EE 599 HW 4 – Report – Tieming Sun

Model fitting problem between pretrained model (ResNet 50) and my own model:

Overfitting is more obvious when we train pretrained model instead of our own model. (See figures below). That is, training accuracy in pretrained model could reach 90% or more, but validation accuracy is much lower, about 65%, whereas our own model (I use VGG 16 structure), is tuned, the training and validation accuracy is almost the same. A pretrained model (ResNet 50) might not be so compatible with this dataset, because it is preliminary trained on ImageNet. But, the pretrained models are trained with a larger dataset, which means model parameter might get some "general" features that could apply to the Polyvore Outfit dataset.

Training speed problem between pretrained model and my own model:

Training speed of the pretrained model is likely to fast than my own model, regardless of model's complexity. Parameters in pretrained model, when initialized, tends to be able to extract some features than our own model. So it would be faster for the pretrained model to get a so-called accuracy plateau.

Learning rate difference:

Pretrained model has a good start point to train, which means it is able to start with a lower learning rate, which is 0.001, with batch size of 64. Whereas in my own model (training from scratch), I set a scheduler to half learning rate every 5 epochs, starting from 0.002, with batch size of 64.

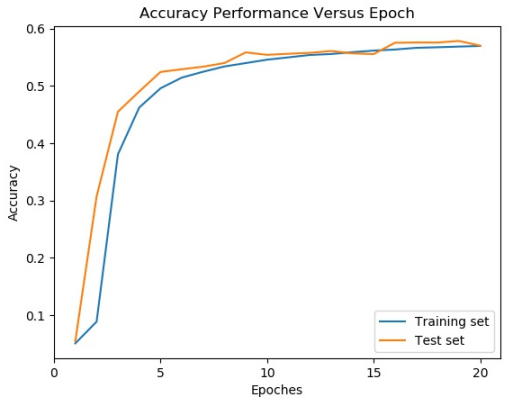

Figure 1. Category finding, train model from scratch (vgg 16)

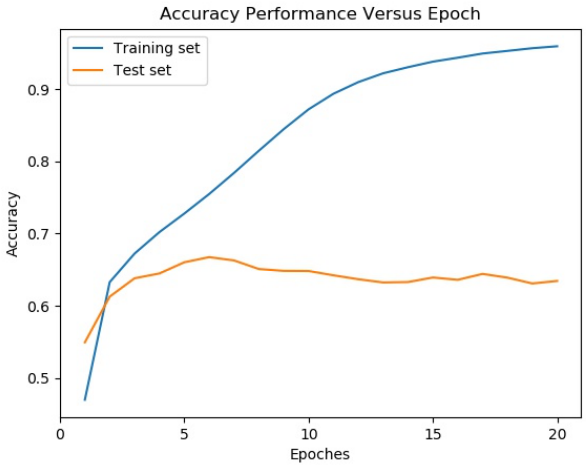

Figure 2. Category finding, pretrained model (ResNet 50)

Pairwise model training:

For this problem, I use a pretrained model (mobile net) to learn compatibility of cloth pairs. To train the model, I overlay a picture on the other picture, so the input channel will be 6 instead of 3. The other modification of the model is that I modify the output class number to 1, and add a sigmoid function to turn the output into a number between 0 and 1. And set threshold as 0.5 to see their class. I trained with learning rate of 0.002, 10 epochs.

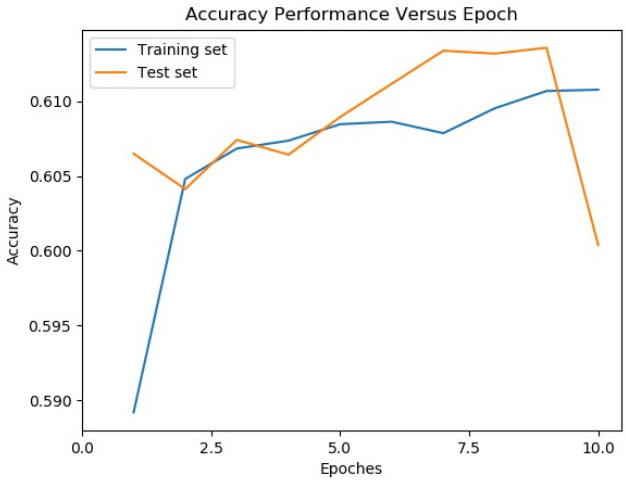

Figure 3. Pairwise training, pretrained model (Mobile Net)

Key part of my code:

Pairwise model training:
I use BCE Loss, Adam optimizer, LR_sheduler with pretrained model mobile_net, and learning rate of 0.002, and it runs about 40 min an epoch.

```
if __name__=='__main__':
```

```
    dataloaders, classes, dataset_size =
get_pairloader(debug=Config['debug'], batch_size=Config['batch_size'],
num_workers=Config['num_workers'])
    # acc_list - global variables
    acc_train_list = []
    acc_test_list = []
    loss_train_list = []
    loss_test_list = []

    # model = net16_pair
    model = mobile
    model.features[0][0] = nn.Conv2d(6, 32, kernel_size=(3, 3),
stride=(2, 2), padding=(1, 1), bias=False)
    fc_features = model.classifier[1].in_features
    model.classifier[1] = nn.Sequential(
        nn.Linear(fc_features, classes),
        nn.Sigmoid())

    criterion = nn.BCELoss()
    optimizer = torch.optim.Adam(model.parameters(),
lr=Config['learning_rate'], weight_decay=0.0001)
    device = torch.device('cuda:0' if torch.cuda.is_available() and
Config['use_cuda'] else 'cpu')

    train_model(dataloaders, model, criterion, optimizer, device,
num_epochs=Config['num_epochs'], dataset_size=dataset_size)
    plot(acc_train_list, acc_test_list, "pairwise.jpg",
num_epochs=Config['num_epochs'])
    plot(loss_train_list, loss_test_list, "pairwise_loss.jpg",
num_epochs=Config['num_epochs'])
```

Category model training:
I use CrossEntropy Loss, Adam optimizer, LR_sheduler with my own model VGG 16, and learning rate of 0.002, and it runs about 13-15 min an epoch.

For pretrained model (resnet 50), CrossEntropy Loss, RMS optimizer, and learning rate of 0.0021, and it runs about 8 min an epoch.

Data.py:

For pairwise model, __getitem__ function of dataset is modified, in order to stack 2 pictures into an input of (6, 224, 224).

```
# For pairset determination
class polyvore_pair_train(Dataset):

    def __init__(self, X_train, y_train, transform):
        self.X_train = X_train
        self.y_train = y_train
        self.transform = transform
        self.image_dir = osp.join(Config['root_path'], 'images')

    def __len__(self):
        return len(self.X_train)

    def __getitem__(self, item):
        file_path1 = osp.join(self.image_dir, self.X_train[item][0])
        file_path2 = osp.join(self.image_dir, self.X_train[item][1])
        new_X = torch.cat((self.transform(Image.open(file_path1)),
self.transform(Image.open(file_path2))), 0)
        return new_X, self.y_train[item]
```