113-1 雲端運算與邊緣運算應用

End computing and edge computing applications.

Lab3

ResNet

Inference and application

授課老師： 王斯弘 老師

學　　生： B11123206 陳冠欣

中 華 民 國 １１３ 年 １２ 月 ２５ 日

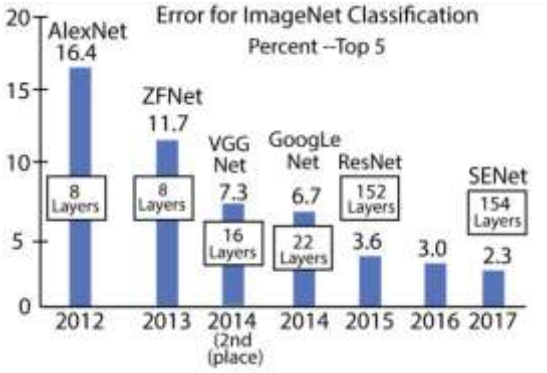
目錄

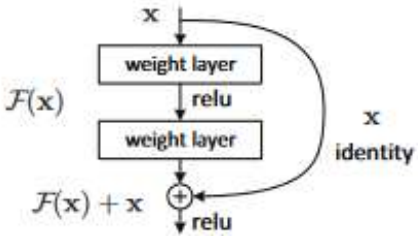
壹、 ResNet 簡介

ResNet（Residual Network）是一種深度卷積神經網路架構，最早由微軟研究院於 2015 年提出，其主要目的是解決隨著網路深度增加而導致的梯度消失和退化問題。ResNet 在許多影像分類與辨識任務中表現出色，並且在 2015 年 ImageNet 比賽中奪冠。

ResNet 的關鍵特性

1. **殘差結構（Residual Block）**
   - ResNet 的核心概念是殘差學習。
   - 傳統網路直接學習輸入到輸出的轉換，ResNet 則學習輸入與輸出的「殘差」，即 $F(x)=H(x)-xF(x) = H(x) - xF(x)=H(x)-x$。
   - 透過「跳躍連接（skip connection）」，輸入可直接繞過中間層，加速模型訓練並解決深層網路的退化問題。
2. **深度優勢**
   - 支援非常深的網路結構，例如 ResNet-50、ResNet-101 等，分別包含 50 層與 101 層。
   - 這些深度使其能捕捉更多高層次的特徵資訊。
3. **性能表現**
   - 在影像分類、物件檢測、語音辨識等多個任務上均有應用。
   - 與傳統的卷積神經網路（如 VGGNet）相比，ResNet 具有更高的準確率，且能在增加深度的情況下保持穩定。





貳、 PC 端 模型訓練&推理

1. 載入 訓練,驗證 資料集

```
[ ]  DATASET_PATH  =  '/content/drive/MyDrive/LAB3/sample'

     IMAGE_SIZE  =  (224,224)

     NUM_CLASSES  =  5

     BATCH_SIZE  =  256

     #  Epoch  數
     NUM_EPOCHS  =  100

     #  模型輸出儲存的檔案
     WEIGHTS_FINAL  =  'model-resnet18_3.h5'

     #  透過  data  augmentation  產生訓練與驗證用的影像資料
     train_datagen  =  ImageDataGenerator(rotation_range=40,
                                          width_shift_range=0.2,
                                          height_shift_range=0.2,
                                          shear_range=0.2,
                                          zoom_range=0.2,
                                          channel_shift_range=10,
                                          horizontal_flip=True,
                                          fill_mode='nearest')
     train_batches  =  train_datagen.flow_from_directory(DATASET_PATH  +  '/train',
                                                         target_size=IMAGE_SIZE,
                                                         interpolation='bicubic',
                                                         class_mode='categorical',
                                                         shuffle=True,
                                                         batch_size=BATCH_SIZE)

     valid_datagen  =  ImageDataGenerator()
     valid_batches  =  valid_datagen.flow_from_directory(DATASET_PATH  +  '/valid',
                                                         target_size=IMAGE_SIZE,
                                                         interpolation='bicubic',
                                                         class_mode='categorical',
                                                         shuffle=False,
                                                         batch_size=BATCH_SIZE)

     #  輸出各類別的索引值
     for  cls,  idx  in  train_batches.class_indices.items():
            print('Class  #{}  =  {}'.format(idx,  cls))
```

```
Found 3100 images belonging to 5 classes.
Found 650 images belonging to 5 classes.
Class #0 = bike
Class #1 = car
Class #2 = cats
Class #3 = dogs
Class #4 = panda
```

2. Database、Train & valid 說明

train 與 valid 資料夾需增加自己的資料集
train 資料集每個類別 500 張
valid 資料集每個類別 150 張
圖片大小約 640*480

3. 定義模型架構

```
[ ]  def block(x,out_filters,k_size=(3,3),downsample=0):
         if(downsample==1):
             x1 = Conv2D(out_filters, k_size, strides=(2,2), padding='same')(x)
             x = Conv2D(out_filters, k_size, strides=(2,2), padding='same')(x)
         else:
             x1 = x
             x = Conv2D(filters=out_filters, kernel_size=k_size,strides=(1,1) ,padding='same')(x)
         x = BatchNormalization()(x)
         x = ReLU()(x)
         x = Conv2D(filters=out_filters, kernel_size=k_size,strides=(1,1) ,padding='same')(x)
         x = BatchNormalization()(x)
         x = add([x1, x])
         x = ReLU()(x)
         return x


     def resnet18(x):
         x = Conv2D(filters=64, kernel_size=(7,7),strides=(2,2), padding='same')(x)
         x = MaxPooling2D()(x)
         x = block(x,out_filters=64,downsample=0)
         x = block(x,out_filters=64,downsample=0)
         x = block(x,out_filters=128,downsample=1)
         x = block(x,out_filters=128,downsample=0)
         x = block(x,out_filters=256,downsample=1)
         x = block(x,out_filters=256,downsample=0)
         x = block(x,out_filters=512,downsample=1)
         x = block(x,out_filters=512,downsample=0)
         x = GlobalAveragePooling2D()(x)
         x = Dense(5, activation='softmax')(x)

         return x
```

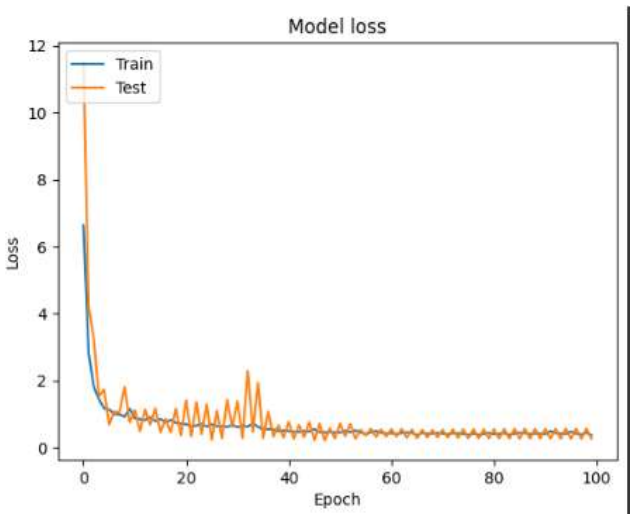
```
[ ]  img_input = Input(shape=IMAGE_SIZE+(3,))
     output = resnet18(img_input)

     model = Model(img_input, output)
     print(model.summary())
```

4. 訓練模型

```
# 使用 Adam optimizer，以較低的 learning rate 進行 fine-tuning
model.compile(optimizer=Adam(learning_rate=0.0001),
                                loss='categorical_crossentropy', metrics=['acc'])

csv_logger = CSVLogger('training.csv')
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2,patience=10, min_lr=1e-7)
earlystop = EarlyStopping(monitor = 'val_acc', patience = 30, verbose = 1)
cbks = [csv_logger,reduce_lr]

# 訓練模型
history=model.fit(train_batches,
                            steps_per_epoch = train_batches.samples // BATCH_SIZE,
                            validation_data = valid_batches,
                            validation_steps = valid_batches.samples // BATCH_SIZE,
                            epochs = NUM_EPOCHS,
                            callbacks = cbks
                            )

# 儲存訓練好的模型
model.save('/content/drive/MyDrive/LAB3/model-resnet18_3.h5')
```

```
12/12 ━━━━━━━━━━━━━━━━━━━━ 2s 90ms/step - acc: 0.8320 - loss: 0.4752 - val_acc: 0.9058 - val_loss: 0.2652 - learning_rate: 1.0000e-07
Epoch 97/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 82s 3s/step - acc: 0.8433 - loss: 0.4313 - val_acc: 0.7871 - val_loss: 0.5655 - learning_rate: 1.0000e-07
Epoch 98/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 2s 84ms/step - acc: 0.8516 - loss: 0.3902 - val_acc: 0.9058 - val_loss: 0.2646 - learning_rate: 1.0000e-07
Epoch 99/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 77s 3s/step - acc: 0.8294 - loss: 0.4549 - val_acc: 0.7891 - val_loss: 0.5663 - learning_rate: 1.0000e-07
Epoch 100/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 2s 62ms/step - acc: 0.8633 - loss: 0.3693 - val_acc: 0.9058 - val_loss: 0.2634 - learning_rate: 1.0000e-07
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recomme
```

5. 模型成果及驗證成果

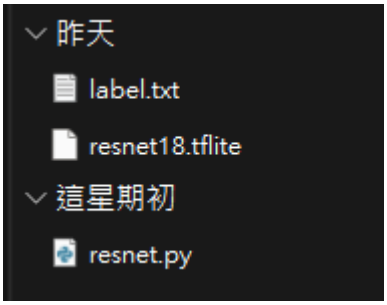



```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 774ms/step
/content/drive/MyDrive/LAB3/test/panda.jpg
  1.000  panda
  0.000  dog
  0.000  car
  0.000  bike
  0.000  cats
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 97ms/step
/content/drive/MyDrive/LAB3/test/dog.jpg
  0.999  dog
  0.001  cats
  0.000  bike
  0.000  panda
  0.000  car
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 94ms/step
/content/drive/MyDrive/LAB3/test/cat.JPEG
  1.000  cats
  0.000  dog
  0.000  bike
  0.000  car
  0.000  panda
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 96ms/step
/content/drive/MyDrive/LAB3/test/car.JPEG
  1.000  car
  0.000  dog
  0.000  panda
  0.000  cats
  0.000  bike
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 94ms/step
/content/drive/MyDrive/LAB3/test/bike.JPEG
  0.999  bike
  0.001  dog
  0.000  cats
  0.000  panda
  0.000  car
```

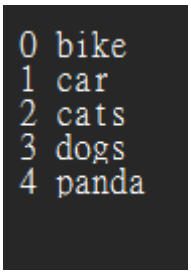
6.

參、 Edge 端 操作流程-

1. 準備檔案(優化後的 tflite 檔、要執行的 Py 檔、Labels 檔)



Labels 內容就是訓練時的類別



2. 進入到 dev board 的系統，進入 dev 中 尋找 sdc1

```
mendel@coy-tang:/$ ls /dev
apex_0              loop1               ptyp4   tty18  tty5    ttypc
autofs              loop2               ptyp5   tty19  tty50   ttypd
block               loop3               ptyp6   tty2   tty51   ttype
bsg                 loop4               ptyp7   tty20  tty52   ttypf
btrfs-control       loop5               ptyp8   tty21  tty53   ttyS0
bus                 loop6               ptyp9   tty22  tty54   ttyS1
cec0                loop7               ptypa   tty23  tty55   ttyS2
char                loop-control        ptypb   tty24  tty56   ttyS3
console             mapper              ptypc   tty25  tty57   ubi_ctrl
cpu_dma_latency     media0              ptypd   tty26  tty58   urandom
cuse                mem                 ptype   tty27  tty59   v4l
disk                memory_bandwidth    ptypf   tty28  tty6    vcs
dri                 mmcblk0             random  tty29  tty60   vcs1
fb0                 mmcblk0boot0        rtc     tty3   tty61   vcs2
fd                  mmcblk0boot1        rtc0    tty30  tty62   vcs3
full                mmcblk0p1           sdc     tty31  tty63   vcs4
fuse                mmcblk0p2           sdc1    tty32  tty7    vcs5
galcore             mmcblk0p3           shm     tty33  tty8    vcs6
gpiochip0           mmcblk0p4           snd     tty34  tty9    vcs7
gpiochip1           mmcblk0rpmb         spidev0.0  tty35  ttyGS0  vcsa
gpiochip2           mqueue              spidev0.1  tty36  ttymxc0  vcsa1
gpiochip3           mxc_hantro          stderr  tty37  ttymxc1  vcsa2
gpiochip4           net                 stdin   tty38  ttymxc2  vcsa3
hugepages           network_latency     stdout  tty39  ttyp0   vcsa4
hwrng               network_throughput  tty     tty4   ttyp1   vcsa5
i2c-0               null                tty0    tty40  ttyp2   vcsa6
i2c-1               port                tty1    tty41  ttyp3   vcsa7
i2c-2               ppp0                tty10   tty42  ttyp4   vga_arbiter
initctl             ptmx                tty11   tty43  ttyp5   vhci
input               ptp0                tty12   tty44  ttyp6   video0
ion                 pts                 tty13   tty45  ttyp7   video1
kmsg                ptyp0               tty14   tty46  ttyp8   watchdog
kvm                 ptyp1               tty15   tty47  ttyp9   watchdog0
log                 ptyp2               tty16   tty48  ttypa   zero
loop0               ptyp3               tty17   tty49  ttypb
mendel@coy-tang:/$
```

3. 創建一個掛載 US 的資料夾 test

```
mendel@coy-tang:/$ ls
bin   home            lib64       opt   sbin  tmp       vmlinuz.old
boot  initrd.img      lost+found  proc  srv   usr
dev   initrd.img.old  media       root  sys   var
etc   lib             mnt         run   test  vmlinuz
```

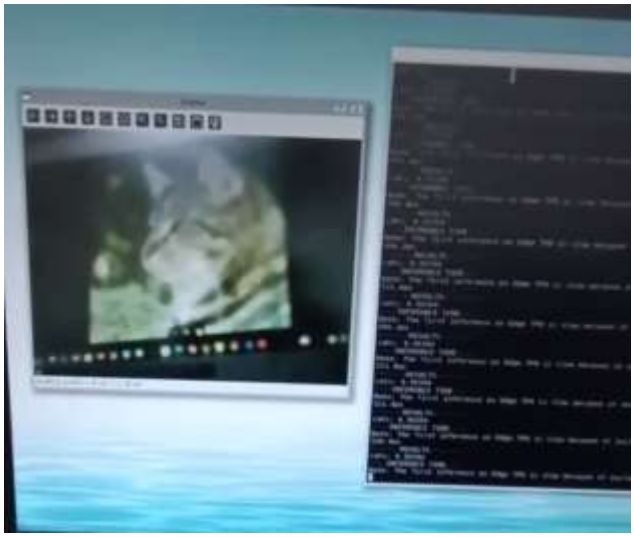
4. 輸入指令 將 USB 掛載到 test，就可以看到 test 已經有 USB 當中的檔案

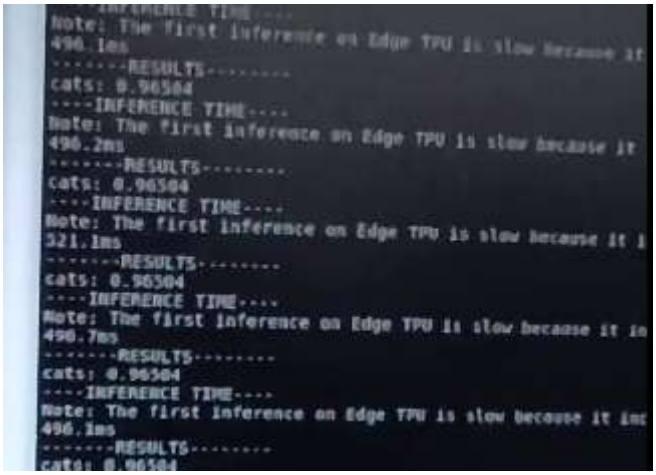
```
mendel@coy-tang:/$ ls /test
 label.txt   resnet18.tflite   resnet.py  'System Volume Information'
```

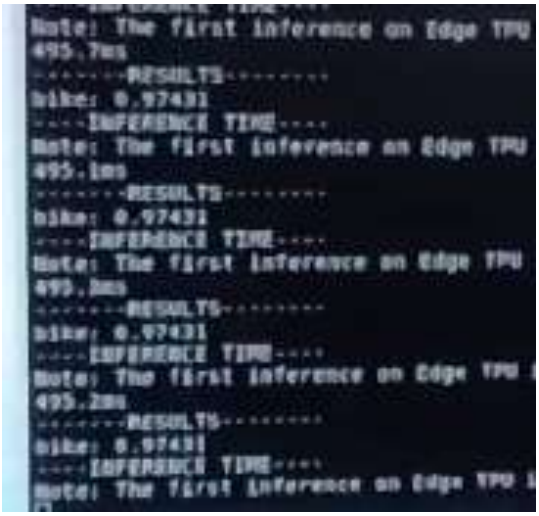
5. 接下來輸入執行指令 Python3 resnet.py -m resnet18.tflite -l label.txt

```
mendel@coy-tang:/test$ python3 resnet.py -m /test/resnet18.tflite -l /test/label.txt
----INFERENCE TIME----
```

肆、 Edge 端 驗證結果







實作影片：https://youtube.com/shorts/bdnmNafnxwU?feature=share

伍、 遇到問題與解決

不管是在驗證模型或是 Edge 端的驗證結果，會一直驗證錯誤，導致模型的在 Dev board 上的應用不佳，會辨識成其他項目

```
1/1 ------------------------------ 1s 749ms/step
/content/drive/MyDrive/LAB3/test/dog.jpg
  0.912  bike
  0.049  panda
  0.024  cats
  0.009  dog
  0.006  car
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 100ms/step
/content/drive/MyDrive/LAB3/test/bike.JPEG
  0.528  panda
  0.377  bike
  0.068  dog
  0.024  cats
  0.002  car
```

最後是清理訓練集的樣本和調整樣本數量和調整模型架構
才成功讓模型準確度增加

陸、 心得

在這次的 Lab3 實驗中，我學習了如何進行 LeNet 模型的推論與應用，並在 PC 端與 Edge 端中進行模型的操作與驗證。實驗過程中，除了熟悉模型的基本架構，還加深了對模型部署流程的理解。

LeNet 是一個經典的卷積神經網路，主要設計用於影像分類任務。透過這次實驗，我瞭解了從模型訓練到推論的整體流程，並掌握了如何在 Edge 端進行模型的優化與部署。在模型驗證階段，我注意到推論的準確度會受到硬體資源和參數設置的影響，這讓我對不同運算環境的限制有了更深的體會。

在實驗中也遇到了一些挑戰，例如模型部署至 Edge 端時，出現了執行效能不如預期的問題。經過與同學的討論與參考資料後，我們嘗試優化推理過程，像是調整模型參數及升級相關套件，最終解決了問題並提升了效能。