Università degli Studi di Padova

Dipartimento di Matematica "Tullio Levi-Civita"

Corso di Laurea in Informatica

Decoding GAN-Generated Malware using Explainable AI Techniques

Tesi di laurea

Relatore
Prof. Alessandro Galeazzi

Correlatore
Prof. Vinod

Laureando
Tiozzo Matteo

Matricola 2042882

Anno Accademico 2023-2024

"I'm convinced that about half of what separates the successful entrepreneurs from the non-successful ones is pure perseverance"

— Steve Jobs

Sommario

Il presente documento descrive il lavoro svolto durante il periodo di stage, della durata complessiva di trecento ore, dal laureando Matteo Tiozzo presso l'Università degli studi di Padova. Il tirocinio è stato condotto sotto la guida del Prof. Alessandro Galeazzi e con la collaborazione del Prof. P.Vinod. Il Prof. Alessandro Brighente ha ricoperto il ruolo di tutor accademico e referente interno al Consiglio del Corso di Studio.

Questa tesi riguarda l'analisi di malware attraverso le Generative Adversarial Network (GAN) nel contesto di malware moderni con la particolare attenzione alle famiglie di malware Windows. Lo studio ha esplorato i pattern che i malware possiedono in modo da poter effettuare un riconoscimento approfondito del malware.

Il tirocinio si è suddiviso in due parti. La prima dedicata alla preparazione del dataset di malware, con conseguente scaricamento dello stesso, suddivisione e analisi dei pattern e caratteristiche di ciascuna famiglia di malware, nonché alla generazione delle immagini in scala di grigi per poter addestrare le GAN. La seconda parte consiste nell'analisi delle tecnologie GAN e nella loro attuale implementazione per la generazione e individuazione di malware. Inoltre la seconda parte ha anche provveduto all'addestramento delle GAN e alla valutazione dei risultati ottenuti.

Ringraziamenti

Giunto al termine di questo mio percorso di studi desidero ringraziare chi mi ha sostenuto e accompagnato in questi anni.

In primis, desidero porgere la mia gratitudine al Prof. Alessandro Galeazzi, relatore della mia tesi, per avermi lasciato la libertà di scegliere un progetto che mi appassionasse realmente e per avermi guidato e supportato durante lo svolgimento di questo lavoro. Un ringraziamento speciale va anche al Prof. Vinod, per avermi aiutato durante lo sviluppo del codice e per avermi fornito preziosi consigli e suggerimenti su come procedere.

Un ringraziamento speciale va ai miei genitori, che hanno reso possibile tutto questo mediante i loro, non pochi, sacrifici. Grazie per avermi sempre sostenuto e per avermi dato la libertà di scegliere il mio percorso. Un ringraziamento particolare va anche a mia sorella, per avermi fornito consigli utili per superare le difficoltà che ho incontrato.

Ringrazio i miei nonni, che nonostante molte volte non capissero cosa stessi studiando mi hanno comunque sostenuto e spronato a dare il meglio di me, permettendomi di arrivare fin qui.

Un caloroso ringraziamento va anche ai miei amici, quelli di una vita e quelli conosciuti durante il percorso universitario, per avermi aiutato nei momenti di difficoltà e per aver condiviso con me esperienze che hanno reso questi anni indimenticabili.

Un grazie sentito lo devo a Diletta, nonostante oramai non stiamo più insieme è stata la persona che più mi ha sostenuto, supportato e sopportato durante questi anni. Grazie davvero.

Infine, un ringraziamento va anche a 3 dei miei compagni del gruppo 7Last, diventato ormai 4Last, con i quali ho passato gli ultimi mesi della mia carriera universitaria.

Padova, Dicembre 2024 Tiozzo Matteo

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

1.1 Motivazione

Negli ultimi anni, il panorama delle minacce informatiche ha subito un'evoluzione significativa, con attacchi malware sempre più sofisticati e distruttivi. Episodi come WannaCry, NotPetya, Stuxnet, Mirai e l'attacco a Travelex hanno evidenziato la vulnerabilità delle infrastrutture digitali globali, causando danni economici ingenti e interruzioni operative su larga scala. Per contrastare efficacemente queste minacce in continua evoluzione, è fondamentale sviluppare strumenti avanzati e metodologie efficaci per identificare e neutralizzare tali pericoli. In questo contesto, le Generative Adversarial Network (GAN) emergono come una soluzione promettente. Le GAN, composte da un generatore e un discriminatore in competizione, possono essere utilizzate per generare campioni di malware realistici. Questi campioni sintetici arricchiscono i dataset di addestramento, migliorando la capacità dei modelli di machine learning di riconoscere e classificare varianti di malware, inclusi quelli sconosciuti o zero-day. L'adozione delle GAN nel rilevamento del malware offre diversi vantaggi:

- **Miglioramento della robustezza dei modelli**: L'esposizione a una gamma più ampia di esempi durante l'addestramento consente ai modelli di identificare con maggiore precisione comportamenti malevoli.
- **Adattamento alle nuove minacce**: Le GAN possono generare varianti di malware che simulano tecniche di offuscamento avanzate, preparando i sistemi di difesa a riconoscere e bloccare nuove minacce.
- **Riduzione della dipendenza da dataset etichettati**: La generazione di campioni sintetici allevia la necessità di grandi quantità di dati etichettati, spesso difficili da ottenere.

In conclusione, l'integrazione delle GAN nelle strategie di cybersecurity rappresenta un passo avanti significativo nella protezione contro le minacce informatiche moderne. Questa tecnologia offre un approccio proattivo e adattivo, essenziale per affrontare l'evoluzione continua del panorama delle minacce.

1.2 Organizzazione del testo

Di seguito, viene presentata la struttura del documento:

Il secondo capitolo fornisce le basi teoriche e concettuali necessarie per comprendere le metodologie e le tecniche utilizzate nell'analisi del malware con le Generative Adversarial Networks (GAN).

Il terzo capitolo presenta quanto trovato di simile nella letteratura attuale.

Il quarto capitolo approfondisce il background e illustra l'idea del progetto.

Il quinto capitolo descrive dettagliatamente l'ambiente di sviluppo e presenta i singoli esperimenti condotti.

Il sesto capitolo riassume e analizza i risultati ottenuti dagli esperimenti.

Il settimo capitolo presenta le conclusioni del lavoro e propone possibili sviluppi futuri.

Capitolo 2

Background

2.1 Malware

Il termine "malware" deriva dall'unione delle parole inglesi *malicious* e *software* e indica qualsiasi programma o codice progettato per infiltrarsi, danneggiare o ottenere l'accesso non autorizzato a un sistema informatico. I malware possono essere creati con diversi scopi, come il furto di informazioni sensibili, il sabotaggio di sistemi, la visualizzazione di pubblicità indesiderata, o il ricatto delle vittime.

2.1.1 Analisi del Malware

Con l'escalation della complessità delle minacce informatiche, è diventato cruciale sviluppare metodologie avanzate per l'analisi dei malware. L'analisi dei malware è il processo sistematico di esame e comprensione di questi software per determinarne il funzionamento, l'origine, gli obiettivi e le vulnerabilità sfruttabili per una difesa efficace.
L'**analisi** dei malware si articola principalmente in due categorie: analisi **statica** e analisi **dinamica**.

2.1.2 Analisi Statica

L'analisi statica del malware implica l'esame del codice sorgente o dei file binari senza eseguirli, permettendo agli analisti di identificare caratteristiche intrinseche del malware. Questo metodo comprende l'analisi di stringhe di testo, importazioni di API, strutture di codice e pattern specifici che possono suggerire attività malevole. Strumenti quali disassemblatori (per esempio, IDA Pro) e decompilatori (come Ghidra) sono essenziali per facilitare questa analisi, convertendo il codice binario in una forma più leggibile e interpretabile. Tuttavia, l'analisi statica incontra notevoli sfide. I creatori di malware adottano spesso tecniche di offuscamento e cifratura per mascherare il vero comportamento del codice, complicando l'interpretazione sia manuale che automatizzata. Tecniche come il packing, il polimorfismo e il metamorfismo sono esempi di metodi usati per aumentare la resistenza del malware all'analisi statica. Le Generative Adversarial Network (GAN) possono essere utilizzate per generare automaticamente campioni sintetici di malware, ampliando i dataset disponibili per l'addestramento dei modelli di machine learning. Questo approccio non solo arricchisce

la varietà dei campioni analizzati ma migliora anche la comprensione delle diverse varianti e tecniche di evasione impiegate dai creatori di malware.

2.1.3 Analisi Dinamica

L'analisi dinamica consiste nell'esecuzione del malware all'interno di un ambiente controllato, come una sandbox, per osservare il suo comportamento in tempo reale. Questo metodo consente di identificare azioni malevole quali modifiche al registro di sistema, creazione di file, comunicazioni di rete sospette e interazioni con altre applicazioni. A differenza dell'analisi statica, l'analisi dinamica può rivelare comportamenti malevoli non immediatamente discernibili dall'analisi del codice sorgente, specialmente quando il malware impiega tecniche di offuscamento o cifratura. Nonostante la sua efficacia, l'analisi dinamica presenta delle limitazioni. Essa richiede risorse computazionali significative per eseguire i campioni di malware in ambienti isolati e per monitorarne le attività. Inoltre, alcuni malware sono capaci di riconoscere la presenza di ambienti di analisi e possono alterare il proprio comportamento per eludere la rilevazione, riducendo così l'efficacia dell'analisi. Le Generative Adversarial Networks (GAN) possono migliorare l'analisi dinamica in vari modi. Per esempio, possono essere utilizzate per simulare ambienti di esecuzione più realistici, rendendo più difficile per il malware riconoscere e adattarsi all'ambiente di analisi. Inoltre, possono generare traffico di rete sintetico e scenari di attacco realistici, che aiutano gli analisti a testare e rafforzare le capacità di rilevamento delle sandbox. Infine, le GAN possono essere impiegate per creare modelli di comportamento dei malware, facilitando la previsione e l'identificazione di nuove varianti basate su schemi comportamentali precedentemente osservati.

2.1.4 Visualizzazione del Malware

La visualizzazione del malware è una tecnica avanzata che sfrutta rappresentazioni grafiche per facilitare l'analisi della struttura e del comportamento dei codici malevoli. Strumenti visivi quali diagrammi di flusso, grafici delle chiamate, mappe di rete e rappresentazioni grafiche delle interazioni tra i componenti sono ampiamente utilizzati in questo ambito. Questi strumenti aiutano gli analisti a identificare pattern ricorrenti, a riconoscere rapidamente le attività sospette e a comprendere le complesse relazioni intrinseche al codice. Una rappresentazione visiva efficace può notevolmente accelerare il processo di analisi, consentendo agli analisti di scoprire anomalie e comportamenti insidiosi che potrebbero non emergere chiaramente attraverso l'analisi testuale o statica. Per esempio, un grafo delle chiamate API può rivelare sequenze anomale di chiamate indicative di attività di Command and Control (C&C) o di esfiltrazione di dati. L'integrazione delle Generative Adversarial Networks (GAN) nella visualizzazione del malware può innalzare il livello delle rappresentazioni a nuove vette di interattività e avanzamento. Questi strumenti sono capaci di generare visualizzazioni sintetiche che simulano scenari di attacco realistici, arricchendo così la comprensione degli analisti sulle possibili modalità di infiltrazione e propagazione del malware. Inoltre, l'uso delle GAN per creare visualizzazioni dinamiche, che si aggiornano in tempo reale con l'acquisizione di nuovi dati durante l'analisi dinamica, migliora notevolmente la capacità di rilevare anomalie e di interpretare le complesse dinamiche interne al codice, agevolando una risposta più tempestiva ed efficace contro le minacce emergenti.

Figura 2.1: *Immagini in scala di grigi di malware di famiglia Adware e Spyware*

2.2 Machine Learning

Il *machine learning* (ML) ha rivoluzionato il campo della sicurezza informatica, fornendo strumenti avanzati per la rilevazione e la classificazione del malware. Gli algoritmi di ML, sia supervisionati che non supervisionati, vengono addestrati su dataset di malware noti per identificare nuove minacce in modo automatico e scalabile. Tecniche quali alberi decisionali, macchine a vettori di supporto (SVM), metodi di clustering e reti neurali sono ampiamente impiegate in questo ambito. Gli algoritmi di ML supervisionati richiedono dataset etichettati, nei quali ogni campione è classificato come benigno o malevolo. Tali algoritmi apprendono a distinguere pattern e caratteristiche specifiche del malware, permettendo loro di classificare nuovi campioni basandosi sulle informazioni acquisite. Al contrario, gli algoritmi di ML non supervisionati operano senza etichette predefinite, identificando autonomamente cluster di comportamenti simili, il che facilita la scoperta di nuove varianti di malware non ancora classificate. Nonostante questi vantaggi, il ML tradizionale può essere limitato dalla qualità e dalla quantità dei dati disponibili. La carenza di campioni di malware etichettati può ridurre l'efficacia degli algoritmi di ML, compromettendo l'identificazione di nuove minacce. Inoltre, i modelli di ML possono essere vulnerabili a tecniche di evasione, nelle quali i creatori di malware modificano i loro campioni per eludere la rilevazione.

2.3 Deep Learning

Il *deep learning* (DL), una branca avanzata del machine learning, sfrutta reti neurali profonde per modellare e riconoscere pattern complessi nei dati. Architetture come le reti neurali convoluzionali (CNN) e le reti neurali ricorrenti (RNN) sono particolarmente efficaci nell'analisi di malware, data la loro capacità di estrarre caratteristiche di alto livello dai campioni di codice. Le CNN sono rinomate per il loro eccellente riconoscimento di pattern spaziali, mentre le RNN sono ottimali per l'analisi di sequenze temporali di dati, rendendole strumenti ideali per il rilevamento di comportamenti sospetti nei flussi di dati. Il DL ha ottenuto risultati notevoli nella classificazione e nella rilevazione di malware, anche quelli sconosciuti. Modelli di DL possono essere addestrati su vasti dataset di codici maligni e benigni per apprendere rappresentazioni

complesse, che aiutano a differenziare tra codice malevolo e non. Inoltre, le tecniche di *transfer learning* consentono di adattare modelli pre-addestrati a nuovi domini o varianti di malware, migliorando sia l'efficienza dell'addestramento sia la precisione delle predizioni. L'introduzione delle *Generative Adversarial Networks* (GAN) nel deep learning può ulteriormente amplificare questi successi. Le GAN possono essere utilizzate per generare nuovi campioni di malware che mettono alla prova i modelli di rilevazione esistenti, facilitando una continua evoluzione e miglioramento dei modelli di DL. Questo processo di *adversarial training* rende i modelli di DL più robusti contro le tecniche di evasione utilizzate dai creatori di malware. In aggiunta, le GAN possono essere impiegate per creare rappresentazioni avanzate di dati di malware, potenziando la capacità dei modelli di DL di estrarre caratteristiche significative e di generalizzare a nuovi tipi di minacce.

2.4 Riproducibilità

La riproducibilità è un principio cardinale nella ricerca scientifica, essenziale per garantire che i risultati ottenuti siano verificabili e replicabili da altri ricercatori. Nel campo dell'analisi del malware, ciò implica la disponibilità di dataset, codici sorgente, configurazioni di esperimenti e ambienti di esecuzione utilizzati negli studi, consentendo ad altri di validare i risultati, confrontare metodologie diverse e costruire su scoperte esistenti. Tuttavia, la natura dinamica delle minacce informatiche può ostacolare la riproducibilità. I dataset di malware possono rapidamente diventare obsoleti con l'emergere di nuove varianti e tecniche di evasione. Le discrepanze negli ambienti di analisi, nelle configurazioni dei tool e nelle versioni del software possono altresì alterare i risultati degli esperimenti, rendendo ardua la replicazione esatta degli studi. L'adozione delle *Generative Adversarial Networks* per generare dataset sintetici standardizzati può essere di grande aiuto nella condivisione e replicazione degli esperimenti. Le GAN sono in grado di produrre campioni di malware che riflettono le caratteristiche dei dataset originali ma introducono variazioni controllate per aumentare la diversità e rappresentatività. Questo metodo assicura che i dataset rimangano attuali e pertinenti anche di fronte a nuove minacce. Inoltre, le GAN possono essere utilizzate per simulare scenari di attacco complessi, facilitando la verifica delle metodologie in ambienti controllati e riproducibili. La creazione di framework open-source che integrano GAN e altre tecnologie di analisi del malware può ulteriormente promuovere la trasparenza e la condivisione di conoscenze. Rendendo accessibili i codici sorgente e le configurazioni sperimentali, si permette ad altri ricercatori di replicare gli studi, identificare bias o limitazioni nei metodi esistenti e suggerire miglioramenti. Questo incoraggia una collaborazione più efficace all'interno della comunità scientifica per affrontare le sfide emergenti nel campo della sicurezza informatica.

2.5 Explainability

L'interpretabilità, o explainability, dei modelli di machine learning e deep learning è fondamentale per comprendere come e perché questi modelli prendono specifiche decisioni. Nel campo della sicurezza informatica, è cruciale che gli analisti siano in grado di interpretare i risultati degli algoritmi di rilevazione del malware per prendere decisioni informate e comprendere le motivazioni dietro le segnalazioni di minacce. Tuttavia, modelli complessi come le reti neurali profonde spesso agiscono come "scatole nere", rendendo arduo discernere il ragionamento dietro le loro predizioni. L'interpre-

tabilità assume un'importanza particolare in contesti dove le decisioni devono essere giustificate, come in ambiti aziendali o legali, dove dimostrare la validità delle analisi è imprescindibile. Senza una chiara comprensione del funzionamento interno dei modelli, può essere difficile identificare errori o bias, limitando così la fiducia e l'adozione di queste tecnologie da parte di analisti e decision makers. Le *Generative Adversarial Networks* (GAN), nonostante siano strumenti generativi potenti, presentano notevoli sfide in termini di interpretabilità. Composte da due reti neurali — il generatore e il discriminatore — che competono tra loro, le GAN complicano il tracciamento e la comprensione dei processi interni che guidano la generazione dei campioni sintetici. Questa complessità può ostacolare la capacità di spiegare come e perché un campione di malware sintetico è stato generato in un determinato modo, o di distinguere le caratteristiche che separano un campione benigno da uno malevolo. Per superare queste sfide, è vitale sviluppare metodi che aumentino la trasparenza nel processo decisionale delle GAN. Tecniche come l'analisi delle attivazioni interne, la visualizzazione delle rappresentazioni latenti e l'impiego di modelli intrinsecamente interpretabili possono migliorare significativamente l'interpretabilità delle GAN. Inoltre, l'adozione di approcci ibridi che combinano GAN con modelli più interpretabili può facilitare una migliore comprensione delle dinamiche interne, permettendo agli analisti di delineare le origini delle decisioni prese dai modelli generativi. L'applicazione di tecniche di spiegazione post-hoc, come LIME (Local Interpretable Model-agnostic Explanations) o SHAP (SHapley Additive exPlanations), ai modelli di GAN offre un ulteriore strumento per fornire spiegazioni locali delle predizioni. Questi metodi aiutano a chiarire quali caratteristiche dei dati di input hanno maggiormente influenzato la generazione dei campioni, offrendo agli analisti insight preziosi per interpretare e validare i risultati. La ricerca nell'ambito dell'*explainable artificial intelligence* (XAI) sta sviluppando nuove strategie per rendere i modelli generativi più trasparenti e comprensibili. Integrare i principi di XAI nelle architetture delle GAN può portare alla creazione di modelli che non solo generano campioni realistici ma che forniscono anche spiegazioni dettagliate e comprensibili delle loro operazioni. Questo passo è fondamentale per incrementare la fiducia degli analisti nella tecnologia e promuovere un'adozione più ampia delle GAN nell'analisi del malware. In conclusione, migliorare l'interpretabilità è un obiettivo cruciale per assicurare che le GAN e altri modelli avanzati di machine learning possano essere impiegati efficacemente e affidabilmente nell'analisi del malware. Una comprensione chiara e trasparente dei processi decisionali interni è essenziale per sfruttare appieno il loro potenziale e garantire la sicurezza e l'affidabilità delle analisi condotte.

Capitolo 3

Related Works

Masataka Kawai, nel 2019, hanno sviluppato una versione migliorata di MalGAN, denominata Improved MalGAN, che mira a eludere i rilevatori di malware incorporando caratteristiche di software legittimi, ovvero cleanware, nelle funzionalità di malware. Questa ricerca, presentata al Muroran Institute of Technology, utilizza un approccio basato su Generative Adversarial Networks (GAN) per generare quantità di caratteristiche che vengono classificate come benign da un rilevatore di malware, mentre conservano la funzionalità del malware originale. L'approccio proposto migliora significativamente le tecniche esistenti impiegando un metodo di apprendimento differenziato che utilizza una singola istanza di malware, a differenza di approcci precedenti che necessitavano di multiple istanze. Questo metodo consente non solo un addestramento più realistico e praticabile per gli attaccanti, ma offre anche prestazioni migliorate in termini di evasione dai sistemi di rilevamento basati su machine learning. I risultati sperimentali indicano che l'Improved MalGAN supera le configurazioni precedenti in termini di performance di evasione, evidenziando la vulnerabilità dei metodi di rilevamento del malware esistenti e la necessità di continuare a sviluppare contromisure più robuste.

Pubblicato su Computers & Security nel 2023, Fascía et al. hanno esplorato l'uso delle Generative Adversarial Networks (GAN) per bypassare le tecniche di rilevazione del malware basate sulla visualizzazione. Queste tecniche, che trasformano file eseguibili in immagini per l'analisi tramite deep learning, risultano vulnerabili alle strategie avanzate di obfuscamento implementate tramite GAN. Gli autori hanno sviluppato una GAN che modifica varianti di malware per nascondere i pattern rilevabili, raggiungendo un tasso di successo del 100% nel loro dataset testato, sottolineando la necessità di evolvere continuamente le tecnologie di rilevazione del malware per affrontare minacce in evoluzione. Questo lavoro evidenzia le vulnerabilità delle tecniche di visualizzazione attuali e stimola lo sviluppo di sistemi più robusti e avanzati.

Nel 2024, pubblicato dall'IEEE, R et al. hanno esplorato l'applicazione delle Deep Neural Networks (DNN) per la classificazione di malware nel contesto del computing perimetrale, dove i dispositivi sono spesso limitati in termini di risorse computazionali. L'articolo analizza diverse architetture DNN, come ResNet, DenseNet, InceptionNet, Xception e MobileNet, evidenziando come ciascuna di queste possa influenzare l'efficienza e l'efficacia della classificazione di malware in tempo reale. Questo studio dimostra il potenziale delle DNN nel migliorare notevolmente la precisione e l'efficacia della rilevazione di malware, classificando con alta accuratezza vari tipi di malware e

sottolineando l'importanza di ottimizzare le prestazioni per il loro utilizzo in dispositivi con risorse limitate. La ricerca sottolinea l'importanza di un rilevamento precoce del malware per prevenire conseguenze negative, e propone un'innovativa distribuzione di compiti di sicurezza su dispositivi periferici per mantenere l'integrità e la disponibilità dei sistemi IoT su larga scala. Inoltre, viene discussa l'applicabilità delle DNN in dispositivi edge, considerando il tempo di computazione e la latenza nella classificazione dei binari di malware, con l'obiettivo di avanzare le procedure di cybersecurity e di rendere gli ecosistemi digitali più resilienti e sicuri contro le minacce cyber sempre più crescenti.

Nel 2024, Sharma et al. hanno introdotto un'innovativa architettura di Generative Adversarial Network (GAN) chiamata MIGAN, per la sintesi di immagini di malware e la classificazione migliorata su un nuovo dataset. MIGAN supera le tradizionali tecniche di riconoscimento malware basate sul machine learning che richiedono notevoli competenze di dominio o analisi comportamentale dispendiosa in termini di tempo. Il framework proposto include una rete generatrice e discriminante accoppiata a un modulo di classificazione. La novità risiede nella struttura della rete GAN, nella funzione di perdita ibrida, nel nuovo dataset e nella struttura della rete di classificazione. Gli autori hanno generato circa 50.000 immagini di malware sintetiche, con le quali hanno addestrato due reti di classificazione: una rete di classificazione personalizzata e una rete Resnet50v2 preaddestrata utilizzando un approccio di transfer learning. I risultati dimostrano l'efficacia del framework, raggiungendo un'Area Under the Curve (AUC) del 99.2%, un punteggio F1 del 99.3% e un'accuratezza del 99.5%. Inoltre, il sistema ha mostrato di poter mitigare efficacemente il problema dell'imbilanciamento delle classi nei dataset Malimg e in quelli personalizzati. Questo studio non solo affronta il problema dell'imbilanciamento delle classi ma apre anche nuove prospettive per lo sviluppo di sistemi di riconoscimento e classificazione di malware efficienti, a basso costo e altamente affidabili per ambienti non controllati.

Capitolo 4

Descrizione del Progetto

Questo capitolo introduce i preliminari tecnici e illustra il problema affrontato. La sezione 4.1 è dedicata alla descrizione del contesto tecnologico mentre la sezione 4.2 analizza le problematiche affrontate nello studio.

4.1 Scenario Tecnico

4.1.1 Malware

Il termine "malware" deriva dall'unione delle parole inglesi *malicious* e *software*[5] e indica qualsiasi programma o codice progettato per infiltrarsi, danneggiare o ottenere l'accesso non autorizzato a un sistema informatico. I malware possono essere creati con diversi scopi, come il furto di informazioni sensibili, il sabotaggio di sistemi, la visualizzazione di pubblicità indesiderata, o il ricatto delle vittime. Esistono varie tipologie di malware, ognuna con specifiche caratteristiche e metodi di azione.

4.1.1.1 Famiglie di Malware

Esistono molte famiglie di malware, ognuna con caratteristiche particolari e comportamenti distinti. Tra le più conosciute, possiamo citare:

- **Adware**: Nasce dall'unione di *advertising* e *software* ed è un'applicazione software in cui un banner pubblicitario o altro materiale pubblicitario vengono visualizzati e scaricati mentre un programma è in esecuzione. L'adware legittimo viene scaricato con il consenso esplicito degli utenti. Questi ultimi scaricano consapevolmente questa forma di adware e di solito ottengono qualcosa in cambio. Potrebbero ottenere uno sconto o software gratuito in cambio della ricezione dell'adware. Gli annunci aiutano a coprire i costi di sviluppo del software o consentono allo sviluppatore di fornire il prodotto gratuitamente. L'adware è malevolo quando viene progettato con l'intento di fornire software dannoso all'utente. Il software viene classificato come spyware se viene eseguito all'insaputa e senza l'autorizzazione dell'utente. I dati degli utenti raccolti in questo modo vengono spesso venduti a terzi.
- **Backdoor**: Un tipo di malware che consente l'accesso remoto non autorizzato a un sistema, permettendo all'attaccante di controllare il dispositivo senza che l'utente ne sia a conoscenza. Tra i potenziali rischi vi è l'installazione di keylogger

nel computer, strumenti che registrano ogni tasto premuto, consentendo ai malintenzionati di sottrarre dati sensibili come le password degli account.

- **Downloader**: Malware progettato per scaricare e installare altri malware su un dispositivo infetto. Spesso funge da primo stadio di un'infezione più complessa, preparando il terreno per l'installazione di altri malware come trojan, ransomware o spyware.
- **Ransomware**: Malware progettato per criptare i dati dell'utente e richiedere un riscatto per la decrittazione. Spesso utilizza tecniche di estorsione per costringere le vittime a pagare somme di denaro in criptovalute.
- **Spyware**: Malware progettato per raccogliere informazioni sull'utente senza il suo consenso, come dati personali, cronologia di navigazione e credenziali, per poi inviarle all'attaccante.
- **Trojan (Cavalli di Troia)**: Programmi che si mascherano come software legittimi per ingannare gli utenti e indurli a installarli. Una volta installati, possono dare accesso al sistema o danneggiarlo.
- **Virus**: Software che si replica infettando altri file eseguibili, spesso con l'intento di danneggiare il sistema o rubare informazioni.
- **Worms**: Simili ai virus, ma non necessitano di un file host per propagarsi. Possono diffondersi autonomamente attraverso la rete, sfruttando vulnerabilità di sicurezza.
- **Rootkit**: Malware progettato per nascondere la presenza di altri malware o del proprio codice su un sistema, integrandosi profondamente nel sistema operativo e rendendosi quasi invisibile agli strumenti di rilevamento tradizionali.
- **Keylogger**: Un tipo di malware che registra ogni battitura dell'utente su tastiera per rubare informazioni sensibili come password e credenziali di accesso.
- **Botnet**: Una rete di dispositivi infetti che possono essere controllati a distanza per compiere azioni malevole, come attacchi DDoS e spam.
- **Fileless Malware**: Malware che non scrive alcun file sul disco, utilizzando invece la memoria e strumenti già esistenti nel sistema operativo, rendendo più difficile la rilevazione.
- **Scareware**: Malware che inganna l'utente facendogli credere che il sistema sia infettato per indurlo a comprare software falsi o fornire informazioni personali.
- **RAT (Remote Access Trojan)**: Trojan che permette all'attaccante di ottenere accesso remoto e controllo completo del dispositivo infetto, consentendo di spiare l'utente e rubare informazioni.
- **Malvertising**: Uso di annunci pubblicitari legittimi per distribuire malware, spesso sfruttando reti pubblicitarie per diffondere link malevoli.
- **Cryptojacker**: Malware che utilizza le risorse di calcolo della vittima per effettuare il mining di criptovalute senza che l'utente ne sia consapevole.
- **Logic Bomb**: Malware che rimane inattivo fino a quando non viene attivato da un evento specifico, come una data o una certa condizione, per poi eseguire azioni malevole come cancellare file o causare crash del sistema.

4.1.1.2 Struttura del malware

La struttura tipica di un malware varia in base alla tipologia di famiglia a cui il malware appartiene, le più comuni sono:

- **Adware**: Include componenti per il monitoraggio delle abitudini di navigazione dell'utente, e un motore per la visualizzazione automatica di annunci pubblicitari.
- **Backdoor**: Contiene un modulo che consente l'accesso remoto non autorizzato, spesso implementato tramite una comunicazione criptata con il server di controllo dell'attaccante.
- **Downloader**: Dispone di un componente per contattare server remoti e scaricare ulteriori malware, spesso utilizzando tecniche di offuscamento per eludere la rilevazione.
- **Ransomware**: Presenta un sistema di criptazione che rende inaccessibili i dati dell'utente, accompagnato da un modulo che mostra un messaggio di riscatto e gestisce i pagamenti.
- **Spyware**: Ha moduli per il keylogging, il monitoraggio della cronologia di navigazione e la raccolta di credenziali, inviando poi queste informazioni agli attaccanti.
- **Trojan**: Include un componente di mimetizzazione, spesso mascherandosi da software legittimo, e può implementare diversi payload per danneggiare il sistema o esfiltrare dati.
- **Virus**: Contiene un modulo di replicazione che gli consente di infettare altri file eseguibili, spesso con funzionalità aggiuntive per danneggiare il sistema.
- **Worms**: Dispone di un motore di propagazione automatica, che sfrutta vulnerabilità note per diffondersi in modo autonomo tra i dispositivi connessi in rete.
- **Rootkit**: Include un modulo per nascondere la presenza del malware all'interno del sistema, modificando componenti del kernel o altri elementi critici del sistema operativo.
- **Keylogger**: Dispone di un modulo per catturare tutte le battiture dell'utente, salvandole in un file log da inviare all'attaccante.
- **Botnet**: Include componenti che consentono di trasformare il dispositivo in un "bot" controllato da remoto, spesso senza che l'utente ne sia consapevole.
- **Fileless Malware**: Utilizza strumenti nativi del sistema operativo (come PowerShell o WMI) per eseguire codice dannoso direttamente in memoria, rendendo più difficile la sua rilevazione.
- **Scareware**: Contiene un modulo per la visualizzazione di avvisi falsi che spingono l'utente ad acquistare software non necessario o dannoso.
- **RAT (Remote Access Trojan)**: Include strumenti per il controllo remoto, consentendo all'attaccante di navigare tra i file, attivare la webcam e installare altri malware.

- **Malvertising**: Contiene codice che viene iniettato tramite banner pubblicitari, spesso sfruttando vulnerabilità del browser per compromettere il sistema dell'utente.
- **Cryptojacker**: Include script di mining che si attivano all'insaputa dell'utente per utilizzare risorse di CPU e GPU per minare criptovalute.
- **Logic Bomb**: Include una condizione preimpostata (come una data specifica o un certo evento) che, una volta soddisfatta, attiva il payload dannoso.

4.1.1.3 Dataset

Nell'ambito di questo progetto di ricerca sono state prese in considerazione solo le prime sette famiglie di malware, in quanto le più comuni e pericolose. Per la raccolta del dataset sono stati utilizzati campioni di malware provenienti da diverse fonti, tra cui:

- **MalwareBazaar**: Una piattaforma gestita da abuse.ch e Spamhaus, dedicata alla condivisione di campioni di malware con la comunità di sicurezza informatica, fornitori di antivirus e provider di intelligence sulle minacce. Gli utenti possono caricare campioni di malware e esplorare il database per ottenere informazioni preziose. La piattaforma offre anche API per integrare i dati nei sistemi di gestione delle informazioni e degli eventi di sicurezza (SIEM) o in altre piattaforme di intelligence sulle minacce.
- **MalShare**: Un progetto comunitario che offre un repository pubblico di malware, fornendo accesso gratuito a campioni di malware e strumenti alla comunità di sicurezza informatica. Gli utenti possono cercare campioni utilizzando hash, nomi di file, nomi di regole Yara o fonti. La piattaforma è progettata per facilitare la condivisione e l'analisi dei campioni di malware tra i professionisti della sicurezza.
- **VirusShare**: Una piattaforma che fornisce accesso a un ampio repository di campioni di malware. È utilizzata da ricercatori di sicurezza informatica, analisti e organizzazioni per ottenere campioni da analizzare e studiare il comportamento del malware. La piattaforma è guidata dalla comunità, con campioni contribuiti da ricercatori e professionisti della sicurezza informatica di tutto il mondo.

Per la classificazione dei campioni di malware sono stati utilizzati diversi strumenti di analisi statica e dinamica, tra cui:

- **Virustotal**: Un servizio online che consente l'analisi di file e URL sospetti per identificare malware e altri tipi di contenuti dannosi. Fondato nel 2004 e acquisito da Google nel 2012, VirusTotal aggrega i risultati di oltre 70 motori antivirus e strumenti di scansione URL, offrendo una panoramica completa sulla sicurezza di un file o di un sito web. Gli utenti possono caricare file o inserire URL per ottenere un rapporto dettagliato che indica quali motori hanno rilevato minacce e quali no. VirusTotal fornisce anche API per l'integrazione con altri strumenti e servizi di sicurezza.
- **AVClass2**: Uno strumento automatico per l'estrazione di tag da etichette antivirus (AV), progettato per categorizzare campioni di malware su larga scala. Sviluppato per migliorare le capacità del suo predecessore, AVClass, AVClass2 non si limita a identificare solo i nomi delle famiglie di malware, ma estrae anche

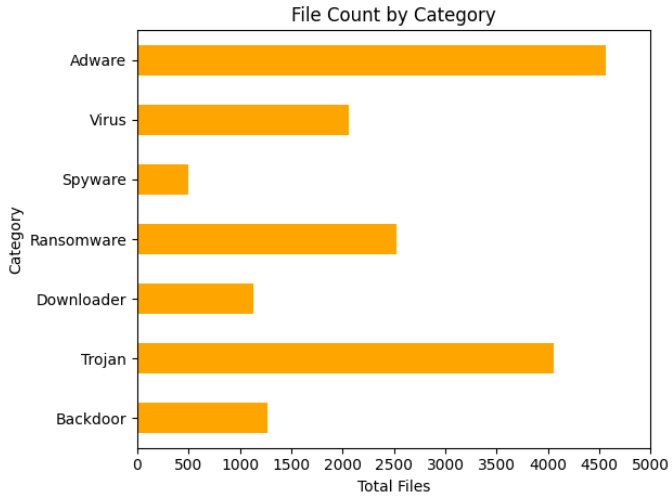

Figura 4.1: *Famiglie di malware presenti nel dataset del progetto di ricerca*

informazioni su classi di malware, comportamenti e proprietà dei file. Utilizza e contribuisce a costruire una tassonomia aperta che organizza i concetti presenti nelle etichette AV, permettendo di gestire nuovi tag introdotti dai fornitori di antivirus nel tempo.

- **Ghidra**: Una suite di strumenti per l'ingegneria inversa di software, sviluppata dalla National Security Agency (NSA) degli Stati Uniti e resa disponibile al pubblico nel 2019. Questa piattaforma open-source consente agli analisti di sicurezza di decompilare, disassemblare e analizzare binari compilati, facilitando la comprensione del funzionamento interno di software sospetti o malevoli. Ghidra supporta una vasta gamma di architetture e formati di file, offrendo funzionalità avanzate come un potente decompilatore e un'interfaccia utente grafica intuitiva.

Il dataset risultante è composto come segue:

Famiglia di Malware	**Numero di Campioni**
Adware	4566
Backdoor	1270
Downloader	1128
Ransomware	2529
Spyware	502
Trojan	4058
Virus	2061

Tabella 4.1: *Dataset di malware impiegato nel progetto di ricerca*

4.1.2 Decompilatore

Nonostante siano disponibili numerose tipologie di decompilatori, in questo progetto si è scelto l'utilizzo di **Ghidra** in quanto open source, già utilizzato in precedenza e compatibile con architettura arm presente nella macchina utilizzata durante tutto il tempo di svolgimento del progetto. È uno strumento di analisi di software, principalmente noto come un potente decompilatore e framework per il reverse engineering, sviluppato e distribuito gratuitamente dalla National Security Agency (NSA) degli Stati Uniti. Il suo utilizzo in questo progetto consiste in:

- Decompilazione del codice eseguibile in **codice assembly** su cui poter effettuare analisi statica in un secondo momento.
- Estrazione del **codice esadecimale**, facilitando ulteriori esami e manipolazioni del codice.

4.1.3 Convolutional Neural Network

Per l'addestramento di questa rete neurale è stato adottato Keras[6] con TensorFlow[7], una libreria open-source per il machine learning e il deep learning. L'architettura della rete neurale convoluzionale (CNN) utilizzata in questo progetto è composta da diversi strati, ciascuno con un ruolo specifico nel processo di apprendimento:

1. **Strati Convoluzionali (Conv2D)**: La rete inizia con tre strati convoluzionali, che sono responsabili dell'estrazione delle caratteristiche principali dalle immagini di input in scala di grigi (16x16 pixel). Primo strato convoluzionale: utilizza da 32 a 128 filtri (incremento di 32) con kernel di dimensione (3x3). Questo strato mantiene l'input della stessa dimensione (16x16) grazie al padding "same" e utilizza l'attivazione ReLU (Rectified Linear Unit). Secondo strato convoluzionale: utilizza da 32 a 64 filtri (incremento di 32) con kernel (3x3) e l'attivazione ReLU, seguito da un'operazione di MaxPooling2D con finestra (2x2), riducendo così la dimensione a (8x8). Terzo strato convoluzionale: impiega lo stesso numero di filtri (32-64, incremento di 32) e kernel, con un'ulteriore riduzione dimensionale tramite MaxPooling2D, portando la dimensione finale a (2x2).
2. **Strato di Pooling (MaxPooling2D)**: Gli strati di pooling riducono la dimensione spaziale delle feature map mantenendo solo le informazioni più rilevanti, riducendo il numero di parametri e migliorando l'efficienza computazionale. Sono utilizzati dopo ogni strato convoluzionale con finestre di dimensione (2x2).
3. **Strato di Flattening**: Dopo la fase convoluzionale, le feature map tridimensionali vengono "appiattite" in un vettore unidimensionale per poter essere processate dagli strati completamente connessi.
4. **Strato Completamente Connesso (Dense)**: Lo strato completamente connesso riceve l'input dallo strato di flattening. Qui, i neuroni collegano ogni unità precedente a quelle successive, consentendo alla rete di apprendere pattern complessi e interazioni tra le caratteristiche estratte. La dimensione del layer è variabile tra 32 e 128 unità, con attivazione ReLU. È anche presente un layer di Dropout, utilizzato per prevenire l'overfitting con un valore di dropout che può variare tra 0.0 e 0.5 (incremento di 0.1).
5. **Strato di Output (Dense)**: Lo strato finale è un layer completamente connesso con un numero di neuroni pari al numero di classi di malware presenti nel dataset,

con attivazione Softmax per ottenere una probabilità normalizzata per ciascuna classe.

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 16, 16, 64)	640
max_pooling2d (MaxPooling2D)	(None, 8, 8, 64)	0
conv2d_1 (Conv2D)	(None, 8, 8, 64)	36,928
max_pooling2d_1 (MaxPooling2D)	(None, 4, 4, 64)	0
conv2d_2 (Conv2D)	(None, 4, 4, 64)	36,928
max_pooling2d_2 (MaxPooling2D)	(None, 2, 2, 64)	0
flatten (Flatten)	(None, 256)	0
dense (Dense)	(None, 96)	24,672
dropout (Dropout)	(None, 96)	0
dense_1 (Dense)	(None, 7)	679

Total params: 99,849 (390.04 KB)
Trainable params: 99,847 (390.03 KB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 2 (12.00 B)

Figura 4.2: *Architettura della rete neurale convoluzionale (CNN) utilizzata*

4.1.3.1 Processo di addestramento

Per garantire un'efficace categorizzazione, il dataset è stato suddiviso in tre sottoinsiemi:

- **Training Set**: 80% del dataset, utilizzato per addestrare il modello.
- **Validation Set**: 10% del dataset, utilizzato per regolare i parametri del modello e prevenire l'overfitting.
- **Test Set**: 10% del dataset, utilizzato per valutare le prestazioni del modello su dati non visti.

Il modello è stato progettato utilizzando una rete neurale convoluzionale (CNN), ottimizzata tramite Keras Tuner per la selezione automatica dei migliori iperparametri. L'ottimizzazione ha incluso la scelta del numero di filtri nelle convoluzioni, le unità dense e il tasso di dropout. Tra i parametri testati vi sono stati i filtri per i layer convoluzionali (con valori da 32 a 128), il numero di unità dense (tra 32 e 128) e il tasso di dropout (da 0,0 a 0,5).

Una volta identificata la configurazione ottimale, il modello è stato addestrato per 100 epoche utilizzando un batch size di 32, adottando l'algoritmo di ottimizzazione Adam e la funzione di perdita categorical_crossentropy, adeguata per la classificazione multiclasse. Durante l'addestramento, la validazione incrociata è stata impiegata per monitorare le prestazioni del modello sui dati di validazione.

Le prestazioni del modello sono state valutate sul dataset di test mediante metriche quali accuratezza, precisione, richiamo e F1-score, calcolate sia in media micro che macro. La matrice di confusione è stata utilizzata per una visione dettagliata della capacità del modello di distinguere tra le diverse categorie di malware.

Per visualizzare l'efficacia dell'addestramento, sono stati generati grafici che mostrano l'andamento dell'accuratezza e della perdita sia per il training che per la validazione.

Questi risultati sono stati salvati in formato grafico e i relativi valori metrici in file di testo, evidenziando le prestazioni complessive del modello.

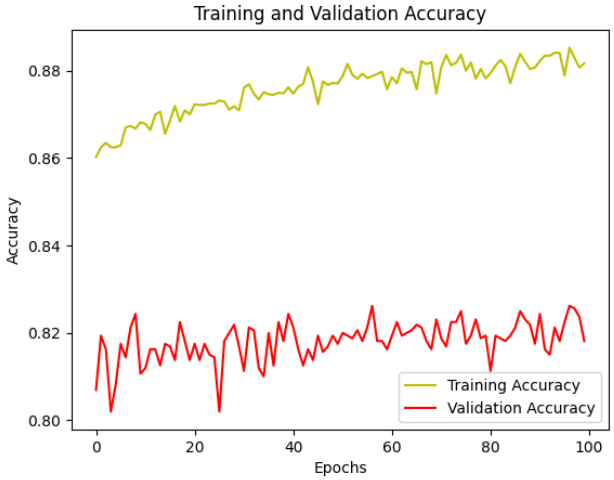

Figura 4.3: *Grafico dell'accuratezza del modello CNN durante l'addestramento e la validazione*

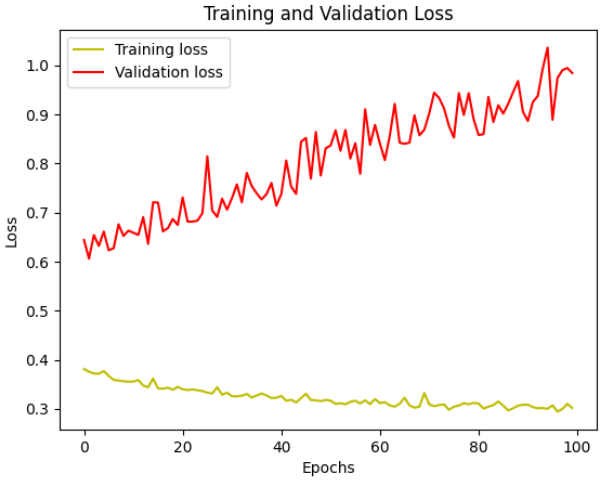

Figura 4.4: *Grafico della perdita del modello CNN durante l'addestramento e la validazione*

4.1.3.2 Metriche di Valutazione del Modello

Nel campo della classificazione, specialmente in scenari complessi come la classificazione del malware, è importante analizzare le prestazioni del modello utilizzando diverse metriche. Ogni metrica offre una prospettiva unica sui punti di forza e debolezza del modello, fornendo una visione completa della sua capacità di generalizzare e discriminare tra le classi. Di seguito, una descrizione approfondita delle metriche adottate.

4.1.3.2.1 Accuratezza

L'accuratezza è la metrica più intuitiva e rappresenta la proporzione di predizioni corrette rispetto al totale delle predizioni effettuate. È calcolata come:

$$\text{Accuratezza} = \frac{TP + TN}{TP + TN + FP + FN}$$

In questo progetto, l'accuratezza raggiunta è stata del **81.64%**, indicando che l'81.64% delle predizioni del modello sono risultate corrette. Tuttavia, in contesti di classificazione sbilanciata (quando alcune classi sono significativamente più numerose di altre), l'accuratezza potrebbe non riflettere la reale efficacia del modello. Pertanto, è importante considerare anche altre metriche.

4.1.3.2.2 Precisione

La precisione misura la capacità del modello di evitare falsi positivi. È definita come il rapporto tra i veri positivi (TP) e la somma dei veri positivi e dei falsi positivi (FP):

$$\text{Precisione} = \frac{TP}{TP + FP}$$

Una precisione elevata indica che il modello commette pochi errori nel classificare come "positiva" una classe. Ad esempio, una precisione alta per la classe "Ransomware" indica che quando il modello predice un sample come ransomware, ha una bassa probabilità di sbagliarsi.

Nel progetto, sono state utilizzate due versioni della precisione:

- **Macro Precision**: Calcola la precisione individualmente per ciascuna classe e poi ne fa la media aritmetica. È utile quando le classi sono bilanciate.
- **Micro Precision**: Considera globalmente il numero totale di veri positivi e falsi positivi su tutte le classi, fornendo una visione più accurata in caso di classi sbilanciate.

4.1.3.2.3 Richiamo/Recall

Il recall, o sensibilità, misura la capacità del modello di identificare correttamente tutti i veri positivi, cioè quanto il modello riesce a recuperare tutti gli esempi positivi:

$$\text{Recall} = \frac{TP}{TP + FN}$$

Un valore di recall elevato indica che il modello è in grado di identificare la maggior parte dei casi positivi, riducendo il numero di falsi negativi. Nel contesto del malware, un alto recall è fondamentale per evitare che potenziali minacce passino inosservate.

Anche qui, sono state adottate due varianti:

- **Macro Recall**: Calcola il recall per ogni classe e ne fa la media, non dando peso alla distribuzione delle classi.
- **Micro Recall**: Tiene conto del totale dei veri positivi e falsi negativi globalmente, utile per scenari con classi squilibrate.

4.1.3.2.4 F1-score

L'F1-score è la media armonica di precision e recall, bilanciando i due aspetti in un'unica metrica. È particolarmente utile quando si desidera trovare un compromesso tra falsi positivi e falsi negativi:

$$\text{F1-score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

Un valore di F1-score elevato suggerisce che il modello è bilanciato sia nella capacità di evitare falsi positivi che di ridurre i falsi negativi. Nel progetto, sono state utilizzate due varianti:

- **Macro F1-score**: Media l'F1-score di ciascuna classe, senza tener conto della distribuzione.
- **Micro F1-score**: Calcola l'F1-score considerando il numero totale di veri positivi, falsi positivi e falsi negativi, ponderato per la distribuzione delle classi.

4.1.3.2.4.1 Interpretazione delle Metriche

L'utilizzo combinato di queste metriche permette di avere un quadro chiaro delle prestazioni del modello:

- **Precisione alta** significa che il modello ha un buon controllo sui falsi positivi, cruciale per evitare falsi allarmi.
- **Recall alto** indica che il modello riesce a identificare la maggior parte dei campioni rilevanti, importante per la rilevazione di malware.
- **F1-score** è particolarmente utile in contesti dove il dataset è sbilanciato, poiché media i compromessi tra precision e recall.
- L'analisi tramite la **matrice di confusione** permette di capire in dettaglio dove il modello può essere migliorato, evidenziando pattern di errore specifici tra le classi.

Queste metriche, combinate, offrono una visione completa e dettagliata delle capacità predittive del modello, evidenziando sia i punti di forza che le aree di miglioramento.

Classe	Precision	Recall	F1-Score	Support
Adware	0.87	0.93	0.90	446
Backdoor	0.81	0.78	0.80	135
Downloader	0.78	0.56	0.65	123
Ransomware	0.85	0.95	0.89	256
Spyware	0.66	0.49	0.56	43
Trojan	0.76	0.82	0.79	404
Virus	0.81	0.65	0.72	205
Accuracy		**0.82**		1612
Macro Avg	0.79	0.74	0.76	1612
Weighted Avg	0.81	0.82	0.81	1612

Tabella 4.2: *Report di classificazione del modello CNN per la rilevazione di malware*

4.1.3.3 Matrice di Confusione

La matrice di confusione fornisce un'analisi visiva dettagliata delle prestazioni del modello. Mostra le predizioni corrette e gli errori di classificazione per ciascuna classe in una tabella. Ogni riga rappresenta i valori reali e ogni colonna le predizioni del modello, evidenziando i seguenti elementi:

- **Vero Positivo (TP)**: Campioni correttamente predetti come appartenenti a una classe.
- **Falso Positivo (FP)**: Campioni che sono stati erroneamente predetti come appartenenti a una classe.
- **Falso Negativo (FN)**: Campioni che appartenevano a una classe ma non sono stati riconosciuti come tali.
- **Vero Negativo (TN)**: Campioni correttamente identificati come non appartenenti a una classe.

Dalla tabella 4.3 emerge che le classi "Adware" e "Ransomware" sono quelle classificate con la maggiore accuratezza, con elevati valori di veri positivi (TP) e bassi valori di falsi negativi (FN). Tuttavia, per alcune classi come "Spyware" e "Downloader", si osserva un maggior numero di errori, evidenziando la difficoltà del modello nel distinguere alcune classi che possono avere caratteristiche simili.

Classe	Adware	Backdoor	Downloader	Ransomware	Spyware	Trojan	Virus
Adware	415	1	3	0	2	15	10
Backdoor	4	105	2	3	1	20	0
Downloader	7	10	69	6	1	27	3
Ransomware	0	0	3	242	1	10	0
Spyware	1	1	3	3	21	14	0
Trojan	13	8	5	25	4	331	18
Virus	38	4	4	7	2	17	133

Tabella 4.3: *Matrice di Confusione: confronto tra classi reali (righe) e classi predette (colonne)*

4.1.4 Generative Adversarial Network

Una Rete Generativa Avversaria è una categoria di modelli specializzati in compiti complessi e può essere applicata alla cybersecurity, alla modellazione, alla simulazione e al processamento del linguaggio naturale. Una caratteristica distintiva delle GAN è che offrono benefici significativi nell'apprendimento non supervisionato e nell'addestramento di reti neurali adottando un approccio avversario che migliora la capacità di generare campioni realistici.

Le reti GAN sono state introdotte per la prima volta da Goodfellow et al. come un framework che consiste in due modelli: un modello generativo G e un modello discriminatorio D. Lo scopo del primo è determinare una distribuzione di probabilità stimata $p(x)$, mentre il secondo è usato per valutare la probabilità che un campione sia reale dato un campione della distribuzione della popolazione.

Nel quadro generale delle GAN, i modelli utilizzati per il generatore e il discriminatore sono percettroni multistrato. Queste reti neurali sono addestrate come segue:

- La procedura di addestramento del modello generativo mira a massimizzare la probabilità di generare campioni indistinguibili dai veri dati di addestramento.
- Il modello discriminatorio è addestrato per massimizzare la sua capacità di distinguere i dati reali da quelli generati.

Il generatore di una rete GAN può essere rappresentato come in Figura 1, dove z è una variabile latente casuale, mentre ogni elemento x dipende da ciascun elemento di z. Tipicamente, ciascun elemento di x dipende da ciascun elemento di z.

Il modello di GAN sviluppato durante questo progetto è stato sviluppato con attraverso Keras[6] e TensorFlow[7], in quanto queste librerie offrono un'implementazione efficiente e flessibile delle GAN.

4.1.5 Funzione di perdita nelle GAN

Nelle GAN, la funzione di perdita considerata finora era basata solo su un punto dati dalla distribuzione dei dati. Per considerare l'intero set di dati, la funzione valore è formulata per includere il valore atteso per ogni quantità applicabile nella formula.

Tenendo conto di queste funzioni, la funzione di perdita $J(D)$ e $J(G)$ possono essere definite come segue:

$$J(D) = \mathbb{E}_{x\sim p_{\text{data}}(\log(D(x)))} + \mathbb{E}_{z\sim p(z)}(\log(1 - D(G(z))))$$

$$J(G) = \mathbb{E}_{z\sim p(z)}(\log(D(G(z))))$$

Infine, si può dire che GAN esegue un gioco minimax a due giocatori tra questi due modelli per ottenere l'efficacia generativa desiderata. Questo gioco può essere formalmente rappresentato dalla seguente funzione valore:

$$\min_G \max_D V(D, G) = \mathbb{E}_{x\sim p_{\text{data}}(\log(D(x)))} + \mathbb{E}_{z\sim p(z)}(\log(1 - D(G(z))))$$

Il training delle GAN, come descritto da Goodfellow et al., suggerisce che il discriminatore dovrebbe essere aggiornato più frequentemente rispetto al generatore, per esempio, aggiornando il discriminatore quattro volte ogni volta che il generatore viene aggiornato. Quindi, per ciascun aggiornamento del generatore, viene utilizzato un minicampione di rumore $p(z)$ per ciascuno degli k passaggi, quindi viene applicato il generatore al minicampione e l'algoritmo di discesa gradiente viene utilizzato per aggiornare il generatore.

4.1.5.1 Architettura

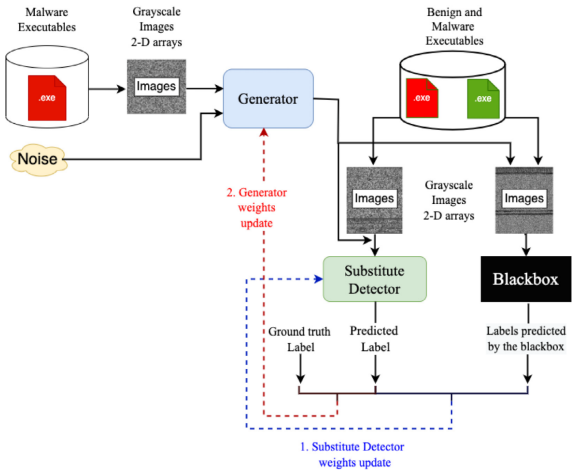

Figura 4.5: *Architettura della Generative Adversarial Network (GAN) utilizzata*

4.1.5.2 Generatore

4.1.5.2.1 Struttura

Il generatore è progettato per creare immagini realistiche aggiungendo rumore controllato a immagini reali. La sua struttura è composta da una sequenza di livelli densi, culminanti in una trasformazione che produce immagini con la stessa dimensione di quelle del dataset. L'architettura combina un input di rumore latente, etichette di classe e immagini reali, producendo immagini alterate attraverso l'operazione di somma.

4.1.5.2.2 Funzionamento

Il generatore opera applicando un rumore progressivo alle immagini reali, guidato da un vettore latente e dalle etichette di classe. Il rumore viene gradualmente adattato durante l'addestramento, ottimizzando il modello per produrre immagini che siano percepite come realistiche dal discriminatore (substitute detector). La perdita del generatore include una componente che misura l'inganno del modello *blackbox* e una regolarizzazione basata sulla somiglianza con le immagini originali.

4.1.5.2.3 Processo di addestramento

Il generatore viene addestrato utilizzando gradienti calcolati sulla base della capacità di ingannare il modello "blackbox". Durante ogni epoca, il rumore viene progressivamente regolato in base a una funzione sigmoidea, incrementando il suo impatto sulle immagini. La loss del generatore viene ottimizzata minimizzando la distanza tra le immagini reali e generate, e massimizzando l'inganno del blackbox. Inoltre, le immagini generate vengono validate per assicurare che rispettino la distribuzione del dataset reale.

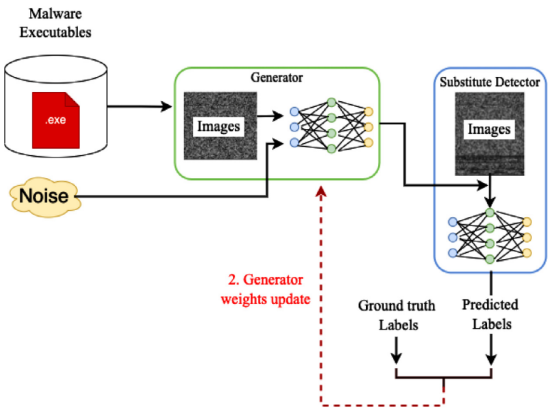

Figura 4.6: *Processo di addestramento del Generatore*

4.1.5.3 Discriminatore

4.1.5.3.1 Struttura

Il discriminatore, chiamato anche *substitute detector*, è un modello convoluzionale progettato per classificare le immagini. La sua struttura include strati convoluzionali per estrarre caratteristiche rilevanti, seguiti da uno strato di pooling e da livelli densi per la classificazione finale. L'output è una distribuzione di probabilità sulle classi del dataset.

4.1.5.3.2 Funzionamento

Il discriminatore riceve sia immagini reali che generate come input e tenta di classificarle in base alle classi del dataset. Il modello è ottimizzato per distinguere accuratamente tra immagini appartenenti a classi diverse, fornendo un feedback cruciale al generatore durante l'addestramento.

4.1.5.3.3 Processo di addestramento

Il discriminatore viene addestrato simultaneamente al generatore, utilizzando immagini reali e generate. La sua funzione di perdita è la cross-entropy categoriale, calcolata sulle etichette reali e le predizioni del modello. Durante il processo, i gradienti vengono propagati per affinare i pesi del discriminatore, migliorandone la capacità di classificazione. Le prestazioni vengono monitorate utilizzando metriche di accuratezza, precisione, richiamo e F1-score sia sulle immagini di training che di validazione.

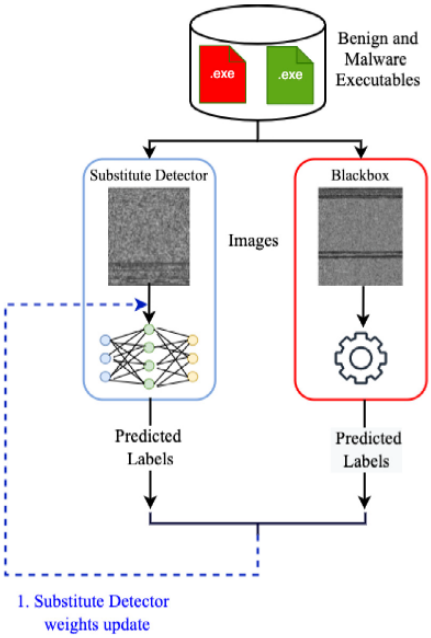

Figura 4.7: *Processo di addestramento del Substitute Detector*

4.1.5.4 Blackbox

Il modello *Blackbox* rappresenta un sistema di classificazione pre-addestrato, utilizzato come riferimento durante il processo di addestramento del generatore e del discriminatore. Questo modello è responsabile della classificazione delle immagini generate, fornendo un feedback critico per valutare l'efficacia del generatore nel creare immagini che ingannino il classificatore. Il blackbox funge quindi da obiettivo principale del generatore, il quale cerca di massimizzare la confusione del modello attraverso l'ottimizzazione delle sue uscite.

4.1.5.5 Applicazioni

Le Generative Adversarial Networks rappresentano una delle architetture di machine learning più innovative e potenti degli ultimi anni. Combinando due reti neurali, il generatore e il discriminatore, le GAN sono state applicate con successo in numerosi ambiti, tra cui la generazione di immagini, la sintesi del linguaggio e, più recentemente, nella rilevazione e creazione di malware. Sebbene le GAN siano principalmente note per la loro capacità di generare dati sintetici altamente realistici, il loro impiego in cybersecurity, e in particolare nella manipolazione e generazione di malware, sta acquisendo sempre più attenzione.

4.1.5.5.1 Generazione di Malware per l'Analisi e la Rilevazione

Una delle applicazioni emergenti delle GAN nel contesto del malware è la generazione di malware sintetico. Le GAN possono essere utilizzate per creare nuovi campioni di malware che replicano le caratteristiche delle minacce informatiche esistenti. Questo approccio consente di arricchire i dataset di malware, che sono spesso sbilanciati o insufficienti, con nuovi esempi che possono essere utilizzati per addestrare modelli di rilevazione più robusti e accurati. Generando malware sintetico, è possibile simulare scenari di attacco reali senza compromettere la sicurezza, permettendo agli esperti di cybersecurity di testare e affinare i sistemi di difesa. Le GAN si distinguono particolarmente nella creazione di varianti di malware che mantengono le stesse caratteristiche funzionali ma con differenze sufficienti da eludere i sistemi di rilevamento tradizionali. Questi modelli generativi sono in grado di apprendere le modalità di evasione dei rilevatori, come quelli basati su firma o comportamentali, e creare malware che possiede capacità di auto-modifica, mimetizzandosi meglio nei sistemi di sicurezza.

4.1.5.5.2 Miglioramento dei Sistemi di Rilevamento

Un'altra area cruciale in cui le GAN stanno trovando applicazione è nel miglioramento dei sistemi di rilevamento del malware. Tradizionalmente, i sistemi di rilevamento si basano su tecniche come l'analisi delle firme o l'analisi comportamentale. Tuttavia, queste tecniche spesso falliscono quando il malware si evolve rapidamente, specialmente nei casi in cui il codice cambia per sfuggire ai sistemi di sicurezza. Le GAN, in questo caso, possono essere utilizzate per addestrare modelli di rilevamento più robusti, generando nuovi campioni di malware che i sistemi di difesa non hanno mai visto prima. L'approccio di adversarial training (addestramento avversario) con le GAN è particolarmente promettente: durante l'addestramento, il generatore crea malware sintetico che viene utilizzato per ingannare il discriminatore, che cerca di classificarlo come malware o come software legittimo. Questo processo aiuta a migliorare la capacità

del modello di rilevare anche minacce sconosciute e mai viste, aumentando l'efficacia dei sistemi di sicurezza.

4.1.5.5.3 Evasione e Attacchi Avanzati

Le GAN sono anche utilizzate in ambito offensivo per creare attacchi adversariali mirati, cioè malware progettato per aggirare specifici sistemi di difesa. Utilizzando la capacità delle GAN di generare nuovi dati in modo iterativo e migliorato, gli attaccanti possono creare malware che riesce a bypassare i sistemi di rilevazione basati su machine learning. In questo caso, il generatore produce varianti del malware che cercano deliberatamente di eludere i filtri di rilevamento, mentre il discriminatore aiuta a ottimizzare il processo di evasione, rendendo l'attacco sempre più difficile da identificare.

4.1.5.6 Tipologie

Esistono numerosi tipi e varianti di GAN, progettati per migliorare le prestazioni in compiti specifici come la generazione di immagini, il miglioramento della qualità delle immagini, o l'adattamento ai dati di input. Di seguito vengono descritti alcuni dei modelli più rilevanti e simili al modello sviluppato durante il periodo di tirocinio: MiGAN, MalGAN, Xception, e InceptionNet.

4.1.5.6.1 MiGAN (Malware Image Generative Adversarial Network)

MiGAN è una variante delle GAN utilizzata per la generazione di immagini di malware. È particolarmente utile nel campo della cybersecurity per migliorare la rilevazione e la generazione di nuove varianti di malware, e si concentra sulla sintesi di immagini di malware che imitano le caratteristiche visive delle minacce esistenti ma con l'intento di eludere i sistemi di rilevamento.

4.1.5.6.1.1 Architettura di MiGAN

L'architettura di MiGAN è composta da un generatore e da un discriminatore, simile a una GAN tradizionale, ma progettata per gestire immagini di malware. La struttura del modello è la seguente:

- **Generatore (G)**: Il generatore crea immagini di malware sintetiche a partire da un vettore di rumore casuale, utilizzando convoluzioni trasposte e batch normalization. La rete è composta da diversi strati di convoluzione, ognuno dei quali espande la dimensione dell'immagine, partendo da un basso livello di dettaglio a uno più alto.
  - Strato di `Dense`: Per iniziare con un vettore di rumore.
  - Strato `Reshape`: Per adattare l'output del `Dense` in una forma adatta alla convoluzione.
  - Strati di `Conv2DTranspose`: Convoluzioni trasposte per ingrandire l'immagine.
  - `BatchNormalization`: Per migliorare la stabilità dell'apprendimento.
  - Funzione di attivazione `ReLU`: Per la maggior parte degli strati.
  - `Tanh`: Per l'output, che restituisce valori tra -1 e 1, tipici delle immagini.

- **Discriminatore (D)**: Il discriminatore è una rete neurale convoluzionale che cerca di distinguere tra immagini di malware reali e quelle sintetiche. Essa utilizza convoluzioni per ridurre progressivamente la dimensione spaziale dell'immagine, utilizzando anche `LeakyReLU` come funzione di attivazione.
  - Strati `Conv2D`: Convoluzioni per l'estrazione delle caratteristiche.
  - `LeakyReLU`: Per prevenire i vanishing gradients.
  - `Dropout`: Per evitare overfitting.
  - `Flatten`: Per appiattire l'immagine.
  - `Dense`: Per ottenere una singola probabilità tra 0 (reale) e 1 (falsa).

Questa architettura consente di generare nuove varianti di malware che possono essere utilizzate per allenare modelli di rilevamento più robusti.

4.1.5.6.2 MalGAN (Malware Generative Adversarial Network)

MalGAN è un'architettura GAN progettata per eludere i sistemi di rilevamento del malware esistenti. A differenza di MiGAN, che genera immagini di malware, MalGAN si concentra sulla creazione di varianti che possano bypassare i modelli di machine learning tradizionali.

4.1.5.6.2.1 Architettura di MalGAN

MalGAN si basa su un'architettura GAN simile a quella di MiGAN, ma con un focus sull'ottimizzazione delle capacità di evasione. In particolare, MalGAN integra un meccanismo di *adversarial training* per creare esempi di malware che sono difficili da rilevare dai modelli di sicurezza.

- **Generatore (G)**: Crea varianti di malware per ingannare il discriminatore. Il generatore è composto da:
  - Strato `Dense` seguito da `Reshape`.
  - Strati `Conv2DTranspose` per amplificare la dimensione dell'immagine.
  - Normalizzazione e `ReLU` per attivazioni.
- **Discriminatore (D)**: È un classificatore binario che distingue tra malware reale e generato. Utilizza:
  - Strati `Conv2D` con `LeakyReLU`.
  - `Dropout` per migliorare la generalizzazione.
  - Funzione di attivazione `Sigmoid` per la probabilità di autenticità.

Il modello di MalGAN è addestrato su varianti di malware esistenti, cercando di ottimizzare la sua capacità di generare malware che può sfuggire ai rilevatori.

4.1.5.6.3 Xception (Extreme Inception)

Xception è una rete neurale convoluzionale basata su una versione avanzata di Inception, ed è progettata per migliorare l'efficienza computazionale attraverso l'uso di convoluzioni separabili in profondità.

4.1.5.6.3.1 Architettura di Xception

Xception[9] si distingue dall'architettura Inception tradizionale, poiché utilizza convoluzioni separabili in profondità, che separano le convoluzioni spaziali e di canale. Questo approccio migliora l'efficienza computazionale e consente una maggiore espressione del modello.

- **Convoluzioni separabili in profondità**: Separano le operazioni di convoluzione in due passaggi: uno per ciascun canale e uno per la combinazione dei canali. Questo riduce il numero di parametri e aumenta l'efficienza.
- Strati `Conv2D`: Applicati ai dati separatamente per ciascun canale.
- Strati `MaxPooling2D`: Per ridurre le dimensioni spaziali.
- `GlobalAveragePooling2D`: Utilizzato per ridurre le dimensioni della feature map, migliorando la generalizzazione.
- Strati `Dense`: Per la classificazione finale con `Softmax`.

Questa architettura è ampiamente utilizzata in compiti di visione artificiale, inclusa la classificazione delle immagini.

4.1.5.6.4 InceptionNet

InceptionNet è una rete neurale convoluzionale progettata per essere altamente efficiente, utilizzando un approccio modulare che combina convoluzioni di diverse dimensioni e pooling.

4.1.5.6.4.1 Architettura di InceptionNet

InceptionNet utilizza moduli Inception che permettono di eseguire convoluzioni di diverse dimensioni, migliorando la capacità della rete di apprendere a più scale.

- **Inception Module**: Ogni modulo esegue convoluzioni di diverse dimensioni (1x1, 3x3, 5x5) e le concatena, permettendo alla rete di esplorare diverse scale di caratteristiche.
- Strati `Conv2D`: Eseguiti con kernel di dimensioni diverse all'interno dello stesso modulo.
- `Pooling Layers`: Combinazione di max-pooling e average-pooling.
- `1x1 Convoluzioni`: Utilizzate per ridurre la dimensionalità, migliorando l'efficienza computazionale.
- Strati `Dense`: Per la classificazione finale.
- `Softmax`: Per la normalizzazione delle probabilità.

Questa architettura è stata utilizzata con successo in competizioni di riconoscimento delle immagini e classificazione di oggetti, come il ImageNet Challenge.

4.2 Problematiche

Durante lo svolgimento del progetto, sono state riscontrate diverse problematiche che hanno influenzato negativamente le prestazioni del modello e la sua efficienza nel rilevamento del malware. Le principali includono:

1. **Squilibrio del dataset:** Il dataset era notevolmente sbilanciato, risultando in una riduzione della precisione e dell'accuratezza del modello. Questo squilibrio ha limitato la capacità del modello di rappresentare efficacemente tutte le classi di malware, compromettendo la sua abilità di generalizzare e rilevare nuove varianti di malware.

2. **Dimensioni del dataset:** Le dimensioni ridotte del dataset hanno impedito una rappresentazione accurata e completa delle diverse classi di malware. Inoltre, la classificazione è stata effettuata per superfamiglie anziché per singole famiglie, il che ha ulteriormente ridotto la precisione e l'accuratezza del modello, poiché le superfamiglie aggregano molteplici classi con caratteristiche simili.

3. **Architettura del modello:** L'architettura attuale del modello potrebbe beneficiare di miglioramenti. L'adozione di architetture più avanzate, come Xception o InceptionNet, potrebbe significativamente migliorare le prestazioni.

4. **Limitazioni delle risorse hardware:** Le limitate capacità computazionali della macchina utilizzata per l'addestramento e i test hanno comportato tempi di addestramento prolungati e hanno ristretto la nostra capacità di esplorare a fondo l'architettura del modello e di ottimizzare i parametri.

Queste limitazioni hanno evidenziato la necessità di un ulteriore sviluppo e ottimizzazione per superare queste sfide e migliorare l'efficacia del modello nel rilevamento del malware.

4.2.1 Miglioramenti

Di seguito sono riportati i principali miglioramenti proposti per affrontare le problematiche identificate e ottimizzare il modello di rilevamento del malware:

1. **Miglioramento del Dataset**:

   - **Bilanciamento**: Ridurre lo sbilanciamento nel dataset, rendendolo non solo più equilibrato ma anche più rappresentativo delle varie classi di malware.
   - **Accuratezza e Dimensioni**: Espansione del dataset, rendendolo più grande e accurato, migliorando significativamente la qualità e l'affidabilità dei dati disponibili per l'addestramento del modello.

2. **Classificazione Raffinata**:

   - **Per Famiglia e Superfamiglia**: Introduzione di una classificazione più dettagliata, sia per singole famiglie sia per superfamiglie di malware, aumenterebbe la precisione del modello nella rilevazione delle minacce, permettendo un' identificazione più specifica e accurata.

3. **Perfezionamento dell'Architettura del Modello**:

- **Miglioramento dei Layer**: Dedicare più tempo all'ottimizzazione della struttura dei layer permetterebbe una migliore elaborazione delle caratteristiche e un'apprendimento più efficace, portando di conseguenza ad un aumento delle prestazioni complessive del sistema.

4. **Utilizzo di Architetture Avanzate**:

   - **Xception e InceptionNet**: L'adozione di architetture più avanzate e performanti, come Xception e InceptionNet, potrebbe migliorare significativamente le prestazioni del modello, consentendo una migliore rappresentazione delle caratteristiche e una maggiore capacità di generalizzazione.

Questi avanzamenti renderebbero il modello non solo più robusto ma anche più efficiente nel rilevare e classificare vari tipi di malware con una maggiore precisione.

Capitolo 5

Processi e Metodi

Questo capitolo fornisce in dettaglio l'ambiente di ricerca utilizzato, le tecnologie impiegate e descrive gli esperimenti condotti. Provvede a dare inoltre tutte le informazioni necessarie per replicare gli esperimenti.

5.1 Ambiente

In questa sezione vengono descritti gli strumenti utilizzati durante il progetto, con le relative versioni riassunte nella Tabella 5.1.
L'intero progetto è stato sviluppato su ***MacOS***[1]. La scelta di questo sistema operativo è stata motivata dalla familiarità con l'ecosistema Apple e dalle elevate prestazioni del processore, oltre che dalla vasta disponibilità di strumenti per l'analisi e lo sviluppo.
Per la decompilazione degli eseguibili è stato utilizzato ***Ghidra***[2], uno strumento open source per l'ingegneria inversa, sviluppato dalla NSA's Research Directorate. La scelta è ricaduta su questo software poiché è open source e già noto al tirocinante.
Il codice è stato condiviso e mantenuto tramite **GitHub**[3].
Per la realizzazione degli esperimenti, il linguaggio di programmazione principale impiegato è stato **Python**[4]. Python è stato scelto per la sua versatilità e l'ampia disponibilità di librerie, che hanno facilitato lo sviluppo rapido di prototipi e script.
Per la classificazione delle sottofamiglie di malware è stato utilizzato **AVClass2**[10], un tool open source sviluppato da MaliciaLab, che consente di classificare i campioni di malware in base a famiglia e sottofamiglia partendo da report in formato JSON.
I campioni di malware sono stati scaricati da tre fonti affidabili e riconosciute: **Malshare**[5], **Malware Bazaar**[6] e **VirusShare**[7].
Per la generazione dei report relativi a ciascun malware è stato utilizzato **VirusTotal**[8], un servizio online di scansione antivirus che analizza file e URL per rilevare malware e fornire dettagli sulle minacce.
Infine, per determinare la presenza di un packer in un malware, è stato impiegato

[1] https://support.apple.com/en-us/111893
[2] https://ghidra-sre.org/
[3] https://github.com/
[4] https://python.org/
[5] https://malshare.com/pull.php
[6] https://bazaar.abuse.ch/
[7] https://virusshare.com/
[8] https://www.virustotal.com/

Detect it Easy (DiE)[11], uno strumento popolare tra analisti di malware, esperti di cybersecurity e reverse engineer. DiE supporta l'analisi sia basata su firme che euristica, ed è compatibile con una vasta gamma di piattaforme, come Windows, Linux e MacOS. Grazie alla sua architettura adattabile e basata su script, DiE si distingue come uno degli strumenti più versatili nel settore.

Tipo	Nome	Versione
Applicativo	*Chromium*	131.0.6778.70
Applicativo	*Ghidra*	11.1.2
Applicativo	*Git*	2.46.0
Applicativo	*Github*	
Applicativo	*AVClass2*	2.0.0
Applicativo	*DiE*	3.10.0
Applicativo	*Visual Studio Code*	1.95.3
Sistema Operativo	*MacOS*	Sonoma 14.6.0
Linguaggio	*Python*	3.13.0

Tabella 5.1: *Tabella riassuntiva tecnologie usate*

5.1.1 Svolgimento del Progetto

Il progetto è stato sviluppato attraverso una serie di fasi distinte, ognuna delle quali ha contribuito in modo significativo alla creazione di un sistema per l'analisi e la generazione di malware. Le fasi del progetto sono descritte di seguito:

1. **Preparazione dell'ambiente di sviluppo**: In questa fase iniziale è stato configurato l'ambiente di sviluppo, includendo l'installazione di un ambiente virtuale, delle librerie necessarie e il setup della repository GitHub. Quest'ultima è stata organizzata in modo strutturato, con cartelle dedicate sia allo sviluppo pratico del progetto sia alla documentazione teorica (la tesi). Inoltre, è stata creata una struttura di base per il progetto, con directory principali e file di configurazione essenziali, per garantire un workflow ordinato ed efficiente.

2. **Ricerca, raccolta e classificazione dei dati**: Sono stati individuati dataset contenenti campioni di malware da fonti affidabili e riconosciute, come **VirusShare**, **MalwareBazaar** e **Malshare**. Ogni campione raccolto è stato analizzato tramite **VirusTotal**, che ha fornito report dettagliati. Successivamente, è stato utilizzato lo strumento **AVClass2** [10] per classificare i malware nelle rispettive famiglie, basandosi sulle loro caratteristiche principali. Questo processo ha garantito un dataset ben strutturato e pronto per le fasi successive.

3. **Preprocessing**: I campioni di malware sono stati disassemblati per estrarre il loro codice in formato esadecimale e assembly. In particolare, dall'assembly sono state selezionate esclusivamente le istruzioni mnemoniche, ignorando altri tipi di dati. Successivamente, sono state generate coppie di mnemonici consecutivi per catturare le relazioni tra le operazioni eseguite dal malware.

   Per analizzare queste coppie, è stata applicata la tecnica **TF-IDF** (Term Frequency - Inverse Document Frequency)[12], che ha permesso di identificare le

coppie più distintive e rilevanti riducendo l'influenza di quelle troppo comuni. Successivamente, è stata utilizzata la **PCA** (Principal Component Analysis) [13] per ridurre la dimensionalità dei dati, identificando le caratteristiche chiave. Dai componenti principali è stata creata una matrice 16x16, che rappresenta una "impronta" comportamentale del malware. Questa matrice è stata normalizzata utilizzando la tecnica **Min-Max Scaling**, mappando i valori su un intervallo [0, 255] per consentire la conversione in immagini in scala di grigi. La formula utilizzata per la normalizzazione è:

$$M'_{ij} = \frac{M_{ij} - \min(M)}{\max(M) - \min(M)} \times 255$$

Dove:

- M_{ij} è il valore originale nella posizione (i, j) della matrice.
- $\min(M)$ e $\max(M)$ sono rispettivamente il valore minimo e massimo della matrice.
- M'_{ij} è il valore normalizzato, corrispondente all'intensità di un pixel.

Infine, le immagini generate sono state utilizzate come input per il modello di rete neurale convoluzionale.

4. **Addestramento della CNN**: È stata progettata e addestrata una rete neurale convoluzionale (**CNN**) per la classificazione dei malware. La CNN ha ricevuto in input le immagini generate durante il preprocessing. Il dataset è stato suddiviso in tre parti: 80% per l'addestramento, 10% per il test e 10% per la validazione. La rete includeva:

   - Due strati convoluzionali con 64 filtri ciascuno, seguiti da strati di max pooling.
   - Un terzo strato convoluzionale con max pooling.
   - Strati fully connected, con uno strato *Flatten*, due strati *Dense* e uno strato *Dropout* per ridurre il rischio di overfitting.

   L'addestramento è stato eseguito con tecniche di regolarizzazione e ottimizzazione degli iperparametri per migliorare le prestazioni. La CNN è stata valutata in termini di accuratezza e precisione sulla classificazione delle famiglie di malware. La struttura del modello è riportata in Figura 4.2.

5. **Creazione e utilizzo della GAN**: Una **Generative Adversarial Network (GAN)** è stata sviluppata per generare nuovi campioni di malware. La GAN è composta da due componenti:

   - **Generatore**, il quale crea immagini di malware sintetiche partendo da un vettore di rumore casuale.
   - **Discriminatore**, che valuta la qualità delle immagini generate confrontandole con campioni reali.

   Le immagini generate sono state utilizzate per mettere alla prova il modello di classificazione pre-addestrato. Attraverso un processo iterativo, è stato introdotto rumore crescente per verificare se il modello potesse essere ingannato. Ogni

immagine è stata sottoposta a un massimo di dieci iterazioni, registrando metriche come accuratezza e numero di iterazioni necessarie per alterare la predizione del modello.

Grafici dettagliati sono stati generati per ogni categoria di malware, includendo:

- **Grafici a barre**: Mostrano l'accuratezza media con intervalli di confidenza.
- **Grafici smussati**: Evidenziano l'andamento generale delle prestazioni del modello.
- **Heatmap**: Rappresentano visivamente le performance per ciascun campione.

6. **Esperimenti e analisi dei risultati**: Sono stati condotti esperimenti per valutare sia la CNN che la GAN. La CNN è stata valutata in termini di accuratezza e precisione nella classificazione dei malware, mentre la GAN è stata analizzata in base alla qualità e somiglianza dei campioni generati rispetto a quelli reali. I risultati sono stati analizzati per identificare punti di forza e aree di miglioramento.

7. **Conclusioni e sviluppi futuri**: Il progetto ha dimostrato come le tecniche di deep learning, incluse CNN e GAN, possano essere utilizzate efficacemente per l'analisi e la generazione di malware. Sono state inoltre discusse potenziali direzioni future, come l'integrazione con analisi dinamiche o l'uso di tecniche multimodali per migliorare ulteriormente le prestazioni.

5.1.2 Tecnologie Specifiche per GAN

Per la realizzazione della componente blackbox all'interno della GAN, è stata utilizzata la libreria Keras di TensorFlow, uno strumento potente e flessibile per la costruzione di modelli di deep learning, disponibile per Python. Keras facilita l'implementazione di reti neurali complesse attraverso un'interfaccia di alto livello e modulare. Il modello di rete neurale convoluzionale è stato sviluppato completamente da zero, addestrando un dataset appositamente creato e classificato dal tirocinante. Per la costruzione della GAN, invece, si è fatto riferimento all' architettura di un modello avanzato e già validato, MalGAN.[14]

5.2 Esperimenti

La sezione corrente esamina in dettaglio gli esperimenti condotti nel corso dello studio, illustrando la logica sottostante e le procedure impiegate per la loro realizzazione. L'obiettivo è fornire una panoramica completa delle attività sperimentali, cosicchè ci sia una maggiore comprensione sia dei metodi utilizzati che degli scopi.

Gli esperimenti sono stati creati con l'obiettivo di identificare e testare diversi metodi per aumentare la precisione nel riconoscimento dei malware.

I relativi risultati vengono analizzati nel Capitolo 6 mentre le conclusioni raggiunte sono discusse nel Capitolo 7.

Gli esperimenti sono stati condotti sotto le stesse condizioni per assicurare la veridicibilità dei risultati, ovvero **dataset suddiviso nello stesso modo**, **random state** e **numero di epoche uguali**.

5.2.1 Addestramento della CNN

5.2.1.1 Esperimento 1

```
model = Sequential([
    Conv2D(128, (3, 3), input_shape=(SIZE, SIZE, 1),
            padding='same', activation='relu'),
    MaxPooling2D(2, 2),
    Conv2D(64, (3, 3), #input_shape=(SIZE//2, SIZE//2, 1),
            padding='same', activation='relu'),
    MaxPooling2D(2, 2),

    Conv2D(32, (3, 3), #input_shape=(SIZE//4, SIZE//4, 1),
            padding='same',  activation='relu'),
    MaxPooling2D(2, 2),
    Flatten(),
    Dense(64, activation='relu'),
    Dropout(0.5),
    Dense(len(categories), activation='softmax')
])
```

Listing 5.1: Primo modello di CNN creato

5.2.1.1.1 Descrizione

La struttura del modello è organizzata come segue:

- **Strati convoluzionali:** La rete include tre strati convoluzionali. Il primo strato applica 128 filtri di dimensione 3×3 sull'immagine di input, preservandone la dimensione originale grazie all'opzione di *padding* `same`. Gli strati successivi applicano rispettivamente 64 e 32 filtri, sempre di dimensione 3×3, continuando a preservare le dimensioni dei dati. Tutti gli strati convoluzionali utilizzano la funzione di attivazione `ReLU`, che introduce non linearità e migliora la capacità del modello di apprendere rappresentazioni complesse.
- **Strati di pooling:** Dopo ogni strato convoluzionale, è presente uno strato di *MaxPooling* con una finestra 2×2. Questo riduce le dimensioni spaziali dell'input, abbassando la complessità computazionale e il rischio di overfitting, preservando al contempo le caratteristiche più significative.
- **Strato *Flatten*:** Dopo gli strati convoluzionali e di pooling, il *Flatten* converte i dati bidimensionali in un vettore unidimensionale, rendendoli compatibili con gli strati *fully connected*.
- **Strati fully connected:** La rete include due strati completamente connessi (*Dense*). Il primo strato ha 64 unità ed è seguito da uno strato di *Dropout* con un tasso di 0.5 per ridurre il rischio di overfitting. Il secondo strato *Dense*, con un numero di unità pari al numero di categorie di classificazione, utilizza la funzione di attivazione `softmax` per produrre le probabilità di appartenenza di ciascuna immagine a una specifica categoria.

L'architettura complessiva è ottimizzata per gestire immagini in scala di grigi con dimensioni $SIZE \times SIZE$ (nel nostro caso, 16×16) come input. La sequenza di strati convoluzionali consente di estrarre caratteristiche locali a diversi livelli di astrazione, mentre gli strati *fully connected* sintetizzano queste caratteristiche per effettuare una classificazione accurata. L'uso di tecniche di regolarizzazione, come il *Dropout*, migliora la capacità del modello di generalizzare ai dati di test, riducendo l'overfitting.

5.2.1.2 Esperimento 2

```
model = Sequential([
    Conv2D(128, (3, 3), input_shape=(SIZE, SIZE, 1),
           padding='same', activation='relu'),
    MaxPooling2D(2, 2),
    Conv2D(64, (3, 3), input_shape=(SIZE//2, SIZE//2, 1),
           padding='same', activation='relu'),
    MaxPooling2D(2, 2),

    Conv2D(32, (3, 3), input_shape=(SIZE//4, SIZE//4, 1),
           padding='same',  activation='relu'),
    MaxPooling2D(2, 2),
    Flatten(),
    Dense(64, activation='relu'),
    Dropout(0.9),
    Dense(len(categories), activation='softmax')
])
```

Listing 5.2: Secondo modello di CNN creato

5.2.1.2.1 Descrizione

La configurazione del modello è descritta come segue:

- **Strati convoluzionali:** Il modello utilizza tre strati convoluzionali per estrarre caratteristiche locali a diversi livelli di astrazione.
  - Il primo strato convoluzionale applica 128 filtri di dimensione 3×3 all'immagine di input con dimensioni $(SIZE, SIZE, 1)$. Grazie all'opzione di *padding* `same`, la dimensione spaziale dell'immagine viene preservata.
  - Il secondo strato convoluzionale applica 64 filtri di dimensione 3×3, anch'essi con *padding* `same`, su un'immagine di dimensioni ridotte $(SIZE/2, SIZE/2, 1)$.
  - Il terzo strato convoluzionale utilizza 32 filtri di dimensione 3×3, sempre con *padding* `same`, su un input ulteriormente ridotto $(SIZE/4, SIZE/4, 1)$.

  Tutti gli strati convoluzionali utilizzano la funzione di attivazione `ReLU` per introdurre non linearità e facilitare l'apprendimento di rappresentazioni complesse.
- **Strati di pooling:** Dopo ciascuno strato convoluzionale, il modello include uno strato di *MaxPooling* con una finestra 2×2. Questa operazione riduce progressivamente la dimensione spaziale dell'input, concentrando l'attenzione sulle caratteristiche più significative e abbassando la complessità computazionale.

- **Strato *Flatten*:** Dopo gli strati convoluzionali e di pooling, i dati vengono appiattiti in un vettore unidimensionale tramite uno strato *Flatten*, rendendoli adatti per l'elaborazione negli strati completamente connessi.
- **Strati fully connected:** Il modello include due strati completamente connessi (*Dense*):
  - Il primo strato *Dense* contiene 64 unità con attivazione `ReLU`.
  - Un *Dropout* con tasso pari a 0.9 è applicato dopo il primo strato fully connected, per ridurre il rischio di overfitting, rendendo il modello più robusto.
  - Il secondo strato *Dense* utilizza un numero di unità pari al numero di categorie (`len(categories)`) e impiega la funzione di attivazione `softmax` per calcolare la probabilità di appartenenza a ciascuna classe.

L'architettura è progettata per gestire immagini in scala di grigi di dimensione $SIZE \times SIZE$, estrarre caratteristiche locali tramite strati convoluzionali e di pooling, e sintetizzare tali informazioni nei livelli completamente connessi per ottenere una classificazione accurata. L'elevato tasso di *Dropout* garantisce una maggiore generalizzazione del modello, riducendo la sensibilità al rumore nei dati di training.

5.2.1.3 Esperimento 3

```
class MalwareModelHyperModel(HyperModel):
    def __init__(self, num_classes):
        self.num_classes = num_classes

    def build(self, hp):
        model = Sequential([
            Conv2D(hp.Int('conv_1_filters', 32, 128, step=32),
                (3, 3), input_shape=(SIZE, SIZE, 1), padding='same
                ', activation='relu'),
            MaxPooling2D(2, 2),
            Conv2D(hp.Int('conv_2_filters', 32, 64, step=32), (3,
                 3), padding='same', activation='relu'),
            MaxPooling2D(2, 2),
            Conv2D(hp.Int('conv_3_filters', 32, 64, step=32), (3,
                 3), padding='same', activation='relu'),
            MaxPooling2D(2, 2),
            Flatten(),
            Dense(hp.Int('dense_units', 32, 128, step=32),
                activation='relu'),
            Dropout(hp.Float('dropout', 0.0, 0.5, step=0.1)),
            Dense(self.num_classes, activation='softmax')
        ])
        model.compile(optimizer='adam', loss='
            categorical_crossentropy', metrics=['accuracy'])
        return model

hypermodel = MalwareModelHyperModel(num_classes)

tuner = RandomSearch(
    hypermodel,
```

```
    objective='val_accuracy',
    max_trials=10,
    executions_per_trial=5,
    directory='my_dir',
    project_name='malware_classification_tuning'
)

tuner.search(X_train, y_train, epochs=30, validation_data=(X_val,
    y_val))
best_model = tuner.get_best_models(num_models=1)[0]
best_model.save(f'{value}/models/
    malware_classification_model_best.h5')

model = best_model
```

Listing 5.3: Terzo modello di CNN creato

5.2.1.3.1 Descrizione

La struttura del modello e il processo di tuning sono descritti come segue:

- **Struttura del modello:** La rete CNN è composta da una sequenza di strati convoluzionali, di pooling e completamente connessi:
  - **Strati convoluzionali:** La rete include tre strati convoluzionali, ciascuno con parametri configurabili:
    - Il numero di filtri per ciascun livello (`conv_1_filters`, `conv_2_filters`, `conv_3_filters`) viene selezionato dinamicamente tra 32 e 128 per il primo strato e tra 32 e 64 per gli altri, con incrementi di 32.
    - Tutti gli strati convoluzionali utilizzano una finestra 3×3, *padding* `same` e la funzione di attivazione `ReLU`.
  - **Strati di pooling:** Dopo ciascun livello convoluzionale, uno strato di *MaxPooling* con una finestra 2×2 riduce progressivamente le dimensioni spaziali dell'input, mantenendo le caratteristiche più significative.
  - **Strato *Flatten*:** I dati bidimensionali risultanti dagli strati convoluzionali e di pooling vengono convertiti in un vettore unidimensionale tramite uno strato *Flatten*.
  - **Strati completamente connessi:**
    - Un primo strato completamente connesso (*Dense*) ha un numero di unità configurabile (`dense_units`) tra 32 e 128, selezionato in incrementi di 32, con funzione di attivazione `ReLU`.
    - Un livello di *Dropout*, con un tasso configurabile (`dropout`) tra 0.0 e 0.5 in incrementi di 0.1, aiuta a ridurre il rischio di overfitting.
    - Lo strato finale *Dense* ha un numero di unità pari al numero di classi (`self.num_classes`) e utilizza la funzione di attivazione `softmax` per produrre la probabilità di appartenenza a ciascuna classe.
- **Compilazione del modello:** Il modello viene compilato utilizzando l'ottimizzatore `adam`, la funzione di perdita `categorical_crossentropy` e la metrica `accuracy`.

- **Tuning degli iperparametri:** Per ottimizzare le prestazioni del modello, è stato utilizzato il *Random Search* tramite la libreria `Keras Tuner`. Il processo di ricerca ha incluso i seguenti dettagli:
  - **Oggetto `tuner`:** Il tuner esplora una combinazione di iperparametri basandosi su 10 prove (`max_trials`), con 5 esecuzioni per ciascuna prova (`executions_per_trial`).
  - **Obiettivo:** Il tuning è stato ottimizzato rispetto alla metrica `val_accuracy`.
  - **Set di dati:** La ricerca degli iperparametri è stata condotta utilizzando il set di addestramento (`X_train`, `y_train`) e di validazione (`X_val`, `y_val`) per 30 epoche.
  - **Miglior modello:** Una volta completata la ricerca, il miglior modello è stato selezionato (`get_best_models`) e salvato per l'utilizzo futuro.
- **Salvataggio del modello:** Il modello con i parametri ottimali è stato salvato nel percorso `malware_classification_model_best.h5` per consentire un utilizzo successivo.

5.2.2 Addestramento della GAN

5.2.2.1 Esperimento 1

Il primo esperimento è stato condotto con un batch size di 32 e 100 epoche.

```
def build_generator(latent_dim, num_classes, image_shape):
    noise = Input(shape=(latent_dim,))
    labels = Input(shape=(num_classes,))
    real_image = Input(shape=image_shape)
    x = Dense(256, activation="relu")(noise)
    x = Dense(np.prod(image_shape), activation="tanh")(x)
    x = Reshape(image_shape)(x)
    noisy_image = Add()([real_image, x])
    model = Model(inputs=[noise, labels, real_image], outputs=
        noisy_image, name="generator")
    return model

def build_substitute_detector(input_shape, num_classes):
    input_img = Input(shape=input_shape)
    x = Conv2D(32, (3, 3), activation="relu", padding="same")(
        input_img)  # Layer convoluzionale
    x = MaxPooling2D((2, 2))(x)  # Pooling
    x = Flatten()(x)
    x = Dense(128, activation="relu")(x)
    x = Dense(num_classes, activation="softmax")(x)
    model = Model(inputs=input_img, outputs=x, name="
        substitute_detector")
    return model
```

Listing 5.4: Modello di GAN generato

5.2.2.1.1 Descrizione

Di seguito sono riportate le caratteristiche strutturali di ciascun componente:

- **Generatore**: L'architettura del generatore è progettata per combinare un vettore di rumore latente, etichette condizionate e un'immagine reale, producendo un'immagine "rumorosa". La struttura può essere descritta come segue:
  - **Input:**
    - Un vettore di rumore latente $\mathbf{z} \in \mathbb{R}^{latent\_dim}$.
    - Un vettore di etichette condizionate $\mathbf{y} \in \mathbb{R}^{num\_classes}$.
    - Un'immagine reale $\mathbf{I} \in \mathbb{R}^{image\_shape}$.
  - **Livelli di elaborazione del rumore latente:**
    - Un livello `Dense` con 256 unità e attivazione ReLU:
      $$\mathbf{h_1} = \text{ReLU}(\mathbf{W_1 z} + \mathbf{b_1}),$$
      dove $\mathbf{h_1} \in \mathbb{R}^{256}$.
    - Un secondo livello `Dense` per trasformare $\mathbf{h_1}$ in un vettore compatibile con la forma dell'immagine:
      $$\mathbf{h_2} = \tanh(\mathbf{W_2 h_1} + \mathbf{b_2}),$$
      dove $\mathbf{h_2} \in \mathbb{R}^{prod(image\_shape)}$.
    - Rimodellamento (`Reshape`) di $\mathbf{h_2}$ per ottenere un'immagine sintetica $\mathbf{G} \in \mathbb{R}^{image\_shape}$.
  - **Combinazione con l'immagine reale:**
    - Somma elemento per elemento (`Add`) tra l'immagine reale $\mathbf{I}$ e l'immagine generata $\mathbf{G}$:
      $$\mathbf{I_{noisy}} = \mathbf{I} + \mathbf{G},$$
      dove $\mathbf{I_{noisy}}$ rappresenta l'immagine "rumorosa" finale.
  - **Output:** L'immagine modificata $\mathbf{I_{noisy}} \in \mathbb{R}^{image\_shape}$.
- **Substitute Detector:** L'architettura del modello `substitute_detector` è progettata per classificare immagini in una serie di categorie predefinite. La struttura del modello è descritta come segue:
  - **Input:** Un'immagine $\mathbf{I} \in \mathbb{R}^{input\_shape}$, dove *input_shape* rappresenta le dimensioni spaziali e il numero di canali dell'immagine.
  - **Livelli convoluzionali e pooling:**
    - Un livello convoluzionale (`Conv2D`) con 32 filtri di dimensione 3×3, attivazione ReLU e `padding="same"`:
      $$\mathbf{C_1} = \text{ReLU}(\text{Conv2D}(\mathbf{I}, \mathbf{K})),$$
      dove $\mathbf{K}$ rappresenta i filtri convoluzionali e $\mathbf{C_1} \in \mathbb{R}^{input\_shape}$.
    - Un livello di pooling massimo (`MaxPooling2D`) con finestra 2×2, che riduce la dimensione spaziale dell'output:
      $$\mathbf{P_1} = \text{MaxPooling2D}(\mathbf{C_1}),$$
      dove $\mathbf{P_1}$ ha dimensioni spaziali dimezzate rispetto a $\mathbf{C_1}$.
  - **Livelli completamente connessi:**

* Un livello di appiattimento (`Flatten`) che trasforma l'output $\mathbf{P_1}$ in un vettore:

$$\mathbf{f} = \text{Flatten}(\mathbf{P_1}),$$

dove $\mathbf{f} \in \mathbb{R}^d$, con d pari al prodotto delle dimensioni spaziali di $\mathbf{P_1}$.

* Un livello denso (`Dense`) con 128 unità e attivazione ReLU:

$$\mathbf{h_1} = \text{ReLU}(\mathbf{W_1 f} + \mathbf{b_1}),$$

dove $\mathbf{h_1} \in \mathbb{R}^{128}$.

* Un secondo livello denso (`Dense`) con *num_classes* unità e attivazione softmax, che produce le probabilità di appartenenza alle classi:

$$\mathbf{o} = \text{Softmax}(\mathbf{W_2 h_1} + \mathbf{b_2}),$$

dove $\mathbf{o} \in \mathbb{R}^{num\_classes}$ rappresenta il vettore di probabilità per ogni classe.

– **Output:** Un vettore $\mathbf{o}$ che rappresenta la distribuzione di probabilità sulle *num_classes* categorie.

Il modello è costruito utilizzando l'API funzionale di Keras e restituito come oggetto `Model` con il nome `substitute_detector`. Esso accetta come input un'immagine $\mathbf{I}$ e produce un output $\mathbf{o}$ per la classificazione.

5.2.2.2 Esperimento 2

Il secondo esperimento è stato progettato per indagare l'effetto di un aumento delle epoche di addestramento sulle prestazioni del modello. È stato utilizzato un batch size pari a 64, mentre il numero di epoche è stato incrementato a 1000. L'architettura del modello è rimasta invariata rispetto al primo esperimento, consentendo un confronto diretto tra le prestazioni ottenute con diverse configurazioni di training.

5.2.2.3 Esperimento 3

Il terzo esperimento ha esplorato l'impatto di una diversa configurazione del batch size sull'addestramento, utilizzando un valore ridotto pari a 32, mantenendo costante il numero di epoche a 1000. Anche in questo caso, l'architettura del modello è stata mantenuta invariata rispetto agli esperimenti precedenti, al fine di isolare e valutare l'influenza del batch size sulle prestazioni complessive del modello.

Capitolo 6

Risultati

In questo capitolo si presenta un'analisi dettagliata dei risultati ottenuti dagli esperimenti descritti in precedenza.

6.1 Risultati degli Esperimenti CNN

6.1.1 Risultati Esperimento 1

6.1.1.1 Metriche di valutazione

Classe	Precision	Recall	F1-Score	Support
Adware	0.84	0.85	0.85	446
Backdoor	0.90	0.45	0.60	135
Downloader	0.52	0.30	0.38	123
Ransomware	0.84	0.90	0.87	256
Spyware	0.00	0.00	0.00	43
Trojan	0.60	0.83	0.70	404
Virus	0.60	0.56	0.58	205
Accuracy		**0.71**		1612
Macro Avg	0.61	0.56	0.57	1612
Weighted Avg	0.71	0.72	0.70	1612

Tabella 6.1: *Report di classificazione del modello CNN per la rilevazione di malware*

La tabella 6.1 riassume le prestazioni del modello CNN per la classificazione delle diverse famiglie di malware, utilizzando metriche standard come precision, recall e F1-Score. I punti principali da evidenziare sono i seguenti:

- **Adware:** Questa classe mostra ottime metriche, con valori di precision, recall e F1-Score pari a 0.85. Ciò indica che il modello è altamente efficace nel riconoscere i campioni di Adware.

- **Backdoor:** Sebbene la precision sia elevata (0.90), il recall è molto più basso (0.45), suggerendo che il modello non riesce a identificare una percentuale significativa di campioni di questa classe.
- **Downloader:** Le performance per questa classe sono scarse, con un F1-Score di 0.38, evidenziando problemi significativi nella corretta classificazione di questi campioni.
- **Spyware:** Questa classe presenta le peggiori performance, con un F1-Score di 0.00, indicando che il modello non è in grado di riconoscere alcun campione di Spyware.
- **Ransomware:** Il modello mostra buone prestazioni su questa classe, con un F1-Score di 0.87, grazie a valori bilanciati di precision (0.84) e recall (0.90).
- **Trojan:** La precision è moderata (0.60), ma il recall elevato (0.83) porta a un F1-Score di 0.70. Questo suggerisce che il modello riconosce correttamente molti campioni di Trojan, ma a costo di alcune classificazioni errate.
- **Virus:** Con un F1-Score di 0.58, questa classe presenta una moderata confusione rispetto ad altre categorie.

Globalmente, l'accuracy del modello è pari a 72%, ma le metriche di *macro average* (0.57 per F1-Score) evidenziano come il modello non gestisca uniformemente tutte le classi, con significative difficoltà per le classi meno rappresentate come Spyware e Downloader.

6.1.1.2 Matrice di Confusione

Classe	Adware	Backdoor	Downloader	Ransomware	Spyware	Trojan	Virus
Adware	380	0	2	0	0	31	33
Backdoor	3	61	12	15	0	39	5
Downloader	8	3	37	1	0	65	9
Ransomware	0	0	1	230	0	16	9
Spyware	0	1	6	3	0	30	3
Trojan	17	2	9	26	0	334	16
Virus	44	1	4	0	0	42	114

Tabella 6.2: *Matrice di Confusione: confronto tra classi reali (righe) e classi predette (colonne)*

La matrice di confusione presentata in Tabella 6.2 offre una visione dettagliata delle predizioni del modello, evidenziando le aree di forza e le principali difficoltà nella classificazione delle famiglie di malware:

- **Adware:** Questa classe è ben classificata, con 380 predizioni corrette su 446. Tuttavia, alcune istanze sono confuse con Trojan (31) e Virus (33), suggerendo una certa sovrapposizione nelle caratteristiche di queste classi.

- **Backdoor:** Solo 61 campioni su 135 sono classificati correttamente. Molti errori coinvolgono Trojan (39) e Downloader (12), evidenziando che queste classi presentano pattern simili.
- **Downloader:** La performance per questa classe è particolarmente scarsa, con solo 37 predizioni corrette su 123. La maggior parte degli errori si verifica verso Trojan (65), indicando che il modello non riesce a distinguere efficacemente tra queste due categorie.
- **Ransomware:** Questa è una delle classi meglio classificate, con 230 predizioni corrette su 256. Gli errori principali si distribuiscono tra Trojan (16) e Virus (9), ma in generale il modello gestisce bene questa classe.
- **Spyware:** Nonostante abbia solo 43 campioni, il modello non riesce a classificare correttamente alcuna istanza. Gli errori principali coinvolgono Trojan (30), suggerendo una sovrapposizione significativa nelle caratteristiche.
- **Trojan:** Con 334 predizioni corrette su 404, il modello gestisce bene questa classe. Tuttavia, vi è confusione con Adware (17) e Downloader (9), evidenziando la necessità di migliorare il modello per separare meglio queste categorie.
- **Virus:** Con 114 predizioni corrette su 205, la classe Virus presenta errori significativi, principalmente verso Trojan (42) e Adware (44). Questo suggerisce che il modello fatica a distinguere tra Virus e altre classi con pattern simili.

In generale, la matrice di confusione mostra che il modello funziona bene per classi come Adware e Ransomware, ma ha difficoltà significative con le classi Spyware, Downloader e Backdoor. Questi risultati evidenziano la necessità di ulteriori ottimizzazioni per migliorare la discriminazione tra classi con pattern sovrapposti e rafforzare la gestione delle classi meno rappresentate.

6.1.2 Risultati Esperimento 2

6.1.2.1 Metriche di valutazione

Classe	Precision	Recall	F1-Score	Support
Adware	0.87	0.91	0.89	446
Backdoor	0.87	0.72	0.79	135
Downloader	0.71	0.61	0.66	123
Ransomware	0.86	0.94	0.90	256
Spyware	0.59	0.51	0.55	43
Trojan	0.77	0.79	0.78	404
Virus	0.73	0.68	0.70	205
Accuracy		**0.81**		1612
Macro Avg	0.77	0.74	0.75	1612
Weighted Avg	0.80	0.81	0.80	1612

Tabella 6.3: *Report di classificazione del modello CNN per la rilevazione di malware*

La tabella 6.3 riassume le prestazioni del modello CNN per la classificazione delle diverse famiglie di malware, utilizzando metriche standard come precision, recall e F1-Score. I punti principali da evidenziare sono i seguenti:

- **Adware:** Questa classe mostra prestazioni eccellenti, con valori di precision, recall e F1-Score pari a 0.87, 0.91 e 0.89 rispettivamente. Ciò indica che il modello è molto efficace nel riconoscere i campioni di Adware.
- **Backdoor:** Nonostante una precision alta (0.87), il recall è inferiore (0.72), evidenziando difficoltà del modello nel riconoscere tutti i campioni di Backdoor.
- **Downloader:** Questa classe presenta prestazioni moderate, con un F1-Score di 0.66. La precision (0.71) e il recall (0.61) suggeriscono che il modello ha problemi a distinguere Downloader da altre classi.
- **Spyware:** Mostra performance deboli, con un F1-Score di 0.55. Sebbene la precision sia accettabile (0.59), il basso recall (0.51) indica che molti campioni di Spyware non vengono correttamente identificati.
- **Ransomware:** Questa classe è una delle meglio riconosciute dal modello, con un F1-Score di 0.90 e recall molto alto (0.94).
- **Trojan:** La precision (0.77) e il recall (0.79) portano a un F1-Score di 0.78. Ciò suggerisce una buona capacità del modello nel classificare questa classe.
- **Virus:** Con un F1-Score di 0.70, questa classe è moderatamente ben gestita, ma i valori di precision e recall (0.73 e 0.68) indicano ancora una certa confusione con altre classi.

L'accuracy complessiva è pari a 80.71%, indicando buone prestazioni globali del modello. Tuttavia, le metriche *macro average* (0.75 per F1-Score) evidenziano che alcune classi meno rappresentate, come Spyware, richiedono ulteriori ottimizzazioni.

6.1.2.2 Matrice di Confusione

Classe	Adware	Backdoor	Downloader	Ransomware	Spyware	Trojan	Virus
Adware	407	0	5	1	3	15	15
Backdoor	7	97	1	4	1	18	7
Downloader	5	2	75	5	2	26	8
Ransomware	0	0	5	240	1	10	0
Spyware	1	2	4	2	22	10	2
Trojan	16	7	10	23	7	320	21
Virus	34	3	5	5	1	17	140

Tabella 6.4: *Matrice di Confusione: confronto tra classi reali (righe) e classi predette (colonne)*

La matrice di confusione riportata in Tabella 6.4 evidenzia le seguenti osservazioni:

- **Adware:** Con 407 predizioni corrette su 446, questa classe è ben classificata. Tuttavia, alcune istanze vengono confuse principalmente con Trojan (15) e Virus (15).
- **Backdoor:** Ha 97 predizioni corrette su 135. Gli errori principali coinvolgono Trojan (18) e Adware (7), suggerendo che queste classi presentano caratteristiche sovrapposte.
- **Downloader:** Mostra 75 predizioni corrette su 123, con errori significativi verso Trojan (26) e Virus (8), indicando difficoltà del modello nel distinguere tra queste categorie.
- **Ransomware:** Questa classe è ben riconosciuta, con 240 predizioni corrette su 256. Gli errori principali si distribuiscono tra Trojan (10) e Virus (5).
- **Spyware:** Con solo 22 predizioni corrette su 43, il modello confonde spesso questa classe con Trojan (10) e Adware (1), evidenziando una scarsa capacità di discriminazione.
- **Trojan:** È una delle classi più rappresentate, con 320 predizioni corrette su 404. Tuttavia, vi è una confusione moderata con Adware (16) e Downloader (10).
- **Virus:** Ha 140 predizioni corrette su 205. Gli errori principali coinvolgono Trojan (17) e Adware (34), suggerendo difficoltà nel distinguere Virus da altre classi con pattern simili.

La matrice di confusione evidenzia che, sebbene il modello funzioni bene per classi come Adware e Ransomware, presenta difficoltà significative nel distinguere classi meno rappresentate o con caratteristiche sovrapposte, come Spyware e Downloader.

6.1.3 Risultati Esperimento 3

Grazie alla configurabilità dinamica degli iperparametri e al processo di tuning, questo approccio consente di adattare il modello alle specifiche del dataset e di massimizzarne le prestazioni nella classificazione dei malware.

6.1.3.1 Metriche di valutazione

La Tabella 6.5 presenta le metriche di classificazione del modello CNN per ciascuna delle classi di malware. Di seguito sono evidenziati i principali risultati:

- **Adware:** Questa classe raggiunge le migliori prestazioni complessive, con precision, recall e F1-Score molto elevati (rispettivamente 0.87, 0.93 e 0.90). Ciò indica che il modello identifica correttamente la maggior parte dei campioni di Adware, con pochi falsi positivi e falsi negativi.
- **Backdoor:** Con un F1-Score di 0.80, il modello mostra una buona capacità di classificare i campioni di Backdoor. Tuttavia, la precision (0.81) e il recall (0.78) evidenziano una certa confusione rispetto ad altre classi.
- **Downloader:** Le prestazioni per questa classe sono moderate, con un F1-Score di 0.65. Il recall più basso (0.56) suggerisce che il modello non riesce a riconoscere una porzione significativa di campioni di Downloader, anche se la precision (0.78) indica che i campioni classificati come Downloader sono in gran parte corretti.

Classe	Precision	Recall	F1-Score	Support
Adware	0.87	0.93	0.90	446
Backdoor	0.81	0.78	0.80	135
Downloader	0.78	0.56	0.65	123
Ransomware	0.85	0.95	0.89	256
Spyware	0.66	0.49	0.56	43
Trojan	0.76	0.82	0.79	404
Virus	0.81	0.65	0.72	205
Accuracy		**0.82**		1612
Macro Avg	0.79	0.74	0.76	1612
Weighted Avg	0.81	0.82	0.81	1612

Tabella 6.5: *Report di classificazione del modello CNN per la rilevazione di malware*

- **Ransomware:** Questa classe è una delle meglio gestite dal modello, con un F1-Score di 0.89 e un recall molto alto (0.95), dimostrando che il modello identifica correttamente quasi tutti i campioni di Ransomware.
- **Spyware:** Le prestazioni per questa classe sono significativamente inferiori rispetto alle altre, con un F1-Score di 0.56 e un recall particolarmente basso (0.49). Ciò suggerisce difficoltà significative nel distinguere i campioni di Spyware da quelli di altre classi.
- **Trojan:** Con un F1-Score di 0.79, il modello dimostra una buona capacità di identificare i Trojan. Il recall (0.82) indica che molti campioni di questa classe vengono correttamente riconosciuti, sebbene vi siano alcune confusioni con altre categorie.
- **Virus:** Le prestazioni per questa classe sono moderate, con un F1-Score di 0.72. Il recall più basso (0.65) suggerisce che alcuni campioni di Virus non vengono identificati correttamente.

L'accuracy globale del modello è del 82%, indicando buone prestazioni complessive. Tuttavia, le metriche *Macro Avg* (F1-Score di 0.76) e *Weighted Avg* (F1-Score di 0.81) riflettono una distribuzione non uniforme delle prestazioni tra le diverse classi, con Spyware e Downloader che rappresentano i maggiori punti critici.

6.1.3.2 Matrice di Confusione

La matrice di confusione riportata in Tabella 6.6 fornisce una visione dettagliata delle predizioni del modello, evidenziando le aree di forza e le principali difficoltà:

- **Adware:** Con 415 predizioni corrette su 446, questa classe è ben classificata. Gli errori principali si distribuiscono tra Trojan (15) e Virus (10), indicando una sovrapposizione moderata nelle caratteristiche.
- **Backdoor:** Ha 105 predizioni corrette su 135. Gli errori principali coinvolgono Trojan (20), evidenziando che queste classi presentano caratteristiche sovrapposte che rendono difficile la discriminazione.

Classe	Adware	Backdoor	Downloader	Ransomware	Spyware	Trojan	Virus
Adware	415	1	3	0	2	15	10
Backdoor	4	105	2	3	1	20	0
Downloader	7	10	69	6	1	27	3
Ransomware	0	0	3	242	1	10	0
Spyware	1	1	3	3	21	14	0
Trojan	13	8	5	25	4	331	18
Virus	38	4	4	7	2	17	133

Tabella 6.6: *Matrice di Confusione: confronto tra classi reali (righe) e classi predette (colonne)*

- **Downloader:** Mostra 69 predizioni corrette su 123, con errori significativi verso Trojan (27). Ciò suggerisce che il modello fatica a distinguere Downloader da altre categorie.
- **Ransomware:** Questa classe è ben gestita dal modello, con 242 predizioni corrette su 256. Gli errori principali si distribuiscono tra Trojan (10), ma il recall elevato riflette la buona capacità del modello nel riconoscere questa classe.
- **Spyware:** Con 21 predizioni corrette su 43, il modello presenta difficoltà significative, con errori che si concentrano su Trojan (14). Questo risultato evidenzia che Spyware è una classe particolarmente complessa da distinguere.
- **Trojan:** È una delle classi più rappresentate, con 331 predizioni corrette su 404. Tuttavia, vi è una confusione moderata con Adware (13) e Downloader (5).
- **Virus:** Ha 133 predizioni corrette su 205, ma subisce una confusione significativa con Adware (38) e Trojan (17), evidenziando che queste classi condividono alcune caratteristiche che rendono la distinzione più difficile.

In generale, la matrice di confusione evidenzia che il modello funziona bene per classi come Adware e Ransomware, ma presenta difficoltà significative nel distinguere classi meno rappresentate o con caratteristiche sovrapposte, come Spyware e Downloader.

6.2 Risultati degli Esperimenti GAN

6.2.1 Risultati Esperimento 1

Di seguito vengono analizzati i risultati del primo esperimento condotto con il modello GAN.

6.2.1.1 Risultati Blackbox

L'analisi dei risultati dell'esperimento mostra un comportamento interessante nel confronto tra il modello Blackbox e il modello sostitutivo (Substitute Detector), in relazione al livello di rumore e al numero di epoche di addestramento. Come visibile nel grafico

"Performance and Noise vs Epochs", la accuratezza del modello Blackbox mostra un calo significativo durante le prime epoche, stabilizzandosi su valori piuttosto bassi man mano che il livello di rumore generato dal modello aumenta. Questo risultato evidenzia la capacità del generatore di degradare l'efficacia del modello Blackbox, introducendo rumore che ne compromette la precisione. All'inizio dell'addestramento (ad esempio, Epoch 0), il modello Blackbox registra una precisione del 54.81%, con un livello di rumore relativamente basso (10.60%). Tuttavia, già dopo poche epoche (Epoch 5-10), l'accuratezza cala rapidamente, stabilizzandosi intorno al 25-30%, accompagnata da un livello di rumore in costante aumento (oltre il 50% intorno a Epoch 50 e prossima al 99% alla fine dell'addestramento). Questo suggerisce che il modello generativo riesce ad aumentare gradualmente l'intensità del rumore, riducendo significativamente l'efficacia del classificatore Blackbox. In termini di metriche complementari, come precisione e F1-score, si osserva un trend simile: entrambe subiscono un degrado progressivo, confermando l'impatto distruttivo del rumore sull'affidabilità delle previsioni del Blackbox.

6.2.1.2 Risultati Substitute Detector

Al contrario, il modello sostitutivo (Substitute Detector) mostra una performance decisamente più stabile. La sua accuratezza iniziale è intorno al 62.57% (Epoch 0) e aumenta progressivamente fino a stabilizzarsi su valori prossimi all'80% nelle ultime epoche, nonostante l'aumento del livello di rumore. Ciò indica che il modello sostitutivo è più robusto all'introduzione di rumore, probabilmente grazie a un'architettura più adattata a questo tipo di perturbazioni. Questa tendenza è confermata anche da altre metriche, come la precisione e il F1-score, che mostrano un miglioramento costante durante l'addestramento, fino a raggiungere valori molto alti (circa 0.80 per entrambe le metriche nelle ultime epoche).

6.2.1.3 Impatto del livello di rumore

Un'analisi combinata dei dati mostra che il livello di rumore generato dal modello GAN aumenta progressivamente con le epoche, partendo da un valore iniziale del 10.60% e raggiungendo il 99.33% alla fine dell'addestramento (Epoch 99). Questo incremento è direttamente correlato alla diminuzione delle prestazioni del Blackbox, che non è in grado di adattarsi all'aumento delle perturbazioni introdotte dal generatore. Tuttavia, il modello sostitutivo riesce a mantenere elevate prestazioni anche in condizioni di rumore elevato, dimostrando una capacità di generalizzazione superiore rispetto al Blackbox. Questo risultato suggerisce che il modello sostitutivo potrebbe essere utilizzato come strategia difensiva contro attacchi che introducono rumore deliberato.

6.2.1.4 Conclusioni

I risultati di questo esperimento evidenziano il successo del generatore nel compromettere l'affidabilità del modello Blackbox, grazie alla progressiva introduzione di rumore. Al contempo, il modello sostitutivo si dimostra più resiliente e adattabile, mantenendo elevate prestazioni anche in condizioni di forte rumore. Questi risultati sottolineano l'importanza di progettare modelli robusti per affrontare scenari in cui l'avversario può sfruttare tecniche di generazione del rumore per compromettere l'accuratezza delle previsioni.

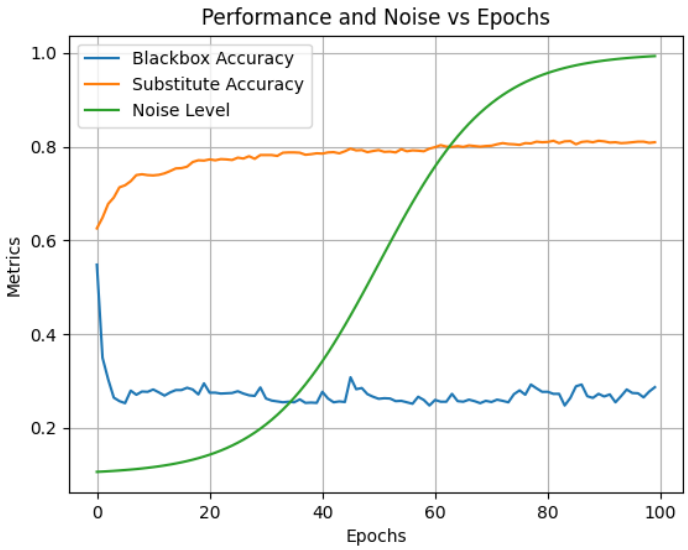

Figura 6.1: *Andamento delle metriche su 100 epoche di addestramento*

6.2.2 Risultati Esperimento 2

Di seguito vengono analizzati i risultati del secondo esperimento condotto con il modello GAN.

6.2.2.1 Risultati Blackbox

I risultati mostrano che il modello Blackbox è fortemente influenzato dall'aumento del livello di rumore generato. Mentre all'inizio dell'addestramento le prestazioni sono moderate, con un'accuratezza intorno al 60%, queste subiscono un degrado significativo con il crescere del rumore. Nelle fasi finali dell'addestramento, quando il livello di rumore raggiunge valori elevati, il modello diventa incapace di fare previsioni significative, con metriche che si stabilizzano su livelli molto bassi. Questo comportamento evidenzia la vulnerabilità del Blackbox a perturbazioni avversarie.

6.2.2.2 Risultati Substitute Detector

Il Substitute Detector, al contrario, si dimostra molto più resiliente al rumore. Fin dalle prime fasi dell'addestramento, le sue prestazioni sono superiori rispetto al Blackbox, e continuano a migliorare progressivamente. Anche in presenza di un rumore significativo, il Substitute mantiene un'elevata accuratezza e robustezza, confermandosi una soluzione più adatta in contesti avversari.

6.2.2.3 Impatto del livello di rumore

L'impatto del rumore è evidente, poiché il suo progressivo aumento compromette rapidamente le prestazioni del Blackbox. Il GAN genera perturbazioni sempre più efficaci, che rendono il modello Blackbox incapace di adattarsi. Tuttavia, il Substitute

Detector dimostra di essere meno sensibile a questo fenomeno, mantenendo alte prestazioni anche in presenza di livelli di rumore molto elevati.

6.2.2.4 Conclusioni

In conclusione, l'esperimento evidenzia la vulnerabilità del modello Blackbox e l'efficacia del Substitute Detector nel mantenere alte prestazioni nonostante l'incremento del rumore. Il GAN si conferma una strategia efficace nel generare perturbazioni avversarie, ma i risultati suggeriscono che modelli più robusti, come il Substitute, possono rappresentare una valida difesa contro questo tipo di attacchi.

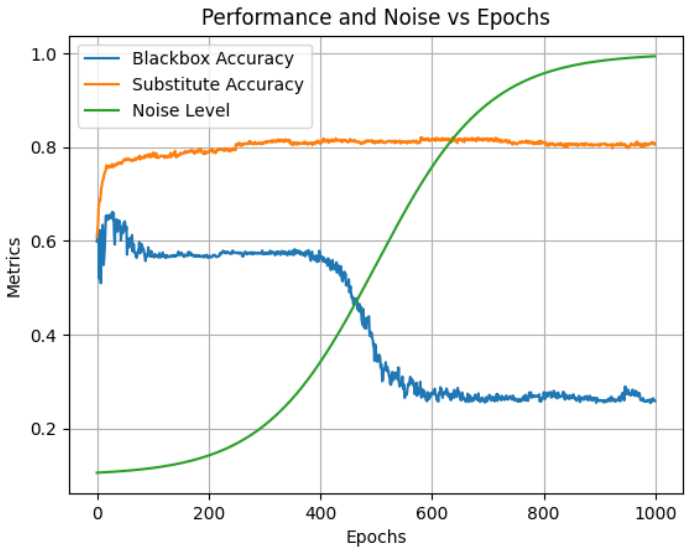

Figura 6.2: *Andamento delle metriche su 1000 epoche con batch size 64*

6.2.3 Risultati Esperimento 3

Di seguito viene presentata un'analisi dei risultati del terzo esperimento condotto con il modello GAN.

6.2.3.1 Risultati Blackbox

Si può notare come, a differenza dell'esperimento precedente, il modello Blackbox (lina blu) parta con un'accuratezza inferiore al 40%, che diminuisce drasticamente nelle prime epoche di addestramento, fino a raggiungere un livello inferiore al 25%. Tuttavia, nonostante l'aumento del rumore nelle ultime epoche finali, si può notare come il modello mantenga una certa capacità di classificazione, seppur molto limitata.

6.2.3.2 Risultati Substitute Detector

L'accuratezza del discriminatore (linea arancione) si stabilizza rapidamente oltre l'80%, mostrando una capacità costante di classificare correttamente sia le immagini reali sia

quelle generate. Questo risultato evidenzia che il discriminatore riesce ad adattarsi efficacemente alle immagini alterate create dal generatore.

6.2.3.3 Impatto del livello di rumore

Il livello di rumore (linea verde) aumenta in modo progressivo durante le epoche, seguendo un andamento sigmoideo. Questo incremento controllato del rumore contribuisce sia alla riduzione dell'accuratezza del Blackbox sia al mantenimento delle prestazioni del discriminatore. La sinergia tra questi elementi è cruciale per il successo dell'addestramento.

6.2.3.4 Conclusioni

Il grafico mostra che il framework proposto riesce a ottenere un compromesso efficace: il generatore inganna il modello Blackbox riducendone l'accuratezza, mentre il discriminatore mantiene prestazioni elevate. L'aumento progressivo del rumore è un fattore determinante nel raggiungimento di questi risultati, dimostrando l'importanza della strategia di addestramento adottata.

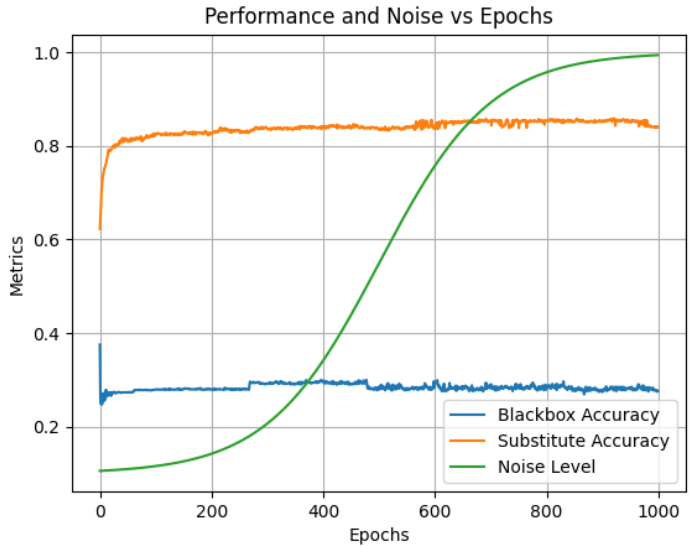

Figura 6.3: *Andamento delle metriche su 1000 epoche con batch size 32*

6.2.4 Applicazione Grad-CAM

La Grad-CAM (Gradient-weighted Class Activation Mapping)[15] è stata utilizzata in questo studio come strumento fondamentale per interpretare il comportamento dei modelli in presenza di rumore crescente. L'obiettivo principale dell'adozione della Grad-CAM è stato quello di ottenere una visualizzazione chiara delle aree di interesse del modello durante la classificazione, specialmente quando il livello di rumore incrementale introdotto dal generatore potrebbe alterare le decisioni del modello.

6.2.4.1 Interpretabilità del modello

La Grad-CAM consente di evidenziare le regioni dell'immagine che contribuiscono maggiormente alla decisione del modello. Questa capacità di visualizzazione aiuta a comprendere se il modello si concentra su caratteristiche rilevanti o viene invece distratto da perturbazioni indotte dal rumore.

6.2.4.2 Valutazione dell'impatto del rumore

L'adozione della Grad-CAM permette di analizzare come l'incremento del rumore influenza le regioni di attenzione del modello. Come mostrato nelle immagini, al crescere del rumore, le attivazioni diventano sempre più distribuite in maniera casuale, indicando che il modello potrebbe perdere la capacità di concentrarsi sulle caratteristiche salienti dell'immagine.

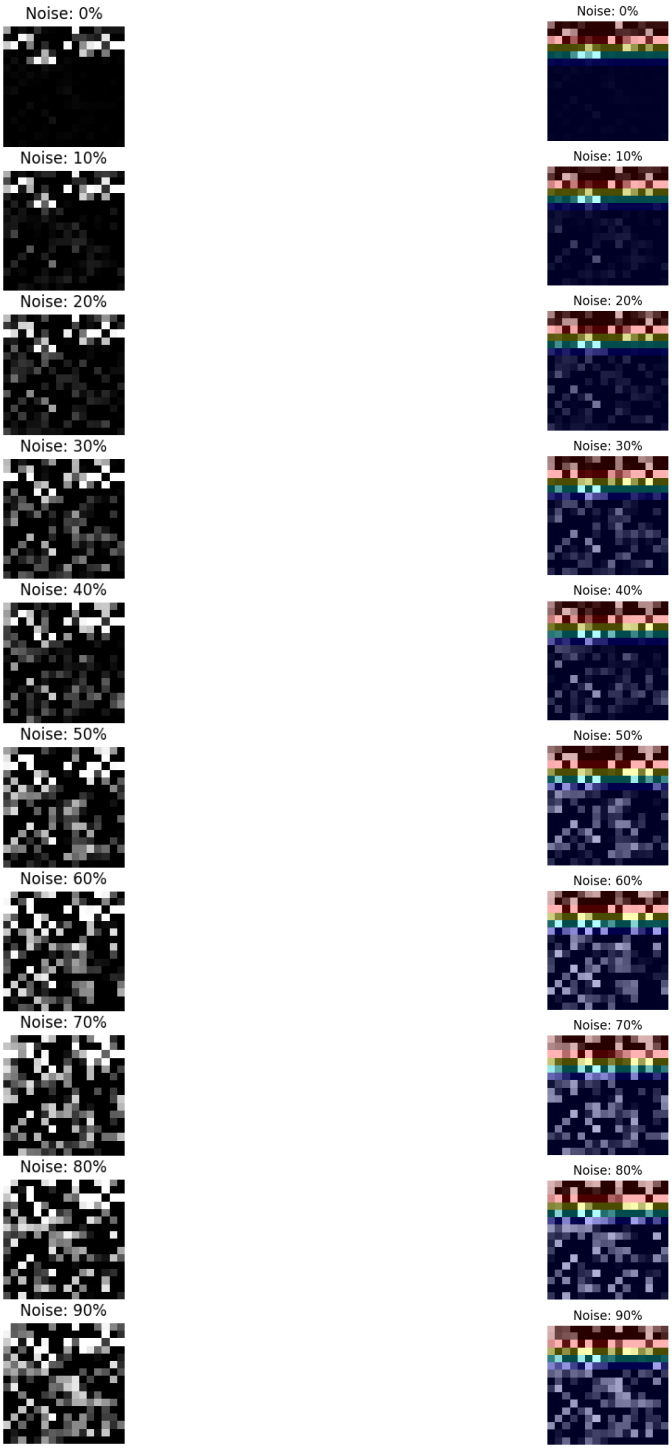

Figura 6.4: *Rumore generato dal generatore (a sinistra) e Grad-CAM applicato al modello sostitutivo (a destra) per un malware di famiglia Trojan*

Capitolo 7

Conclusioni e Sviluppi Futuri

7.1 Consuntivo finale

Questo studio ha esplorato l'efficacia delle Generative Adversarial Networks (GAN) nel contesto della rilevazione, classificazione e generazione di malware, con un focus specifico sulle capacità del generatore di eludere il discriminatore. L'analisi dettagliata ha permesso di confermare le ipotesi iniziali, dimostrando che le GAN possono efficacemente identificare e classificare le diverse famiglie di malware presenti nel dataset utilizzato. Importante è stato il riscontro sulla capacità del generatore di incrementare il livello di rumore nelle immagini in scala di grigi per eludere il riconoscimento del discriminatore. Questo ha evidenziato non solo la vulnerabilità dei sistemi di rilevamento basati su apprendimento automatico, ma anche la potenzialità delle GAN nel generare varianti di malware che possono bypassare i meccanismi di sicurezza tradizionali. Le implicazioni di queste scoperte sono significative per la sicurezza informatica. Sottolineano la necessità di continuare a sviluppare e affinare gli algoritmi di apprendimento automatico per tenere il passo con le tecniche sofisticate di evasione. Inoltre, i risultati suggeriscono che l'impiego di GAN potrebbe rivoluzionare il modo in cui gli esperti di sicurezza approcciano la creazione di sistemi di difesa, promuovendo un modello in cui i sistemi di rilevazione sono costantemente testati e migliorati contro attacchi generati artificialmente.

7.2 Sviluppi Futuri

Nonostante il presente studio si sia soffermato principalmente su come un modello possa riconoscere le famiglie di malware presenti su un dataset molto piccolo, ci sono numerose direzioni che si potrebbero esplorare in futuro per ampliare e potenziare le applicazioni di questa ricerca. Alcuni sviluppi interessanti includono:

- **Generazione di campioni di malware realistici**: Le GAN potrebbero essere sfruttate per generare nuovi campioni di malware che simulano il comportamento di malware esistenti. Questo permetterebbe di:
  - Ampliare artificialmente dataset di malware, facilitando l'addestramento di modelli più robusti.
  - Simulare varianti future di malware, aiutando i ricercatori a prevedere come le famiglie di malware potrebbero evolversi.

- **Creazione di campioni adversariali:** Le GAN possono essere utilizzate per generare campioni adversariali, ovvero input progettati per ingannare i sistemi di rilevazione. Questo approccio può aiutare a:
  - Identificare le vulnerabilità dei modelli attuali di rilevazione malware.
  - Migliorare la robustezza dei modelli di classificazione, rendendoli meno suscettibili agli attacchi.
- **Rilevazione di comportamenti anomali:** Una GAN può essere addestrata a generare rappresentazioni "normali" di codice o dati. Eventuali discrepanze tra i dati reali e quelli generati dalla GAN possono evidenziare anomalie, indicando la presenza di comportamenti sospetti o potenzialmente dannosi.
- **Ricostruzione di payload parziali:** Nel caso in cui vengano rilevati solo frammenti di malware (ad esempio, file corrotti o parzialmente cancellati), le GAN potrebbero essere utilizzate per ricostruire i payload completi. Questo aiuterebbe ad analizzare le minacce in modo più efficace, anche quando i dati non sono completi.
- **Ottimizzazione dei sistemi di rilevazione:** Integrando le GAN in un framework di rilevazione, si potrebbero creare scenari di apprendimento continuo, in cui i sistemi di difesa vengono costantemente messi alla prova da campioni generati in modo dinamico. Questo approccio potrebbe essere particolarmente utile per sistemi che monitorano reti aziendali in tempo reale.
- **Sviluppo di nuove tecniche di steganografia e anti-steganografia:** Le GAN possono essere utilizzate per sviluppare e analizzare tecniche di steganografia (cioè nascondere codice malevolo all'interno di dati innocui). Al contempo, possono essere utilizzate per costruire sistemi in grado di rilevare e prevenire tali tecniche.

Infine, oltre agli usi diretti delle GAN, ulteriori sviluppi futuri potrebbero includere:

- **Espansione dei dataset:** Creazione di dataset più ampi e diversificati, includendo campioni di malware più recenti e provenienti da diverse piattaforme.
- **Rilevazione in tempo reale:** Integrazione dei modelli con sistemi di monitoraggio in tempo reale, per identificare e bloccare il malware durante la sua esecuzione.
- **Evoluzione verso modelli multimodali:** Combinare diverse fonti di informazione (come analisi statiche e dinamiche) per migliorare la capacità del modello di distinguere tra software legittimo e malevolo.

In sintesi, questi strumenti potrebbero preparare il terreno per nuove generazioni di tecnologie di sicurezza informatica.

Glossario

Adam Optimizer Un algoritmo di ottimizzazione che combina i benefici di AdaGrad e RMSProp, adattando i tassi di apprendimento per ciascun parametro.. 16

Batch Size Il numero di campioni utilizzati per aggiornare i pesi di un modello di machine learning durante l'addestramento.. 16

Blackbox Un modello di machine learning il cui funzionamento interno non è trasparente o comprensibile, ma è valutato solo in base agli input e agli output.. 24

Confusion Matrix Una tabella utilizzata per valutare le prestazioni di un modello di classificazione, confrontando i valori predetti con quelli reali.. 16

Cybersecurity L'insieme delle azioni volte a difendere computer, server, dispositivi mobili, sistemi elettronici, reti e dati dalle minacce informatiche. 1

Dropout Una tecnica di regolarizzazione utilizzata nelle reti neurali per prevenire l'overfitting, disattivando casualmente neuroni durante l'addestramento.. 15

Dynamic Analysis Una tecnica di analisi del software che studia il comportamento di un programma durante la sua esecuzione.. 4

Generative Adversarial Networks (GAN) Una classe di algoritmi di machine learning in cui due reti neurali, un generatore e un discriminatore, competono per migliorare le loro prestazioni.. 1

Hyperparameter Un parametro che controlla il processo di addestramento di un modello di machine learning, non appreso direttamente dai dati.. 16

Overfitting Una condizione in cui un modello di machine learning si adatta troppo ai dati di addestramento, riducendo le prestazioni sui dati di test.. 15

Reverse Engineering Il processo di analisi di un sistema software o hardware per comprenderne la struttura, le funzionalità e il funzionamento.. 15

Static Analysis Una tecnica di analisi del software che esamina il codice sorgente o binario senza eseguire il programma.. 3

Bibliografia

Articoli scientifici consultati

[1] Mianxing Dong Masataka Kawai Kaoru Ota. «Improved MalGAN: Avoiding Malware Detector by Leaning Cleanware Features». In: (2019). DOI: 10.1109/IITCEE59897.2024.10467665 (cit. a p. 8).

[2] Lara Saidia Fascía et al. «Disarming visualization-based approaches in malware detection systems». In: (2022). DOI: 10.1109/IITCEE59897.2024.10467665 (cit. a p. 8).

[3] Akhil M R et al. «Deep Learning for Cyberthreats: Performance Analysis and Application of Malware Classification in Edge Computing». In: (2024). DOI: 10.1109/IITCEE59897.2024.10467665 (cit. a p. 8).

[4] Osho Sharma, Akashdeep Sharma e Arvind Kalia. «MIGAN: GAN for facilitating malware image synthesis with improved malware classification on novel dataset». In: (2024). DOI: 10.1109/IITCEE59897.2024.10467665 (cit. a p. 9).

[8] Ian J. Goodfellow et al. «Generative Adversarial Nets». In: (2014). DOI: 10.1109/IITCEE59897.2024.10467665. URL: https://proceedings.neurips.cc/paper_files/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf (cit. alle pp. 22, 23).

[13] «Principal Component Analysis». In: (). URL: https://www.dmf.unisalento.it/~straf/allow_listing/pub/did/PhD/capPCA.pdf (cit. a p. 34).

Siti web consultati

[5] *Malware*. URL: https://www.avg.com/it/signal/what-is-malware (cit. a p. 10).

[6] *Getting started with KerasTuner*. URL: https://keras.io/keras_tuner/getting_started/ (cit. alle pp. 15, 22).

[7] *TensorFlow*. URL: https://www.tensorflow.org/ (cit. alle pp. 15, 22).

[9] *Machine Learning Tutorials*. URL: https://github.com/maelfabien/Machine_Learning_Tutorials/blob/master/4-DeepLearning/GANs/GAN_MNIST.ipynb (cit. a p. 29).

[10] *AVClass2*. URL: https://github.com/malicialab/avclass (cit. alle pp. 32, 33).

[11] *Detect It Easy*. URL: https://github.com/horsicq/Detect-It-Easy (cit. a p. 33).

[12] *TF-IDF*. URL: https://www.geeksforgeeks.org/visualizing-tf-idf-scores-a-comprehensive-guide-to-plotting-a-document-tf-idf-2d-graph/ (cit. a p. 33).

[14] *Malware GAN attack*. URL: https://github.com/lzylucy/Malware-GAN-attack (cit. a p. 35).

[15] *Grad-CAM class activation visualization*. URL: https://keras.io/examples/vision/grad_cam/ (cit. a p. 53).

[16] *Python Graph Gallery*. URL: https://python-graph-gallery.com/.

[17] *Machine Learning Tutorials*. URL: https://github.com/bnsreenu/python_for_microscopists.

[18] *Image-based Malware Classification using CNN*. URL: https://github.com//malware-image-detection.

[19] *VirusTotal*. URL: https://www.virustotal.com/gui/home/upload.

[20] *VirusShare*. URL: https://virusshare.com/.

[21] *MalShare*. URL: https://malshare.com/.

[22] *MalwareBazaar*. URL: https://bazaar.abuse.ch/.

[23] *Deep Learning Fusion For Effective Malware Detection: Leveraging Visual Features*. URL: https://ar5iv.labs.arxiv.org/html/2405.14311v1.