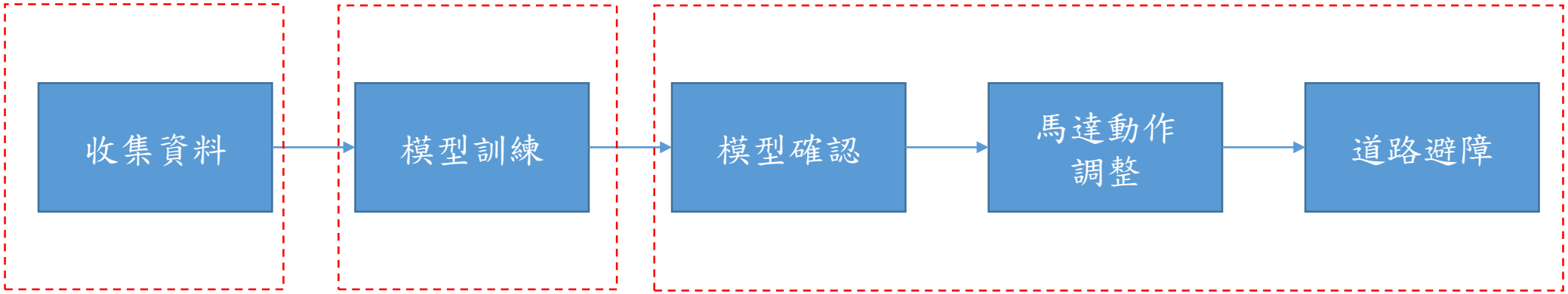
嵌入式智慧影像分析與實境界面
Fall 2021

Instructor：Yen-Lin Chen(陳彥霖), Ph.D.
Professor
Dept. Computer Science and Information Engineering
National Taipei University of Technology

Project 6

應用輕量化深度學習網路技術進行道路避障

道路避障建議流程圖



data_collection.ipynb　train_model.ipynb　live_demo.ipynb

道路避障

- 使用程式檔案路徑:Nvidia/jetbot/notebooks/collision_avoidance
- 按照順序使用的檔案名稱:
  - 1. data_collection.ipynb
  - 2. train_model.ipynb
  - 3. live_demo.ipynb

收集資料

data_collection.ipynb

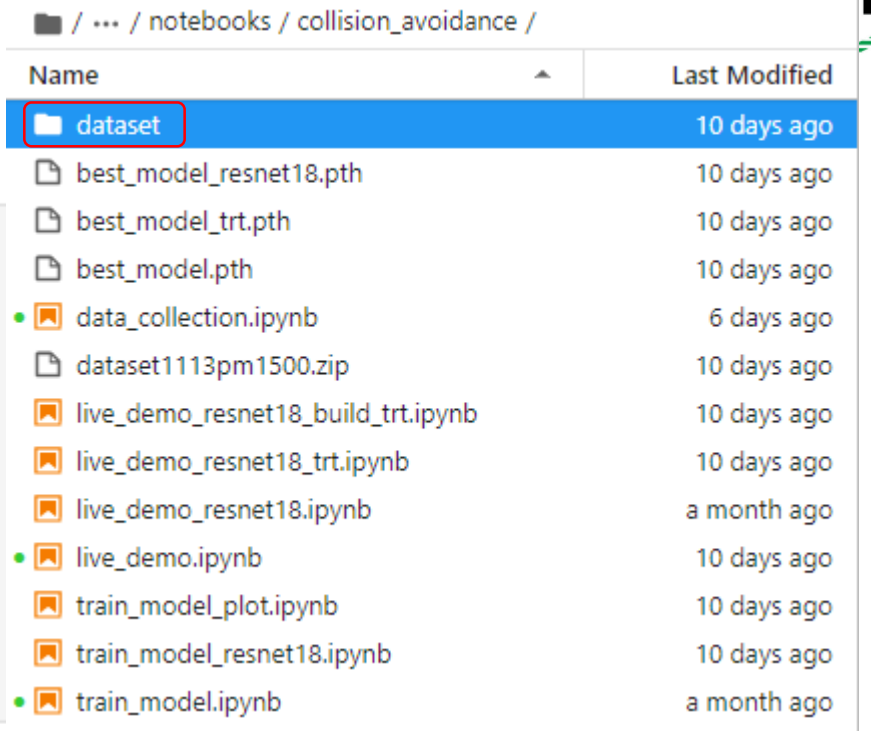
定義相機格式

```
1  import traitlets
2  import ipywidgets.widgets as widgets
3  from IPython.display import display
4  from jetbot import Camera, bgr8_to_jpeg
5
6  camera = Camera.instance(width=224, height=224)
7
8  image = widgets.Image(format='jpeg', width=224, height=224)  # this width and height doesn't necessarily have to match the camera
9
10 camera_link = traitlets.dlink((camera, 'value'), (image, 'value'), transform=bgr8_to_jpeg)
11
12 display(image)
```

- [6]初始化相機影像大小，由於神經網路輸入影像為224*224，所以使用widgets.Image將影像儲存為224*224且jpeg的格式。
- 相機影像大小將會影響記憶體的使用多寡 。
- [8]定義用於[12]顯示的影像格式。
- [10]將兩個目標(camera, image)進行連接，且影像從bgr8轉成jpeg格式。

建立資料夾

```
1  import os
2
3  blocked_dir = 'dataset/blocked'
4  free_dir = 'dataset/free'
5
6  # we have this "try/except" statement because these next functions
7  try:
8      os.makedirs(free_dir)
9      os.makedirs(blocked_dir)
10 except FileExistsError:
11     print('Directories not created becasue they already exist')
```



- 在基本範例中，[3]~[9]建立一個dataset資料夾，其中包含free和blocked的子資料夾，用來儲存每種類別的影像。
- [10][11]如果資料夾存在於目錄中，則顯示’Directories not created because they already exist’。

產生按鍵與連結功能

```
1  button_layout = widgets.Layout(width='128px', height='64px')
2  free_button = widgets.Button(description='add free', button_style='success', layout=button_layout)
3  blocked_button = widgets.Button(description='add blocked', button_style='danger', layout=button_layout)
4  free_count = widgets.IntText(layout=button_layout, value=len(os.listdir(free_dir)))
5  blocked_count = widgets.IntText(layout=button_layout, value=len(os.listdir(blocked_dir)))
6
7  display(widgets.HBox([free_count, free_button]))
8  display(widgets.HBox([blocked_count, blocked_button]))
```

- [1]~[3]建立free與blocked按鍵，free表示對於Jetbot可行駛的道路影像，blocked表示對於Jetbot不可行駛的道路影像。
- [4][5]在按鍵左側會各自產生一個文字框，上方的文字框表示free的道路影像數量;下方的文字框表示blocked的道路影像數量。
- 建議在收集影像時，各類別的影像數量盡量一致，確保模型不會偏向某一類別，導致物件辨識結果不佳。
- 影像的數量也影響到辨識的準確度，各組可以找出最佳準確度的影像數量。
- 現在這些按鍵不會執行任何動作，後面會將按鍵加入到on_click功能中。

定義按鍵功能

- save_snapshot : 將儲存每張影像的數值。
- save_free : 儲存free (可通過的影像)類別的影像數值。
- save_blocked : 儲存blocked(障礙物的影像)類別的影像數值。
- 在產生影像時為了不重複任何檔案名稱，使用python中的uuid功能，此功能是根據日期、時間和MAC address的資訊產生，用來確保影像名稱是唯一的。

```python
from uuid import uuid1

def save_snapshot(directory):
    image_path = os.path.join(directory, str(uuid1()) + '.jpg')
    with open(image_path, 'wb') as f:
        f.write(image.value)

def save_free():
    global free_dir, free_count
    save_snapshot(free_dir)
    free_count.value = len(os.listdir(free_dir))

def save_blocked():
    global blocked_dir, blocked_count
    save_snapshot(blocked_dir)
    blocked_count.value = len(os.listdir(blocked_dir))

# attach the callbacks, we use a 'lambda' function to ignore the
# parameter that the on_click event would provide to our function
# because we don't need it.
free_button.on_click(lambda x: save_free())
blocked_button.on_click(lambda x: save_blocked())
```

收集資料

```
display(image) 顯示即時影像
display(widgets.HBox([free_count, free_button]))
display(widgets.HBox([blocked_count, blocked_button]))
```

計數功能與按鍵連接

30 add free 可通過的影像

30 add blocked 障礙物的影像

```
camera.stop() 停止相機
```

收集資料:

1.將機器人放置於障礙物的情境中，然後按下add block

2.將機器人放置於可通過的情境中，然後按下add free

3.然後重複動作1.、2.，直到影像數量足夠為止(各組請自行決定數量)

標記資料的技巧:

1.嘗試不同的方向

2.嘗試不同的光線

3.嘗試各種物體/碰撞類型，例如:牆壁、其他物體等

4.嘗試使用其他含有紋理的地板/物體，例如:玻璃等

訓練模型

train_model.ipynb

引用Pytorch函示庫

- 在範例中，使用Pytorch函示庫來完成接下來的模型訓練。

```
1  import torch   提供多維的資料結構及數學運算
2  import torch.optim as optim  最佳化演算法
3  import torch.nn.functional as F  卷積函數
4  import torchvision  開放原始碼的機器學習框架
5  import torchvision.datasets as datasets 自定義資料集
6  import torchvision.models as models  模型架構
7  import torchvision.transforms as transforms  用於影像變換
```

建立資料集格式

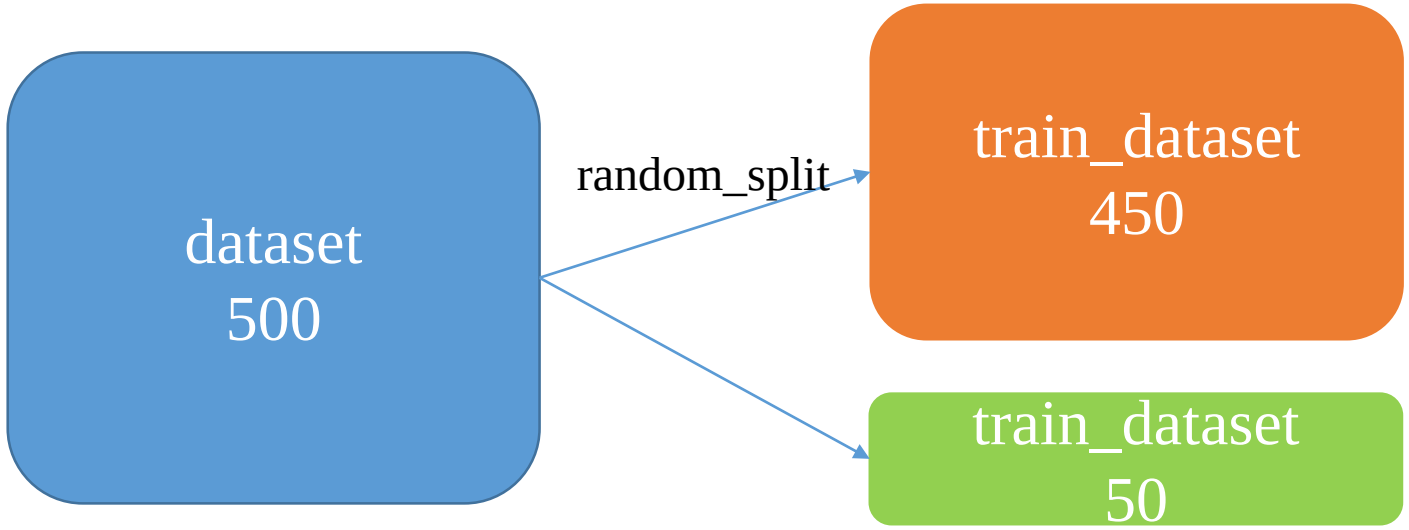
```
1  dataset = datasets.ImageFolder(
2      'dataset', 資料夾名稱
3      transforms.Compose([ 組合transforms所需的變數
4          transforms.ColorJitter(0.1, 0.1, 0.1, 0.1), 隨機更改影像亮度、對比度、飽和度和色調
5          transforms.Resize((224, 224)), 設定影像大小
6          transforms.ToTensor(), 轉換影像至Tensor
7          transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
8      ]) 使用平均值和標準差對tensor影像進行正規化
9  )
```

- [1][2]指定要使用的資料夾。
- [3]~[8]組合轉換所需要的變數。
- [4]隨機更改影像數值。
- [5]重新設定影像大小。
- [6]轉換影像至Tensor。
- [7]使用平均值和標準差對tensor影像進行正規化。

分割訓練和測試資料集

```
train_dataset, test_dataset = torch.utils.data.random_split(dataset, [len(dataset) - 50, 50])
```

- torch.utils.data.random_split( ) : 將資料集隨機分割，並依照參數產生不重疊且新的資料集。
- 假設dataset大小為500張影像，則train_dataset的影像數量為450張、test_dataset為50張。(建議為Training 90%, Testing 10%)



建立資料讀取格式

```
train_loader = torch.utils.data.DataLoader(
    train_dataset,
    batch_size=8,
    shuffle=True,
    num_workers=0
)

test_loader = torch.utils.data.DataLoader(
    test_dataset,
    batch_size=8,
    shuffle=True,
    num_workers=0
)
```

Train_dataset : 訓練資料集。
Batch_size :訓練批次大小。
Shuffle :是否隨機順序讀取影像。
Num_works :設定執行緒，0代表只用單執行緒進行訓練。
Nano最大可以設定到4，使用全部的核心進行訓練。

- 使用DataLoader來批次讀取資料，混淆資料並允許使用多個執行緒。
- Batch_size將影響到模型的最佳化程度和速度，範例將批次大小設定為8。批次處理大小是基於輸入影像大小與GPU實際可用的記憶體，設定值太大可能會造成訓練錯誤，太小訓練次數會變多，這會影響模型的準確性。

使用Pre-train model

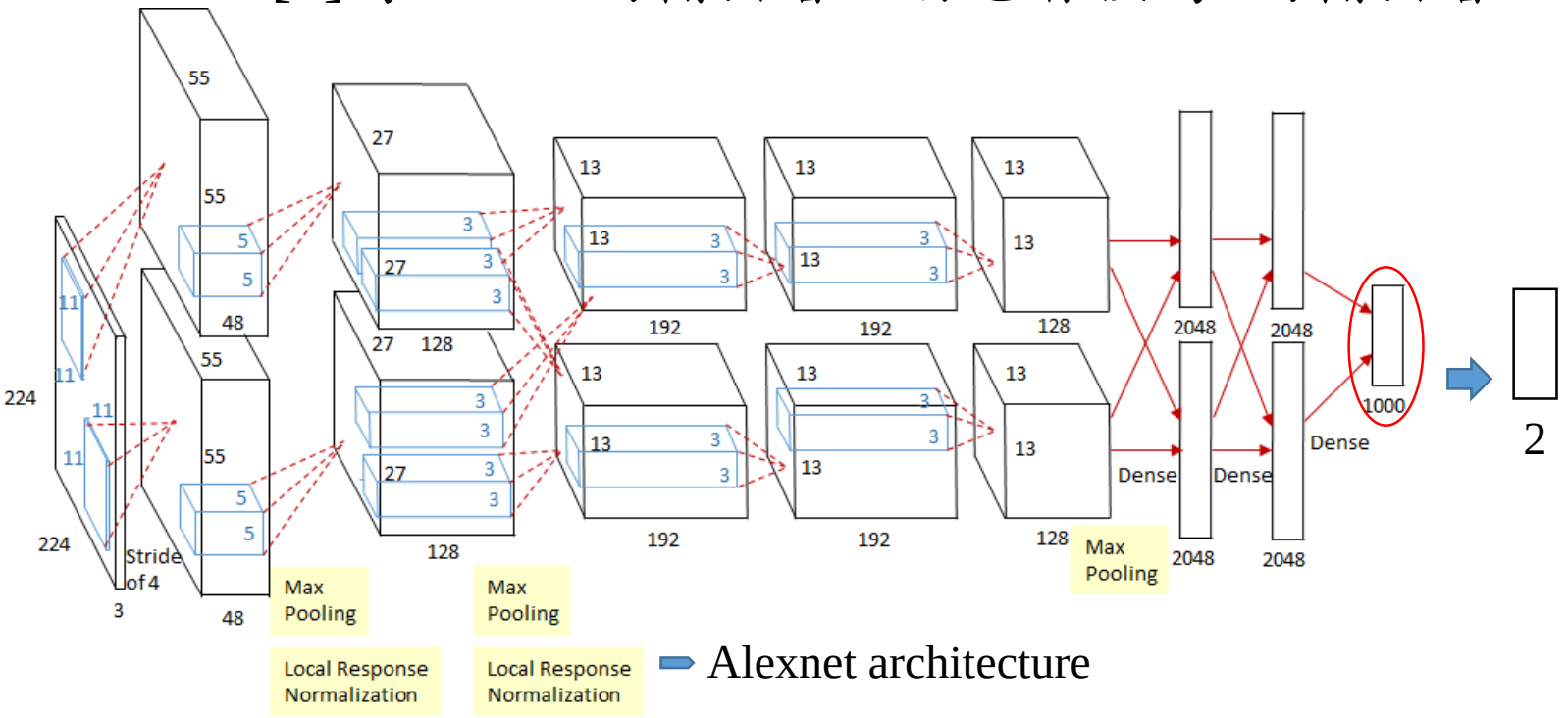
```
model = models.alexnet(pretrained=True)
```

- torchvision提供了可以使用的已訓練模型。
- 在遷移學習的過程中，可以將預先訓練的模型重新用於其他訓練任務。
- 在範例中將使用alexnet模型，並使用已訓練模型功能。

更換神經網路的輸出層

```
model.classifier[6] = torch.nn.Linear(model.classifier[6].in_features, 2)
```

- alexnet最初是為了訓練1000個類別的資料集而產生，但資料集只有兩個類別。
- Model.classifier[6]為alexnet的輸出層，將它替換為2的輸出層。



啟動GPU

```
1  device = torch.device('cuda')
2  model = model.to(device)
```

- [1]torch.device代表將torch.Tensor分配到的設備，’cuda’表示將所有的Tensor變數複製到所指定的GPU上面，之後的運算都在GPU上進行。
- [1]torch.device包含了設備的類別(‘cpu’ or ’cuda’)和可選的設備編號。
- [2]model使用cuda核心進行運算。
- Pytorch可透過字串或是設備編號來指定使用的運算資源，例:

```
>>> torch.device('cuda', 0)
device(type='cuda', index=0)

>>> torch.device('cpu', 0)
device(type='cpu', index=0)
```

最佳化模型介紹

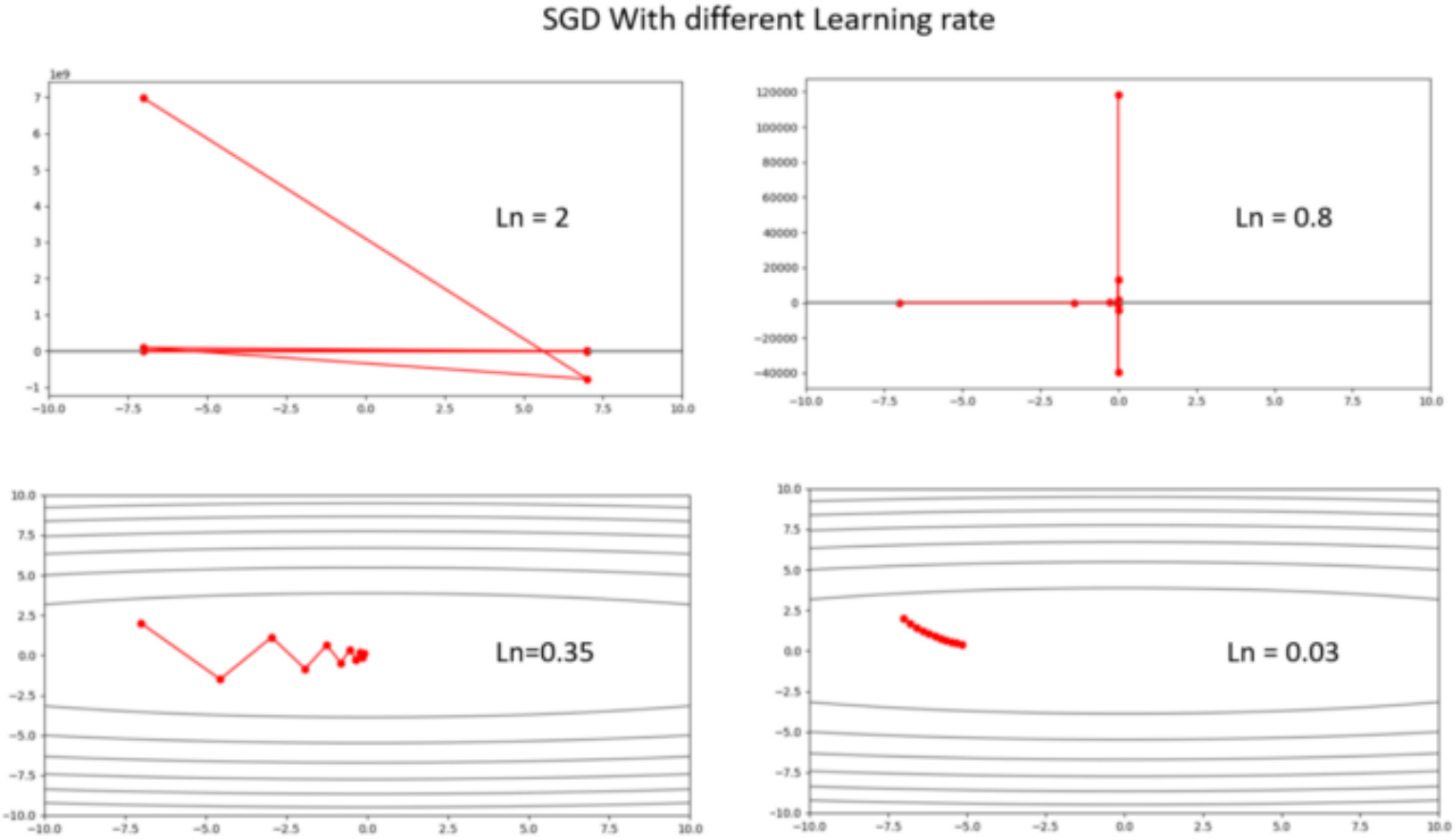
- SGD-隨機梯度下降法 (stochastic gradient decent)
  - SGD 也就是最單純的梯度下降方法，找出參數的梯度(利用微分的方法)，往梯度的方向去更新參數(weight)，即：

$$W \leftarrow W - \eta \frac{\partial L}{\partial W}$$

  - ***W*** 為權重(weight)參數，***L*** 為損失函數(loss function)，η 是學習率(learning rate)，$\partial L/\partial W$ 是損失函數對參數的梯度(微分)。
  - SGD的缺點是如果學習率太大，容易造成參數呈現鋸齒狀的更新，這會使模型的訓練效率降低。

Learning rate(學習率)

- 以下為SGD在調整參數時的震盪，從不同Ln中看得出來SGD容易造成震盪，如果設定太小的話收斂會更慢。※Learning rate = Lr = Ln



Momentun

- Momentum 是「運動量」的意思，此最佳化模型為模擬物理動量的概念，在同方向的維度上學習速度會變快，方向改變的時候學習速度會變慢。

$$V_t \leftarrow \beta V_{t-1} - \eta \frac{\partial L}{\partial W}$$

$$W \leftarrow W + V_t$$

Momentum Weight update equation

- 這裡多了一個 ***Vt*** 的參數，可以將他想像成「方向速度」，會跟上一次的更新有關，如果上一次的梯度跟這次同方向的話，***|Vt|***(速度)會越來越大(代表梯度增強)，***W***參數的更新梯度便會越來越快，如果方向不同，***|Vt|***便會比上次更小(梯度減弱)，***W***參數的更新梯度則會變小，β 可以想像成空氣阻力或是地面摩擦力，通常設定成0.9。

訓練模型

- 本範例訓練30次的神經網路，並在每個訓練週期之後保存準確率最佳的模型。

```
1  NUM_EPOCHS = 30
2  BEST_MODEL_PATH = 'best_model.pth'
3  best_accuracy = 0.0
4
5  optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
6
7  for epoch in range(NUM_EPOCHS):
8
9      for images, labels in iter(train_loader):
10         images = images.to(device)
11         labels = labels.to(device)
12         optimizer.zero_grad()
13         outputs = model(images)
14         loss = F.cross_entropy(outputs, labels)
15         loss.backward()
16         optimizer.step()
17
18     test_error_count = 0.0
19     for images, labels in iter(test_loader):
20         images = images.to(device)
21         labels = labels.to(device)
22         outputs = model(images)
23         test_error_count += float(torch.sum(torch.abs(labels - outputs.argmax(1))))
24
25     test_accuracy = 1.0 - float(test_error_count) / float(len(test_dataset))
26     print('%d: %f' % (epoch, test_accuracy))
27     if test_accuracy > best_accuracy:
28         torch.save(model.state_dict(), BEST_MODEL_PATH)
29         best_accuracy = test_accuracy
```

讀取訓練資料集

初始化error

讀取測試資料集

[5]lr是學習率，momentum是運動量。

[10][11]複製images、labels到GPU記憶體中。

[12]在Pytorch中避免前一次的訓練梯度結果影響到這次的訓練梯度。

[13]使用這次iteration的資料進行訓練。

[14]使用損失函數評估目前模型的好與壞。

[15][16]更新本次iteration訓練參數。

[23]計算測試集的錯誤率。

[25]計算測試集的準確率。

[28]只保存模型參數

執行模型

live_demo.ipynb

使用已訓練模型

```
1  import torch
2  import torchvision
3
4  model = torchvision.models.alexnet(pretrained=False)
5  model.classifier[6] = torch.nn.Linear(model.classifier[6].in_features, 2)
```

- 本範例使用Pytorch提供的alexnet已訓練模型。
- [4]使用alexnet並設定為pretrained=False，目的是為了只保留網路架構。
- [5]model.classifier[6]為了建立跟訓練時一樣的輸出層。

讀取模型

```
model.load_state_dict(torch.load('best_model.pth'))
```

- 從best_model.pth讀取訓練後的權重。
- .pth(模型參數檔)包含了四個主要參數(Key value)，範例如下:

Key	Type	Size	Value
Iteration	int	1	8000
Model	OrderedDict	221	orderedDict object of collection module
Optimizer	dict	2	{‘state’:{140671658900000:{…}, 140671658899352:{...
scheduler	dict	7	{‘milestones’:(15000, 80000), ‘gamma’:0.1, ‘warmup…}

啟動GPU

```
device = torch.device('cuda')
model = model.to(device)
```

- 讓模型能使用GPU記憶體，達到提升運算效能之目的。

定義前處理

```python
import cv2
import numpy as np

mean = 255.0 * np.array([0.485, 0.456, 0.406])
stdev = 255.0 * np.array([0.229, 0.224, 0.225])

normalize = torchvision.transforms.Normalize(mean, stdev)

def preprocess(camera_value):
    global device, normalize
    x = camera_value
    x = cv2.cvtColor(x, cv2.COLOR_BGR2RGB)
    x = x.transpose((2, 0, 1))
    x = torch.from_numpy(x).float()
    x = normalize(x)
    x = x.to(device)
    x = x[None, ...]
    return x
```

[7]透過平均值與標準差進行正規化
[9]~[18]定義前處理
[12]將相機的色彩格式BGR轉成RGB
[13]轉置矩陣
[14]將矩陣放入浮點數，tensor矩陣中
[15]正規化
[16]啟動GPU
[17]將矩陣的第1個設為None

Example:

```
>>> x = torch.randn(2, 3)
>>> x
tensor([[ 1.0028, -0.9893,  0.5809],
        [-0.1669,  0.7299,  0.4942]])
>>> torch.transpose(x, 0, 1)
tensor([[ 1.0028, -0.1669],
        [-0.9893,  0.7299],
        [ 0.5809,  0.4942]])
```

- 因為訓練影像與相機的格式不同，因此我們需要進行前處理。

定義相機與滑軌(slider bar)

```
import traitlets
from IPython.display import display
import ipywidgets.widgets as widgets
from jetbot import Camera, bgr8_to_jpeg

camera = Camera.instance(width=224, height=224)
image = widgets.Image(format='jpeg', width=224, height=224)
blocked_slider = widgets.FloatSlider(description='blocked', min=0.0, max=1.0, orientation='vertical')

camera_link = traitlets.dlink((camera, 'value'), (image, 'value'), transform=bgr8_to_jpeg)

display(widgets.HBox([image, blocked_slider]))
```

- 建立一個滑軌(slider bar)，該滑軌(slider bar)將顯示機器人blocked的數值。
- 建立即時影像視窗，從影像中可以知道對應的blocked數值。
- 當Jetbot越靠近障礙物時，blocked數值越高，反之亦然。

啟動Jetbot馬達

```python
from jetbot import Robot

robot = Robot()
```

- 定義Robot並提供Jetbot驅動馬達。

定義更新功能

```python
import torch.nn.functional as F
import time

def update(change):
    global blocked_slider, robot
    x = change['new']
    x = preprocess(x)
    y = model(x)

    # we apply the `softmax` function to normalize the output vector so it sums to 1 (which makes it a probability distribution)
    y = F.softmax(y, dim=1)

    prob_blocked = float(y.flatten()[0])

    blocked_slider.value = prob_blocked

    if prob_blocked < 0.6:
        robot.forward(0.27)
    else:
        robot.left(0.27)

    time.sleep(0.001)

update({'new': camera.value})  # we call the function once to intialize
```

- 建立update功能，當影像發生變化時，會依照blocked的數值對馬達進行控制。
- 注意!現在還未讀取相機資訊，請同學把Jetbot拿在手上，以避免Jetbot摔落。

提供相機資訊

```
camera.observe(update, names='value')  # this attaches the 'update' function to the 'value' traitlet of our camera
```

- The observe method of the widget can be used to register a callback.
- 使用者可以透過observe(觀察者功能)將camera.value加到update function中。
- 當camera.value改變時，update function則會依照camera.value的資訊來動作。

停止相機與Jetbot

```
import time

camera.unobserve(update, names='value')

time.sleep(0.1)  # add a small sleep to make sure frames have finished processing

robot.stop()
```

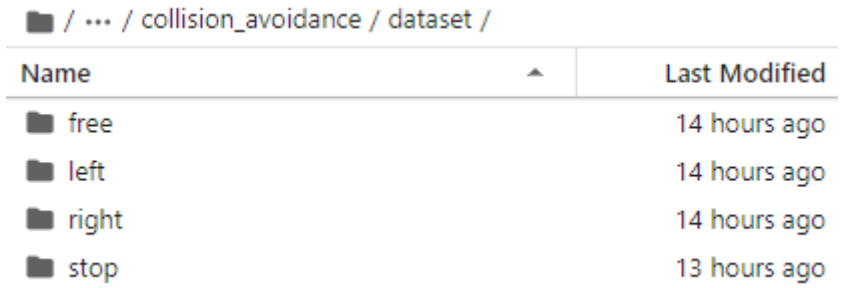
- 如果要停止Jetbot時，可以透過執行以上程式碼來完成。

專案實作-道路避障

- 專案項目:道路避障
- 各小組需修改道路避障程式，定義Free space、紅色角椎、綠色角錐、自訂障礙物等4種label
- 透過障礙物避障與辨識模型使Jetbot行駛於紅綠角椎所建成的道路內，並於自訂障礙物前停下。

收集資料 & 模型訓練 調整

- 修改data_collection.ipynb，將預設label數量由2改為4(如下圖)
- AlexNet模型output由2改為4
- Training data 與 Testing data數量根據dataset數量決定(助教dataset總數量為24張)

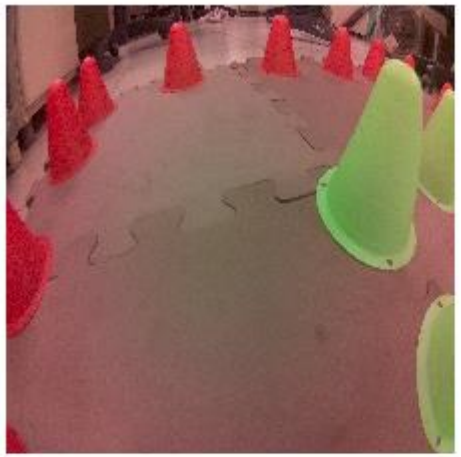


模型訓練結果檢視

- 修改live_demo.ipynb，將訓練結果之輸出使用slider bar顯示於widget，以確認模型訓練結果是否正確
- 根據輸出類別調整馬達動作

blocked free 0.57
blocked right 0.18
blocked left 0.22
blocked stop 0.03

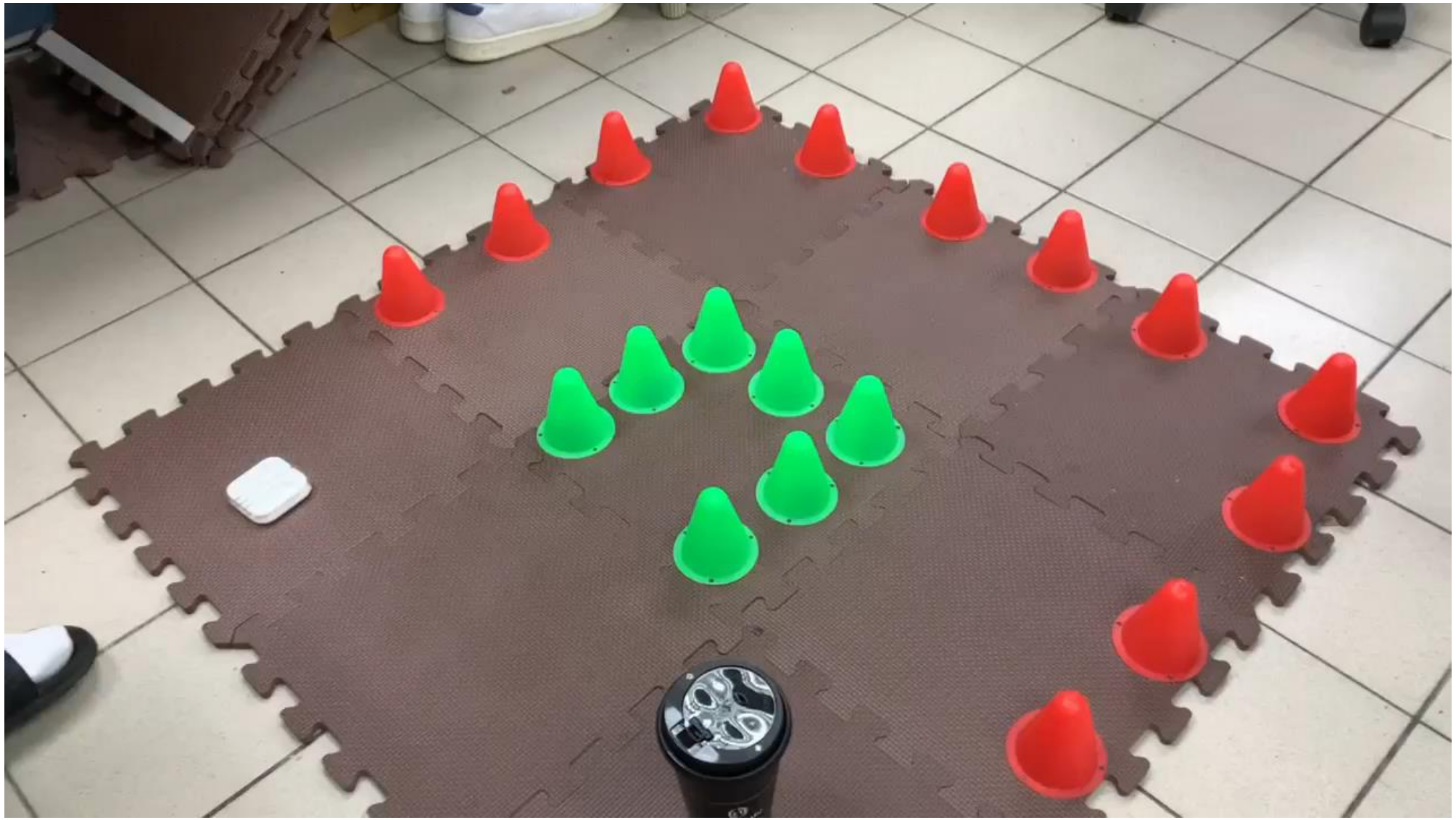
blocked free 0.09
blocked right 0.72
blocked left 0.10
blocked stop 0.03



blocked free 0.17
blocked right 0.18
blocked left 0.57
blocked stop 0.07

blocked free 0.02
blocked right 0.00
blocked left 0.00
blocked stop 0.99

Project Demo



小組報告格式規定

- 專案情境
  - 小組所討論出來的議題，並簡明扼要描述議題的情境。
- 定義問題
  - 將議題中的問題定義出來，並收斂問題方向。
- 方案構思
  - 簡單描述如何解決定義好的問題，並預計使用的技術。
- 解決方法
  - 說明實際上如何完成此議題的方案構思。
- 分工
  - 說明小組成員分工內容與比例。

個人報告內容

- 個人報告內容須要有以下內容：
  - 你一開始所提出的議題是?你的議題是否有被選為小組議題候選?
  - 你在小組議題中，提出了那些問題與解決方案?是否有被小組接受?
    - 如個人所提出的方案沒被接受，是因為那些原因?
    - 為了這個議題，你去找了那些資料?你是如何分析找到的資料?
    - 其他小組成員所提出的提議有哪些?而你對於其他人的提議意見如何?
  - 在小組決定小組議題過程中，你對於小組最後提出的議題討論是否能接受?接受理由為何?不接受理由為何?
  - 你是否能接受最後的議題與方案?如接受請說明接受與否的理由?
  - 本次專案個人的心得
  - 本次你認為小組成員的貢獻比例及理由

專案繳交規則

- 專案成果實體驗收請於110/12/10課程結束前找助教檢查
- 小組報告繳交期限:110/12/10 23:59(以I學園上傳時間為基準)
- 個人報告繳交期限:110/12/10 23:59(以I學園上傳時間為基準)
- 補交規則
  - 超過正常繳交期限兩周內成績**打8折**
  - 超過正常期限兩周後不接受補交