Computer Vision HW2 Report

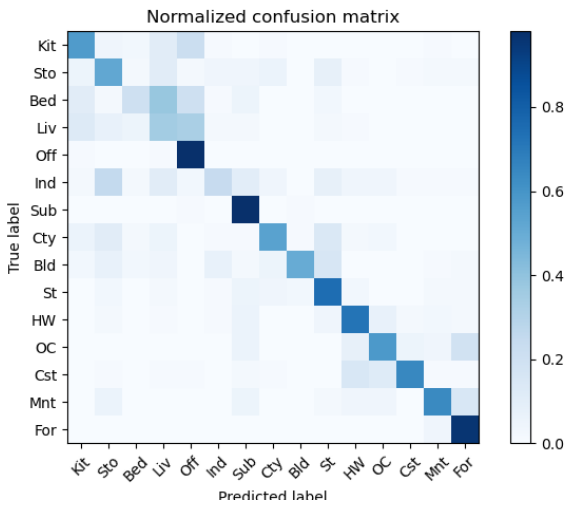
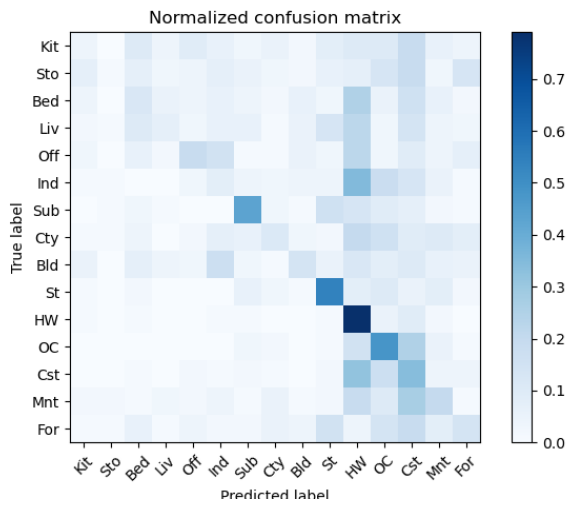
Student ID: B09901075

Name: 陳駿瑋

Part 1. (10%)

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

Ans:

bag of sift	tiny image
	

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

Ans:

Setting	Accuracy
tiny image	24.5%
bag of sift	61.2%

首先關於 tiny image 的部分，基本上就是把圖片 resize - flatten - normalize，當成圖片 feature。然後用 cdist 算 test feature 和 train feature 間的距離，最後用 kNN 決定結果。主要決定 accuracy 的因素是 kNN 中的 k 值和算距離使用的方法。cdist 的部分，我試過 hint 中的 minkowski distance (1-norm and 2-norm and 3-norm)，結果最佳的是 minkowski distance (1-norm)， accuracy 結果大概會落在 0.21~0.25 左右 (k = 7)。另外 k 值則是從 1~11，從 1 開始 accuracy 會慢慢提升，但大概到 8 左右會慢慢開始下降，不過雖然有變化，也都大概是 0.2 ~ 0.25 左右。總結來說，在我測試結果中，此方法可以讓 accuracy 落在大概 20%~25%，最後選擇的組合是 minkowski distance (1-norm) 和 k = 7。

再來關於 bag of sift 的部分，首先使用 dsift 取得 SIFT feature，然後用 kmeans 分群，得到 vocab.pkl (像是圖片字典)。接下來取得預測結果的部分，一樣先取得圖片的 SIFT feature，與 vocab 中的 feature 計算距離，統計出距離最近的 feature，最後再使用 kNN 取得結果。在此方法中，我認為影響 accuracy 的因素主要有 dsift 中的 step 和與 tiny image 一樣，計算距離的方法和 k 值。(其實還有一個是在取 SIFT feature 時，可以不用全部使用，加快訓練速度，同時也不會使 accuracy 下降太

多，但因為我選取的 step 花的時間還可以接受，所以我是直接使用全部 feature) 而 step 我測試過 2 和 3，step = [2,2]時的準確率較高，畢竟 step 越小，取 feature 越仔細，能提升 accuracy 還滿符合直覺，只是使用較小的 step 真的會大幅增加產出 vocab.pkl 的時間。另外 k 值的選取結果與 tiny image 類似，提高會讓 accuracy 上升，但太高的 k 可能會讓 accuracy 下降。(k = 1 和 k = 7 時，accuracy 大概差到 3%) 最後是我認為最重要的 distance 計算方法的選擇，我測試了 minkowski (1-norm and 2-norm)，其中在固定 k = 7 時，minkowski (1-norm)的 accuracy 達到 0.612，是最佳結果，其他兩個會使 accuracy 降到 0.51 (2-norm)和 0.47 (3-norm)，差別非常大，而針對 1-norm 表現最好，我認為是因為 SIFT feature 是由 histogram 統計出來，所以計算距離時，也就使用 1-norm 不同 dimension 直接相減相加就好。

而針對 tiny image 和 bag of sift 的比較，畢竟 tiny image 只是單純 resize 圖片，沒有分析圖片，而 bag of sift 有分析圖片，取出 sift feature，並且做分類，所以 bag of sift 比 tiny image 高不少算是可預期的結果。

Part 2. (25%)

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

Model	Accuracy
Mynet (follow LeNet in lec06.pdf, p.98)	63.82%
Resnet18 (with revised architecture)	91.28%

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

相關的圖在文字下方。

Resnet 跟一般的 CNN 主要差在使用 residual/skip connections，從 Resnet 的 source code 可以發現，每個 basic block 的輸出都是 output (input 經過 basic block 中的 Conv, BatchNorm, Relu 後的結果) + identity (input 或是 input 經過 downsample 層) 後再經過 Relu 層。使用 residual/skip connections 是因為如果 model 太深，梯度往前傳時不斷減小，可能會導致梯度變零 (gradient vanish)，而透過 residual/skip connections，可以解決此問題，也讓優化參數的過程更輕鬆。

Resnet18 (with revised architecture)	Mynet (follow LeNet in lec06.pdf, p.98)

```
ResNet18(
  (resnet): ResNet(
    (conv1): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
    (maxpool): Identity()
    (layer1): Sequential(
      (0): BasicBlock(
        (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
      (1): BasicBlock(
        (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (layer2): Sequential(
      (0): BasicBlock(
        (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (downsample): Sequential(
          (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
          (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        )
      )
      (1): BasicBlock(
        (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (layer3): Sequential(
      (0): BasicBlock(
        (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (downsample): Sequential(
          (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
          (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        )
      )
      (1): BasicBlock(
        (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (layer4): Sequential(
      (0): BasicBlock(
        (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (downsample): Sequential(
          (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
          (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        )
      )
      (1): BasicBlock(
        (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
    (fc): Linear(in_features=512, out_features=10, bias=True)
  )
)
```

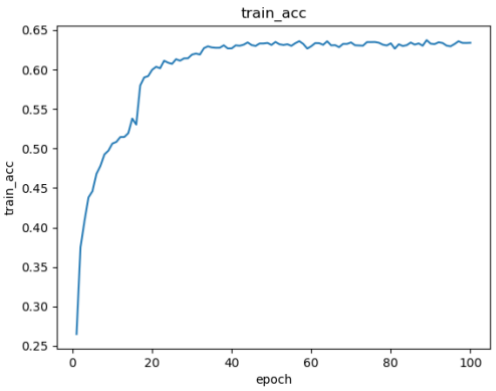
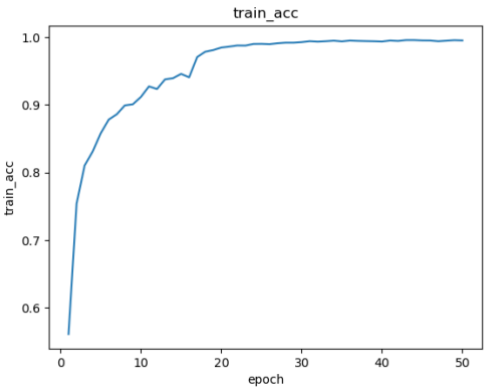
```
total parameters: 11173962
trainable parameters: 11173962
```

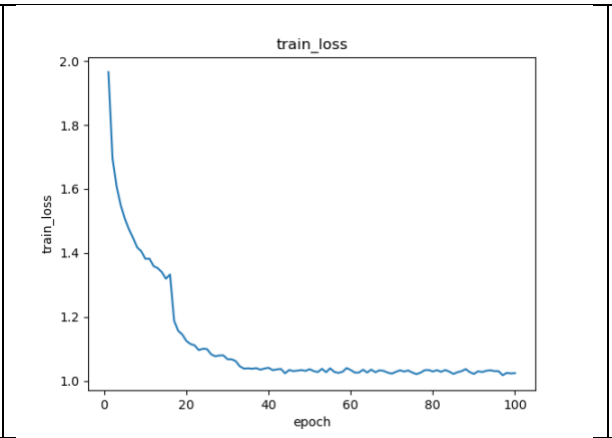
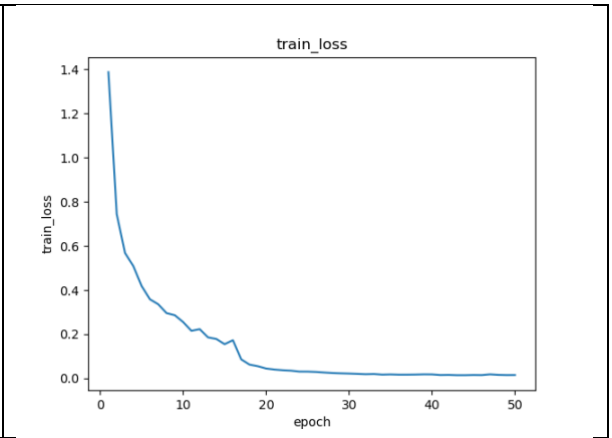
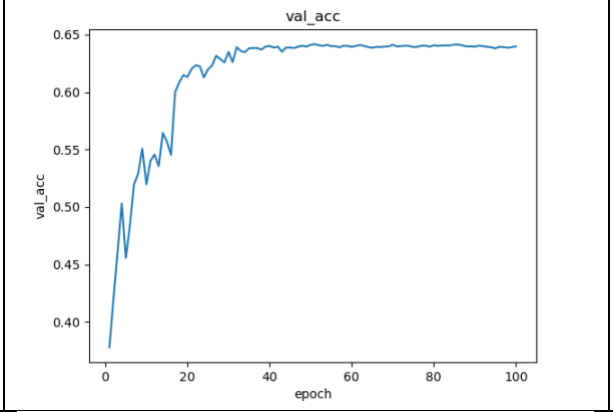
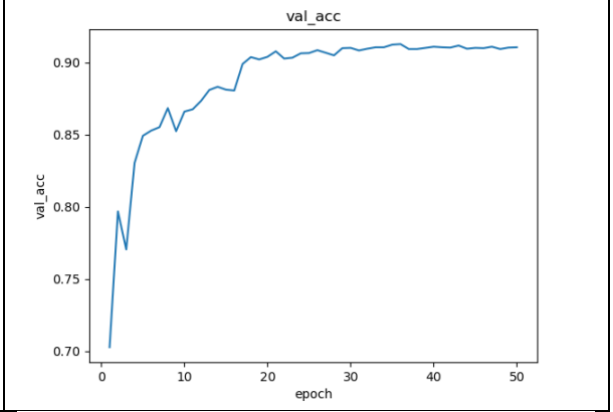
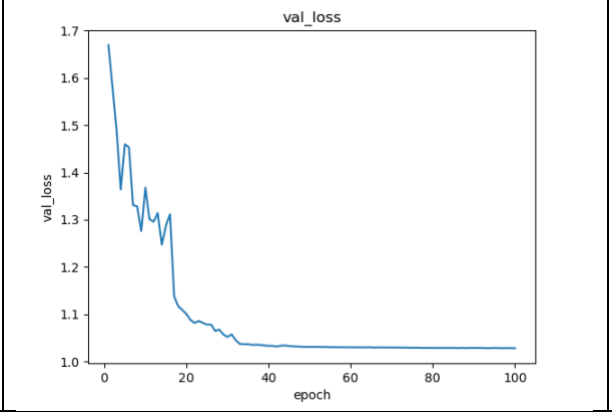
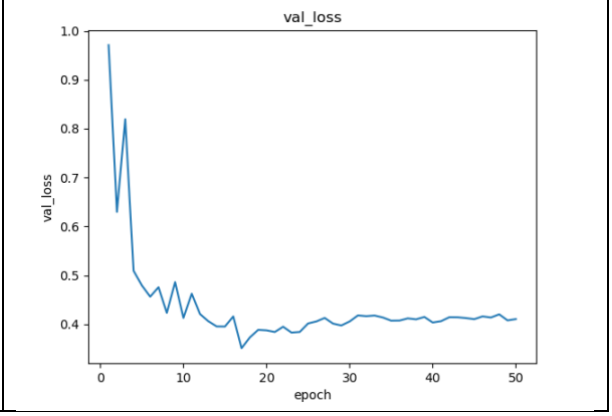
```
MyNet(
  (Conv): Sequential(
    (0): Conv2d(3, 6, kernel_size=(5, 5), stride=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
    (4): ReLU()
    (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (FC): Sequential(
    (0): Linear(in_features=400, out_features=120, bias=True)
    (1): ReLU()
    (2): Linear(in_features=120, out_features=84, bias=True)
    (3): ReLU()
    (4): Linear(in_features=84, out_features=10, bias=True)
  )
)
```

```
total parameters: 62006
trainable parameters: 62006
```

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

Ans:

	Mynet	Resnet18
Train accuracy		

Train loss		
Valid accuracy		
Valid loss		

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

Ans:

1. **Data augmentation**

   原本有任意測試 Rotation, ColorJitter, GrayScale 等等，發現越多 transform 不一定較佳，因為 transform 種類很多，選取幾個也是問題，所以我查詢相關資料，最後選擇以下組合。

```
transforms.Pad(4),
transforms.RandomHorizontalFlip(),
transforms.RandomCrop(32),
```

2. **Model architecture**

   我最好的 model 達到 0.9128 的 accuracy，是使用根據 hint 修改的 resnet18，修改部分如下：

```
# (batch_size, 3, 32, 32)
self.resnet = models.resnet18(pretrained=True)
# (batch_size, 512)
self.resnet.conv1 = nn.Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
self.resnet.maxpool = Identity()
self.resnet.fc = nn.Linear(self.resnet.fc.in_features, 10)
# (batch_size, 10)
```

3. Loss function

optimizer = torch.optim.SGD(model.parameters(), lr=cfg.lr, momentum=0.9, weight_decay=1e-6)

4. Parameters

```
epochs     = 50           # train how many epochs
batch_size = 32           # batch size for dataloader, original = 32
use_adam   = False        # Adam or SGD optimizer
lr         = 1e-2         # learning rate, original = 1e-2
milestones = [16, 32, 45] # reduce learning rate at 'milestones' epochs
```

雖然這裡 epoch 有到 50，但實際 train 時，大概 18 就能達到 9 成的 accuracy。