

Computer Vision HW2 Report

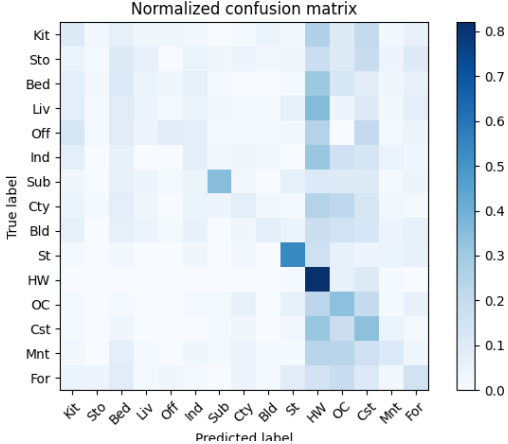
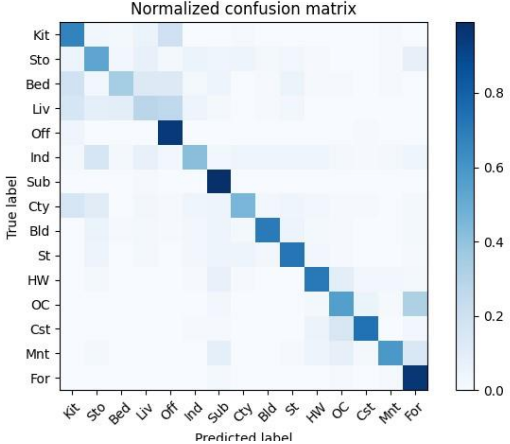
Student ID: R11521201

Name: 廖沁柔

Part 1. (10%)

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

Ans:

Tiny Image	Bag of SIFT
	
Accuracy	
21.73%	64.20%

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

Ans:

(1) Tiny Image (Accuracy)

➤ For different k values:

k=1	k=2	k=3	k=4	k=5
21.73%	21.00%	21.67%	21.67%	21.73%
k=6	k=7	k=8	k=9	k=10
20.87%	21.47%	21.00%	21.27%	21.27%

The key to affecting accuracy in this part lies in the choice of the k-value.

However, there is no clear trend to interpret changes in the setting of the k-value. It can only be concluded that smaller k-values (k=1~5) may lead to higher accuracy in this part.

(2) Bag of SIFT (Accuracy)

➤ Dense SIFT (dsift) setting of step size (Fix k = 1 for KNN):

step=(1,1)	step=(3,3)	step=(5,5)
61.87%	61.87%	61.87%

Changing the step size refers to the pixel distance between one point and the next when extracting SIFT features. From the above table presented, it was found that using a step size of (5,5) can speed up the process without sacrificing performance. This suggests that there may be redundant information present in the images.

➤ **Fix step=(5,5); Change different k values:**

k=1	k=2	k=3	k=4	k=5
61.87%	61.00%	61.80%	63.13%	62.80%
k=6	k=7	k=8	k=9	k=10
63.60%	64.20%	64.00%	63.33%	63.00%

In this part, k=7 achieves the highest accuracy. It is speculated that k=7 finds a balance point, effectively resisting noise while maintaining sufficient sensitivity to recognize different categories.

➤ **Fix k=7 and metric = 'minkowski'; Change p values:**

p = 1	p = 2
64.20%	56.60%

In this part, when p=1, the Minkowski distance becomes the Manhattan distance, and when p=2, it becomes the Euclidean distance. It has been observed that the accuracy is higher with p=1, indicating that the Manhattan distance is more effective for this dataset. This might be because the Manhattan distance is less sensitive to outliers and better handles high-dimensional data, making it more advantageous in situations where different dimensions contribute unevenly.

Part 2. (25%)

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

Ans:

Model	Accuracy
ResNet18	89.28%
MyNet	82.28%

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

Ans:

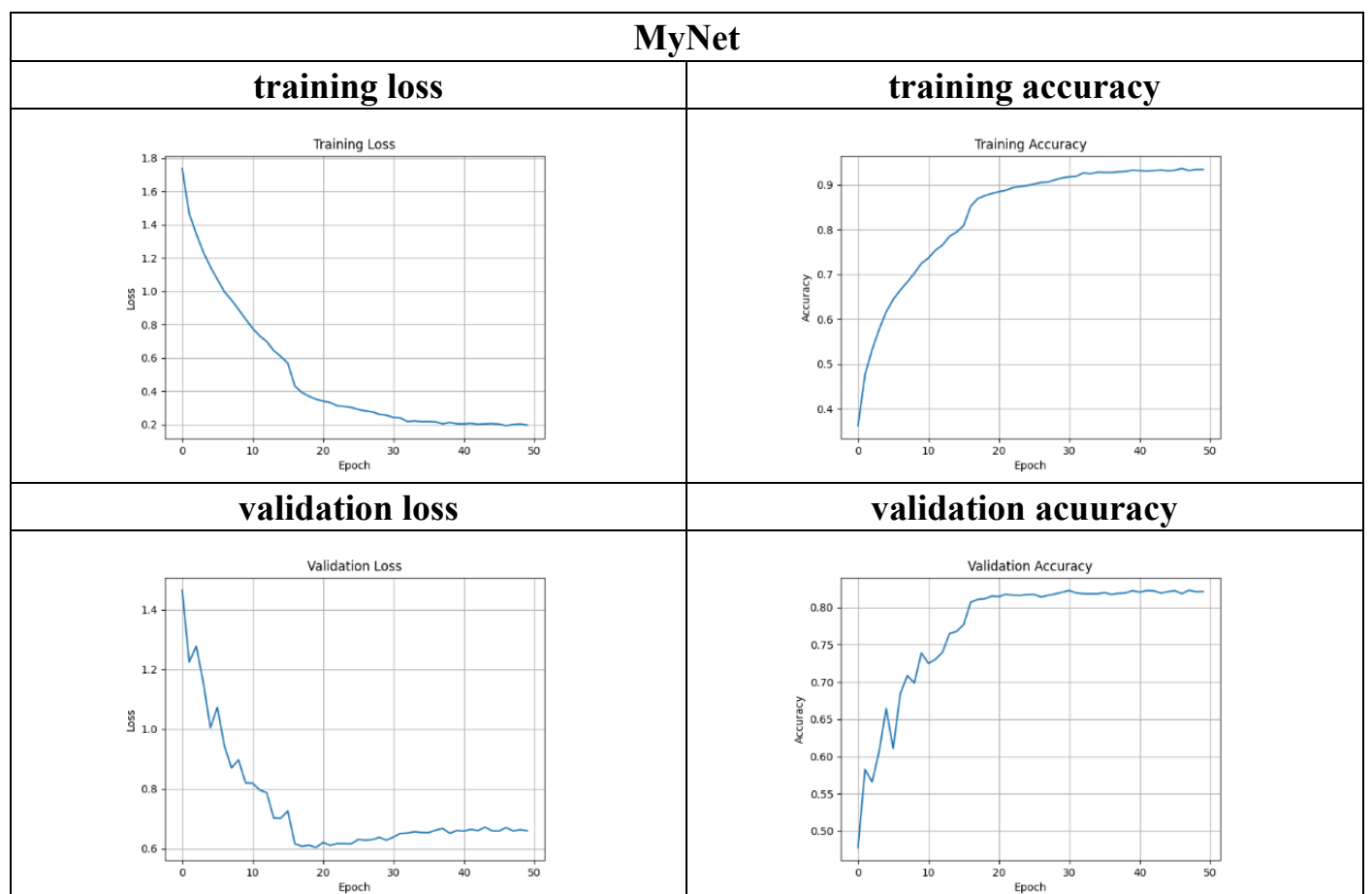
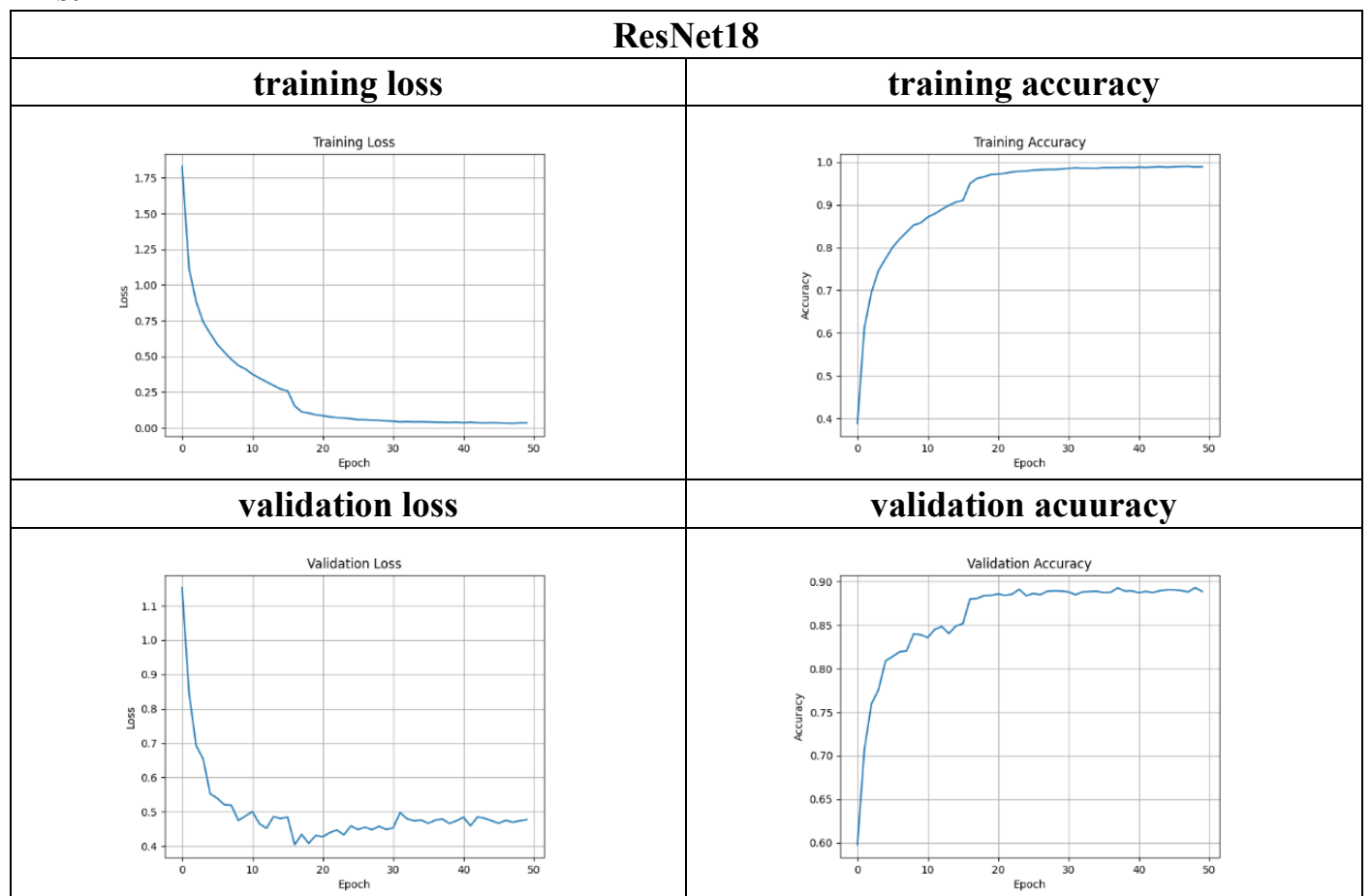
Model	Network architecture & # of parameters
ResNet18	<pre> ===== Layer (type:depth-idx) Output Shape Param # ===== └─ResNet: 1-1 [-1, 10] -- └─Conv2d: 2-1 [-1, 64, 32, 32] 1,728 └─BatchNorm2d: 2-2 [-1, 64, 32, 32] 128 └─ReLU: 2-3 [-1, 64, 32, 32] -- └─Identity: 2-4 [-1, 64, 32, 32] -- └─Sequential: 2-5 [-1, 64, 32, 32] -- └─BasicBlock: 3-1 [-1, 64, 32, 32] 73,984 └─BasicBlock: 3-2 [-1, 64, 32, 32] 73,984 └─Sequential: 2-6 [-1, 128, 16, 16] -- └─BasicBlock: 3-3 [-1, 128, 16, 16] 230,144 └─BasicBlock: 3-4 [-1, 128, 16, 16] 295,424 └─Sequential: 2-7 [-1, 256, 8, 8] -- └─BasicBlock: 3-5 [-1, 256, 8, 8] 919,040 └─BasicBlock: 3-6 [-1, 256, 8, 8] 1,180,672 └─Sequential: 2-8 [-1, 512, 4, 4] -- └─BasicBlock: 3-7 [-1, 512, 4, 4] 3,673,088 └─BasicBlock: 3-8 [-1, 512, 4, 4] 4,720,640 └─AdaptiveAvgPool2d: 2-9 [-1, 512, 1, 1] -- └─Linear: 2-10 [-1, 10] 5,130 ===== Total params: 11,173,962 Trainable params: 11,173,962 Non-trainable params: 0 Total mult-adds (M): 582.80 ===== Input size (MB): 0.01 Forward/backward pass size (MB): 8.50 Params size (MB): 42.63 Estimated Total Size (MB): 51.14 ===== </pre>
MyNet	<pre> ===== Layer (type:depth-idx) Output Shape Param # ===== └─Sequential: 1-1 [-1, 512, 2, 2] -- └─Conv2d: 2-1 [-1, 64, 32, 32] 1,792 └─BatchNorm2d: 2-2 [-1, 64, 32, 32] 128 └─LeakyReLU: 2-3 [-1, 64, 32, 32] -- └─MaxPool2d: 2-4 [-1, 64, 16, 16] -- └─Conv2d: 2-5 [-1, 128, 16, 16] 73,856 └─BatchNorm2d: 2-6 [-1, 128, 16, 16] 256 └─LeakyReLU: 2-7 [-1, 128, 16, 16] -- └─MaxPool2d: 2-8 [-1, 128, 8, 8] -- └─Conv2d: 2-9 [-1, 256, 8, 8] 295,168 └─BatchNorm2d: 2-10 [-1, 256, 8, 8] 512 └─LeakyReLU: 2-11 [-1, 256, 8, 8] -- └─MaxPool2d: 2-12 [-1, 256, 4, 4] -- └─Conv2d: 2-13 [-1, 512, 4, 4] 1,180,160 └─BatchNorm2d: 2-14 [-1, 512, 4, 4] 1,024 └─LeakyReLU: 2-15 [-1, 512, 4, 4] -- └─MaxPool2d: 2-16 [-1, 512, 2, 2] -- └─Sequential: 1-2 [-1, 10] -- └─Linear: 2-17 [-1, 1024] 2,098,176 └─ReLU: 2-18 [-1, 1024] -- └─Linear: 2-19 [-1, 512] 524,800 └─ReLU: 2-20 [-1, 512] -- └─Dropout: 2-21 [-1, 512] -- └─Linear: 2-22 [-1, 10] 5,130 ===== Total params: 4,181,002 Trainable params: 4,181,002 Non-trainable params: 0 Total mult-adds (M): 65.20 ===== Input size (MB): 0.01 Forward/backward pass size (MB): 1.89 Params size (MB): 15.95 Estimated Total Size (MB): 17.85 ===== </pre>

➤ **Main difference – Total parameters**

The ResNet18 model is more complex, potentially offering better performance on more complex tasks, but it also requires more computational resources. The custom CNN is simpler and may train faster, but it might not perform as well on tasks that benefit from a deeper architecture.

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

Ans:



• **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

- **Random Horizontal Flip**

This transformation mirrors the image across the vertical axis. It is especially beneficial for training models that need to understand objects irrespective of their horizontal orientation. By flipping images randomly, the model learns to recognize objects even when their orientation changes, thus enhancing its ability to generalize better from the training data to real-world scenarios.

- **Random Rotation**

Rotating images randomly introduces rotational variance to the training set. This helps in training the model to recognize objects and patterns even when they are not perfectly aligned with the image grid, which is common in real-life captures. It reduces the model's sensitivity to the orientation of objects, thereby making it more robust to positional changes.

- **Color Jitter**

Adjusting the brightness, contrast, and saturation of images randomly can mimic various lighting conditions. This is particularly useful because it trains the model to perform well under different lighting conditions and color settings. By doing so, it reduces the model's dependency on specific lighting conditions for recognizing features, making it more versatile and capable of operating effectively across diverse environments.

(2) Model architecture

- **ResNet18**

Pre-trained weight of ResNet18: The model starts with a pre-trained ResNet18 backbone. Using a pre-trained model leverages transfer learning, where the model has already learned a significant amount of relevant features from a large and diverse dataset (commonly ImageNet). This approach can significantly improve model performance, especially when training data is limited.

Modified First Convolutional Layer: The first standard convolutional layer of the original ResNet18 has been modified to have a kernel size of 3x3, a stride of 1, and padding of 1 with no bias. This modification ensures that the spatial dimensions of the output feature map are the same as the input, unlike the original ResNet18, which reduces the dimensions early in the network. This can help in preserving more fine-grained details in the input images.

BatchNorm2d: The batch normalization layer normalizes the output of the first convolutional layer, helping to speed up training by stabilizing the learning process. It also helps in reducing the internal covariate shift which often affects deep networks.

ReLU: It introduces non-linearity to the model, helping it to learn more complex patterns in the data.

Identity MaxPooling: Instead of using a traditional max pooling layer which reduces the spatial dimensions of the feature maps, this model employs an identity mapping for the pooling operation. The identity function simply passes the data through without change, preserving the feature map dimensions. This is particularly useful for maintaining resolution for tasks where finer detail is crucial.

Final Fully Connected Layer: The last layer in ResNet18 has been replaced with a new fully connected layer that reduces the output features to 10, corresponding to the number of target classes in your

specific task.

- **MyNet**

BatchNorm2d: Utilizes batch normalization to enhance the stability of the model. This technique normalizes the activations of the previous layer at each batch, maintaining the mean output close to 0 and the output standard deviation close to 1. This can lead to improvements in training speed and overall model performance.

LeakyReLU: Acts as the activation function, which helps to prevent issues related to the dying ReLU problem. Unlike the standard ReLU that outputs zero for any negative input, LeakyReLU allows a small, non-zero, constant gradient when the unit is not active and the input is less than zero.

MaxPool2d: Reduces the spatial dimensions (width and height) of the input feature maps, decreasing the amount of computation and model parameters. This also helps to make the detection of features invariant to scale and orientation changes.

Dropout: A regularization technique that involves randomly setting a fraction of input units to 0 at each update during training time, which helps to prevent overfitting. The neurons thus learn to function independently, leading to a more robust network.

Fully Connected Layers: These layers follow the convolutional and pooling layers, transforming the 2D feature maps into a flat vector. The network then uses these fully connected layers to derive the final output classifications. They are vital for combining features effectively to make predictions.

(3) Loss Function

- **Cross-Entropy**

The cross-entropy loss function is highly effective for classification tasks because it targets the optimization of predictive accuracy by penalizing deviations between the predicted probability distribution and the actual distribution of labels. Its use of the natural logarithm ensures moderate gradients, which prevents extreme updates during training and supports stable convergence, making it a robust choice for improving a model's ability to classify correctly.