
UNIVERSITÀ DEGLI STUDI DI SALERNO

Dipartimento di Informatica

Corso di Laurea Magistrale in Sicurezza Informatica

# MALWARE CLASSIFICATION USING API CALL AND CONVOLUTIONAL NEURAL NETWORK

Docenti di riferimento:

Prof. Arcangelo Castiglione

Prof. Gianni D'Angelo

Candidato:

Aniello Giugliano

Anno Accademico 2019-2020

# INDICE DEI CONTENUTI

# INDICE DELLE FIGURE

# ABSTRACT

Con l'aumento delle dimensioni di Internet, i malware si sono trasformati in una delle principali minacce ai computer e ai sistemi informativi in tutto il mondo. Attualmente la maggior parte dei sistemi di rilevamento dei virus sono basati sulle signature , il che significa che cercano di identificare il malware in base ad una singola funzionalità. Il principale svantaggio di tali sistemi di rilevamento è che non possono rilevare malware sconosciuti (0-day) o polimorfici, ma identificano solo le varianti di malware già identificate in precedenza. Ecco perché metodi di detection basati sul machine-learning stanno aumentando sempre di più. Lo scopo di questo lavoro di tesi è stato quello di creare un metodo di detection e classificazione per i malware efficace, basato su alcune caratteristiche estratte dai malware in fasi di analisi dinamica,eseguita utilizzando una Sandbox. L'implementazione di questa tecnica di classificazione è stata fatta tramite tecniche di Machine Learning, in particolare utilizzando la Convolutional Neural Network (CNN).

# INTRODUZIONE

Con il rapido sviluppo di Internet, i malware sono diventati una delle principali minacce informatiche al giorno d'oggi. Qualsiasi software che esegue azioni dannose, inclusi furto di informazioni, spionaggio ecc, può essere definito malware. Kaspersky Labs definisce il malware come "un tipo di programma per computer progettato per infettare il computer di un utente legittimo e infliggergli danni in più modi"( Kaspersky Labs 2017). Mentre la diversità del malware è in aumento, gli scanner antivirus non possono soddisfare le esigenze di protezione, con il risultato che milioni di host vengono attaccati. Secondo Kaspersky Labs, 7 milioni di host diversi sono stati attaccati e nel 2015 sono stati identificati quattro milioni di malware diversi (Kaspersky Labs 2016). Le violazioni dei dati e gli incidenti di sicurezza stanno diventando sempre più costosi. Il costo medio di una violazione dei dati nel 2020 è di 3,86 milioni di dollari, secondo un nuovo rapporto di IBM e Ponemon Institute . Sebbene il costo medio di una violazione sia relativamente invariato, IBM afferma che i costi stanno diminuendo per le aziende preparate e diventano molto più grandi per quelle che non prendono precauzioni. Inoltre, c'è una diminuzione del livello di abilità richiesto per lo sviluppo di malware, a causa dell'elevata disponibilità di strumenti di attacco su Internet al giorno d'oggi. L'elevata disponibilità di tecniche anti-rilevamento, nonché la capacità di acquistare malware sul mercato nero, offrono l'opportunità di diventare un attaccante per chiunque, indipendentemente dal livello di abilità. Gli studi attuali mostrano che sempre più attacchi vengono lanciati da script kiddies o sono automatizzati. Pertanto, la protezione dai malware dei sistemi informatici è una delle attività di sicurezza informatica più importante per i singoli utenti e per le aziende, poiché anche un singolo attacco può causare gravi perdite sia economiche che di immagine. Attacchi frequenti e perdite massicce impongono la necessità di metodi di rilevamento accurati e tempestivi. Gli attuali metodi statici non forniscono un rilevamento efficiente, soprattutto quando si tratta di attacchi zero-day e virus polimorfi. Per questo motivo, è possibile utilizzare tecniche basate sul machine-learning per il rilevamento di questi malware più complessi.

## Obbiettivo della Tesi

Questo lavoro di tesi discute i punti e le preoccupazioni principali del rilevamento e della categorizzazione dei malware basato sul machine-learning, oltre a cercare i migliori metodi di rappresentazione e classificazione delle funzionalità. Lo *scopo principale* di questo lavoro è quello di creare un metodo di detection e classificazione di malware efficace utilizzando la Convolutional Neural Network. I malware vengono analizzati utilizzando una sandbox(CuckooDroid Sandbox) e ne vengono estratte alcune feature. In particolare le feature a cui siamo interessati sono le api-call sequenziali,tramite le quali creeremo delle matrici(api-image) che passeremo successivamente alla rete per addestrarla. L'obbiettivo in particolare è quello di categorizzare 5 famiglie di malware diverse (BaseBridge,OpFake, DroidKungFu,FakeInstaller,Plankton).

## Struttura della Tesi

Nel *Capitolo 1* viene fatta una breve panoramica dell'analisi dei malware, esponendo i concetti generali di analisi statica e dinamica, e vedendone le differenze principali nei due approcci. Nel *Capitolo 2* viene data una definizione generale di Sandbox e virtualizzazione e poi viene approdondito nella specifico l'architettura utilizzata (Cuckoo Sandbox). Nel *Capitolo 3* viene data un overview sui concetti principali del Machine Learning che poi servirà nel capitolo successivo per implementare la soluzione proposta. In particolare,nel *Capitolo 4*, viene spiegato passo per passo tutta l'attività progettuale svolta, dall'inizio alla fine, viene mostrato l' ambiente di sviluppo, il dataset utilizzato, il processo per l'estrazione delle feature, la creazione dell'input per la CNN, un breve approfondimento sulle funzioni principali della CNN. Il training ed il testing del modello vengono mostrati nel *Capitolo 5* ,dove vengono anche mostrati i risultati ottenuti.Nel *Capitolo 6* viene dato uno spunto su quali possono essere gli sviluppi futuri di questo progetto. Nell'*Appendice A* viene riportato il codice completo per la creazione della rete CNN.

# CAPITOLO 1
# MALWARE ANALYSIS

I criminali informatici stanno diventando più sofisticati e innovativi, stanno emergendo varietà nuove ed avanzate di malware ed il rilevamento del malware si sta rivelando una vera sfida. L'analisi del malware, che comporta l'analisi dell'origine (source code), delle funzionalità e del potenziale impatto di qualsiasi campione del malware, è di fondamentale importanza per quanto riguarda la sicurezza informatica nel mondo moderno. I professionisti della sicurezza si affidano all'analisi del malware per vari scopi. Potrebbero usarlo per valutare l'entità dell'infezione ogni volta che si verifica un attacco o per identificare la natura del malware coinvolto. Allo stesso modo, una corretta comprensione delle funzionalità e dell'impatto di qualsiasi campione di malware li aiuta ad affrontare gli attacchi informatici in modo migliore. Esistono due diversi tipi di analisi del malware, vale a dire *l'analisi statica* e quella *dinamica*. L'analisi statica implica tutti quegli esami del malware in cui non si esegue effettivamente il programma malevolo ma si cerca di capire cosa sta cercando di fare.
L'analisi dinamica si effettua eseguendo realmente il malware in ambiente virtualizzato (Sandbox) e poi si cerca di capire le varie funzionalità di quest'ultimo.

## 1.1 Definizione e tipi di Malware

Per definizione, il malware è l'abbreviazione di "malicious software", cioè software malevolo, progettato e creato per danneggiare un dispositivo o il suo proprietario. Si tratta di un termine generico utilizzato per classificare i file o i software che danneggiano un sistema (una volta che si trovano al suo interno) in vari modi, ad esempio rubando i dati dal computer dell'utente, crittografandoli o semplicemente eliminandoli. Un malware è anche in grado di modificare le funzioni all'interno di un computer o di prenderne completamente il controllo, come nel caso di botnet e rootkit. Esistono molti tipi di malware, che si differenziano in base alle caratteristiche o alla modalità operativa. Tra questi:

- **Virus informatico**: è la forma di malware più classica. Una volta infettato il dispositivo dell'utente, può danneggiarlo in vari modi: può rallentare il sistema,

disattivarne parti specifiche fino ad arrivare al controllo completo. Proprio come per i virus biologici, è progettato affinché continui a diffondersi automaticamente su reti e dispositivi.

- **Spyware**: è un malware progettato per raccogliere dati da un computer e dai suoi utenti infiltrandosi nel computer dell'utente e monitorandone l'attività. Viene installato sul computer dell'utente in maniera diretta o sfruttando eventuali vulnerabilità della vittima.
- **Ransomware**: proprio come suggerisce il nome ("ransom" significa riscatto in inglese), un ransomware è un software creato allo scopo di prendere in ostaggio i dati sul computer dell'utente. Tale software esegue la crittografia di dati sensibili e mirati; in seguito, i suoi creatori richiedono denaro all'utente per decifrare i dati.
- **Trojan**: è un tipo di malware che appare come un normalissimo programma, quindi saranno proprio gli utenti inconsapevoli a installarlo sui loro computer. Una volta installato ed eseguito, il trojan può avviare la funzione malevola per la quale è stato pensato. A differenza dei virus e dei worm, raramente i trojan cercano di replicarsi e di diffondersi.
- **Rootkit**: questo tipo di malware garantisce ai criminali informatici l'autorizzazione a livello di amministratore sul computer di un obiettivo, così possono modificarne il sistema operativo. Inoltre, è in grado di nascondere la presenza di altri malware all'interno del sistema informatico.
- **Backdoor**: questo tipo di malware crea una "backdoor" nel computer di una vittima, attraverso la quale un attaccante è in grado di accedere al pc di quest'ultima da remoto.

## 1.2 Analisi Statica

L'analisi statica del malware implica l'esame di un dato campione di malware senza eseguire effettivamente il codice. Questo di solito viene fatto determinando la firma del binario del malware; la firma è un'identificazione univoca per il file binario. Il calcolo dell'hash crittografico del file e la comprensione di ciascuno dei suoi componenti aiuta a determinarne la firma. L'eseguibile del file binario del malware viene caricato in un disassembler (ad esempio IDA) e quindi il codice dalla macchina viene convertito in codice in linguaggio

assembly. Pertanto, eseguendo questo reverse engineering su un malware, è reso facile da leggere e comprendere per una persona. L'analista, osservando il codice del linguaggio assembly, riesce a comprendere meglio il malware,quindi si fa un'idea migliore sulle funzionalità che è programmato per fare e sul potenziale impatto che può avere su qualsiasi sistema e rete. Gli analisti utilizzano diverse tecniche per l'analisi statica,che includono il fingerprinting dei file,scansione antivirus , dumping della memoria, rilevamento packer e debug.

## 1.3 Analisi Dinamica

L'analisi dinamica del malware implica l'analisi durante l'esecuzione del codice in un ambiente controllato. Il malware viene eseguito in un ambiente virtuale chiuso e isolato(sandbox) e quindi viene studiato il suo comportamento. L'intenzione è di comprenderne il funzionamento e il comportamento e utilizzare questa conoscenza per fermare la sua diffusione o per rimuovere l'infezione. I debugger vengono utilizzati, nell'analisi avanzata del malware dinamico, per determinare la funzionalità dell'eseguibile del malware.

## 1.4 Differenze tra analisi statica e dinamica

- **Analisi**: L'analisi statica del malware è un modo abbastanza semplice e diretto per analizzare un campione di malware senza eseguirlo effettivamente, quindi il processo non richiede all'analista di passare attraverso ogni fase. Osserva semplicemente il comportamento del malware per determinare di cosa è capace o cosa può fare al sistema. L'analisi dinamica del malware, d'altra parte, implica un'analisi approfondita che utilizza il comportamento e le azioni del campione di malware durante l'esecuzione per avere una migliore comprensione del campione. Il sistema è configurato in un ambiente chiuso e isolato con un monitoraggio adeguato.

- **Approccio**: L'analisi statica utilizza un approccio basato sulla firma per il rilevamento e l'analisi del malware. Una firma non è altro che un identificatore univoco per un

malware specifico che è una sequenza di byte. Diversi modelli vengono utilizzati per la scansione delle firme. I programmi antimalware basati su firme sono efficaci contro i tipi più comuni di malware, ma sono inefficaci contro i programmi malware sofisticati e avanzati. È qui che entra in gioco l'analisi dinamica. Invece di un approccio basato sulla firma, l'analisi dinamica utilizza un approccio basato sul comportamento per determinare la funzionalità del malware studiando le azioni eseguite dal malware in questione.

Nella *Figura 1* sono mostrate le due tecniche di analisi statica e dinamica con le loro rispettive sottocategorie.

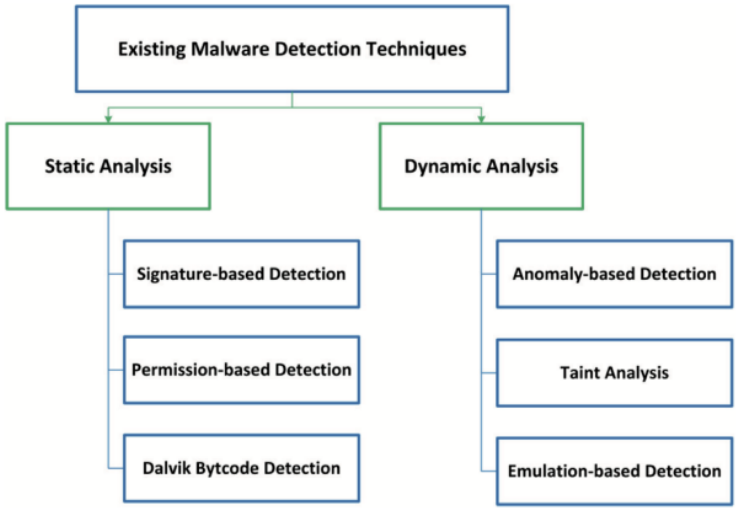



*Figura 1 Analisi Statica e Dinamica dei Malware*

# CAPITOLO 2
# CUCKOO SANDBOX

## 2.1 Definizione di Sandbox

Una sandbox è un meccanismo per eseguire applicazioni in uno spazio limitato. Solitamente fornisce un ristretto e controllato set di risorse al programma che deve essere testato, come un'area ristretta di memoria o un insieme di chiamate di sistema limitate. Di norma, l'accesso alla rete, la possibilità di ispezionare il sistema ospite o leggere dai dispositivi di input, sono disabilitati o altamente ristretti. Data la capacità di fornire un ambiente estremamente controllato, le sandbox possono essere viste come uno esempio specifico di virtualizzazione, solitamente utilizzata per eseguire programmi non testati o non attendibili, non verificati o provenienti da terze parti non riconosciute (come utenti o siti web), senza rischiare di infettare il dispositivo dove viene eseguita l'applicazione, ad esempio effettuare test su programmi non verificati che possono contenere virus o codice maligno, senza permettere al software di infettare il dispositivo ospite. Nella *Figura 2* viene mostrata l'architettura di una sandbox.

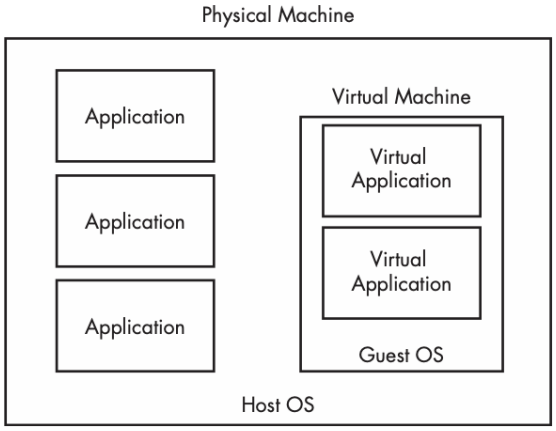



*Figura 2 Sandbox*

## 2.2 Vantaggi nell'utilizzo di una Sandbox

Esistono vari motivi per utilizzare una macchina virtuale come ambiente sandbox per testare il malware anziché utilizzare un sistema reale. Uno dei motivi chiari è che l'utilizzo di macchine reali è troppo costoso se confrontato con una macchina virtuale. L'altro motivo più evidente per utilizzare una macchina virtuale è che quando un malware infetta una macchina virtuale e va fuori controllo, possiamo semplicemente eliminare il malware e ricominciare tutto da capo, ma non è così con macchine reali poiché il malware può ancora persistere nelle macchine su cui è stato eseguito e rimuovere il malware dal sistema al 100% potrebbe essere abbastanza complesso. Le macchine virtuali inoltre forniscono la funzione di "snapshot" che è un enorme vantaggio in quanto permettono di ripristinare il sistema ad uno stato precedente.

## 2.3 Svantaggi nell'utilizzo di una Sandbox

Anche se utilizzare una Sandbox per l'analisi dei malware è molto vantaggioso, c'è un problema, cioè che ogni volta che si eseguono malware in una macchina virtuale ci sono delle possibilità che il malware rilevi che viene eseguito all'interno di un ambiente virtualizzato e non esegua tutte le azioni che eseguirebbe se fosse in un sistema reale. Il malware è in grado di rilevare questo osservando le seguenti caratteristiche della macchina:

- **Quantità di spazio di archiviazione sul disco rigido**. Ad esempio, una macchina normale avrebbe almeno 256 GB di spazio di archiviazione in SSD o circa 512 GB se HDD, ma se il malware rileva che l'archiviazione su disco rigido è di soli 20 GB o giù di lì, capirà di essere eseguito in un ambiente virtuale.Per risolvere questo problema è necessario assegnare alla macchina virtuale una quantità di memoria piu grande,ad esempio 100 gb,per evitare che il malware noti questa differenza con una macchina reale.
- **Test per la connettività ad internet**. Le macchine virtuali in cui testiamo il malware vengono tenute disconnesse da Internet poiché non sappiamo cosa potrebbe tentare di fare il malware se rileva una connessione Internet attiva. Potrebbe provare a propagarsi ad altre macchine tramite Internet ed ovviamente non possiamo consentirlo. La maggior parte delle volte,quindi, quando eseguiamo il malware, manteniamo la macchina virtuale in una rete interna. La soluzione a questo problema è l'uso di strumenti come ApateDNS o Netcat. Questi strumenti vengono utilizzati per impostare

una connessione Internet falsa in modo che se il malware cerca di capire se la macchina ha una connessione Internet ,considererà che c'è una connessione anche se in realtà non esiste.

## 2.4 Introduzione a Cuckoo

Cuckoo è un sistema di analisi del malware automatizzato open source. Viene utilizzato per eseguire ed analizzare automaticamente i file e raccogliere risultati di analisi completi che descrivono ciò che fa il malware durante l'esecuzione all'interno di un sistema operativo isolato. Può recuperare i seguenti tipi di risultati:

- Chiamate API eseguite da tutti i processi generati dal malware.
- File creati, eliminati e scaricati dal malware durante la sua esecuzione.
- Dump della memoria dei processi malware.
- Traccia del traffico di rete in formato PCAP.
- Screenshot acquisiti durante l'esecuzione del malware.
- Dump di memoria completi delle macchine.

Cuckoo è un progetto che è stato avviato per automatizzare e analizzare rapidamente il codice dannoso.Cuckoo Sandbox è stato sviluppato nel 2010 e CuckooDroid, una sandbox per l'analisi dei malware Android, è stato sviluppato nel 2012.

## 2.5 Cuckoo Droid

CuckooDroid è un SO guest, un emulatore Android che permette di vedere il comportamento dei malware in esecuzione . Le app dannose possono rilevare varie azioni come l'estrazione della chiave di crittografia, l'ispezione SSL, la traccia delle chiamate API e la firma. Il comportamento dannoso di un'app può essere compreso dal risultato dell'analisi. CuckooDroid supporta l'analisi tramite vari gestori di virtualizzazione tra cui KVM, Xen, VirtualBox e VMware.

Le capacità di analisi di CuckooDroid sono suddivise in analisi statica, dinamica e del traffico a seconda del metodo. Secondo la classificazione dell'analisi, le informazioni vengono raccolte in base a ciascuna funzione. L'analisi statica fornisce dati per le chiamate API statiche tramite la raccolta della struttura dei dati tramite la struttura dei file e la decompilazione dell'applicazione. Fornisce inoltre informazioni generali sul file Android e fornisce componenti generali per l'applicazione come Attività o Servizio. L'analisi dinamica fornisce i risultati monitorati tramite il modulo Droidmon. Droidmon è un modulo di monitoraggio che collega le chiamate API Dalvik per estrarre le informazioni sul comportamento di un'app. Se l'app è dannosa, stampa l'output effettivo o il percorso del file a cui si accede. Inoltre, se il contenuto crittografato è presente, la chiave di crittografia e il contenuto crittografato vengono emessi come testo normale. Fornisce quindi uno screenshot del movimento complessivo dell'app per mostrare le azioni e gli impatti che possono verificarsi sui dispositivi reali. Fornisce inoltre analisi del traffico di rete. Analizza il protocollo di comunicazione tramite l'output dell'URL dell'app e restituisce le informazioni specifiche del protocollo. Infine, tramite l'analisi del comportamento, i risultati dell'analisi precedente vengono ordinati per stringa relativa. Estrae le impostazioni basate su Androguard o le classifica in base alle firme associate utilizzando le informazioni raccolte dalle app in esecuzione per l'estrazione dinamica.

## 2.6 Architettura di Cuckoo

Ogni analisi viene avviata in una macchina virtuale nuova e isolata. L'infrastruttura di Cuckoo è composta da una macchina host (il software di gestione) e da una serie di macchine ospiti (macchine virtuali che eseguono l'analisi). L'host esegue il componente principale della sandbox che gestisce l'intero processo di analisi, mentre i guest sono gli ambienti isolati in cui i campioni di malware vengono eseguiti in modo sicuro e quindi analizzati. Ogni guest è composto da una macchina virtuale Linux che esegue l'emulatore Android, che è controllato dal modulo della macchina. Nella *Figura 3* viene mostrata l'architettura di Cuckoo Droid.

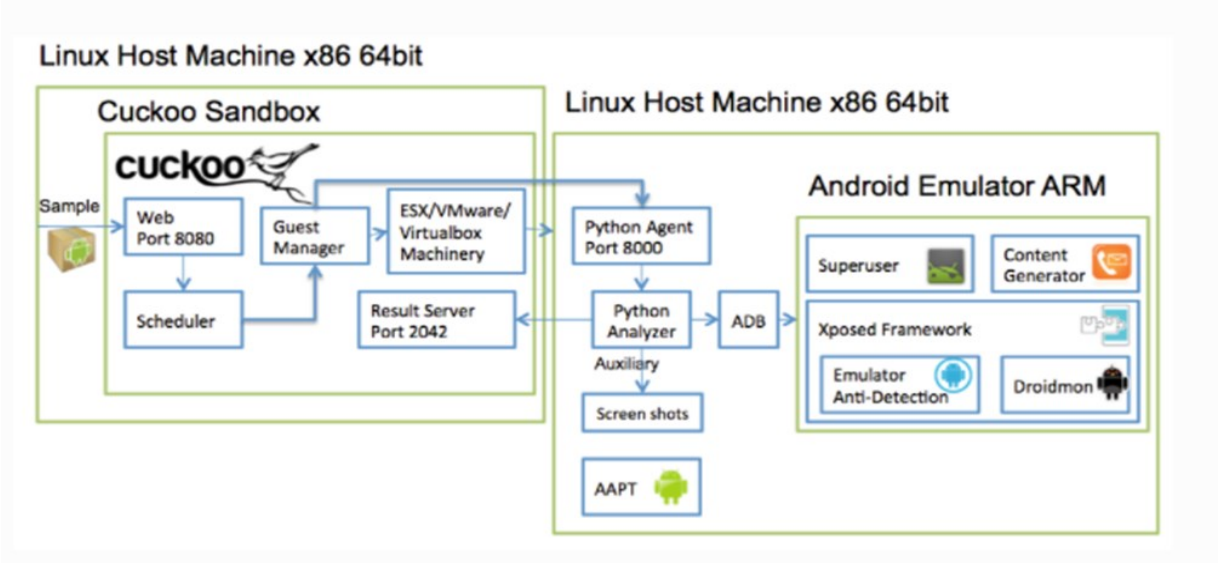



*Figura 3 Architettura Cuckoo Sandbox*

# CAPITOLO 3
# MACHINE LEARNING

Il Machine Learning (ML) è definito come l'uso di algoritmi e statistiche computazionali per apprendere dai dati senza essere programmati esplicitamente. È una sottosezione del dominio dell'intelligenza artificiale (AI). Sebbene il campo dell'apprendimento automatico non sia esploso fino a tempi più recenti, il termine è stato coniato per la prima volta nel 1959 e molti studi sono stati condotti tra il 1970 ed il 1980. L'ascesa alla ribalta del machine learning oggi è stata resa possibile dall'abbondanza di dati, da un'archiviazione dei dati più efficiente e da calcolatori più veloci. A seconda di ciò che si sta tentando di ottenere, esistono molti modi diversi per fare in modo che un computer apprenda dai dati. Questi vari modi possono essere classificati in tre sottosezioni principali dell'apprendimento automatico: *apprendimento supervisionato*, *apprendimento non supervisionato* e *apprendimento di rinforzo*.

## 3.1 Apprendimento Supervisionato

L'apprendimento supervisionato è un approccio alla creazione di intelligenza artificiale (AI), in cui al programma vengono forniti dati di input etichettati e i risultati di output attesi. Al sistema di intelligenza artificiale viene detto in modo specifico cosa cercare, quindi il modello viene addestrato fino a quando non è in grado di rilevare i modelli e le relazioni sottostanti, consentendogli di produrre buoni risultati quando vengono presentati dati mai visti prima. Come tutti gli algoritmi di apprendimento automatico, l'apprendimento supervisionato si basa sulla formazione. Il sistema viene alimentato con enormi quantità di dati durante la sua fase di addestramento, che istruiscono il sistema su quale output deve essere ottenuto da ogni specifico valore di input. Il modello addestrato viene quindi presentato con i dati del test per verificare il risultato dell'addestramento e misurare l'accuratezza. Negli algoritmi di rete neurale, il processo di apprendimento supervisionato viene migliorato misurando costantemente l'output risultante del modello e ottimizzando il sistema per avvicinarsi alla precisione del target. Il livello di accuratezza ottenibile dipende da due cose: i dati disponibili e l'algoritmo in uso. Gli algoritmi di apprendimento supervisionato generano principalmente

due tipi di risultati: *classificazione e regressione*. Un algoritmo di *classificazione* cerca di determinare la classe o la categoria dei dati con cui viene presentato. Ad esempio, gli algoritmi di riconoscimento degli oggetti sono problemi di classificazione, in cui l'algoritmo ha il compito di determinare a quale categoria di oggetti appartiene l'elemento che gli viene presentato. Il riconoscimento dei caratteri, la classificazione dello spam e-mail, la classificazione dei farmaci sono esempi di problemi che richiedono all'IA di determinare a quale classe appartengono i dati. Molte volte, un oggetto potrebbe appartenere a diverse categorie e l'IA deve determinare quali sono tali categorie e quanta fiducia l'algoritmo ha nelle sue previsioni. Le attività di *regressione* sono diverse, poiché si aspettano che il modello produca un valore numerico,ad esempio se si desidera prevedere valori continui, come prevedere il costo di una casa o la temperatura esterna.Questo tipo di problema non ha un vincolo di valore specifico perché il valore potrebbe essere qualsiasi numero senza limiti.

## 3.2 Apprendimento non Supervisionato

L'apprendimento senza supervisione è utilizzato quando abbiamo a che fare con dati che non sono stati etichettati o classificati. L'obiettivo è trovare modelli e creare una struttura nei dati per ricavarne il significato. Due forme di apprendimento senza supervisione sono il *clustering* e *la riduzione della dimensionalità* . Il *clustering* raggruppa dati simili. I dati in un gruppo dovrebbero avere proprietà o caratteristiche simili tra loro. Tuttavia, rispetto ai dati di un altro gruppo, dovrebbero avere proprietà molto diverse. La *riduzione della dimensionalità* è la compressione dei dati rimuovendo le variabili casuali e mantenendo quelle principali senza perdere la struttura e la significatività del set di dati. L'utilità della riduzione della dimensionalità deriva dal rendere i dati più facili da memorizzare, più veloci per eseguire i calcoli e più facili da visualizzare nella visualizzazioni dei dati. È anche utile per rimuovere falsi segnali o rumore da un modello che aiuta a migliorarne le prestazioni.

## 3.3 Apprendimento per rinforzo

Il Reinforcement Machine Learning è un sottoinsieme dell'intelligenza artificiale.Questo tipo di apprendimento automatico richiede l'uso di un sistema di ricompensa / penalità . L'obiettivo è premiare la macchina quando impara correttamente e penalizzare la macchina quando apprende in modo errato. Con l'ampia gamma di risposte possibili dai dati, il processo di questo tipo di apprendimento è un passaggio iterativo,poichè il sistema impara continuamente. Esempi di apprendimento per rinforzo sono ad esempio,insegnare ad una macchina a giocare a scacchi o ai videogame o le auto a guida autonoma.

Nella *Figura 4* sono mostrati i tre tipi di apprendimento (Supervisionato,Non Supervisionato e per Rinforzo) con le rispettive caratteristiche.

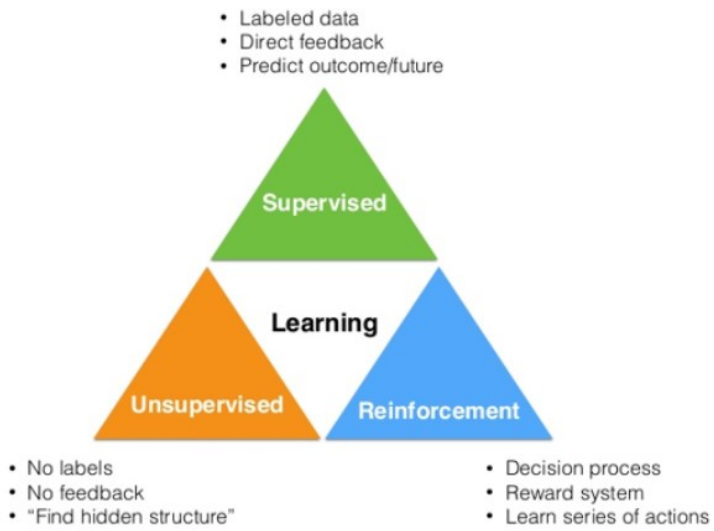



*Figura 4 Supervised,Unsupervised e Rinforcement Learning*

## 3.4 Neural Network

Le reti neurali sono un insieme di algoritmi ispirati al funzionamento del cervello umano.Le reti neurali a volte chiamate reti neurali artificiali (ANN), perché non sono naturali come i neuroni nel cervello, sono composte da un gran numero di elementi di elaborazione altamente interconnessi (neuroni) che lavorano all'unisono per risolvere problemi specifici. Le reti neurali, con la loro notevole capacità di ricavare significato da dati complicati o imprecisi, possono essere utilizzate per estrarre modelli e rilevare tendenze troppo complesse per essere notate dagli essere umani o da altre tecniche informatiche. Il tipo più comune di rete neurale artificiale è costituito da tre gruppi(layer) di neuroni: un layer di input,connesso ad un layer intermedio(hidden layer) connesso ad un layer di output. *Nella Figura 5* viene mostrata l'architettura di una rete neurale.

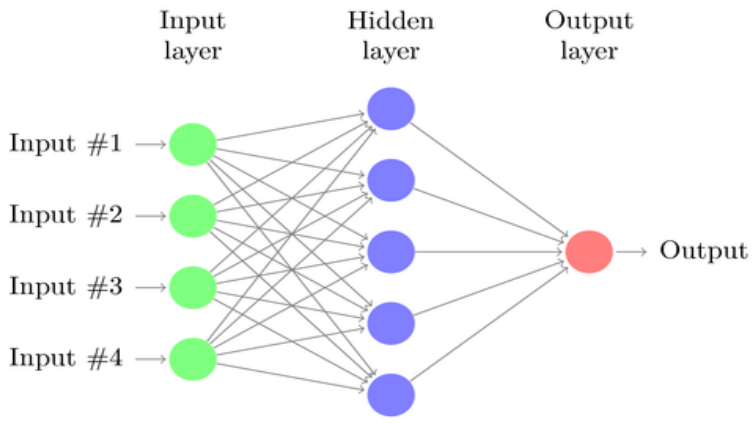



*Figura 5 Neural Network*

L'unità di base del calcolo in una rete neurale è il neurone , spesso chiamato nodo . Riceve input da altri nodi o da una sorgente esterna e calcola un output. Ad ogni ingresso è associato un peso (w), che viene assegnato in base alla sua importanza relativa agli altri ingressi. Il nodo applica una funzione f (*funzione di attivazione*) alla somma ponderata dei suoi input. La funzione di attivazione ha lo scopo di introdurre la non linearità nell'output di un neurone. Questo è importante perché la maggior parte dei dati del mondo reale sono non lineari e vogliamo che i neuroni apprendano queste rappresentazioni non lineari.

Ogni funzione di attivazione (o non linearità ) prende un singolo numero ed esegue su di esso una determinata operazione matematica fissa. Esistono diverse funzioni di attivazione in pratica(sigmoid,softmax ecc). Nella *Figura 6* viene mostrata in dettaglio la formula della funzione di attivaizone.

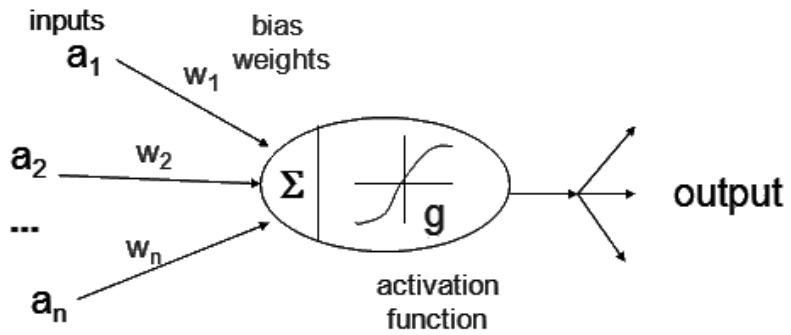



*Figura 6 Funzione di Attivazione*

## 3.5 Deep Learning

Il deep learning è un tipo di machine learning (ML) che imita il modo in cui gli esseri umani acquisiscono determinati tipi di conoscenza. Il deep learning è un elemento importante della data science, che include statistiche e modelli predittivi. È estremamente vantaggioso per i data scientist che hanno il compito di raccogliere, analizzare e interpretare grandi quantità di dati poichè il deep learning rende questo processo più veloce e più facile. Nella sua forma più semplice, il deep learning può essere pensato come un modo per automatizzare l' analisi predittiva . Mentre gli algoritmi di machine learning tradizionali sono lineari, gli algoritmi di deep learning sono impilati in una gerarchia di complessità e astrazione crescenti. Per raggiungere un livello accettabile di precisione, i programmi richiedono l'accesso a immense quantità di dati di addestramento e potenza di elaborazione, nessuno dei quali era facilmente disponibile per i programmatori fino all'era dei big data e del cloud computing . Poiché la programmazione del deep learning può creare modelli statistici complessi direttamente dal proprio output iterativo, è in grado di creare modelli predittivi accurati da grandi quantità di dati non etichettati e non strutturati. Questo è importante poiché l'Internet delle cose (IoT)

continua a diventare più pervasivo, perché la maggior parte dei dati che gli esseri umani e le macchine creano non sono strutturati e non sono etichettati. La rete neurale artificiale è una rete di neuroni artificiali (o nodi) interconnessi in cui ogni neurone rappresenta un'unità di elaborazione delle informazioni. Questi nodi interconnessi si scambiano le informazioni imitando il cervello umano. I nodi interagiscono tra loro e condividono le informazioni. Ogni nodo riceve l'input ed esegue alcune operazioni su di esso prima di trasmetterlo. L'operazione viene eseguita da quella che viene chiamata funzione di attivazione (non linearità).Tale funzione converte l'input in output ,che può essere quindi utilizzato come input per altri nodi. I collegamenti tra i nodi sono per lo più ponderati. Questi pesi vengono regolati in base alle prestazioni della rete. Se le prestazioni (o la precisione) sono elevate, i pesi non vengono regolati, ma se le prestazioni sono basse, i pesi vengono regolati tramite calcoli specifici. Il livello di neuroni più a sinistra è chiamato livello di input e, analogamente, il livello più a destra è chiamato livello di output. Tutti gli altri livelli intermedi sono chiamati livelli nascosti. In poche parole, un neurone artificiale riceve l'input da altri nodi e applica la funzione di attivazione alla somma ponderata dell'input (funzione di trasferimento) e quindi passa l'output. Una soglia (chiamata Bias) viene aggiunta alla somma ponderata per evitare di non passare alcun output (zero). Per l'apprendimento profondo, diversi livelli di rete neurale ,di solito maggiore di 100, sono collegati in stile *feedforward* o *feedback* per passare le informazioni l'uno all'altro.

- **Feedforward**: questo è il tipo più semplice di ANN. Qui, le connessioni non formano un ciclo e quindi non hanno loop. L'ingresso viene alimentato direttamente all'uscita (in un'unica direzione) attraverso una serie di pesi. Sono ampiamente utilizzati nel riconoscimento dei modelli.
- **Feedback** (o ricorrente): le connessioni nella rete di feedback possono muoversi in entrambe le direzioni. L'output derivato dalla rete viene reimmesso nella rete per migliorare le prestazioni (loop). Queste reti possono diventare molto complicate ma sono relativamente più potenti del feedforward. Le reti di feedback sono dinamiche e sono ampiamente utilizzate per molti problemi.

Le ANN più popolari utilizzate per il deep learning sono:

- **Perceptrons multistrato**: sono le reti neurali più elementari con reti feedforward. Generalmente utilizzano funzioni di attivazione non lineari (come Tang o Relu) e calcolano le perdite tramite Mean Square Error (MSE) o Logloss. La perdita viene nuovamente propagata per regolare i pesi e rendere il modello più accurato. Sono generalmente utilizzati come parte di una rete di apprendimento profondo più ampia.
- **Rete neurale Convoluzionale**(CNN): sono simili alle reti neurali ordinarie ma la loro architettura è progettata specificamente per le immagini come input. In particolare, a differenza di una normale rete neurale, gli strati di una CNN hanno neuroni disposti in 3 dimensioni: larghezza, altezza, profondità. Sono particolarmente adatti per dati spaziali, riconoscimento di oggetti e analisi di immagini utilizzando strutture neuronali multidimensionali. Uno dei motivi principali della popolarità del deep learning ultimamente è dovuto proprio alle CNN. Alcuni degli usi comuni delle reti neurali convoluzionali sono auto a guida autonoma, droni, visione artificiale e analisi del testo.
- **Reti neurali ricorrenti**(RNN): sono una rete feedforward, tuttavia con loop di memoria ricorrenti che prendono l'input dai livelli precedenti (*backpropagation*). Ciò conferisce loro una capacità unica di modellare lungo la dimensione temporale e la sequenza arbitraria di eventi e input. In termini più semplici, per ogni dato istante, la rete mantiene una memoria fino a quel momento e quindi può prevedere l'azione successiva.

# CAPITOLO 4
# SOLUZIONE PROPOSTA

In questo capitolo verranno mostrati passo-passo tutti gli step per la soluzione al nostro problema(classificazione dei malware). In particolare, quando si vanno ad applicare algoritmi di Machine Learning si eseguono i seguenti passi: generazione del Dataset, pulizia dei dati, estrazione delle feature (api call), creazione del modello, allenamento e testing del modello. Nel caso in cui i risultati non sono soddisfacenti viene applicata una fase di ottimizzazione dei parametri. Il modello scelto per classificazione dei malware è la CNN(Convolutional Neural Network). Le CNN sono un modello di Deep Learning, che fa parte della classe di algoritmi per l'apprendimento supervisionato, ciò significa che il modello verrà allenato su dei dati etichettati, con l'obbiettivo di classificare dati che non ha mai visto prima.

## 4.1 Ambiente di Sviluppo

Il lavoro di implementazione della rete neurale è stato svolto utilizzando il linguaggio di programmazione Python. L'IDE utilizzato è PyCharm. PyCharm presenta un insieme di strumenti che permettono di testare i propri script tramite una comoda Python console integrata. All'interno di PyCharm è possibile integrare varie librerie, tra cui:

- **Numpy**: permette di manipolare agevolmente vettori ad n-dimensioni.
- **Tensorflow** : è una piattaforma open source end-to-end per il machine learning e rappresenta un sistema completo e flessibile di strumenti, librerie e altre risorse che forniscono flussi di lavoro con API di alto livello.
- **Keras**, è una libreria open source di componenti di rete neurale ed è in grado di funzionare su Tensorflow. Keras ha un'interfaccia semplice e coerente ottimizzata per i casi d'uso comuni che fornisce un feedback chiaro e utilizzabile per gli errori dell'utente.

## 4.2 Dataset

Il dataset "Drebin" contiene 5560 file di 179 diverse famiglie di malware. I campioni di malware sono stati raccolti da agosto 2010 a ottobre 2012. Inoltre, il dataset fornisce l'hash SHA256 di tutti i campioni di malware e le corrispondenti etichette della famiglia AV. Le etichette sono state create manualmente unificando l'output di diversi scanner AV. Nella nostra attività di tesi sono state prese in considerazione 5 famiglie di malware: BaseBridge, DroidKungFu, FakeInstaller, Opfake e Plankton. E da ognuna famiglia abbiamo analizzato all'incirca 210 malware.

## 4.3 Feature Estratte

Le feature estratte dall'analisi statica e dinamica dei malware effettuate utilizzando la Sandbox *CuckooDroid* sono:

- File Details
- Informazioni sull'applicazione android
- Informazioni dell'analisi dinamica
- Screenshot
- Network Analysis

*File Details* contiene:

- Nome del file
- Dimensione del file
- Tipo del file
- Hash MD5, SHA1, SHA256, SHA512

Nella *Figura 7* viene mostrata la schermata di CuckooDroid che mostra le informazioni sul file in analisi(File Details).

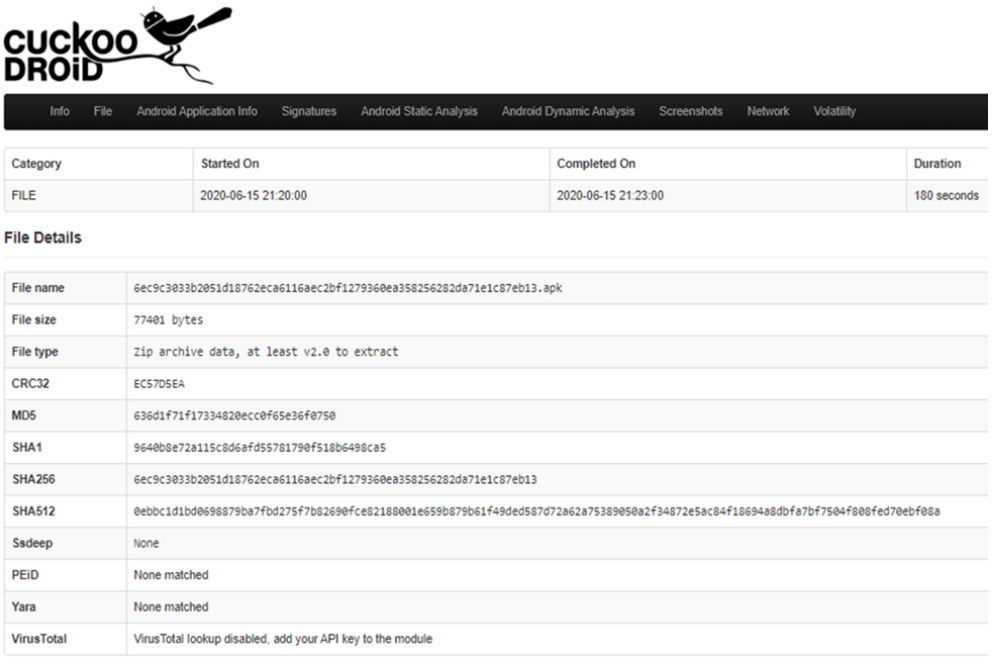



*Figura 7 Schermata CuckooDroid*

Le informazioni estratte dall'*analisi statica* dall'app Android sono:

- **Activities**: La classe Activity è un componente cruciale di un'app Android e il modo in cui le attività vengono avviate e messe insieme è una parte fondamentale del modello applicativo della piattaforma. Il sistema Android avvia il codice in un'istanza Activity invocando metodi di callback specifici che corrispondono a fasi specifiche del suo ciclo di vita.
- **Services** : Un servizio è un componente dell'applicazione che può eseguire operazioni di lunga durata in background e non fornisce un'interfaccia utente.
- **Receivers**: I ricevitori consentono alle applicazioni di ricevere messaggi trasmessi dal sistema o da altre applicazioni, anche quando altri componenti dell'applicazione non sono in esecuzione.
- **Permissions**: Lo scopo di un'autorizzazione è proteggere la privacy di un utente Android. Le app Android devono richiedere l'autorizzazione per accedere ai dati sensibili dell'utente (come contatti e SMS), nonché a determinate funzionalità del sistema (come videocamera e Internet). A seconda della funzione, il sistema potrebbe concedere l'autorizzazione automaticamente o richiedere all'utente di approvare la richiesta.

Nella *Figura 8* vengono mostrate le informazioni sull'applicazione Android ricavate dall'analisi di CuckooDroid.

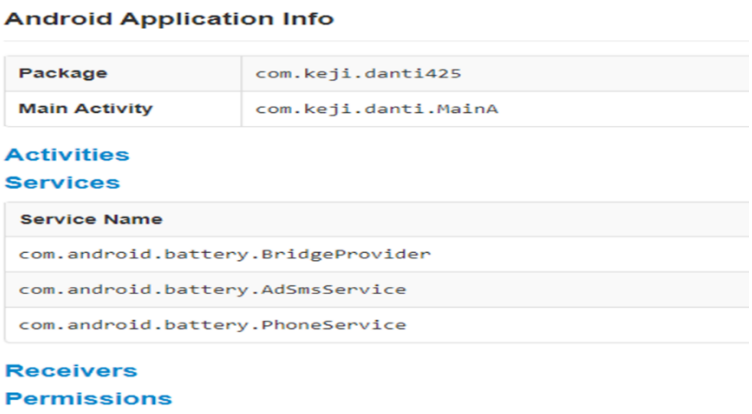



*Figura 8 Android Application Info*

Le informazioni estratte dall'*analisi dinamica* sono:

- Chiave di cifratura
- Testo cifrato
- Fingerprints
- File e cartelle accedute dal malware
- Proprietà del sistema

Nella *Figura 9* vengono mostrati i risultati dell'analisi dinamica di CuckooDroid.

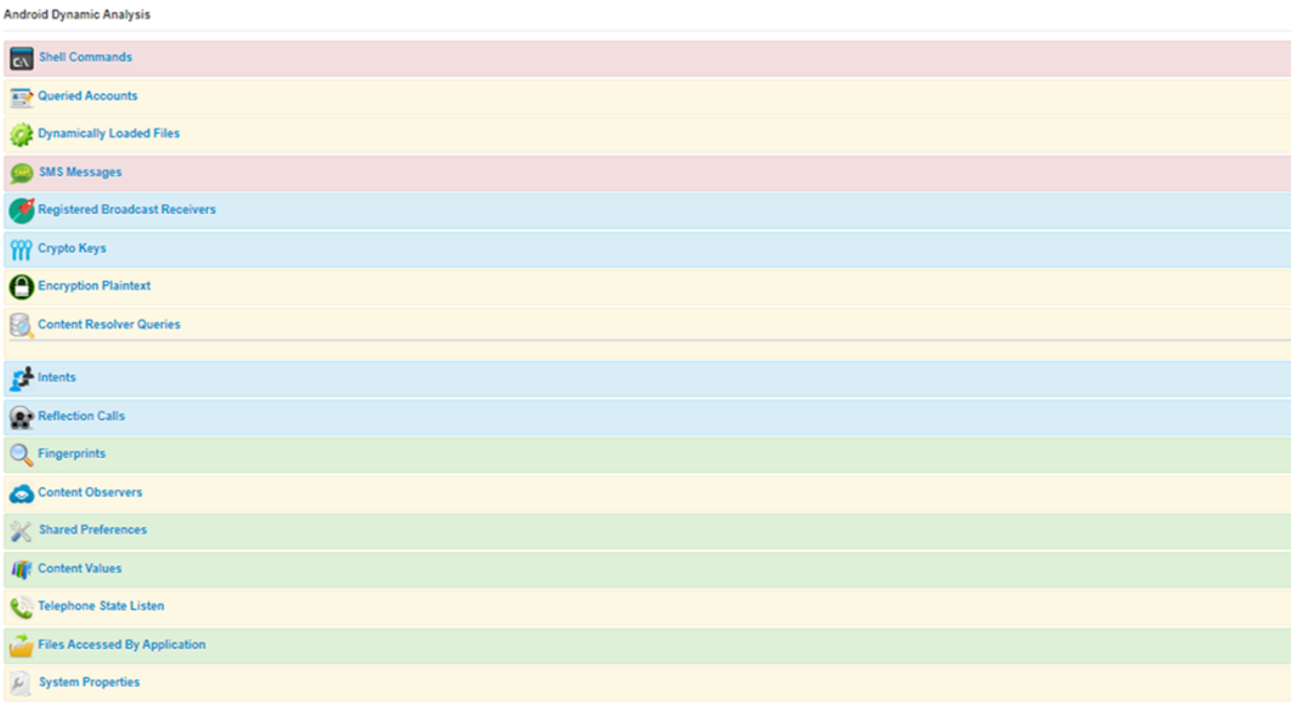



*Figura 9 Android Dinamic Analysis*

Gli *screenshot* eseguiti durante l'esecuzione dell'applicazione sono mostrati nella *Figura 10*.

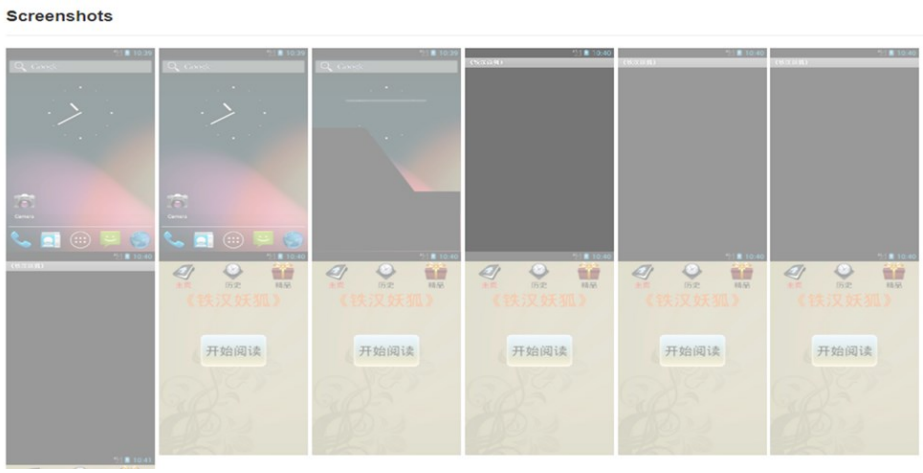



*Figura 10 Screenshot CuckooDroid*

Infine il *network analysis* riguarda le informazioni di quando il malware accede alla rete:

- **Host Involved**: Tutti gli host che vengono richiamati durante l'esecuzione dell'app
- **Richieste DNS ed HTTP/HTTPS**

Nella *Figura 11* vengono mostrati i risultati della Network Analysis.

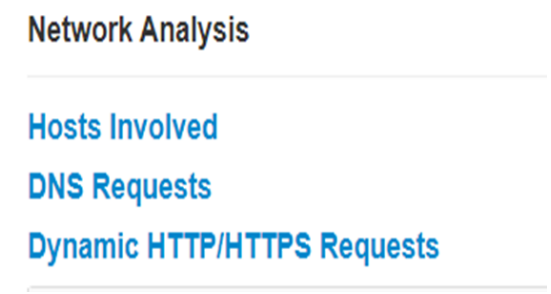
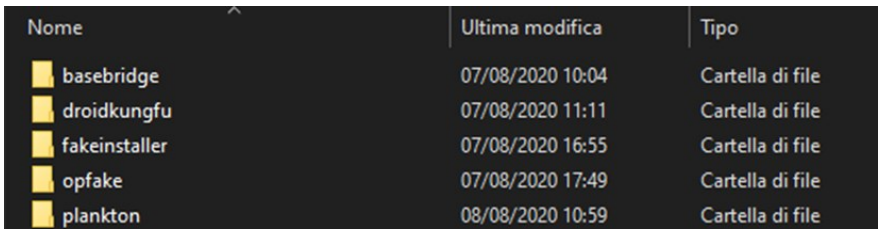


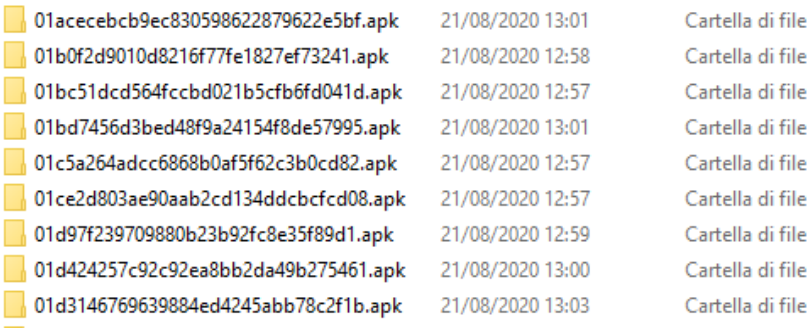

*Figura 11 Network Analysis*

## 4.4 Generazione dati di Input

Dopo aver analizzato le 5 famiglie di malware, abbiamo raggruppato le analisi relative a ciascuna famiglia nella propria cartella




All'interno di ogni cartella sono presenti tutte le analisi estratte da *CuckooDroid*:




Le informazioni di cui abbiamo bisogno nel nostro lavoro di tesi sono presenti all'interno della cartella *Logs* nel file *Droidmon.*In particolare nel suddetto file sono presenti per ogni malware :

- Timestamp
- Classe
- Metodo

- Tipo e Path

Nella *Figura 12* vengono mostrate le informazioni presenti all'interno del file Droidmon.


```
{"timestamp":1592894238499,"result":"false","class":"java.io.File","method":"exists","type
{"timestamp":1592894238506,"result":"false","class":"java.io.File","method":"exists","type
{"timestamp":1592894238518,"result":"false","class":"java.io.File","method":"exists","type
{"timestamp":1592894238540,"result":"Europe\/Rome","class":"android.os.SystemProperties","
{"timestamp":1592894238593,"class":"android.content.ContentResolver","method":"registerCon
{"timestamp":1592894239076,"result":"false","class":"java.io.File","method":"exists","type
{"timestamp":1592894239110,"result":"false","class":"java.io.File","method":"exists","type
```


*Figura 12 Informazioni presenti all'interno del file Droidmon*

Per ogni file *Droidmon* estraiamo le Api call(*Method*) attraverso lo script mostrato nella *Figura 13*:

analysis.py

```
import json
import re

MalwareList = []
print("Started Reading JSON file which contains multiple JSON document")
with open('P79.log') as f:
    for jsonObj in f:
        malwareDict = json.loads(jsonObj)
        MalwareList.append(malwareDict)

print("Printing each JSON Decoded Object")
for malware in MalwareList:
    print(malware["method"],end=",")
```

*Figura 13 Script per l'estrazione delle Api-Call*

Il risultato finale dell'estrazione delle Api-Call è mostrato nella *Figura 14*.


```
exists,exists,exists,sendBroadcast,exists,exists,invoke,exists,exists,exists,exists,exists,ex
exists,exists,exists,startService,exists,exists,invoke,putInt,exists,open,invoke,invoke,get,g
exists,exists,exists,sendBroadcast,exists,exists,invoke,exists,exists,exists,exists,exists,ex
exists,exists,exists,sendBroadcast,exists,exists,invoke,exists,exists,exists,exists,exists,ex
exists,exists,exists,exists,exists,invoke,findResource,openConnection,openConnection,open,exi
exists,exists,exists,exists,exists,exists,exists,get,putLong,exists,open,invoke,exists,exists
exists,exists,exists,sendBroadcast,exists,exists,invoke,exists,exists,exists,exists,exists,ex
exists,exists,exists,sendBroadcast,exists,exists,invoke,exists,exists,exists,exists,exists,ex
exists,exists,exists,startService,get,get
exists,exists,exists,sendBroadcast,exists,exists,invoke,exists,exists,exists,exists,exists,ex
exists,exists,exists,exists,findLibrary,exists,findLibrary,exists,findLibrary,startService,ex
exists,exists,exists,startService,exists,exists,exists,exists,exists,exists,exists,exists,exe
exists,exists,exists,startService,exists,exists,invoke,exists,exists,putInt,exists,open,invok
exists,exists,exists,exists,exists,exists,exists,get,putLong,exists,open,invoke,exists,exists
exists,exists,exists,exists,exists,invoke,findResource,openConnection,openConnection,open,exi
exists,exists,exists,exists,exists,putString,putString,putString,putBoolean,exists,open,exec,
exists,exists,exists,exists,exists,invoke,findResource,openConnection,openConnection,open,exi
exists,exists,exists,exists,exists,exists,exists,get,putLong,exists,open,invoke,exists,exists
```


*Figura 14 Api Call in formato testuale*

Data questa sequenza di Api-Call in formato testuale,il passo successivo è quello di creare un *Dizionario*. Il dizionario è una raccolta non ordinata, modificabile e indicizzata. In Python i dizionari sono scritti con parentesi graffe e hanno chiavi e valori. Nel nostro caso le *chiavi* sono rappresentate dai singoli Metodi della Api-Call,i *valori* sono rappresentati da interi univoci che vanno da 1 a 113(poichè ci sono 113 metodi distinti in totale). Nella *Figura 15* viene mostrata una parte del Dizionario che è stato creato.

```
dizionario_malware={"exists":1,"sendbroadcast":2,
"invoke":3,"findresource":4,"openconnection":5,"open":6,"opendexfile":7,
"dalvik.system.dexfile":8,"loaddex":9,"dalvik.system.dexclassloader":10,"get":11,"getline1number":12,
"getdeviceid":13,"getsimserialnumber":14,"getnetworkoperatorname":15,"getdevicesoftwareversion":16,"getmacaddress":17,"put":18
```

*Figura 15 Dizionario delle Api Call*

Dopo aver creato il dizionario,il passo successivo è quello di sostituire i valori del dizionario con i metodi(stringhe) nel testo,ottenendo il risultato ottenuto in *Figura 16.*

```
1,1,1,66,1,1,3,1,1,1,1,1,1,1,68,73,73,6,6,6,6,6,6,1,68,73,73,6,6,6,6,6,6,6,1,1,6,6,6,69,70,7
1,1,1,67,1,1,3,78,1,6,3,3,11,11
1,1,1,66,1,1,3,1,1,1,1,1,1,1,68,73,73,6,6,6,6,6,6,1,68,73,73,6,6,6,6,6,6,6,1,1,6,6,6,69,70,7
1,1,1,66,1,1,3,1,1,1,1,1,1,1,68,73,73,6,6,6,6,6,6,1,68,73,73,6,6,6,6,6,6,6,1,1,6,6,6,69,70,7
1,1,1,1,1,3,68,73,73,6,1,1,1,6,104,1,1,3,30,30,22,73,3,3,11,30,30,22,73,30,30,22,73,30,30,22,
1,1,1,1,1,1,1,11,86,1,6,3,1,1,1,1,1,1,1,68,73,73,6,6,6,6,6,6,1,68,73,73,6,6,6,6,6,6,6,1,1,6,
1,1,1,66,1,1,3,1,1,1,1,1,1,1,68,73,73,6,6,6,6,6,6,1,68,73,73,6,6,6,6,6,6,6,1,1,6,6,6,69,70,7
1,1,1,66,1,1,3,1,1,1,1,1,1,1,68,73,73,6,6,6,6,6,6,1,68,73,73,6,6,6,6,6,6,6,1,1,6,6,6,69,70,7
1,1,1,67,11,11
1,1,1,66,1,1,3,1,1,1,1,1,1,1,68,73,73,6,6,6,6,6,6,1,68,73,73,6,6,6,6,6,6,6,1,1,6,6,6,69,70,7
1,1,1,1,75,1,75,1,75,67,1,1,1,1,18,76,77,78,1,6,1,1,1,1,11,22,22,73,73,22,73,22,73,1,1,11
1,1,1,67,1,1,1,1,1,1,1,1,34,34,34,1,1,1,1,1,11,6,11,11,93,11,103,22,22,1,6,11,11,104,22,22,3,
1,1,1,67,1,1,3,1,1,78,1,6,3,3,3,11,11,11
1,1,1,1,1,1,1,11,86,1,6,3,1,1,1,1,1,1,1,68,73,73,6,6,6,6,6,6,1,68,73,73,6,6,6,6,6,6,6,1,1,6,
1,1,1,1,1,3,68,73,73,6,1,1,1,6,104,1,1,3,30,30,22,73,3,3,11,30,30,22,73,30,30,22,73,30,30,22,
```

*Figura 16 Api Call in formato numerico*

Infine, per ogni riga(che rappresenta una sequenza di api call numerica) si crea una matrice di dimensione 113*113 (numero massimo di elementi all'interno del dizionario). Pertanto, alla fine di questa fase si ottiene un file contenente 1050 matrici 113*113( 210 per ogni famiglia di malware). Le matrici vengono salvate all'interno del file "matrci_fin_out2.txt" per essere poi caricate successivamente come input per la rete neurale CNN , in quanto tali matrici possono essere considerate come immagini "grayscale". Nella *Figura 17* viene

mostrato lo script per la creazione delle matrici ed il loro salvataggio nel file "matrici_fin_out2.txt".

creazione_matrici_dalfile.py

```
import numpy as np
import pickle
with open("matrici_finali.txt", "r") as ins:
    # matrice_temp = np.zeros((10, 10))
    f=[]
    array = []
    obj = {}
    matrici_sparse={}
    for line in ins:
        line = line.strip()
        words = line.split(",")
        array.append(words)

    for i in range(0, len(array)):
        mylist2 = [int(item) for item in array[i]]
        obj[int(i)] = mylist2

for i in range(len(obj)):
    matrici_sparse[int(i)] = np.zeros((113, 113))
    l=obj[i]
    for first, second in zip(l, l[1:]):
        matrici_sparse[i][first][second] += 1

# Step 2
with open('matrci_fin_out2.txt', 'wb') as config_dictionary_file:
    # Step 3
    pickle.dump(matrici_sparse, config_dictionary_file)
```

*Figura 17 Codice per la creazione delle matrici*

Infine, per ogni record del dataset aggiungiamo un ulteriore feature, detta label, che indica il record a quale famiglia di malware appartiene, in particolare:

- **BaseBridge** = [10000]
- **DroidKungFu** = [01000]
- **FakeInstaller** = [00100]
- **Opfake** = [00010]
- **Plankton** = [00001]

# CAPITOLO 5
# RISULTATI SPERIMENTALI

Avendo a disposizione una serie di dati(api-image) appertenenti ad un numero finito di categorie note (5 famigle di Malware), l'obbiettivo della fase di training e testing del modello è quello di assegnare ogni campione alla giusta categoria (*Classificazione*). Innanzitutto vengono creati il *training set* ed il *test set* che servono per addestrare il modello. Il training set è un insieme di dati che vengono utilizzati per addestrare un sistema supervisionato, e generalmente corrisponde ad un vettore di input a cui è associata una risposta o una determinata classificazione. Una volta eseguito, l'algoritmo apprende, in base alla risposta o alla classificazione, quali caratteristiche discriminano gli elementi appartenenti alle differenti categorie. Una volta effettuata la fase di apprendimento, la correttezza dell'algoritmo viene verificata eseguendo lo stesso su un test set per verificare eventuale overfitting con l'insieme di addestramento.

## 5.1 Training e Testing del Modello

Il primo passo consiste nell'importare il train set e il test set che corrispondono al file "matrci_fin_out2.txt". Nella *Figura 18* viene riportato lo script per il caricamento delle matrici e la divisione in test e training set.

```
#carichiamo le matrici dal file
  with open("matrci_fin_out2.txt", "rb") as matrici:
      matrci_sparse = pickle.load(matrici)

  #carichiamo la label dal file
  with open("array_label.txt", "rb") as array:
      array_label2 = pickle.load(array)

  #dividiamo in training e test set e dividiamo per 255,per diminuire i valori dell input
  X_train,X_test,y_train,y_test=train_test_split(matrci_sparse,array_label2,test_size=0.3,random_state=42)

  X_train = [x /255 for x in X_train]
  X_test = [x /255 for x in X_test]
```

*Figura 18 Codice per il caricamento dei dati e la divisione del set di input*

Dopo aver caricato le matrici dal file *matrci_fin_out2.txt* e le corrispettive label dal file *array_label.txt*, tramite la funzione *train_test_split* , si va a creare il *training set* ed il *test set*. Inoltre per questa funzione vengono utilizzati *test_size* = 0.3, in cui si va a dividere l'input per il 70% nel training set e per il 30% nel test set, mentre la funzione *random_state* = 42,

serve a garantire pseudocasualità in quanto rende i risultati riproducibili. Inoltre, dividiamo sia il training set che il test set per 255, cosi facendo tutti i valori all'interno della matrice saranno compresi tra 0 e 1. A questo punto abbiamo 735 matrici all'interno del training set e 315 matrici nel test set. Inoltre, è stata utilizzata la funzione di *reshape* che permette di modificare la dimensione, il numero di righe e di colonne di un array,per passare correttamente l'input alla rete neurale. Nella *Figura 19* viene mostrato il codice della funzione reshape.

```
X_train=X_train.reshape(735, 113, 113,1)
X_test=X_test.reshape(315,113,113,1)
```

*Figura 19 Reshape Function*

Il passo successivo è stato quello di creare il modello. Generalmente l'architettura di una rete neurale convoluzionale è composta da una seria di livelli convoluzionali, un livello di pooling e una ulteriore serie di livelli *convoluzionali*, un livello di *pooling* e così via. Nella *Figura 20* viene mostrata l' architettura della CNN.

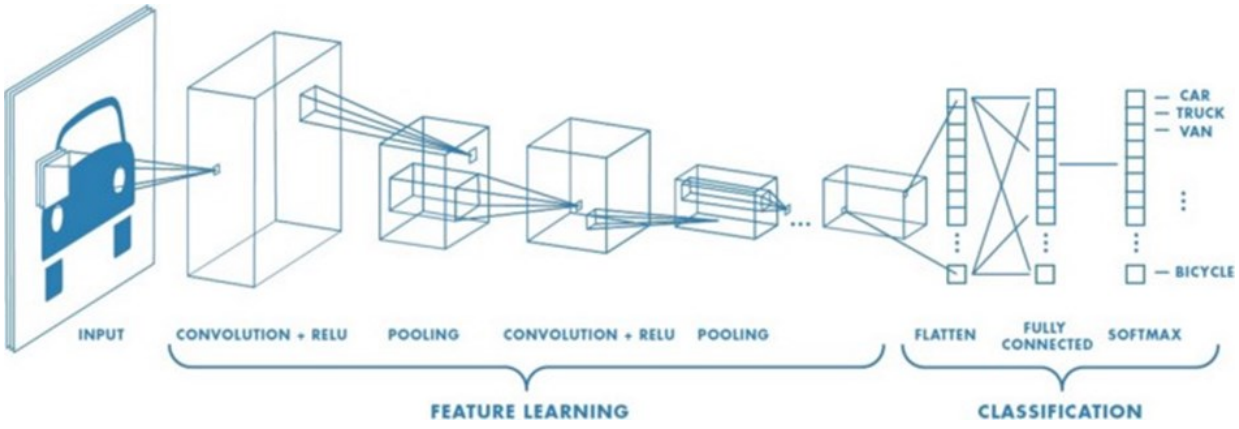



*Figura 20 Convolutional Neural Network CNN*

Il livello di **Convoluzione** è il primo livello per estrarre le caratteristiche da un'immagine di input. La convoluzione preserva la relazione tra i pixel apprendendo le caratteristiche dell'immagine utilizzando piccoli quadrati di dati di input. È un'operazione matematica che richiede due input ,ossia la matrice dell'immagine e un filtro(kernel). Quest'operazione

permetterà di creare una insieme di feature-map,sarà poi la rete stessa a determinare quale sia la migliore da utilizzare per classificare l'immagine. Nella *Figura 21* viene mostrato un approfondimento sul livello di convoluzione.

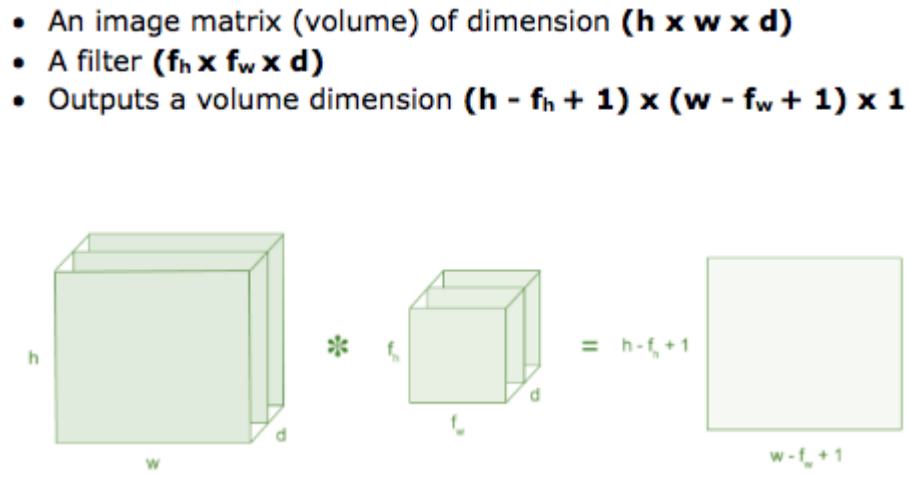



*Figura 21 Livello di Convoluzione*

Lo scopo del livello di **pooling**, invece, è quello di ridurre la dimensione delle matrici delle caratteristiche restituita dal livello convoluzionale così da rendere più rapido il passaggio successivo e concentrarsi sulle feature più caratterizzanti. Nella *Figura 22* viene mostrato il funzionamento del livello di Pooling.

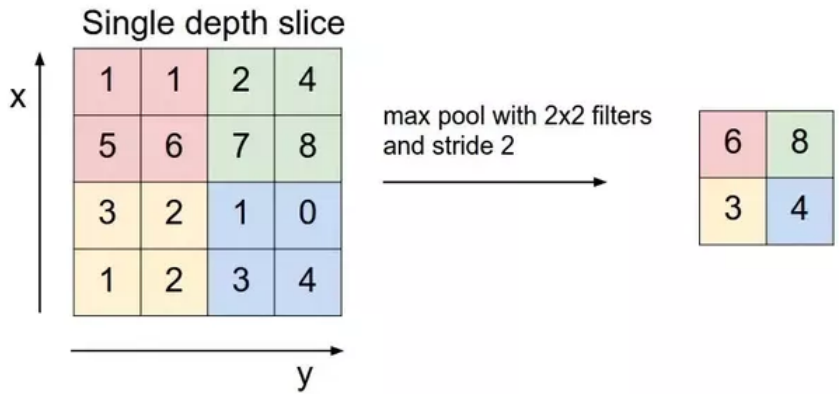



*Figura 22 Livello di Pooling*

Solimatemene quanto piu grande è l'immagine di input, tanto più questo ciclo viene ripetuto. L'immagine diventa sempre più piccola man mano che avanza attraverso la rete. Nella parte terminale della rete viene aggiunta una rete neurale, composta da alcuni livelli fortemente connessa(FC) e un livello terminale di output,ossia una funzione di attivazione come softmax o sigmoid per classificare l'output.Nella *Figura 23* viene mostrato il codice della CNN.

```
# implementazione della rete
model = Sequential()
model.add(Conv2D(filters=32, kernel_size=(4, 4), input_shape=(113, 113, 1), activation="relu"))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Conv2D(filters=64, kernel_size=(4, 4),activation="relu"))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(120, activation="relu"))
model.add(Dropout(0.5))
model.add(Dense(120, activation="relu"))
model.add(Dropout(0.5))
model.add(Dense(120, activation="relu"))
model.add(Dropout(0.3))
#output layer
model.add(Dense(5,activation="softmax"))
model.compile(loss="categorical_crossentropy",optimizer="adam",metrics=["accuracy"])
```

*Figura 23 Codice per la creazione del Modello*

*Kernel_size*: indica la dimensione dei filtri,viene impostato a '4' siccome la nostra matrice di input ha dimensioni abbastanza limitate.

*Input_shape:* i dati di input hanno una forma di (batch_size, height, width, depth) , dove la prima dimensione rappresenta la dimensione batch dell'immagine e le altre tre dimensioni rappresentano le dimensioni dell'immagine che sono altezza, larghezza e profondità. Nel nostro caso la profondità è settata ad 1 poichè stiamo utilizzando una "grayscale".

*Activation*: "relu", è una funzione di attivazione definita come la parte positiva del suo argomento,la formula matematica è la seguente:

$$f(x) = x^+ = \max(0, x)$$

dove x è l'input per un neurone. Nella *Figura 24* viene mostrata la funzione di attivazione RELU.

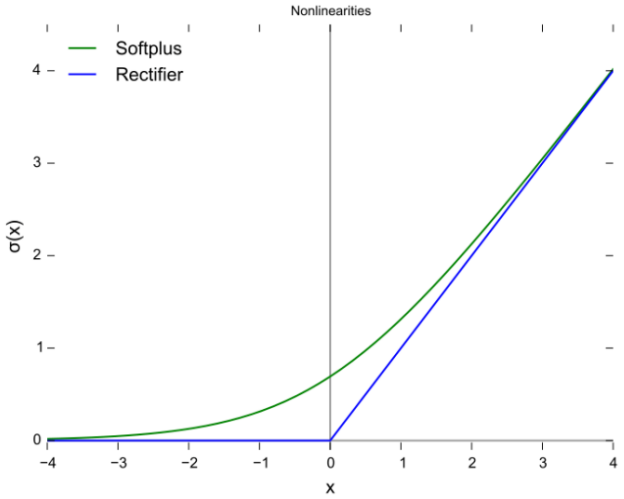



*Figura 24 Activation Function RELU*

Dopo il livello di convoluzione e quello di pooling abbiamo 3 layer Dense (fortemente connessi) con 128 nodi per ogni livello. Poichè i livelli sono fortemente connessi, bisogna effettuare il flattern per convertire l'output bidimensionale dei livello convoluzionale in un array monodimensionale, perché è quello che la rete fortemente connessa si aspetta di ricevere. Dopo ogni livello Dense cè un livello di *Dropout* impostato a 0.5,il quale serve ad eliminare l'overfitting nella rete.

Il *dropout* si riferisce all'ignorare le unità (cioè i neuroni) durante la fase di addestramento di un certo insieme di neuroni che viene scelto a caso. Con "ignorare", si intende che queste unità non sono considerate durante un particolare passaggio in avanti o all'indietro. Più tecnicamente, in ogni fase di addestramento, i singoli nodi vengono eliminati dalla rete con probabilità 1-p o mantenuti con probabilità p , in modo da lasciare una rete ridotta; vengono rimossi anche i bordi in entrata e in uscita verso un nodo abbandonato.

Il layer di output è formato da 5 neuroni fortemente connessi,con funzione di attivazione *softmax*. Tale funzione trasforma un vettore di K valori reali in un vettore di K valori reali che si sommano a 1. I valori di input possono essere positivi, negativi, zero o maggiori di uno, ma softmax li trasforma in valori compresi tra 0 e 1, in modo che possano essere interpretati come probabilità . Se uno degli input è piccolo o negativo, il softmax lo trasforma in una piccola probabilità, e se un input è grande, lo trasforma in una grande probabilità, ma rimarrà sempre tra 0 e 1. Softmax è una generalizzazione della regressione logistica che può essere utilizzata

per la classificazione multi-classe e la sua formula è molto simile alla funzione sigmoide utilizzata per la regressione logistica. La funzione softmax può essere utilizzata in un classificatore solo quando le classi si escludono a vicenda.

La formula di Softmax è la seguente:

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

dove tutti i valori zi sono gli elementi del vettore di input e possono assumere qualsiasi valore reale. Il termine in fondo alla formula è il termine di normalizzazione che assicura che tutti i valori di output della funzione si sommino a 1, costituendo così una valida distribuzione di probabilità. L' ultimo passo, prima di effettuare il training, è quello di compilare il modello. In questa operazione settiamo come metrica da misurare l'accuratezza, e come loss la 'categorical_crossentropy', siccome il nostro è un problema di categorizzazione di 5 famiglie di malware diversi.

La *categorical_cross*-entropy è un'attivazione Softmax più una perdita di Entropia incrociata . Se usiamo questa perdita, addestreremo una CNN a produrre una probabilità su

C classi per ogni immagine. Viene utilizzata per la classificazione multi-classe. Nella *Figura 25* viene mostrata la struttura della categorical cross entropy.

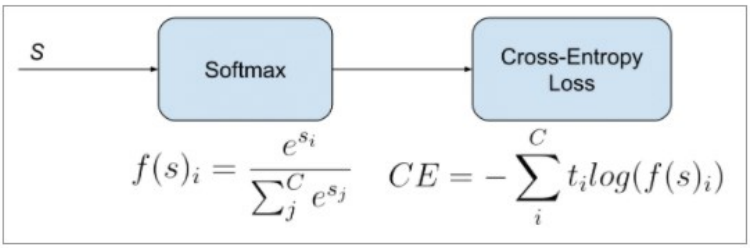



*Figura 25 Categorical Cross Entropy*

Una volta compilato il modello, possiamo iniziare ad addestrarlo. Quest'operazione viene effettuata utilizzando il metodo *fit*, mostrata nella *Figura 26.*

```
history = model.fit(X_train,y_train,epochs=200,batch_size=380,
                    validation_data=(X_test,y_test),verbose=2, shuffle=True)
```

*Figura 26 Fit del Modello*

Oltre alla matrice contenente tutte le feature X e il vettore delle label Y del train-set, tale metodo riceve come parametri:

- **epoche**: nel nostro caso 200, cioè il numero di iterazioni che effettuerà il modello sul train-set.
- **batch size**: nel nostro caso 380, ciò significa che i pesi della rete vengono aggiornati in blocchi di 380, quest'operazione consente di velocizzare il training della rete.
- **verbose** : impostato a 2, ci permette di stampare a video i dettagli di ogni epoche.

Il report dell'allenamento del nostro modello viene mostrato nella *Figura 27*:

```
Epoch 190/200
735/735 - 9s - loss: 0.0340 - accuracy: 0.9932 - val_loss: 0.2235 - val_accuracy: 0.9492
Epoch 191/200
735/735 - 8s - loss: 0.0347 - accuracy: 0.9905 - val_loss: 0.2307 - val_accuracy: 0.9524
Epoch 192/200
735/735 - 8s - loss: 0.0472 - accuracy: 0.9837 - val_loss: 0.2362 - val_accuracy: 0.9524
Epoch 193/200
735/735 - 9s - loss: 0.0339 - accuracy: 0.9905 - val_loss: 0.2326 - val_accuracy: 0.9524
Epoch 194/200
735/735 - 8s - loss: 0.0385 - accuracy: 0.9878 - val_loss: 0.2328 - val_accuracy: 0.9524
Epoch 195/200
735/735 - 8s - loss: 0.0435 - accuracy: 0.9864 - val_loss: 0.2348 - val_accuracy: 0.9524
Epoch 196/200
735/735 - 8s - loss: 0.0482 - accuracy: 0.9823 - val_loss: 0.2232 - val_accuracy: 0.9492
Epoch 197/200
735/735 - 8s - loss: 0.0390 - accuracy: 0.9850 - val_loss: 0.2306 - val_accuracy: 0.9524
Epoch 198/200
735/735 - 8s - loss: 0.0307 - accuracy: 0.9918 - val_loss: 0.2350 - val_accuracy: 0.9492
Epoch 199/200
735/735 - 9s - loss: 0.0310 - accuracy: 0.9932 - val_loss: 0.2305 - val_accuracy: 0.9524
Epoch 200/200
735/735 - 9s - loss: 0.0439 - accuracy: 0.9864 - val_loss: 0.2249 - val_accuracy: 0.9524
```

*Figura 27 Report dell'allenamento del Modello*

Generalmente, quando l'allenamento procede bene, il valore della loss diminuisce, mentre il valore dell'accuratezza aumenta.

## 5.2 Valutazione delle Performance

Dopo aver allenato e testato il modello, si passa alla valutazione delle prestazioni,per comprendere quanto è "buono" il nostro modello. Il modello è stato testato su 1050 record(matrici di malware), ed i risultati ottenuti sono molto buoni. Abbiamo un accuratezza di circa il 96/98%. Un altro fattore molto importante riscontrato è il valore della loss che risulta molto basso pari a 0.04/0.05. La funzione loss esprime la difformità tra i valori predetti dal modello in fase di allenamento e i valori attesi per ciascuna istanza di esempio. L'obiettivo finale è dunque quello di insegnare al modello la capacità di predire correttamente i valori attesi su un set di istanze non presenti nel training set (test set) mediante la minimizzazione della loss function in questo insieme di istanze. Questo porta ad una maggiore generalizzazione delle capacità di predizione. Per confermare la bontà dei risultati ottenuti, oltre all'accuratezza sono state valutate anche ulteriori metriche:

- **Precision**=95%
- **Recall**=95%
- **F1-Score**=95%

Andiamo a definire i seguenti valori:

- **Veri Positivi(TP):** Questi sono i valori positivi previsti correttamente, il che significa che il valore della classe effettiva è sì e anche il valore della classe prevista è sì.
- **Veri Negativi(TN):** Questi sono i valori negativi previsti correttamente, il che significa che il valore della classe effettiva è no e anche il valore della classe prevista è no.
- **Falsi Positivi(FP):** Quando la classe effettiva è no e la classe prevista è sì.
- **Falsi Negativi(FN):** Quando la classe effettiva è sì ma la classe prevista è no.

I valori dei falsi positivi e falsi negativi si verificano quando la classe prevista è in contrasto con quella effettiva. Una volta compresi questi quattro parametri, si possono calcolare *Accuracy,Precision*, *Recall* e *F1-Score*.

- **Accuracy**: è la misura delle prestazioni più intuitiva ed è semplicemente un rapporto tra l'osservazione prevista correttamente e le osservazioni totali. Si potrebbe pensare

che, se abbiamo un'elevata precisione, il nostro modello è buono. L'accuratezza è un'ottima misura, ma solo quando si dispone di set di dati simmetrici in cui i valori di falsi positivi e falsi negativi sono quasi gli stessi. Pertanto, si devono esaminare altri parametri per valutare le prestazioni del modello. Accuratezza = TP + TN / TP + FP + FN + TN

- **Precision**: è il rapporto tra le osservazioni positive previste correttamente e le osservazioni positive totali previste.
  Precisione = TP / TP + FP
- **Recall**: è il rapporto tra le osservazioni positive previste correttamente e tutte le osservazioni nella classe attuale
  Recall= TP / TP + FN
- **F1-Score**: è la media ponderata di Precisione e Richiamo. Pertanto, questo punteggio tiene conto sia dei falsi positivi che dei falsi negativi. Intuitivamente non è così facile da capire come l'accuratezza, ma F1 di solito è più utile dell'accuratezza, soprattutto se si dispone di una distribuzione delle classi non uniforme.
  F1-Score= 2 * (Recall * Precision) / (Recall + Precision)

Questi valori vengono ottenuti tramite il metodo *classification_report.*Viene anche mostrata la matrice con i valori dei falsi positivi,falsi negativi,veri positivi e veri negativi,ottenuta tramile il metodo *confusion_matrix.*Entrambi i metodi vengono utilizzati importando la libreria *sklearn.metrics.*Nella *Figura 28* viene mostrata la Classification Report,che mostra per ognuna delle 5 famiglie il grado raggiunto da ogni metrica (precision,recall ecc). Vediamo che la media dei valori è del 95%,quindi un risultato molto positivo.

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.95 | 0.94 | 0.94 | 63 |
| 1 | 0.92 | 0.94 | 0.93 | 69 |
| 2 | 0.99 | 0.93 | 0.96 | 75 |
| 3 | 0.96 | 1.00 | 0.98 | 52 |
| 4 | 0.95 | 0.96 | 0.96 | 56 |
| accuracy | | | 0.95 | 315 |
| macro avg | 0.95 | 0.96 | 0.95 | 315 |
| weighted avg | 0.95 | 0.95 | 0.95 | 315 |

*Figura 28 Classification Report*

La Confusion Matrix(Matrice di confusione) viene riportata nella *Figura 29*. Nella diagonale principale sono presenti tutti i true positive (TP) calcolati per ogni famiglia. Da questa matrice si evince che la nostra rete categorizza molto bene i vari malware tra di loro,infatti ci sono pochissimi falsi positivi e negativi.

```
[[59  4  0  0  0]
 [ 0 65  1  0  3]
 [ 2  2 70  1  0]
 [ 0  0  0 52  0]
 [ 1  0  0  1 54]]
```

*Figura 29 Confusion Matrix*

Di seguito,nella *Figura 30*, c'è il codice per stampare le due matrici.

```
printLoss(history)
printAccuracy(history)
predictions=model.predict_classes(X_test)
print(classification_report(Y_test_label,predictions))
print(confusion_matrix(Y_test_label,predictions))
```

*Figura 30 Codice per la stampa delle due matrici Confusion e Classification*

Inoltre sono stati generati anche i grafici che mostrano l'accuratezza e la loss,utilizzando la libreria *matplotlib*.Nella *Figura 31* è presente il modello della Loss, nella *Figura 32* quello dell'Accuracy. Dal modello della Loss si evince che è presente un pò di overfitting all'interno della rete,questo è dovuto al fatto che il campione di dati utilizzati per fare l'analisi è troppo ridotto. Lo stesso vale anche nel caso dell'Accuracy.

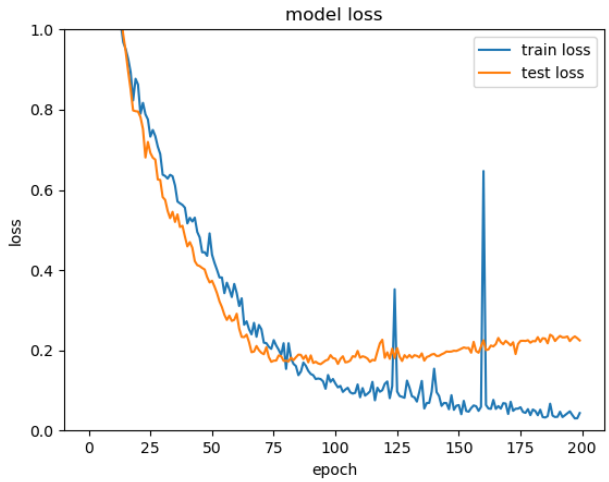



*Figura 31 Model Loss*

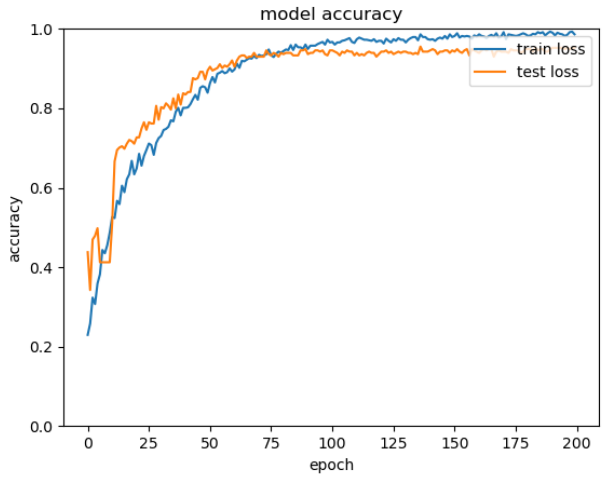



*Figura 32 Model Accuracy*

Il codice utilizzato per generare i grafici viene mostrato nelle *Figure 33 e 34*.

```
def printLoss(history):

    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.ylim(0, 1)
    plt.title("model loss")
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train loss', 'test loss'], loc='upper right')
    plt.show()
```

*Figura 33 Codice per stampare la Model Loss*

```
def printAccuracy(history):

    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.ylim(0, 1)
    plt.title("model accuracy")
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train loss', 'test loss'], loc='upper right')
    plt.show()
```

*Figura 34 Codice per stampare la Model Accuracy*

Il codice completo della rete viene riportato nell'Appendice A.

# CAPITOLO 6
# CONCLUSIONI E SVILUPPI FUTURI

In questo lavoro di tesi è stata affrontata la tematica del Malware Analysis nell'ambiente Android attraverso l'utilizzo di una rete neurale convoluzionale (CNN). La rete è stata addestrata sulle matrici sparse che rappresentano immagini bidimensionali (dette API-image). Queste immagini sono state costruite a partire dall'analisi dinamica eseguita su CuckooDroid. Attraverso l'utilizzo della CNN sono stati ottenuti buoni risultati di classificazione. E' possibile identificare diversi miglioramenti futuri relativi all'implementazione pratica di questo progetto:

- **Implementare l'estrazione delle features in modalità inline**: attualmente, l'estrazione delle features viene eseguita dopo che i file sono stati eseguiti nella sandbox e sono stati generati i report. Questo approccio comporterà ritardi nell'analisi dei file una volta implementato. Invece, sarebbe piu conveniente estrarre le features man mano che vengono elaborate dalla sandbox, in modo che non sia necessario esaminare nuovamente i report.
- **Utilizzare un Dataset piu ampio**: Il dataset utilizzato in questo progetto conteneva 1050 malware che ovviamente sono troppo limitati rispetto ad applicazioni del mondo reale. Per ottenere risultati piu accurati sarebbe meglio quindi lavorare su dataset piu ampi.

# APPENDICE A

```
import pickle
import numpy as np
import seaborn as sns
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense,MaxPool2D,Conv2D,Flatten,Activation,Dropout
from tensorflow.keras.utils import to_categorical
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix,classification_report
```

```
def printLoss(history):

    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.ylim(0, 1)
    plt.title("model loss")
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train loss', 'test loss'], loc='upper right')
    plt.show()

def printAccuracy(history):

    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.ylim(0, 1)
    plt.title("model accuracy")
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train loss', 'test loss'], loc='upper right')
    plt.show()
```

```
if __name__ == '__main__':

    #carichiamo le matrici dal file
    with open("matrci_fin_out2.txt", "rb") as matrici:
        matrci_sparse = pickle.load(matrici)

    #carichiamo la label dal file
    with open("array_label.txt", "rb") as array:
        array_label2 = pickle.load(array)

    y_example = to_categorical(array_label2, 5)

    #dividiamo in training e test set e dividiamo per 255,per diminuire i valori dell input
    X_train,X_test,y_train,y_test=train_test_split(matrci_sparse,y_example,test_size=0.3,random_state=42
    Y_train_label,Y_test_label=train_test_split(array_label2,test_size=0.3,random_state=42)

    X_train = [x /255 for x in X_train]
    X_test = [x /255 for x in X_test]

     #trasformiamo in oggetti numpy
     X_train=np.array(X_train)
     X_test=np.array(X_test)
     y_train=np.array(y_train)
     y_test=np.array(y_test)
     Y_train_label=np.array(Y_train_label)
     Y_test_label=np.array(Y_test_label)
     print(Y_train_label.shape)

     #reshape in quanto abbiamo un immagine in greyscale con valori da 0 a 255
     X_train=X_train.reshape(735, 113, 113,1)
     X_test=X_test.reshape(315,113,113,1)

    # implementazione della rete
    model = Sequential()
    model.add(Conv2D(filters=32, kernel_size=(4, 4), input_shape=(113, 113, 1), activation="relu"))
    model.add(MaxPool2D(pool_size=(2, 2)))
    model.add(Conv2D(filters=64, kernel_size=(4, 4),activation="relu"))
    model.add(MaxPool2D(pool_size=(2, 2)))
    model.add(Flatten())
    model.add(Dense(120, activation="relu"))
    model.add(Dropout(0.5))
    model.add(Dense(120, activation="relu"))
    model.add(Dropout(0.5))
    model.add(Dense(120, activation="relu"))
    model.add(Dropout(0.3))
    #output layer
    model.add(Dense(5,activation="softmax"))
    model.compile(loss="categorical_crossentropy",optimizer="adam",metrics=["accuracy"])

    history = model.fit(X_train,y_train,epochs=200,batch_size=380,
                        validation_data=(X_test,y_test),verbose=2, shuffle=True)
```

```
85      printLoss(history)
86      printAccuracy(history)
87      predictions=model.predict_classes(X_test)
88      print(classification_report(Y_test_label,predictions))
89      print(confusion_matrix(Y_test_label,predictions))
90
```

# RIFERIMENTI BIBLIOGRAFICI


[1] Aurélien Géron,"Hands-On Machine Learning with Scikit-Learn, Keras, and Tensorflow",O'Reilly(2019)

[2] Gopinath Rebala, Ajay Ravi, Sanjay Churiwala,"An Introduction to Machine Learning"

[3] Gianni D'Angelo,Massimo Ficco,Francesco Palmieri "Malware detection in mobile environments based on Autoencoders and API-images" Journal of Parallel and Distributed Computing 137 (2020) 26–33

[4] Sanchit Gupta1, Harshit Sharma and Sarvjeet Kaur "Malware Characterization Using Windows API Call Sequences"

[5] Nawfal Turki Obeis and Wesam Bhaya "Malware Analysis Using Apis Pattern Mining"

[6] Mohit Sewak, Md.Rezaul Karim and Pradeep Pujari "Practical Convolutional Neural Networks: Implement advanced deep learning models using Python"

[7] Aditya Anand "Malware Analysis 101 - Sandboxing" Medium.com

[8] Adam Geitgey "Machine Learning is Fun!" Medium.com

[9] Iffat Zafar, Giounona Tzanidou , Richard Burton , Nimesh Patel "Hands-on Convolutional Neural Networks with Tensorflow"

[10] Purnasai Gudikandula "A Beginner Intro to Neural Networks" Medium.com

[11] Manuel Torres "IL PROCESSO DI DATA SCIENCE: MODELLI DI MACHINE LEARNING IN AZIONE" Techedgegroup.com

[12] CuckooDroid cuckoo-droid.readthedocs.io

[13] Sumit Saha "A Comprehensive Guide to Convolutional Neural"