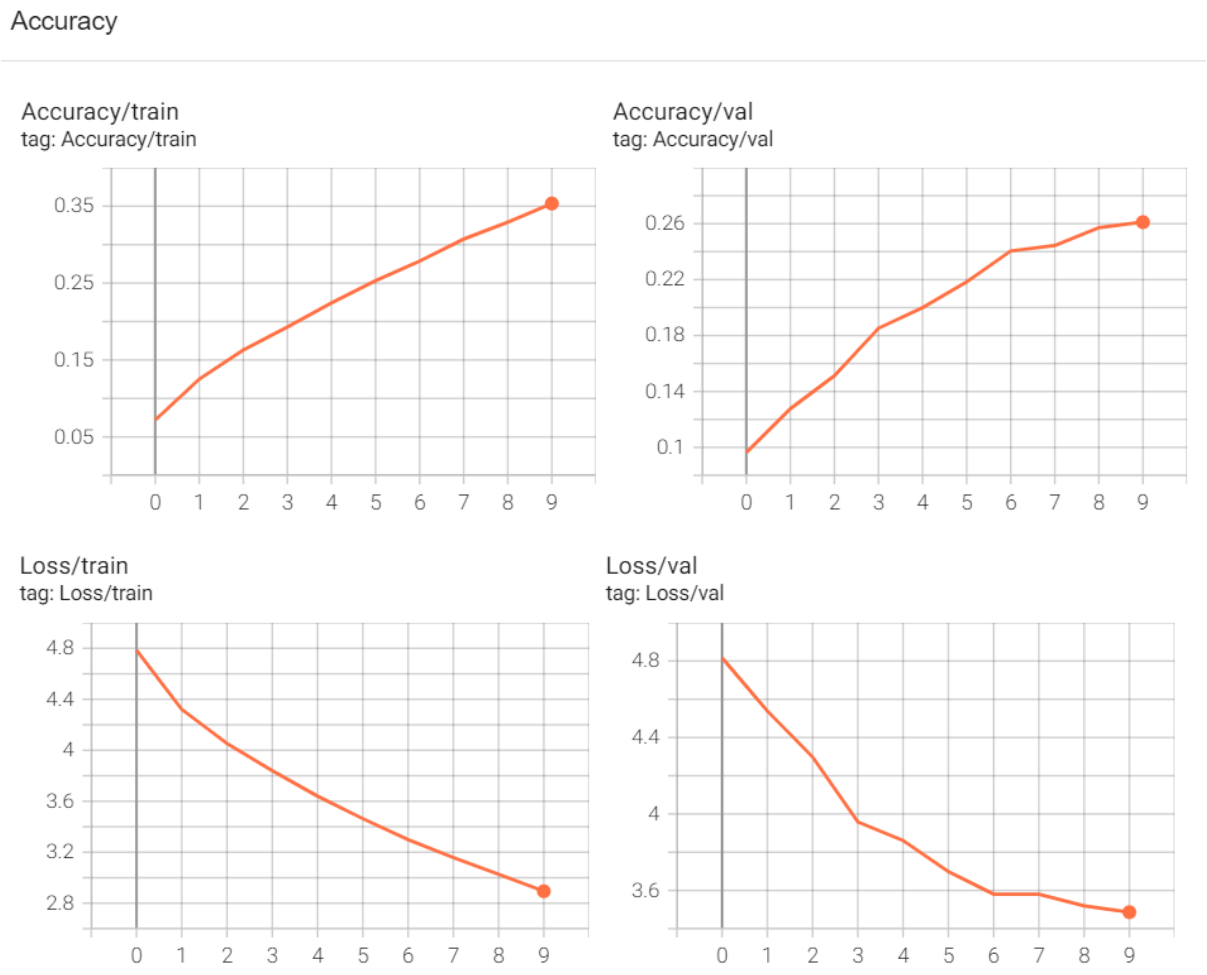
Deep Learning course. Homework 1. Report

VladislavTrifonov

In the begging, I would like to mention that I took some parts of my own code from my home tasks of a course “Machine Learning” from term 3. There studetns were provided with the certain templates (so, this template may be coincide with other students). Also I might borrow some examples from torch tutorials.

I believe, I came out with my initiall net inspired by VGG. I read some other papers, but from them I just borrowed at least some basic intuition how to stack layers.

The very first run.

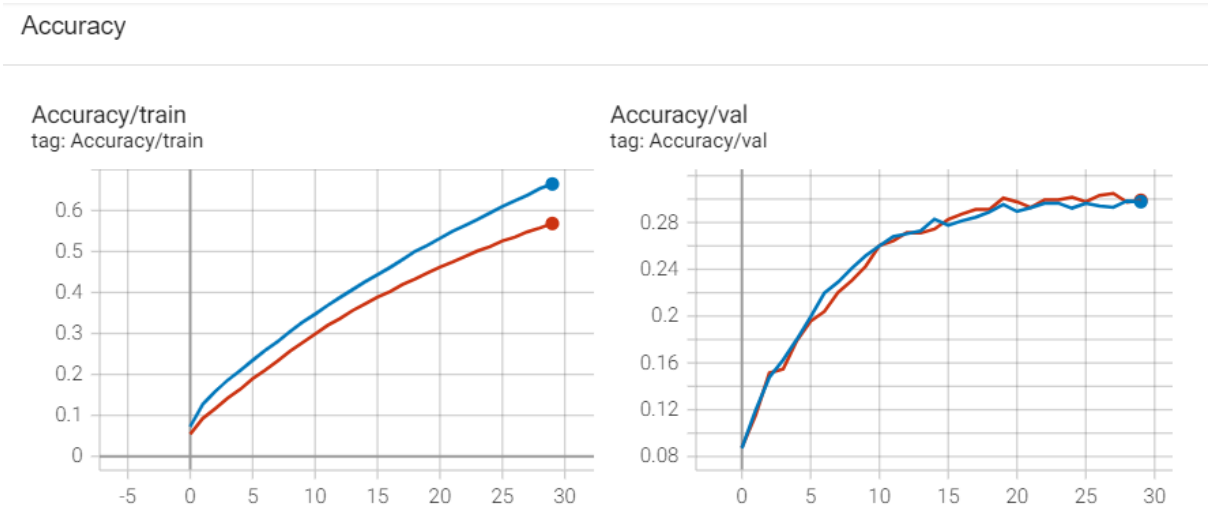


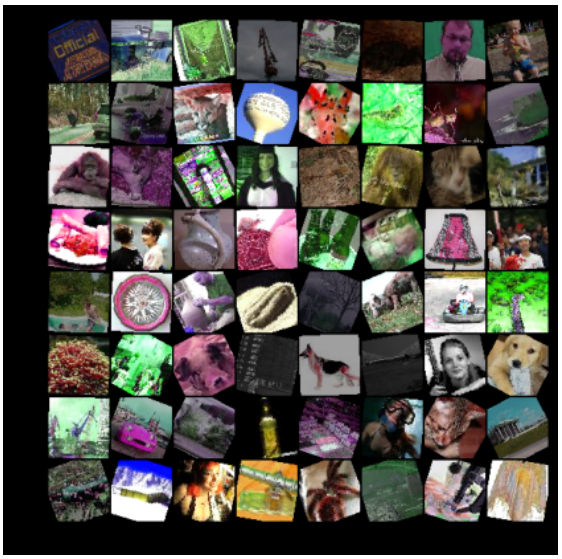
```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 32, 30, 30]           2,432
       BatchNorm2d-2           [-1, 32, 30, 30]              64
            Conv2d-3           [-1, 32, 28, 28]          25,632
              ReLU-4           [-1, 32, 28, 28]               0
         MaxPool2d-5           [-1, 32, 14, 14]               0
       BatchNorm2d-6           [-1, 32, 14, 14]              64
            Conv2d-7           [-1, 96, 14, 14]          27,744
       BatchNorm2d-8           [-1, 96, 14, 14]             192
            Conv2d-9          [-1, 256, 14, 14]         221,440
          Dropout-10          [-1, 256, 14, 14]               0
      BatchNorm2d-11          [-1, 256, 14, 14]             512
           Conv2d-12          [-1, 384, 14, 14]         885,120
             ReLU-13          [-1, 384, 14, 14]               0
        MaxPool2d-14            [-1, 384, 7, 7]               0
      BatchNorm2d-15            [-1, 384, 7, 7]             768
          Flatten-16                [-1, 18816]               0
          Dropout-17                [-1, 18816]               0
           Linear-18                   [-1, 20]         376,340
             ReLU-19                   [-1, 20]               0
      BatchNorm1d-20                   [-1, 20]              40
           Linear-21                   [-1, 10]             210
================================================================
Total params: 1,540,558
Trainable params: 1,540,558
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 4.08
Params size (MB): 5.88
Estimated Total Size (MB): 9.97
----------------------------------------------------------------
```

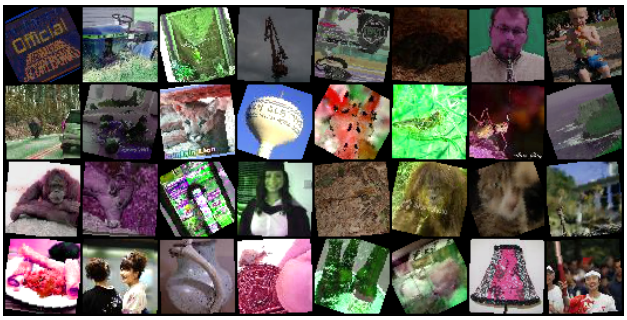
I tried a lot of things actually. In the very begining I did too much of augmentation and my NN did not train at all. Also I used small batches (firstly 8, than 20, 40, 50 and achived good enough accuracy at very first time with batch 60 I guess). Than I read, that some data scientist believe thats batch size has to be a power of two. That explain for me the numbers.

As you can see from my optimizer, I used weight_decay, which did not help in the begging (but at this moment I was with bad augmentation and small batches. So, I did not know what to start with). I would like to mention that it’s still confusing for me how to choose augmentation. I understand, that it totally depends of the dataset at hand, but torchvixion.transforms.ColorJitter not the first time confusing me.

Results with first tries of augmentation.







That’s this augmentation

I belive that at this moment most of all helped these things: 1. decrease of augmentation; 2. stop of messing with hyperparameters of optimizer; 3. increasing a batch size.

Till the every end I wanted to implement my own CNN, but not just use one from torchs' zoo – ugh, thats tough.

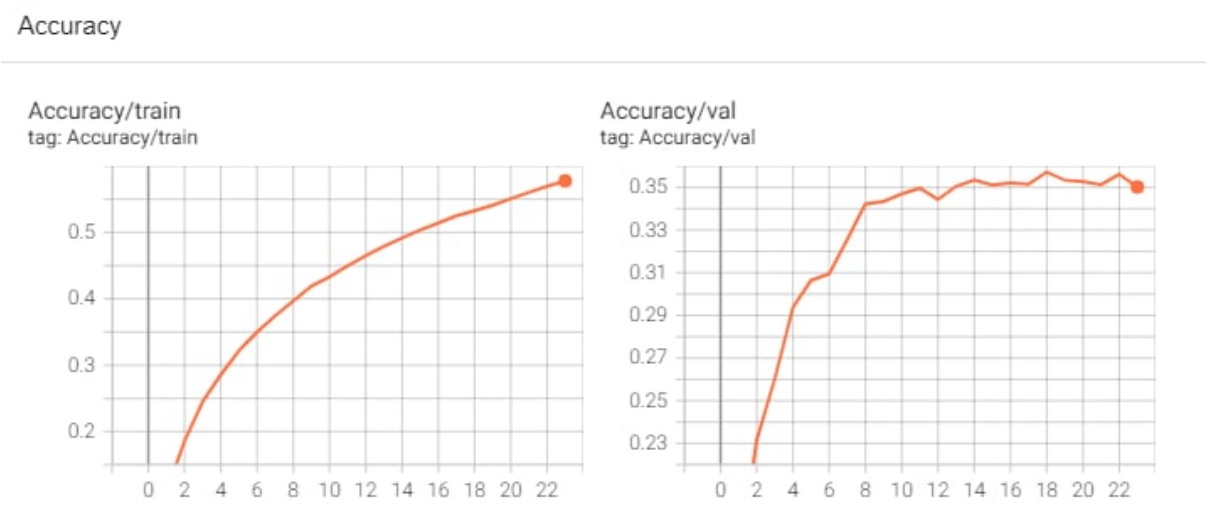
I also trained in parallel some ResNets. I tried resnet101 and resnet50 – both of them were too complex for free colab resources. Resnet34 worked for me, but my implementation at this moment started to perform better with all the other parameters kept the same. (For now I can't manage parallel tuning of parameters, so both nets had the same besides the architecture).

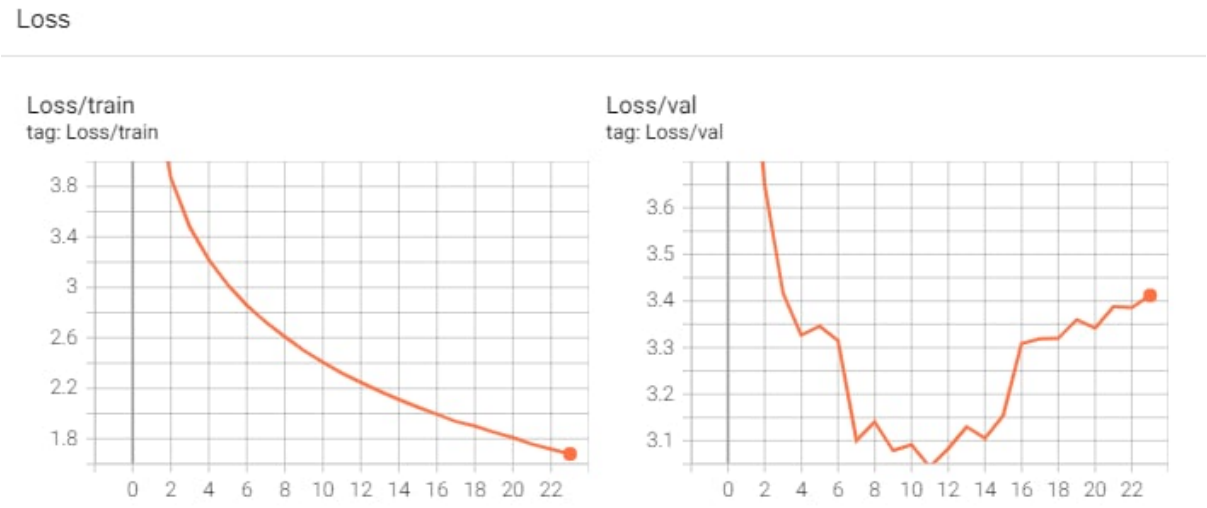
I played with position of regualrization in architecture. Most of all there helped me proper usage of BatchNormalization (just before the layer). And of course tuning of Droput's (firstly I dropped to much weights and did it too often).

At one point helped to increase a number of channels up to 256\384 (depend on architecture). And I liked the way with the separated so-to-say "blocks" of convolutions between which there are Poolings.

I also messed a little with classification block. Not sure what is better: only one hidden layer or couple of them. From my side it depends on the amount of Pooilngs. More poolings - more hidden linear layers and vice versa. In the end I had 3 fully-connected layers.

Increased number of fully-connected layers and played with prob in dropout.

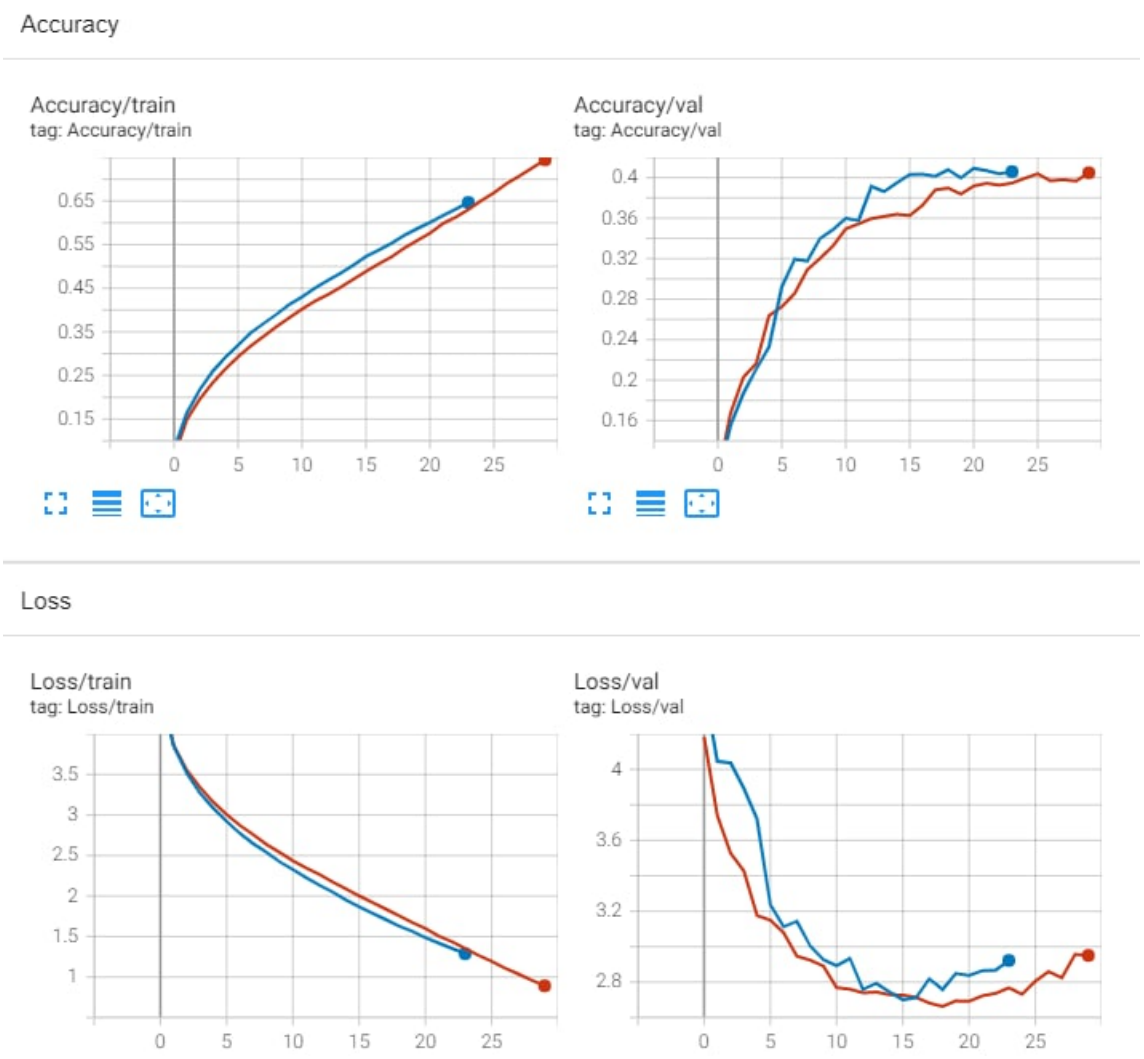




I tried these batch sizes: {64, 128, 256, 512}. It was surprising that higher batch size does not mean higher accuracy. 256 is my choice.

When I stuck on 39 accuracy on val I decided to increase number of conv “blocks”. That made things worse. What really helped is to decreasing sizes of tesnors after convolution with strides in MaxPooling. Also more higher value for 2 dropouts in fully-connceted layers help. But with if the prob is higher than 0.3 – than I just overegularized and stopped train.

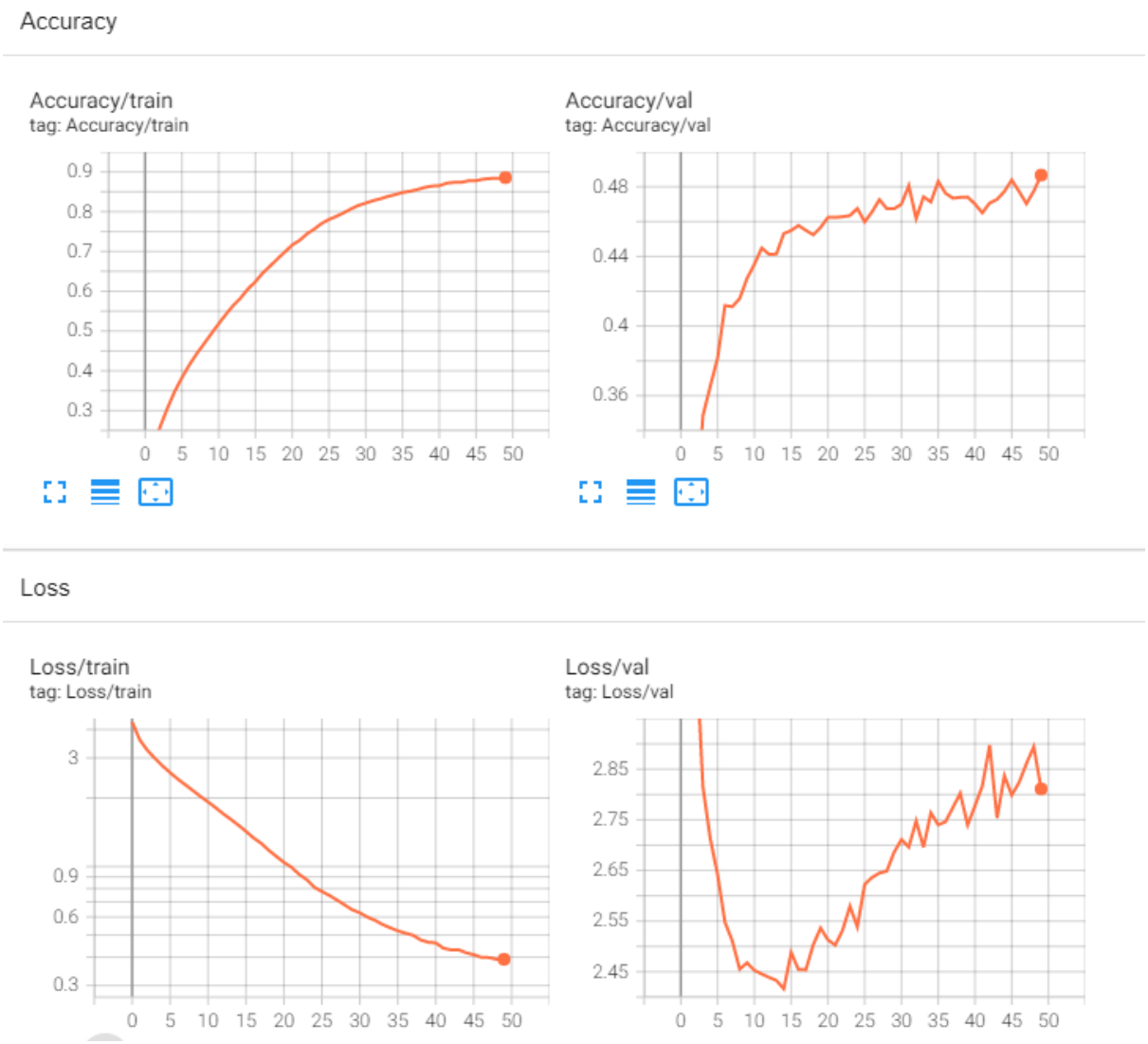
Stuck below 0.4 val acc



I used both ReLU and LeakyReLUm since I read on towardsdatascience article, that 0 for negative inputs to ReLU speed-ups a net which is significant for multi-layer networks. But as far as I understood learning is not happening in the are of 0 in ReLU (zero gradient) - this is its main disadvantage. And LeakyReLU is introduced to solve this problem. However, I used ReLU in the last architecture, because experiments showed it to be better choice.

Run before the last run: I changed two Dropouts in Conv blocks to BatchNorms and it allowed me to achieve val score 0.48

Run with 0.48 val accuracy

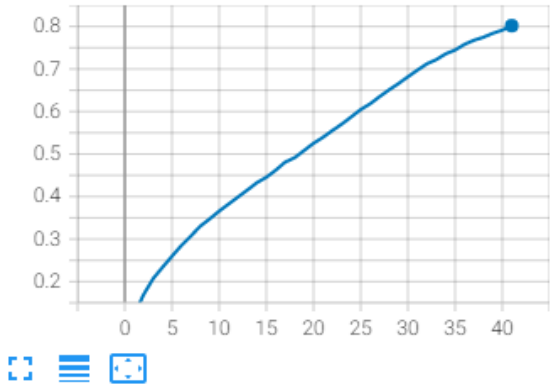


In the end I had time to conduct the very last experiment. Here I replace order of BathcNorms and MaxPoolings. The score is not higher than 0.48, but it rather more stable. So, if I had more time may be I cloud add regularization a bit and achieve score for val > 0.50.
So, the final difference between the last and one beofre it just 1% but the last run is really way more stable. I will choose the run before the last one, because it accuracy higher than 0.46 for the half of the training.

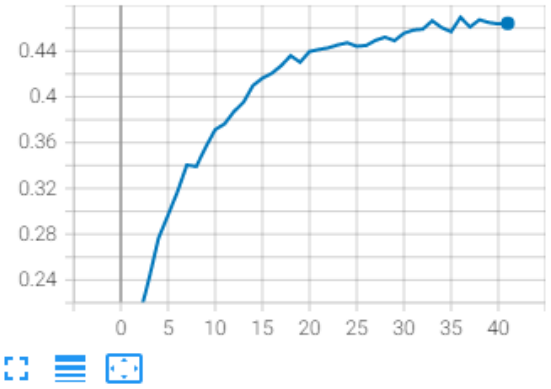
The very last experiment.

Accuracy

Accuracy/train
tag: Accuracy/train

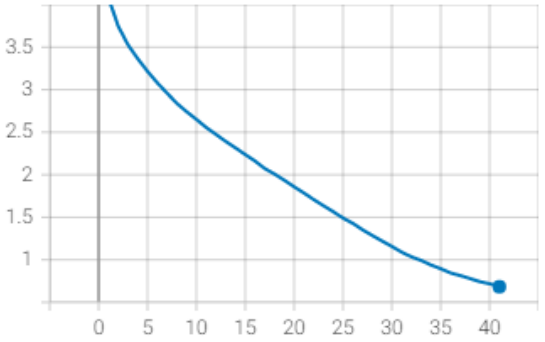


Accuracy/val
tag: Accuracy/val



Loss

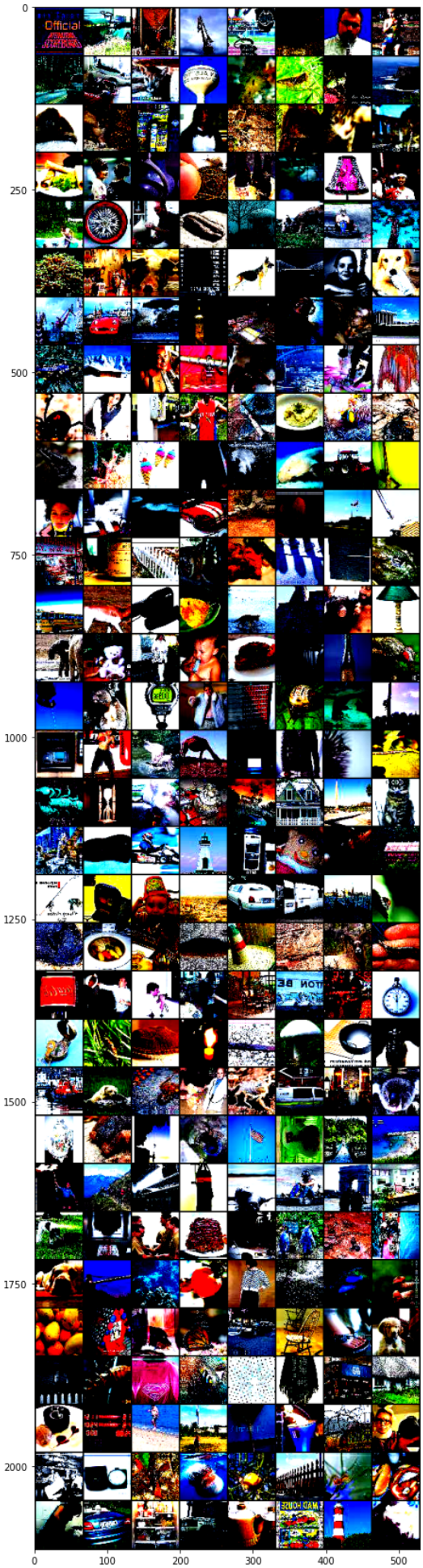
Loss/train
tag: Loss/train



Loss/val
tag: Loss/val

The final model

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 32, 62, 62]           2,432
              ReLU-2           [-1, 32, 62, 62]               0
       BatchNorm2d-3           [-1, 32, 62, 62]              64
            Conv2d-4           [-1, 32, 60, 60]          25,632
              ReLU-5           [-1, 32, 60, 60]               0
         MaxPool2d-6           [-1, 32, 30, 30]               0
       BatchNorm2d-7           [-1, 32, 30, 30]              64
            Conv2d-8           [-1, 96, 30, 30]          27,744
              ReLU-9           [-1, 96, 30, 30]               0
      BatchNorm2d-10           [-1, 96, 30, 30]             192
           Conv2d-11          [-1, 128, 30, 30]         110,720
             ReLU-12          [-1, 128, 30, 30]               0
        MaxPool2d-13          [-1, 128, 15, 15]               0
      BatchNorm2d-14          [-1, 128, 15, 15]             256
           Conv2d-15          [-1, 256, 15, 15]         295,168
             ReLU-16          [-1, 256, 15, 15]               0
      BatchNorm2d-17          [-1, 256, 15, 15]             512
           Conv2d-18          [-1, 384, 15, 15]         885,120
             ReLU-19          [-1, 384, 15, 15]               0
        MaxPool2d-20            [-1, 384, 3, 3]               0
      BatchNorm2d-21            [-1, 384, 3, 3]             768
          Flatten-22                 [-1, 3456]               0
           Linear-23                 [-1, 3456]      11,947,392
             ReLU-24                 [-1, 3456]               0
          Dropout-25                 [-1, 3456]               0
           Linear-26                 [-1, 3456]      11,947,392
             ReLU-27                 [-1, 3456]               0
          Dropout-28                 [-1, 3456]               0
           Linear-29                  [-1, 200]         691,400
================================================================
Total params: 25,934,856
Trainable params: 25,934,856
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.05
Forward/backward pass size (MB): 12.06
Params size (MB): 98.93
Estimated Total Size (MB): 111.04
----------------------------------------------------------------
```



Final Augmented (batch 256)

Conclusion: this task allowed me to get some insights about hyper-parametetrs of layers (such as stride, pad, etc.) – before it I just tried something that may work. I again faced the problem wit the data augmentation (next time I won't dive right into training but will spend some time to check and analyse the dataset and augmentation for it). I also faced my self that this thesis is wrong: more layers – better net. See the different ways to reularization, and had way more insight about it than I had on ML course.

Links which I found useful. This list is not comprehensive:

1. https://machinelearningmastery.com/how-to-reduce-overfitting-in-deep-learning-with-weight-regularization/#:~:text=The%20most%20common%20type%20of,range%20between%200%20and%200.1.
2. https://ml-cheatsheet.readthedocs.io/en/latest/activation_functions.html
3. https://towardsdatascience.com/a-quick-guide-to-activation-functions-in-deep-learning-4042e7addd5b
4. chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/viewer.html?pdfurl=https%3A%2F%2Farxiv.org%2Fpdf%2F2012.06782.pdf&clen=2525212&chunk=true