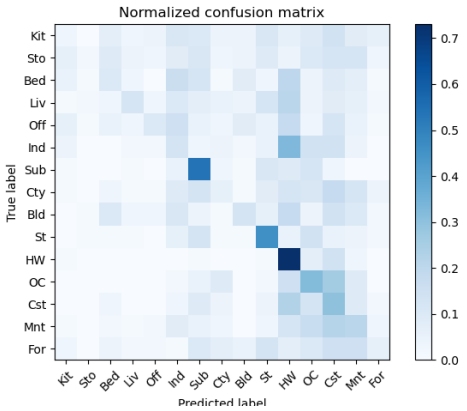
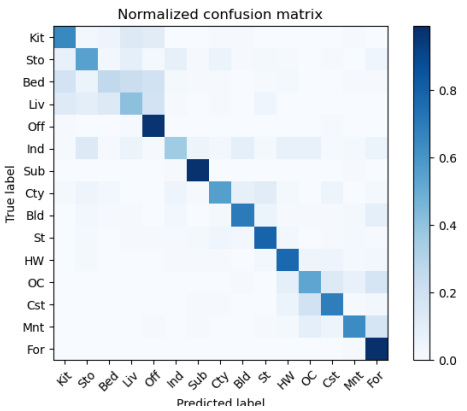
Computer Vision HW2 Report

Student ID: R11543003
Name: 羅正淯

Part 1. (10%)

• Plot confusion matrix of two settings. (i.e. Bag of sift and tiny image) (5%)

Ans:

Tiny image confusion matrix	Bow image confusion matrix
	

• Compare the results/accuracy of both settings and explain the result. (5%)

Ans:

Tiny Image	(myenv) C:\Users\User\OneDrive - NTUMEMS.NET\碩一下\課業\電腦視覺\CV_HW02\hw2\hw2\p1>python p1.py Loading all data paths and labels... Feature: tiny_image Classifier: nearest_neighbor Accuracy = 0.22066666666666668
Bag of SIFT	(myenv) C:\Users\User\OneDrive - NTUMEMS.NET\碩一下\課業\電腦視覺\CV_HW02\hw2\hw2\p1>python p1.py Loading all data paths and labels... Feature: bag_of_sift Classifier: nearest_neighbor Accuracy = 0.6566666666666666

上面兩張 confusion matrix 很明確的顯示 Bag of word 在準確度大於 tiny image 的差異。那在 tiny image 的訓練圖像經過高斯模糊跟圖像零均值化，若不使用此方法，就算是相同類別的圖像，可能因為各自圖片的角度跟形狀，單從像素來看差異很大，取 KNN 時只有 0.15 左右。若有做上述的兩個影像預處理，削弱圖片自己的強邊緣，且讓灰階影像的大小差異不要這麼大，做 KNN 可能有更高的機會把相同類別的圖像算進來，那計算得出的精確度是 0.22。那使用 Bow 的方法，先使用 SIFT 提取 128 維度的

特徵點，使用 Kmeans 把相似的特徵攏聚在一起，像是把他們建立詞彙。然後再將 train 的資料再做一次 sift，再把每張圖片的特徵跟這些詞彙的距離做成值方圖，紀錄每個類別值方圖的表現，因此可以將相似值方圖的圖片做歸類。且 SIFT 可以保留類似物體在不同角度的特徵，在 KNN 做歸類時有較好的表現，表現提升至 0.6566。

Part 2. (25%)

• Report accuracy of both models on the validation set. (2%)

Ans:

將 p2_train.py 訓練出來的 best model 放進 checkpoint 裡，在 p2_inference.py 裡選擇 model type，讓 test 資料庫再進行一次比對，輸出 csv 檔。最後使用 p2_eval.py 將 test 的結果與 ground truth 做比對得到以下兩個模型的結果。

ResNet18	`(base) C:\Users\User\OneDrive - NTUMEMS.NET\碩一下\課業\電腦視覺\CV_HW02\hw2\hw2\p2>python p2_eval.py` `Accuracy = 0.8594`
MyNet	`(base) C:\Users\User\OneDrive - NTUMEMS.NET\碩一下\課業\電腦視覺\CV_HW02\hw2\hw2\p2>python p2_eval.py` `Accuracy = 0.8404`

• Print the network architecture & number of parameters of both models. What is the main difference between ResNet and other CNN architectures? (5%)

Ans:

資料來源: https://towardsdatascience.com/the-w3h-of-alexnet-vggnet-resnet-and-inception-7baaaecccc96

ResNet18

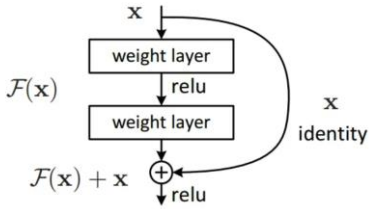
ResNet18 - Structural Details

#	Input Image			output			Layer	Stride	Pad	Kernel		in	out	Param
1	227	227	3	112	112	64	conv1	2	1	7	7	3	64	9472
	112	112	64	56	56	64	maxpool	2	0.5	3	3	64	64	0
2	56	56	64	56	56	64	conv2-1	1	1	3	3	64	64	36928
3	56	56	64	56	56	64	conv2-2	1	1	3	3	64	64	36928
4	56	56	64	56	56	64	conv2-3	1	1	3	3	64	64	36928
5	56	56	64	56	56	64	conv2-4	1	1	3	3	64	64	36928
6	56	56	64	28	28	128	conv3-1	2	0.5	3	3	64	128	73856
7	28	28	128	28	28	128	conv3-2	1	1	3	3	128	128	147584
8	28	28	128	28	28	128	conv3-3	1	1	3	3	128	128	147584
9	28	28	128	28	28	128	conv3-4	1	1	3	3	128	128	147584
10	28	28	128	14	14	256	conv4-1	2	0.5	3	3	128	256	295168
11	14	14	256	14	14	256	conv4-2	1	1	3	3	256	256	590080
12	14	14	256	14	14	256	conv4-3	1	1	3	3	256	256	590080
13	14	14	256	14	14	256	conv4-4	1	1	3	3	256	256	590080
14	14	14	256	7	7	512	conv5-1	2	0.5	3	3	256	512	1180160
15	7	7	512	7	7	512	conv5-2	1	1	3	3	512	512	2359808
16	7	7	512	7	7	512	conv5-3	1	1	3	3	512	512	2359808
17	7	7	512	7	7	512	conv5-4	1	1	3	3	512	512	2359808
	7	7	512	1	1	512	avg pool	7	0	7	7	512	512	0
18	1	1	512	1	1	1000	fc					512	1000	513000
							Total							11,511,784

MyNet

```
self.SEQ1 = nn.Sequential(
    nn.Conv2d(3, 64, kernel_size = 3, padding = 1),
    nn.ReLU(),
    nn.BatchNorm2d(64),
    nn.Dropout(0.1))

self.SEQ2 = nn.Sequential(
    nn.Conv2d(64, 64, kernel_size = 3, padding = 1),
    nn.ReLU(),
    nn.BatchNorm2d(64),
    nn.MaxPool2d(2, 2),
    nn.Dropout(0.1))

self.SEQ3 = nn.Sequential(
    nn.Conv2d(64, 128, kernel_size=3, padding=1),
    nn.ReLU(),
    nn.BatchNorm2d(128))

self.SEQ4 = nn.Sequential(
    nn.Conv2d(128, 128, kernel_size = 3, padding=1),
    nn.ReLU(),
    nn.BatchNorm2d(128),
    nn.MaxPool2d(2, 2),
    nn.Dropout(0.5))

self.SEQ5 = nn.Sequential(
    nn.Conv2d(128, 256, kernel_size = 3, padding=1),
    nn.ReLU(),
    nn.BatchNorm2d(256),
    nn.Dropout(0.5))

self.SEQ6 = nn.Sequential(
    nn.Conv2d(256, 256, kernel_size = 3, padding=1),
    nn.ReLU(),
    nn.BatchNorm2d(256),
    nn.AvgPool2d(8, 8))

self.fc = nn.Linear(256, 10)
```

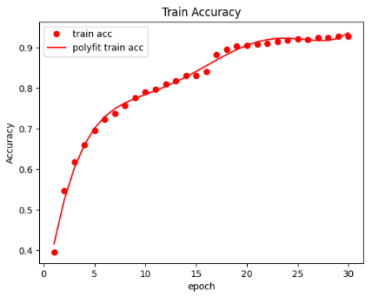
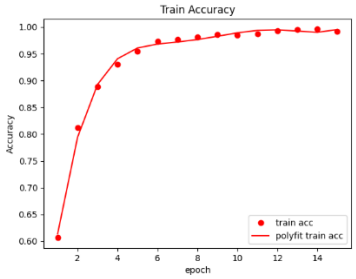
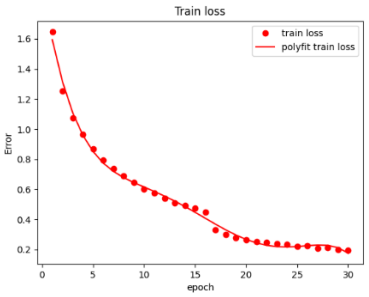
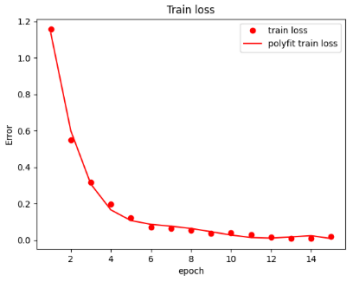
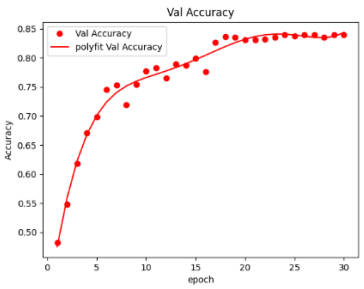
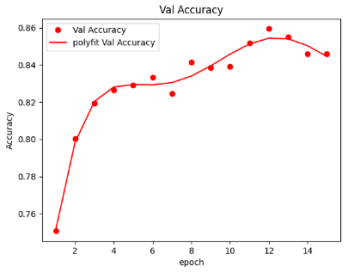
可以看到 ResNet18 的參數約有一千一百萬個，他先是由一個 7x7 的捲積再經最大池化，再進到四層各有四個捲積的結構，在進到每一層結構，都會與前面的資料相加

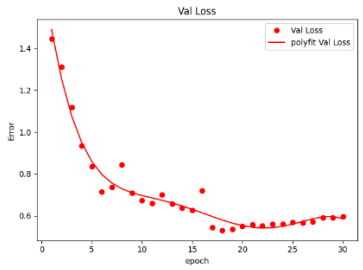
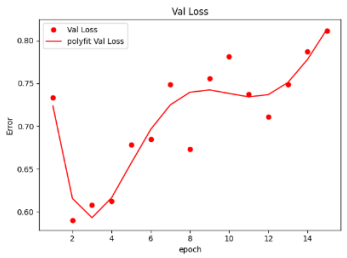


像是建立高速公路讓後前面的資訊，不會因為多層結構使訊息消失，反而可以保留梯度，整個架構重複這個殘差塊，避免梯度消失。其他傳統 CNN 結構，單純的捲積、池化、正則等方法，雖然與 ResNet 提取方法都一樣，但少了殘差塊的訊息流動，使得梯度容易因為多層消失，也是過往認為 CNN 結過不需要超過三層的因素。

• Plot four learning curves (loss & accuracy) of the training process (train/validation) for both models. Total 8 plots. (8%)

Ans:

	Mynet	ResNet18
Train Acc		
Train Loss		
Val Acc		

Val Loss		

• Briefly describe what method do you apply on your best model? (e.g. data augmentation, model architecture, loss function, etc) (10%)
Ans:

參考上課老師及作業建議將部分 train 的影像作水平翻轉、調整飽和度還有高斯模糊的處理，避免擬合曲線過度貼合 train。

```
if split == 'train':
    transform = transforms.Compose([
        transforms.RandomHorizontalFlip(p=0.3),
        transforms.ColorJitter(saturation=0.1),
        transforms.GaussianBlur(3, sigma=(1)),
        transforms.Resize((32,32)),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
    ])
else: # 'val' or 'test'
    transform = transforms.Compose([
        transforms.Resize((32,32)),
        # we usually don't apply data augmentation on test or val data
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
    ])
```

也因為 train 的資料庫是有 augmentation 的，使得一開始 train acc 比 val acc 還要低的情形，因為 validation 的資料庫是沒有做 augmentation 的，使得在驗證上一開始有較高的準確度，但經過數個 epoch，擬合曲線會持續向 train 資料庫做權重的調整，因此 train acc 會逐漸超過 val acc。

```
super(ResNet18, self).__init__()

weights = models.ResNet18_Weights.IMAGENET1K_V1
self.resnet = models.resnet18(weights)
self.resnet.conv1=nn. Conv2d(3, 64, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2), bias=False)
self.resnet.maxpool=Identity()
self.resnet.fc = nn.Linear(self.resnet.fc.in_features, 10)
```

那在我的 resnet18，有參考作業給定的更改預設定的結構，將第一層的捲積核函數跟 stride 縮小，並將最大池化層移除，在 15 次的 epoch 內，就讓準確度大幅的從 0.75 提升到 0.85。那損失函數跟優化器則沒有做額外的更動。

那在 MyNet 則是稍微參考網路他人的結構，做了六層的捲積，並在中間引入

BatchNormal 做資料的正則化，還有做一次的最大池化層。那在 train 資料庫相同情形下，將 batch size 縮小一倍希望每次 epoch 做更多次的 Iteration，並將 epoch 稍微提高一倍，大概是 30 次左右，由 MyNet 的 val acc 可以看到在 epoch 15 次時，就已達到 0.80 的要求，但要達到 0.84 則幾乎多花一倍的 epoch 次數才達到。但時間仍控制在十分鐘內達到作業的要求。