資料科學, Fall 2017

Homework #3 B03901023 許秉鈞

Model Design

• 1. Describe how you preprocess your data and the model architecture. (30%)

我利用 keras 的 cifar10.load_data() 直接把 train, test 讀進來，他會自己分成 50000, 10000 筆。 preprocess 部分我做了不少 augmentation（因為這樣能提高不少 accuracy），我使用 **VGG16** 最後 best testing set accuracy = **0.9329**。以下是augmentation，請參考我的操作

```
#data augmentation
datagen = ImageDataGenerator(
    featurewise_center=False,  # set input mean to 0 over the dataset
    samplewise_center=False,  # set each sample mean to 0
    featurewise_std_normalization=False,  # divide inputs by std of the dataset
    samplewise_std_normalization=False,  # divide each input by its std
    zca_whitening=False,  # apply ZCA whitening
    rotation_range=15,  # randomly rotate images in the range (degrees, 0 to 180)
    width_shift_range=0.1,  # randomly shift images horizontally (fraction of total
    width)
    height_shift_range=0.1,  # randomly shift images vertically (fraction of total
    height)
    horizontal_flip=True,  # randomly flip images
    vertical_flip=False)  # randomly flip images
```

因為 keras 官方就已經提供了 CNN 和 ResNet 的版本了，而且 fine-tune 的很好，所以我繳交的版本是 VGG16 的，然後做了些自己的 fine-tune，舉例來說，我做了以下兩個操作：

```
def normalize(self, X_train, X_test):
    # this function normalize inputs for zero mean and unit variance
    # it is used when training a model.

    mean = np.mean(X_train, axis=(0, 1, 2, 3))
    std = np.std(X_train, axis=(0, 1, 2, 3))
    X_train = (X_train - mean)/(std+1e-7)
    X_test = (X_test - mean)/(std+1e-7)
    return X_train, X_test

def normalize_production(self, x):
    #this function is used to normalize instances in production according to saved
    training set statistics

    #these values produced during first training and are general for the standard
    cifar10 training set normalization
    mean = 120.707 # PIN-CHUN: this is experimental results parameter
    std = 64.15
    return (x - mean)/(std + 1e-7)
```

其中 normalize() 是用來把 mean 調到 0，variance 調到 1。然後 normalize_production() 是專門為 cifar-10 調出來的參數 mean=120.707 和 std=64.15（資料來源在此：https://github.com/geifmany/cifar-vgg）。

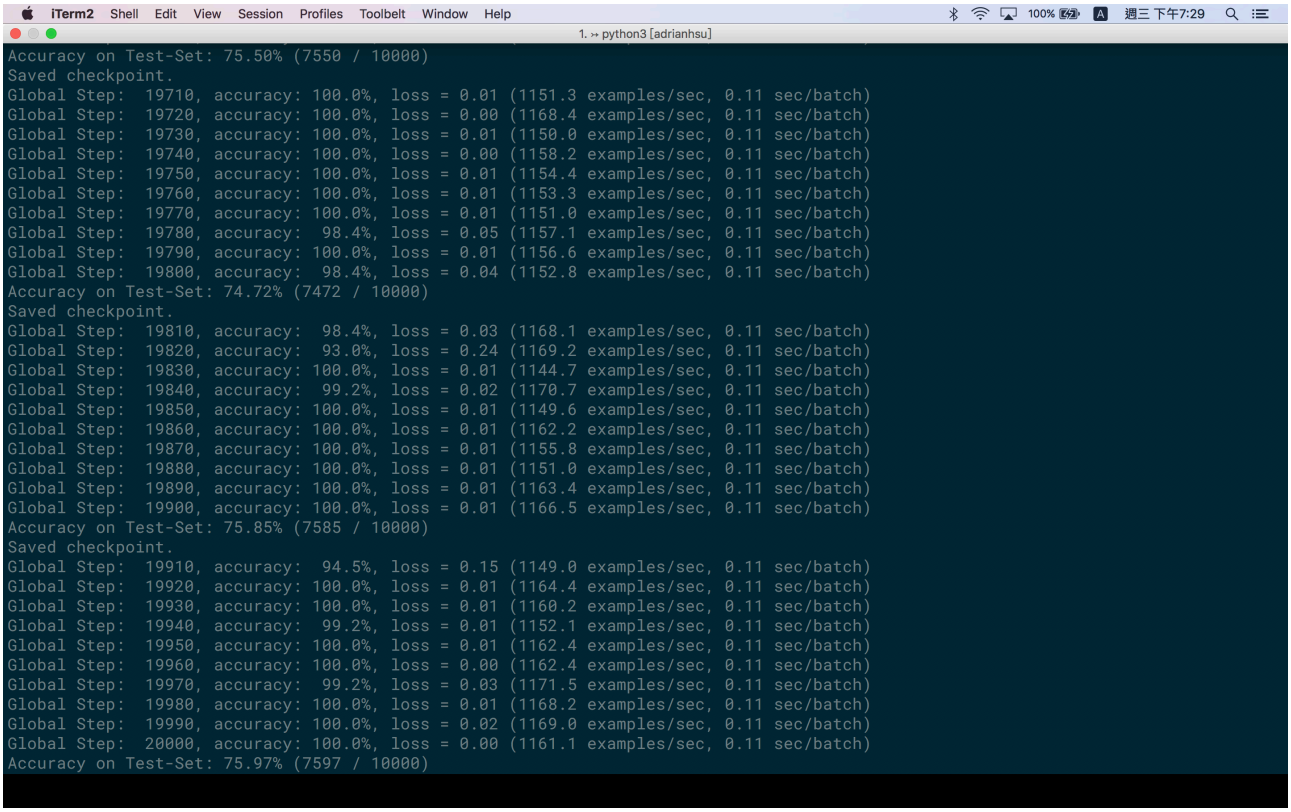
我的 code 最後會把結果存為 cifar10vgg.h5 這個 pre-trained model，因為檔案有些大就沒有上傳 ceiba 了。

●2. Compare different architectures or parameters and record their performance (30%)

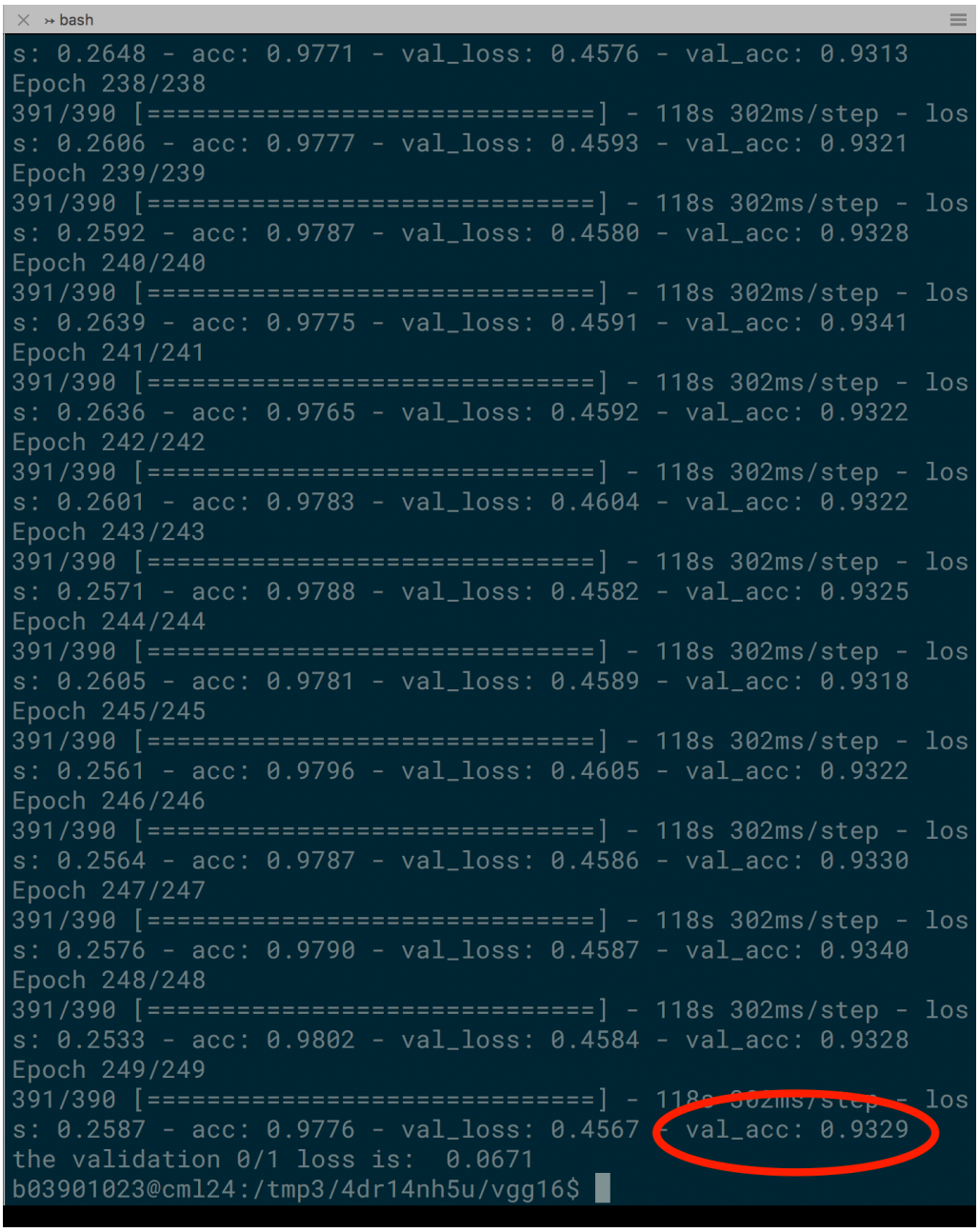
我使用了三種 Model，分別是 CNN, VGG16 以及 ResNet
其他的基本參數如下：（這邊主要是 VGG16的參數調整，也就是繳交的這份code）

```
#training parameters
batch_size = 128
maxepoches = 250
learning_rate = 0.1 # 0.2, 0.3 … etc
lr_decay = 1e-6 # 1e-5, 1e-4

optimizers.SGD(lr=lrf, decay=lr_decay, momentum=0.9, nesterov=True)

# training process in a for loop with learning rate drop every 25 epoches.
if epoch % 25 == 0 and epoch > 0:
                lrf /= 2
```

一、 CNN 的結果是 75.97 % (iteration = 20000, no data augmentation)



二、 VGG16 的結果是 93.29 % (# of epochs = 250, with data augmentation)



```
bash
s: 0.2648 - acc: 0.9771 - val_loss: 0.4576 - val_acc: 0.9313
Epoch 238/238
391/390 [==============================] - 118s 302ms/step - los
s: 0.2606 - acc: 0.9777 - val_loss: 0.4593 - val_acc: 0.9321
Epoch 239/239
391/390 [==============================] - 118s 302ms/step - los
s: 0.2592 - acc: 0.9787 - val_loss: 0.4580 - val_acc: 0.9328
Epoch 240/240
391/390 [==============================] - 118s 302ms/step - los
s: 0.2639 - acc: 0.9775 - val_loss: 0.4591 - val_acc: 0.9341
Epoch 241/241
391/390 [==============================] - 118s 302ms/step - los
s: 0.2636 - acc: 0.9765 - val_loss: 0.4592 - val_acc: 0.9322
Epoch 242/242
391/390 [==============================] - 118s 302ms/step - los
s: 0.2601 - acc: 0.9783 - val_loss: 0.4604 - val_acc: 0.9322
Epoch 243/243
391/390 [==============================] - 118s 302ms/step - los
s: 0.2571 - acc: 0.9788 - val_loss: 0.4582 - val_acc: 0.9325
Epoch 244/244
391/390 [==============================] - 118s 302ms/step - los
s: 0.2605 - acc: 0.9781 - val_loss: 0.4589 - val_acc: 0.9318
Epoch 245/245
391/390 [==============================] - 118s 302ms/step - los
s: 0.2561 - acc: 0.9796 - val_loss: 0.4605 - val_acc: 0.9322
Epoch 246/246
391/390 [==============================] - 118s 302ms/step - los
s: 0.2564 - acc: 0.9787 - val_loss: 0.4586 - val_acc: 0.9330
Epoch 247/247
391/390 [==============================] - 118s 302ms/step - los
s: 0.2576 - acc: 0.9790 - val_loss: 0.4587 - val_acc: 0.9340
Epoch 248/248
391/390 [==============================] - 118s 302ms/step - los
s: 0.2533 - acc: 0.9802 - val_loss: 0.4584 - val_acc: 0.9328
Epoch 249/249
391/390 [==============================] - 118s 302ms/step - los
s: 0.2587 - acc: 0.9776 - val_loss: 0.4567 - val_acc: 0.9329
the validation 0/1 loss is:  0.0671
b03901023@cml24:/tmp3/4dr14nh5u/vgg16$
```

我發現在 VGG16 這個 model 下，調整參數的話（learning_rate = 0.1, 0.2, 0.3, lr_decay = 1e-6, 1e-5, 1e-4），可能是因為我有做 data augmentation 還有每過 25 個 epochs 就做 lrf /= 2 的關係，這些參數雖然影響初期的值，但最後 20000 次 iteration 後的 accuracy 差距不大，都在 0.01 之內（ 也就是 0.92 to 0.94 這個範圍之間）。

三、（keras 官方 code） ResNet 的結果是 91.77% (# of epochs = 200, lr scheduling)

```
bash
Epoch 195/200
1562/1563 [==============================>.] - ETA: 0s - loss: 0.15
52 - acc: 0.9867Epoch 00195: val_acc did not improve
1563/1563 [==============================] - 104s 66ms/step - los
s: 0.1552 - acc: 0.9867 - val_loss: 0.4355 - val_acc: 0.9168
Learning rate:  5e-07
Epoch 196/200
1562/1563 [==============================>.] - ETA: 0s - loss: 0.15
46 - acc: 0.9871Epoch 00196: val_acc did not improve
1563/1563 [==============================] - 104s 67ms/step - los
s: 0.1546 - acc: 0.9871 - val_loss: 0.4373 - val_acc: 0.9177
Learning rate:  5e-07
Epoch 197/200
1562/1563 [==============================>.] - ETA: 0s - loss: 0.15
63 - acc: 0.9869Epoch 00197: val_acc did not improve
1563/1563 [==============================] - 104s 66ms/step - los
s: 0.1563 - acc: 0.9869 - val_loss: 0.4379 - val_acc: 0.9171
Learning rate:  5e-07
Epoch 198/200
1562/1563 [==============================>.] - ETA: 0s - loss: 0.15
50 - acc: 0.9872Epoch 00198: val_acc did not improve
1563/1563 [==============================] - 105s 67ms/step - los
s: 0.1550 - acc: 0.9872 - val_loss: 0.4372 - val_acc: 0.9171
Learning rate:  5e-07
Epoch 199/200
1562/1563 [==============================>.] - ETA: 0s - loss: 0.15
60 - acc: 0.9868Epoch 00199: val_acc did not improve
1563/1563 [==============================] - 103s 66ms/step - los
s: 0.1560 - acc: 0.9868 - val_loss: 0.4355 - val_acc: 0.9183
Learning rate:  5e-07
Epoch 200/200
1562/1563 [==============================>.] - ETA: 0s - loss: 0.15
61 - acc: 0.9864Epoch 00200: val_acc did not improve
1563/1563 [==============================] - 105s 67ms/step - los
s: 0.1561 - acc: 0.9864 - val_loss: 0.4354 - val_acc: 0.9177
10000/10000 [==============================] - 6s 586us/step
Test loss: 0.435383516264
Test accuracy: 0.9177
b03901023@cml24:/tmp3/4dr14nh5u/resnet$
```

硬體資訊：[0], [2] Tesla K80 in CMLab

```
cml24  Thu Nov 16 09:46:01 2017
[0] Tesla K80          | 42'C,   0 % |     2 / 11439 MB |
[1] Tesla K80          | 52'C,  99 % | 10948 / 11439 MB |
[2] Tesla K80          | 33'C,   0 % |     2 / 11439 MB |
[3] Tesla K80          | 29'C,   0 % |     2 / 11439 MB |
cml25  Thu Nov 16 09:46:02 2017
```