

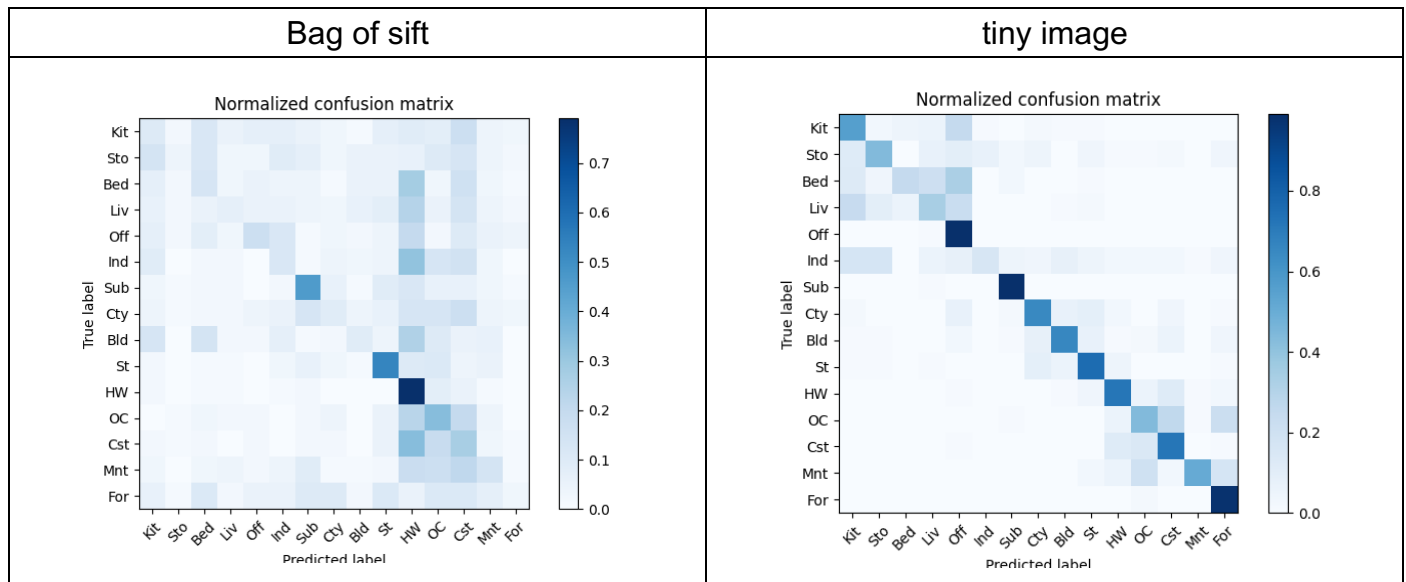
Computer Vision HW2 Report

Student ID: R12942159

Name: 邱亮茗

Part 1. (10%)

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



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

Conclusion:

- The Bag of SIFT method is clearly superior to the Tiny Image method for this classification task.
- Tiny images lose too much information, leading to poor performance.
- Bag of SIFT captures essential features, making it a better choice for image classification.

1. Tiny Image

- This method uses raw pixel values as features, which are highly sensitive to lighting, scale, and orientation variations.
- Since tiny images are down sampled representations, they lack detailed information, making it difficult to differentiate classes.
- The poor accuracy (22.06%) confirms that this method is too simplistic for robust classification.
- The confusion matrix is more dispersed, meaning that predictions are scattered across various classes.

2. Bag of SIFT

- This method extracts local key points and represents images as a collection of features.
- It is more robust to transformations such as scaling, rotation, and lighting changes.

- The improved accuracy (61.13%) reflects its ability to capture more meaningful representations of images.
- The confusion matrix shows a strong diagonal, indicating that the model correctly classifies most categories.

Part 2. (25%)

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

	A	B
accuracy	89.16 %	90.48 %

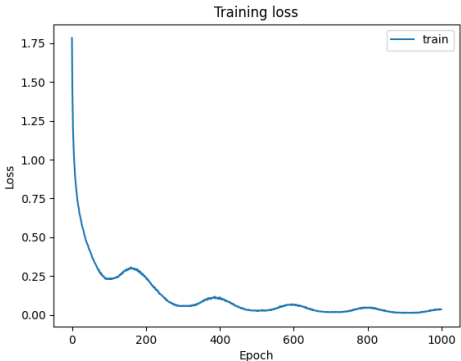
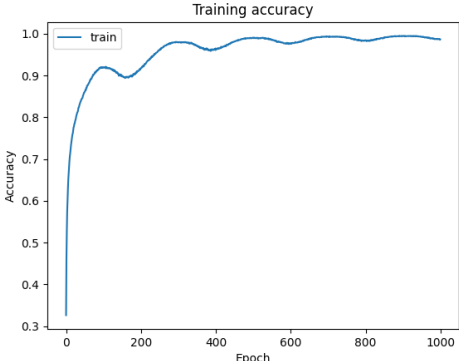
• Print the network architecture & number of parameters of both models. What is the main difference between ResNet and other CNN architectures? (5%)

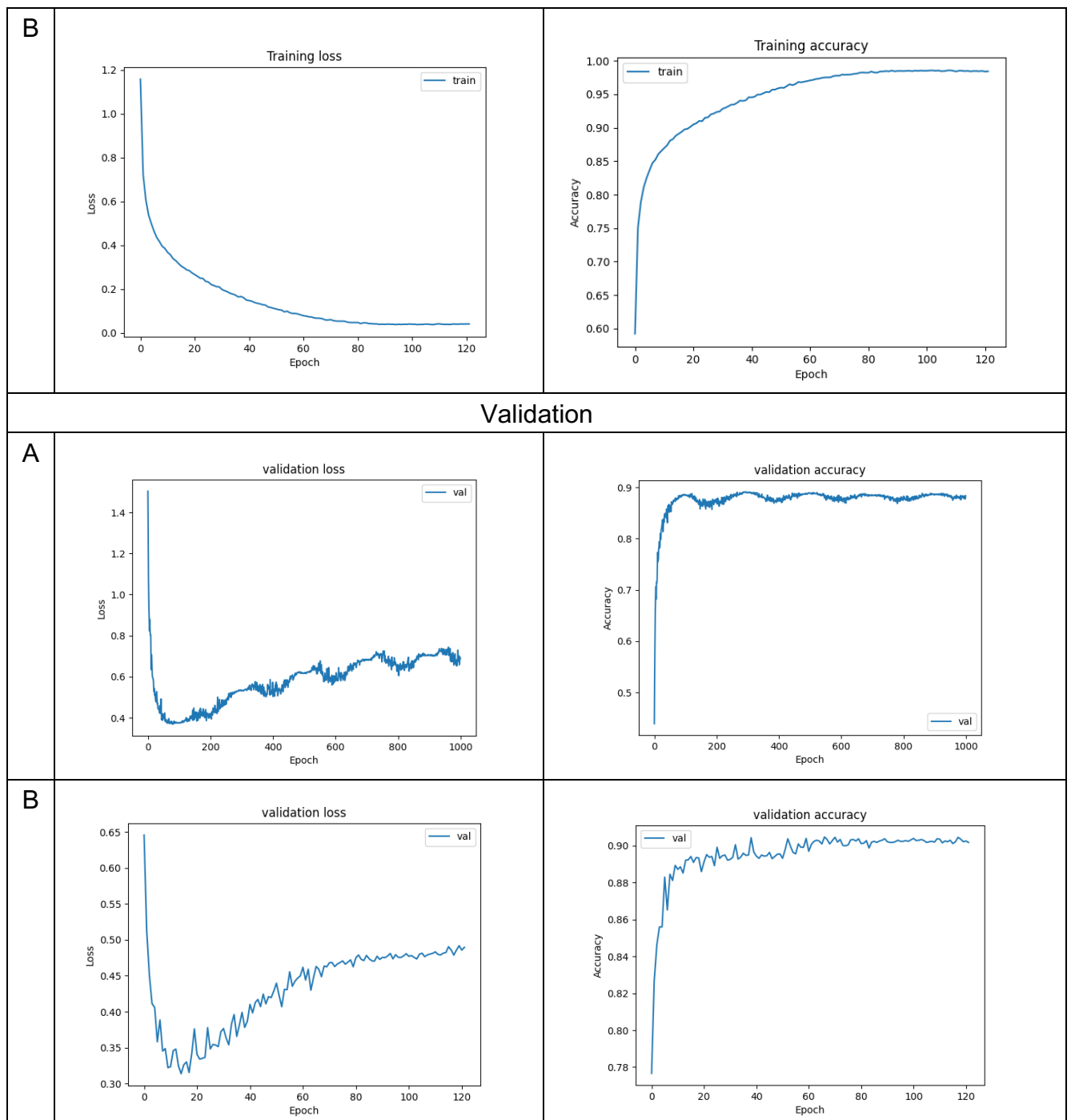
	A	B
Parameters	13,423,882	11,173,962
Architecture	<pre> ===== Model Architecture ===== MyNet((feature_extractor): Sequential((0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)) (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True) (2): ReLU() (3): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)) (4): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True) (5): ReLU() (6): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False) (7): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)) (8): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True) (9): ReLU() (10): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)) (11): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True) (12): ReLU() (13): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False) (14): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)) (15): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True) (16): ReLU() (17): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)) (18): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True) (19): ReLU() (20): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)) (flatten): Flatten(start_dim=1, end_dim=-1) (classifier): Sequential((0): Linear(in_features=8192, out_features=1024, bias=True) (1): ReLU() (2): Dropout(p=0.5, inplace=False) (3): Linear(in_features=1024, out_features=512, bias=True) (4): ReLU() (5): Dropout(p=0.5, inplace=False) (6): Linear(in_features=512, out_features=10, bias=True))) </pre>	<pre> ===== Model Architecture ===== 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))) (layer2): Sequential((0): BasicBlock((conv1): Conv2d(64, 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)) (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=(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)) (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=(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)) (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))))) </pre>

		<pre>(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))</pre>
--	--	---

The main difference between ResNet and other CNN architectures lies in its use of residual connections, which help mitigate the vanishing gradient problem in deep networks. Unlike traditional CNNs such as VGG, which stack convolutional layers sequentially, ResNet introduces shortcut connections that allow gradients to flow directly through the network, making it easier to train deeper models without performance degradation. This design enables ResNet, to achieve high accuracy while maintaining computational efficiency. Compared to architectures like VGG-16 or AlexNet, ResNet uses significantly fewer parameters while achieving better generalization, making it a popular choice for image classification tasks.

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

Train		
	Loss	Accuracy
A	 <p>Training loss</p> <p>The plot shows the training loss starting at approximately 1.75 at epoch 0 and rapidly decreasing to about 0.25 by epoch 100. It then continues to decrease more gradually, reaching near 0 by epoch 1000. The x-axis is labeled 'Epoch' and ranges from 0 to 1000. The y-axis is labeled 'Loss' and ranges from 0.00 to 1.75. A legend indicates the 'train' series.</p>	 <p>Training accuracy</p> <p>The plot shows the training accuracy starting at approximately 0.3 at epoch 0 and rapidly increasing to about 0.9 by epoch 100. It then continues to increase more gradually, reaching near 1.0 by epoch 1000. The x-axis is labeled 'Epoch' and ranges from 0 to 1000. The y-axis is labeled 'Accuracy' and ranges from 0.3 to 1.0. A legend indicates the 'train' series.</p>



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

- Data Augmentation:

1. RandomHorizontalFlip(50% change)
2. RandomVerticalFlip (30% chance)
3. RandomCrop (with padding)
4. RandomRotation ($\pm 15^\circ$)
5. ColorJitter (modifying brightness, contrast, saturation, and hue).

These augmentations help improve generalization and robustness against overfitting.

- Normalization:

The input images are normalized using mean = [0.485, 0.456, 0.406] and std = [0.229, 0.224, 0.225], which are standard values used in ImageNet-trained models. This ensures the input distribution matches the pretrained ResNet18 model, improving training stability and convergence.

- Model Architecture:

Uses a pretrained ResNet18 with modifications — the first convolution layer is adjusted, and the initial max pooling layer is removed. The final fully connected layer is modified to output 10 classes.

- Semi-Supervised Learning:

A semi-supervised learning approach is applied, leveraging both labeled and unlabeled data to improve performance. This method enhances generalization by allowing the model to learn from a larger dataset, reducing reliance on labeled samples while improving robustness.

- Loss Function:

1. CrossEntropyLoss is used, which is the standard choice for multi-class classification tasks.
2. Optimizer: Stochastic Gradient Descent (SGD) with momentum (0.9) and weight decay (1e-6) is applied for stable convergence and reducing overfitting.
3. Learning Rate Scheduler: CosineAnnealingLR is used with T_max=100, gradually decreasing the learning rate in a cosine fashion to enhance training stability and final model performance.

- Training Configuration:

The model is trained for 500 epochs with a batch size of 128, ensuring a balance between convergence speed and stability.