Lab6 CNN Architectures and Transfer Learning

In practice, people usually use **standard CNN architectures** such that has been shown to deliver superior performance in a wide range of applications. In general, use these standard networks rather than design your own. Some examples of CNN architectures are:

- AlexNet (Winning entry in ILSVRC 2012)
- VGGNet (1st runner up in ILSVRC 2014)
- GoogLeNet (Winning entry in ILSVRC 2014
- ResNet (Winning entry in ILSVRC 2015)
- SqueezeNet (First successful attempt at optimising model size, 2016)
- ResNeXt (2nd place in ILSVRC 2016 classification task, 2016)
- DenseNet (Best paper award CVPR2017, Similar accuracy as ResNet but half the parameters)
- ShuffleNet V2 (Used channel shuffle operations & group convolusion to increase accuracy, 2018)
- MobileNet V2 (State of the art performance of mobile models on multiple tasks)

When training these model, typically, we do not train from scratch (with random initialization). This is because it is not common have a dataset of sufficient size to train these models.

> Training a deep architecture on a small dataset from scratch leads to overfitting issue.

Instead, it is advisable to **pretrain** a ConvNet on a very large dataset, e.g. ImageNet, which contains 1.2 million images with 1000 categories. Then, we can use the CNN in two ways:

1. **Finetuning the convnet**: Instead of random initialization, initialize the network with the pretrained network.
2. **Fixed feature extractor**: Freeze the weights for all of the layers of the network except for the final fully connected (fc) layer. Replace the last fc layer so that the output size is the same as the number of classes for the new task. The new layer is initialized with random weights and only this layer is trained.

> Training a deep architecture using a pre-trained model allows us to train on a small dataset with less overfitting.

Objectives:

In this practical, students learn how to:

1. Customize the standard CNN Network to their task
2. Train a standard CNN Network:
   A. Train from scratch
   B. Finetune the whole model
   C. Finetune the upper layers of the model
   D. As a feature extractor

Method	Accuracy
Train from scratch	**41%**
Finetune the whole model	**84.4%**

Method	Accuracy
Finetune the upper layers of the model	**70.2%**
As a feature extractor	**78.8%**

References:

1. PyTorch Official Tutorial: Transfer Learning Tutorial (https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html)
2. PyTorch Official Tutorial: Finetuning torchvision model (https://pytorch.org/tutorials/beginner/finetuning_torchvision_models_tutorial.html)

Content:

Mount google drive onto virtual machine

In [1]:
```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client
_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&
redirect_uri=urn%3aietf%3awg%3aoauth%3a2.0%3aoob&response_type=code&scope=ema
il%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdocs.test%20https%3a%2f%2fwww.
googleapis.com%2fauth%2fdrive%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdri
ve.photos.readonly%20https%3a%2f%2fwww.googleapis.com%2fauth%2fpeopleapi.read
only

Enter your authorization code:
··········
Mounted at /content/gdrive
```

Change current directory to Lab 6

In [2]:
```
cd "gdrive/My Drive/UCCD3074_Lab6"
```

```
/content/gdrive/My Drive/UCCD3074_Lab6
```

Load required libraries

In [2]:

```
%load_ext autoreload
%autoreload 2

import numpy as np
import torchvision.models as models

import torch, torchvision
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.utils.data import DataLoader
import torchvision.transforms as transforms
from torchsummary import summary

from cifar10 import CIFAR10
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

Helper Functions

Define the train function

In [0]:
```
loss_iter = 1

def train(net, num_epochs, lr=0.1, momentum=0.9, verbose=True):

    history = []

    loss_iterations = int(np.ceil(len(trainloader)/loss_iter))

    # transfer model to GPU
    if torch.cuda.is_available():
        net = net.cuda()

    # set the optimizer
    optimizer = optim.SGD(net.parameters(), lr=lr, momentum=momentum)

    # set to training mode
    net.train()

    # train the network
    for e in range(num_epochs):

        running_loss = 0.0
        running_count = 0.0

        for i, (inputs, labels) in enumerate(trainloader):

            # Clear all the gradient to 0
            optimizer.zero_grad()

            # transfer data to GPU
            if torch.cuda.is_available():
                inputs = inputs.cuda()
                labels = labels.cuda()

            # forward propagation to get h
            outs = net(inputs)

            # compute loss
            loss = F.cross_entropy(outs, labels)

            # backpropagation to get dw
            loss.backward()

            # update w
            optimizer.step()

            # get the loss
            running_loss += loss.item()
            running_count += 1

             # display the averaged loss value
            if i % loss_iterations == loss_iterations-1 or i == len(trainloade
r) - 1:
                train_loss = running_loss / running_count
                running_loss = 0.
                running_count = 0.
```

```
                if verbose:
                    print(f'[Epoch {e+1:2d}/{num_epochs:d} Iter {i+1:5d}/{len
(trainloader)}]: train_loss = {train_loss:.4f}')

                history.append(train_loss)

    return history
```

Define the evaluate function

In [0]:
```
def evaluate(net):
    # set to evaluation mode
    net.eval()

    # running_correct
    running_corrects = 0

    for inputs, targets in testloader:

        # transfer to the GPU
        if torch.cuda.is_available():
            inputs = inputs.cuda()
            targets = targets.cuda()

        # perform prediction (no need to compute gradient)
        with torch.no_grad():
            outputs = net(inputs)
            _, predicted = torch.max(outputs, 1)
            running_corrects += (targets == predicted).double().sum()

    print('Accuracy = {:.2f}%'.format(100*running_corrects/len(testloader.data
set)))
```

1. Load CIFAR10 dataset

Here, we use a sub-sample of CIFAR10 where we use a sub-sample of 1000 training and testing samples. The sample size is small and hence is expected to face overfitting issue. Using a pretrained model alleviates the problem.

In [0]:
```
# transform the model
transform = transforms.Compose([
    transforms.Resize(256),
    transforms.RandomCrop(224),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

# dataset
trainset = CIFAR10(train=True,  transform=transform, num_samples=1000)
testset  = CIFAR10(train=False,  transform=transform, num_samples=1000)

# dataloader]
trainloader = DataLoader(trainset, batch_size=32, shuffle=True, num_workers=4)
testloader  = DataLoader(testset, batch_size=128, shuffle=True, num_workers=4)
```

2. The ResNet18 model

In this section, we shall build our network using a standard network architectures. We customize resnet18 by replacing its classifier layer, i.e., the last fully connected layer with our own. The original classifier layer has 1000 outputs (ImageNet has 1000 output classes) whereas our model has only 10.

Network Architecture of ResNet18

We shall use resnet18 as our base network. Before we customize it, let's print out the summary of all layers of the model to view its architecture. Bear in mind that to customize the network, we need to replace the last linear layer.

First, let's review the resnet18 network architecture.

In [7]:
```
resnet18 = models.resnet18()
summary(resnet18, (3, 224, 224), device="cpu")
```

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 64, 112, 112]	9,408
BatchNorm2d-2	[-1, 64, 112, 112]	128
ReLU-3	[-1, 64, 112, 112]	0
MaxPool2d-4	[-1, 64, 56, 56]	0
Conv2d-5	[-1, 64, 56, 56]	36,864
BatchNorm2d-6	[-1, 64, 56, 56]	128
ReLU-7	[-1, 64, 56, 56]	0
Conv2d-8	[-1, 64, 56, 56]	36,864
BatchNorm2d-9	[-1, 64, 56, 56]	128
ReLU-10	[-1, 64, 56, 56]	0
BasicBlock-11	[-1, 64, 56, 56]	0
Conv2d-12	[-1, 64, 56, 56]	36,864
BatchNorm2d-13	[-1, 64, 56, 56]	128
ReLU-14	[-1, 64, 56, 56]	0
Conv2d-15	[-1, 64, 56, 56]	36,864
BatchNorm2d-16	[-1, 64, 56, 56]	128
ReLU-17	[-1, 64, 56, 56]	0
BasicBlock-18	[-1, 64, 56, 56]	0
Conv2d-19	[-1, 128, 28, 28]	73,728
BatchNorm2d-20	[-1, 128, 28, 28]	256
ReLU-21	[-1, 128, 28, 28]	0
Conv2d-22	[-1, 128, 28, 28]	147,456
BatchNorm2d-23	[-1, 128, 28, 28]	256
Conv2d-24	[-1, 128, 28, 28]	8,192
BatchNorm2d-25	[-1, 128, 28, 28]	256
ReLU-26	[-1, 128, 28, 28]	0
BasicBlock-27	[-1, 128, 28, 28]	0
Conv2d-28	[-1, 128, 28, 28]	147,456
BatchNorm2d-29	[-1, 128, 28, 28]	256
ReLU-30	[-1, 128, 28, 28]	0
Conv2d-31	[-1, 128, 28, 28]	147,456
BatchNorm2d-32	[-1, 128, 28, 28]	256
ReLU-33	[-1, 128, 28, 28]	0
BasicBlock-34	[-1, 128, 28, 28]	0
Conv2d-35	[-1, 256, 14, 14]	294,912
BatchNorm2d-36	[-1, 256, 14, 14]	512
ReLU-37	[-1, 256, 14, 14]	0
Conv2d-38	[-1, 256, 14, 14]	589,824
BatchNorm2d-39	[-1, 256, 14, 14]	512
Conv2d-40	[-1, 256, 14, 14]	32,768
BatchNorm2d-41	[-1, 256, 14, 14]	512
ReLU-42	[-1, 256, 14, 14]	0
BasicBlock-43	[-1, 256, 14, 14]	0
Conv2d-44	[-1, 256, 14, 14]	589,824
BatchNorm2d-45	[-1, 256, 14, 14]	512
ReLU-46	[-1, 256, 14, 14]	0
Conv2d-47	[-1, 256, 14, 14]	589,824
BatchNorm2d-48	[-1, 256, 14, 14]	512
ReLU-49	[-1, 256, 14, 14]	0
BasicBlock-50	[-1, 256, 14, 14]	0
Conv2d-51	[-1, 512, 7, 7]	1,179,648
BatchNorm2d-52	[-1, 512, 7, 7]	1,024
ReLU-53	[-1, 512, 7, 7]	0
Conv2d-54	[-1, 512, 7, 7]	2,359,296

```
      BatchNorm2d-55            [-1, 512, 7, 7]           1,024
           Conv2d-56            [-1, 512, 7, 7]         131,072
      BatchNorm2d-57            [-1, 512, 7, 7]           1,024
             ReLU-58            [-1, 512, 7, 7]               0
       BasicBlock-59            [-1, 512, 7, 7]               0
           Conv2d-60            [-1, 512, 7, 7]       2,359,296
      BatchNorm2d-61            [-1, 512, 7, 7]           1,024
             ReLU-62            [-1, 512, 7, 7]               0
           Conv2d-63            [-1, 512, 7, 7]       2,359,296
      BatchNorm2d-64            [-1, 512, 7, 7]           1,024
             ReLU-65            [-1, 512, 7, 7]               0
       BasicBlock-66            [-1, 512, 7, 7]               0
AdaptiveAvgPool2d-67            [-1, 512, 1, 1]               0
           Linear-68                 [-1, 1000]         513,000
================================================================
Total params: 11,689,512
Trainable params: 11,689,512
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.57
Forward/backward pass size (MB): 62.79
Params size (MB): 44.59
Estimated Total Size (MB): 107.96
----------------------------------------------------------------
```

From previous cell, our next task is to replace Layer `Linear-68` in the model. But what is its name? To get the name, let's print the names of the name of the layers (modules) at the root of the network.

In [8]:
```
for name, _ in resnet18.named_children():
    print(name)
```

```
conv1
bn1
relu
maxpool
layer1
layer2
layer3
layer4
avgpool
fc
```

It seems that the name of the last layer is called `fc` . Let's probe deeper into the module tree of the network to confirm that `fc` is really the classifier (linear) layer.

In [9]:
```
print(resnet18)
```

```
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), b
ias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_
stats=True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_m
ode=False)
  (layer1): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_runn
ing_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_runn
ing_stats=True)
    )
    (1): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_runn
ing_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_runn
ing_stats=True)
    )
  )
  (layer2): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (downsample): Sequential(
        (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
```

```
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
    )
  )
  (layer3): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=
(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (downsample): Sequential(
        (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
    )
  )
  (layer4): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=
(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (downsample): Sequential(
        (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=
```

```
(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Linear(in_features=512, out_features=1000, bias=True)
)
```

We will see that:

- `layer1` to `layer4` contains two blocks each. Each block is contains two convolutional layers.
- The second last layer (`avgpool`) performs *global average pooling* to average out the spatial dimensions.
- The last layer (`fc`) is a linear layer and indeed, it functions as a classifier. This is the layer that we want to replace to fit our model.

To customize the network, we need to replace the `fc` layer with our own classifier layer.

Customizing ResNet18

In the following, we shall replace the last layer with a new classifier layer. The original layer is designed to classify ImageNet's 1000 image categories. The new layers will be used to classify Cifar10's 10 classes

In [0]:
```
def build_network(pretrained=True):
    resnet18 = models.resnet18(pretrained=pretrained)
    in_c = resnet18.fc.in_features
    resnet18.fc = nn.Linear(in_c, 10)
    return resnet18
```

let's visualize what we have built. Note that the last layer of the network (`fc`) now has 10 instead of 1000 neurons.

In [11]:

```
print(build_network())
```

```
Downloading: "https://download.pytorch.org/models/resnet18-5c106cde.pth" to /
root/.cache/torch/checkpoints/resnet18-5c106cde.pth
```

```
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), b
ias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_
stats=True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_m
ode=False)
  (layer1): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_runn
ing_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_runn
ing_stats=True)
    )
    (1): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_runn
ing_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_runn
ing_stats=True)
    )
  )
  (layer2): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (downsample): Sequential(
        (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
```

```
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
    )
  )
  (layer3): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=
(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (downsample): Sequential(
        (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
    )
  )
  (layer4): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=
(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (downsample): Sequential(
        (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=
```

```
(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_run
ning_stats=True)
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Linear(in_features=512, out_features=10, bias=True)
)
```

Model 1: Training from scratch

Let's build the network **without** loading the pretrained model. To do this, we set `pretrained=False` .

In [0]:

```
resnet18 = build_network(pretrained=False)
```

Train the model

In [13]:
```
history1 = train(resnet18, num_epochs=50, lr=0.001, momentum=0.9)
```

```
[Epoch  1/50 Iter    32/32]: train_loss = 2.2672
[Epoch  2/50 Iter    32/32]: train_loss = 2.1142
[Epoch  3/50 Iter    32/32]: train_loss = 1.9972
[Epoch  4/50 Iter    32/32]: train_loss = 1.9236
[Epoch  5/50 Iter    32/32]: train_loss = 1.8568
[Epoch  6/50 Iter    32/32]: train_loss = 1.8228
[Epoch  7/50 Iter    32/32]: train_loss = 1.7615
[Epoch  8/50 Iter    32/32]: train_loss = 1.7087
[Epoch  9/50 Iter    32/32]: train_loss = 1.6858
[Epoch 10/50 Iter    32/32]: train_loss = 1.6436
[Epoch 11/50 Iter    32/32]: train_loss = 1.6338
[Epoch 12/50 Iter    32/32]: train_loss = 1.5795
[Epoch 13/50 Iter    32/32]: train_loss = 1.5222
[Epoch 14/50 Iter    32/32]: train_loss = 1.4971
[Epoch 15/50 Iter    32/32]: train_loss = 1.4747
[Epoch 16/50 Iter    32/32]: train_loss = 1.4142
[Epoch 17/50 Iter    32/32]: train_loss = 1.4335
[Epoch 18/50 Iter    32/32]: train_loss = 1.3976
[Epoch 19/50 Iter    32/32]: train_loss = 1.3786
[Epoch 20/50 Iter    32/32]: train_loss = 1.3682
[Epoch 21/50 Iter    32/32]: train_loss = 1.3348
[Epoch 22/50 Iter    32/32]: train_loss = 1.2664
[Epoch 23/50 Iter    32/32]: train_loss = 1.2884
[Epoch 24/50 Iter    32/32]: train_loss = 1.2245
[Epoch 25/50 Iter    32/32]: train_loss = 1.2211
[Epoch 26/50 Iter    32/32]: train_loss = 1.2117
[Epoch 27/50 Iter    32/32]: train_loss = 1.1558
[Epoch 28/50 Iter    32/32]: train_loss = 1.1471
[Epoch 29/50 Iter    32/32]: train_loss = 1.1207
[Epoch 30/50 Iter    32/32]: train_loss = 1.0994
[Epoch 31/50 Iter    32/32]: train_loss = 1.1273
[Epoch 32/50 Iter    32/32]: train_loss = 1.0570
[Epoch 33/50 Iter    32/32]: train_loss = 1.0492
[Epoch 34/50 Iter    32/32]: train_loss = 0.9788
[Epoch 35/50 Iter    32/32]: train_loss = 0.9806
[Epoch 36/50 Iter    32/32]: train_loss = 0.9600
[Epoch 37/50 Iter    32/32]: train_loss = 0.9518
[Epoch 38/50 Iter    32/32]: train_loss = 0.9422
[Epoch 39/50 Iter    32/32]: train_loss = 0.9236
[Epoch 40/50 Iter    32/32]: train_loss = 0.9001
[Epoch 41/50 Iter    32/32]: train_loss = 0.8671
[Epoch 42/50 Iter    32/32]: train_loss = 0.8284
[Epoch 43/50 Iter    32/32]: train_loss = 0.8102
[Epoch 44/50 Iter    32/32]: train_loss = 0.8270
[Epoch 45/50 Iter    32/32]: train_loss = 0.7770
[Epoch 46/50 Iter    32/32]: train_loss = 0.7573
[Epoch 47/50 Iter    32/32]: train_loss = 0.7311
[Epoch 48/50 Iter    32/32]: train_loss = 0.7315
[Epoch 49/50 Iter    32/32]: train_loss = 0.7145
[Epoch 50/50 Iter    32/32]: train_loss = 0.6886
```

Evaluate the model

In [14]:
```
evaluate(resnet18)
```

```
Accuracy = 41.00%
```

Model 2: Finetuning the pretrained model

Typically, a standard network come with a pretrained model trained on ImageNet's large-scale dataset for the image classification task.

- In the following, we shall load resnet18 with the pretrained model and use it to **initialize** the network. To do this, we set `pretrained=True` .
- The training will update the parameters **all layers** of the network.

For Windows system, the pretrained model will be saved to the following directory: `C:\Users\<user name>\.cache\torch\checkpoints` . A PyTorch model has an extension of `.pt` or `.pth` .

In [0]:
```
resnet18 = build_network(pretrained=True)
```

By default, all the layers are set to `requires_grad=True`

In [16]:

```
for name, param in resnet18.named_parameters():
    print(name, ':', param.requires_grad)
```

```
conv1.weight : True
bn1.weight : True
bn1.bias : True
layer1.0.conv1.weight : True
layer1.0.bn1.weight : True
layer1.0.bn1.bias : True
layer1.0.conv2.weight : True
layer1.0.bn2.weight : True
layer1.0.bn2.bias : True
layer1.1.conv1.weight : True
layer1.1.bn1.weight : True
layer1.1.bn1.bias : True
layer1.1.conv2.weight : True
layer1.1.bn2.weight : True
layer1.1.bn2.bias : True
layer2.0.conv1.weight : True
layer2.0.bn1.weight : True
layer2.0.bn1.bias : True
layer2.0.conv2.weight : True
layer2.0.bn2.weight : True
layer2.0.bn2.bias : True
layer2.0.downsample.0.weight : True
layer2.0.downsample.1.weight : True
layer2.0.downsample.1.bias : True
layer2.1.conv1.weight : True
layer2.1.bn1.weight : True
layer2.1.bn1.bias : True
layer2.1.conv2.weight : True
layer2.1.bn2.weight : True
layer2.1.bn2.bias : True
layer3.0.conv1.weight : True
layer3.0.bn1.weight : True
layer3.0.bn1.bias : True
layer3.0.conv2.weight : True
layer3.0.bn2.weight : True
layer3.0.bn2.bias : True
layer3.0.downsample.0.weight : True
layer3.0.downsample.1.weight : True
layer3.0.downsample.1.bias : True
layer3.1.conv1.weight : True
layer3.1.bn1.weight : True
layer3.1.bn1.bias : True
layer3.1.conv2.weight : True
layer3.1.bn2.weight : True
layer3.1.bn2.bias : True
layer4.0.conv1.weight : True
layer4.0.bn1.weight : True
layer4.0.bn1.bias : True
layer4.0.conv2.weight : True
layer4.0.bn2.weight : True
layer4.0.bn2.bias : True
layer4.0.downsample.0.weight : True
layer4.0.downsample.1.weight : True
layer4.0.downsample.1.bias : True
layer4.1.conv1.weight : True
layer4.1.bn1.weight : True
layer4.1.bn1.bias : True
```

```
layer4.1.conv2.weight : True
layer4.1.bn2.weight : True
layer4.1.bn2.bias : True
fc.weight : True
fc.bias : True
```

Train the model

In [17]:
```
history2 = train(resnet18, num_epochs=50, lr=0.001, momentum=0.9)
```

```
[Epoch  1/50 Iter    32/32]: train_loss = 2.1624
[Epoch  2/50 Iter    32/32]: train_loss = 1.4112
[Epoch  3/50 Iter    32/32]: train_loss = 0.9895
[Epoch  4/50 Iter    32/32]: train_loss = 0.7223
[Epoch  5/50 Iter    32/32]: train_loss = 0.5630
[Epoch  6/50 Iter    32/32]: train_loss = 0.4510
[Epoch  7/50 Iter    32/32]: train_loss = 0.3467
[Epoch  8/50 Iter    32/32]: train_loss = 0.2806
[Epoch  9/50 Iter    32/32]: train_loss = 0.2160
[Epoch 10/50 Iter    32/32]: train_loss = 0.1919
[Epoch 11/50 Iter    32/32]: train_loss = 0.1545
[Epoch 12/50 Iter    32/32]: train_loss = 0.1298
[Epoch 13/50 Iter    32/32]: train_loss = 0.1068
[Epoch 14/50 Iter    32/32]: train_loss = 0.1050
[Epoch 15/50 Iter    32/32]: train_loss = 0.0938
[Epoch 16/50 Iter    32/32]: train_loss = 0.0768
[Epoch 17/50 Iter    32/32]: train_loss = 0.0625
[Epoch 18/50 Iter    32/32]: train_loss = 0.0714
[Epoch 19/50 Iter    32/32]: train_loss = 0.0598
[Epoch 20/50 Iter    32/32]: train_loss = 0.0453
[Epoch 21/50 Iter    32/32]: train_loss = 0.0557
[Epoch 22/50 Iter    32/32]: train_loss = 0.0484
[Epoch 23/50 Iter    32/32]: train_loss = 0.0424
[Epoch 24/50 Iter    32/32]: train_loss = 0.0324
[Epoch 25/50 Iter    32/32]: train_loss = 0.0426
[Epoch 26/50 Iter    32/32]: train_loss = 0.0401
[Epoch 27/50 Iter    32/32]: train_loss = 0.0333
[Epoch 28/50 Iter    32/32]: train_loss = 0.0284
[Epoch 29/50 Iter    32/32]: train_loss = 0.0302
[Epoch 30/50 Iter    32/32]: train_loss = 0.0262
[Epoch 31/50 Iter    32/32]: train_loss = 0.0270
[Epoch 32/50 Iter    32/32]: train_loss = 0.0313
[Epoch 33/50 Iter    32/32]: train_loss = 0.0400
[Epoch 34/50 Iter    32/32]: train_loss = 0.0263
[Epoch 35/50 Iter    32/32]: train_loss = 0.0215
[Epoch 36/50 Iter    32/32]: train_loss = 0.0188
[Epoch 37/50 Iter    32/32]: train_loss = 0.0160
[Epoch 38/50 Iter    32/32]: train_loss = 0.0227
[Epoch 39/50 Iter    32/32]: train_loss = 0.0178
[Epoch 40/50 Iter    32/32]: train_loss = 0.0267
[Epoch 41/50 Iter    32/32]: train_loss = 0.0254
[Epoch 42/50 Iter    32/32]: train_loss = 0.0191
[Epoch 43/50 Iter    32/32]: train_loss = 0.0257
[Epoch 44/50 Iter    32/32]: train_loss = 0.0240
[Epoch 45/50 Iter    32/32]: train_loss = 0.0321
[Epoch 46/50 Iter    32/32]: train_loss = 0.0210
[Epoch 47/50 Iter    32/32]: train_loss = 0.0170
[Epoch 48/50 Iter    32/32]: train_loss = 0.0328
[Epoch 49/50 Iter    32/32]: train_loss = 0.0267
[Epoch 50/50 Iter    32/32]: train_loss = 0.0230
```

Evaluate the network

In [18]:
```
evaluate(resnet18)
```

```
Accuracy = 84.40%
```

Model 3: As a fixed feature extractor

When the dataset is too small, fine-tuning the model may still incur overfitting. In this case, you may want to try to use the pretrained as a fixed feature extractor where we train only the classifier layer (i.e., **last layer**) that we have newly inserted into the network.

In [0]:
```
# Load the pretrained model
resnet18 = build_network(pretrained=True)
```

We set `requires_grad=False` for all parameters except for the newly replaced layer `fc` , i.e., the last two parameters in `resnet.parameters()` .

In [0]:
```
parameters = list(resnet18.parameters())
for param in parameters[:-2]:
    param.requires_grad = False
```

In [21]:
```
for name, param in resnet18.named_parameters():
    print(name, ':', param.requires_grad)
```

```
conv1.weight : False
bn1.weight : False
bn1.bias : False
layer1.0.conv1.weight : False
layer1.0.bn1.weight : False
layer1.0.bn1.bias : False
layer1.0.conv2.weight : False
layer1.0.bn2.weight : False
layer1.0.bn2.bias : False
layer1.1.conv1.weight : False
layer1.1.bn1.weight : False
layer1.1.bn1.bias : False
layer1.1.conv2.weight : False
layer1.1.bn2.weight : False
layer1.1.bn2.bias : False
layer2.0.conv1.weight : False
layer2.0.bn1.weight : False
layer2.0.bn1.bias : False
layer2.0.conv2.weight : False
layer2.0.bn2.weight : False
layer2.0.bn2.bias : False
layer2.0.downsample.0.weight : False
layer2.0.downsample.1.weight : False
layer2.0.downsample.1.bias : False
layer2.1.conv1.weight : False
layer2.1.bn1.weight : False
layer2.1.bn1.bias : False
layer2.1.conv2.weight : False
layer2.1.bn2.weight : False
layer2.1.bn2.bias : False
layer3.0.conv1.weight : False
layer3.0.bn1.weight : False
layer3.0.bn1.bias : False
layer3.0.conv2.weight : False
layer3.0.bn2.weight : False
layer3.0.bn2.bias : False
layer3.0.downsample.0.weight : False
layer3.0.downsample.1.weight : False
layer3.0.downsample.1.bias : False
layer3.1.conv1.weight : False
layer3.1.bn1.weight : False
layer3.1.bn1.bias : False
layer3.1.conv2.weight : False
layer3.1.bn2.weight : False
layer3.1.bn2.bias : False
layer4.0.conv1.weight : False
layer4.0.bn1.weight : False
layer4.0.bn1.bias : False
layer4.0.conv2.weight : False
layer4.0.bn2.weight : False
layer4.0.bn2.bias : False
layer4.0.downsample.0.weight : False
layer4.0.downsample.1.weight : False
layer4.0.downsample.1.bias : False
layer4.1.conv1.weight : False
layer4.1.bn1.weight : False
layer4.1.bn1.bias : False
```

```
layer4.1.conv2.weight : False
layer4.1.bn2.weight : False
layer4.1.bn2.bias : False
fc.weight : True
fc.bias : True
```

Train the model

In [22]:
```
history3 = train(resnet18, num_epochs=50, lr=0.001, momentum=0.9)
```

```
[Epoch  1/50 Iter    32/32]: train_loss = 2.3111
[Epoch  2/50 Iter    32/32]: train_loss = 1.8610
[Epoch  3/50 Iter    32/32]: train_loss = 1.5722
[Epoch  4/50 Iter    32/32]: train_loss = 1.3819
[Epoch  5/50 Iter    32/32]: train_loss = 1.2790
[Epoch  6/50 Iter    32/32]: train_loss = 1.1993
[Epoch  7/50 Iter    32/32]: train_loss = 1.0880
[Epoch  8/50 Iter    32/32]: train_loss = 1.0487
[Epoch  9/50 Iter    32/32]: train_loss = 0.9966
[Epoch 10/50 Iter    32/32]: train_loss = 0.9547
[Epoch 11/50 Iter    32/32]: train_loss = 0.9562
[Epoch 12/50 Iter    32/32]: train_loss = 0.8831
[Epoch 13/50 Iter    32/32]: train_loss = 0.8700
[Epoch 14/50 Iter    32/32]: train_loss = 0.8726
[Epoch 15/50 Iter    32/32]: train_loss = 0.8171
[Epoch 16/50 Iter    32/32]: train_loss = 0.8263
[Epoch 17/50 Iter    32/32]: train_loss = 0.7972
[Epoch 18/50 Iter    32/32]: train_loss = 0.7898
[Epoch 19/50 Iter    32/32]: train_loss = 0.7867
[Epoch 20/50 Iter    32/32]: train_loss = 0.7959
[Epoch 21/50 Iter    32/32]: train_loss = 0.7342
[Epoch 22/50 Iter    32/32]: train_loss = 0.7551
[Epoch 23/50 Iter    32/32]: train_loss = 0.7161
[Epoch 24/50 Iter    32/32]: train_loss = 0.7272
[Epoch 25/50 Iter    32/32]: train_loss = 0.7190
[Epoch 26/50 Iter    32/32]: train_loss = 0.7254
[Epoch 27/50 Iter    32/32]: train_loss = 0.6980
[Epoch 28/50 Iter    32/32]: train_loss = 0.7193
[Epoch 29/50 Iter    32/32]: train_loss = 0.6977
[Epoch 30/50 Iter    32/32]: train_loss = 0.6820
[Epoch 31/50 Iter    32/32]: train_loss = 0.6547
[Epoch 32/50 Iter    32/32]: train_loss = 0.6487
[Epoch 33/50 Iter    32/32]: train_loss = 0.6591
[Epoch 34/50 Iter    32/32]: train_loss = 0.6682
[Epoch 35/50 Iter    32/32]: train_loss = 0.6707
[Epoch 36/50 Iter    32/32]: train_loss = 0.6535
[Epoch 37/50 Iter    32/32]: train_loss = 0.6329
[Epoch 38/50 Iter    32/32]: train_loss = 0.6291
[Epoch 39/50 Iter    32/32]: train_loss = 0.6370
[Epoch 40/50 Iter    32/32]: train_loss = 0.6176
[Epoch 41/50 Iter    32/32]: train_loss = 0.6032
[Epoch 42/50 Iter    32/32]: train_loss = 0.6177
[Epoch 43/50 Iter    32/32]: train_loss = 0.6097
[Epoch 44/50 Iter    32/32]: train_loss = 0.6255
[Epoch 45/50 Iter    32/32]: train_loss = 0.6000
[Epoch 46/50 Iter    32/32]: train_loss = 0.5974
[Epoch 47/50 Iter    32/32]: train_loss = 0.5723
[Epoch 48/50 Iter    32/32]: train_loss = 0.5853
[Epoch 49/50 Iter    32/32]: train_loss = 0.5563
[Epoch 50/50 Iter    32/32]: train_loss = 0.6231
```

Evaluate the model

In [23]:
```
evaluate(resnet18)
```

```
Accuracy = 70.20%
```

Model 4: Finetuning the top few layers

We can also tune the top few layers of the network. The following tunes all the layers in the block `layer 4` as well as the `fc` layer.

In [0]:
```
# Load the pretrained model
resnet18 = build_network(pretrained=True)
```

Then, we freeze all tha layers except for `layer4` and `fc` layers

In [0]:
```
for name, param in resnet18.named_parameters():
    if not any(name.startswith(ext) for ext in ['layer4', 'fc']):
        param.requires_grad = False
```

In [26]:
```
for name, param in resnet18.named_parameters():
    print(name, ':', param.requires_grad)
```

```
conv1.weight : False
bn1.weight : False
bn1.bias : False
layer1.0.conv1.weight : False
layer1.0.bn1.weight : False
layer1.0.bn1.bias : False
layer1.0.conv2.weight : False
layer1.0.bn2.weight : False
layer1.0.bn2.bias : False
layer1.1.conv1.weight : False
layer1.1.bn1.weight : False
layer1.1.bn1.bias : False
layer1.1.conv2.weight : False
layer1.1.bn2.weight : False
layer1.1.bn2.bias : False
layer2.0.conv1.weight : False
layer2.0.bn1.weight : False
layer2.0.bn1.bias : False
layer2.0.conv2.weight : False
layer2.0.bn2.weight : False
layer2.0.bn2.bias : False
layer2.0.downsample.0.weight : False
layer2.0.downsample.1.weight : False
layer2.0.downsample.1.bias : False
layer2.1.conv1.weight : False
layer2.1.bn1.weight : False
layer2.1.bn1.bias : False
layer2.1.conv2.weight : False
layer2.1.bn2.weight : False
layer2.1.bn2.bias : False
layer3.0.conv1.weight : False
layer3.0.bn1.weight : False
layer3.0.bn1.bias : False
layer3.0.conv2.weight : False
layer3.0.bn2.weight : False
layer3.0.bn2.bias : False
layer3.0.downsample.0.weight : False
layer3.0.downsample.1.weight : False
layer3.0.downsample.1.bias : False
layer3.1.conv1.weight : False
layer3.1.bn1.weight : False
layer3.1.bn1.bias : False
layer3.1.conv2.weight : False
layer3.1.bn2.weight : False
layer3.1.bn2.bias : False
layer4.0.conv1.weight : True
layer4.0.bn1.weight : True
layer4.0.bn1.bias : True
layer4.0.conv2.weight : True
layer4.0.bn2.weight : True
layer4.0.bn2.bias : True
layer4.0.downsample.0.weight : True
layer4.0.downsample.1.weight : True
layer4.0.downsample.1.bias : True
layer4.1.conv1.weight : True
layer4.1.bn1.weight : True
layer4.1.bn1.bias : True
```

```
layer4.1.conv2.weight : True
layer4.1.bn2.weight : True
layer4.1.bn2.bias : True
fc.weight : True
fc.bias : True
```

Train the model

In [27]:
```python
# Load the pretrained model
history4 = train(resnet18, num_epochs=50, lr=0.001, momentum=0.9)
```

```
[Epoch  1/50 Iter    32/32]: train_loss = 2.0768
[Epoch  2/50 Iter    32/32]: train_loss = 1.4822
[Epoch  3/50 Iter    32/32]: train_loss = 1.0887
[Epoch  4/50 Iter    32/32]: train_loss = 0.8927
[Epoch  5/50 Iter    32/32]: train_loss = 0.7659
[Epoch  6/50 Iter    32/32]: train_loss = 0.6563
[Epoch  7/50 Iter    32/32]: train_loss = 0.5566
[Epoch  8/50 Iter    32/32]: train_loss = 0.4853
[Epoch  9/50 Iter    32/32]: train_loss = 0.4875
[Epoch 10/50 Iter    32/32]: train_loss = 0.4122
[Epoch 11/50 Iter    32/32]: train_loss = 0.3643
[Epoch 12/50 Iter    32/32]: train_loss = 0.3143
[Epoch 13/50 Iter    32/32]: train_loss = 0.2922
[Epoch 14/50 Iter    32/32]: train_loss = 0.2609
[Epoch 15/50 Iter    32/32]: train_loss = 0.2869
[Epoch 16/50 Iter    32/32]: train_loss = 0.2316
[Epoch 17/50 Iter    32/32]: train_loss = 0.2137
[Epoch 18/50 Iter    32/32]: train_loss = 0.2019
[Epoch 19/50 Iter    32/32]: train_loss = 0.1722
[Epoch 20/50 Iter    32/32]: train_loss = 0.1641
[Epoch 21/50 Iter    32/32]: train_loss = 0.1521
[Epoch 22/50 Iter    32/32]: train_loss = 0.1391
[Epoch 23/50 Iter    32/32]: train_loss = 0.1422
[Epoch 24/50 Iter    32/32]: train_loss = 0.1298
[Epoch 25/50 Iter    32/32]: train_loss = 0.1368
[Epoch 26/50 Iter    32/32]: train_loss = 0.1305
[Epoch 27/50 Iter    32/32]: train_loss = 0.1137
[Epoch 28/50 Iter    32/32]: train_loss = 0.0827
[Epoch 29/50 Iter    32/32]: train_loss = 0.0873
[Epoch 30/50 Iter    32/32]: train_loss = 0.0932
[Epoch 31/50 Iter    32/32]: train_loss = 0.0731
[Epoch 32/50 Iter    32/32]: train_loss = 0.0767
[Epoch 33/50 Iter    32/32]: train_loss = 0.0703
[Epoch 34/50 Iter    32/32]: train_loss = 0.0685
[Epoch 35/50 Iter    32/32]: train_loss = 0.0639
[Epoch 36/50 Iter    32/32]: train_loss = 0.0699
[Epoch 37/50 Iter    32/32]: train_loss = 0.0694
[Epoch 38/50 Iter    32/32]: train_loss = 0.0570
[Epoch 39/50 Iter    32/32]: train_loss = 0.0663
[Epoch 40/50 Iter    32/32]: train_loss = 0.0748
[Epoch 41/50 Iter    32/32]: train_loss = 0.0555
[Epoch 42/50 Iter    32/32]: train_loss = 0.0469
[Epoch 43/50 Iter    32/32]: train_loss = 0.0495
[Epoch 44/50 Iter    32/32]: train_loss = 0.0442
[Epoch 45/50 Iter    32/32]: train_loss = 0.0591
[Epoch 46/50 Iter    32/32]: train_loss = 0.0455
[Epoch 47/50 Iter    32/32]: train_loss = 0.0424
[Epoch 48/50 Iter    32/32]: train_loss = 0.0425
[Epoch 49/50 Iter    32/32]: train_loss = 0.0362
[Epoch 50/50 Iter    32/32]: train_loss = 0.0343
```

Evaluate the model

In [28]:
```
evaluate(resnet18)
```

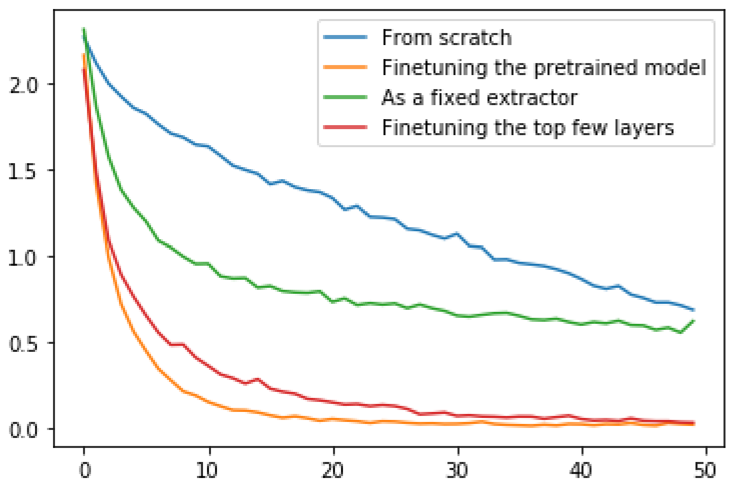
```
Accuracy = 78.80%
```

Plotting training loss

Lastly, we plot the training loss history for each of the training schemes above.

In [29]:
```
import matplotlib.pyplot as plt

plt.plot(history1, label='From scratch')
plt.plot(history2, label='Finetuning the pretrained model')
plt.plot(history3, label='As a fixed extractor')
plt.plot(history4, label='Finetuning the top few layers')
plt.legend()
plt.show()
```



Conclusion

You can try with different network architecture and compare their performances

In [0]:
```

```