

國立雲林科技大學機械工程系

碩士論文

Department of Mechanical Engineering

National Yunlin University of Science & Technology

Master Thesis

PyTorch 機器深度學習架構應用之研究

The Research of Machine Deep Learning by

PyTorch Libraries

孫健晟

Chien-Cheng Sun

指導教授：吳英正 博士

Advisor：Ying-Jeng Wu, Ph.D.

中華民國 109 年 6 月

June 2020

摘要

近來深度學習框架如雨後春筍般的出現，其中以 Tensorflow 最廣為人知，而在眾多的學習框架中，PyTorch 於論文的引用數逐漸追上 Tensorflow，由於 PyTorch 撰寫於 AI 程式的靈活性及使用體驗十分優秀，為此將對 PyTorch 進行研究與討論。本論文將會從安裝 PyTorch、建立模型到如何載入自己的資料庫，在過程中了解建立模型的方法及其傳達的概念，不再是只有增加層數與調整參數，而是需要自行建立網路中一個個的迴圈，並控制每個細節，雖然比起其他學習框架多了一些須注意的步驟，但換來的是更優質的靈活度與使用體驗，這也是為什麼論文的引用數會越來越多的原因。論文的最後將會介紹動態圖(dynamic graph)的概念，並使用 PyTorch 實現此概念以加強網路的學習效果，以更直觀的感受 PyTorch 的靈活性。

關鍵字：PyTorch、CNN、動態圖。

Abstract

Recently, deep learning libraries have been sprung up, and Tensorflow is the most widely known among them. Among the many learning librareis, PyTorch has gradually been caught up with Tensorflow according to the number of citations in the paper. Due to the flexibility of PyTorch for AI coding and its excellent experience, PyTorch is studied. This paper discusses how to install PyTorch, how to build a model and how to load our own database. In the process, PyTorch arrows user to the method of building a model and the concepts of PyTorch are explored. Create loops one by one on your own network and control every detail of it. Although, there are more desired procedures than other deep learning libraries, it has better coding flexibility and user experience, which is why the number of citations increase. At the end of this paper, the concept of dynamic graph will be introduced, and PyTorch will be used to realize this concept to enhance the learning effect of the network, such that the user may feel the flexibility of PyTorch more intuitively.

Keyword: PyTorch, CNN, Dynamic Graph.

目錄

摘要.....	i
Abstract	ii
目錄.....	iii
表目錄.....	iv
圖目錄.....	v
第 1 章 緒論	1
1.1 研究目的與動機.....	1
1.2 研究方法	1
1.3 文獻回顧	2
1.4 論文架構	3
第 2 章 PyTorch 介紹與安裝.....	4
2.1 PyTorch 介紹	4
2.2 PyTorch 安裝流程	4
第 3 章 於 PyTorch 建立 CNN 模型.....	7
3.1 CNN 模型介紹	7
3.2 使用 PyTorch 實現 CNN 模型.....	9
第 4 章 於 PyTorch 載入資料集之方法	20
4.1 台灣限速牌資料集	20
4.2 建立圖片路徑文字檔與標籤.....	21
4.3 建立符合 PyTorch 讀取資料之類別.....	22
4.4 台灣限速牌資料集訓練結果.....	23
第 5 章 PyTorch 動態圖	27
5.1 動態圖與靜態圖	27
5.2 PyTorch 動態圖之應用 – 改變學習率	27
第 6 章 結論與建議.....	31
6.1 結論	31
6.2 建議.....	32
參考文獻.....	33
附錄 A.....	34

表目錄

表 2-1	安裝環境與軟體版本.....	4
表 2-2	Anaconda 安裝流程.....	5
表 2-3	Pytorch 安裝流程	5
表 2-4	Cuda 與 Cudnn 安裝流程	6
表 3-1	使用 PyTorch 時需匯入的函式庫	9
表 3-2	確認電腦有無 GPU 並設定 device 變數.....	11
表 3-3	載入圖片資料集與進行資料前處理.....	11
表 3-4	建立 CNN 模型所需之函數.....	12
表 3-5	訓練開始前的調適參數設定	15
表 3-6	訓練開始前設定 GPU、損失函數與最佳化方法.....	16
表 3-7	訓練流程之程式碼介紹.....	17
表 4-1	建立 PyTorch 讀取資料之類別	22
表 4-2	限速牌資料集對應 RGB 三色道的平均值與標準差.....	25

圖目錄

圖 1-1	對應不同框架與網路模型每秒處理影像的數量.....	2
圖 2-1	PyTorch 安裝版本選擇介面.....	5
圖 3-1	捲積層計算過程示意圖.....	7
圖 3-2	訓練完成後 kernal 中的參數具有訓練資料的特徵.....	8
圖 3-3	參考 VGG 所建立的 CNN 流程圖.....	9
圖 4-1	台灣限速牌資料集.....	20
圖 4-2	圖片路徑檔之示意圖.....	21
圖 4-3	限速牌的訓練模型.....	23
圖 4-4	對應 epoch 的訓練與測試準確率.....	24
圖 4-5	對應 epoch 的損失量.....	24
圖 4-6	資料於 Normalize 後對應 epoch 所得到的準確率.....	25
圖 4-7	資料於 Normalize 後對應 epoch 所得到的損失量.....	26
圖 5-1	局部最小值與全域最小值.....	27
圖 5-2	模型訓練沒有調整學習率對應 epoch 所呈現的準確率.....	29
圖 5-3	模型訓練沒有調整學習率對應 epoch 所呈現的損失量.....	29
圖 5-4	模型於訓練過程中調整學習率對應 epoch 所呈現的準確率.....	30
圖 5-5	模型於訓練過程中調整學習率對應 epoch 所呈現的損失量.....	30

第1章 緒論

1.1 研究目的與動機

近年來 AI 學習框架如雨後春筍般地出現，例如 Tensorflow、MXNet.....等，眾多的學習框架中 Tensorflow 在學界與業界都有不錯的表現，其開發團隊為世界知名公司 Google。雖然 Tensorflow 的功能強大，但是對於初入 AI 領域的新手，會因為其嚴謹的撰寫方式而不易入手，不過最近有一款學習框架於學術領域上的引用數逐漸與 Tensorflow 持平甚至有超過的跡象，此框架名為 PyTorch。PyTorch 的前身為 Torch，於 2017 年時 Torch 的執行團隊決定將程式結構重建與優化，轉型成為了 Python 而打造的學習框架，使運作效率與靈活度比 Tensorflow 優秀許多，且 PyTorch 重建後對於 Python 的開發者相當的友善能以有十分優秀的使用體驗。

由於以上原因，此論文將以「導入 PyTorch 的 AI 程式撰寫」與「PyTorch 撰寫 AI 程式碼的優勢」為討論的重點。

1.2 研究方法

網路上有大量且豐富的資源，欲學習 PyTorch，不論是官方網站或網路上分享的開源碼都是相當合適的參考資料，閱讀這些資料並學習後，得以建立屬於自己的深度學習網路。製作模型的過程中將參考羅偉宸(2019，[6])「台灣限速牌資料集」，放入模型內進行訓練，以初步了解 PyTorch 的使用方法，並記錄過程中所遇到問題與重點以完成「導入 PyTorch 的 AI 程式撰寫」等目標。

對於 PyTorch 有明確的掌握後，進入核心「動態圖(Dynamic graph)」靜態圖(Static graph)相對應於動態圖，靜態圖的概念是 AI 在訓練的過程中超參數(Hyperparameter)保持不變，例如：學習率(Learning rate)、損失函數(Loss function)和最佳化方法(Optimal function)等，而動態圖比起靜態圖靈活許多，得以在訓練過程中改變模型的超參數，將動態圖之概念應用於 PyTorch，並記錄過程中所遇到問題與重點以完成「PyTorch 於撰寫 AI 程式碼的優勢」等目標。

1.3 文獻回顧

機械深度學習因電腦速度的提升，近年來得以有不錯的表現，4、50 年前其理論已經有初步的架構，但是限制於當時電腦的速度，使機械學習無法實際應用。目前建立深度學習的網路時，通常會使用他人所打造的深度學習框架，本論文於眾多的框架中，選擇編寫靈活性與運算效率較優秀的 PyTorch 作為應用的對象。

2019 年 Adam Paszke、Sam Gross、Francisco Massa ...許多人共同發表的文章中，介紹 PyTorch 有著許多優點，使編寫程式的靈活性與使用體驗十分優秀，關於文章中討論運算效率的結果如圖 1-1。

Framework	Throughput (higher is better)					
	AlexNet	VGG-19	ResNet-50	MobileNet	GNMTv2	NCF
Chainer	778 ± 15	N/A	219 ± 1	N/A	N/A	N/A
CNTK	845 ± 8	84 ± 3	210 ± 1	N/A	N/A	N/A
MXNet	1554 ± 22	113 ± 1	218 ± 2	444 ± 2	N/A	N/A
PaddlePaddle	933 ± 123	112 ± 2	192 ± 4	557 ± 24	N/A	N/A
TensorFlow	1422 ± 27	66 ± 2	200 ± 1	216 ± 15	9631 ± 1.3%	4.8e6 ± 2.9%
PyTorch	1547 ± 316	119 ± 1	212 ± 2	463 ± 17	15512 ± 4.8%	5.4e6 ± 3.4%

圖 1-1 對應不同框架與網路模型每秒處理影像的數量，”PyTorch: An Imperative Style, High-Performance Deep Learning Library”(2019，[1])

PyTorch 於眾多學習框架中處理影像的速度大多位居前 1、2 名，並且論文中提到 arXiv (論文提報網站) 於 2019 年與深度學習相關論文提及 PyTorch 的比率多達 40% 以上，因此本論文將以 PyTorch 做為討論及研究的核心。

1.4 論文架構

本論文第一章將介紹為何 PyTorch 現今被學界的 AI 研究者所愛好以及如何學習與研究 PyTorch；第二章將介紹 PyTorch 與安裝過程以及需注意的細節；第三章將參考 VGG 模型以應用 PyTorch 實現 CNN 模型，並說明製作過程中應注意的細節與使用 PyTorch 製作模型之概念；第四章將引用非 PyTorch 所提供的資料庫進行網路訓練，並介紹引用的方法，使 PyTorch 的應用能更加廣泛；第五章中介紹動態圖與靜態圖，並應用動態圖的概念改變訓練中模型的學習率，使訓練效果提升；第六章為結論與建議為本論文的總結。



第2章 PyTorch 介紹與安裝

2.1 PyTorch 介紹

PyTorch 是 2017 年來於學術界迅速竄起的深度學習框架，程式撰寫方面比多數的學習框架靈活並有更好的使用體驗。PyTorch 的前身為 Torch，建立於 Lua 語言的張量計算資料庫，且 Lua 語言被稱為世界上最快的手稿語言，是一款相當優質的程式語言，但是 Torch 的團隊比較後發現 Python 的使用數遠多於 Lua，因此團隊決定為 Python 重建與優化自己的資料庫，並命名為 PyTorch。

由上述可知 PyTorch 於 AI 編碼時雖然比起其他學習框架靈活，但其為較新的資料庫，因此不論是在其他平台上的部署或產品化的穩定度都沒有比 Tensorflow 來的優秀，所以目前 PyTorch 使用者大多是為了更好的調適性以及「Python 化」的使用體驗。

2.2 PyTorch 安裝流程

安裝任何一款深度學習框架時都需要注意電腦的作業系統、程式語言的版本與顯示卡的型號等資訊，並於安裝過程中注意目前安裝的資料庫版本是否與以上的資訊相符合，表 2-1 為安裝成功的範例。

表 2-1 安裝環境與軟體版本

安裝環境與軟體版本		
電腦資訊		軟體版本
作業系統	Windows10	Pytorch 1.4.0
程式語言	Python3.6	Torchvision 0.5.0
顯示卡	Nvidia GTX1060	Cuda 10.1
		Cudnn 10.0


確認硬體與軟體資訊後則開始安裝 Anaconda，其為許多 Python 直譯軟體的組合包，此組合包中較常使用的直譯軟體為 Spyder，安裝流程如表 2-2。

表 2-2 Anaconda 安裝流程

Anaconda 安裝流程		
步驟	流程簡述	敘述
1.	安裝 Anaconda	於 Anaconda 官方網站下載安裝檔後安裝。
2.	建立 Python3.6 環境	<p>Anaconda 安裝完成後，可以在「開始」找到 Anaconda 的資料夾，在裡面尋找「Anaconda Prompt」將其開啟，Anaconda Prompt 類似 Console Mode，開啟後在裡面輸入指令：</p> <p>「conda create -n py36 python=3.6 anaconda」</p> <p>指令完成後則代表 python3.6 環境建立成功，並且此環境名稱為「py36」</p>

Anaconda 安裝完成後則開始安裝 PyTorch，安裝流程如表 2-3。

表 2-3 Pytorch 安裝流程

PyTorch 安裝流程		
步驟	流程簡述	敘述
1.	進入 PyTorch 官方網站	上網搜尋 PyTorch 進入官方網站，於網站內尋找「Get Start」按鈕並點下。(參考網站： https://pytorch.org/)
2.	選擇 PyTorch 版本	<p>進入「Get Start」的頁面後會看到如圖 2-1 的畫面，請根據需求選擇。</p>  <p>圖 2-1 PyTorch 安裝版本選擇介面</p>

3.	安裝 PyTorch	<p>圖 2-1 選擇完成後，介面的最下方會出現安裝 PyTorch 的指令並複製，如：</p> <pre>「 pip install torch==1.4.0 torchvision==0.5.0 -f https://download.pytorch.org/whl/torch_stable.html 」</pre> <p>開啟 Anaconda Prompt，並輸入「conda activate py36」以開啟 python3.6 環境，再輸入複製的指令以安裝 PyTorch。</p>
----	------------	---

根據圖 2-1 的選擇使得 PyTorch 為 GPU 版本，但是下載 PyTorch-GPU 版本還不足以在訓練的過程中使用 GPU，因此還需要 Cuda 以及深度學習加速資源包 Cudnn，Cuda 是由 Nvidia 所推出的整合技術為溝通 GPU 的介面，以使用 GPU 進行加速，而 Cudnn 是加快深度學習速度的資源包，安裝流程如表 2-4。

表 2-4 Cuda 與 Cudnn 安裝流程

Cuda 與 Cudnn 安裝流程		
步驟	流程簡述	敘述
1.	安裝 Cuda	<p>安裝 Cuda 請參考以下網站：</p> <p>https://developer.nvidia.com/cuda-downloads</p>
2.	下載 Cudnn 資源包	<p>下載 cudnn 之前請先註冊成為 Novidia 的會員，下載的網站請參考以下網站：</p> <p>https://developer.nvidia.com/rdp/cudnn-download</p>
3.	移動 Cudnn 的檔案至 Novian 資料夾	根據資源包內的資料夾名稱尋找 Cuda 資料夾內相應的名稱，將檔案複製至 Cuda。
4.	更改環境變數	進入電腦內更改環境變數中的「Path」，在「Path」添加 Cuda 資料夾內的「x64」與「bin」資料夾路徑。

完成以上步驟後即可於 PyTorch 上使用 GPU 進行訓練。

第3章 於 PyTorch 建立 CNN 模型

3.1 CNN 模型介紹

CNN(Convolutional Neural Network)是目前應用於圖像辨識最有效的神經網路，其中包含捲積層(Convolutional Layer)與全連接層(Full Connection Layer)。捲積層的訓練參數由許多較小的矩陣組成，名為 kernel，將 kernel 與輸入矩陣點對點相乘後加總，再根據步伐(stride)大小移動 kernel 的位置，當 kernel 掃描過輸入矩陣中的每個位置後結束計算，例如圖 3-1 kernel 大小為 3x3 的矩陣、輸入矩陣的大小為 6x6 以及步伐為 1，於計算的過程中，將 kernel 與輸入矩陣的紅色區域點對點相乘後相加，得到輸出矩陣中紅色位置的數值，再將紅色區域往右移動步伐的格數進行同樣的運算，直到 kernel 掃描過輸入矩陣全部的位置。

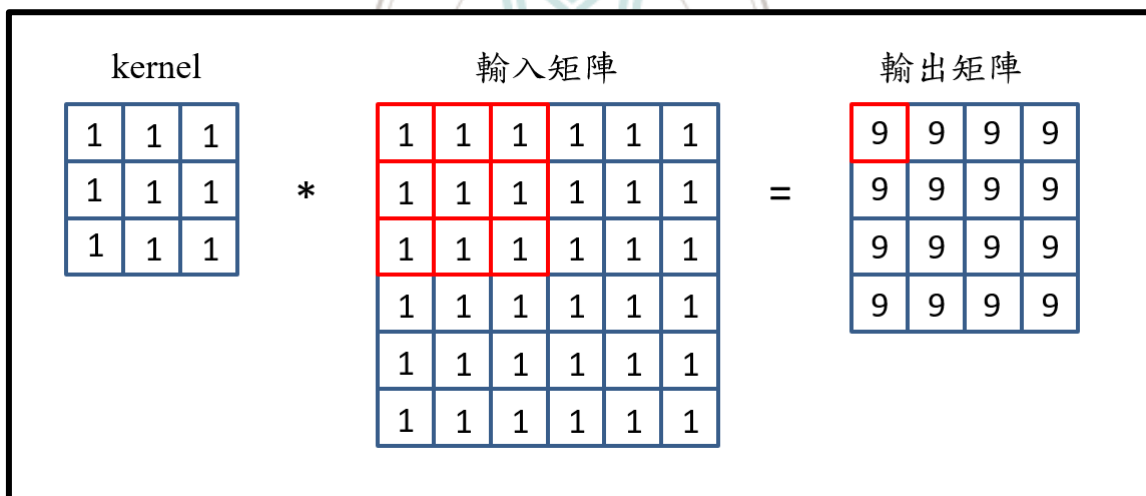


圖 3-1 捲積層計算過程示意圖

CNN 是由多個捲積層所建立的神經網路，當捲積層進行至最後一層時會將全部的訓練參數轉換為一維的向量，連接至全連接層進行分類以判斷圖像或訊號的種類。訓練完成的 CNN 模型於圖像分類時將捲積層的輸出結果以圖像的方式展開，可以發現捲積層會將訓練資料的特徵萃取出來，如圖 3-2。

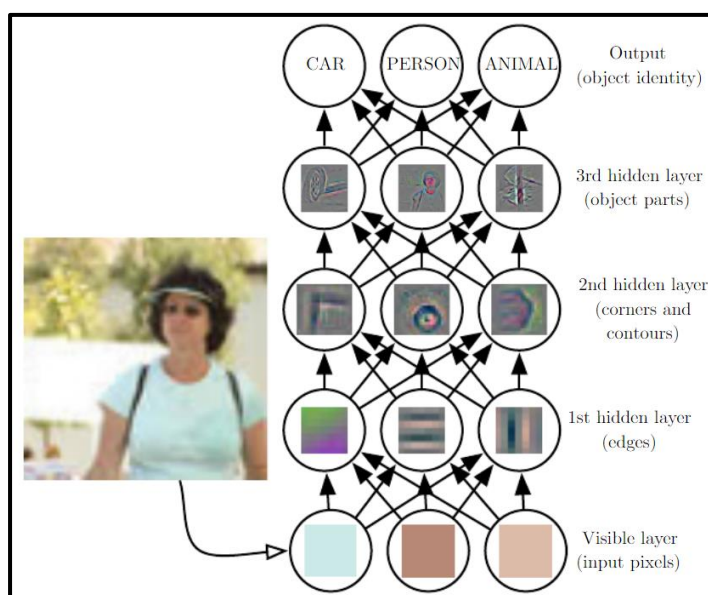


圖 3-2 訓練完成後捲積層輸出結果具有訓練資料的特徵, Ian Goodfellow, Yoshua Bengio, and Aaron Courville(2016, [2])

本論文 CNN 模型使用 VGG16，一般 VGG16 為 13 層捲積層以及 3 層全連接層所組合，但全連接層較容易使模型的訓練參數過度增加，因此本論文只用了 1 層全連接層。由於 VGG 中捲積層的層數較多，因此 VGG 需使用小的 kernel 以換取更深的層數。關於 CNN 的深度(層數)需對應訓練資料複雜的程度以判斷使用的層數，層數越多通常會使學習的效果越好，但使用過高的層數時有可能導致模型無法收斂，反之層數過低時訓練的效果可能較差，因此應選擇對應資料所需的層數及可。

使用 VGG16 主要的原因在於本論文訓練的資料是網路上的開源圖片資料集 CIFAR-10，其中有 10 種物件的圖片，因為其中每種圖片的樣式複雜，所以最終選擇參考 VGG16 以建立深度模型。此資料集中有 50000 張的訓練資料與 10000 張的測試資料，對於訓練深度神經網路相當足夠，圖 3-3 為參考 VGG 所建立的 CNN 流程圖。

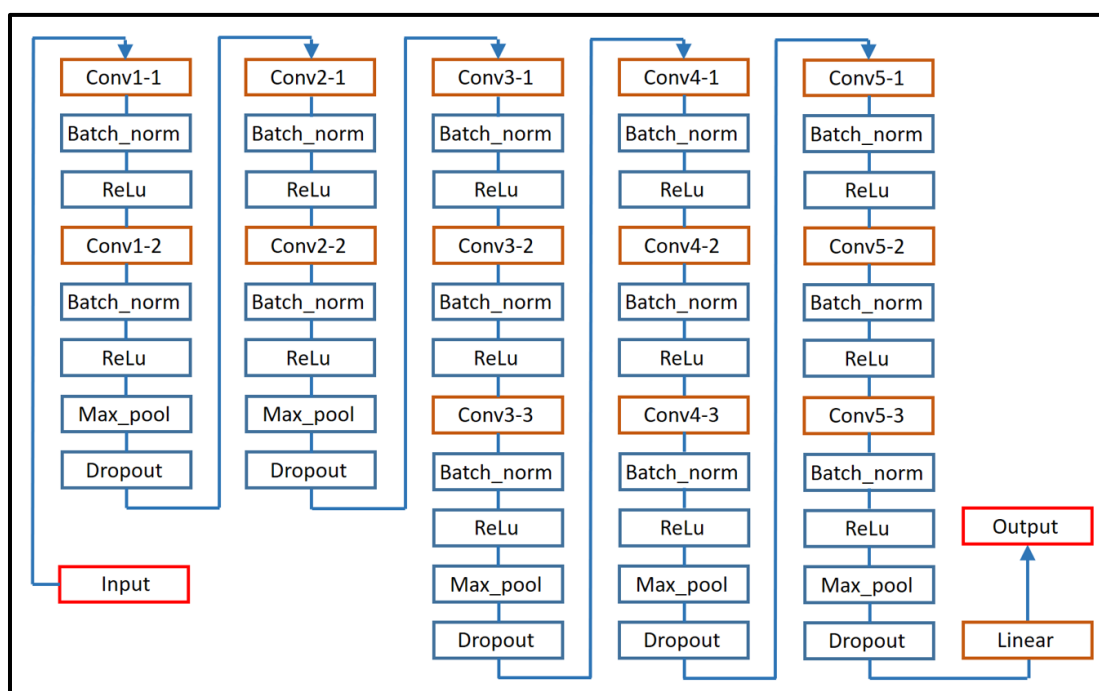


圖 3-3 參考 VGG 所建立的 CNN 流程圖

3.2 使用 PyTorch 實現 CNN 模型

本節說明如何使用 PyTorch 實現 CNN 模型。撰寫 CNN 模型前首先要匯入 PyTorch 與其他顯示圖片、計算數據的函數庫，需要匯入的函數庫如表 3-2。

表 3-1 使用 PyTorch 時需匯入的函式庫

行數	內容
1.	import torch
2.	import torch.nn as nn
3.	import torch.nn.functional as F
4.	import torchvision.datasets as datasets
5.	import torchvision.transforms as transforms
6.	import matplotlib.pyplot as plt
7.	import numpy as np
8.	import time
9.	from os import path
10.	from os import mkdir

匯入的函數庫如表 3-2，首先將「torch」匯入至程式內，此類別為所有函數的父類別，因為其底下的子類別會經常使用，所以為了方便則將這些類別以單獨的方式匯入，例如：nn 類別、nn.functional 類別.....等，單獨引用的類別中最為重要的是「nn」，存放了所有層(layer)的類別，例如：Conv_2d、BatchNorm2d等，因此使用 PyTorch 建立神經網路 nn 為不可缺少的部分。

於 nn 底下的子類別 functional 可以說是 nn 的靈魂，當 nn 需使用任何有關深度學習的函數時大部分都會從 functional 內引用，雖然 nn 的設計概念是將所有訓練所需的函式建立為不同的類別，但是有些類別因為不需更新參數，所以建立網路時則可以直接使用 nn.functional，以減少多建立層的計算時間。

關於「torchvision」是 PyTorch 的函數庫，功能在於補足於深度學習中與神經網路無關的函數庫，例如：載入圖片資料集、資料前處理.....等，於本章節中引用 CIFAR-10 也需使用此函數庫進行處理，關於此函數庫進行資料前處理的方法在於其子類別「transforms」，此類別中的函數有「灰階化」、「修改亮度、飽和度」以及「標準化」.....等，可以說是相當方便。

以上描述為引用 PyTorch 函數庫的介紹，但是建立完整的神經網路程式還需要其他的函數庫補足，其中有畫圖的函數庫「matplotlib」、建立變數與計算數學的函數庫「numpy」、紀錄時間的函數庫「time」以及處理檔案位置的函數庫「os」，有了以上的函數庫幫忙方能開始撰寫神經網路程式。

建立神經網路前需要先確認電腦中是否有 GPU 幫忙加速網路學習，有無 GPU 對於學習的速度可以相差至 8 到 9 倍。確認有無 GPU 的程式碼如表 3-3，假如有 GPU 時，device 變數將設定為使用 GPU，反之則設定為使用 CPU 的方法。

表 3-2 確認電腦有無 GPU 並設定 device 變數

行數	內容
1.	<code>if torch.cuda.is_available():</code>
2.	<code>device = torch.device("cuda:0")</code>
3.	<code>else:</code>
4.	<code>device = torch.device("cpu")</code>

確認有無 GPU 後則開始載入圖片資料集與進行資料前處理，如表 3-4。

表 3-3 載入圖片資料集與進行資料前處理

行數	內容
1.	<code>data_transform = transforms.Compose([transforms.ToTensor() , transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010))])</code>
2.	<code>set_batch_size = 16</code>
3.	<code>trainset = datasets.CIFAR10(root = './data', train = True, download = True, transform = data_transform)</code>
4.	<code>trainloader = torch.utils.data.DataLoader(trainset, batch_size = set_batch_size, shuffle = True)</code>
5.	<code>len_trainloader = len(trainloader.dataset)</code>

關於資料前處理的部分使用了 ToTensor 函數以及 Normalize 函數，ToTensor 的目的在於將變數轉換為 torch 可以計算的變數，一般的變數種類通常為 numpy.ndarray，而 torch 可以計算的變數種類為 torch.FloatTensor，並將輸入變數的數值轉換至[0, 1]之間，而 Normalize 函數的目的在於將數據進行標準化，處理的方法會依照數據整體的標準差與平均值進行計算，且因為 CIFAR-10 為 RGB 的彩色圖片，所以標準差與平均值根據三種不同的色道進行標準化運算。

關於資料載入的部分，載入時須使用 torchvision 的子類別「dataset」，dataset 存有許多現已公開的圖片資料集，這次使用的是 CIFAR-10 資料集，開啟時需要

輸入四個資訊「root」、「train」、「download」以及「trnasform」，分別為「開啟圖片資料集的路徑」、「此資料是用來訓練或是測試」、「是否已經下載」以及「資料前處理的方法」，資料載入後設定訓練時需要使用的參數如表 3-4 的第 4 行，對函數輸入圖片資料集的設定後，決定小批次(mini-batch)的數值以及每次訓練時是否打亂資料順序，以上完成後建立變數記錄此資料集的資料數量。

表 3-4 為開啟 CIFAR-10 資料集訓練資料的方法，而開啟測試資料的方法與之相似，詳細的程式碼將記錄於附錄 A，接下來介紹如何建立神經網路，如表 3-5。

表 3-4 建立 CNN 模型所需之函數

行數	內容
1.	<code>class Net(nn.Module):</code>
2.	<code>def __init__(self):</code>
3.	<code>super(Net, self).__init__()</code>
4.	<code>self.conv1_1 = nn.Conv2d(3, 32, 3, padding = 1)</code>
5.	<code>self.BatchNorm2d1_1 = nn.BatchNorm2d(32)</code>
6.	<code>self.conv1_2 = nn.Conv2d(32, 32, 3, padding = 1)</code>
7.	<code>self.BatchNorm2d1_2 = nn.BatchNorm2d(32)</code>
	<code>⋮</code>
8.	<code>self.classifier = nn.Linear(512, 10)</code>
9.	<code>def forward(self, x, batch_size, set_dropout, train):</code>
10.	<code>x = F.relu(self.BatchNorm2d1_1(self.conv1_1(x)), inplace = True)</code>
11.	<code>x = F.relu(self.BatchNorm2d1_2(self.conv1_2(x)), inplace = True)</code>
12.	<code>x = F.max_pool2d(x, kernel_size = 2, stride = 2)</code>
13.	<code>if train :</code>
14.	<code>x = F.dropout(x, p = set_drop_out)</code>

	⋮
15.	<code>x = x.view(batch_size, -1)</code>
16.	<code>x = self.classifier(x)</code>
17.	<code>return x</code>

關於 PyTorch 建立神經網路較為常用的方法有兩種，其一使用 `nn` 類別下的函數 `Sequential` 以排序的方式逐一將需要的層加入至神經網路內，另一則是以類別的方式建立神經網路，以此方式建立的網路會靈活於 `Sequential`，因為 `Sequential` 的概念是將網路以序列(Sequence)的方式包住，所以網路進行正向傳遞(forward)時無需自行定義每一個步驟，相對於使用類別的方式，如表 3-5 第 9 行中則自行定義正向傳遞的步驟，選用什麼方式定義網路依個人喜好與使用情況決定，而本論文使用以類別的方式建立神經網路。

此方式建立網路時需要將網路的類別繼承於 `nn.Module` 類別之下，如表 3-5 的 1 至 3 行，建立網路的流程可以參考圖 3-2，過程中不需將所有函數都建立為不同的層這樣只會增加電腦的運算時間，因此以層的方式建立時，此函數必須要包含訓練參數，例如：捲積層、批次標準化(Batch Normalization)以及全連接層，捲積層需要更新 kernel 內的參數以找到物件的特徵，而批次標準化在訓練的過程中以批次的數量計算訓練參數的平均數與標準差，並對資料進行標準化，再依照目前數據的大小以點乘的方式將每一個數據乘以訓練標準差後加上訓練平均值以重新賦予因標準化消失的訓練特徵，標準化公式如式 3.1

$$\hat{x}_i = \frac{x_i - u_B}{\sqrt{\sigma_B^2 + \epsilon}} \quad (3.1)$$

如式 3.1 所示 \hat{x}_i 為標準化後的 x_i ，其中 u_B 為原始數據的平均值、 σ_B^2 為原始數據的變異數以及 ε 是為了防止標準差為零而設定的很小的數值，標準化完成後乘以訓練變異數並加上訓練平均值以得到最後的輸出，如式 3.2。

$$y_i = \alpha_i * \hat{x}_i + \beta_i \quad (3.2)$$

如式 3.2 所示 y_i 為 Batch Normalize 的輸出結果，其中 α_i 與 β_i 為訓練標準差與訓練平均值，此數值需經由訓練的過程持續更新參數，因為深度學習的參數過於龐大導致與無法以數學精準的描述訓練與參數更新的過程，但是經由反覆的實驗得知，目前使用的時機為「當網路訓練的過程十分緩慢時可以考慮加入網路中，使訓練的速度加快。」

關於全連接層的功能在於將捲積層的結果進行分類，計算的公式如式 3.3。

$$Y = X^T \cdot W + B \quad (3.3)$$

如式 3.3 所示 W (權重)與 X 進行內積後，再將內積結果加上 B (偏權值)，此兩個矩陣在訓練時持續更新參數。全連接層會將訓練的結果轉換至輸出維度，但是由於此函數容易使神經網路的參數數量暴增，導致訓練的結果產生過度擬和，因此須謹慎使用。

神經網路初始化時通常會將需要更新參數的層預先建立完成，如表 3-5 第 4 至 8 行，完整的程式碼請參考附錄 A。初始化完成後建立 forward 函數，需要遵守圖 3-2 的順序將每個訓練時所需使用的函數逐一用上，其中有用到的函數為

「ReLU」、「Max_pooling」以及「Dropout」，每個函數所對應的意義如下：

※ ReLu：由於深度神經網路於學習的過程中，只有線性的函數會解決不了非線性的問題，因此需要非線性函數加入其中，而 ReLu 為常用且優質的非線性函數。

※ Max_pooling：此函數為幫助神經網路抓取最大特徵值，並將其他抹去的函數。

※ Dropout：此函數的目的在於解決深度學習中 Over-Fitting 的問題，使用方法以隨機的方式從訓練參數中抹去部分參數，以降低 Over-Fitting 的效果。

表 3-5 訓練開始前的調適參數設定

行數	內容
1.	<code>lr_set = 0.01</code>
2.	<code>set_drop_out = 0.5</code>
3.	<code>SLTimes = 100</code>
4.	<code>number = int((len_trainloader/SLTimes)/set_batch_size)</code>
5.	<code>show_loss = np.array([0.0])</code>
6.	<code>show_train_accuracy = np.array([0.0])</code>
7.	<code>show_test_accuracy = np.array([0.0])</code>
8.	<code>loss_stop = 0.08</code>
9.	<code>epoch_save = 16</code>
10.	<code>epoch_num = 256</code>
11.	<code>train_last_batch_size = len_trainloader % set_batch_size</code>
12.	<code>test_last_batch_size = len_testloader % set_batch_size</code>

調適深度神經網路時，通常會以訓練準確率、測試準確率以及損失量做為調適的標準，其他學習框架觀察此標準的方法都會被固定於函數，意味著需要判斷神經網路的品質時，只能透過特定函數以進行觀察，有時候會為調適超參數帶來困擾，而 PyTorch 於 `torchvision.transforms` 類別底下提供了的兩個函數 `ToTensor` 與 `ToPILImage`，此函數的作用在於將變數的類型自由地切換成 `torch.FloatTensor` 或 `numpy.ndarray`，當訓練的過程中參數或輸出的結果可以切換成 `numpy.ndarray` 時，意味著觀察標準的時機可以依程式的需求自行定義，因此增加了調適神經網路的靈活性。表 3-6 中定義了一些變數做為調適神經網路的參考，以下為這些變數的解釋。

- ※ lr_set：設定 learging rate，決定網路學習的快慢。
- ※ set_drop_out：設定 drop_out 參數，決定丟棄參數的比例。
- ※ SLTimes：每 epoch 所顯示 loss 的次數。
- ※ number：根據此參數計算顯示 epoch 的時機。
- ※ show_loss：紀錄每次 epoch 的損失量。
- ※ show_train_accuracy：紀錄每次 epoch 的訓練準確率。
- ※ show_test_accuracy：紀錄每次 epoch 的測試準確率。
- ※ loss_stop：當損失量低於此參數時則停止訓練
- ※ epoch_save：當 epoch 的數量整除此參數時，則儲存目前的模型與參數。
- ※ epoch_num：當 epoch 的數量超過此變數時則停止訓練。
- ※ train_last_batch_size：神經網路每次 epoch 進行最後一次正向傳遞時，除非小批次與資料數量整除，否則最後一次小批次之數值應使用此參數。
- ※ test_last_batch_size：同上(測試時最後一次小批次之數值應使用此參數)

表 3-6 訓練開始前設定 GPU、損失函數與最佳化方法

行數	內容
1.	if torch.cuda.is_available():
2.	net = Net()
3.	net.cuda()
4.	loss_function = nn.CrossEntropyLoss()
5.	optimizer = torch.optim.SGD(net.parameters(), lr=lr_set, \ momentum=0.9, weight_decay=5e-4)
6.	time_Start = time.time()
Training neuron network...	
7.	time_End = time.time()

關於神經網路訓練前需先設定損失函數、最佳化方法以及訓練網路所使用的硬體，本次實驗所設定的損失函數為「Cross-Entropy-Loss」與最佳化方法為「SGD-Momentum」，因為訓練所使用的硬體預設為 CPU，所以欲使用 GPU 支援網路訓練時，如表 3-6 第 3 行所示，將初始化的神經網路轉換為「cuda」使 GPU 支援神經網路進行訓練，完成以上步驟後開始訓練網路，此時設定 time_Start 為訓練開始的時間，訓練結束後設定 time_End 為訓練結束的時間，以記錄本次訓練總共花費的時間。

表 3-7 訓練流程之程式碼介紹

行數	內容
1.	while True:
2.	net.train()
3.	for i, data in enumerate(trainloader, 0):
4.	inputs, labels = data
5.	inputs = inputs.to(device)
6.	labels = labels.to(device)
7.	net.zero_grad()
8.	outputs = net.forward(inputs, set_batch_size, set_dropout, train)
9.	_, predicted = torch.max(outputs.data, 1)
10.	train_accuracy += (predicted == labels).sum().item()
11.	loss = loss_function(outputs, labels)
12.	loss.backward()
13.	optimizer.step()
14.	net.eval()
Testing...	
15.	epoch = epoch + 1
16.	if epoch >= epoch_num or epoch_loss/SLTimes < loss_stop:
17.	break

關於 PyTorch 訓練網路時與其他學習框架不同的地方在於撰寫 AI 程式碼的模式，大部分學習框架是將神經網路架設完成後，使用特定方法進行網路訓練與顯示準確率與損失量等，例如：Keras 的 compile 函數與 fit 函數(Kears 是以 Tensorflow 為背景的高階深度學習框架)，而 PyTorch 進行訓練時如表 3-8，forward 函數、計算準確率、計算損失(loss)、進行最佳化，每一個步驟都有條有理，使設計者調整程式碼或是調適 Hyperparameter 都會比其他學習框架靈活許多。

接下來將開始解釋其中的內容，最外層的迴圈為永遠執行的 while，停下的條件有兩個只要達成一條則跳出此迴圈，其一為設定 epoch 最高的上限數，while 每執行一次迴圈 epoch 的數量就會加一，當 epoch 的數量超過設定的上限時，則跳出迴圈停止訓練；另一則是設定 loss 的最低下限，loss 的意思是此模型預測的結果距離標準答案還差多遠的估計值，在理想的訓練過程中 loss 會隨著訓練的次數使數值降低，當此數值低於設定的下限時，則跳出迴圈停止訓練。

網路學習的過程中訓練與測試完成時稱為一次 epoch，當網路於訓練或是測試開始前需將網路轉換至訓練模式或測試模式，如第 2 行與第 14 行所示。訓練時所使用的資料為表 3-4 中的 trainloader 變數，此變數會依照小批次將資料分割，此時再用 for 迴圈搭配 enumerate 函數將 trainloader 變數的資料取出，enumerate 函數所回饋的變數有兩個，其一是以 list 的方式將資料一份一份地取出並儲存於 data 變數，另一則是回報目前 for 迴圈正在取出 trainloader 變數內的第幾份資料，使用此方法可以自動的將所有資料取出使用。

以上程式碼所產生的 data 變數中包含了資料的內容與標籤，為方便使用分別設定為 inputs 與 labels 變數。根據 PyTorch 的特性，導數的數值會自動儲存於網路之中，所以每次訓練開始前需將網路中所有的導數設定為零以避免與下次的導數相加，使結果不如預期。

接下來進行預測，將 inputs 放入網路內並呼叫 forward 函數，由於 forward 是自己製作的函數，因此如小批次以及測試網路時是否不開啟 dropout 的功能都需自行設定，雖然 PyTorch 需要注意這些細節，但是這些細節都是由創作者所控制，使撰寫程式碼時可以更加的靈活。

預測完成後則記錄下損失量與準確率，當損失計算完成後代表神經網路來到了最後一個節點，從這裡往前求導，使用最佳化方法計算參數更新的數值並將其更新，如表 3-8 第 10 至 13 行。以上為訓練網路的流程，測試網路的步驟與之相似，而不相同的地方在於測試網路時不要求導與更新網路的參數，完整的內容請參閱附錄 A。

以上所有的講解為 PyTorch 建立 CNN 網路的流程，曾經使用過 Keras 的 AI 研究者，初步接觸 PyTorch 時可能有些微地不習慣，因為大部分深度學習架構都偏向於包裝網路訓練的流程，以避免使用者於設計模型時遺漏了部分細節導至網路無法訓練，雖然 PyTorch 於建立網路時需要注意一些細節，但是這不意味著 PyTorch 是一個不易入手的學習框架，只要練習官方網站提供的範例，則漸漸可以了解 PyTorch 的核心概念，再藉由網路上許多創作者的開源碼，也可以變成靈活運用 PyTorch 的高手。

第4章 於 PyTorch 載入資料集之方法

4.1 台灣限速牌資料集

PyTorch 載入資料的方法，已經於第三章講解如何使用 torchvision.dataset 函數載入開源圖片資料集，但是 torchvision.dataset 函數所蒐集的資料集內容大多時候無法應用於實際的需求，因此除了使用提供的資料集，也要有能力將其它資料集載入至神經網路中進行訓練。

本論文使用的資料集為羅偉宸(2019, [6])「台灣限速牌圖片資料集」，其中蒐集了 20 至 110 公里共 11 種不同的限速牌，且有晴天、雨天、早上、晚上……等不同的路況，收集以上等資訊，使神經網路能夠應付不同的情況，此資料集共有訓練資料 26400 張圖片、測試資料 6600 張圖片，限速牌圖片如圖 4-1。



圖 4-1 台灣限速牌資料集

4.2 建立圖片路徑文字檔與標籤

PyTorch 載入資料之前，需建立圖片路徑文字檔並在每個路徑的後方標記標籤(label)，如圖 4-2，建立文字檔的作業可以藉由 os 函數庫底下的 path 類別與 listdir 函數完成，於 path 類別底下的 abspath 函數輸入「.」，可以得到目前程式位於電腦的絕對路徑，將此程式與資料集放在同一個資料夾，則能夠藉由程式將所有的圖片路徑寫入至文字檔內，而 listdir 函數的功用是在輸入資料夾路徑後，取得資料夾中所有項目的名稱，以方便建立圖片路徑文字檔。

關於寫入文字檔的方法有許多，其中較為廣泛使用的是 open 函數，該函數開啟一個副檔名為 txt 的檔案，並設定寫入模式，寫入完成後開啟文字檔確認圖片路徑以及路徑後方的標籤是否與圖片互相符合，如圖 4-2。



圖 4-2 圖片路徑檔之示意圖

4.3 建立符合 PyTorch 讀取資料之類別

關於 PyTorch 載入其他資料集的方法，首先需建立類別繼承於 `torch.utils.data.Dataset` 類別底下，功用是將所有資料包裝以方便使用，PyTorch 的資料集亦會繼承此類別。根據官方網站說明，必須建立兩個函數提供類別使用，分別為「`__getitem__(self, index)`」與「`__len__(self)`」，`getitem` 函數根據 `index` 變數將相對應的資料路徑與標籤取出，而 `len` 函數則會顯示此資料集共有幾筆資料，程式碼如表 4-1。

表 4-1 建立 PyTorch 讀取資料之類別

行數	內容
1.	<code>from PIL import Image</code>
2.	<code>from torch.utils.data import Dataset</code>
3.	<code>class MyDataset(Dataset):</code>
4.	<code>def __init__(self, txt_path, transform = None):</code>
	<code>initializing</code>
5.	<code>def __getitem__(self, index):</code>
6.	<code>image_location, label = self.imgs[index]</code>
7.	<code>img = Image.open(image_location).convert('RGB')</code>
8.	<code>if self.transform is not None:</code>
9.	<code>img = self.transform(img)</code>
10.	<code>return img, label</code>
11.	<code>def __len__(self):</code>
12.	<code>return len(self.imgs)</code>

從表 4-1 第 6 行發現 self.imgs 變數是根據 image 變數取出對應的資料路徑與標籤，其中 self.imgs 於 MyDataset() 初始化的過程中所建立，建立的過程只需將文字檔內的資料路徑與標籤分開後分別儲存於 list 即可，程式碼請參考附錄 A。

4.4 台灣限速牌資料集訓練結果

開始訓練之前建議觀察訓練資料的複雜程度，CIFAR-10 是一個訓練資料相當複雜的資料庫，內容有 10 種不同的動物、交通工具等物體，其中貓狗等動物有時會分不清楚，因此 CIFAR-10 需使用多層網路進行訓練，但是限速牌與 CIFAR-10 的图片相比，限速牌的圖樣相較簡單(如圖 4-1)，若使用 VGG16 模型進行訓練其結果大多時候較差，因此參考羅偉宸(2019, [6])論文中所使用的 CNN 模型對限速牌資料集進行訓練。如圖 4-3。

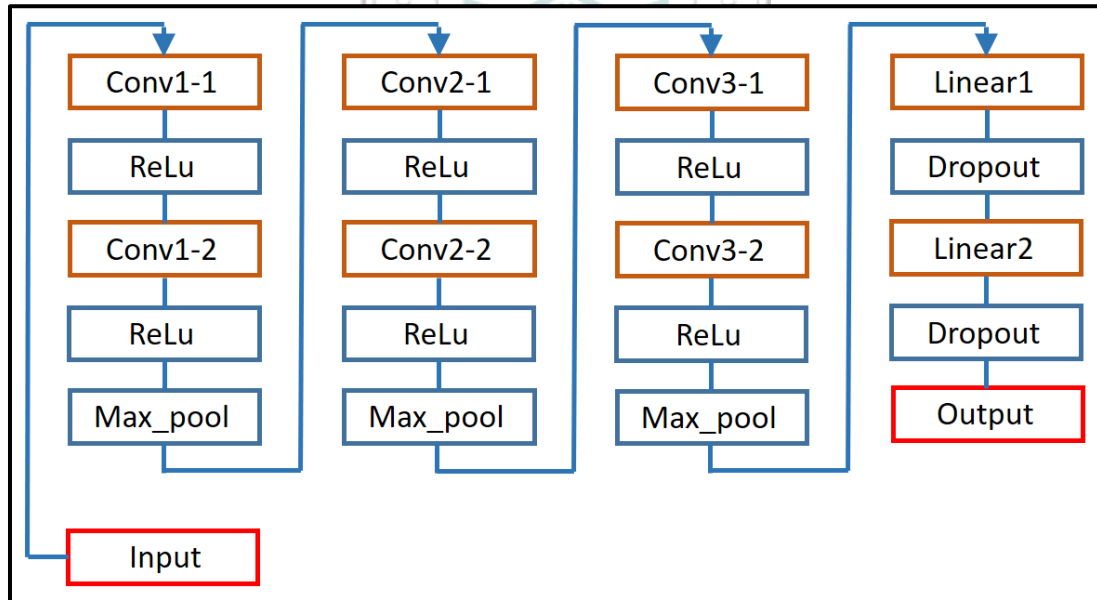


圖 4-3 限速牌的訓練模型

此模型訓練時設定最低的損失量為 0.08，且訓練只使用了 5 至 10 分鐘就達到設定值，因此停止訓練，模型的訓練準確率與測試準確率終值是「97.6%」與「95.1%」相當不錯，epoch 所對應的準確率如圖 4-4、對應損失量如圖 4-5。

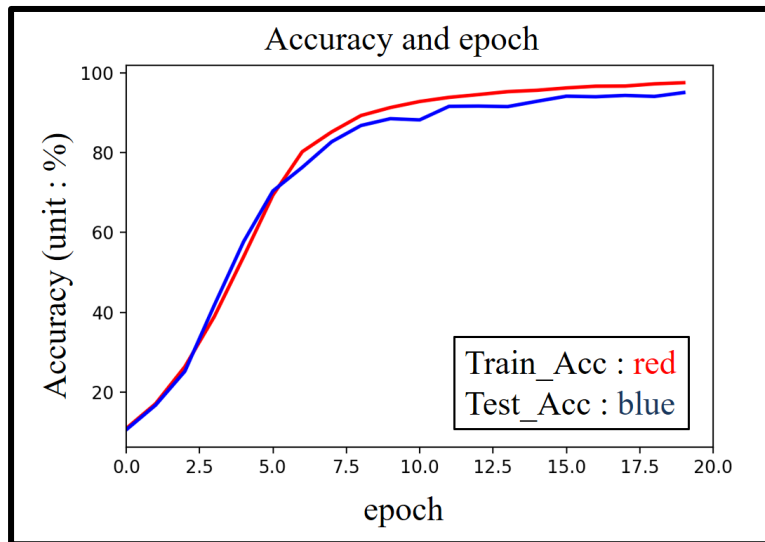


圖 4-4 對應 epoch 的訓練與測試準確率

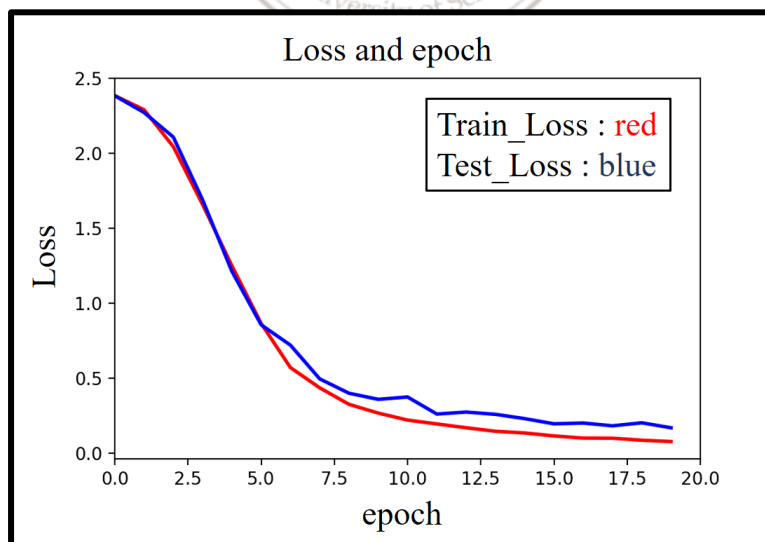


圖 4-5 對應 epoch 的損失量

由於本模型訓練時間相當快速，因此將探討有無使用表 3-4 第 1 行中的 Normalize 函數對訓練效果的影響，圖片對應 RGB 三色道，分別加總資料數據再除以「資料張數*圖片大小*255」(將數據轉換至[0, 1]區間)以得到三色道各別的平均值，再計算標準差，經過 Normalize 函數計算後得到對應資料集 RGB 的平均值與標準差，如表 4-2。

表 4-2 限速牌資料集對應 RGB 三色道的平均值與標準差

色道	平均數	標準差
R	0.46762	0.05939
G	0.39931	0.06074
B	0.42414	0.05936

圖 4-4 與 4-5 為沒有 Normalize 函數計算的實驗結果，而圖 4-6 與 4-7 為資料進行 Nomalize 函數計算後的實驗結果。

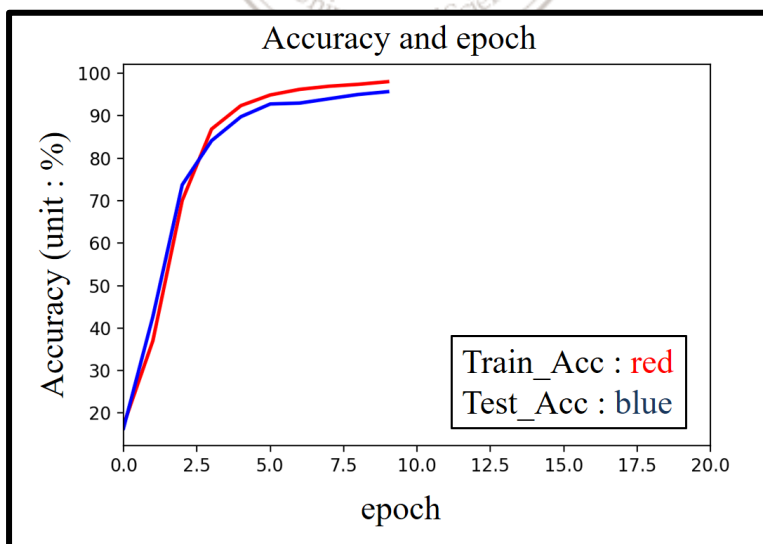


圖 4-6 資料於 Normalize 後對應 epoch 所得到的準確率

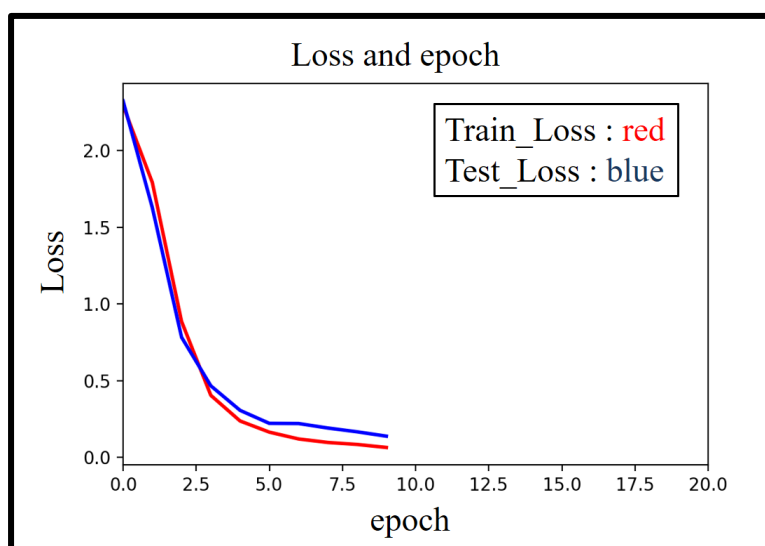


圖 4-7 資料於 Normalize 後對應 epoch 所得到的損失量

不管資料集在訓練前有沒有進行 Normalize 函數，網路的訓練成果都相當不錯，但是有進行 Normalize 的資料集，網路的收斂速度有明顯上升如圖 4-6，因此當訓練速度緩慢，建議使用 Normalize 函數幫助模型進行訓練。

第5章 PyTorch 動態圖

5.1 動態圖與靜態圖

討論動態圖(Dynamic Graph)與靜態圖(Static Graph)之前需先定義訓練流程圖(Graph)的意思，訓練流程圖如同圖 3-2 與圖 4-3，建立網路時，思考網路需要的神經層與函數以及使用的數量。靜態圖的概念是網路在訓練的過程中不改變網路的超參數，而動態圖是網路訓練中調整超參數以得到更好的結果，觀察附錄 A 程式撰寫的流程後可以發現，每一個迴圈是由程式設計者所控制，因此可以自由調整訓練流程與超參數，5.2 節介紹如何使用 PyTorch 於訓練中改變學習率。

5.2 PyTorch 動態圖之應用 – 改變學習率

討論學習率之前，需先探討局部最小值(Local minimum)與全域最小值(Global minimum)，以峽谷做為比喻，局部最小值是峽谷內較低的山溝，而全域最小值為峽谷中最低的山溝，如圖 5-1 所示。

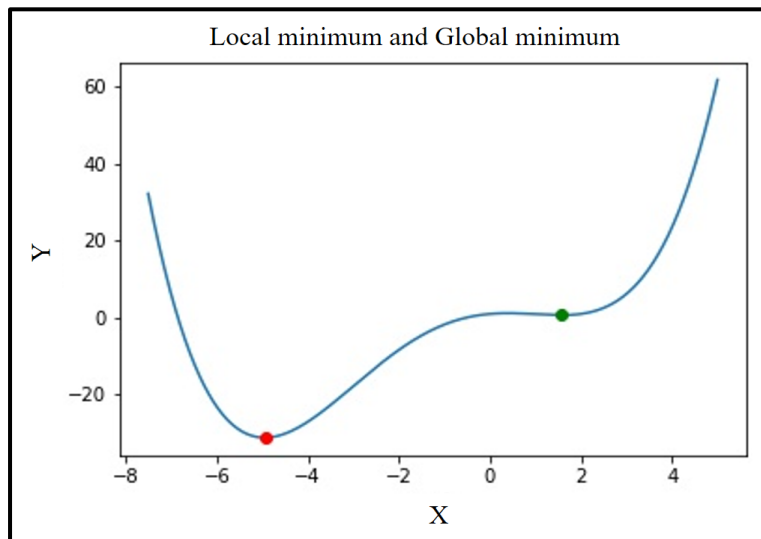


圖 5-1 局部最小值與全域最小值

如圖 5-1 綠色的點為局部最小值、紅色的點為全域最小值，神經網路的訓練過程是根據損失量作為指標尋找全域最小值的位置，而學習率可以想像成在山谷中移動的步伐大小，當步伐太大時可能會找不到全域最小值，或是在全域最小值的附近來回走動一直找到全域最小值，但是當步伐太小時可能會使模型收斂在損失量較高的地方，並且走得太慢也會導致訓練的時間增加，因此將採測試的方法嘗試出最好的學習率。

學習率太大可能導致網路無法收斂，而太小則使訓練的時間增加，因此在訓練開始時設定較大的學習率，並在訓練的過程中使用「監控測試準確率的趨勢以調整學習率的大小」讓網路知道調整學習率的時機，使網路可以更加靈活地學習。

使用固定長度的一維矩陣，訓練開始後矩陣會慢慢地被測試準確率所填滿，當矩陣被填滿後，對矩陣內的數值進行線性擬合(Curve Fitting)並記錄斜率，當斜率低於預設的數值時，則代表模型的學習狀況已經停滯，需降低學習率使模型繼續學習，於本論文矩陣的長度預設為「8」及斜率閾值為「0.03」，此方法不光使用在判斷學習率更新的時機，也可以判斷模型停止訓練的時機，當學習率更新二至三次之後會發現再降低學習率的數值，準確率也不會有明顯的提升時則可以作為停止訓練的指標。

圖 5-2 與圖 5-3 為 VGG-16 模型訓練 CIFAR-10 資料集沒有調整學習率之結果，當 epoch 超過 30 之後準確率就沒有再上升的趨勢，此情形很適合藉由降低學習率以增加學習效果。

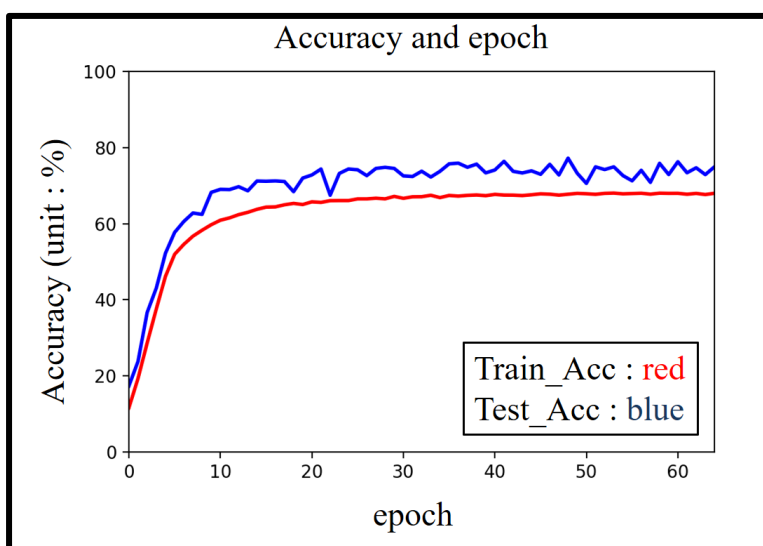


圖 5-2 模型訓練沒有調整學習率對應 epoch 所呈現的準確率

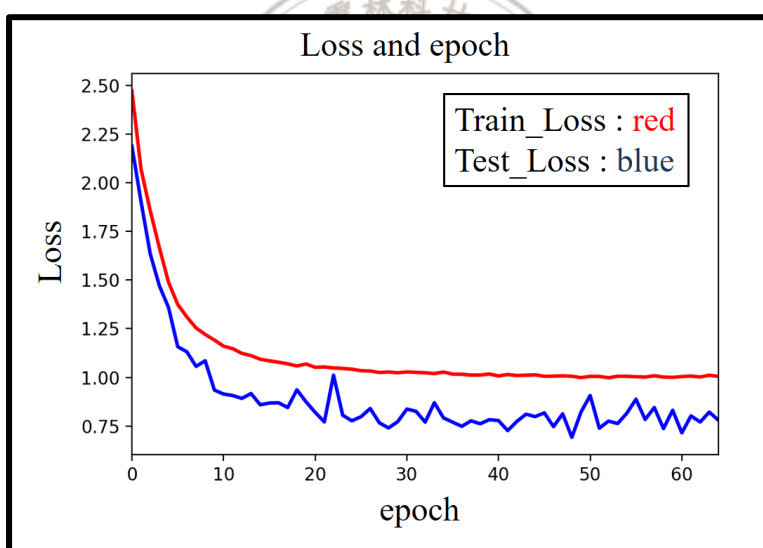


圖 5-3 模型訓練沒有調整學習率對應 epoch 所呈現的損失量

圖 5-4、5-5 為應用動態圖的結果，將原本只有 60 至 70%的準確率，最終提高至 85%以上，由此可知使用動態圖的概念，會讓網路發揮出更好的效果。

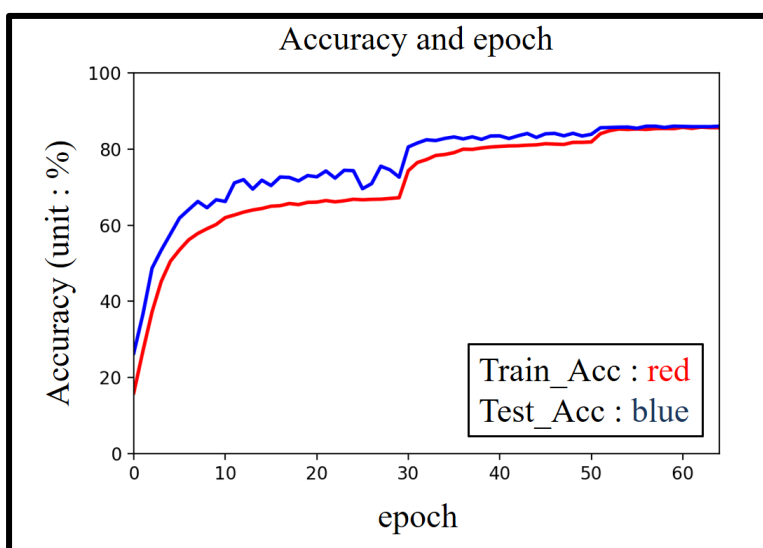


圖 5-4 模型於訓練過程中調整學習率對應 epoch 所呈現的準確率

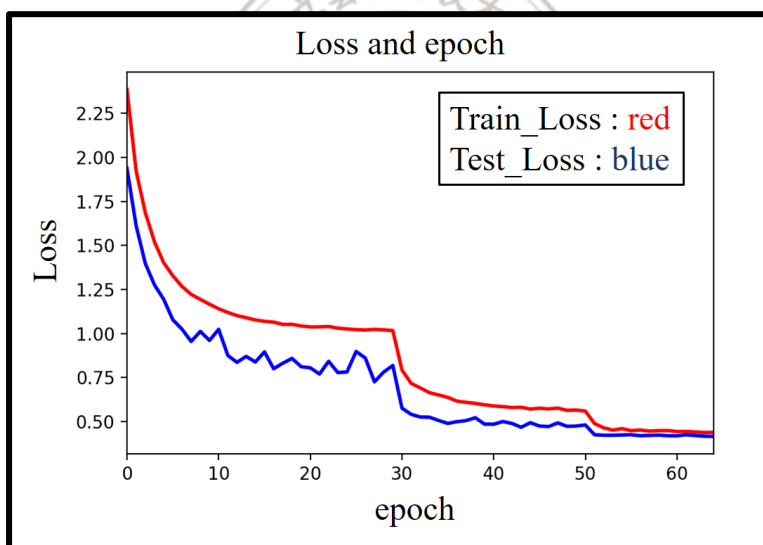


圖 5-5 模型於訓練過程中調整學習率對應 epoch 所呈現的損失量

第6章 結論與建議

6.1 結論

關於初入 PyTorch 的 AI 研究者，從如何安裝 PyTorch 到建立模型與載入自己的資料集，本論文的第二至四章中都有明確的說明，使用 PyTorch 建立神經網路的過程中，可以從中體悟到 PyTorch 提供給研究者們的概念是「建立神經網路不應該是單純的增加層數與設定參數，而是可以將自己的想法實現於訓練網路的過程」。第五章中動態圖的概念是於訓練中更新超參數，以幫助模型有更好的訓練效果，此概念也存在於 Keras 的函數中，但是與 Keras 不同的地方是 PyTorch 可以依照自己的想法決定在甚麼時候更改超參數，且訓練過程是由迴圈所建立，因此可以靈活的編輯程式碼。

PyTorch 的靈活度與使用體驗都十分地優秀，如圖 5-6 與圖 5-7 使用自己的想法決定更新學習率的時機，此使用體驗與效果是在其他框架中前所未有的，但是這並非說明 PyTorch 屬於最好的學習框架，與目前最廣為人知的學習框架 Tensorflow 做比較，Tensorflow 的優點在於擁有許多擴充應用程式幫助觀察模型的品質、支援其他平台以及作為產品的穩定度，這些都是 PyTorch 尚需加強的部分，因此 PyTorch 的使用者大多都是為了以下兩點優點而來：

1. 以更加靈活的方式建立網路，對建立網路的過程有更好的使用體驗。
2. 更多「Python 化」的使用方式。

上述說明都是為了更加了解 PyTorch 的使用方法及其概念，並期待研究者們可以實際操作 PyTorch 建立屬於自己的深度神經網路。

6.2 建議

使用 PyTorch 建立神經網路之前，建議閱讀 AI 領域的相關書籍，並不依靠深度學習框架練習建立神經網路，因為 PyTorch 的宗旨在於「靈活地使用自己的想法進行網路訓練」，所以沒有了解 AI 的理論或是自行建立深度網路的過程，會導致只能按照範例或網路上的開源資料依樣畫葫蘆，而無法體會 PyTorch 傳達給研究者們的概念。



參考文獻

- [1] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang Junjie Bai Soumith Chintala, 2019, “PyTorch: An Imperative Style, High-Performance Deep Learning Library”, 33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.
- [2] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep Learning. MIT Press, 2016. <http://www.deeplearningbook.org>.
- [3] Johan Bjorck, Carla Gomes, Bart Selman, Kilian Q. Weinberger, 2018, “Understanding Batch Normalization”, 32nd Conference on Neural Information Processing Systems (NeurIPS 2018), Montréal, Canada.
- [4] Yanzhao Wu+, Ling Liu+, Juhyun Bae+, Ka-Ho Chow+, Arun Iyengar*, Calton Pu+, Wenqi Wei+, Lei Yu*, Qi Zhang*, 2019, “Demystifying Learning Rate Policies for High Accuracy Training of Deep Neural Networks”, +School of Computer Science, Georgia Institute of Technology, Atlanta, GA, USA, * IBM Thomas. J. Watson Research, Yorktown Heights, NY, USA
- [5] 齋藤康毅，2017，Deep Learning：用 Python 進行深度學習的基礎理論實作，台灣，美商歐萊禮股份有限公司台灣分公司
- [6] 羅偉宸，2019，人工智慧於道路限速標誌辨識之研究，雲林科技大學機械工程系
- [7] Karen Simonyan *, Andrew Zisserman +, 2015, Very Deep Convolutional Networks for Large-Scale Image Recognition, * current affiliation: Google DeepMind + current affiliation: University of Oxford and Google DeepMind

附錄 A

```
1
2 """
3 Created on Sat Feb 22 16:01:47 2020
4
5 @author: mps6401
6 """
7
8 import torch
9 import torch.nn as nn #建立關於深度學習需要使用的類別
10 import torch.nn.functional as F #關於深度學習需要使用的函數
11 import torch.optim as optim #尋找gradient的函數
12 import torchvision.datasets as datasets #載入pytorch的預置資料庫
13 import torchvision.transforms as transforms #存有對資料進行處理的函數
14 import matplotlib.pyplot as plt
15 import numpy as np
16 import time
17 from os import path #尋找資料夾位置
18 from os import mkdir #建立新的資料夾
19
20
21 class Net(nn.Module):
22     def __init__(self):
23         super(Net, self).__init__()
24         self.conv1_1 = nn.Conv2d(3, 32, 3, padding = 1)
25         self.BatchNorm2d1_1 = nn.BatchNorm2d(32)
26         self.conv1_2 = nn.Conv2d(32, 32, 3, padding = 1)
27         self.BatchNorm2d1_2 = nn.BatchNorm2d(32)
28
29         self.conv2_1 = nn.Conv2d(32, 64, 3, padding = 1)
30         self.BatchNorm2d2_1 = nn.BatchNorm2d(64)
31         self.conv2_2 = nn.Conv2d(64, 64, 3, padding = 1)
32         self.BatchNorm2d2_2 = nn.BatchNorm2d(64)
33
34         self.conv3_1 = nn.Conv2d(64, 128, 3, padding = 1)
35         self.BatchNorm2d3_1 = nn.BatchNorm2d(128)
36         self.conv3_2 = nn.Conv2d(128, 128, 3, padding = 1)
37         self.BatchNorm2d3_2 = nn.BatchNorm2d(128)
38         self.conv3_3 = nn.Conv2d(128, 128, 3, padding = 1)
39         self.BatchNorm2d3_3 = nn.BatchNorm2d(128)
40
41         self.conv4_1 = nn.Conv2d(128, 256, 3, padding = 1)
42         self.BatchNorm2d4_1 = nn.BatchNorm2d(256)
43         self.conv4_2 = nn.Conv2d(256, 256, 3, padding = 1)
44         self.BatchNorm2d4_2 = nn.BatchNorm2d(256)
45         self.conv4_3 = nn.Conv2d(256, 256, 3, padding = 1)
46         self.BatchNorm2d4_3 = nn.BatchNorm2d(256)
47
48         self.conv5_1 = nn.Conv2d(256, 512, 3, padding = 1)
49         self.BatchNorm2d5_1 = nn.BatchNorm2d(512)
50         self.conv5_2 = nn.Conv2d(512, 512, 3, padding = 1)
51         self.BatchNorm2d5_2 = nn.BatchNorm2d(512)
52         self.conv5_3 = nn.Conv2d(512, 512, 3, padding = 1)
53         self.BatchNorm2d5_3 = nn.BatchNorm2d(512)
54
55         self.classifier = nn.Linear(512, 10)
56
57
58     def forward(self, x, batch_size, set_dropout, train):
59
60         x = F.relu(self.BatchNorm2d1_1(self.conv1_1(x)), inplace = True)
61         x = F.relu(self.BatchNorm2d1_2(self.conv1_2(x)), inplace = True)
62         x = F.max_pool2d(x, kernel_size = 2, stride = 2)
63         if train:
64             x = F.dropout(x, p = set_dropout)
65
66         x = F.relu(self.BatchNorm2d2_1(self.conv2_1(x)), inplace = True)
67         x = F.relu(self.BatchNorm2d2_2(self.conv2_2(x)), inplace = True)
68         x = F.max_pool2d(x, kernel_size = 2, stride = 2)
69         if train:
70             x = F.dropout(x, p = set_dropout)
71
72         x = F.relu(self.BatchNorm2d3_1(self.conv3_1(x)), inplace = True)
73         x = F.relu(self.BatchNorm2d3_2(self.conv3_2(x)), inplace = True)
74         x = F.relu(self.BatchNorm2d3_3(self.conv3_3(x)), inplace = True)
75         x = F.max_pool2d(x, kernel_size = 2, stride = 2)
76         if train:
77             x = F.dropout(x, p = set_dropout)
78
79         x = F.relu(self.BatchNorm2d4_1(self.conv4_1(x)), inplace = True)
80         x = F.relu(self.BatchNorm2d4_2(self.conv4_2(x)), inplace = True)
81         x = F.relu(self.BatchNorm2d4_3(self.conv4_3(x)), inplace = True)
82         x = F.max_pool2d(x, kernel_size = 2, stride = 2)
83         if train:
84             x = F.dropout(x, p = set_dropout)
85
86         x = F.relu(self.BatchNorm2d5_1(self.conv5_1(x)), inplace = True)
87         x = F.relu(self.BatchNorm2d5_2(self.conv5_2(x)), inplace = True)
88         x = F.relu(self.BatchNorm2d5_3(self.conv5_3(x)), inplace = True)
89         x = F.max_pool2d(x, kernel_size = 2, stride = 2)
90         if train:
91             x = F.dropout(x, p = set_dropout)
92
93         x = x.view(batch_size, -1)
94         x = self.classifier(x)
95         return x
96
97
```



```

108 #####
109 if __name__ == '__main__':
110     # check cuda is available
111     if torch.cuda.is_available():
112         device = torch.device("cuda:0")
113     else:
114         device = torch.device("cpu")
115
116     # set transforms for datas
117     data_transform = transforms.Compose([transforms.ToTensor(),
118                                         transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010))])
119
120     # set batch size
121     set_batch_size = 8
122
123     # Load and set train datas
124     trainset = datasets.CIFAR10(root = './data', train = True,
125                                download = False, transform = data_transform)
126     trainloader = torch.utils.data.DataLoader(trainset, batch_size = set_batch_size,
127                                              shuffle = True)
128     len_trainloader = len(trainloader.dataset)
129
130     # Load and set test datas
131     testset = datasets.CIFAR10(root = './data', train = False,
132                                download = False, transform = data_transform)
133     testloader = torch.utils.data.DataLoader(testset, batch_size = set_batch_size,
134                                              shuffle = False)
135     len_testloader = len(testloader.dataset)
136
137     # parameter for count and record informations
138
139     lr_set = 0.01 # 設定learning rate
140     set_dropout = 0.5
141     SLTimes = 100 # 每一個epoch顯示loss的次數
142     number = int((len_trainloader/SLTimes)/set_batch_size) # 顯示loss的參數
143     show_train_loss = np.array([0.0]) # 紀錄每次epoch的訓練損失量
144     show_test_loss = np.array([0.0]) # 紀錄每次epoch的測試損失量
145     show_train_accuracy = np.array([0.0]) # 紀錄每次epoch的訓練準確率
146     show_test_accuracy = np.array([0.0]) # 紀錄每次epoch的測試準確率
147     epoch_save = 16 # 每當epoch的數量整除此參數時儲存一次模型與參數
148     epoch_num = 128 # 當epoch的數量超過此參數時則停止訓練
149     loss_stop = 0.08 # 當loss小於此數值則停止訓練
150
151     # parameter for mini-batch
152     train_last_batch_size = len_trainloader % set_batch_size # 最後一個 epoch 的batch size
153
154     test_last_batch_size = len_testloader % set_batch_size # 最後一個 epoch 的batch size
155
156     #####
157     epoch = 0
158     if torch.cuda.is_available():
159         net = Net() # 網路初始化
160         net.cuda() # 透過cuda使用GPU
161         loss_function = nn.CrossEntropyLoss() # 設定Loss function
162         optimizer = optim.SGD(net.parameters(), lr=lr_set, momentum=0.9, weight_decay=5e-4) # 設定最佳化方法
163         time_Start = time.time() # 訓練開始時間
164
165         # build txt file with write mode
166         f = open("info.txt", "w")
167
168         while True: # do while loop
169             print("epoch : " + str(epoch+1))
170
171             # parameters
172             train_total = 0 # 訓練的總數
173             train_accuracy = 0 # 訓練正確的總數
174             test_total = 0 # 測試的總數
175             test_accuracy = 0 # 測試正確的總數
176             batch_loss = 0.0 # 累加每一個batch的Loss
177             train_epoch_loss = 0.0 # epoch的訓練損失量
178             test_epoch_loss = 0.0 # epoch的測試損失量
179             batch_counter = 0
180             slope = 0
181
182             ##### start training #####
183             time1_batch_train = time.time()
184             net.train() # switch mode of model is training
185             training = True
186             # enumerate : 根據 trainloader 的數量回傳 trainloader 的內容與此內容的 index
187             # 第二個參數為 0 代表index從 0 開始
188             for i, data in enumerate(trainloader, 0):
189                 # get the inputs and labels per batch size
190                 inputs, labels = data # 根據batch_size決定inputs與labels的數量
191
192                 # CPU device convert to GPU
193                 inputs = inputs.to(device)
194                 labels = labels.to(device)
195
196                 # zero the parameter gradients

```

```

191 net.zero_grad() #將網路中的每個gradient歸零，不歸零則會使gradient疊加
192
193 # forward
194 if batch_counter < int(len_trainloader/set_batch_size):
195     outputs = net.forward(inputs, set_batch_size, set_dropout, training)
196 else:
197     outputs = net.forward(inputs, train_last_batch_size, set_dropout, training)
198 batch_counter = batch_counter+1
199
200 # loss
201 loss = loss_function(outputs, labels)
202 batch_loss += loss.item()
203
204 # predicted
205 _, predicted = torch.max(outputs.data, 1)
206 # 回傳最大值，回傳最大值的index = torch.max(outputs.data, 0) 直(柱)向
207 # 回傳最大值，回傳最大值的index = torch.max(outputs.data, 1) 橫向
208
209 # count train accuracy
210 train_total += labels.size(0) #根據每次 batch size 的數量增加
211 train_accuracy += (predicted == labels).sum().item() #根據每次的「predicted == labels」的數量增加
212
213 # backward
214 loss.backward() #計算backpropagation
215
216 # optimize
217 optimizer.step() #更新參數
218
219 # show loss by SLTimes
220 if i % number == number-1:
221     time2_batch_train = time.time()
222     print("[ " + str(i*set_batch_size) + "/" + str(len_trainloader) + " ] loss : " + str(batch_loss / number) +
223           " use_time : " + str(time2_batch_train - time1_batch_train))
224
225     train_epoch_loss = train_epoch_loss + batch_loss / number
226     batch_loss = 0.0
227     time1_epoch_train = time.time()
228
229 batch_counter = 0
230 ##### end training per epoch #####
231
232
233 ##### start testing predict #####
234 # show train accuracy and loss per epoch
235 print('Train Accuracy : ' + str(100 * train_accuracy / train_total) + '%')
236 print("Train loss : " + str(train_epoch_loss/SLTimes))
237
238 # record loss and train accuracy to array
239
240 show_train_loss = np.append(show_train_loss, [train_epoch_loss/SLTimes], axis = 0)
241 show_train_accuracy = np.append(show_train_accuracy, [100 * train_accuracy / train_total], axis = 0)
242
243 # predict the test datas
244 net.eval() # switch mode of model is testing
245 training = False
246 #根據pytorch的特性，每個有關網路架構的物件都會自動計算梯度，而 torch.no_grad() 告訴torch以下的程式不用計算梯度
247 with torch.no_grad():
248     for data in testloader:
249         # get the inputs and labels per batch size
250         inputs, labels = data
251         inputs = inputs.to(device)
252         labels = labels.to(device)
253
254         # forward
255         if batch_counter < int(len_testloader/set_batch_size):
256             outputs = net.forward(inputs, set_batch_size, set_dropout, training)
257         else:
258             outputs = net.forward(inputs, test_last_batch_size, set_dropout, training)
259         batch_counter = batch_counter+1
260
261         # loss
262         loss = loss_function(outputs, labels)
263         test_epoch_loss += loss.item()
264
265         # predicted
266         _, predicted = torch.max(outputs.data, 1)
267         # 回傳最大值，回傳最大值的index = torch.max(outputs.data, 0) 直(柱)向
268         # 回傳最大值，回傳最大值的index = torch.max(outputs.data, 1) 橫向
269
270         # count test accuracy
271         test_total += labels.size(0) #根據每次的Labels數量增加
272         test_accuracy += (predicted == labels).sum().item() #根據每次的(predicted == labels)的總數增加
273
274 ##### end testing predict per epoch #####
275
276 # show test loss and test accuracy per epoch
277 print('Test Accuracy : ' + str(100 * test_accuracy / test_total) + '%')
278 print('Test Loss : ' + str(test_epoch_loss/batch_counter))
279
280 # record test accuracy to array
281 show_test_loss = np.append(show_test_loss, [test_epoch_loss/batch_counter], axis = 0)
282 show_test_accuracy = np.append(show_test_accuracy, [100 * test_accuracy / test_total], axis = 0)
283
284 # count epochs
285 epoch = epoch + 1
286
287 # write txt

```

```

287         f.writelines(["epoch : " + str(epoch) + "\n",
288                       "Train_loss : " + str(train_epoch_loss/SLTimes) + "\n",
289                       "Test_loss : " + str(test_epoch_loss/betch_counter) + "\n",
290                       "Train Accuracy : " + str(100 * train_accuracy / train_total) + "%\n",
291                       "Test Accuracy : " + str(100 * test_accuracy / test_total) + "%\n"])
292     f.write("\n")
293
294     # save model, parameters and informations by epoch_save
295     if epoch%epoch_save == 0:
296
297         # build save folder
298         localtion_save = path.abspath('.') + '\\dir_save_epoch_' + str(epoch) #建立儲存資料夾
299         if not path.isdir(localtion_save): #如果沒有資料夾
300             mkdir(localtion_save) #則建立一個資料夾
301
302         # save model and parameters
303         PATH = localtion_save + '\\paper_use_eopch' + str(epoch) + '.pth'
304         torch.save(net, PATH)
305
306         # the limit to close train process
307         if epoch>=epoch_num or train_epoch_loss/SLTimes < loss_stop: # Loop for do_while
308             # close txt
309             f.close()
310             break
311
312         time_End = time.time() #訓練結束時間
313         print('Finished Training')
314         print("GPU traning time : " + str(time_End - time_Start))
315     else:
316         print("This computer havn't GPU")
317
318     #####
319
320     # 刪除 show_loss, show_train_accuracy, show_test_accuracy 的第一個元素
321     show_train_loss = np.delete(show_train_loss ,0)
322     show_test_loss = np.delete(show_test_loss ,0)
323     show_train_accuracy = np.delete(show_train_accuracy ,0)
324     show_test_accuracy = np.delete(show_test_accuracy ,0)
325
326     # show Loss
327     plt.plot(np.arange(0, len(show_train_loss)), show_train_loss, color="red", linewidth=2)
328     plt.plot(np.arange(0, len(show_test_loss)), show_test_loss, color="blue", linewidth=2)
329     plt.xlim((0,100))
330     plt.xlabel("Epoch")
331     plt.ylabel("Loss")
332     plt.title("Loss and epoch")
333     plt.savefig('loss.png', dpi=200)
334     plt.show()
335
336     # show Accuracy
337     plt.plot(np.arange(0, len(show_train_accuracy)), show_train_accuracy, color="red", linewidth=2)
338     plt.plot(np.arange(0, len(show_test_accuracy)), show_test_accuracy, color="blue", linewidth=2)
339     plt.xlim((0,100))
340     plt.xlabel("Epoch")
341     plt.ylabel("Accuracy")
342     plt.title("Accuracy and epoch")
343     plt.savefig('Accuracy.png', dpi=200)
344     plt.show()
345
346     # save final model and parameters
347     PATH = 'final_paper_use.pth' # 保存整個網路
348     torch.save(net, PATH)
349
350

```