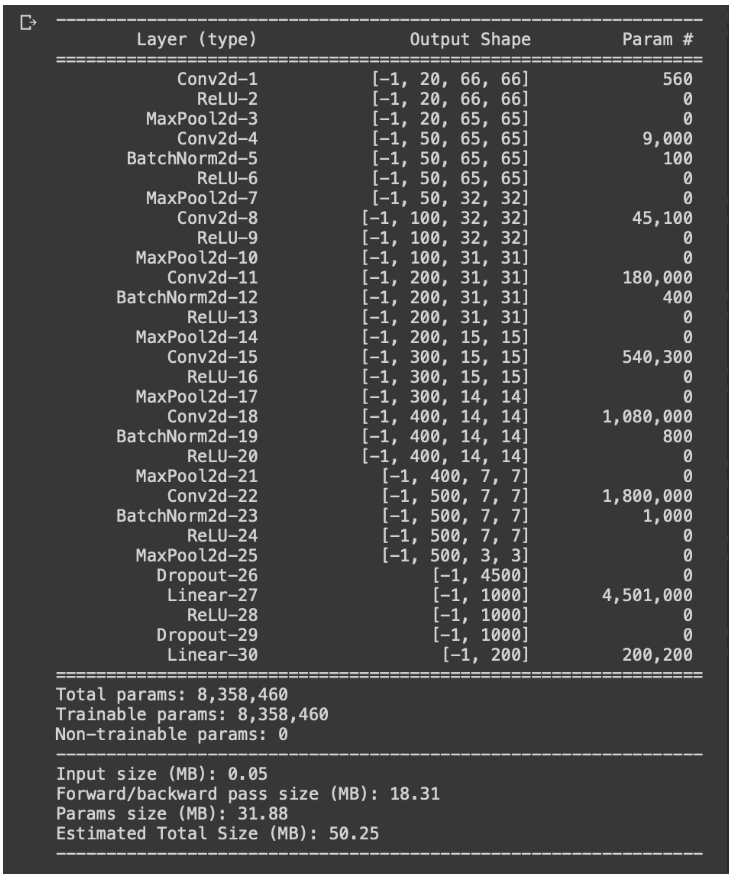
My history of tweaks and improvements

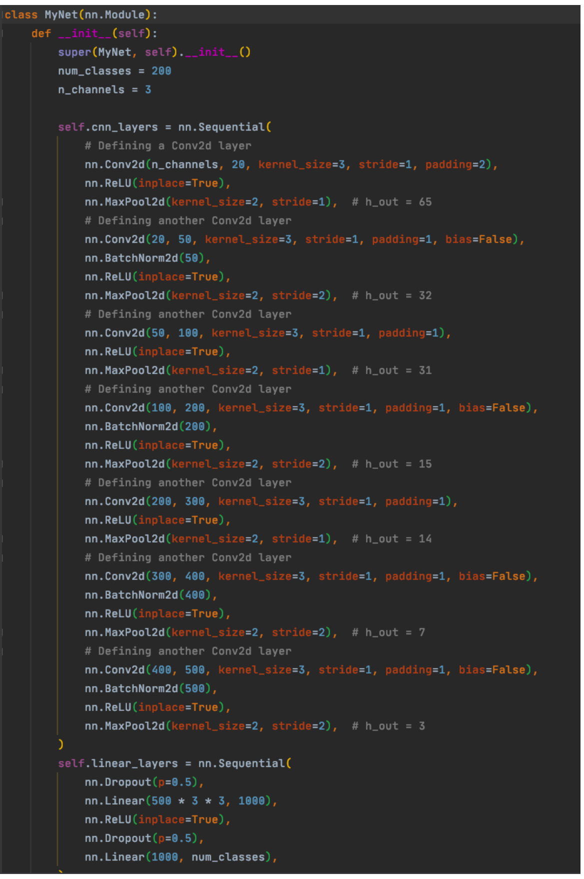
I decided to check that my code works on a small ConvNet with 3 convolutional layers (+BatchNorms, MaxPool, Dropout, ReLU). As data transform for train set I took RandomVerticalFlip, and Normalization. I got 30.210 % accuracy on the validation set after 20 epochs. Not too bad, I thought, and decided to increase the learning rate from 1e-4 to 1e-3, and got 35.17 on after 60 epochs)).

Then I decided to extend the number of convolutional layers and made 7 layers with the following architecture:



```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 20, 66, 66]             560
              ReLU-2           [-1, 20, 66, 66]               0
         MaxPool2d-3           [-1, 20, 65, 65]               0
            Conv2d-4           [-1, 50, 65, 65]           9,000
       BatchNorm2d-5           [-1, 50, 65, 65]             100
              ReLU-6           [-1, 50, 65, 65]               0
         MaxPool2d-7           [-1, 50, 32, 32]               0
            Conv2d-8          [-1, 100, 32, 32]          45,100
              ReLU-9          [-1, 100, 32, 32]               0
        MaxPool2d-10          [-1, 100, 31, 31]               0
           Conv2d-11          [-1, 200, 31, 31]         180,000
      BatchNorm2d-12          [-1, 200, 31, 31]             400
             ReLU-13          [-1, 200, 31, 31]               0
        MaxPool2d-14          [-1, 200, 15, 15]               0
           Conv2d-15          [-1, 300, 15, 15]         540,300
             ReLU-16          [-1, 300, 15, 15]               0
        MaxPool2d-17          [-1, 300, 14, 14]               0
           Conv2d-18          [-1, 400, 14, 14]       1,080,000
      BatchNorm2d-19          [-1, 400, 14, 14]             800
             ReLU-20          [-1, 400, 14, 14]               0
        MaxPool2d-21            [-1, 400, 7, 7]               0
           Conv2d-22            [-1, 500, 7, 7]       1,800,000
      BatchNorm2d-23            [-1, 500, 7, 7]           1,000
             ReLU-24            [-1, 500, 7, 7]               0
        MaxPool2d-25            [-1, 500, 3, 3]               0
          Dropout-26                 [-1, 4500]               0
           Linear-27                 [-1, 1000]       4,501,000
             ReLU-28                 [-1, 1000]               0
          Dropout-29                 [-1, 1000]               0
           Linear-30                  [-1, 200]         200,200
================================================================
Total params: 8,358,460
Trainable params: 8,358,460
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.05
Forward/backward pass size (MB): 18.31
Params size (MB): 31.88
Estimated Total Size (MB): 50.25
----------------------------------------------------------------
```

Summary for the network (using torchsummary library)



```python
class MyNet(nn.Module):
    def __init__(self):
        super(MyNet, self).__init__()
        num_classes = 200
        n_channels = 3

        self.cnn_layers = nn.Sequential(
            # Defining a Conv2d layer
            nn.Conv2d(n_channels, 20, kernel_size=3, stride=1, padding=2),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=1),  # h_out = 65
            # Defining another Conv2d layer
            nn.Conv2d(20, 50, kernel_size=3, stride=1, padding=1, bias=False),
            nn.BatchNorm2d(50),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),  # h_out = 32
            # Defining another Conv2d layer
            nn.Conv2d(50, 100, kernel_size=3, stride=1, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=1),  # h_out = 31
            # Defining another Conv2d layer
            nn.Conv2d(100, 200, kernel_size=3, stride=1, padding=1, bias=False),
            nn.BatchNorm2d(200),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),  # h_out = 15
            # Defining another Conv2d layer
            nn.Conv2d(200, 300, kernel_size=3, stride=1, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=1),  # h_out = 14
            # Defining another Conv2d layer
            nn.Conv2d(300, 400, kernel_size=3, stride=1, padding=1, bias=False),
            nn.BatchNorm2d(400),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),  # h_out = 7
            # Defining another Conv2d layer
            nn.Conv2d(400, 500, kernel_size=3, stride=1, padding=1, bias=False),
            nn.BatchNorm2d(500),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),  # h_out = 3
        )
        self.linear_layers = nn.Sequential(
            nn.Dropout(p=0.5),
            nn.Linear(500 * 3 * 3, 1000),
            nn.ReLU(inplace=True),
            nn.Dropout(p=0.5),
            nn.Linear(1000, num_classes),
```

Convolutional and linear layers in the network

This convolutional network gave 47.15 % accuracy.

After that I tried a bunch of other improvements, but they did not beat this score. These improvements were:

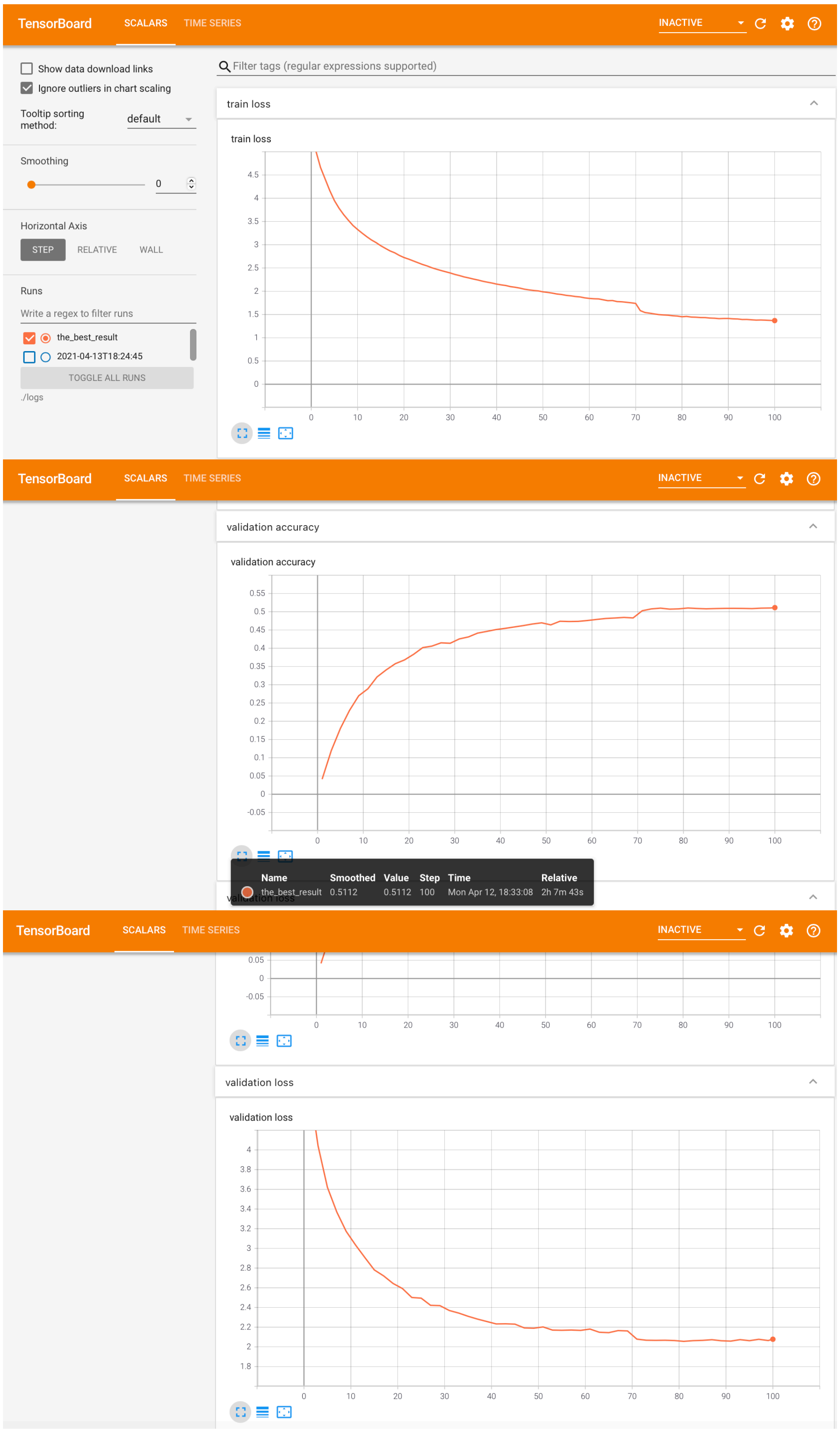
- Adding 8th convolutional layer - approx. 10 % drop in accuracy (perhaps, it was too much parameters for this dataset)
- Adding another one linear layer - drop in accuracy (probably the same problem)
- Replacing one linear layer with average pooling - this one I heard on the lecture - approx. 0.004 % drop in accuracy
- Adding data augmentations (ColorJitter, GaussianBlur, RandomRotation, RandomErasing) - these augmentations gave me lower accuracy. I tried to apply them all and only some of them, and it seems like RandomRotation gave this drop in accuracy. Perhaps, with these augmentations, the neural network will take longer to learn, but it will not ovefit, however I didn't want to wait more than 120 epochs.

In the end, I decided to go back to the previous good architecture and a small list of augmentations (RandomVerticalFlip, RandomHorizontalFlip, Normalization). I trained this network for 100 epochs and saw that in the last 30 epochs the loss of validation goes neither up nor down. I added a scheduler that reduces the learning rate by a factor of 10 in the 70th epoch and then got 51.12 %. Hooray))

Finally, there are my insights:

- In any unclear situation, try changing the learning rate first)
- Making your network deeper does not always mean that it will perform better than a smaller one.
- Average pooling at the end of the convolutional layer is not bad idea, I'll try to experiment more with it next time.

Screenshots from TensorBoard:



☐ Show data download links
☑ Ignore outliers in chart scaling
Tooltip sorting method: default
Smoothing 0
Horizontal Axis: STEP
Runs
Write a regex to filter runs
☑ the_best_result
☐ 2021-04-13T18:24:45
./logs

Name	Smoothed	Value	Step	Time	Relative
the_best_result	0.5112	0.5112	100	Mon Apr 12, 18:33:08	2h 7m 43s