使用動態分析資料於卷積神經網路上進行惡意程式家族分類

蕭舜文*
國立政治大學資訊管理學系
hsiaom@nccu.edu.tw

摘要

傳統上惡意程式的病毒碼特徵擷取與惡意行為分析需要耗費大量的人力與時間，分析過程通常需要借助資訊安全專家多年對於惡意程式分析的經驗。資安專家通常會比對過去已知的惡意特徵將新發現的惡意程式歸類到已知的惡意程式家族。然而現今新的惡意程式變種數量已經大幅超越人工分析的能力，面對如此資安挑戰，本論文的目的是藉助卷積神經網路對惡意程式進行家族進行自動分類並產生行為特徵，將過去人工的動作轉為自動，與其他過去的研究不同，本論文先對惡意程式進行動態側寫分析並產出其高階的 Windows API 呼叫序列紀錄，而卷積神經網路將視 Windows API 呼叫序列為輸入資料並最終輸出惡意程式家族分類的結果。本文亦利用卷積神經網路的學習結果來解釋其惡意程式之特徵行為。在實驗上我們採用國網中心以及資策會於真實世界蒐集的惡意程式，進行動態分析側寫後進行監督式的訓練以及驗證，其家族分類準確率超過 99%。我們的實驗並證明可以使用有限的 Windows API 呼叫序列就能進行正確的家族分類，如此我們的研究成果可以進一步導入至入侵防禦系統，進行早期的入侵偵測。

關鍵詞：惡意程式、動態分析、卷積神經網路、行為分類

Using dynamic analysis data for malware family classification by convolution neural network

Shun-Wen Hsiao*
Department of Information Management Systems, National Chengchi University
hsiaom@nccu.edu.tw

Abstract

Conventionally, it takes lots of time and human resources to analyze malware to extract its byte signature and malicious behavior. Usually, such analysis process relies on years of experience of malware analysis by the cybersecurity domain experts. They usually classify the unseen malware sample into a known malware family by checking against known behavior characteristics. However, nowadays the number of new malware is too large for human experts

* 通訊作者 (Corresponding author.)

to manually analyze them. To face such cybersecurity challenge, the purpose of this paper is to provide a method to automatically classify malware by using convolution neural network (CNN) and generate behavior characteristics with the help of CNN. Unlike previous research works, we firstly perform dynamic analysis on malware sample and produce its high-level Windows API call sequences as its behavior profile. Then, the API call sequences are fed into the convolution neural network as input to generate the malware family classification result. We also use the learning result of the convolution neural network to explain the behavior characteristics of the malware families. In our experiments, we use the malware samples collected from the real world by the National Center for High-Performance Computing (Taiwan) to generate malware profiles and perform supervised training and validation. The family classification accuracy is over 99%. Our experiments also show that we can use a limited number of Windows API call sequences to perform malware classification; in this case, our result can be used in an intrusion prevention system for early malware detection.

Keywords: Malware, dynamic analysis, convolution neural network, behavior classification

壹、前言

傳統上惡意程式的分析需要耗費大量的人力與時間，分析過程通常需要借助資訊安全專家多年對於惡意程式分析的經驗。然而現今的惡意程式變種數量已經大幅超越人工分析的能力，但是資訊安專家的知識卻無法如同惡意程式般迅速的傳承。因此資訊安全的分析人員需要有一個有效的方案與智能工具來進行惡意程式的分析、研究與側寫，進而判斷一個可疑的程式是否具有某些行為特徵，而可以被分類至已知的惡意程式家族。若已知某一新的惡意程式變種為已知的惡意程式家族，那麼資訊安全人員可以採取相似的防範措施，或是更進一步可以預先暫停該惡意程式的執行，而達到入侵預防的目的。

本論文的目的是提供一套完善的動態程式行為側寫與卷積神經網路分析工具，本文的分析目標是主機的可執行程式，程式將在虛擬機器中進行動態的行為側寫，並將側寫結果導入學習式的卷積神經網路分析工具來進行惡意程式家族的分類。本論文提出之系統具備以下重要的能力：第一、有能力對惡意程式進行動態分析並產出側寫檔案，本論文所產出的側寫檔案是 Windows API 呼叫序列，而非組合語言或系統呼叫之序列；第二、利用類神經網路之演算方法判別所蒐集之側寫資料是否屬於已知的惡意程式家族；第三、利用類神經網路之結果進行惡意程式之特徵分析，以利資安人員在分析惡意程式家族行為時，能盡速抓取惡意程式家族之特徵。

由於惡意程式變種個體產生的速度已經超越人工分析的速度，因此本文認為採取智

能的分析手段是必要的。根據過去的研究發現惡意程式的行為通常是由較小的行為所組合而成的，而不同的組合行為可以被惡意程式的撰寫者利用，而相似的組合行為可以進而產生隸屬於同一惡意程式家族的新變種。因此同一惡意程式家族的惡意程式具有一定程度的相似行為，並且其 Windows API 函式庫的使用方式有相似的順序，所以惡意程式變種間會具有某種相似的組合關係。舉例來說，在 Windows 系統中，若要使用網路傳輸資訊必定會根據 Windows 所定義的網路使用 API 規範來撰寫其程式。若要讀檔案其情況亦是類同，而先「讀檔案」再行「網路傳輸」則可能隱含竊取資料的惡意行為。目前有相當部分的惡意程式是來自於各惡意程式家族的混種以及拼裝，因此本論文使用卷積神經網路針對惡意程式的動態分析行為進行特徵切割與分析，最終有效地對此類組裝行為特徵來分類未知的惡意程式至已知的惡意程式家族。

本論文之特色為使用動態分析之 Windows API 呼叫序列，而不使用系統呼叫或組合語言之序列，雖然過去的研究表示使用系統呼叫仍可以辨別是否為同一之程式，但本文認為系統呼叫之層級過低。例如：以系統呼叫之 write 函式而言，可能表示檔案的寫入或硬體裝置的寫入，因此比較難從系統呼叫回推至惡意程式行為。而在小片段的時間之內，系統呼叫的數目通常也較 Windows API 呼叫多，因此在序列分析上可能產生必非要之雜訊，且序列較長也不利卷積神經網路分析，亦較難萃取出惡意程式的意義。而 Windows API 擁有明確的意義，較容易幫助我們回推惡意程式的行為。舉例來說，本文所紀錄的 Windows API 包含：載入的動態函式庫、檔案的讀寫與存取、程序 (process) 的執行、Windows Registry Key 的使用、網路存取行為等。本文是利用虛擬環境之虛擬機器內省機制 (Virtual Machine Introspection, VMI) 記錄惡意程式於執行時期的 Windows API 呼叫行為，因此同時也可以支援即時的入侵防禦。

本論文使用卷積神經網路進行惡意程式家族分類的原因如下：其一、卷積神經網路為一成熟的神經網路，因此在理論與計算上都有卓越的成效，本論文以 GPU 進行最佳化之運算可以在有限時間內完成卷積神經網路的訓練與測試。第二、卷積神經網路過去常用在手寫數字的辨識以及圖型的辨識，其神經網路中含有卷積核 (kernel) 或稱過濾器 (filter) 的設計，該過濾器可以表示手寫數字的片段特徵，並進一步比對該片段特徵是否出現在整個手寫數字中。過濾器以遮罩的概念在整個手寫數字中移動，若特徵的片段與手寫數字的片段起反應，則其下一層輸出會顯現出該特徵反應，表示有相似於過濾器特徵的片段被發現。本文認為過濾器之設計與惡意程式之短行為片段可相互配合，若設定適當之過濾器尺寸，可以利用過濾器找出惡意程式家族分類之特徵。

本論文之創新在於利用卷積神經網路釋家族之行為特徵，利用 Windows API 進行高階語意的分析，並討論惡意行為是具有組合性的行為概念，對惡意行為進行切割與分析。以上綜合的面向是過去的研究並未著墨的部分。

本論文第二章為文獻探討，將討論過去惡意程式家族行為分析下，使用動態行為分析技術之文獻，並介紹使用類神經網路用於惡意程式家族分類之文獻。第三章為本論文使用之動態分析系統之側寫檔案說明。第四章為本論文使用之卷積神經網路設計以及參

數之規劃。第五章為惡意程式家族分類之實驗數據以及結果。第六章為討論以及總結。

貳、文獻探討

2.1 惡意程式之動態側寫方法

根據不同的程式行為，側寫機制應該針對程式主體 (subject) 的各種標的 (objects) 製作客製化的側寫檔案 (profile) 以期能精準地描述該程式之行為或特徵，例如：檔案存取、程序命令、各項資源的操作。惡意程式的分析與偵測有數種不同的研究方向，各研究分別針對不同的主體和標的為基礎做行為分析，但基本上可以分為動態分析與靜態分析兩種主要分析方法。

動態分析是蒐集該程式正在運行時所能取得的資訊作為行為分析的依據，例如記錄該程式所使用的系統呼叫 (system call) [7] 便是常見的一種動態分析方式。Panorama [24] 提出了在系統中追蹤資訊流 (information flow) 的方法，抓取程式中可疑的資訊存取模式和處理行為用以發掘惡意程式的執行行為。而靜態分析 [22] 則是以應用程式的原始碼或是執行檔為分析的基礎，直接分析內部運作的流程或是資料使用的情況。

動態分析的缺點在於其不完整性，也就是在執行程式時無法有效而完整的把所有的行為都記錄下來，肇因於程式可能有不同的輸入參數而導致不同的執行路徑 (execution path)，而並非每一條執行路徑都可以在執行時期可被觀察，因此動態分析需要時間和大量的測試來取得較完整的程式行為。動態分析其優點是可以取得實質的參數值或訊息，例如：當下從網路傳來的訊息、駭客使用的檔案。靜態分析的優缺點則恰與動態分析相對，靜態分析可以完整地了解程式的運作行為，因為所有的執行內容都可以在原始碼或執行檔中取得，但是靜態分析缺乏執行時期的參數以及執行路徑，並且有時執行的程式碼可能是另外由網路傳輸過來。過去的研究 [7] 亦指出採用動靜態分析方法實際上是取決應用程式本身的特性，端看程式是否容易在動態分析中能挖掘出所有的執行行為。

本文使用 Windows API 呼叫序列為側寫檔案之主體，並採用虛擬機器內省技術錄製惡意程式於虛擬機器中執行時期所呼叫的 Windows API，以達到側寫的目的。本論文提出惡意程式在執行時，即使是較短的動態分析側寫檔案，亦可以達到分辨不同惡意程式家族的結果。本論文只專注在動態分析的側寫檔案，而靜態分析之內容並非為本論文之討論範圍。依照動態分析的主體或標的，分析的對象可分為系統呼叫以及 API 呼叫。

系統呼叫分析研究的假設是：一隻程式的執行歷程可以用系統呼叫來表示。系統呼叫是程式與作業系統之間溝通的橋樑，因此系統呼叫被用來表達一隻程式使用作業系統的特徵。Forrest et al. 的研究 [7] 使用連續的系統呼叫當作程式的特徵，並實驗在信件伺服器軟體 sendmail 之上。由於 sendmail 程式的複雜性很高，因此他們採用動態分析的方法輸入 112 組正常寄送郵件的 messages，來取得程式的系統呼叫側寫檔案以當作正常

的執行模型。於實際偵測時，他們計數系統呼叫發生的次數比對於正常的模型，來評估當下的執行是否為攻擊。此研究是以系統呼叫為基礎而執行入侵偵測的濫觴，其後作[10] 亦以數學模式研究產生程式的執行模型時，該採用各項參數：如：連續系統呼叫的個數、判別正常與異常的閥值。Lee [14] 在模型描述上更進一步採用資料探勘的方式，將惡意以及正常的 sendmail 系統呼叫行為做分群，用來區分攻擊時的系統呼叫順序以及正常程式進行時的系統呼叫順序。Kruegel et al. [13] 提出的學習模型亦將系統呼叫的文字參數考慮在內，但他們並不考慮呼叫的順序。

系統呼叫分析是對程式來說最基礎的分析，由於作業系統可以攔截該程式所有的系統呼叫，因此對於紀錄和分析來說是相對簡單而方便的。不過，我們認為系統呼叫的層級太低，無法反映上層程式的實際執行狀況以及語意，尤其程式中某些執行步驟並不牽涉到系統呼叫或是攻擊者可以任意呼叫不相干的系統呼叫來干擾，因此用系統呼叫建立側寫檔案作為了解程式執行的過程並不足夠。此外，我們認為系統呼叫比較適合用來建立固定而正常的程式模型，作為建立異常偵測的正常模型，但是只有使用系統呼叫來描述惡意程式並不足夠。因此從偵測的角度上，我們希望有更上層的語意可以被推導出。

Windows API 呼叫分析的起源是因為 Windows 作業系統是惡意程式寄生主要標的，因此有部分學者將系統呼叫的側寫方法改為 Windows API 呼叫。Bayer et al. [1] 採用動態分析的方式將惡意程式轉為 DLL 檔案並放置在電腦模擬器 (emulator) 中執行，並將惡意程式的系統呼叫與 Windows API 呼叫記錄下來，記錄包含有：檔案操作、Windows Registry 操作、網路操作、service 操作等。Bayer et al. [1] 亦將側寫檔案輸入至分群演算法中，用以偵測同類型的惡意行為。Willems et al. [23] 開發 CWSandbox 用以實行 Windows API hooking 以及 DLL 檔案的 injection，如此可以讓惡意程式在沙箱內執行並擷取執行的內容。BotTracer [15] 亦將殭屍程式在主機上蒐集資料的行為用磁碟存取相關的系統呼叫以及 Windows API 呼叫記錄下來。Hsiao et al. [11] 將惡意程式放置在虛擬機器中執行，並在擷取指定的 Windows API 成為側寫檔案，計算不同惡意程式以及正常程式的相似性，用以偵測惡意程式的存在。本論文採用修改過的模擬器 [11] 於虛擬機器當中擷取所需要的 Windows API 呼叫紀錄，並確保被側寫之惡意程式無法察覺身處於監控之下。

虛擬機器內省的技術通常被用來執行動態分析。該技術可讓管理者在虛擬機器外監控運作中的虛擬機器內部的所有軟硬體資料。如此我們可以不需要在虛擬機器中安插代理程式，也不需要取得虛擬機器中作業系統的管理權限就可以進行各種安全監控的功能，例如：掃毒、異常偵測、被動式側寫等。虛擬機器內省的技術目前已經被用在惡意程式的研究上，虛擬機器內省技術目前可以監控的內容包含有：CPU 暫存器、主記憶體、磁碟機、網路以及硬體層的事件。在外的監控程式可以拼湊以上的資訊將虛擬機器中執行的樣貌完整的表現出來，以補足虛擬機器內外的語意隔閡 (semantic gap)[4]。

虛擬機器內省的動態分析研究大致分為模擬器與真實虛擬機器。模擬器以 QEMU 為學術界最常使用。而真實虛擬機器常以 Xen 以及 KVM 兩種公開原始碼的虛擬機器監

控軟體 (virtual machine monitor, VMM)為主。QEMU 則是採用模擬的執行環境，因此在最精細的開發條件下，開發測試人員可以針對每一個在 QEMU 上所執行的每一個 CPU 指令作分析及測試。但是在這樣的監控環境之下，其效能不佳，雖然常用於動態分析惡意程式，但通常是實施離線的研究。TEMU [19] 則是將 QEMU 大幅改寫，可以安插 plug-ins 至 QEMU 中，進行 API hooking 與汙點分析。Hsiao et al. [11] 則改寫 TEMU，使其具有 process tracking 的功能，能將汙點分析由記憶體區塊或檔案繼續延伸至所有相關的程序，並將 API hooking 的機制延伸至 Windows API 以及其輸入輸出參數。

在虛擬機器的側寫使用上，Chen et al. [4]最早論述虛擬機器可以幫助資安進行安全的記錄以及偵測的問題。Garfinkel and Rosenblum [8] 首先提供了一個基於虛擬機器內省技術的入侵偵測系統，他們在 VMM 的平台上將觀測的虛擬機器資訊透過 VMM 傳遞給另一台具有入侵偵測功能的虛擬機器，並由該虛擬機器來辨識是否有安全問題產生。在 VMM 中，他們製作了一份 OS interface library，利用 VMM 能取得的低階資訊（CPU register、記憶提存取、硬碟存取），並轉換成高階資訊（程序、檔案、函式庫引用）。他們比對了虛擬機器內部以及虛擬機器外部所取得的資訊兩相比較，若有不一致，則表示虛擬機器可能遭受到攻擊。ReVirt [6] 則將安全紀錄的機制從虛擬機器中，移動至 VMM 當中。如此可以避免記錄檔被惡意的程式銷毀或是竄改。VMwatcher [12] 同樣克服了 semantic gap 的問題，可以重新建立 guest 的 view，以檔案、程序、核心模組等高階資訊來監視 guest OS。Ether [5] 可將 VMI 的技術應用在有硬體加速的虛擬機器上，如此可以增快其側寫速度。

2.2 類神經網路於惡意程式家族分類之方法

過去數十載，對抗惡意程式的策略有漸漸導入使用機器學習的趨勢，以下討論數種不同的機器學習方法以及資料對象用以執行惡意程式家族分類。

惡意程式檔案位元分析是早期常見的分析手法，將惡意程式的執行檔案每一個位元做為一個單位，編碼為一連串 16 進位的值之後，讓機器學習演算法進行訓練。通常為了保留短暫的序列，編碼也可能採用 n-gram 的方式，將 n 個連續的位元一同編碼。當 n 為 1 時，一個位元需要 256 種編碼。N-gram 的編碼方式首見於 IBM [20]，他們在一款防毒程式當中，採用 3-gram 的編碼處理惡意程式的分類問題。其分類演算法是採用類神經網路，但要特別注意的是 3 位元的特徵數量是 256^3，其特徵數量相當大，因此通常還需要搭配降維的技巧來處理資料集。

惡意程式檔案 opcode 分析採取另外一個策略只看執行檔案中的 opcode，因此必須先將 machine code 對應至 opcode 序列，通常是以組合語言表示之。根據文獻 [3] 表示，正常程式以及惡意程式所使用的個別 opcode 其分佈是略有不同的，因此同樣可以使用 n-gram 的方式將 opcode 序列編碼，再進行分析。

可攜式執行檔案 (portable executable, PE) 是 Windows 的可執行檔案格式，常見於

exe、dll 以及 sys 檔案。PE 檔案除了包含可執行之 CPU instruction 之外，還包含了一組標頭檔案，該標頭檔案記載包含編譯的日期、引用的函示庫、區段資料、檔案大小等資訊。過去的研究常自標頭中萃取有用的欄位來進行惡意程式分析，例如：檔案位於硬碟的位置、從外部載入的函式庫、匯出的函式庫、執行碼的長度。Saxe and Berlin [18] 利用 PE 檔案之中的數值型資料進行深度學習，共擷取 1024 種不同的特徵，製作 2 層的隱藏層類神經網路，來判別資料集是否為惡意或正常程式。

函式呼叫亦是常用的分析對象，但是過去利用函式呼叫來進行類神經網路的惡意程式分析多半使用靜態分析的手法，而本論文則是採取動態分析的手法。他們利用所收集到的 PE 檔案，將其內部會使用到的 Windows API 呼叫擷取出來，並且計算其使用頻率。在總數約十二萬種不同的 Windows API 呼叫當中，進行類神經網路的分析。Ravi and Manoharan [17] 發展一套惡意程式入侵偵測系統，他們採用 4-gram 的方式編碼所在 PE 檔案內所收集到的 Windows API 呼叫，並進行惡意程式的偵測，其準確率約為 90%。另有文獻 [21] 採用相同的特徵擷取策略，但是最後使用的是 SVM 的分類演算法。

惡意程式圖形化是將 PE 檔案的每一個 byte 視作為一個 pixel，並將該 byte 的值對應為數值為 0 至 255 的灰階圖像點，然後進行圖像的相似度分析。Nataraj et al. [16] 在將惡意程式轉換為灰階圖像後，進行紋理特偵萃取，再利用萃取之特徵進行 k-nearest neighbor 演算法進行分類，來將相似的惡意程式分至同一家族當中。Gibert [9] 則是同樣採用將惡意程式轉換至灰階圖像的做法，但他直接使用卷積神經網路進行惡意程式的分類。他使用了三種不同的卷積神經網路設定，基本上兩層卷積層再加上一層全連結層有最好的分類效果，其準確度約在 98%至 99%附近。Microsoft Malware Classification Challenge [25] 最終獲勝者其所提出之方法為以上之混合方法，他們使用了 PE 的 binary 檔案、反組譯之組合語言檔案以及圖像式的分析。最終他們的特徵為 2-gram, 3-gram, 4-gram 的 opcode、每一個 PE 區段的長度、反組譯位元（前 800 位元）所對應之灰階圖像，其混合之惡意程式分類家族演算法之準確率為 99.87%。

本論文與上述研究之不同處在於採用了動態分析之 Windows API 側寫檔案，而大多先前的研究都只著眼於靜態分析的研究，並無討論動態分析之分類能力。本論文使用卷積神經網路之設計，利用卷積神經網路之過濾器將 Windows API 呼叫序列的片段視為區域特徵，不需要拘泥各別 API 呼叫的細節差異以及 API 呼叫出現的位置，而是將小片段的行為特徵抓取出來，不論是該特徵序列為有些微的變化（插入、刪減）、平移等，都可以由卷積神經網路自行判別捕捉過濾器的共通要素。而卷積神經網路可以自行根據訓練資料計算出過濾器的值，這也可以使得資訊安全人員不需要花費額外的人工分析時間來進行特徵值的萃取，這對於資訊安全人員來說是相當大的助益。

參、動態分析側寫檔案

本論文之動態分析側寫系統是使用 Hsiao et al. [11] 所設計之虛擬機器內省機制。其輸入為惡意程式之可執行檔案，其輸出資訊為該惡意程式的 Windows API 呼叫側寫檔案。該虛擬機器內省機制共設定 62 項 Windows API 的 hook。因同一類之 Windows API 有眾多版本（例如：LoadLibraryA、LoadLibraryW），其不重複之 Windows API 版本見表一。表一之 Windows API 涵蓋主要五大類的行為：函式載入行為、程序處理行為、檔案讀寫行為、Windows Registry 存取行為以及網路存取行為。本論文認為該五大類的行為足以表示惡意程式在活動的期間能記錄下該惡意程式對於系統產生的影響。

表一：Windows API Hooking 表

Library	Process	File	Registry	Network
Load Library	CreateProcess OpenProcess ExitProcess WinExec CloseHandle CreateRemote Thread TerminateProcess TerminateThread CreateThread OpenThread	CopyFile CreateFile WriteFile ReadFile DeleteFile	RegOpenCurrent User RegQueryValue RegEnumValue RegOpenKey RegCloseKey RegSetValue RegCreateKey RegDeleteKey RegDeleteValue	WinHttpConnect WinHttpCreateUrl WinHttpOpen WinHttpOpenRequest WinHttpReadData WinHttpSenRequest WinHttpWriteData WinHttpGetProxyForUrl InternetOpen InternetConnect HttpSendRequest GetUrlCacheEntryInfo

其產出之動態分析側寫檔案如圖一所示。該動態分析側寫檔案為一 XML 文件，於 <execution> 標籤內部包含了該惡意程式於執行時期的 Windows API 呼叫序列，包含呼叫的時間、API 名字、使用的 API 參數、回傳值等資訊。圖一之範例為惡意程式家族 Morstar 的一個變種，其執行檔案之雜湊值為 6e68e0ff5efce1297faa3b49，當時於虛擬機器內執行時，其 process ID 為 1876，整個動態分析共進行 300 秒，總共包含 1,121 個如表一所示的 Windows API 被記錄下來。圖一共節錄其中 6 個 Windows API 呼叫，第一個呼叫為 CreateFile，其目的是讀取 (GENERIC_READ) 一個 Windows 系統內建的設定檔案 (WindowsShell.Manifest)，該檔案標明了在連接函式庫時，應該使用哪一個版本的 dll 檔案，該 API 呼叫執行成功 (Return="SUCCESS")，其時間戳記為 352080000。第二個呼叫為 LoadLibrary，表示該程式欲載入 comctl32.dll。第三個呼叫為 RegQueryValue，其目的是測試 Windows 之安全性設定 (DisableImprovedZoneCheck)，但該惡意程式未能正確存取該設定。第四個呼叫同為 RegQueryValue 函式，但其機碼值為惡意程式自訂之

s5924.exe，其目的是確認該惡意程式是否之前已經感染過該主機，若不曾感染過該主機則繼續動作。第五個呼叫是 RegQueryValue，但其機碼值為 User Agent，其目的是探索該主機之瀏覽器訊息，根據回應可知該主機使用 Mozilla/4.0 (compatible; MSIE 6.0; Win32) 版本之瀏覽器。圖一之最後一個範例呼叫為 CreateFile，該惡意程式會將存取系統開機之自動執行檔案 c:\autoexec.bat，並且稍後將進行修改。

```
<profile>
 <meta> <hashID>6e68e0ff5efce1297faa3b49</hashID>
  <processID>1876</processID><duration unit=”second”>300</duration>  </meta>
 <execution>
  <CreateFile
hName=“C:\WINDOWS\WindowsShell.Manifest“ desiredAccess=“GENERIC_READ”
creationDisposition=“OPEN_EXISTING” Return=“SUCCESS” Time=“352080000” />
  <LoadLibrary lpFileName=“comctl32.dll” Return=“SUCCESS” Time=“354700000”
/>
  <RegQueryValue
hKey=”HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Intern
et Settings\DisableImprovedZoneCheck” Return=”FAILURE” Time=”354910000” />
  <RegQueryValue hKey=”HKEY_LOCAL_MACHINE\Software\Microsoft\Internet
Explorer\Main\FeatureControl\FEATURE_OBJECT_CACHING\s5924.exe”
Return=”FAILURE” Time=”354980000” />
  <RegQueryValue
hKey=”HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Inte
rnet Settings\User Agent” Return=”SUCCESS” type=”REG_SZ” data=”Mozilla/4.0
(compatible; MSIE 6.0; Win32)” Time=“381260000”
  <CreateFile hName=”c:\autoexec.bat” desiredAccess=”GENERIC_READ”
creationDisposition=”OPEN_EXISTING” Return=”SUCCESS” Time=”461060000” />
…
```

圖一：動態分析側寫檔案（節錄）

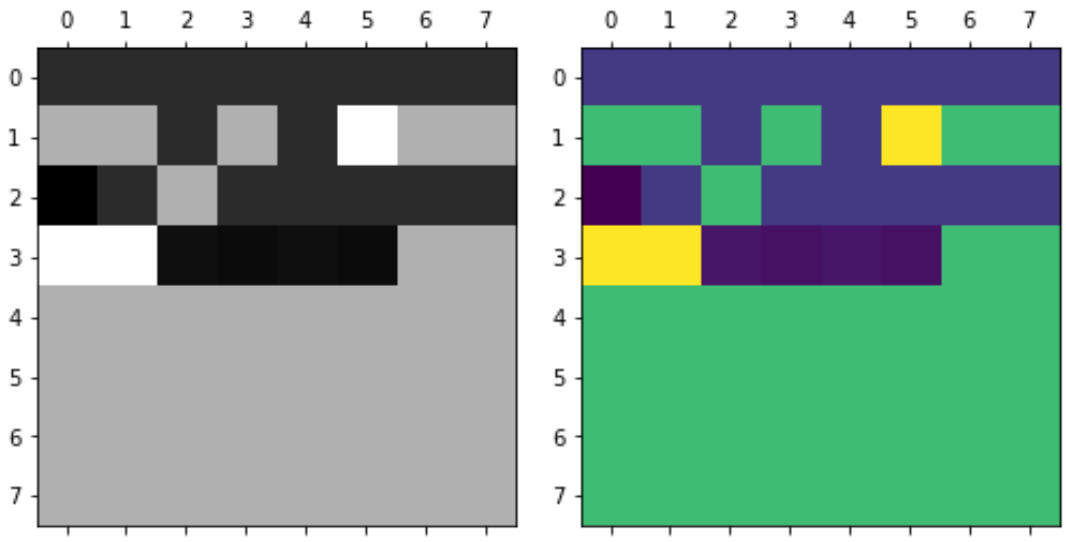
以上為動態分析側寫之檔案之範例，本論文取得所有側寫檔案之後，將對其每一個 Windows API 呼叫進行編碼。首先針對一資料集，計算每一個 Windows API 呼叫之累積出現次數，並作排序。其排序之順序如表二所示，所有的動態分析側寫檔案的 API 呼叫將依照該頻率順序重新編碼為相對應之小數，因此 API 的編碼值會介於 0.0 至 1.0 之間，本論文並預留 0.0 之值於卷積神經網路 padding 之用，其轉換之後之純小數陣列將做為卷積神經網路的輸入值。表二為某資料集編碼表，只列出部分的 API 編碼。

表二：API 編碼表（節錄）

API	頻率	編碼
CreateProcess	89	0.0015223303627935618
CreateProcessInternal	125	0.002138104442125789
DeleteFile	143	0.0024459914817919026
ExitProcess	144	0.002463096317328909
RegDeleteKey	252	0.004310418555325591
CreateThread	262	0.004481466910695653
RegEnumValue	344	0.005884063424730171
OpenProcess	1,210	0.02069685099977764
RegSetValue	1,471	0.025161213074936285
RegCreateKey	1,806	0.0308913329798334
LoadLibrary	4,449	0.07609941330414108
…	…	…
RegQueryValue	16,955	0.290012486529942
CreateFile	24,462	0.4184184869062484

以圖一之動態分析側寫檔案為例，其六個 Windows API 呼叫序列將重新編碼為一純小數之陣列：[0.4184184869062484, 0.07609941330414108, 0.290012486529942, 0.290012486529942, 0.290012486529942, 0.4184184869062484]。

若將一長度為 64 之 Windows API 呼叫序列依照表二之編碼，可得一長度為 64 之陣列。本論文將該陣列做視覺化轉換，換至灰階調 (grey) 與綠色調 (viridis) 可得下圖二之視覺化圖型。Windows API 呼叫序列由左至右再依序向下排列，不同之顏色表示不同之 API 編碼。以灰階調為例，全黑之顏色為編碼最小之數值，而全白為編碼最大之數值，中間值為灰色；以綠色調之圖為例，深藍色為編碼最小之數值，而黃色為編碼最大之數值，中間值為綠色。由此我們可以利用視覺化的方式，進行數據的檢視以及比對，視覺化的圖型只是幫助資安分析人員可以迅速地辨別使用的 Windows API 呼叫種類，並不影響後續卷積神經網路的運算。



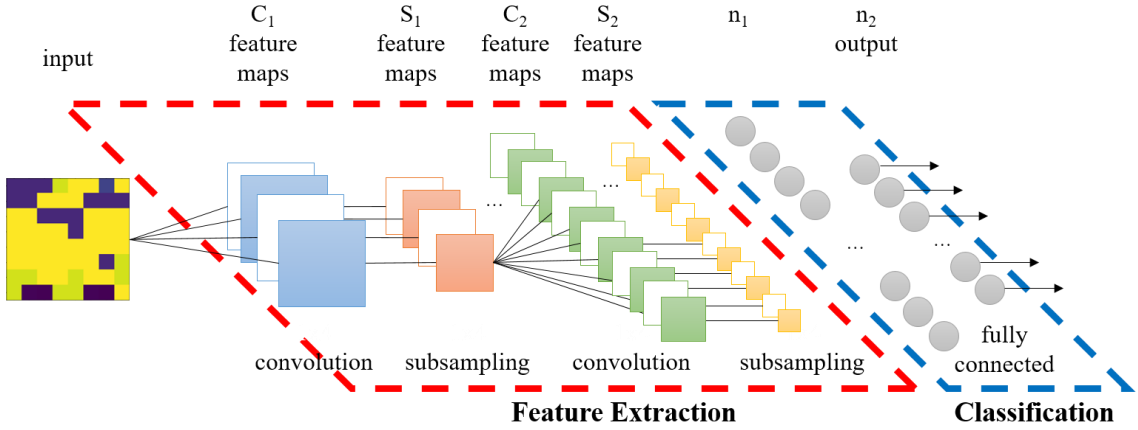
圖二：視覺化之 Windows API 呼叫序列之編碼結果。左圖為灰階調，右圖為綠色調。

肆、卷積神經網路設計

4.1 卷積神經網路

本論文採用 Google Tensorflow 之架構設計卷積神經網路，採用標準之兩層式卷積神經網路設計，如圖三。將第參章所設計之編碼矩陣作為輸入層，輸入之長度為 n 之一維矩陣（註：並非為 8x8 的方陣，方陣僅在視覺化時使用），並加入一卷積層 (C_1)，該層有 K_1 個過濾器，每一個過濾器大小為長度 F_1 之一維矩陣。在卷積層中我們使用 padding 技巧，並每次移動一個 API 進行過濾，卷積層同樣輸出長度為 n 之矩陣。C_1 後為池化 (pooling) 層 (S_1)，進行 subsampling 動作，其 pooling 大小為長度 P_1，其跨度為 D_1。

其後有第二卷積層 (C_2)，該層有 K_2 個過濾器，過濾器大小為長度 F_2 之一維矩陣，卷積層同樣使用 padding 技巧。C_2 後為池化層 (S_2)，其 pooling 大小為 P_2，其跨度為 D_2。最後是全連結層，共 n_1 個神經元，最終輸出 n_2 個 output 的值。輸出部分採用 one-hot 模式，亦即若有 n_2 個類別，則輸出長度為 n_2 的陣列，其數值最高的類別即為分類之結果。



圖三：卷積神經網路

在兩組卷積層後以及全連結層後，本論文均採用線性整流函數 (Rectified Linear Unit, ReLU) 作為激活函數 (activation function)，以去除負數的值。為了減少過適 (overfitting) 的現象，在全連結層前加入 dropout 層，該 dropout 層除了會遮蔽神經元的輸出外，也會自動對輸入值做 scaling，dropout rate 為 DR。在最後的輸出層，我們採用 softmax 的方法。其整體的 loss function 為 cross entropy with logits，用以計算分類預測值以及分類標籤之差異，其訓練步驟為最小化 cross entropy。圖四為本論文使用之卷積神經網路之 Tensorflow 設計，程式碼之變數如同本章節前述之參數。

```
import tensorflow as tf
x = tf.placeholder(tf.float32, shape=[None, 1*n])
y_ = tf.placeholder(tf.float32, shape=[None, n2])

# First Convolutional Layer
W_conv1 = weight_variable([1, F1, 1, K1])
b_conv1 = bias_variable([K1])
x_image = tf.reshape(x, [-1, 1, n, 1])
output_conv1 = tf.nn.conv2d(x, W_conv1, strides=[1, 1, 1, 1],
padding='SAME')+b_conv1
h_conv1 = tf.nn.relu(output_conv1)
h_pool1 = tf.nn.max_pool(h_conv1, [1, 1, K1, 1], [1, 1, D1, 1], padding='SAME')

# Second Convolutional Layer
W_conv2 = weight_variable([1, F2, K1, K2])
b_conv2 = bias_variable([K2])
```

```
output_conv2 = tf.nn.conv2d(pool1, W_conv2, [1, 1, 1, 1], padding='SAME')+b_conv2
h_conv2 = tf.nn.relu(output_conv2)
h_pool2 = tf.nn.max_pool(h_conv2, [1, 1, K2, 1], [1, 1, D2, 1], padding='SAME')

# Connected Layer
s = n/D1/D2
W_fc1 = weight_variable([s * K2, n1])
b_fc1 = bias_variable([n2])
h_pool2_flat = tf.reshape(h_pool2, [-1, s*K2])
h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)

# Dropout
keep_prob = tf.placeholder(tf.float32)
h_fc1_drop = tf.nn.dropout(h_fc1, DR)

# Readout Layer
W_fc2 = weight_variable([n1, n2])
b_fc2 = bias_variable([n2])
y_conv = tf.matmul(h_fc1_drop, W_fc2) + b_fc2

# Train and Evaluate
cross_entropy = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels=y_,
logits=y_conv))
train_step = tf.train.AdamOptimizer(1e-4).minimize(cross_entropy)
correct_prediction = tf.equal(tf.argmax(y_conv, 1), tf.argmax(y_, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

圖四：Tensorflow 之卷積神經網路設計

卷積神經網路之特色在於卷積層的輸入與前一層相連，並提取前一層之特徵結果。原先卷積神經網路是適用於識別二維圖型的位移、縮放、扭曲等動作，因此本論文將片段的 Windows API 呼叫序列視為是圖型上的模式，因此可以藉由過濾器（或稱為卷積核）來自動學習特徵抽取，進行局部感知分析。本論文認為惡意程式的行為是局部性的，API 呼叫之間的順序聯繫就如同局部的圖型特徵識別，距離較近 API 呼叫應被視為一體，距離較遠的 API 呼叫則相關性較弱。因此在卷積進行的過程當中，每一個神經元沒有必要對全部的 API 呼叫序列進行感知，只要透過過濾器進行局部的感知即可，然後在更高層（池化層）將局部的訊息綜合起來，便可以得到全域的 API 呼叫樣貌。

過濾器的好處是不需要設定過濾器的內容，在圖四當中，每一個卷積層只需要指定過濾器的數目（也就是 K_1 以及 K_2）以及過濾器的大小（也就是 F_1 以及 F_2）。稍後於訓練階段，每一個過濾器會調整其矩陣值，並輸出最佳的預測結果。

惡意程式當中由於常常有程式碼使用重複或是迴圈的狀況，因此過濾器還有一個優勢，API 呼叫的部分片段可能會出現在不只一處，可能在整個側寫檔案的任一個位置，該片段有可能是駭客刻意移動位置，也可能是一個迴圈，或是一個函式的呼叫。然而過濾器在區域片段提取特徵的方式，和該片段位於整體側寫檔案的何處位置無關，這也意味著若某一過濾器對於提取特徵是有用的，那麼該過濾器可在整個側寫檔案中提取該特徵，不會只是限定在某一區域而已。若某一特徵在整個側寫檔案中出現多次，那麼每一次出現該特徵，過濾器均會起反應。

在惡意程式家族的分析上，本文認為多個過濾器可以提取不同家族的特徵行為，因此使用多個過濾器是必要的（亦即 K_1 以及 K_2 應大於 1）。而過濾器的大小必須要與惡意程式 API 呼叫片段長度相當，也就是說資安分析專家應該設定一個片段的大小，可以表示一個程式使用 API 呼叫時，足以判斷一個區域小行為的長度。此長度不應過長，否則會囊括太多行為在過濾器當中，有可能會導致過濾器無法針對特徵起反應。同樣的道理，在進行池化時，跨度的長短也應該與一個區域小行為的 API 呼叫長度相當。

4.2 訓練環節

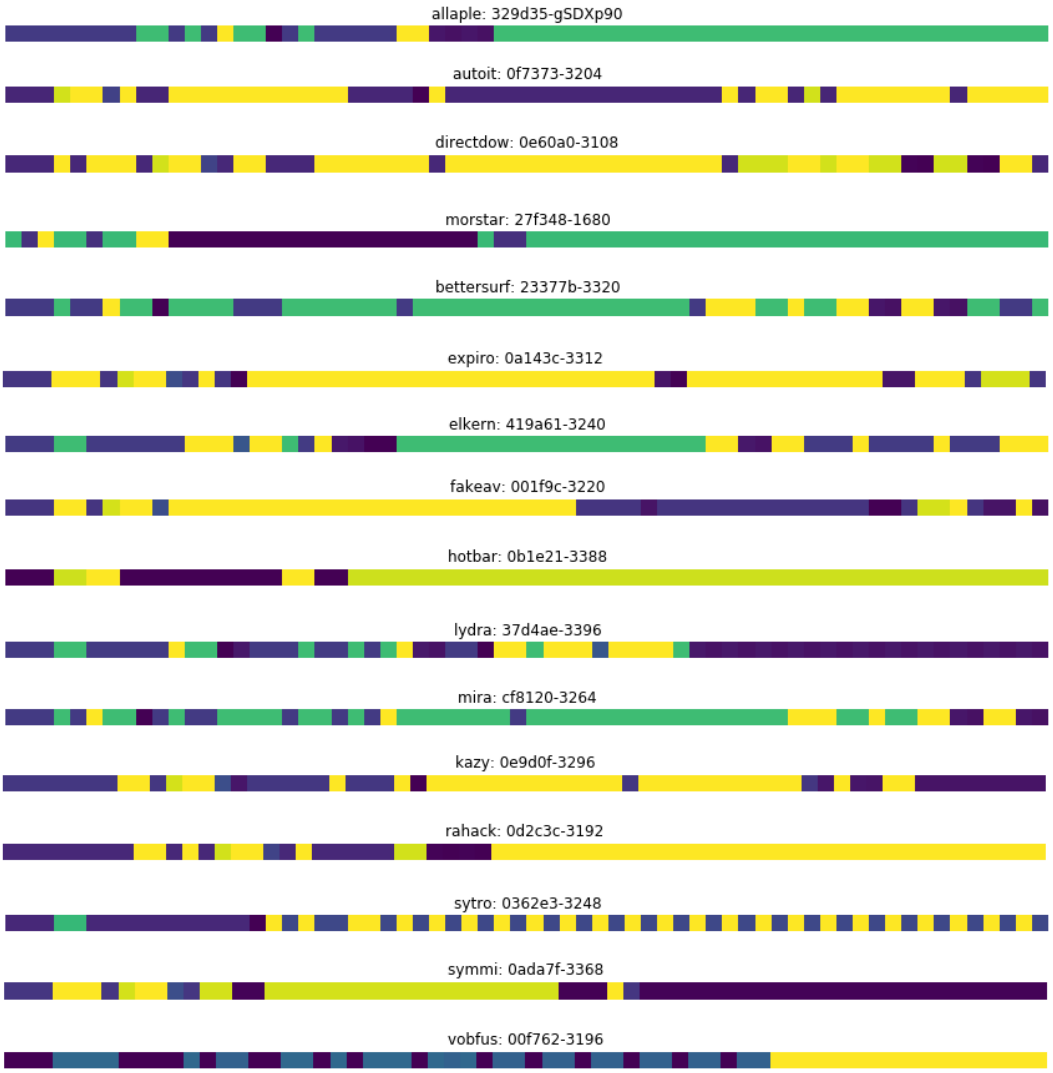
訓練環節將採取標準之 10-fold 測試，共有 10%的資料被留作測試使用，測試資料並不參與訓練。假設訓練資料共 TN 筆，測試資料共 TT 筆。訓練環節有一特殊的設計，TN 筆訓練資料一開始先排序過，在每一輪 (epoch) 的訓練程序中，會從排序的訓練資料中依序取出 batch_size 個訓練資料，並使用於卷積神經網路之訓練。下一輪的訓練將再取出下 batch_size 個訓練資料，直至 TN 筆資料被使用完畢為止。若第 e 輪訓練時，剩下的訓練資料數小於 batch_size，則對 TN 筆訓練資料重新做亂序的排序，並重新自第一筆訓練資料開始取出 batch_size 個訓練資料進行第 e 輪訓練。依照相同規則，第 (e+1) 輪的訓練將繼續使用該亂序之下 batch_size 筆訓練資料，直至使用完所有亂序資料為止。而每次用罄 TN 筆訓練資料，均重新對 TN 筆資料進行亂序排序。以上之設計可以逐步針對訓練資料進行卷積神經網路權重之調整，並適用於少量資料的訓練。

伍、實驗

5.1 實驗資料集與分析環境

本論文的惡意程式樣本由網路蒐集取得，由於本論文之分析資料為動態行為分析，因此必須蒐集可執行檔案，並於所設計之動態分析側寫系統中取得動態側寫檔案。由於

網路蒐集之惡意程式樣本並無家族分類之資訊，因此所蒐集之惡意程式雜湊值會被上傳至 VirusTotal 網站用來擷取各防毒軟體偵測引擎之偵測結果。本論文採取較嚴謹之家族分類標記，若某一惡意程式樣本其各防毒軟體偵測引擎之測試結果中（例如：Win32.CryptDoma.dc），有二分之一以上的偵測引擎結果均出現某子字串（以前例來說應為 CryptDoma 字串），則本論文將該惡意程式樣本的家族標記為該子字串。而常見之非家族名稱之子字串將事先被剃除，例如：Trojan、Win32、Adware、Tool。



圖五：家族樣本之視覺化 Windows API 呼叫序列

根據以上之資料蒐集方法，本實驗共取得 36 個家族分類，共 850 個惡意程式樣本。圖五中為其中 16 個家族樣本之圖像化結果（礙於空間並不呈現全部家族之樣本圖形），每一個家族樣本只顯示前 64 個 Windows API 呼叫序列。由圖五可知每一家族的 Windows API 呼叫序列之樣態均 (pattern) 不同，因此卷積神經網路應能學習每一家族之分類特

徵。圖五之標記為每一家族之名稱、該家族之某樣本編號（為其樣本雜湊值十六進位之前六碼數值)以及該樣本於執行時期之 ID 或執行檔名稱。例如：allaple: 329d35-gSDXp90 表示其下之圖形為 allaple 家族中，雜湊值前六碼編號為 329d35 之樣本，其側寫時使用之執行檔名為 gSDXp90。

本論文使用之惡意程式檔案均為 Windows 32 位元之 PE 可執行檔案，所執行側寫之環境為 Windows XP SP2 (32-bit) 作業系統。側寫時間為每個惡意程式樣本 300 秒，所產生之惡意程式動態分析側寫檔案共 850 個。平均每一側寫檔案包含 259 個 Windows API 呼叫序列，最長序列長度為 1,650，最短序列長度為 20。

卷積神經網路使用 Python 3.6 撰寫，使用 Tensorflow 套件，運算使用 NVIDIA GeForce GTX 1080 Ti 顯示卡進行卷積神經網路分析。由於側寫檔案數量並不多，利用 GPU 執行一次卷積神經網路分析的時間在 30 秒之內。

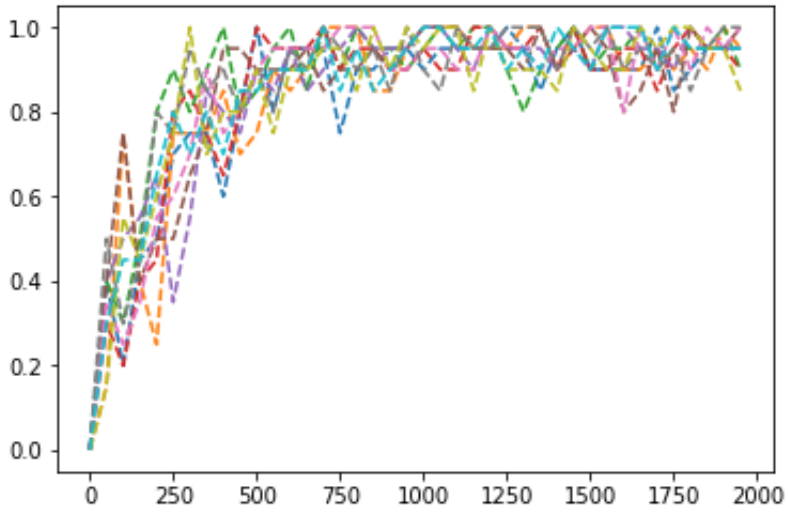
5.2 實驗結果

以下表三為卷積神經網路使用之參數，其參數已為適配之結果。本論文之目的在於討論動態分析資料使用於卷積神經網路之關係，並非針對卷積神經網路調整策略討論。

表三：卷積神經網路使用之參數

參數	參數值	說明
test_rate	0.1	測試樣本數占所有樣本數的比例。
n	64	樣本長度，Windows API 呼叫序列長度。
K_1	32	第一層過濾器（卷積核）個數
F_1, F_2	4, 4	過濾器（卷積核）長度
P_1, P_2	4	池化長度
D_1, D_2	4, 4	池化層之跨度長度
K_2	64	第二層過濾器（卷積核）個數
n_1	1024	全連結層節點數
DR	0.5	Dropout rate
batch_size	20	每輪訓練使用的訓練資料個數

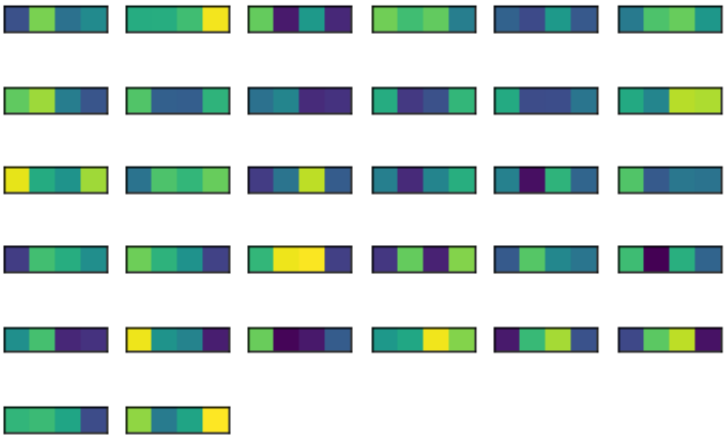
圖六為資料集之卷積神經網路學習之歷程，共顯示十次不同的 10-fold 資料其個別學習歷程。由圖可知以上的參數設定可以在約 500 輪學習之後（每輪 20 個訓練資料），達到平均 90%以上的訓練準確率。而該網路之最終平均測試準確率為 99.5%。



圖六：卷積神經網路學習之歷程

圖七為卷積神經網路第一層之過濾器（卷積核）視覺化之結果，可由過濾器之樣態得知，卷積神經網路根據學習資料的訓練可以自動得出不同樣態的過濾器，而每一個過濾器實際上可以觸發不同的 Windows API 呼叫序列，並傳遞至下一層。舉例來說：圖七過濾器的視覺化樣態中，比較常出現不同綠色的連續區塊，較少出現連續的黃色區塊。利用表二可將樣態解譯回 Windows API，連續的黃色區塊表示過濾器會針對連續的 RegQueryValue 或是 CreateFile 呼叫起反應，並傳遞至下一層。而根據事後的分析，RegQueryValue 為 Windows 程式中常見的呼叫，但是在分析時若不考慮其參數值，即使是有經驗的資訊安全分析師也無法判別連續的 RegQueryValue 呼叫是否為惡意家族行為。在觀察側寫檔案後可以發現，各家族的樣本均有連續呼叫 RegQueryValue 的現象，而這應被視為是行為的雜訊，該行為不足以分辨不同的惡意程式家族。因此在過濾器中並沒有看到這樣的樣態被訓練出來。

被挑選出來的每一個過濾器應可以表示一個 Windows API 呼叫的片段行為（長度為 K1），以第一個過濾器的第一個值為例，其值為 0.0795，這個過濾器會與 LoadLibrary 這一個 API 起反應，以此類推可得到過濾器對 API 序列所反應的樣態。



圖七：卷積神經網路第一層之過濾器（卷積核）

陸、結論

本論文的目的是藉助卷積神經網路對惡意程式進行家族進行自動分類並產生行為特徵，與其他過去的研究不同，本論文先對惡意程式進行動態側寫分析並產出其高階的 Windows API 呼叫序列紀錄，而卷積神經網路將視 Windows API 呼叫序列為輸入資料並最終輸出惡意程式家族分類的結果。本論文利用卷積神經網路的學習結果來解釋其惡意程式之特徵行為。在實驗上我們採用國網中心以及資策會於真實世界蒐集的惡意程式，進行動態分析側寫後進行監督式的訓練以及驗證，其家族分類準確率超過 99%。我們的實驗並證明可以使用有限的 Windows API 呼叫序列（64 個）就能進行正確的家族分類，如此我們的研究成果可以進一步導入至入侵防禦系統，進行早期的入侵偵測。

[誌謝]

本論文之部分惡意程式樣本由「財團法人資訊工業策進會」之「智慧型資安與新興應用整合技術研發計畫」支持；部分惡意程式樣本由「國家實驗研究院高速網路與計算中心」之「惡意程式知識庫」支持。本論文受「科技部」106-2218-E-004-001 之計畫補助。

參考文獻

[1] U. Bayer, P. M. Comparetti, C. Hlauschek, C. Kruegel and E. Kirda, “Scalable, Behavior-

Based Malware Clustering," *in Proc. Network and Distributed System Security Symposium (NDSS)*, pp. 8-11, 2009.

[2] U. Bayer, C. Kruegel, and E. Kirda, "TTAnalyze: A Tool for Analyzing Malware," *in Proc. European Institute for Computer Antivirus Research (EICAR 2006) Annual Conference*, April 2006.

[3] D. Billar, "Opcodes as predictor for malware," *International Journal of Electronic Security and Digital Forensics*, vol. 1, pp. 156–168, 2007.

[4] P. M. Chen and B. D. Noble, "When virtual is better than real," *in Proc. of 8th Workshop on Hot Topics in Operating Systems (HotOS)*, pp. 133-138, May 2001.

[5] A. Dinaburg, P. Royal, M. Sharif and W. Lee, "Ether: Malware Analysis via Hardware Virtualization Extensions," *in Proc. of ACM Conference on Computer and Communications Security*, pp. 51–62, 2008.

[6] G. W. Dunlap, S. T. King, S. Cinar, M. A. Basrai and P. M. Chen, "ReVirt: Enabling Intrusion Analysis through Virtual-Machine Logging and Replay," *in Proc. of USENIX Symposium on Operating Systems Design and Implementation*, pp. 211-224, 2002.

[7] S. Forrest, S. A. Hofmeyr, A. Somayaji and T. A. Longstaff, "A Sense of Self for Unix Processes," *in Proc. of IEEE Symposium on Security and Privacy (S&P)*, pp. 120-128, May 1996.

[8] T. Garfinkel and M. Rosenblum, "A Virtual Machine Introspection Based Architecture for Intrusion Detection," *in Proc. of NDSS*, pp. 191-206, 2003.

[9] D. Gibert, "Convolutional Neural Networks for Malware Classification," MS thesis. Universitat Politècnica de Catalunya, 2016.

[10] S. A. Hofmeyr, S. Forrest and A. Somayaji, "Intrusion Detection using Sequences of System Calls," *Journal of Computer Security*, vol. 6, pp. 155-180, 1998.

[11] S.-W. Hsiao, Y.-N. Chen, Y. S. Sun and M. C. Chen, "A Cooperative Botnet Profiling and Detection in Virtualized Environment," *in Proc. of IEEE Conference on Communications and Network Security (IEEE CNS)*, pp. 154-162, Oct. 2013.

[12] X. Jiang, X. Wang and D. Xu, "Stealthy Malware Detection through VMM-based 'out-of-the-box' Semantic View Reconstruction," *in Proc. of ACM CCS*, pp. 128-138, 2007.

[13] C. Kruegel, D. Mutz, F. Valeur and G. Vigna, "On the Detection of Anomalous System Call Arguments," *in Proc. of European Symposium on Research in Computer Security*, pp. 101-118, 2003.

[14] W. Lee and S. J. Stolfo, "Data Mining Approaches for Intrusion Detection," *in Proc. of USENIX Security Symposium*, pp. 79-93, July 1998.

[15] L. Liu, S. Chen, G. Yan and Z. Zhang, "BotTracer: Execution-Based Bot-Like Malware Detection," *in Proc. of Int. Conf. on Information Security (ISC)*, pp. 97-113, 2008.

[16] L. Nataraj, S. Karthikeyan, G. Jacob and B. S. Manjunath, "Malware images: Visualization and automatic classification," *in Proc. of International symposium on Visualization for Cyber Security*, 2011.

[17] C. Ravi and R. Manoharan, "Malware detection using windows api sequence and machine learning," *International Journal of Computer Applications*, vol. 43, 2012.

[18] J. Saxeand and K. Berlin, "Deep neural network based malware detection using two dimensional binary program features," *in Proc. of MALWARE*, pp. 11-20, 2015.

[19] D. Song, D. Brumley, H. Yin, J. Caballero, I. Jager, M. G. Kang, Z. Liang, J. Newsome, P. Poosankam and P. Saxena, "BitBlaze: A New Approach to Computer Security via Binary Analysis," *in Proc. of International Conference on Information Systems Security*, pp. 1-25, 2008.

[20] G. J. Tesauro, J. O. Kephart and G. B. Sorkin, "Neural networks for computer virus recognition," *IEEE expert*, vol. 11, 1996.

[21] R. Veeramani and N. Rai, "Windows api based malware detection and framework analysis," *International Journal of Scientific & Engineering Research*, vol. 3, 2012.

[22] D. Wagner and D. Dean, "Intrusion Detection via Static Analysis," *in Proc. of IEEE Symposium on Security and Privacy (IEEE S&P)*, pp. 156-168, May 2001.

[23] C. Willems, T. Holz and F. Freiling, "Toward Automated Dynamic Malware Analysis Using CWSandbox," *IEEE Security & Privacy*, vol. 5, no. 2, pp. 32--39, 2007.

[24] H. Yin, D. Song, M. Egele, C. Kruegel and E. Kirda, "Panorama: Capturing System-Wide Information Flow for Malware Detection and Analysis," *in Proc. of ACM Conference on Computer and Communications Security (ACM CCS)*, pp. 116-127, 2007.

[25] https://www.kaggle.com/c/malware-classification.

[26] https://www.tensorflow.org/get_started/mnist/pros

[作者簡介]

蕭舜文博士於國立臺灣大學資訊管理學系所學士(2004 年)與博士畢業(2012 年)。自 2006 年自 2008 年參與 iCAST 合作計畫於美國卡內基美隆大學 (CMU) 之 CyLab 進行研究工作,自 2012 年 7 月加入中央研究院資訊科學研究所擔任博士後研究員。於 2017 年 2 月加入國立政治大學資訊管理學系擔任助理教授,其主要研究專長為網路安全、惡意程式行為分析、電腦虛擬化技術、資料科學、區塊鏈與智能合約。