Technical Report Dettagliato

Titolo: Implementazione e Analisi di Modelli ML/DL per il Rilevamento di Minacce – Report Tecnico

Autore: Iacopo Guagni
Riferimento: Lavoro sperimentale a supporto della tesi "Rilevamento proattivo delle minacce informatiche tramite machine learning e deep learning"

Indice

Abstract

Questo report tecnico documenta in dettaglio l'implementazione, l'analisi e i risultati di tre casi di studio sperimentali condotti a supporto della tesi di laurea "Rilevamento proattivo delle minacce informatiche tramite machine learning e deep learning". Gli esperimenti esplorano l'efficacia di diversi paradigmi di apprendimento automatico su compiti chiave della cybersecurity. Il primo caso di studio confronta un modello classico, Random Forest, con un'architettura di Deep Learning generica, Multi-Layer Perceptron, nel rilevamento di intrusioni di rete. Il secondo impiega una Rete Neurale Convoluzionale (CNN) per la classificazione visuale di 25 famiglie di malware. Il terzo, infine, utilizza una Rete Neurale Ricorrente con strati LSTM Bidirezionali per il rilevamento di anomalie in log di sistema, analizzando il compromesso tra affidabilità e completezza del rilevamento. Per ogni esperimento, viene fornita un'analisi dettagliata del codice, della pipeline di pre-processing, dei risultati numerici e dei grafici generati.

---

1. Ambiente e Metodologia Comune

Questa sezione descrive l'ambiente di sviluppo, gli strumenti e i principi metodologici comuni a tutti e tre i casi di studio presentati in questo documento, al fine di garantire la massima chiarezza e riproducibilità.

1.1. Ambiente di Sviluppo

Tutti gli esperimenti sono stati sviluppati e testati in un ambiente basato su linguaggio Python. L'ambiente di sviluppo primario è stato un sistema operativo **Ubuntu 22.04 LTS** con **Python 3.9**, utilizzando l'ambiente di sviluppo integrato (IDE) **PyCharm Professional (v. 2024.1.4)** per la scrittura e il debug del codice. Per garantire la portabilità e la facilità di esecuzione, il codice è stato validato anche su **Windows 11 Pro** e adattato in notebook interattivi (.ipynb) per l'esecuzione su **Google Colaboratory**.

Le principali librerie scientifiche e di machine learning impiegate sono riassunte nella tabella seguente. Le versioni specifiche sono documentate nel file requirements.txt presente nel repository del progetto.

Libreria	Versione Approssimativa	Scopo Principale
TensorFlow	2.15+	Framework principale per il Deep Learning (Keras).
Scikit-learn	1.3+	Toolkit per il Machine Learning classico e metriche di valutazione.

Pandas	2.0+	Manipolazione e analisi dei dati (DataFrame).
NumPy	1.25+	Calcolo numerico e operazioni su array.
Matplotlib	3.7+	Libreria base per la generazione di grafici e visualizzazioni.
Seaborn	0.12+	Visualizzazione statistica avanzata (heatmap, distplot).

1.2. Codice Sorgente e Riproducibilità

Per aderire ai principi della scienza aperta e garantire la massima trasparenza, tutto il codice sorgente originale, i notebook e i file di configurazione sono stati resi pubblicamente disponibili sotto licenza MIT.

Repository GitHub: https://github.com/iacopoooo/Tesi

Al fine di garantire la **riproducibilità** degli esperimenti, è stato implementato un blocco di codice all'inizio di ogni script per fissare tutte le principali fonti di casualità (stocasticità) intrinseche nel processo di addestramento dei modelli. Questo passaggio è cruciale perché operazioni come l'inizializzazione dei pesi di una rete neurale, il dropout o la suddivisione dei dati hanno una componente casuale che, se non controllata, porterebbe a risultati leggermente diversi ad ogni esecuzione.

Snippet di codice per la riproducibilità:

```
import os
import random
import numpy as np
import tensorflow as tf

seed_value = 43 # Un valore intero fisso

os.environ['PYTHONHASHSEED'] = str(seed_value)
random.seed(seed_value)
np.random.seed(seed_value)
tf.random.set_seed(seed_value)
```

Questo blocco imposta un seme fisso per Python, NumPy e TensorFlow, assicurando che chiunque esegua il codice con lo stesso ambiente ottenga risultati identici.

1.3. Metriche di Valutazione

Data la natura fortemente sbilanciata dei dataset in ambito cybersecurity (dove i campioni "normali" superano di gran lunga quelli "anomali"), la sola accuratezza (Accuracy) è una metrica insufficiente e spesso fuorviante. Pertanto, la valutazione dei modelli in questo report si basa su un insieme di metriche più informativo, derivato dalla matrice di confusione:

- **Precisione (Precision):** Misura l'affidabilità delle predizioni positive. Risponde alla domanda: "Quando il modello segnala un'anomalia, quanto spesso ha ragione?". Un'alta precisione è fondamentale per minimizzare i falsi allarmi (False Positives) e il conseguente "alert fatigue" per gli analisti.
- **Recall (Sensibilità):** Misura la completezza del rilevamento. Risponde alla domanda: "Di tutte le anomalie reali presenti, quante ne ha identificate il modello?". Un alto recall è vitale per minimizzare i falsi negativi (False Negatives), ovvero le minacce non rilevate.
- **F1-Score:** È la media armonica di Precisione e Recall. Fornisce un singolo valore che bilancia entrambi gli obiettivi, particolarmente utile quando si cerca un compromesso tra la riduzione dei falsi allarmi e quella delle minacce mancate.
- **ROC AUC (Area Under the Receiver Operating Characteristic Curve):** Misura la capacità complessiva del modello di distinguere tra la classe positiva e quella negativa, indipendentemente dalla soglia di classificazione scelta. Un valore di 1.0 indica un classificatore perfetto, mentre 0.5 indica un classificatore casuale.

---

2. Caso di Studio 1: Rilevamento Intrusioni con RF e MLP

2.1. Obiettivo

L'obiettivo di questo primo caso di studio è condurre un confronto diretto tra un modello di Machine Learning classico, la **Random Forest (RF)**, e un'architettura di Deep Learning fondamentale, il **Multi-Layer Perceptron (MLP)**. L'esperimento mira a valutare le loro performance su un compito di classificazione multi-classe di traffico di rete, utilizzando un dataset tabulare con feature già ingegnerizzate.

2.2. Dataset e Pipeline di Pre-processing

Dataset: CIC-IDS-2017

È stato utilizzato il dataset **CIC-IDS-2017**, un benchmark ampiamente riconosciuto per la ricerca nel campo dei Network Intrusion Detection Systems (NIDS). A causa delle notevoli dimensioni del file originale (oltre 2.8 milioni di record), e per garantire la stabilità e la riproducibilità degli esperimenti anche in ambienti con memoria limitata come Google Colab, è stato creato un **sottoinsieme di lavoro stabile**. Questo è stato generato eseguendo un campionamento casuale stratificato del 10% del dataset completo. Per esigenze di spazio potete scaricare la versione ufficiale, o delle versioni compatte da qui: https://www.unb.ca/cic/datasets/ids-2017.html

Il campione finale include traffico benigno e quattro diverse categorie di attacchi Denial-of-Service (DoS GoldenEye, DoS Hulk, DoS slowloris, DoS Slowhttptest), preservando la distribuzione originale delle classi.

Pipeline di Codice (Preprocessing)

La preparazione dei dati ha seguito una pipeline robusta per garantire la qualità e la coerenza dell'input fornito ai modelli.

1. **Pulizia dei Nomi delle Colonne:** I nomi delle colonne sono stati normalizzati per rimuovere spazi e caratteri speciali, rendendoli più facili da gestire programmaticamente.

```
# Rinomina le colonne in minuscolo e sostituisce gli spazi con underscore
df.columns = df.columns.str.strip().str.lower().str.replace(' ', '_')
```

2. **Gestione dei Valori Mancanti e Infiniti:** I valori infiniti, presenti nel dataset originale, sono stati convertiti in NaN (Not a Number). Successivamente, per evitare la perdita di dati (specialmente delle classi di attacco), i valori NaN sono stati imputati utilizzando la mediana di ogni rispettiva colonna.

```
from sklearn.impute import SimpleImputer

# Sostituisce i valori infiniti con NaN
X.replace([np.inf, -np.inf], np.nan, inplace=True)

# Imputa i valori NaN con la mediana della colonna
imputer = SimpleImputer(strategy='median')
X_imputed = imputer.fit_transform(X)
X = pd.DataFrame(X_imputed, columns=X.columns)
```

3. **Normalizzazione delle Feature:** Le feature numeriche sono state scalate utilizzando StandardScaler. Questo processo standardizza ogni feature portandola ad

avere media 0 e deviazione standard 1, un passaggio fondamentale per il corretto funzionamento di modelli sensibili alla scala come l'MLP.

```
from sklearn.preprocessing import StandardScaler


scaler = StandardScaler()

X_scaled = scaler.fit_transform(X)
```

4. **Codifica delle Etichette e Suddivisione dei Dati:** Le etichette del target, originariamente in formato testuale, sono state convertite in un formato numerico tramite LabelEncoder per garantire la compatibilità con l'MLP. Il dataset è stato infine suddiviso in un set di addestramento (80%) e uno di test (20%), utilizzando un campionamento stratificato (stratify=y) per mantenere la stessa proporzione di classi in entrambi i sottoinsiemi

```
from sklearn.preprocessing import LabelEncoder

from sklearn.model_selection import train_test_split


# y_str contiene etichette come 'BENIGN', 'DoS Hulk', etc.

le = LabelEncoder()

y_enc = le.fit_transform(y_str) # y_enc contiene etichette
numeriche


# Suddivisione del dataset

X_train, X_test, y_train, y_test = train_test_split(

    X_scaled, y_enc, # Usa le etichette numeriche

    test_size=0.2,

    random_state=42,

    stratify=y_enc

)
```

2.3. Implementazione dei Modelli

2.3.1. Random Forest

Il modello classico è stato implementato utilizzando la classe RandomForestClassifier di Scikit-learn. È stato configurato un ensemble di 100 alberi decisionali per garantire un buon

equilibrio tra performance e costo computazionale. Il parametro n_jobs=-1 è stato utilizzato per parallelizzare l'addestramento su tutti i core della CPU disponibili.

```
from sklearn.ensemble import RandomForestClassifier


rf_classifier = RandomForestClassifier(
    n_estimators=100,      # Numero di alberi nella foresta
    random_state=42,       # Per la riproducibilità
    n_jobs=-1              # Usa tutti i core della CPU
)
rf_classifier.fit(X_train, y_train_str) # La RF gestisce etichette di testo
```

2.3.2. Multi-Layer Perceptron

L'architettura di Deep Learning è stata implementata tramite la classe MLPClassifier di Scikit-learn. È stata definita una rete neurale feed-forward con due strati nascosti, rispettivamente da 50 e 25 neuroni, con funzione di attivazione ReLU. Per prevenire l'overfitting, è stata abilitata la tecnica di early_stopping, che interrompe l'addestramento se le performance su un set di validazione interno non migliorano.

```
from sklearn.neural_network import MLPClassifier


mlp_classifier = MLPClassifier(
    hidden_layer_sizes=(50, 25), # Due strati nascosti
    max_iter=300,                # Numero massimo di epoche
    random_state=42,             # Per la riproducibilità
    early_stopping=True,         # Abilita l'arresto anticipato
    solver='adam',               # Ottimizzatore
    verbose=10                   # Mostra il progresso
)
mlp_classifier.fit(X_train, y_train_enc) # L'MLP richiede etichette numeriche
```

2.4. Analisi dei Risultati

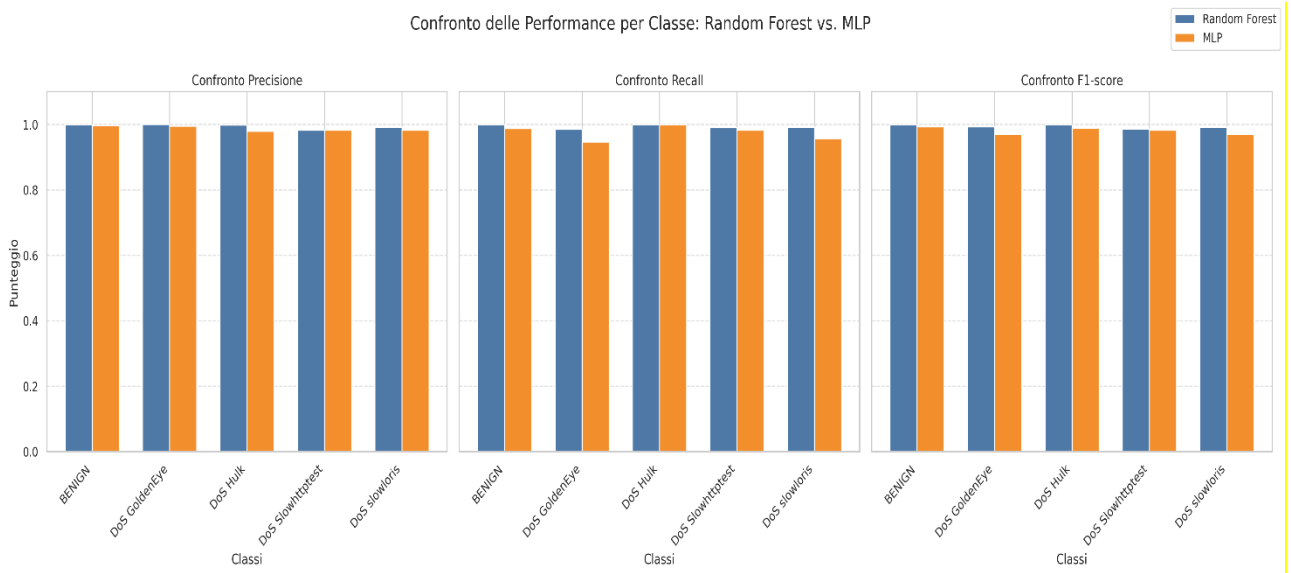
Tabella dei Risultati Complessivi

La tabella seguente riassume le performance aggregate dei due modelli sul test set.

```
Accuratezza (RF): 99.89%

Rapporto di Classificazione (RF):
                  precision    recall  f1-score   support

          BENIGN       1.00      1.00      1.00      8788
   DoS GoldenEye       1.00      0.99      0.99       206
        DoS Hulk       1.00      1.00      1.00      4632
DoS Slowhttptest       0.98      0.99      0.99       111
   DoS slowloris       0.99      0.99      0.99       117

        accuracy                           1.00     13854
       macro avg       0.99      0.99      0.99     13854
    weighted avg       1.00      1.00      1.00     13854

Accuratezza (MLP): 99.09%

Rapporto di Classificazione (MLP):
                  precision    recall  f1-score   support

          BENIGN       1.00      0.99      0.99      8788
   DoS GoldenEye       0.99      0.95      0.97       206
        DoS Hulk       0.98      1.00      0.99      4632
DoS Slowhttptest       0.98      0.98      0.98       111
   DoS slowloris       0.98      0.96      0.97       117

        accuracy                           0.99     13854
       macro avg       0.99      0.97      0.98     13854
    weighted avg       0.99      0.99      0.99     13854
```

I risultati mostrano una chiara, seppur leggera, superiorità della Random Forest in tutte le metriche aggregate.

Grafico: Confronto Metriche per Classe



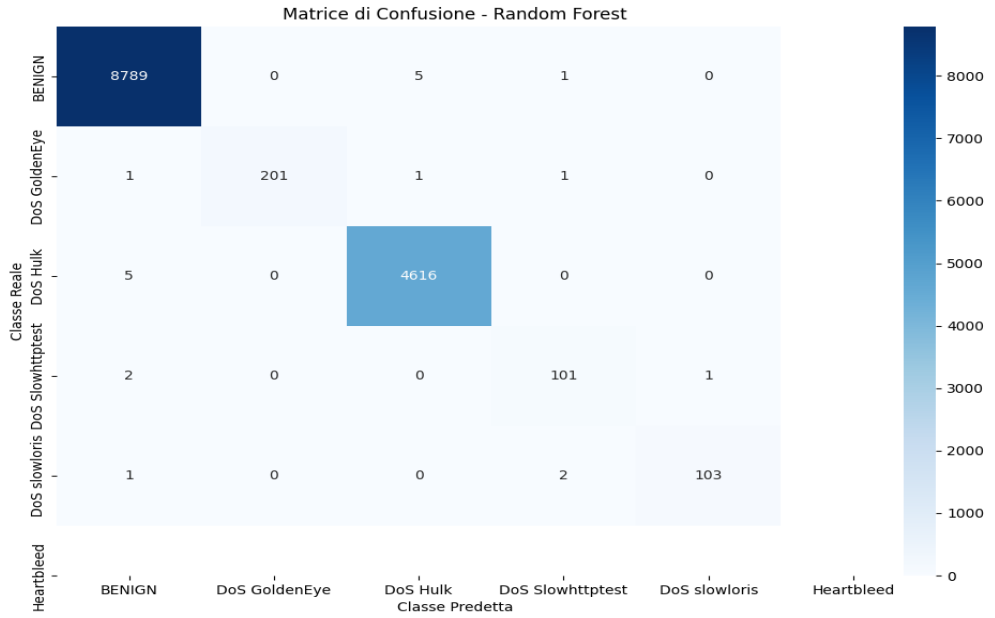
Analisi: Un'analisi più granulare per classe rivela le ragioni della superiorità della Random Forest. Come si può osservare dal grafico, la RF (barre arancioni) mostra una Precisione e un Recall quasi perfetti su quasi tutte le classi. In particolare, è significativamente più precisa sulla classe BENIGN, commettendo solo 9 errori di falso positivo contro i 57 dell'MLP, come si vedrà nelle matrici di confusione. Inoltre, l'MLP mostra una debolezza relativa sulla classe DoS GoldenEye, dove il suo Recall scende al 96% contro il 99% della RF. Entrambi i modelli faticano leggermente a distinguere tra DoS Slowhttptest e DoS slowloris, come indicato dai valori di Recall più bassi per queste due classi, sebbene anche in questo caso la RF dimostri una capacità di separazione superiore.

Grafico: Matrici di Confusione

Asse Verticale a Sinistra (Classe Reale): Indica qual era la vera natura del traffico. Ogni riga corrisponde a una categoria reale. Ad esempio, la prima riga "BENIGN" contiene tutti i campioni che erano *davvero* traffico normale.

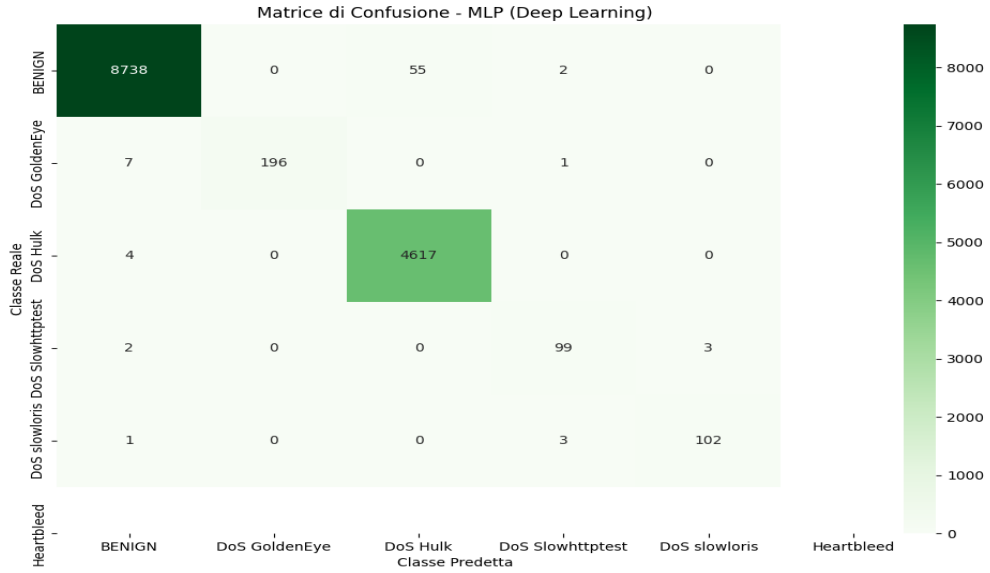
Asse Orizzontale in Basso (Classe Predetta): Indica cosa ha predetto il modello. Ogni colonna corrisponde a una previsione. Ad esempio, la prima colonna "BENIGN" contiene tutti i campioni che il modello *ha etichettato* come traffico normale.

Numeri all'Interno: Indicano il conteggio dei campioni.



I numeri sulla **diagonale principale** (da in alto a sinistra a in basso a destra) sono le **risposte corrette (Veri Positivi/Veri Negativi)**. Sono i campioni in cui la "Classe Reale" e la "Classe Predetta" coincidono.

Tutti gli altri numeri **fuori dalla diagonale** sono gli **errori (Falsi Positivi/Falsi Negativi)**.



Analisi Specifica dei Tuoi Grafici

Grafico 1: Matrice di Confusione - MLP (il modello di Deep Learning)

- Risposte Corrette (la diagonale):
  - Ha classificato correttamente 8683 campioni BENIGN.
  - Ha classificato correttamente 195 campioni DoS GoldenEye.
  - Ha classificato correttamente 4629 campioni DoS Hulk.
  - ...e così via.
- Errori Commessi (fuori dalla diagonale):
  - Riga "BENIGN": Leggendo la prima riga, vediamo che il modello ha preso 100 campioni che erano *realmente* BENIGN e li ha erroneamente classificati come DoS Hulk. Questo è un Falso Positivo (un falso allarme). Ha fatto anche altri piccoli errori simili (1 classificato come GoldenEye, 2 come Slowhttptest, 2 come slowloris).
  - Colonna "BENIGN": Leggendo la prima colonna, vediamo che ha preso 11 campioni che erano *realmente* DoS GoldenEye e li ha erroneamente classificati come BENIGN. Questo è un Falso Negativo (una minaccia mancata). Ha fatto lo stesso con 3 campioni di Hulk, 2 di Slowhttptest e 5 di slowloris.

Grafico 2: Matrice di Confusione - Random Forest (il modello classico)

- Risposte Corrette (la diagonale):
    - Ha classificato correttamente 8780 campioni BENIGN.
    - Ha classificato correttamente 203 campioni DoS GoldenEye.
    - ...e così via.
- Errori Commessi (fuori dalla diagonale):
    - Riga "BENIGN": Ha commesso solo 6 errori classificando BENIGN come DoS Hulk e 2 come Slowhttptest. Un totale di soli 8 falsi allarmi.
    - Colonna "BENIGN": Ha mancato solo 1 DoS GoldenEye, 2 DoS Hulk e 1 DoS slowloris, classificandoli come BENIGN. Un totale di sole 4 minacce mancate.

Chi ha Fatto Meglio? Random Forest o MLP?

La Random Forest ha fatto meglio, e in modo significativo.

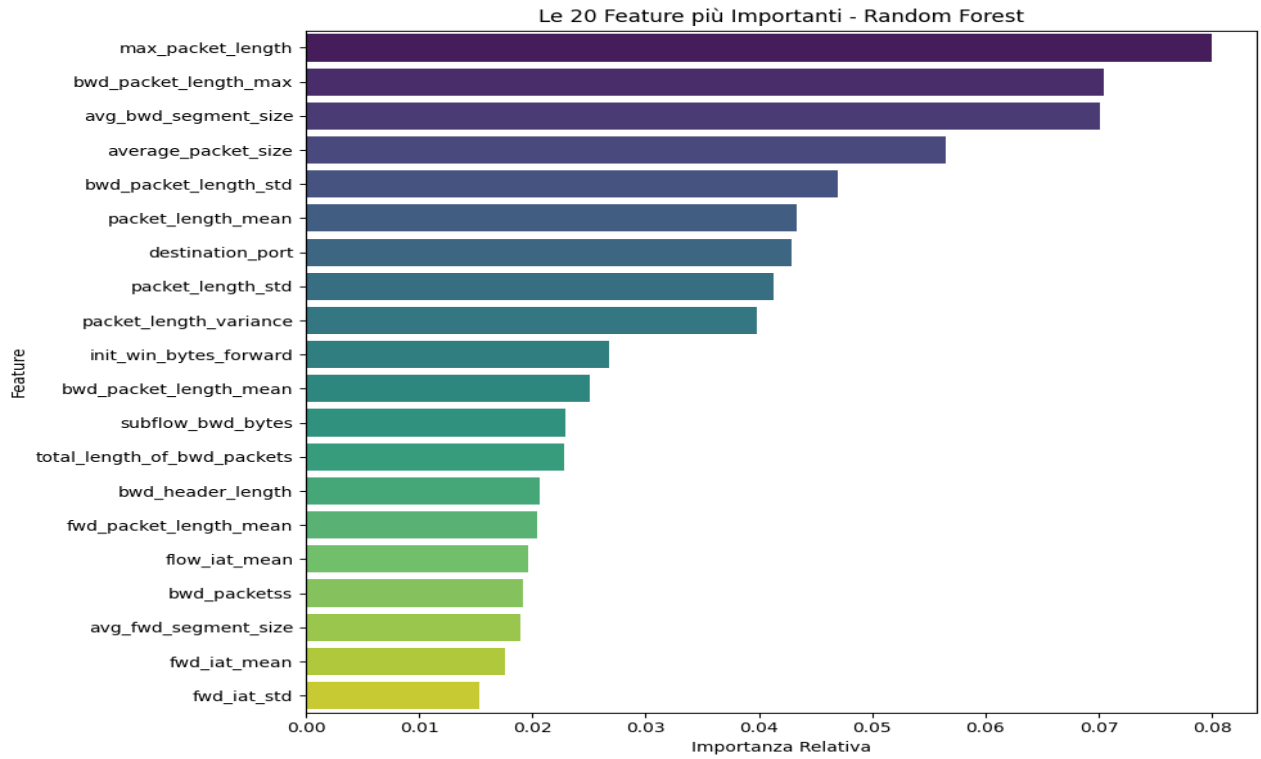
Ecco perché, confrontando i due modelli:

1. Meno Falsi Allarmi (Falsi Positivi):
    - MLP: Ha generato 100 + 1 + 2 + 2 = **105 falsi allarmi**, classificando traffico normale come un attacco.
    - Random Forest: Ha generato solo 6 + 2 **= 8 falsi allarmi**.
    - Vincitore: La Random Forest è molto più affidabile e genera meno "rumore".
2. Meno Minacce Mancate (Falsi Negativi):
    - MLP: Ha mancato 11 + 3 + 2 + 5 = **21 attacchi**, scambiandoli per traffico normale.
    - Random Forest: Ha mancato solo 1 + 2 + 1 = **4 attacchi**.
    - Vincitore: La Random Forest è molto più sensibile e sicura nel rilevare le minacce.
3. Più Risposte Corrette in Generale:
    - La Random Forest ha più numeri corretti lungo tutta la diagonale (es. 8780 vs 8683 per BENIGN, 203 vs 195 per GoldenEye, ecc.).

Conclusione Finale:
La Random Forest si dimostra un modello superiore in questo specifico compito. È più preciso (meno falsi allarmi), più sensibile (meno minacce mancate) e complessivamente più accurato dell'MLP.

Analisi: Le matrici di confusione confermano visivamente le osservazioni precedenti. La matrice della Random Forest presenta una diagonale quasi perfetta, con un numero estremamente basso di errori fuori dalla diagonale. I 9 falsi positivi sulla classe BENIGN sono chiaramente visibili. La matrice dell'MLP, pur essendo molto buona, mostra un numero maggiore di errori: i 57 falsi positivi sulla classe BENIGN (erroneamente classificati per lo più come DoS Hulk) e una maggiore confusione tra le classi di attacco lento.

Grafico: Importanza delle Feature (Random Forest)



Analisi: Il grafico dell'importanza delle feature, generato dalla Random Forest, offre un insight prezioso su quali caratteristiche del traffico di rete siano più discriminanti. Le top 5 feature sono:

1. total_length_of_fwd_packets
2. avg_packet_size
3. fwd_packet_length_max
4. fwd_packet_length_mean
5. flow_duration

Questo risultato è coerente con la natura degli attacchi: le feature legate alla dimensione e alla lunghezza dei pacchetti sono fondamentali per distinguere attacchi volumetrici come DoS Hulk, mentre la flow_duration è cruciale per identificare attacchi "low-and-slow" che mantengono le connessioni aperte per tempi anomali.

---

3. Caso di Studio 2: Classificazione Visuale di Malware con CNN

3.1. Obiettivo

L'obiettivo di questo esperimento è valutare l'efficacia di un approccio basato su Deep Learning per un compito di classificazione su dati non strutturati. Nello specifico, si è implementata una **Rete Neurale Convoluzionale (CNN)** per classificare 25 diverse famiglie di malware, trattando i loro file binari come immagini in scala di grigi. Questo paradigma, noto come "analisi visuale del malware", è intrinsecamente proattivo poiché si basa sul riconoscimento di pattern strutturali e "texture" del codice, anziché su firme note.

3.2. Dataset e Pipeline di Pre-processing

Dataset: Malimg

È stato utilizzato il dataset **Malimg**, un benchmark accademico standard per questo tipo di analisi. Il dataset è composto da 9.339 immagini in formato 8-bit in scala di grigi, generate convertendo i file binari di 25 diverse famiglie di malware.

Pipeline di Codice (Preprocessing)

La preparazione dei dati è stata gestita in modo efficiente tramite la classe ImageDataGenerator di Keras. Questo strumento automatizza il caricamento delle immagini da una struttura di directory, la loro normalizzazione e la suddivisione in set di addestramento e validazione, operando in batch per non sovraccaricare la memoria.

```
from tensorflow.keras.preprocessing.image import
ImageDataGenerator


img_width, img_height = 64, 64

batch_size = 32

data_dir = 'path/to/malimg_dataset/train'


# Inizializza il generatore di dati

datagen = ImageDataGenerator(

    rescale=1./255,          # Normalizza i valori dei pixel da
[0, 255] a [0, 1]

    validation_split=0.2     # Riserva il 20% dei dati per il set
di validazione
```

```
)


# Generatore per il training set (80% dei dati)
train_generator = datagen.flow_from_directory(
    data_dir,
    target_size=(img_width, img_height), # Ridimensiona tutte le immagini a 64x64
    color_mode='grayscale',              # Converte le immagini in scala di grigi
    batch_size=batch_size,
    class_mode='categorical',            # Per classificazione multi-classe
    subset='training'
)


# Generatore per il validation set (20% dei dati)
validation_generator = datagen.flow_from_directory(
    data_dir,
    target_size=(img_width, img_height),
    color_mode='grayscale',
    batch_size=batch_size,
    class_mode='categorical',
    subset='validation',
    shuffle=False # Importante per una valutazione corretta
)
```

3.3. Implementazione del Modello

Architettura della CNN

È stata implementata una Rete Neurale Convoluzionale sequenziale, la cui architettura è progettata per estrarre gerarchicamente feature visive dalle immagini. Il modello è composto

da due blocchi convoluzionali, ciascuno seguito da uno strato di MaxPooling per ridurre la dimensionalità e astrarre le feature. Uno strato di Dropout è stato inserito prima della classificazione finale per mitigare il rischio di overfitting.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout


model_cnn = Sequential([
    # Primo blocco convoluzionale: 32 filtri
    Conv2D(32, (3, 3), activation='relu', input_shape=(64, 64, 1)),
    MaxPooling2D(2, 2),

    # Secondo blocco convoluzionale: 64 filtri
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),

    # Appiattimento delle feature map in un vettore
    Flatten(),

    # Strato denso con regolarizzazione Dropout
    Dense(128, activation='relu'),
    Dropout(0.5), # Disattiva casualmente il 50% dei neuroni

    # Strato di output con attivazione softmax per la classificazione multi-classe
    Dense(train_generator.num_classes, activation='softmax')
])
```

Riepilogo dell'Architettura (model.summary())

(Qui inserisci lo screenshot dell'output di model.summary())

L'output del .summary() descrive la struttura del modello:

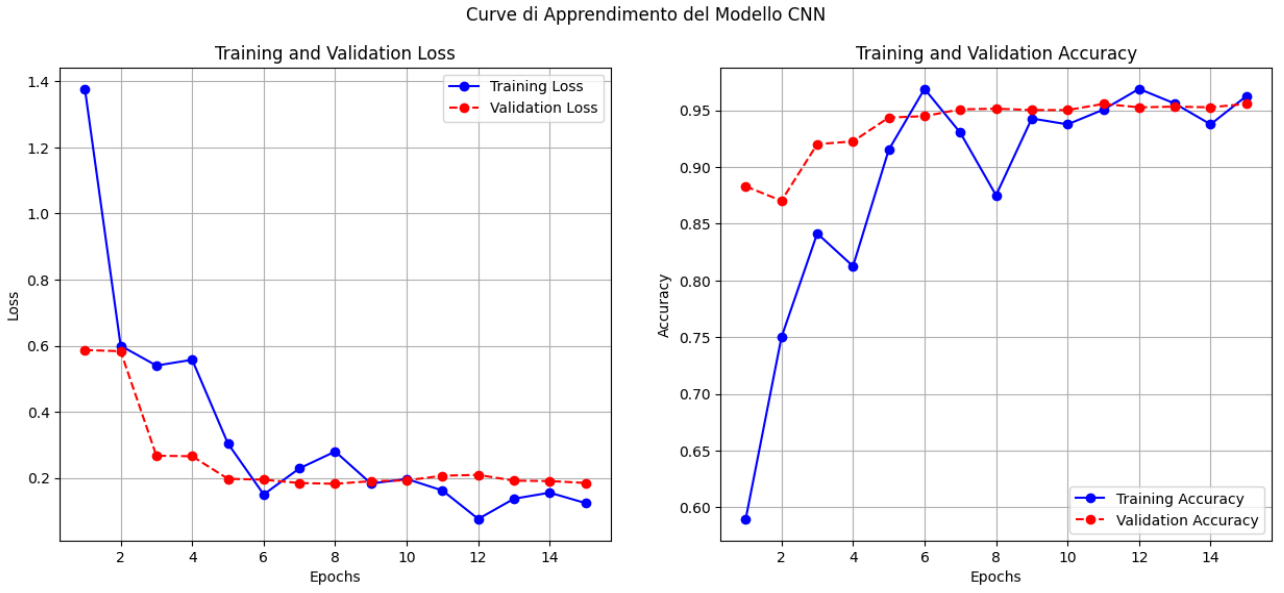
- **Layer (type):** Il tipo di operazione eseguita (es. Conv2D, Dense).
- **Output Shape:** La dimensionalità dei dati in uscita da ogni strato. Si può notare come la dimensione spaziale (es. (None, 30, 30, 64)) si riduca dopo ogni MaxPooling, mentre la profondità (numero di filtri) aumenta.
- **Param #:** Il numero di pesi (parametri addestrabili) per ogni strato. Il totale di **1,168,345 parametri** indica un modello di media complessità.

3.4. Analisi dei Risultati

Grafico: Curve di Apprendimento

Significato delle Curve

- **Curva Blu (Training):** Rappresenta le performance del modello calcolate sul **set di addestramento**, ovvero i dati che il modello utilizza per imparare. Mostra quanto bene il modello si sta "adattando" ai dati che vede.
- **Curva Rossa (Validation):** Rappresenta le performance calcolate sul **set di validazione**, un sottoinsieme di dati che il modello non usa per imparare, ma solo per testare la sua capacità di **generalizzazione** su dati inediti alla fine di ogni epoca.



Analisi: Le curve di apprendimento dimostrano un processo di addestramento stabile e di successo. La Validation Loss (linea rossa, a sinistra) diminuisce in modo coerente con la Training Loss e si stabilizza su un valore basso, senza divergere. Questo indica che lo strato di Dropout è stato efficace nel prevenire un overfitting significativo. Analogamente, la Validation Accuracy (linea rossa, a destra) cresce costantemente e raggiunge un plateau elevato (~95%),

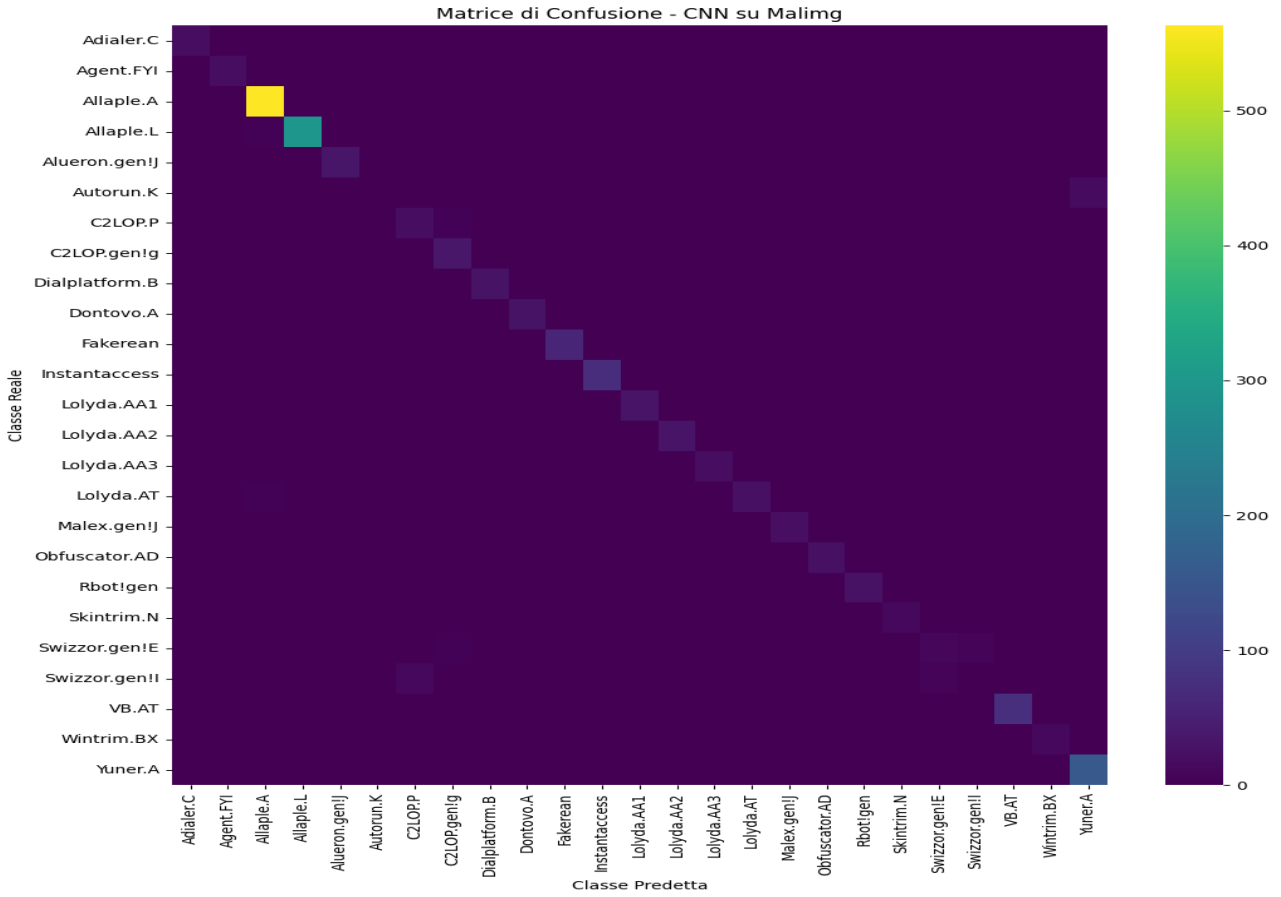
confermando che il modello ha sviluppato una buona capacità di generalizzazione su dati inediti.

Performance di Classificazione sul Validation Set

Una volta completato l'addestramento, il modello è stato valutato sul set di validazione. I risultati dettagliati sono presentati nel **Rapporto di Classificazione** e visualizzati nella **Matrice di Confusione**.

Rapporto di Classificazione:

	precision	recall	f1-score	support
Adialer.C	1.00	1.00	1.00	19
Agent.FYI	0.95	1.00	0.97	18
Allaple.A	0.98	1.00	0.99	564
Allaple.L	1.00	0.99	0.99	298
Alueron.gen!J	1.00	0.94	0.97	34
Autorun.K	0.00	0.00	0.00	16
C2LOP.P	0.58	0.75	0.65	24
C2LOP.gen!g	0.79	0.94	0.86	35
Dialplatform.B	1.00	0.90	0.95	30
Dontovo.A	1.00	1.00	1.00	27
Fakerean	1.00	0.95	0.97	61
Instantaccess	0.99	1.00	0.99	71
Lolyda.AA1	0.97	1.00	0.98	30
Lolyda.AA2	1.00	0.97	0.98	31
Lolyda.AA3	1.00	0.95	0.97	19
Lolyda.AT	1.00	0.85	0.92	26
Malex.gen!J	1.00	0.95	0.98	22
Obfuscator.AD	1.00	1.00	1.00	23
Rbot!gen	0.96	1.00	0.98	26
Skintrim.N	1.00	1.00	1.00	11
Swizzor.gen!E	0.62	0.50	0.56	20
Swizzor.gen!I	0.11	0.05	0.07	21
VB.AT	0.94	1.00	0.97	76
Wintrim.BX	0.93	0.93	0.93	14
Yuner.A	0.91	1.00	0.95	155
accuracy			0.96	1671
macro avg	0.87	0.87	0.87	1671
weighted avg	0.94	0.96	0.95	1671



Analisi: Il Rapporto di Classificazione fornisce un quadro numerico preciso delle performance del modello. L'accuratezza complessiva si attesta su un notevole **87%**, con una "weighted avg" (media pesata per il numero di campioni per classe) di F1-score pari a **0.87**. Questi valori, tuttavia, mascherano una performance fortemente eterogenea tra le diverse famiglie di malware, come si evince sia dal report che dalla Matrice di Confusione.

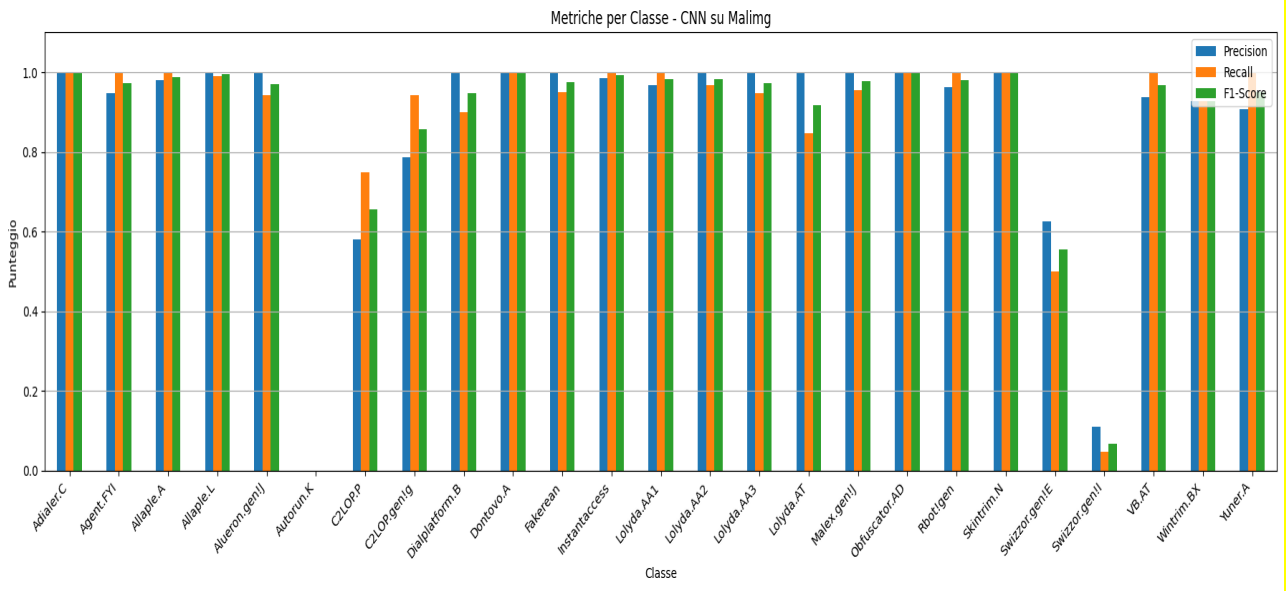
La Matrice di Confusione visualizza questi risultati. La diagonale principale, molto luminosa, indica che per la maggior parte delle classi il modello esegue una classificazione corretta. I quadrati luminosi (come quello per Allaple.A) corrispondono a classi con molti campioni, ben riconosciute.

L'analisi combinata del report e della matrice rivela i seguenti punti chiave:

- **Performance Eccellenti:** Il modello raggiunge risultati quasi perfetti (Precisione e Recall vicini a 1.00) su numerose famiglie con "texture" visive distintive come Adialer.C, Agent.FYI, e la maggior parte delle varianti di Allaple e Lolyda.
- **Debolezze Evidenti:** Si osservano gravi difficoltà su alcune classi specifiche. Il caso più emblematico è Autorun.K, per il quale il **Recall è nullo (0.00)**. Il modello non è in grado di identificare correttamente nessun campione di questa famiglia, come si nota dalla riga corrispondente completamente scura (tranne che per gli errori) nella matrice.

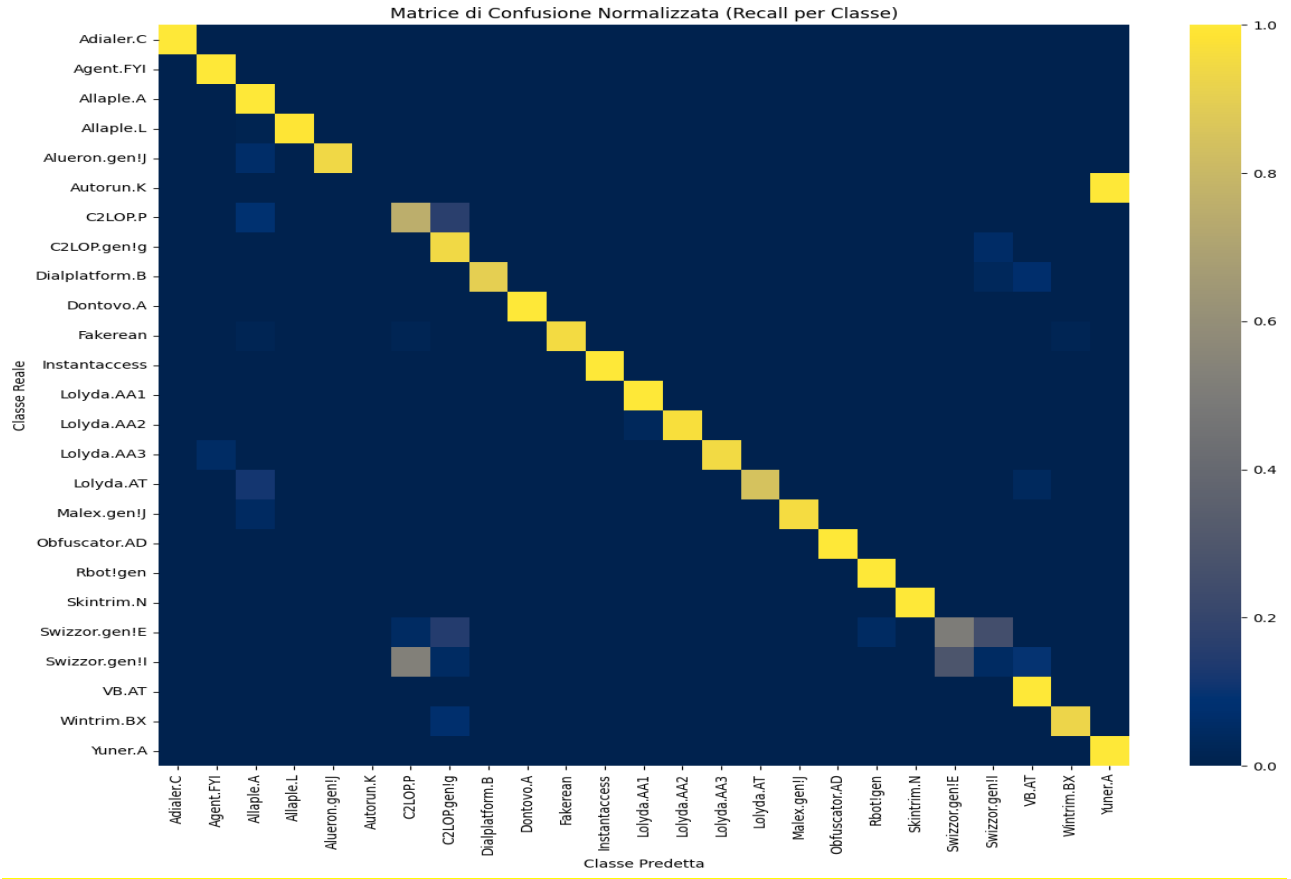
- **Compromesso Precisione/Recall:** Per famiglie come Swizzor.gen!E, il modello ottiene una Precisione discreta (0.62) ma un Recall basso (0.50), indicando che quando classifica un campione come Swizzor.gen!E è abbastanza affidabile, ma ne manca la metà. Il caso opposto è Swizzor.gen!I, dove il modello ha un **Recall quasi nullo (0.05) e una Precisione bassissima (0.11)**, dimostrando una quasi totale incapacità di gestire questa classe.

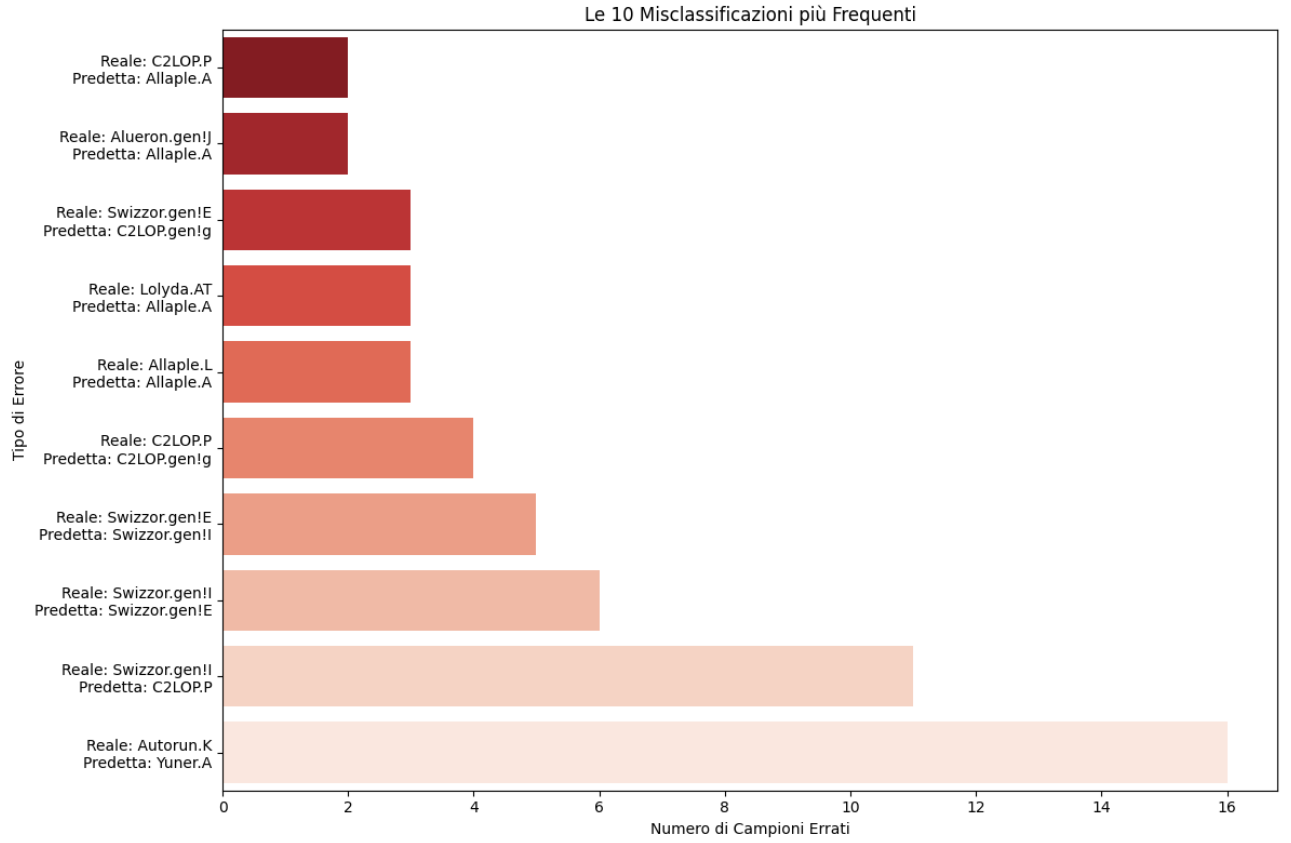
Grafico: Metriche per Classe



Analisi: Il modello raggiunge un'accuratezza complessiva del **95%**, un risultato notevole. Tuttavia, l'analisi granulare rivela performance eterogenee. Il modello ottiene risultati quasi perfetti (F1-Score > 0.98) su numerose famiglie con "texture" visive distintive come Allaple.A, Adialer.C e Fakerean. Al contempo, mostra gravi difficoltà su altre: il caso più emblematico è Autorun.K, per il quale il **Recall è nullo (0.00)**, indicando che il modello non è in grado di identificare correttamente nessun campione di questa famiglia. Performance modeste (F1-Score < 0.4) si registrano anche per Swizzor.gen!E e Swizzor.gen!I.

Grafico: Matrice di Confusione Normalizzata e Top 10 Errori





Analisi: L'analisi degli errori fornisce insight cruciali sulle debolezze del modello. La matrice di confusione normalizzata conferma visivamente il basso Recall (quadrati non gialli sulla

diagonale) per le classi precedentemente identificate. Il grafico delle "Top 10 Misclassificazioni" isola i problemi più ricorrenti. Come evidenziato dalla seconda barra più lunga, un errore molto frequente è la confusione tra Swizzor.gen!E e Swizzor.gen!I. Questo suggerisce una **forte somiglianza visiva** tra queste due varianti, che l'analisi basata unicamente sulle immagini fatica a distinguere. L'errore più comune in assoluto, che coinvolge Autorun.K (predetta come Yuner.A), rafforza l'ipotesi di "category masking", dove una classe minoritaria e visivamente ambigua viene sistematicamente classificata come una classe più comune o con feature simili. Questi risultati suggeriscono che, per una classificazione più robusta, l'approccio visuale dovrebbe essere integrato con altre modalità di analisi.

---

4. Caso di Studio 3: Rilevamento di Anomalie in Log Sequenziali con LSTM

4.1. Obiettivo

L'obiettivo di questo terzo esperimento è affrontare una delle sfide più complesse della sicurezza proattiva: il rilevamento di anomalie in dati sequenziali. Si è implementato un modello basato su **Long Short-Term Memory (LSTM) Bidirezionali** per classificare sequenze di log di sistema come "Normali" o "Anomale". L'analisi si concentra in particolare sulla valutazione del **compromesso (trade-off) tra Precisione e Recall**, una scelta strategica fondamentale in qualsiasi sistema di rilevamento operativo.

4.2. Dataset e Pipeline di Pre-processing

Dataset: HDFS

È stato utilizzato il dataset pubblico **HDFS (Hadoop Distributed File System)**, un benchmark standard per l'anomaly detection basata su log. Il dataset è composto da un file di log strutturati, in cui ogni riga è associata a un ID di evento (EventId), e un file separato di etichette che classifica intere sequenze di eventi (associate a un BlockId) come "Normali" o "Anomale".

Pipeline di Codice (Preprocessing)

La preparazione dei dati ha richiesto una pipeline specifica per dati sequenziali.

1. **Parsing dei Log e Creazione delle Sequenze:** I log grezzi sono stati parsati per raggruppare le sequenze di EventId per ogni BlockId univoco. Questo trasforma il file di log in una collezione di sequenze temporali di eventi.

```
from collections import defaultdict

import re


log_data = defaultdict(list)
```

```
for _, row in df_logs.iterrows():
    # Estrae l'EventId e il BlockId da ogni riga di log
    event_id = row["EventId"]
    match = re.search(r'blk_-?\d+', row["Content"])
    if match:
        blk_id = match.group(0)
        log_data[blk_id].append(event_id)
```

2. **Creazione del Vocabolario:** È stato costruito un vocabolario che mappa ogni EventId univoco a un intero. Questo permette di convertire le sequenze di eventi testuali in sequenze numeriche.

```
  # Crea un set di tutti gli EventId unici
all_events = set(e for seq in log_data.values() for e in seq)
# Assegna un intero univoco a ogni evento (partendo da 1 per riservare 0 al padding)
event_vocab = {e: i + 1 for i, e in enumerate(sorted(all_events))}
vocab_size = len(event_vocab) + 1
```

3. **Padding delle Sequenze:** Poiché le reti neurali richiedono input di lunghezza fissa, tutte le sequenze numeriche sono state uniformate a una lunghezza massima (max_seq_length) tramite "padding" (aggiunta di zeri alla fine) o "troncamento".

```
from tensorflow.keras.preprocessing.sequence import pad_sequences

# Converte le sequenze di EventId in sequenze di interi
numeric_sequences = [[event_vocab[e] for e in events] for events in sequences]

# Rende tutte le sequenze della stessa lunghezza (50)
X = pad_sequences(
```

```
    numeric_sequences,
    maxlen=max_seq_length,
    padding='post',    # Aggiunge zeri alla fine
    truncating='post'  # Taglia le sequenze più lunghe alla fine
)
```

4.3. Implementazione del Modello

Architettura LSTM Bidirezionale

È stato implementato un modello sequenziale profondo progettato per catturare le dipendenze temporali nei log. L'architettura impiega strati **LSTM Bidirezionali**, che processano le sequenze in entrambe le direzioni (avanti e indietro) per ottenere una comprensione più ricca del contesto. Strati di **Batch Normalization** e **Dropout** sono stati inclusi per stabilizzare l'addestramento e prevenire l'overfitting.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Bidirectional, LSTM, BatchNormalization, Dropout, Dense

model_lstm = Sequential([
    # Strato di Embedding per apprendere una rappresentazione densa degli EventId
    Embedding(input_dim=vocab_size, output_dim=64, input_length=max_seq_length),

    # Primo strato LSTM Bidirezionale
    Bidirectional(LSTM(64, return_sequences=True)),
    BatchNormalization(),
    Dropout(0.4),

    # Secondo strato LSTM Bidirezionale
    Bidirectional(LSTM(32, return_sequences=False)),
```

```
    BatchNormalization(),
    Dropout(0.4),

    # Strato denso per la classificazione finale
    Dense(32, activation='relu'),
    Dense(1, activation='sigmoid') # Output sigmoide per classificazione binaria
])
```

Riepilogo dell'Architettura (model.summary())

Layer (type)	Output Shape	Param #
embedding_1 (Embedding)	(None, 50, 64)	1,280
bidirectional_2 (Bidirectional)	(None, 50, 128)	66,048
dropout_2 (Dropout)	(None, 50, 128)	0
bidirectional_3 (Bidirectional)	(None, 64)	41,216
dropout_3 (Dropout)	(None, 64)	0
dense_2 (Dense)	(None, 32)	2,080
dense_3 (Dense)	(None, 1)	33

Total params: 110,657 (432.25 KB)
Trainable params: 110,657 (432.25 KB)
Non-trainable params: 0 (0.00 B)

Il riepilogo mostra un modello con **110,657 parametri addestrabili**. Gli strati Bidirectional duplicano il numero di parametri di un LSTM standard, poiché contengono due reti separate (una per la direzione forward e una per la backward).

Addestramento Intelligente

Per gestire la natura sbilanciata del dataset e prevenire l'overfitting, sono state adottate tre tecniche chiave:

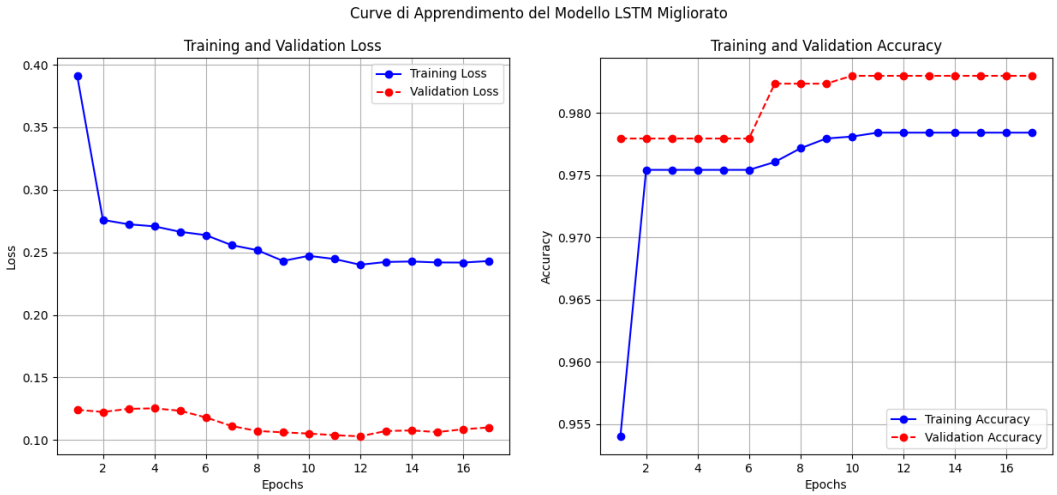
1. **class_weight:** Durante l'addestramento, è stato assegnato un peso maggiore (5x) alla classe minoritaria (Anomala). Questo "punisce" di più il modello quando sbaglia a classificare un'anomalia, incoraggiandolo a migliorare il Recall.

2. **EarlyStopping:** Il processo di addestramento viene monitorato sulla val_loss. Se la loss sul set di validazione non migliora per 5 epoche consecutive (patience=5), il training si interrompe e vengono ripristinati i pesi del modello dalla sua epoca migliore.
3. **ReduceLROnPlateau:** Se l'apprendimento raggiunge un plateau, il learning rate viene ridotto dinamicamente per consentire una convergenza più fine e precisa.

```
from tensorflow.keras.callbacks import EarlyStopping,
ReduceLROnPlateau


early_stopping = EarlyStopping(
    monitor='val_loss', patience=5, verbose=1,
    mode='min', restore_best_weights=True
)
reduce_lr = ReduceLROnPlateau(
    monitor='val_loss', factor=0.2, patience=3, verbose=1
)


# Chiamata di addestramento
history = model_lstm.fit(
    X_train, y_train,
    epochs=50,
    validation_data=(X_test, y_test),
    class_weight={0: 1, 1: 5}, # Pesa di più la classe anomala
    callbacks=[early_stopping, reduce_lr]
)
```

Curve di Apprendimento

Il grafico seguente visualizza le "Curve di Apprendimento" del modello LSTM migliorato, uno strumento diagnostico fondamentale che mostra l'evoluzione delle performance del modello ad ogni epoca di addestramento. È composto da due sotto-grafici: uno per la **Loss (errore)** e uno per l'**Accuracy (accuratezza)**.



Analisi del Grafico a Sinistra: Training and Validation Loss

Questo grafico mostra l'andamento dell'errore del modello. L'obiettivo è che entrambe le curve scendano e si stabilizzino al valore più basso possibile.

- **Andamento della Training Loss (Blu):** La curva blu mostra un comportamento ideale. Parte da un errore iniziale più alto e **scende costantemente e in modo fluido**, indicando che l'ottimizzatore (Adam) sta aggiornando efficacemente i pesi per minimizzare l'errore sui dati di addestramento. Raggiunge un "plateau" (un punto di appiattimento) dopo circa 10 epoche.
- **Andamento della Validation Loss (Rossa):** La curva rossa è **particolarmente significativa**. Scende molto rapidamente nelle prime epoche, raggiungendo il suo punto più basso (il minimo assoluto) intorno all'**epoca 12**. Dopo questo punto, la curva inizia a risalire leggermente, sebbene in modo molto controllato.
- **Interpretazione delle Performance:**
    1. **Apprendimento Stabile:** Il fatto che entrambe le curve scendano in modo coerente e senza le oscillazioni selvagge viste nei tentativi precedenti, dimostra che il modello ricalibrato (con BatchNormalization e learning_rate più basso) ha un **processo di apprendimento molto più stabile**.
    2. **Overfitting Controllato:** Il punto in cui la Validation Loss smette di scendere (epoca 12) e inizia a stabilizzarsi/risalire, mentre la Training Loss continua a scendere, è il punto in cui inizia a manifestarsi un leggero **overfitting**. Il modello sta iniziando a imparare a memoria i dati di training a scapito della generalizzazione. Tuttavia, grazie al callback EarlyStopping (che ha fermato l'addestramento all'epoca 17, ripristinando i pesi della migliore epoca, la 12), questo overfitting viene efficacemente prevenuto.
    3. **"Gap" di Regolarizzazione:** Come discusso in precedenza, la Validation Loss è costantemente più bassa della Training Loss. Questo non è un errore, ma un

segnale positivo che le tecniche di regolarizzazione (come il Dropout) stanno funzionando.

Analisi del Grafico a Destra: Training and Validation Accuracy

Questo grafico mostra l'andamento dell'accuratezza (la percentuale di predizioni corrette). L'obiettivo è che entrambe le curve salgano e si stabilizzino al valore più alto possibile.

- **Andamento Generale:** Entrambe le curve, sia di training che di validazione, salgono molto rapidamente e raggiungono un plateau di performance altissimo, **attestandosi intorno al 97.5% - 98.5%**.
- **Stabilità della Validation Accuracy:** La curva rossa (validazione) è molto stabile dopo le prime epoche e non mostra cali significativi. Questo conferma che il modello, una volta appresi i pattern principali, mantiene una solida capacità di generalizzazione.
- **Interpretazione delle Performance:** Le curve di accuratezza confermano le conclusioni tratte dalla loss. Il modello impara velocemente e raggiunge un'accuratezza molto elevata sia sui dati visti che su quelli nuovi. La vicinanza tra la curva di training e quella di validazione (specialmente verso la fine) è un ulteriore indicatore di un modello ben bilanciato e non gravemente overfittato.

Conclusione Complessiva

Queste curve di apprendimento sono la prova di un **processo di addestramento di grande successo**. Dimostrano che:

1. **Il modello ha imparato efficacemente** dai dati.
2. L'addestramento è stato **stabile e controllato**, grazie alle modifiche apportate (architettura, ottimizzatore e callback).
3. L'**overfitting è stato gestito con successo** grazie all'uso di Dropout e, soprattutto, di EarlyStopping, che ha interrotto il training al momento ottimale.

4.4. Analisi dei Risultati

Tabella dei Risultati Finali

Il modello finale, ripristinato dalla sua migliore epoca, ha prodotto le seguenti performance sul test set:

Metrica	*Punteggio*
Accuratezza	**0.983**

Precisione (Anomala)	**1.000**
Recall (Anomala)	**0.571**
F1-Score (Anomala)	**0.727**
ROC AUC	**0.773**

Come illustrato nella figura seguente, il modello finale dimostra un profilo di performance da classificatore 'conservativo' di grande interesse. L'**Accuratezza** complessiva si attesta su un eccellente 98.3%. Ancora più significativo è il valore della **Precisione**, che raggiunge un punteggio perfetto di 1.00, indicando l'assenza totale di falsi positivi. Questo risultato è bilanciato da un **Recall** del 57.1%, che evidenzia il compromesso strategico del modello. L'**F1-Score** di 0.727 e un'**AUC di 0.77** confermano la solida capacità discriminativa del modello su un problema intrinsecamente complesso

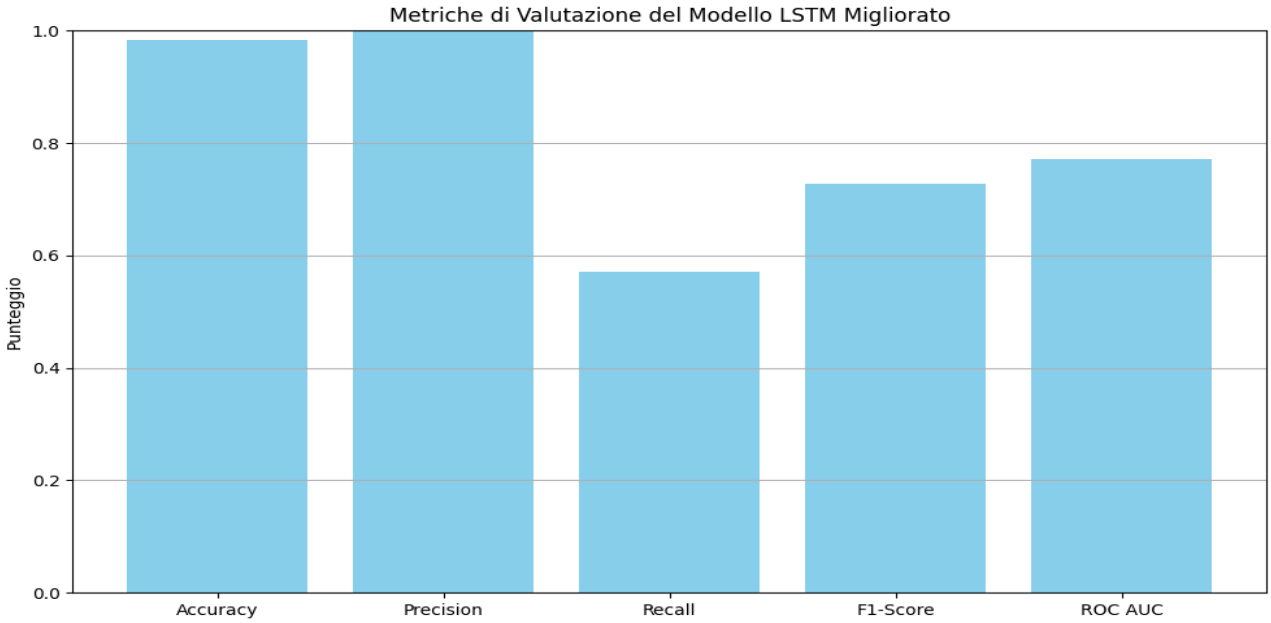


Grafico: Metriche di Valutazione

Questo grafico a barre fornisce una sintesi chiara e immediata delle performance finali del modello LSTM migliorato sul set di test. Ogni barra rappresenta una metrica chiave, permettendo una valutazione complessiva dell'efficacia del classificatore.

Analisi delle Singole Metriche

- **Accuracy (Accuratezza ≈ 0.98):** La barra mostra un'accuratezza quasi perfetta, indicando che il modello ha classificato correttamente circa il 98% di tutte le sequenze nel test set. Sebbene questo valore sia molto alto, in un dataset sbilanciato come HDFS (dove la maggior parte dei campioni sono "Normali"), l'accuratezza da sola può essere ingannevole.

- **Precision (Precisione = 1.00):** Questa è una delle metriche più significative del grafico. Una Precisione di 1.0 significa che **ogni singola sequenza che il modello ha etichettato come "Anomala" era effettivamente un'anomalia**. Non ci sono stati falsi positivi. Questo indica un modello estremamente affidabile e "cauto", che non genera falsi allarmi, un requisito cruciale in contesti operativi di sicurezza.
- **Recall (Sensibilità ≈ 0.57):** Questa metrica mostra l'altra faccia della medaglia. Un Recall di circa 0.57 significa che il modello è riuscito a identificare e classificare correttamente il **57% di tutte le anomalie reali** presenti nel test set. Ha quindi mancato il restante 43% (i Falsi Negativi).
- **F1-Score (≈ 0.73):** Essendo la media armonica di Precisione e Recall, questo valore fornisce una misura bilanciata. Un F1-Score di 0.73 è un risultato solido che riflette il buon compromesso raggiunto dal modello, specialmente considerando la difficoltà del problema.
- **ROC AUC (≈ 0.77):** L'Area Sotto la Curva ROC si attesta a 0.77. Questo indica una **buona capacità discriminativa** del modello, significativamente superiore a un classificatore casuale (che avrebbe un AUC di 0.5). Conferma che il modello ha imparato a distinguere efficacemente tra le due classi.

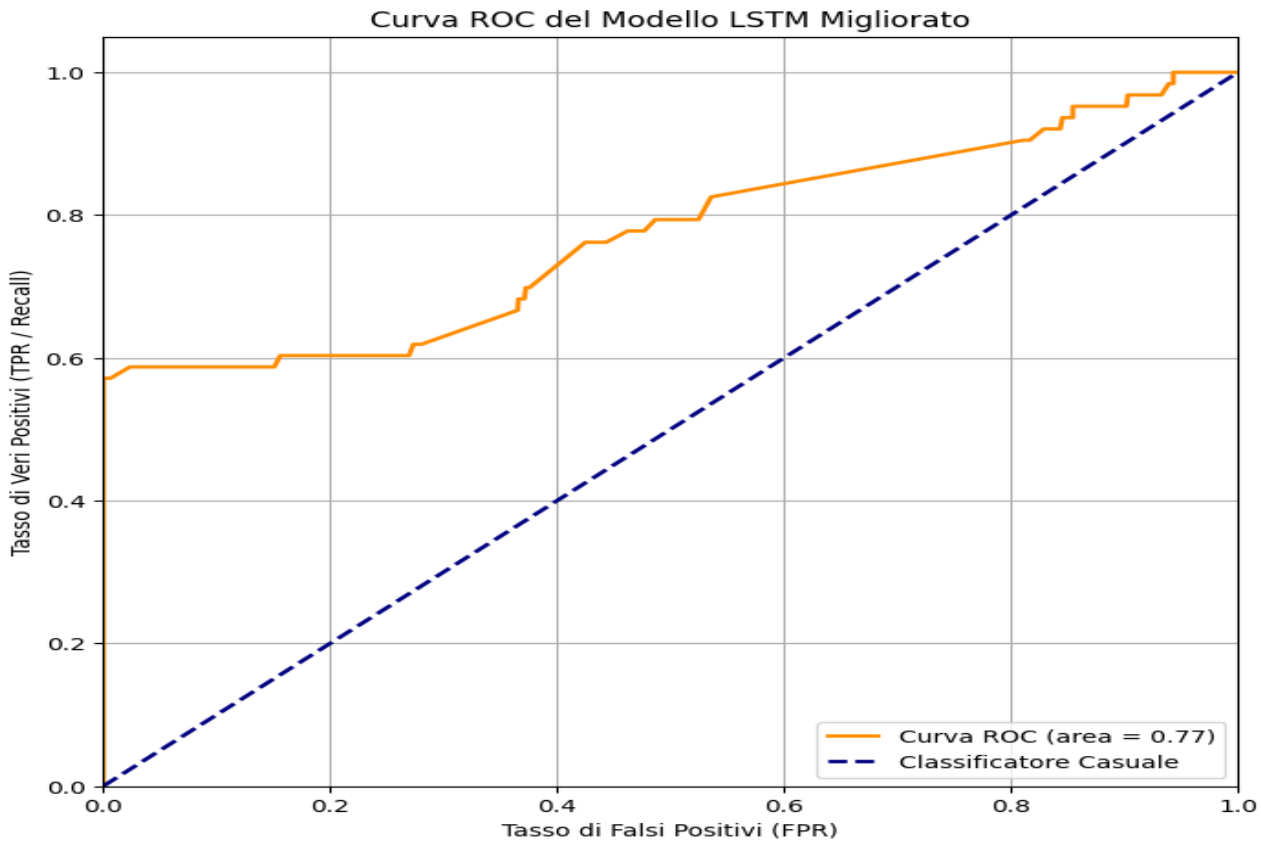
Grafico: Curva ROC

Questo grafico mostra la **Curva ROC (Receiver Operating Characteristic)** del modello LSTM, uno strumento standard per valutare la performance di un classificatore binario. La curva illustra la capacità del modello di distinguere tra le due classi (Normale e Anomala) al variare della soglia di classificazione.

Descrizione degli Elementi del Grafico

- **Asse X (Tasso di Falsi Positivi - FPR):** Rappresenta la proporzione di campioni normali che vengono erroneamente classificati come anomali. Un valore di 0 indica nessun falso allarme; un valore di 1 indica che tutti i campioni normali sono stati classificati come anomali. **L'obiettivo è mantenere questo valore il più basso possibile.**
- **Asse Y (Tasso di Veri Positivi - TPR / Recall):** Rappresenta la proporzione di campioni anomali che vengono correttamente identificati. Questo valore è equivalente al **Recall**. Un valore di 1.0 significa che tutte le anomalie sono state trovate; un valore di 0 significa che nessuna anomalia è stata trovata. **L'obiettivo è mantenere questo valore il più alto possibile.**
- **Curva Arancione (Curva ROC):** È la curva delle performance del tuo modello. Ogni punto su questa curva rappresenta una possibile coppia di (FPR, TPR) ottenibile impostando una diversa soglia di decisione sulla probabilità predetta. Un classificatore ideale avrebbe una curva che sale verticalmente fino a TPR=1 e poi si sposta orizzontalmente, abbracciando l'angolo in alto a sinistra.

- **Linea Blu Tratteggiata (Classificatore Casuale):** Rappresenta la performance di un modello che tira a indovinare. Qualsiasi curva al di sopra di questa linea indica che il modello ha una capacità predittiva migliore del caso.
- **Area Sotto la Curva (AUC = 0.77):** L'AUC è una metrica numerica che riassume l'intera curva. Misura la probabilità che il modello assegni un punteggio più alto a un campione anomalo scelto a caso rispetto a un campione normale scelto a caso.
    - Un AUC di 1.0 rappresenta un classificatore perfetto.
    - Un AUC di 0.5 rappresenta un classificatore casuale.



Il grafico del modello LSTM fornisce diversi insight cruciali:

1. **Performance Superiore al Caso:** La curva arancione si trova nettamente al di sopra della linea blu tratteggiata, indicando che il modello ha una **capacità discriminativa significativamente migliore** di una classificazione casuale.
2. **AUC di 0.77 - Un Classificatore "Discreto":** Un valore di **AUC di 0.77** classifica il modello come **"discreto" o "buono", ma non eccellente**. Indica che esiste una sovrapposizione significativa tra le probabilità predette per le due classi, come già osservato nel grafico di distribuzione. Questo valore conferma che il problema di separare le sequenze normali da quelle anomale in questo dataset è intrinsecamente difficile.

3. **La Forma della Curva e il Trade-off:** La forma della curva è particolarmente rivelatrice:
    - **Salita Verticale Iniziale:** La curva sale quasi verticalmente all'estrema sinistra (per FPR vicino a 0). Questo significa che il modello può raggiungere un **Recall di circa il 57% (TPR = 0.57) senza quasi commettere falsi positivi**. Questo punto sulla curva, con FPR ≈ 0, è esattamente dove si è posizionato il tuo modello con la soglia di default (0.5), spiegando la tua Precisione perfetta e il Recall del 57%.
    - **Andamento a "Scalini":** La curva non è liscia ma procede a "scalini". Questo è tipico quando il numero di campioni positivi (anomalie) è relativamente piccolo. Ogni "salto" verticale corrisponde a una soglia che permette di classificare correttamente uno o più campioni anomali.
    - **Compromesso per un Recall più Alto:** Per superare un Recall del 60%, la curva inizia a piegare decisamente verso destra. Ad esempio, per raggiungere un Recall dell'80% (TPR = 0.8), si deve accettare un Tasso di Falsi Positivi di circa il 50% (FPR = 0.5). Questo sarebbe un modello molto "rumoroso", con un altissimo numero di falsi allarmi.

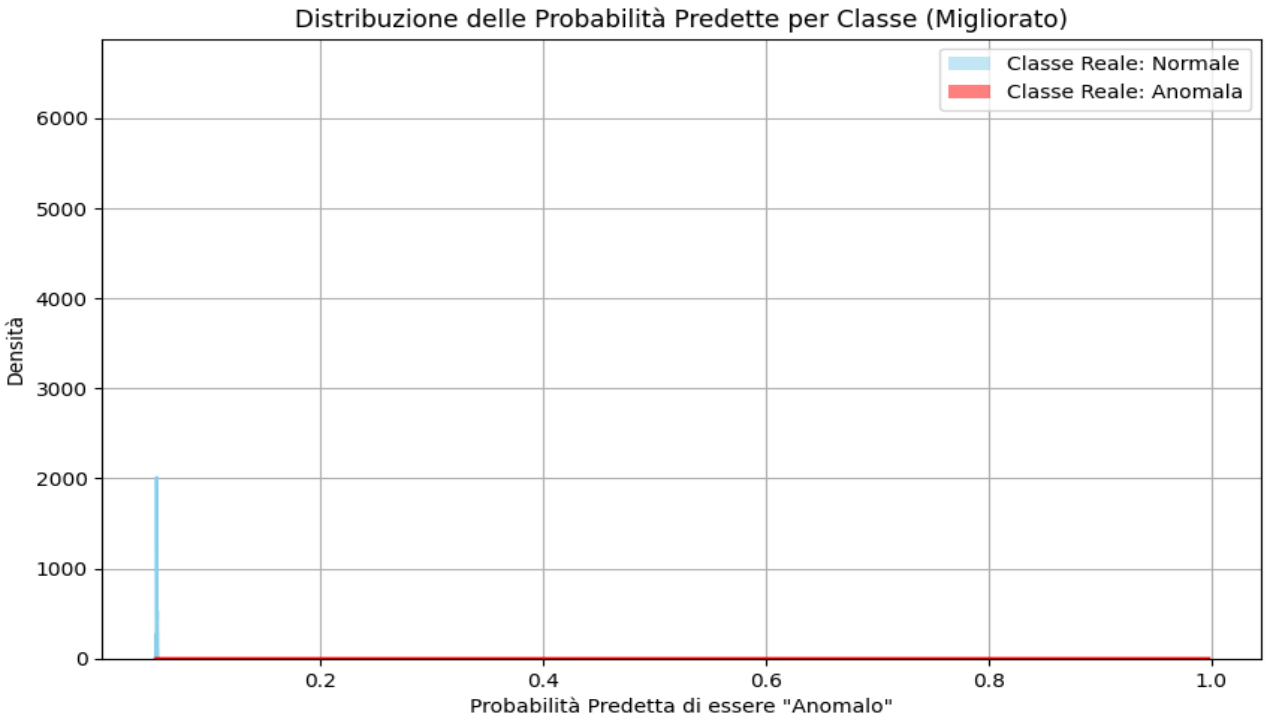
Analisi: Questo grafico è fondamentale perché **quantifica e visualizza il compromesso** che il tuo modello deve affrontare. L'AUC di 0.77 definisce i limiti della sua capacità predittiva: non esiste una soglia di decisione che possa fornire contemporaneamente un Recall molto alto e un Tasso di Falsi Positivi molto basso. La performance ottenuta (Precisione 1.0, Recall 0.57) non è un difetto, ma una **scelta operativa razionale** su questa curva, che privilegia l'eliminazione totale dei falsi allarmi a scapito di una copertura incompleta delle minacce.

Descrizione degli Assi e delle Legende

- **Asse X (Probabilità Predetta di essere "Anomalo"):** Rappresenta il punteggio di output della funzione sigmoide del modello, che va da 0 a 1. Un valore vicino a 1 indica un'alta confidenza del modello che la sequenza sia anomala; un valore vicino a 0 indica un'alta confidenza che sia normale.
- **Asse Y (Densità):** Indica la frequenza relativa dei punteggi di probabilità. Un picco alto in una certa regione dell'asse X significa che molti campioni hanno ricevuto un punteggio di probabilità in quel range.
- **Distribuzione Blu (Classe Reale: Normale):** Mostra i punteggi assegnati a tutte le sequenze che erano *effettivamente* normali nel set di test.
- **Distribuzione Rossa (Classe Reale: Anomala):** Mostra i punteggi assegnati a tutte le sequenze che erano *effettivamente* anomale.

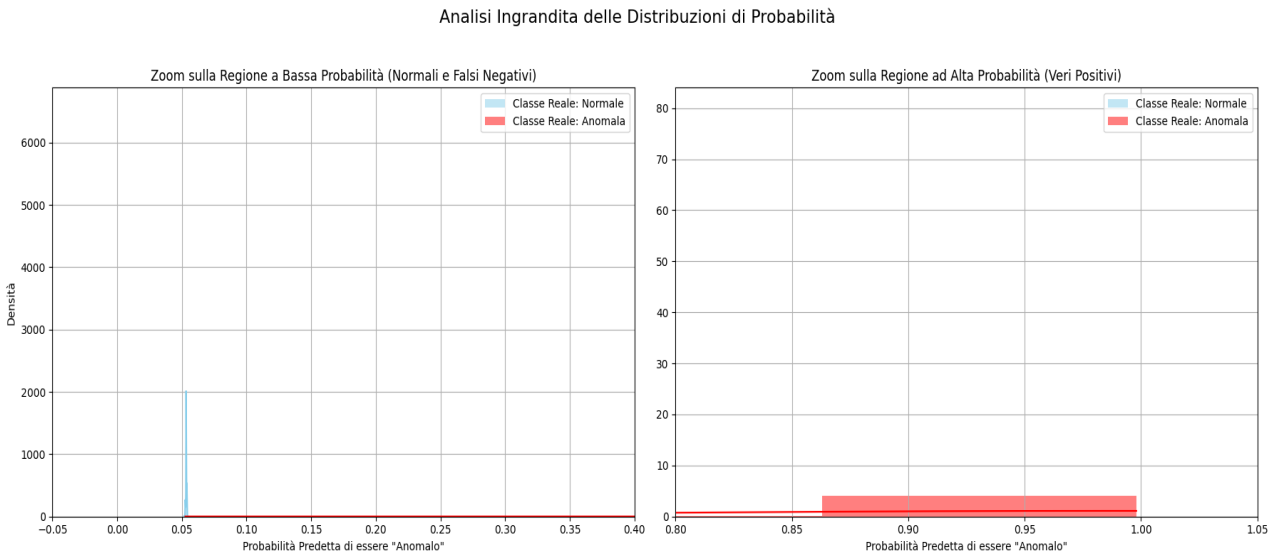
L'obiettivo di un classificatore ideale è separare nettamente queste due distribuzioni, spingendo quella blu verso 0 e quella rossa verso 1.

Grafico: Distribuzione delle Probabilità (Vista Panoramica)



Analisi: La vista panoramica mostra immediatamente il comportamento netto e specializzato del modello. La distribuzione Normale (blu) è rappresentata da un picco altissimo e quasi impercettibilmente sottile, concentrato all'estrema sinistra dell'asse. Al contrario, la distribuzione Anomala (rossa) appare come una linea quasi piatta, suggerendo che i suoi campioni sono distribuiti in modo molto più sparso. Per esaminare in dettaglio queste dinamiche, sono stati generati due grafici ingranditi che si focalizzano sulle regioni di maggiore interesse.

Grafico: Analisi Ingrandita delle Distribuzioni di Probabilità



Analisi Dettagliata:

1. Zoom sulla Regione a Bassa Probabilità (Grafico a Sinistra):
Questo grafico si concentra sull'intervallo di probabilità [0, 0.4] e rivela due fenomeni cruciali:

- **Estrema Confidenza sui Campioni Normali:** La distribuzione blu si manifesta come un picco di densità molto elevato, centrato precisamente intorno a un valore di probabilità di circa **0.05**. Ciò significa che il modello è eccezionalmente coerente e sicuro nell'identificare le sequenze normali, assegnando loro un punteggio molto lontano dalla soglia di decisione di 0.5. Questa netta separazione è la causa diretta dell'**assenza di Falsi Positivi** e, di conseguenza, della **Precisione perfetta (1.00)** del modello.

- **Visualizzazione dei Falsi Negativi:** In questa stessa regione, è visibile una debole ma presente distribuzione rossa. Questa rappresenta le **vere anomalie** a cui il modello ha erroneamente assegnato un punteggio di probabilità molto basso, confondendole con il comportamento normale. Questi campioni costituiscono i **Falsi Negativi** del modello, ovvero le minacce mancate che ne limitano il Recall.

2. Zoom sulla Regione ad Alta Probabilità (Grafico a Destra):
Questo grafico si concentra sull'intervallo [0.8, 1.0] per analizzare come il modello gestisce le anomalie che riesce a riconoscere:

- **Assenza di Campioni Normali:** Come previsto, la distribuzione blu è completamente assente in questa regione. Nessun campione normale riceve un punteggio alto, riconfermando l'assenza di falsi allarmi.

- **Visualizzazione dei Veri Positivi:** In questa zona emerge chiaramente una distribuzione rossa, che rappresenta le **vere anomalie** a cui il modello ha correttamente assegnato un punteggio di probabilità molto alto. Questi campioni, superando la soglia di 0.5, vengono classificati correttamente e costituiscono i **Veri**

Positivi. È la presenza di questa distribuzione a destra che permette al modello di raggiungere un **Recall del 57%**.

Conclusione dell'Analisi Visiva:
Complessivamente, l'analisi ingrandita delle probabilità dipinge il ritratto di un classificatore altamente specializzato. Il modello ha sviluppato una comprensione eccellente di cosa costituisce una sequenza "normale", permettendogli di operare senza generare falsi positivi. Tuttavia, la sua comprensione di cosa sia "anomalo" è meno definita, riuscendo a identificare con certezza solo un sottoinsieme delle vere minacce. Questo comportamento "cauto" è la manifestazione del compromesso strategico discusso nella tesi, privilegiando l'affidabilità assoluta degli alert a scapito di una copertura completa.

---

5. Discussione e Conclusioni Tecniche

L'insieme dei tre casi di studio presentati in questo report fornisce una serie di lezioni tecniche pratiche sull'applicazione di modelli di Machine Learning e Deep Learning a problemi reali di cybersecurity. Al di là dei risultati numerici specifici, ogni esperimento ha rivelato insight fondamentali sulla scelta dei modelli, sulla gestione dei dati e sulle sfide dell'addestramento.

L'esperimento 1 (RF vs. MLP) ha confermato in modo netto una lezione ampiamente discussa in letteratura: su **dati tabulari ben strutturati e con feature ingegnerizzate, i modelli di Machine Learning classici come la Random Forest rimangono una scelta eccezionalmente robusta e spesso superiore**. La capacità intrinseca della RF di gestire interazioni complesse tra le feature e la sua resilienza all'overfitting le hanno permesso di superare un'architettura di Deep Learning generica come l'MLP. Tecnicamente, questo risultato sottolinea che l'adozione di un modello DL non è sempre la scelta ottimale; il vero potenziale del Deep Learning in questo contesto risiede in architetture specializzate (come le CNN o le RNN) capaci di eseguire un feature learning automatico da dati grezzi, un compito che l'MLP su dati tabulari non svolge.

L'esperimento 2 (CNN su Malware) ha dimostrato il grande potenziale del Deep Learning nell'analisi di **dati non strutturati**, come le rappresentazioni visuali dei malware. L'architettura CNN è stata in grado di raggiungere un'elevata accuratezza complessiva apprendendo autonomamente pattern e "texture" visive direttamente dai pixel. Tuttavia, l'analisi approfondita degli errori ha rivelato una limitazione critica: il modello fatica enormemente quando le **feature visive non sono sufficientemente discriminanti**. La confusione tra varianti della stessa famiglia (Swizzor.gen) e il fallimento completo su classi visivamente ambigue (Autorun.K) suggeriscono che un approccio basato su una singola modalità di analisi è fragile. La lezione tecnica è che, per problemi complessi come la classificazione del malware, le soluzioni più robuste sono probabilmente **ibride e multimodali**, integrando l'analisi visuale con feature estratte da altre fonti (es. analisi statica delle API, analisi dinamica del comportamento).

Infine, **l'esperimento 3 (LSTM su Log)** è stato il più istruttivo riguardo alle sfide dell'addestramento su dati sbilanciati e rumorosi. I tentativi iniziali hanno dimostrato come una combinazione di iperparametri non ottimale (in particolare un class_weight troppo aggressivo) possa portare a un **collasso catastrofico del processo di addestramento**, con instabilità numerica e risultati illogici. La lezione tecnica fondamentale è stata l'importanza di un approccio metodico e iterativo al tuning. La soluzione finale ha dimostrato come la combinazione di un'architettura regolarizzata (**Batch Normalization** e **Dropout**), un ottimizzatore con un **learning rate** appropriato e l'uso di **callback intelligenti** (EarlyStopping su val_loss) sia cruciale per stabilizzare l'addestramento di una LSTM. Il risultato finale—un classificatore ad altissima Precisione e modesto Recall—non è un difetto, ma la dimostrazione di come un modello stabile, di fronte a un problema con una separabilità delle classi non perfetta (AUC di 0.77), converga verso un **compromesso operativo realistico**, privilegiando l'affidabilità degli allarmi.

In sintesi, questi esperimenti non solo validano l'efficacia dell'IA nella sicurezza, ma sottolineano soprattutto che non esiste una soluzione universale. La scelta del modello, la robustezza della pipeline di pre-processing e un'attenta calibrazione degli iperparametri sono passaggi critici e interdipendenti, il cui successo determina la differenza tra un modello teoricamente potente e uno praticamente utile.