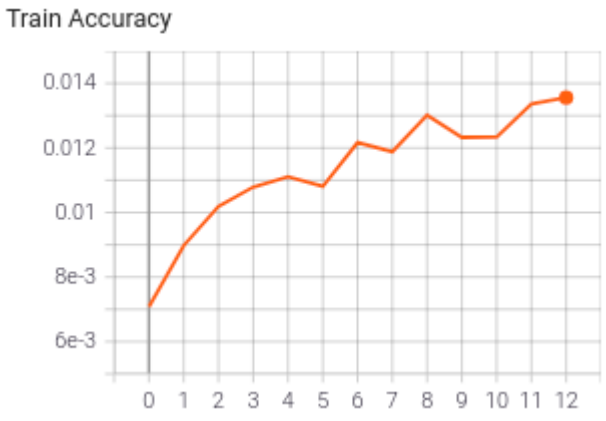
As we solve the task for ImageNet data, it seems to be logical to search models that have already faced this problem. There are a lot of models in *torchvision.models* module and their performance on ImageNet data.

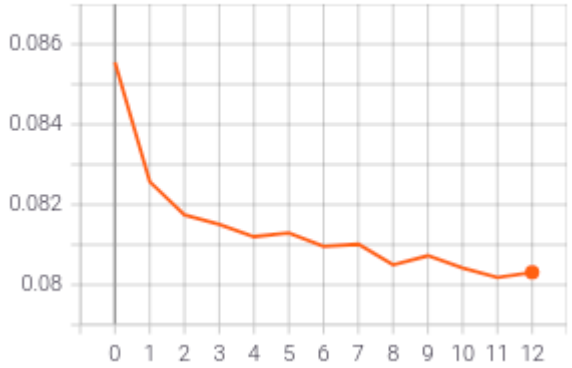
I started with VGG-19 with BatchNorm that gives high performance (according to *torchvision.models* module page) and has the simplest structure, comparing to other high-performance models.

Model	Acc@1	Acc@5
AlexNet	56.522	79.066
VGG-11	69.020	88.628
VGG-13	69.928	89.246
VGG-16	71.592	90.382
VGG-19	72.376	90.876
VGG-11 with batch normalization	70.370	89.810
VGG-13 with batch normalization	71.586	90.374
VGG-16 with batch normalization	73.360	91.516
VGG-19 with batch normalization	74.218	91.842

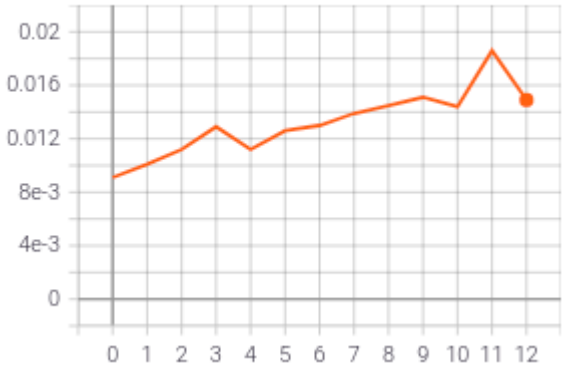
However, as can be seen from tensorboard charts below, both train and validation accuracy increased, but it seems to be trained for a long time. That's why I decided to move on to more complex models.



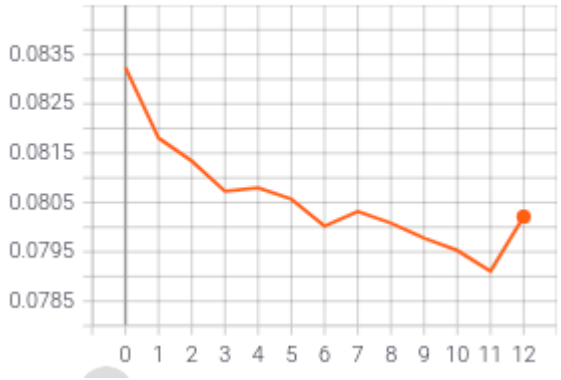
Train Loss



Validation Accuracy



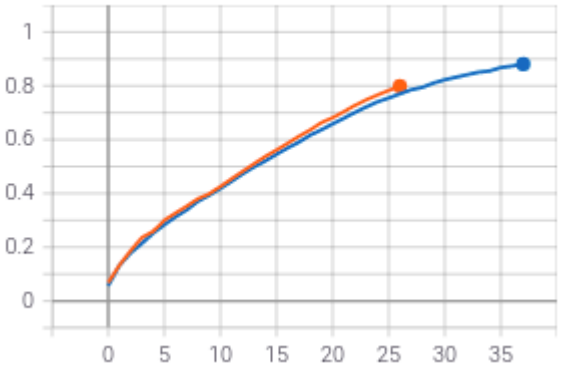
Validation Loss



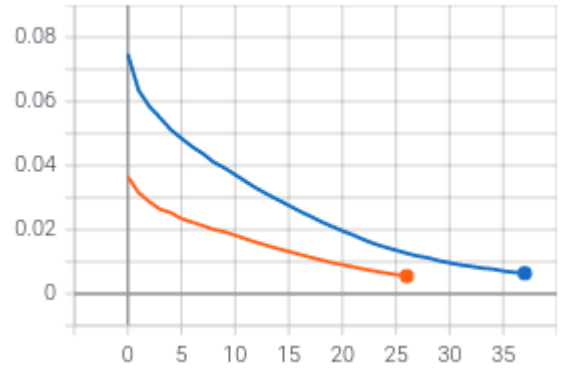
Then I tried to use ResNet-152, however the one epoch lasted approximately 15 minutes. That's rather bigger than I expected to see. After this, DenseNet-161 was selected for testing their performance with the default experiment parameters (SGD optimizer with learning rate=0.1 and momentum=0.9, batch size=64, number of workers=2).

It showed good and quick (comparing to ResNet) performance. But validation accuracy achieved a peak at 0.38. There are tensorboard charts of loss and accuracy below. The blue line batch size is equal to 64, the orange line batch size is equal to 128.

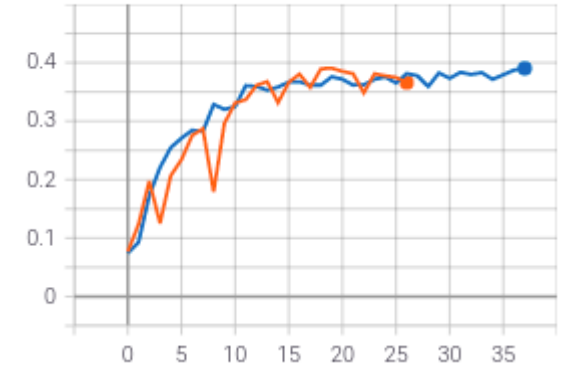
Train Accuracy



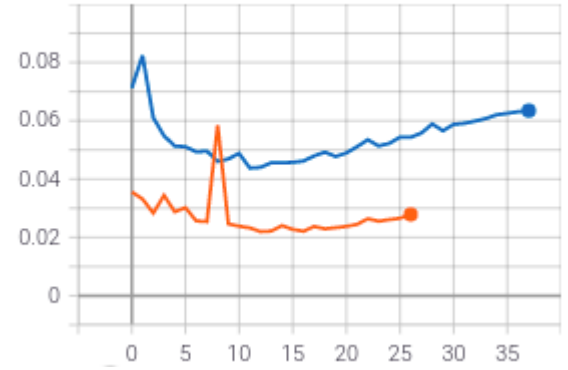
Train Loss



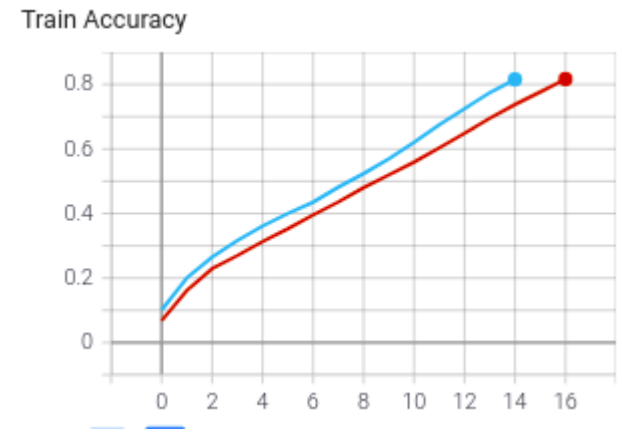
Validation Accuracy

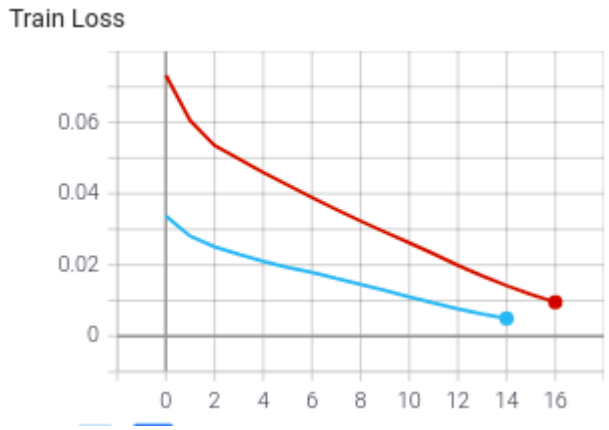


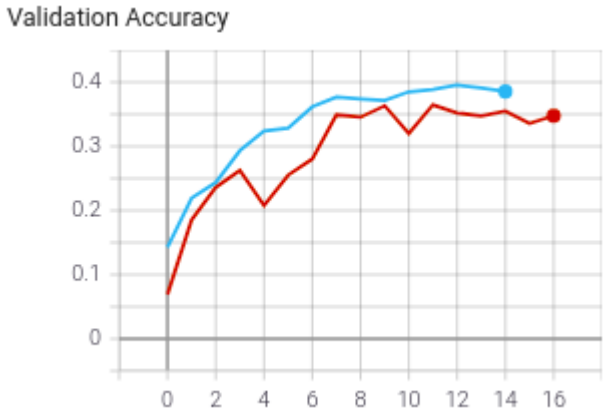
Validation Loss



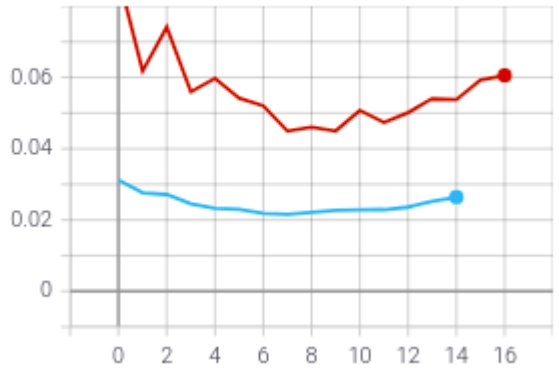
Then there were some experiments with DenseNet-161 with transforms, optimizer and batch size. The blue line includes SGD with learning rate 0.1, batch size 64 and the following set of transforms (random rotation, random vertical flip, random perspective). The red line includes Adam with learning rate 0.001, batch size 128 and the following set of transforms (random affine, random rotation, random horizontal flip).





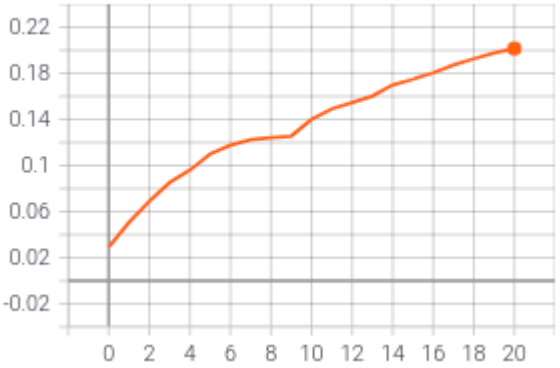


Validation Loss

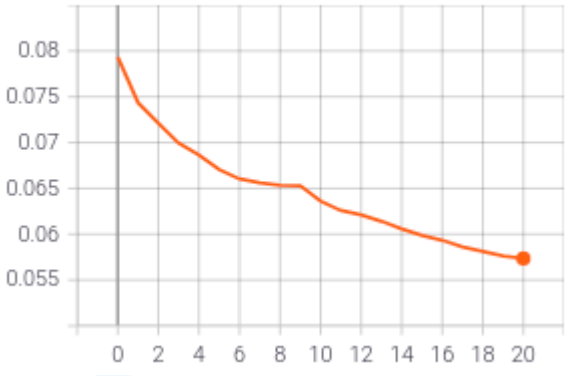


Also I tried to use Network 1 (DenseNet) transforms from the paper. However, it was a bad attempt because of slow training.

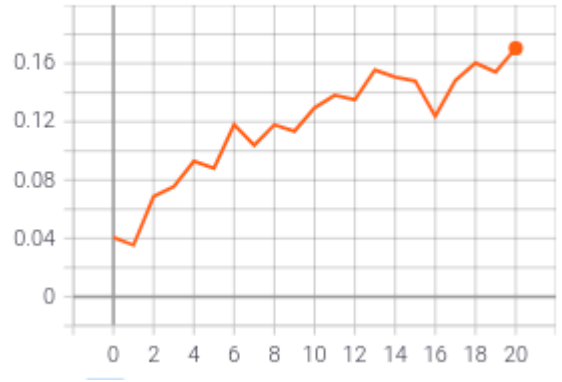
Train Accuracy



Train Loss



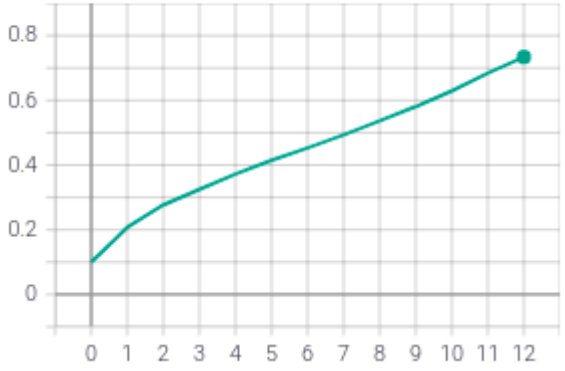
Validation Accuracy



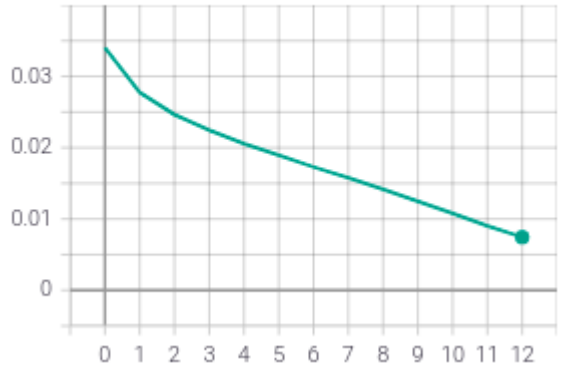
Validation Loss

The final model that I selected was DenseNet-201: Adam optimizer (learning rate 1e-3), batch size 128, the following set of transforms (RandomAffine(15), RandomRotation(10), RandomHorizontalFlip), the following critetrion (CrossEntropyLoss).

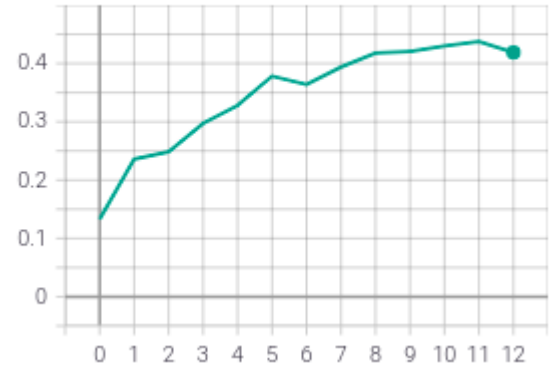
Train Accuracy



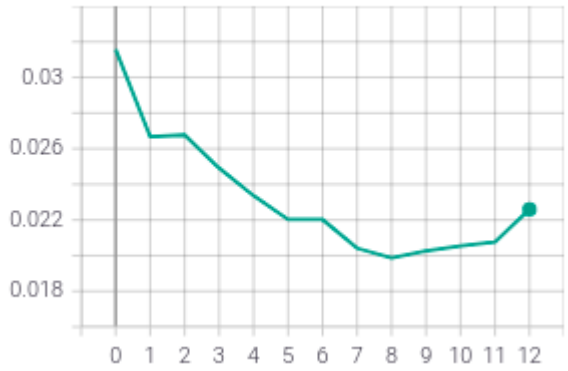
Train Loss



Validation Accuracy



Validation Loss



For the current checkpoint the epoch 11 was selected with train accuracy 0.6845 and validation accuracy 0.4375. Then I tried to re-train model with the lower learning rate, but deadline time is over.

As for optimizers, after multiple experiments, Adam seems to be a more universal tool than SGD. It was easy to use Adam with default parameters, which are recommended in the lecture, and get good accuracy.

As for batch size, I preferred to use 64 or 128. I tried different values, but 64 and 128 gave a better result.

As for the number of iterations, it would be more convenient to use early stopping, however, I tried to do it “by eye”. That’s not best practice, of course, but it was sometimes difficult to manage all experiment setups and in some cases “early stopping by eye” was the only way.

The main trick that I gained after this task is how we can re-train model with different setup. I’ve never done this before, so it’s a really useful experience for me.

Sources of code:

1. Training loop structure. https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html
2. Torchvision models. https://pytorch.org/vision/stable/models.html
3. Torchvision transforms. https://pytorch.org/vision/stable/transforms.html