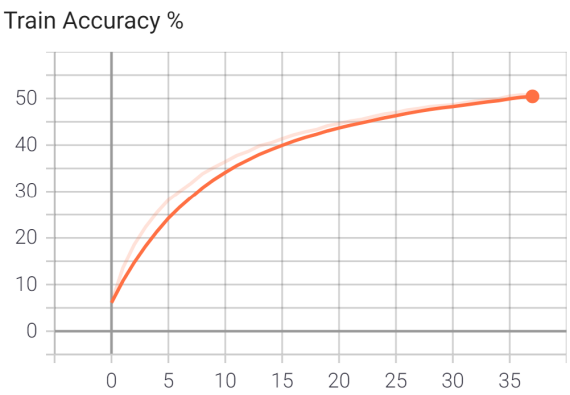
All the time I tested models from torchvision lib. First, I tried with ResNet18, Adam optimizer, lr = 10e-3. It was around 20 % most of the time. Second, I added augmentation (http://cs231n.stanford.edu/reports/2017/pdfs/12.pdf - roughly speaking: bigger number of augmentations is better) and change to DenseNet161 - it is slower than ResNet18 but more deep and can give a better accuracy (based on articles e.g. https://arxiv.org/pdf/2002.08991.pdf). The accuracy was around 25-30%. Third, I increased the number of epochs from 10 till 30, decreased lr to lr=10e-4. The final accuracy was 49%. Last, I increased the number of epochs and the accuracy became >50% on the train and val score at the end of the notebook was 45.85% (if model pre-train weights loaded):

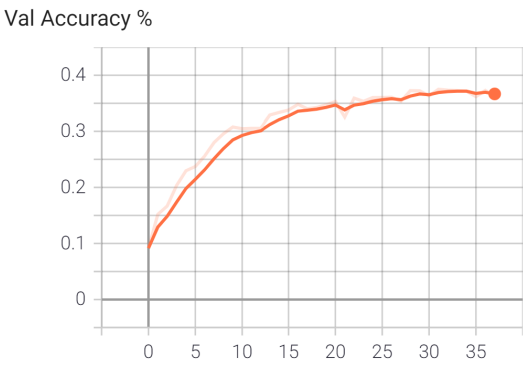
```
val_accuracy, _ = validate(val_dataloader, model)
assert 1.5 <= val_accuracy <= 100.0
print("Validation accuracy: %.2f%%" % val_accuracy)
```

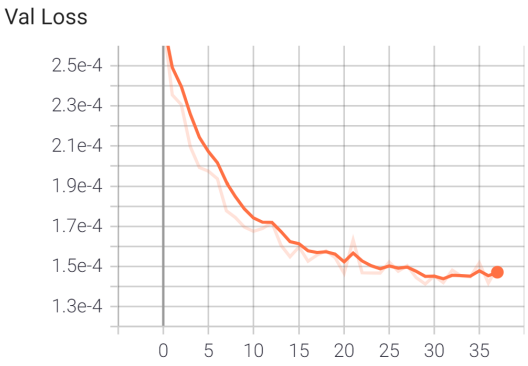
```
Validation accuracy: 45.85%
```

Some time was spended with batch size - tried to take as big as it possible, but there were memory problems on val. Then it was decreased, batch size: 320 -> 300 -> 256 -> 128, and 128 is the final batch size. Data augmentation was some kind counterintuitive at first, because there is so much data (I thought) but it is a useful tool in net training and increased accuracy. As for technical problems, there were issues with loss and accuracy calculation - that is why I have a huge accuracy first, because it was calculated from from epoch to epoch. The same story with loss. Multiple colab accounts for experimenting really helps!

So, the optimization is done via Adam (it can be also checked SGD with momentum, but in my practice it most always performs worse). The regularization is augmentation and architecture structure (batch normalization). The net is chosen in principle that it should be significantly big to learn necessary patterns and train in some adequate amount of time. DenseNet161 is chosen in this way. PS Sorry for the wrong scale on val graphs (accuracy should be multiplied by 100 to show %, and some bag with loss numbers, nevertheless tendency is right). Not changed because of limited time :(







New (additional) comments.
I also tried a number of other models - VGG with/without batch norm, Resnet152, MobileNetv3 (mostly inspired by the 3rd HW). All the models showed ~ same or worse accuracy. Also I tried another optimizer - Adadelta, made a bruteforce on learning rate and scheduler. The problem is always the same - train accuracy increases, but on validation it is saturated on some point and cannot go up. I thought that augmentations would be enough for it; of course it was helpful, but not enough. Learning rate is rather difficult to tune optimally, just based on obtained scores you took it bigger/lower or use a scheduler when it is saturating and accuracy is not improved.

For me insights are the following: experiments are rather long, need to stop bad ones early (not waiting for the miracle till the end), googling more (if the problem with overfiting, maybe there are guys who dealt with the same issues and even on the same dataset - just spend more time on searching). Also, increased batch size is not always a way to improve accuracy (first, it is counterintuitive - you take more samples to train and it should lead to better generalization of unknown data distribution, but on practice I faced roughly speaking the opposite - till some number it can increase, but further it also can decrease too; so, it is not a silver bullet - distributions can be rather exotic and complex.). Also, for better generalization property of the network and accuracy score it is better to test really deep nets, not spend much time on shallow or middle size ones.

In terms of overfitting were tried all the most popular receipts - early stopping, dropout, augmentations. Maybe I should also try L1/L2 regularization. All the techniques give their impact into final accuracy, but not always (dropout can also make your network train longer and sometimes even achieve lower accuracy score; also simplification of net can help for preventing overfitting, but there should be also a balance - to have big enough net for generalization and small enough to prevent overfitting). Nevertheless, these all mostly empirical rules and all of them should be tested manually on each particular examples, but the main idea is that there is a pack of techniques which you can start with,