Lab 3-2 Report

Introduction:

這次Lab的目的在於讓我們學會實作scalable ConvNets，讓我們可以去對model的各個面向進行擴展，以提升準確率。

Experiment setup:

我是在 win10 下使用 Anaconda 的 jupyter notebook 進行實驗，顯卡是 GTX1070。

Implementation:

我們實做的是 Resnet18 的架構，首先會讓使用者輸入想要的 depth, width 跟 input size，因為原先 resnet18 的架構資料總共會縮小 32 倍，因此我們限定資料大小是 32 的倍數 x 32 的倍數。

```
def main():
    str1 = input('Enter depth(number of layers, default: 2 2 2 2): ')
    layerlist = str1.split()
    layerlist = [int(i) for i in layerlist]

    str2 = input('Enter width(number of channels, default: 64 ): ')
    width = int(str2)

    str3 = input('Enter resolution(input size, default: 224, the number have to be divisible by 32): ')
    input_size = int(str3)
```

然後下方是 Resnet 的前半段內容，主要就是將 layers, width, input_size 作為參數傳入，width 每一個 layers 會乘以 2，跟原先架構相同，input_size 用來計算最後 avg_pool 的 kernel 大小，讓資料最後變成 1X1

```python
class ResNet(nn.Module):

    def __init__(self, block, layers=[2,2,2,2], width=64, input_size=224, num_classes=1000):
        self.inplanes = width
        super(ResNet, self).__init__()
        self.conv1 = nn.Conv2d(3, width, kernel_size=7, stride=2, padding=3,
                               bias=False)
        self.bn1 = nn.BatchNorm2d(width)
        self.relu = nn.ReLU(inplace=True)
        self.maxpool = nn.MaxPool2d(kernel_size=3, stride=2, padding=1)
        self.layer1 = self._make_layer(block, width, layers[0])
        self.layer2 = self._make_layer(block, width*2, layers[1], stride=2)
        self.layer3 = self._make_layer(block, width*2*2, layers[2], stride=2)
        self.layer4 = self._make_layer(block, width*2*2*2, layers[3], stride=2)
        self.avgpool = nn.AvgPool2d(int(input_size/32), stride=1)
        self.fc = nn.Linear(width*2*2*2*block.expansion, num_classes)
```

Results:

首先是維持原本 Resnet18 設置的結果(layers=[2,2,2,2],width=64,input_size=224)

(p.s. 使用的資料是 skewed_food11)

```
Training complete in 34m 8s
Best val Acc: 0.520700
Test set: Top1 Accuracy: 1738/3347 (51 %) , Top3 Accuracy: 2691/3347 (80 %)
class 0 : 131/368 35 %
class 1 : 10/148  6 %
class 10 : 59/231 25 %
class 2 : 331/500 66 %
class 3 : 137/335 40 %
class 4 : 143/287 49 %
class 5 : 308/432 71 %
class 6 : 69/147 46 %
class 7 : 11/96 11 %
class 8 : 143/303 47 %
class 9 : 396/500 79 %
```

接著分別對各個面向進行擴展

Layers = [3,3,3,3]

```
Training complete in 44m 13s
Best val Acc: 0.463848
Test set: Top1 Accuracy: 1586/3347 (47 %) , Top3 Accuracy: 2510/3347 (74 %)
class 0 : 134/368 36 %
class 1 : 0/148  0 %
class 10 : 0/231  0 %
class 2 : 250/500 50 %
class 3 : 135/335 40 %
class 4 : 141/287 49 %
class 5 : 322/432 74 %
class 6 : 54/147 36 %
class 7 : 8/96  8 %
class 8 : 126/303 41 %
class 9 : 416/500 83 %
```

Layers = [3,4,6,3]

```
Training complete in 53m 11s
Best val Acc: 0.435860
Test set: Top1 Accuracy: 1472/3347 (43 %) , Top3 Accuracy: 2352/3347 (70 %)
class 0 : 148/368 40 %
class 1 : 0/148  0 %
class 10 : 0/231  0 %
class 2 : 286/500 57 %
class 3 : 97/335 28 %
class 4 : 99/287 34 %
class 5 : 324/432 75 %
class 6 : 0/147  0 %
class 7 : 0/96  0 %
class 8 : 160/303 52 %
class 9 : 358/500 71 %
```

Input size=256

```
Training complete in 36m 45s
Best val Acc: 0.504665
Test set: Top1 Accuracy: 1669/3347 (49 %) , Top3 Accuracy: 2624/3347 (78 %)
class 0 : 145/368 39 %
class 1 : 3/148  2 %
class 10 : 14/231  6 %
class 2 : 312/500 62 %
class 3 : 151/335 45 %
class 4 : 133/287 46 %
class 5 : 319/432 73 %
class 6 : 59/147 40 %
class 7 : 0/96  0 %
class 8 : 150/303 49 %
class 9 : 383/500 76 %
```

Width(channels) = 128

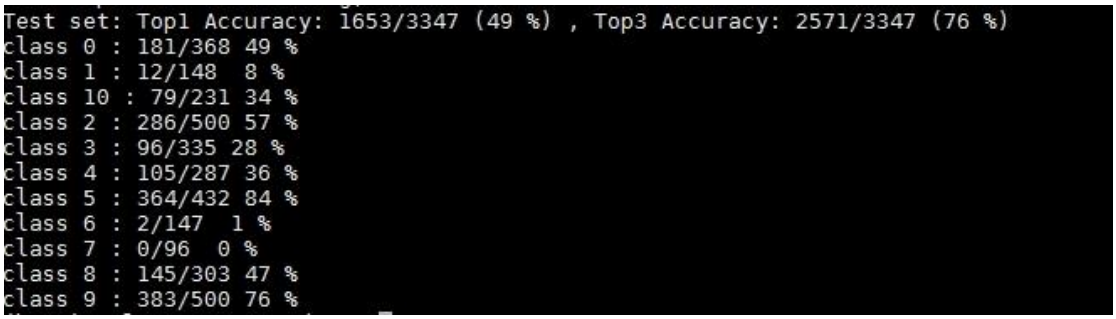
```
Training complete in 69m 18s
Best val Acc: 0.497668
Test set: Top1 Accuracy: 1701/3347 (50 %) , Top3 Accuracy: 2584/3347 (77 %)
class 0 : 155/368 42 %
class 1 : 0/148  0 %
class 10 : 34/231 14 %
class 2 : 289/500 57 %
class 3 : 130/335 38 %
class 4 : 136/287 47 %
class 5 : 329/432 76 %
class 6 : 45/147 30 %
class 7 : 6/96  6 %
class 8 : 177/303 58 %
class 9 : 400/500 80 %
```

Width(channels) = 80

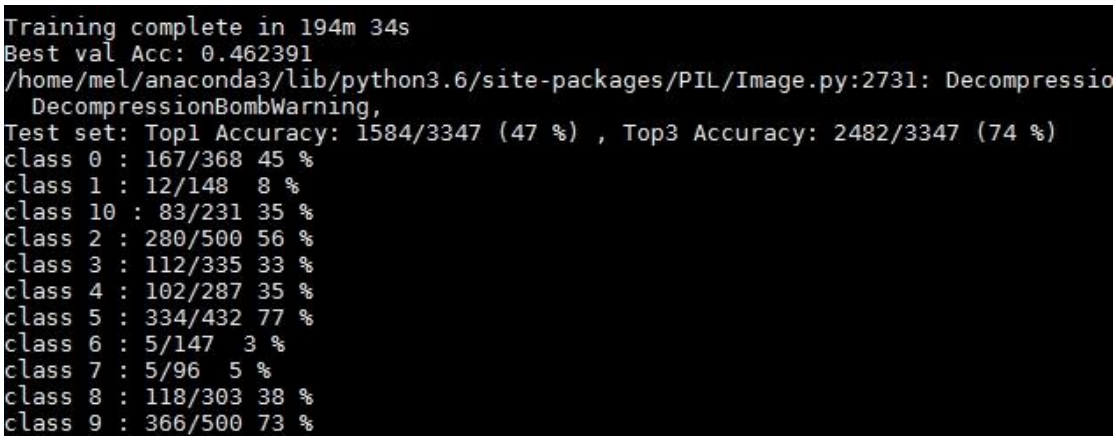
```
Training complete in 40m 59s
Best val Acc: 0.521574
Test set: Top1 Accuracy: 1801/3347 (53 %) , Top3 Accuracy: 2690/3347 (80 %)
class 0 : 123/368 33 %
class 1 : 7/148  4 %
class 10 : 102/231 44 %
class 2 : 301/500 60 %
class 3 : 139/335 41 %
class 4 : 129/287 44 %
class 5 : 331/432 76 %
class 6 : 86/147 58 %
class 7 : 10/96 10 %
class 8 : 159/303 52 %
class 9 : 414/500 82 %
```

可以發現只有當 width=80 時得到較好的結果，因此影響最大的是 width(channel)的大小，而 width=128(乘 2)的結果是較差的，說明我們擴展的幅度不能太大。另外，當提升 layers 跟 width 時會增加訓練的時間，而提升 input size 則不會。

若將 input size 提升到 2 倍 512 發現結果並不會提升太多



提升到 1024 訓練時間極長 結果僅與 512 的效果差不多反而下降了一點



因此可以發現僅調整 input size 對於訓練結果影響不大，小幅度的提升 input size 會提高精確度 但是過多的調升反而會達到反效果

接下來比較同時提升三個參數參數的結果

Layers = 3 3 3 3

Width = 76

Input size = 288

```
Training complete in 85m 59s
Best val Acc: 0.462682
/home/mel/anaconda3/lib/python3.6/site-packages/PIL/Image.py:2731: Decompressi
  DecompressionBombWarning,
Test set: Top1 Accuracy: 1599/3347 (47 %) , Top3 Accuracy: 2539/3347 (75 %)
class 0 : 192/368 52 %
class 1 : 2/148  1 %
class 10 : 0/231  0 %
class 2 : 313/500 62 %
class 3 : 93/335 27 %
class 4 : 114/287 39 %
class 5 : 319/432 73 %
class 6 : 67/147 45 %
class 7 : 3/96  3 %
class 8 : 145/303 47 %
class 9 : 351/500 70 %
```

可以發現結果比單純提高 layers 好 收斂曲線也更平滑

最後根據 efficient net 的 paper 中提出的公式推導出之倍率（1.2/1.5/1.15）套入求得之結果進行訓練

Layer = 2 3 2 2

Width = 76

Input size = 288

```
Training complete in 73m 6s
Best val Acc: 0.530321
/home/mel/anaconda3/lib/python3.6/site-packages/PIL/Image.py:2731: Decompress
  DecompressionBombWarning,
Test set: Top1 Accuracy: 1795/3347 (53 %) , Top3 Accuracy: 2703/3347 (80 %)
class 0 : 133/368 36 %
class 1 : 9/148  6 %
class 10 : 77/231 33 %
class 2 : 296/500 59 %
class 3 : 161/335 48 %
class 4 : 141/287 49 %
class 5 : 332/432 76 %
class 6 : 62/147 42 %
class 7 : 13/96 13 %
class 8 : 171/303 56 %
class 9 : 400/500 80 %
```

可以看到得到了目前最好的結果 但是可能 Resnet18 之模型較小 提升的效果不顯著 需要更大幅度的提高 width 如果使用 80 可能會得到更好的結果