國立臺灣師範大學
資訊工程學系碩士論文

指導教授：　紀博文　　博士

通用於第一人稱射擊遊戲外掛檢測機制之研究

A Study on Cheating Detection Mechanism for Generic FPS Games

研究生：　陳逸文　　撰

中華民國　　111　　年　9　月

致謝

首先要感謝指導教授紀博文教授，每次報告的建言讓我的研究方向更加明確並讓我確認了論文題目，之後的資料蒐集也是在教授的幫助下才得以完成，在線上工作坊的投稿與畢業論文的修改您都給予我很大的幫助，讓我能一步步地完成這本碩士論文。

感謝實驗室的秦昊、炫豪、惠安學長，協助我在密碼學與資訊安全方面的學習，也感謝瑞鴻、儀君、昀修、哲瑋、哲銘同學對我論文研究的建議與幫助，沒有你們的幫助，我無法完成這本論文。

摘 要

隨著科技的飛速發展，玩家可以在一台個人電腦上遊玩各種類型的遊戲，在各類型遊戲中，網路遊戲是大多數玩家最喜愛的遊戲類型，玩家為了在網路遊戲中獲得更好的成就，開始使用外掛程式達成個人無法實現的目標，基於上訴原因，作弊偵測成為了遊戲廠商的重大課題。

本研究提出了一種基於影像辨識並以數據檢測輔助的作弊檢測系統，並分別使用 VGG16、ResNet50、MobileNet V2、Xception 和 Inception v3 對誠實玩家和作弊玩家的瞄準軌跡進行檢測，研究結果表明，Inception V3 能最準確的分辨誠實玩家與作弊玩家。

關鍵字：機器學習、作弊偵測、Inception V3、自動瞄準、FPS

ABSTRACT

With the rapid development of technology, players can use a personal computer to play a variety of games. Of all kinds of games, online games are the most popular game type for most players. To obtain better achievements in online games, players begin to use game cheat to achieve goals that cannot be achieved by individuals. Due to the above, cheat detection becomes the most important issue for game manufacturers.

This research proposes a cheat detection system based on image recognition and supplemented by data detection and compared VGG16, ResNet50, MobileNet V2, Xception, and Inception V3 in an attempt to classify honest players and cheater aiming trajectories. The results of the research show that Inception V3 is the most accurate detector of honest player aiming trajectories.

Keywords: machine learning, cheat detection, InceptionV3, Aimbot, FPS

目 錄

附表目錄

附圖目錄

第一章　緒論

1.1 研究動機

電子遊戲隨著科技的進步蓬勃發展，從早期在大型機台投幣遊玩 ，轉變成各式家用主機相互競爭，再到個人電腦逐漸普及，玩家只需要一台個人電腦就可遊玩多樣的遊戲，在眾多的遊戲類型中，可以讓多人同時在相同的環境中一同遊玩的線上遊戲成為多數玩家最受歡迎的遊戲類型，一些玩家為了在線上遊戲中獲得更好的成就，而開始使用外掛程式去達成個人無法達成的目標，像是在射擊遊戲和 Multiplayer Online Battle Arena（以下簡稱 MOBA）獲得更好的對戰表現、在 Massively Multiplayer Online Role-Playing Game（以下簡稱 MMORPG）獲得難以取得的裝備與大量的製作材料等等。

在各類遊戲之中，射擊遊戲、MOBA類型遊戲、卡牌遊戲等因為其遊戲模式，可以給觀眾帶來優良的觀賞體驗而成為現在電子競技最常舉辦比賽的遊戲類型，在各類競技遊戲中，又以射擊遊戲的外掛最為猖獗，原因在於遊戲整體設計不同導致外掛程式的實作難度有所不同，以 MOBA 類遊戲來說，玩家對角色的操作都會上傳至伺服器進行運算後再將運算結果回傳至玩家的電腦中，但以射擊遊戲來說，需要計算子彈軌跡、槍枝後座力、玩家移動等大量的數據，同時也對動畫幀數與網路延遲的要求極高，在這些前提下，如果將玩家對角色的每個操作上傳至伺服器進行運算，以現今的技術水平無法保證射擊遊戲的流暢運作，所以會直接使用玩家個人的電腦進行本地運算，將這些數據運算整合後將關鍵數據上傳至伺服器[1]，這樣的運行方式給予外掛程式可以直接修改這些本地運算的數據的能力，這也使得在射

擊遊戲中作弊的門檻非常低，總結來說，MOBA遊戲是由伺服器告訴玩家電腦，玩家在做的動作，射擊遊戲則是完全相反，由玩家電腦告訴伺服器，玩家在做的動作。

射擊遊戲涉及到玩家的反應速度、意識以及準度，可說是所有遊戲中最能體現玩家個人技術能力的類型，技術強弱的差距會讓人們難以分辨對手是否使用外掛程式，因此對於遊戲開發商來說，外掛程式的偵測可說是最重要的目標，為了達成這個目的，遊戲開發商一般會使用兩種方式防治外掛程式：

1. 安裝遊戲時，將遊戲主程式與保護程式同時以封包傳輸至玩家電腦中，保護程式會在遊戲開啟時啟動，防止外掛程式對遊戲主程式進行駭入。
2. 選擇在遊戲中達到一定排名水準且並沒有任何不良紀錄的玩家作為審查員，審查被其他玩家檢舉或是被數據檢測程式發現數據有作弊疑慮的玩家數據。

1.2 玩家視角差別

在常見的射擊遊戲中，畫面的呈現方式有兩種：

1. 第一人稱視角：玩家所控制的遊戲角色不會出現在畫面中，玩家所看到的畫面是由遊戲角色能看到的景象形成，所以玩家也可以看到角色的雙手與拿著的物品等，特點：代入感強，可看到場景的細節描繪。
2. 第三人稱視角：玩家會以旁觀者的視角觀看場景與角色動作，第三人稱視角又以呈現方式分為上帝視角、過肩視角，上帝視角：玩家從正上方俯視場地，可獲得大量資訊用於排兵布陣，故常用於戰略遊戲的呈現；過肩視角：玩家的畫面一般會以操作角色後方一定距離的地方觀看，玩家可看到

操作角色的全身，也可觀察障礙物後的部分場景，特點：寬廣的視野帶來大量額外的資訊。

在本研究中，我們選擇以第一人稱視角設計遊戲，我們想讓玩家專注於視野中可看到的東西，避免過多的資訊分散玩家注意力。

1.3 射擊遊戲常見外掛類型

射擊遊戲的外掛可分為讀取本地端運算資料或是修改運算資料進而影響回傳至伺服器的資料，例如：無限彈藥、無後座力、加速瞬移、飛天遁地與我們的檢測目標透視與自動瞄準等等，都是透過讀取或修改於本地端運行的數據所實現，以下會主要介紹本研究的研究目標，透視與自動瞄準外掛：

1. 透視：藉由修改參數或是讀取遊戲數據，將不該出現在畫面中的敵方角色繪製在畫面中，常見有以下兩種：

   i. 人物透視外掛[2]：渲染類透視外掛，遊戲使用 Direct3D （Microsoft 在 Winodws 操作系統上開發的3D 繪圖程式，以下簡稱 D3D）渲染物體，D3D 提供多種渲染狀態，此種外掛利用其原理，通過修改相應的遊戲文件參數實現，遊戲中啟用時，玩家可直接在畫面中看到其他敵人與玩家操作角色的相對位置。

   ii. 方框透視外掛[3]：標記類外掛，藉由讀取遊戲數據，並透過計算獲取敵人位置資訊，並以方框的形式在玩家畫面中標示其位置。

本研究為我們自行開發遊戲，所以我們選擇以修改人物的渲染方式實作玩家使用的透視外掛系統。

2. 自動瞄準（Aimbot）[4]：藉由讀取在電腦進行本地運算的資料，確定敵人的位置後，直接控制玩家電腦進行視野的轉動，將瞄準的準心直接固定到敵人身上，讓使用者不用自行搜索敵人及瞄準，自動瞄準外掛又可以其自動化的程度分為下列四種[5]：

   i. OnPressed：當玩家按下射擊鍵，才會開始進行瞄準。

   ii. FollowTarget：當敵人進入玩家視野，會自動進行瞄準。

   iii. AutoAtk：自動對視野內的敵人進行瞄準與射擊。

   iv. FullyAuto：除了自動瞄準射擊以外，還會控制玩家角色移動。

本研究考慮到每個玩家的反應時間有所不同，所以選擇在遊戲中以 OnPressed 的模型實作自動瞄準外掛，進而增加資料的多樣性。

1.4 研究目標

在這個研究中，我們針對第一人稱射擊遊戲進行實驗環境的建構，並且選取人物透視外掛與自動瞄準兩種外掛進行針對性的研究，原因有以下幾點：

1. 第三人稱視角會給玩家帶來過多額外的資訊，我們希望玩家專注於藉由個人觀察發現敵人的動向，並且也可讓玩家更貼近於現實中士兵在戰場搜索敵人、擊敗敵人。

2. 人物透視外掛與自動瞄準兩種外掛都是建立在讀取遊戲文件，獲取敵人位置資料並在畫面中繪製或是將瞄準準心修改到可與敵人形成直線的位置。

我們的研究目標是製作出可用於進行檢測玩家數據，並分辨是否使用外掛程式

進行遊戲的數據檢測程式，並將其運用於市面上的第一人稱射擊遊戲。

1.5 研究貢獻

在本研究中我們做出了以下三點貢獻：

1. 開發一款第一人稱射擊遊戲，內容包含隨機地圖、有團隊合作能力的 AI、於遊戲中實作的透視與自動瞄準外掛等等。
2. 提出一款以 machine learning 的方法進行影像辨識為主，以數據檢測程式為輔的外掛檢測系統，此系統經測試可分辨外掛使用者與誠實玩家。
3. 以 accuracy、F1-score、AUC（area under curve）與 train time 作為評分標準測試了五種用於影像辨識的模型，Inception V3在測試中表現最佳。

第二章　背景知識

在這一章節會介紹用於瞄準軌跡影響辨識的各種模型，包含 VGGNet、Inception、ResNet、MobileNet 和 Xception。

2.1 VGGNet[6]

2014年，牛津大學計算機視覺組（Visual Geometry Group）和 Google DeepMind 一起研發出新的深度卷積神經網路—VGGNet（圖1），VGGNet 有以下特點：

1. 結構簡潔：由5層卷積層、3層全連接層、softmax 輸出層構成，層與層之間以 max-pooling 分開，並使用 ReLU 作為 activation function。
2. 小卷積核和多卷積子層：卷積層以多個小卷積核的卷積層替代一個卷積核較大的卷積層，用以減少參數，並藉由更多的非線性映射，增加網路的擬合能力。
3. 小池化層：採用2*2的池化核。
4. 通道數多：VGG 網路第一層的通道為64，後面每層都進行了翻倍，最多到512個通道，通道數增加，可提取更多的信息。
5. 層數更深、特徵圖更寬：卷積核專注於擴大通道數、池化核專注於縮小寬和高，在模型架構更深更寬的同時，控制了計算量的規模增加。

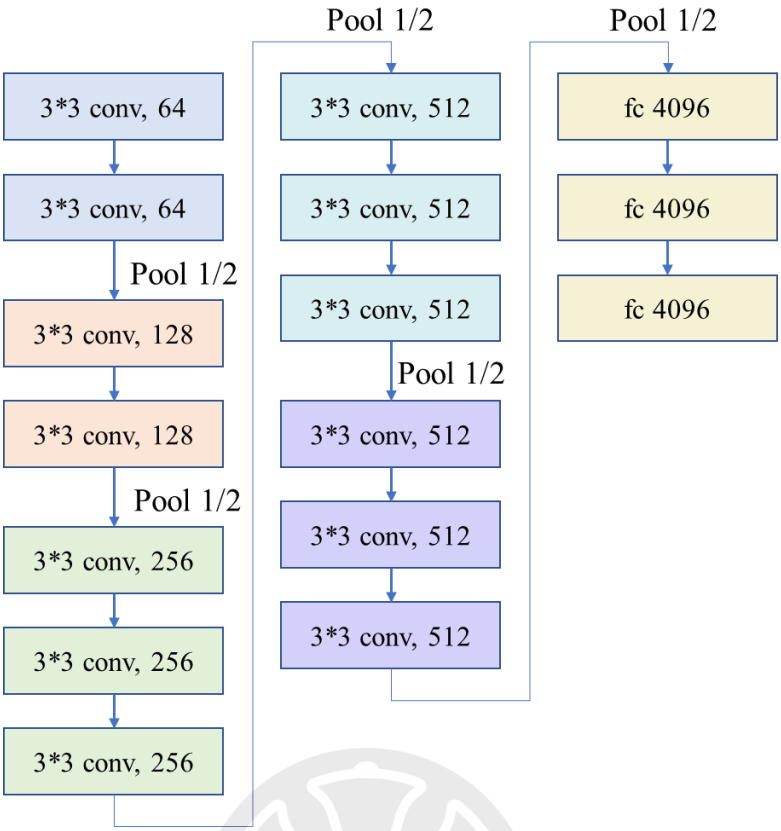


圖 1 VGGNet 結構圖

2.2 Inception [7] [8] [9]

GoogLeNet 在2014ILSVRC 分類大賽中使用 Inception 獲得良好成績，以下會介紹 Inception 各版本的不同：

1. Inception V1[7]：2014年提出，結構如圖2，藉由將1*1的卷積層與3*3、5*5卷積層和3*3的池化層堆疊，增加了網路的寬度， 其中選用1*1的卷積層去減少資料維度，因此增加了網路對輸入資料尺度的適應性，並使用線性整流函式（Rectified Linear Unit，簡稱 ReLU）作為 activation function。

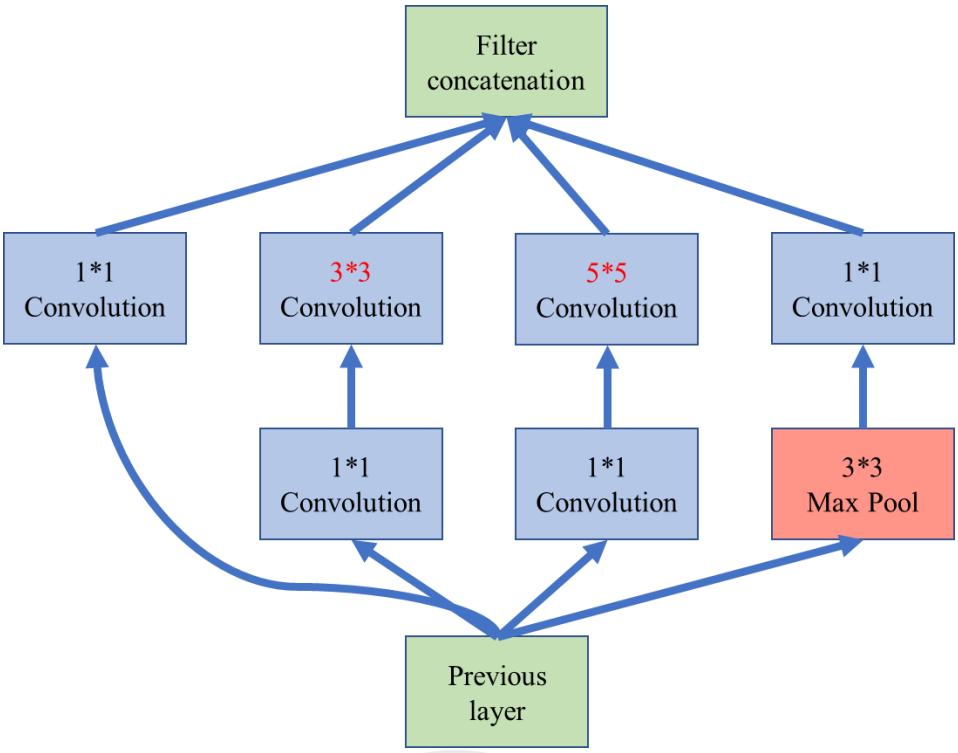


圖 2 Inception V1 model

以1*1卷積進行資料降維

2. Inception V2[8]：2015年2月提出，基於 Inception V1做了兩項改進，形成下圖3的結構：

   i. 引入 Batch Normalization，將每個 mini-batch 進行正規化到平均值0、標準差1的常態分佈，有助於減緩梯度消失與解決 Internal Covariate Shift 的問題，同時加速收斂並有正則化（regularization）的效果。

   ii. 使用兩個連續的3*3卷積層代替5*5的卷積層，如圖4，從圖中可看出兩個3*3的卷積可覆蓋的範圍是5*5，假設5*5與兩個3*3卷積可輸出相同數量的特徵值，那兩個3*3卷積的計算量是5*5的（3*3 + 3*3）/（5*5）=18/25。

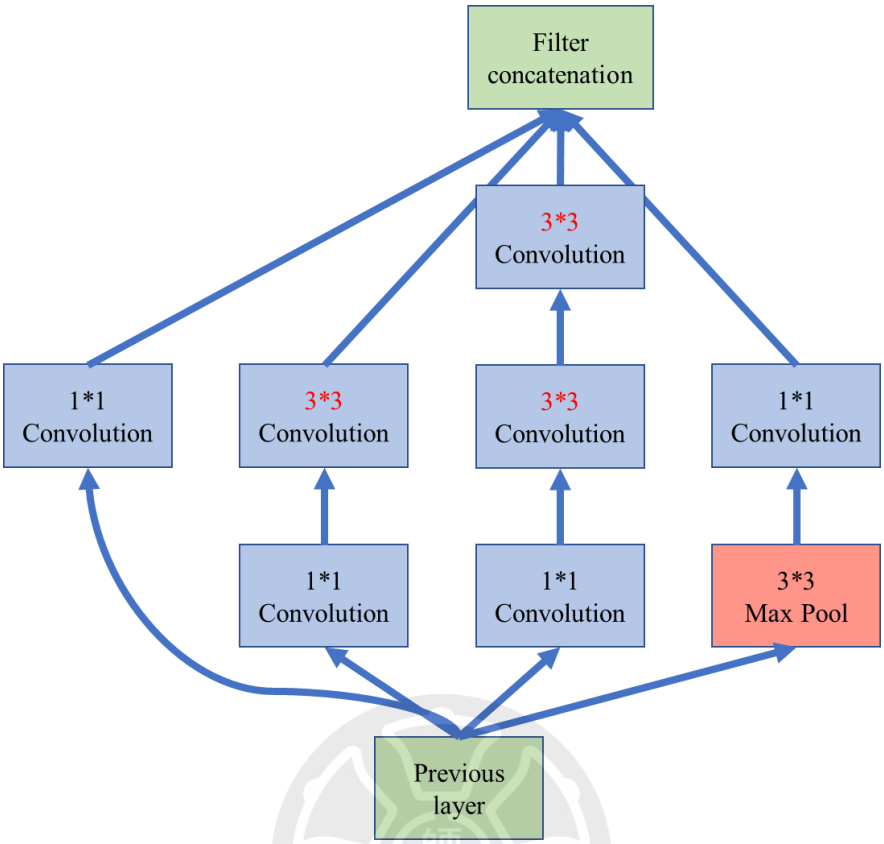


圖 3 Inception V2 model

（將5*5卷積分解為兩個連續的3*3卷積）

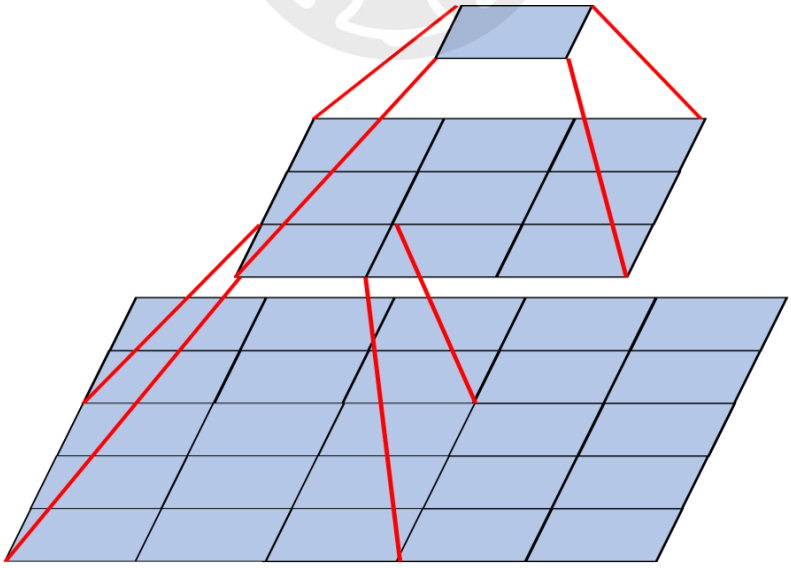


圖 4 卷積分解

（以兩個3*3卷積分解5*5卷積）

3. Inception V3[9]：2015年12月提出，基於四個網路設計的原則：

   i. 避免在前面層數的網路架構中使用 bottlenecks，因為 bottlenecks 雖然可以有效降低參數計算量，但是在過程中也會失去過多的特徵訊息，網路架構應避免維度過度壓縮，保持輸入到輸出特徵維度緩慢下降。

   ii. 高維度的特徵適合用於網路的局部處理，並在網路中增加非線性結構獲取更多特徵值，進而加速訓練過程。

   iii. 空間聚合（Spatial aggregation）可藉由低維度進行降維，且不會影響模型功能。

   iv. 增加網路的寬度與深度都能達到提升效能的結果，平衡兩者能獲得更好的效能。

基於上述4個網路設計原則與 Inception V2，Inception V3作了以下改進：

1. 將卷積分解為不對稱卷積：由 Inception V2得知將大卷積和拆解為多個3*3卷積和可減少計算量，在 Inception V3嘗試分解成不對稱的卷積核並比較計算量，如下圖5，將3*3卷積層分解為1*3及3*1卷積層，（1*3+3*1）/（3*3）=2/3，減少了33%的計算量，若分解為兩個2*2的卷積層，（2*2+2*2）/（3*3）=8/9，只能減少11%的計算量。

   不對稱卷積在實驗中驗證出適合用於尺寸範圍在12~20的特徵圖，並且不適合用於前面層的卷積，Inception V3在17*17特徵圖使用了1*7及7*1的卷積，降低了參數量並增加模型深度，如圖6。

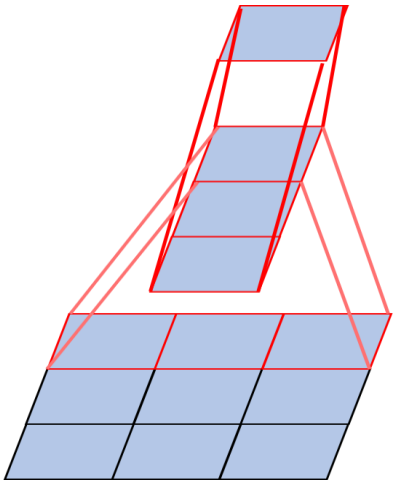


圖 5 不對稱卷積

（以連續的1*3 3*1卷積替代3*3卷積）

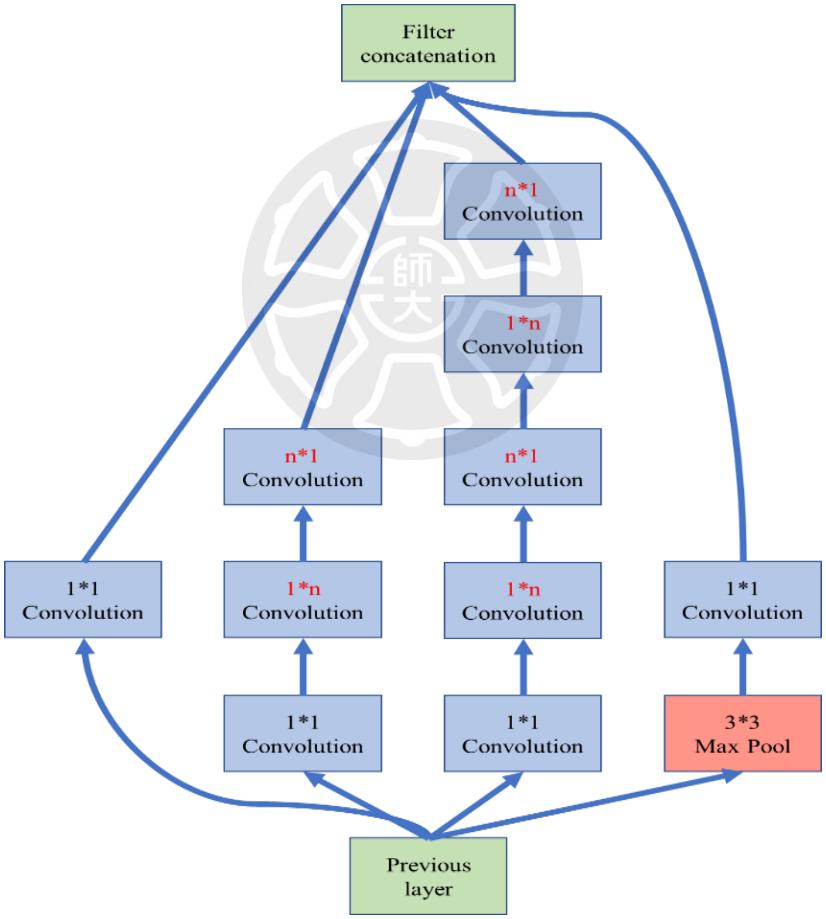


圖 6 Inception model after the factorization

（n = 7用於17*17卷積）

根據網路設計原則—高維度的特徵更容易處理並有利於訓練，因此在8*8卷積層採用可增加網路寬度的結構，以產生高維度特徵，如圖7。

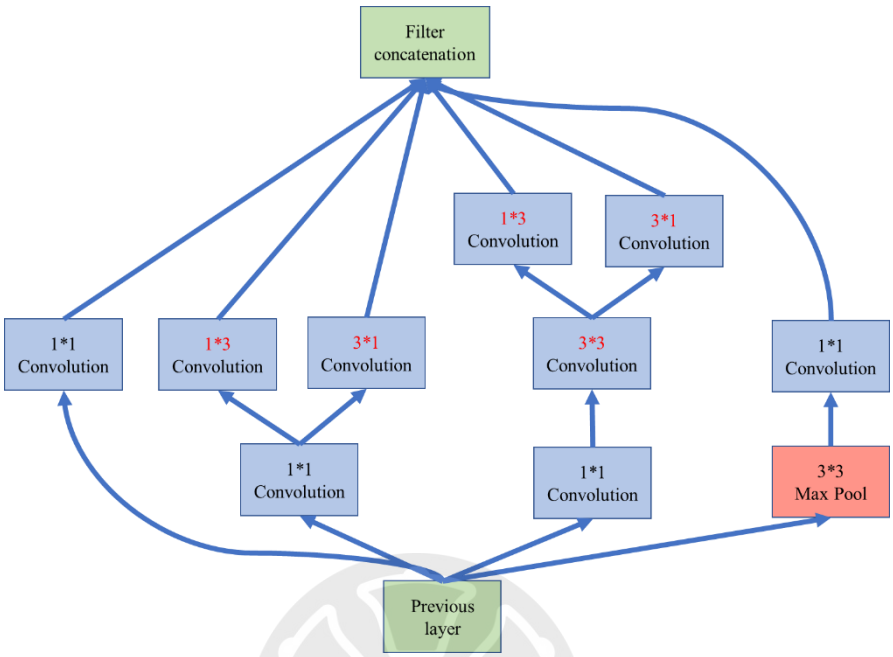


圖 7 Inception model

（以並聯的1*3 3*1卷積增加網路寬度）

2. 輔助分類器的作用：GoogLeNet 在訓練的早期與後期分別設置了輔助分類器，當拿掉較低層的輔助分類器並沒有對模型結果有負面影響，推測出輔助分類器有助於較低特徵的演變是不正確的，並且輔助分類器若具有 Batch Normalization 或是 Dropout 層，可使模型效能變好，輔助分類器應有正則化的作用。

3. 縮小特徵圖：常見的縮小特徵圖有兩種方法：

   i. 先進行池化減少特徵數目進行採樣，在進行 Inception，如圖8a，這種做法會違背網路設計原則—保持輸入到輸出特徵維度緩慢下降。

ii. 先進行 Inception，在通過池化層採樣，如圖8b，計算量是圖8a 的3倍。

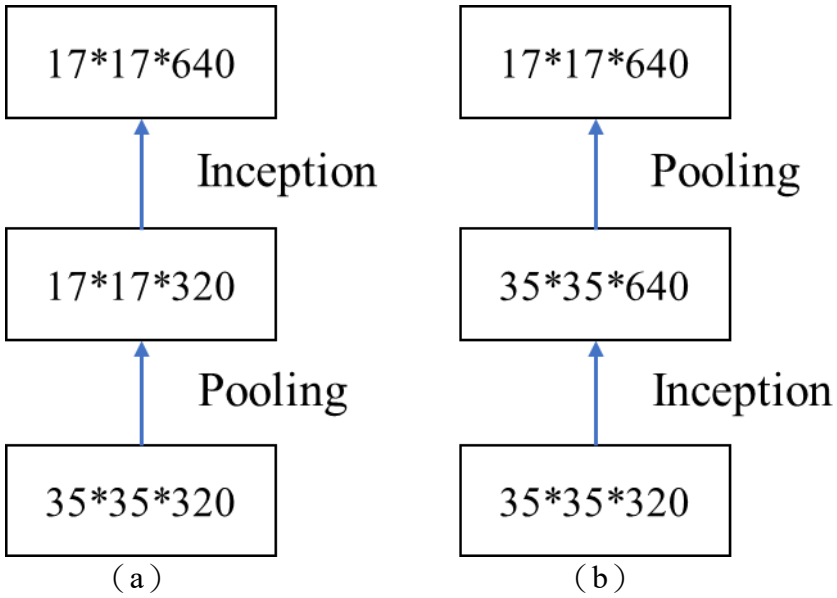


圖 8 常見縮小特徵圖方法

因此 Inception V3使用 stride=2的卷積層與池化層並行操作以縮小特徵圖，結構如圖9，在減少參數量的圖時，還能避免使用 bottlenecks。

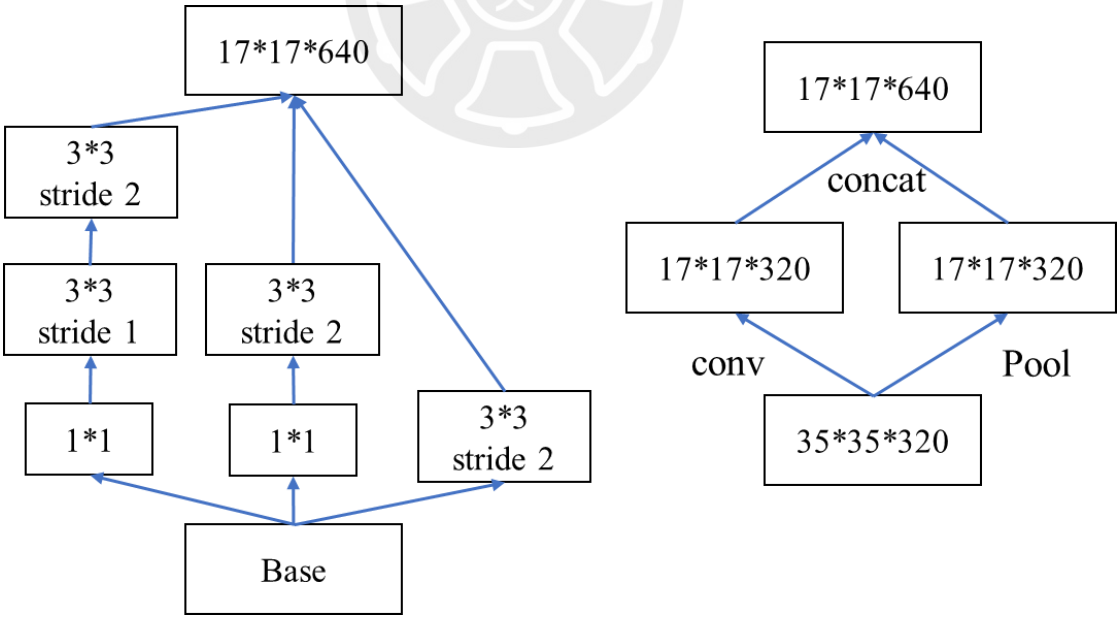


圖 9 Inception model

（圖左為網路架構 圖右為網路大小）

2.3 ResNet[10]

ResNet 全名 Deep Residual Neural Network，在 VGG 的架構上，提出了殘差映射（Residual Mapping），這種架構不會新增參數，但可以解決梯度消失的問題，圖10為一個基本的 Residual Block，假設模型的輸入為 x，模型所學習到的特徵為 H（x），定義 Residual function 為 F（x） = H（x） - x，進而將學習特徵寫為 F（x） + x。

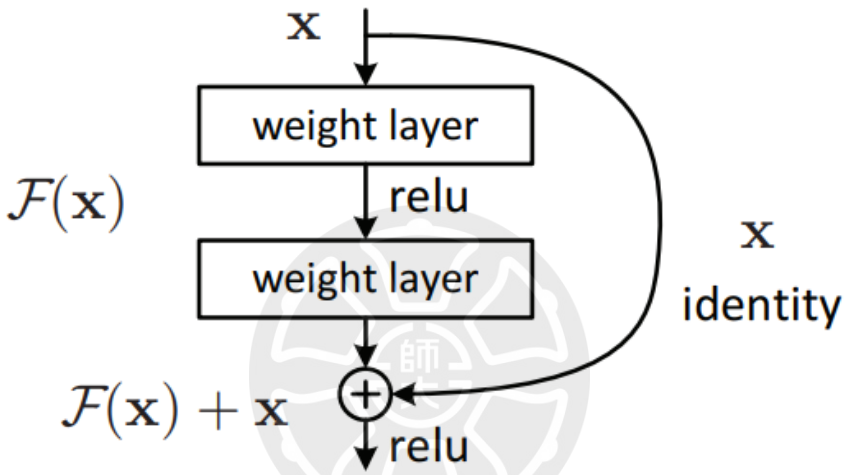
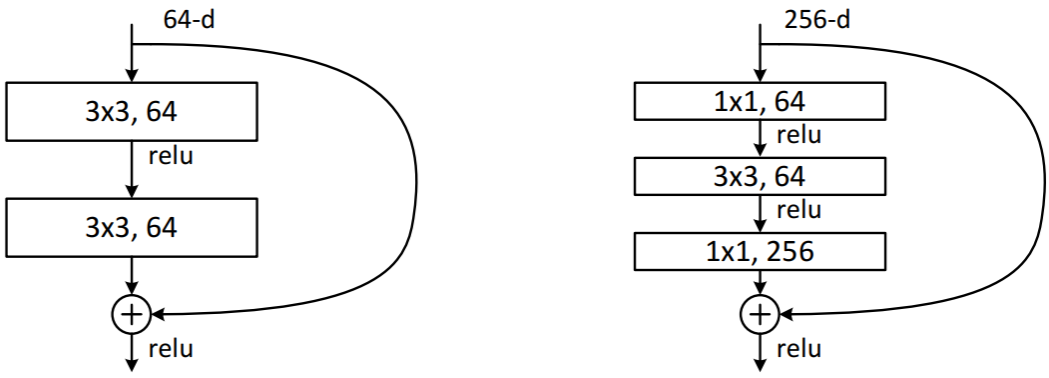


圖 10 Resodual block

因為顧及到訓練深度與訓練時間，在大於50層的網路中，提出了圖11b 的 bottleneck 的設計，用於節省大量的參數，此架構使用了1*1的卷積進行了降維後再升維的過程，在不增加時間複雜度的狀況下，同時兼顧了相較於左邊架構更高的維度與效能，從圖12可看出，在 ResNet50的寬度與深度增大的狀況下，ResNet34與ResNet50運算量仍相差不遠。



（a）　　（b）

圖 11 deep residual function

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		[3×3, 64; 3×3, 64] ×2	[3×3, 64; 3×3, 64] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3
conv3_x	28×28	[3×3, 128; 3×3, 128] ×2	[3×3, 128; 3×3, 128] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×8
conv4_x	14×14	[3×3, 256; 3×3, 256] ×2	[3×3, 256; 3×3, 256] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×23	[1×1, 256; 3×3, 256; 1×1, 1024] ×36
conv5_x	7×7	[3×3, 512; 3×3, 512] ×2	[3×3, 512; 3×3, 512] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

圖 12 ResNet 網路架構（ResNet34與 ResNet50計算量相差不多）

2.4 Xception[11]

2016年提出，Xception 取自 Extreme Inception，在 Inception V3的基礎上，採用了 depthwise separable convolution 取代了原本 Inception V3的卷積，並加入 ResNet 的 Residual Learning，整體結構見圖13，由多個 module 組成，總共分為3個 flow—

Enity、Middle 和 Exit，Middle flow 會重複8次，從圖中可看出其架構與傳統的 depthwise separable convolution 不同，depthwise separable convolution 會先將通道進行卷積後再進行1*1卷積運算，兩個卷積之間不使用激活函數，Xception 則是先進行1*1卷積運算，再將通道卷積，並在其中使用 ReLU 連接。

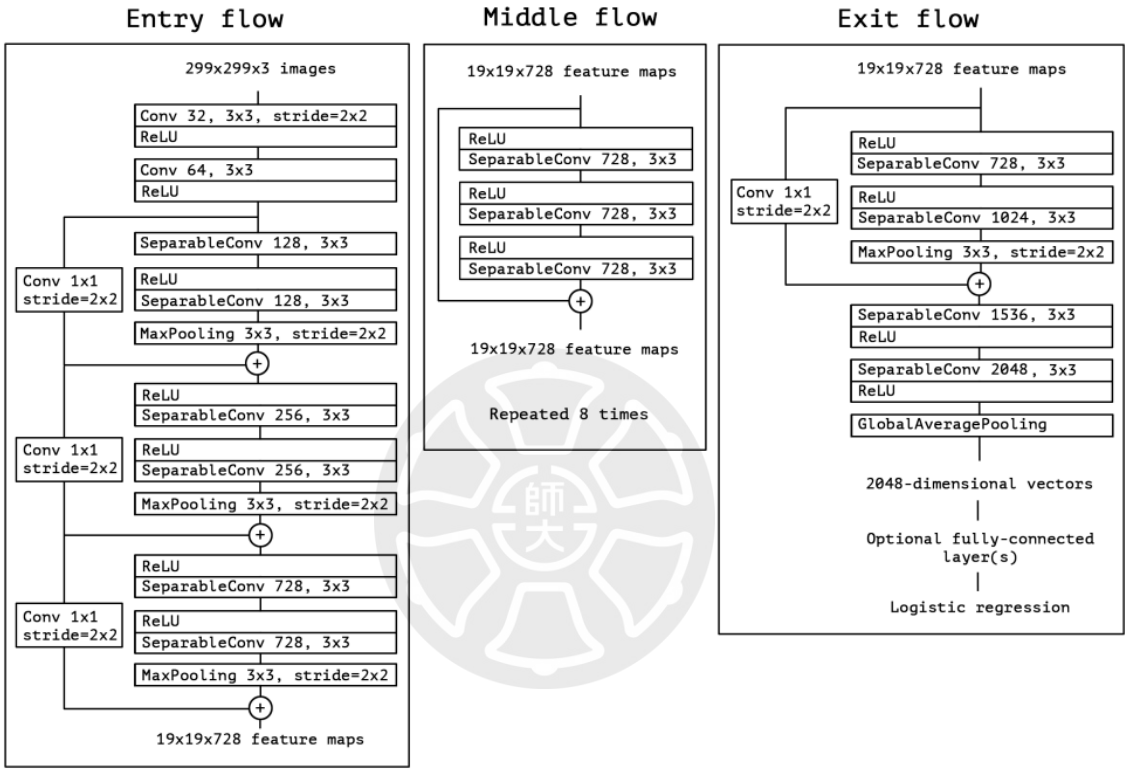


圖 13 Xception 結構

SeparableConv 為 depthwise separable convolution，加號為 Residual Learning

2.5 MobileNet [12] [13]

為了在行動裝置中運行深度學習網路模型，Google 於2017年提出了 MobileNet V1，此模型可在保證準確率的狀況下，將模型的體積縮小、速度加快，以下會介紹 MobileNet 的各版本的不同：

1. MobileNet V1[12]：2017年於 Xception 提出，透過2個超參數（Hyper parameters）：寬度因子（Width multiplier）和解析度因子（Resolution multiplier）控制模型的整體架構與參數數量，並採用深度可分離卷積（Depthwise Separable Convolution）取代傳統的卷積。

i. 深度可分離卷積：由深度卷積（depthwise convolution）跟逐點卷積（pointwise convolution）組成，假設傳統的卷積輸入輸出如圖14，深度卷積則會針對每個輸入的通道分別用一個 k*k 的 kernel 進行卷積，如下圖15，逐點卷積會將每個通道做完深度卷積的輸出資料連接一個1*1的卷積層，分別計算後形成一個 kernel map，以上過程會進行 N_k 次，獲得尺寸為 $W_{out}*H_{out}*N_k$ 的 kernel map，過程如圖16，對比傳統卷積與深度可分離卷積的參數數量與計算量，傳統卷積參數數量為 $k*k*N_{ch}*N_k$，計算量為 $W_{in}*H_{in}*N_{ch}*k*k*N_k$，深度可分離卷積的參數數量為 $k*k*N_k + N_{ch}*N_k$，計算量為 $W_{in}*H_{in}*N_{ch}*k*k+W_{in}*H_{in}*N_{ch}*N_k$，參數數量比為:

$$\frac{(k * k * N_{ch} + N_{ch} * N_k)}{(k * k * N_{ch} * N_k)} = \frac{1}{N_k} + \frac{1}{k * k}$$

計算量比為:

$$\frac{(W_{in} * H_{in} * N_{ch} * k * k + W_{in} * H_{in} * N_{ch} * N_k)}{(W_{in} * H_{in} * N_{ch} * k * k * N_k)} = \frac{1}{N_k} + \frac{1}{k * k}$$

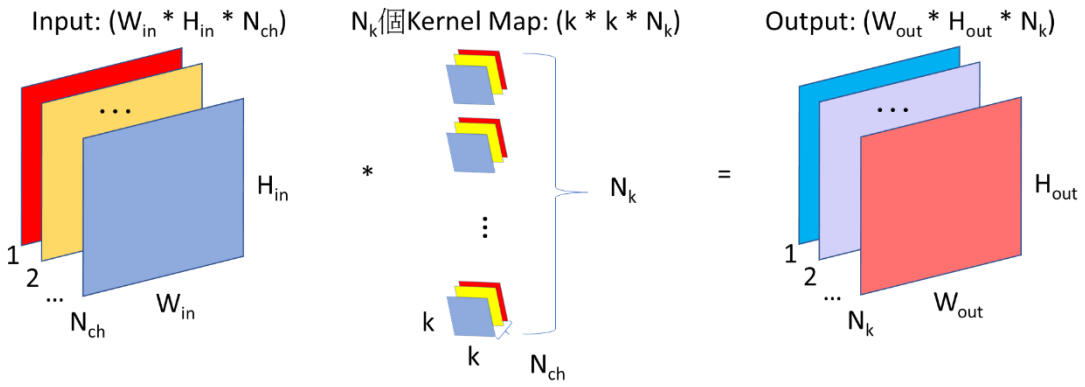


圖 14 Convolution 架構

ii. Width multiplier：以 α 表示，α ∈（0, 1]，用於縮減輸入輸出的通道，輸入通道由 N_{ch} 到 $\alpha*N_{ch}$，輸出通道由 N_k 到 $\alpha*N_k$，縮減後計算量為 $W_{in}*H_{in}*\alpha N_{ch}*k*k + W_{in}*H_{in}*\alpha N_{ch}*\alpha N_k$。

iii. Resolution multiplier：以 ρ 表示，ρ∈（0, 1]，用於控制 feature map 縮減比例，同時使用 Width multiplier 和 Resolution multiplier，可將計算量縮減為 ρWin*ρHin*αNch*k*k + ρWin*ρHin*αNch*αNk。

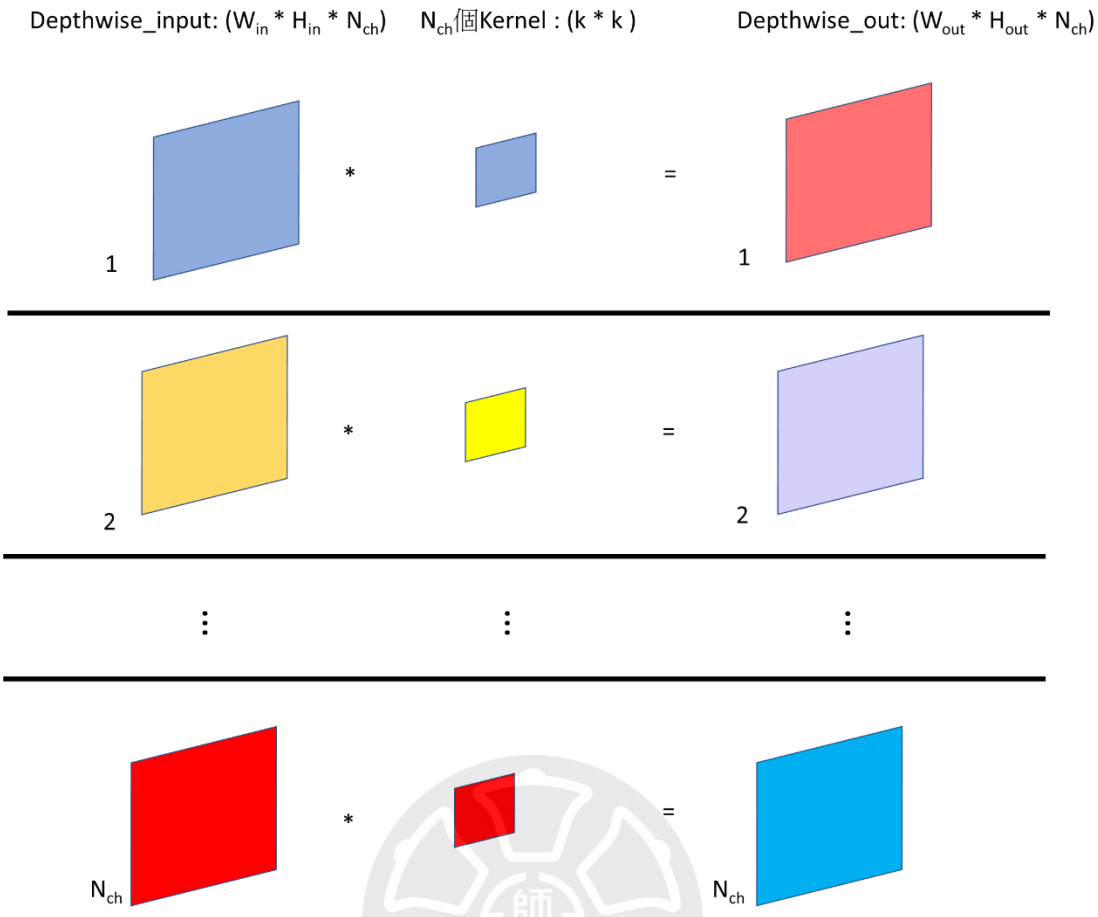


圖 15 DepthWise Convolution 架構

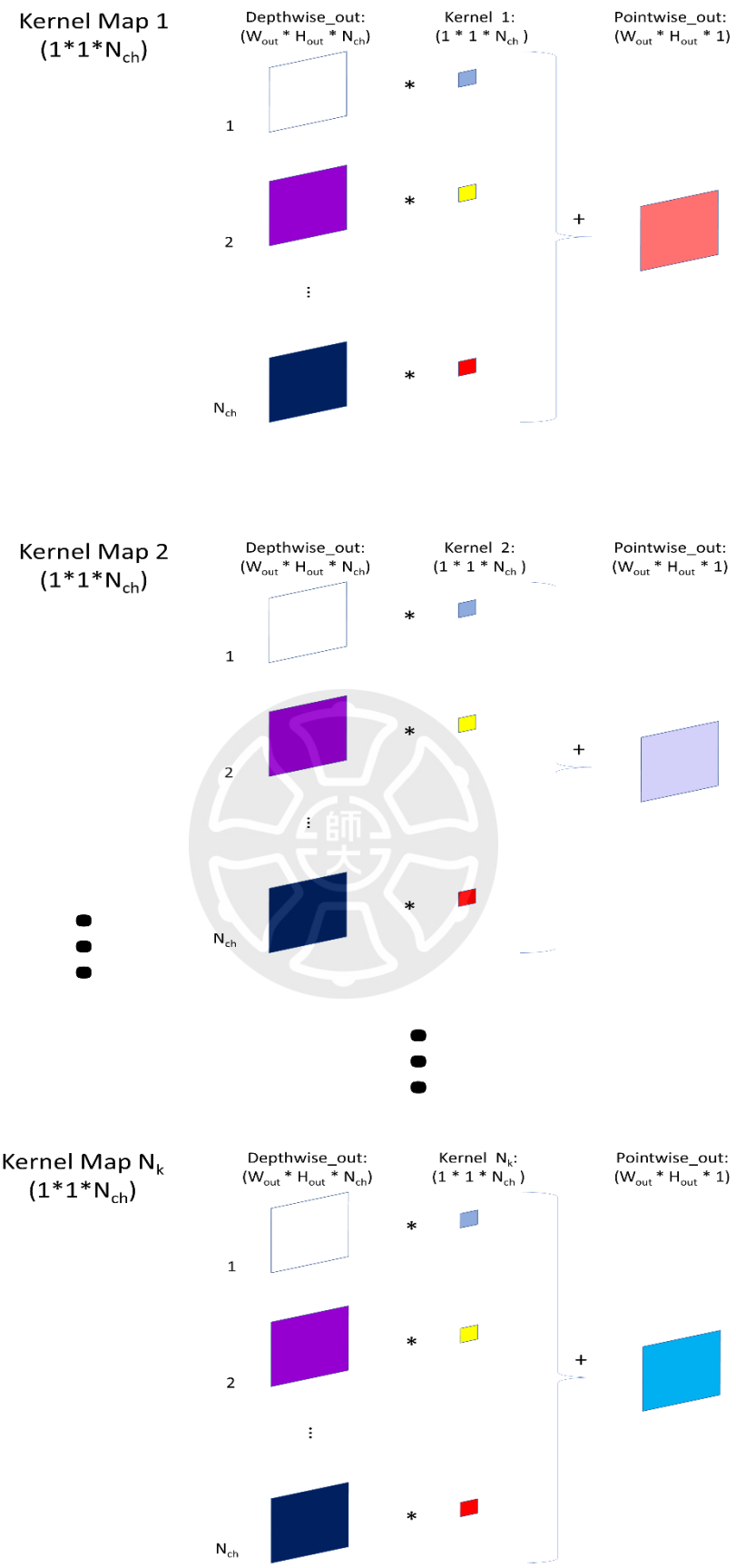


圖 16 PointWise Convolution 架構

2. MobileNet V2[13]：2018年提出，基於 MobileNet V1的架構，並加入了 Inverted Residuals 和 Linear Bottlenecks。

   i. Inverted Residuals：原本的 Resudual 是使用 pointwise 進行壓縮，再利用3*3卷積進行特徵提取，最後使用 pointwise 卷積進行通道擴張，整個過程就是壓縮卷積擴張，Inverted Residual 則是使用 pointwise 進行通道擴張，再使用3*3 depthwise，最後使用 pointwise 壓縮，過程變成擴張卷積壓縮，因為在過程中將通道數提升，特徵值的維度提升，不會像原本的架構下降過快。

   ii. Linear Bottlenecks：MobileNet V1中採用 ReLU 作為 activation function，ReLU 在對特徵進行轉換時，輸出是低維度的特徵會導致大量訊息流失，圖17為相同輸入經由 ReLU 轉換至不同維度輸出的結果，在低維度中，特徵信息流失嚴重，在高維度中可保存較多的特徵信息。

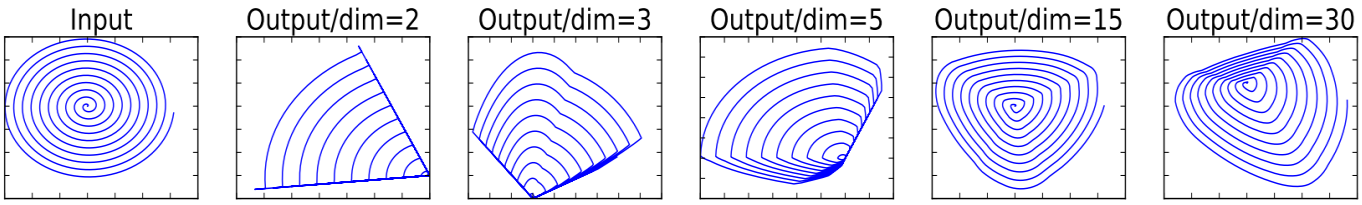


圖 17 ReLU 轉換結果

（低維度轉換後特徵消失明顯 高維度保留較多特徵）

在本研究中，我們選取了五種在影像辨識中在比賽中表現良好或是有特色的模型，帶動了神經網路越深越好的理念的 VGGNet，用多種不同大小的卷積層讀取特徵的 Inception，藉由引入 Residual 解決因為網路的深度增加而導致梯度下降過多的狀況的 ResNet，在 Inception 的架構上加入了 depthwise separable convolution 與 Residual 的 Xception，以與 Xception 不同的 depthwise separable convolution 減少參數量進而讓訓練時間減少的 MobileNet。

第三章　文獻回顧

為了防止外掛對遊戲環境造成破壞而使玩家的遊戲體驗變差，遊戲開發商引入了兩套系統用於找出外掛使用者，一種是在安裝遊戲時，在封包中加入保護程式後傳輸到玩家電腦中，保護程式會在遊戲開啟時啟動，防止外掛程式對遊戲主程式進行駭入，常見的有 Ubisoft 的 BattleEye、Epic Games 的 EasyAnti-Cheat （以下簡稱 EAC）、Value 的 Value Anti-Cheat（以下簡稱 VAC）等等；另一種是選擇在遊戲中達到一定排名水準且並沒有任何不良紀錄的玩家作為審查員，審查被其他玩家檢舉或是被數據檢測程式發現數據有作弊疑慮的玩家數據。

S. Park 等人提出了 BlackMirror[14]，其中採用了 Intel 的 SGX（Software Guard Extensions）技術，藉由這種以硬體為基礎的記憶體加密技術，將重要資料進行加密防止外掛程式進行存取。

K.-T. Chen 等人在 Quake II 中通過分析角色行動軌跡進行檢測[15]，因為玩家不會採取極為相似或相同的路徑在地圖中移動，而機器人會這樣做，作者使用 Isomap、k-NN 演算法與 SVM 進行角色行動軌跡的分析，在論文中並未針對使用外掛程式進行輔助的玩家群體。

L. Galli 等人提出了一個由收集數據的遊戲伺服器、進行預先處理數據與檢測作弊行為的後端程式與讓使用者進行遊戲監控的前端介面組成的外掛檢測系統[16]，其中使用了 Supervised learning 並將其與 decision trees、Naive Bayes、random forest、neural networks、support vector machines 等五種技術形成新的系統，此種系統因為需要處理多種數據無法在短時間完成外掛檢測。

M. Willman 使用了神經網路在 Counter-Strike Global Offensive 進行自動瞄準外掛的識別[17]，但因為訓練數據數量不夠，只獲得了50%的準確度，未達成原本預估的90%準確度。

S. Najari 等人提出了 GAN-Aimbots[18]，其中採用了 Generative Adversarial Networks 模擬玩家滑鼠移動，進而產生可以假亂真的滑鼠移動方式，並將其用於外掛程式的製作。

H. Alayed 等人專注於研究儲存了玩家行為資料的遊戲日誌[19]，從日誌中提取資料做為特徵並採用了 reinforcement learning 進行訓練，R. Spijkerman 等人使用了 decision tree、SVM 和 Naïve Bayes 分析遊戲日誌[20]。

上述的各項論文包含了使用機器學習進行外掛程式的訓練、以新種的加密算法防止外掛駭入、針對機器人的檢測程式與針對遊戲日誌的機器學習程式，在本研究中，為了開發出一款可通用於第一人稱射擊遊戲的外掛檢測系統，我們採用了對於遊戲日誌的研究作為基礎，並提出了以瞄準軌跡影像辨識為主，以遊戲日誌數據檢測為輔的檢測系統。

第四章　實驗環境

本論文中提出一套外掛檢測系統，此系統分為兩部分，第一部分以分析玩家數據以找出可能為外掛使用者的玩家，第二部分則是針對瞄準軌跡進行影像辨識進而分辨誠實玩家與外掛使用者，為了檢驗這個系統，我們設計了一套第一人稱射擊遊戲，並在遊戲中加入了透視外掛與自動瞄準外掛以蒐集誠實玩家與外掛使用者的數據。

4.1 概述

我們的檢測系統專注於辨識第一人稱射擊遊戲的透視與自動瞄準的外掛使用者，所以我們設計了一款第一人稱射擊遊戲作為實驗環境，其中包含了會團隊合作的敵人與可使用的透視與自動瞄準的外掛程式，玩家的角色會被放置於隨機地圖中，玩家須在三分鐘內盡可能地找出地圖上的敵人並擊倒。

玩家會在四種設定下進行遊戲：未開啟作弊程式、透視程式開啟、自動瞄準程式開啟和同時開啟兩種作弊程式，其中自動瞄準的視角移動速度設定為每秒10、20和30度，瞄準目標固定於玩家身體而不是頭部，我們會收集玩家瞄準與射擊的數據，並用於訓練檢測系統。

4.2 遊戲環境建構

為了獲得我們所需的遊戲資料，我們使用 unreal engine 4.25.4[20]開發遊戲環境，玩家角色會被放到由圖18 a~d 隨機排列產生的10*10的地圖中（圖19），除了玩家出生點外，其餘地圖板塊都會產生一名敵人，共99名，玩家會分別在8種狀況下進行遊戲，目標是在3分鐘內盡可能得搜索敵人並擊敗敵人，8種狀況如表1。

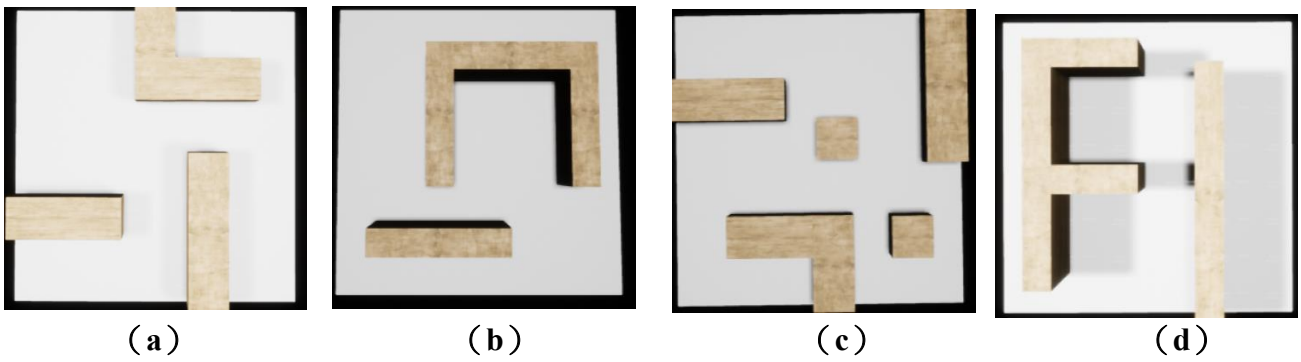


圖 18 地圖板塊

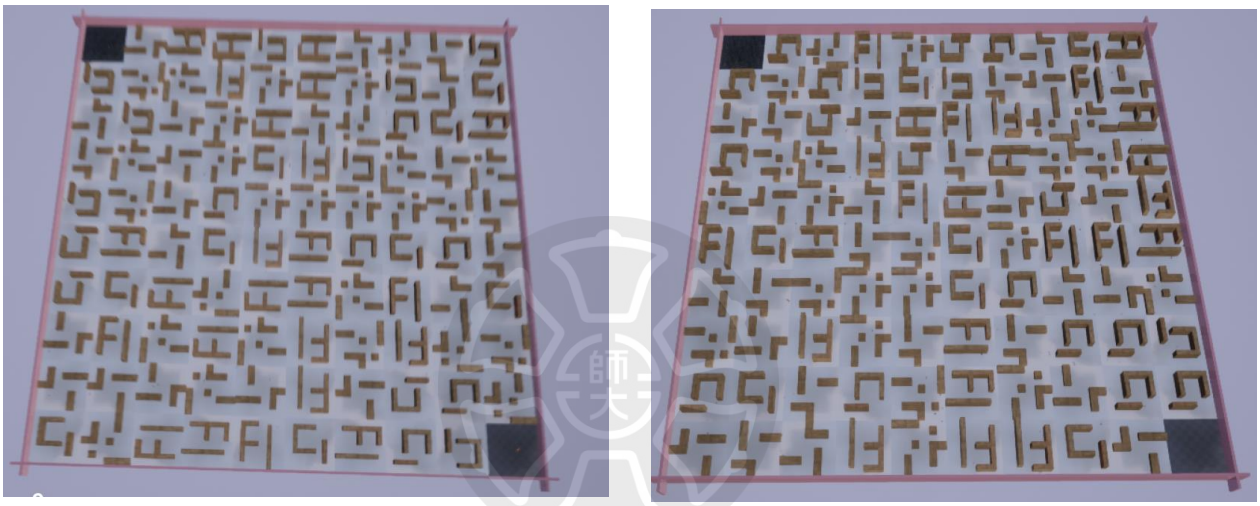


圖 19 隨機地圖

表 1 實驗狀況設定

i	沒有開啟外掛。
ii	開啟透視外掛。
iii	開啟自動瞄準外掛，鏡頭移動速度慢。
iv	開啟自動瞄準外掛，鏡頭移動速度中。
v	開啟自動瞄準外掛，鏡頭移動速度快。
vi	開啟透視外掛與自動瞄準外掛，鏡頭移動速度慢。
vii	開啟透視外掛與自動瞄準外掛，鏡頭移動速度中。
viii	開啟透視外掛與自動瞄準外掛，鏡頭移動速度快。

4.3 AI 行為樹建構

為了完整模擬第一人稱射擊遊戲的對戰，除了隨機地圖外，還需要可作為敵人的 AI 角色，我們參考了於2019 IEEE Conference of Games 發布的 Desirable Behaviors for Companion Bots in First-Person Shooters 所提出的 Jude AI 設計[21]，此種設計的實驗結果表明，在總體評分中的團隊合作度拿到較好的成績，能讓參與實驗的玩家有較好的遊戲體驗，並設計出我們的 AI 的整體行為具體設計如表2。

表 2 AI 行為樹

NO.	當前階段	下一階段	狀況
1	巡邏	往敵人移動	收到同伴呼叫
2	巡邏	射擊	看見敵人
3	往敵人移動	射擊	看見敵人
4	射擊	往敵人移動	看不見敵人
5	射擊	搜尋掩體並呼叫隊友	角色血量低於50%
6	搜尋掩體並呼叫隊友	於掩體後射擊	躲於掩體後
7	於掩體後射擊	搜尋掩體並呼叫隊友	玩家繞道掩體後
8	任意階段	死亡	角色血量為0

4.4 透視與自動瞄準外掛實作

我們於遊戲中加入了於第一章第三小節提到的人物透視外掛與自動瞄準外掛，人物透視外掛我們是以修改 AI 的渲染方式進行實作，下圖20為透視關閉的遊戲畫面，圖21則為透視開啟的遊戲畫面，自動瞄準外掛我們使用的是 OnPressed 模式，當玩家按下滑鼠右鍵進行縮放瞄準時，自動瞄準就會啟動，當視野中出現敵人時，會自動選取離玩家最近的敵人進行瞄準，圖22與23為開啟自動瞄準後的畫面移動。

圖 20 透視關閉遊戲畫面

圖 21 透視開啟遊戲畫面

圖 22 自動瞄準鏡頭移動前

圖 23 自動瞄準鏡頭移動後

4.5 玩家資料蒐集

在本研究中，將第一人稱射擊遊戲的玩家數據分為九個部分於表3中表示，蒐集資料如表4：

表 3 玩家數據格式

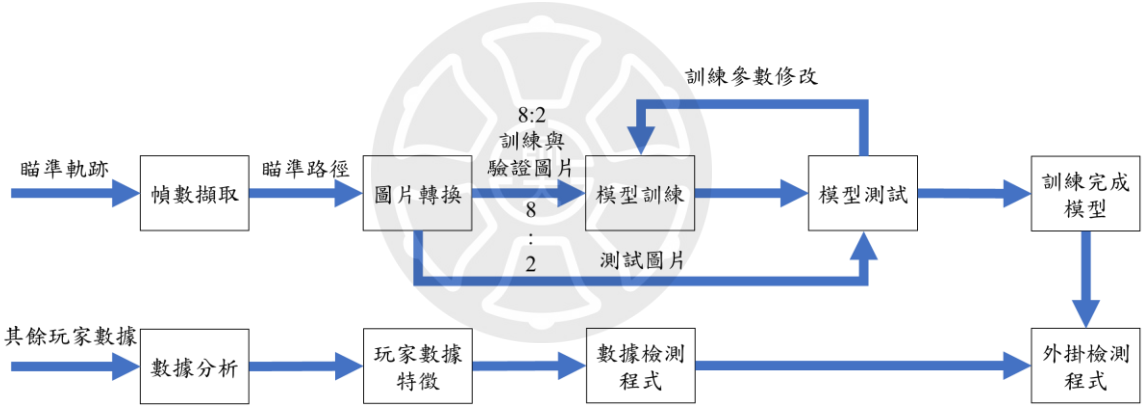
命中率	總命中數除以總擊發數。
擊中頭部率	命中頭部數除以總命中數。
擊敗敵人時間	視野中出現敵人到擊敗敵人所需時間。
擊敗敵人距離	擊敗敵人時玩家與敵人距離。
擊敗敵人到下次擊中時間	擊敗敵人且視野中仍有其他敵人，到玩家下次射擊命中所需時間。
反擊時間	玩家視野中沒有敵人，遭到敵人攻擊後到玩家射擊命中敵人所需時間。
首次射擊命中率	玩家視野中出現敵人，玩家的第一發射擊命中率。
首次射擊命中時間	玩家視野中出現敵人到玩家射擊命中敵人所需時間。
瞄準軌跡	視野中出現敵人後的視野鏡頭移動軌跡。

表 4 玩家數據

命中率	21.82%
擊中頭部率	18.99%
擊敗敵人時間（sec）	11.4425s
擊敗敵人距離	485.8689
擊敗敵人到下次擊中時間（sec）	2.3696s
反擊時間（sec）	44.0794s

首次射擊命中率	2.38%					
首次射擊命中時間	6.144097063					
瞄準軌跡	X 軸	(0.736	0.736	0.736	0.676	0.348
		0.379	0.409	0.417	0.426	0.415
		0.413	0.413	0.413	0.413	0.413)
	Y 軸	(-0.677	-0.677	-0.677	-0.737	-0.937
		-0.925	-0.913	-0.909	-0.905	-0.91
		-0.911	-0.911	-0.911	-0.911	-0.911)

4.6 實驗運作方式



4.7 檢測方法

檢測方法分為兩部份，第一部分以在第一章第四節中提過的對玩家的遊戲日誌進行分析，我們會使用在前一小節提出的玩家數據，其中瞄準軌跡會被用於檢測系統，其餘的八個數據會以誠實玩家的數據為基準，將作弊使用者的數據與其進行比較找出可用於檢驗誠實玩家與作弊使用者的特徵，第二部分則是檢驗我們提出的檢測系統，這個檢測系統會擷取玩家瞄準敵人的畫面移動軌跡，將資料轉換成圖片後，

以影像辨識模型進行訓練，我們採用 Inception V3、MobileNet V2、Xception、ResNet50與 vgg16進行訓練並比較其成果，我們會擷取前10、20、…、50的瞄準軌跡，並使用五個模型進行訓練，評分標準有以下四個：

1. Accuracy：$\frac{\mathrm{TP+TN}}{TP+TN+FP+FN}$ 。
2. F1-score：$2 * \frac{\mathrm{Precision*Recall}}{Precision+\ Recall}$ （$\mathrm{Precision} = \frac{\mathrm{TP}}{TP+FP}$；$\mathrm{Recall} = \frac{\mathrm{TP}}{TP+FN}$）。
3. AUC（area under curve）：ROC（Receiver operator characteristic）曲線，以 $\mathrm{FPR} = \frac{\mathrm{FP}}{FP+TN}$ 為 x 軸，以$\mathrm{TPR} = \frac{\mathrm{TP}}{TP+FB}$ 為 y 軸，AUC 為此曲線下的面積。
4. Train time：模型訓練時間。

第五章　研究結果

在本論文中我們會使用第四章提出的外掛檢測系統進行實驗，檢測方法於第四章第五小節完整敘述，本章的實驗結果分為玩家數據分析與我們提出的數據檢測程式的實際測試結果。

5.1 玩家數據分析

我們蒐集了60位參與者的資料，參與者為國立台灣師範大學師生，我們將表2的玩家數據，以誠實玩家為基準分別與作弊使用者進行比較，分析哪些數據可用於區分誠實玩家與外掛使用者，瞄準軌跡會做為檢測程式的訓練數據，於第五章第二小節介紹。

從圖24可觀察到，自動瞄準的外掛可讓玩家命中率顯著提升，其中又以每秒20度表現最佳。從圖25可觀察到，兩種外掛對首次射擊命中率的表現沒有顯著影響。從圖26可觀察到，透視外掛可對命中頭部率的提升有顯著的影響。從圖27可觀察到，自動瞄準外掛可減少玩家瞄準所需時間。從圖28可觀察到，兩種外掛類型都有助於減少擊敗敵人所需時間。從圖29可觀察到，外掛使用者與誠實玩家的在擊敗敵人的距離有明顯差距。從圖30可觀察到，兩種外掛都可減少擊敗敵人到下次命中其他敵人的時間。從圖31可觀察到，兩種外掛都可將反擊所需時間減少，但減少幅度不大。

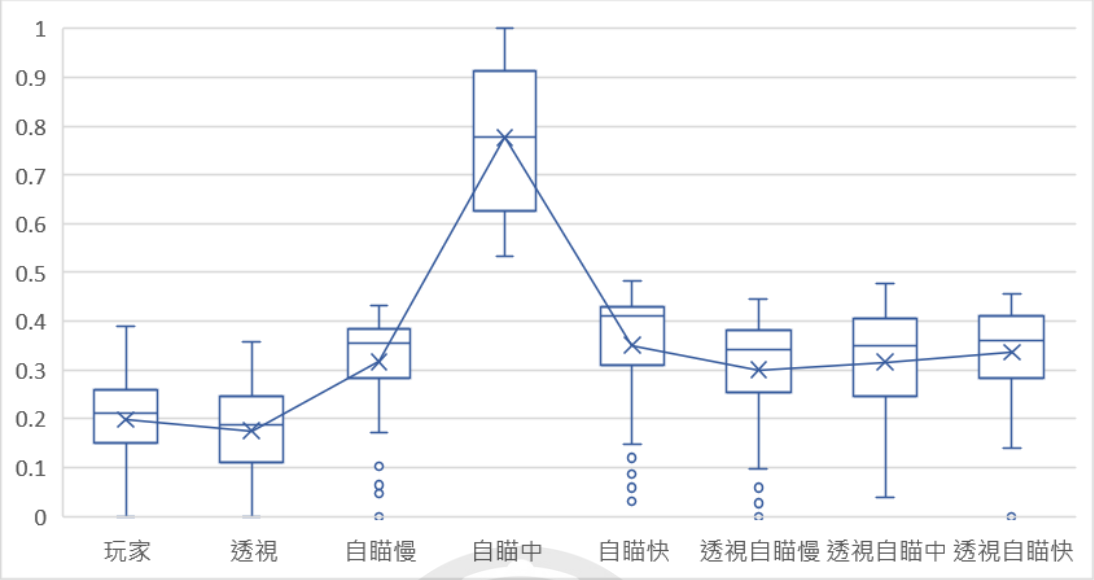


圖 24 命中率人數分布

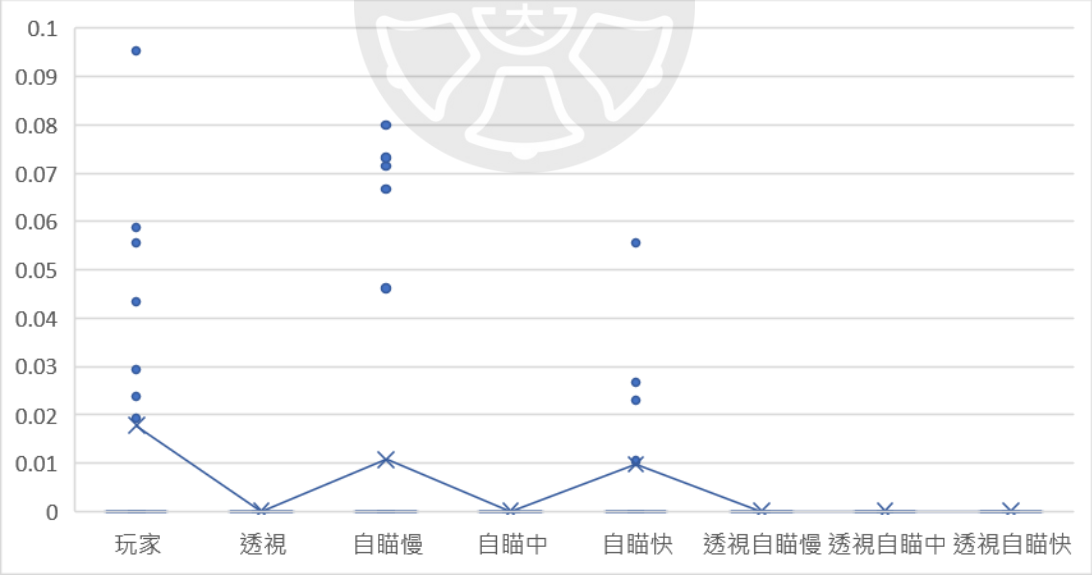


圖 25 首次射擊命中率人數分布

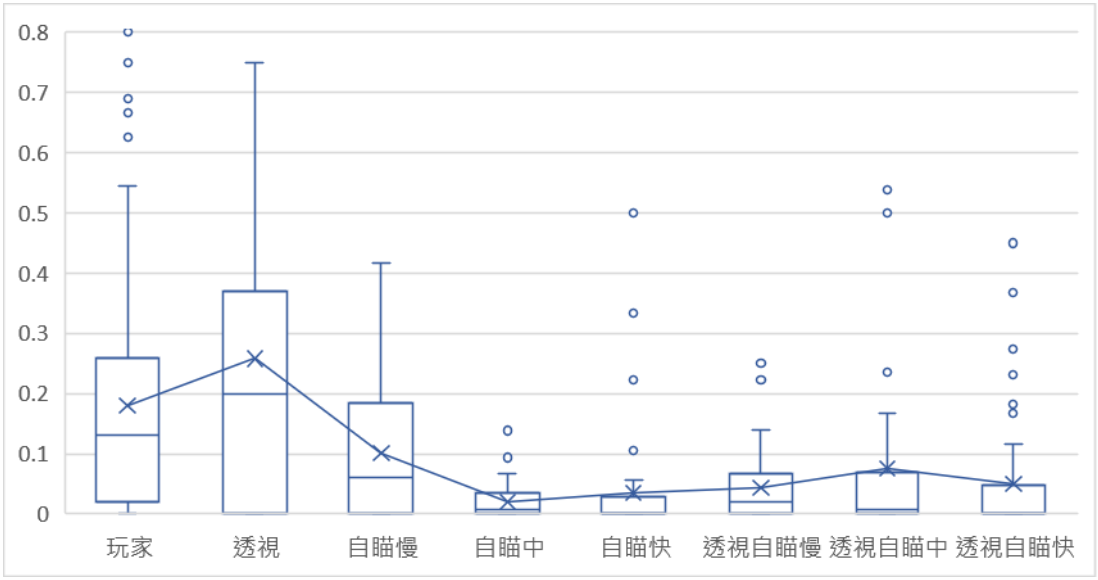


圖 26 命中頭部率人數分布

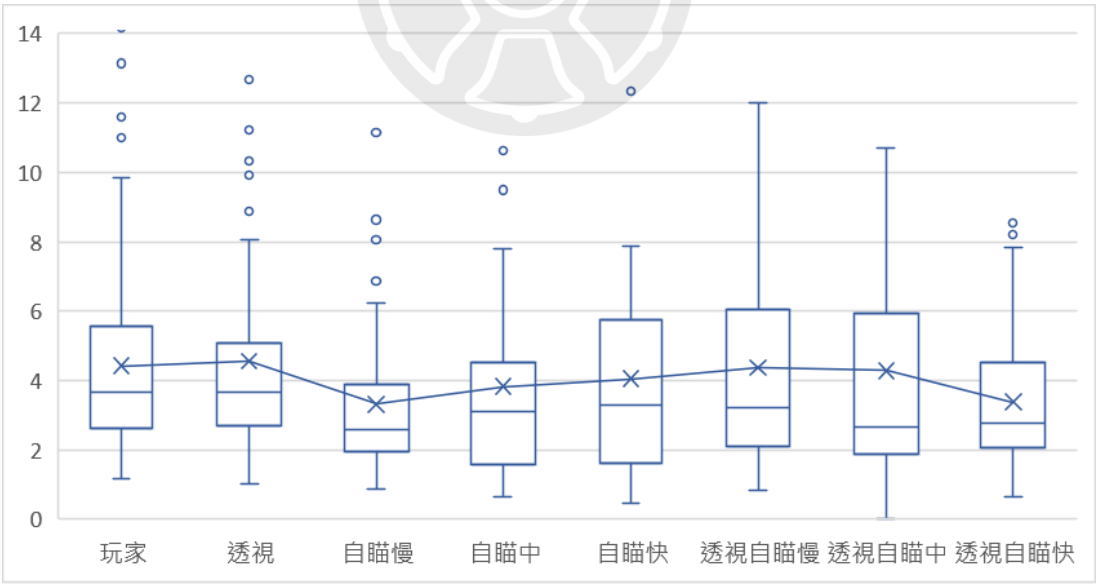


圖 27 首次命中時間人數分布

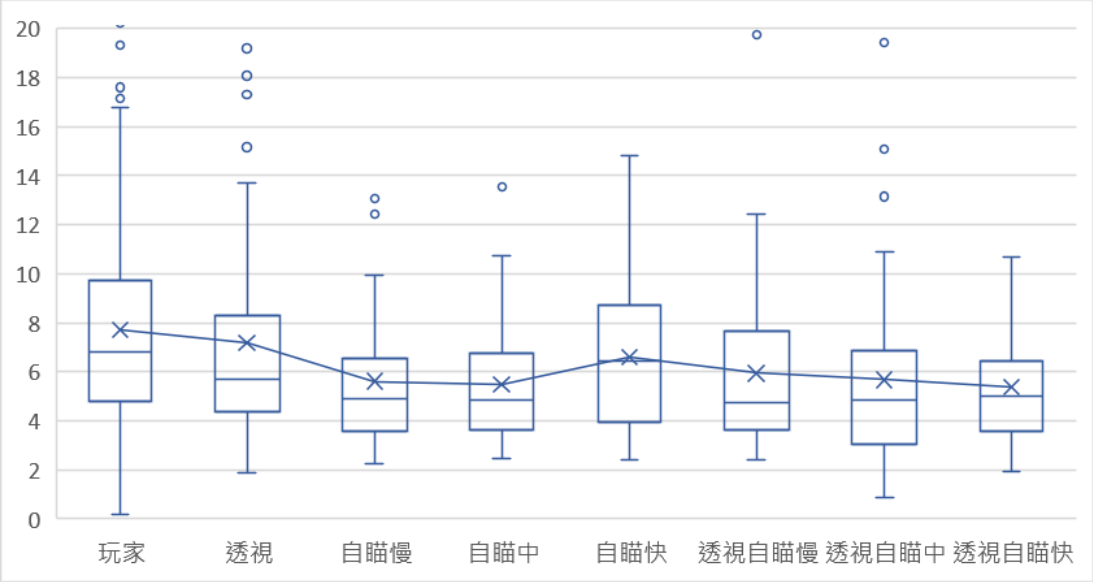


圖 28 擊敗時間人數分布

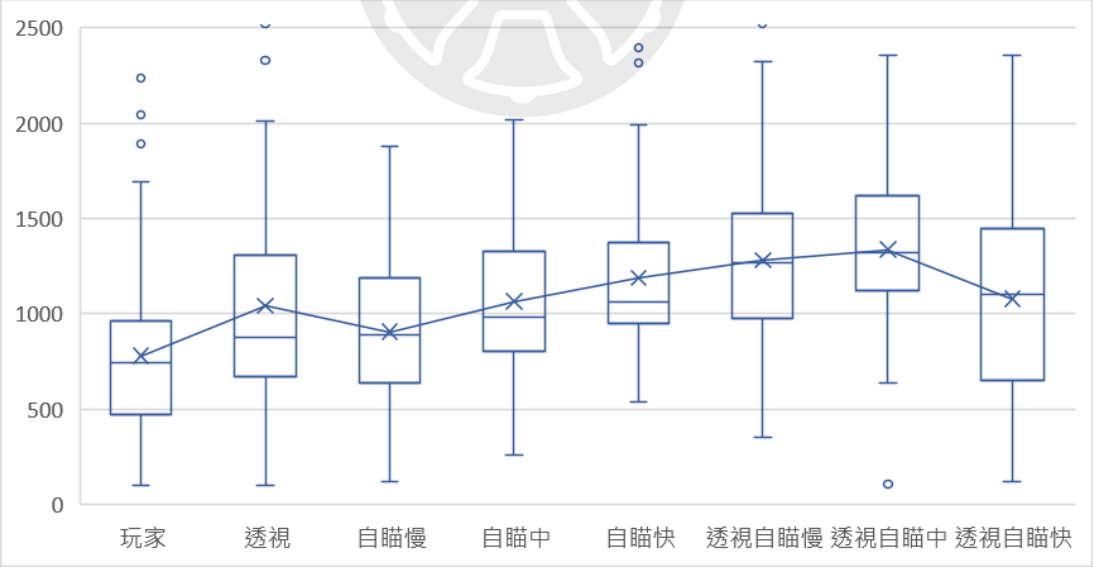


圖 29 擊敗敵人距離人數分布

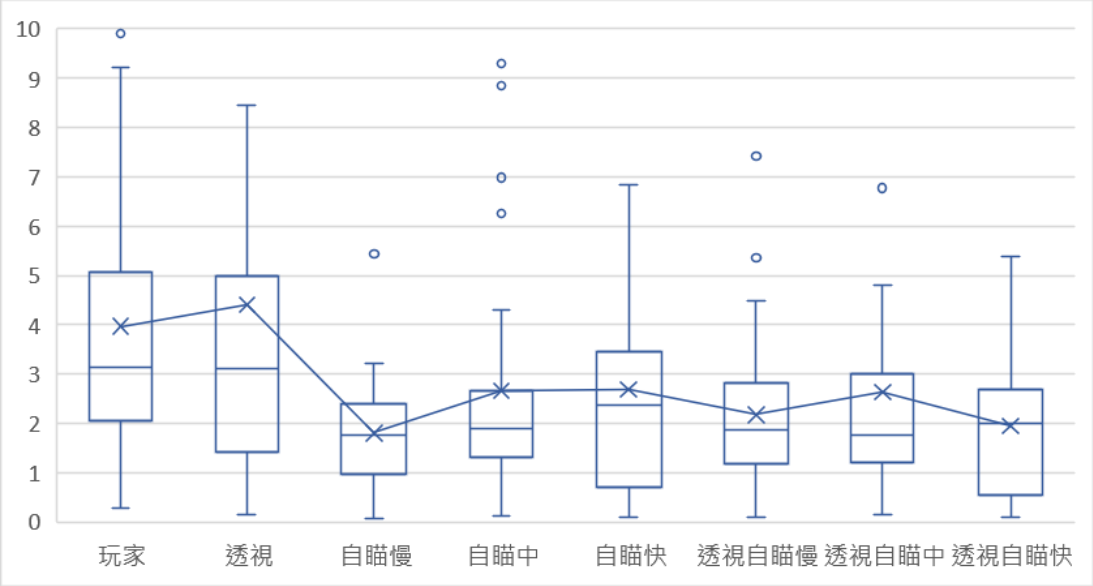


圖 30 擊敗到下次命中人數分布

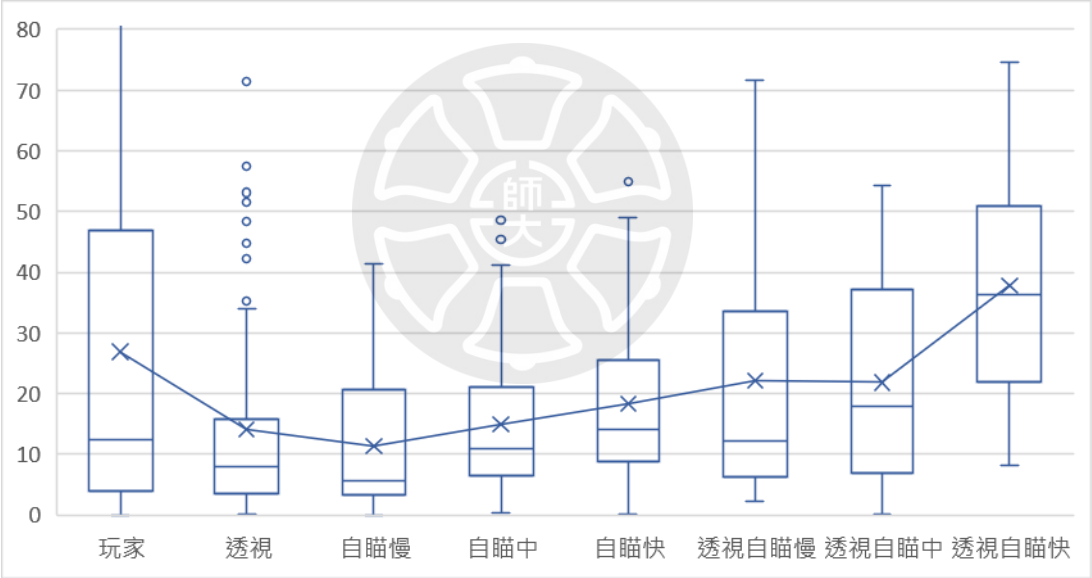


圖 31 反擊時間人數分布

5.2 數據檢測程式測試

我們分別擷取前10、20、30、40和50幀的瞄準軌跡，並將擷取後的資料轉換成圖片，並將其放入五個模型中做為訓練數據，訓練結果於表5~10、圖32~34，在五個模型中，Inception V3的準確率和AUC表現最好，Xception的F1-score表現最好，

MobileNet V2訓練時間最短。

表 5 擷取前10幀實驗結果

10	acc	F1	auc	train time
MobileNetV2	0.8671	0.7670	0.5013	1036.07
Inception V3	0.8674	0.7677	0.5032	2045.97
ResNet50	0.8667	0.7666	0.5000	2838.83
VGG16	0.8672	0.7664	0.5000	2548.31
Xception	0.8356	0.8909	0.7510	1521.35

表 6 擷取前20幀實驗結果

20	acc	F1	auc	train time
MobileNetV2	0.8740	0.8090	0.6621	714.08
Inception V3	0.8995	0.8479	0.7372	1615.47
ResNet50	0.8461	0.7354	0.5000	1868.43
VGG16	0.8462	0.7393	0.5081	1889.24
Xception	0.8246	0.8847	0.7343	1322.84

表 7 擷取前30幀實驗結果

30	acc	F1	auc	train time
MobileNetV2	0.8744	0.8099	0.6854	662.44
Inception V3	0.8935	0.8470	0.7512	1353.27
ResNet50	0.8370	0.7221	0.5000	1596.37
VGG16	0.8375	0.7266	0.5109	1679.40
Xception	0.8281	0.8909	0.7169	1194.15

表 8 擷取前40幀實驗結果

40	acc	F1	auc	train time
MobileNetV2	0.8685	0.8069	0.6923	558.30
Inception V3	0.8983	0.8548	0.7754	1153.44
ResNet50	0.8294	0.7110	0.5000	1409.22
VGG16	0.8295	0.7197	0.5235	1431.37
Xception	0.8197	0.8901	0.6921	1106.73

表 9 擷取前50幀實驗結果

50	acc	F1	auc	train time
MobileNetV2	0.8717	0.8175	0.7179	491.63
Inception V3	0.8986	0.8635	0.7922	725.72
ResNet50	0.9231	0.7021	0.5000	1131.61
VGG16	0.8249	0.7029	0.5000	1290.62
Xception	0.8297	0.8853	0.7012	1047.02

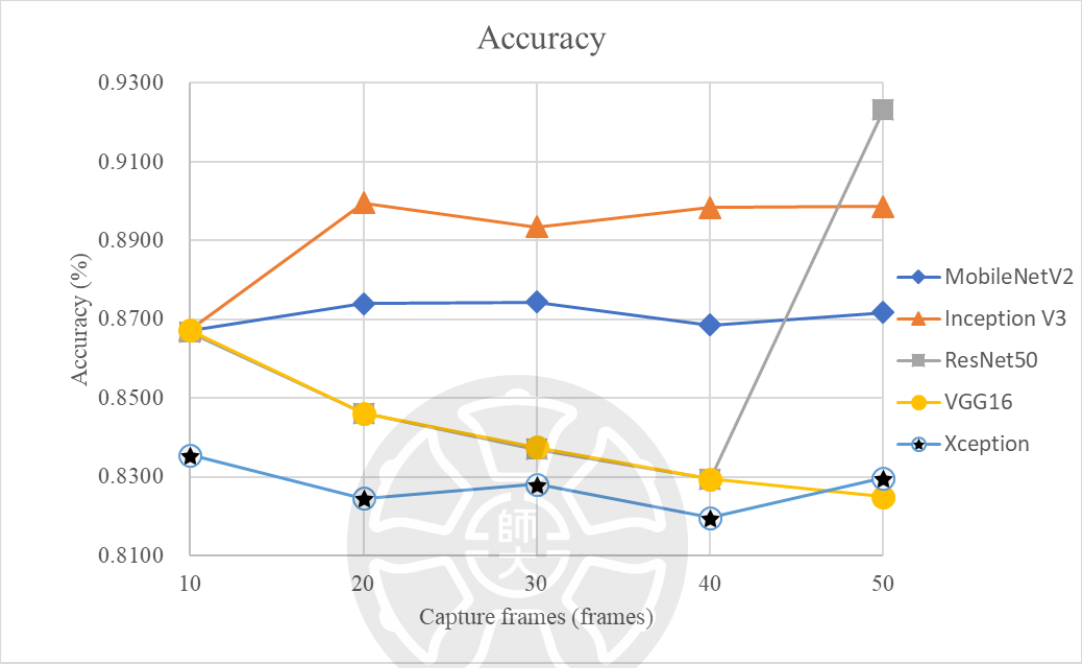


圖 32 Accuracy 比較

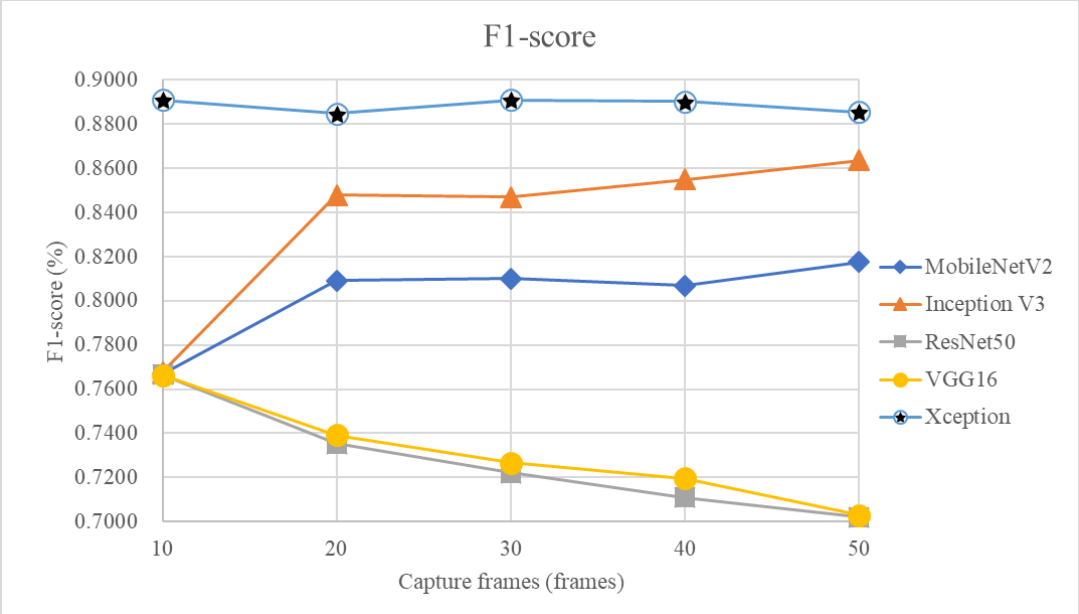


圖 33 F1-score 比較

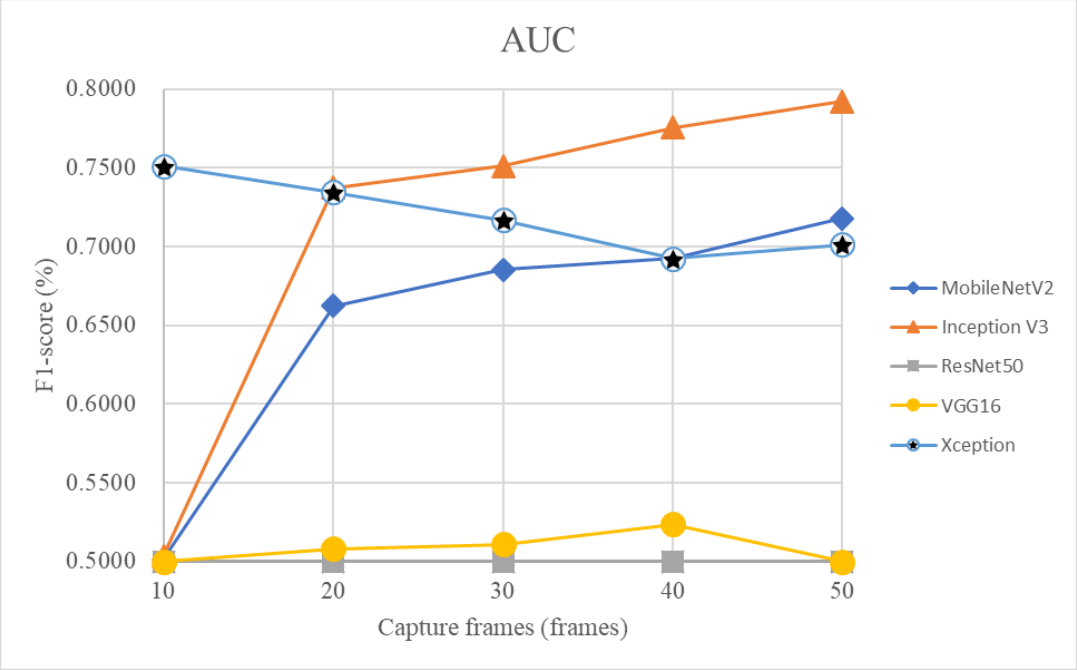


圖 34 AUC 比較

表 10 訓練時間

Train time	50	40	30	20	10
MobileNetV2	491.63	558.30	662.44	714.08	1036.07
Inception V3	725.72	1153.44	1353.27	1615.47	2045.97
ResNet50	1131.61	1409.22	1596.37	1868.43	2838.83
VGG16	1290.62	1431.37	1679.40	1889.24	2548.31
Xception	1047.02	1106.73	1194.15	1322.84	1521.35

表 11 準確性比較

	Accuracy
Recurrent Neural Network	53.08%
Decision Tree accuracy	83.33%
Feature pair SVM	75%
Ensemble Learning using Feature Pair SVMs	77.77%
Multidimensional SVMs	77%
Our system	89.95%

結果表明，自動瞄準外掛在命中率、首次命中時間、擊敗時間、擊敗距離、擊敗到下次命中和反擊時間都能有比誠實玩家更好的表現，透視外掛則在命中頭部率、擊敗時間、擊敗距離和反擊時間有較好的表現，總體來說，命中率、命中頭部率、

首次命中時間、擊敗時間、擊敗距離、擊敗到下次命中和反擊時間可作為分辨特徵。

從圖32中可知，MobileNet V2在30到40幀表現變差，Inception V3在20到30幀表現變差，Xception 分別在20和40幀表現變差，我們認為在擷取幀數上升時，會導致在一張圖片中有不只一條瞄準路徑使得辨識作弊使用者能力下降，但是誠實玩家需要更長的瞄準時間，所以擷取幀數辨常使得誠實玩家的瞄準特徵更加明顯，使得辨識誠實玩家的能力上升；從表11可知，我們所設計的系統在分辨作弊使用者與誠實玩家的表現最好。

第六章　總結與未來工作

只要科技不斷更新，外掛技術就會不斷變化，使得外掛更加真假難辨，這會導致外掛防治工作需要不斷發展更新才能維持遊戲環境的良好風氣，給遊戲玩家更好的遊戲體驗。

在本研究中提出以瞄準軌跡為主玩家分級為輔的數據檢測程式，並比較了五種影像辨識模型得準確率、F1-score、AUC 和訓練時間，其中 Inception V3整體表現最佳，驗證了瞄準軌跡可用於分辨誠實玩家與作弊玩家。

在未來的目標，我們會繼續處理不同幀數導致的分辨率誤差與改進玩家分級的各項參數，修改影像辨識模型以辨識更多像是 GANBOT 的作弊程式，並嘗試將檢測程式運用於市面上的第一人稱射擊遊戲。

參考文獻

[1] 阿新, "FPS 遊戲原理漫談：玩家延時與伺服器同步" 796t.com https://www.796t.com/content/1548091452.html (accessed July. 20, 2022)

[2] Zertalious, "Shellshock.IO Aimbot & ESP." greasyfork.org. https://greasyfork.org/zh-TW/scripts/436330-shellshock-io-aimbot-esp (accessed July. 20, 2022)

[3] "K-40精品輔助" i-faka.com. https://www.i-faka.com/links/k4Ohack (accessed July. 20, 2022)

[4] "萊恩高端輔助定製" liongame.bixone.com https://liongame.bixone.com/0520product/index.php?item=1 (accessed July. 20, 2022)

[5] D. Liu, X. Gao, M. Zhang, H. Wang, and A. Stavrou, "Detecting passive cheats in online games via performance-skillfulness inconsistency," in 2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks（DSN）, 2017: IEEE, pp. 615-626.

[6] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," arXiv preprint arXiv:1409.1556, 2014.

[7] C. Szegedy et al., "Going deeper with convolutions," in Proceedings of the IEEE conference on computer vision and pattern recognition, 2015, pp. 1-9.

[8] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in International conference on machine learning, 2015: PMLR, pp. 448-456.

[9] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 2818-2826.

[10] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 770-778.

[11] A. G. Howard et al., "Mobilenets: Efficient convolutional neural networks for mobile vision applications," arXiv preprint arXiv:1704.04861, 2017.

[12] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in Proceedings of the IEEE conference on computer vision and pattern recognition, 2018, pp. 4510-4520.

[13] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in Proceedings of the IEEE conference on computer vision and pattern recognition, 2017, pp. 1251-1258.

[14] S. Park, A. Ahmad, and B. Lee, "Blackmirror: Preventing wallhacks in 3d online fps games," in Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, 2020, pp. 987-1000.

[15] K.-T. Chen, A. Liao, H.-K. K. Pao, and H.-H. Chu, "Game bot detection based on avatar trajectory," in International Conference on Entertainment Computing, 2008: Springer, pp. 94-105.

[16] L. Galli, D. Loiacono, L. Cardamone, and P. L. Lanzi, "A cheating detection framework for Unreal Tournament III: A machine learning approach," in 2011 IEEE Conference on Computational Intelligence and Games （CIG'11）, 2011: IEEE, pp. 266-272.

[17] M. Willman, "Machine Learning to identify cheaters in online games," ed, 2020.

[18] S. Najari, M. Salehi, and R. Farahbakhsh, "GANBOT: a GAN-based framework for social bot detection," Social Network Analysis and Mining, vol. 12, no. 1, pp. 1-11, 2022.

[19] H. Alayed, F. Frangoudes, and C. Neuman, "Behavioral-based cheating detection in online first person shooters using machine learning techniques," in 2013 IEEE conference on computational inteligence in games （CIG）, 2013: IEEE, pp. 1-8.

[20] R. Spijkerman and E. Marie Ehlers, "Cheat Detection in a Multiplayer First-Person Shooter Using Artificial Intelligence Tools," in 2020 The 3rd International Conference on Computational Intelligence and Intelligent Systems, 2020, pp. 87-92.

[21] "Unreal Engine 4.25 Release Notes" docs.unrealengine.com https://docs.unrealengine.com/4.27/en-US/WhatsNew/Builds/ReleaseNotes/4_25/ (accessed July. 18, 2022)

[22] A. Friedman and J. Schrum, "Desirable behaviors for companion bots in first-person shooters," in 2019 IEEE Conference on Games （CoG）, 2019: IEEE, pp. 1-8.