Computer Vision HW2 Report

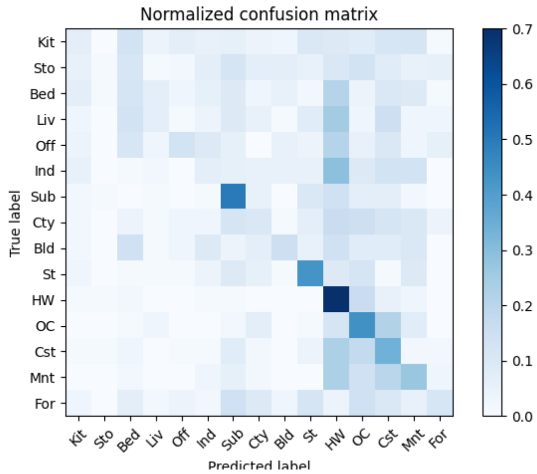
Student ID: R10943019
Name: 李奕勳

Part 1. (10%)

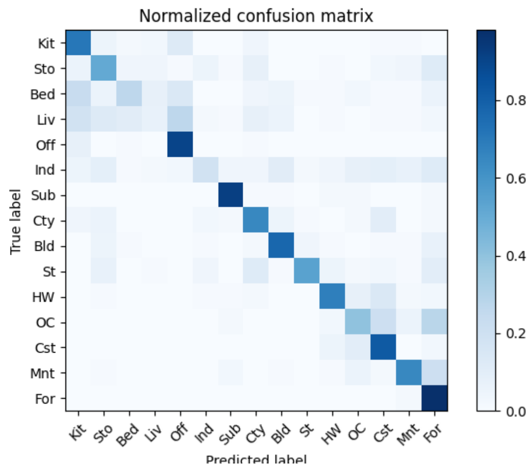
• Plot confusion matrix of two settings. (i.e. Bag of sift and tiny image representation) (5%)

Ans:

Tiny image



Bag of sift



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

Ans:

Tiny image:

```
$ python3 p1.py --feature tiny_image --classifier nearest_neighbor
Getting paths and labels for all train and test data
knn calculating.....
Accuracy =  0.234
```

只透過 resize 圖片來當成特徵，無法有效地得到有意義的特徵表示，即使是同一類別的圖片，彼此之間還是有很多變異，僅透過 tiny image 當作特徵來分群，很難將同一類別的圖片聚集在一起，因此在 testing 時，準確率自然就不會高，但優點是每張圖片只有一個特徵向量，因此運算速度很快。

Bag of sift:

```
$ python3 p1.py --feature bag_of_sift --classifier nearest_neighbor
Getting paths and labels for all train and test data
(1, 400)
knn calculating.....
Accuracy =  0.604
```

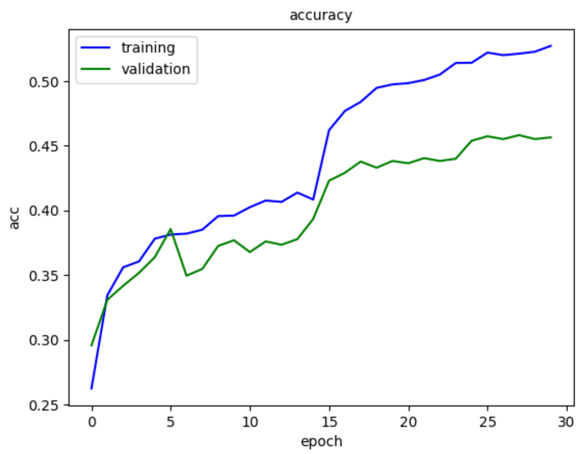
SIFT 能找到圖片中不同維度的特徵，例如:edge、corner 等等……，透過 SIFT 得到的特徵點相較於 tiny image 更有意義，因此就能更容易地將同類別的圖片分群在一起，在 testing 時，準確率就相對的高上許多，但缺點是同一張圖片中，SIFT 會找到很多特徵點，若將所有特徵點納入考慮，運算複雜度會很高。

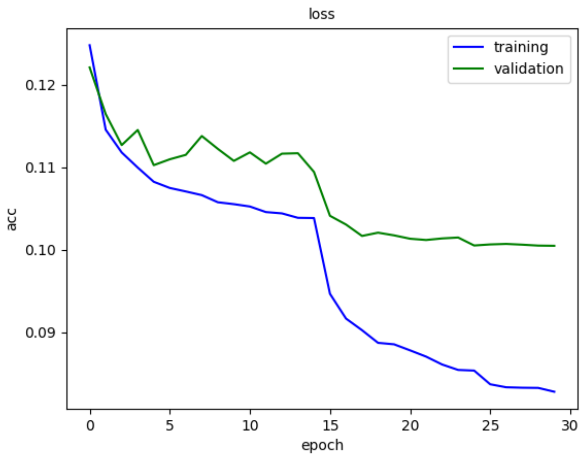
Part 2. (35%)

• Compare the performance on residual networks and LeNet. Plot the learning curve (loss and accuracy) on both training and validation sets for both 2 schemes. 8 plots in total. (20%)

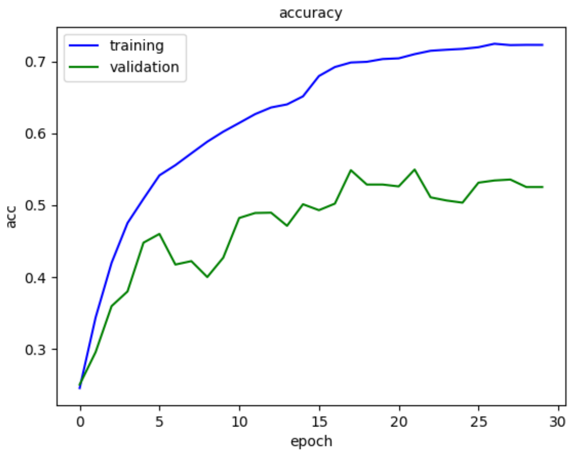
Ans:

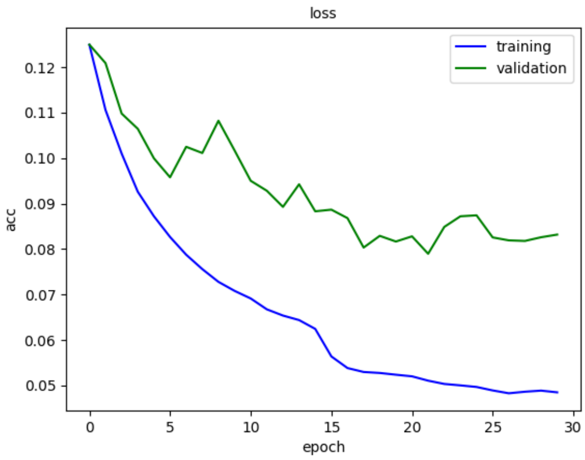
LeNet





residual networks





從圖中可以發現，無論是 training 或是 validation，在 accuracy 的表現上都是 residual network 較好，透過 residual block 可以避免梯度消失的問題，因此相較於 LeNet，residual network 較可以建立起深層的網路來學習不同語意的特徵，在分類問題上得到更好的準確率。

• Attach basic information of the model you use including model architecture and number of the parameters. (5%)

Ans:

Model I use : EfficientNetB0 pretrained on ImageNet

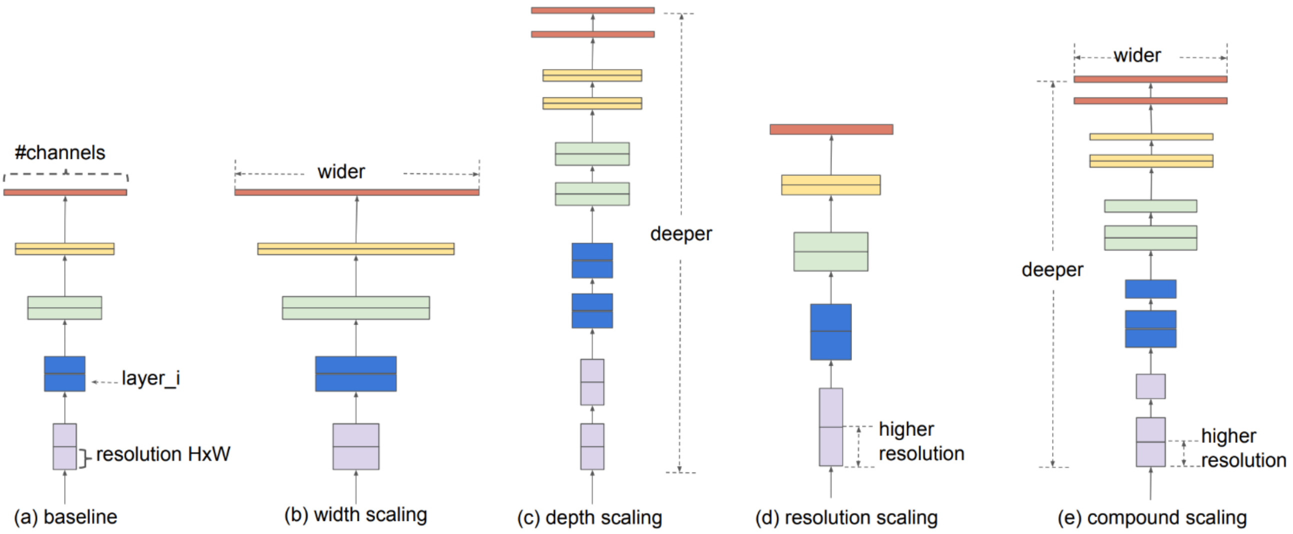


Figure 2. **Model Scaling.** (a) is a baseline network example; (b)-(d) are conventional scaling that only increases one dimension of network width, depth, or resolution. (e) is our proposed compound scaling method that uniformly scales all three dimensions with a fixed ratio.

Ref. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks (arxiv.org)

Key idea : compound scaling

在神經網路的 scaling 有三個維度:深度、寬度、解析度

在 EfficientNet 中提到 compound scaling，必須平衡三個維度的 scaling 幅度，才能得到更有效率且準確率高的網路。

Model architecture for EfficientNetB0:

```
│    └─MBConvBlock: 2-14                              --
│    │    └─Conv2dStaticSamePadding: 3-107            221,184
│    │    └─BatchNorm2d: 3-108                        2,304
│    │    └─Conv2dStaticSamePadding: 3-109            28,800
│    │    └─BatchNorm2d: 3-110                        2,304
│    │    └─Conv2dStaticSamePadding: 3-111            55,344
│    │    └─Conv2dStaticSamePadding: 3-112            56,448
│    │    └─Conv2dStaticSamePadding: 3-113            221,184
│    │    └─BatchNorm2d: 3-114                        384
│    │    └─MemoryEfficientSwish: 3-115               --
│    └─MBConvBlock: 2-15                              --
│    │    └─Conv2dStaticSamePadding: 3-116            221,184
│    │    └─BatchNorm2d: 3-117                        2,304
│    │    └─Conv2dStaticSamePadding: 3-118            28,800
│    │    └─BatchNorm2d: 3-119                        2,304
│    │    └─Conv2dStaticSamePadding: 3-120            55,344
│    │    └─Conv2dStaticSamePadding: 3-121            56,448
│    │    └─Conv2dStaticSamePadding: 3-122            221,184
│    │    └─BatchNorm2d: 3-123                        384
│    │    └─MemoryEfficientSwish: 3-124               --
│    └─MBConvBlock: 2-16                              --
│    │    └─Conv2dStaticSamePadding: 3-125            221,184
│    │    └─BatchNorm2d: 3-126                        2,304
│    │    └─Conv2dStaticSamePadding: 3-127            28,800
│    │    └─BatchNorm2d: 3-128                        2,304
│    │    └─Conv2dStaticSamePadding: 3-129            55,344
│    │    └─Conv2dStaticSamePadding: 3-130            56,448
│    │    └─Conv2dStaticSamePadding: 3-131            221,184
│    │    └─BatchNorm2d: 3-132                        384
│    │    └─MemoryEfficientSwish: 3-133               --
│    └─MBConvBlock: 2-17                              --
│    │    └─Conv2dStaticSamePadding: 3-134            221,184
│    │    └─BatchNorm2d: 3-135                        2,304
│    │    └─Conv2dStaticSamePadding: 3-136            10,368
│    │    └─BatchNorm2d: 3-137                        2,304
│    │    └─Conv2dStaticSamePadding: 3-138            55,344
│    │    └─Conv2dStaticSamePadding: 3-139            56,448
│    │    └─Conv2dStaticSamePadding: 3-140            368,640
│    │    └─BatchNorm2d: 3-141                        640
│    │    └─MemoryEfficientSwish: 3-142               --
├─Conv2dStaticSamePadding: 1-4                        --
│    └─Identity: 2-18                                 --
├─BatchNorm2d: 1-5                                    2,560
├─AdaptiveAvgPool2d: 1-6                              --
├─Dropout: 1-7                                        --
├─Linear: 1-8                                         12,810
├─MemoryEfficientSwish: 1-9                           --
```

To release GPU memory

Number of parameters for EfficientNetB0:

```
=========================================================================
Total params: 3,609,894
Trainable params: 3,609,894
Non-trainable params: 0
=========================================================================
```

Model architecture for EfficientNetB1:

```
================================================================================
Layer (type:depth-idx)                                      Param #
================================================================================
├─Conv2dStaticSamePadding: 1-1                              --
│    └─ZeroPad2d: 2-1                                       --
├─BatchNorm2d: 1-2                                          64
├─ModuleList: 1-3                                           --
│    └─MBConvBlock: 2-2                                     --
│    │    └─Conv2dStaticSamePadding: 3-1                    288
│    │    └─BatchNorm2d: 3-2                                64
│    │    └─Conv2dStaticSamePadding: 3-3                    264
│    │    └─Conv2dStaticSamePadding: 3-4                    288
│    │    └─Conv2dStaticSamePadding: 3-5                    512
│    │    └─BatchNorm2d: 3-6                                32
│    │    └─MemoryEfficientSwish: 3-7                       --
```

To release GPU memory

.

.

23 MBConvBlock in a series

.

.

```
│    └─MBConvBlock: 2-24                                    --
│    │    └─Conv2dStaticSamePadding: 3-195                  614,400
│    │    └─BatchNorm2d: 3-196                              3,840
│    │    └─Conv2dStaticSamePadding: 3-197                  17,280
│    │    └─BatchNorm2d: 3-198                              3,840
│    │    └─Conv2dStaticSamePadding: 3-199                  153,680
│    │    └─Conv2dStaticSamePadding: 3-200                  155,520
│    │    └─Conv2dStaticSamePadding: 3-201                  614,400
│    │    └─BatchNorm2d: 3-202                              640
│    │    └─MemoryEfficientSwish: 3-203                     --
├─Conv2dStaticSamePadding: 1-4                              --
│    └─Identity: 2-25                                       --
├─BatchNorm2d: 1-5                                          2,560
├─AdaptiveAvgPool2d: 1-6                                    --
├─Dropout: 1-7                                              --
├─Linear: 1-8                                               12,810
├─MemoryEfficientSwish: 1-9                                 --
```

Number of parameters for EfficientNetB1:

```
================================================================================
Total params: 6,115,530
Trainable params: 6,115,530
Non-trainable params: 0
================================================================================
```

- **Briefly describe what method do you apply? (e.g. data augmentation, model architecture, loss function, semi-supervised etc.) (10%)**

Ans:

Data augmentation :

TOO MUCH augmentation will lead to poor training progress!!

```
transforms.RandomPerspective(distortion_scale = 0.6, p = 1.0),
transforms.RandomRotation(degrees = (0, 180)),
transforms.RandomAffine(degrees = (30, 70), translate = (0.1, 0.3), scale = (0.5, 0.75)),
transforms.RandomHorizontalFlip(p=0.5),
transforms.RandomVerticalFlip(p=0.5),
```

```
epoch = 13
100%|| 1294/1294 [01:35<00:00, 13.52it/s]
100%|| 144/144 [00:01<00:00, 77.04it/s]
**********
time = 1.0000 MIN 37.6128 SEC, total time = 22.0000 Min 57.8580 SEC
training loss : 0.1230   train acc = 0.2817
val loss : 0.1307   val acc = 0.2300
========================

epoch = 14
100%|| 1294/1294 [01:30<00:00, 14.36it/s]
100%|| 144/144 [00:01<00:00, 76.01it/s]
**********
time = 1.0000 MIN 32.0195 SEC, total time = 24.0000 Min 29.8777 SEC
training loss : 0.1222   train acc = 0.2911
val loss : 0.1273   val acc = 0.2587
========================
```

Final data augmentation - only vertical and horizontal flip

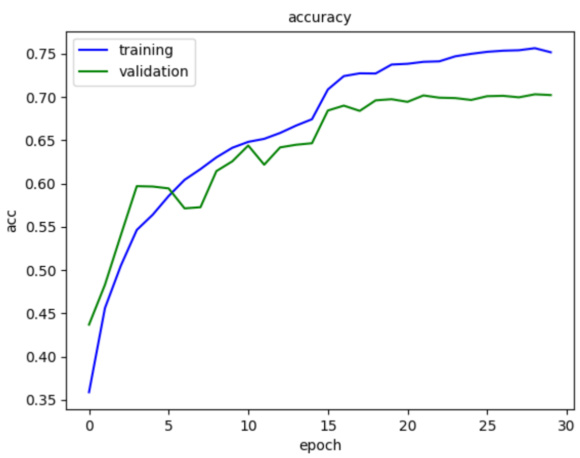
```
transforms.RandomHorizontalFlip(p=0.5),
transforms.RandomVerticalFlip(p=0.5),
```

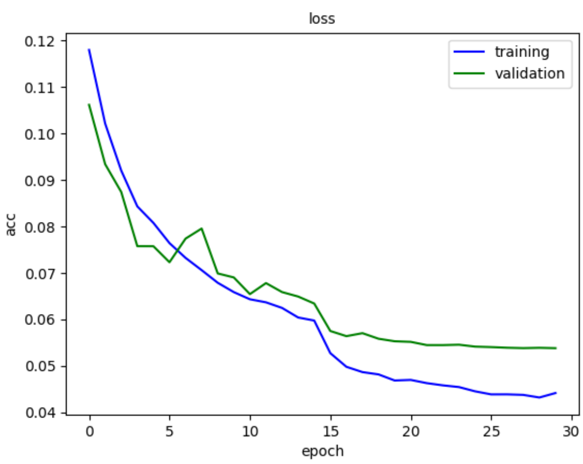
```
epoch = 13
100%|| 1294/1294 [01:42<00:00, 12.56it/s]
100%|| 144/144 [00:02<00:00, 62.00it/s]
**********
time = 1.0000 MIN 45.3177 SEC, total time = 25.0000 Min 27.8183 SEC
training loss : 0.0588   train acc = 0.6832
val loss : 0.0608   val acc = 0.6639
========================

epoch = 14
100%|| 1294/1294 [01:43<00:00, 12.55it/s]
100%|| 144/144 [00:02<00:00, 62.41it/s]
**********
time = 1.0000 MIN 45.3926 SEC, total time = 27.0000 Min 13.4020 SEC
training loss : 0.0575   train acc = 0.6842
val loss : 0.0591   val acc = 0.6700
========================
```

Model architecture :

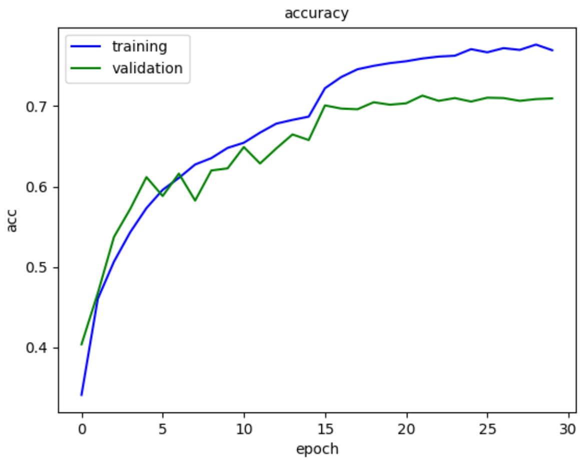
Except from LeNet and residual block, I also train on EfficientNet family (B0、B1、B4)

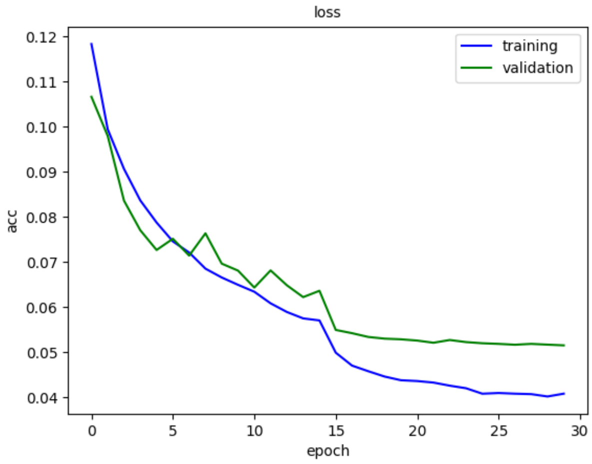
EfficientNetB0:



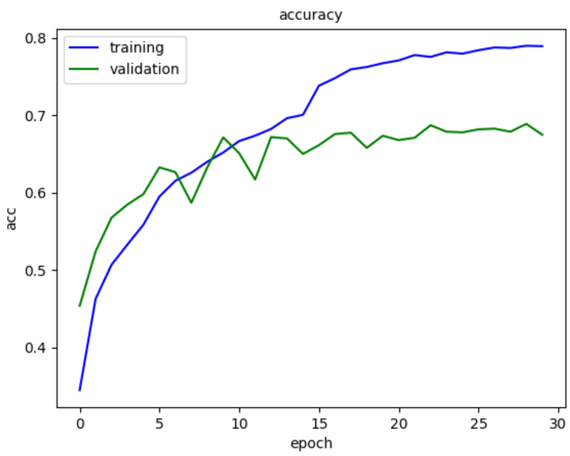


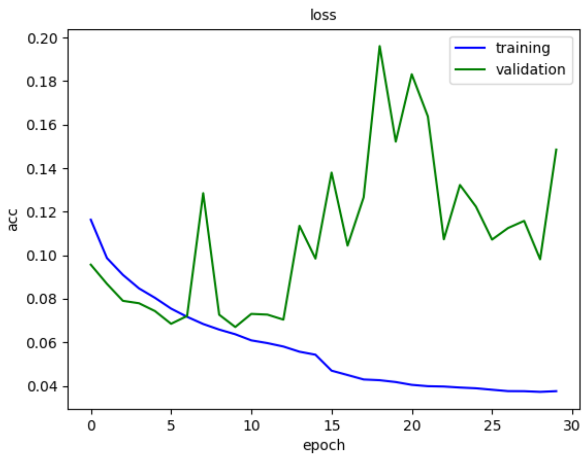
EfficientNetB1:





EfficientNetB4:





由上圖發現，如果使用同一個 training dataset 去訓練，越深層(B4)的網路越容易 overfitting

Loss function : use cross-entropy for classification problem