of epochs, new augmented images generated by randomly choosing several augmentation methods were used to further train the model. This technique increases the validation accuracy by 9.5%.

For Network2, the author applied transformation at the beginning of the model run and kept 64x64 image resolution as input size throughout the complete training run. They totally performed 11 augmentations, like horizontal and vertical flip, Gaussian blur, crop and pad, scale, translate, rotate, shear, coarse dropout, multiply and contrast normalization at a random sequence with manually tuned parameters.

The authors also constrained their computation power by using only free services provided by Google Colab and training two models from scratch instead of using transfer learning of pre-trained models.

2.2 Key Results of the Original Paper

In the original paper, they ran network 1 for 235 epochs with 256 batch sizes for 32x32 and 16x16 image resolution and a batch size of 64 for 64x64 resolution. The best validation accuracy for network 1 is 59.5%. For network 2, they trained with 108 epochs with a batch size of 128 images. The top validation accuracy is 62.73%. For error analysis, they found out that almost 30% of images are not classified correctly and the top worst classified classes are ‘Plunger’, ‘Umbrella’, ‘Water Jug’, ‘Bucket’, ‘Wooden Spoon’, and ‘Bannister’.

3. Methodology (of the Students’ Project)

We added a few dense layers and dropout layers into the network, but the network efficiency doesn’t improve much. We found out adding dense layers and dropout layers can only increase the computation amount in each iteration. As a result, the validation accuracy increased faster for the first few epochs. However, the upper bound of top validation accuracy of the whole network doesn’t change and the computation time increases for each epoch. Thus, the network efficiency doesn’t improve much. We also tried other different kinds of learning schedulers and only the triangular 2 cyclical learning rate scheduler callback can improve the system performance and efficiency more obviously.

3.1. Objectives and Technical Challenges

Our objective is to replicate the neural network structure from the original paper, and we want to achieve similar validation accuracy for the system. The first and the biggest challenge we faced is the limited amount of GCP budget. We solved this problem by using the most efficient running method and algorithm. The other problem we faced is using triangle 2 cyclical learning rate callback. At first, we used a Cyclical Learning Rate function in tensorflow_addons and built a learning rate scheduler callback. However, the top validation accuracy is always 2% less than the accuracy from the original paper. We then decided to use CLR in keras-contrib just like what they did in the original paper. But only keras version 2.3.1 supports keras-contrib from github. It took us a while to figure out.

4. Implementation

TensorFlow keras API was used to build and train the Network1 and Network2 models. There are 3 “bottleneck” blocks in the models and each block contains 5 convolution layers(all 3x3 filters) with gradually increasing channels. The ranges of the increasing channel of 3 blocks are slightly different, the first block is from 32 to 512, the second block is from 64 to 1024, the third block is from 32 to 1024. That is because we want different blocks to extract different features and improve the robustness. Each convolution layer in all 3 blocks is followed by a batch normalization layer and a Relu activation layer. The batch normalization layer can constrain the output to follow Gaussian distribution and limit the range, so a more accurate result and faster training process can be achieved. We treat the last layer of each block as the skipped node and concatenate the output of this node to the next block’s skipped node to implement the DenseNet structure. Because the output sizes of the adjacent skipped node are different and can

not be directly concatenated, we reshape a wider and shallow output of skipped node to a half spatial dimension but 4 times deeper layer(more channels) by using TensorFlow space_to_depth function so that each skipped node can be correctly concatenated. After the third block, we also mimic the paper to use a convolution layer with 200 channels followed by a global average pooling layer to replace a fully connected layer to get the probability of 200 classes.

We use keras ImageDataGenerator to generate input flow where we can easily change the image resolution and choose what kind of image augmentation method to use, compared to directly feeding all images at one time. We firstly use reduced resolution images with 32x32 pixels to train the model for 15 epochs followed by original 64x64 pixels images for another 15 epochs. Then we further reduce the resolution to only 16x16 for 15 epochs and finally feed the original images again for 15 epochs.

Next, we feed the augmented images to the model to further improve the robustness and validation accuracy. We used imgaug API as the preprocessing function in the keras ImageDataGenerator to feed the model with augmented images of both the training dataset and validation dataset. We randomly choose the following method: scale, coarse dropout, rotate, additive Gaussian noise and crop & pad to each image.

4.1 Data

We used the same dataset as the paper which is a subset of the ImageNet dataset called Tiny ImageNet dataset. There are 100,000 training examples and 10,000 validation examples with 64x64 pixels resolution and 200 different classes. Many images in our dataset are even difficult for humans to detect objects due to the background domination and low resolution.

4.1 Deep Learning Network

The Tiny ImageNet dataset is used as input data. The structure construction is as instructed in the original paper. And the structures and receptive fields of both network 1 and 2 are as follows:

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 2. Receptive Field of Networks[2]

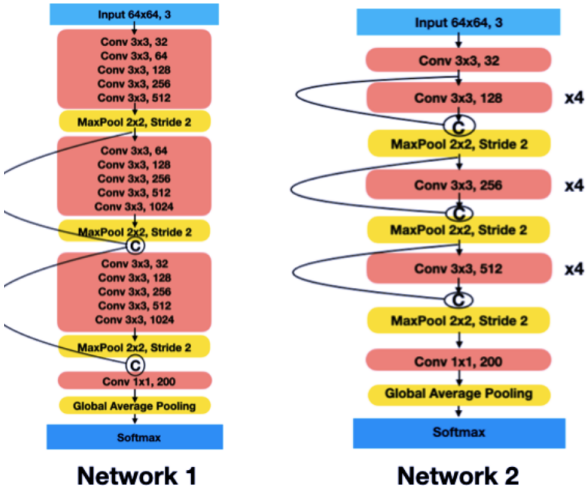

Figure 3. The structure of two custom DenseNet models(Network1 and Network2)[2]

4.2 Software Design

The coding structure for this project is quite simple. All the functions we need to carry out the experiment can be found in the existing modules. We used imguag to do image augmentation. Used Keras to set up and train models and used keras-contrib to call Cyclical Learning rate Callbacks.

Figure 4. The flowchart of programs for Network 1(left) and Network 2(right)

5. Results

5.1 Project Results

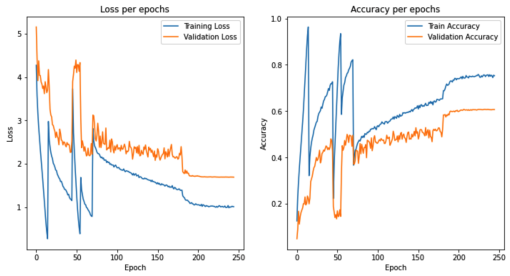

Figure 5. Loss and accuracy of training and validation plot of Network1

For Network1, both loss and accuracy plot experience huge fluctuations before 80 epochs and gradually change after that. At around 180 epochs, there are sudden changes for both loss and accuracy plots and reach a plateau. The sudden improvement of accuracy could be due to the smaller learning rate.

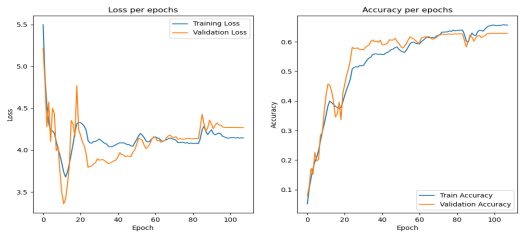

Figure 6. Loss and accuracy of training and validation plot for Network2

For Network2, our validation loss reached a plateau for almost 10 epochs after 70 epochs, and a sudden increase at around 84th epoch, which is due to our change of base learning rate and maximum lr. At the end of 108 epochs, we achieved a validation accuracy of 62.91% whereas training accuracy remained at 65.68%.

Test and validation. We validate the model, and the following Table shows the top-5 worst classified classes by network2 are 'Plunger', 'Pole', 'Umbrella', 'Punching bag' and 'Pop bottle'.

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

	Comparison	Original	Project
Network1	Train accuracy	67%	87.50%
	Validation accuracy	59.5%	60.67%
Network2	Train accuracy	68.11%	65.68%
	Validation accuracy	62.73%	62.91%
Error analysis	Plunger	25%	18.75%
	Umbrella	27.27%	31.82%
	Water Jug	30%	30.43%
	Bucket	30.43%	35.09%
	Wooden Spoon	31.03%	34.88%

5.3 Discussion of Insights Gained

For both Networks, the final results are better than the results mentioned in the original paper. Our validation accuracy is better, especially for Network 1. More than 1% accuracy difference of validation accuracy is definitely not caused by random errors during the computing procedures. My explanation for this phenomenon is that the upgraded versions of tensorflow and keras have better running efficiency. We run the same code structure using the same module and our computation time for each step is much less than theirs. They used Google Colab and we used GCP, thus our computation time must be less. However, our virtual machine performance is not good enough to improve from 2s per step to 793ms per step. The renewed algorithms of the Keras module are possibly responsible for this improvement.

6. Conclusion

Both models, Network1 and Network2, demonstrate the power of DenseNet structure and ResNet structure. We train two models from scratch and get to a satisfactory validation accuracy with 200 epochs. Through the analysis of error, a lot of misclassification cases are explainable and they are even difficult for humans to detect. We achieve our target with our carefully designed network structures, image augmentation, and tricky training sample feeding policy. We also learn how to use a convolution layer and global pooling layer to replace the fully connected layer as the final output layer. I think we can further improve the robustness of the model and avoid overfitting by combining the resolution reduction and image augmentation methods together.

7. References

[1]https://github.com/ecbme4040/e4040-2021fall-project-CDNM-zw2781-dh3027-zy2447

[2] Zoheb Abai, Nishad Rajmalwar,DenseNet Models for Tiny ImageNet Classification, arXiv:1904.10429v2

[3] G. Huang, Z. Liu, L. v. d. Maaten, K. Q.Weinberger, Densely Connected Convolutional Networks, arXiv:1608.06993v5, 2018

8. Appendix

8.1 Individual Student Contributions in Fractions

	UNI1	UNI2	UNI3
Last Name	zw2781	dh3027	zy2447
Fraction of (useful) total contribution	1/3	1/3	1/3
What I did 1	Trained Network2	Trained Network1	Trained network2
What I did 2	finished report	Network1 in the report	Error analysis
What I did 3			