Project: Transfer learning for image classification task

Hanna Pylieva

Motivation

Understand how transfer learning works in practice, try it on different pretrained models with different data augmentation and parameters. I am going to resolve the task of classification as a work project, that is why I decided to try how transfer learning works in this project.

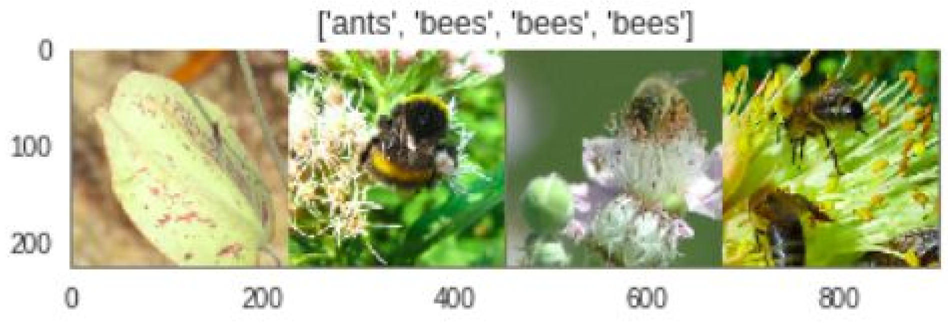
Base code and data taken from:
https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html#finetuning-the-convnet

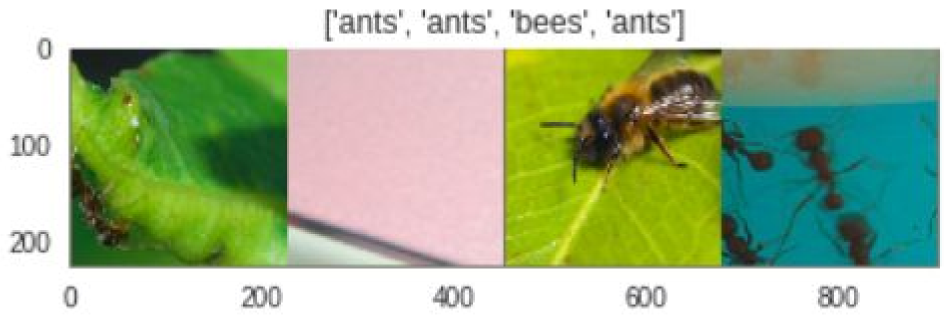
Data

Ants and bees images.
Train:
- ants - 124 images
- bees - 121 images

Val:
- ants - 70 images
- bees - 83 images
-

Few images from the dataset:





Result evaluation

I will track cross-entropy loss and accuracy on validation set. In the end I calculate the model's best validation accuracy and according to this will compare models.

Experiments

Important Notes:
My experiments were run in Google Colaboratory, visualization is done on TensorBoard. I used PyTorch.

I firstly ran experiments 1-6 and descibed them. Then Google Colaboratory refused to connect properly and when it started to work again all my progress (logs from the previous runs) were lost, as they were stored in Google Colaboratory environment and I didn't found how to load them properly on Google Drive and back in new session. So in a new session I ran experiments 0,1 and compared them, 5,7 and compared them. That is why

- in experiment (1) plots for net in exp.(1) doesn't match plots of the same net (exp. (1)) in exp-s 2-4.

- in experiment (7) plots for net for exp.(5) doesn't match plots of the same net (exp. (5)) in exp-s 5-6.

But I assumed that this is due to some random staff and won't impact comparison and conclusions in each of experiments. If I had more time, I would have rerun all the experiments :) Anyway, sorry for any inconveniences this can cause for perception of this report.

0. Load a pretrained Resnet18 net and change the number of classes it is expected to predict. Train as usual. No augmentation here is used (**pretrained_resnet-no-augm** - this is the name of this network on plots; the name is specified in each experiment).

    - Number of epochs: 25

```
data_transforms = {
    'train': transforms.Compose([
        transforms.RandomResizedCrop(224),
        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])
    ]),
}
```

    - Pretrained model: resnet18
    - Optimizer: SGD
        - LR: 0.001
        - Momentum: 0.9
    - LR in multiplied by 0.1 on every 7th epoch

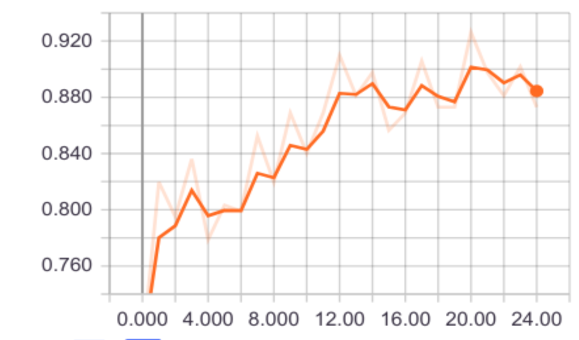
```
model_ft = models.resnet18(pretrained=True)
num_ftrs = model_ft.fc.in_features
model_ft.fc = nn.Linear(num_ftrs, 2)
```

```
model_ft = model_ft.to(device)
criterion = nn.CrossEntropyLoss()

optimizer_ft = optim.SGD(model_ft.parameters(), lr=0.001, momentum=0.9)

# Decay LR by a factor of 0.1 every 7 epochs
exp_lr_scheduler = lr_scheduler.StepLR(optimizer_ft, step_size=7, gamma=0.1)
```

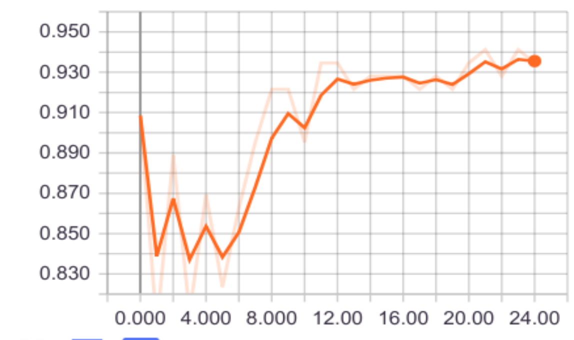
Results:
Training complete in 3m 56s
Best val Acc: 0.941176

✓ ○ pretrained-resnet-no-augm

train_acc



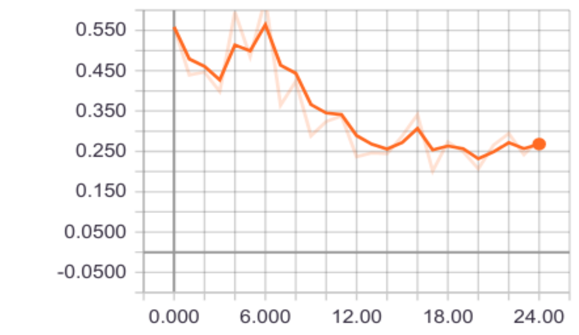
val_acc



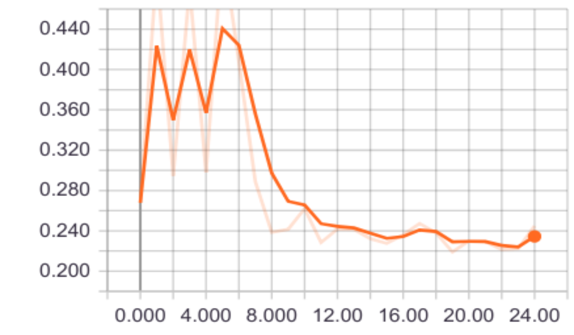
train_loss

val_loss

train_loss



val_loss



Few images to validate classification visually:

Analysis: after struggling on the first 5 epochs network starts to perform good. It seems that it reached its maximum in 25 epochs, so no need to train it longer. I was astonished that the network provided such good results on such a small dataset. But I want to try a little augmentation.

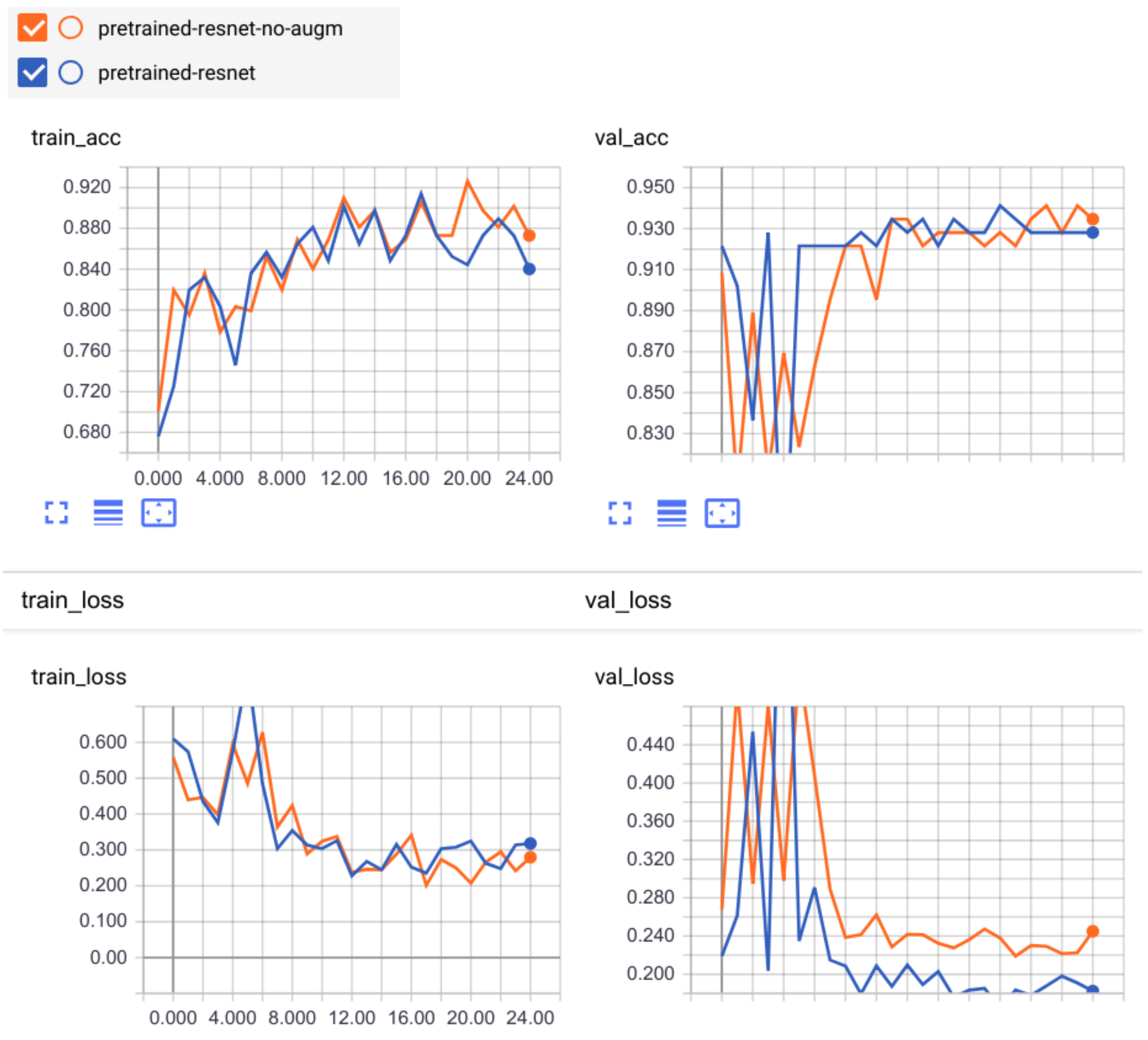
1. To base of exp.(0) add `RandomHorizontalFlip` data augmentation (**pretrained-resnet**):

```
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])
    ]),
}
```

Everything else is the same.

Results:

Training complete in 3m 4s
Best val Acc: 0.941176

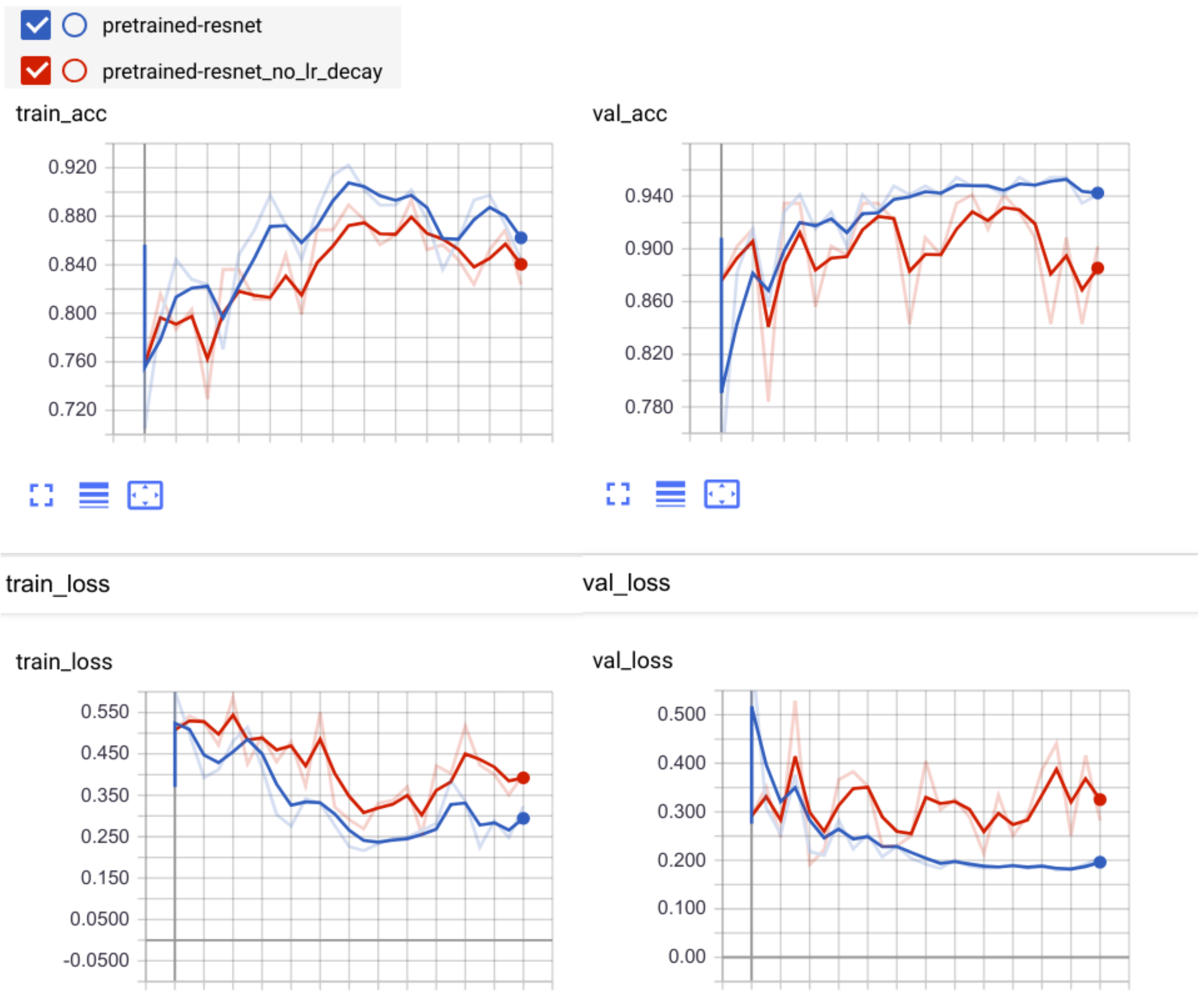


Analysis: Validation accuracy went much lower than in exp. (0), but train accuracy started to decrease after 17th epoch. But in the end, the maximum achieved val_acc in this exp. Is the same as in exp. (0), but it took 52 seconds less to train and still val_loss is lower, so ***this model will be used as a baseline in all the rest of experiments***.

2. On base of exp. (1) remove LR decay at all (so it is the same on all epochs) (**pretrained-resnet_no_lr_decay**).

Results:

Training complete in 3m 13s
Best val Acc: 0.941176



Few images to validate classification visually:

Analysis: Model shows much poorer performance than in exp. (1), especially starting from 20th epoch where the validation accuracy starts to drop. Visual overview of results for 6 random images (obviously not enough to derive some strong decisions, but still allows to build some overview whether the net clasifies correctly) shows that net still classifies correctly in most of cases.

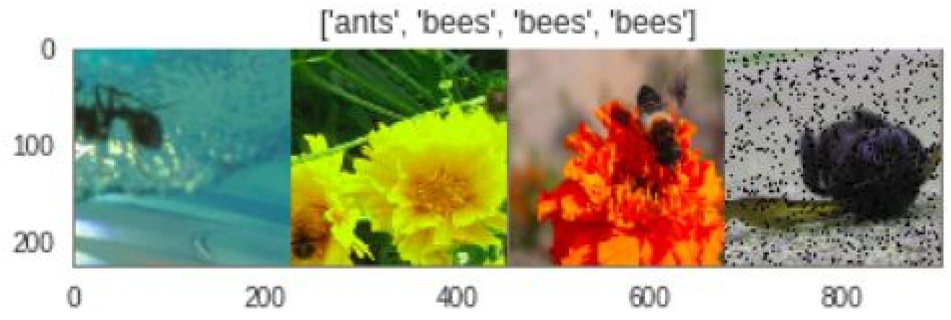
3. On base of experiment (1) add more data augmentation (**pretrained-resnet-more-augm**).

RandomHorizontalFlip was removed and other augmentation added.

```
from imgaug import augmenters as iaa
import imgaug as ia

class ImgAugTransform:
    def __init__(self):
        self.aug = iaa.Sequential([
            iaa.Scale((224, 224)),
            iaa.Sometimes(0.25, iaa.GaussianBlur(sigma=(0, 3.0))),
            iaa.Fliplr(0.5),
            iaa.Affine(rotate=(-20, 20), mode='symmetric'),
            iaa.Sometimes(0.25,
                          iaa.OneOf([iaa.Dropout(p=(0, 0.1)),
                                     iaa.CoarseDropout(0.1, size_percent=0.5)])),
            iaa.AddToHueAndSaturation(value=(-10, 10), per_channel=True)
        ])

    def __call__(self, img):
        img = np.array(img)
        return self.aug.augment_image(img)
```

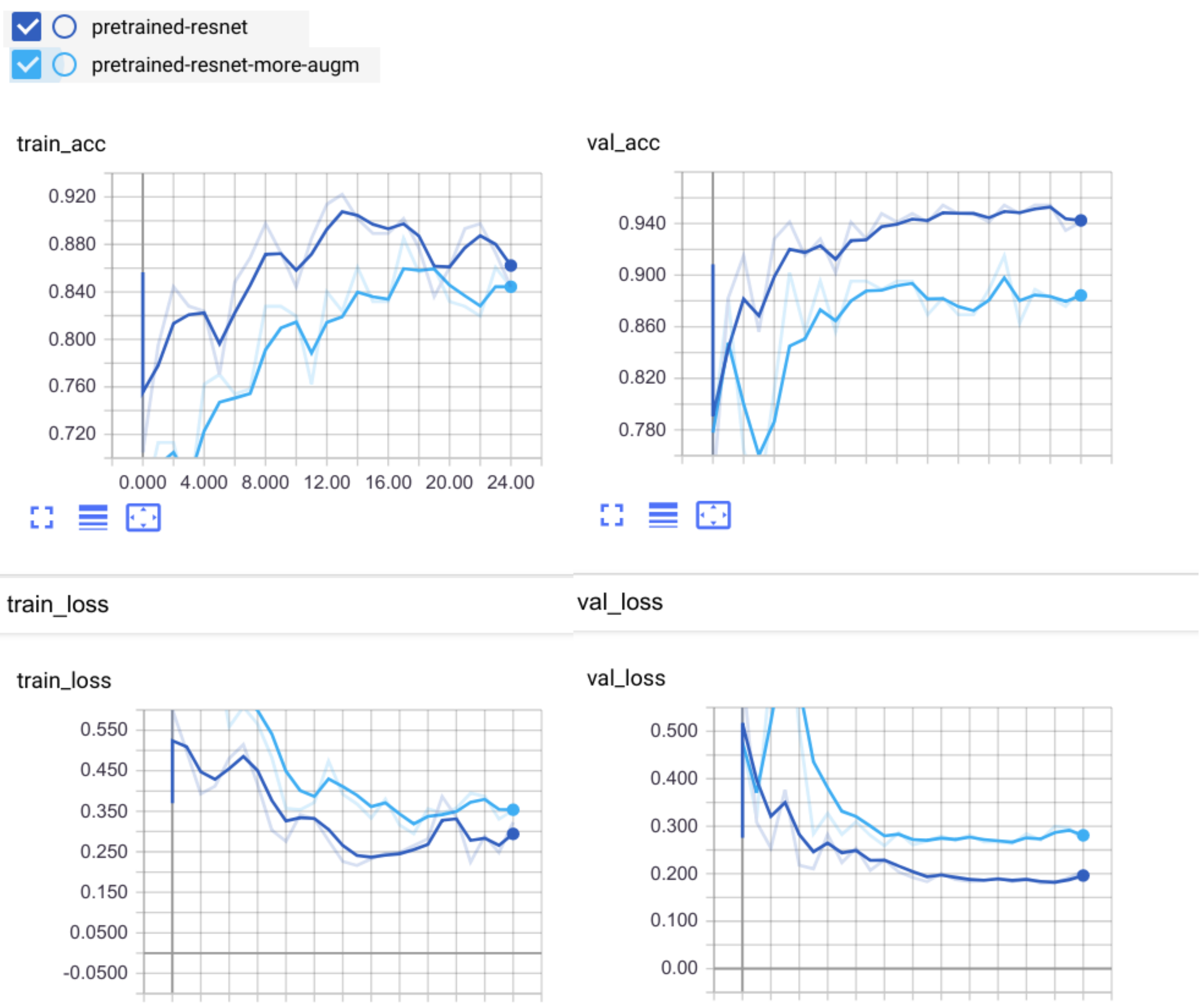
```
data_transforms = {
    'train': transforms.Compose([
        transforms.RandomResizedCrop(224),
        ImgAugTransform(),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ]),
    'val': transforms.Compose([
        transforms.Resize(256),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

Few input images for now:

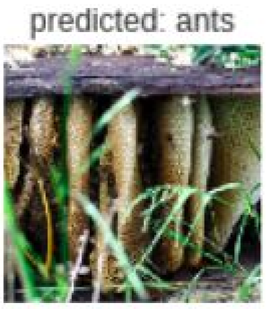


Results:

Training complete in 3m 42s
Best val Acc: 0.915033



Few images to validate classification visually:

Analysis: Model shows steadily poorer performance, than in exp. (1). Validation loss is on the higher level than in exp. (1) throughput all learning. Validation accuracy doesn't increase higher than 91.5%, which is 4% lower than in exp. (1) on the same epoch.

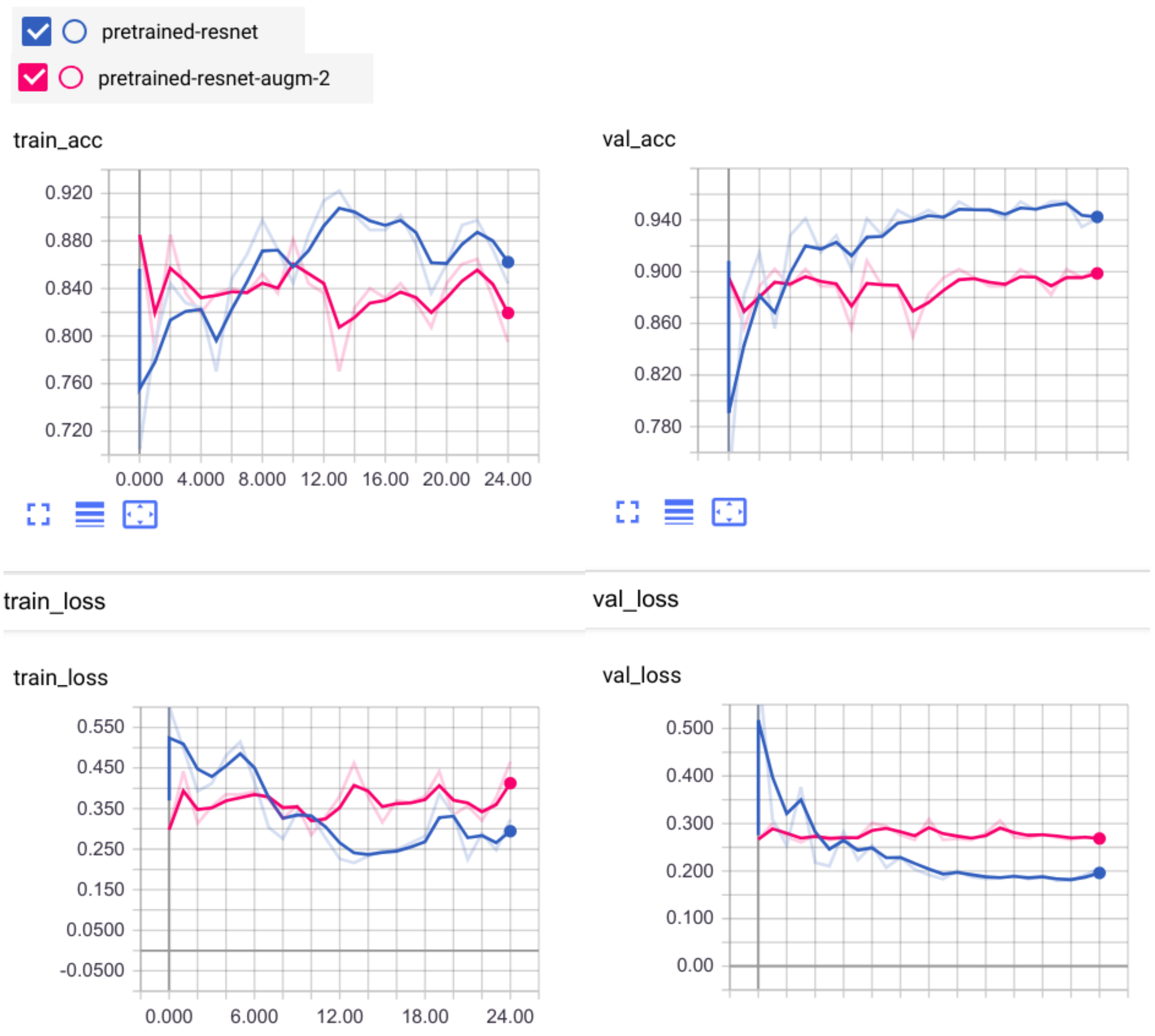
But from the few random images we can see that the model still classifies correctly.

4. On base of exp. (1) add another data augmentation - simplier than in exp. (3) (**pretrained-resnet-augm-2**)

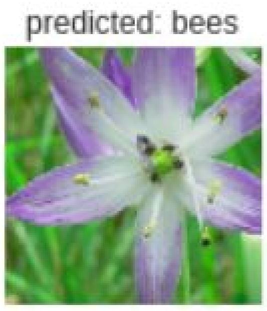
```
data_transforms = {
    'train': transforms.Compose([
        transforms.RandomResizedCrop(224),
        transforms.RandomHorizontalFlip(),
#         ImgAugTransform(),
        transforms.RandomRotation(20),
        transforms.ColorJitter(hue=.05, saturation=.05),
        transforms.ToTensor(),
        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])
    ]),
}
```

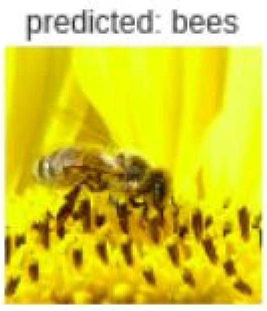
Results:

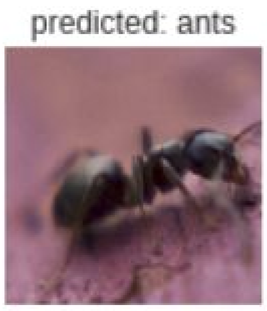
Training complete in 3m 26s
Best val Acc: 0.908497

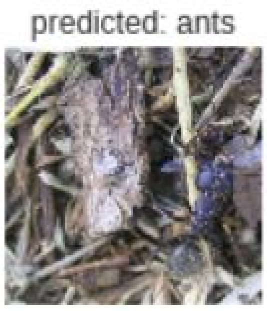


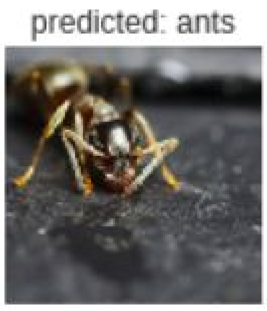
Few images to validate classification visually:

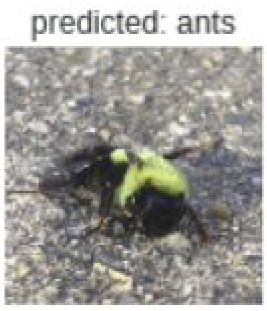






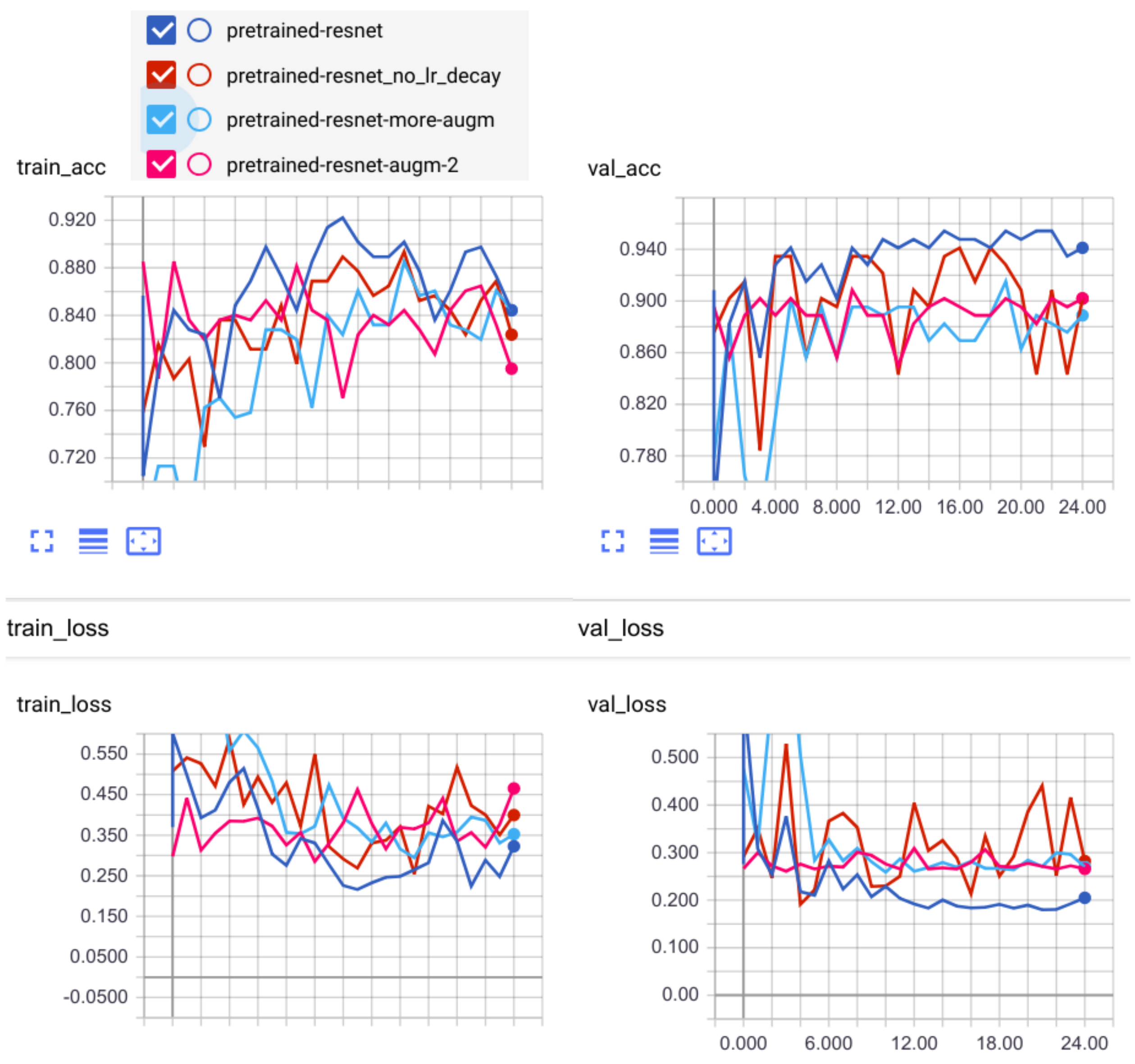






Analysis: Validation Loss doesn't decrease and validation accuracy almost doesn't grow. But val-loss starts on lower position than in exp. (1). Max val accuracy is the lowest so far. Even on random set of images we can see that the last bee was classified incorrectly.

Comparison of experiments 1,2,3 and 4 on one set of graphs:



5. Freeze all the network except the final FC layer. This last layer is replaced with a new one with random weights and only this layer is trained (**freezed-resnet**).

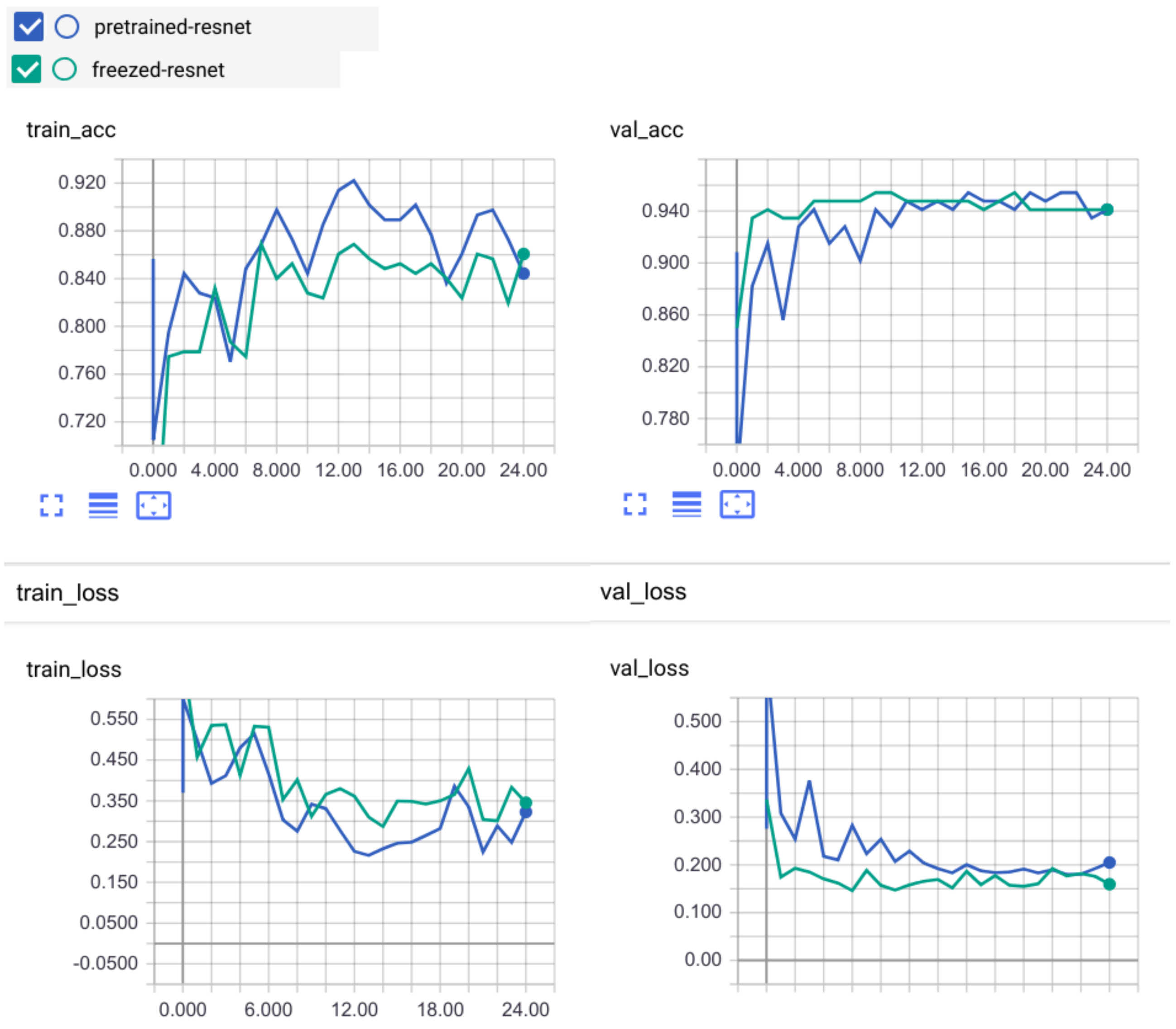
So the only part that changed from exp. (1) is:

```
model_conv = torchvision.models.resnet18(pretrained=True)
for param in model_conv.parameters():
    param.requires_grad = False

# Parameters of newly constructed modules have requires_grad=True by default
num_ftrs = model_conv.fc.in_features
model_conv.fc = nn.Linear(num_ftrs, 2)
```

Rest of parameters (optimizer, input data augmentation, ..) remained the same as in exp. (1)

Results:

Training complete in 2m 46s
Best val Acc: 0.954248



Analysis: This model shows better accuracy than the one from exp. (1) from the very beginning, but then doesn't improve much. Considering that the val-loss is lower than train loss, it doesn't seem to me that the model started to overfit. Probably it just fitted to it's best val-acc on the 9th epoch and cannot perform better.

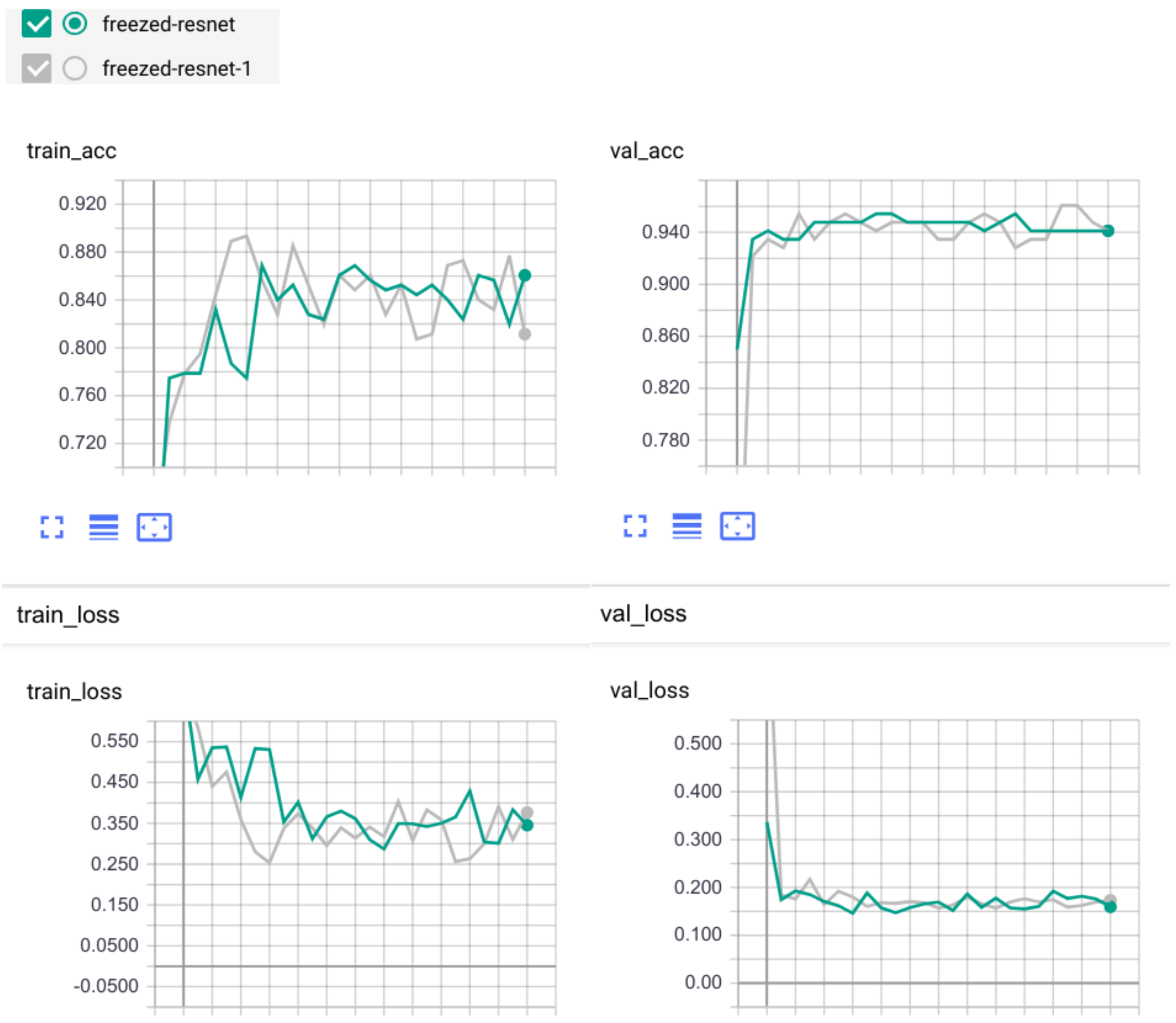
Also it is noticeable that not only the best accuracy was achieved 5 epochs faster than in exp. (5), but also all 25 epochs took less time to run (by 27 seconds).

6. On base of exp. (5) make LR decay more frequent but on less gamma (**freezed-resnet-1**).

```
# Decay LR by a factor of 0.07 every 5 epochs
exp_lr_scheduler = lr_scheduler.StepLR(optimizer_conv, step_size=5, gamma=0.07)
```

Results:

Training complete in 2m 51s
Best val Acc: 0.960784

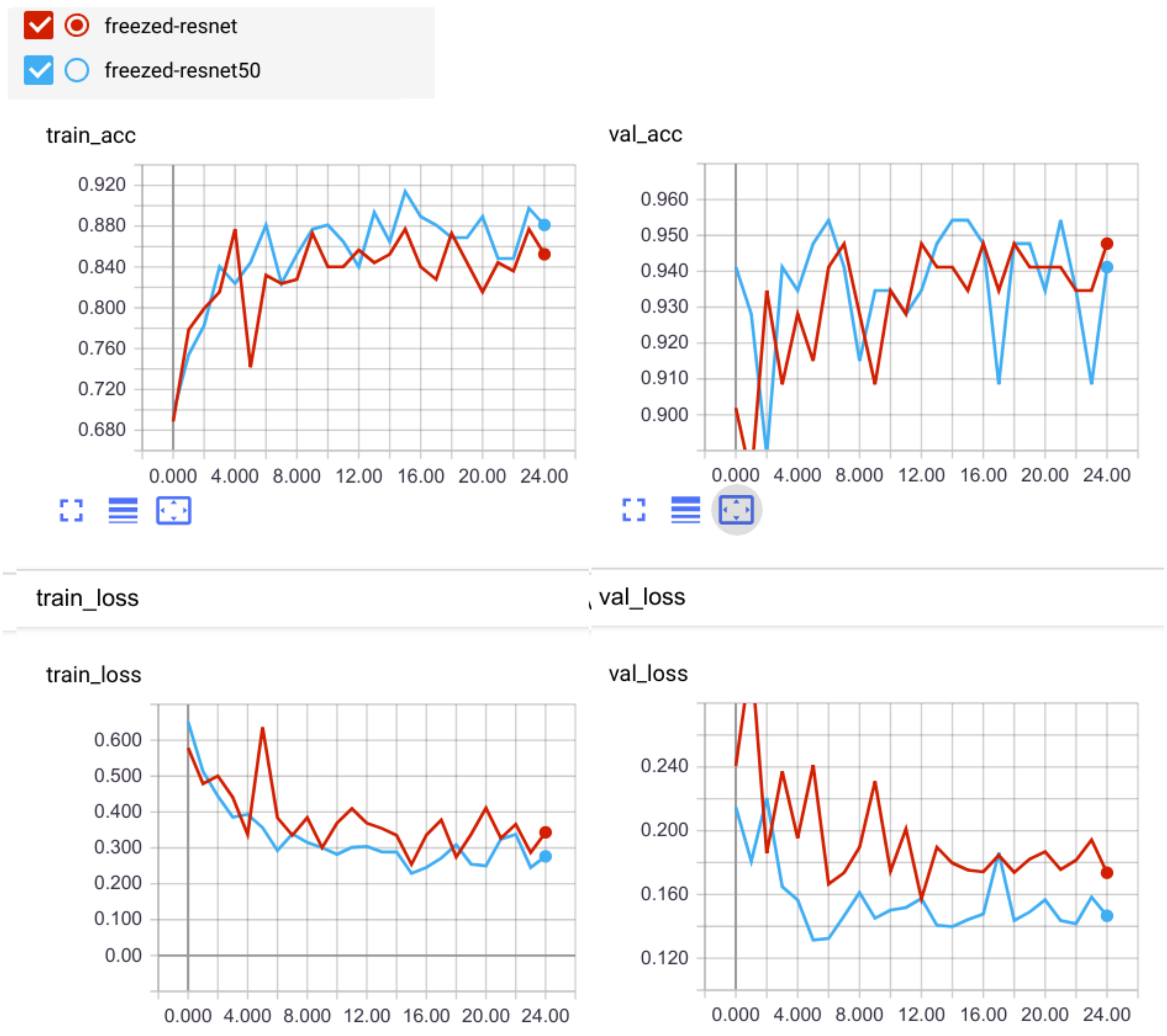


Analysis: The best accuracy achieved is a little smaller than in exp. (5), but this seems to be insignificant.

7. Change the pretrained model used in exp. (5). Now I will use resnet50 instead resnet18. Everything else remains the same (**freezed-resnet50**).

Results:

Training complete in 3m 4s
Best val Acc: 0.954248



Analysis: Val-loss went lower with resnet50, but maximum validation accuracy didn't improve. This model took 18 seconds more to run through all 25 epochs, but achieved the bets val_acc 1 epoch earlier than with resnet18. But the last difference doesn't seem significant to me and seem to be variable due to random seed chosen.

Conclusions from all the experiments:

- Data augmentation doesn't always improve the model performance. Especially when the performance without data augmentation is already good.
- On small dataset and a little classes (2 in my case) deeper network won't improve the classification result significantly.

The full code I used for performing the experiments can be found on github.

Thanks!

Veronika, Andrii, thank you for the course and this task.

I learned a lot from your lectures even though I watched them in record. This task was very helpful to summarize a lot of information we’ve received on lectures and to play around with different parameters and tools (I really felt like Wonder Woman when managed to run PyTorch models on external dataset in Google Colaboratory and visualize results on TensorBoard XD).

So I found this place the best for expressing my gratitude and highly positive impression of the course :)