DLP Lab2 Report

1.Introduction:

在這次的 lab 中，需要訓練 Butterfly & Months 的分類類神經網路模型。在作業中，需要完成 data preprocess 和 data loader 的設計，接著實作出 VGG19 和 Resnet50 的 architecture，並且從頭訓練模型的權重。

2.Implement Details:

A. The details of your model (VGG19, ResNet50)

A-1 VGG19:

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

VGG19 是由 16 convolution layer + 3 fully connected layer 所組成，因此，只要照著上圖的 E 依序由上做下來即可，而在經過每一個 convolution layer 和 FC layer 時，會使用 Relu 作為 activate function。

Weight initialization:

For random initialisation (where applicable), we sampled the weights from a normal distribution with the zero mean and 10^{-2} variance. The biases were initialised with zero. It is worth noting that after the paper submission we found that it is possible to initialise the weights without pre-training by using the random initialisation procedure of Glorot & Bengio (2010).

VGG 的作者認為網路權重的 initialization 相當重要，因此，在上圖中，作者也對

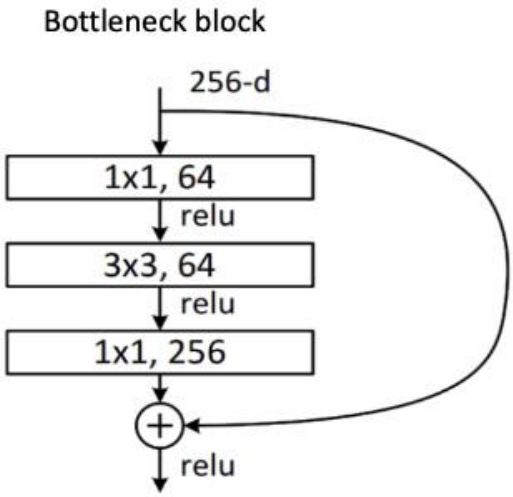
weight 有一系列的設置，但其在最後也指出，Glorot & Bengio 提出的方式更有效。因此在原始的 model 無法訓練起來的狀況下，我選擇了手動設定 weight distribution 的方式，看看 model 是否能夠達到 baseline。

```
80          for m in self.modules():
81              if isinstance(m, (nn.Conv2d, nn.Linear)):
82                  nn.init.kaiming_uniform_(m.weight, mode='fan_in', nonlinearity='relu')
```

Normalization:

在 VGG 的 paper 中，作者在 VGG11 使用了 Local Response Normalization(LRN)的技巧進行 normalization，不過作者發現 LRN 並不能改善 model 的表現，因此，在較深層的網路(B-E)並未使用 normalization 的技巧，而在 **Weight Initialization** 後無法超過 baseline 後，我嘗試添加了 **Batch Normalization**。

A-2 ResNet50



這次 lab 中，需要實作的是 Resnet50，而在 50 layer 以上的 Resnet architecture 中需要實作 Bottleneck。

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

在 Resnet 的架構中，可以發現 conv2_x、conv3_x、conv4_x、conv5_x 是使用類似的相同邏輯去設計，因此我在 Model 的設計上使用了兩個 class，class Bottleneck 主要處理這種類似的構造，而 class Resnet50_net 則將整個完整的架構串在一起。

Class Bottleneck

```
6   class Bottleneck(nn.Module):
7       expansion = 4
8
9       def __init__(self, input_channels, output_channels, stride=1):
10          super(Bottleneck, self).__init__()
11
12          self.conv_1 = nn.Conv2d(
13              input_channels, output_channels, kernel_size=1, stride=1, bias=False)
14          self.bn_1 = nn.BatchNorm2d(output_channels)
15
16          self.conv_2 = nn.Conv2d(output_channels, output_channels,
17                                  kernel_size=3, padding=1, stride=stride, bias=False)
18          self.bn_2 = nn.BatchNorm2d(output_channels)
19
20          self.conv_3 = nn.Conv2d(
21              output_channels, output_channels*self.expansion, kernel_size=1, stride=1, bias=False)
22          self.bn_3 = nn.BatchNorm2d(output_channels*self.expansion)
23          self.relu = nn.ReLU(inplace=True)
24
25          self.downsample = None
26          if stride != 1 or input_channels != output_channels*self.expansion:
27              self.downsample = nn.Sequential(nn.Conv2d(
28                  input_channels, output_channels*self.expansion, kernel_size=1, stride=stride, bias=False),
29                  nn.BatchNorm2d(output_channels*self.expansion),
30              )
```

```
32      def forward(self, x):
33          identity = x.clone()
34
35          x = self.conv_1(x)
36          x = self.bn_1(x)
37          x = self.relu(x)
38
39          x = self.conv_2(x)
40          x = self.bn_2(x)
41          x = self.relu(x)
42
43          x = self.conv_3(x)
44          x = self.bn_3(x)
45
46          if self.downsample is not None:
47              identity = self.downsample(identity)
48
49          x += identity
50          x = self.relu(x)
51          return x
```

其中比較重要的部分是 Line 25-30 行的部分，因為在 bottleneck block 進行 forward 時，需要將 bottleneck 的 output 和 input 相加，因此，需要注意 input shape 和 output shape 是否相等的問題，才需要 25-30 對不相等的情形做處理。

B. The details of your Dataloader

```
45  class BufferflyMothLoader(data.Dataset):
64      def __getitem__(self, index):
84          path = os.path.join(self.root, self.img_name[index])
85          image = Image.open(path).convert('RGB')
86          label = self.label[index]
87          if self.inference == False:
88              # print("Training time")
89              img = train_transform(image)
90          else:
91              # print("inference time")
92              img = test_transform(image)
93          return img, label
```

在這次的作業中，助教提供的 code 已經包含了 BufferflyMonthLoader，因此，僅需要實作**__getitem__**的 function 即可，Line84-85 是讀取圖片的部分，而 line86 則是取得該圖片的 label。而在 Line87-92 行，則是利用 transforms.Compose()將我的 data preprocessing 串起來，形成一個 data preprocessing pipeline，至於分成 train_transform 和 test_transform 的原因，我則留到 3-A 進行說明。

```
train_loader = DataLoader(train_data, batch_size=32, shuffle=True)
valid_loader = DataLoader(valid_data, batch_size=16, shuffle=False)
test_loader = DataLoader(test_data, batch_size=16, shuffle=False)
```

完成好 dataset loading 後，接著需需要使用 **torch.utils.data.DataLoader**，將 dataset 分成一組組的 batch 進行訓練和評估，而在 train loader 的部分，使用 shuffle 會將整個 dataset 進行打亂，根據過往的例子，這能有助於提升 model 的穩健性，不過在這次的 lab 中，助教提供的 train.csv 的順序本身即為亂數排列，因此，在 train loader 有無使用 shuffle，我認為沒有太大的影響。

3.Data Preprocessing

A. How you preprocessed your data:

Train phase

```
7   train_transform = transforms.Compose([
8       transforms.Resize([224, 224]),
9       transforms.RandomHorizontalFlip(),
10      transforms.RandomRotation(degrees=30),
11      transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.1),
12      transforms.ToTensor(),
13      transforms.Normalize(mean=[0.485, 0.456, 0.406],
14                           std=[0.229, 0.224, 0.225]),
15  ])
```

Test phase

```python
test_transform = transforms.Compose([
    transforms.Resize([224, 224]),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225]),
])
```

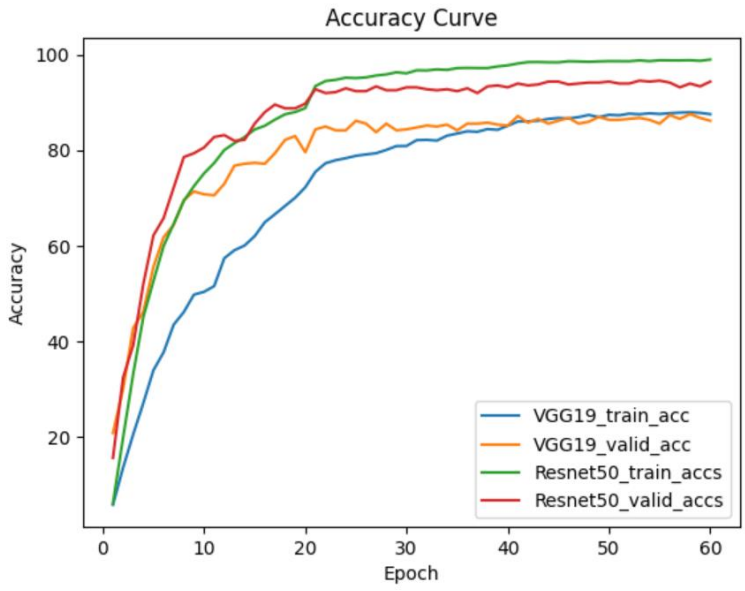
在 train phase 中，首先我會對 Image 進行 resize 到(224*224)，這是因為 VGG19 中包含了 fully connected layer，而 FC layer 的 input channels 會因為 input size 有所改變，因此，必須使用 Resize 的方式統一 dataset 的 input size，而 Resne50 的部分，因為其會多加一層 avg pool，因此，FC layer 的 input channels 不會受到 input size 的影響。接著，則是使用一些影像處理的方式，像是水平翻轉、隨機旋轉以及顏色抖動，這種方式有助於增強 model 的穩健性，接著使用 **transforms.ToTensor**(這一步能夠將 pixel value 轉到[0,1]之間並且使 image shape 從[H, W, C] 轉置成 [C, H, W])，最後再使用 normalization 的手段，其中 mean = [0.485, 0.456, 0.406]和 std = [0.229, 0.224, 0.225]，這個數值是 ImageNet 統計出來的數值，因為考量到自己的 dataset 需要重新計算，我便直接採取這個參數了。

4.Experimental results:

A. The highest testing accuracy

```
---VGG19---
VGG19     |train accuracy:     97.141%|valid accuracy:     87.600%|test accuracy:     90.400%
---Resnet50---
Resnet50  |train accuracy:     99.722%|valid accuracy:     94.600%|test accuracy:     94.600%
```

B. Comparison figures



從 4-A 的圖上可以發現，無論是在 train data 還是 test data 上 Resnet50 的表現皆比 VGG19 好上一截，而從 4-B 的 Comparison figures 中除了可以發現和 4-A 相同的結論:Resnet50 的表現更好，此外在 20 個 epochs 以前時，可以發現 valid accuracy 比 train accuracy 好，我想這是因為 train data 在訓練時會經過 random rotation、color jitter 等預處理方式，因此在 accuracy 上會有些許的損失。

5.Discussion:

在這次的 lab 中，我實作了 VGG19 和 Resnet50 的 model 架構，VGG19 就是基本的深層網路，因此，在實作起來相對的輕鬆，不過我在訓練 VGG19 時，發現怎麼樣調整 training hyper parameter 都發現訓練不起來，因此也使用了各種方式去嘗試，像是更改 data preprocess、weight initialization、learning rate schedule、Clip the gradient norm，不過都沒有明顯的差距，直到最後使用了 Batch Normalization 的方式，model 終於能有效的訓練。而 Resnet50 則是在實作上遇到比較大的麻煩，一開始沒有注意到 downsample 的問題，導致一直出現 shape 相關的 error，但訓練上則沒遇到太大的阻礙，在相同的設定下，Resnet50 輕鬆地打敗了 VGG19 的表現。

6.Reference:

1.https://arxiv.org/abs/1409.1556
2.https://arxiv.org/abs/1512.03385
3.https://proceedings.mlr.press/v9/glorot10a.html
4.https://zhuanlan.zhihu.com/p/53712833