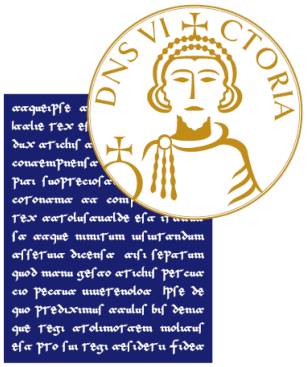


Università degli Studi del Sannio

Dipartimento di Ingegneria

Corso di Laurea Magistrale in Ingegneria Informatica

Data science

Homework 4

Prof:
Pecchia Antonio

Studenti:
Cinelli Jessica, 399000529
Mazzitelli Francesco C., 399000532

Anno Accademico 2022-2023

Indice

Elenco delle figure

1 Introduzione

Il seguente studio[1] ha avuto come obiettivo l'analisi e l'applicazione di alcune tipologie di algoritmi di Deep Learning per la classificazione di dati.
L'approccio ha previsto la creazione di un'architettura volta ad emulare il funzionamento del cervello umano tramite delle componenti chiamate **Percettroni** o **Neuroni**.
Un **neurone** è un classificatore lineare binario, il quale riceve in input un vettore di valori reali e restituisce una decisione booleana. Queste componenti possono essere raggruppate in tre layer fondamentali:

- **Layer di input**: non è un livello elaborativo, si occupa della distribuzione dei dati ai livelli successivi
- **Layer nascosto**: effettuano elaborazioni sull'input, classificandolo. E' un layer altamente parametrizzato in quanto c'è la possibilità di impostare il numero di neuroni presenti nel livello, selezionare la funzione di attivazione, specificare se selezionare un determinato input da processore o meno e all'evenienza creare ulteriori layer nascosti collegandoli in cascata.
- **Layer di output**: effettua la somma pesata dei livelli precedenti generando la predizione.

Il processo di apprendimento associato al modello può essere di due tipi:

- **Supervisionato**: se è disponibile un insieme di valori attesi con il quale confrontare i valori ottenuti dall modello previsionale.
- **Non supervisionato**: s il processo di classificazione si basa solo sul valore dei dati in input. Un esempio sono gli algoritmi di clustering in quanto producono direttamente in output dei raggruppamenti.

Sono stati analizzati dati relativi al traffico di rete associato alle connessioni ad un server, con lo scopo di riuscire a classificare e separare le connessioni benigne, a carico di normali user, da quelle maligne. Nei file utilizzati sono state individuate diverse tipologie di *Denial of Service*:

- DoS GoldenEye
- DoS Hulk
- DoS Slowloris
- DoS Slowhttptest

La seprazione dei vari insiemi di addestramento, validazione e test è stata effettuata staticamente, passando singolarmente i file al modello di interesse e richiamando delle funzioni specifiche per ogni fase, in esso definite.

[1] È possibile visionare l'intero progetto al link https://github.com/jessicacinelli/Homework4.git.

2 Inserimento dei file

Le operazioni di lettura del contenuto dei file e di selezione delle features di interesse è stata demandata allo script python "utils.py". I file non contenevano all'interno labels, quindi le stesse sono state riportate in una lista

```
features = ['Flow Duration', 'Total Fwd Packet','Total Bwd packets','Total Length of Fwd
    Packet','Total Length of Bwd Packet', 'Fwd Packet Length Max','Fwd Packet Length Min','Fwd
    Packet Length Mean','Fwd Packet Length Std','Bwd Packet Length Max','Bwd Packet Length
    Min','Bwd Packet Length Mean','Bwd Packet Length Std', 'Flow Bytes/s','Flow Packets/s','Flow
    IAT Mean','Flow IAT Std','Flow IAT Max','Flow IAT Min', 'Fwd IAT Total','Fwd IAT Mean','Fwd
    IAT Std','Fwd IAT Max','Fwd IAT Min','Bwd IAT Total','Bwd IAT Mean', 'Bwd IAT Std','Bwd IAT
    Max','Bwd IAT Min','Fwd PSH Flags','Bwd PSH Flags','Fwd URG Flags','Bwd URG Flags', 'Fwd RST
    Flags','Bwd RST Flags','Fwd Header Length','Bwd Header Length','Fwd Packets/s','Bwd
    Packets/s', 'Packet Length Min','Packet Length Max','Packet Length Mean','Packet Length
    Std','Packet Length Variance', 'FIN Flag Count','SYN Flag Count','RST Flag Count','PSH Flag
    Count','ACK Flag Count','URG Flag Count', 'CWR Flag Count','ECE Flag Count','Down/Up
    Ratio','Average Packet Size','Fwd Segment Size Avg', 'Bwd Segment Size Avg','Fwd Bytes/Bulk
    Avg','Fwd Packet/Bulk Avg','Fwd Bulk Rate Avg','Bwd Bytes/Bulk Avg','Bwd Packet/Bulk Avg','Bwd
    Bulk Rate Avg','Subflow Fwd Packets','Subflow Fwd Bytes','Subflow Bwd Packets', 'Subflow Bwd
    Bytes','FWD Init Win Bytes','Bwd Init Win Bytes','Fwd Act Data Pkts','Fwd Seg Size
    Min','Active Mean','Active Std','Active Max','Active Min','Idle Mean','Idle Std','Idle
    Max','Idle Min','ICMP Code','ICMP Type','Total TCP Flow Time']
```

Grazie alla definizione delle features di interesse è stato possibile effettuare la lettura dei dataset

```
def transformData ( training, training_ae, validation, test ):
    dfTrain = pd.read_csv(training, names=names, header=None, sep=',', index_col=False,
        dtype='unicode')
    dfTrainAe = pd.read_csv(training_ae, names=names, header=None, sep=',', index_col=False,
        dtype='unicode')
    dfValidation = pd.read_csv(validation, names=names, header=None, sep=',', index_col=False,
        dtype='unicode')
    dfTest = pd.read_csv(test, names=names, header=None, sep=',', index_col=False, dtype='unicode')
    x_train, y_train, L_train = getXY(dfTrain)
    x_train_ae, y_train_ae, L_train_ae = getXY(dfTrainAe)
    x_val, y_val, L_val = getXY(dfValidation)
    x_test, y_test, L_test = getXY(dfTest)
    scaler = MaxAbsScaler()
    x_train = scaler.fit_transform(x_train)
    x_train_ae = scaler.fit_transform(x_train_ae)
    x_val = scaler.transform(x_val)
    x_test = scaler.transform(x_test)
    return x_train, y_train, L_train, x_train_ae, y_train_ae, L_train_ae, x_val, y_val, L_val,
        x_test, y_test, L_test
```

Una volta effettuata la lettura dei dataset, sono state estratte le variabili di interesse x, y, L. Per l'estrazione di y (colonne) è stato ritenuto apportuno aggiungere altre due colonne con due bit per l'identificazione della connessione come benigna o maligna

```
def getXY ( inDataframe ):
    # 1) indataframe -> x (solo le features)
    x = inDataframe[features].values.astype(float)

    # 2) y    (1,0) -> BENIGN   /    (0,1) -> DoS
    bitA = np.where(inDataframe['Label'] == 'BENIGN', 1, 0)
    bitB = np.where(inDataframe['Label'] == 'BENIGN', 0, 1)

    y = np.column_stack((bitA, bitB))

    # 3) Label originali in forma String
    L = inDataframe['Label'].values
    return x, y, L
```

Infine è stata definita una funzione in grado di valutare le performance del modello

```
def evaluatePerformance(outcome, evaluationLabels):
       eval = pd.DataFrame( data={'prediction':outcome, 'Class':evaluationLabels} )
       ...
       for c in classes:
           if c != 'BENIGN':
               A = eval[(eval['prediction'] == True) & (eval['Class'] == c)].shape[0]
               B = eval[(eval['prediction'] == False) & (eval['Class'] == c)].shape[0]
               print ( '%6d %10d %10.3f %26s' %(A, B, B / (A + B), c) )
               FN = FN + A    # cumulative FN
               TP = TP + B    # cumulative TP
           else:
               TN = eval[(eval['prediction'] == True) & (eval['Class'] == 'BENIGN')].shape[0]
               FP = eval[(eval['prediction'] == False) & (eval['Class'] == 'BENIGN')].shape[0]
       print('%6s %10s' % ('----', '----'))
       print('%6d %10d %10s' % (FN, TP, 'total'))
       print('')
       print('Confusion matrix:')
       print('%42s' % ('prediction'))
       print('%36s | %14s' % (' | BENIGN (neg.)','ATTACK (pos.)'))
       print('      --------------|---------------|---------------')
       print('%28s %6d | FP = %9d' % ('BENIGN (neg.) | TN = ', TN, FP))
       print('label --------------|---------------|---------------')
       print('%28s %6d | TP = %9d' % ('ATTACK (pos.) | FN = ', FN, TP))
       print('      --------------|---------------|---------------')
       recall = TP / (TP + FN)
       precision = 0
       if TP + FP != 0: precision = TP / (TP + FP)
       f1 = 0
       if precision + recall != 0:
           f1=2 * ((precision * recall) / (precision + recall))
       fpr = FP / (FP + TN)
       print('R = %5.3f P = %5.3f F1 score = %5.3f FPR = %5.3f' % (recall, precision, f1, fpr))
```

3 Creazione e addestramento del modello FeedForward

In questa rete neurale le informazioni si muovono solo in avanti rispetto a nodi d'ingresso, attraversano i layer nascosti e infine raggiungono i nodi d'uscita. Una caratteristica della rete è che non sono presenti cicli.

```
class FeedforwardNN():

    def __init__(self, input_dim):
        input_layer = Input(shape=(input_dim, ))
        layer = Dense(80, activation='relu',
             kernel_initializer=initializers.RandomNormal())(input_layer)
        layer = Dense(20, activation='relu', kernel_initializer=initializers.RandomNormal())(layer)
        layer = Dense(4, activation='relu', kernel_initializer=initializers.RandomNormal())(layer)
        layer = Dense(2, activation='relu', kernel_initializer=initializers.RandomNormal())(layer)
        output_layer = Activation(activation='softmax')(layer)
        self.classifier = Model(inputs=input_layer, outputs=output_layer)

    def summary(self, ):
        self.classifier.summary()

    def train(self, x, y):
        epochs = 100
        batch_size = 1024
        validation_split = 0.1
        self.classifier.compile(optimizer='rmsprop', loss='categorical_crossentropy')
        history = self.classifier.fit(x, y, batch_size=batch_size, epochs=epochs,
             validation_split=validation_split, shuffle=True, verbose=2)
        plt.plot(history.history['loss'])
        plt.plot(history.history['val_loss'])
        plt.title('model loss')
        plt.ylabel('loss')
        plt.xlabel('epoch')
        plt.legend(['training', 'validation'], loc='upper right')
        plt.show()
        df_history = pd.DataFrame(history.history)
        return df_history

    def predict ( self, x_evaluation ):
        predictions = self.classifier.predict(x_evaluation)
        outcome = predictions[:, 0] > predictions[:, 1]
        return outcome
```

4 Creazione e addestramento del modello AutoEncoder

L'architettura di un Autoencoder prevede due componenti principali:

- **Encoder**: riceve i dati in input e ne fornisce una rappresentazione compressa o latent space representation
- **Decoder**: riceve in input la versione compressa del dato e lo riproduce con il minor errore possibile.

Un Autoencoder è caratterizzato da:

- Specificità del dato, non è possibile utilizzare il modello addestrato per la classificazione di altri oggetti;
- Un output non è esattamente uguale all'input;
- Apprendimento non supervisionato, non necessita di etichette di classificazione.

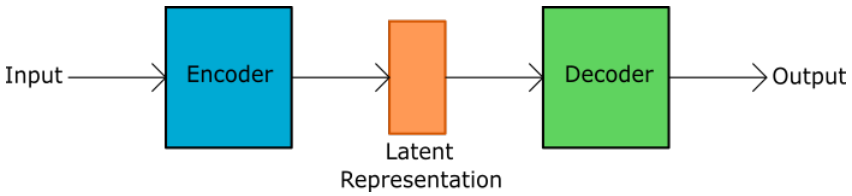


Figura 1: Schema di funzionamento del modello

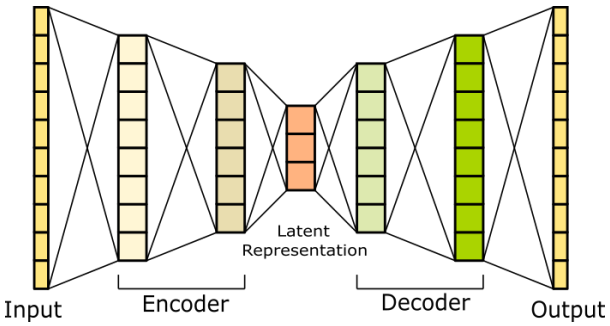


Figura 2: Schema di una rete AutoEncoder

La classe contenente la definizione e l'addestramento del modello è la seguente:

```
class AutoEncoder():
    def __init__(self, input_dim):
        input_layer= Input (shape=(input_dim,))
        layer = Dense( 64, activation='relu',
            kernel_initializer=initializers.RandomNormal)(input_layer)
        layer = Dense( 32, activation='relu', kernel_initializer=initializers.RandomNormal)(layer)
        layer = Dense( 16, activation='relu', kernel_initializer=initializers.RandomNormal)(layer)
        layer = Dense( 8, activation='relu', kernel_initializer=initializers.RandomNormal)(layer)
        layer = Dense( 16, activation='relu', kernel_initializer=initializers.RandomNormal)(layer)
        layer = Dense( 32, activation='relu', kernel_initializer=initializers.RandomNormal)(layer)
        layer = Dense( 64, activation='relu', kernel_initializer=initializers.RandomNormal)(layer)
        output_layer= Dense(input_dim,
            activation='relu',kernel_initializer=initializers.RandomNormal)(layer)
        self.autoencoder = Model(inputs=input_layer, outputs = output_layer)

    def summary(self, ):
        self.autoencoder.summary()

    def train(self, x, y):

        epochs = 200
        batch_size=1024
        validation_split= 0.1 #(10-15 % dei dati)

        #rmsprop funziona propagando l'errore
        #tipicamente la loss dipende dal problema. Con la loss valutiamo la diff tra la
            ricostruzione e l'input vero e proprio
        #conviene scegliere la funzioen di loss che  la mean squared error
        self.autoencoder.compile(optimizer='Nadam', loss='mean_squared_error')
        #shuffle serve a prelevare in maniera random per non polarizzare il train
        history= self.autoencoder.fit(x,y, epochs=epochs, batch_size=batch_size,
            validation_split=validation_split, shuffle=True, verbose=2)

        plt.plot(history.history['loss'])
        plt.plot(history.history['val_loss'])
        plt.title('model loss')
        plt.ylabel('loss')
        plt.xlabel('epoch')
        plt.legend(['training', 'validation'], loc='upper right')
        plt.show()

        x_thSet = x[x.shape[0]-(int)(x.shape[0]*validation_split):x.shape[0]-1, :]
        self.threshold = self.computeThreshold (x_thSet)
        print("Threashold: " + str(self.threshold))
        df_history = pd.DataFrame(history.history)
        return df_history
```

```
def predict(self, x_evaluation):
    #1.
    #predizioni : x ricostruite in uscita
    reconstructions =self.autoencoder.predict(x_evaluation)
    #2.
    #calcoliamo RE input vs reconstr.
    #media dei quadrati delle differenze:
    #x_evaluation-reconstructions -> vettore delle differenzr
    RE = np.mean(np.power(x_evaluation-reconstructions, 2) ,axis=1)
    #3.
    #confronto RE-threshold
    #confronta RE con la soglia e ricostruisce
    outcome = RE<=self.threshold
    return outcome

def computeThreshold ( self, x_thSet ):
    x_thSetPredictions = self.autoencoder.predict(x_thSet)
    mse = np.mean(np.power(x_thSet - x_thSetPredictions, 2),axis=1)
    threshold = np.percentile(mse, 95)
    return threshold

def plot_reconstruction_error(self, x_evaluation, evaluationLabels):
    predictions = self.autoencoder.predict(x_evaluation)
    mse = np.mean(np.power(x_evaluation - predictions, 2), axis=1)
    trueClass = evaluationLabels != 'BENIGN'
    errors = pd.DataFrame({'reconstruction_error': mse, 'true_class': trueClass})
    groups = errors.groupby('true_class')
    fig, ax = plt.subplots(figsize=(8, 5))
    right = 0
    for name, group in groups:
        if max(group.index) > right: right = max(group.index)
        ax.plot(group.index, group.reconstruction_error, marker = 'o', ms = 5, linestyle = '',
            markeredgecolor = 'black', #alpha = 0.5,
            label = 'Normal' if int(name) == 0 else 'Attack', color = 'green' if int(name) == 0
                else 'red')
    ax.hlines(self.threshold, ax.get_xlim()[0], ax.get_xlim()[1], colors = 'red', zorder = 100,
         label = 'Threshold',linewidth=4,linestyles='dashed')
    ax.semilogy()
    ax.legend()
    plt.xlim(left = 0, right = right)
    plt.title('Reconstruction error for different classes')
    plt.grid(True)
    plt.ylabel('Reconstruction error')
    plt.xlabel('Data point index')
    plt.show()
```

5 Esecuzione dei modelli

Dopo aver definito i dataset da analizzare e i vari modelli, gli stessi vengono mandati in esecuzione all'interno di un script python "maindl.py"

```
train = 'hw4Data\\TRAIN.csv'
train_ae = 'hw4Data\\TRAIN_AE.csv'
validation = 'hw4Data\\VALIDATION.csv'
test = 'hw4Data\\TEST.csv'

x_train, y_train, L_train, x_train_ae, y_train_ae, L_train_ae, x_val, y_val, L_val, x_test, y_test,
     L_test = transformData(train, train_ae, validation, test)

input_dim = x_train.shape[1]

ffnn = FeedforwardNN(input_dim=input_dim)
ffnn.summary()
ffnn.train(x_train, y_train)
outcome = ffnn.predict(x_val)
evaluatePerformance(outcome, L_val)


-----------------------------------------------------------------------------------------------------

ae=AutoEncoder(input_dim = input_dim)
ae.summary()
ae.train(x_train_ae, x_train_ae)
outcome=ae.predict(x_val)
evaluatePerformance(outcome, L_val)
ae.plot_reconstruction_error(x_val, L_val)
```

6 Configurazioni

La precisione della rete e valutata mediante i valori di Recall (R), Precision (P) e False Positive Rate (FPR) che vengono calcolati durante la fase di validazione. Sono stati posti i seguenti obiettivi:

- nel caso di Feedforward Neural Network: $Recall \geq 0.980$ per almeno tre dei quattro attacchi DoS e $FPR \leq 0.001$
- nel caso di Autoender: $Recall \geq 0.950$ per almeno tre dei quattro attacchi Dos e $FPR \leq 0.050$

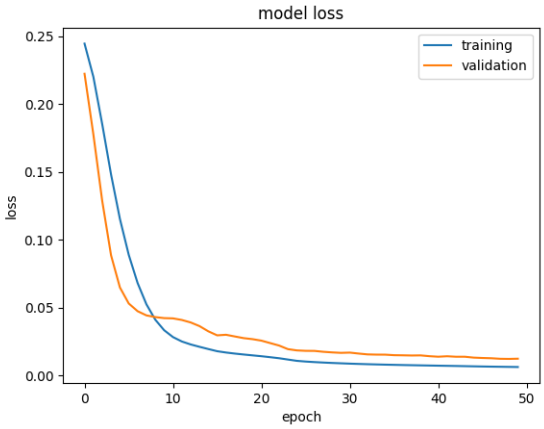
6.1 Feedforward neural network

6.1.1 Macchina 1

Come configurazione iniziale è stata scelta la rete con i seguenti layers:

- input layer: 81 attributi
- hidden layer: 8 neuroni con funzione di attivazione *relu*
- hidden layer: 2 neuroni con funzione di attivazione *relu*
- softmax layer

Il training è stato effettuato con un numero di epoche pari a 50 e batch size pari a 2048. Come ottimizzatore è stato scelto **rmsprop** e l'errore è stato valutato con la metrica **mean_squared_error**. La figura 13(a) riporta l'andamento della loss, mentre la figura 13(b) mostra i risultati ottenuti in fase di validazione del modello.



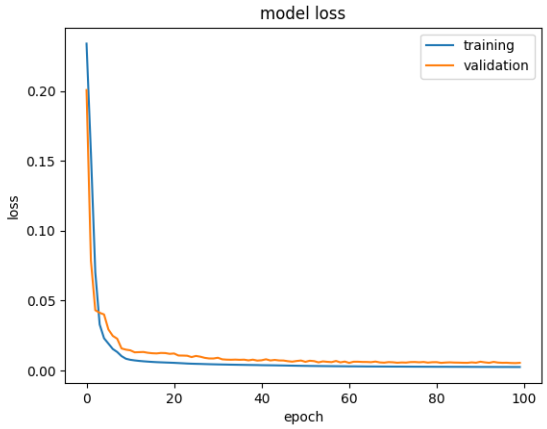
((a)) Loss

```
*** EVALUATION RESULTS ***

Coverage by attack (positive classes)
  FN     TP    recall
  13    191    0.936          DoS GoldenEye
   1   4625    1.000               DoS Hulk
  12     44    0.786        DoS Slowhttptest
  31     83    0.728           DoS Slowloris
----   ----
  57   4943    total

Confusion matrix:
                              prediction
                    | BENIGN (neg.) |  ATTACK (pos.)
      --------------|---------------|---------------
      BENIGN (neg.) | TN =     4984 | FP =        16
label --------------|---------------|---------------
      ATTACK (pos.) | FN =       57 | TP =      4943
      --------------|---------------|---------------
Metrics:
R = 0.989  P = 0.997  F1 score = 0.993  FPR = 0.003
```

((b)) Matrice di confusione e risultati

Figura 3: Risultati sulla macchina 1: Prova 1

Successivamente è stato inserito un ulteriore hidden layer di 16 neuroni, con funzione di attivazione *relu*. La rete è stata addestrata con 100 epoche e batch size pari a 1024. La figura 4(a) riporta l'andamento della loss, mentre la figura 4(b) mostra i risultati ottenuti in fase di validazione del modello.



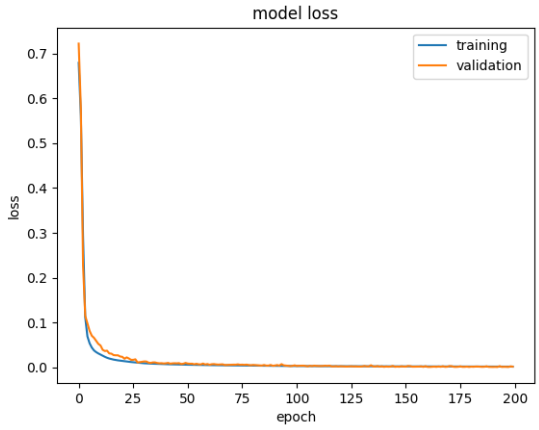
((a)) Loss

```
*** EVALUATION RESULTS ***

Coverage by attack (positive classes)
  FN         TP     recall
   0        204      1.000              DoS GoldenEye
   0       4626      1.000                   DoS Hulk
   2         54      0.964           DoS Slowhttptest
  19         95      0.833              DoS Slowloris
----       ----
  21       4979      total

Confusion matrix:
                                prediction
                     | BENIGN (neg.) |  ATTACK (pos.)
      --------------|---------------|---------------
      BENIGN (neg.) | TN =     4994 | FP =          6
label ---------------|---------------|---------------
      ATTACK (pos.) | FN =       21 | TP =       4979
      --------------|---------------|---------------
Metrics:
R = 0.996  P = 0.999  F1 score = 0.997  FPR = 0.001
```

((b)) Matrice di confusione e risultati

Figura 4: Risultati sulla macchina 1: Prova 2

Per poter raggiungere gli obiettivi prefissati sono state effettuate delle prove utilizzando come funzione di attivazione *elu* e *tanh*. In entrambi i casi non sono stati riscontrati miglioramenti. Si è pensato, dunque, di utilizzare come metrica di loss **categorical_crossentropy**: il modello è stato addestrato utilizzando come funzioni di attivazione la funzione *relu*, un numero di epoche pari a 200 e batch size pari a 1024. Si riporta l'andamento della loss in figura 5(a) e il risultato dell'addestramento in figura 5(b).



((a)) Loss

```
*** EVALUATION RESULTS ***

Coverage by attack (positive classes)
  FN         TP     recall
   0        204      1.000              DoS GoldenEye
   0       4626      1.000                   DoS Hulk
   0         56      1.000           DoS Slowhttptest
   0        114      1.000              DoS Slowloris
----       ----
   0       5000      total

Confusion matrix:
                                prediction
                     | BENIGN (neg.) |  ATTACK (pos.)
      --------------|---------------|---------------
      BENIGN (neg.) | TN =     4997 | FP =          3
label ---------------|---------------|---------------
      ATTACK (pos.) | FN =        0 | TP =       5000
      --------------|---------------|---------------
Metrics:
R = 1.000  P = 0.999  F1 score = 1.000  FPR = 0.001
```

((b)) Matrice di confusione e risultati

Figura 5: Risultati sulla macchina 1: Prova 3

Il modello che riesce a raggiungere gli obiettivi prefissati[2] è il seguente:

- input layer: 81 attributi
- hidden layer: 16 neuroni con funzione di attivazione *relu*
- hidden layer: 8 neuroni con funzione di attivazione *relu*
- hidden layer: 2 neuroni con funzione di attivazione *relu*
- softmax layer

Sono stati utilizzati l'ottimizzatore *rmsprop* e la loss **categorical_crossentropy**, con 200 epoche e batch size pari a 1024. Come step successivo sono state calcolate le metriche sul test set (figura 6).

```
                *** EVALUATION RESULTS ***

           Coverage by attack (positive classes)
      FN         TP     recall
       8        196      0.961              DoS GoldenEye
       2       4624      1.000                   DoS Hulk
       0         56      1.000           DoS Slowhttptest
       3        111      0.974              DoS Slowloris
    ----       ----
      13       4987      total

Confusion matrix:
                                  prediction
                        | BENIGN (neg.) |  ATTACK (pos.)
       --------------|---------------|---------------
       BENIGN (neg.) | TN =     4997 | FP =         3
label  --------------|---------------|---------------
       ATTACK (pos.) | FN =       13 | TP =      4987
       --------------|---------------|---------------
Metrics:
R = 0.997  P = 0.999  F1 score = 0.998  FPR = 0.001
```

Figura 6: Risultati ottenuti sul test set con la configurazione ottimale

È possibile osservare una diminuzione dei valori di recall: i risultati ottenuti sul test set non rispettano i valori soglia fissati come obiettivo.

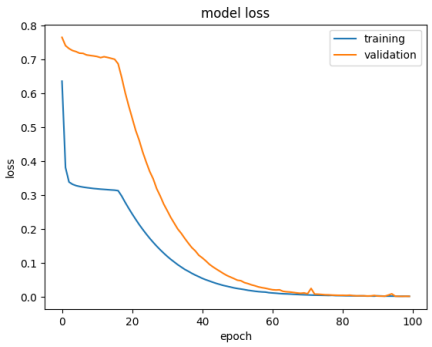
[2]Sono state effettuate ulteriori prove aumentando il numero di neuroni e il numero di epoche e diminuendo la dimensione del batch. In nessun caso sono stati indivituati miglioramenti.

6.1.2 Macchina 2

Come configurazione iniziale è stata scelta la rete con i seguenti layers:

- input layer: 81 attributi
- hidden layer: 64 neuroni con funzione di attivazione *tanh*
- hidden layer: 32 neuroni con funzione di attivazione *relu*
- hidden layer: 16 neuroni con funzione di attivazione *tanh*
- hidden layer: 2 neuroni con funzione di attivazione *relu*
- softmax layer

Il training è stato effettuato con un numero di epoche pari a 100 e batch size pari a 1024. Come ottimizzatore è stato scelto **Nadam** e l'errore è stato valutato con la metrica **categorical_crossentropy**.
La figura 7(a) riporta l'andamento della loss, mentre la figura 7(b) mostra i risultati ottenuti in fase di validazione del modello.



((a)) Loss

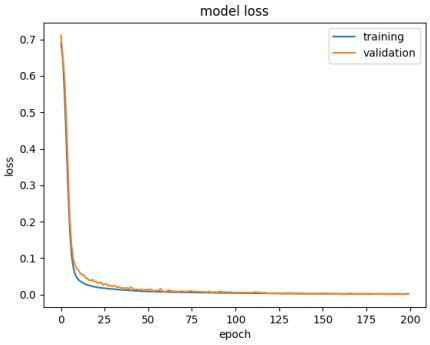
```
*** EVALUATION RESULTS ***

Coverage by attack (positive classes)
  FN     TP    recall
   0    204     1.000          DoS GoldenEye
   0   4626     1.000               DoS Hulk
   0     56     1.000       DoS Slowhttptest
   2    112     0.982          DoS Slowloris
 ----   ----
   2   4998     total

Confusion matrix:
                        prediction
                | BENIGN (neg.) |  ATTACK (pos.)
      ----------|---------------|---------------
      BENIGN (neg.) | TN =    4714 | FP =       286
label ----------|---------------|---------------
      ATTACK (pos.) | FN =       2 | TP =      4998
      ----------|---------------|---------------
Metrics:
R = 1.000  P = 0.946  F1 score = 0.972  FPR = 0.057
```

((b)) Matrice di confusione e risultati

Figura 7: Risultati sulla macchina 2: Prova 1

Successivamente, al fine di raggiungere gli obbiettivi e dopo aver sperimentato modificando i parametri, sono stati ridotti a 3 gli hidden layers. La rete è stata addestrata con 200 epoche e batch size pari a 1024. La figura 8(a) riporta l'andamento della loss, mentre la figura 8(b) mostra i risultati ottenuti in fase di validazione del modello. Nel dettaglio la configurazione si presenta come segue:

- input layer: 81 attributi
- hidden layer: 80 neuroni con funzione di attivazione *tanh*
- hidden layer: 16 neuroni con funzione di attivazione *tanh*
- hidden layer: 2 neuroni con funzione di attivazione *relu*
- softmax layer



((a)) Loss

```
*** EVALUATION RESULTS ***

Coverage by attack (positive classes)
  FN     TP     recall
   0    204      1.000              DoS GoldenEye
   0   4626      1.000                   DoS Hulk
   0     56      1.000           DoS Slowhttptest
   2    112      0.982              DoS Slowloris
 ----   ----
   2   4998      total

Confusion matrix:
                          prediction
                | BENIGN (neg.) |  ATTACK (pos.)
      ----------|---------------|---------------
      BENIGN (neg.) | TN =     4973 | FP =         27
label ----------|---------------|---------------
      ATTACK (pos.) | FN =        2 | TP =       4998
      ----------|---------------|---------------
Metrics:
R = 1.000  P = 0.995  F1 score = 0.997  FPR = 0.005
```

((b)) Matrice di confusione e risultati

Figura 8: Risultati sulla macchina 2: Prova 2

Sfortunatamente su questa macchina non è stato possibile riuscire a raggiungere l'obiettivo, in quanto il valore minimo relativo alla frequenza di falsi positivi corrisponde a quello riportato in figura 8(b).

Sono state testate configurazioni con combinazioni dei seguenti parametri:

- Funzione di attivazione: relu, elu, selu, tanh
- Epoche: 100, 150, 200, 300, 400
- Batch size: 512, 1024, 2048, 4096
- Ottimizzatore: Nadam, rmsprop, sgd, adadelta

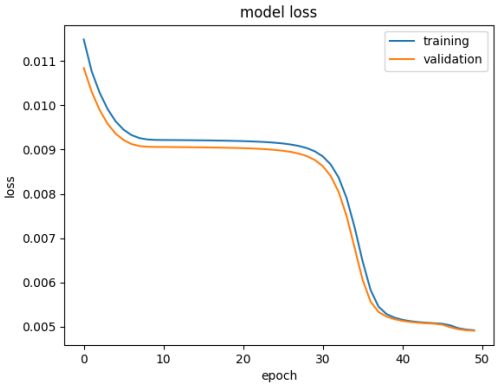
6.2 Autoencoder

6.2.1 Macchina 1

Come configurazione iniziale è stata scelta la rete con i seguenti layers:

- input layer: 81 attributi
- hidden layer: 16 neuroni con funzione di attivazione *relu*
- hidden layer: 8 neuroni con funzione di attivazione *relu*
- hidden layer: 16 neuroni con funzione di attivazione *relu*
- output layer: 81 neuroni con funzione di attivazione *relu*

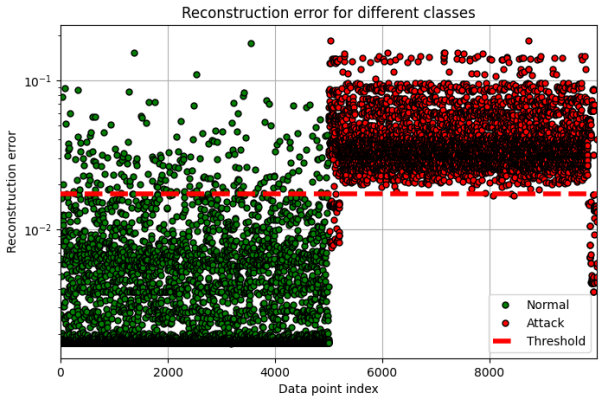
Il training è stato effettuato con un numero di epoche pari a 50 e batch size pari a 2048. Come ottimizzatore è stato scelto **rmsprop** e l'errore è stato valutato con la metrica **mean_squared_error**. La figura 9 riporta i risultati ottenuti.



((a)) Loss

```
*** EVALUATION RESULTS ***

Coverage by attack (positive classes)
  FN    TP    recall
  53   151    0.740          DoS GoldenEye
   2  4624    1.000               DoS Hulk
  15    41    0.732       DoS Slowhttptest
  77    37    0.325          DoS Slowloris
 ---   ---
 147  4853    total

Confusion matrix:
                          prediction
                  | BENIGN (neg.) |  ATTACK (pos.)
      ------------|---------------|---------------
      BENIGN (neg.) | TN =   4771 | FP =      229
label ------------|---------------|---------------
      ATTACK (pos.) | FN =    147 | TP =     4853
      ------------|---------------|---------------
Metrics:
R = 0.971  P = 0.955  F1 score = 0.963  FPR = 0.046
```

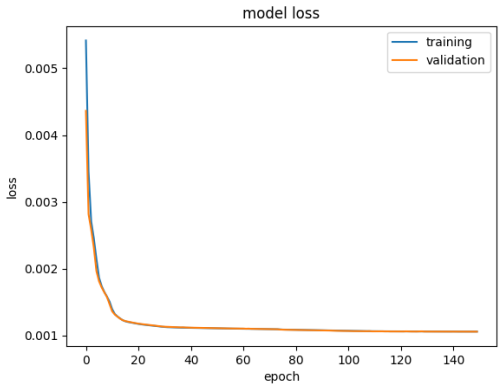
((b)) Matrice di confusione e risultati



((c)) Errore di ricostruzione

Figura 9: Risultati sulla macchina 1: Prova 1

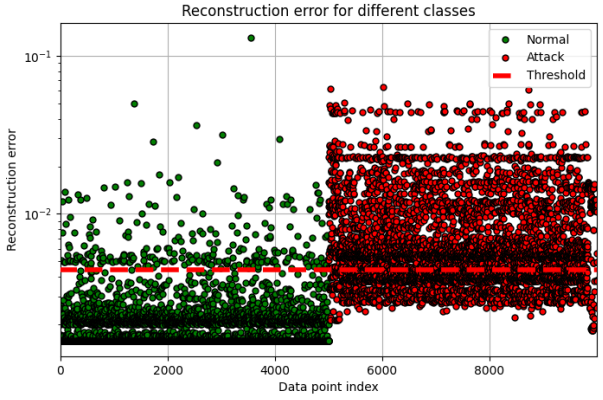
Successivamente sono state effettuate delle prove aumentando il numero di epoche a 100, diminuendo la batch size a 1024, cambiando la funzione di attivazione con *elu*. In nessun caso sono stati individuati miglioramenti. Si è deciso quindi di utilizzare l'ottimizzatore *Nadam*, la funzione di loss *huber* e funzioni di attivazione *relu*. La rete è stata addestrata con 150 epoche e batch size pari a 1024. La figura 12 riporta l'andamento della loss(10(a)), i risultati ottenuti in fase di validazione del modello (10(b)) e l'errore di ricostruzione (10(c)).



((a)) Loss

```
*** EVALUATION RESULTS ***

Coverage by attack (positive classes)
  FN     TP     recall
  51     153    0.750          DoS GoldenEye
1654     2972   0.642          DoS Hulk
   8     48     0.857          DoS Slowhttptest
  26     88     0.772          DoS Slowloris
----   ----
1739     3261   total

Confusion matrix:
                              prediction
                    | BENIGN (neg.) |  ATTACK (pos.)
      --------------|---------------|---------------
      BENIGN (neg.) | TN =     4737 | FP =       263
label --------------|---------------|---------------
      ATTACK (pos.) | FN =     1739 | TP =      3261
      --------------|---------------|---------------
Metrics:
R = 0.652  P = 0.925  F1 score = 0.765  FPR = 0.053
```

((b)) Matrice di confusione e risultati



((c)) Errore di ricostruzione

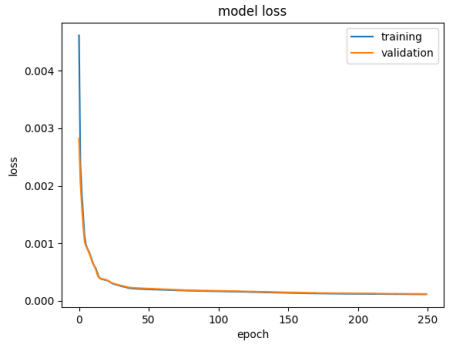
Figura 10: Risultati sulla macchina 1: Prova 2

Per poter raggiungere gli obiettivi prefissati sono state effettuate delle prove inserendo un nuovo hidden layer di 32 neuroni e utilizzando un numero di epoche pari a 200 e come funzione di attivazione *elu* e *selu*. Nel primo caso non sono stati riscontrati miglioramenti.

Nel secondo caso, invece, i valori sono risultati molto vicini agli obiettivi. Sono stati effettuati nuovi addestramenti, aumentando il numero di neuroni e il numero di epoche e diminuendo la dimensione del batch. In nessun caso sono stati indivituati miglioramenti[3].
La configurazione che più si avvicina agli obiettivi fissati è la seguente:

- input layer: 81 attributi
- hidden layer: 32 neuroni con funzione di attivazione *selu*
- hidden layer: 16 neuroni con funzione di attivazione *selu*
- hidden layer: 8 neuroni con funzione di attivazione *selu*
- hidden layer: 16 neuroni con funzione di attivazione *selu*
- hidden layer: 32 neuroni con funzione di attivazione *selu*
- output layer: 81 neuroni con funzione di attivazione *selu*

La rete è stata addestrata con l'ottimizzatore *Nadam* e la loss **huber**, con un numero di epoche pari 200 epoche e batch size pari a 1024. Si riporta l'andamento della loss in figura 11(a), il risultato dell'addestramento in figura 11(b) e l'errore di ricostruzione in figura 11(c).



((a)) Loss

```
*** EVALUATION RESULTS ***

Coverage by attack (positive classes)
 FN         TP      recall
 11        193       0.946              DoS GoldenEye
  2       4624       1.000                   DoS Hulk
  0         56       1.000           DoS Slowhttptest
 17         97       0.851              DoS Slowloris
----       ----
 30       4970       total

Confusion matrix:
                                prediction
                      | BENIGN (neg.) |  ATTACK (pos.)
      ----------------|---------------|---------------
      BENIGN (neg.)   | TN =     4763 | FP =        237
label ----------------|---------------|---------------
      ATTACK (pos.)   | FN =       30 | TP =       4970
      ----------------|---------------|---------------
Metrics:
R = 0.994  P = 0.954  F1 score = 0.974  FPR = 0.047
```

((b)) Matrice di confusione e risultati

((c)) Errore di ricostruzione

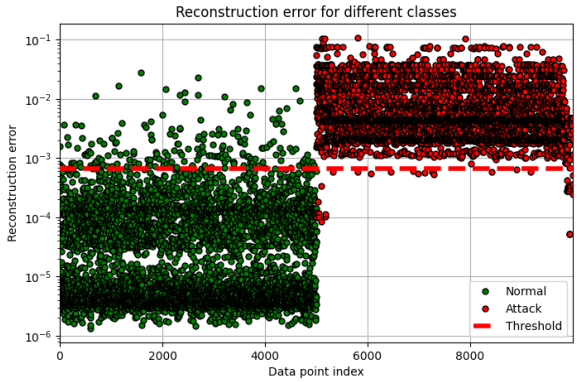
Figura 11: Risultati sulla macchina 1: Prova 3

[3]L'addestramento ha prodotto gli stessi risultati con la funzione di loss *log_cosh*, per cui le due configurazioni risultano identiche.

Come step successivo sono state calcolate le metriche sul test set (figura 12(a)). Si riporta anche il grafico che mostra l'errore di ricostruzione sul test set(figura 12(b)).

```
          *** EVALUATION RESULTS ***

      Coverage by attack (positive classes)
   FN         TP     recall
    9        195      0.956              DoS GoldenEye
   13       4613      0.997                   DoS Hulk
    0         56      1.000           DoS Slowhttptest
   26         88      0.772              DoS Slowloris
  ----       ----
   48       4952      total

Confusion matrix:
                                prediction
                     | BENIGN (neg.) |  ATTACK (pos.)
      --------------|---------------|---------------
      BENIGN (neg.) | TN =     4787 | FP =       213
label --------------|---------------|---------------
      ATTACK (pos.) | FN =       48 | TP =      4952
      --------------|---------------|---------------
Metrics:
R = 0.990  P = 0.959  F1 score = 0.974  FPR = 0.043
```

((a)) Matrice di confusione e risultati



((b)) Errore di ricostruzione

Figura 12: Risultati ottenuti sul test set con la configurazione ottimale

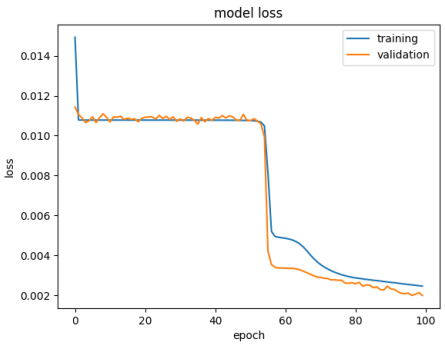
È possibile osservare che in fase di test i valori sono più alti e rientrano nei valori soglia fissati come obiettivo.

6.2.2 Macchina 2

Come configurazione iniziale è stata scelta la rete con i seguenti layers:

- input layer: 81 attributi
- hidden layer: 50 neuroni con funzione di attivazione *relu*
- hidden layer: 25 neuroni con funzione di attivazione *relu*
- hidden layer: 5 neuroni con funzione di attivazione *relu*
- hidden layer: 25 neuroni con funzione di attivazione *relu*
- hidden layer: 50 neuroni con funzione di attivazione *relu*
- output layer: 81 neuroni con funzione di attivazione *relu*

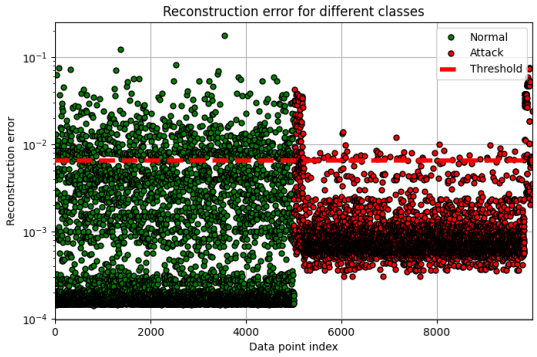
Il training è stato effettuato con un numero di epoche pari a 100 e batch size pari a 1024. Come ottimizzatore è stato scelto **rmsprop** e l'errore è stato valutato con la metrica **mean_squared_error**. La figura 13 riporta i risultati ottenuti.



((a)) Loss

```
        *** EVALUATION RESULTS ***

     Coverage by attack (positive classes)
  FN          TP      recall
 132          72       0.353              DoS GoldenEye
4567          59       0.013                   DoS Hulk
   9          47       0.839           DoS Slowhttptest
  44          70       0.614              DoS Slowloris
----        ----
4752         248       total

Confusion matrix:
                                prediction
                    | BENIGN (neg.) |  ATTACK (pos.)
              --------------|---------------|---------------
       BENIGN (neg.) | TN =     4308 | FP =        692
label         --------------|---------------|---------------
       ATTACK (pos.) | FN =     4752 | TP =        248
              --------------|---------------|---------------
Metrics:
R = 0.050  P = 0.264  F1 score = 0.084  FPR = 0.138
```

((b)) Matrice di confusione e risultati



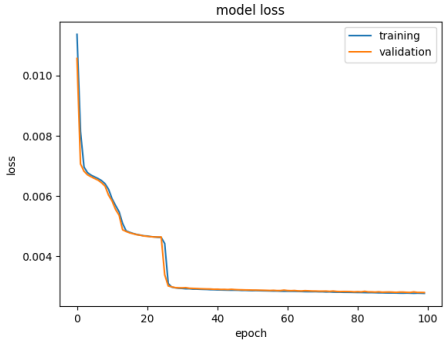
((c)) Errore di ricostruzione

Figura 13: Risultati sulla macchina 2: Prova 1

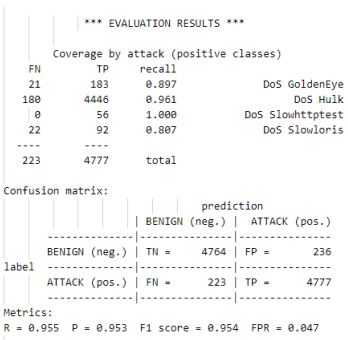
Successivamente sono state effettuate delle prove cambiando numero di layer, numero di neuroni, l'ottimizzatore utilizzando: *Nadam* e la funzione di loss con: *huber*. La rete è stata addestrata con 100 epoche e batch size pari a 1024.

- input layer: 81 attributi
- hidden layer: 64 neuroni con funzione di attivazione *relu*
- hidden layer: 32 neuroni con funzione di attivazione *relu*
- hidden layer: 16 neuroni con funzione di attivazione *relu*
- hidden layer: 8 neuroni con funzione di attivazione *relu*
- hidden layer: 16 neuroni con funzione di attivazione *relu*
- hidden layer: 32 neuroni con funzione di attivazione *relu*
- hidden layer: 64 neuroni con funzione di attivazione *relu*
- output layer: 81 neuroni con funzione di attivazione *relu*

La figura 14 riporta l'andamento della loss(14(a)), i risultati ottenuti in fase di validazione del modello (14(b)) e l'errore di ricostruzione (14(c)).



((a)) Loss



*** EVALUATION RESULTS ***

Coverage by attack (positive classes)

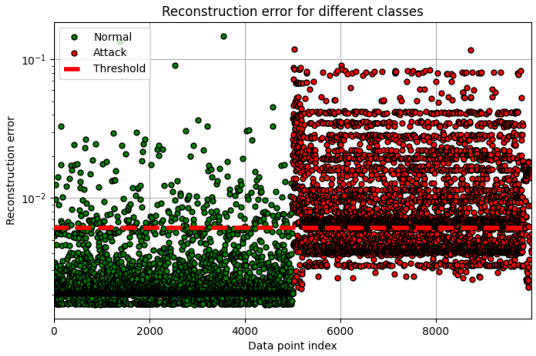
FN	TP	recall	
21	183	0.897	DoS GoldenEye
180	4446	0.961	DoS Hulk
0	56	1.000	DoS Slowhttptest
22	92	0.807	DoS Slowloris
223	4777	total	

Confusion matrix:

label \ prediction	BENIGN (neg.)	ATTACK (pos.)
BENIGN (neg.)	TN = 4764	FP = 236
ATTACK (pos.)	FN = 223	TP = 4777

Metrics:
R = 0.955 P = 0.953 F1 score = 0.954 FPR = 0.047

((b)) Matrice di confusione e risultati



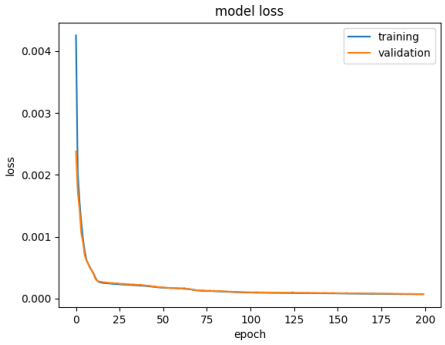
((c)) Errore di ricostruzione

Figura 14: Risultati sulla macchina 2: Prova 2

La configurazione che ha prodotto i risulati migliori è:

- input layer: 81 attributi
- hidden layer: 40 neuroni con funzione di attivazione *selu*
- hidden layer: 20 neuroni con funzione di attivazione *selu*
- hidden layer: 10 neuroni con funzione di attivazione *tanh*
- hidden layer: 20 neuroni con funzione di attivazione *selu*
- hidden layer: 40 neuroni con funzione di attivazione *selu*
- output layer: 81 neuroni con funzione di attivazione *elu*

Il numero di epoche è stato aumentato a 200; i restanti parametri sono rimasti invariati dall'osservazione precedente.



((a)) Loss

```
        *** EVALUATION RESULTS ***

    Coverage by attack (positive classes)
   FN          TP     recall
   11         193      0.956              DoS GoldenEye
   17        4609      0.996                   DoS Hulk
    0          56      1.000           DoS Slowhttptest
   18          96      0.842              DoS Slowloris
  ----        ----
   46        4954      total

Confusion matrix:
                                prediction
                    | BENIGN (neg.) |  ATTACK (pos.)
       -------------|---------------|---------------
       BENIGN (neg.) | TN =     4774 | FP =        226
label  -------------|---------------|---------------
       ATTACK (pos.) | FN =       46 | TP =       4954
       -------------|---------------|---------------
Metrics:
R = 0.991  P = 0.956  F1 score = 0.973  FPR = 0.045
```

((b)) Matrice di confusione e risultati

((c)) Errore di ricostruzione

Figura 15: Risultati sulla macchina 2: Prova 3

7 Configurazioni hardware delle macchine utilizzate

In questa sezione si riportano le caratteristiche delle macchine utilizzate per la configurazione delle reti neurali.

- **Macchina 1**:
  Acer Aspire A515-56G
  Windows 11 Home
  11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz 2.80 GHz
  Python 3.11.0
  Keras 2.12.0
  TensorFlow 2.12.0

- **Macchina 2**: Acer Nitro 5 AN515-54
  Windows 11 Home
  Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz 2.59 GHz
  Python 3.11.0
  Keras 2.12.0
  TensorFlow 2.12.0

8 Risultati ottenuti: Feedforward neural network

La tabella 1 riporta le configurazioni ottimali ottenute sulle macchine utilizzate.

Machine	Layers	Epochs	Batch size	Optimizer	Loss
Acer Aspire 5	81/16/8/2/2 -/relu/relu/relu/softmax	200	1024	rmsprop	categorical crossentropy
Acer Nitro 5	81/80/16/2/2 -/tanh/tanh/relu/softmax	200	1024	Nadam	categorical crossentropy

Tabella 1: Configurazioni ottimali della FNN

La tabella 2 riporta i risultati ottenuti in fase di validazione e test su entrambe le macchine, utilizzando la configurazione ottimale ricavata dalla macchina 1.

Platform	Score	
	Validation	Test
Macchina 1: Acer Aspire 5	Golden Eye: 1 Hulk: 1 Slowhttptest: 1 Slowloris: 1 FPR: 0.001	Golden Eye: 0.961 Hulk: 1 Slowhttptest: 1 Slowloris: 0.974 FPR: 0.001
Macchina 2: Acer Nitro 5	Golden Eye: 1 Hulk: 1 Slowhttptest: 1 Slowloris: 0.982 FPR: 0.098	Golden Eye: 1 Hulk: 1 Slowhttptest: 1 Slowloris: 1 FPR: 0.114

Tabella 2: Risultati con configurazione ottimale ricavata dalla macchina 1

La tabella 3 riporta i risultati ottenuti in fase di validazione e test su entrambe le macchine, utilizzando la configurazione ottimale ricavata dalla macchina 2.

Platform	Score	
	Validation	Test
Macchina 1: Acer Aspire 5	Golden Eye: 1 Hulk: 1 Slowhttptest: 1 Slowloris: 1 FPR: 0.033	Golden Eye: 1 Hulk: 1 Slowhttptest: 1 Slowloris: 1 FPR: 0.033
Macchina 2: Acer Nitro 5	Golden Eye: 1 Hulk: 1 Slowhttptest: 1 Slowloris: 1 FPR: 0.004	Golden Eye: 1 Hulk: 1 Slowhttptest: 1 Slowloris: 0.982 FPR: 0.005

Tabella 3: Risultati con configurazione ottimale ricavata dalla macchina 2

9 Risultati ottenuti: Autoencoder

La tabella 4 riporta le configurazioni ottimali ottenute sulle macchine utilizzate.

Machine	Layers	Epochs	Batch size	Optimizer	Loss
Acer Aspire 5	81/32/16/8/16/32/81 -/selu/selu/selu/selu/selu/selu	200	1024	Nadam	huber
Acer Nitro 5	81/40/20/10/20/40/81 -/selu/selu/tanh/selu/selu/elu	100	1024	Nadam	huber

Tabella 4: Configurazioni ottimali della rete Autoencoder

La tabella 5 riporta i risultati ottenuti in fase di validazione e test su entrambe le macchine, utilizzando la configurazione ottimale ricavata dalla macchina 1.

Platform	Score	
	Validation	Test
Macchina 1: Acer Aspire 5	Golden Eye: 0.946 Hulk: 1 Slowhttptest: 1 Slowloris: 0.851 FPR: 0.047	Golden Eye: 0.956 Hulk: 0.997 Slowhttptest: 1 Slowloris: 0.772 FPR: 0.043
Macchina 2: Acer Nitro 5	Golden Eye: 0.941 Hulk: 0.996 Slowhttptest: 1 Slowloris: 0.807 FPR: 0.048	Golden Eye: 0.951 Hulk: 0.967 Slowhttptest: 0.786 Slowloris: 0.781 FPR: 0.048

Tabella 5: Risultati con configurazione ottimale ricavata dalla macchina 1

La tabella 6 riporta i risultati ottenuti in fase di validazione e test su entrambe le macchine, utilizzando la configurazione ottimale ricavata dalla macchina 2.

Platform	Score	
	Validation	Test
Macchina 1: Acer Aspire 5	Golden Eye: 0.941 Hulk: 0.986 Slowhttptest: 1 Slowloris: 0.868 FPR: 0.044	Golden Eye: 0.956 Hulk: 0.960 Slowhttptest: 1 Slowloris: 0.816 FPR: 0.0.047
Macchina 2: Acer Nitro 5	Golden Eye: 0.0946 Hulk: 0.998 Slowhttptest: 1 Slowloris: 0.857 FPR: 0.042	Golden Eye: 0.956 Hulk: 0.996 Slowhttptest: 1 Slowloris: 0.842 FPR: 0.045

Tabella 6: Risultati con configurazione ottimale ricavata dalla macchina 2