# 18

January 23, 2024

[1]:
```
%pylab inline
import torch
import sys
sys.path.append('..')
sys.path.append('../..')
from data import load
device = torch.device('cuda') if torch.cuda.is_available() else torch.
 ↪device('cpu')
print('device = ', device)
```

```
%pylab is deprecated, use %matplotlib inline and import the required libraries.
Populating the interactive namespace from numpy and matplotlib
device =  cuda
```

[2]:
```
class ConvNet(torch.nn.Module):
    class Block(torch.nn.Module):
        def __init__(self, n_input, n_output, stride=1):
            super().__init__()
            self.net = torch.nn.Sequential(
              torch.nn.Conv2d(n_input, n_output, kernel_size=3, padding=1,␣
 ↪stride=stride, bias=False),
              torch.nn.BatchNorm2d(n_output),
              torch.nn.ReLU(),
              torch.nn.Conv2d(n_output, n_output, kernel_size=3, padding=1,␣
 ↪bias=False),
              torch.nn.BatchNorm2d(n_output),
              torch.nn.ReLU()
            )
            self.downsample = None
            if stride != 1 or n_input != n_output:
                self.downsample = torch.nn.Sequential(torch.nn.Conv2d(n_input,␣
 ↪n_output, 1),
                                                      torch.nn.
 ↪BatchNorm2d(n_output))

        def forward(self, x):
            identity = x
```

```
            if self.downsample is not None:
                identity = self.downsample(x)
            return self.net(x) + identity

    def __init__(self, layers=[32,64,128], n_input_channels=3):
        super().__init__()
        L = [torch.nn.Conv2d(n_input_channels, 32, kernel_size=7, padding=3,␣
↪stride=2, bias=False),
             torch.nn.BatchNorm2d(32),
             torch.nn.ReLU(),
             torch.nn.MaxPool2d(kernel_size=3, stride=2, padding=1)
            ]
        c = 32
        for l in layers:
            L.append(self.Block(c, l, stride=2))
            c = l
        self.network = torch.nn.Sequential(*L)
        self.classifier = torch.nn.Linear(c, 1)

    def forward(self, x):
        # Compute the features
        z = self.network(x)
        # Global average pooling
        z = z.mean(dim=[2,3])
        # Classify
        return self.classifier(z)[:,0]

net = ConvNet()
net.train()
print( net.training )
net.eval()
print( net.training )
```

```
True
False
```

[ ]: