# LAB 3-2 : Transfer Learning

Dataset is sampling from Food-11/training

Testing dataset still use from Food-11/evaluation

ResNet-18 as network

Use and not use pre-trained model

# The First Implementation

## Prepare sampled training dataset

*import os*

*import shutil*

*fo = open("sample11@100.dms")*

*src\_files = []*

*for line in fo:*

*src\_files.append('/tmp/dataset-nctu/Food-11/training/'+line.strip('\n'))*

*fo.close()*

*dest = '/tmp/work/LAB3-2/sampled\_train'*

*for file\_name in src\_files:*

*if (os.path.isfile(file\_name)):*

*shutil.copy(file\_name, dest)*

## ResNet18 model

It is easy to construct a whole new ResNet18 by

*model = models.resnet18()*

It also used *model = models.resnet18(pretrained=True)* to load pre-trained model. Due to the final output classes number is 11, it can change the final FC layer by *model.fc = nn.Linear(model.fc.in\_features, 11)*.

In this implementation, there are three different type transfer learning, trained from scratch (SCRATCH mode), trained the whole network (ALL mode) and trained the last FC layer (LAST mode). Only the last one did not update CONV layer’s gradient:

*for param in model.parameters():*

*param.requires\_grad = False*

Using SGD as the optimizer by *optim.SGD(model.parameters(), lr=LR, momentum=MOMENTUM)*. But in the LAST mode, it just only optimized the last FC layer by *optim.SGD(model.fc.parameters(), LR, momentum=MOMENTUM)*.

## Training phase

Due to the GPU memory size limitation, it cannot run these three models in the same time. It can train model one by one and *del model* in each run.

## The results

After 200 epoches, SCRATCH mode is

*200 epoch, training accuracy: 69.3%*

*loss : 0.852*

*testing accuracy: 41.5%*

*loss : 1.979*

ALL mode is

*200 epoch, training accuracy: 20.9%*

*loss : 2.155*

*testing accuracy: 16.1%*

*loss : 2.203*

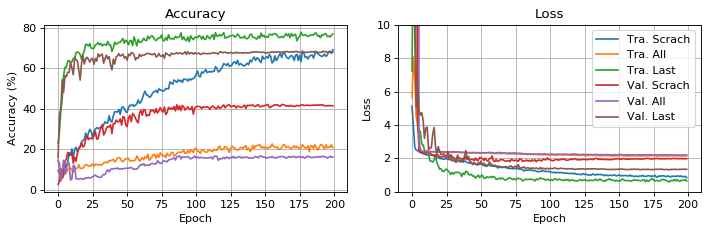
LAST mode is

*200 epoch, training accuracy: 77.1%*

*loss : 0.644*

*testing accuracy: 68.1%*

*loss : 1.344*



And the testing results are

*Evaluation of mode scratch. testing accuracy: 42.4% loss : 1.855*

*Evaluation of mode all. testing accuracy: 16.2% loss : 2.251*

*Evaluation of mode last. testing accuracy: 69.2% loss : 1.182*

# The Second Implementation

## Transfer Learning

In practice, very few people train an entire Convolutional Network from scratch (with random initialization), because it is relatively rare to have a dataset of sufficient size. Instead, it is common to pretrain a Convolutional Network on a very large dataset (e.g. ImageNet, which contains 1.2 million images with 1000 categories), and then use the Convolutional Network either as an initialization or a fixed feature extractor for the task of interest.

These two major transfer learning scenarios look as follows:

1. Fine tuning the convolutional network: Instead of random initialization, we initialize the network with a pretrained network, like the one that is trained on imagenet 1000 dataset. Rest of the training looks as usual.
2. Convolutional network as fixed feature extractor: We freeze the weights for all of the network except that of the final fully connected layer. This last fully connected layer is replaced with a new one with random weights and only this layer is trained.

## Prepare the dataset

*../data/food/train-sample (for training)*

*../data/food/val (for validation)*

*../data/food/test (for evaluation)*

*data\_transforms = {*

*'train-sample': transforms.Compose([*

*transforms.RandomResizedCrop(224),*

*transforms.RandomHorizontalFlip(),*

*transforms.ToTensor(),*

*transforms.Normalize([0.5548, 0.4508, 0.3435], [0.2281, 0.2384, 0.2376])*

*]),*

*'val': transforms.Compose([*

*transforms.Resize(256),*

*transforms.CenterCrop(224),*

*transforms.ToTensor(),*

*transforms.Normalize([0.5604, 0.4540, 0.3481], [0.2260, 0.2367, 0.2352])*

*]),*

*'test': transforms.Compose([*

*transforms.Resize(256),*

*transforms.CenterCrop(224),*

*transforms.ToTensor(),*

*transforms.Normalize([0.5604, 0.4540, 0.3481], [0.2260, 0.2367, 0.2352])*

*]),*

*}*

*data\_dir = '../data/food/'*

*image\_datasets = {x: datasets.ImageFolder(os.path.join(data\_dir, x),*

*data\_transforms[x])*

*for x in ['train-sample', 'val', 'test']}*

*dataloaders = {x: torch.utils.data.DataLoader(image\_datasets[x], batch\_size=4,*

*shuffle=True, num\_workers=1)*

*for x in ['train-sample', 'val', 'test']}*

*dataset\_sizes = {x: len(image\_datasets[x]) for x in ['train-sample', 'val', 'test']}*

*class\_names = image\_datasets['train-sample'].classes*

*device = torch.device("cuda:0" if torch.cuda.is\_available() else "cpu")*

## Re-train the whole convolutional network from scratch

As mentioned, it’s rare to train an entire Convolutional Network from scratch (with random initialization), we do it here to understand why we need transfer learning especially when we have small training dataset.

*model\_new = models.resnet18(pretrained=False)*

*num\_ftrs = model\_new.fc.in\_features*

*model\_new.fc = nn.Linear(num\_ftrs, 11)*

*model\_new = model\_new.to(device)*

*criterion = nn.CrossEntropyLoss()*

*optimizer\_new = optim.SGD(model\_new.parameters(), lr=0.001, momentum=0.9)*

*exp\_lr\_scheduler = lr\_scheduler.StepLR(optimizer\_new, step\_size=10, gamma=0.5)*

*model\_new = train\_model(model\_new, criterion, optimizer\_new, exp\_lr\_scheduler,*

*num\_epochs=100)*

## Fine tuning the convolutional network

Load a pretrained model and reset final fully connected layer.

*model\_ft = models.resnet18(pretrained=True)*

*num\_ftrs = model\_ft.fc.in\_features*

*model\_ft.fc = nn.Linear(num\_ftrs, 11)*

*model\_ft = model\_ft.to(device)*

*criterion = nn.CrossEntropyLoss()*

*optimizer\_ft = optim.SGD(model\_ft.parameters(), lr=0.001, momentum=0.9)*

*exp\_lr\_scheduler = lr\_scheduler.StepLR(optimizer\_ft, step\_size=10, gamma=0.5)*

*model\_ft = train\_model(model\_ft, criterion, optimizer\_ft, exp\_lr\_scheduler,*

*num\_epochs=25)*

## Fixed feature extractor

We need to freeze all the network except the final layer. We need to set requires\_grad == False to freeze the parameters so that the gradients are not computed in backward().

Reference: <https://pytorch.org/docs/master/notes/autograd.html#excluding-subgraphs-from-backward>

*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, 11)*

*model\_conv = model\_conv.to(device)*

*criterion = nn.CrossEntropyLoss()*

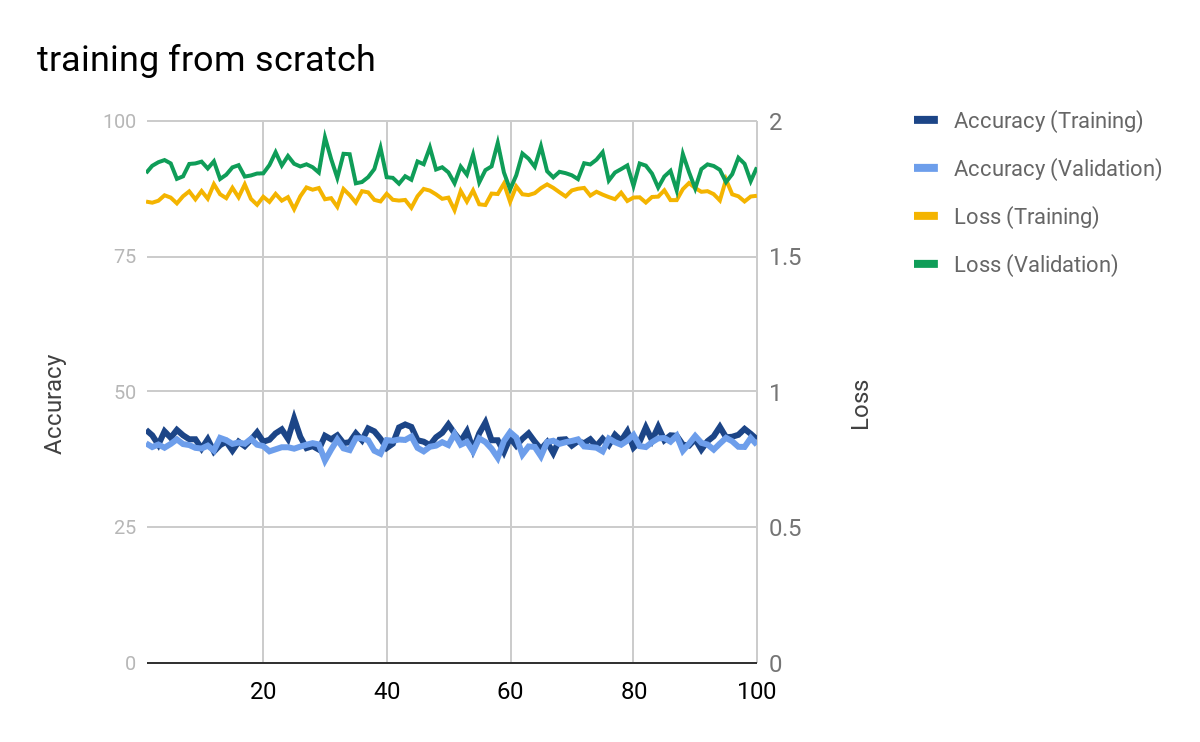
*optimizer\_conv = optim.SGD(model\_conv.fc.parameters(), lr=0.001, momentum=0.9)*

*exp\_lr\_scheduler = lr\_scheduler.StepLR(optimizer\_conv, step\_size=7, gamma=0.1)*

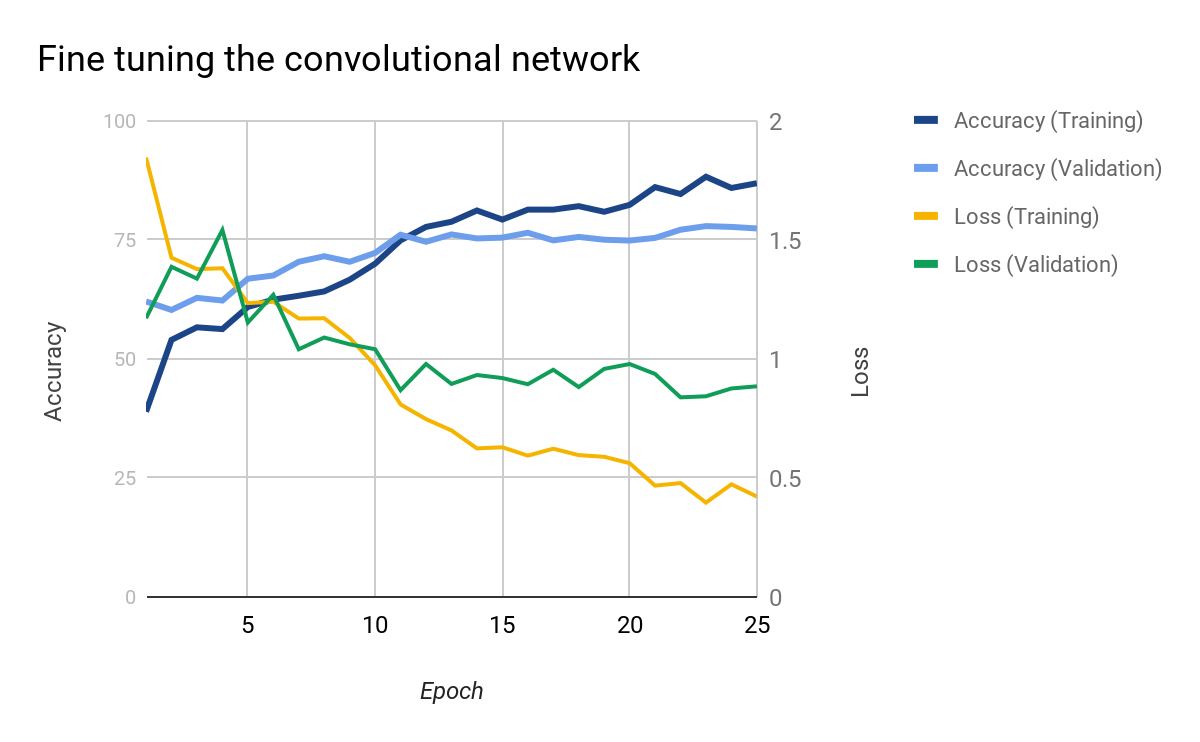
*model\_conv = train\_model(model\_conv, criterion, optimizer\_conv, exp\_lr\_scheduler, num\_epochs=25)*

## The results

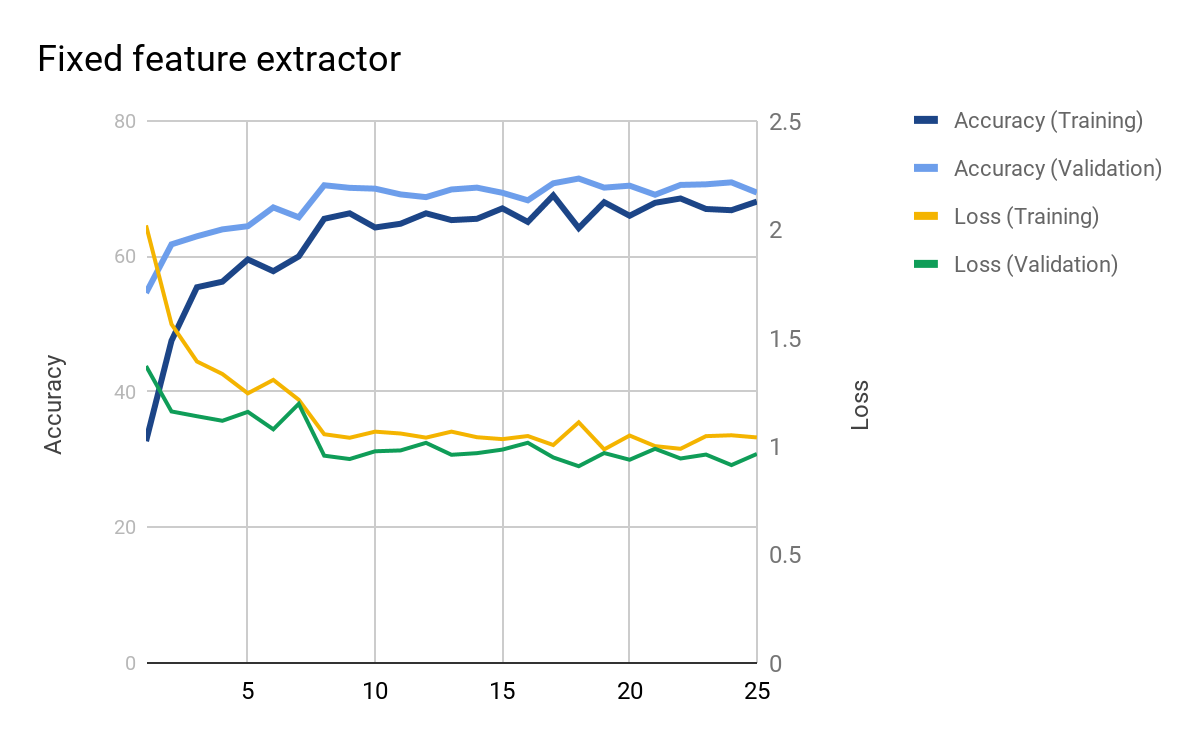
### Accuracy and Loss - Training from scratch



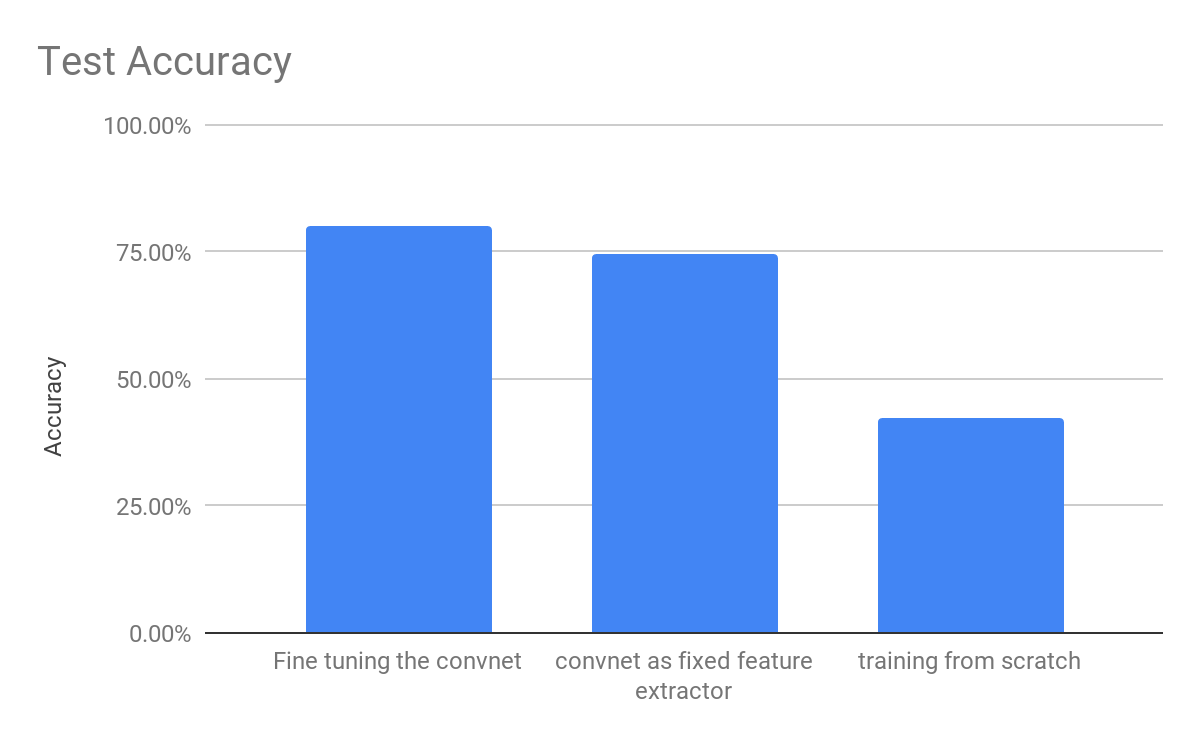
### Accuracy and Loss - Fine tuning the convolutional network



### Accuracy and Loss - Fixed feature extractor



### Test Accuracy



### Test Accuracy in each class

|  |  |  |  |
| --- | --- | --- | --- |
| Type | Fine tuning the convnet | convnet as fixed feature extractor | training from scratch |
| Total | 80.13% | 74.45% | 42.34% |
| Bread | 54 % | 66 % | 14 % |
| Dairy Product | 74 % | 75 % | 13 % |
| Dessert | 68 % | 51 % | 36 % |
| Egg | 76 % | 65 % | 11 % |
| Fried Food | 88 % | 73 % | 47 % |
| Meat | 82 % | 75 % | 44 % |
| Noodles Pasta | 98 % | 95 % | 63 % |
| Rice | 96 % | 94 % | 42 % |
| Seafood | 85 % | 75 % | 65 % |
| Soup | 92 % | 93 % | 55 % |
| Vegetable Fruit | 91 % | 82 % | 79 % |