國立雲林科技大學電子工程系

人工智慧深度學習

Lab7

Deeplab

第十二組

指導教授: 夏世昌 教授

王斯弘 助理教授

組員:四電子三 A B10913014 林廷緯

四電子三 B B10913158 陳維翔

四電子三 B B10913154 曹宸維

中華民國 111 年 12 月 20 日

一、 題目

二、 基本介紹

1. Deeplab 介紹

2. Deeplab 實作方式

3. EdgeAI 平台介紹

4. EdgeAI 實作方式

5. 輕量化 tflite 轉換

三、 程式說明

1. Deeplab 模型及訓練結果

2. h5 檔轉換至 tflite 檔

四、 EdgeAI 平台驗證結果

五、 心得與討論

一、 題目

Deeplab 架構模型訓練及 EdgeAI 平台驗證

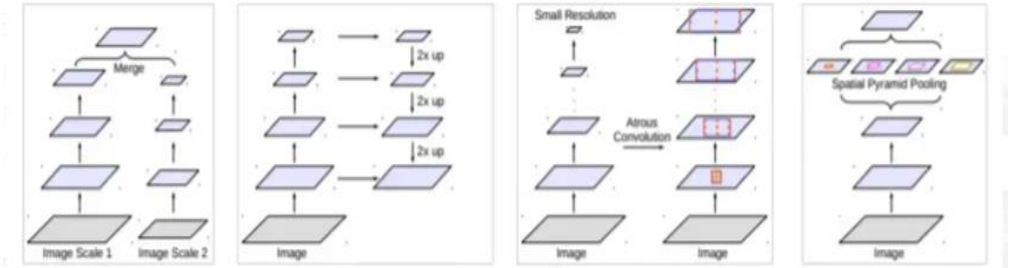
二、 基本介紹

1. Deeplab 介紹

Deeplab 是 google 提出用 DCNN 來解決語義分割任務的解決方案，主要是為了解決任務中的兩個挑戰:

I. 連續池化或卷積操作，讓特徵分辨維持不變性，使得深層網路能學習更抽象的特徵。然而這種不變性在像素級別的密集預測任務反而造成了阻礙，也導致詳細空間信息的預測不穩定，為了克服這個問題，google 建議使用空洞卷積。

II. 多尺度物體的存在。



(a) Image Pyramid (b) Encoder-Decoder (c) Deeper w. Atrous Convolution (d) Spatial Pyramid Pooling

Figure 2. Alternative architectures to capture multi-scale context.

Image Pyramid:將輸入圖片放縮成不同比例，分別應用在 DCNN 上，將預測結果融合得到最終輸出。

Encoder-Decoder:將 Encoder 階段的多尺度特徵運用到 Decoder 階段上來恢復空間分辨率。

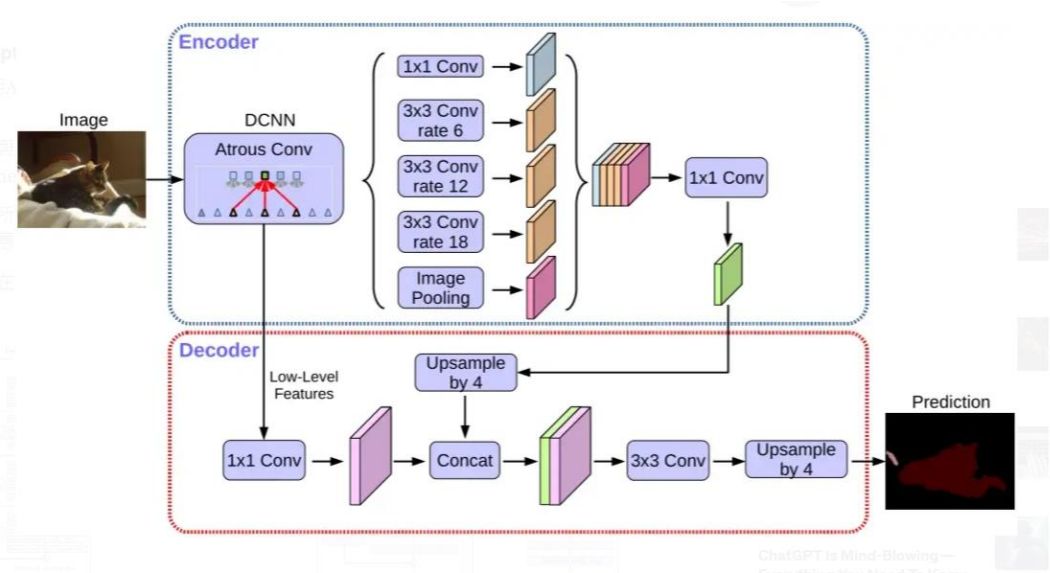
Deeper w. Atrous Convolution: 在原始模型的頂端疊加額外的模塊，以捕捉像素間長距離信息。例如 Dense CRF，或者疊加一些其他的捲積層。

Spatial Pyramid Pooling:使用不同採樣率和多種視野的捲積核，以捕捉多尺度對象。

2. Deeplab 實作方式

Encoder → DeepLab V3，修改 ResNet101 最後兩(一)個 block 的 stride，使得 output stride 為 8(研究發現 stride=16 在速度和精度之間可取得最佳平衡，步幅=8 用於編碼器模塊時，性能略有提高，但代價是額外的計算量)。之後在 block4 後使用改良的 Atrous Spatial Pyramid Pooling，將所得的特徵圖 concatenate，再用 1×1 的捲積得到 256 個通道的特徵圖。

decoder → 特徵圖首先上採樣 4 倍，然後與 encoder 中對應分辨率低級特徵 concatenate。在 concatenate 之前，由於低級特徵圖的通道數通常太多(256 或 512)，從 encoder 中得到的富含語義信息的特徵圖通道數只有 256，這樣會淡化語義信息，讓訓練變得更困難。因此在 concatenate 之前，需要將低級特徵圖通過 1×1 的捲積減少通道數。在 concatenate 之後用 3×3 的捲積改善特徵，最後上採樣 4 倍恢復到原始圖像大小。



3. EdgeAI 平台介紹

EdgeAI 平台採用樹莓派 3B+，搭配 Google TPU，而將 AI 模型移動至樹莓派時，需使用 TensorFlowLite 檔，壓縮模型大小，以減少樹莓派空間的浪費並加快讀取速度。

4. EdgeAI 實作方式

下載 win32 燒入器，至樹梅派官網下載作業系統，在 win32 燒入器上選擇載下來的作業系統壓縮包，插上磁碟卡並燒入作業系統。

將記憶卡插入樹莓派並開機，經過簡單的初始設定後開啟 Terminal，並依序輸入以下指令，安裝相關套件。

```
1 echo "deb https://packages.cloud.google.com/apt coral-edgetpu-stable main" | sudo tee /etc/apt/sources.list.d/coral-edgetpu.list
2 curl https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -
3 sudo apt-get update
4 sudo apt-get install libedgetpu1-std
5 sudo apt-get install python3-edgetpu
6 sudo apt-get install edgetpu-examples
7 sudo pip3 install jupyterlab opencv-python matplotlib
8 sudo apt-get install python3-pycoral
```

建好環境後，將「tflite 模型」、「label」、「測試圖片」、「py 執行檔」複製到樹莓派後，並下達以下指令，等待一段時間後即可看到影像辨識的結果。

```
python3 py執行檔 --model tflite模型 --labels label --input 測試圖片
```

5. 輕量化 tflite 轉換

利用虛擬機安裝 ubuntu 系統，下達以下指令來建置環境:

```
1 echo "deb https://packages.cloud.google.com/apt coral-edgetpu-stable main" | sudo tee /etc/apt/sources.list.d/coral-edgetpu.list
2 curl https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -
3 sudo apt-get update
4 sudo apt-get install edgetpu-compiler
```

將 h5 檔案轉換成 tflite 檔案並傳輸至虛擬機中，執行以下指令來進行壓縮優化:

```
edgetpu_compiler tflite模型
```

最後再將優化後的模型複製到樹梅派即可。

三、 程式說明

1. Deeplab 模型及訓練結果

匯入套件

```
import datetime
import os
from functools import partial

import numpy as np
import tensorflow as tf
import tensorflow.keras.backend as K
from tensorflow.keras.callbacks import (EarlyStopping, LearningRateScheduler,
                                        TensorBoard)
from tensorflow.keras.optimizers import SGD, Adam

from nets.deeplab import Deeplabv3
from nets.deeplab_training import (CE, Focal_Loss, dice_loss_with_CE,
                                   dice_loss_with_Focal_Loss, get_lr_scheduler)
from utils.callbacks import EvalCallback, LossHistory, ModelCheckpoint
from utils.dataloader import DeeplabDataset
from utils.utils import show_config
from utils.utils_fit import fit_one_epoch
from utils.utils_metrics import Iou_score, f_score
```

建立/設定參數、資料集

```
if __name__ == "__main__":
    #---------------------------------------------------------------------#
    #   是否使用eager模式训练
    #---------------------------------------------------------------------#
    eager           = False
    #-----------------------------------------------------------------------#
    #   train_gpu   训练用到的GPU
    #                   默认为第一张卡、双卡为[0, 1]、三卡为[0, 1, 2]
    #                   在使用多GPU时，每个卡上的batch为总batch除以卡的数量。
    #-----------------------------------------------------------------------#
    train_gpu   = [0,]
    #---------------------------------------------------------------------#
    #   num_classes     训练自己的数据集必须要修改的
    #                       自己需要的分类个数+1，如2+1
    #---------------------------------------------------------------------#
    num_classes = 21
    #-----------------------------#
    #   所使用的的主干网络：
    #   mobilenet
    #   xception
    #-----------------------------#
    backbone    = "mobilenet"
    #----------------------------------------------------------------------------------------------------------------------------#
    #   权值文件的下载请看README，可以通过网盘下载。模型的 预训练权重 对不同数据集是通用的，因为特征是通用的。
    #   模型的 预训练权重 比较重要的部分是 主干特征提取网络的权值部分，用于进行特征提取。
    #   预训练权重对于99%的情况都必须要用，不用的话主干部分的权值太过随机，特征提取效果不明显，网络训练的结果也不会好
    #   训练自己的数据集时提示维度不匹配正常，预测的东西都不一样了自然维度不匹配
    #
    #   如果训练过程中存在中断训练的操作，可以将model_path设置成logs文件夹下的权值文件，将已经训练了一部分的权值再次载入。
    #   同时修改下方的 冻结阶段 或者 解冻阶段 的参数，来保证模型epoch的连续性。
    #
    #   当model_path = ''的时候不加载整个模型的权值。
    #
    #   此处使用的是整个模型的权重，因此是在train.py进行加载的。
    #   如果想要让模型从主干的预训练权值开始训练，则设置model_path为主干网络的权值，此时仅加载主干。
    #   如果想要让模型从0开始训练，则设置model_path = ''，下面的Freeze_Train = Fasle，此时从0开始训练，且没有冻结主干的过程。
    #   一般来讲，从0开始训练效果会很差，因为权值太过随机，特征提取效果不明显。
    #
    #   网络一般不从0开始训练，至少会使用主干部分的权值，有些论文提到可以不用预训练，主要原因是他们 数据集较大 且 调参能力优秀，
    #   如果一定要训练网络的主干部分，可以了解imagenet数据集，首先训练分类模型，分类模型的 主干部分 和该模型通用，基于此进行训练。
    #----------------------------------------------------------------------------------------------------------------------------#
    model_path  = "model_data/deeplabv3_mobilenetv2.h5"
    #---------------------------------------------------------#
    #   downsample_factor   下采样的倍数8、16
    #                       8下采样的倍数较小、理论上效果更好。
    #                       但也要求更大的显存
    #---------------------------------------------------------#
    downsample_factor   = 16
    #-----------------------------#
    #   输入图片的大小
    #-----------------------------#
    input_shape         = [512, 512]
```

```
#------------------------------------------------------------------------------------------------------------------#
#       训练分为两个阶段，分别是冻结阶段和解冻阶段，设置冻结阶段是为了满足机器性能不足的同学的训练需求。
#       冻结训练需要的显存较小，显卡非常差的情况下，可设置Freeze_Epoch等于UnFreeze_Epoch，此时仅仅进行冻结训练。
#
#       在此提供若干参数设置建议，各位训练者根据自己的需求进行灵活调整：
#       （一）从整个模型的预训练权重开始训练：
#             Adam：
#                   Init_Epoch = 0，Freeze_Epoch = 50，UnFreeze_Epoch = 100，Freeze_Train = True，optimizer_type = 'adam'，Init_lr = 5e-4。（冻结）
#                   Init_Epoch = 0，UnFreeze_Epoch = 100，Freeze_Train = False，optimizer_type = 'adam'，Init_lr = 5e-4。（不冻结）
#             SGD：
#                   Init_Epoch = 0，Freeze_Epoch = 50，UnFreeze_Epoch = 100，Freeze_Train = True，optimizer_type = 'sgd'，Init_lr = 7e-3。（冻结）
#                   Init_Epoch = 0，UnFreeze_Epoch = 100，Freeze_Train = False，optimizer_type = 'sgd'，Init_lr = 7e-3。（不冻结）
#             其中：UnFreeze_Epoch可以在100-300之间调整。
#       （二）从主干网络的预训练权重开始训练：
#             Adam：
#                   Init_Epoch = 0，Freeze_Epoch = 50，UnFreeze_Epoch = 100，Freeze_Train = True，optimizer_type = 'adam'，Init_lr = 5e-4。（冻结）
#                   Init_Epoch = 0，UnFreeze_Epoch = 100，Freeze_Train = False，optimizer_type = 'adam'，Init_lr = 5e-4。（不冻结）
#             SGD：
#                   Init_Epoch = 0，Freeze_Epoch = 50，UnFreeze_Epoch = 120，Freeze_Train = True，optimizer_type = 'sgd'，Init_lr = 7e-3。（冻结）
#                   Init_Epoch = 0，UnFreeze_Epoch = 120，Freeze_Train = False，optimizer_type = 'sgd'，Init_lr = 7e-3。（不冻结）
#             其中：由于从主干网络的预训练权重开始训练，主干的权值不一定适合语义分割，需要更多的训练跳出局部最优解。
#                   UnFreeze_Epoch可以在120-300之间调整。
#                   Adam相较于SGD收敛的快一些。因此UnFreeze_Epoch理论上可以小一点，但依然推荐更多的Epoch。
#       （三）batch_size的设置：
#             在显卡能够接受的范围内，以大为好。显存不足与数据集大小无关，提示显存不足（OOM或者CUDA out of memory）请调小batch_size。
#             受到BatchNorm层影响，batch_size最小为2，不能为1。
#             正常情况下Freeze_batch_size建议为Unfreeze_batch_size的1-2倍。不建议设置的差距过大，因为关系到学习率的自动调整。
#------------------------------------------------------------------------------------------------------------------#
#------------------------------------------------------#
#       冻结阶段训练参数
#       此时模型的主干被冻结了，特征提取网络不发生改变
#       占用的显存较小，仅对网络进行微调
#       Init_Epoch          模型当前开始的训练世代，其值可以大于Freeze_Epoch，如设置：
#                                   Init_Epoch = 60、Freeze_Epoch = 50、UnFreeze_Epoch = 100
#                                   会跳过冻结阶段，直接从60代开始，并调整对应的学习率。
#                                   （断点续练时使用）
#       Freeze_Epoch        模型冻结训练的Freeze_Epoch
#                                   （当Freeze_Train=False时失效）
#       Freeze_batch_size   模型冻结训练的batch_size
#                                   （当Freeze_Train=False时失效）
#------------------------------------------------------#
Init_Epoch          = 0
Freeze_Epoch        = 50
Freeze_batch_size   = 8
#------------------------------------------------------#
#       解冻阶段训练参数
#       此时模型的主干不被冻结了，特征提取网络会发生改变
#       占用的显存较大，网络所有的参数都会发生改变
#       UnFreeze_Epoch          模型总共训练的epoch
#       Unfreeze_batch_size     模型在解冻后的batch_size
#------------------------------------------------------#
UnFreeze_Epoch      = 100
Unfreeze_batch_size = 4
#------------------------------------------------------#
```

```
#------------------------------------------------------------------#
#       Freeze_Train        是否进行冻结训练
#                           默认先冻结主干训练后解冻训练。
#------------------------------------------------------------------#
Freeze_Train        = True

#------------------------------------------------------------------#
#       其它训练参数：学习率、优化器、学习率下降有关
#------------------------------------------------------------------#
#------------------------------------------------------------------#
#       Init_lr             模型的最大学习率
#                           当使用Adam优化器时建议设置  Init_lr=5e-4
#                           当使用SGD优化器时建议设置   Init_lr=7e-3
#       Min_lr              模型的最小学习率，默认为最大学习率的0.01
#------------------------------------------------------------------#
Init_lr             = 7e-3
Min_lr              = Init_lr * 0.01
#------------------------------------------------------------------#
#       optimizer_type      使用到的优化器种类，可选的有adam、sgd
#                           当使用Adam优化器时建议设置  Init_lr=5e-4
#                           当使用SGD优化器时建议设置   Init_lr=7e-3
#       momentum            优化器内部使用到的momentum参数
#------------------------------------------------------------------#
optimizer_type      = "sgd"
momentum            = 0.9
#------------------------------------------------------------------#
#       lr_decay_type       使用到的学习率下降方式，可选的有'step'、'cos'
#------------------------------------------------------------------#
lr_decay_type       = 'cos'
#------------------------------------------------------------------#
#       save_period         多少个epoch保存一次权值
#------------------------------------------------------------------#
save_period         = 5
#------------------------------------------------------------------#
#       save_dir            权值与日志文件保存的文件夹
#------------------------------------------------------------------#
save_dir            = 'logs'
#------------------------------------------------------------------#
#       eval_flag           是否在训练时进行评估，评估对象为验证集
#       eval_period         代表多少个epoch评估一次，不建议频繁的评估
#                           评估需要消耗较多的时间，频繁评估会导致训练非常慢
#       此处获得的mAP会与get_map.py获得的会有所不同，原因有二：
#       （一）此处获得的mAP为验证集的mAP。
#       （二）此处设置评估参数较为保守，目的是加快评估速度。
#------------------------------------------------------------------#
eval_flag           = True
eval_period         = 5

#------------------------------------------------------------------#
#       VOCdevkit_path  数据集路径
#------------------------------------------------------------------#
VOCdevkit_path      = 'VOCdevkit'
```

```
#------------------------------------------------------------------#
#       建议选项：
#       种类少（几类）时，设置为True
#       种类多（十几类）时，如果batch_size比较大（10以上），那么设置为True
#       种类多（十几类）时，如果batch_size比较小（10以下），那么设置为False
#------------------------------------------------------------------#
dice_loss               = False
#------------------------------------------------------------------#
#       是否使用focal  loss来防止正负样本不平衡
#------------------------------------------------------------------#
focal_loss              = False
```

訓練模型

```
#------------------------------#
#       开始模型训练
#------------------------------#
for epoch in range(start_epoch, end_epoch):
        #---------------------------------#
        #       如果模型有冻结学习部分
        #       则解冻，并设置参数
        #---------------------------------#
        if epoch >= Freeze_Epoch and not UnFreeze_flag and Freeze_Train:
                batch_size          = Unfreeze_batch_size

                #-------------------------------------------------------------------#
                #       判断当前batch_size，自适应调整学习率
                #-------------------------------------------------------------------#
                nbs                         = 16
                lr_limit_max        = 5e-4 if optimizer_type == 'adam' else 1e-1
                lr_limit_min        = 3e-4 if optimizer_type == 'adam' else 5e-4
                if backbone == "xception":
                        lr_limit_max        = 1e-4 if optimizer_type == 'adam' else 1e-1
                        lr_limit_min        = 1e-4 if optimizer_type == 'adam' else 5e-4
                Init_lr_fit         = min(max(batch_size / nbs * Init_lr, lr_limit_min), lr_limit_max)
                Min_lr_fit          = min(max(batch_size / nbs * Min_lr, lr_limit_min * 1e-2), lr_limit_max * 1e-2)
                #---------------------------------#
                #       获得学习率下降的公式
                #---------------------------------#
                lr_scheduler_func = get_lr_scheduler(lr_decay_type, Init_lr_fit, Min_lr_fit, UnFreeze_Epoch)

                for i in range(len(model.layers)):
                        model.layers[i].trainable = True

                epoch_step          = num_train // batch_size
                epoch_step_val      = num_val // batch_size

                if epoch_step == 0 or epoch_step_val == 0:
                        raise ValueError("数据集过小，无法继续进行训练，请扩充数据集。")

                train_dataloader.batch_size         = batch_size
                val_dataloader.batch_size           = batch_size
```

```
                gen         = tf.data.Dataset.from_generator(partial(train_dataloader.generate), (tf.float32, tf.float32))
                gen_val     = tf.data.Dataset.from_generator(partial(val_dataloader.generate), (tf.float32, tf.float32))

                gen         = gen.shuffle(buffer_size = batch_size).prefetch(buffer_size = batch_size)
                gen_val     = gen_val.shuffle(buffer_size = batch_size).prefetch(buffer_size = batch_size)

                if ngpus_per_node > 1:
                        gen         = strategy.experimental_distribute_dataset(gen)
                        gen_val     = strategy.experimental_distribute_dataset(gen_val)

                UnFreeze_flag = True

        lr = lr_scheduler_func(epoch)
        K.set_value(optimizer.lr, lr)

        fit_one_epoch(model, loss, loss_history, eval_callback, optimizer, epoch, epoch_step, epoch_step_val, gen, gen_val,
                                end_epoch, f_score(), save_period, save_dir, strategy)

        train_dataloader.on_epoch_end()
        val_dataloader.on_epoch_end()
```

```
else:
        start_epoch = Init_Epoch
        end_epoch = Freeze_Epoch if Freeze_Train else UnFreeze_Epoch

        if ngpus_per_node > 1:
                with strategy.scope():
                        model.compile(loss = loss,
                                      optimizer = optimizer,
                                      metrics = [f_score()])
        else:
                model.compile(loss = loss,
                              optimizer = optimizer,
                              metrics = [f_score()])

        #-------------------------------------------------------------------------------#
        #       训练参数的设置
        #       logging                 用于设置tensorboard的保存地址
        #       checkpoint              用于设置权值保存的细节，period用于修改多少epoch保存一次
        #       lr_scheduler            用于设置学习率下降的方式
        #       early_stopping      用于设定早停，val_loss多次不下降自动结束训练，表示模型基本收敛
        #-------------------------------------------------------------------------------#
        time_str                = datetime.datetime.strftime(datetime.datetime.now(),'%Y_%m_%d_%H_%M_%S')
        log_dir                 = os.path.join(save_dir, "loss_" + str(time_str))
        logging                 = TensorBoard(log_dir)
        loss_history        = LossHistory(log_dir)
        checkpoint          = ModelCheckpoint(os.path.join(save_dir, "ep{epoch:03d}-loss{loss:.3f}-val_loss{val_loss:.3f}.h5"),
                                              monitor = 'val_loss', save_weights_only = True, save_best_only = False, period = save_period)
        checkpoint_last = ModelCheckpoint(os.path.join(save_dir, "last_epoch_weights.h5"),
                                              monitor = 'val_loss', save_weights_only = True, save_best_only = False, period = 1)
        checkpoint_best = ModelCheckpoint(os.path.join(save_dir, "best_epoch_weights.h5"),
                                              monitor = 'val_loss', save_weights_only = True, save_best_only = True, period = 1)
        early_stopping    = EarlyStopping(monitor='val_loss', min_delta = 0, patience = 10, verbose = 1)
        lr_scheduler        = LearningRateScheduler(lr_scheduler_func, verbose = 1)
        eval_callback     = EvalCallback(model, input_shape, num_classes, val_lines, VOCdevkit_path, log_dir, \
                                                                        eval_flag=eval_flag, period=eval_period)
        callbacks             = [logging, loss_history, checkpoint, checkpoint_last, checkpoint_best, lr_scheduler, eval_callback]
```

```
if start_epoch < end_epoch:
        print('Train on {} samples, val on {} samples, with batch size {}.'.format(num_train, num_val, batch_size))
        model.fit(
                x                                       = train_dataloader,
                steps_per_epoch         = epoch_step,
                validation_data         = val_dataloader,
                validation_steps        = epoch_step_val,
                epochs                                  = end_epoch,
                initial_epoch                   = start_epoch,
                use_multiprocessing = True if num_workers > 1 else False,
                workers                                 = num_workers,
                callbacks                       = callbacks
        )
#---------------------------------#
#       如果模型有冻结学习部分
#       则解冻，并设置参数
#---------------------------------#
if Freeze_Train:
        batch_size    = Unfreeze_batch_size
        start_epoch = Freeze_Epoch if start_epoch < Freeze_Epoch else start_epoch
        end_epoch     = UnFreeze_Epoch

        #-------------------------------------------------------------------#
        #       判断当前batch_size，自适应调整学习率
        #-------------------------------------------------------------------#
        nbs                     = 16
        lr_limit_max        = 5e-4 if optimizer_type == 'adam' else 1e-1
        lr_limit_min        = 3e-4 if optimizer_type == 'adam' else 5e-4
        if backbone == "xception":
                lr_limit_max        = 1e-4 if optimizer_type == 'adam' else 1e-1
                lr_limit_min        = 1e-4 if optimizer_type == 'adam' else 5e-4
        Init_lr_fit         = min(max(batch_size / nbs * Init_lr, lr_limit_min), lr_limit_max)
        Min_lr_fit          = min(max(batch_size / nbs * Min_lr, lr_limit_min * 1e-2), lr_limit_max * 1e-2)
        #---------------------------------#
        #       获得学习率下降的公式
        #---------------------------------#
        lr_scheduler_func = get_lr_scheduler(lr_decay_type, Init_lr_fit, Min_lr_fit, UnFreeze_Epoch)
        lr_scheduler        = LearningRateScheduler(lr_scheduler_func, verbose = 1)
        callbacks             = [logging, loss_history, checkpoint, checkpoint_last, checkpoint_best, lr_scheduler, eval_callback]
```

```
for i in range(len(model.layers)):
        model.layers[i].trainable = True
if ngpus_per_node > 1:
        with strategy.scope():
                model.compile(loss = loss,
                              optimizer = optimizer,
                              metrics = [f_score()])
else:
        model.compile(loss = loss,
                      optimizer = optimizer,
                      metrics = [f_score()])

epoch_step          = num_train // batch_size
epoch_step_val    = num_val // batch_size

if epoch_step == 0 or epoch_step_val == 0:
        raise ValueError("数据集过小，无法继续进行训练，请扩充数据集。")

train_dataloader.batch_size        = Unfreeze_batch_size
val_dataloader.batch_size          = Unfreeze_batch_size

print('Train on {} samples, val on {} samples, with batch size {}.'.format(num_train, num_val, batch_size))
model.fit(
        x                                  = train_dataloader,
        steps_per_epoch          = epoch_step,
        validation_data          = val_dataloader,
        validation_steps         = epoch_step_val,
        epochs                           = end_epoch,
        initial_epoch              = start_epoch,
        use_multiprocessing = True if num_workers > 1 else False,
        workers                          = num_workers,
        callbacks                    = callbacks
)
```

2. h5 檔轉換至 tflite 檔

匯入套件

```
import tensorflow as tf
from tensorflow.keras import Model
import numpy as np
import glob
tf.__version__
```

轉換副程式

```
IMAGE_SIZE = 512
def representative_data_gen():
        dataset_list=glob.glob('VOCdevkit/VOC2007/JPEGImages/*')
        dataset_list_index=np.random.choice(range(len(dataset_list)),100)
        for i in range(100):
                image = dataset_list[dataset_list_index[i]]
                print(image)
                image = tf.io.read_file(image)
                image = tf.io.decode_jpeg(image,channels=3)
                image = tf.image.resize(image,[IMAGE_SIZE, IMAGE_SIZE])
                image = tf.cast(image,tf.float32)
                image = tf.expand_dims(image,0)
                #with tf.Session() as sess:
                #       image = sess.run(image)
                yield [image]
```

載入模型檔案

```
save_keras_model = tf.keras.models.load_model('./export/deeplab.h5',custom_objects={'tf':tf})
save_keras_model.summary()
```

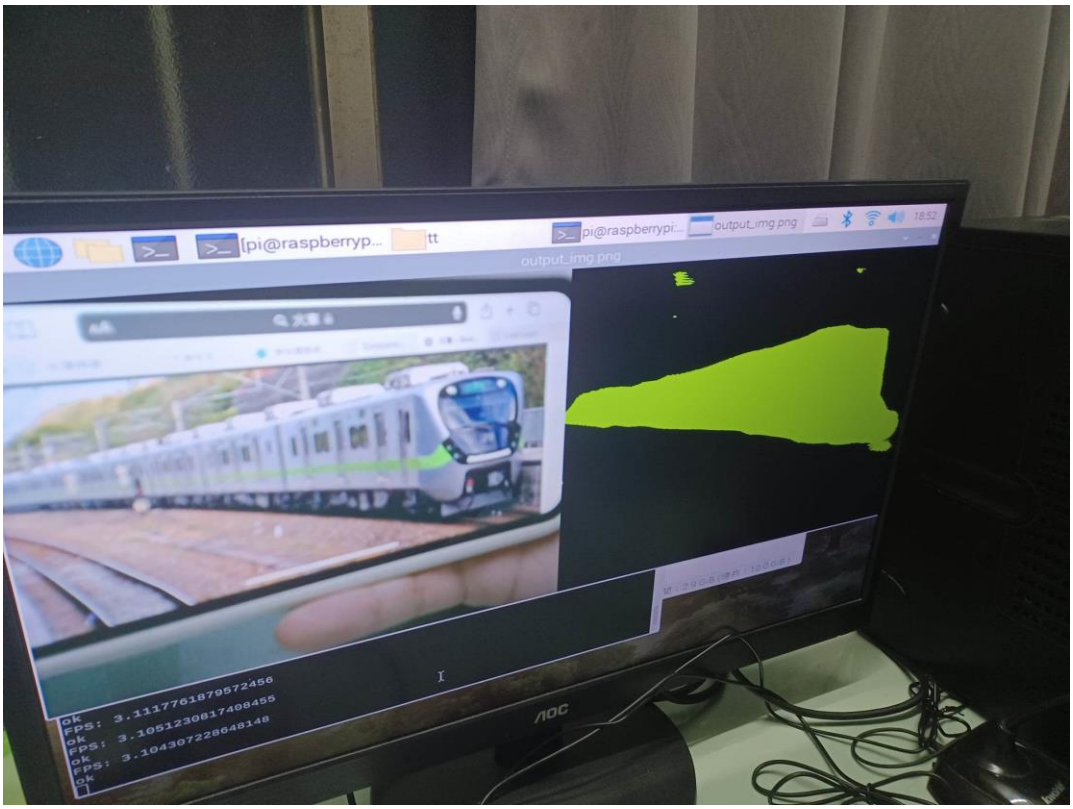
轉換模型

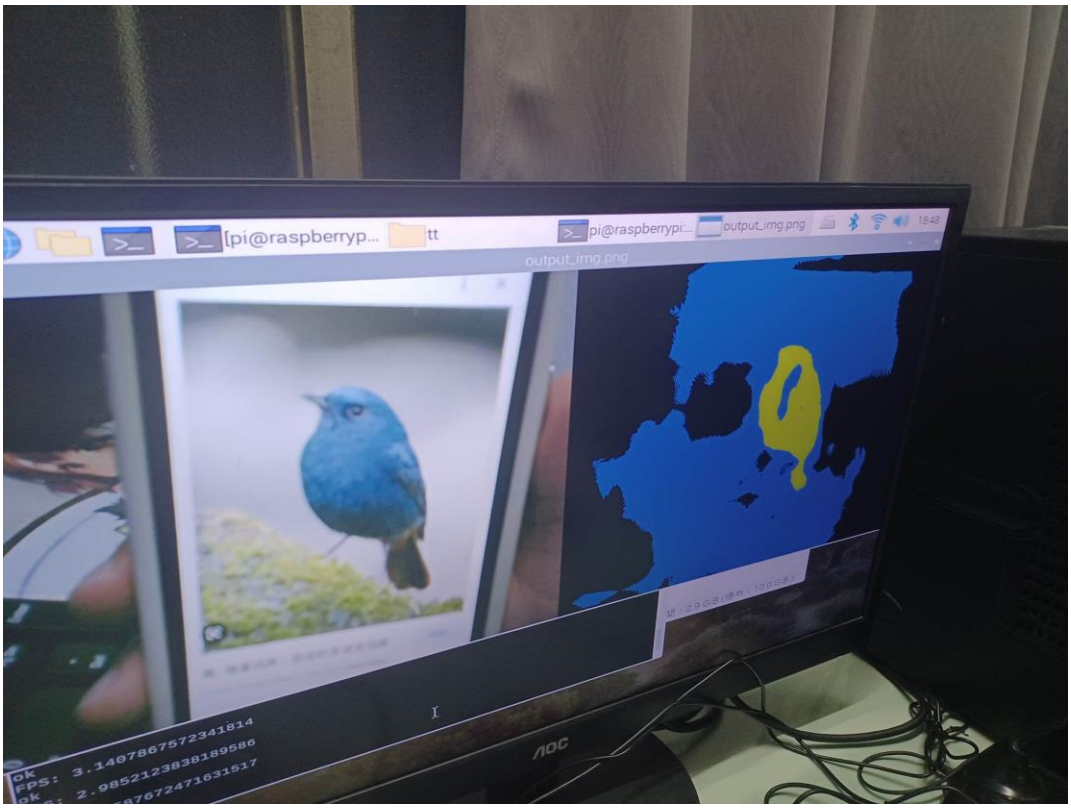
```
if tf.__version__[0]=='2':
        converter = tf.lite.TFLiteConverter.from_keras_model(save_keras_model)
elif tf.__version__[0]=='1':
        converter = tf.lite.TFLiteConverter.from_keras_model_file('./export/deeplab.h5')
converter.optimizations = [tf.lite.Optimize.DEFAULT]
converter.target_spec.supported_ops = [tf. lite.OpsSet.TFLITE_BUILTINS_INT8]
converter.inference_input_type = tf.uint8
converter.inference_output_type = tf.uint8
converter.representative_dataset = representative_data_gen
tflite_model = converter.convert()
```

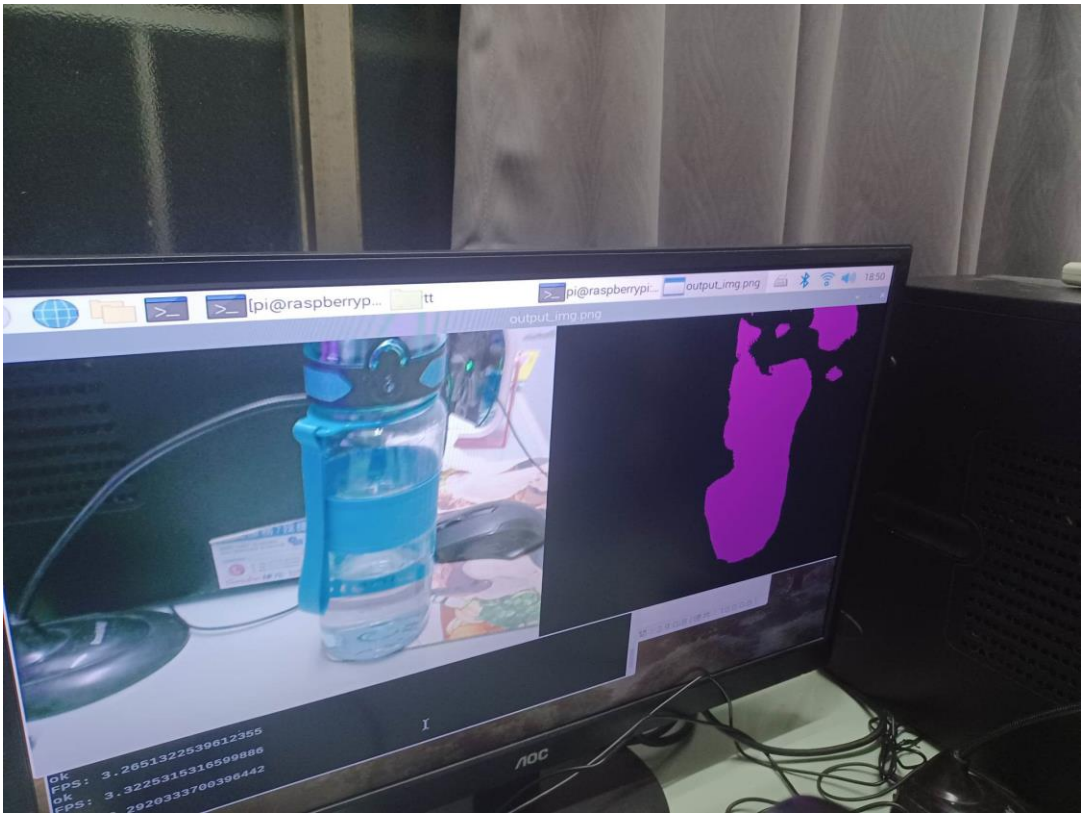
寫入模型

```
with open('deeplab.tflite','wb') as f:
        f.write(tflite_model)
```

四、 EdgeAI 平台驗證結果


[pi@raspberryp...
tt
pi@raspberrypi:...
output_img.png
output_img.png
ok
FPS: 3.1117761879572456
ok
FPS: 3.1051230817408455
ok
FPS: 3.1043072228648148
ok
AOC


[pi@raspberryp...
tt
pi@raspberrypi:...
output_img.png
output_img.png
ok
FPS: 3.1407867572341814
AOC


[pi@raspberryp...
tt
pi@raspberrypi...
output_img.png
18:50
output_img.png
ok
FPS: 3.265132263961235 5
ok
FPS: 3.3225315316599886
AOC

五、 心得與討論

陳維翔:

這次做語意切割,我覺得結果很有趣,用不同的顏色去識別物體，很有特色，這次 lab 做得很快，因為學長有給訓練完的模型，謝謝學長。

曹宸維:

這次的 lab 因為學長說訓練要很久，所以我們直接拿學長訓練完的結果來做測試，用攝像頭時感覺它的 fps 很低，應該是因為模型的容量太大，導致運算量太大讓它跑得慢。

林廷緯:

從這次的 lab 中我們學到了語意切割的用法，可以針對你想要去識別的物體用顏色和周圍的事物做區分,也知道了其架構和原理,我們也成功實做完了，總之很有幫助。