0656092 蔣宜桓

1. Introduction

這次作業主要的目的是要解決深層神經網路出現的梯度爆炸或消失問題。當神經網路越來越深層深層，欲更新 weight 等參數而做 backpropagation 時，會遇到梯度爆炸或消失問題，倒置參數不更新或參數更新幅度過大，最終使正確率下降。

這次的作業為了解決這類型的問題，在神經網路間加了 shortcut 變成 residual neural network，使得做 backpropagation 時的微分為 1 以避免梯度爆炸或消失。

2. Experiment Setup

2.1 Resnet model

參考 https://github.com/kuangliu/pytorch-cifar ，其中 Resnet model 選擇 resnet.py ，最主要的核心內容如下圖。

```
class ResNet(nn.Module):
    def __init__(self, block, num_blocks, num_classes=10):
        super(ResNet, self).__init__()
        self.in_planes = 64

        self.conv1 = nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(64)
        self.layer1 = self._make_layer(block, 64, num_blocks[0], stride=1)
        self.layer2 = self._make_layer(block, 128, num_blocks[1], stride=2)
        self.layer3 = self._make_layer(block, 256, num_blocks[2], stride=2)
        self.layer4 = self._make_layer(block, 512, num_blocks[3], stride=2)
        self.linear = nn.Linear(512*block.expansion, num_classes)

    def _make_layer(self, block, planes, num_blocks, stride):
        strides = [stride] + [1]*(num_blocks-1)
        layers = []
        for stride in strides:
            layers.append(block(self.in_planes, planes, stride))
            self.in_planes = planes * block.expansion
        return nn.Sequential(*layers)

    def forward(self, x):
        out = F.relu(self.bn1(self.conv1(x)))
        out = self.layer1(out)
        out = self.layer2(out)
        out = self.layer3(out)
        out = self.layer4(out)
        out = F.avg_pool2d(out, 4)
        out = out.view(out.size(0), -1)
        out = self.linear(out)
        return out
```

首先是 block 的格式，分為 Basicblock 以及 Bottleneck，前者為2層convolution layers（kernel size 3*3 以及 3*3）後者為3層（kernel size 1*1, 3*3 以及 1*1）。

一開始 new Resnet Class 時，會依據使用者給定的參數，產生相對的四大層 layers，每大層 layer 再根據 block 型態 以及 (stride * num_blocks-1) 生成相對應的 layers 。舉例，Block: basicblock, num_blocks: [2, 2, 2, 2]，則層數為 2*2 + 2*2+ 2*2+ 2*2 + 1(input) + 1(output) = 18。

2.2 CNN model

將Resnet model 轉換為 CNN model，我的作法是將下圖的36行 shortcut 註解。

```
16 class BasicBlock(nn.Module):
17     expansion = 1
18 
19     def __init__(self, in_planes, planes, stride=1):
20         super(BasicBlock, self).__init__()
21         self.conv1 = nn.Conv2d(in_planes, planes, kernel_size=3, s
22         self.bn1 = nn.BatchNorm2d(planes)
23         self.conv2 = nn.Conv2d(planes, planes, kernel_size=3, stri
24         self.bn2 = nn.BatchNorm2d(planes)
25 
26         self.shortcut = nn.Sequential()
27         if stride != 1 or in_planes != self.expansion*planes:
28             self.shortcut = nn.Sequential(
29                 nn.Conv2d(in_planes, self.expansion*planes, kernel_
30                 nn.BatchNorm2d(self.expansion*planes)
31             )
32 
33     def forward(self, x):
34         out = F.relu(self.bn1(self.conv1(x)))
35         out = self.bn2(self.conv2(out))
36         #out += self.shortcut(x)
37         out = F.relu(out)
38         return out
```

2.3 接下來介紹 training 及 testing 的參數

2.3.1 RGB color channel

```
transform_train = transforms.Compose([
    transforms.RandomCrop(32, padding=4),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize((0.4914, 0.4824, 0.4467), (0.2471, 0.2435, 0.2616)),
])

transform_test = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.4914, 0.4824, 0.4467), (0.2471, 0.2435, 0.2616)),
])
```

2.3.2 使用SGD方法或取資料，weight decay = 0.0001，momentum = 0.9

```
optimizer = optim.SGD(net.parameters(), lr=args.lr, momentum=0.9, weight_decay=1e-4)
```

2.3.3 Loss 計算方法為 cross entropy

```
criterion = nn.CrossEntropyLoss()
```

2.4 Resnet layer 層數的參數

```
def ResNet20():
    return ResNet(BasicBlock, [2,2,3,2])

def ResNet34():
    return ResNet(BasicBlock, [3,4,6,3])

def ResNet50():
    return ResNet(Bottleneck, [3,4,6,3])

def ResNet56():
    return ResNet(BasicBlock, [6,6,9,6])
    #return ResNet(Bottleneck, [3,4,9,3])

def ResNet101():
    return ResNet(Bottleneck, [3,4,23,3])

def ResNet110():
    return ResNet(BasicBlock, [9,16,20,9])
    #return ResNet(Bottleneck, [3,4,26,3])
```

3. Result

3.1 Final test error

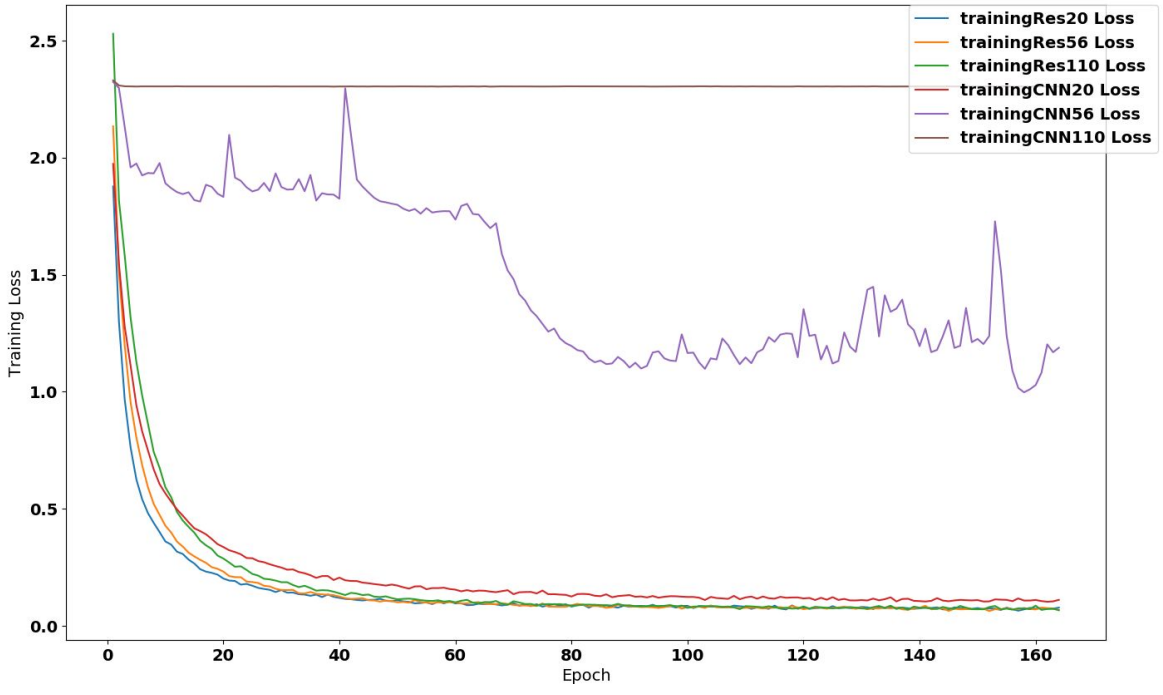
最後結果，Resnet 110 的 test error 最小， CNN 110 的 test error 最大。

```
testingRes20 test error: 8.08326686108
testingRes56 test error: 7.73773704366
testingRes110 test error: 7.46300281706
testingCNN20 test error: 8.91529343872
testingCNN56 test error: 39.0302302229
testingCNN110 test error: 89.6647951919
```

```
testingRes20 accuracy: 91.9167331389%
testingRes56 accuracy: 92.2622629563%
testingRes110 accuracy: 92.5369971829%
testingCNN20 accuracy: 91.0847065613%
testingCNN56 accuracy: 60.9697697771%
testingCNN110 accuracy: 10.3352048081%
```

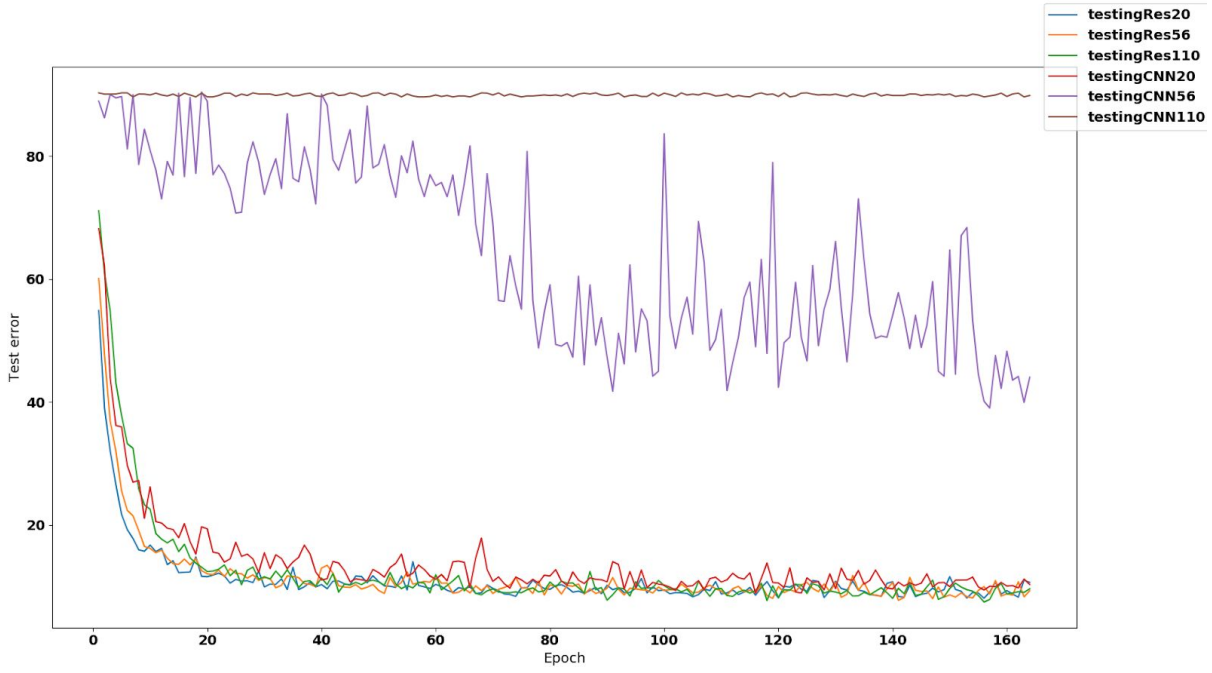
3.2 Training loss curve

下圖為全部的 Loss curve 比較圖，可以看到 CNN 110 的 loss 很平穩，無論參數怎麼更新 loss也降不下來。CNN 56 的 loss 有隨著 epoch 增加而往下降的趨勢，但只到一定的程度就降不下去了。其餘的 training loss 都隨著epoch增加而有往下降的趨勢。

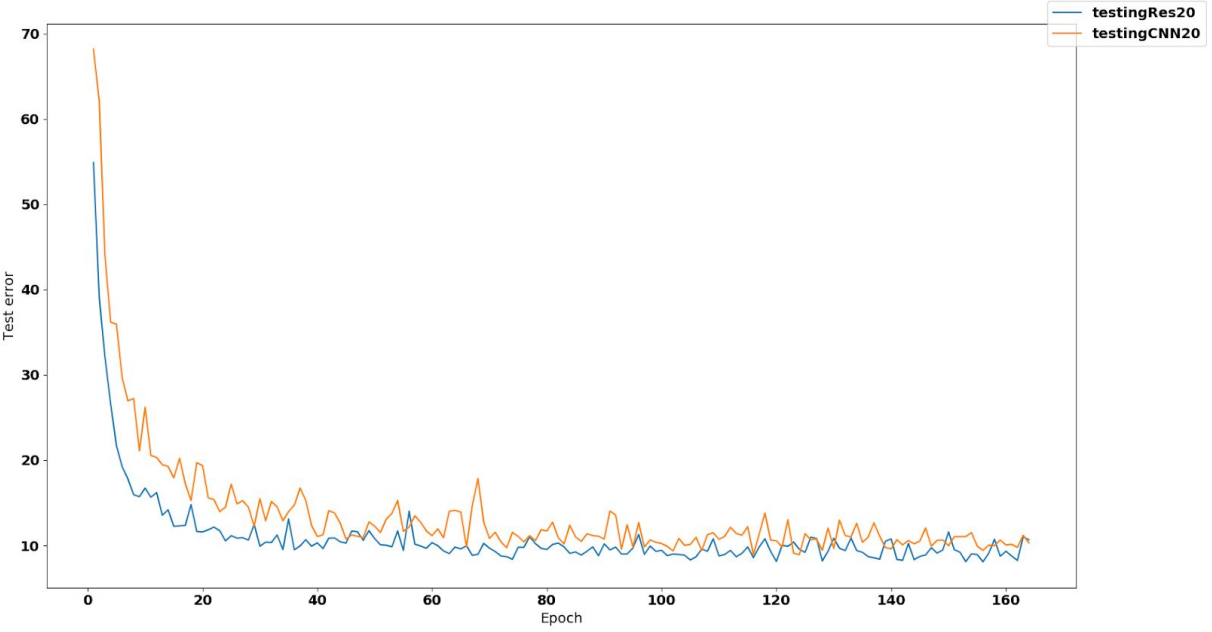


3.3 Test error curve

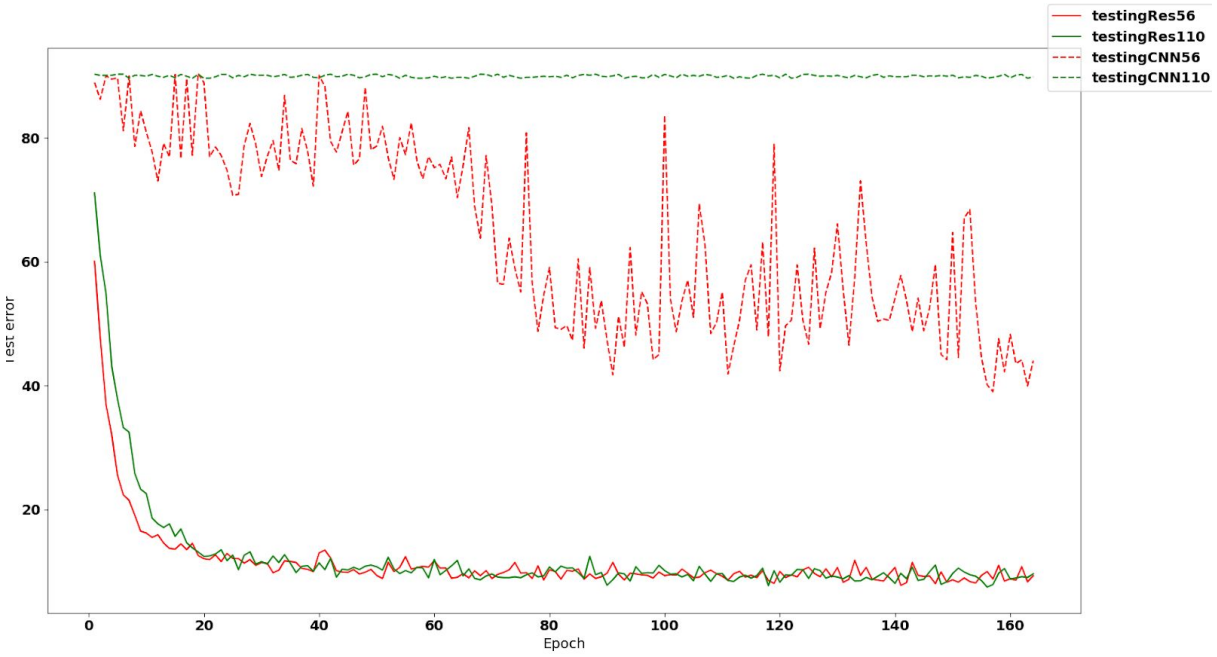
下圖為全部的 Test error curve 比較圖，跟 Loss curve 差不多，可以看到 CNN 110 的 error 很平穩，無論參數怎麼更新 test error 也降不下來。CNN 56 的 test error 有隨著 epoch 增加而往下降的趨勢，但只到一定的程度就降不下去了。其餘的 test error 都隨著 epoch 增加而有往下降的趨勢。



下圖可以看到 CNN 20 與 Resnet 20 兩者的 error 差異不大，Resnet 大概比CNN 結果好一點而已。



下圖可以看到 CNN 56 與 Resnet 56 以及 CNN 110 與 Resnet 110 兩者的 error 差異就很明顯，沒有做 shortcut 的 CNN 結果很顯著的比 Resnet 差。



4. Discussion

遇到最大的問題莫過於設備限制。首先，因為 bottleneck 所需要的 GPU 記憶體龐大，當 layers 數為 56 時剛好需要 6G，而我所擁有的資源，剛好只有 6G 的記憶體，所以當 layers 110 層很明顯不夠用，因此只好改用 basicblock。其次，因為顯示卡型號為 GTX 1060，沒有助教們的顯示卡強大，因此 training 110 layers 的網路大概耗時800分鐘，算是蠻久的，等到發現結果不理想要修改參數或是儲存的結果格式錯誤，再來重新 training 一次會哭死。