Homework Report

Before thinking about neural net architecture, I researched the dataset for this problem. In this task, we have to construct and fit a neural network for image classification on the Tiny ImageNet dataset. This dataset consists of images with low resolution (3 channels with 64x64 pixels) of 200 different classes. And in the train set, we have 100K images in total and 10K images in the validation set [1].

This problem makes sense to be like ImageNet that was solved quite effectively with popular algorithms (AlexNet, VGG, ResNet, GoogleNet, etc), so I found it reasonable to use some of these architectures. So, ResNet18 [2] was chosen. I like this kind of architecture because it's invariant to image size, and I can train it on my pictures as it is, without any changes. Before training, I decided to use data augmentation to get more data "by free" and prevent overfitting.

After researching the pictures, I decided to use the following transforms because they change data not very sufficiently – RandomRotation, RandomHorizontalFlip, RandomAffine. As for the optimizer, I chose Adam with standard configuration ($lr = 1 \cdot 10^{-3}$, $beta1 = 0.9$, $beta2 = 0.99$) because it is an easy way to have some quite good results without tuning the optimizer's parameters. As a loss function, I've chosen CrossEntropyLoss. During the training procedure, the wandb library was used to observe results.

As a result (figure 1, purple curve), Resnet18 achieved maximum validation accuracy of 37.7% on the 36th epoch, and with the next epochs, validation accuracy started to decrease while train loss continued decreasing fast. It's a common symptom of overfitting and I decided to use L2 regularization to prevent it. At first, the weight decay was chosen as $1 \cdot 10^{-3}$, but it turned out that it affects huge penalize of weights and as a result, maximal validation accuracy was decreased to 10%, and also train loss stopped decreasing. (figure 1, orange curve) Then weight decay was chosen to be $1 \cdot 10^{-4}$ and it helped to prevent overfitting and an accuracy of 39% was achieved. But this result was quite far from expected, I also found some papers [3] with residuals nets and there was image size adaptation applied by bicubic interpolation to small images in order to run them on original ResNets, that were used for ImageNet problem, but I couldn't use this approach because of the restriction in our task. That's why I decided to use another architecture.

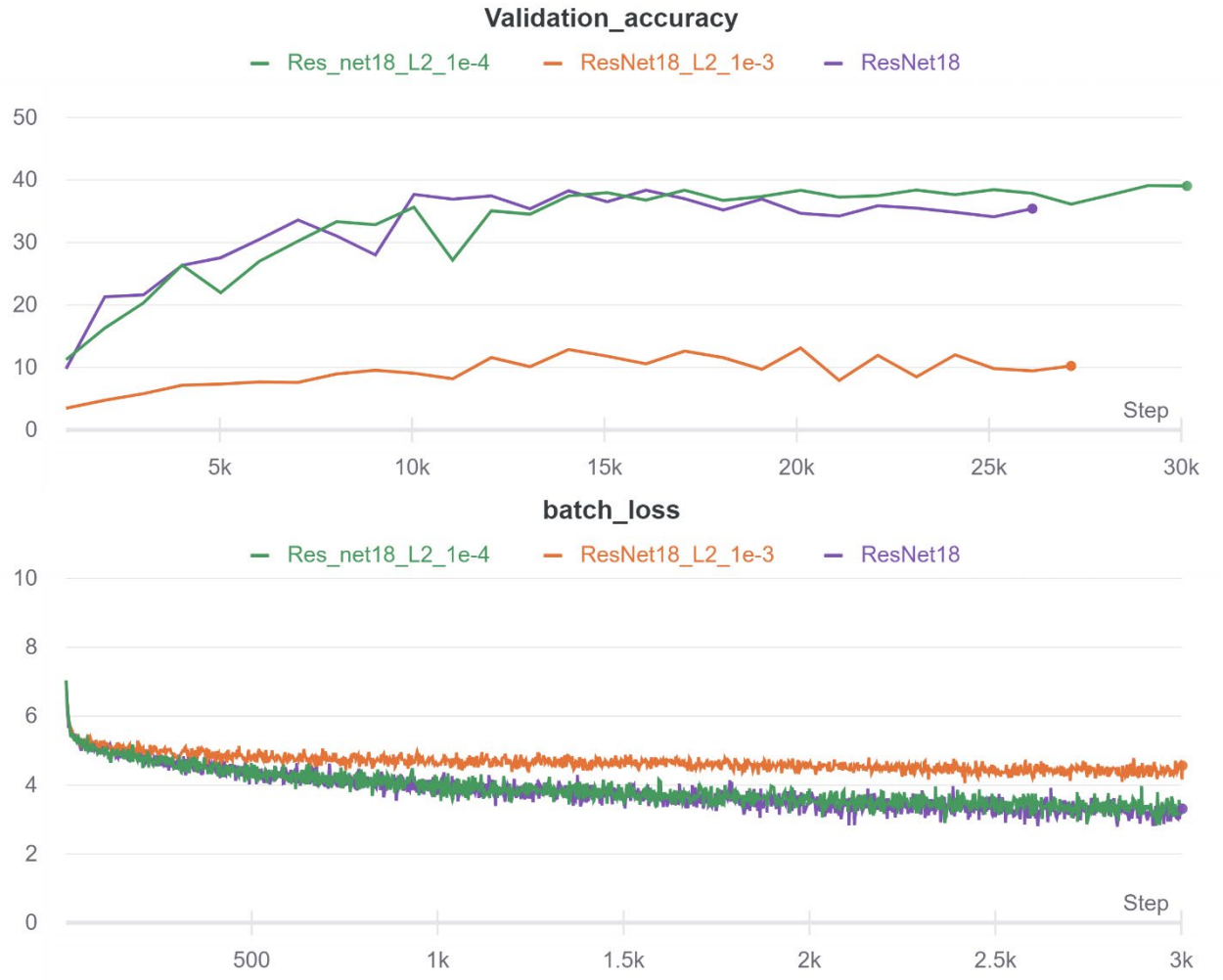

Figure 1 Graphs of train loss and validation accuracy for ResNet architectures.

Then I created my network inspired by VGG [4]. I used the following architecture.

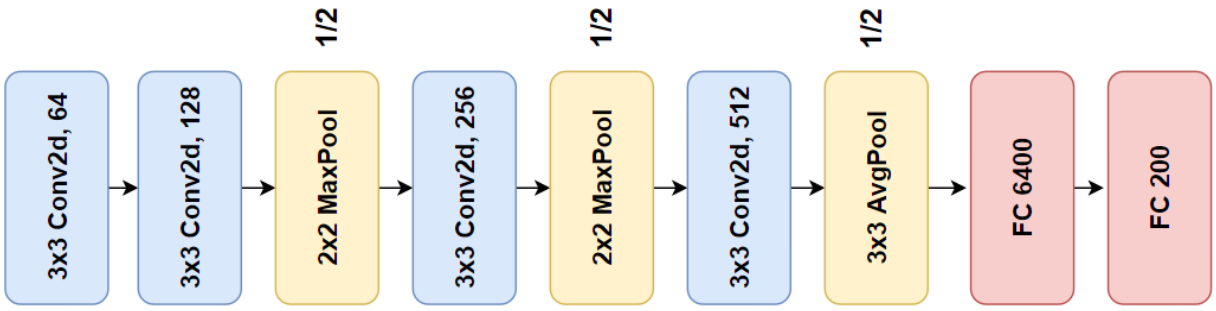

Figure 2 My VGG architecture. Conv blocks contains also the ReLU activation function.

I also used the same optimizer as in the previous attempt and also L2 regularizations, because it showed good performance.

As a result, it performed worse than previous models. It achieved 32.9% validation accuracy (figure 3, purple curve). Then I decided to add Batch normalization before all ReLU activation functions. And it helped to increase the result and it was higher than even for ResNet18 – 40.67% (figure 3, green curve).

Figure 3 Graphs of train loss and validation accuracy for VGG architectures

Reading the literature [5,6] I found an interesting improvement of ResNet – DenseNet. It uses not the summation of residuals but concatenating. I decided to try this approach as the idea of concatenating outputs of residual blocks along channels seemed to me much reasonable than the summation because we are not blending the features from different kernels.

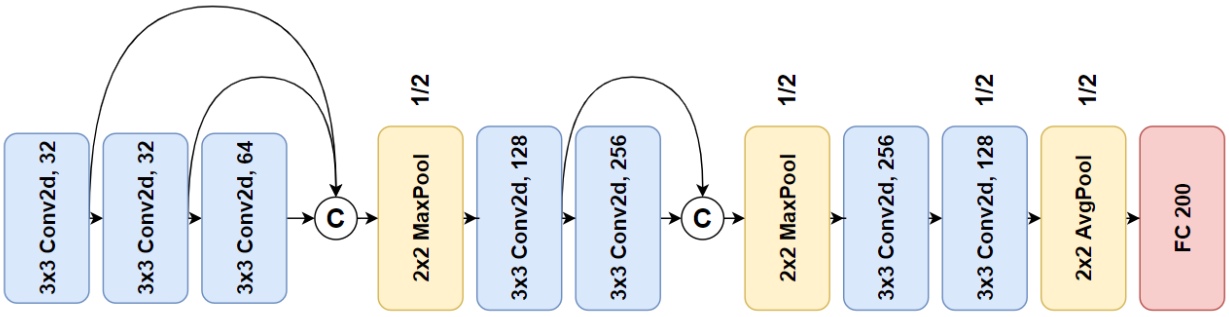

Figure 4 My DenseNet architecture. All conv blocks contain ReLU activation function and BatchNormalization

Also, I find out that all the nets during the training procedure achieve some plateau of validation accuracy and during all next epochs it fluctuated near this level.

This optimization challenge I think can be solved by tuning the optimizer. I would suggest that SGD with momentum will show a better result with some parameters but it takes a lot of time to find optimal parameters. Therefore, I decided to use Scheduler to decrease the learning rate after achieving this plateau.

In my DenseNet, I used these approaches with L2 regularization and batch Normalization. As a result, this architecture beats all previous ones. It achieved 49.65% validation accuracy (figure 5, gray curve).

Also, I tried to improve this net by using additional augmentation transforms. I used color Jitter to make the net invariant to colors and Gaussian blur because the pictures look very sharp to my mind and I thought it would help. But it turned out that Gaussian Blur effects as a kind of dropout (we lose information) and the result was badly damaged (figure 5, blue curve). Also, I didn't notice any improvements from color Jitter, that's because some images can be recognized just by colors. For example, most pictures with frogs had a lot of green color.

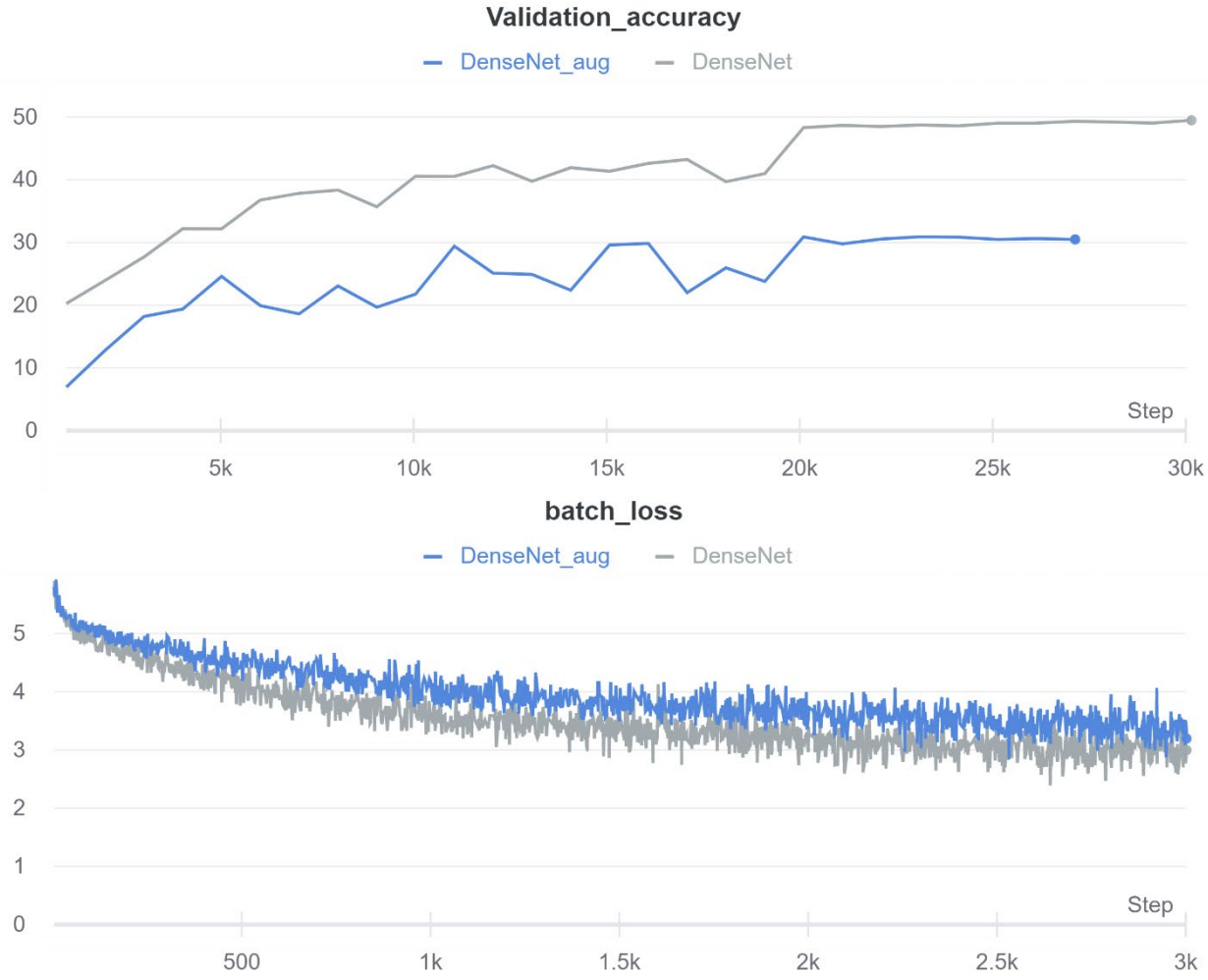

Figure 5 Graphs of train loss and validation accuracy for DenseNet architectures.

Figure 6 Best models from each kind of architectures. (For loss curve smoothness was applied)

References

[1] A. Banerjee, V. Iyer. Tiny Imagenet Challenge. CS231N Project Report - cs231n.stanford.edu, 2015.

[2] K. He, X. Zhang, S. Ren, J. Sun. Deep Residual Learning for Image Recognition/ IEEE Conference on Computer Vision and Pattern Recognition (CVPR). arXiv:1512.03385v1, 2016.

[3] H. Kim. Residual Networks for Tiny ImageNet. CS-231N Final Project Report - cs231n.stanford.edu, 2016.

[4] K. Simonyan, A. Zisserman. Very deep convolutional networks for large-scale image recognition. arXiv:1409.1556, 2015.

[5] Z. Abai, N. Rajmalwar. DenseNet Models for Tiny ImageNet Classification. arXiv:1904.10429, 2019.

[6] G. Huang, Z. Liu, L. Maaten. Densely Connected Convolutional Networks. arXiv:1608.06993v5, 2018.