

電腦視覺-HW2 Report

R08921053 電機丙研二 梁峻瑋

#Part1

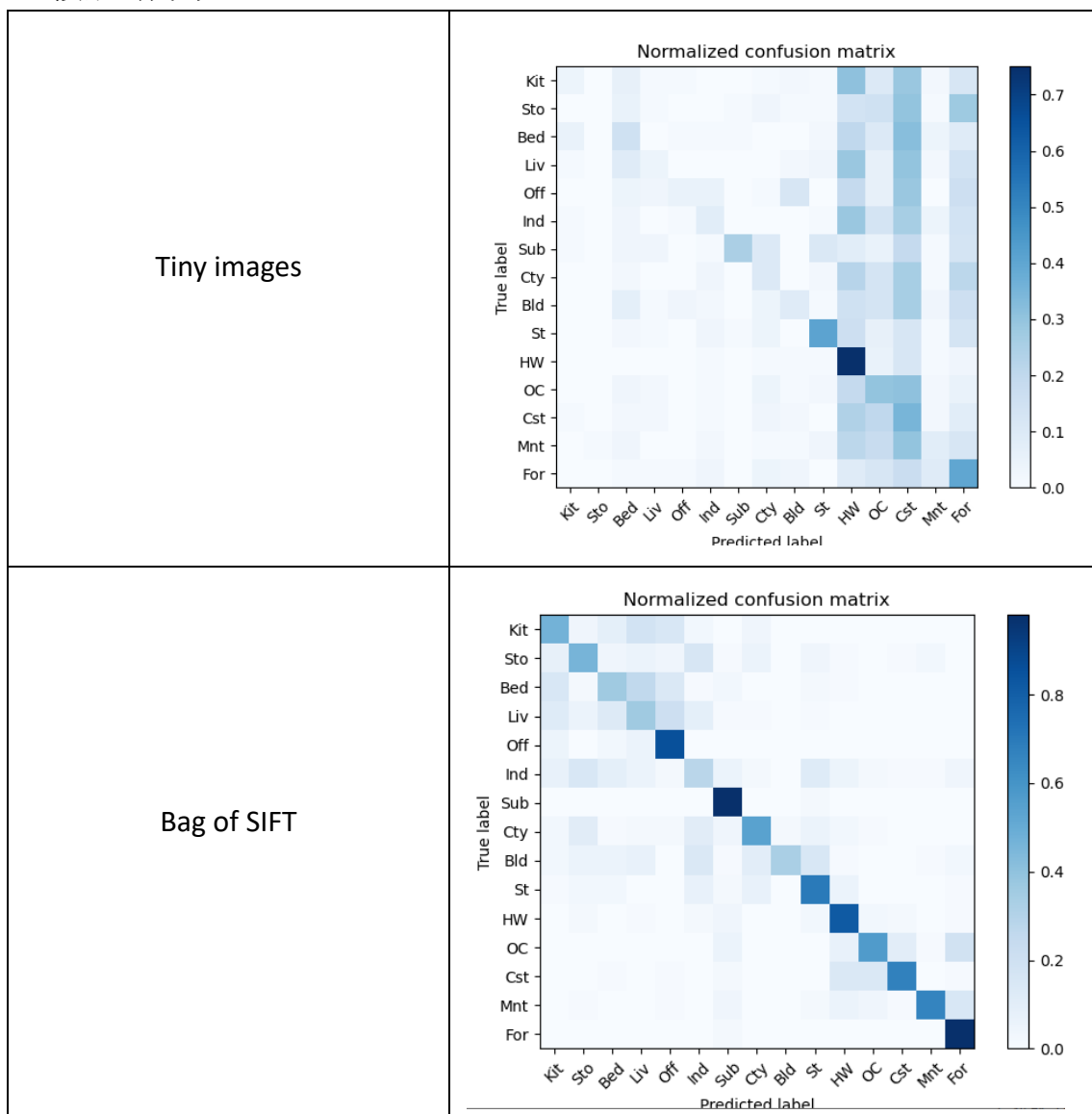
1. (5%) Report accuracy of two settings

直接回答我最終的準確度:

Tiny images	20.86%
Bag of SIFT	62.53%

2. (5%) Plot confusion matrix of two settings

直接給出我的 confusion matrix:



3. (5%) Compare the results of both settings and explain the result

(i) Tiny images

先附上表格:

	k=1	k=2	k=3	k=4	k=5
準確率	22.6%	18.8%	19.73%	20.6%	20.86%
	k=6	k=7	k=8	k=9	k=10
準確率	21%	20.6%	21.33%	21.33%	21.2%

在這個部分, 首先是 `get_tiny_images.py`, 我使用 `cv2.resize()` 來壓縮圖片, 再用 `tiny_img2D.flatten()` 來攤平成一維向量. 再來是 `nearest_neighbor_classify.py`, 我使用 `distance.cdist()` 來算距離, 再用 `np.argmaxpartition(row, k)[:k]` 及 `mode` 來取得預測結果.

影響準確率的關鍵, 應該是 kNN 的 k 值選擇, 因此在上表中列出 k=1~5 的準確度. 直接解讀圖表, k 值的選擇大約會影響 2% 的準確度. 另外, k 值越大或 k 值越小, 好像也不代表準確度就會單調變化. 但能觀察到, k 值大到一個程度以後, 影響的效果就會趨近於無(收斂).

(ii) Bag of SIFT

先附上表格:

//Fix k=4 for kNN

Image_feats dsift setting vocab.pkl dsift setting	step=(3,3)	step=(1,1) mod 5	step=(1,1) mod 2
step=(5,5) mod 2	50%	51.6%	
step=(3,3)	51.13%	52.06%	
step=(1,1) mod 2	52.86%	53.46%	53.33%

*mod 2 代表每兩筆資料取一筆(每兩條 row 取一條)

//固定 53.46% 的數據, 把 kNN 的 norm 從 L2 改成 L1, 再跑 k 值

	k=1	k=2	k=3	k=4	k=5
準確率	60.53%	56.33%	60.73%	61.53%	62.53%
	k=6	k=7	k=8	k=9	k=10
準確率	61.8%	61.53%	61.46%	61.33%	60.8%

在這個部分，首先是 `build_vocabulary.py`，使用 `dsift()` 取得 feature，再根據 `mod N` 的設定來篩選 feature(如果需要)，最後送進 `kmeans()`，完成 `vocab.pkl`。再來是 `get_bags_of_sift.py`，也是用 `dsift()` 來取得 feature，再用 `distance.cdist()` 跟 `vocab.pkl` 裡面的資料庫 feature 計算距離，最後再統計最近的資料庫 feature，輸出。最後則是前面(i)部分提過的 KNN。

影響準確度的關鍵，除了 kNN 的 k 值選擇，我認為主要有三點：

- (1) `dsift()` 的 step size，通常越小越高準確度
- (2) 相同 step size 時，`mod 2` 篩選 feature 會提升準確度。*或者其他質數
- (3) 最後，把 metric setting 從 L2 norm 改成 L1，準確度大幅上升約 6~8%

關於(1)，很直覺的，對每一張圖片更仔細的挑選 feature，以及給出更多的 feature，兩方面顯然都有助於提升準確度。

關於(2)，很顯然的解釋是，由於原先的 feature 太多，會造成 `overfitting`，因此需要篩選過 feature，才會讓準確度上升。另一方面，這也提升了程式速度！

關於(3)，直觀的看法是，在我們這個專題中，量測距離的對象是兩個 histogram，也就是對於資料庫 feature 的統計次數。此時，舉個例子來看，假設只有兩個特徵值，(1,0)應該與(3,0)，(2,1)有一樣的距離，才比較符合常理。甚至於我會認為(3,0)應該更接近(1,0)，因為他們同樣都沒使用到第二個特徵值。

#Part2

1. (4%) Print the network architectures & number of parameters of both models

直接給出 network information

conv

```
ConvNet(
  (cnn): Sequential(
    (0): Conv2d(1, 6, kernel_size=(5, 5), stride=(1, 1))
    (1): ReLU(inplace=True)
    (2): AvgPool2d(kernel_size=2, stride=2, padding=0)
    (3): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
    (4): ReLU(inplace=True)
    (5): AvgPool2d(kernel_size=2, stride=2, padding=0)
  )
  (fc): Sequential(
    (0): Linear(in_features=256, out_features=120, bias=True)
    (1): ReLU(inplace=True)
    (2): Linear(in_features=120, out_features=84, bias=True)
    (3): ReLU(inplace=True)
    (4): Linear(in_features=84, out_features=10, bias=True)
  )
)
```

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 6, 24, 24]	156
ReLU-2	[-1, 6, 24, 24]	0
AvgPool2d-3	[-1, 6, 12, 12]	0
Conv2d-4	[-1, 16, 8, 8]	2,416
ReLU-5	[-1, 16, 8, 8]	0
AvgPool2d-6	[-1, 16, 4, 4]	0
Linear-7	[-1, 120]	30,840
ReLU-8	[-1, 120]	0
Linear-9	[-1, 84]	10,164
ReLU-10	[-1, 84]	0
Linear-11	[-1, 10]	850

Total params: 44,426
Trainable params: 44,426
Non-trainable params: 0

mynet

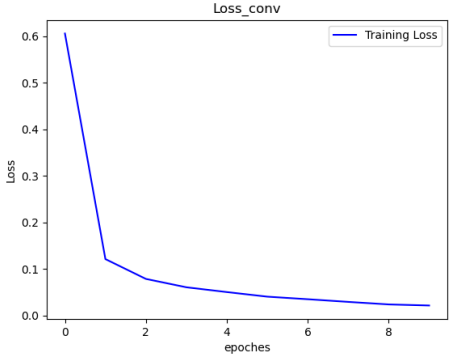
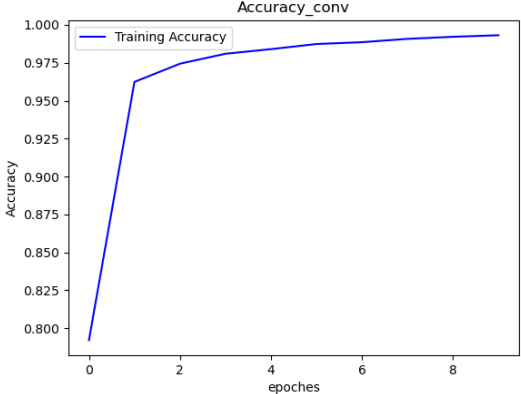
```
MyNet(
  (cnn): Sequential(
    (0): Conv2d(1, 6, kernel_size=(5, 5), stride=(1, 1))
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
    (4): ReLU(inplace=True)
    (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Conv2d(16, 40, kernel_size=(3, 3), stride=(1, 1))
    (7): ReLU(inplace=True)
  )
  (fc): Sequential(
    (0): Dropout(p=0.5, inplace=False)
    (1): Linear(in_features=160, out_features=120, bias=True)
    (2): ReLU(inplace=True)
    (3): Dropout(p=0.5, inplace=False)
    (4): Linear(in_features=120, out_features=84, bias=True)
    (5): ReLU(inplace=True)
    (6): Linear(in_features=84, out_features=10, bias=True)
  )
)
```

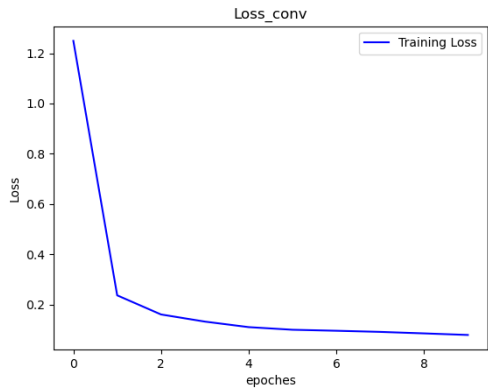
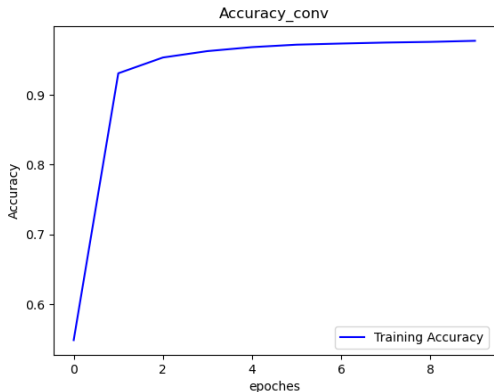
Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 6, 24, 24]	156
ReLU-2	[-1, 6, 24, 24]	0
MaxPool2d-3	[-1, 6, 12, 12]	0
Conv2d-4	[-1, 16, 8, 8]	2,416
ReLU-5	[-1, 16, 8, 8]	0
MaxPool2d-6	[-1, 16, 4, 4]	0
Conv2d-7	[-1, 40, 2, 2]	5,800
ReLU-8	[-1, 40, 2, 2]	0
Dropout-9	[-1, 160]	0
Linear-10	[-1, 120]	19,320
ReLU-11	[-1, 120]	0
Dropout-12	[-1, 120]	0
Linear-13	[-1, 84]	10,164
ReLU-14	[-1, 84]	0
Linear-15	[-1, 10]	850

Total params: 38,706
Trainable params: 38,706
Non-trainable params: 0

2. (10%) Plot the learning curve (loss, accuracy) of the training process(train/validation).

直接在下一頁給出 picture

	conv																						
Learning curve loss	 <p>Loss_conv</p> <p>Training Loss</p> <table border="1"><thead><tr><th>epoques</th><th>Loss</th></tr></thead><tbody><tr><td>0</td><td>0.6</td></tr><tr><td>1</td><td>0.12</td></tr><tr><td>2</td><td>0.08</td></tr><tr><td>3</td><td>0.07</td></tr><tr><td>4</td><td>0.06</td></tr><tr><td>5</td><td>0.055</td></tr><tr><td>6</td><td>0.05</td></tr><tr><td>7</td><td>0.048</td></tr><tr><td>8</td><td>0.045</td></tr><tr><td>9</td><td>0.042</td></tr></tbody></table>	epoques	Loss	0	0.6	1	0.12	2	0.08	3	0.07	4	0.06	5	0.055	6	0.05	7	0.048	8	0.045	9	0.042
epoques	Loss																						
0	0.6																						
1	0.12																						
2	0.08																						
3	0.07																						
4	0.06																						
5	0.055																						
6	0.05																						
7	0.048																						
8	0.045																						
9	0.042																						
Learning curve accuracy	 <p>Accuracy_conv</p> <p>Training Accuracy</p> <table border="1"><thead><tr><th>epoques</th><th>Accuracy</th></tr></thead><tbody><tr><td>0</td><td>0.80</td></tr><tr><td>1</td><td>0.965</td></tr><tr><td>2</td><td>0.975</td></tr><tr><td>3</td><td>0.98</td></tr><tr><td>4</td><td>0.985</td></tr><tr><td>5</td><td>0.988</td></tr><tr><td>6</td><td>0.99</td></tr><tr><td>7</td><td>0.992</td></tr><tr><td>8</td><td>0.993</td></tr><tr><td>9</td><td>0.995</td></tr></tbody></table>	epoques	Accuracy	0	0.80	1	0.965	2	0.975	3	0.98	4	0.985	5	0.988	6	0.99	7	0.992	8	0.993	9	0.995
epoques	Accuracy																						
0	0.80																						
1	0.965																						
2	0.975																						
3	0.98																						
4	0.985																						
5	0.988																						
6	0.99																						
7	0.992																						
8	0.993																						
9	0.995																						
Training process Train+validate	<pre>Epoch: 0 Training batch index: 500, train loss: 1.193516, acc: 0.580 Training batch index: 1000, train loss: 0.718621, acc: 0.751 Training batch index: 1250, train loss: 0.605715, acc: 0.792 Validing batch index: 313, train loss: 0.172809, acc: 0.948 Epoch: 1 Training batch index: 500, train loss: 0.135365, acc: 0.958 Training batch index: 1000, train loss: 0.124463, acc: 0.962 Training batch index: 1250, train loss: 0.120876, acc: 0.962 Validing batch index: 313, train loss: 0.089463, acc: 0.972 Epoch: 2 Training batch index: 500, train loss: 0.080326, acc: 0.974 Training batch index: 1000, train loss: 0.080430, acc: 0.973 Training batch index: 1250, train loss: 0.078285, acc: 0.974 Validing batch index: 313, train loss: 0.071141, acc: 0.978 Epoch: 3 Training batch index: 500, train loss: 0.065688, acc: 0.980 Training batch index: 1000, train loss: 0.060978, acc: 0.981 Training batch index: 1250, train loss: 0.060297, acc: 0.981 Validing batch index: 313, train loss: 0.066589, acc: 0.981 Epoch: 4 Training batch index: 500, train loss: 0.047176, acc: 0.985 Training batch index: 1000, train loss: 0.050030, acc: 0.984 Training batch index: 1250, train loss: 0.049990, acc: 0.984 Validing batch index: 313, train loss: 0.066737, acc: 0.980 Epoch: 5 Training batch index: 500, train loss: 0.039628, acc: 0.987 Training batch index: 1000, train loss: 0.039492, acc: 0.988 Training batch index: 1250, train loss: 0.040117, acc: 0.987 Validing batch index: 313, train loss: 0.061635, acc: 0.984 Epoch: 6 Training batch index: 500, train loss: 0.032768, acc: 0.989 Training batch index: 1000, train loss: 0.034272, acc: 0.989 Training batch index: 1250, train loss: 0.034609, acc: 0.989 Validing batch index: 313, train loss: 0.057744, acc: 0.984 Epoch: 7 Training batch index: 500, train loss: 0.026173, acc: 0.992 Training batch index: 1000, train loss: 0.029146, acc: 0.991 Training batch index: 1250, train loss: 0.028868, acc: 0.991 Validing batch index: 313, train loss: 0.051250, acc: 0.986 Epoch: 8 Training batch index: 500, train loss: 0.025150, acc: 0.992 Training batch index: 1000, train loss: 0.023668, acc: 0.992 Training batch index: 1250, train loss: 0.023433, acc: 0.992 Validing batch index: 313, train loss: 0.054216, acc: 0.986 Epoch: 9 Training batch index: 500, train loss: 0.020965, acc: 0.994 Training batch index: 1000, train loss: 0.021624, acc: 0.993 Training batch index: 1250, train loss: 0.021077, acc: 0.993 Validing batch index: 313, train loss: 0.061598, acc: 0.984</pre>																						

	mynet
Learning curve loss	 <p>Loss_conv</p> <p>Training Loss</p>
Learning curve accuracy	 <p>Accuracy_conv</p> <p>Training Accuracy</p>
Training process Train+validate	<pre> Epoch: 0 Training batch index: 500, train loss: 2.251719, acc: 0.153 Training batch index: 1000, train loss: 1.477986, acc: 0.461 Training batch index: 1250, train loss: 1.249213, acc: 0.549 Validing batch index: 313, train loss: 0.186166, acc: 0.947 Epoch: 1 Training batch index: 500, train loss: 0.277152, acc: 0.918 Training batch index: 1000, train loss: 0.249551, acc: 0.927 Training batch index: 1250, train loss: 0.236934, acc: 0.931 Validing batch index: 313, train loss: 0.093487, acc: 0.972 Epoch: 2 Training batch index: 500, train loss: 0.172730, acc: 0.950 Training batch index: 1000, train loss: 0.164211, acc: 0.952 Training batch index: 1250, train loss: 0.160885, acc: 0.953 Validing batch index: 313, train loss: 0.090356, acc: 0.974 Epoch: 3 Training batch index: 500, train loss: 0.136673, acc: 0.960 Training batch index: 1000, train loss: 0.133506, acc: 0.962 Training batch index: 1250, train loss: 0.132487, acc: 0.963 Validing batch index: 313, train loss: 0.058308, acc: 0.983 Epoch: 4 Training batch index: 500, train loss: 0.111589, acc: 0.968 Training batch index: 1000, train loss: 0.111013, acc: 0.968 Training batch index: 1250, train loss: 0.110491, acc: 0.968 Validing batch index: 313, train loss: 0.075153, acc: 0.978 Epoch: 5 Training batch index: 500, train loss: 0.103671, acc: 0.971 Training batch index: 1000, train loss: 0.102203, acc: 0.971 Training batch index: 1250, train loss: 0.100031, acc: 0.972 Validing batch index: 313, train loss: 0.052122, acc: 0.985 Epoch: 6 Training batch index: 500, train loss: 0.097409, acc: 0.974 Training batch index: 1000, train loss: 0.098307, acc: 0.973 Training batch index: 1250, train loss: 0.096047, acc: 0.974 Validing batch index: 313, train loss: 0.047901, acc: 0.987 Epoch: 7 Training batch index: 500, train loss: 0.084227, acc: 0.977 Training batch index: 1000, train loss: 0.090181, acc: 0.975 Training batch index: 1250, train loss: 0.091561, acc: 0.975 Validing batch index: 313, train loss: 0.055939, acc: 0.984 Epoch: 8 Training batch index: 500, train loss: 0.092855, acc: 0.974 Training batch index: 1000, train loss: 0.087948, acc: 0.976 Training batch index: 1250, train loss: 0.085591, acc: 0.976 Validing batch index: 313, train loss: 0.052829, acc: 0.987 Epoch: 9 Training batch index: 500, train loss: 0.080828, acc: 0.977 Training batch index: 1000, train loss: 0.078449, acc: 0.977 Training batch index: 1250, train loss: 0.079351, acc: 0.977 Validing batch index: 313, train loss: 0.044157, acc: 0.988 </pre>

3. (6%) Compare the results of both model and explain the result

(i) conv

在這個部分, 我實作了 LeNet-5 模型, 也就是 2 層 conv2D layer(&AvgPool)以及 3 層的 Full Connected layer.

初步觀察 train data set 和 test data set, 可以發現 train data set 的資料非常全面, 涵蓋了 test data set 裡面的各類資料, 因此準確度應該可以達到非常高. 實際上, 在訓練 10 個回合後, 也確實在 test data 上達到 98.4%的準確度.

(ii) Mynet

在這個部分, 我修改並優化了 LeNet-5 模型, 主要有三個部分:

- (1) 把 AvgPool 改成效果較好的 MaxPool
- (2) 加了第三層的 conv2D layer (參考自網路上的建議)
- (3) 加入 Dropout, 避免 overfitting 問題

首先, 根據以往的經驗, 以及老師的說法, MaxPool 往往會有比 AvgPool 更好的效果, 因此我們做了這部分的優化. 另外, 在這個專題中, 由於 train data set 太過全面, 因此我們反而會擔心 overfitting 的問題. 所以, 做了(3)的優化, 而準確度也確實提升了一些.