Experiments with transfer learning

Dataset:

Dataset consisted of two classes: Bees and Ants. It contained 195 images of each class, 120 were used for training and 75 for validation.

Data Augmentation:

```
data_transforms = {
    'train': transforms.Compose([
        transforms.RandomResizedCrop(224),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ]),
    'val': transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ]),
}
```

Model:

I used ResNet18 network which was pretrained on Imagenet. I replaced last fully connected layer in order to predict only two classes. I made two runs: in first run model was updating all weights while in the second I freezed all weights except the last layer. This was done using next code:

```
for param in model.parameters():
    param.requires_grad = False
num_ftrs = model.fc.in_features
model.fc = nn.Linear(num_ftrs, 2)
```

Both models were trained for 25 epochs.

Learning rate - 0.001

Momentum - 0.9

Optimizer - SGD.

Learning rate scheduler - multiply the learning rate by 0.1 each 7th epoch.

Results:

1. Time

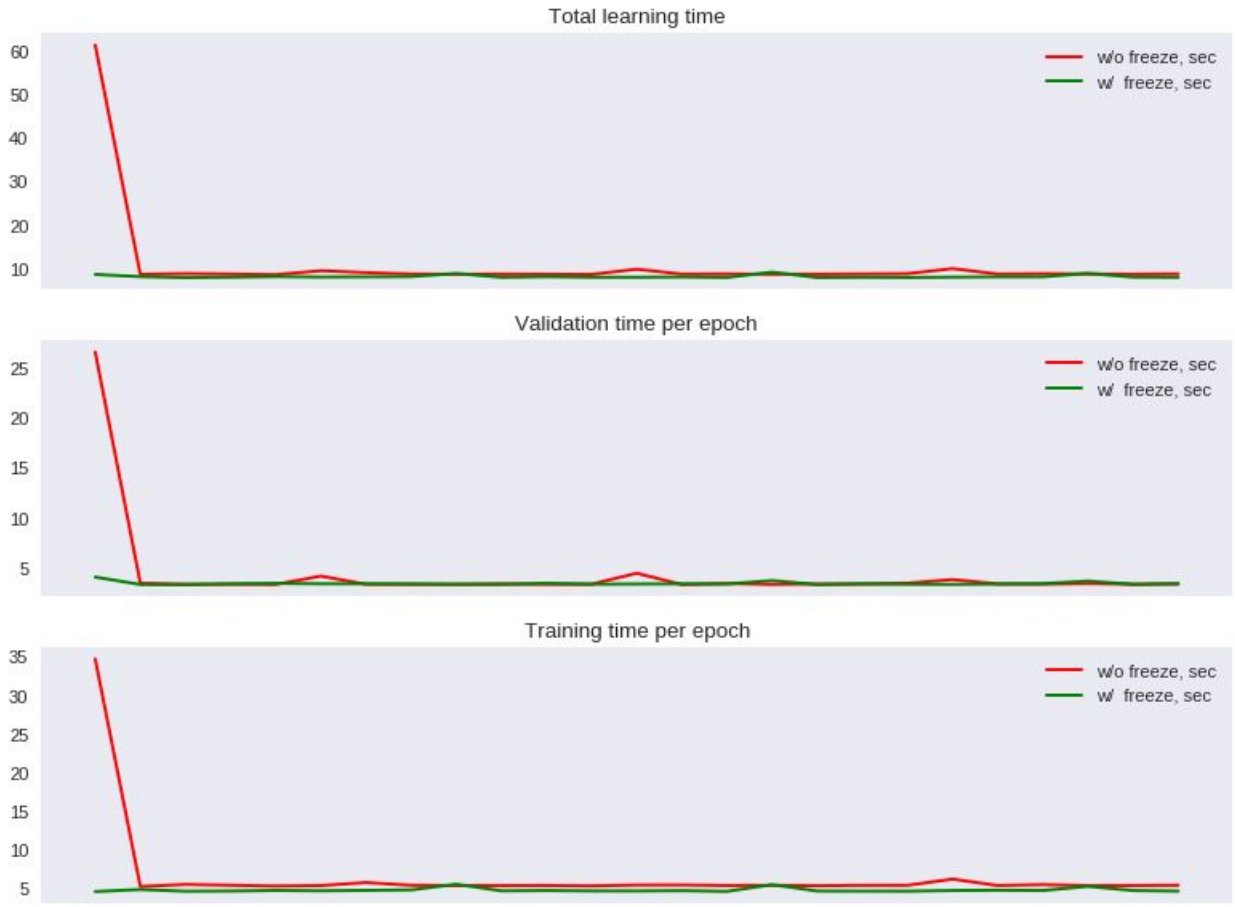

Fig 1. How long models were running

Model without frozen weights completed training in 4m 38s.
Model with frozen weights on all layers except the last one completed training in 3m 28s, which is 1.34 times faster.

2. Loss and accuracy

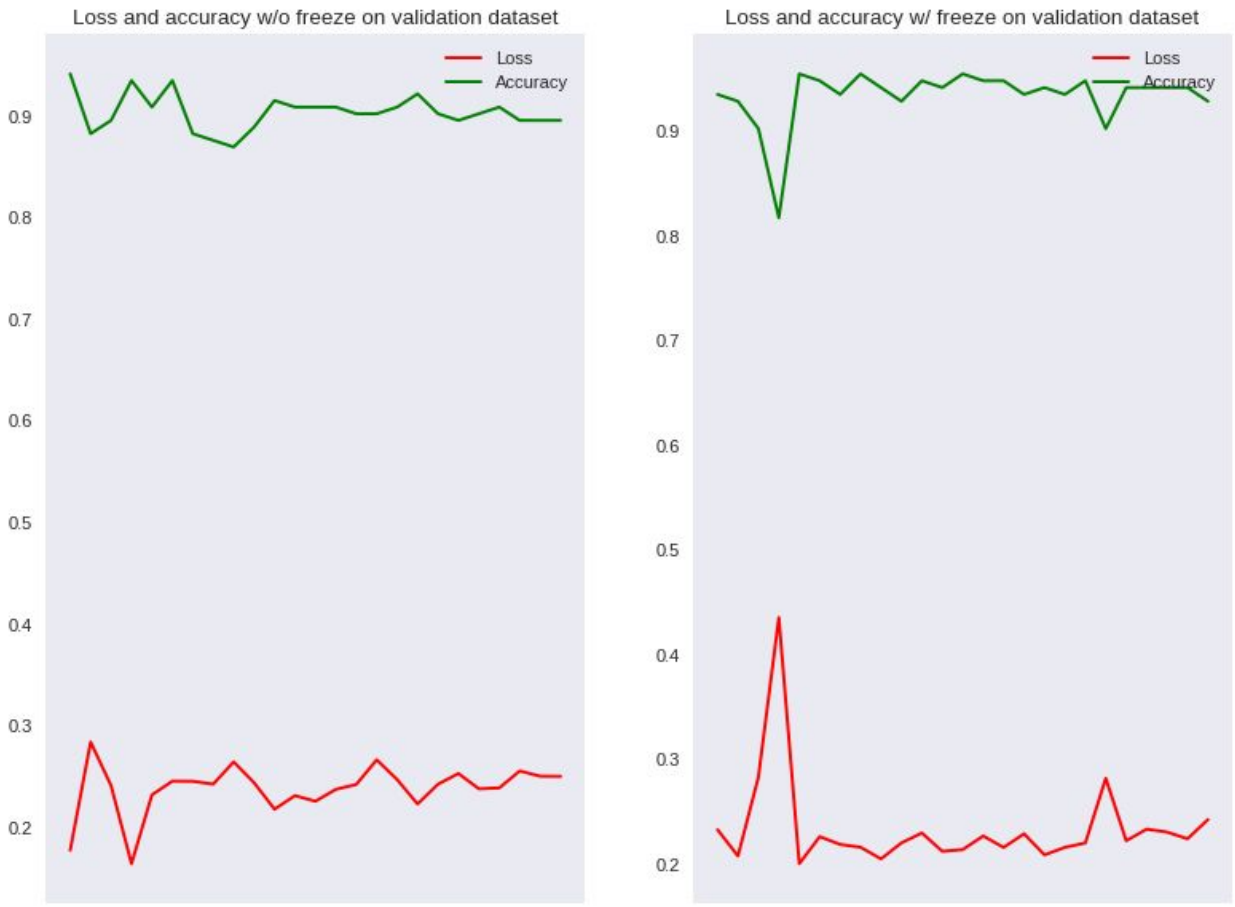

Fig 2. Loss and accuracy for both models

Model without frozen weights reached best validation accuracy of 0.941176.
Model with frozen weights reached best validation accuracy of 0.954248.

Experiment with scheduler
Model used:
Lenet_in3x32x32_out10 from https://github.com/lyubonko/ucu2018_dl4cv
The first run was done using params_cifar10.yaml
For the second I've added scheduler, multiplying learning rate by 0.2 each 5th epoch.

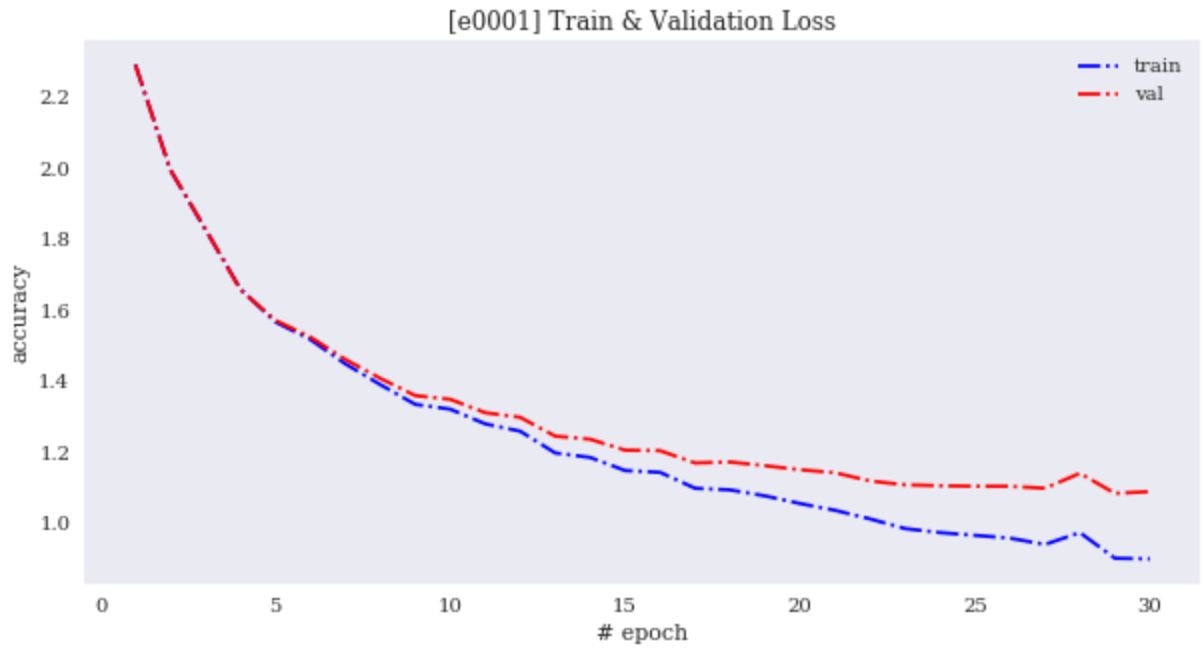

Fig 3. Loss, the first model

Fig 3. Loss, the second model

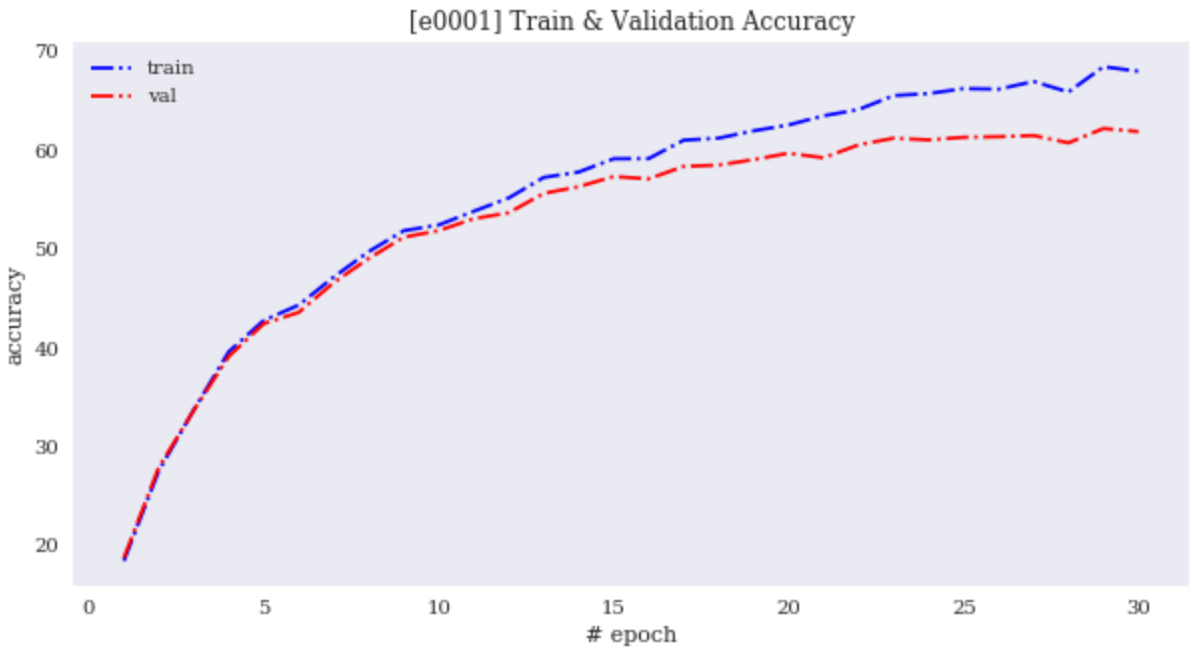

Fig 4. Accuracy, the first model

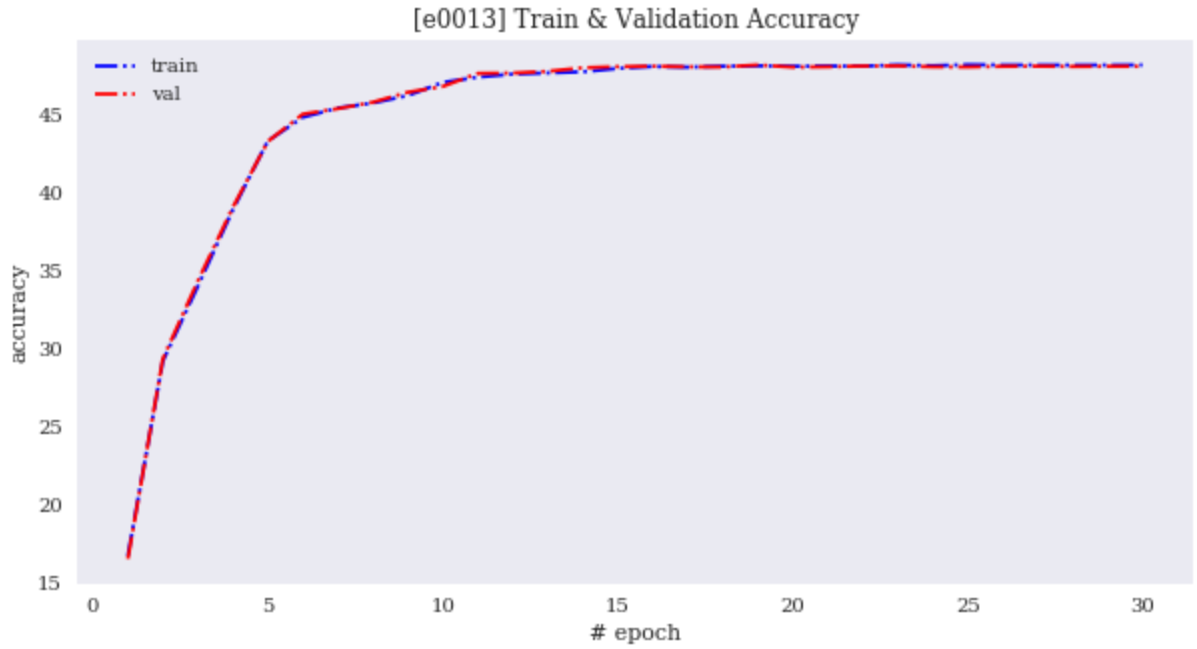

Fig 4. Accuracy, the second model

From this plots, we can see that the second model's final loss is much bigger than the first one. And also accuracy is much smaller. So, I decided to increase the starting learning rate from 0.001 to 0.1 and to see what will be the results.

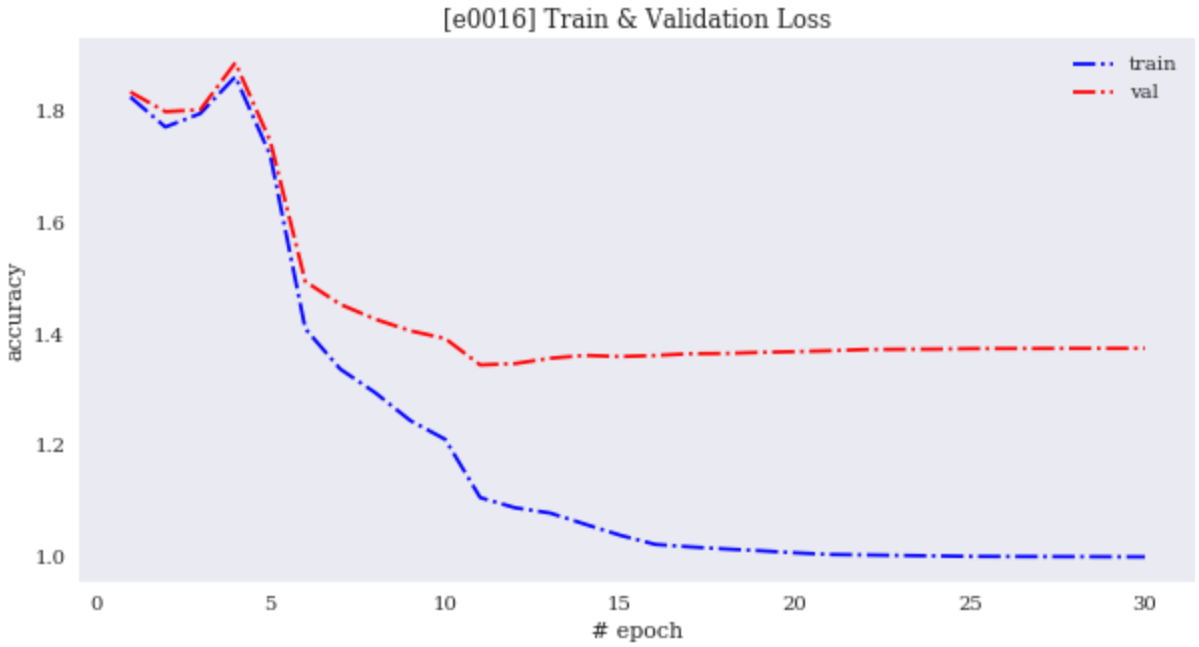

Fig 5. Loss after changing initial learning rate

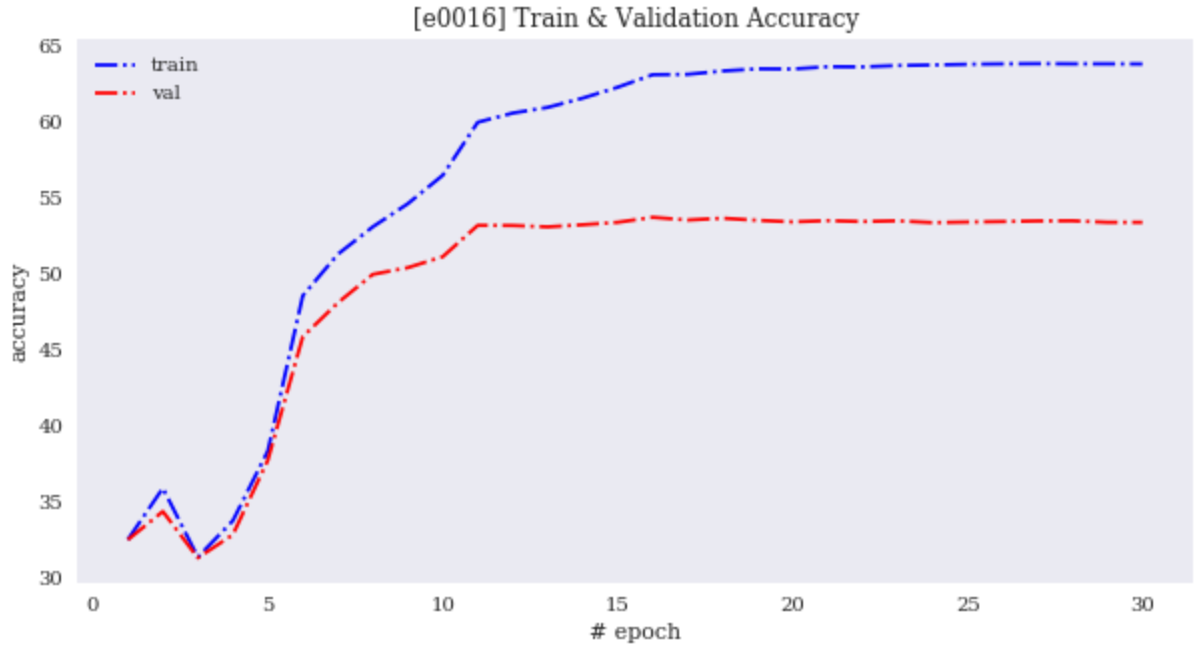

Fig 5. Accuracy after changing initial learning rat

Looks like this doesn't help much. I think that the problem is in the too big initial learning rate, but the deadline is coming and I don't have enough time to check this hypothesis :(