

UNIVERSITÀ DEGLI STUDI DI MILANO
FACOLTÀ DI SCIENZE E TECNOLOGIE

DIPARTIMENTO DI INFORMATICA
GIOVANNI DEGLI ANTONI



*CORSO DI LAUREA MAGISTRALE IN
SICUREZZA INFORMATICA*

ANALISI DELLA DATA AUGMENTATION BASATA SU
GAN APPLICATA AGLI INTRUSION DETECTION
SYSTEM

Relatore: Prof. Stelvio CIMATO

Tesi di:
Giulia Francesca CONTARDI
Matr. Nr. 984239

ANNO ACCADEMICO 2023-2024

*A Mamma e Papà che hanno sempre creduto in me
e mi hanno insegnato a non mollare mai.*

A noi donne.

Indice

Indice	ii
Elenco delle tabelle	v
Elenco delle figure	vii
1 Introduzione	1
1.1 Organizzazione della tesi	3
2 Intrusion Detection System	4
2.1 Tipologie NIDS e HIDS	6
2.2 Rilevamento “Signature-based” o “Anomaly-based”	7
3 ML e DL per la Sicurezza Informatica	9
3.1 Concetti fondamentali	10
3.1.1 Tipologie di apprendimento	11
3.1.2 Il Deep Learning e le Reti Neurali	14
3.1.3 Il meccanismo di Self-attention	17
3.2 ML-IDS e DL-IDS allo stato dell’arte	19
3.3 Approccio centralizzato e distribuito	21
4 Federated Learning	23
4.1 Caratteristiche	23
4.2 Algoritmi progettati allo stato dell’arte	25
4.2.1 FedAvg	26

4.2.2	FedProx	26
4.2.3	Altri algoritmi rilevanti	28
4.3	Vulnerabilità	29
4.4	Problemi generali	30
4.5	FL-IDS	31
5	GAN - Generative Adversarial Network	33
5.1	Tipologie	33
5.1.1	Vanilla GAN	34
5.1.2	Altri tipi di GAN per la Cybersecurity	35
5.2	Vantaggi e problemi riscontrati	37
5.3	Applicazione negli IDS allo stato dell'arte	38
6	Analisi di tecniche NIDS basate su GAN	40
6.1	Caratteristiche tecniche del framework	40
6.2	Dataset e Pre-processing	41
6.3	Caso di studio ed architettura	46
6.4	Metriche di valutazione	47
6.5	Sviluppo del modello GAN	51
6.5.1	Valutazione dei dati generati	60
6.6	Sviluppo degli IDS e Data Augmentation	75
6.6.1	Valutazione della Data Augmentation	91
6.6.2	Conclusioni dell'analisi	104
7	Conclusioni e sviluppi futuri	108
A	Preprocessing	112
B	GAN	115
C	IDS	127
Bibliografia		142

Elenco delle tabelle

1	Elenco iniziale dei label del traffico.	42
2	Elenco delle 35 feature selezionate.	45
3	Iperparametri scelti che causano un apprendimento non efficiente relativi alla figura 9.	56
4	Iperparametri ottimizzati dopo l'esempio n°1 relativi alla figura 10. .	56
5	Iperparametri finali configurati per il modello GAN.	57
6	Analisi statistiche sui dati generati di tipo Botnet nelle due configurazioni di iperparametri.	63
7	Analisi statistiche sui 10000 dati generati di tipo Botnet nelle due configurazioni di iperparametri.	66
8	Analisi statistiche sui dati generati di tipo Bruteforce nelle due configurazioni di iperparametri.	66
9	Analisi statistiche sui 10000 dati generati di tipo Bruteforce nelle due configurazioni di iperparametri.	68
10	Iperparametri relativi alla fase di training del modello IDS-MLP. . .	75
11	Risultati della fase di testing del modello IDS-MLP.	77
12	Valori TP, TN, FP, FN relativi alle performance dell'IDS-MLP. . .	79
13	Valutazione delle performance di classificazione dell'IDS-MLP. . .	80
14	Risultati della fase di testing del modello IDS-MLP con Class Weighting.	80
15	Valori TP, TN, FP, FN relativi alle performance dell'IDS-MLP con Class Weighting.	83
16	Valutazione delle performance di classificazione dell'IDS-MLP con Class Weighting.	84

17	Configurazione di parametri del modello RFC usato come secondo IDS.	85
18	Confronto Test Set e Samples predetti dall'IDS-RFC.	85
19	Valutazione delle performance di classificazione dell'IDS-RFC.	85
20	Valori TP, TN, FP, FN relativi alle performance dell'IDS-RFC.	88
21	Analisi della presenza di overfitting nell'IDS-RFC.	89
22	Samples per classe nella Data Augmentation con incremento del 100%.	92
23	Samples per classe nella Data Augmentation con incremento di 10000 samples per Botnet e Bruteforce.	92
24	Valutazione delle performance di classificazione dell'IDS-MLP applicando D.A. con e senza CW.	97
25	Valutazione delle performance di classificazione dell'IDS-MLP applicando D.A. con e senza CW solo su Botnet e Bruteforce.	98
26	Valutazione delle performance di classificazione dell'IDS-MLP aumentando di 10000 samples le classi Botnet e Bruteforce (con e senza CW).	100
27	Valutazione delle performance di D.A. migliori: analisi delle metriche delle sole classi minori dell'IDS-MLP.	101
28	Metriche rilevate da K-Cross Validation con K=5 dell'IDS-MLP per verificare l'attendibilità della performance esaminata.	103
29	Confronto dei risultati relativi alle prestazioni con ricerche allo stato dell'arte.	107
30	Confronto dei risultati relativi alle prestazioni con l'articolo accademico di Barkah et al.	107
31	Analisi statistiche sui dati generati di tipo DDoS nelle due configurazioni di iperparametri.	117
32	Analisi statistiche sui dati generati di tipo DoS nelle due configurazioni di iperparametri.	119
33	Analisi statistiche sui dati generati di tipo Portscan nelle due configurazioni di iperparametri.	123
34	Valutazione delle performance di classificazione dell'IDS-MLP aumentando di 10000 samples le classi Botnet e Bruteforce e del 100% le classi maggiori (con e senza CW).	138

Elenco delle figure

1	Struttura tipica di un IDS.	6
2	Relazione tra AI, ML e DL.	11
3	Modello GAN	35
4	Differenza tra Vanilla GAN e cGAN	36
5	Fasi di pre-processing adottate	41
6	Distribuzione del traffico malevolo e benigno dopo la pulizia del dataset.	43
7	Distribuzione del traffico per le 6 classi dopo la pulizia del dataset. . .	44
8	Distribuzione del traffico nel Train Set e nel Test Set.	46
9	Apprendimento fallace n°1 riscontrato durante l'ottimizzazione degli iperparametri in fase di sperimentazione del modello GAN.	55
10	Apprendimento con iperparametri ottimizzati del modello GAN successivamente all'esempio n°1.	56
11	Architettura del modello GAN.	58
12	Grafico Loss ed Accuratezza del modello GAN relativi alla classe Bruteforce con alpha=0,01.	60
13	Confronto tra distribuzioni della feature BwdPacketLengthStd per la classe Botnet nelle due configurazioni di iperparametri.	62
14	Confronto tra matrici di correlazione della classe Botnet nelle due configurazioni di iperparametri.	62
15	Confronto tra scatter plot dei dati generati e dei dati reali della classe Botnet nelle due configurazioni di iperparametri.	64
16	Confronto tra scatter plot della classe Botnet nelle due configurazioni di iperparametri per la generazione di 10000 samples.	65

17	Confronto tra matrici di correlazione della classe Bruteforce nelle due configurazioni di iperparametri.	67
18	Confronto delle distribuzioni dei dati generati rispetto ai dati reali di due feature per la classe Bruteforce nelle due configurazioni di iperparametri.	68
19	Confronto tra scatter plot della classe Bruteforce nelle due configurazioni di iperparametri.	69
20	Confronto tra scatter plot della classe Bruteforce nelle due configurazioni di iperparametri per la generazione di 10000 samples.	70
21	Confronto tra matrici di correlazione della classe Benign nelle due configurazioni di iperparametri.	72
22	Confronto tra distribuzioni di due feature per la classe Benign nelle due configurazioni di iperparametri.	73
23	Confronto tra scatter plot della classe Benign nelle due configurazioni di iperparametri.	74
24	Architettura del modello IDS-MLP	75
25	Grafici dell'apprendimento del IDS-MLP basilare.	76
26	Risultati di testing relativi alle performance dell'IDS-MLP.	77
27	Grafico performance dell'IDS-MLP.	78
28	Grafici dell'apprendimento del IDS-MLP che sfrutta Class Weighting.	81
29	Risultati di testing relativi alle performance dell'IDS-MLP che sfrutta ClassWeightiting.	81
30	Grafico performance dell'IDS-MLP con Class Weighting.	82
31	Risultati di testing relativi alle performance dell'IDS-RFC.	86
32	Grafico performance dell'IDS-RFC.	87
33	Curve di Apprendimento dell'IDS-RFC.	90
34	Confronto D.A. del 100% di tutte le classi nei due diversi valori di alpha.	93
35	Confronto D.A. del 100% delle sole classi Botnet e Bruteforce nei due diversi valori di alpha.	94
36	Confronto D.A. con l'aumento di 10000 samples, senza aumentare le classi maggiori, nei due diversi valori di alpha.	95

37	Confronto sui Report di Classificazione della D.A. di tutte le classi su IDS-MLP con e senza CW, nei due diversi valori di alpha.	97
38	Confronto sui Report di Classificazione della D.A. di Botnet e Bruteforce su IDS-MLP con e senza CW, nei due diversi valori di alpha. . .	98
39	Confronto sulle Matrici di Confusione della D.A. di Botnet e Bruteforce su IDS-MLP con CW nei due diversi valori di alpha.	99
40	Confronto sui Report di Classificazione della D.A. con aumento a 10000 samples delle classi Botnet e Bruteforce su IDS-MLP con e senza CW.	101
41	Confronto sulle Matrici di Confusione della D.A. di Botnet e Bruteforce su IDS-MLP nei due diversi valori di alpha.	102
42	Esito dell'analisi di importanza delle feature su Random Forest. Le feature sono ordinate in ordine decrescente secondo il punteggio ottenuto.	113
43	Grafico di correlazione delle feature prima della selezione.	114
44	Confronto tra distribuzioni di due feature per la classe DDoS nelle due configurazioni di iperparametri.	116
45	Confronto tra matrici di correlazione della classe DDoS nelle due configurazioni di iperparametri.	117
46	Confronto tra scatter plot della classe DDoS nelle due configurazioni di iperparametri.	118
47	Confronto tra distribuzioni di due feature per la classe DoS nelle due configurazioni di iperparametri.	120
48	Confronto tra matrici di correlazione della classe DoS nelle due configurazioni di iperparametri.	121
49	Confronto tra scatter plot della classe DoS nelle due configurazioni di iperparametri.	122
50	Confronto tra distribuzioni di tre feature per la classe Portscan nelle due configurazioni di iperparametri.	124
51	Confronto tra matrici di correlazione della classe Portscan nelle due configurazioni di iperparametri.	125
52	Confronto tra scatter plot della classe Portscan nelle due configurazioni di iperparametri.	126

53	Grafici Curve Loss e Accuratezza della D.A. al 100% di tutte le classi senza CW nei due diversi valori di alpha.	128
54	Grafici Curve Loss e Accuratezza della D.A. al 100% di tutte le classi con CW nei due diversi valori di alpha.	129
55	Grafici Curve Loss e Accuratezza della D.A. al 100% di Botnet e Bruteforce senza CW nei due diversi valori di alpha.	130
56	Grafici Curve Loss e Accuratezza della D.A. al 100% di Botnet e Bruteforce con CW nei due diversi valori di alpha.	131
57	Grafici Curve Loss e Accuratezza della D.A. con aumento di 10000 delle classi minori e 100% delle maggiori senza CW nei due diversi valori di alpha.	132
58	Grafici Curve Loss e Accuratezza della D.A. con aumento di 10000 delle classi minori e 100% delle maggiori con CW nei due diversi valori di alpha.	133
59	Grafici Curve Loss e Accuratezza della D.A. con aumento di 10000 delle sole classi minori senza CW nei due diversi valori di alpha. . . .	134
60	Grafici Curve Loss e Accuratezza della D.A. con aumento di 10000 delle sole classi minori con CW nei due diversi valori di alpha. . . .	135
61	Confronto sulle Matrici di Confusione della D.A. del 100% di tutte le classi su IDS-MLP con e senza CW nei due diversi valori di alpha. . .	136
62	Confronto sulle Matrici di Confusione della D.A. del 100% di Botnet e Bruteforce su IDS-MLP senza CW nei due diversi valori di alpha, in aggiunta alla figura 39.	137
63	Confronto sui Report di Classificazione della D.A. con aumento a 10000 samples delle classi Botnet e Bruteforce e del 100% sulle classi maggiori con e senza CW.	139
64	Confronto sulle Matrici di Confusione della D.A. del 100% sulle classi maggiori e di 10000 samples sui minori con e senza CW, nei due diversi valori di alpha.	140
65	Confronto D.A. del 100% delle classi maggiori e di 10000 samplese delle due classi minori nei due diversi valori di alpha.	141

Capitolo 1

Introduzione

Nel corso degli anni si è assistito non solo ad un aumento degli attacchi informatici ma anche ad un’evoluzione degli stessi attraverso l’Intelligenza Artificiale (AI). Secondo un report dell’ENISA – European Union Agency for Cybersecurity – la frequenza degli attacchi informatici è aumentata nel biennio 2020/2021 [1]. L’impiego delle nuove tecnologie come l’AI, le soluzioni Cloud ed un aumento proporzionale dell’interconnessione hanno comportato la nascita di problematiche per la Sicurezza Informatica in quanto gli attacchi informatici sono diventati sempre più sofisticati, complessi e con impatto maggiore per le vittime. In particolare, sono emerse otto specifiche minacce, tra le quali spiccano: Ransomware, Malware, Cryptojacking, Data Breach e DoS. L’AI, secondo l’ENISA, rimane uno strumento attraverso il quale gli hacker potranno migliorare gli attacchi informatici. L’arrivo dell’Internet of Things (IoT) e dell’industria 4.0 (IIoT) ha portato non solo vantaggi, ma ha anche aperto punti deboli. Infatti, si prospetta che entro il 2030 il campo della Sicurezza Informatica vedrà un’insufficiente preparazione e comprensione delle vulnerabilità dei sistemi IoT e della loro manutenzione. Un esempio riportato dagli studi evidenzia come gli attaccanti potranno sfruttare l’AI usando tecniche come le Generative Adversarial Networks (GAN) per ridurre notevolmente il rilevamento di attacchi informatici [2]. Questi ultimi verranno migliorati sfruttando il Deep Learning (DL) e, combinando insieme più tecniche, si potrà assistere alla realizzazione di attacchi mai visti, detti zero-day, ancor più efficaci. Tali tecnologie però vengono sfruttate anche da ricercatori

in tutto il mondo per contrastare di pari passo l’evoluzione delle minacce informatiche. Malintenzionati o hackers sviluppano, contemporaneamente ai ricercatori, nuove e migliori tecniche. Il DL è uno degli approcci maggiormente utilizzati negli ultimi anni per realizzare soluzioni per la Sicurezza Informatica. Tale approccio offre eccellenti performance spesso migliori del tradizionale Machine Learning (ML): uno dei vantaggi è la grande quantità di dati sfruttata per prendere decisioni. In un ambiente dinamico come quello della Sicurezza Informatica, che vede in gioco attaccanti con l’obiettivo di superare le attuali difese, è necessario che i sistemi di sicurezza sfruttino concetti di apprendimento automatico adatti alla dinamicità richiesta [3]. L’impiego degli Intrusion Detection System (IDS) risulta perciò uno dei mezzi necessari per contrastare azioni malevoli. L’integrazione dell’AI con gli IDS può essere un’ottima strada percorribile dai ricercatori per rendere le risposte più dinamiche e migliorare le performance. Infatti, si è potuto notare nell’ultimo decennio come le ricerche abbiano portato effettivamente a buoni risultati grazie alla loro implementazione sfruttando algoritmi di Machine Learning ed in seguito di Deep Learning. Rimangono però ancora problematiche relative all’utilizzo di dataset realistici per addestrare tali algoritmi: spesso tali dataset non sono aggiornati, sono molto sbilanciati o con dati non del tutto puliti e ciò richiede un elevato lavoro di preparazione degli stessi nonché l’utilizzo di tecniche per aumentarne le quantità. Le attuali ricerche suggeriscono soluzioni che sfruttano architetture basate su Cloud più centralizzate, dove i dati raccolti presso i dispositivi vengono trasmesse ai Server Cloud al fine dell’elaborazione: tale approccio però non sempre risulta essere il migliore perché mina la privacy dei dati [4]. Il concetto di privacy dei dati è stato recentemente affrontato da molti Paesi nel mondo: nell’Unione Europea è stato emanato il GDPR – General Data Protection Regulation – il Regolamento UE n. 2016/679 del 2016, successivamente aggiornato nel 2018, che espone delle linee guida per l’archiviazione e l’elaborazione dei dati personali. L’attenzione alla privacy è cresciuta ed ha influenzato anche la ricerca portando a ideare soluzioni che non espongano a rischi i dati degli utenti. Oltre al concetto di privacy dei dati, vi sono anche problematiche relative all’approccio basato sul cloud, ad esempio: la rilevante latenza di risposta nei sistemi real-time ed il costo legato al trasferimento e archiviazione di dati “grezzi” non anonimizzati o cifrati. Per superare

tali ostacoli si è iniziato a sfruttare l’edge computing che può effettivamente portare vantaggi attraverso l’elaborazione più vicina a quei dispositivi che producono effettivamente i dati coinvolti e, quindi, distribuita. Tale approccio può esser ancor più efficace se usato con una nuova tecnica di apprendimento e di “training” chiamata Federated Learning (FL), ideata da Google nel 2016 [5]. Il FL permette di avere maggior privacy e garantisce una gestione più efficiente delle comunicazioni. Per questi motivi risulta opportuno focalizzare l’attenzione sul miglioramento delle performance degli IDS rendendoli più efficaci, riducendone i falsi allarme, ottimizzando i tempi di risposta agli attacchi ed allo stesso tempo garantendo la privacy dei dati sfruttando le nuove tecnologie e nuovi approcci presenti allo stato dell’arte.

1.1 Organizzazione della tesi

La tesi è articolata come segue:

- il secondo capitolo affronta le tecniche di Intrusion Detection System e le tipologie esistenti;
- il terzo capitolo concentra l’attenzione sul Machine Learning e Deep Learning: i concetti teorici fondamentali, gli IDS che sfruttano ML e DL allo stato dell’arte e gli approcci architetturali centralizzati e distribuiti;
- nel quarto capitolo si descrive il Federated Learning in maniera esaustiva focalizzandosi sui vantaggi e le vulnerabilità associate ad esso;
- il quinto capitolo affronta le reti neurali Generative Adversarial Network: le tipologie, gli usi ed i benefici che introduce;
- nel sesto capitolo viene descritto il progetto principale di questa tesi, ovvero: l’analisi di tecniche di Network Intrusion Detection System basate su GAN e l’utilizzo della Data Augmentation;
- il settimo capitolo è dedicato alla conclusione ed agli sviluppi futuri in questo campo di ricerca.

Capitolo 2

Intrusion Detection System

Gli Intrusion Detection System (IDS) sono strumenti o applicazioni software che monitorano il sistema e la rete alla ricerca di attività malevole o anomalie e ne notificano la presenza; controllano sia le attività interne che esterne al sistema sotto esame [6]. Essi sono ormai ritenuti fondamentali nell'IoT, nelle architetture Cloud ed in generale per tutti quei sistemi considerati critici.

Emergono due concetti differenti dalla definizione sopra presentata: l'attacco e l'anomalia [7]. Un attacco può essere *interno*, quindi, un rischio dentro l'organizzazione che può essere sfruttato per causare danni oppure *esterno* in quanto coinvolge un attaccante all'esterno del perimetro del 'target', il quale tenta di ottenere l'accesso al sistema. Gli attaccanti, inoltre, possono esser ritenuti razionali o malevoli ed attivi o passivi: la prima distinzione riguarda l'avere o meno una specifica vulnerabilità da sfruttare in uno specifico punto della rete e la seconda, invece, distingue tra azioni diverse eseguite sul sistema e il semplice 'ascolto' della rete in attesa di lanciare un attacco sfruttando le informazioni carpite. Invece, l'anomalia si ha quando vi è un tentativo di bypassare i controlli di sicurezza ottenendo il controllo del sistema e si divide in: *anomalia di un punto* se un singolo dato devia dalla normalità; *anomalia contestuale* se il dato anomalo è inserito in uno specifico contesto e, infine, *anomalia collettiva* se un insieme di dati risultano anormali in confronto ad altri [7]. Gli IDS sono un valido strumento per analizzare e rilevare attività che non rispettano le politiche di sicurezza e scovare così anomalie che altri sistemi di sicurezza non troverebbero

come, ad esempio, i firewall. Gli IDS a loro volta possono essere classificati in attivi o passivi: la prima tipologia comprende gli IDS ed IPS che notificano la presenza di attività malevole, salvano i log degli eventi e “reagiscono” eseguendo, o consigliando all’Amministratore, comandi per configurare la rete al meglio con l’obiettivo di prevenire l’attacco; mentre l’altra tipologia notifica la minaccia ed effettua solamente il log degli eventi [8].

Nel corso degli anni gli IDS sono stati ottimizzati sfruttando le nuove tecnologie. Questo ha segnato una netta distinzione tra gli IDS convenzionali e quelli basati su Machine Learning ovvero più ‘intelligenti’ [8]: i primi sono principalmente basati sull’analisi della “firma” (teoricamente detta: ricerca della “hard-coded signature”) di attacchi conosciuti e richiedono aggiornamenti regolari del Database delle firme; i secondi basati invece su analisi statistiche, permettono il rilevamento di attacchi non conosciuti ma presentano un’elevata frequenza di falsi allarmi. Le principali categorie d’attacchi che si intende rilevare e che minano le tre proprietà fondamentali CIA (Confidenzialità, Integrità e Disponibilità) della sicurezza informatica sono [7]:

- DoS – Denial of Service: ha l’obiettivo di portare ‘offline’ componenti informatici inviando una elevata quantità di richieste fino alla saturazione;
- R2L – Remote to Local: sfrutta script automatici per indovinare la password dell’host sotto attacco ed ottenere il controllo della macchina;
- U2R – User to Root: ha l’obiettivo di ottenere il controllo non autorizzato di un account amministrativo e con i privilegi acquisiti sfruttare risorse critiche;
- Probe/Scan: utilizzato come attacco di ricognizione al fine di ottenere informazioni attraverso una scansione della rete.

I componenti coinvolti negli IDS sono: i *Sensori* per il monitoraggio delle attività e la raccolta dei dati; i *Server Database* che contengono tutti i log di incidenti ed attività malevole registrate dai sensori; la *Console*, o interfaccia utente, per permettere all’Amministratore di monitorare gli eventi ed eseguire azioni ed infine, opzionale, il *Server di Gestione* che prende decisioni a fronte di un allarme di sicurezza scatenato da un sensore [6], come illustrato in figura 1.

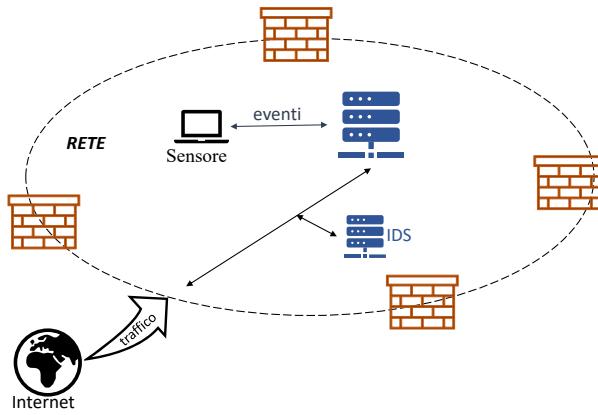


Figura 1: Struttura tipica di un IDS.

Inoltre, gli IDS possono essere classificati secondo diversi aspetti: architettura, conoscenza, tipo di attività monitorate, real-time o offline e tipo di rilevamento degli attacchi. Vengono distinti principalmente in Network-IDS e Host-based-IDS ed in rilevamento “Signature-based” o “Anomaly-based”, ovvero: basato su “firma” o sul comportamento anomalo nella rete.

2.1 Tipologie NIDS e HIDS

Gli Host-based-IDS sono distribuiti sui singoli host e monitorano le attività svolte, salvando i log degli eventi. Essi controllano il traffico della rete, i programmi ed i processi in esecuzione sull'host ed altre informazioni rilevanti per determinare attività sospette. Tale tipologia presenta degli svantaggi: sono in funzione solo se l'host è attivo e ciò non permette di rilevare costantemente dei possibili attacchi; comporta problematiche se vi sono host molto diversi tra loro che richiedono differenti configurazioni aumentando il rischio di avere conflitti con preesistenti sistemi di sicurezza; infine, si considerano tendenzialmente più facili da disabilitare durante un attacco.

I Network-IDS, invece, analizzano il traffico di rete catturato per trovare eventuali intrusioni ed effettuano il log degli eventi. I NIDS più recenti comprendono attività

di ‘anomaly detection’, analisi di protocolli stateful e ‘signature-based detection’ in modo da migliorare le performance. Inoltre, i NIDS possono svolgere il loro ruolo in *real-time* o *offline* analizzando i dati raccolti durante l’acquisizione del traffico. Il loro punto debole è la non efficacia in presenza di traffico elevato o se presenti informazioni cifrate. L’uso di algoritmi Machine Learning all’interno dei NIDS può migliorare l’accuratezza dei rilevamenti, tenendo in considerazione però la capacità computazionale dei dispositivi coinvolti per ottenere ottime performance ed evitare ritardi nella notifica degli allarmi.

2.2 Rilevamento “Signature-based” o “Anomaly-based”

Un’ulteriore distinzione viene fatta tra rilevamento “Signature-based” o “Anomaly-based” sia per HIDS che per NIDS, ma in questa tesi ci si focalizzerà sulla seconda variante. Il primo è un approccio ben noto e presente fin dai primi IDS commercializzati. Tale tipologia ha ottimo successo con attacchi noti, ma risulta inefficace in presenza di attacchi zero-day. Viene sfruttato un algoritmo di ricerca di stringhe che le confronti con un Database. Quest’ultimo, di conseguenza, dovrà essere aggiornato costantemente per poter rilevare anche nuovi attacchi.

La seconda tipologia, invece, permette di rilevare anche attacchi sconosciuti andando ad analizzare come il traffico della rete si discosti dalla normalità: ciò ovviamente risulta complesso da attuare in quanto è necessario definire un profilo dell’attività normale della rete e qualunque scostamento scatena perciò un allarme risultando purtroppo in un elevato tasso di falsi rilevamenti. Le principali tecniche usate negli IDS sono basate sulla statistica e su metodi di Machine Learning, su cui questa tesi si concentrerà. Vi sono anche versioni ibride di IDS che sfruttano entrambi le tipologie sopra descritte permettendo così di avere uno strumento completo che efficacemente rilevi attacchi noti tramite firme, ma che soprattutto sia in grado di osservare il traffico di rete alla ricerca di attacchi sconosciuti, con l’unico difetto di avere un costo computazionale più elevato.

Attualmente gli IDS basati su ML non sono così largamente diffusi e commercializzati e tipicamente sfruttano algoritmi, quali: Random Forest, Decision Tree, Algoritmi Evolutivi, SVM, ANN e DNN [7, 8]; essi vengono valutati in base all'accuratezza ed ai falsi allarmi scatenati sfruttando dataset contenenti dati di traffico normale e dati relativi ad attacchi. Essi si prestano come soluzione per la loro configurazione flessibile e facilmente adattabile; di contro però persistono allo stato dell'arte problemi legati alla necessità di dispositivi con una determinata potenza computazionale, alla mancanza di dati aggiornati per il training nonché a problemi di rallentamento della rete stessa e, come largamente descritto, elevata frequenza di falsi positivi [8].

Capitolo 3

ML e DL per la Sicurezza Informatica

Come anticipato nel capitolo precedente, gli algoritmi di Machine Learning (ML) vengono impiegati per migliorare e rendere più efficaci, ad esempio, gli Intrusion Detection System e i filtri antispam. Inoltre possono essere usati per rafforzare le politiche di sicurezza adottate nelle aziende e/o organizzazioni: usare l'AI anziché definire le politiche manualmente o sfruttando automatismi può ridurre le ridondanze e migliorare la protezione offerta. La ricerca nel campo della sicurezza informatica che sfrutta il ML si concentra su NIDS e HIDS; analisi e classificazione di malware; analisi di vulnerabilità; analisi delle supply chains e identificazione di Botnet [8]. I sistemi basati su ML hanno portato vantaggi come la riduzione del carico di lavoro degli esperti in sicurezza ed aumentato l'efficienza e l'efficacia di queste tecniche; spicca tra queste la maggior flessibilità e velocità nel risolvere problemi di sicurezza molto complessi. Di contro, però, hanno comportato la crescita dei falsi positivi: l'accuracy dimostrata nelle varie ricerche non risulterebbe sufficientemente affidabile per l'applicazione in un contesto reale ma, nonostante ciò, rimane un valido approccio date le ottime possibilità che si prospettano.

Si ritiene, perciò, necessario approfondire alcuni concetti fondamentali per poter sfruttare questa tecnologia al meglio. Innanzitutto, differenziando tra AI, ML e DL e proseguendo con il definire le tipologie di apprendimento, le reti neurali ed un'analisi

dello stato dell'arte dei Machine Learning-IDS e Deep Learning-IDS.

3.1 Concetti fondamentali

La definizione di Intelligenza Artificiale (in inglese, Artificial Intelligence - AI) non è unica e prende diversi significati in base al contesto in cui viene utilizzata. Secondo l'organizzazione per la standardizzazione ISO – International Organization for Standardization – la definizione di AI (ISO/IEC 2382:2015 [9]) può essere:

1. branca dell'informatica dedicata allo sviluppo di sistemi di elaborazione dati che svolgono funzioni normalmente associate all'intelligenza umana, come ragionare, apprendere e migliorare autonomamente;
2. campo interdisciplinare, di solito considerato una branca dell'informatica, che si occupa di modelli e sistemi per l'esecuzione di funzioni generalmente associate all'intelligenza umana, come il ragionamento e l'apprendimento.

Si può, quindi, considerare il Machine Learning come un ramo dell'AI, “*che si occupa dello sviluppo di algoritmi e tecniche finalizzate all'apprendimento automatico mediante la statistica computazionale e l'ottimizzazione matematica*” [10]. Inoltre, è emerso successivamente il Deep Learning ed è stato considerato come un nuovo campo all'interno del vasto settore del ML (si veda Figura 2) [11]. Il DL sostanzialmente sfrutta le reti neurali per apprendere: quest'ultime composte da neuroni artificiali che simulano il cervello dell'essere umano.

In generale un sistema AI ha uno specifico ciclo di vita e, in particolare, la fase di sviluppo comprende degli step fondamentali [7, 8, 12]:

1. *Data Collection*: nella quale vengono generati o raccolti tutti i dati necessari creando dataset diversi in base alle necessità;
2. *Pre-processing*: fase importante che permette di pulire, strutturare e rifinire i dati al fine di evitare che l'apprendimento automatico si dilunghi a causa di

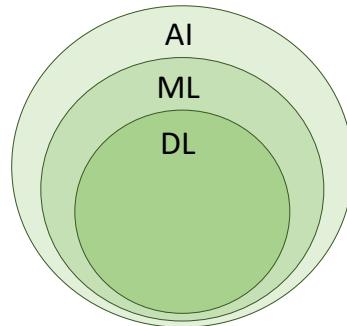


Figura 2: Relazione tra AI, ML e DL.

dati “grezzi” e ridondanti. Le tecniche utilizzate sono principalmente: normalizzazione, pulizia di ridondanze o valori incorretti, riduzione della dimensione dei dataset e molte altre;

3. *Model Training*: nella quale il modello scelto viene addestrato sfruttando i dati raccolti ed ha un elevato costo computazionale;
4. *Testing*: il modello viene quindi testato attraverso un dataset opportunamente scelto, diverso dagli altri, al fine di vedere concretamente come si comporta il modello ad esempio nel classificare il traffico di rete in benigno o maligno;
5. *Validazione*: vengono visualizzati i risultati del testing ed eventualmente apportate modifiche per ottimizzare il funzionamento del modello.

3.1.1 Tipologie di apprendimento

Tutto ruota intorno al concetto di apprendimento da parte delle macchine. Tali tecniche prendono spunto da come l’essere umano impara, ovvero: arricchisce la propria conoscenza innanzitutto memorizzando eventi e casi diversi e, successivamente, applica una generalizzazione considerando la similitudine di una situazione con un caso incontrato. Vi sono diversi approcci di apprendimento: supervisionato, non supervisionato, semi-supervisionato e il cosiddetto “Reinforcement learning”, tradotto dall’inglese in “per rinforzo” o “rinforzato”.

Supervisionato

L'apprendimento supervisionato è la tipologia più usata in ML. Fornisce in ingresso (X, Y) al modello, dove i dati di input sono X ed i dati di output desiderati Y considerati come etichette. Per capire meglio questo concetto basti pensare ad un insieme X di foto di animali ed un insieme Y di etichette delle foto che classificano le tipologie di animali (cane, gatto, ...). Sostanzialmente, l'obiettivo è quello di estrarre una regola generale che associa l'input all'output corretto. Gli algoritmi che ricadono in questa categoria sono prevalentemente associati a *problemi di regressione* nel quale si cerca di predire un valore continuo o *problemi di classificazione* dove, invece, l'obiettivo è quello di distinguere un valore discreto in un insieme limitato di possibilità [7, 8, 12]. Focalizzandoci sui NIDS, oggetto di questa tesi, si può pensare a dati relativi al traffico di rete come input ed alla loro classificazione in “normale” e “anormale” oppure ad una classificazione in base al tipo di attacco.

Non Supervisionato

La tipologia non supervisionata invece riceve solamente dati di input X e durante la fase di addestramento non sfrutta le classi o le etichette associate agli input x , ma investiga le relazioni presenti tra attributi. Di conseguenza, per poter prendere decisioni con adeguata accuratezza è necessario avere un dataset di elevate dimensioni. È anche utilizzato con ottimi risultati nell'anomaly detection per i NIDS in quanto rilevano più facilmente gli attacchi zero-day ed in generale nuovi pattern di comportamento malevolo. I principali usi riguardano il raggruppamento dei dati per *associazione* o “*clustering*” andando a mettere nello stesso gruppo quei dati che presentano similitudini.

Semi-Supervisionato

Questa tipologia è una versione ibrida che unisce i due concetti poc'anzi introdotti. Vi sono quindi dati di input X e dati output etichettati Y , ma non tutti gli input $x \in X$ sono associati ad una etichetta y . Si usa per problemi di classificazione e

clustering. Si reputa possa essere una buona soluzione da sfruttare in future ricerche nonostante la sua maggior complessità.

Reinforcement Learning

L’ apprendimento rinforzato permette al modello di prendere decisioni sfruttando un meccanismo basato sull’accumulare ricompense: il modello apprende autonomamente, attraverso tentativi e commettendo errori, a svolgere la funzione assegnata e riceve una “ricompensa” ogni qualvolta un’azione è correttamente svolta oppure viene penalizzato se sbaglia (si può considerare come se imparasse dall’esperienza). I dati vengono raccolti dall’ambiente, il quale viene ulteriormente analizzato in cerca di una soluzione con l’obiettivo di adattarsi a cambiamenti dell’ambiente stesso.

Vantaggi e svantaggi

Di seguito vengono riassunti i vantaggi (✓) e svantaggi (✗) principali di ogni tipologia di apprendimento [6, 7, 8]:

- Supervisionato

- ✓ Efficace per applicazioni basate su un’attività specifica
- ✓ Necessita di meno dati
- ✓ Più controllo sui risultati
- ✓ Accurato e affidabile
- ✗ Problemi di over-fitting
- ✗ Richiede tempo per preparare i dati
- ✗ Necessita di dataset con output diviso in categorie

- Non supervisionato

- ✓ Non è necessario che il dataset sia diviso in categorie
- ✓ Adatto per trovare nuove relazioni tra i dati

- ✓ Lavora autonomamente
 - ✗ Richiede più dati del supervisionato
 - ✗ Non è scalabile
 - ✗ Prono ad errori
- Semi-supervisionato
 - ✓ Facile implementazione
 - ✓ Veloce nella risoluzione
 - ✓ Necessita di meno dati per l'allenamento
 - ✗ Meno accurato
 - ✗ Richiede un dataset bilanciato nelle sue categorie
 - ✗ Aumenta il rumore nei dati
- Rinforzato
 - ✓ Veloce nel prendere decisioni in ambienti realistici
 - ✓ Costo minore relativo alla generazione dei dati
 - ✗ Vulnerabile ad attacchi
 - ✗ Difficile da applicare per la sicurezza informatica
 - ✗ I dati possono essere facilmente corrotti

3.1.2 Il Deep Learning e le Reti Neurali

Come già anticipato, il DL è una sottocategoria del ML che sfrutta algoritmi ispirati al cervello umano, ovvero le Reti Neurali Artificiali. Queste ultime sono composte da vari livelli di unità a cascata che eseguono estrazione di caratteristiche e trasformazione. La loro applicazione comprende la classificazione e l'analisi di pattern.

Le reti neurali vengono impiegate in quanto utili a processare grandi quantità di dati grazie al fatto che lavorano in parallelo. I difetti riscontrati riguardano la difficoltà nell'interpretare il ragionamento che le ha portate al risultato; la complessità nel trattare variabili categoriche e, infine, la necessità di un tempo elevato per l'addestramento qualora vi siano moltissime variabili da analizzare. Il vantaggio principale, invece, è quello di svolgere autonomamente l'estrazione delle cosiddette “feature” (ovvero caratteristiche) rispetto al tradizionale ML che necessita l'estrazione manuale. Conoscendo il ciclo di vita di ogni sistema AI (Data Collection; Pre-processing; Model Training; Testing e Validazione) si può affermare che le reti neurali, quindi, permettono di avere un Pre-processing più snello.

In generale, allenare una rete neurale significa sfruttare un certo insieme di esempi (il dataset di training) di cui si conosce già l'esito della classificazione e, attraverso tentativi ed errori, impostare una serie di parametri al fine di avere una rete neurale che classifichi in autonomia esempi mai visti prima. Ciò significa che deve essere in grado di generalizzare il più possibile. I parametri da definire sono diversi tra cui: i pesi dei neuroni e gli iperparametri. Inoltre, le reti neurali hanno una loro funzione da calcolare che rappresenta l'approssimazione di un problema o funzione non nota a priori che si cerca di trovare: la si può vedere come una funzione obiettivo che rappresenta, ad esempio, la classificazione di oggetti.

Per capire come impostare tali valori ed effettuare un ottimo training è necessario descrivere com'è fatta una rete neurale: il mattone alla base della rete neurale è il *percettrone* che è la versione matematica di un neurone umano. Esso esegue una classificazione binaria mappando un vettore in ingresso in un valore in output considerando un peso. Ogni nodo di una rete neurale è un 'neurone' a cui è associata una funzione di attivazione quale ad esempio: la funzione ReLu - *Rectified Linear Unit*; la funzione Sigmoidea; la Softmax usata spesso per problemi di classificazione ed altre varianti. In particolare, l'output della funzione di attivazione del neurone applicata alla somma pesata del vettore in input viene propagata al neurone successivo.

I pesi della rete possono essere modificati al fine di ottenere una buona rete neurale che lavori correttamente. Per ottenere ciò è necessario definire i pesi in modo tale che la rete classifichi correttamente il maggior numero di esempi conoscendo l'input: ciò

si ottiene cercando una approssimazione adatta della probabilità a posteriori per la funzione calcolata dalla rete neurale giungendo così, attraverso calcoli, a trovare una cosiddetta funzione di errore o *loss function* da minimizzare. Tale loss function è una funzione che aiuta a capire se i pesi sono stati impostati correttamente.

Trovare il minimo della loss function in reti neurali molto grandi è complesso, di conseguenza si sfruttano tecniche matematiche come la *discesa del gradiente* per trovare i minimi: essa si basa sul calcolo del gradiente di una funzione f (∇f) che rappresenta il verso di crescita della funzione stessa. Ne consegue che il metodo di discesa del gradiente permette di determinare iterativamente un punto di minimo di f , partendo da una soluzione iniziale e proseguendo di un certo “passo di discesa”; quest’ultimo incide sulla velocità con cui convergerà l’algoritmo. Un modo per calcolare i gradienti è quello di usare il metodo della *Back Propagation* che permette contemporaneamente di aggiornare i pesi dei neuroni a ritroso. Tale metodo può causare però il problema della “*Scomparsa del Gradiente*” che crea un’elevata difficoltà nell’addestramento. È perciò necessario capire come far coincidere più impostazioni ottenendo una rete neurale che funzioni.

Oltre ai pesi, vi sono anche gli *iperparametri* da considerare in quanto incidono sulle performance della rete stessa. Vi sono molti parametri che si possono definire:

- il numero dei neuroni e la disposizione negli strati;
- il numero delle epoche (in inglese “epoch”): un’epoca si ha quando l’intero dataset di training viene elaborato dalla rete neurale;
- le funzioni di attivazione dei nodi;
- la loss function;
- la dimensione del batch, ovvero: il numero di esempi presenti per ogni sottinsieme ottenuto dalla divisione dell’intero dataset quando troppo ampio;
- l’algoritmo di discesa del gradiente ed il suo “learning rate”;
- la dimensione del dataset per la validazione;

e molti altri a seconda della complessità della rete.

Per capire se la rete neurale lavora correttamente si consiglia di usare *tecniche di Cross Validation*: essa permette di monitorare la loss function e determinare la bontà del modello in base agli iperparametri fissati; ed eventualmente modificarli in base all'esito della validazione. Vi sono, inoltre, una serie di accorgimenti per ottimizzare l'apprendimento quali: Pre-processing dei dati al fine di normalizzarli; la regolarizzazione della loss function; inizializzare opportunamente i pesi prima dell'allenamento; la Data Augmentation per ampliare la quantità di dati a disposizione e molti altri.

Problemi correlati

Vi sono però alcuni problemi da considerare quando si sviluppano modelli Machine Learning e Deep Learning che possono dare false aspettative sui risultati:

- L'*Over-fitting* (in italiano: adattamento eccessivo) si ha quando il modello “impara a memoria” senza essere in grado di generalizzare, causando molti errori durante la validazione in presenza di un dataset mai visto;
- L'*Under-fitting* si presenta con errori elevati sia in fase di Training che di Validazione causati dalla complessità dei dati.

Le soluzioni da adottare riguardano la gestione stessa dei dati o dell'addestramento, ad esempio, effettuando un partizionamento dei dataset sfruttando sempre le *tecniche di Cross Validation* per accorgersi della presenza di over-fitting o arrestando anzitempo il processo d'addestramento. Vi sono diverse implementazioni di Cross Validation: Holdout; K-fold e Leave-one-out [11]. Tale accorgimento permette di rimuovere dal dataset di addestramento alcuni dati e presentarli solo in validazione.

3.1.3 Il meccanismo di Self-attention

Il meccanismo di self-attention è stato introdotto da Vaswani et al. [13] nel 2017 ed è sempre più utilizzato nel campo del NLP - Natural Language Processing; sono però stati fatti tentativi anche in contesti nei quali vengono sfruttati modelli deep. Tale meccanismo permette al modello di dare un certo livello di importanza a diverse parti

di una sequenza in input e dinamicamente calibrare la loro influenza sull'output. È evidente perciò come sia un'ottima soluzione per attività di "language processing" nelle quali il significato di una parola cambia in base al contesto. Esistono diversi tipi di meccanismi di self-attention realizzati con lo scopo di aumentare l'efficienza della versione originale, nonostante quest'ultima mantenga comunque una notevole accuratezza [14]. La sua applicazione permette perciò di aumentare l'accuratezza a discapito di una più lenta convergenza.

Innanzitutto, una funzione di "Attenzione" (in inglese 'Attention function') è descritta come un mappaggio di tre componenti in input Q , K , V in un ouput. Le componenti di input sono matrici: Q è una matrice di interrogazioni (di seguito "query"); K è una matrice di chiavi ottenute trasformando le query ed, invece, V è di valori. Esistono due principali funzioni di attenzione:

- *Scaled dot product attention* (SDPA): funzione che fornisce in output una matrice C di contesto che contiene i pesi dei valori della matrice V ;
- *Multi-Head Attention* (MHA) esegue in parallelo più SDPA e concatena gli ouput; essa sfrutta proiezioni lineari degli input in quanto permette di estrarre più efficientemente relazioni tra i dati.

Tale meccanismo di self-attention assume che gli input provengano tutti da una stessa fonte e, di conseguenza, ciò permette di catturare informazioni di contesto importanti e le possibili dipendenze tra i dati; il tutto in maniera parallela. [13, 15].

Sebbene sia prevalentemente usato per l'NLP, vi sono altri contesti in cui viene applicato come, ad esempio, la Sicurezza Informatica: CS-FL [16] è una proposta di IDS basato su Federated Learning che sfrutta un modello ibrido composto da Self-Attention ed una CNN ed, invece, altri ricercatori propongono l'uso del meccanismo all'interno di una rete WGAN per migliorare le prestazioni degli IDS [15].

Infine, è stato anche sfruttato per realizzare una tipologia di rete GAN particolare detta SAGAN ("Self-Attention Generative Adversarial Networks") [17].

3.2 ML-IDS e DL-IDS allo stato dell'arte

Il Machine Learning ed il Deep Learning vengono impiegati anche per realizzare Intrusion Detection System. Allo stato dell'arte vi sono moltissime ricerche che trattano di questi argomenti e che hanno evidenziato notevoli progressi e miglioramenti. In particolare, per quanto riguarda il ML, gli algoritmi più utilizzati per realizzare gli IDS sono [18, 19]:

- le *Support Vector Machine* (SVM) utilizzata per la sua elevata accuratezza nel classificare dataset complessi, ma rende poco affidabili gli IDS qualora vi siano dati “rumorosi”;
- i *Decision Tree* (DT) vengono usati per fare previsioni partendo da delle osservazioni; adatti per gestire grandi quantità di dati sia continui che categorici, però risultano essere soggetti ad over-fitting ed avere costo computazionale elevato;
- le *Random Forest* (RF) sono un insieme di DT usati negli IDS per gestire dati rumorosi e per individuare anomalie in grandi dataset: purtroppo però la loro lentezza può essere un grande svantaggio;
- l'algoritmo *K-nearest neighbors* (KNN) è diffuso in questo contesto applicativo soprattutto per la sua efficienza e velocità che gli permette di essere usato in real-time ed è un'ottima soluzione a basso impatto computazionale e temporale; ciò nonostante, l'accuratezza si basa completamente sulla qualità dei dati correndo così il rischio di peggiorare la classificazione a causa di dati non adeguati;
- le *Artificial Neural Network* (ANN) sono un valido aiuto per la loro capacità di rilevare relazioni tra i dati, ma hanno un costo computazionale elevato e non è possibile capire il loro processo decisionale: un esempio di ANN è la rete Feed-forward neural network.

I DT e le RF, confrontati con gli altri algoritmi, risultano essere le migliori in termini di accuratezza in seguito ad una valutazione sia per classificazione binaria che multi-classe. Inoltre i DT hanno il più basso costo computazionale tra le tipologie

indicate [18]. Secondo uno studio, nel quale è stato eseguito un attacco sfruttando tecniche di ML per evadere il rilevamento, le SVM e i DT non dovrebbero essere impiegati come classificatori nei NIDS in quanto troppo vulnerabili ad attacchi basati su dati malevoli generati da algoritmi evolutivi [20]. Un'altra ricerca ha evidenziato come gli attacchi basati su reti GAN possano ridurre notevolmente l'efficacia degli IDS: Mari et al. [21] hanno studiato il comportamento di RF, KNN e ANN sotto attacco e consigliano di eseguire il training sfruttando non solo i dataset, ma anche dati generati da reti GAN per aumentare le prestazioni e prevenire attacchi di questo tipo.

Un ulteriore passo verso il miglioramento è stato raggiunto con l'uso di algoritmi Deep Learning grazie alle loro capacità di gestire grossi e complessi problemi e ciò fa presumere che potrà essere un ottimo punto di partenza per futuri sviluppi. Infatti si è arrivati ad avere moltissimi avanzamenti in letteratura. I modelli principalmente utilizzati sono [22]:

- *Convolutional Neural Network* (CNN) usata per la sua velocità nel training;
- *Recurrent Neural Network* (RNN) nonostante soffra del problema del vanishing gradient; poi superate con l'uso dell'architettura *Long Short-Term Memory* (LSTM) che garantisce buone prestazioni in contesti dove i dati sono in relazione tra loro in base al tempo;
- *Autoencoder* (AE) e le varianti *AdversarialAE*, *VariationalAE* e molte altre: sono composti da due parti dette encoder e decoder, ognuna delle quali implementata con delle reti neurali e sono impiegate per le loro ottime capacità di generazione;
- *Generative Adversarial Network* (GAN), affrontate in maniera completa nel quarto capitolo, usate soprattutto per la Data Augmentation e per eseguire attacchi.

Oltre a queste reti neurali sono state impiegate in letteratura moltissime combinazioni di altre tecniche Deep Learning come anche le DNN o le RNN-GRU - *Gated recurrent units*. Alcuni esempi di DL-IDS possono essere, ad esempio, quelli studiati

da Abdalgawad et al. [23] nel quale si evince come l'uso di metodi deep come AAE e BiGAN insieme a KNN possa aumentare l'accuratezza. Vi sono anche approcci ibridi come DRL-GAN [24] nel quale vengono sfruttati sia Deep Learning che Reinforcement Learning per realizzare un NIDS che risolve alcune problematiche (discusse nei capitoli successivi) utilizzando reti GAN. Altre soluzioni riguardano la realizzazione di IDS sfruttando l'edge-computing: gli autori de Araujo-Filho et al. [15] hanno studiato un IDS realizzato usando una TCN - *Temporal Convolutional Network* - ed una WGAN che integra un blocco self-attention; è emerso che tale soluzione rileva gli attacchi zero-day in maniera molto efficace con un tasso di rilevazione a 0.9993 se realizzato con TCN. Inoltre nella survey di Macas et al. [22] si può constatare come allo stato dell'arte vi siano diverse architetture per NIDS: centralizzata, decentralizzata, distribuita e federata che sfrutta l'edge-computing. In queste soluzioni vengono impiegate SAE, RBM, DBN e molte altre.

3.3 Approccio centralizzato e distribuito

Come già brevemente citato in alcuni esempi nel capitolo precedente, esistono soluzioni allo stato dell'arte che sfruttano approcci centralizzati o distribuiti.

In un'architettura centralizzata il server mantiene tutti i dati raccolti dai client ed effettua la fase di addestramento. In questo modo l'intero dataset, ovvero l'insieme di tutti i dati, viene elaborato direttamente dal server stesso, o da un insieme di server, qualora servissero per ridurre il tempo di training. Questo approccio espone i dati grezzi sul server e, perciò, è consigliabile che il proprietario del server sia lo stesso di quello dei dati. Qualora invece fosse un'entità autorizzata alla raccolta ed elaborazione dei dati emergono una serie di scenari possibili relativi all'affidabilità e fiducia che si ha della stessa, avendo un aumento del rischio legato alla privacy; non è inusuale che in queste situazioni si considerino le entità centralizzate come oneste oppure oneste-ma-curiose (dette “honest-but-curious” in inglese). La versione centralizzata con cluster di server ha comunque accesso all'intero dataset e perciò presenta gli stessi svantaggi della versione con unico centro di elaborazione. Emergono così evidenti alcune lacune quali: l'avere il problema del cosiddetto SPoF – Single point-of-failure –

nel quale il server rappresenta un singolo punto vulnerabile nell’architettura; problemi di privacy dei dati e la loro condivisione tra più server e problemi legati all’aumento della quantità di dati da elaborare ed al trasferimento degli stessi che potrebbe comportare una riduzione delle performance dell’entità centrale, in particolare: un aumento della latenza e della banda di trasmissione usufruita.

L’approccio distribuito, invece, nel contesto del ML permette ai dispositivi partecipanti di creare un modello globale sul server centrale addestrandone uno locale sfruttando il proprio dataset privato. Il server ha il ruolo di unire i modelli locali creandone uno globale unico e coordina l’operato dei partecipanti. Ne consegue un aumento del livello di privacy in quanto il server non accede direttamente ai dati locali di ogni client e vi è un miglioramento delle performance grazie alla riduzione dell’overhead nella comunicazione. Tra gli approcci decentralizzati rientra una tipologia chiamata Federated Learning (FL) che sfrutta l’edge-computing: essa aumenta ancor di più il livello di privacy dei dati privati dei partecipanti inviando solo i parametri del modello. Un FL-NIDS risulterebbe così meno vulnerabile ad un attacco diretto al server di aggregazione in quanto persisterebbe comunque un’analisi a livello locale sui singoli client.

Capitolo 4

Federated Learning

Il FL è un approccio machine learning collaborativo, ideato da Google nel 2016 [5, 25], che ha portato notevoli benefici soprattutto legati alla privacy dei dati, ormai oggetto di discussione in tutto il mondo. Un esempio ben noto dell'applicazione del FL è GBoard [26], la tastiera sviluppata da Google [4, 27].

4.1 Caratteristiche

Il Federated Learning permette di mantenere i dati di training sui client (chiamati anche partecipanti) ed eseguire la fase di addestramento (o ‘training’) collaborando: ciò significa che i dispositivi che partecipano hanno un loro dataset locale ed un loro modello locale. Ogni partecipante computa un aggiornamento locale (o ‘update’) e lo invia ad un server centrale che funge da coordinatore. Gli ‘update’ locali del modello vengono così aggregati sul server costituendo un aggiornamento per il modello globale che viene poi comunicato ai partecipanti; da qui il processo continua per una serie di round fino alla conclusione del training [5]. Di conseguenza, si ha la possibilità di eseguire algoritmi di ML in maniera collaborativa senza condividere dati privati dei partecipanti ad un server centrale di aggregazione; per questi motivi è ritenuto un valido approccio per la protezione della privacy.

Questo approccio collaborativo può essere principalmente classificato in: *orizzontale* se tutti i partecipanti condividono la stessa struttura di dati (stesso spazio delle

feature e diversi sample) ed in *verticale* se, invece, lo spazio delle feature è diverso ma sugli stessi sample: ciò significa che i partecipanti hanno differenti informazioni su stessi dati. Si può intuire la differenza con un esempio: nel caso della tipologia verticale si hanno “due organizzazioni che lavorano nella stessa città possono avere gli stessi clienti ma detenere diverse informazioni su questi ultimi” [4].

Inoltre è importante discutere come avviene la trasmissione dei dati e la loro elaborazione: il MultiAccess Edge Computing (MEC) e la Secure MultiParty Computation (SMC) sono le tecniche più utilizzate nelle soluzioni che applicano il Federated Learning. Il MEC permette di eseguire il lavoro più vicino ai dispositivi come gli smartphone, tablet ed in generale dell’IoT ed aiuta nel ridurre le congestioni della rete e la latenza nelle trasmissioni dei dati; purtroppo, però, se usato in contesti in cui il training è elevato e continuo risulta avere elevati costi di comunicazione nonché presenta problemi di privacy dei dati causata dalla trasmissione degli stessi e la loro conservazione. La SMC è un protocollo crittografico che protegge da Server di Aggregazione onesti-ma-curiosi: permette di eseguire una computazione distribuita tra più nodi che collaborano tra loro, senza però accedere ai reciproci dati privati; ad esempio, per il FL è stata utilizzata la *crittazione omomorfica* [28] che richiede però elevata capacità computazionale.

Prima di affrontare i principali algoritmi usati nel FL è necessario approfondire alcuni concetti fondamentali sul suo funzionamento: esso coinvolge un insieme di partecipanti $N = \{1, \dots, n\}$ con i loro dataset privati $D_n, n \in N$ e il loro modello locale w_n^t , ed un *Server di Aggregazione* che conserva un *Modello Globale* w_G^t , dove t è il round di aggregazione considerato durante il training [4, 5]. Per ogni round di aggregazione viene scelto un sottoinsieme di partecipanti $S^t \in N$ a cui il server invia lo stato del modello globale in quel momento. Viene scelto solo un gruppo di partecipanti per motivi di efficienza in quanto vi è un degrado delle prestazioni se il numero di client è al di sopra di una specifica soglia [5]. Successivamente, ogni partecipante esegue τ computazioni locali basate sul modello globale ricevuto ed il suo dataset locale ed invia i parametri del modello locale al server di aggregazione. Dopo che il server ha eseguito l’aggregazione globale invia il modello w_G^t ai partecipanti ed il processo ricomincia col round $t + 1$. Tutto questo viene eseguito assumendo che i

partecipanti siano “onesti” e, quindi, non intendano “avvelenare” i dati o il modello stesso. Possiamo riassumere il training in quattro passaggi fondamentali:

1. Inizializzazione
2. Aggiornamento Locale
3. Selezione dei partecipanti
4. Aggregazione Globale

Nella fase di Inizializzazione il server crea il modello globale w_G^0 impostandolo con parametri casuali da una distribuzione normale e specifica gli iperparametri: numero di update locali (τ), tasso di apprendimento (η) e dimensione del mini-batch (B). Le successive tre fasi rientrano in un singolo round di aggregazione.

La fase di Aggiornamento locale esegue, per ogni client in S^t , l’aggiornamento dei parametri del modello locale per (τ) iterazioni ottenendo w_n^t con l’obiettivo di definire dei parametri ottimizzati che minimizzano la funzione di errore locale – *local loss function* – $F_n = (w_n^t)$.

La successiva fase di Selezione dei partecipanti permette al server di scegliere casualmente un sottoinsieme $S^t \in N$ di client che invieranno al server i loro parametri di aggiornamento per l’aggregazione globale.

Infine, l’ultima fase di Aggregazione Globale consente al server di aggregare i parametri ricevuti, aggiornare il modello globale w_G^{t+1} ed inviarlo ai partecipanti per il round successivo. Anche il server ha l’obiettivo di minimizzare la funzione di errore globale, in questo caso $F_n = (w_G^t)$.

Le tre fasi principali (2, 3, 4) vengono ripetute fino alla convergenza della ‘*global loss function*’ o fino a quando l’algoritmo non incontra una condizione di terminazione.

4.2 Algoritmi progettati allo stato dell’arte

Le fasi principali coinvolte nel funzionamento del FL illustrate precedentemente vengono implementate con l’algoritmo Federated Averaging (*FedAvg*) che è stato il primo ad essere ideato ed è il più utilizzato, nato in seguito a *FedSGD* dagli stessi ideatori

del FL e del FedAvg. FedSGD sfrutta la funzione di ottimizzazione SGD ma consuma in maniera elevata la banda di trasmissione richiedendo maggior capacità computazionale a causa dell'aggregazione del gradiente ad ogni epoch. Vi sono però altri algoritmi che sono stati realizzati nel corso degli anni per migliorare le performance e risolvere alcuni problemi emersi.

4.2.1 FedAvg

È necessario, però, approfondire l'algoritmo FedAvg ed il suo funzionamento al fine di poterlo confrontare con le altre alternative. Tale algoritmo è il primo realizzato e presenta problematiche relative alla convergenza in contesti in cui i dati sono eterogenei [29]. È stato studiato un ulteriore algoritmo chiamato FedProx che garantisce convergenza, modificando leggermente FedAvg. In particolare, dall'algoritmo 1 si può notare che: il server inizia il training (righe 1-6) e solo i partecipanti scelti iniziano l'addestramento locale (9-16) andando ad ottimizzare la loss function sui mini-batches del dataset locale. Il server all'iterazione t , quindi, punta a minimizzare l'errore totale aggregando i gradienti ricevuti dai partecipanti (riga 7). Il tutto continua fino al raggiungimento di un numero massimo di round di aggregazione oppure quando la loss function globale raggiunge una errore minimo desiderato.

4.2.2 FedProx

L'algoritmo FedProx può esser visto come una versione generalizzata di FedAvg [30]. Quest'ultimo è difficile da approfondire teoricamente e non garantisce la convergenza in scenari in cui vi è eterogeneità dei dispositivi nella rete che a loro volta generano (o catturano, se consideriamo i NIDS) dati localmente: ci si pone dunque in una situazione nella quale i dati non sono identicamente distribuiti (detti, in inglese, Non-IID data) tra tutti i partecipanti coinvolti. Ciò permette così di avere una garanzia di convergenza in uno scenario federato realistico dove i dispositivi coinvolti sono eterogenei, dimostrata teoricamente e attraverso sperimentazione [30]. In particolare, dall'algoritmo 2 si può notare che possiede gli stessi passaggi del server di FedAvg: seleziona un sottoinsieme di partecipanti ad ogni round, fa eseguire il training locale

Algorithm 1 Federated Averaging (FedAvg) [4, 5]**Input:** B, τ, η , numero di round di aggregazione T **Output:** modello globale w_G

[**Server:** calcola la media globale pesata usando i parametri dei modelli locali dei partecipanti selezionati]

- 1: Inizializzazione w_G^0
 - 2: **for** $t = 1, T$ **do**
 - 3: Seleziona casualmente un sottoinsieme di client $S^t \in N$
 - 4: **for all** $n \in S^t$ **do**
 - 5: $w_n^{t+1} \leftarrow ClientUpdate(n, w_G^t)$
 - 6: **end for**
 - 7: $w_G^{t+1} = \sum_{n=1}^{S^t} \frac{D_n}{D} w_n^{t+1}$
 - 8: **end for**
 - [**Client:** aggiornamento del modello locale]
 - 9: **procedure** CLIENTUPDATE(n, w_G^t)
 - 10: $B_n \leftarrow$ Divide il dataset locale D_n in piccoli insiemi di dimensione B
 - 11: **for all** epoch locale, τ **do**
 - 12: **for all** batch $b \in B_n$ **do**
 - 13: $w_n^{t+1} \leftarrow w_n^t - \eta \nabla F_n(w_n^t; b)$
 - 14: **end for**
 - 15: **end for**
 - 16: **return:** w_n^t
 - 17: **end procedure**
-

Algorithm 2 FedProx [30]**Input:** $N = \{1, \dots, n\}, T, w_g^0, p_n, \mu, \gamma$

- 1: **for** $t = 0, T - 1$ **do**
 - 2: **[Server]**
 - 3: Sceglie un sottoinsieme di client $S^t \in N$, ogni client scelto con probabilità p_n
 - 4: Invia ad ogni partecipante estratto w_G^t
 - 5: **[Client]**
 - 6: Ogni client scelto $n \in S^t$ cerca un w_n^{t+1} che sia un minimizzatore γ_n^t -inesatto
di: $w_n^{t+1} \approx argmin_w h_n(w; w^t) = F_n(w) + \frac{\mu}{2} \|w - w^t\|^2$
 - 7: Ogni client scelto $n \in S^t$ invia w_n^{t+1} al server
 - 8: **[Server]**
 - 9: Aggrega ciò che ha ottenuto come $w_G^{t+1} = \frac{1}{|S^t|} \sum_{n \in S^t} w_n^{t+1}$
 - 10: **end for**
-

e gli aggiornamenti locali li aggreda per ottenere un modello globale aggiornato. La parte diversa riguarda il client (riga 4) nel quale vengono aggiunti due dettagli:

1. Tolleranza al lavoro incostante dei dispositivi eterogenei: è poco realistico forzare tutti i client a lavorare uniformemente considerando le differenze di connessioni, capacità computazionali e batterie; perciò è stato permesso un lavoro disomogeneo definendo γ_n^t come la quantità di computazione locale del device n al round t ;
2. Controllo degli aggiornamenti locali usando un “Proximal term”: contemporaneamente, si cerca di limitare la variabilità degli aggiornamenti locali causati dall’eterogeneità dei dati stessi in quanto potrebbero portare ad una divergenza;

Le dimostrazioni teoriche vengono ampiamente discusse dai ricercatori di Fed-Prox [30].

4.2.3 Altri algoritmi rilevanti

Oltre ai due algoritmi di aggregazione presentati ve ne sono altri ed ognuno ha un proprio specifico scopo.

Uno di questi è *CDW-FedAvg*, variante realizzata basandosi su FedAvg, che permette di ridurre l’impatto relativo all’eterogeneità in contesto IoT [31, 32]. Esso è basato sulla media pesata CDW (Centroid Distance Weighted).

Ulteriori alternative risultano essere gli algoritmi *CMFL* [33, 34] e *Fed-SOINN* [35]: il primo riduce il costo relativo alla comunicazione permettendo la condivisione solo degli aggiornamenti locali più rilevanti basandosi su un confronto con il modello globale; il secondo, invece, aumenta l’accuratezza e converge più velocemente, sfruttando il meccanismo di self-attention ed attua apprendimento incrementale in contesto federato basandosi sull’algoritmo SOINN.

Allo stato dell’arte esistono altre possibili strategie di aggregazione in contesto Federated Learning: alcuni di questi come CMFL si focalizzano sul rendere più efficiente la comunicazione quali Fed-Dropout [36] e FedMMD [37].

4.3 Vulnerabilità

L’approccio federato ampiamente descritto presenta delle vulnerabilità che possono essere sfruttate per lanciare attacchi. Questi ultimi possono essere divisi in due categorie: attacchi alle performance del modello ed attacchi alla privacy dei dati [4].

La prima tipologia di attacchi ha l’obiettivo di ridurre le prestazioni del modello globale. Rientrano in questa categoria i seguenti attacchi:

- *Data Poisoning Attack*: ovvero un attacco che mira a corrompere il training del modello globale “avvelenandolo” con update fuorvianti ottenuti dall’uso di dati etichettati scorrettamente durante l’addestramento locale. Le reti GAN possono essere un valido strumento per realizzare questi attacchi [38];
- *Model Poisoning Attack*: più efficace del precedente, coinvolge un partecipante malevolo che ha l’obiettivo di avvelenare il modello globale sfruttando gli aggiornamenti dei modelli locali ed eventualmente modificare i parametri in modo tale da far prevalere il modello malevolo durante l’aggregazione [39];
- *Free-Riding Attack*: si ha in presenza di un partecipante pigro che non contribuisce attivamente ed invia parametri randomici, ma vuole beneficiare della collaborazione;

La seconda tipologia punta ad inferire informazioni dai dati privati dei partecipanti. Questa categoria comprende le seguenti classi di attacchi:

- *Inversione del Modello e Inferenza del Gradiente*: l’attacco di inversione permette di inferire qualcosa sui dati di training dai parametri del modello locale;
- *GAN Reconstruction Attack*: permette di ricostruire i dati dei partecipanti sfruttando le reti GAN e risulta essere molto più efficace degli attacchi di inversione del modello [40].

Le soluzioni proposte per risolvere tali vulnerabilità rientrano tutte nell’utilizzo di tecniche che rendano robusta l’aggregazione e nell’uso di meccanismi di *Differential*

Privacy; inoltre, si consiglia lo sviluppo di metodi per rilevare la presenza di partecipanti malevoli attraverso il confronto dei modelli ricevuti o valutando le performance dei training locali. Altre soluzioni riguardano l’impiego della crittografia e delle stesse reti GAN per creare un dataset simile a quello dei partecipanti in modo da non usare i dati veri privati, mantenendo accuratezza ed avendo così più privacy. Alcuni studi reputano inadeguato l’utilizzo delle blockchain come soluzione perché può comportare problemi di privacy dei dati; inoltre, l’uso di tecniche che vanno ad aggiungere rumore per ottenere privacy rischierebbero di compromettere le prestazioni del modello stesso. Tra alcune soluzioni realizzate per queste vulnerabilità vi è FED-IIoT degli autori Taheri et al. [41] che applica l’algoritmo di difesa A3GAN per rilevare partecipanti malevoli. Vi sono, inoltre, altre ricerche che hanno l’obiettivo di rilevare e ridurre gli attacchi di avvelenamento come quella di Zhao et al. [42] che impiega una GAN per generare un dataset di auditing da usare per controllare i modelli locali; questi ultimi vengono scartati dall’aggregazione qualora non superassero una specifica soglia di accuratezza.

4.4 Problemi generali

Vi sono, però, ulteriori problematiche non legate ad attacchi da parte di utenti malevoli. Essi riguardano la comunicazione, l’eterogeneità dei dispositivi ed il problema dei dati Non-IID (Non Indipendentemente e Identicamente Distribuiti) [4].

La comunicazione è una problematica molto importante in quanto onerosa ed è necessario che diventi più efficiente [34]. Il problema risiede nella fase di addestramento nella quale una grande quantità di dati viene elaborata su più turni e ciò ovviamente rischia di impattare a cascata sullo svolgimento stesso di questa fase. Inoltre risulta essere un grosso limite quello delle caratteristiche inadeguate della rete. Sono stati studiati alcune possibili precauzioni tra cui: la compressione dei modelli locali, anche se ritenuto non sicuro in quanto può agevolare i partecipanti malevoli a non esser rilevati, e l’utilizzo dei soli aggiornamenti più significativi.

L’eterogeneità dei dispositivi coinvolti comporta una serie di problemi: i client innanzitutto possono non riuscire a collaborare attivamente a causa di connessioni

debolì, batteria e memoria limitata [34] e, in generale, hardware diverso. Queste differenze possono portare a riduzioni delle performance e rallentamenti.

Da quest'ultimo problema ne consegue la presenza di dati Non-IID che rende il tutto ancor più difficile e riduce notevolmente l'accuratezza [43]. Questa tipologia di dati è scaturita dal fatto che la distribuzione dei dati è totalmente diversa per ogni partecipante e causa divergenza del modello globale, soprattutto nel FL orizzontale che usa modelli parametrici. Vi sono una serie di soluzioni sviluppate per risolvere il problema della convergenza in presenza di questo tipo di dati come, ad esempio, la soluzione proposta dagli autori Quyen et al. [44] nella quale si sfruttano le reti GAN per bilanciare i dati Non-IID prima di usarli nel training (andando quindi ad effettuare Data Augmentation) scegliendo il sottoinsieme di client con il Deep Reinforcement Learning. Non solo, ma anche nella soluzione DRL-GAN [24] vengono sfruttate due tipologie diverse di reti GAN e ve ne sono molte altre tra cui FEDGAN-IDS [45] che usa le GAN non solo per classificare, ma anche per risolvere le problematiche relative ai dati. Un'altra soluzione può essere l'utilizzo di algoritmi appositamente studiati per gestire questi dati come FedProx, discusso precedentemente. Allo stato dell'arte vi sono numerose possibili soluzioni adottabili per superare queste difficoltà.

4.5 FL-IDS

Il FL è stato sfruttato anche per la realizzazione di Intrusion Detection System e risulta essere un promettente ambito di ricerca che potrebbe risolvere i principali problemi legati alle soluzioni ML-IDS [31]. FEDGAN-IDS [45], già citato precedentemente, è una soluzione IDS che sfrutta il Federated Learning per fare classificazione multiclass e è realizzata attraverso l'uso delle reti GAN: ogni dispositivo coinvolto ed il nodo di aggregazione possiedono una GAN; tale soluzione raggiunge una notevole accuratezza del 99% confrontata con FED-IDS che raggiunge solo l'85% circa nei tre dataset utilizzati. Gli autori Idrissi et al. [46] però reputano FEDGAN-IDS una soluzione meno performante della loro Fed-ANIDS che, sfruttando il FL, usa diversi tipi di Autoencoder per realizzare l'ANIDS: in particolare hanno studiato vari

modelli quali AE, VAE e AAE combinate con gli algoritmi di aggregazione FedAvg e FedProx. Inoltre rispetto a FEDGAN-IDS sfruttano dataset di traffico vero e non trasformano tali dati in immagini. L'accuratezza di FedANIDS varia in base ai dataset usati ed il suo picco lo raggiunge con il 99.95% (contro il 97.66% di FEDGAN-IDS) se implementata con AAE e FedProx sfruttando il dataset USTC-TFC2016 [47]; invece, con il dataset CSE-CIC-IDS2018 [48] mostra un'accuratezza del 94.5% se realizzata con VAE e FedProx contro l'80.7% di FEDGAN-IDS. Altri esempi riportano l'uso delle CNN e MLP - *Multilayer Perceptrons* - per realizzare un FL-IDS mettendole a confronto con la versione tradizionale centralizzata ottenendo così un'accuratezza del 96%, maggiore della controparte [49]. Belenguer et al. hanno proposto GöwFed [50] che impiega il modello tradizionale MLP per il FL-NIDS combinandolo ad un meccanismo di attenzione. Deng et al. [16] proposero un FL-IDS che sfrutta un meccanismo di Self-Attention per una rete Fusion Convolutional Neural Network e trasforma il dataset in immagini. Poongodi et al. [51], invece, realizza un IDS dislocando dei Multilevel discriminator sui partecipanti per effettuare analisi del traffico in maniera continua, combinandolo con un metodo self-adaptive. Ciò porta ad un miglioramento della generazione dei sample sul nodo centrale d'aggregazione. In letteratura esistono, inoltre, altre soluzioni FL-IDS che cercano di mitigare alcuni attacchi: FedDef [52] è una soluzione per FL-NIDS che utilizza un meccanismo di difesa basato sulla perturbazione dell'input con la perdita di accuratezza del solo 3% rispetto ad altre tecniche.

Capitolo 5

GAN - Generative Adversarial Network

La Generative Adversarial Network (GAN) è una metodologia proposta nel 2014 da Goodfellow et al. [53] con l'obiettivo di generare grosse quantità di dati il più possibile simili a dei dati realistici di riferimento. Come anticipato nel capitolo 4.3, le GAN possono essere impiegate per eseguire attacchi, ma anche per migliorare le performance delle odierne soluzioni in ambito DL. Per questi motivi sono largamente utilizzate in ricerche legate agli Intrusion Detection System: tale modello è stato realizzato principalmente per generare immagini, ma può essere impiegato anche per elaborare dati come il traffico di rete.

5.1 Tipologie

Le reti GAN, ideate per la prima volta nel 2014, hanno subito miglioramenti nel corso degli anni e sono state sviluppate diverse varianti. Di seguito vengono elencate le più rilevanti allo stato dell'arte nel campo dei NIDS, ma ne esistono molte altre tra cui: AC-GAN, BiGAN e BEGAN [54].

5.1.1 Vanilla GAN

La prima versione delle GAN viene detta “Vanilla” in termini tecnici e individua la versione originale e/o standard. Tale modello è costituito da due reti neurali dette *Generatore* e *Discriminatore* che svolgono il ruolo di avversari in un “gioco” (fig. 3):

- il Generatore G ha il compito di generare dati sempre più simili ad un dataset di riferimento cercando di confondere il Discriminatore. Generando i dati e sottoponendoli al discriminatore, essa riceve dei feedback che sfrutta per migliorare la qualità di ciò che ha generato modificando i suoi pesi;
- il Discriminatore D ha l’obiettivo di distinguere i dati falsi, ovvero generati da G , da quelli reali del dataset;

Il Generatore riceve in input del rumore che segue una distribuzione normale e consegna in output dati simili a quelli di training. Invece, il Discriminatore riceve sia i dati generati che il dataset di riferimento ed in output restituisce un feedback alla rete generatrice ed un responso sulla genuinità dei dati [45]. Il gioco si conclude quando il Discriminatore non riesce più a distinguere correttamente i dati generati dal Generatore da quelli reali con un’accuratezza del solo 50%. Queste due reti neurali sono semi-supervisionate e sfruttano Back-Propagation. La convergenza è raggiunta quando l’output della funzione del Discriminatore, ovvero un valore di probabilità, si appiattisce appunto a 0,5.

Dopo la convergenza del modello, è possibile tenere anche solo una delle due reti neurali in base allo scopo che si vuole raggiungere: qualora si volesse classificare si potrebbe mantenere solo il Discriminatore, a patto che quest’ultimo vinca contro il Generatore, oppure si potrebbe mantenere solo il Generatore se lo scopo fosse la sola generazione dei dati.

La tipologia basilare di GAN però presenta delle problematiche:

- *Collasso del Modello* causato dal fatto che il Generatore non apprende la distribuzione dei dati in input;
- *Catastrophic Forgetting* nel quale, mentre i modelli imparano, dimenticano ciò che hanno imparato in precedenza causando così divergenza [54];

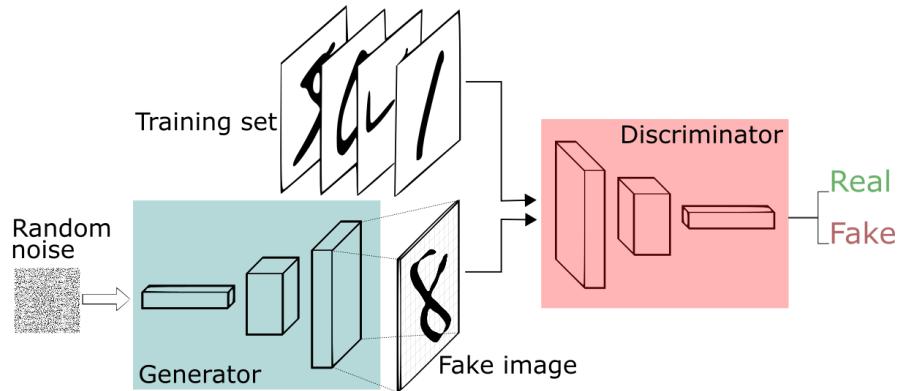


Figura 3: Modello GAN
[55]

- *Divergenza del modello*: quando nessuna delle due componenti (D, G) vince sull'altra;
- *Vanishing Gradients* nel quale il Discriminatore risulta più potente ed il generatore non riesce ad imparare dai feedback ricevuti [56].

Si è cercato di risolvere questi difetti intrinseci del modello realizzando delle varianti.

5.1.2 Altri tipi di GAN per la Cybersecurity

cGAN

Il modello cGAN - *Conditional GAN* - è stato realizzato al fine di avere più controllo sull'output permettendo così di focalizzarsi su specifici aspetti dei dati generati. Questo controllo fa sì che vi sia la possibilità di considerare un contesto di input [57]. La differenza con l'originale è mostrata nello schema in figura 4.

DCGAN

Questa tipologia di architettura GAN - *Deep Convolutional Generative Network* - sfrutta le CNN che sono alla base della struttura delle reti usate nella GAN. La

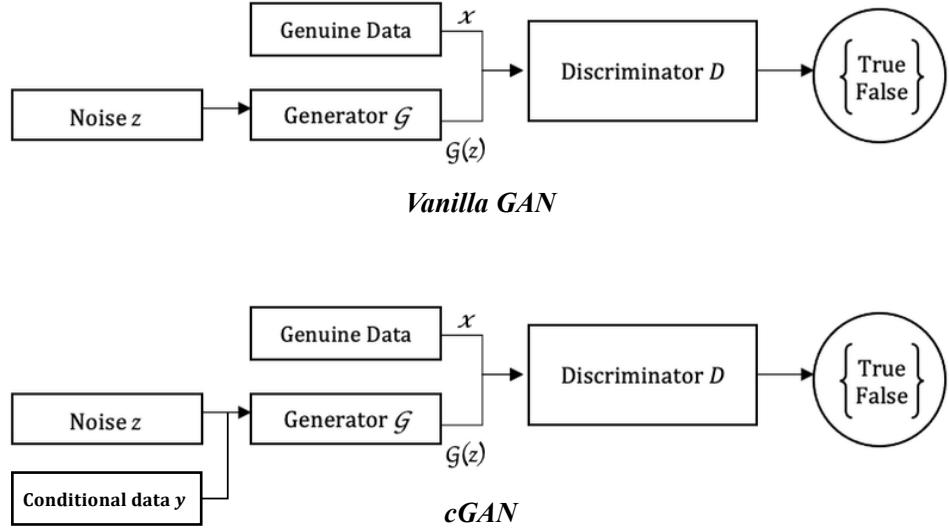


Figura 4: Differenza tra Vanilla GAN e cGAN
[54]

caratteristica di questa architettura è la capacità di generalizzare con elevata accuratezza. Nonostante siano state pensate per la elaborazione delle immagini, sono spesso impiegate anche in campi di ricerca legati alla Sicurezza Informatica. [54, 58].

WGAN

Questo tipo di modello - *Wasserstein GAN* - è stato impiegato in molti campi e si basa sulla *Distanza di Wasserstein* [59]. Esso sfrutta una nuova funzione di costo usando la distanza di Wesserstein anzichè aggiungere rumore: in questo modo il Discriminatore apprende comunque nonostante il Generatore produca immagini di bassa qualità. Ne consegue che la WGAN sia più facile da addestrare e molto più stabile della sua versione standard, risolvendo contemporaneamente il problema del Vanishing Gradients [15].

CTGAN

La tipologia CTGAN [60], invece, è stata studiata per migliorare la generazione dei dati in forma tabulare e sembra superare gli altri metodi di creazione di dati sintetici. Allo stato dell'arte esiste una libreria open-source chiamata 'SDV'¹ [61] che permette di sfruttare modelli GAN di vario tipo ed effettuare una valutazione di ciò che è stato generato. Tale libreria verrà impiegata nel capitolo 6 al fine di effettuare valutazioni sui dati che sono stati generati. In particolare, Bourou et al. [62] hanno studiato le performance di diverse tipologie di GAN per la generazione di dati 'tabulari' ed hanno osservato che le tipologie CTGAN, TableGAN e CopulaGAN sono adatte per la generazione di dati per i sistemi IDS, seppur con qualche lieve differenza e problematiche presenti tra esse.

SAGAN

Questo modello di GAN - *Self-Attention Generative Adversarial Network* - è stato proposto nel 2019 e aggiunge alle fondamenta di una GAN convoluzionale un modulo di self-attention in modo da gestire caratteristiche e/o dettagli a grana fine delle immagini generate. [17]. In questo modo è possibile generare immagini molto dettagliate considerando moltissime "feature" e contemporaneamente permettere al Discriminatore di valutare tutte queste caratteristiche. Tutto ciò svolto con un carico computazionale abbastanza basso [54].

5.2 Vantaggi e problemi riscontrati

Come già ampiamente discusso, il modello GAN possiede un'ottima abilità nel generare anche dati relativi ad attacchi informatici e questo comporta un ottimo addestramento a fronte di eventuali zero-day. Un'altra caratteristica risiede nella prevenzione dell'over-fitting in quanto è possibile generare dati anche per quelle categorie che hanno pochi esempi. Non solo, ma è possibile fare anche classificazione sfruttando il Discriminatore. Tale modello, quindi, può essere impiegato per [54]:

¹<https://docs.sdv.dev/sdv/>

- eseguire Pre-processing dei dati: risulta essere ottimo per bilanciare i dati e creare dati di traffico senza trasformarli in immagini nonchè adatto a risolvere problemi come i dati non bilanciati effettuando di fatto la Data Augmentation;
- creare attacchi ed esempi di traffico malevolo: usata per generare attacchi zero-day mai visti, variazioni di traffico noto e creare dati relativi ad attacchi polimorfici;

Oltre ad i problemi di apprendimento accennati nel capitolo 5.1.1, vi sono dei difetti: l'alta capacità computazionale richiesta ed il tempo di addestramento elevato, soprattutto in presenza di un'attività di multi-classificazione e pre-processing in contesti real-time IDS. Infine, risulta difficile l'elaborazione di attributi categorici come gli indirizzi IP e le porte [24, 63].

5.3 Applicazione negli IDS allo stato dell'arte

Il modello GAN è stato applicato non solo nel campo della Network Intrusion Detection System, ma anche nell'IoT, nella ricerca di Botnet e nel campo dei veicoli a guida autonoma. In particolare, per quanto riguarda lo sviluppo di tecniche NIDS si possono trovare diverse soluzioni: Yang et al. [64] proposero l'utilizzo di una DC-GAN e il LSTM per la classificazione al fine di realizzare un IDS real time, ottenendo un'accuratezza maggiore del 99,5%. Park et al. [65], invece, proposero un modello che per alcune categorie IoT raggiunse il 100% di accuratezza attraverso uno schema GAN-NIDS che sfruttava GAN e classificatori quali CNN, DNN e LSTM. Arafa et al. [66] hanno svolto una valutazione tra VanillaGAN, BiGAN e WGAN in diversi contesti AIDS (tipi di classificazioni, classificatori e dataset) dalla quale è emerso che il modello WGAN rileva molti più attacchi. Taheri et al. [41] hanno proposto uno schema di rilevamento di malware Android che sfrutta algoritmi GAN e FL chiamato Fed-IIoT.

Allo stesso modo molti altri autori citati in questa tesi nei capitoli precedenti hanno sfruttato il modello GAN per eseguire attacchi quali, ad esempio, Mari et al. [21] che suggeriscono di sfruttare i dati generati da reti GAN per aumentare le

prestazioni e prevenire attacchi di evasione. Allo stesso modo Chen et al. [52] hanno proposto una difesa chiamata FedDef studiando attacchi di ricostruzione ed evasione con WGAN.

Altri autori quali Quyen et al. [44] hanno risolto il problema dei dati Non-IID sfruttando il modello GAN e associandolo al RL ed FL al fine di migliorare le performance del FL-IDS sfruttando il RL per scegliere i partecipanti. Autori come Tabassum et al. [45], [24] e Poongodi et al. [51] hanno sfruttato tale modello per aumentare l'accuratezza degli IDS e risolvere problemi di sbilanciamento dei dati.

Capitolo 6

Analisi di tecniche NIDS basate su GAN

La domanda che ci si pone in questa ricerca è la seguente: *È possibile migliorare le prestazioni di un NIDS basato su GAN? Inoltre, l'utilizzo del GAN è utile per migliorare la problematica dei dati sbilanciati nei dataset utilizzati per i NIDS?*

In questo capitolo si approfondiscono tutte le fasi di sviluppo e studio del sistema realizzato nonchè le prestazioni.

6.1 Caratteristiche tecniche del framework

Per la sperimentazione è stata utilizzata la piattaforma open-source di Google chiamata Google Colab¹: servizio Jupyter Notebook ospitato che non richiede alcuna configurazione e fornisce accesso gratuito alle risorse di elaborazione, incluse GPU e TPU. In particolare, è stata utilizzata la sua versione a pagamento Pro per usufruire di RAM elevata e meno limitazioni sulla durata del runtime, nonostante il limite di massimo di sessioni contemporanee. Associato ad esso è stato sfruttato Google Drive nella sua versione da 200 GB per poter salvare facilmente i dataset e poterli caricare direttamente su Google Colab, effettuando così il montaggio di Google Drive sul

¹<https://colab.google/>

runtime. Attraverso questi strumenti è stato possibile utilizzare Python come linguaggio di programmazione e sfruttare librerie quali Tensorflow, Keras e scikit-learn. Ogni elaborazione è stata eseguita sfruttando un “seed” per mantenere consistenza dei risultati.

6.2 Dataset e Pre-processing

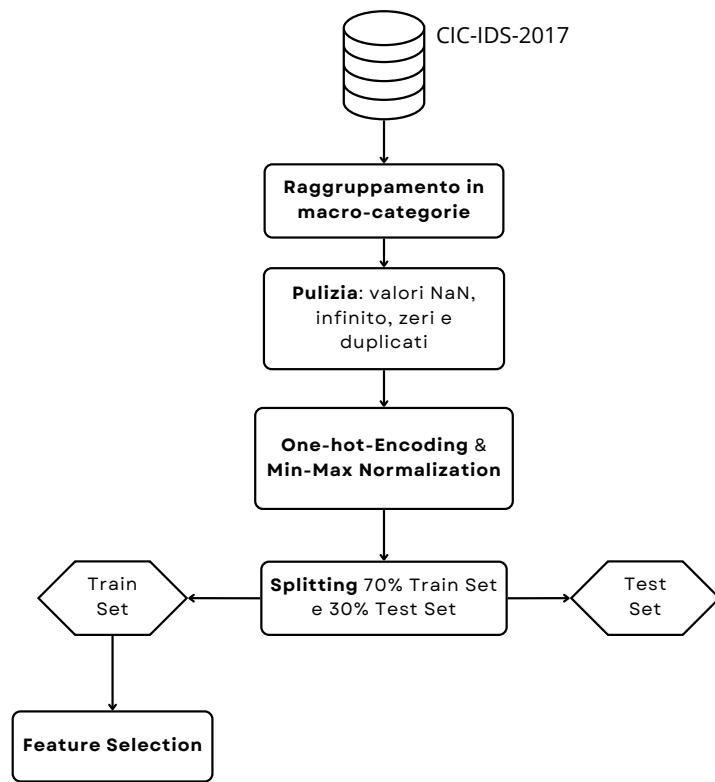


Figura 5: Fasi di pre-processing adottate

Il dataset considerato è il CIC-IDS-2017 [48] nella sua versione migliorata dai ricercatori Liu et al. [67]. Questi ultimi hanno analizzato a fondo i dataset di riferimento nel campo di ricerca dei NIDS, ovvero il CIC-IDS-2017 ed il CIC-CSE-IDS-2018, al fine di correggere errori che compromettono le performance dei sistemi di rilevamento

BENIGN	1594545
Portscan	159066
DoS Hulk	158468
DDoS	95144
Infiltration - Portscan	71767
DoS GoldenEye	7567
FTP-Patator	3972
DoS Slowloris	3859
SSH-Patator	2961
DoS Slowhttptest	1740
Botnet	736
Web Attack - Brute Force	73
Infiltration	36
Web Attack - XSS	18
Web Attack - SQL Injection	13
Heartbleed	11

Tabella 1: Elenco iniziale dei label del traffico.

delle intrusioni. Tale versione risulta risolvere problemi di “labeling” degli attacchi. I file csv divisi per giornate contenute nel dataset sono stati perciò scaricati, uniti e mescolati casualmente in modo che non rimangano concatenati. Si è deciso di considerare gli attacchi etichettati con “Attempted” come “Benigni”, come suggerito dagli stessi ricercatori, ottenendo le classi di attacchi riportate in tabella 1.

Come da grafico riportato in figura 5, la fase di preprocessing presenta diversi step. Tali tecniche sono state adottate in seguito ad una attenta analisi del modus operandi comune applicato dai ricercatori allo stato dell’arte.

1. *Raggruppamento degli attacchi e scrematura:* seguendo il preprocessing svolto dagli autori Ali et al. [68], vengono raccolti gli attacchi nelle corrispondenti macro-categorie d’attacco e ciò significa che, ad esempio, la voce ‘DoS GoldenEye’ è stata accorpata insieme alle altre sotto la voce ‘DoS’. Inoltre, seguendo l’analisi di Kurniabudi et al. [69], viene eliminata la macro-categoria ‘WebAttack’ in quanto presenta un numero insufficiente di samples per far sì che il modello GAN apprenda correttamente, ottenendo così sei classi d’attacco finali. Il dataset si presenta con 2099774 record e 89 feature totali;

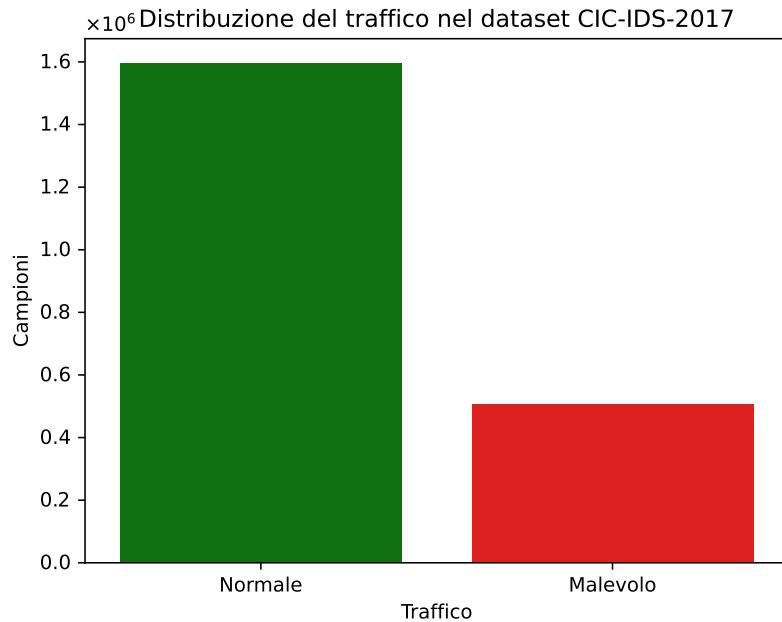


Figura 6: Distribuzione del traffico malevolo e benigno dopo la pulizia del dataset.

2. *Pulizia del dataset:* seguendo Ali et al. [68] vengono eliminate le feature relative agli indirizzi IP, al timestamp ed al flowID in quanto possono essere spoofati. Sfruttando del codice presente sul sito Kaggle², si procede a controllare ed eliminare quei dati che presentano valori NaN, infinito o righe/colonne di tutti zero e duplicate; tali tecniche sono uno standard di pulizia dei dataset e sono state utilizzate anche da Li et al. [70]. In seguito alla pulizia, si ottengono le distribuzione in figura 6 e 7;
3. *Encoding e Normalizzazione:* viene eseguita la cosiddetta *one-hot-encoding* per la multiclassificazione e viene applicata la *normalizzazione min-max*, quest'ultima eseguita da molti ricercatori [70, 68];
4. *Divisione in Train Set e Test Set:* a causa dello sbilanciamento evidente dei dati tra le sei classi di traffico si applica la funzione *StratifiedShuffleSplit* per poter dividere in due set (Train set 70% e Test set 30%) mantenendo una percentuale

²<https://www.kaggle.com/code/filipekoriginal/cicids2017-preprocessing/notebook>

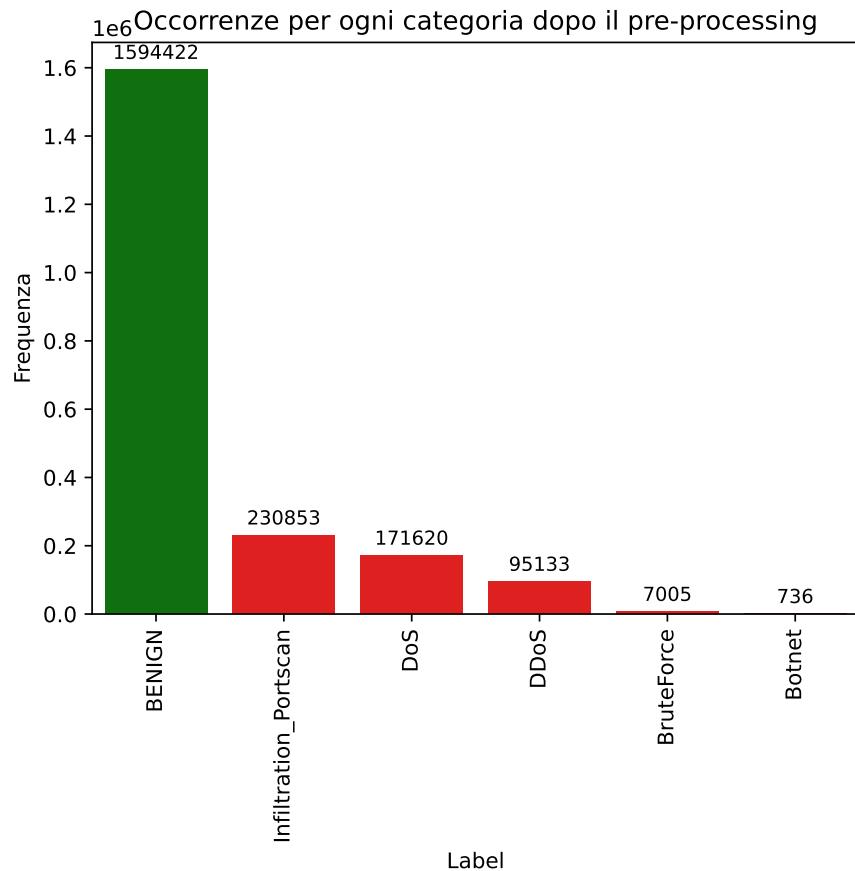


Figura 7: Distribuzione del traffico per le 6 classi dopo la pulizia del dataset.

omogenea di sample per ogni classe. La distribuzione delle classi nei set è visibile in figura 8 dove la classe '0' rappresenta i benigni e le successive classi rispettano l'ordine decrescente di sample (1-5). Il tutto è stato eseguito in modo tale da evitare data leakage in sessione;

5. *Feature Selection:* come metodi di feature engineering sono state scelte: la selezione per importanza sfruttando una Random Forest e per varianza uguale a 0. La prima metodologia classifica per importanza tutte le features e, invece, la seconda permette di analizzare la varianza delle feature ed eliminare quelle con varianza zero o molto vicina allo zero in quanto non incidono sulla classificazione [68]. Le feature rimaste sono 73 di cui vengono scelte solo le prime 35

in ordine di importanza (tale cifra è da considerarsi una media tra un minimo di circa 8 e massimo di 70 feature). La scelta di tali tecniche è basata su diversi studi e modus operandi comune a diversi ricerche [68, 71, 72]. Viene riportato in Appendice A in figura 42 l'esito dell'analisi d'importanza delle feature.

Un'analisi a posteriori seguendo lo studio di Peppes et al. [72] rivela, come visibile dal grafico delle correlazioni in Appendice A in figura 43, che le feature maggiormente in relazione con l'etichetta 'Label' e, quindi, con le categorie d'attacco sono tra quelle scelte finali, riportate in tabella 2. Risultano essere perciò state correttamente scelte come rilevanti ai fini della rilevazione degli attacchi. Sono state escluse solamente due feature tra quelle ad alta correlazione.

FlowDuration	BwdPacketLengthStd	DstPort
TotalLengthofFwdPacket	FlowIATMax	PacketLengthVariance
PacketLengthStd	BwdPacketLengthMax	FlowPackets/s
PacketLengthMax	FwdPacketLengthMax	BwdPacketLengthMean
PacketLengthMean	BwdSegmentSizeAvg	FwdPackets/s
AveragePacketSize	FlowBytes/s	FwdRSTFlags
FwdPacketLengthStd	SubflowBwdBytes	TotalLengthofBwdPacket
FlowIATMean	BwdInitWinBytes	FwdIATMax
FwdIATStd	FwdHeaderLength	BwdBulkRateAvg
BwdHeaderLength	FwdIATTTotal	FwdSegSizeMin
FWDInitWinBytes	BwdPackets/s	FwdPacketLengthMean
FwdSegmentSizeAvg	SubflowFwdBytes	-

Tabella 2: Elenco delle 35 feature selezionate.

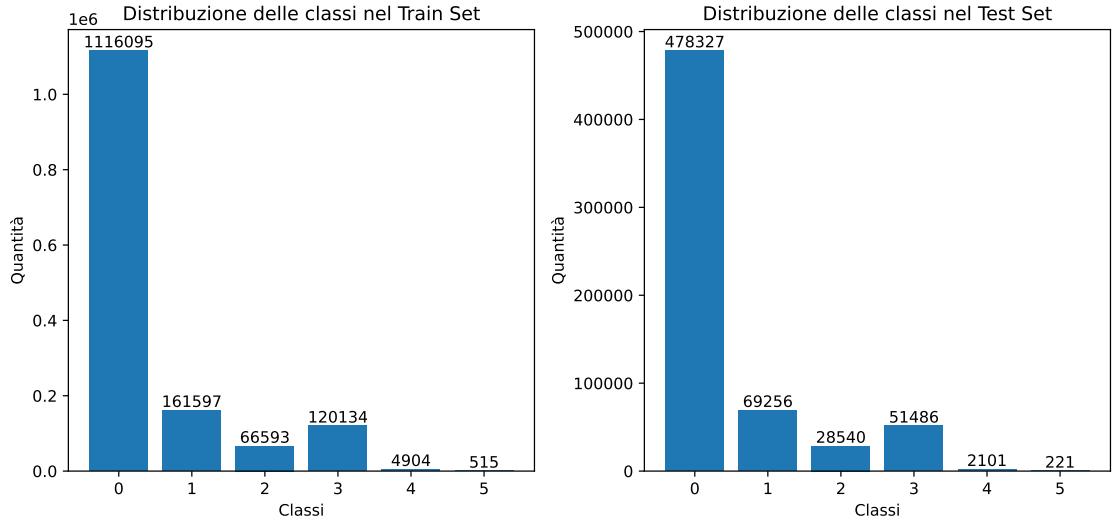


Figura 8: Distribuzione del traffico nel Train Set e nel Test Set.

6.3 Caso di studio ed architettura

Questa tesi prende ispirazione da alcuni articoli accademici: l’architettura FEDGAN-IDS basata su immagini proposto da Tabassum et al. [45], il sistema Fed-ANIDS di Idrissi et al. [46], lo studio di Bacevicius et al. [73] nel quale viene analizzato il comportamento di diversi modelli ML in presenza di un dataset sbilanciato e quello di Liu et al. [74], nel quale viene usata una ACGAN per realizzare un framework di Data Augmentation per il Federated Learning. Tali testi accademici sfruttano una serie di diverse soluzioni per arginare un problema comune, ovvero quello della performance dei NIDS in presenza di dati non bilanciati. Tali soluzioni comportano l’uso di modelli GAN, delle tecniche di Data Augmentation nonché Federated Learning.

Questo studio, quindi, verte sulla verifica delle performance di un NIDS, realizzato con due tipologie di algoritmi AI, che sfrutta il modello Vanilla GAN per effettuare Data Augmentation al fine di arginare il problema dello sbilanciamento dei dati sotto forma di tabelle.

Di seguito vengono descritti brevemente i componenti del sistema impiegati per la sperimentazione:

- *IDS*: al fine di studiare il comportamento degli IDS a cui si applica Data Augmentation si è deciso, dopo una serie di tentativi, di prendere come riferimento un IDS che sfrutta *MLP* ed uno basato su *Random Forest Classifier*;
- *GAN*: il modello iniziale della Vanilla GAN è stato realizzato sfruttando il progetto di tesi del Dottore Swart [75], il progetto del Dr. Greig Fotheringham³ nonché quello del ricercatore Emami⁴. La Vanilla GAN viene perciò impiegata come strumento per la Data Augmentation;

Nei successivi capitoli viene descritto lo sviluppo dei singoli componenti in maniera esaustiva.

6.4 Metriche di valutazione

Prima di addentrarsi nei capitoli relativi allo sviluppo del sistema, è necessario approfondire le metriche di valutazione scelte per analizzare le prestazioni di ogni singola componente. Ogni metrica è stata scelta tra quelle più utilizzate allo stato dell'arte nel contesto di ricerca che si sta analizzando. Di seguito vengono elencate ed esaustivamente descritte le metriche utilizzate, divise per componente sul quale sono state impiegate.

Metriche relative al modello GAN

La GAN è di per sé difficile da valutare sfruttando i classici grafici delle curve loss e di accuratezza perché coinvolge due reti neurali differenti che cercano di prevalere una sull'altra. Ciò comporta grafici molto traballanti e di difficile interpretazione, perciò ci si basa spesso su analisi qualitative che mirano a verificare la similitudine dei dati generati con quelli reali di riferimento nonché analisi quantitative basate su formule statistiche che calcolano distanze tra le distribuzioni dei dati per capire quanto esse siano differenti.

³<https://github.com/mm32-star/Network-Data-Generation-Vanilla-GAN>

⁴<https://www.kaggle.com/code/samanemami/gan-on-tabular-data/notebook>

Uno studio approfondito della letteratura allo stato dell’arte ha portato alla scelta di più metodi per avere un quadro complessivo della bontà della GAN [76]: sono state sfruttate sia analisi quantitative che qualitative, sfruttando anche la libreria SDV⁵.

- *Analisi qualitativa*: per l’esame visuale sono stati sfruttati i confronti con matrici di correlazione, grafici delle distribuzioni per specifiche feature e scatter plot delle relazioni più interessanti. Esse hanno permesso di vedere visivamente quanto i dati generati fossero simili rispetto ai dati reali. I grafici delle loss sono comunque stati generati per completezza dell’analisi;
- *Analisi quantitativa*: una ricerca ha portato a scegliere la *Distanza di Wasserstein* sia per la distribuzione completa dei dati generati che per singole features rispetto ai dati reali attesi e l’*Errore Quadratico Medio (MSE)* solo tra singole feature.
- *Libreria SDV*: sono stati elaborati i report di diagnostica e qualità forniti dalla libreria per avere dettagli sulla similitudine delle feature e la loro correlazione nonché basilari controlli di validità sulla struttura dei dati.

In letteratura si discute molto sul riuscire a far sì che il modello GAN generi dati simili a quelli reali, ma che abbiano anche la caratteristica di essere sufficientemente diversi e non uguali ai dati reali di riferimento.

Ciò che si vuole ottenere dall’analisi qualitativa è evidenziare una generazione di dati quanto più simile a quella reale, ma garantendo allo stesso tempo la diversità degli stessi. Invece, per la Libreria SDV si vuole ottenere un punteggio quanto più alto possibile, come sarà visibile dalle tabelle nel capitolo 6.5.1.

Infine, per l’analisi quantitativa o statistica sono state impiegate:

- *Distanza di Wasserstein*: che computa la distanza tra due distribuzioni ed, infatti, è usata come funzione di loss per il modello alternativo WGAN con l’obiettivo di ridurre tale valore. Per questi motivi si cerca di ottenere una distanza quanto più possibile bassa che indichi come le due distribuzioni siano uguali;

⁵<https://docs.sdv.dev/sdv>

- *Errore Quadratico Medio (MSE)*: usata per calcolare la differenza tra due distribuzioni. Un MSE basso indica una buona generazione della feature in esame e, quindi, poca differenza con la distribuzione reale;

Oltre ad essi viene impiegata anche l'analisi delle performance di classificazione [76] degli IDS che però vengono descritte nella sezione successiva con un'approfondita analisi dei risultati nel capitolo 6.6.

Metriche relative all'IDS

Per quanto concerne le metriche relative alla valutazione delle performance degli IDS impiegati, si sono sfruttate le seguenti metriche comuni allo stato dell'arte:

- *Grafici delle Curve Loss e di Accuratezza* impiegati per monitorare l'addestramento dei modelli che lo necessitavano;
- *Accuracy e Balanced Accuracy* per analizzare le performance nel contesto complesso dello sbilanciamento dei dati. $Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$;
- *Precision, Recall e F1-Score* nelle loro versioni 'weighted' e anche 'macro' per una visione d'insieme sui risultati a causa dello sbilanciamento.
 - *Precision*⁶: è il rapporto $Precision = \frac{TP}{TP+FP}$, ovvero la capacità di non segnalare come positivo un campione negativo;
 - *Recall*⁷: è il rapporto $Recall = \frac{TP}{TP+FN}$ ed evidenzia la capacità di trovare correttamente tutti i campioni positivi;
 - *F1-score*⁸: può essere considerata come la media ponderata di Precision e Recall ed è utile soprattutto in contesti dove vi sono classi sbilanciate

$$F1 = \frac{2*Precision*Recall}{Precision+Recall} = \frac{2*TP}{2*TP+FP+FN};$$

⁶https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_score.html

⁷https://scikit-learn.org/stable/modules/generated/sklearn.metrics.recall_score.html

⁸https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html

- *Report di Classificazione* per un quadro completo delle performance;
- *Grafico della Confusion Matrix* per poter analizzare le performance da un punto di vista di *True Positive*, *True Negative*, *False Positive* e *False Negative*.

6.5 Sviluppo del modello GAN

Il modello GAN è stato sviluppato seguendo più fonti di codice tra cui: il progetto di tesi del Dr. Swart [75], il progetto del Dr. Greig Fotheringham⁹ e del ricercatore Emami¹⁰. Il codice di partenza è stato ulteriormente modificato durante il processo per ottimizzare il modello ed adattarlo al dataset nonché al tipo di pre-processing eseguito sullo stesso.

La complessa sperimentazione delle diverse architetture del modello GAN nonché il tuning degli iperparametri al fine di realizzare un modello che meglio si adattasse allo scopo finale ha inciso notevolmente sulle tempistiche a causa della difficoltà e delle moltissime variabili da considerare. In totale sono state svolte circa 100 prove *trial-and-error* raccogliendo i risultati sia positivi che negativi in un file excel: dapprima si è sperimentato sulla possibilità di utilizzare una WGAN, successivamente si è passati ad una Vanilla GAN ragionando sulla sua architettura ed infine, per effettuare il fine-tuning, si è sperimentato l'utilizzo di *KerasTuner* (un framework che permette di ottimizzare gli iperparametri automaticamente) optando poi per una ricerca manuale degli iperparametri più adatti. Risulta evidente, perciò, che il lavoro svolto necessitava di adeguato tempo al fine di raggiungere gli obiettivi che ci si era posti, considerando il tempo di computazione del modello stesso. Tale approccio viene descritto in maniera più esaustiva di seguito per poter apprezzare meglio il lavoro svolto.

WGAN e VanillaGAN: studio delle diverse architetture

Inizialmente ci si era posti la possibilità di sfruttare il modello WGAN per le migliori performance riscontrate in letteratura rispetto alla sua versione originale Vanilla GAN. Durante la sperimentazione sono stati realizzati 6 file ipynb tramite Google Colab per studiare la WGAN. Sono stati svolti tentativi sfruttando *TF-GAN* (libreria Tensorflow per l'uso di modelli GAN) nonché codice open-source presente su github andando a creare una propria funzione di loss basata appunto sulla Distanza di Wasserstein. Tali approcci seppur interessanti non sono stati sfruttati per la loro

⁹<https://github.com/mm32-star/Network-Data-Generation-Vanilla-GAN>

¹⁰<https://www.kaggle.com/code/samanemami/gan-on-tabular-data/notebook>

complessità. Un miglior background ed una maggior esperienza nel campo dell’Intelligenza Artificiale avrebbe sicuramente influito positivamente sull’utilizzo di tali risorse. Successivamente, al fine di non impattare negativamente sui tempi di realizzazione di questo studio, si è optato per realizzare la versione Vanilla GAN: in seguito allo studio di diverso codice python si è riusciti a realizzare una Vanilla GAN funzionante da cui partire per sperimentare diverse architetture ed iperparametri. Anche in questo caso sono stati svolti diversi tentativi che hanno visto coinvolti 5 file ipynb prima di trovare una combinazione adatta su cui lavorare; sperimentando anche diverse metriche di valutazione rispetto a quelle finali riportate in questa tesi come *KS-Statistic* e *KL Divergence*. Le architetture studiate inizialmente hanno visto coinvolti meno layer e la funzione di attivazione *ReLU* (come riportato nel listing 6.1) anzichè quella finale *LeakyReLU*. Una prima versione vedeva la presenza di tre livelli per il Generatore rispettivamente con 64, 128, 256 neuroni successivamente aumentati a 128, 256, 512; il Discriminatore presentava anch’esso solamente tre livelli interni poi aumentati a cinque aggiungendo due layer di *DropOut*. È stato effettuato un tentativo di introduzione di un layer di Self-Attention al Generatore che però non ha riscontrato successo. Si è proseguito perciò con la valutazione parallela dell’architettura e dei primi iperparametri fondamentali come: il *learning rate* del Generatore e del Discriminatore (inizialmente posti come uguali), il *batch size* e le *epoches*. A questo punto è stato necessario approfondire l’argomento sfruttando ricerche accademiche e forum per poter apprendere come migliorare tale architettura: sono stati perciò aggiunti layer di *BatchNormalization* al Generatore e cambiata la funzione di attivazione in *LeakyReLU*, come visibile al listing 6.2.

La situazione a metà processo di sperimentazione del modello GAN riportava risultati scarsi, con grosse difficoltà di generazione e risultati delle metriche non soddisfacenti e spesso riscontrando il problema del *Collasso del Modello*.

Prima di raggiungere l’architettura finale sono state svolte ulteriori prove aggiungendo più layer sia al Generatore che al Discriminatore aumentandoli sequenzialmente e studiandone gli effetti ed i risultati ottenuti. Si è perciò arrivati gradualmente a definire, ad esempio, un Generatore con sette layer totali: tre di BatchNormalization

ed i restanti di tipo *Dense* con rispettivamente 32, 64, 128 neuroni; ed al Discriminatore, invece, vi si presentavano tre layer di *DropOut* anzichè due come in precedenza. Un'ultima sperimentazione ha portato a valutare l'aumento dei neuroni nuovamente a 128, 256, 512.

Giunti ad aver definito un'architettura simile a molte ricerche ed ottenendo risultati promettenti, che non comportavano Mode Collapse, si è reso necessario il fine-tuning degli iperparametri in quanto settati ad un valore standard o suggerito da articoli accademici, ovvero: alpha della LeakyRelu=0,1; momentum del BatchNormalization=0,8 e dropout Rate=0,1.

```

class Generator():
    def __init__(self, batch_size):
        self.batch_size = batch_size

    def build_model(self, input_shape, dim, data_dim):
        input_layer = Input(shape=input_shape, batch_size=self.batch_size)
        x = Dense(neuroni, activation='relu')(input_layer)
        x = Dense(neuroni * 2, activation='relu')(x)
        x = Dense(neuroni * 4, activation='relu')(x)
        output_layer = Dense(data_dim, activation='sigmoid')(x)
        return Model(inputs=input_layer, outputs=output_layer)

class Discriminator():
    def __init__(self, batch_size):
        self.batch_size=batch_size

    def build_model(self, input_shape, neuroni):
        input = Input(shape=input_shape, batch_size=self.batch_size)
        x = Dense(neuroni * 4, activation='relu')(input)
        x = Dropout(0.1)(x)
        x = Dense(neuroni * 2, activation='relu')(x)
        x = Dropout(0.1)(x)
        x = Dense(neuroni, activation='relu')(x)
        x = Dense(1, activation='sigmoid')(x)
        return Model(inputs=input, outputs=x)

```

Listing 6.1: Codici del Generatore e Discriminatore della prima versione di GAN.

```

class Discriminator():
    def __init__(self, batch_size):
        self.batch_size=batch_size

    def build_model(self, input_shape, neuroni):
        input = Input(shape=input_shape, batch_size=self.batch_size)
        x = Dense(neuroni * 4)(input) #512
        x = LeakyReLU(alpha=alpha_leakyRelu)(x)
        x = Dropout(dropoutRate)(x)
        x = Dense(neuroni * 2)(x) #256
        x = LeakyReLU(alpha=alpha_leakyRelu)(x)
        x = Dropout(dropoutRate)(x)
        x = Dense(neuroni)(x) #128
        x = LeakyReLU(alpha=alpha_leakyRelu)(x)
        x = Dense(1, activation='sigmoid')(x)
        return Model(inputs=input, outputs=x)

class Generator():
    def __init__(self, batch_size):
        self.batch_size = batch_size

    def build_model(self, input_shape, neuroni, data_dim):
        input_layer = Input(shape=input_shape, batch_size=self.batch_size)
        x = Dense(neuroni)(input_layer) #neuroni=128
        x = LeakyReLU(alpha=alpha_leakyRelu)(x)
        x = BatchNormalization(momentum=momentumBN)(x)
        x = Dense(neuroni * 2)(x) #256
        x = LeakyReLU(alpha=alpha_leakyRelu)(x)
        x = BatchNormalization(momentum=momentumBN)(x)
        x = Dense(neuroni * 4)(x) #512
        x = LeakyReLU(alpha=alpha_leakyRelu)(x)
        x = BatchNormalization(momentum=momentumBN)(x)
        output_layer = Dense(data_dim, activation='sigmoid')(x)
        return Model(inputs=input_layer, outputs=output_layer)

```

Listing 6.2: Codici del Generatore e Discriminatore del modello GAN con architettura migliorata.

VanillaGAN: fine-tuning iperparametri

L'ottimizzazione degli iperparametri ha richiesto molto più tempo rispetto alla definizione dell'architettura al fine di trovare il giusto equilibrio tra i valori di ogni iperparametro e la qualità dei dati generati che si otteneva. Sono state svolte molte

prove considerando i nove iperparametri coinvolti. Inizialmente sono stati definiti dei valori massimi e minimi entro il quale rientrare per evitare problemi distruttivi nella generazione che vedevano coinvolti i seguenti iperparametri: *learning rate*, *batch size*, *epoches* e *noise dimension*. In particolare, per quanto riguarda le epoches si è lavorato all'interno dell'intervallo [300 - 8000]; per il noise dimension l'intervallo è stato [1 - 64]; il learning rate invece [0,00001 - 0,008] ed infine per il batch size si è usato [64 - 2048]. I restanti iperparametri analizzati, invece, presentano i seguenti intervalli: *alpha* della LeakyRelu = [0,01 - 0,3]; *momentum* del BatchNormalization = [0,65 - 0,85] e *dropout Rate* = [0,1 - 0,2].

In figura 9 si riporta un esempio di una situazione di apprendimento fallace ed i relativi iperparametri configurati sono riportati in tabella 3.

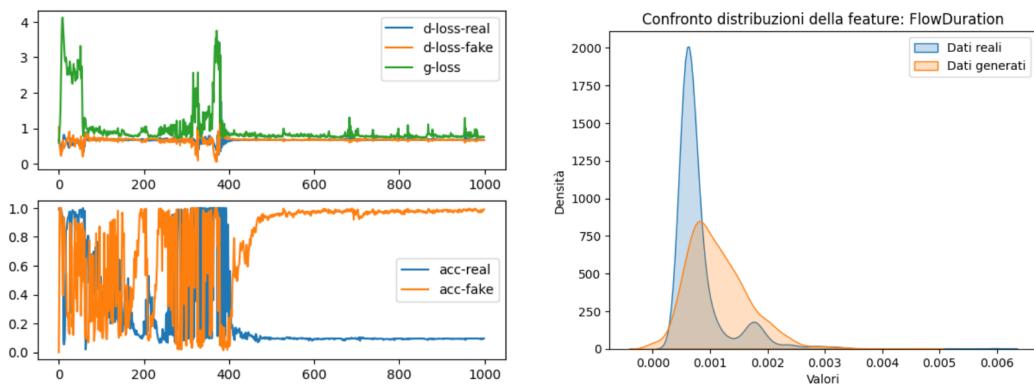


Figura 9: Apprendimento fallace n°1 riscontrato durante l'ottimizzazione degli iperparametri in fase di sperimentazione del modello GAN.

Un secondo esempio di ottimizzazione successiva alla n°1 lo si riporta in figura 10 dove si evidenzia un miglioramento; i relativi iperparametri configurati sono riportati in tabella 4.

Alpha LeakyRelu	0.01
BatchNormalization Momentum	0.8
Dropout Rate	0.1
Neuroni	128
Batch Size	1024
Learning Rate Discriminator	0.001
Learning Rate Generator	0.003
Noise Dimension	16
Epoche	1000

Tabella 3: Iperparametri scelti che causano un apprendimento non efficiente relativi alla figura 9.

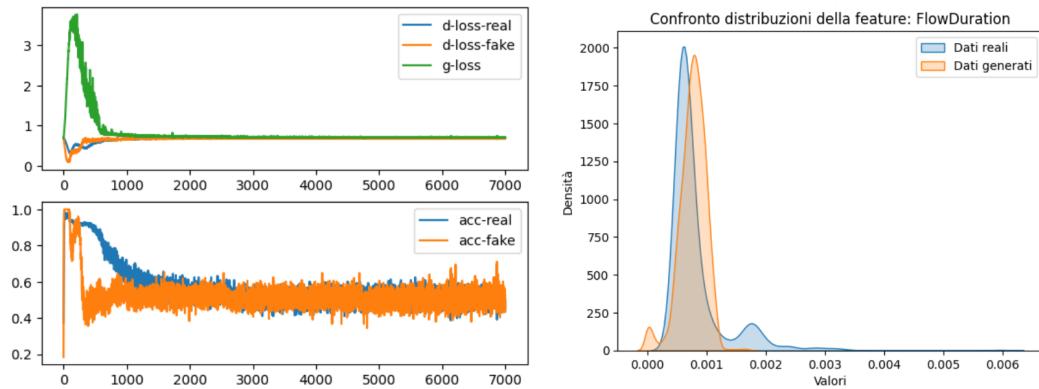


Figura 10: Apprendimento con iperparametri ottimizzati del modello GAN successivamente all'esempio n°1.

Alpha LeakyRelu	0.01
BatchNormalization Momentum	0.8
Dropout Rate	0.1
Neuroni	64
Batch Size	256
Learning Rate Discriminator	0.0001
Learning Rate Generator	0.0003
Noise Dimension	16
Epoche	7000

Tabella 4: Iperparametri ottimizzati dopo l'esempio n°1 relativi alla figura 10.

VanillaGAN finale

Tutto il lavoro precedentemente illustrato ha perciò portato a definire l’architettura del modello come riportato in figura 11. In particolare, sono stati scelti per il modello layer con funzione di attivazione LeakyReLU, layer di Batch Normalization con uno specifico momentum per il Generatore e layer di Dropout con un suo specifico rate per il Discriminatore. Tale architettura permette di generare dati soddisfacenti in due configurazioni differenti dell’iperparametro *alpha* relativo alla *LeakyReLU*: in particolare, con alpha=0,01 e alpha=0,15. Tutti gli iperparametri adottati sono riassunti in tabella 5. Si riportano, inoltre, i codici del Generatore e Discriminatori ai listing 6.3 e 6.4. L’ottimizzatore utilizzato è di tipo Adam con i learning rate evidenziati sempre in tabella 5.

Tali configurazioni finali del modello GAN sono state definite in seguito ad una lunga sperimentazione di diverse architetture e iperparametri, come già anticipato ed illustrato all’inizio del capitolo, considerando la difficoltà nell’addestramento dello stesso [77]: diverse architetture sia per il Generatore che per il Discriminatore comportano una netta differenza nella generazione di dati di qualità; inoltre, tali scelte dipendono dalla complessità delle distribuzioni dei dati e la loro quantità nonché dalla qualità che si vuole ottenere, le risorse disponibili e la stabilità del modello [72].

Alpha LeakyRelu	0.01/0.15
BatchNormalization Momentum	0.8
Dropout Rate	0.1
Neuroni	128
Batch Size	2048
Learning Rate Discriminator	0.0001
Learning Rate Generator	0.0003
Noise Dimension	16
Epoche	6000

Tabella 5: Iperparametri finali configurati per il modello GAN.

Model: "GAN"

Layer (type)	Output Shape	Param #
<hr/>		
Generatore (Sequential)	(None, 35)	188323
Discriminatore (Sequential)	(None, 1)	182785
<hr/>		
Total params: 371108 (1.42 MB)		
Trainable params: 186531 (728.64 KB)		
Non-trainable params: 184577 (721.00 KB)		

(a) Architettura GAN completa.

Model: "Generatore"

Layer (type)	Output Shape	Param #
<hr/>		
dense (Dense)	(None, 128)	2176
batch_normalization (Batch Normalization)	(None, 128)	512
dense_1 (Dense)	(None, 256)	33024
batch_normalization_1 (Batch chNormalization)	(None, 256)	1024
dense_2 (Dense)	(None, 512)	131584
batch_normalization_2 (Batch chNormalization)	(None, 512)	2048
dense_3 (Dense)	(None, 35)	17955
<hr/>		
Total params: 188323 (735.64 KB)		
Trainable params: 186531 (728.64 KB)		
Non-trainable params: 1792 (7.00 KB)		

(b) Architettura Generatore.

Model: "Discriminatore"

Layer (type)	Output Shape	Param #
<hr/>		
dense_4 (Dense)	(None, 512)	18432
dropout (Dropout)	(None, 512)	0
dense_5 (Dense)	(None, 256)	131328
dropout_1 (Dropout)	(None, 256)	0
dense_6 (Dense)	(None, 128)	32896
dropout_2 (Dropout)	(None, 128)	0
dense_7 (Dense)	(None, 1)	129
<hr/>		
Total params: 182785 (714.00 KB)		
Trainable params: 0 (0.00 Byte)		
Non-trainable params: 182785 (714.00 KB)		

(c) Architettura Discriminatore.

Figura 11: Architettura del modello GAN.

```
def build_generator(data_dim, noise_dim):
    model = Sequential(name="Generatore")
    model.add(Dense(neuroni,
                   activation=LeakyReLU(alpha_leakyRelu),
                   kernel_initializer="he_uniform",
                   input_dim=noise_dim))
    model.add(BatchNormalization(momentum=momentumBN))
    model.add(Dense(neuroni*2,
                   activation=LeakyReLU(alpha_leakyRelu),
                   kernel_initializer="he_uniform"))
    model.add(BatchNormalization(momentum=momentumBN))
    model.add(Dense(neuroni*4,
                   activation=LeakyReLU(alpha_leakyRelu),
                   kernel_initializer="he_uniform"))
    model.add(BatchNormalization(momentum=momentumBN))
    model.add(Dense(data_dim, activation="sigmoid"))
    return model
```

Listing 6.3: Codice del Generatore del modello GAN.

```
def build_discriminator(data_dim):
    model = Sequential(name="Discriminatore")
    model.add(Dense(neuroni*4,
                   activation=LeakyReLU(alpha_leakyRelu),
                   kernel_initializer="he_uniform",
                   input_dim=data_dim))
    model.add(Dropout(dropoutRate))
    model.add(Dense(neuroni*2,
                   activation=LeakyReLU(alpha_leakyRelu),
                   kernel_initializer="he_uniform"))
    model.add(Dropout(dropoutRate))
    model.add(Dense(neuroni,
                   activation=LeakyReLU(alpha_leakyRelu),
                   kernel_initializer="he_uniform"))
    model.add(Dropout(dropoutRate))
    model.add(Dense(1, activation="sigmoid"))
    model.compile(loss="binary_crossentropy",
                  optimizer=optimizerD,
                  metrics=["accuracy"])
    return model
```

Listing 6.4: Codice del Discriminatore del modello GAN.

6.5.1 Valutazione dei dati generati

Di seguito vengono riportate alcune valutazioni interessanti sui dati generati: in particolare, si riportano le analisi approfondite delle classi di traffico Botnet, Bruteforce e Benign. In questa sezione perciò, per ogni classe sotto esame, verrà svolta una valutazione quantitativa e qualitativa dei dati generati in confronto a quelli reali del dataset utilizzando le metriche illustrate nel capitolo 6.4. In questo modo sarà possibile dare una valutazione completa sulla qualità dei dati generati dalla GAN. Le restanti analisi delle classi sono descritte in maniera completa in Appendice B.

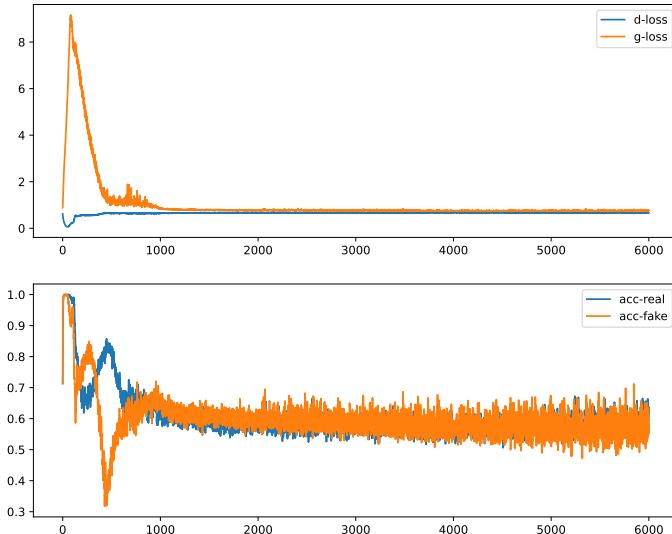


Figura 12: Grafico Loss ed Accuratezza del modello GAN relativi alla classe Bruteforce con alpha=0,01.

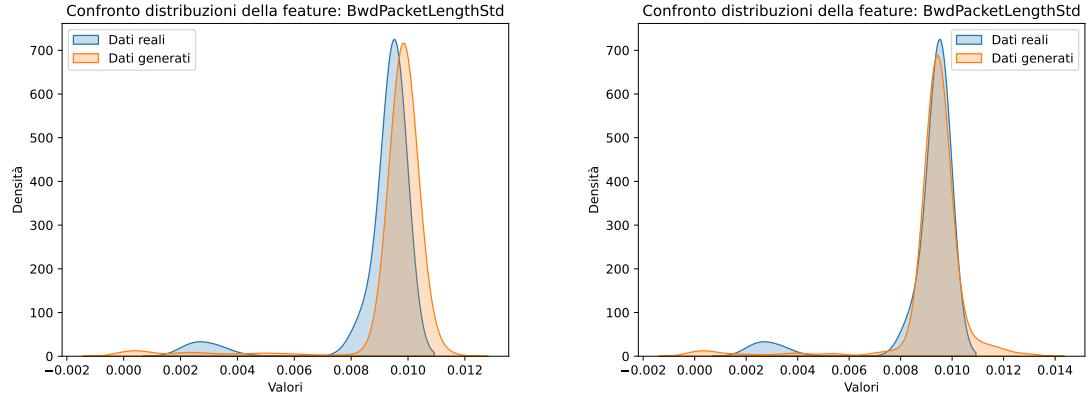
Una prima considerazione da fare sull'apprendimento del modello GAN e la generazione dei dati è quella di analizzare i grafici delle Curve Loss e dell'Accuratezza: essi permettono di capire visivamente se il modello è in difficoltà e se fallisce nell'apprendimento; si intende perciò l'identificazione di un eventuale Collasso del Modello o difficoltà di Convergenza. Per questi motivi si riporta, come esempio di analisi, il grafico in figura 12 relativo all'apprendimento da parte della GAN della classe Bruteforce

con alpha=0,01 dal quale si evince un training stabile, raggiungendo un'accuratezza media nella generazione dei samples (riportata come “acc-fake”) di circa 60%.¹¹. Per ogni classe generata sono stati analizzati tali grafici che, grazie al tuning approfondito della GAN descritto precedentemente, non presentano alcuna problematica di sorta evidenziando un buon apprendimento.

Una volta constatato che il modello GAN non ha alcun problema di apprendimento per ogni classe generata, successivamente è stata affrontata l’analisi dei dati generati. In generale, il modello GAN ha avuto una difficoltà attesa nella generazione dei dati relativi alle classi minori d’attacco come Botnet e Bruteforce a causa della quantità ridotta di samples; si evidenzia altresì una netta difficoltà nella generazione della classe Portscan in entrambe le configurazioni di iperparametri. Ci si è focalizzati sul raggiungere una buona qualità dei dati generati relativi alle classi minori e, perciò, una prima generazione di dati è stata eseguita mantenendo la stessa quantità dei dati reali per una verifica immediata ed una seconda generazione con 10000 samples. Partendo ad analizzare la classe Botnet, che contiene 515 samples nel Train Set, si può notare come i dati generati dal modello GAN presentano similitudini tra la generazione con alpha=0,01 e con alpha=0,15 ed, anzi, si evidenzia una migliore generazione per alpha=0,15, visibile in figura 13; tali grafici rappresentano visivamente il confronto delle distribuzioni dei valori della feature BwdPacketLengthStd tra i dati reali del dataset (in azzurro) e quelli generati dalla GAN (in arancione) da cui si vuole rilevare una curva quanto più sovrapposta possibile. Inoltre, dai grafici delle matrici di correlazione in figura 14, il cui utilizzo ha lo scopo di vedere come il grafico dei dati generati assomigli a quello dei dati reali del dataset, si nota come in 14b vi sia una maggiore similitudine visiva del grafico dei dati generati posto sulla destra con quello dei dati reali sulla sinistra, rispetto al 14a.

In particolare, si nota in tabella 6 che statisticamente sembra esser leggermente migliore la generazione dei dati con alpha=0,01 nonostante la miglior similitudine evidenziata dai grafici. Il valore della Wasserstain Distance totale si intende sulla distribuzione di tutto il dataset della classe Botnet; mentre i valori di Wasserstain

¹¹<https://machinelearningmastery.com/practical-guide-to-gan-failure-modes/>



(a) Distribuzione della feature BwdPacketLengthStd Botnet con alpha=0,01.
(b) Distribuzione della feature BwdPacketLengthStd Botnet con alpha=0,15.

Figura 13: Confronto tra distribuzioni della feature BwdPacketLengthStd per la classe Botnet nelle due configurazioni di iperparametri.

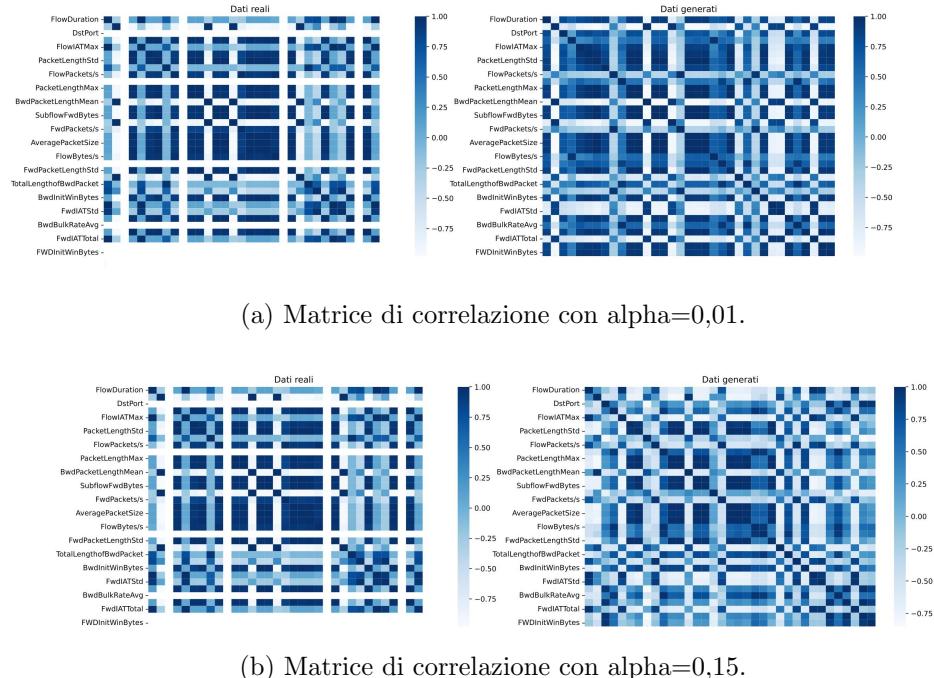


Figura 14: Confronto tra matrici di correlazione della classe Botnet nelle due configurazioni di iperparametri.

Distance ed MSE sono relativi alla feature BwdPacketLengthStd presa in esame nei grafici.

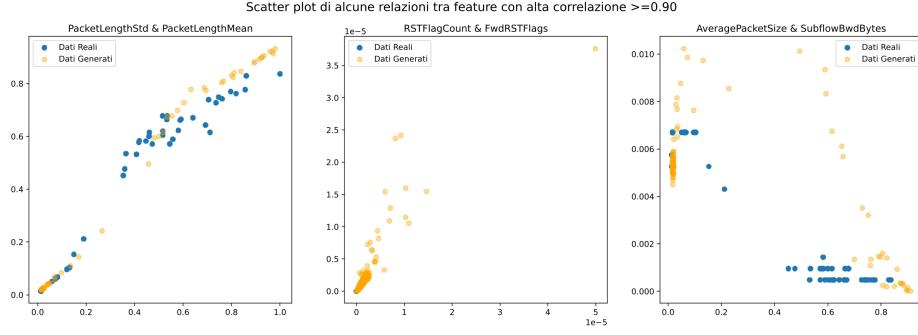
	Alpha=0.01	Alpha=0.15
Wasserstein Distance totale	0.003555	0.005616
Wasserstein Distance	0.000544	0.000426
Mean Squared Error	5.883532705408811e-06	5.00902869601238e-06
Punteggio SDV Qualità	42.32%	51.29%
Punteggio SDV Validità	80.01%	78.55%

Tabella 6: Analisi statistiche sui dati generati di tipo Botnet nelle due configurazioni di iperparametri.

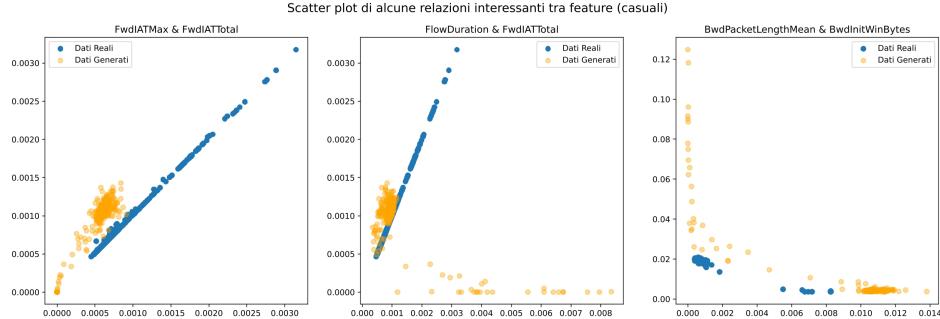
Gli scatter plot utilizzati per l'analisi di tutte le classi prendono in considerazione sei relazioni tra feature. Le prime tre coinvolgono feature ad alta correlazione ≥ 0.90 e le altre tre sono state scelte casualmente tra quelle ritenute interessanti: “FwdIATMax & FwdIATTot” con correlazione=0.66; “FlowDuration & FwdIAT-Total” con correlazione=1.00 e “BwdPacketLengthMean & BwdInitWinBytes” con correlazione=0.00.

Dagli scatter plot in figura 15 si nota come i dati generati della classe Botnet dal modello GAN siano leggermente migliori per alpha=0.15 (15c e 15d) in quanto si evidenziano similitudini più distinte tra i dati generati (in arancione) e quelli reali (in azzurro); volendo perciò rilevare una generazione quanto più vicina a quella dei reali. Tuttavia, la generazione di 10000 samples invece presenta leggere differenze mantenendo però un soddisfacente esito: i valori statistici, presenti in tabella 7, rimangono molto vicini a quelli con i soli 515 samples e dall'analisi qualitativa con scatter plot in figura 16 si nota un mantenimento dell'andamento dei dati nel dataset di riferimento, ovvero quello reale.

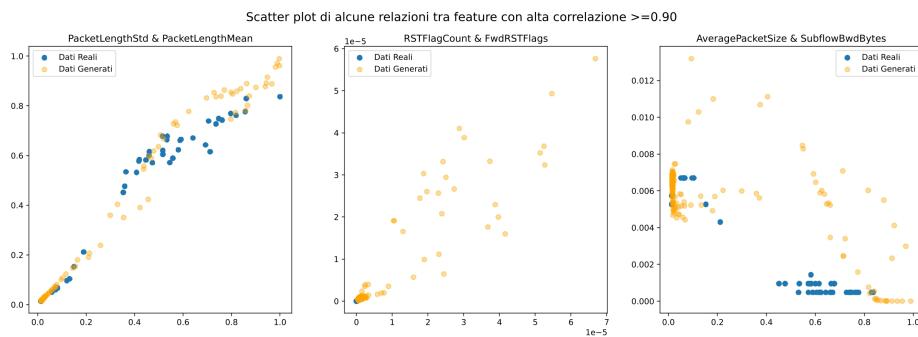
Si considerano perciò entrambe le configurazioni adatte per la generazione dei dati della classe Botnet, ma si conviene che da un'analisi visiva i dati generati con alpha=0.15 siano migliori seppur in maniera molto sottile.



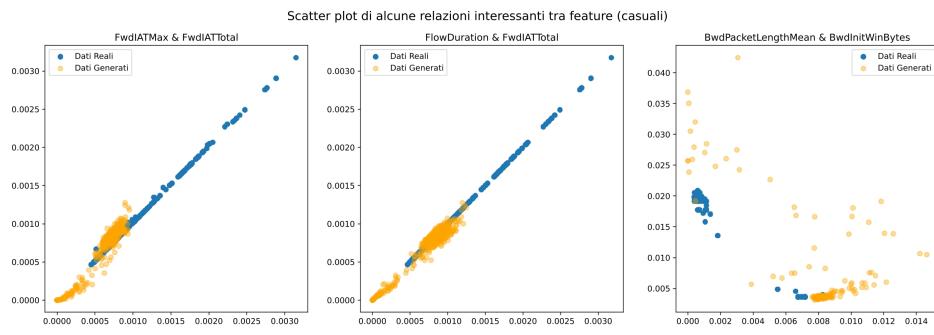
(a) Scatter plot con alpha=0,01 con alta correlazione.



(b) Scatter plot con alpha=0,01 casuali.

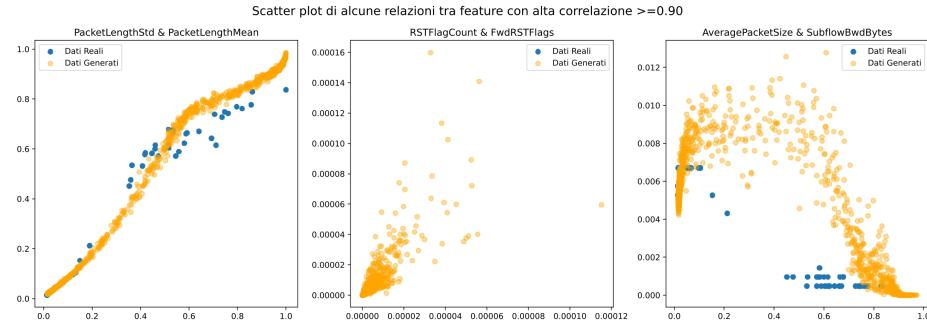


(c) Scatter plot con alpha=0,15 con alta correlazione.

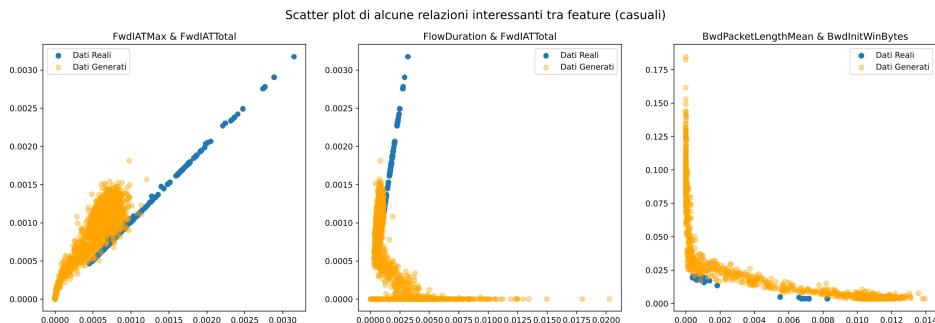


(d) Scatter plot con alpha=0,15 casuali.

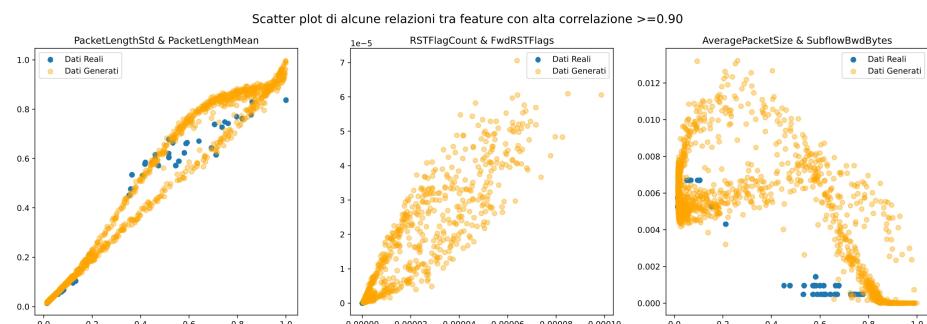
Figura 15: Confronto tra scatter plot dei dati generati e dei dati reali della classe Botnet nelle due configurazioni di iperparametri.



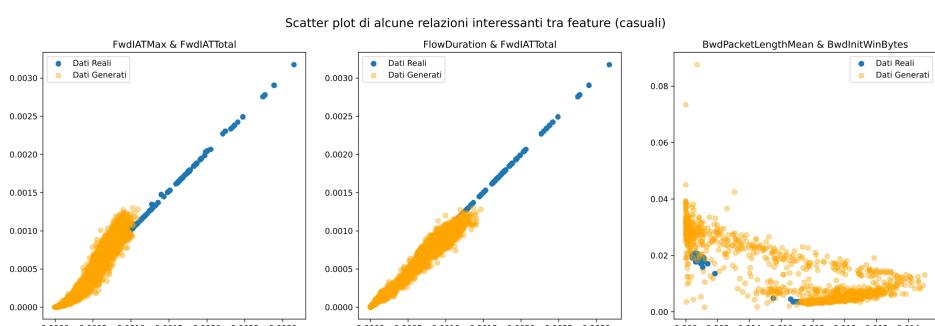
(a) Scatter plot con alpha=0,01 con alta correlazione.



(b) Scatter plot con alpha=0,01 casuali.



(c) Scatter plot con alpha=0,15 con alta correlazione.



(d) Scatter plot con alpha=0,15 casuali.

Figura 16: Confronto tra scatter plot della classe Botnet nelle due configurazioni di iperparametri per la generazione di 10000 samples.

	Alpha=0.01	Alpha=0.15
Wasserstein Distance totale	0.003537	0.002807
Wasserstein Distance	0.000540	0.000451
Punteggio SDV Validità	80.08%	78.86%

Tabella 7: Analisi statistiche sui 10000 dati generati di tipo Botnet nelle due configurazioni di iperparametri.

Relativamente alla seconda classe con minor numero di samples, ovvero la classe Bruteforce con 4904 samples nel Train Set, è emerso dalla generazione di dati tramite GAN che l'iperparametro alpha=0,15 risulta essere migliore considerando il confronto visivo per mezzo di matrice di correlazione in figura 17, dove appunto si evidenzia come il grafico 17a di destra dei dati generati riproponga più dettagliatamente le caratteristiche peculiari della matrice dei reali di sinistra rispetto al 17b. Inoltre, analizzando le feature FlowDuration e BwdPacketLengthStd è possibile notare come per entrambi i valori scelti per l'iperparametro alpha vengano generati dati con delle buone distribuzioni simili ai reali, come visibile in figura 18, dove si nota una maggior similitudine per alpha=0,01. In tabella 8 vengono riportati i valori statistici calcolati dove (0) si intende la feature FlowDuration ed, invece, (1) la feature BwdPacketLengthStd. Da quest'ultima si evince che alpha=0,01 risulti essere leggermente migliore se non equiparabile alla generazione con alpha=0,15.

	Alpha=0.01	Alpha=0.15
Wasserstein Distance totale	0.001210	0.001982
Wasserstein Distance (0)	0.002904	0.004601
Mean Squared Error (0)	0.001648	0.001036
Wasserstein Distance (1)	0.002650	0.003376
Mean Squared Error (1)	0.000458	0.000465
Punteggio SDV Qualità	72.39%	71.95%
Punteggio SDV Validità	94.80%	91.01%

Tabella 8: Analisi statistiche sui dati generati di tipo Bruteforce nelle due configurazioni di iperparametri.

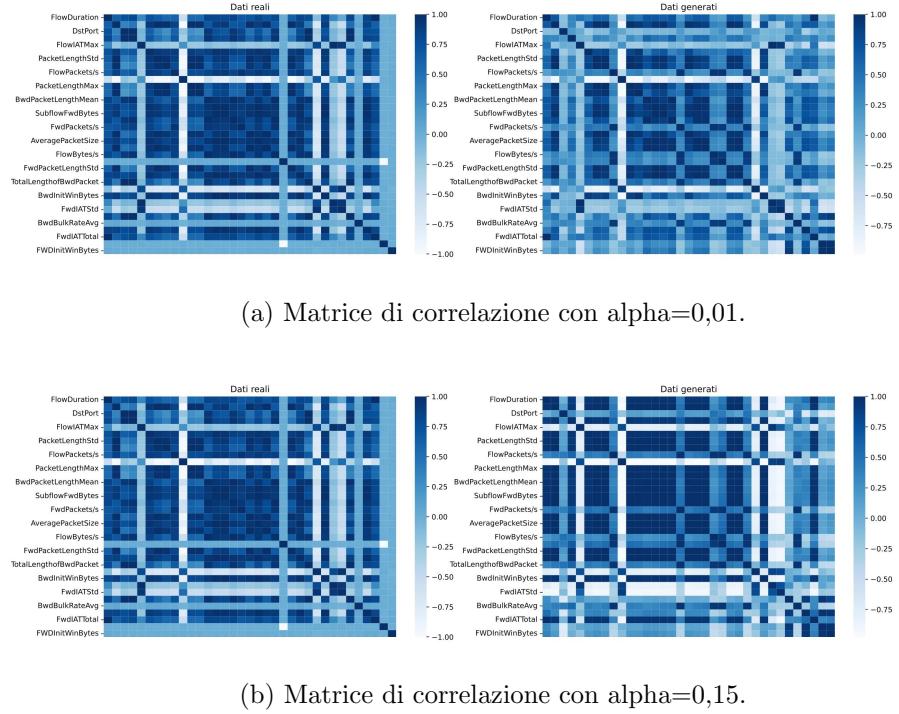


Figura 17: Confronto tra matrici di correlazione della classe Bruteforce nelle due configurazioni di iperparametri.

Dagli scatter plot in figura 19, invece, si nota come i dati generati siano tendenzialmente migliori con $\alpha=0,01$ in quanto i dati generati (in arancione) ripropongono un pattern simile a quello dei reali rispetto ai grafici per $\alpha=0,15$.

La generazione di 10000 samples presenta, invece, leggere differenze mantenendo le caratteristiche relative alla generazione minima di samples: dall’analisi statistica in tabella 9 risulta essere lievemente migliore la generazione sfruttando $\alpha=0,01$; i grafici delle distribuzioni e delle matrici di correlazione risultano essere praticamente uguali ed, invece, dal confronto degli Scatter Plot in figura 20 si evince un miglior apprendimento delle relazioni con $\alpha=0,01$ dove anche qui i dati generati seguono il pattern dei dati reali, nonostante qualche difficoltà per due relazioni: $RSTFlagCount \& FwdRSTFlags$ e $BwdPacketLengthMean \& BwdInitWindBytes$.

Si considera perciò che la configurazione con $\alpha=0,01$ sia lievemente migliore considerando l’analisi effettuata.

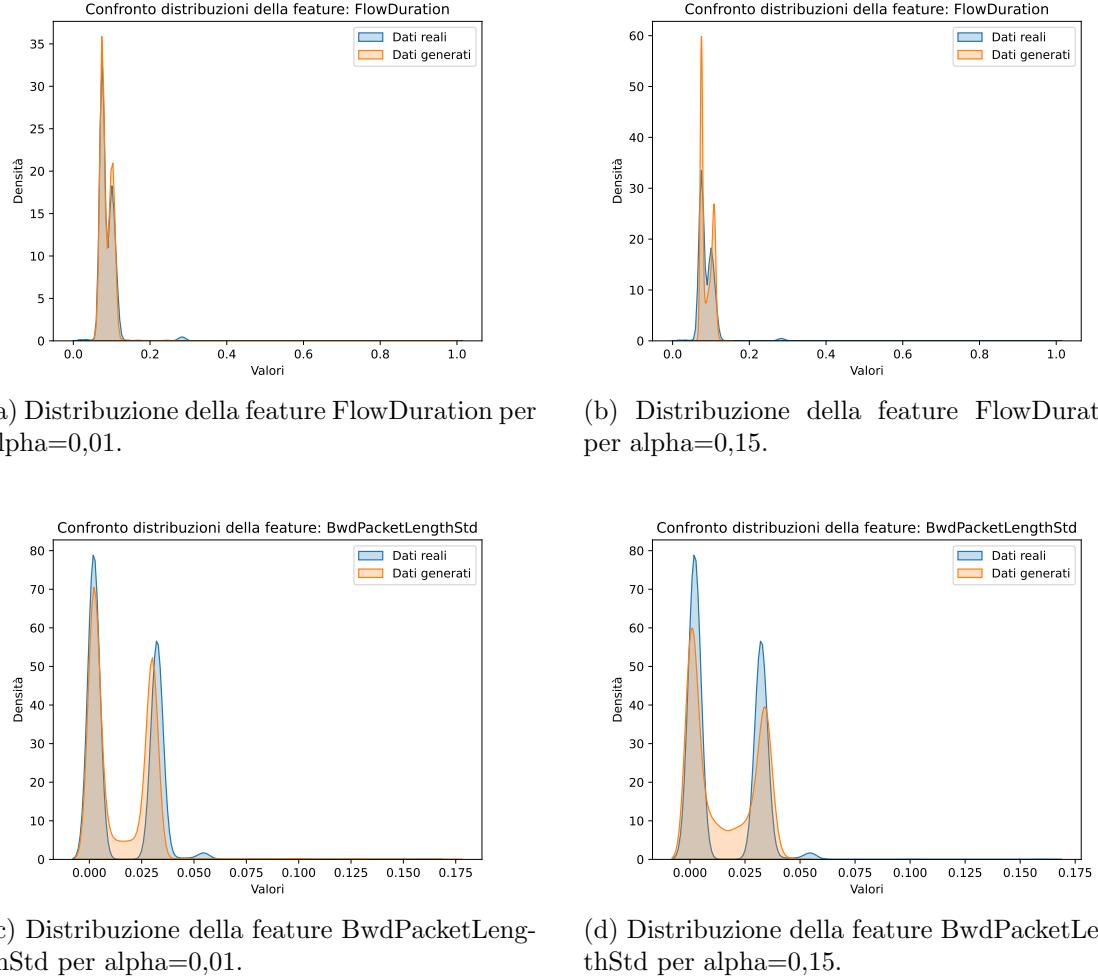


Figura 18: Confronto delle distribuzioni dei dati generati rispetto ai dati reali di due feature per la classe Bruteforce nelle due configurazioni di iperparametri.

	Alpha=0.01	Alpha=0.15
Wasserstein Distance totale	0.001217	0.001963
Wasserstein Distance (0)	0.003167	0.004439
Wasserstein Distance (1)	0.002649	0.003321
Punteggio SDV Validità	94.79%	90.84%

Tabella 9: Analisi statistiche sui 10000 dati generati di tipo Bruteforce nelle due configurazioni di iperparametri.

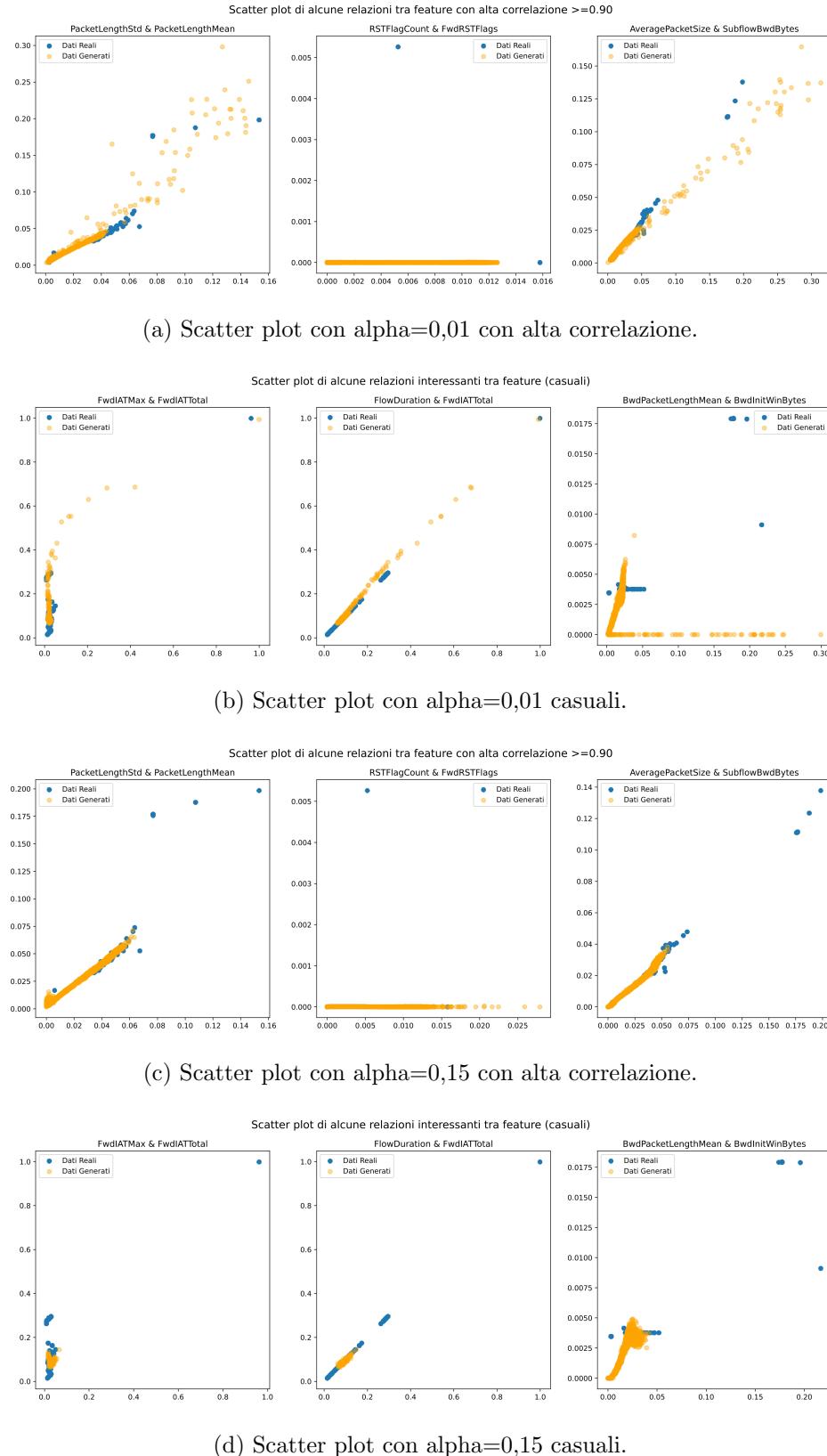


Figura 19: Confronto tra scatter plot della classe Bruteforce nelle due configurazioni di iperparametri.

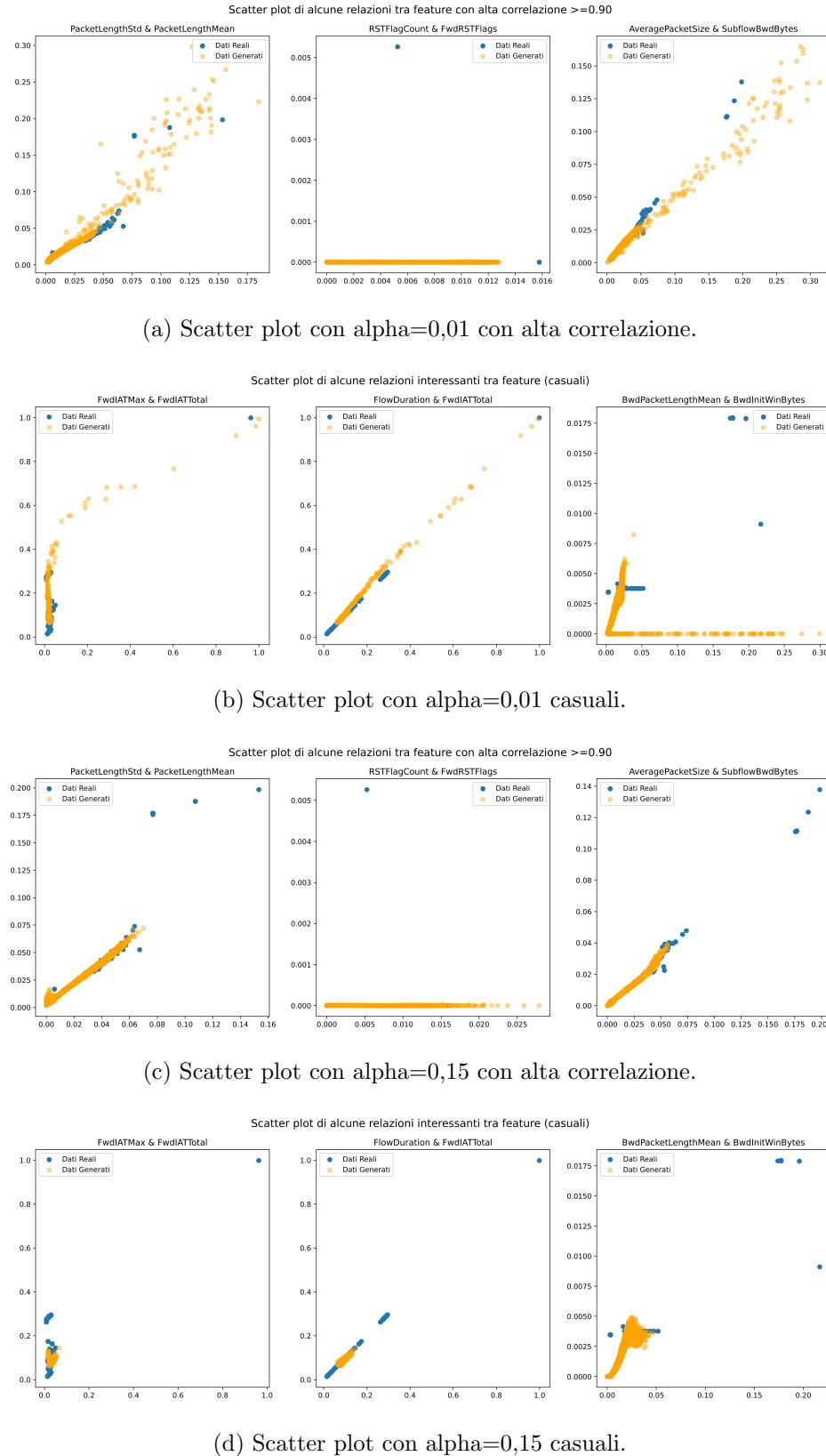
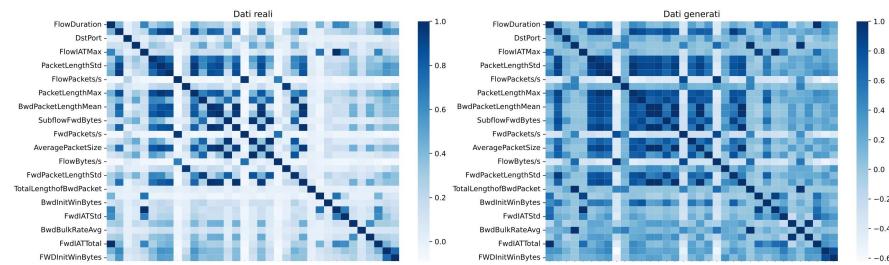


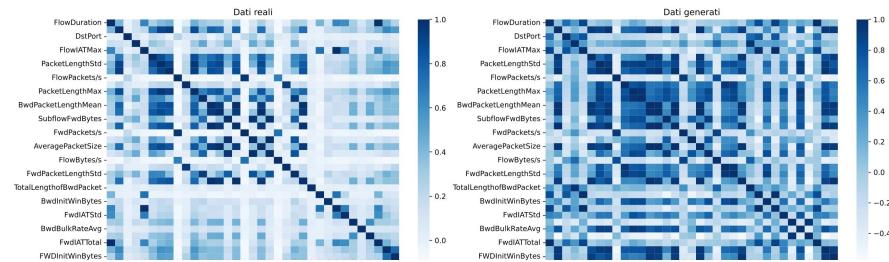
Figura 20: Confronto tra scatter plot della classe Bruteforce nelle due configurazioni di iperparametri per la generazione di 10000 samples.

Si conclude la valutazione dei dati generati con la classe Benign lasciando gli approfondimenti all'Appendice B. La classe di dati Benign, ovvero tutto il traffico benigno, risulta essere quella con la maggior quantità di samples e dall'analisi è evidente come la quantità elevata incida in maniera molto rilevante sull'apprendimento del modello GAN. Tale classe, infatti, risulta essere quella generata con i migliori risultati. La presenza di elevati dati non ha necessitato la generazione di una specifica quantità di dati (come i 10000 scelti per le due classi minori) e si è optato per generare la stessa quantità dei dati originali ai fini della valutazione. Innanzitutto, considerando la figura 22 si evidenzia come le distribuzioni dei dati reali e dei dati generati delle due feature coinvolte nel confronto siano molto simili soprattutto con $\alpha=0,01$ visibili in 22a e 22c. Ciò è visibile anche nei grafici delle matrici di correlazione in figura 21 nella quale si presenta una matrice dei dati generati a destra più simile a quella dei dati reali di sinistra e meglio rappresentata nel 21a con $\alpha=0,01$ rispetto alla 21b.

Inoltre, si vuole evidenziare i grafici Scatter Plot in figura 23. Anch'essi evidenziano che per $\alpha=0,01$ si ha una migliore generazione di dati, nonostante le imperfezioni presenti. Ciò lo si nota nei grafici 23a e 23b dove i dati generati (in arancione) risultano ricalcare il pattern dei dati reali del dataset in maniera molto più simile rispetto alla situazione presente per $\alpha=0,15$.



(a) Matrice di correlazione con alpha=0,01.



(b) Matrice di correlazione con alpha=0,15.

Figura 21: Confronto tra matrici di correlazione della classe Benign nelle due configurazioni di iperparametri.

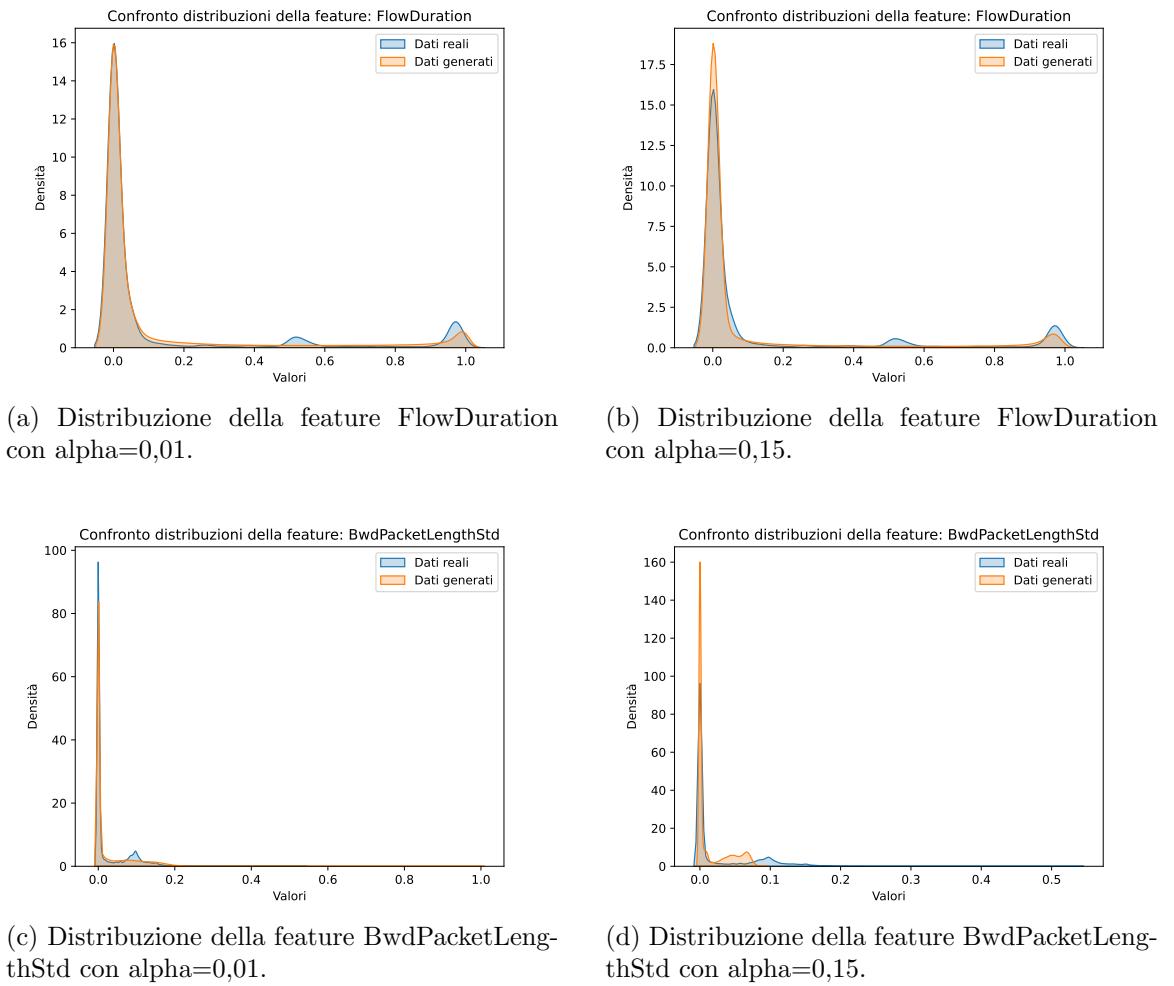


Figura 22: Confronto tra distribuzioni di due feature per la classe Benign nelle due configurazioni di iperparametri.

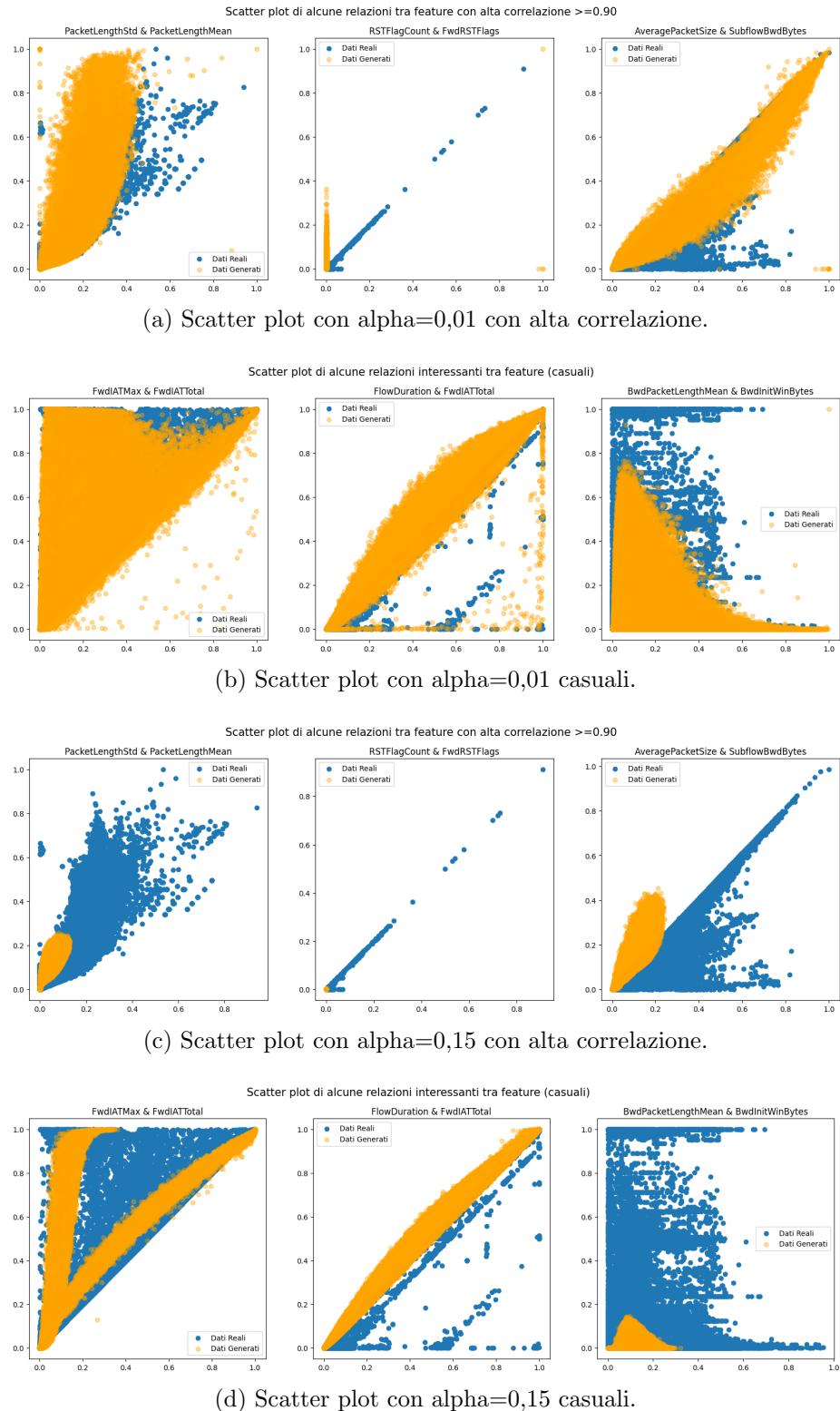


Figura 23: Confronto tra scatter plot della classe Benign nelle due configurazioni di iperparametri.

6.6 Sviluppo degli IDS e Data Augmentation

Per lo sviluppo del primo IDS è stato sfruttato il modello ANN di tipo MLP realizzato da Bacevicius et al. [73] settando il parametro *L2 Regularization* a 0.01. L’architettura è visibile in figura 24.

Riscontrando un’elevata difficoltà del modello preso in esame nell’apprendere le categorie di attacco con pochi dati si è deciso di applicare una miglior tecnica di Cross-Validation, sfruttando lo *splitting omogeneo* delle classi tra Train Set e Validation Set attraverso l’impiego della funzione *train_test_split stratificata* nonchè la tecnica del *Class Weighting* [78] bilanciato in base alla distribuzione delle classi nei dati.

Layer (type)	Output Shape	Param #
<hr/>		
dense (Dense)	(None, 64)	2304
dense_1 (Dense)	(None, 128)	8320
dense_2 (Dense)	(None, 128)	16512
dense_3 (Dense)	(None, 64)	8256
dense_4 (Dense)	(None, 6)	390
<hr/>		
Total params: 35782 (139.77 KB)		
Trainable params: 35782 (139.77 KB)		
Non-trainable params: 0 (0.00 Byte)		

Figura 24: Architettura del modello IDS-MLP

Numero delle Epoche	300
Validation Split	0.20
Early Stopping	10
Batch Size	1024
Learning Rate	0.001

Tabella 10: Iperparametri relativi alla fase di training del modello IDS-MLP.

In particolar modo, per la fase di addestramento sono stati impostati gli iperparametri riportati in tabella 10.

Si analizzano di seguito i risultati emersi con e senza la tecnica di Class Weighting.

IDS-MLP basico

Come si evince dai grafici in figura 25, dove viene rappresentato l'apprendimento per mezzo della Curva di Loss e di Accuratezza, si può constatare che esso avviene in maniera molto regolare e senza interrompersi nonostante all'Early Stopping, confermando così un corretto apprendimento senza overfitting. Si riscontrano però nei risultati di testing riportati in tabella 11 ed in figura 26, la quale riporta la performance di classificazione per ogni classe, che lo sbilanciamento dei dati comporta seri problemi di rilevamento degli attacchi con minori dati. Infatti, si può notare tale difficoltà di apprendimento delle due classi minori dalla matrice di confusione e le relative TP, TN, FP, FN in figura 27 e tabella 12. Infine, si conclude l'analisi con una valutazione delle metriche calcolate, visibili in tabella 13, che sottolineano come globalmente il modello sia in difficoltà nel classificare le classi minori.

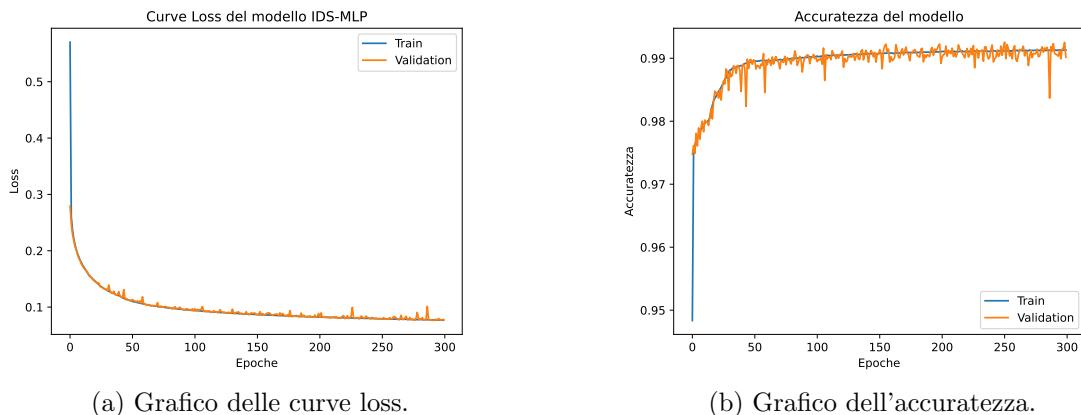


Figura 25: Grafici dell'apprendimento del IDS-MLP basilare.

Loss	0.0768
Test Loss	0.0765
Accuracy	0.9913
Test Accuracy	0.9905
<i>Balanced Accuracy</i>	0.6613

Tabella 11: Risultati della fase di testing del modello IDS-MLP.

	precision	recall	f1-score	support
0	1.00	1.00	1.00	478327
1	0.95	0.99	0.97	69256
2	1.00	0.99	0.99	28540
3	0.99	0.99	0.99	51486
4	0.00	0.00	0.00	2101
5	0.00	0.00	0.00	221
accuracy			0.99	629931
macro avg	0.66	0.66	0.66	629931
weighted avg	0.99	0.99	0.99	629931

Figura 26: Risultati di testing relativi alle performance dell'IDS-MLP.

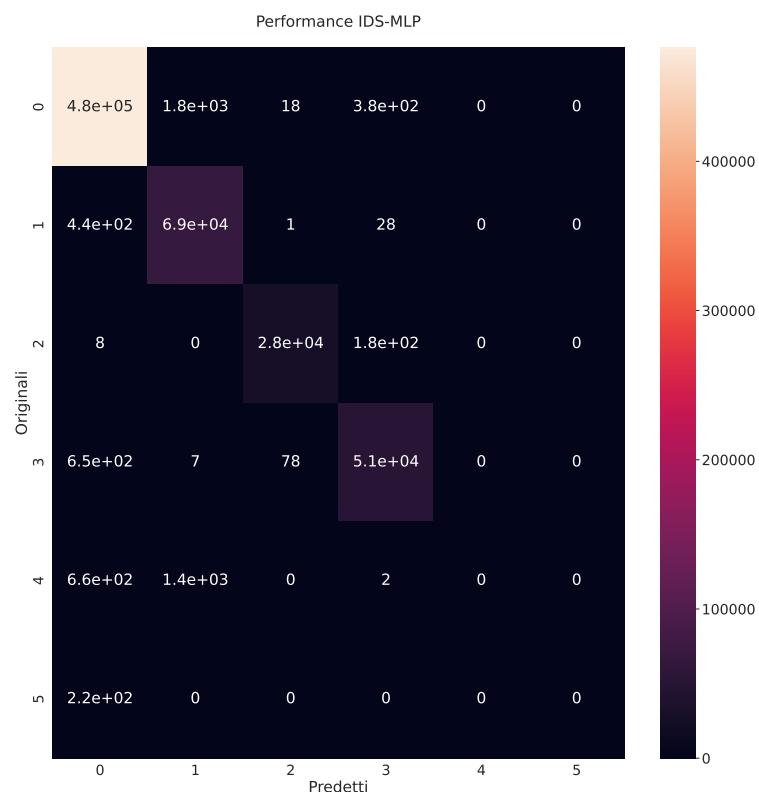


Figura 27: Grafico performance dell'IDS-MLP.

Classe	Metriche	Valore
BENIGN	TP	476082
	TN	149615
	FP	1989
	FN	2245
Portscan	TP	68783
	TN	557390
	FP	3285
	FN	473
DoS	TP	50749
	TN	577849
	FP	596
	FN	737
DDoS	TP	28350
	TN	601294
	FP	97
	FN	190
Bruteforce	TP	0
	TN	627830
	FP	0
	FN	2101
Botnet	TP	0
	TN	629710
	FP	0
	FN	221

Tabella 12: Valori TP, TN, FP, FN relativi alle performance dell'IDS-MLP.

Metrica di Valutazione	Valore misurato
Accuracy	0.990528
Balanced Accuracy	0.661251
Weighted Precision	0.987040
Macro Precision	0.655873
Weighted Recall	0.990528
Macro Recall	0.661251
Weighted F1 score	0.988741
Macro F1 score	0.658497

Tabella 13: Valutazione delle performance di classificazione dell'IDS-MLP.

IDS-MLP con tecnica di Class Weighting

L'applicazione della tecnica di Class Weighting evidenzia un apprendimento stabile, visibile in figura 28, che viene interrotto all'epoca 164 grazie all'EarlyStopping. Esso ottiene in fase di testing risultati più verosimili e con un miglior rilevamento delle classi minori: tali dati vengono riportati in tabella 14 e figura 29, dalla quale si riscontra la rilevazione delle due classi minori seppur con fatica. Si riportano per completezza dell'analisi la matrice di confusione e le relative TP, TN, FP, FN in figura 30 e tabella 15: essi confermano come la capacità di rilevare le classi minori sia migliorata, ad esempio con i TP=221 della classe Botnet che evidenziano un riconoscimento di tutti i samples del Test Set seppur con qualche difficoltà nel distinguere i dati di altre categorie. Infine, si conclude l'analisi riportando tutte le metriche calcolate, visibili in tabella 16.

Loss	0.1575
Test Loss	0.2103
Accuracy	0.9467
Test Accuracy	0.9539
<i>Balanced Accuracy</i>	0.9850

Tabella 14: Risultati della fase di testing del modello IDS-MLP con Class Weighting.

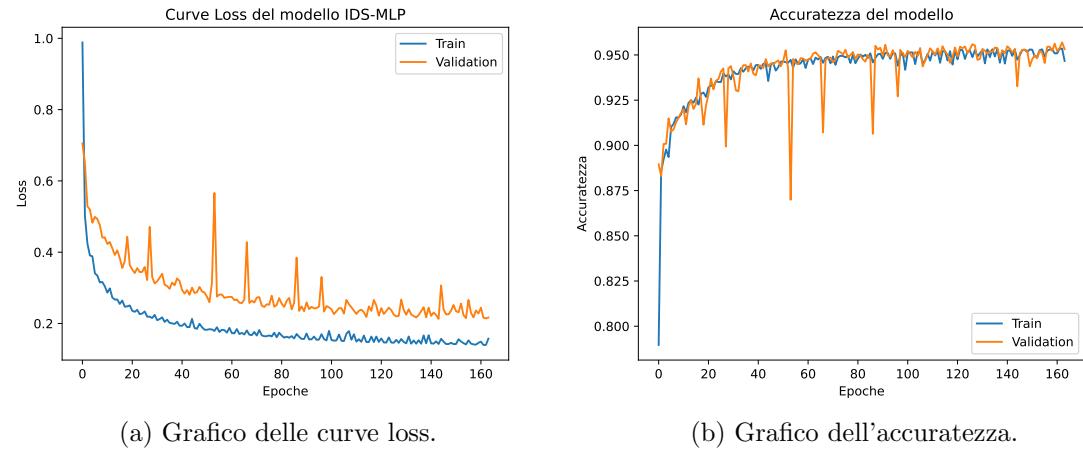


Figura 28: Grafici dell'apprendimento del IDS-MLP che sfrutta Class Weighting.

	precision	recall	f1-score	support
0	1.00	0.94	0.97	478327
1	0.89	0.99	0.94	69256
2	0.96	1.00	0.98	28540
3	0.89	0.98	0.93	51486
4	0.16	1.00	0.28	2101
5	0.11	1.00	0.20	221
accuracy			0.95	629931
macro avg	0.67	0.98	0.72	629931
weighted avg	0.97	0.95	0.96	629931

Figura 29: Risultati di testing relativi alle performance dell'IDS-MLP che sfrutta ClassWeighting.

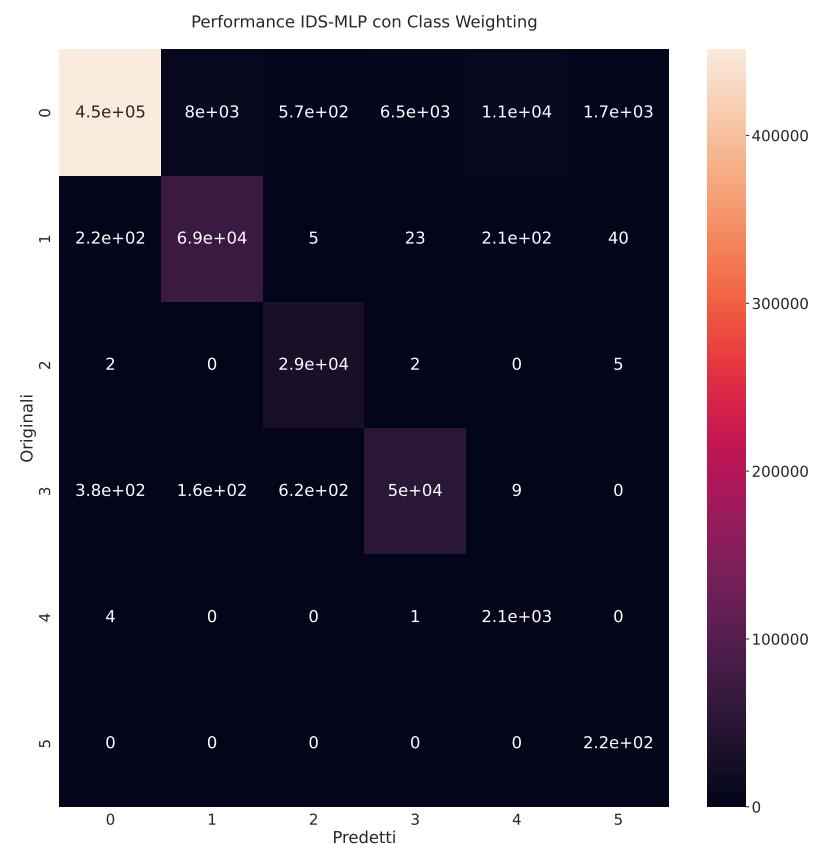


Figura 30: Grafico performance dell'IDS-MLP con Class Weighting.

Classe	Metriche	Valore
BENIGN	TP	450981
	TN	150991
	FP	613
	FN	27346
Portscan	TP	68755
	TN	552460
	FP	8215
	FN	501
DoS	TP	50304
	TN	571944
	FP	6501
	FN	1182
DDoS	TP	28531
	TN	600190
	FP	1201
	FN	9
Bruteforce	TP	2096
	TN	617060
	FP	10770
	FN	5
Botnet	TP	221
	TN	627967
	FP	1743
	FN	0

Tabella 15: Valori TP, TN, FP, FN relativi alle performance dell'IDS-MLP con Class Weighting.

Metrica di Valutazione	Valore misurato
Accuracy	0.953895
Balanced Accuracy	0.984991
Weighted Precision	0.972948
Macro Precision	0.668752
Weighted Recall	0.953895
Macro Recall	0.984991
Weighted F1 score	0.961196
Macro F1 score	0.716831

Tabella 16: Valutazione delle performance di classificazione dell'IDS-MLP con Class Weighting.

Secondo IDS: Random Forest Classifier

Per lo sviluppo del secondo tipo di IDS è stata impiegata una Random Forest [79]: in particolare il modello *Random Forest Classifier* della libreria *sklearn*, di seguito chiamato IDS-RFC. Le RF permettono di combinare più alberi decisionali al fine di aumentarne l'accuratezza, riducendo overfitting e migliorando la generalizzazione del modello stesso.

Si è utilizzato del codice¹² di partenza per poter studiare la configurazione definita dagli autori Bulavas et al. [80]: questi ultimi hanno sfruttato lo stesso dataset impiegato per questo studio ma nella sua versione originale, ovvero il CIC-IDS-2017, per analizzare come lo sbilanciamento dei dati influisca sulle performance di diversi algoritmi di classificazione tra cui RFC, CART, KNN ed altri; dove RFC risulta essere dall'analisi uno dei migliori metodi. Si precisa però che il pre-processing eseguito dai ricercatori risulta essere diverso da quello messo in atto in questo studio. I parametri della configurazione vengono riportati in tabella 17.

¹²<https://github.com/noushinpervez/Intrusion-Detection-CIC-IDS2017/blob/main/Intrusion-Detection-CIC-IDS2017.ipynb>

criterion	entropy
class_weight	balanced
min_samples_leaf	7
n_estimators	120
max_depth	15
max_features	0,5
random_state	0

Tabella 17: Configurazione di parametri del modello RFC usato come secondo IDS.

Classi	Samples Test Set	Samples predetti
BENIGN	478327	478277
Infiltration_Portscan	69256	69307
DoS	51486	51486
DDoS	28540	28540
BruteForce	2101	2100
Botnet	221	221

Tabella 18: Confronto Test Set e Samples predetti dall'IDS-RFC.

Metrica di Valutazione	Valore misurato
Accuracy	0.999878
Balanced Accuracy	0.999789
Weighted Precision	0.999881
Macro Precision	0.999763
Weighted Recall	0.999879
Macro Recall	0.999789
Weighted F1 score	0.999879
Macro F1 score	0.999776

Tabella 19: Valutazione delle performance di classificazione dell'IDS-RFC.

Le performance di tale IDS possono esser ritenute più che soddisfacenti se non ottime fin da subito, infatti, si può notare come in tabella 18 vengano predetti con un'ottima percentuale quasi tutti i samples del Test Set. Inoltre, si può vedere chiaramente anche dalle metriche di valutazione calcolate in tabella 19. Si riportano per completezza dell'analisi la matrice di confusione e le relative TP, TN, FP, FN in figura 32 e tabella 20 nonché il report di classificazione in figura 31.

	precision	recall	f1-score	support
0	1.00	1.00	1.00	478327
1	1.00	1.00	1.00	69256
2	1.00	1.00	1.00	28540
3	1.00	1.00	1.00	51486
4	1.00	1.00	1.00	2101
5	1.00	1.00	1.00	221
micro avg	1.00	1.00	1.00	629931
macro avg	1.00	1.00	1.00	629931
weighted avg	1.00	1.00	1.00	629931
samples avg	1.00	1.00	1.00	629931

Figura 31: Risultati di testing relativi alle performance dell'IDS-RFC.

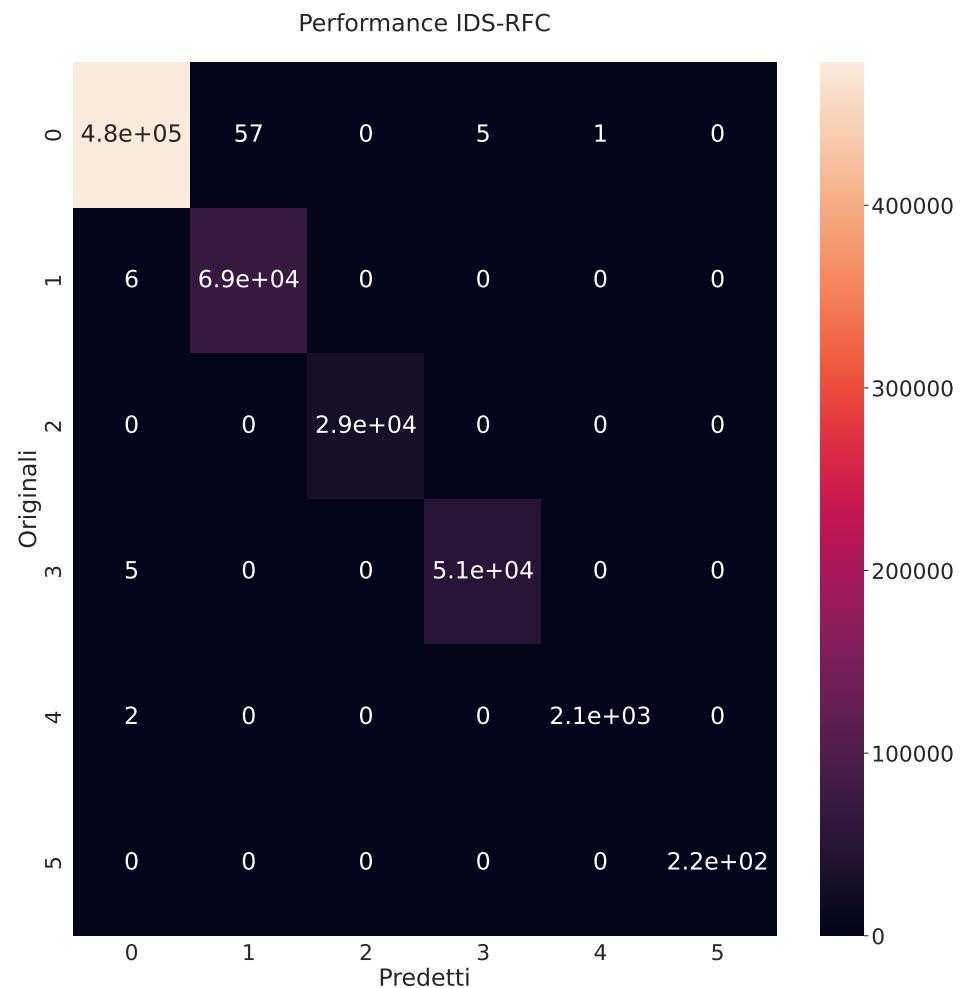


Figura 32: Grafico performance dell'IDS-RFC.

Classe	Metriche	Valore
BENIGN	TP	478264
	TN	151591
	FP	13
	FN	63
Portscan	TP	69250
	TN	560618
	FP	57
	FN	6
DoS	TP	51481
	TN	578440
	FP	5
	FN	5
DDoS	TP	28540
	TN	601391
	FP	0
	FN	0
Bruteforce	TP	2099
	TN	627829
	FP	1
	FN	2
Botnet	TP	221
	TN	629710
	FP	0
	FN	0

Tabella 20: Valori TP, TN, FP, FN relativi alle performance dell'IDS-RFC.

Per concludere l’analisi dell’IDS-RFC si riportano i valori del Training e Test Accuracy nonchè quelli relativi al K-Cross Validation Accuracy con K=5 sul Train Set in modo tale da confermare che le ottime performance del modello non siano dovute ad overfitting: tali misurazioni sono visibili in tabella 21. Dall’esame emerge perciò che non vi è alcun overfitting, confermato anche dai grafici delle Curve di Apprendimento in figura 33, nel quale si sono sfruttati il 20%, 40%, 60%, 80%, ed il 100% dei dati di training per bilanciare il costo computazionale della generazione del grafico e la profondità dell’analisi. Per generare tali grafici è stata impiegata la funzione *learning_curve* della libreria *scikit-learn*, che permette di capire come si comporta il modello all’aumentare dei samples nella fase di training. In figura 36a è stato settato il parametro “cv” coinvolto dalla funzione, che rappresenta la strategia di cross-validation, a cv=2 per ridurre il tempo di generazione del grafico ed avere una prima analisi con un valore minimo, per poi passare a cv=5 (valore di default, ovvero 5-fold cross validation) in figura 36b per avere così un’analisi completa: entrambe le curve convergono a valori alti, sintomo di una buona performance senza overfitting, nonostante si evidenzi instabilità nel grafico con cv=5 sull’accuratezza del Cross-Validation.

Metrica di Valutazione	Valore
Training Accuracy	0.999905
Test Accuracy	0.999879
K Cross-Validation Accuracy	0.999857 0.999840 0.999820 0.999898 0.999850
Average Cross-Validation Accuracy	0.999853

Tabella 21: Analisi della presenza di overfitting nell’IDS-RFC.

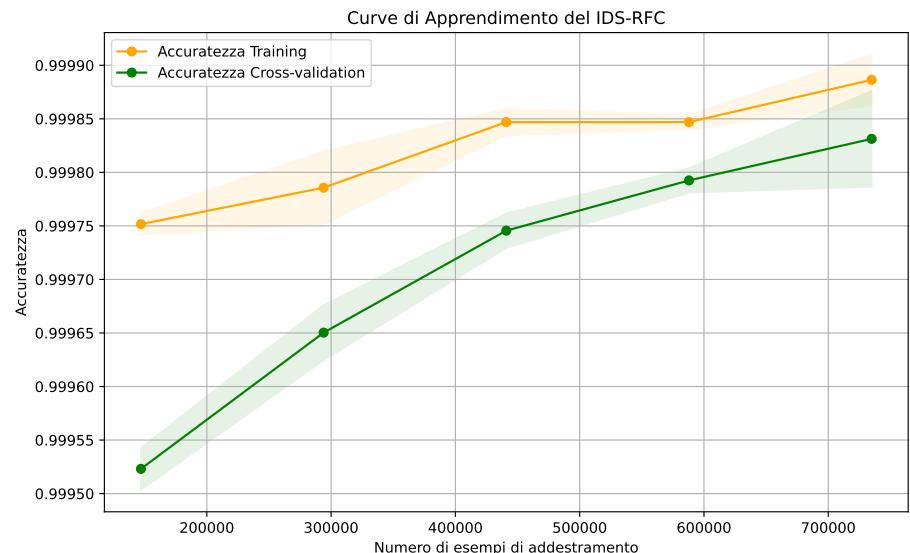
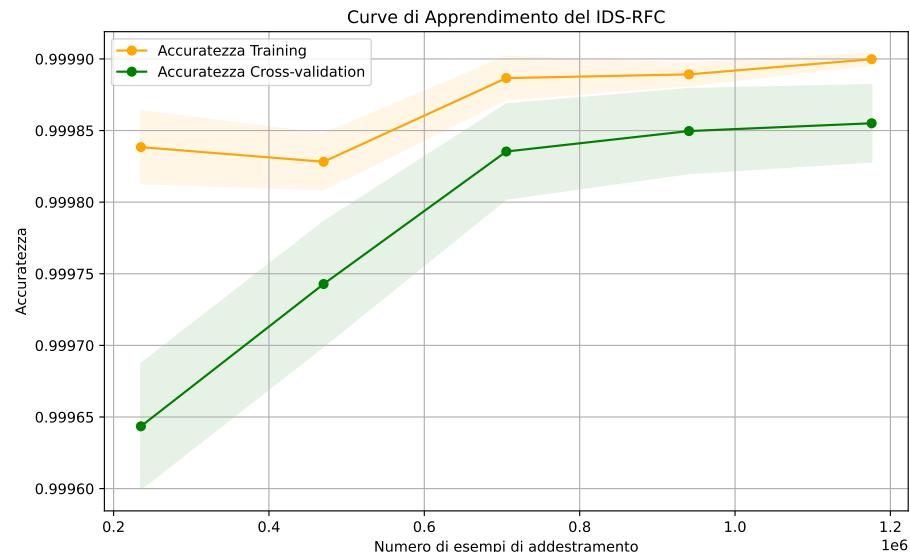
(a) Curve di Apprendimento con $cv=2$.(b) Curve di Apprendimento con $cv=5$ (default).

Figura 33: Curve di Apprendimento dell'IDS-RFC.

6.6.1 Valutazione della Data Augmentation

La Data Augmentation (di seguito D.A.) è stata eseguita in diverse modalità su entrambi gli IDS al fine di valutare sia la generazione dei dati da parte della GAN sia le performance degli IDS focalizzandosi sul migliorare la classificazione delle classi maggiormente sbilanciate. Innanzitutto, si è deciso di analizzare il comportamento dei due set di iperparametri separatamente per capire se i risultati riscontrati nel capitolo 6.5.1 venissero confermati. La strategia di Data Augmentation adottata è stata la seguente: una prima modalità è stata quella di fare un incremento del 100% ovvero raddoppiando le quantità già presenti per classe nel train dataset; successivamente si è valutato un incremento di 10000 samples per le classi minori Botnet e Bruteforce in modo tale da mantenere la stessa proporzione rispetto alle classi con maggiori dati: si vedano tabelle 22, 23. Tale decisione è stata presa in seguito allo studio di diverse ricerche accademiche [80, 81, 82, 83, 84] dalle quali però non emerge alcun modus operandi comune di incremento dei dati (hanno utilizzato incrementi del 100%, incrementi fissi oppure riduzione delle classi maggiori ed incremento casuale), perciò si è optato per questa strategia di D.A. che potesse incidere significativamente sulle performance.

Tali modalità sono:

1. *Data Augmentation* dapprima con alpha=0,01 e poi con alpha=0,15
 - (a) Generazione del doppio di samples presenti
 - i. Generazione solo delle classi minori
 - ii. Generazione di tutte le classi
 - (b) Generazione di 10000 samples per Botnet e Bruteforce
 - i. Generazione solo delle classi minori
 - ii. Generazione di tutte le classi

Di seguito viene effettuata la valutazione delle performance: eventuali approfondimenti vengono riportati in Appendice C.

	Train	Train+Generati100%	Train+Generati100% Classi Minori
Benign	1116095	2232190	1116095
Portscan	161597	323194	161597
DoS	120134	240268	120134
DDoS	66593	133186	66593
Bruteforce	4904	9808	9808
Botnet	515	1030	1030

Tabella 22: Samples per classe nella Data Augmentation con incremento del 100%.

	Train	Train+Generati10000	Train+Generati10000 Classi Minori
Benign	1116095	2232190	1116095
Portscan	161597	323194	161597
DoS	120134	240268	120134
DDoS	66593	133186	66593
Bruteforce	4904	14904	14904
Botnet	515	10515	10515

Tabella 23: Samples per classe nella Data Augmentation con incremento di 10000 samples per Botnet e Bruteforce.

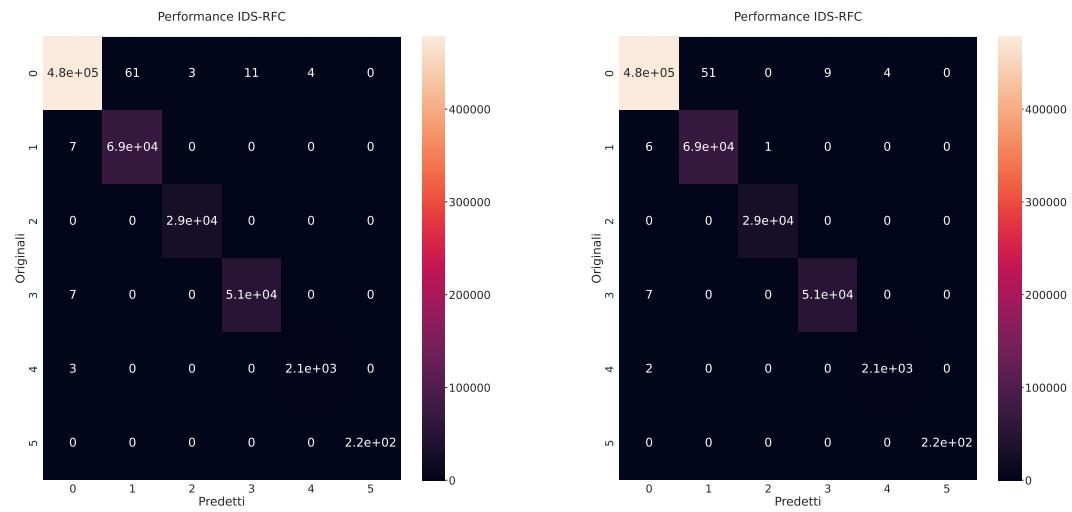
IDS-RFC

Una prima analisi viene fatta sulla generazione di tutte le classi nei due diversi set di iperparametri con un'aumento del 100%.

In figura 34 si nota una performance leggermente migliore per alpha=0,15. Si specifica che le classi mostrate sull'asse y di questo e dei futuri grafici delle matrici di confusione mostrati sono: Benign=0, Portscan=1, DDoS=2, DoS=3, Bruteforce=4 e Botnet=5. Inoltre non si è ritenuto rilevante includere i report di classificazione in quanto tutti uguali e tutti con esito ottimo, nonostante si mostreranno di seguito leggeri problemi di FP e FN.

Rispetto alla versione originale senza D.A, visibile nel capitolo 6.6 alla tabella 32, si evidenzia un leggero peggioramento nell'applicare tale tecnica. Si ritiene che questo peggioramento sia attendibile a causa della generazione non troppo brillante da

parte della Vanilla GAN, che non impatta drasticamente sulle performance generali dell'IDS-RFC. Si può perciò affermare che, nonostante la tecnica di D.A. non sia necessaria per l'IDS-RFC preso in esame, i dati generati dalla Vanilla GAN possono esser considerati accettabili perché non riducono radicalmente le prestazioni di classificazione degli attacchi. Si conclude che tale esame conferma la valutazione del modello GAN nel capitolo 6.5.1.

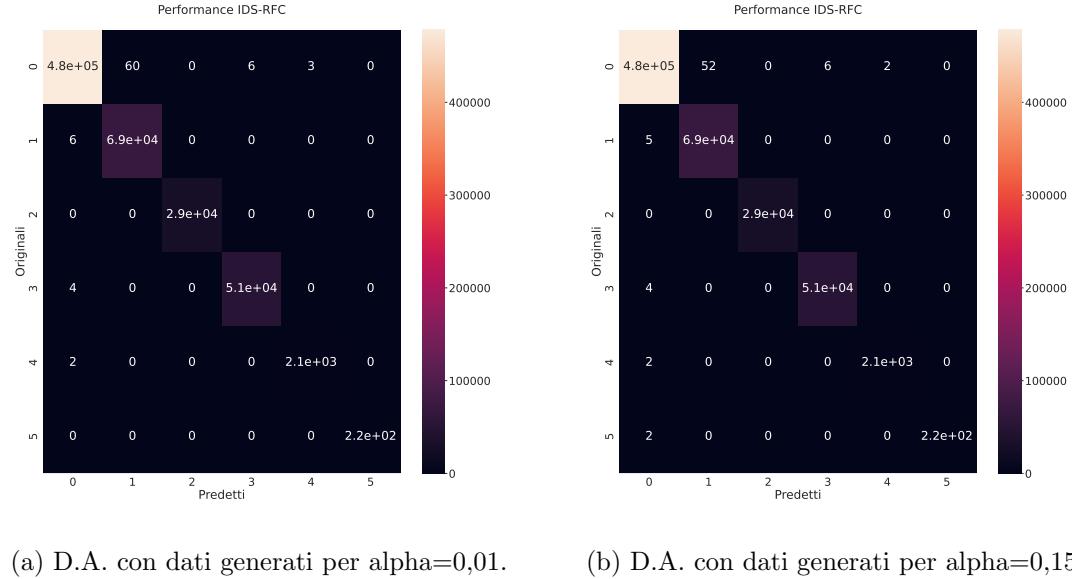


(a) D.A. con dati generati per alpha=0,01.

(b) D.A. con dati generati per alpha=0,15.

Figura 34: Confronto D.A. del 100% di tutte le classi nei due diversi valori di alpha.

Segue l'analisi effettuata sulla generazione delle sole classi con minori samples con incremento del 100%, da cui emerge in figura 35 una migliore prestazione rispetto alla generazione di tutte le classi. Ciò viene ritenuto attendibile in quanto la generazione delle classi maggiori può esser ritenuta superflua ai fini del miglioramento della classificazione data appunto l'enorme quantità di dati già presenti per il training, ma rilevante per la valutazione della GAN. Inoltre, la D.A. che sfrutta alpha=0,01 risulta migliore per la classificazione della classe Botnet. Rispetto al IDS-RFC senza D.A. si nota un leggero peggioramento delle due classi minori aumentate, soprattutto per la classe Botnet con alpha=0,15.



(a) D.A. con dati generati per alpha=0,01.

(b) D.A. con dati generati per alpha=0,15.

Figura 35: Confronto D.A. del 100% delle sole classi Botnet e Bruteforce nei due diversi valori di alpha.

Invece, per quanto riguarda la D.A. delle sole classi minori aumentate di 10000 samples si ha anche in questo caso una riduzione delle prestazioni rispetto all'originale.

Tale aumento comporta delle difficoltà, evidenziate in figura 36, da cui si può affermare che non risulta necessaria la D.A. per il modello in esame ma rimane essere comunque migliore dell'aumento di tutte le classi come già ampiamente affermato precedentemente.

In conclusione, si evidenzia che la Data Augmentation non risulta essere efficace per il modello IDS-RFC. Si nota un leggero peggioramento che non incide in maniera sostanziale sulle performance di classificazione come atteso a causa della non perfetta generazione dei dati da parte della Vanilla GAN impiegata. Tra le diverse modalità applicate si nota che il solo aumento delle classi minori da' migliori risultati come da attese, ma non incide nel migliorare la classificazione e invece comportando qualche minima difficoltà.

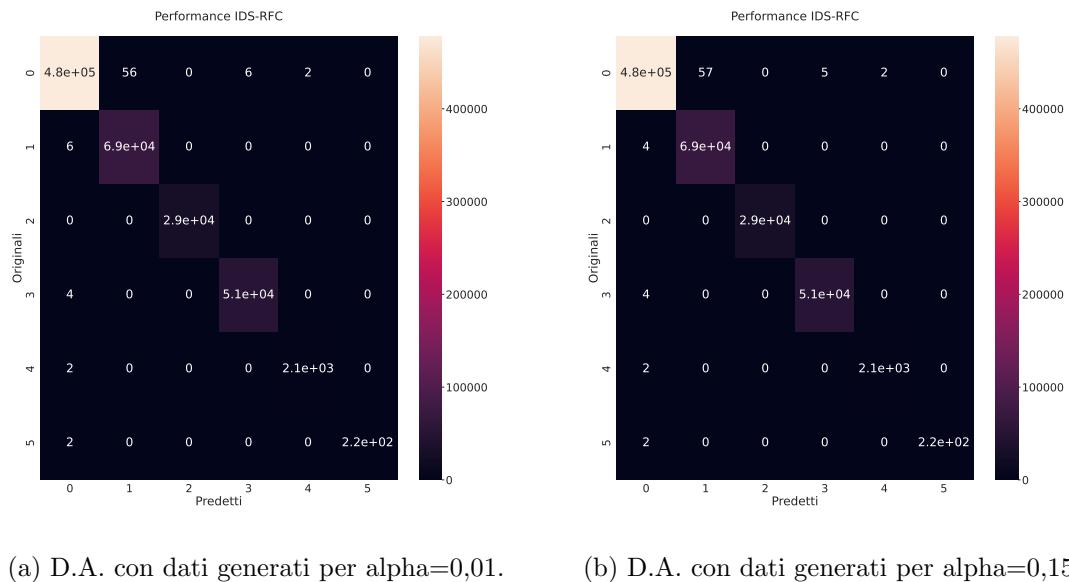


Figura 36: Confronto D.A. con l'aumento di 10000 samples, senza aumentare le classi maggiori, nei due diversi valori di alpha.

IDS-MLP

La Data Augmentation applicata all'IDS-MLP, invece, offre buoni risultati andando a migliorare le performance del modello.

Vengono analizzate separatamente le prestazioni del IDS-MLP con e senza Class Weighting (di seguito CW) ed allo stesso tempo confrontando l'impiego dei due valori di alpha. In particolare, si sfrutteranno Matrici di Confusioni, metriche standard per la valutazione e Report di Classificazione. Eventuali Grafici delle Curve Loss e di Accuratezza vengono riportate in appendice C.

Per quanto riguarda la Data Augmentation applicata a tutte le classi con aumento del 100% si vede dai Report di classificazione in figura 37 la presenza di problemi nel rilevare le classi minori per l'IDS-MLP senza l'utilizzo del Class Weighting ed, in particolare, in tabella 24 si possono osservare le metriche calcolate in dettaglio. Le matrici di confusione vengono riportate in appendice C in figura 61. Inoltre, si vuole sottolineare come nell'applicazione del CW si noti un lieve miglioramento nella rilevazione della classe Bruteforce per alpha=0,01 e quella Botnet per alpha=0,15 visibile in 37b e 37d, come affermato durante la valutazione dei dati generati al capitolo 6.5.1. Infine, si afferma che rispetto all'IDS-MLP originale senza CW si osserva un leggero miglioramento per le classi minori, ma con qualche problema di rilevamento delle classi maggiori: considerando, invece, la tecnica del CW si ha un miglioramento più evidente per alpha=0,15.

Di contro, se la D.A. viene applicata solo alle due classi minori sempre con aumento del 100% si ha comunque una performance migliore con l'applicazione del Class Weighting: ciò è visibile in tabella 25 ed in figura 38, nella quale emerge la rilevazione effettiva della classe Botnet. Questa modalità però risulta essere migliore rispetto alla generazione di tutti i dati come da attese ed, anzi, migliora le prestazioni in maniera molto evidente per alpha=0,15 con CW visibile in 38d. Per completezza si riportano le matrici di confusione dell'IDS-MLP a cui viene applicato il CW nelle due configurazioni di alpha in figura 39, il resto viene riportato in appendice C.

	per alpha=0,01		per alpha=0,15	
	senza CW	con CW	senza CW	con CW
Accuracy	0.988489	0.957216	0.984089	0.956419
Balanced Accuracy	0.725737	0.983769	0.657912	0.981689
Weighted Precision	0.988278	0.977965	0.980994	0.977302
Macro Precision	0.807876	0.675490	0.648686	0.678866
Weighted Recall	0.988489	0.957216	0.984089	0.956419
Macro Recall	0.725737	0.983768	0.657912	0.981689
Weighted F1 score	0.987974	0.965618	0.982380	0.964989
Macro F1 score	0.750531	0.713053	0.653039	0.715809

Tabella 24: Valutazione delle performance di classificazione dell'IDS-MLP applicando D.A. con e senza CW.

	precision	recall	f1-score	support		precision	recall	f1-score	support
0	0.99	0.99	0.99	478327	0	1.00	0.95	0.97	478327
1	0.94	0.99	0.97	69256	1	0.89	0.99	0.94	69256
2	1.00	0.99	1.00	28540	2	0.98	1.00	0.99	28540
3	1.00	0.96	0.98	51486	3	0.95	0.98	0.96	51486
4	0.92	0.41	0.57	2101	4	0.15	0.99	0.26	2101
5	0.00	0.00	0.00	221	5	0.08	1.00	0.16	221
accuracy			0.99	629931	accuracy			0.96	629931
macro avg	0.81	0.73	0.75	629931	macro avg	0.68	0.98	0.71	629931
weighted avg	0.99	0.99	0.99	629931	weighted avg	0.98	0.96	0.97	629931
(a) senza CW per alpha=0,01.					(b) con CW per alpha=0,01.				
	precision	recall	f1-score	support		precision	recall	f1-score	support
0	0.99	0.99	0.99	478327	0	0.99	0.95	0.97	478327
1	0.93	1.00	0.96	69256	1	0.90	0.99	0.94	69256
2	0.98	1.00	0.99	28540	2	0.97	1.00	0.98	28540
3	0.99	0.97	0.98	51486	3	0.96	0.95	0.96	51486
4	0.00	0.00	0.00	2101	4	0.14	1.00	0.25	2101
5	0.00	0.00	0.00	221	5	0.11	1.00	0.19	221
accuracy			0.98	629931	accuracy			0.96	629931
macro avg	0.65	0.66	0.65	629931	macro avg	0.68	0.98	0.72	629931
weighted avg	0.98	0.98	0.98	629931	weighted avg	0.98	0.96	0.96	629931
(c) senza CW per alpha=0,15.					(d) con CW per alpha=0,15.				

Figura 37: Confronto sui Report di Classificazione della D.A. di tutte le classi su IDS-MLP con e senza CW, nei due diversi valori di alpha.

	per alpha=0,01		per alpha=0,15	
	senza CW	con CW	senza CW	con CW
Accuracy	0.994380	0.957335	0.993758	0.974235
Balanced Accuracy	0.825905	0.983859	0.780872	0.988808
Weighted Precision	0.994071	0.976117	0.993411	0.978966
Macro Precision	0.817574	0.680454	0.828800	0.743454
Weighted Recall	0.994380	0.957335	0.993758	0.974235
Macro Recall	0.825905	0.983859	0.780872	0.988808
Weighted F1 score	0.994215	0.964769	0.993501	0.975714
Macro F1 score	0.821650	0.724721	0.801605	0.806059

Tabella 25: Valutazione delle performance di classificazione dell'IDS-MLP applicando D.A. con e senza CW solo su Botnet e Brutforce.

	precision	recall	f1-score	support		precision	recall	f1-score	support
0	1.00	1.00	1.00	478327	0	1.00	0.95	0.97	478327
1	0.98	0.99	0.99	69256	1	0.89	0.99	0.94	69256
2	0.99	1.00	1.00	28540	2	0.97	1.00	0.99	28540
3	0.99	0.98	0.99	51486	3	0.94	0.97	0.95	51486
4	0.94	0.99	0.96	2101	4	0.16	1.00	0.27	2101
5	0.00	0.00	0.00	221	5	0.13	1.00	0.23	221
accuracy			0.99	629931	accuracy			0.96	629931
macro avg	0.82	0.83	0.82	629931	macro avg	0.68	0.98	0.72	629931
weighted avg	0.99	0.99	0.99	629931	weighted avg	0.98	0.96	0.96	629931
(a) senza CW per alpha=0,01.					(b) con CW per alpha=0,01.				
	precision	recall	f1-score	support		precision	recall	f1-score	support
0	0.99	1.00	1.00	478327	0	1.00	0.97	0.98	478327
1	0.99	0.99	0.99	69256	1	0.91	0.99	0.95	69256
2	1.00	0.99	1.00	28540	2	0.97	1.00	0.99	28540
3	0.99	0.97	0.98	51486	3	0.93	0.98	0.95	51486
4	0.99	0.74	0.85	2101	4	0.43	1.00	0.60	2101
5	0.00	0.00	0.00	221	5	0.23	1.00	0.37	221
accuracy			0.99	629931	accuracy			0.97	629931
macro avg	0.83	0.78	0.80	629931	macro avg	0.74	0.99	0.81	629931
weighted avg	0.99	0.99	0.99	629931	weighted avg	0.98	0.97	0.98	629931
(c) senza CW per alpha=0,15.					(d) con CW per alpha=0,15.				

Figura 38: Confronto sui Report di Classificazione della D.A. di Botnet e Bruteforce su IDS-MLP con e senza CW, nei due diversi valori di alpha.

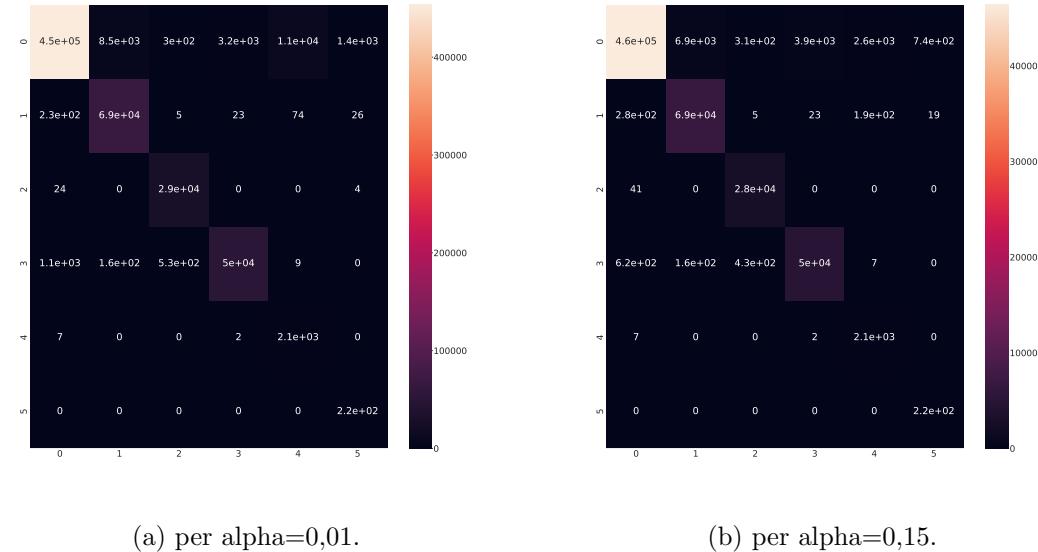


Figura 39: Confronto sulle Matrici di Confusione della D.A. di Botnet e Bruteforce su IDS-MLP con CW nei due diversi valori di alpha.

L'analisi successiva invece riguarda la Data Augmentation con Botnet e Bruteforce aumentati di 10000 samples da cui emerge un sostanziale miglioramento della rilevazione delle classi minori Botnet e Bruteforce: in particolare, si nota come il Class Weighting vada a peggiorare la classificazione avendo perciò una netta disparità di performance dalla versione con solo Data Augmentation. Tale comportamento può ritenersi atteso. Le performance della sola D.A. con aumento delle classi minori a 10000 samples superano tutte le altre modalità esaminate e ciò è visibile sia in tabella 26 che in figura 40 soprattutto in 40a e 63c. Si noti altresì che le performance dell'IDS-MLP con D.A. basato su dati generati per $\alpha=0,15$ (63c) superano leggermente quelle con $\alpha=0,01$ (40a). Si riportano per completezza le matrici di confusione in figura 41. Si vuole perciò evidenziare in tabella 27 le differenze di metriche calcolate tra le due classi minori nelle varie modalità analizzate: si evince perciò che la D.A. con aumento di 10000 samples è una buona soluzione al problema rispetto al Class Weighting ed, inoltre, viene descritto un comportamento emerso anche nella valutazione dei dati generati al capitolo 6.5.1 ovvero quello di una migliore rilevazione

dei Botnet per alpha=0,15 seppur sottile. Come ci si aspettava dalla tabella risulta chiaro quindi che la tecnica di Class Weighting non giova alla performance se unita alla Data Augmentation: il CW però rimane un valido approccio di risoluzione dello sbilanciamento considerando le performance dell'IDS-MLP basilare.

	per alpha=0,01		per alpha=0,15	
	senza CW	con CW	senza CW	con CW
Accuracy	0.994636	0.964261	0.995654	0.974265
Balanced Accuracy	0.993055	0.985596	0.992723	0.989422
Weighted Precision	0.994711	0.978694	0.995682	0.980339
Macro Precision	0.944393	0.692714	0.953356	0.730659
Weighted Recall	0.994636	0.964261	0.995653	0.974265
Macro Recall	0.993055	0.985596	0.992723	0.989422
Weighted F1 score	0.994657	0.969830	0.995660	0.976286
Macro F1 score	0.966155	0.737776	0.971302	0.788486

Tabella 26: Valutazione delle performance di classificazione dell'IDS-MLP aumentando di 10000 samples le classi Botnet e Brutforce (con e senza CW).

	precision	recall	f1-score	support		precision	recall	f1-score	support
0	1.00	1.00	1.00	478327	0	1.00	0.96	0.98	478327
1	0.98	0.99	0.99	69256	1	0.89	0.99	0.94	69256
2	1.00	0.99	1.00	28540	2	0.98	1.00	0.99	28540
3	0.98	0.99	0.99	51486	3	0.97	0.97	0.97	51486
4	0.93	0.99	0.96	2101	4	0.20	1.00	0.33	2101
5	0.77	1.00	0.87	221	5	0.13	1.00	0.23	221
accuracy			0.99	629931	accuracy			0.96	629931
macro avg	0.94	0.99	0.97	629931	macro avg	0.69	0.99	0.74	629931
weighted avg	0.99	0.99	0.99	629931	weighted avg	0.98	0.96	0.97	629931
(a) senza CW per alpha=0,01.					(b) con CW per alpha=0,01.				
	precision	recall	f1-score	support		precision	recall	f1-score	support
0	1.00	1.00	1.00	478327	0	1.00	0.97	0.98	478327
1	0.99	0.99	0.99	69256	1	0.90	0.99	0.94	69256
2	0.99	1.00	0.99	28540	2	0.98	1.00	0.99	28540
3	0.99	0.98	0.99	51486	3	0.95	0.98	0.97	51486
4	0.94	0.99	0.96	2101	4	0.38	1.00	0.55	2101
5	0.81	1.00	0.90	221	5	0.18	1.00	0.30	221
accuracy			1.00	629931	accuracy			0.97	629931
macro avg	0.95	0.99	0.97	629931	macro avg	0.73	0.99	0.79	629931
weighted avg	1.00	1.00	1.00	629931	weighted avg	0.98	0.97	0.98	629931
(c) senza CW per alpha=0,15.					(d) con CW per alpha=0,15.				

Figura 40: Confronto sui Report di Classificazione della D.A. con aumento a 10000 samples delle classi Botnet e Bruteforce su IDS-MLP con e senza CW.

		D.A 100% senza CW	D.A 100% con CW	D.A. 10000 senza CW	D.A. 10000 con CW
Botnet per alpha=0,01	Precision	0.000000	0.130847	0.772727	0.129619
	Recall	0.000000	1.000000	1.000000	1.000000
	F1 Score	0.000000	0.231414	0.871795	0.229491
Bruteforce per alpha=0,01	Precision	0.943938	0.155504	0.932524	0.195411
	Recall	0.985721	0.995716	0.986673	0.997144
	F1 Score	0.964377	0.268998	0.958834	0.326782
Botnet per alpha=0,15	Precision	0.000000	0.226434	0.812500	0.179238
	Recall	0.000000	1.000000	1.000000	1.000000
	F1 Score	0.000000	0.369256	0.896552	0.303989
Bruteforce per alpha=0,15	Precision	0.993573	0.427200	0.935047	0.375448
	Recall	0.735840	0.995716	0.986673	0.997144
	F1 Score	0.845502	0.597885	0.960167	0.545502

Tabella 27: Valutazione delle performance di D.A. migliori: analisi delle metriche delle sole classi minori dell'IDS-MLP.

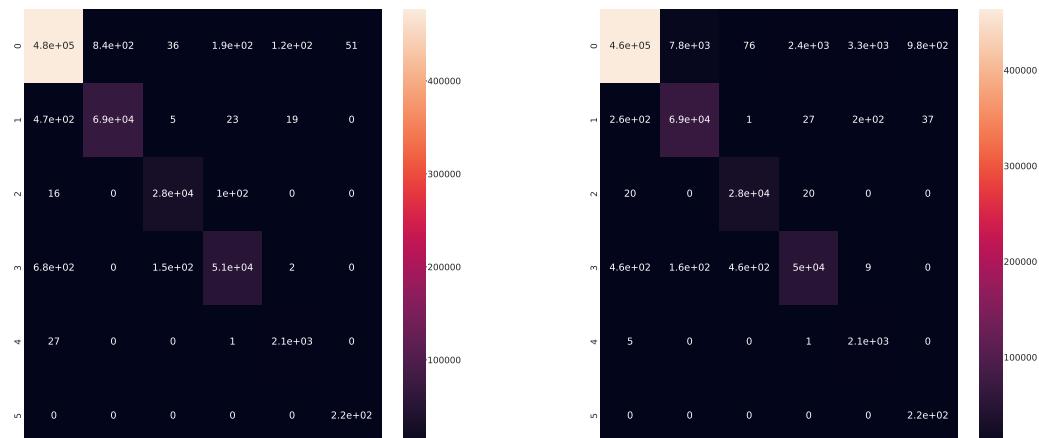
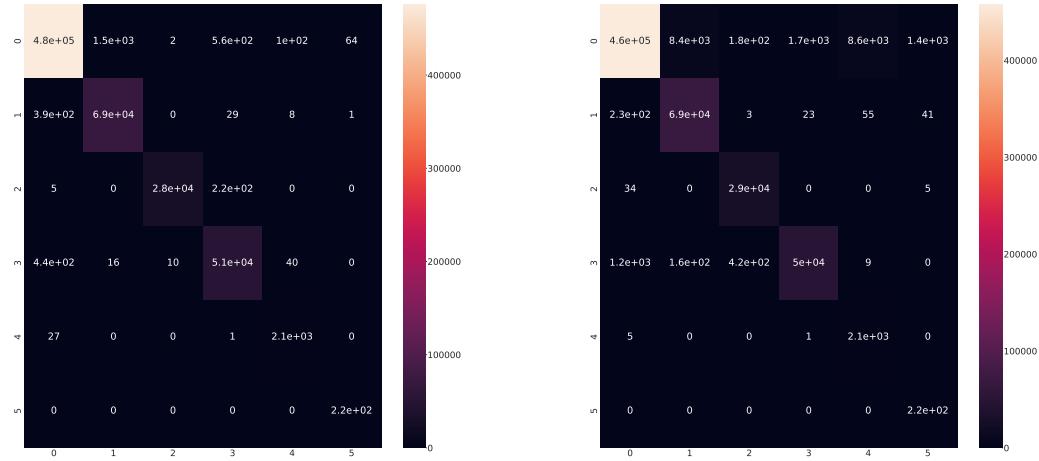


Figura 41: Confronto sulle Matrici di Confusione della D.A. di Botnet e Bruteforce su IDS-MLP nei due diversi valori di alpha.

Per concludere l'analisi si è applicato il K-Cross Validation con k=5 sul Train Set al fine di sincerarsi che la performance registrata dell'IDS-MLP sia attendibile ed, infatti, le metriche di valutazione rilevate confermano quanto riportato in questo capitolo. Tali valori sono presenti in tabella 28 e riportano le performance dell'IDS-MLP basilare senza CW e la D.A. nella sua versione migliore, ovvero con un aumento di 10000 samples delle sole classi minori con generazione per alpha=0,15 (senza CW).

Media delle Metriche di valutazione	IDS-MLP basilare (senza CW)	D.A. migliore
Average Accuracy	0.991	0.993
Average Macro F1 score	0.661	0.984

Tabella 28: Metriche rilevate da K-Cross Validation con K=5 dell'IDS-MLP per verificare l'attendibilità della performance esaminata.

6.6.2 Conclusioni dell’analisi

La valutazione dei dati generati dalla GAN per mezzo di metriche di classificazione ha fatto emergere le stesse problematiche evidenziate nel capitolo 6.5.1. Tali problematiche hanno però impattato in maniera molto sottile sulla rilevazione, senza degradare totalmente le performance degli IDS. Tra i due valori di alpha proposti, il valore $\alpha=0,15$ risulta produrre migliori risultati, seppur in maniera ridotta per alcune casistiche. Inoltre, si conferma che per $\alpha=0,15$ i dati generati di tipo Botnet risultano esser rilevati più facilmente ed, allo stesso modo, i dati generati per Bruteforce con $\alpha=0,01$ vengono rilevati con migliori performance ma non per tutti i casi analizzati. Si conclude perciò che i dati generati dal modello GAN possano ritenersi utili per la Data Augmentation nonostante sia necessario tenere in considerazione la loro non ottimale qualità.

Dall’analisi della Data Augmentation, applicata ai due tipi di IDS per risolvere problemi di apprendimento delle classi minori Botnet e Bruteforce, si osserva che l’IDS-MLP giova di tale aiuto soprattutto nel caso dell’aumento a 10000 unità di dati delle sole classi minori, che inizialmente non venivano rilevate dall’IDS-MLP basilare. Come da attese, l’applicazione della Data Augmentation risulta sufficiente per la risoluzione dei problemi e non necessita di ulteriori tecniche come il Class Weighting che, invece, degrada le performance se combinato.

Per quanto riguarda l’IDS-RFC, invece, la Data Augmentation risulta essere superflua seppur non peggiori in maniera sostanziale le performance superando quelle dell’IDS-MLP. Tale comportamento risulta esser stato rilevato in diverse ricerche in questo campo anche senza l’utilizzo della Data Augmentation. Nonostante il modello MLP sia più complesso dal punto di vista architettonale rispetto all’RFC, quest’ultimo fornisce migliori prestazioni soprattutto in scenari dove i dataset NIDS sono sbilanciati riducendo anche il rischio di overfitting.

Di seguito si fornisce un’analisi completa di tale comportamento basata sul confronto di quanto emerso in questa tesi con altre cinque ricerche accademiche [48, 73, 80, 82, 83]: Bacevicius et al. [73] hanno studiato il comportamento di diversi modelli ML usati come NIDS in presenza di un dataset sbilanciato come CIC-IDS-2017 senza l’ausilio di Data Augmentation. Si ricorda che dalla ricerca accademica di Bacevicius

et al. è stata presa l'architettura dell'**IDS-MLP** ed, invece, da quella di Bulavas et al. [80] è stata presa quella dell'**IDS-RFC**. Dalla ricerca di Bacevicius et al. è emerso che gli algoritmi basati su alberi decisionali sono quelli che ottengono una migliore performance. Allo stato dell'arte vi sono moltissime comparazioni dalle quali emerge come tali algoritmi risultino essere adatti per quei dataset che presentano sbilanciamenti e comprendono elevate quantità di dati. In tabella 29 si nota come, a parità di architettura dell'MLP, i risultati emersi dalla sperimentazione in questa tesi risultino essere più proficui, ma comunque in linea con quelli presentati dall'articolo accademico di Bacevicius et al. ed in particolare si noti come l'applicazione del miglior caso di Data Augmentation, ovvero quello che prevede l'aumento a 10000 samples delle sole classi Botnet e Bruteforce con alpha=0,15, comporti un miglioramento evidente delle prestazioni. Inoltre, si vuole specificare che l'MLP con i relativi iperparametri della ricerca di Bacevicius et al. è leggermente più complesso rispetto a quello adottato da Bulavas et al. ed, invece, l'RFC di Bacevicius et al. risulta essere lievemente meno performante di quello di Bulavas et al. che invece presenta iperparametri adatti a gestire lo sbilanciamento dei dati, seppur entrambi presentino quasi le stesse caratteristiche a livello di complessità.

La motivazione di questo comportamento risiede perciò nel pre-processing ed in tutte quelle scelte di preparazione dei dati applicate in questa tesi nonchè alla possibilità che il dataset CIC-IDS2017 migliorato di Liu et al. [67]. possa aver contribuito in maniera importante sui risultati ottenuti. A tal proposito, sempre dall'articolo preso in esame, si riportano delle considerazioni in merito ai motivi che si celano dietro al funzionamento ottimale di alcuni algoritmi ML usati per NIDS: *«dalle osservazioni emerge che l'impatto sulla accuratezza dell'architettura ed eventuali iperparametri è minima. Ciò che impatta maggiormente risulta essere il pre-processing ed in generale la preparazione dei dati, la loro divisione in training e test nonchè il numero di feature. Permangono però ancora dubbi sull'impatto che possono avere i modelli ibridi complessi, l'applicazione di una profonda pulizia dei dati, la riduzione delle feature e la fusione di classi di dati»*. Ciò fa dedurre che il lavoro di pre-processing e preparazione dei dati adottato in questa tesi incida in maniera sostanziale sulle prestazioni evidenziate, ma il comportamento riscontrato, nel quale le prestazioni del

modello RFC superano quelle dell'MLP, rimane in linea con quello osservato allo stato dell'arte.

Allo stesso modo, analizzando in maniera approfondita il lavoro di Bulavas et al., nel quale vengono impiegati diversi algoritmi per valutare il loro comportamento in presenza di dati sbilanciati nel dataset CIC-IDS-2017, si delinea la stessa dinamica nonostante venga applicata la tecnica SMOTE per effettuare Data Augmentation ed un under-sampling alle classi maggiori per ridurle. Tale tecnica SMOTE [85] permette di creare nuovi dati nella classe rara per bilanciare le occorrenze tra classi. Viene scelto un caso di classe rara e i suoi punti K più vicini per generare nuovi casi sulla linea che li collega. Gli autori precisano che le metriche più adatte risultano essere quelle *Macro Average* in quanto più sensibili e più adatte a rilevare difficoltà in presenza di classi sbilanciate nel dataset. Inoltre, gli stessi autori precisano che le metriche riportate nel loro articolo non sono direttamente comparabili con quelle di altre ricerche per la diversità di approccio utilizzato per la valutazione delle performance e che i valori messi a confronto hanno solo il mero scopo di confrontare ricerche allo stato dell'arte. Tali metriche sono riportate in tabella 29 e da esse si evidenzia come anche in questo caso l'RFC adottato dai ricercatori superi l'MLP. Il lavoro di preparazione dei dati svolto in questa tesi risulta essere diverso da quello proposto anche in questo articolo e perciò si ritiene che tali azioni incidano sulle performance riscontrate durante la sperimentazione, seppur rimangano in linea con quelle nello studio a parità di architettura RFC. Si riportano per completezza d'analisi anche le metriche di valutazione rilevate da Sharafaldin et al. [48] al fine di un confronto.

Infine, si vogliono analizzare gli studi di Barkah et al. [82] ed Alabrah [83] che sfruttano il modello GAN e sue varianti per effettuare Data Augmentation sui dati al fine di studiarne gli effetti sulla multi-classificazione con diversi algoritmi. Per quanto riguarda la ricerca di Barkah et al., essi sfruttano il dataset CIC-IDS-2017 per studiare il modello DGM che sfrutta CGAN da loro elaborato. I dati vengono preparati con tecniche diverse da quelle adottate in questa tesi e successivamente aumentati del 100%. Dai risultati di classificazione degli algoritmi impiegati è emerso che il modello DGM con RF è migliore di quello con MLP. Le metriche riscontrate sono visibili in tabella 30 ed evidenziano quanto detto poc'anzi. La Data Augmentation applicata in

Modello	Metriche di Valutazione	Bacevicius et al.	Bulavas et al.	Sharafaldin et al.	In questa tesi		
					senza CW	con CW	con D.A. migliore
MLP	Accuracy	0.93029	-	-	0.990528	0.953895	0.995654
	Balanced Accuracy	-	0.937	-	0.661251	0.984991	0.992723
	Weighted Precision	0.87388	0.981	0.77	0.987040	0.972948	0.995682
	Macro Precision	0.27089	0.879	-	0.655873	0.668752	0.953356
	Weighted Recall	0.93018	0.980	0.83	0.990528	0.953895	0.995653
	Macro Recall	0.27662	-	-	0.661251	0.984991	0.992723
	Weighted F1 Score	0.90038	0.980	0.76	0.988741	0.961196	0.995660
	Macro F1 Score	0.27127	-	-	0.658497	0.716831	0.971302
RFC	Accuracy	0.98489	-	-	0.999878	0.999876	0.999876
	Balanced Accuracy	-	0.991	-	0.999789	0.998288	0.998288
	Weighted Precision	0.98106	0.998	0.98	0.999881	0.999879	0.999879
	Macro Precision	0.88152	0.913	-	0.999763	0.999684	0.999684
	Weighted Recall	0.98489	0.998	0.97	0.999879	0.999879	0.999879
	Macro Recall	0.81499	-	-	0.999789	0.998288	0.998288
	Weighted F1 Score	0.98237	0.998	0.97	0.999879	0.999879	0.999879
	Macro F1 Score	0.83882	-	-	0.999776	0.998983	0.998983

Tabella 29: Confronto dei risultati relativi alle prestazioni con ricerche allo stato dell’arte.

questa tesi sull’IDS-MLP ha permesso un aumento delle prestazioni. Si precisa che tale articolo non approfondisce l’architettura dei modelli di classificazione utilizzati e perciò non è possibile confrontarli con quelli sfruttati durante la sperimentazione.

Per quanto riguarda lo studio svolto dal Dottor Alabrah, seppur utilizzando un dataset diverso da quello utilizzato in questa tesi e nelle ricerche analizzate finora, anche in questo caso si osserva come l’MLP utilizzato sia migliore del RF in tutti gli scenari analizzati con e senza Data Augmentation.

Modello	Metriche di Valutazione	Barkah et al.	In questa tesi		
			con D.A.	senza CW	con CW
MLP	Precision	90%	66%	67%	95%
	Recall	88%	66%	98%	99%
	F1 Score	87%	66%	72%	97%
RF			senza D.A.		con D.A. migliore
	Precision	98%	99%		99%
	Recall	97%	99%		99%
	F1 Score	97%	99%		99%

Tabella 30: Confronto dei risultati relativi alle prestazioni con l’articolo accademico di Barkah et al.

Capitolo 7

Conclusioni e sviluppi futuri

L’obiettivo finale di questa sperimentazione era quello di verificare che la Data Augmentation eseguita con Vanilla GAN su un dataset fortemente sbilanciato, come il CIC-IDS-2017, potesse migliorare le performance di due NIDS realizzati con i modelli MLP ed RFC. Il dataset sfruttato è stato impiegato nella sua forma migliorata dai ricercatori Liu et al. [67], ma presenta comunque un forte sbilanciamento. I vari step applicati durante la sperimentazione hanno permesso di filtrare il dataset e prepararlo adeguatamente al fine di permettere al modello GAN realizzato di apprendere più facilmente i dati e generarne di nuovi al fine di effettuare Data Augmentation delle classi minori. Dall’analisi dei dati generati è emerso come la GAN impiegata possa ritenersi accettabile in quanto genera dati buoni nonostante la loro non ottimale qualità: sia i risultati emersi da una pura analisi quantitativa e qualitativa dei dati generati, sia quelli derivanti dalla valutazione delle performance degli IDS mostrano buone prestazioni e nessun peggioramento evidente delle stesse. Si è evidenziato inoltre che il modello RFC supera quello MLP ed, anzi, non giova della Data Augmentation. Tale comportamento è stato riscontrato in larga scala in letteratura e si deduce perciò che la sperimentazione eseguita rimane in linea con i risultati allo stato dell’arte ed, anzi, presenta performance interessanti.

Siffatto rendimento perciò può ritenersi soddisfacente rispetto agli obiettivi che ci si era posti in questa tesi: si è riusciti perciò ad ottenere un miglioramento delle performance applicando Data Augmentation realizzata con dati generati dal modello

Vanilla GAN soprattutto per l'IDS-MLP.

I motivi sono da ricercarsi prevalentemente nella fase di pre-processing: il dataset utilizzato può aver contribuito a migliorare le prestazioni rispetto alle ricerche accademiche analizzate che sfruttano la versione originale del CIC-IDS2017 nonchè il pre-processing eseguito in maniera profonda e la tecnica di feature extraction applicata possono aver migliorato l'apprendimento ottenendo così un potenziamento delle prestazioni a confronto con gli articoli studiati, a parità di architettura dei modelli utilizzati. In secondo luogo, si può riscontrare una efficace Data Augmentation prevalentemente per l'IDS-MLP grazie alla sperimentazione svolta sul modello GAN e perciò si ritiene che l'architettura utilizzata nonchè gli iperparametri siano adatti a questo scopo sebbene possano essere ottimizzati.

I possibili sviluppi futuri comprendono un'analisi comparativa dell'attuale esperimento con l'utilizzo del solo dataset originale CIC-IDS2017 al fine di validare quanto supposto poc'anzi e confermare così che il dataset migliorato abbia inciso sulle prestazioni ed, inoltre, confrontare il pre-processing adottato in questa tesi con altre versioni in modo da appurare quanto sostenuto nelle conclusioni, assicurandosi così che le performance dipendano anche dagli step impiegati nella preparazione dei dati.

Come da attese, l'utilizzo della versione originale Vanilla GAN non ha permesso di ottenere dati ottimali per tutte le classi e perciò si reputa necessario lavorare in futuro sull'ottimizzazione del modello presentato, eseguendo così fine-tuning degli iperparametri in maniera più approfondita come pure sviluppare una tipologia diversa di GAN che possa generare dati migliori sotto forma di csv come quelli adottati.

La Data Augmentation eseguita in questa tesi è stata svolta per mezzo del modello GAN, ma esistono altre tecniche come la SMOTE e le sue varianti. Si reputa perciò un valido progetto futuro quello di effettuare un confronto sulle prestazioni adottando altre tecniche e sincerandosi quale sia la migliore, anche a fronte di diversi modelli GAN utilizzati.

Infine, un lavoro futuro interessante potrebbe essere quello di realizzare un sistema NIDS a sè stante che sfrutti edge-computing nonchè Federated Learning. Come

spiegato nel capitolo 4, il FL può essere un’ottima soluzione per la privacy in quanto garantisce un ambiente collaborativo tra più partecipanti senza la necessità di condividere dati privati tra loro o con un server centrale.

Un primo possibile sviluppo potrebbe essere quello di applicare l’idea dei ricercatori Liu et al. [74] a questa tesi, ovvero: un framework FL che esegue Data Augmentation sui dati Non-IID sfruttando il modello ACGAN il quale è stato addestrato sfruttando un dataset aumentato con tecnica SMOTE. A livello di Server rimane presente un ACGAN completa che viene addestrata e per ogni client, invece, viene rilasciato solo il Generatore che andrà ad aumentare i dati localmente considerando gli aggiornamenti forniti. In questo modo, i ricercatori hanno aumentato i dati dei client coinvolti e ridotto l’impatto negativo dei dati Non-IID. La classificazione degli attacchi è eseguita sfruttando una CNN e raggiunge un’accuratezza di oltre il 98%. Considerando i vantaggi di questa nuova tecnologia, ovvero il Federated Learning, si ritiene interessante approfondire questo aspetto in futuro andando a realizzare un framework simile impiegando le risorse realizzate in questa tesi oppure sperimentando altre combinazioni di tipologie di GAN, iperparametri e soluzioni di Data Augmentation.

Oltre a questo risulta essere non banale l’idea di integrare un meccanismo di Self-attention spesso incontrato durante lo studio della letteratura di questa tesi per migliorare le prestazioni del modello GAN nonchè di un eventuale sistema NIDS completo come, ad esempio, nella ricerca di Araujo et al [15] sugli attacchi DoS: essi hanno realizzato un NIDS sfruttando una WGAN; quest’ultima realizzata con un layer TCN oppure con meccanismo di Self-Attention. Dai risultati emerge come due blocchi TCN superino le performance di un singolo blocco e di un livello di self-attention ottenendo il 97% di accuratezza. Tale soluzione è stata adottata sfruttando edge-computing, ma si ritiene un ottimo sviluppo futuro quello di adottare tale framework ed inserirlo in un contesto di Federated Learning studiando come poter migliorare i risultati ottenuti in questa tesi.

Infine, un ultimo valido progetto realizzabile potrebbe essere quello di cogliere idee dal lavoro di Idrissi et al. [46]: innanzitutto, attuando il lavoro svolto in questa tesi ad un contesto FL (valutando diversi algoritmi conosciuti come FedProx), lo si potrebbe mettere a confronto con quello svolto dagli studiosi a parità di dataset utilizzato ed,

inoltre, si potrebbero studiare le eventuali differenze relative al pre-processing nonché eventuali comparazioni con tipologie di GAN diverse.

Considerare il FL come possibile tecnologia da attuare per uno sviluppo futuro però apre a molteplici combinazioni di risorse diverse quali: tipologie di GAN; diversi algoritmi di FL; algoritmi di classificazione ML/DL e tecniche di Data Augmentation. Ci si aspetta perciò in futuro moltissimi studi in merito a possibili soluzioni in questo campo di ricerca e nella sicurezza informatica nel suo complesso, soprattutto a fronte dell'avanzamento tecnologico dell'intelligenza artificiale che stiamo vivendo.

Appendice A

Preprocessing

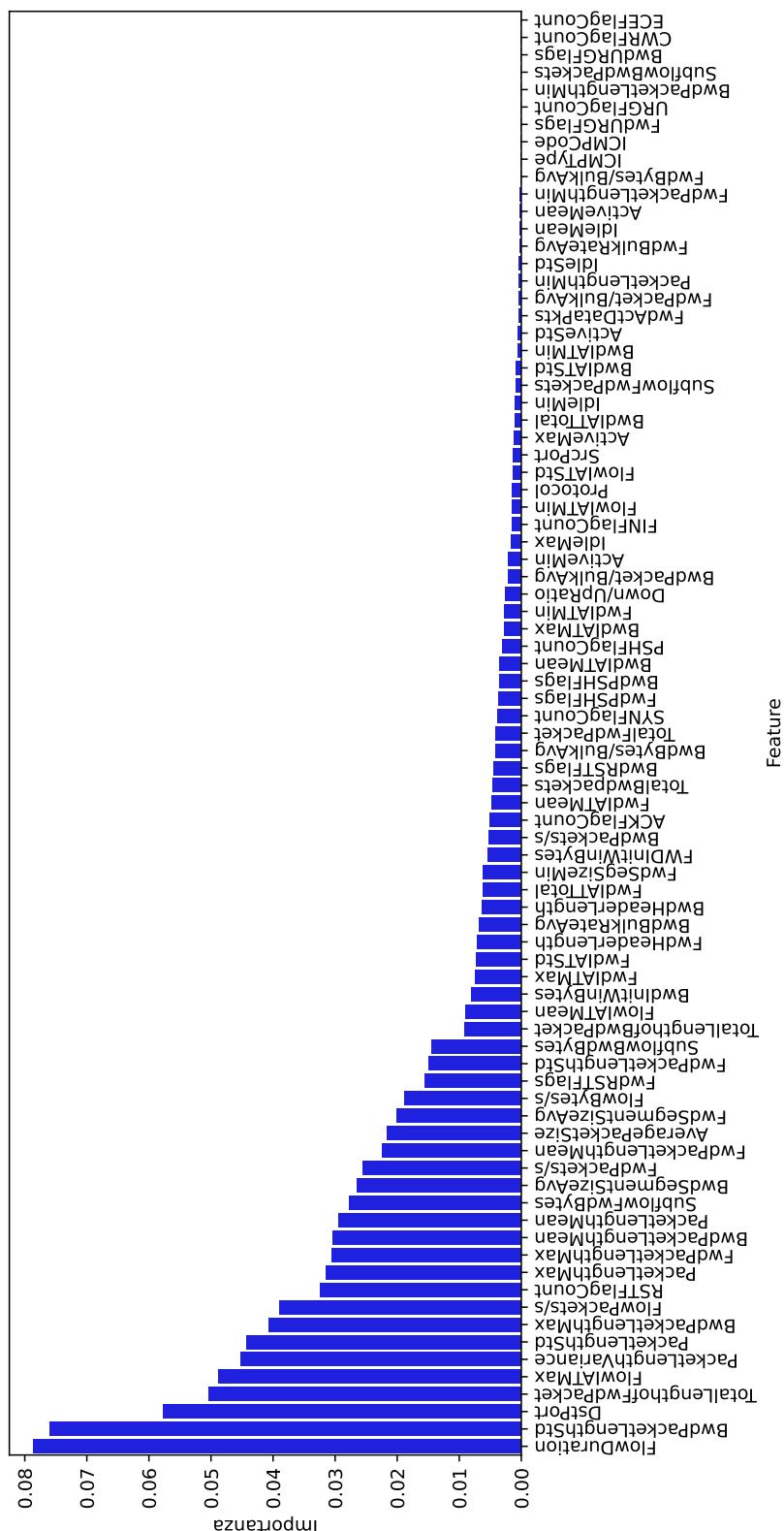


Figura 42: Esito dell’analisi di importanza delle feature su Random Forest. Le feature sono ordinate in ordine decrescente secondo il punteggio ottenuto.

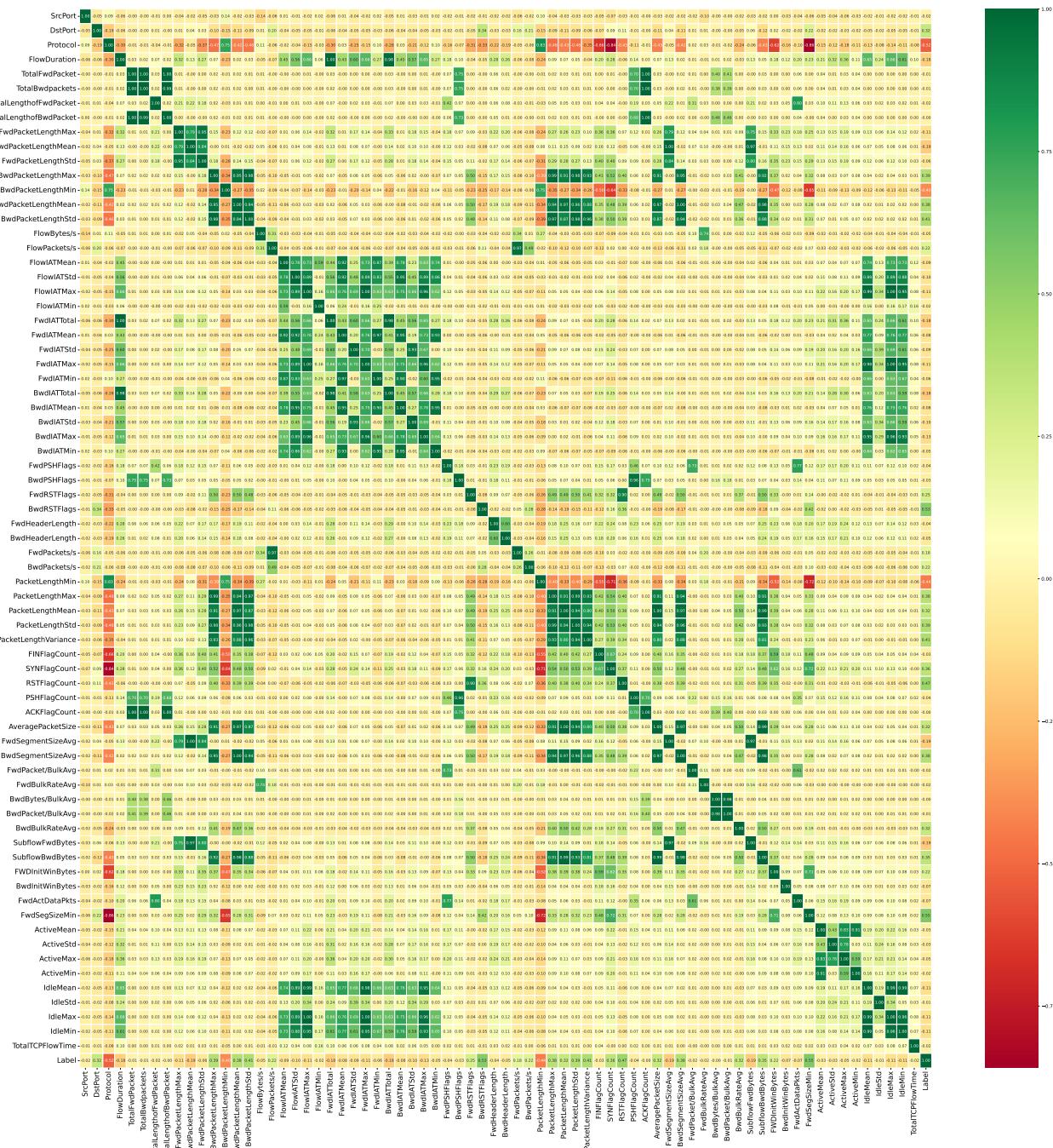


Figura 43: Grafico di correlazione delle feature prima della selezione.

Appendice B

GAN

Vengono di seguito analizzate le restanti classi generate dal modello Vanilla GAN sviluppato. Le valutazioni mancanti nel capitolo 6 riguardano quelle relative alle classi DDoS, DoS e Portscan, di cui non sono stati generati un numero specifico di samples (come per Botnet e Bruteforce) in quanto presentano già una soddisfacente quantità.

Per quanto riguarda la classe DDoS, si può affermare che vi è qualche difficoltà di generazione dei dati come visibile in figura 44 e 45. Dall'analisi con scatter plot in figura 46 non emerge una netta distinzione tra le due configurazioni. Infine, in tabella 31 si riportano i valori statistici calcolati.

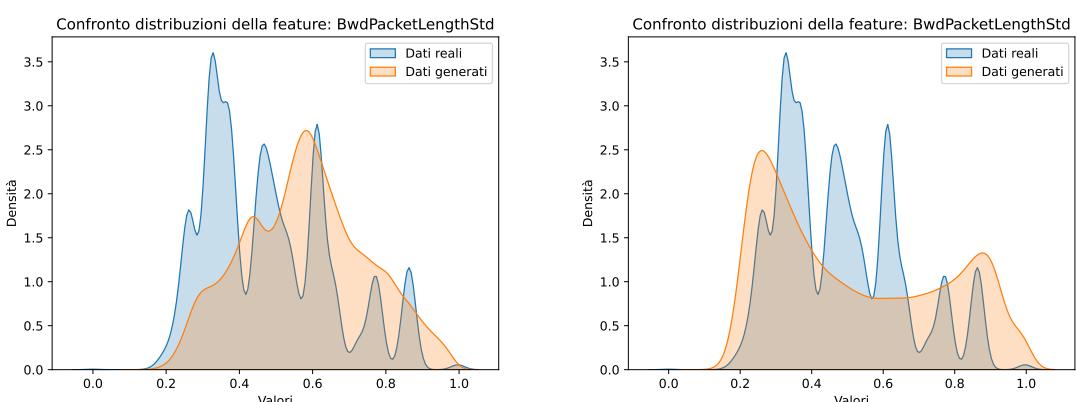
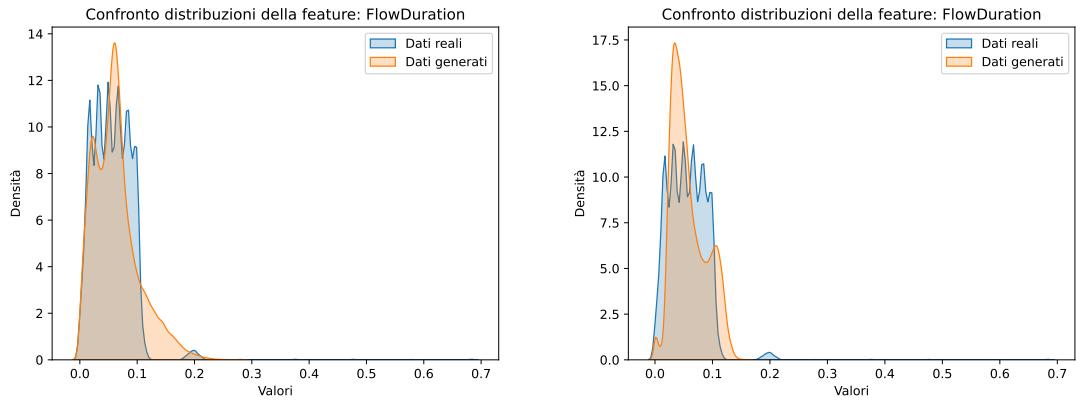


Figura 44: Confronto tra distribuzioni di due feature per la classe DDoS nelle due configurazioni di iperparametri.

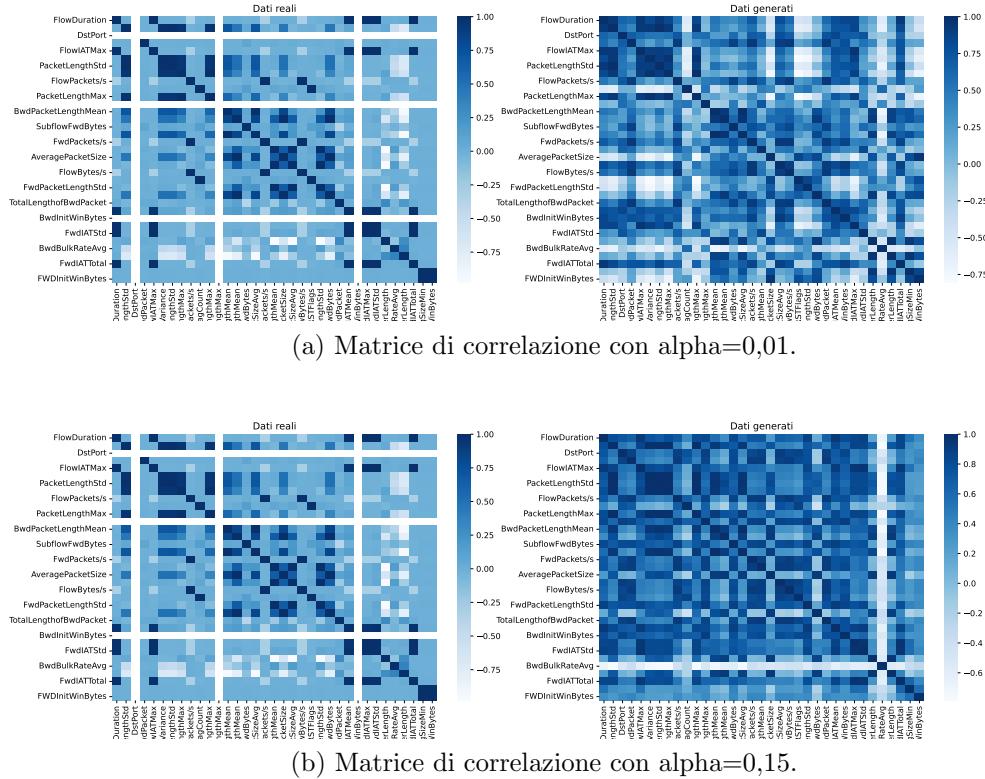
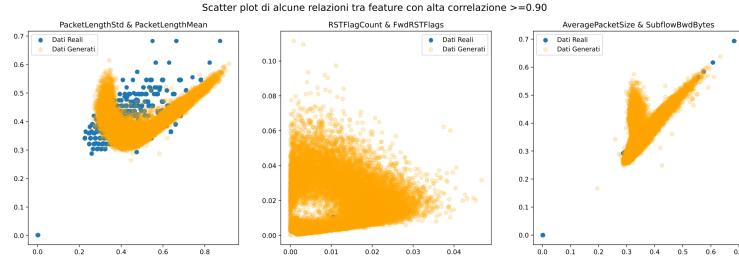


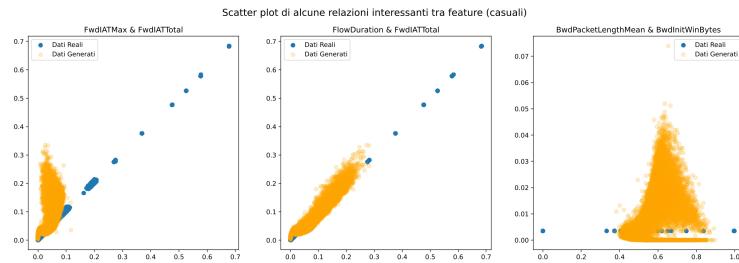
Figura 45: Confronto tra matrici di correlazione della classe DDoS nelle due configurazioni di iperparametri.

	Alpha=0.01	Alpha=0.15
Wasserstein Distance totale	0.011944	0.008930
Wasserstein Distance (0)	0.007041	0.006986
Mean Squared Error (0)	0.002851	0.002117
Wasserstein Distance (1)	0.101131	0.072571
Mean Squared Error (1)	0.067723	0.091038
Punteggio SDV Qualità	58.73%	60.33%
Punteggio SDV Validità	84.62%	81.6%

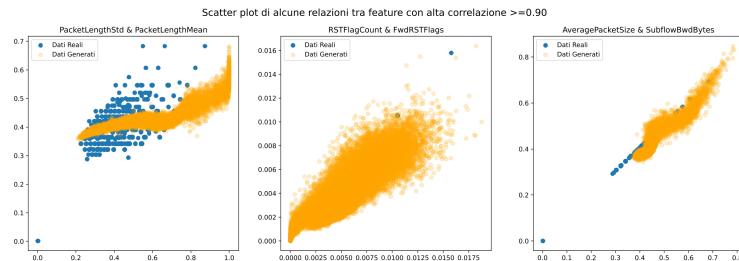
Tabella 31: Analisi statistiche sui dati generati di tipo DDoS nelle due configurazioni di iperparametri.



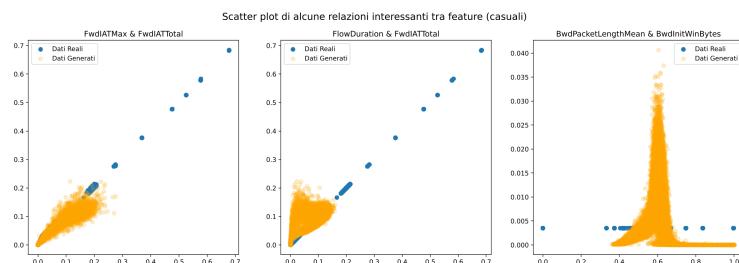
(a) Scatter plot con alpha=0,01 con alta correlazione.



(b) Scatter plot con alpha=0,01 casuali.



(c) Scatter plot con alpha=0,15 con alta correlazione.



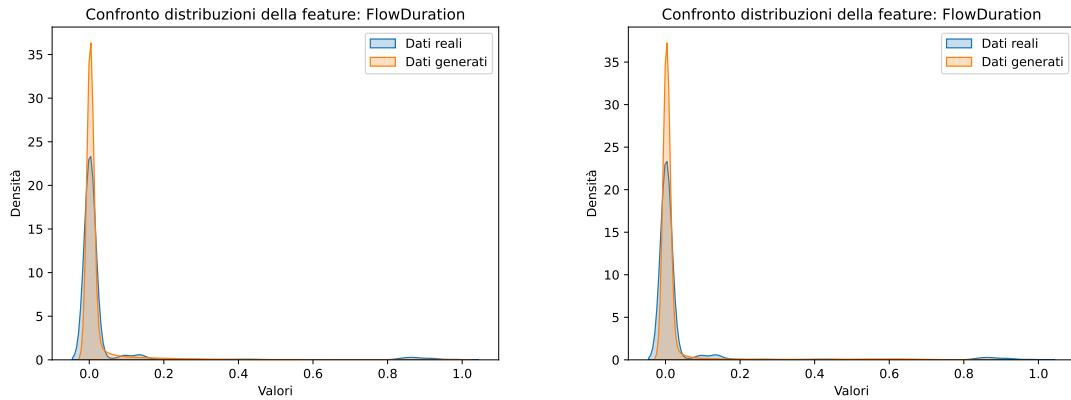
(d) Scatter plot con alpha=0,15 casuali.

Figura 46: Confronto tra scatter plot della classe DDoS nelle due configurazioni di iperparametri.

	Alpha=0.01	Alpha=0.15
Wasserstein Distance totale	0.010138	0.012041
Wasserstein Distance (0)	0.017565	0.016801
Mean Squared Error (0)	0.033348	0.033987
Wasserstein Distance (1)	0.078562	0.061150
Mean Squared Error (1)	0.023725	0.053745
Punteggio SDV Qualità	65.96%	63.14%
Punteggio SDV Validità	92.81%	93.9%

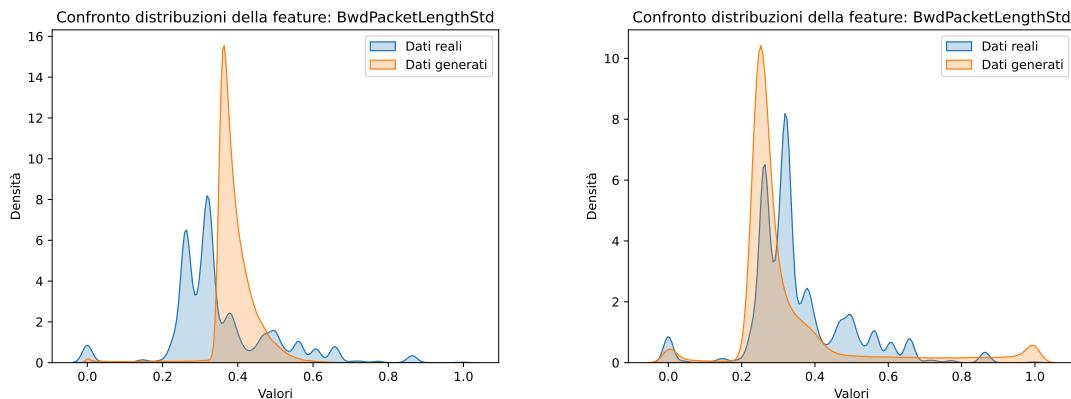
Tabella 32: Analisi statistiche sui dati generati di tipo DoS nelle due configurazioni di iperparametri.

Per la classe DoS, invece, dai grafici in figura 47 si nota come per alpha=0,15 vi sia un miglior apprendimento delle distribuzioni soprattutto considerando 47d rispetto alla sua generazione con alpha=0,01. Tale affermazione viene ribaltata dal grafico delle matrici di correlazione che evidenzia una migliore performance per alpha=0,01, visibile in figura 48, con qualche lieve imperfezione. Nonostante ciò, però il grafico degli scatter plot in figura 49 presenta una miglior generazione per alpha=0,01. In tabella 32 si riportano i dati statistici calcolati. Si conviene perciò che si abbia una migliore generazione per alpha=0,01.



(a) Distribuzione della feature FlowDuration con
alpha=0,01.

(b) Distribuzione della feature FlowDuration con
alpha=0,15.



(c) Distribuzione della feature BwdPacketLengthStd con alpha=0,01.

(d) Distribuzione della feature BwdPacketLengthStd con alpha=0,15.

Figura 47: Confronto tra distribuzioni di due feature per la classe DoS nelle due configurazioni di iperparametri.

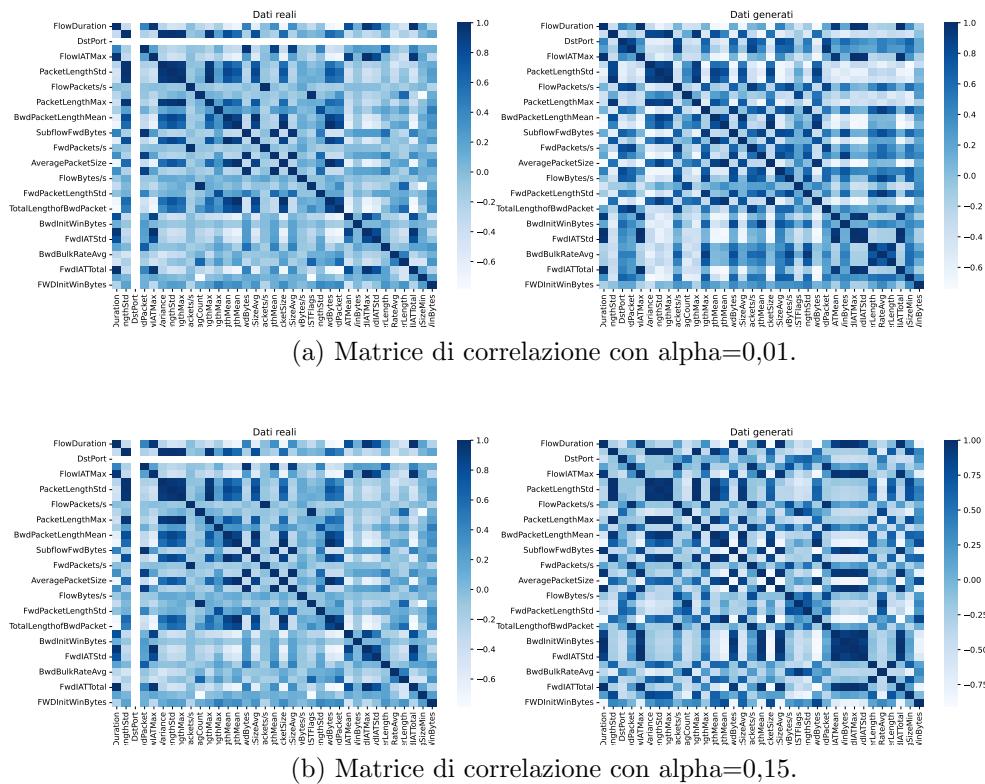
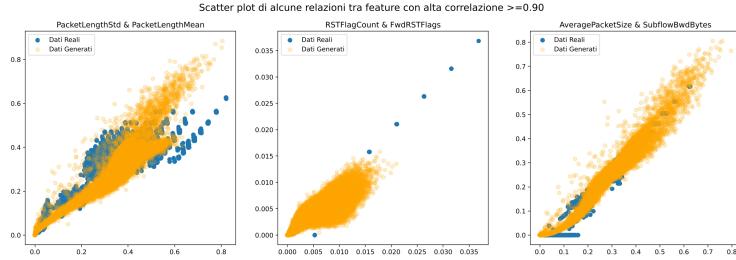
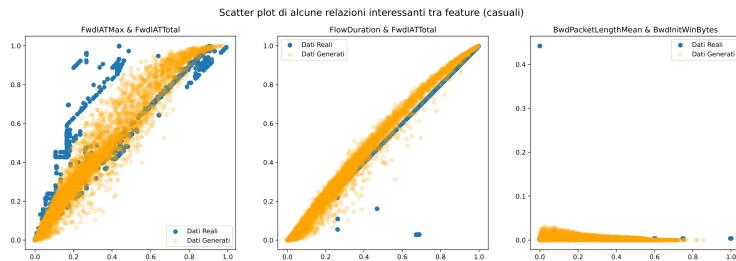


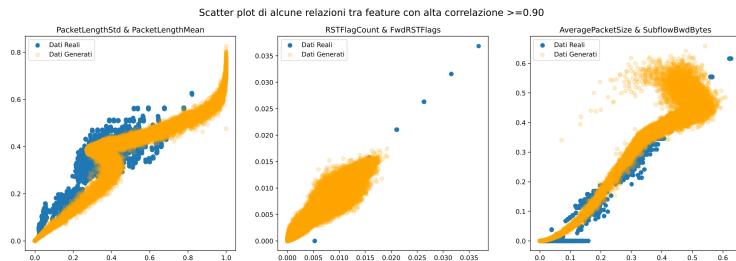
Figura 48: Confronto tra matrici di correlazione della classe DoS nelle due configurazioni di iperparametri.



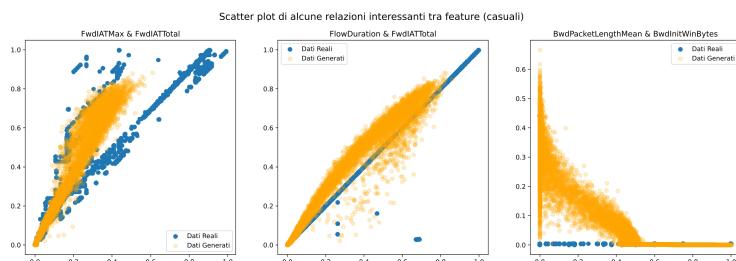
(a) Scatter plot con alpha=0,01 con alta correlazione.



(b) Scatter plot con alpha=0,01 casuali.



(c) Scatter plot con alpha=0,15 con alta correlazione.



(d) Scatter plot con alpha=0,15 casuali.

Figura 49: Confronto tra scatter plot della classe DoS nelle due configurazioni di iperparametri.

	Alpha=0.01	Alpha=0.15
Wasserstein Distance totale	0.003422	0.004988
Wasserstein Distance (0)	0.000638	0.001351
Mean Squared Error (0)	0.000198	0.000201
Wasserstein Distance (1)	0.000178	0.000210
Mean Squared Error (1)	4.515356121837748e-05	4.5166237648203756e-05
Wasserstein Distance (2)	0.024742	0.032494
Mean Squared Error (2)	0.081940	0.069606
Punteggio SDV Qualità	50.23%	48.75%
Punteggio SDV Validità	99.69%	99.47%

Tabella 33: Analisi statistiche sui dati generati di tipo Portscan nelle due configurazioni di iperparametri.

Infine, come anticipato nel capitolo 6, la classe Portscan non ha ottenuto una generazione soddisfacente e sembra non essere stata completamente appresa dal modello. Per poter dare una visione più completa delle distribuzioni è stata aggiunta una terza feature, ovvero DstPort (2), al fine di una comprensione migliore della generazione. Dai grafici in figura 50 emerge una difficoltà nella generazione, ma è visibile un certo miglioramento per alpha=0,15 soprattutto nel grafico 50f ed allo stesso modo è visibile in figura 51. In tabella 33 si riportano i valori statistici calcolati. Dallo scatter plot in figura 52 si evidenzia appunto la difficoltà di generazione. Si ritiene perciò che il modello così sviluppato non sia stato in grado di apprendere tale classe correttamente.

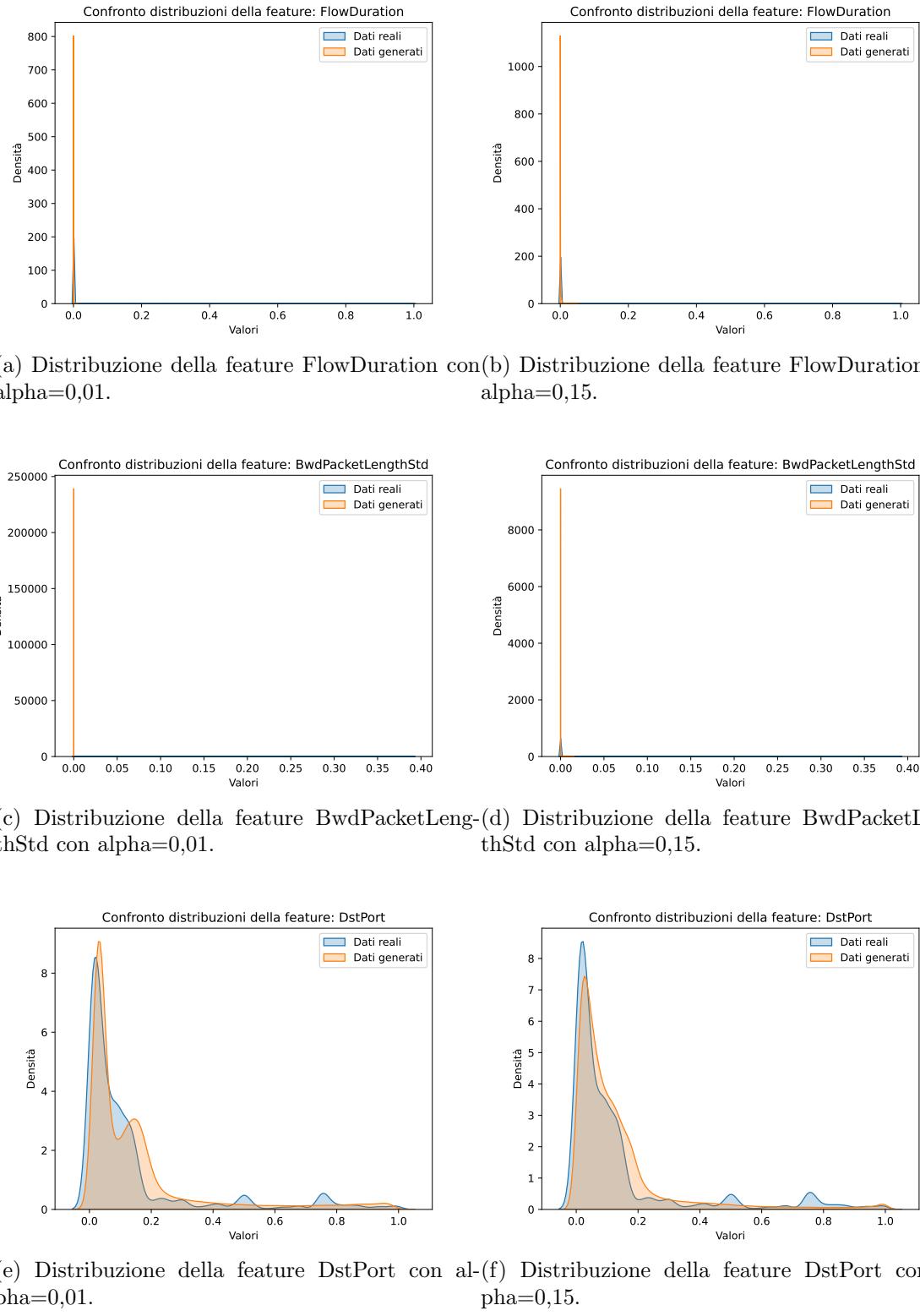


Figura 50: Confronto tra distribuzioni di tre feature per la classe Portscan nelle due configurazioni di iperparametri.

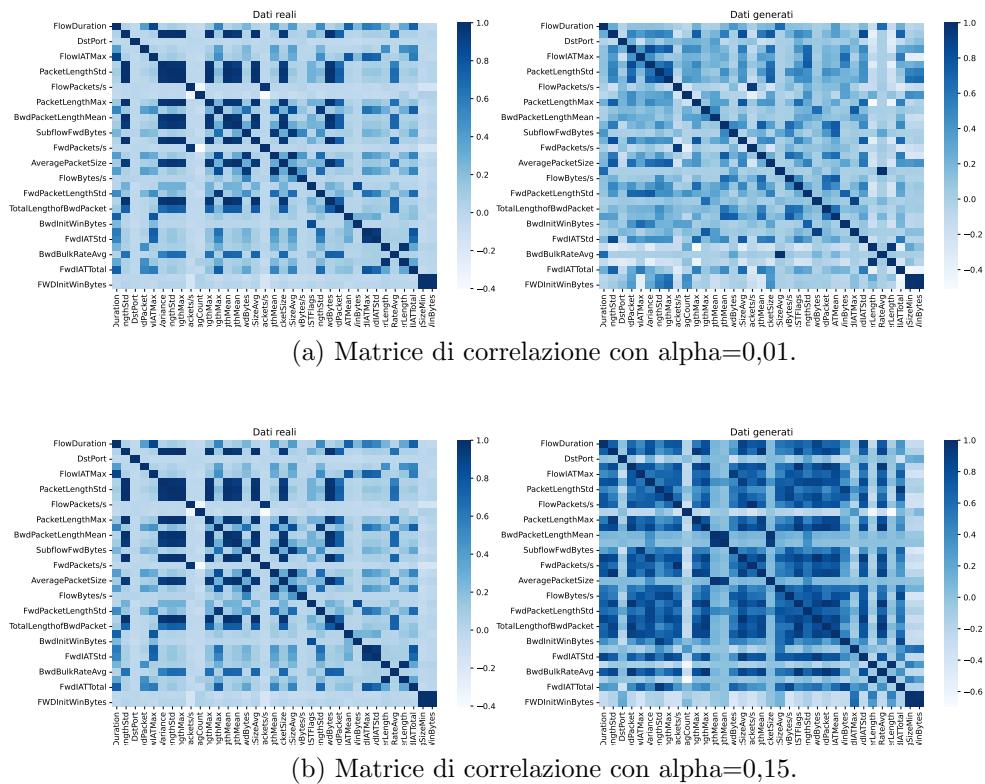


Figura 51: Confronto tra matrici di correlazione della classe Portscan nelle due configurazioni di iperparametri.

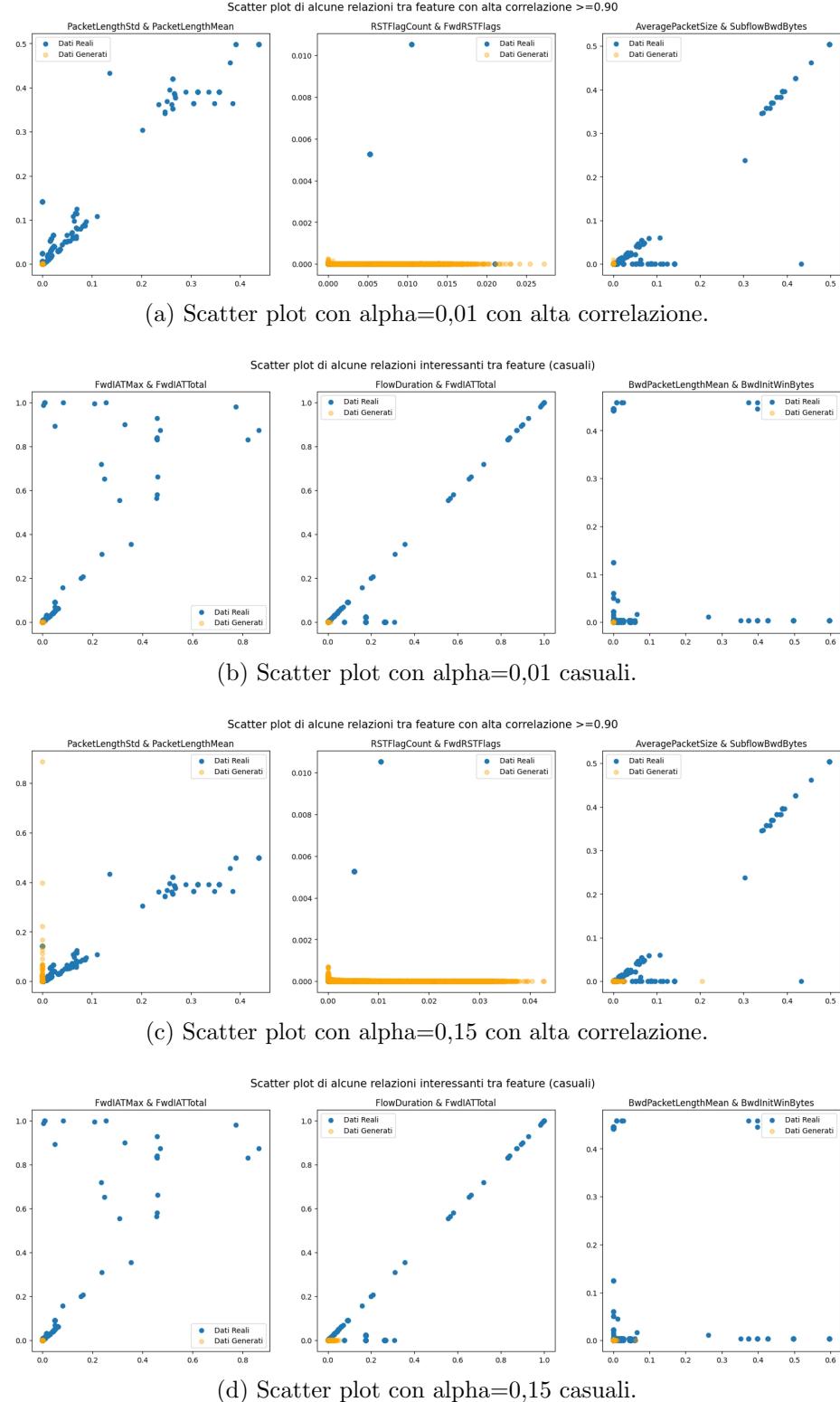


Figura 52: Confronto tra scatter plot della classe Portscan nelle due configurazioni di iperparametri.

Appendice C

IDS

Si riportano di seguito i grafici, non menzionati nel capitolo 6.6.1, per completezza dell’analisi: in particolare vengono riportati i grafici delle curve loss e di accuratezza del modello IDS-MLP nonchè le restanti analisi non approfondite nel capitolo.

IDS-MLP

Di seguito si riportano i grafici relativi alle curve loss e di accuratezza dell’IDS-MLP che dimostrano il corretto apprendimento e non evidenziano problematiche quali overfitting.

Successivamente, si riportano le analisi relative al modello IDS-MLP ed in particolare l’analisi della D.A. su tutte le classi con l’aumento delle sole classi minori a 10000 samples: dalla tabella 34 e dalle figure 63 e 64 emerge la presenza di qualche difficoltà con la rilevazione delle classi maggiori aumentate con miglioramenti visibili per le classi minori rispetto all’originale IDS-MLP senza alcuna tecnica. Come già evidenziato nel capitolo 6.6.1 la tecnica di Data Augmentation esclude quella di Class Weighting e viceversa, perciò si ritiene più utile l’utilizzo della D.A. In generale, si ritiene la performance per alpha=0,01 leggermente migliore rispetto all’altra, nonostante qualche imprecisione. Si può affermare perciò che anche in questo caso i dati generati dalla GAN non inficiano negativamente sulle prestazioni bensì aiutano la rilevazione delle classi minori.

Infine, valutando la migliore performance visibile in 63a rispetto alla figura 40 nel capitolo 6.6.1 si ritiene che il solo aumento delle classi minori senza class weighting sia sufficiente per migliorare le prestazioni di tale modello, come già largamente affermato nel capitolo citato.

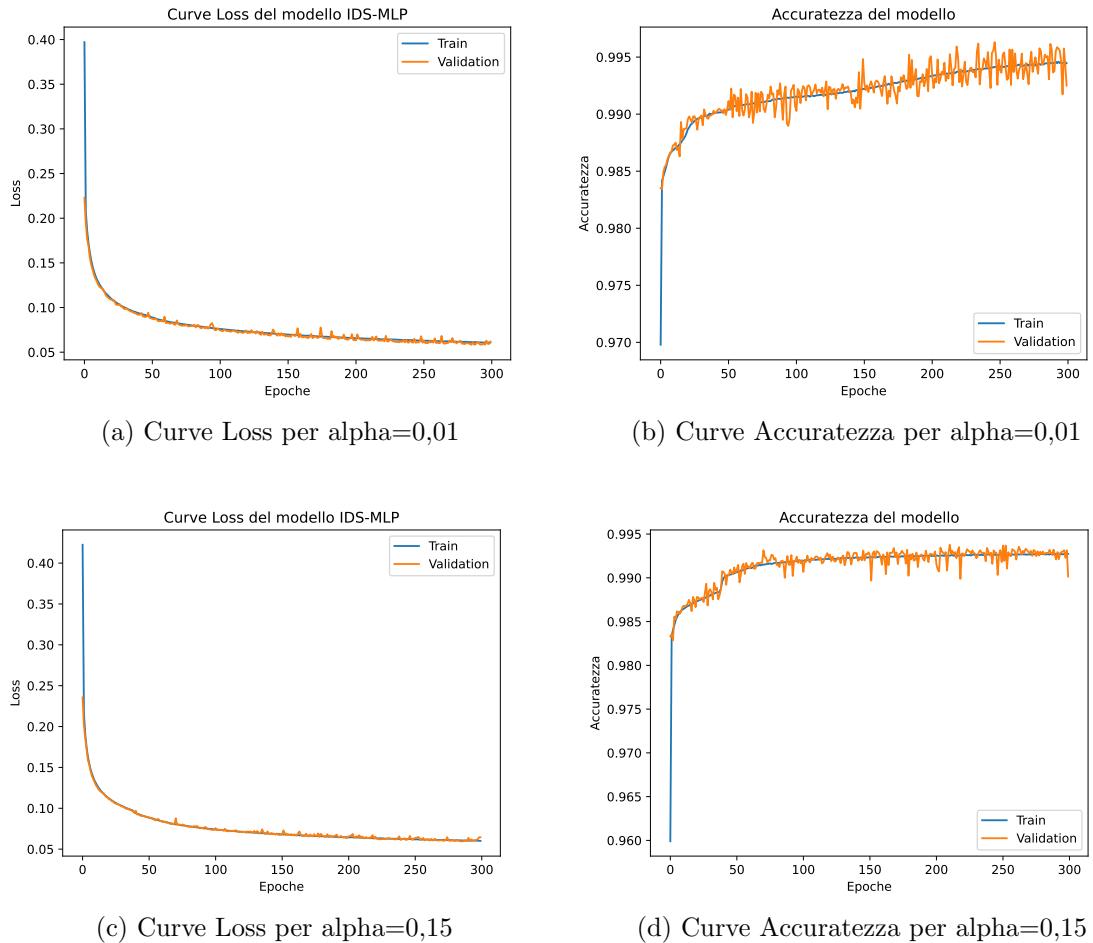


Figura 53: Grafici Curve Loss e Accuratezza della D.A. al 100% di tutte le classi senza CW nei due diversi valori di alpha.

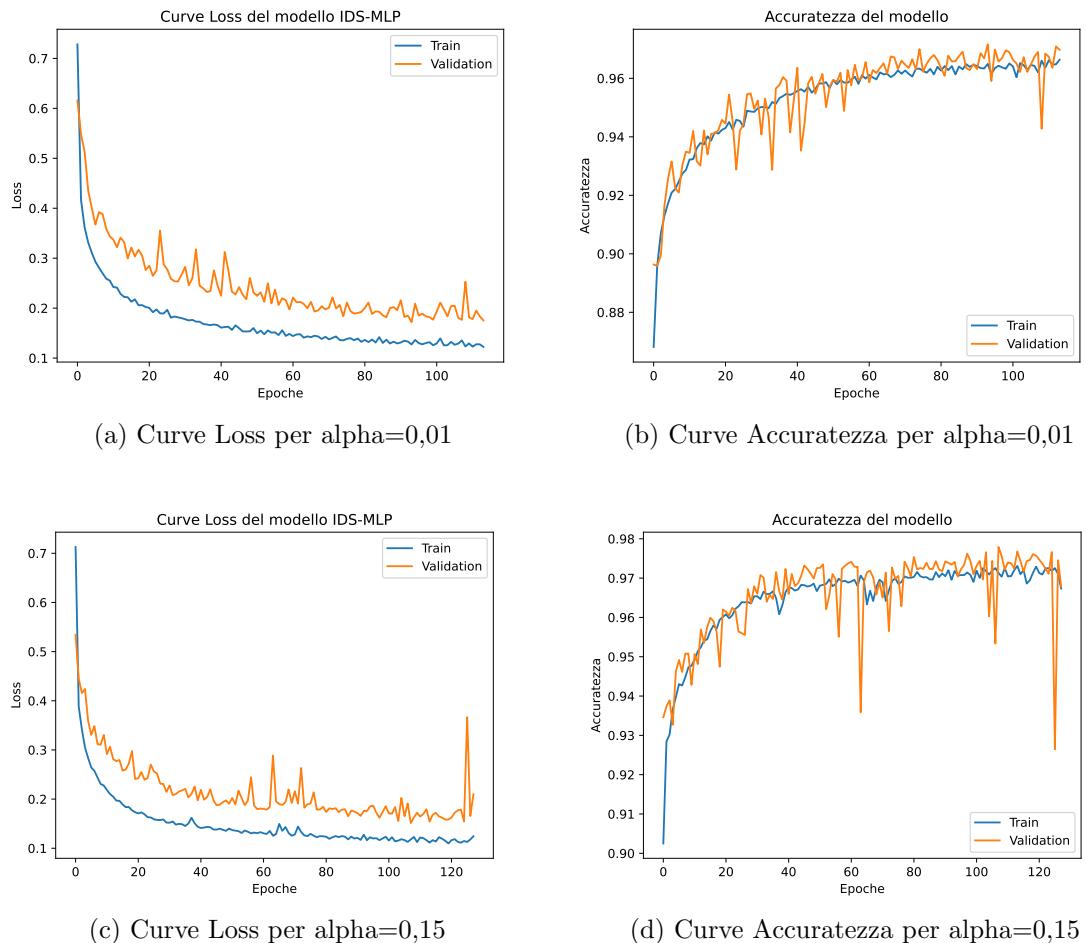


Figura 54: Grafici Curve Loss e Accuratezza della D.A. al 100% di tutte le classi con CW nei due diversi valori di alpha.

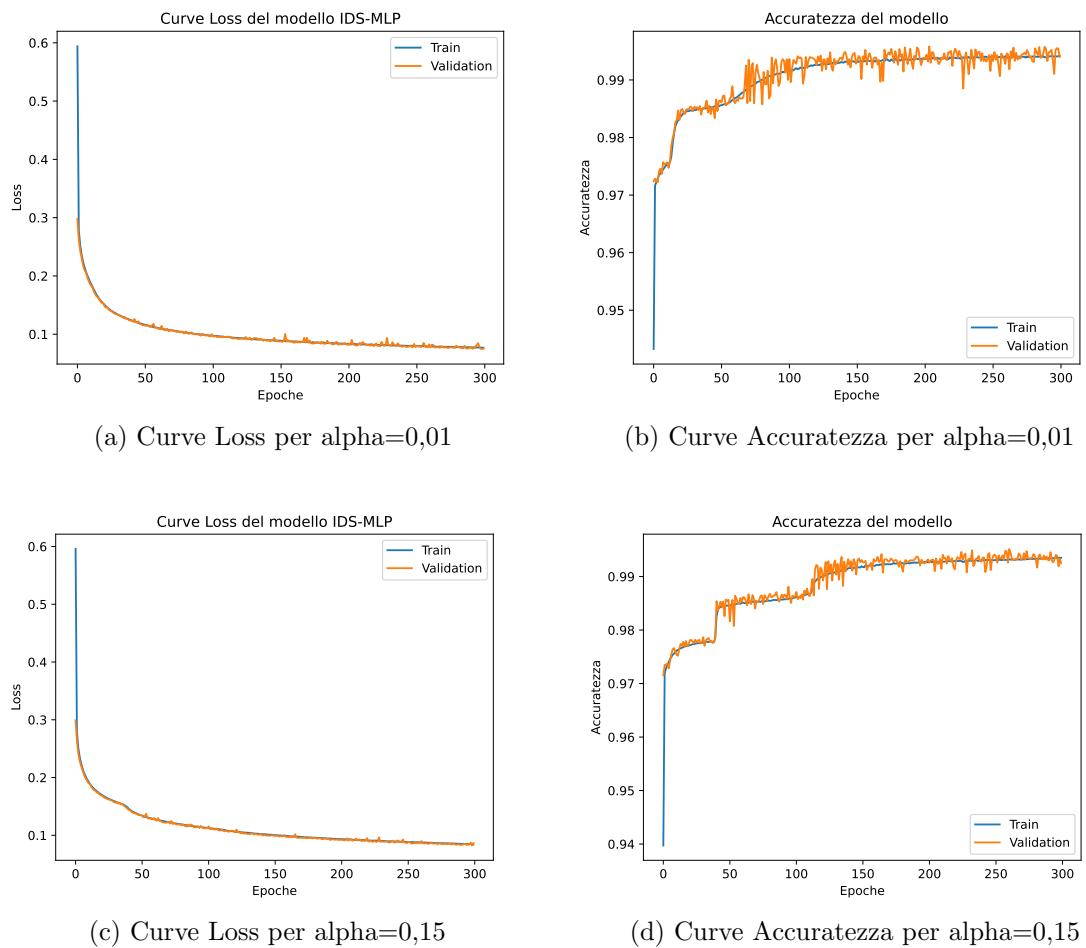


Figura 55: Grafici Curve Loss e Accuratezza della D.A. al 100% di Botnet e Bruteforce senza CW nei due diversi valori di alpha.

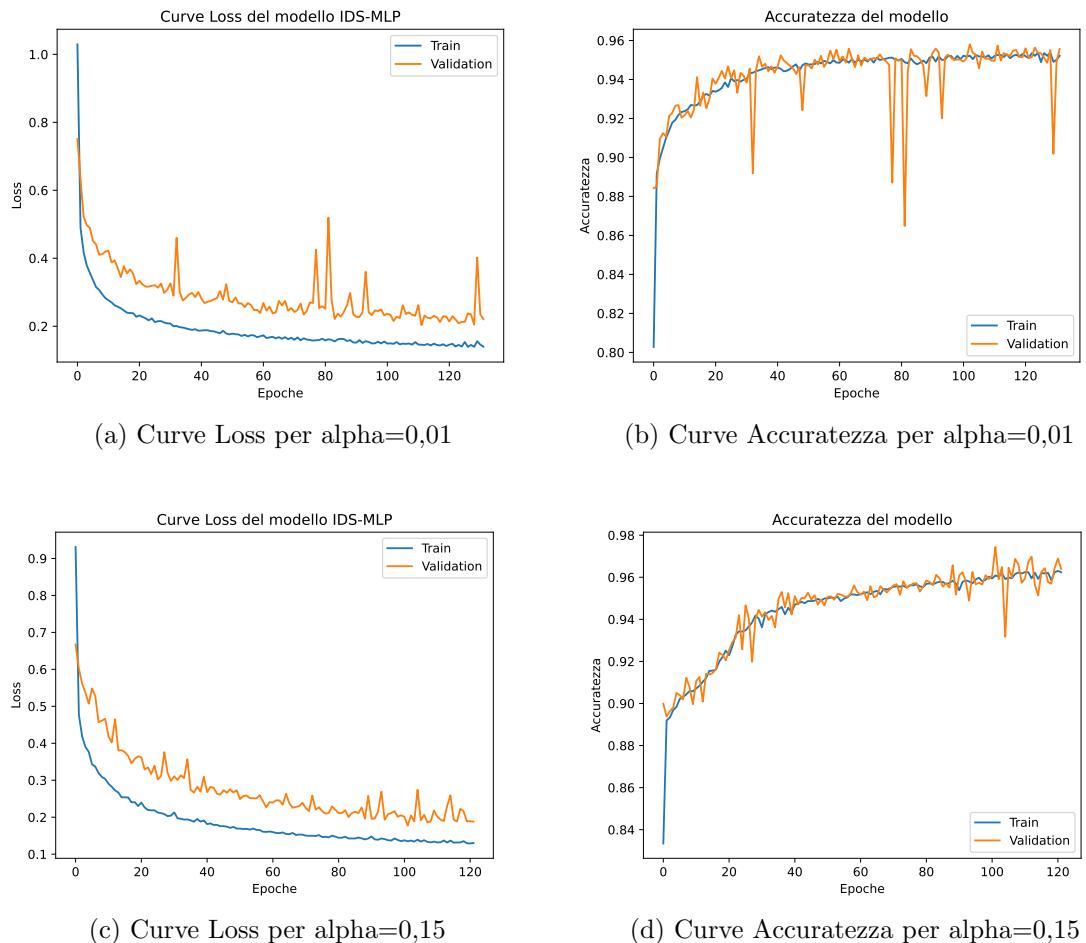


Figura 56: Grafici Curve Loss e Accuratezza della D.A. al 100% di Botnet e Bruteforce con CW nei due diversi valori di alpha.

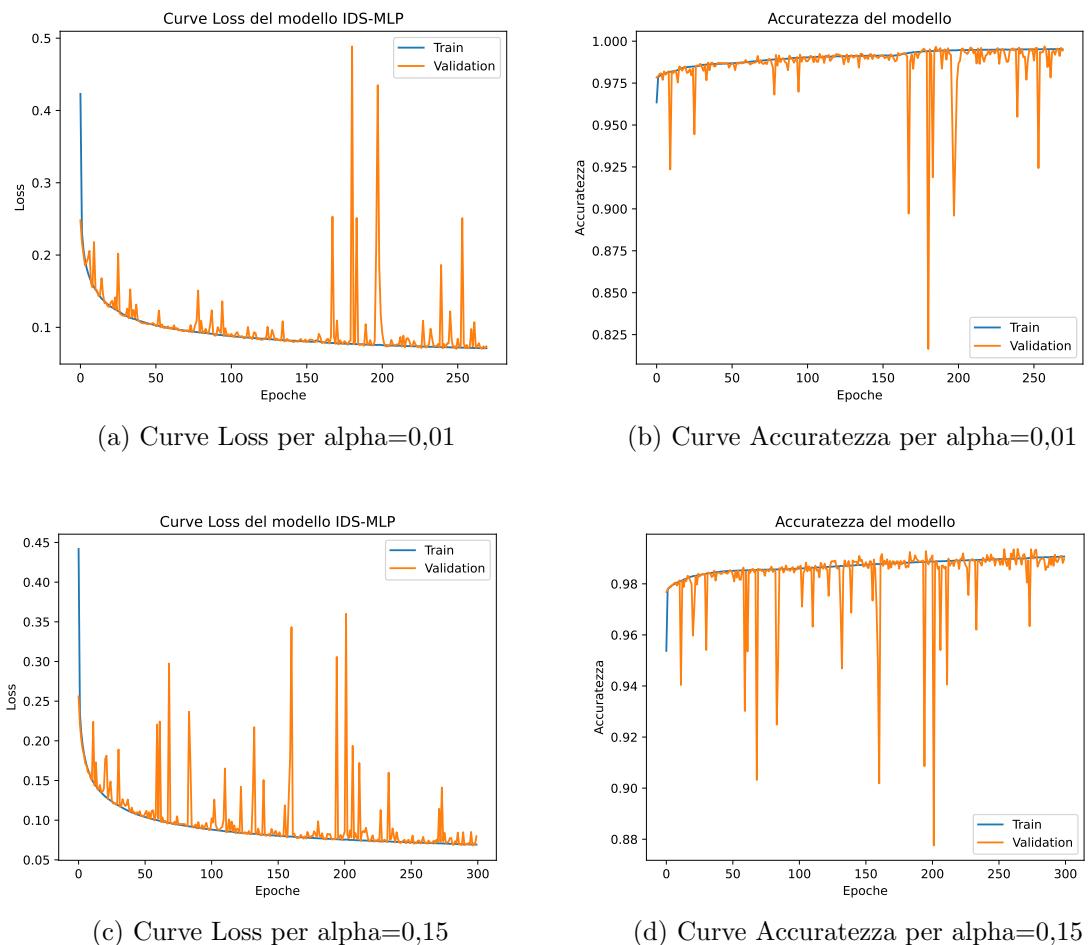


Figura 57: Grafici Curve Loss e Accuratezza della D.A. con aumento di 10000 delle classi minori e 100% delle maggiori senza CW nei due diversi valori di alpha.

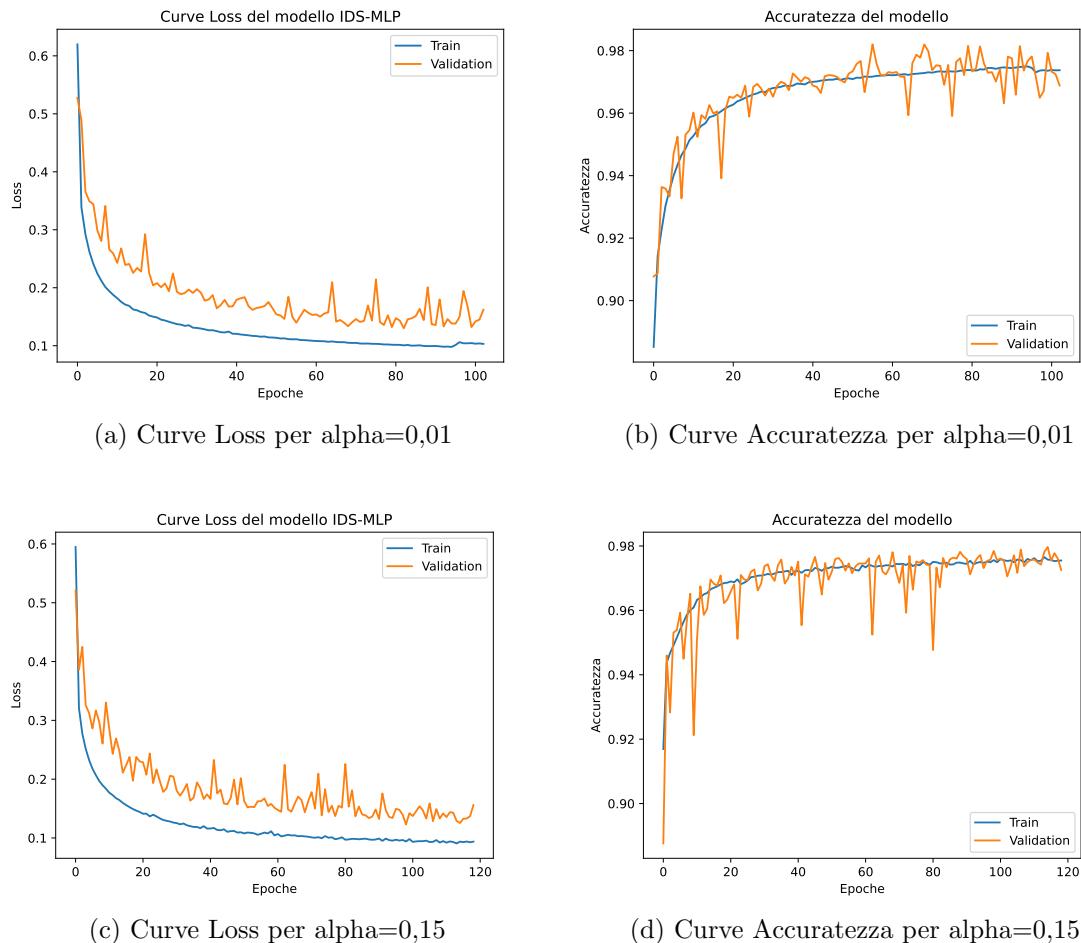


Figura 58: Grafici Curve Loss e Accuratezza della D.A. con aumento di 10000 delle classi minori e 100% delle maggiori con CW nei due diversi valori di alpha.

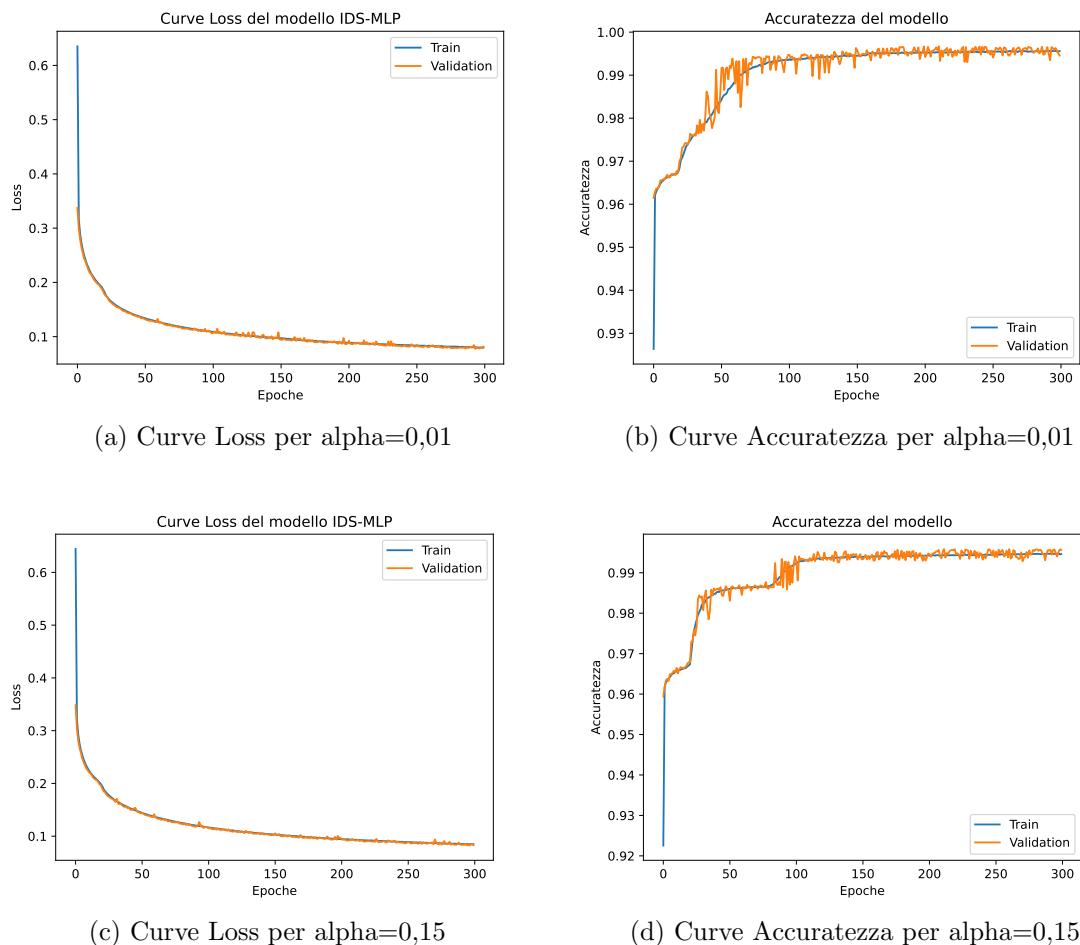


Figura 59: Grafici Curve Loss e Accuratezza della D.A. con aumento di 10000 delle sole classi minori senza CW nei due diversi valori di alpha.

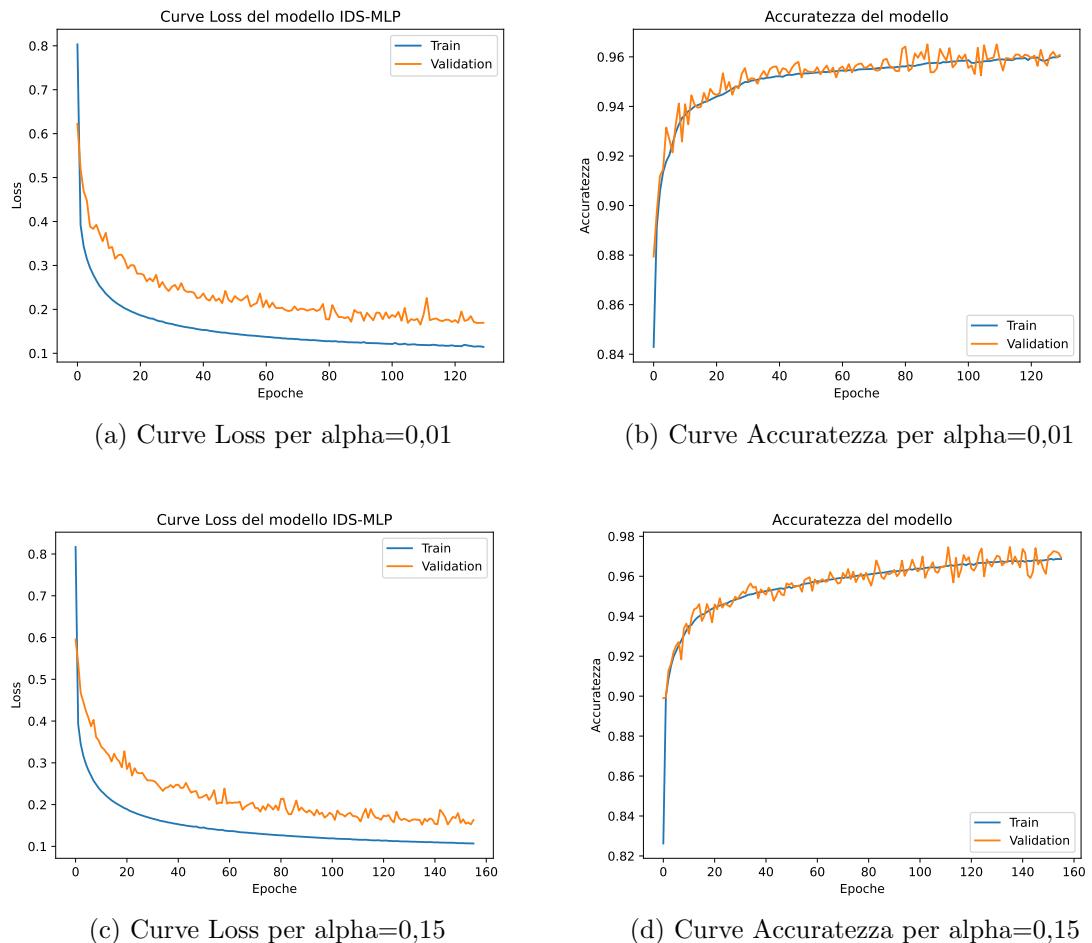


Figura 60: Grafici Curve Loss e Accuratezza della D.A. con aumento di 10000 delle sole classi minori con CW nei due diversi valori di alpha.

Vengono riportati di seguito i grafici delle matrici di confusione relativi alla valutazione della D.A.

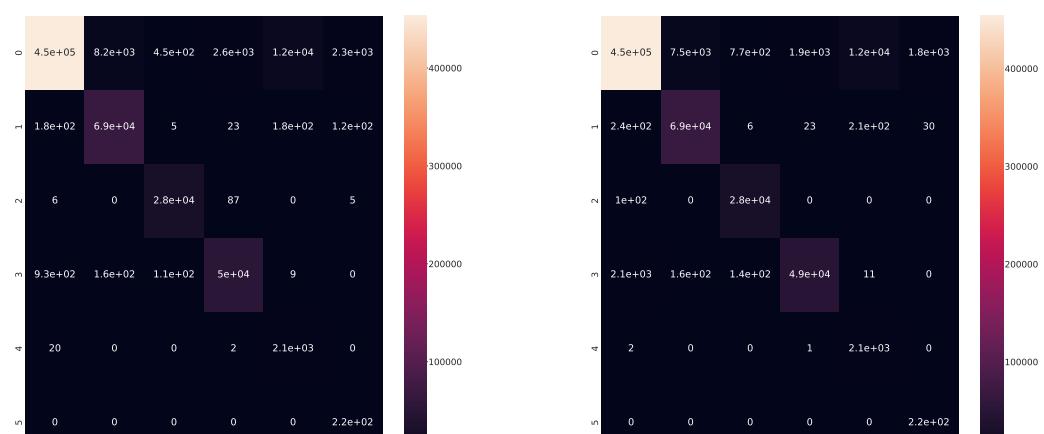
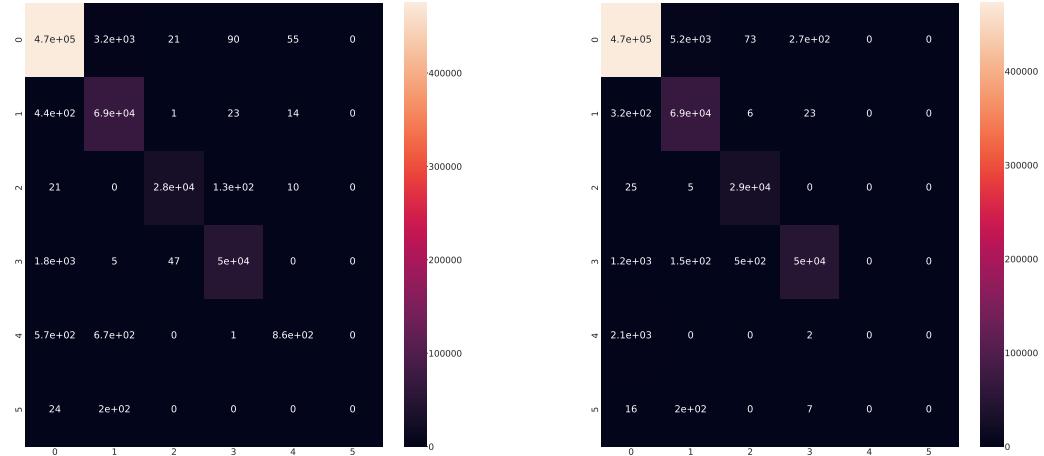


Figura 61: Confronto sulle Matrici di Confusione della D.A. del 100% di tutte le classi su IDS-MLP con e senza CW nei due diversi valori di α .

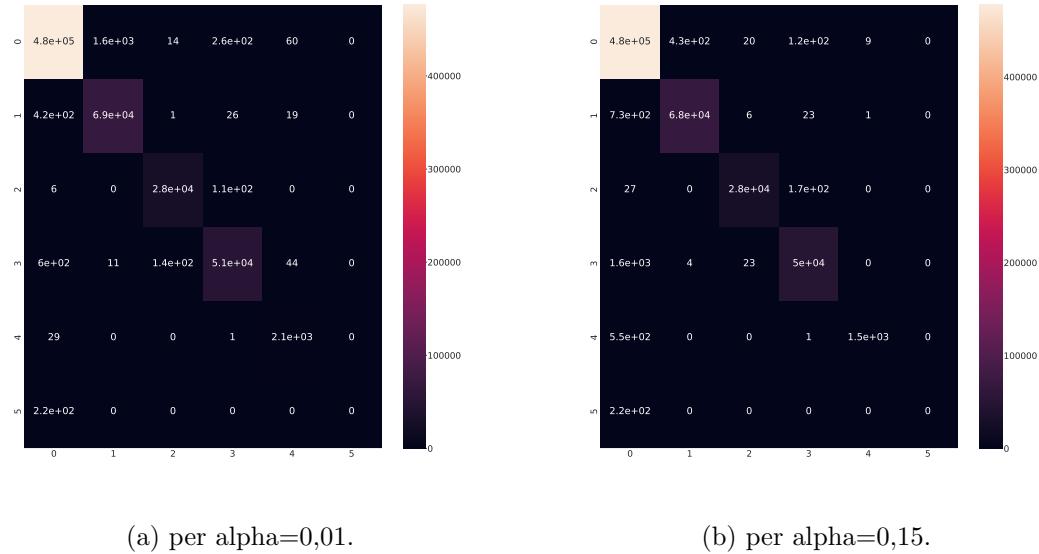


Figura 62: Confronto sulle Matrici di Confusione della D.A. del 100% di Botnet e Bruteforce su IDS-MLP senza CW nei due diversi valori di alpha, in aggiunta alla figura 39.

Di seguito si riportano i grafici relativi alla valutazione della D.A. su tutte le classi al 100% con l'incremento delle sole classi Botnet e Bruteforce a 10000; i risultati di tale analisi sono presenti all'inizio di questo capitolo.

	per alpha=0,01		per alpha=0,15	
	senza CW	con CW	senza CW	con CW
Accuracy	0.992301	0.971073	0.983930	0.957116
Balanced Accuracy	0.986980	0.989795	0.837463	0.984145
Weighted Precision	0.992622	0.979369	0.985875	0.972409
Macro Precision	0.894257	0.712463	0.751144	0.674111
Weighted Recall	0.992301	0.971073	0.983930	0.957116
Macro Recall	0.986980	0.989795	0.837463	0.984145
Weighted F1 score	0.992398	0.973991	0.984490	0.962755
Macro F1 score	0.922841	0.764322	0.781472	0.730759

Tabella 34: Valutazione delle performance di classificazione dell'IDS-MLP aumentando di 10000 samples le classi Botnet e Bruteforce e del 100% le classi maggiori (con e senza CW).

	precision	recall	f1-score	support
0	1.00	0.99	1.00	478327
1	0.97	0.99	0.98	69256
2	1.00	1.00	1.00	28540
3	0.99	0.97	0.98	51486
4	0.98	0.98	0.98	2101
5	0.43	0.98	0.60	221
accuracy			0.99	629931
macro avg	0.89	0.99	0.92	629931
weighted avg	0.99	0.99	0.99	629931

(a) senza CW per alpha=0,01.

	precision	recall	f1-score	support
0	1.00	0.96	0.98	478327
1	0.89	0.99	0.94	69256
2	0.99	1.00	1.00	28540
3	0.94	0.99	0.96	51486
4	0.34	1.00	0.51	2101
5	0.11	1.00	0.20	221
accuracy			0.97	629931
macro avg	0.71	0.99	0.76	629931
weighted avg	0.98	0.97	0.97	629931

(b) con CW per alpha=0,01.

	precision	recall	f1-score	support
0	1.00	0.98	0.99	478327
1	0.93	0.99	0.96	69256
2	1.00	0.99	0.99	28540
3	0.96	0.99	0.97	51486
4	0.51	0.99	0.68	2101
5	0.10	0.08	0.09	221
accuracy			0.98	629931
macro avg	0.75	0.84	0.78	629931
weighted avg	0.99	0.98	0.98	629931

(c) senza CW per alpha=0,15.

	precision	recall	f1-score	support
0	1.00	0.95	0.97	478327
1	0.91	0.99	0.95	69256
2	0.91	1.00	0.95	28540
3	0.89	0.98	0.93	51486
4	0.19	0.99	0.32	2101
5	0.15	1.00	0.26	221
accuracy			0.96	629931
macro avg	0.67	0.98	0.73	629931
weighted avg	0.97	0.96	0.96	629931

(d) con CW per alpha=0,15.

Figura 63: Confronto sui Report di Classificazione della D.A. con aumento a 10000 samples delle classi Botnet e Bruteforce e del 100% sulle classi maggiori con e senza CW.

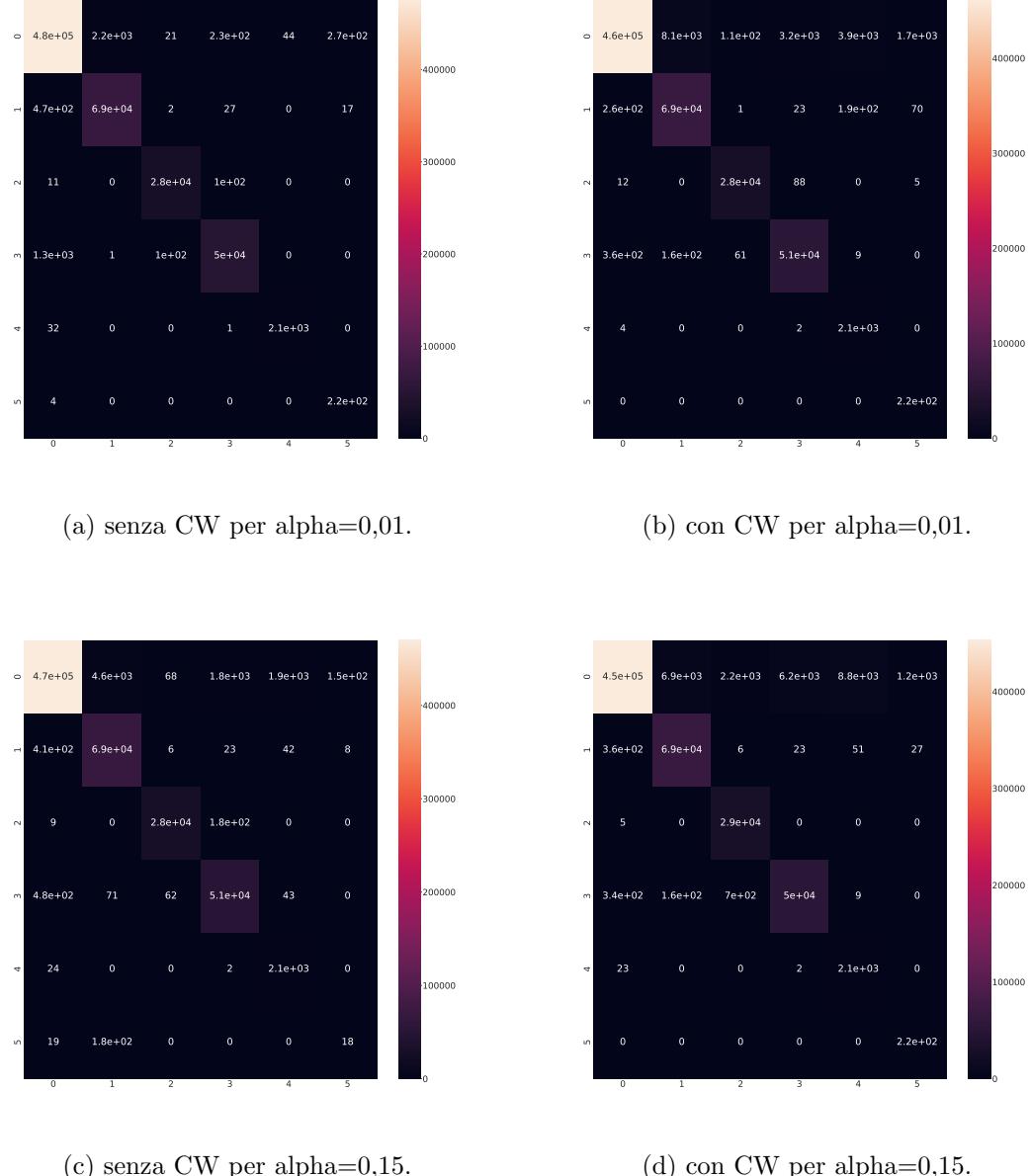
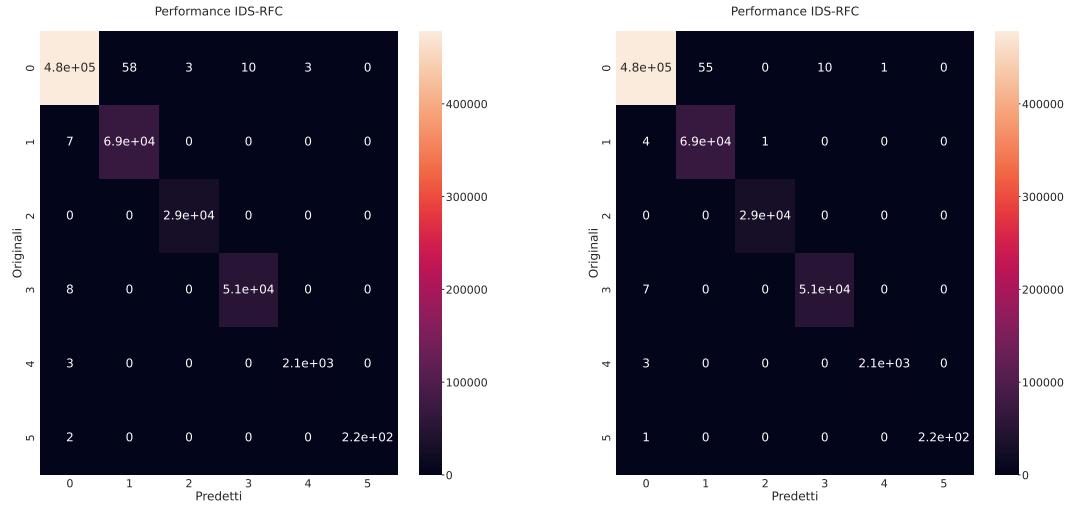


Figura 64: Confronto sulle Matrici di Confusione della D.A. del 100% sulle classi maggiori e di 10000 samples sui minori con e senza CW, nei due diversi valori di alpha.

IDS-RFC

Infine, si analizza la modalità di D.A. con aumento di 10000 samples per le classi minori ed aumento del 100% delle classi maggiori: dai grafici si evince il comportamento già esposto nel capitolo 6.6.1 ovvero una leggera difficoltà di rilevamento delle classi maggiori sempre a causa della non ottimalità dei dati generati dal modello Vanilla GAN. Risulta essere migliore la performance per alpha=0,15 ma l'aumento di 10000 samples per le classi Botnet e Bruteforce non ha introdotto vantaggi evidenti.



(a) D.A. con dati generati per alpha=0,01.

(b) D.A. con dati generati per alpha=0,15.

Figura 65: Confronto D.A. del 100% delle classi maggiori e di 10000 samplese delle due classi minori nei due diversi valori di alpha.

Bibliografia

- [1] ENISA, C. Ardagna, S. Corbiaux, A. Sfakianakis, and C. Douligeris, “Enisa threat landscape 2021,” 10 2021.
- [2] R. Mattioli, A. Malatras, E. N. Hunter, M. G. B. Penso, D. Bertram, I. Neubert, and E. U. A. for Cybersecurity, *Identifying emerging cybersecurity threats and challenges for 2030.*, E. U. A. for Cybersecurity, Ed., 3 2023.
- [3] S. Ntalamp, G. Misuraca, and P. Rossel, “Artificial intelligence and cybersecurity research,” 2023.
- [4] H. N. C. Neto, J. Hribar, I. Dusparic, D. M. F. Mattos, and N. C. Fernandes, “A survey on securing federated learning: Analysis of applications, attacks, challenges, and trends,” *IEEE Access*, vol. 11, pp. 41 928–41 953, 2023. [Online]. Available: <https://ieeexplore.ieee.org/document/10107622/>
- [5] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, “Communication-efficient learning of deep networks from decentralized data,” *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, AISTATS 2017*, 2 2016. [Online]. Available: <http://arxiv.org/abs/1602.05629>
- [6] L. Reznik, *Intelligent Security Systems*. Wiley, 10 2021. [Online]. Available: <https://onlinelibrary.wiley.com/doi/book/10.1002/9781119771579>
- [7] M. Ahmed, S. R. Islam, A. Anwar, N. Moustafa, and A.-S. K. P. Editors, *Explainable Artificial Intelligence for Cyber Security: Next Generation Artificial*

- Intelligence*, 2022, vol. 1025. [Online]. Available: <https://link.springer.com/bookseries/7092>
- [8] S. Abaimov and M. Martellini, *Machine Learning for Cyber Agents*. Springer International Publishing, 2022.
 - [9] ISO/IEC, “Iso 2382:2015 - information technology,” 2015. [Online]. Available: <https://www.iso.org/obp/ui/en/#iso:std:iso-iec:2382:ed-1:v2:en>
 - [10] E. Treccani, “Machine learning,” 2019, istituto della Enciclopedia Italiana fondata da Giovanni Treccani. [Online]. Available: [https://www.treccani.it/vocabolario/machine-learning_\(Neologismi\)/](https://www.treccani.it/vocabolario/machine-learning_(Neologismi)/)
 - [11] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. [Online]. Available: <http://www.deeplearningbook.org>
 - [12] Bishop, M. Jordan, J. Kleinberg, and B. Schölkopf, *Pattern Recognition and Machine Learning*, 2006.
 - [13] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention is all you need,” 6 2017. [Online]. Available: <http://arxiv.org/abs/1706.03762>
 - [14] S. Raschka, “Understanding and coding the self-attention mechanism,” 2 2023. [Online]. Available: <https://sebastianraschka.com/blog/2023/self-attention-from-scratch.html>
 - [15] P. F. de Araujo-Filho, M. Naili, G. Kaddoum, E. T. Fapi, and Z. Zhu, “Unsupervised gan-based intrusion detection system using temporal convolutional networks and self-attention,” *IEEE Transactions on Network and Service Management*, pp. 1–1, 2023. [Online]. Available: <https://ieeexplore.ieee.org/document/10077779/>
 - [16] J. Deng, R. Guo, and Z. Jin, “An intrusion detection scheme based on federated learning and self-attention fusion convolutional neural network for iot,” *Journal on Internet of Things*, vol. 4, pp. 141–153, 2022.

- [17] H. Zhang, I. Goodfellow, D. Metaxas, and A. Odena, “Self-attention generative adversarial networks,” *36th International Conference on Machine Learning, ICML 2019*, vol. 2019-June, pp. 12 744–12 753, 5 2018. [Online]. Available: <http://arxiv.org/abs/1805.08318>
- [18] P. Dini, A. Elhanashi, A. Begni, S. Saponara, Q. Zheng, and K. Gasmi, “Overview on intrusion detection systems design exploiting machine learning for networking cybersecurity,” *Applied Sciences*, vol. 13, p. 7507, 6 2023. [Online]. Available: <https://www.mdpi.com/2076-3417/13/13/7507>
- [19] M. A. Manjramkar and K. C. Jondhale, *Cyber Security Using Machine Learning Techniques*, 2023, pp. 680–701. [Online]. Available: https://www.atlantis-press.com/doi/10.2991/978-94-6463-136-4_59
- [20] E. Alhajjar, P. Maxwell, and N. Bastian, “Adversarial machine learning in network intrusion detection systems,” *Expert Systems with Applications*, vol. 186, 12 2021.
- [21] A.-G. Mari, D. Zinca, and V. Dobrota, “Development of a machine-learning intrusion detection system and testing of its performance using a generative adversarial network,” *Sensors*, vol. 23, p. 1315, 1 2023. [Online]. Available: <https://www.mdpi.com/1424-8220/23/3/1315>
- [22] M. Macas, C. Wu, and W. Fuertes, “A survey on deep learning for cybersecurity: Progress, challenges, and opportunities,” *Computer Networks*, vol. 212, p. 109032, 7 2022. [Online]. Available: <https://linkinghub.elsevier.com/retrieve/pii/S1389128622001864>
- [23] N. Abdalgawad, A. Sajun, Y. Kaddoura, I. A. Zualkernan, and F. Aloul, “Generative deep learning to detect cyberattacks for the iot-23 dataset,” *IEEE Access*, vol. 10, pp. 6430–6441, 2022.
- [24] C. Strickland, C. Saha, M. Zakar, S. Nejad, N. Tasnim, D. Lizotte, and A. Haque, “Drl-gan: A hybrid approach for binary and multiclass network intrusion detection,” 1 2023. [Online]. Available: <http://arxiv.org/abs/2301.03368>

- [25] G. Research, “Federated learning: Collaborative machine learning without centralized training data,” 4 2017. [Online]. Available: <https://blog.research.google/2017/04/federated-learning-collaborative.html>
- [26] Google, “Gboard.” [Online]. Available: <https://support.google.com/gboard/answer/12373137?hl=en#zippy=%2Cfederated-learning>
- [27] M. Alazab, S. P. Rm, M. Parimala, P. K. R. Maddikunta, T. R. Gadekallu, and Q. V. Pham, “Federated learning for cybersecurity: Concepts, challenges, and future directions,” *IEEE Transactions on Industrial Informatics*, vol. 18, pp. 3501–3509, 5 2022.
- [28] J. Zhang, B. Chen, S. Yu, and H. Deng, “Pefl: A privacy-enhanced federated learning scheme for big data analytics.” IEEE, 12 2019, pp. 1–6. [Online]. Available: <https://ieeexplore.ieee.org/document/9014272/>
- [29] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, “Federated learning: Challenges, methods, and future directions,” *IEEE Signal Processing Magazine*, vol. 37, pp. 50–60, 8 2019. [Online]. Available: <http://arxiv.org/abs/1908.07873http://dx.doi.org/10.1109/MSP.2020.2975749>
- [30] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, “Federated optimization in heterogeneous networks,” *Arxiv.Org*, 12 2018. [Online]. Available: <http://arxiv.org/abs/1812.06127>
- [31] L. Lavaur, M. O. Pahl, Y. Busnel, and F. Autrel, “The evolution of federated learning-based intrusion detection and mitigation: A survey,” *IEEE Transactions on Network and Service Management*, vol. 19, pp. 2309–2332, 9 2022.
- [32] W. Zhang, Q. Lu, Q. Yu, Z. Li, Y. Liu, S. K. Lo, S. Chen, X. Xu, and L. Zhu, “Blockchain-based federated learning for device failure detection in industrial iot,” 9 2020. [Online]. Available: <http://arxiv.org/abs/2009.02643>
- [33] L. WANG, W. WANG, and B. LI, “Cmfl: Mitigating communication overhead for federated learning,” vol. 2019-July. IEEE, 7 2019, pp. 954–964. [Online]. Available: <https://ieeexplore.ieee.org/document/8885054/>

- [34] M. Asad, A. Moustafa, T. Ito, and M. Aslam, “Evaluating the communication efficiency in federated learning algorithms.” Institute of Electrical and Electronics Engineers Inc., 5 2021, pp. 552–557.
- [35] M. yuan Zhu, Z. Chen, K. fan Chen, N. Lv, and Y. Zhong, “Attention-based federated incremental learning for traffic classification in the internet of things,” *Computer Communications*, vol. 185, pp. 168–175, 3 2022.
- [36] S. Caldas, J. Konečny, H. B. McMahan, and A. Talwalkar, “Expanding the reach of federated learning by reducing client resource requirements,” 12 2018.
- [37] X. Yao, T. Huang, C. Wu, R.-X. Zhang, and L. Sun, “Federated learning with additional mechanisms on clients to reduce communication costs,” 8 2019.
- [38] J. Zhang, J. Chen, D. Wu, B. Chen, and S. Yu, “Poisoning attack in federated learning using generative adversarial nets,” 2019.
- [39] M. Goldblum, D. Tsipras, C. Xie, X. Chen, A. Schwarzchild, D. Song, A. Madry, B. Li, and T. Goldstein, “Dataset security for machine learning: Data poisoning, backdoor attacks, and defenses,” 12 2020. [Online]. Available: <http://arxiv.org/abs/2012.10544>
- [40] B. Hitaj, G. Ateniese, and F. Perez-Cruz, “Deep models under the gan: Information leakage from collaborative deep learning.” ACM, 10 2017, pp. 603–618. [Online]. Available: <https://dl.acm.org/doi/10.1145/3133956.3134012>
- [41] R. Taheri, M. Shojafar, M. Alazab, and R. Tafazolli, “Fed-iiot: A robust federated malware detection architecture in industrial iot,” *IEEE Transactions on Industrial Informatics*, 2020.
- [42] Y. Zhao, J. Chen, J. Zhang, D. Wu, M. Blumenstein, and S. Yu, “Detecting and mitigating poisoning attacks in federated learning using generative adversarial networks,” vol. 34. John Wiley and Sons Ltd, 3 2022.
- [43] H. Zhu, J. Xu, S. Liu, and Y. Jin, “Federated learning on non-iid data: A survey,” 6 2021. [Online]. Available: <http://arxiv.org/abs/2106.06843>

- [44] N. H. Quyen, P. T. Duy, N. C. Vy, D. T. T. Hien, and V.-H. Pham, *Federated Intrusion Detection on Non-IID Data for IIoT Networks Using Generative Adversarial Networks and Reinforcement Learning*. Springer Science and Business Media Deutschland GmbH, 2022, vol. 13620 LNCS, pp. 364–381. [Online]. Available: https://link.springer.com/10.1007/978-3-031-21280-2_20
- [45] A. Tabassum, A. Erbad, W. Lebda, A. Mohamed, and M. Guizani, “Fedgan-ids: Privacy-preserving ids using gan and federated learning,” *Computer Communications*, vol. 192, pp. 299–310, 8 2022. [Online]. Available: <https://linkinghub.elsevier.com/retrieve/pii/S0140366422002171>
- [46] M. J. Idrissi, H. Alami, A. E. Mahdaouy, A. E. Mekki, S. Oualil, Z. Yartaoui, and I. Berrada, “Fed-anids: Federated learning for anomaly-based network intrusion detection systems,” *Expert Systems with Applications*, vol. 234, p. 121000, 12 2023. [Online]. Available: <https://linkinghub.elsevier.com/retrieve/pii/S0957417423015026>
- [47] W. Wang, M. Zhu, X. Zeng, X. Ye, and Y. Sheng, “Malware traffic classification using convolutional neural network for representation learning,” in *2017 International Conference on Information Networking (ICOIN)*, Jan 2017, pp. 712–717.
- [48] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, “Toward generating a new intrusion detection dataset and intrusion traffic characterization.” SCITEPRESS - Science and Technology Publications, 2018, pp. 108–116. [Online]. Available: <https://www.unb.ca/cic/datasets/ids-2018.htmlhttps://registry.opendata.aws/cse-cic-ids2018/>
- [49] J. Chen, Q. Guo, Z. Fu, Q. Shang, H. Ma, and D. Wu, “Campus network intrusion detection based on federated learning,” vol. 2022-July. Institute of Electrical and Electronics Engineers Inc., 2022.

- [50] A. Belenguer, J. A. Pascual, and J. Navaridas, “Göwfed: A novel federated network intrusion detection system,” *Journal of Network and Computer Applications*, vol. 217, 8 2023, cita FEDGAN ma solo come statoDell’Arte... bella intro e alcune precisazioni su tools simulazione.
- [51] M. Poongodi and M. Hamdi, “Intrusion detection system using distributed multilevel discriminator in gan for iot system,” *Transactions on Emerging Telecommunications Technologies*, 2023.
- [52] J. Chen, Y. Zhao, Q. Li, X. Feng, and K. Xu, “Feddef: Defense against gradient leakage in federated learning-based network intrusion detection systems,” 10 2022. [Online]. Available: <http://arxiv.org/abs/2210.04052><http://dx.doi.org/10.1109/TIFS.2023.3297369>
- [53] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial networks,” 6 2014. [Online]. Available: <http://arxiv.org/abs/1406.2661>
- [54] A. Dunmore, J. Jang-Jaccard, F. Sabrina, and J. Kwak, “A comprehensive survey of generative adversarial networks (gans) in cybersecurity intrusion detection,” *IEEE Access*, vol. 11, pp. 76 071–76 094, 2023. [Online]. Available: <https://ieeexplore.ieee.org/document/10187144/>
- [55] T. S. Silva, “A short introduction to generative adversarial networks,” <https://sthalles.github.io>, 2017. [Online]. Available: <https://sthalles.github.io/intro-to-gans/>
- [56] R. Guerraoui, A. Guirguis, A.-M. Kermarrec, and E. L. Merrer, “Fegan: Scaling distributed gans.” ACM, 12 2020, pp. 193–206. [Online]. Available: <https://dl.acm.org/doi/10.1145/3423211.3425688>
- [57] M. Mirza and S. Osindero, “Conditional generative adversarial nets,” 11 2014. [Online]. Available: <http://arxiv.org/abs/1411.1784>

- [58] A. Radford, L. Metz, and S. Chintala, “Unsupervised representation learning with deep convolutional generative adversarial networks.” International Conference on Learning Representations, ICLR, 2016.
- [59] M. Arjovsky, S. Chintala, and L. Bottou, “Wasserstein generative adversarial networks,” in *Proceedings of the 34th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, D. Precup and Y. W. Teh, Eds., vol. 70. PMLR, 06–11 Aug 2017, pp. 214–223. [Online]. Available: <https://proceedings.mlr.press/v70/arjovsky17a.html>
- [60] L. Xu, M. Skoularidou, A. Cuesta-Infante, and K. Veeramachaneni, “Modeling tabular data using conditional gan,” 6 2019. [Online]. Available: <http://arxiv.org/abs/1907.00503>
- [61] N. Patki, R. Wedge, and K. Veeramachaneni, “The synthetic data vault.” IEEE, 10 2016, pp. 399–410. [Online]. Available: <http://ieeexplore.ieee.org/document/7796926/>
- [62] S. Bourou, A. E. Saer, T.-H. Velivassaki, A. Voulkidis, and T. Zahariadis, “A review of tabular data synthesis using gans on an ids dataset,” *Information*, vol. 12, p. 375, 9 2021. [Online]. Available: <https://www.mdpi.com/2078-2489/12/9/375>
- [63] M. Ring, D. Schlör, D. Landes, and A. Hotho, “Flow-based network traffic generation using generative adversarial networks,” *Computers and Security*, vol. 82, 2019.
- [64] J. Yang, T. Li, G. Liang, W. He, and Y. Zhao, “A simple recurrent unit model based intrusion detection system with dcgan,” *IEEE Access*, vol. 7, pp. 83 286–83 296, 2019.
- [65] C. Park, J. Lee, Y. Kim, J. G. Park, H. Kim, and D. Hong, “An enhanced ai-based network intrusion detection system using generative adversarial networks,” *IEEE Internet of Things Journal*, vol. 10, pp. 2330–2345, 2 2023.

- [66] M. Arafah, I. Phillips, and A. Adnane, “Evaluating the impact of generative adversarial models on the performance of anomaly intrusion detection,” *IET Networks*, 8 2023. [Online]. Available: <https://ietresearch.onlinelibrary.wiley.com/doi/10.1049/ntw2.12098>
- [67] L. Liu, G. Engelen, T. Lynar, D. Essam, and W. Joosen, “Error prevalence in nids datasets: A case study on cic-ids-2017 and cse-cic-ids-2018.” IEEE, 10 2022, pp. 254–262. [Online]. Available: <https://intrusion-detection.distrinet-research.be/CNS2022/>
- [68] M. Ali, M. ul Haque, M. H. Durad, A. Usman, S. M. Mohsin, H. Mujlid, and C. Maple, “Effective network intrusion detection using stacking-based ensemble approach,” *International Journal of Information Security*, vol. 22, pp. 1781–1798, 12 2023. [Online]. Available: <https://github.com/muhammad-ali/ML-NIDS/>
- [69] Kurniabudi, D. Stiawan, Darmawijoyo, M. Y. B. Idris, A. M. Bamhdi, and R. Budiarto, “Cicids-2017 dataset feature analysis with information gain for anomaly detection,” *IEEE Access*, vol. 8, pp. 132 911–132 921, 2020. [Online]. Available: <https://ieeexplore.ieee.org/document/9142219/>
- [70] S. Li, Q. Li, and M. Li, “A method for network intrusion detection based on gan-cnn-bilstm,” *International Journal of Advanced Computer Science and Applications*, vol. 14, pp. 507–515, 2023.
- [71] B. Reis, E. Maia, and I. Praça, *Selection and Performance Analysis of CICIDS2017 Features Importance*, 2020, pp. 56–71. [Online]. Available: http://link.springer.com/10.1007/978-3-030-45371-8_4
- [72] N. Peppes, T. Alexakis, K. Demestichas, and E. Adamopoulou, “A comparison study of generative adversarial network architectures for malicious cyber-attack data generation,” *Applied Sciences*, vol. 13, p. 7106, 6 2023. [Online]. Available: <https://www.mdpi.com/2076-3417/13/12/7106>
- [73] M. Bacevicius and A. Paulauskaite-Taraseviciene, “Machine learning algorithms for raw and unbalanced intrusion detection data in a multi-class classification

- problem,” *Applied Sciences*, vol. 13, p. 7328, 6 2023. [Online]. Available: <https://www.mdpi.com/2076-3417/13/12/7328>
- [74] Y. Liu, G. Wu, W. Zhang, and J. Li, “Federated learning-based intrusion detection on non-iid data,” vol. 13777 LNCS. Springer Science and Business Media Deutschland GmbH, 2023, pp. 313–329.
- [75] C. Swart, “Using generative adversarial networks to improve malware detection in the iot focused intrusion detection dataset iot-23,” 2022. [Online]. Available: <https://github.com/CameronSwart/GAN-IDS>
- [76] A. Borji, “Pros and cons of gan evaluation measures,” 2 2018. [Online]. Available: <http://arxiv.org/abs/1802.03446>
- [77] M. Castelli, L. Manzoni, T. Espindola, A. Popović, and A. D. Lorenzo, “Generative adversarial networks for generating synthetic features for wi-fi signal quality,” *PLOS ONE*, vol. 16, p. e0260308, 11 2021. [Online]. Available: <https://dx.plos.org/10.1371/journal.pone.0260308>
- [78] G. J, “Using class weight to compensate for imbalanced data,” 7 2022. [Online]. Available: https://medium.com/@bubbapora_76246/using-class-weight-to-compensate-for-imbalanced-data-6eff370185d3
- [79] L. Breiman, “Random forests,” *Machine Learning*, vol. 45, pp. 5–32, 2001.
- [80] V. Bulavas, V. Marcinkevičius, and J. Rumiński, “Study of multi-class classification algorithms’ performance on highly imbalanced network intrusion datasets,” *Informatica*, pp. 441–475, 9 2021. [Online]. Available: <https://informatica.vu.lt/doi/10.15388/21-INFOR457>
- [81] M. A. Olaimat, D. Lee, Y. Kim, J. Kim, and J. Kim, “A learning-based data augmentation for network anomaly detection.” IEEE, 8 2020, pp. 1–10. [Online]. Available: <https://ieeexplore.ieee.org/document/9209598/>
- [82] A. S. Barkah, S. R. Selamat, Z. Z. Abidin, and R. Wahyudi, “Data generative model to detect the anomalies for ids imbalance cicids2017

- dataset,” *TEM Journal*, pp. 80–89, 2 2023. [Online]. Available: https://www.temjournal.com/content/121/TEMJournalFebruary2023_80_89.html
- [83] A. Alabrah, “A novel study: Gan-based minority class balancing and machine-learning-based network intruder detection using chi-square feature selection,” *Applied Sciences*, vol. 12, p. 11662, 11 2022. [Online]. Available: <https://www.mdpi.com/2076-3417/12/22/11662>
- [84] Z. Ma, J. Li, Y. Song, X. Wu, and C. Chen, “Network intrusion detection method based on fcwgan and bilstm,” *Computational Intelligence and Neuroscience*, vol. 2022, pp. 1–17, 4 2022. [Online]. Available: <https://www.hindawi.com/journals/cin/2022/6591140/>
- [85] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, “Smote: Synthetic minority over-sampling technique,” *Journal of Artificial Intelligence Research*, vol. 16, pp. 321–357, 6 2002.

Ringraziamenti

Innanzitutto, ringrazio il mio relatore per esser sempre stato disponibile ed avermi trasmesso calma in questi lunghi mesi di lavoro. Il supporto sia accademico che a livello umano ricevuto è stato per me molto importante.

Inoltre, ringrazio il mio migliore amico Simone per essermi stato vicino ed avermi supportato tutto questo tempo. Grazie per avermi aiutato e consigliato durante questi lunghi mesi. Ringrazio la mia amica Giovanna per tutte le parole gentili e di conforto che mi hanno sempre fatto sentire bene.

Ringrazio i miei genitori per essermi stati vicino in questi mesi difficili e pieni di insicurezze. Li ringrazio per avermi trasmesso fiducia e per tutte le parole di incoraggiamento di cui mi hanno inondato in questi mesi. Farò tesoro di tutto quello che mi avete detto per affrontare il futuro. Grazie mamma e papà per esser costantemente fonte di supporto e coraggio nella mia vita, grazie per la vostra immensa pazienza. Vi voglio molto bene.

Un grande grazie lo rivolgo anche a tutti gli zii ed i cugini per il sostegno ricevuto.

Infine, prendo un piccolo spazio per parlare alla me del futuro: ti auguro di vivere giorno per giorno con positività e serenità. Rialzati dalle cadute come hai sempre fatto, come i tuoi genitori ti hanno insegnato. Hai superato tanti ostacoli e stai lavorando molto su te stessa. In bocca al lupo.

*We face stormy weather
Rain's pouring but it never lasts
Only gets better from here
... Just take it slow, one day at a time*

Milano, Luglio 2024

Giulia Francesca Contardi

Codici della sperimentazione, excel, dataset e varie risorse sono disponibili al link:
https://github.com/GiuliaFContardi/FL_IDS-GAN.git