

Università Politecnica delle Marche

Dipartimento di Ingegneria dell'Informazione

Facoltà di Ingegneria Informatica e dell'Automazione

Corso di Advanced Cyber Security



Analisi e Classificazione di Domini Malevoli mediante Deep Learning

Professore:

Prof. Spalazzi Luca

Studenti:

Bernovschi Denis
Incicco Emanuele
Pinciaroli Andrea
Fratini Lorenzo
Miscia Federico

Indice

1 Introduzione	9
1.1 Tecnologie	11
2 Prima Parte	13
2.1 Dataset GARR	13
2.1.1 ETL	14
2.1.2 Divisione in 1-grams, 2-grams, 3-grams	15
2.2 FastText	16
2.2.1 Introduzione	16
2.2.2 Skipgram vs CBOW	16
2.2.3 Caso di Studio	16
3 Seconda Parte	21
3.1 Dataset UMUDGA	21
3.2 Architettura Multi-Input	21
3.3 Architettura Multi-Input con Embedding 1-gram Casuale	22
3.4 Iperparametri	22
3.5 Risultati	23
3.6 Tempi Computazionali	23
3.6.1 Tempi Addestramento FastText	23
3.6.2 Tempi Addestramento Multi-Input Architecture	23
3.6.3 Tempi Addestramento Multi-Input con Random Embedding	23
3.7 Risultati Classificazione con Multi-Input	24
3.7.1 Risultati Classificazione - Percentuale 0.017	24
3.7.2 Risultati Classificazione - Percentuale 0.034	25
3.7.3 Risultati Classificazione - Percentuale 0.068	26
3.7.4 Risultati Classificazione - Percentuale 0.135	27
3.8 Risultati Classificazione con Multi-Input e Random Embedding	28
3.8.1 Risultati Classificazione - Percentuale 0.017	28
3.8.2 Risultati Classificazione - Percentuale 0.034	29
3.8.3 Risultati Classificazione - Percentuale 0.068	30
3.8.4 Risultati Classificazione - Percentuale 0.135	31
3.9 Confronto Risultati Architetture	32
3.9.1 Risultati Classificazione	32
4 Conclusioni e Sviluppi Futuri	35

A Risultati Multi Input	39
A.1 Risultati Multi Input 0.017	39
A.2 Risultati Multi Input 0.034	40
A.3 Risultati Multi Input 0.068	41
A.4 Risultati Multi Input 0.135	43
B Risultati Multi Input with Random Embedding	45
B.1 Multi Input with Random Embedding 0.017	45
B.2 Multi Input with Random Embedding 0.034	46
B.3 Multi Input with Random Embedding 0.068	47
B.4 Multi Input with Random Embedding 0.135	49
C Ulteriori Informazioni	51

Elenco delle figure

1.1	Funzionamento di un DGA	10
1.2	Linguaggio di Programmazione Python	11
1.3	Google Colab	11
2.1	Skipgram vs CBOW	17
3.1	Architettura Multi-Input	22
3.2	Architettura Multi-Input con Embedding Random	22
3.3	CM 0.017 Multi Input	25
3.4	CM 0.034 Multi Input	26
3.5	CM 0.068 Multi Input	27
3.6	CM 0.135 Multi Input	28
3.7	CM 0.017 Multi Input con Random Embeddings	29
3.8	CM 0.034 Multi Input con Random Embeddings	30
3.9	CM 0.068 Multi Input con Random Embeddings	31
3.10	CM 0.135 Multi Input con Random Embeddings	32

Listings

2.1	Script per ETL	14
2.2	Script per la creazione dei n-gramss	15
2.3	Implementazione	16
2.4	Implementazione FastText	17
2.5	Training del Modello	18
2.6	Conversione da <i>Bin</i> a <i>Vec</i>	18

Capitolo 1

Introduzione

Oggi giorno, gli attacchi informatici sono sempre più frequenti e con conseguenze sempre più importanti.

Volendo fare un'estrema sintesi, gli attacchi quali phishing, trojan e worm non fanno altro che tentare di recuperare informazioni sensibili, il cui utilizzo può essere svariato: da una semplice rivendita nel dark web (i dati sanitari sono i più richiesti e pagati) fino alla predisposizione di ulteriori azioni malevoli, come ad esempio attacchi DoS (Denial of Service). Attacchi del genere, tipicamente, utilizzano come mezzo di trasmissione la rete. Ne consegue che sempre più dispositivi utilizzati sia in ambito lavorativo che per scopi personali sono esposti al rischio di compromissione.

In questo elaborato l'attenzione è posta, nello specifico, sui malware che permettono di creare le *Botnet*, cioè reti di macchine infette (bot) che poi devono essere coordinate dall'hacker tramite un server di comando e controllo. Per contrastare un'infezione di questo tipo, spesso, si tenta di individuare ed offuscare il server di comando e controllo; ciò, però, non è così facile perché in risposta vengono adottate delle tecniche evasive che ne rendono difficile l'individuazione. Una di queste tecniche consiste nel cambiare il nome di dominio dopo un certo intervallo di tempo, attraverso un DGA (Domain Name Generation Algorithm).

Il cambio continuo di *domain name*, ad ogni modo, causa problemi anche al malware stesso che ha infettato una macchina: quando esso si attiva e cerca di contattare il server di comando e controllo, infatti, non sa esattamente a quale nome di dominio debba rivolgersi. A tal proposito, il malware, utilizzando anch'esso il medesimo DGA, genera tutta una serie di nomi di dominio e di volta in volta interroga il DNS¹ per vedere se c'è qualcuno registrato sotto il nome generato. Quando viene trovato un riscontro positivo, il DNS traduce il nome di dominio in un indirizzo IP.

¹DNS ≡ Domain Name Server

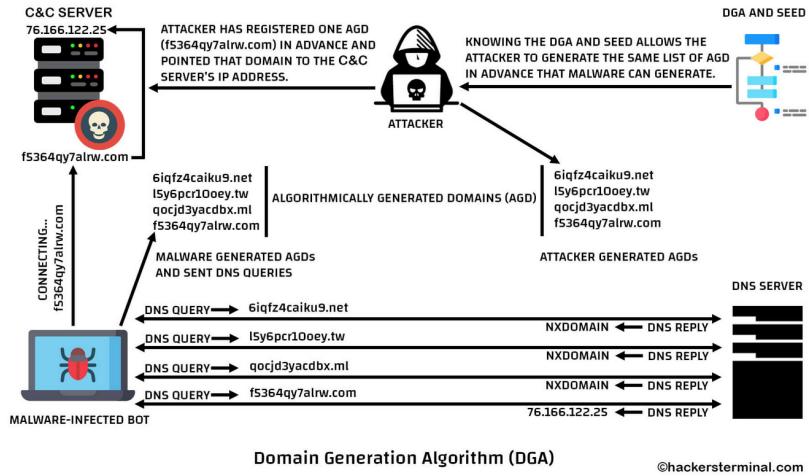


Figura 1.1: Funzionamento di un DGA

Per descrivere come funziona il sistema DNS di Internet, basti pensare ad una rubrica telefonica che gestisce la mappatura tra nomi e numeri. I server DNS traducono le richieste di nomi di dominio in indirizzi IP, controllando a quale server si connette un utente finale nel momento in cui egli digiterà un certo nome di dominio nel proprio browser Web. A questo punto, dunque, appare chiaro che se il dominio digitato è in realtà malevolo si è vittima di un attacco. Andare a scovare questi domini, tentando di rintracciarne la forma o le caratteristiche peculiari, dunque, diventa fondamentale nel contrastare questa tipologia di attacco.

Il quadro è ulteriormente complicato dal fatto che un DGA può generare diverse centinaia di domini, rendendo di fatto l'analisi un'operazione onerosa e assai complessa per l'uomo, dato che bisogna verificare la legittimità di ogni nome di dominio sulla base delle richieste che lo interessano.

I DGA sviluppati odiernamente, inoltre, sfruttano un meccanismo che combina lettere o parole in modo pseudo-casuale, fino ad arrivare alla creazione di un dizionario, in modo da complicare ulteriormente il lavoro di analisi: sulla base delle parole contenute all'interno del dizionario, infatti, vengono creati nomi di dominio dall'aspetto molto simile a quelli reali e talvolta facilmente scambiabili per legittimi. Da ciò si intuisce come l'evoluzione dei sistemi di detection da una parte e il progresso continuo di questi algoritmi, dall'altra, siano in una rincorsa reciproca e continua.

L'elaborato qui presente, dunque, ha l'obiettivo di testare una tipologia di architettura basata su Deep Learning che permetta di classificare i domini malevoli e non.

1.1 Tecnologie

Per lo sviluppo del progetto e l'analisi degli approcci di apprendimento automatico utili alla questione, è stato utilizzato il tool Google Colab, il quale mette a disposizione un ambiente di sviluppo condiviso per il linguaggio Python con all'interno disponibili già diverse librerie di interesse. Altri aspetti a favore di Google Colab sono legati alle computational resources e alla possibilità di storage sharing, semplificando il lavoro del team.



Figura 1.2: Linguaggio di Programmazione Python



Figura 1.3: Google Colab

Capitolo 2

Prima Parte

2.1 Dataset GARR

Per quanto concerne i datasets utilizzati, nella prima parte del lavoro si è fatto uso dei file di log forniti dalla rete GARR. Dopo un'attenta analisi e una fase di ETL¹, si è attuata anche una pulizia dei duplicati e dei nomi di dominio non conformi alle specifiche assegnate. Il dataset ottenuto in questa sede, ad ogni modo, non risulta etichettato, ossia non distingue tra nomi di dominio benevoli o malevoli.

Selezione Log Per la selezione e la creazione del dataset, sono stati selezionati i logs relativi a sei giorni distinti di ciascuno degli ultimi 12 mesi disponibili. Successivamente, è stato considerato solo il 10% dei records presenti all'interno di ogni file di log scaricato.

Precisazioni

1. Per quanto riguarda le ore selezionate, esse rappresentano l'intero intervallo giornaliero, ovvero dalle 00 : 00 alle 23 : 00;
2. Per la scelta dei giorni, si è proceduto in modo ciclico, selezionandoli tutti dal lunedì alla domenica, estremi inclusi;
3. Relativamente alle mensilità, ripercorrendo a ritroso, l'intervallo considerato va da Novembre 2021 fino a Febbraio 2021, dopodiché da Settembre 2020 ad Agosto 2020. La ragione di tale scelta è un'assenza di dati nell'arco temporale che va da Febbraio 2021 a Settembre 2020 all'interno del repository del GARR.
4. Di seguito un resoconto del numero di logs selezionati per ciascun giorno della settimana:

Giorno	Lunedì	Martedì	Mercoledì	Giovedì	Venerdì	Sabato	Domenica
Nº Logs	9	10	10	10	12	12	9

Tabella 2.1: N Logs considerati per ciascuna giorno della settimana

¹ETL ≡ Extraction, Transform e Loading

2.1.1 ETL

Nella seguente sezione è possibile visionare lo script utilizzato per la fase di ETL. Come già anticipato, oltre alla fase di selezione dei dati di log da considerare, è stata realizzata anche la fase di pulizia di questi ultimi. Come riportato nel listato seguente, sono state effettuate diverse tipologie di operazioni di pulizia e di verifica della correttezza dei dati a disposizione. In estrema sintesi, è possibile individuare:

- Una clausola per la lunghezza massima del nome di dominio $LEN_THRESHOLD = 100$;
- Una clausola per la lunghezza minima del nome di dominio: $len(DN_name) < 3$. Sono stati esclusi, cioè, i nomi di dominio più corti di 3 caratteri, così facendo elidiamo i possibili problemi nel implementazione della divisione in n-grams;
- Implementazione di una REGEX, in modo da considerare solo i nomi di dominio che non contengono all'interno indirizzi IP;
- Infine, la selezione del 10% dei records, come da specifiche richieste.

```

1 import os
2 import csv
3 import re
4 from random import sample
5 import math
6
7 # PARAMS
8 PATH = os.path.join(path_progettoADV, '_log_files')
9 OUT_FILENAME = 'out_temp_file.csv'
10 LEN_THRESHOLD = 100
11 # get the list of files
12 files_list = os.listdir(PATH)
13 # open output file
14 out_file = open(os.path.join(path_progettoADV, OUT_FILENAME), 'w', newline=',')
15 writer = csv.writer(out_file)
16 # domain names list
17 DNS_list = []
18 DNS_list_samples = []
19
20 # for each file
21 for filename in files_list:
22     # skip not '.log' files
23     if '.log' not in filename:
24         continue
25
26     # open file
27     file = open(os.path.join(PATH, filename), 'r')
28
29     # for each line
30     for i, line in enumerate(file):
31         # get domain name
32         DN_name = line.split(';')[5]
33
34         # clear final dot in the domain name
35         if DN_name[-1] == '.':
36             DN_name = DN_name[0:-1]
37         # skip domain names with length greater or lower than threshold
38         if len(DN_name) > LEN_THRESHOLD or len(DN_name) < 3:
39             continue
40         # skip domain names with addresses

```

```

41     if re.match(r'.*\d*\.\d*\.\d*', DN_name):
42         continue
43     else:
44         DNs_list.append(DN_name)
45
46     # random choice 10% of DNs
47     DNs_list_sample = sample(DNs_list, math.floor(len(DNs_list)/100))
48     DNs_list_samples += DNs_list_sample
49
50 # get unique domain names
51 set_of_DNs = set(DNs_list_samples)
52 print(f'Estratti {len(set_of_DNs)} nomi di dominio.')
53 # write domain names and domain names without dots
54 writer.writerows([value, f'{value}'.replace('.', '')] for value in set_of_DNs)

```

Listing 2.1: Script per ETL

A seguito delle operazioni di ETL, sono risultati 4548258 nomi di dominio a partire da una cifra iniziale di circa 28 milioni. I record che non hanno rispettato i vincoli sulla lunghezza (maggiore di 100 o minore di 3) sono stati 816473. I record scartati tramite l'utilizzo di espressioni regolari sono stati 9359778. La quota rimanente è stata filtrata grazie alla rimozione dei duplicati.

2.1.2 Divisione in 1-grams, 2-grams, 3-grams

Una volta terminata la fase di ETL, si è passati ad implementare uno script per la creazione di 1-grams, 2-grams e 3-grams, cioè la suddivisione di ciascun nome di dominio in sotto-unità costituite rispettivamente da una, due e tre lettere. Nel listato seguente è visibile tale implementazione.

La peculiarità all'interno di questo script è l'utilizzo della libreria **nltk**: essa infatti mette a disposizione la funzione *ngrams*, la quale permette di estrapolare gli n-grams in maniera agevole, specificando direttamente il parametro *n* desiderato.

```

1 import csv
2 from nltk import ngrams
3
4
5 # PARAMS
6 IN_FILENAME = 'out_temp_file.csv'
7 OUT_FILENAME = 'out_final_file.csv'
8
9 # open input and output files
10 with open(os.path.join(path_progettoADV, IN_FILENAME), 'r') as in_file, open(os.
    path.join(path_progettoADV, OUT_FILENAME), 'w', newline='') as out_file:
11     reader = csv.reader(in_file)
12     writer = csv.writer(out_file)
13     for row in reader:
14         # get domain name without dots
15         domain_name = row[1]
16
17         # extract 1-grams from the domain name
18         n = 1
19         one_grams = ngrams(domain_name, n)
20         one_gram = [','.join(el) for el in one_grams]
21
22         # extract 2-grams from the domain name
23         n = 2
24         two_grams = ngrams(domain_name, n)
25         two_gram = [','.join(el) for el in two_grams]
26

```

```

27     # extract 3-grams from the domain name
28     n = 3
29     three_grams = ngrams(domain_name, n)
30     three_gram = [','.join(el) for el in three_grams]
31
32     # write extracted info to the output file
33     writer.writerow([row[0], row[1], ', '.join(one_gram), ', '.join(two_gram),
34                      ', '.join(three_gram)])

```

Listing 2.2: Script per la creazione dei n-gramss

2.2 FastText

2.2.1 Introduzione

FastText è una libreria open-source, sviluppata dal laboratorio AI Research di Facebook (FAIR), che consente agli utenti di apprendere a partire da rappresentazioni di testo. FastText, più in dettaglio, permette l'apprendimento delle incorporazioni di parole e la classificazione dei testi. Il modello offerto da FastText fa affidamento su un algoritmo di apprendimento non supervisionato o supervisionato al fine di ottenere rappresentazioni vettoriali delle parole. Nel moderno Machine Learning e nel Natural Language processing, infatti, ha trovato riscontro positivo l'idea di rappresentare le parole tramite vettori. Tali vettori catturano informazioni nascoste di un linguaggio, come analogie sintattiche o analogie semantiche. FastText *viene anche utilizzato per migliorare le prestazioni dei classificatori di testo*[1].

2.2.2 Skipgram vs CBOW

FastText fornisce due modelli per il calcolo delle rappresentazioni di parole: *skipgram* e *cbow* ("continuous-bag-of-words").

Il modello *skipgram* impara a prevedere una parola target grazie ad una parola vicina. D'altra parte, il modello *cbow* predice la parola d'interesse in base al suo contesto. Il contesto è rappresentato tramite un insieme di parole contenute in una finestra di dimensione fissa intorno alla parola target. È possibile mostrare questa differenza con un esempio: data la frase "*I am selling these fine leather jackets*" e la parola target "*fine*", un modello *skipgram* cerca di predirla usando una parola casuale vicina, come "*selling*" o "*these*". Il modello *cbow* invece prende tutte le parole in una finestra circostante, come {*selling*, *these*, *leather*, *jackets*}, e utilizza la somma dei loro vettori per prevedere la parola target. La figura seguente riassume questa differenza[2].

Implementazione

Una possibile implementazione di FastText potrebbe essere la seguente. Come si può notare, oltre a passare i dati su cui addestrarsi, è possibile settare il parametro relativo a CBOW o SkipGram.

```

1 import fasttext
2 model = fasttext.train_unsupervised("data/fil9", "cbow")

```

Listing 2.3: Implementazione

2.2.3 Caso di Studio

Nel lavoro svolto, l'utilizzo della FastText ha permesso di attuare un pre-addestramento da sfruttare successivamente negli strati di embeddings dell'architettura che verrà descritta nella

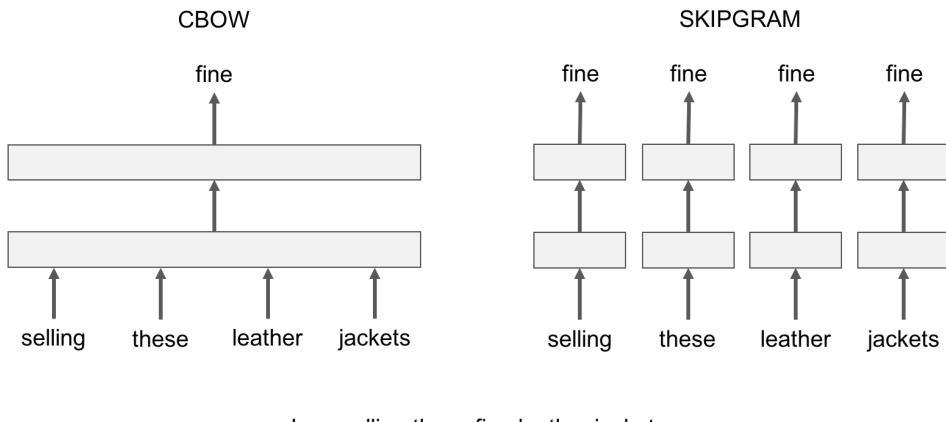


Figura 2.1: Skipgram vs CBOW

seconda parte della relazione. L’addestramento della FastText, inoltre, è stato eseguito per ogni tipologia di dato a disposizione, ovvero per *1-grams*, *2-grams* e *3-grams*. Nello specifico, si è realizzato un addestramento non supervisionato, utilizzando la modalità *skipgram* messa a disposizione da FastText. Il risultato sono dei vettori pesati per ciascun elemento di *1-grams*, *2-grams* e *3-grams*, da caricare poi come pesi nella seconda architettura addestrata nella parte successiva del progetto.

Implementazione

Nella seguente sezione si discute l’implementazione della FastText per il progetto in esame. Per una maggiore comprensione e chiarezza, di seguito è riportata la road-map seguita in questa parte del lavoro.

1. Implementazione dell’algoritmo per effettuare il training, con la lettura dei file *n-grams* e la dichiarazione del modello utilizzato;
2. Training del modello;
3. Trasformazione del file con i pesi della FastText da un formato *.bin* ad uno *.vec*.

Nella parte sottostante sono descritti nel dettaglio tutti gli steps. Lo script relativo al primo passaggio è riportato nel seguente listato.

```

1 #installazione pacchetto
2 !pip install fasttext -q
3
4 import fasttext
5 import pandas as pd
6 import os
7
8 #lettura file
9 df = pd.read_csv('out_final_file.csv', header=None)
10
11 #split per n-gram
12 _label_ = [',', '1-gram', '2-gram', '3-gram']
13 j = 1;
14 for i in range (2, 5):
15     columns_ = df[i]
16     print(columns_.values)

```

```

17 print('len di {} : {}'.format(_label_[j], len(columns_.values)))
18 with open('out_final_file_{}.txt'.format(j), 'w') as f:
19     for line in columns_.values:
20         f.write(str(line).strip('\n'))
21         f.write('\n')
22     j = j + 1

```

Listing 2.4: Implementazione FastText

Come si nota nel listato precedente, il file da leggere è **out_final_file.csv** (output dei tasks precedenti, relativi alla fase di ETL); di tale csv, vengono considerate solo le colonne dalla seconda alla quarta, ovvero quelle contenenti gli *n-grams* (rispettivamente 1-gram nella colonna 2, 2-gram nella colonna 3, 3-gram nella colonna 4).

Come definito nella road map, successivamente si passa all’allenamento dei modelli (tre modelli, rispettivamente per 1-grams, 2-grams e 3-grams). Per fare ciò, è sufficiente una semplice invocazione del metodo *train_unsupervised* dell’oggetto *fasttext*, passando come parametro in primo luogo il file creato precedentemente per la tipologia di n-grams in questione. I restanti parametri sono relativi alla scelta tra il modello *skipgram* o *cbow* ed infine la dimensione di *embedding* che, da specifiche progettuali, è impostata a 128. Di seguito è riportato il listato del training del modello con input 1-gram.

```

1 model_1 = fasttext.train_unsupervised(
2     'out_final_file_1.txt',
3     "skipgram",
4     dim=128)
5 model_1.words
6 model_1.save_model("out_final_file_1.bin")

```

Listing 2.5: Training del Modello

Terminata la fase di training dei tre modelli, i file ottenuti in output, contenenti i pesi generati come risultato dell’addestramento, sono stati convertiti dal formato *.bin* a *.vec*. Per fare ciò, è stato utilizzato lo script fornito da FastText². Per una maggior fruizione dell’elaborato, tale script è riportato nel listato successivo.

```

1 #!/usr/bin/env python
2
3 # Copyright (c) 2017-present, Facebook, Inc.
4 # All rights reserved.
5 # This source code is licensed under the MIT license found in the
6 # LICENSE file in the root directory of this source tree.
7
8 from __future__ import absolute_import
9 from __future__ import division
10 from __future__ import print_function
11 from __future__ import unicode_literals
12 from __future__ import division, absolute_import, print_function
13
14 from fasttext import load_model
15 import argparse
16 import errno
17
18 if __name__ == "__main__":
19     parser = argparse.ArgumentParser(
20         description=("Print fasttext .vec file to stdout from .bin file"))
21     )
22     parser.add_argument(
23         "model",

```

²URL https://github.com/facebookresearch/fastText/blob/main/python/doc/examples/bin_to_vec.py

```

24         help="Model to use",
25     )
26 args = parser.parse_args()
27
28 f = load_model(args.model)
29 words = f.get_words()
30 print(str(len(words)) + " " + str(f.get_dimension()))
31 for w in words:
32     v = f.get_word_vector(w)
33     vstr = ""
34     for vi in v:
35         vstr += " " + str(vi)
36     try:
37         print(w + vstr)
38     except IOError as e:
39         if e.errno == errno.EPIPE:
40             pass

```

Listing 2.6: Conversione da *Bin* a *Vec*

Lo script appena presentato permette di realizzare la conversione di formato desiderata per mezzo del comando:

```
1 python bin_to_vec.py nome_modello_fastText.bin > nome_file_vec.vec
```

Si è deciso di convertire il file ".bin" in un file ".vec" in quanto nel repository che ospita l'architettura utilizzata nella seconda fase del progetto si fa uso di un file con estensione ".vec". In questo modo, dunque, si evitano modifiche ulteriori al codice presente nel repository.

Capitolo 3

Seconda Parte

Nella seconda parte dell'elaborato vengono descritti i passi per la realizzazione e l'addestramento dell'architettura di classificazione assegnata, ovvero la Multi-Input. L'addestramento di tale architettura avverrà sfruttando i pesi ottenuti dal precedente addestramento della FastText (rispettivamente su 1-grams, 2-grams e 3-grams) i quali andranno ad alimentare lo strato di embedding da agganciare alla LSTM¹. L'LSTM, infine, deve essere a sua volta addestrata come classificatore: in ingresso, infatti, riceverà un dataset etichettato.

3.1 Dataset UMUDGA

Per la seconda parte del lavoro si è utilizzato il dataset UMUDGA [3], differente rispetto a quello creato per mezzo dei file di log della rete GAR.R.

Tale dataset è composto da 50 classi di DGA, oltre alla classe dei nomi di dominio legittimi; in totale, quindi, si hanno 51 classi.

Il dataset, inoltre, risulta bilanciato in quanto presenta lo stesso numero di nomi di dominio appartenenti a ciascuna delle 51 classi.

Al fine di utilizzare il dataset UMUDGA, l'addestramento dell'architettura è avvenuto mediante il file "*umudga_1k.csv*" presente nel repository e considerando quattro differenti percentuali di training.

Il file "*umudga_1k.csv*" è formato dai seguenti campi: label binaria (dga o legit), label multiclass (relativa ai singoli DGA), nome di dominio senza punti, nome di dominio originario, 1-grams, 2-grams, 3-grams e words ottenuti dal nome di dominio senza punti.

3.2 Architettura Multi-Input

Come architettura da addestrare è stata utilizzata la Multi-Input. Essa è caratterizzata dal prendere in input tipologie differenti di dato.

Come si può osservare dalla figura 3.1, tale architettura è caratterizzata dall'avere 4 rami di input, il primo prende sequenze di words ed è caratterizzato da uno strato di embedding basato su ELMo, gli altri 3 prendono in input sequenze di 1-grams, 2-grams e 3-grams e sono caratterizzati da strati di embeddings basati su FastText.

¹LSTM = Long short-term memory, si tratta di una rete neurale

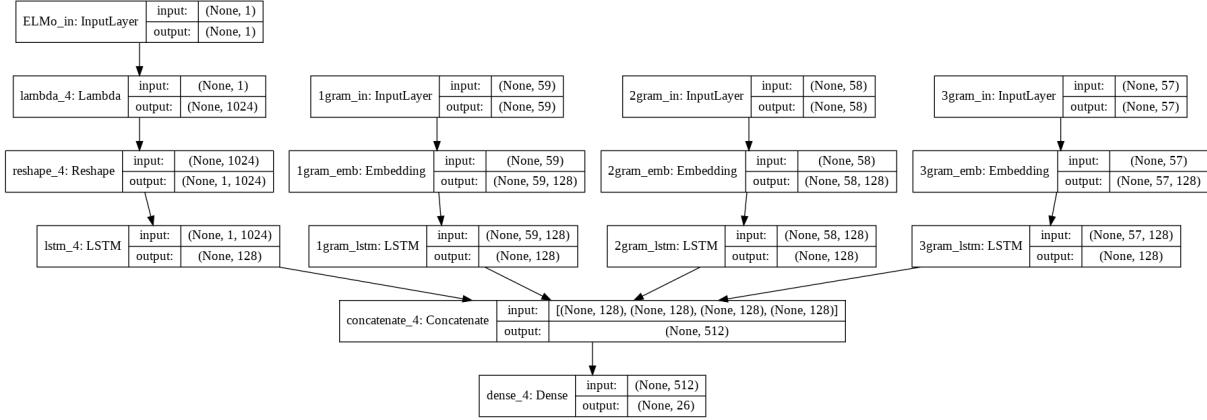


Figura 3.1: Architettura Multi-Input

3.3 Architettura Multi-Input con Embedding 1-gram Casuale

Come test aggiuntivo, si è deciso di addestrare una configurazione aggiuntiva dell'architettura Multi-Input. Questa ulteriore architettura, a differenza della precedente, possiede un ramo in più caratterizzato dal prendere in input sequenze di 1-grams con uno strato di embedding randomico. Avendo addestrato ambo le tipologie di Multi-Input, è stato possibile eseguire un confronto dei risultati per le diverse percentuali di training set utilizzate.

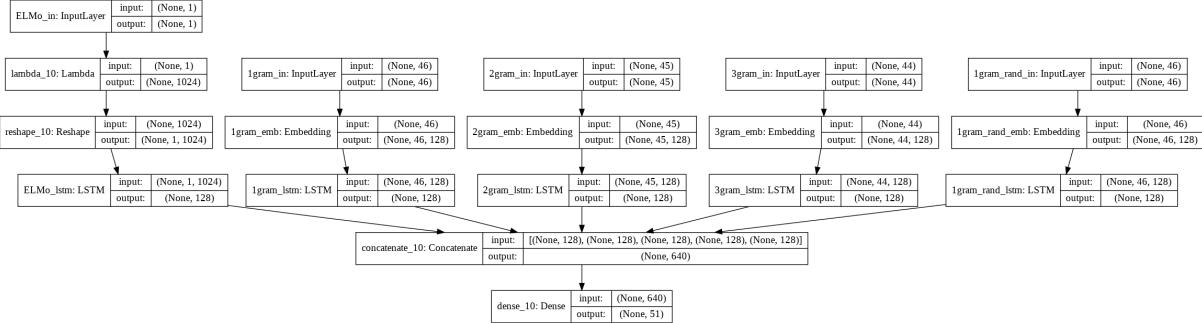


Figura 3.2: Architettura Multi-Input con Embedding Random

Nella figura 3.2 è visibile l'architettura del secondo modello addestrato in questo lavoro.

3.4 Iperparametri

Negli esperimenti effettuati sono stati utilizzati diversi iper-parametri, alcuni fissati secondo le specifiche del progetto, altri lasciando i valori di default.

I parametri utilizzati sono riportati di seguito.

- **Numero di epoch:** settato al valore di default, ovvero 20;
- **Batch size:** stato settato al valore di default, ovvero 128;
- **Dimensioni fastText-based embeddings:** settato secondo specifiche progettuali a 128.

3.5 Risultati

Nelle sezioni successive sono riportati i risultati, sia in termini computazionali che in termini di metriche ordinarie, quali ad esempio accuracy, precision, etc. per ambo le architetture descritte nelle sezioni (Sez. 3.2, Sez.3.3), divisi in base alla percentuale di training set considerato.

3.6 Tempi Computazionali

In questa sezione sono riportati i tempi computazionali impiegati per gli addestramenti della FastText e dell'architettura Multi-Input.

Gli addestramenti della FastText e dell'architettura Multi-Input sono stati eseguiti su Colab. In particolare, si è sfruttata un'istanza Pro di Colab, il che ha permesso di avere a disposizione GPU più performanti e, di conseguenza, ridurre i tempi di addestramento rispetto ad un'istanza gratuita.

3.6.1 Tempi Addestramento FastText

In questa sezione si riportano i tempi di addestramento impiegati dalla FastText per eseguire l'addestramento non supervisionato dei dati estratti dal GARR. I tempi sono divisi in base alla tipologia di suddivisione con cui si è addestrata la FastText.

Tabella 3.1: Tempi Addestramento fastText

Tipologia Dato	Durata
1-grams	~10 secondi
2-grams	~5 minuti
3-grams	~20 minuti

3.6.2 Tempi Addestramento Multi-Input Architecture

In questa sezione si riportano i tempi di addestramento impiegati dall'architettura Multi-Input per eseguire l'addestramento supervisionato utilizzando il dataset UMUDGA. I tempi vengono divisi in base alla percentuale di training utilizzata.

Tabella 3.2: Tempi Addestramento Multi-Input UMUDGA

Percentuale Train	Durata
0.017	~22 minuti
0.034	~26 minuti
0.068	~33 minuti
0.135	~50 minuti

3.6.3 Tempi Addestramento Multi-Input con Random Embedding

In questa sezione si riportano i tempi di addestramento impiegati dall'architettura **Multi-Input con Random Embedding** per eseguire l'addestramento supervisionato utilizzando il dataset UMUDGA. I tempi vengono divisi in base alla percentuale di training utilizzata.

Tabella 3.3: Tempi Addestramento Multi-Input con Random Embedding - UMUDGA

Percentuale Train	Durata
0.017	~26 minuti
0.034	~29 minuti
0.068	~35 minuti
0.135	~49 minuti

3.7 Risultati Classificazione con Multi-Input

In questa sezione vengono riportati i risultati della classificazione ottenuti tramite l'architettura descritta nella sezione (Sez.3.2), suddividendoli per ciascun intervallo di dataset considerato.

3.7.1 Risultati Classificazione - Percentuale 0.017

Tabella 3.4: Risultati Metriche di Classificazione - 0.017

Metrica	Risultato
Accuracy	0.594858
F1-Score	0.578561
Precision	0.609030
Recall	0.594858

Matrice di Confusione Nel seguente paragrafo è riportata la matrice di confusione, così da avere un overview migliore sui risultati della classificazione (Fig.3.3)

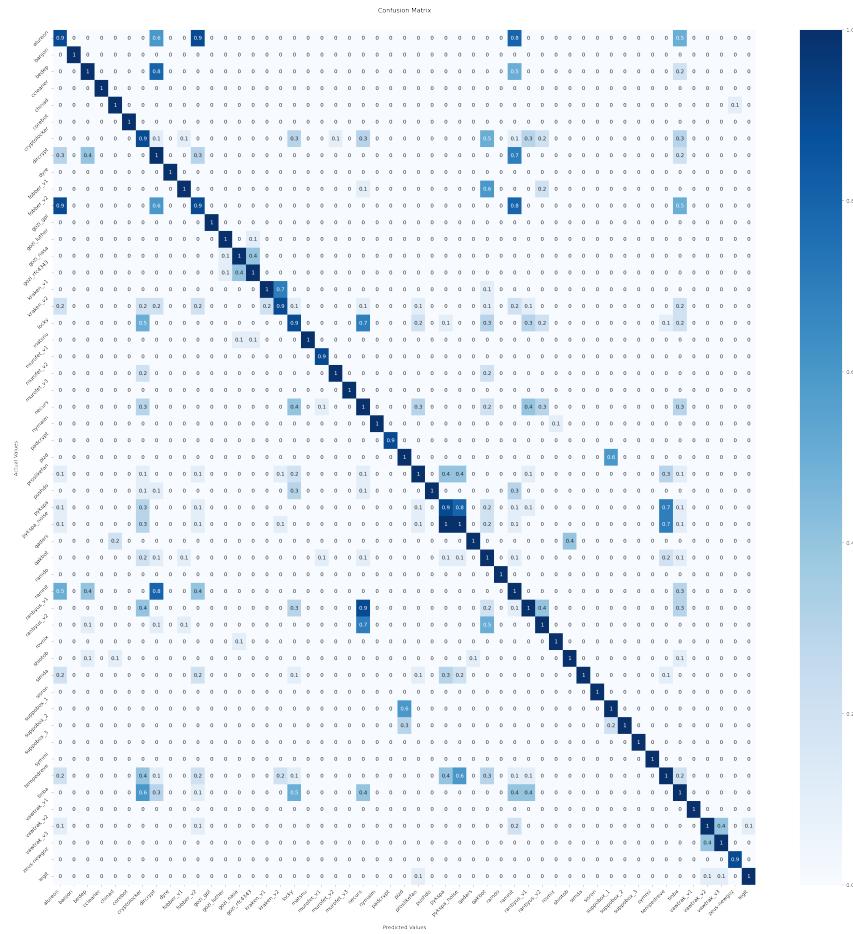


Figura 3.3: CM 0.017 Multi Input

3.7.2 Risultati Classificazione - Percentuale 0.034

Tabella 3.5: Risultati Metriche di Classificazione - 0.034

Metrica	Risultato
Accuracy	0.658689
F1-Score	0.648831
Precision	0.680713
Recall	0.658690

Matrice di Confusione Nel seguente paragrafo è riportata la matrice di confusione, così da avere un overview migliore sui risultati della classificazione (Fig.3.4)

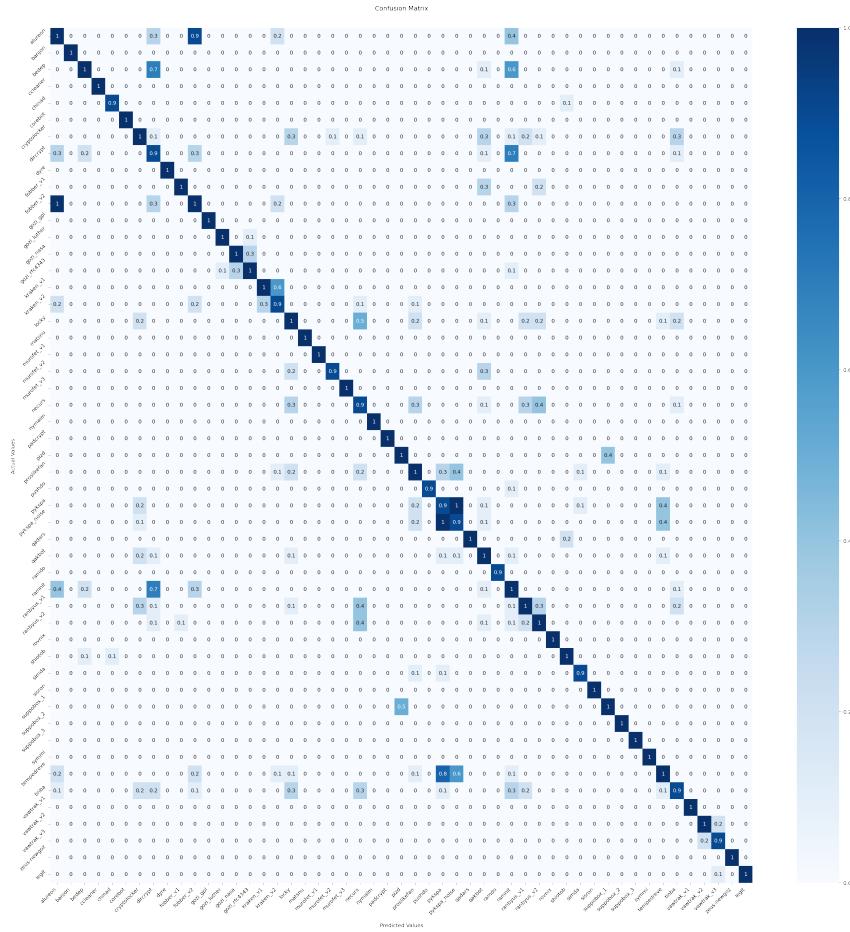


Figura 3.4: CM 0.034 Multi Input

3.7.3 Risultati Classificazione - Percentuale 0.068

Tabella 3.6: Risultati Metriche di Classificazione - 0.068

Metrica	Risultato
Accuracy	0.719612
F1-Score	0.708704
Precision	0.741019
Recall	0.719612

Matrice di Confusione Nel seguente paragrafo è riportata la matrice di confusione, così da avere un overview migliore sui risultati della classificazione (Fig.3.5)

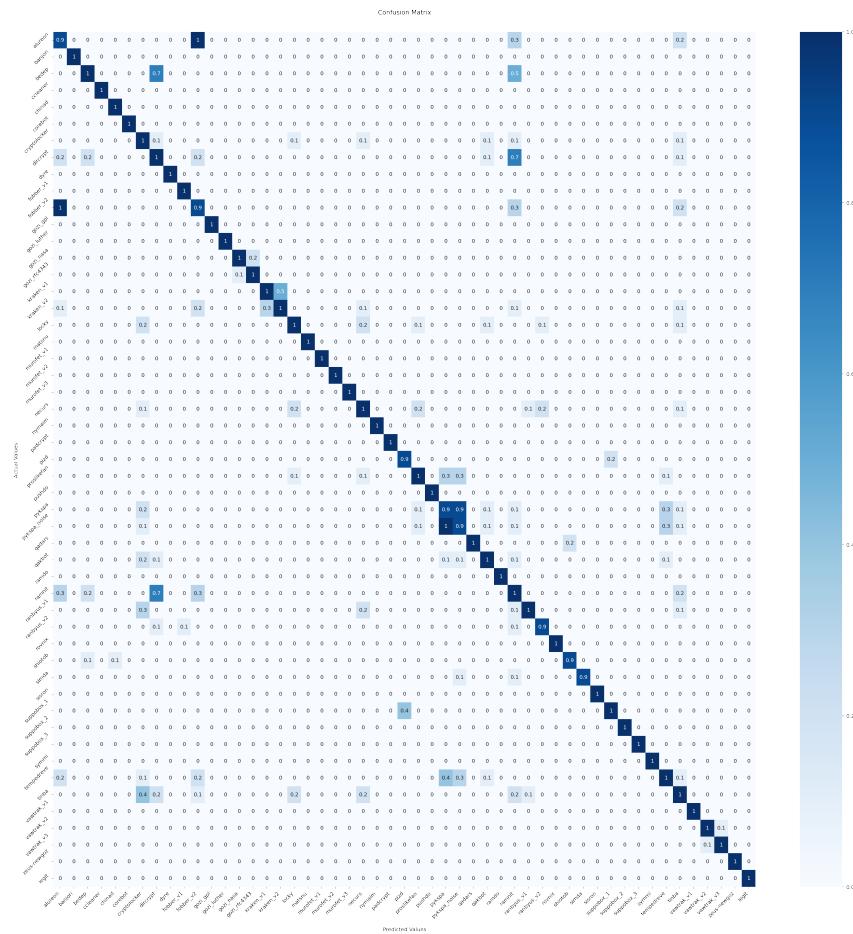


Figura 3.5: CM 0.068 Multi Input

3.7.4 Risultati Classificazione - Percentuale 0.135

Tabella 3.7: Risultati Metriche di Classificazione - 0.135

Metrica	Risultato
Accuracy	0.779504
F1-Score	0.773470
Precision	0.784416
Recall	0.779504

Matrice di Confusione Nel seguente paragrafo è riportata la matrice di confusione, così da avere un overview migliore sui risultati della classificazione (Fig.3.6)

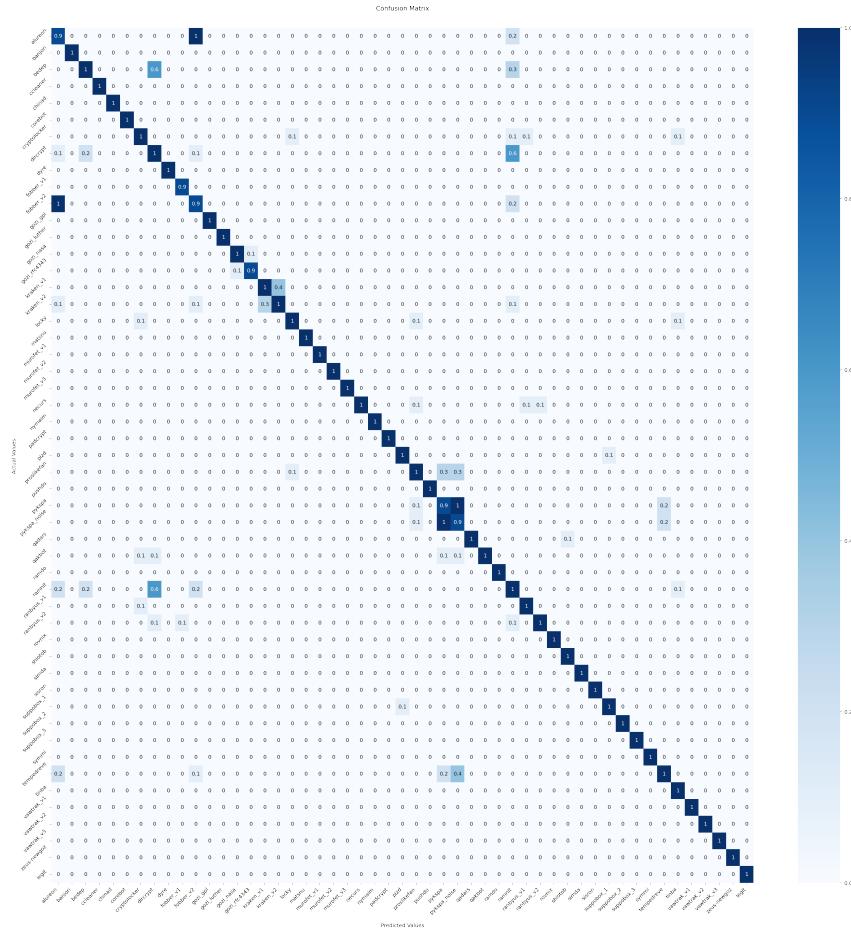


Figura 3.6: CM 0.135 Multi Input

3.8 Risultati Classificazione con Multi-Input e Random Embedding

In questa sezione sono riportati i risultati della classificazione con Random Embedding, dividen-doli per ciascun intervallo di dataset considerato.

3.8.1 Risultati Classificazione - Percentuale 0.017

Tabella 3.8: Risultati Metriche di Classificazione - 0.017

Metrica	Risultato
Accuracy	0.613173
F1-Score	0.601629
Precision	0.631673
Recall	0.613173

Matrice di Confusione Nel seguente paragrafo è riportata la matrice di confusione, così da avere un overview migliore sui risultati della classificazione (Fig.3.7)

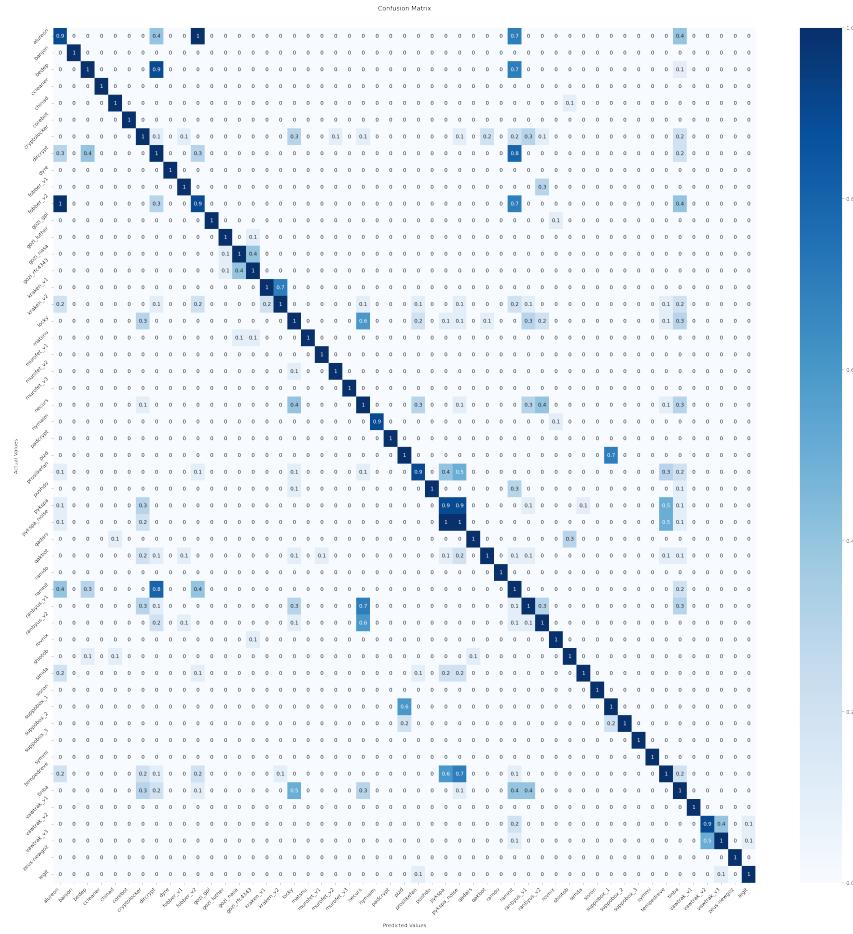


Figura 3.7: CM 0.017 Multi Input con Random Embeddings

3.8.2 Risultati Classificazione - Percentuale 0.034

Tabella 3.9: Risultati Metriche di Classificazione - 0.034.

Metrica	Risultato
Accuracy	0.671993
F1-Score	0.661358
Precision	0.685101
Recall	0.671993

Matrice di Confusione Nel seguente paragrafo è riportata la matrice di confusione, così da avere un overview migliore sui risultati della classificazione (Fig.3.8)

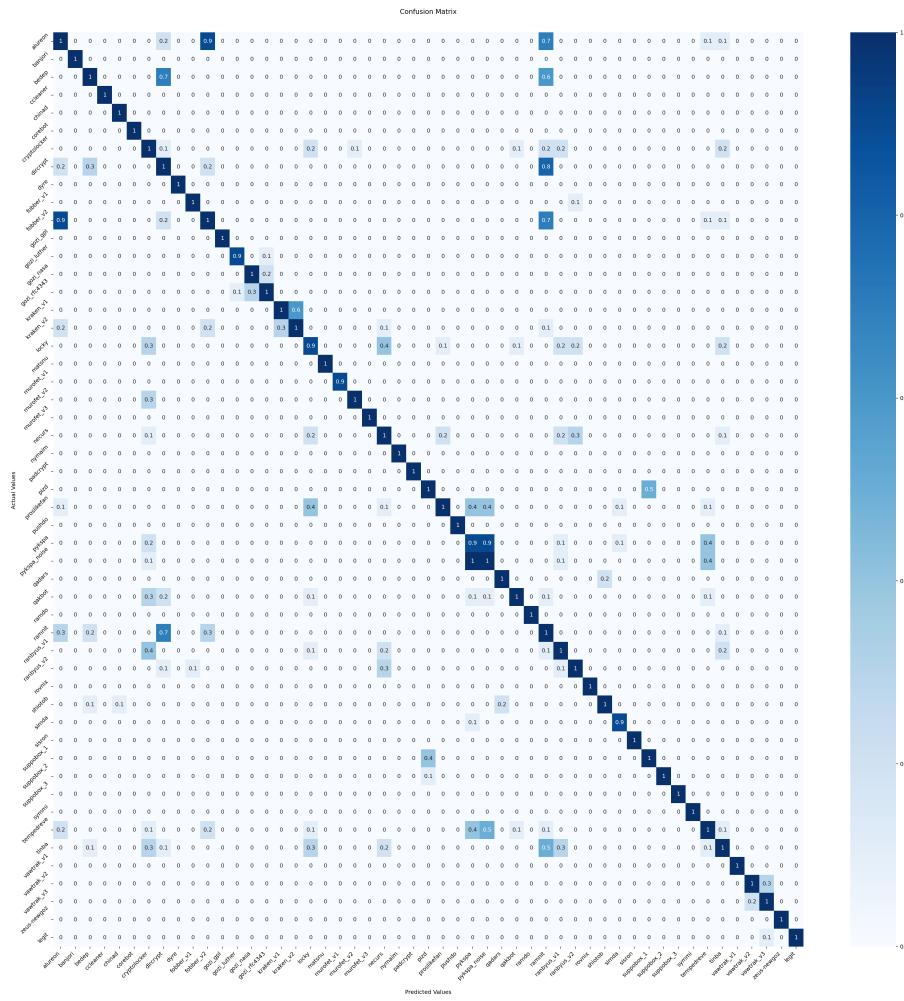


Figura 3.8: CM 0.034 Multi Input con Random Embeddings

3.8.3 Risultati Classificazione - Percentuale 0.068

Tabella 3.10: Risultati Metriche di Classificazione - 0.068

Metrica	Risultato
Accuracy	0.697278
F1-Score	0.692756
Precision	0.748620
Recall	0.697278

Matrice di Confusione Nel seguente paragrafo è riportata la matrice di confusione, così da avere un overview migliore sui risultati della classificazione (Fig.3.9)

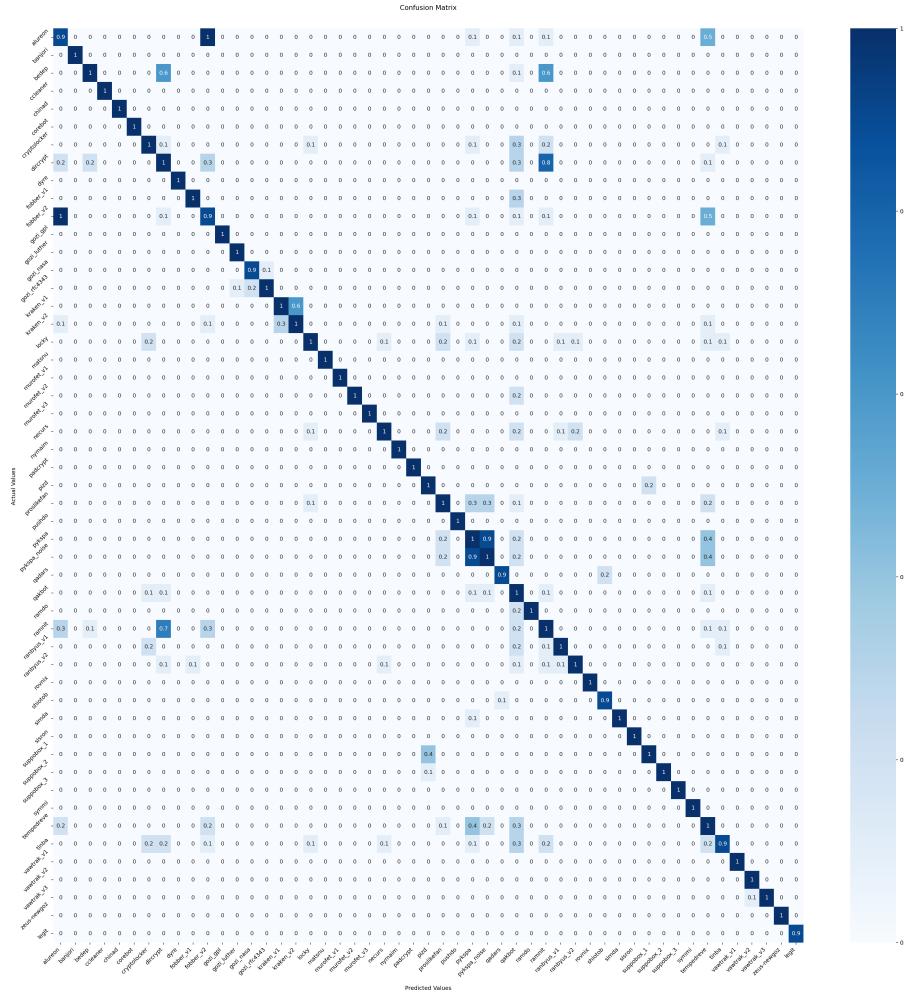


Figura 3.9: CM 0.068 Multi Input con Random Embeddings

3.8.4 Risultati Classificazione - Percentuale 0.135

Tabella 3.11: Risultati Metriche di Classificazione - 0.135

Metrica	Risultato
Accuracy	0.782414
F1-Score	0.780383
Precision	0.788942
Recall	0.782414

Matrice di Confusione Nel seguente paragrafo è riportata la matrice di confusione, così da avere un overview migliore sui risultati della classificazione (Fig.3.10)

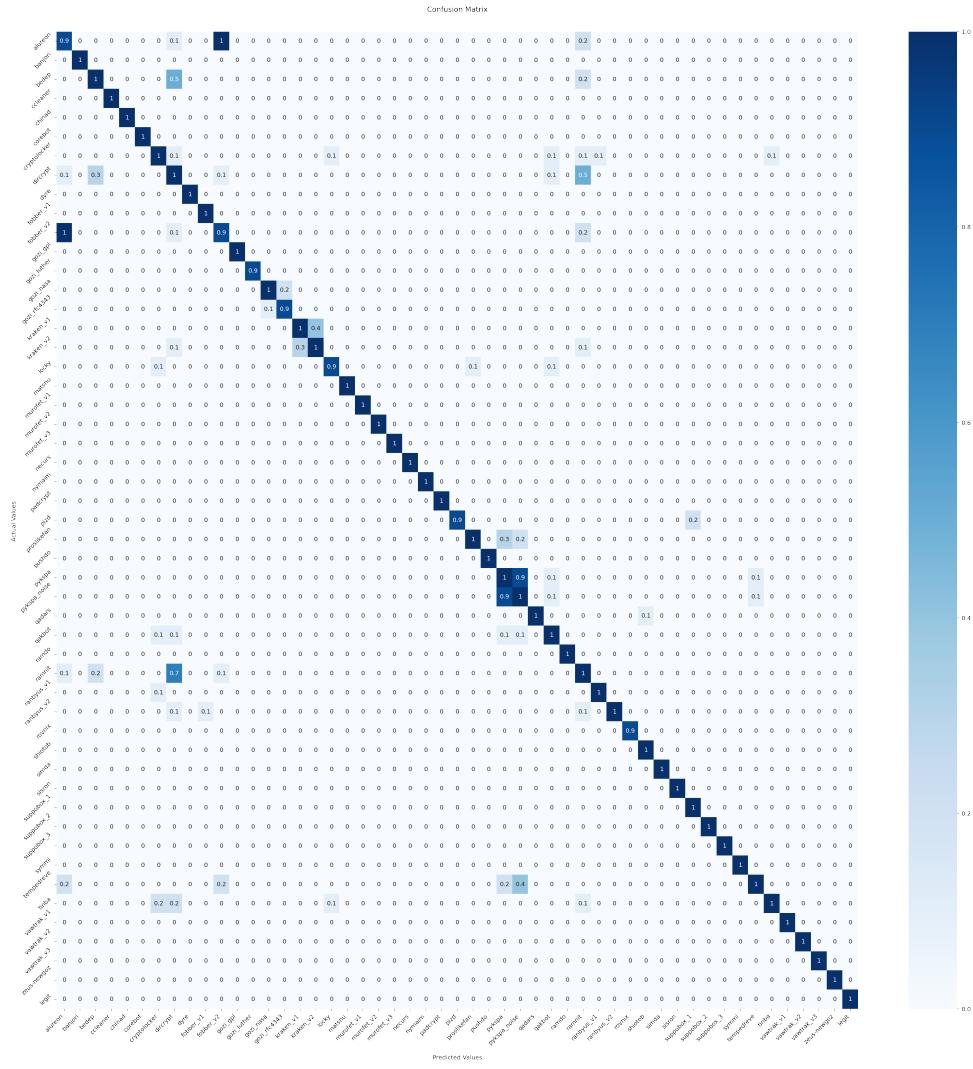


Figura 3.10: CM 0.135 Multi Input con Random Embeddings

3.9 Confronto Risultati Architetture

3.9.1 Risultati Classificazione

Tabella 3.12: Risultati Metriche di Classificazione

Architettura	Percentuale Train	Accuracy	F1-Score	Precision	Recall
Multi-Input	0.017	0.594858	0.578561	0.609030	0.594858
Multi-Input con Random Emb.	0.017	0.613173	0.601629	0.631673	0.613173
Multi-Input	0.034	0.658689	0.648831	0.680713	0.658690
Multi-Input con Random Emb.	0.034	0.671993	0.661358	0.685101	0.671993
Multi-Input	0.068	0.719612	0.708704	0.741019	0.719612
Multi-Input con Random Emb.	0.068	0.697278	0.692756	0.748620	0.697278
Multi-Input	0.135	0.779504	0.773470	0.784416	0.779504
Multi-Input con Random Emb.	0.135	0.782414	0.780383	0.788942	0.782414

Nella tabella 3.12 si possono confrontare i valori delle metriche ottenuti con le due differenti architetture addestrate precedentemente.

Ponendo particolare attenzione sulle metriche di accuracy e f1-score, si può osservare come, per i tagli più piccoli (0.017 e 0.034), si ottengono risultati migliori utilizzando l'architettura che racchiude anche lo strato di embedding randomico. Per le altre percentuali si ottengono invece valori poco differenti.

Capitolo 4

Conclusioni e Sviluppi Futuri

In conclusione, con il lavoro realizzato è possibile addestrare e testare differenti architetture al fine di effettuare una classificazione multi-classe per vari nomi di dominio appartenenti al dataset UMUDGA.

Dal confronto fra le diverse architetture utilizzate è possibile osservare che l'architettura Multi-Input che include lo strato di embedding casuale riesce ad ottenere risultati migliori soprattutto per percentuali di training più piccole, quindi nei casi in cui si hanno a disposizione pochi nomi di dominio (si pensi a quelle situazioni in cui un nuovo DGA è stato individuato). Il vantaggio di tale architettura tende a diminuire, confrontato con i valori dell'architettura Multi-Input senza strato di embedding casuale, al crescere della percentuale di train e quindi del numero di nomi di dominio.

Si può concludere, quindi, che l'obiettivo prefissato di classificare i nomi di dominio presenti nel dataset sfruttando un pre-addestramento della FastText, è stato raggiunto.

Possibili sviluppi futuri potrebbero riguardare l'allenamento di nuove architetture, i cui risultati possono essere confrontati con quelli presenti in questo lavoro. Altri possibili sviluppi potrebbero riguardare la prima fase compiuta in questo lavoro, ovvero l'addestramento della FastText da sfruttare poi negli strati di embedding. Si potrebbe pensare di ampliare il dataset utilizzato o effettuare un'operazione di pulizia dei nomi di dominio estratti dal GARR più profonda, ad esempio tenendo soltanto i nomi di dominio di primo livello. Così facendo, si potrebbero confrontare i risultati ottenuti con quelli presentati in questo elaborato, così da osservare i cambiamenti e la dipendenza dei risultati della seconda fase con il dataset utilizzato nella prima fase.

Bibliografia

- [1] FastText. Word representations, May 2022. URL <https://fasttext.cc/docs/en/unsupervised-tutorial.html>.
- [2] FastText. Cbow vs skipgram, May 2022. URL <https://fasttext.cc/docs/en/unsupervised-tutorial.html#advanced-readers-skipgram-versus-cbow>.
- [3] Mattia Zago, Manuel Gil Pérez, and Gregorio Martínez Pérez. Umudga: A dataset for profiling dga-based botnet. *Computers & Security*, 92:101719, 2020. ISSN 0167-4048. doi: <https://doi.org/10.1016/j.cose.2020.101719>. URL <https://www.sciencedirect.com/science/article/pii/S0167404820300067>.

Appendice A

Risultati Multi Input

A.1 Risultati Multi Input 0.017

	precision	recall	f1-score	support
alureon	0.212	0.305	0.236	983.000
banjori	0.996	1.000	0.998	983.000
bedep	0.459	0.333	0.342	983.000
ccleaner	0.973	1.000	0.986	983.000
chinad	0.742	0.722	0.722	983.000
corebot	0.983	0.977	0.980	983.000
cryptolocker	0.177	0.329	0.214	983.000
dircrypt	0.195	0.275	0.182	983.000
dyre	0.995	1.000	0.997	983.000
fobber_v1	0.587	0.838	0.684	983.000
fobber_v2	0.213	0.169	0.178	983.000
gozi_gpl	0.828	0.892	0.854	983.000
gozi_luther	0.752	0.771	0.749	983.000
gozi_nasa	0.631	0.351	0.419	983.000
gozi_rfc4343	0.479	0.695	0.547	983.000
kraken_v1	0.743	0.638	0.669	983.000
kraken_v2	0.371	0.229	0.279	983.000
locky	0.309	0.168	0.208	983.000
matsnu	0.798	0.781	0.746	983.000
murofet_v1	0.843	0.955	0.894	983.000
murofet_v2	0.741	0.862	0.791	983.000
murofet_v3	0.995	0.990	0.992	983.000
necurs	0.222	0.159	0.163	983.000
nymaim	0.857	0.659	0.738	983.000
padcrypt	0.904	0.949	0.924	983.000
pizd	0.483	0.558	0.484	983.000
proslikefan	0.440	0.362	0.381	983.000
pushdo	0.681	0.660	0.653	983.000
pykspa	0.242	0.101	0.126	983.000
pykspa_noise	0.248	0.247	0.228	983.000
qadars	0.785	0.582	0.646	983.000
qakbot	0.415	0.203	0.212	983.000

Table A.1 continued from previous page

	precision	recall	f1-score	support
ramdo	0.810	0.966	0.878	983.000
ramnit	0.169	0.186	0.168	983.000
ranbyus_v1	0.316	0.104	0.138	983.000
ranbyus_v2	0.326	0.500	0.365	983.000
rovnix	0.804	0.805	0.798	983.000
shiotob	0.566	0.553	0.539	983.000
simda	0.691	0.573	0.612	983.000
sisron	0.948	0.970	0.958	983.000
suppobox_1	0.524	0.605	0.538	983.000
suppobox_2	0.859	0.621	0.717	983.000
suppobox_3	0.938	0.994	0.965	983.000
symmi	0.969	0.989	0.978	983.000
tempedreve	0.225	0.171	0.189	983.000
tinba	0.178	0.127	0.140	983.000
vawtrak_v1	0.970	0.981	0.975	983.000
vawtrak_v2	0.512	0.415	0.402	983.000
vawtrak_v3	0.503	0.704	0.563	983.000
zeus-newgoz	0.917	0.986	0.950	983.000
legit	0.751	0.444	0.545	983.000
accuracy	0.597	0.597	0.597	0.597
macro avg	0.613	0.597	0.581	50 133.000
weighted avg	0.613	0.597	0.581	50 133.000

Tabella A.1: Multi Input Class Report 0.017

A.2 Risultati Multi Input 0.034

	precision	recall	f1-score	support
allureon	0.261	0.388	0.272	966.000
banjori	0.999	1.000	1.000	966.000
bedep	0.469	0.573	0.476	966.000
ccleaner	0.986	1.000	0.993	966.000
chinad	0.816	0.893	0.842	966.000
corebot	0.995	0.984	0.989	966.000
cryptolocker	0.410	0.235	0.284	966.000
dircrypt	0.255	0.218	0.232	966.000
dyre	0.997	1.000	0.998	966.000
fobber_v1	0.724	0.888	0.794	966.000
fobber_v2	0.266	0.345	0.265	966.000
gozi_gpl	0.908	0.942	0.924	966.000
gozi_luther	0.776	0.882	0.824	966.000
gozi_nasa	0.622	0.737	0.658	966.000
gozi_rfc4343	0.741	0.479	0.572	966.000
kraken_v1	0.754	0.660	0.692	966.000
kraken_v2	0.462	0.392	0.411	966.000
locky	0.383	0.205	0.229	966.000

Table A.2 continued from previous page

	precision	recall	f1-score	support
matsnu	0.852	0.949	0.895	966.000
murofet_v1	0.954	0.967	0.960	966.000
murofet_v2	0.765	0.957	0.848	966.000
murofet_v3	0.999	0.987	0.993	966.000
necurs	0.381	0.249	0.258	966.000
nymaim	0.927	0.801	0.859	966.000
padcrypt	0.943	0.976	0.959	966.000
pizd	0.642	0.553	0.557	966.000
proslikefan	0.444	0.463	0.440	966.000
pushdo	0.800	0.802	0.799	966.000
pykspa	0.262	0.224	0.189	966.000
pykspa_noise	0.261	0.163	0.179	966.000
qadars	0.869	0.663	0.729	966.000
qakbot	0.609	0.320	0.357	966.000
ramdo	0.860	0.994	0.920	966.000
ramnit	0.168	0.136	0.138	966.000
ranbyus_v1	0.488	0.416	0.415	966.000
ranbyus_v2	0.487	0.552	0.504	966.000
rovnix	0.897	0.850	0.870	966.000
shiotob	0.669	0.566	0.605	966.000
simda	0.738	0.701	0.717	966.000
sisron	0.985	0.997	0.991	966.000
suppobox_1	0.626	0.709	0.631	966.000
suppobox_2	0.875	0.825	0.846	966.000
suppobox_3	0.970	0.992	0.981	966.000
symmi	0.981	0.998	0.989	966.000
tempedreve	0.347	0.364	0.308	966.000
tinba	0.337	0.318	0.278	966.000
vawtrak_v1	0.975	0.996	0.986	966.000
vawtrak_v2	0.641	0.740	0.671	966.000
vawtrak_v3	0.662	0.659	0.647	966.000
zeus-newgoz	0.999	0.982	0.991	966.000
legit	0.735	0.613	0.658	966.000
accuracy	0.673	0.673	0.673	0.673
macro avg	0.686	0.673	0.659	49 266.000
weighted avg	0.686	0.673	0.659	49 266.000

Tabella A.2: Multi Input Class Report 0.034

A.3 Risultati Multi Input 0.068

	precision	recall	f1-score	support
alureon	0.257	0.388	0.264	932.000
banjori	0.997	1.000	0.999	932.000
bedep	0.553	0.567	0.502	932.000
ccleaner	0.983	1.000	0.991	932.000

Table A.3 continued from previous page

	precision	recall	f1-score	support
chinad	0.893	0.926	0.906	932.000
corebot	0.998	0.994	0.996	932.000
cryptolocker	0.397	0.360	0.365	932.000
dircrypt	0.349	0.255	0.271	932.000
dyre	0.999	1.000	0.999	932.000
fobber_v1	0.712	0.971	0.810	932.000
fobber_v2	0.296	0.302	0.247	932.000
gozi_gpl	0.955	0.922	0.936	932.000
gozi_luther	0.803	0.916	0.852	932.000
gozi_nasa	0.768	0.716	0.722	932.000
gozi_rfc4343	0.774	0.615	0.675	932.000
kraken_v1	0.749	0.760	0.726	932.000
kraken_v2	0.593	0.384	0.400	932.000
locky	0.511	0.344	0.382	932.000
matsnu	0.896	0.959	0.926	932.000
murofet_v1	0.954	0.996	0.974	932.000
murofet_v2	0.845	0.929	0.883	932.000
murofet_v3	1.000	0.995	0.998	932.000
necurs	0.589	0.306	0.369	932.000
nymaim	0.878	0.846	0.857	932.000
padcrypt	0.957	0.976	0.966	932.000
pizd	0.671	0.744	0.672	932.000
proslikefan	0.530	0.485	0.489	932.000
pushdo	0.835	0.905	0.859	932.000
pykspa	0.293	0.212	0.240	932.000
pykspa_noise	0.297	0.179	0.205	932.000
qadars	0.849	0.759	0.780	932.000
qakbot	0.728	0.388	0.502	932.000
ramdo	0.909	0.998	0.951	932.000
ramnit	0.283	0.052	0.085	932.000
ranbyus_v1	0.555	0.670	0.590	932.000
ranbyus_v2	0.637	0.605	0.576	932.000
rovnix	0.873	0.925	0.893	932.000
shiotob	0.701	0.607	0.599	932.000
simda	0.816	0.827	0.819	932.000
sisron	0.984	0.997	0.991	932.000
suppobox_1	0.768	0.646	0.628	932.000
suppobox_2	0.924	0.904	0.912	932.000
suppobox_3	0.984	0.990	0.987	932.000
symmi	0.991	0.998	0.995	932.000
tempedreve	0.468	0.445	0.442	932.000
tinba	0.387	0.498	0.340	932.000
vawtrak_v1	0.987	0.999	0.993	932.000
vawtrak_v2	0.829	0.765	0.784	932.000
vawtrak_v3	0.696	0.917	0.787	932.000
zeus-newgoz	0.999	0.997	0.998	932.000
legit	0.866	0.570	0.678	932.000

Table A.3 continued from previous page

	precision	recall	f1-score	support
accuracy	0.716	0.716	0.716	0.716
macro avg	0.737	0.716	0.702	47532.000
weighted avg	0.737	0.716	0.702	47532.000

Tabella A.3: Multi Input Class Report 0.068

A.4 Risultati Multi Input 0.135

	precision	recall	f1-score	support
alureon	0.318	0.388	0.315	865.000
banjori	1.000	1.000	1.000	865.000
bedep	0.529	0.624	0.569	865.000
ccleaner	0.995	1.000	0.998	865.000
chinad	0.873	0.971	0.918	865.000
corebot	0.999	0.996	0.997	865.000
cryptolocker	0.548	0.401	0.435	865.000
dircrypt	0.304	0.311	0.292	865.000
dyre	0.999	1.000	0.999	865.000
fobber_v1	0.800	0.965	0.874	865.000
fobber_v2	0.322	0.391	0.333	865.000
gozi_gpl	0.964	0.957	0.960	865.000
gozi_luther	0.863	0.926	0.890	865.000
gozi_nasa	0.782	0.791	0.773	865.000
gozi_rfc4343	0.796	0.702	0.735	865.000
kraken_v1	0.788	0.743	0.750	865.000
kraken_v2	0.607	0.483	0.498	865.000
locky	0.641	0.419	0.502	865.000
matsnu	0.925	0.963	0.944	865.000
murofet_v1	0.976	0.997	0.987	865.000
murofet_v2	0.824	0.982	0.895	865.000
murofet_v3	1.000	0.996	0.998	865.000
necurs	0.732	0.548	0.622	865.000
nymaim	0.939	0.877	0.906	865.000
padcrypt	0.966	0.994	0.980	865.000
pizd	0.821	0.871	0.837	865.000
proslikefan	0.602	0.525	0.552	865.000
pushdo	0.945	0.938	0.941	865.000
pykspa	0.321	0.269	0.271	865.000
pykspa_noise	0.330	0.242	0.240	865.000
qadars	0.912	0.868	0.886	865.000
qakbot	0.615	0.477	0.524	865.000
ramdo	0.968	1.000	0.983	865.000
ramnit	0.269	0.173	0.177	865.000
ranbyus_v1	0.572	0.819	0.655	865.000
ranbyus_v2	0.720	0.745	0.729	865.000
rovnix	0.911	0.950	0.929	865.000

Table A.4 continued from previous page

	precision	recall	f1-score	support
shiotob	0.821	0.661	0.725	865.000
simda	0.893	0.918	0.905	865.000
sisron	0.994	1.000	0.997	865.000
suppobox_1	0.871	0.865	0.863	865.000
suppobox_2	0.964	0.942	0.953	865.000
suppobox_3	0.988	0.999	0.993	865.000
symmi	0.994	0.999	0.996	865.000
tempedreve	0.492	0.546	0.512	865.000
tinba	0.486	0.465	0.452	865.000
vawtrak_v1	0.991	1.000	0.996	865.000
vawtrak_v2	0.932	0.965	0.948	865.000
vawtrak_v3	0.903	0.964	0.931	865.000
zeus-newgoz	1.000	0.998	0.999	865.000
legit	0.867	0.703	0.774	865.000
accuracy	0.771	0.771	0.771	0.771
macro avg	0.778	0.771	0.764	44115.000
weighted avg	0.778	0.771	0.764	44115.000

Tabella A.4: Multi Input Class Report 0.135

Appendice B

Risultati Multi Input with Random Embedding

B.1 Multi Input with Random Embedding 0.017

	precision	recall	f1-score	support
alureon	0.221	0.305	0.212	983.000
banjori	0.993	1.000	0.996	983.000
bedep	0.361	0.638	0.434	983.000
ccleaner	0.967	1.000	0.983	983.000
chinad	0.784	0.701	0.733	983.000
corebot	0.992	0.965	0.978	983.000
cryptolocker	0.260	0.150	0.164	983.000
dircrypt	0.217	0.082	0.105	983.000
dyre	0.992	1.000	0.996	983.000
fobber_v1	0.612	0.886	0.716	983.000
fobber_v2	0.207	0.248	0.207	983.000
gozi_gpl	0.787	0.972	0.867	983.000
gozi_luther	0.705	0.870	0.771	983.000
gozi_nasa	0.543	0.579	0.549	983.000
gozi_rfc4343	0.546	0.504	0.509	983.000
kraken_v1	0.621	0.694	0.627	983.000
kraken_v2	0.354	0.228	0.268	983.000
locky	0.258	0.293	0.207	983.000
matsnu	0.899	0.794	0.837	983.000
murofet_v1	0.801	0.907	0.824	983.000
murofet_v2	0.765	0.832	0.773	983.000
murofet_v3	0.989	0.975	0.982	983.000
necurs	0.299	0.103	0.135	983.000
nymaim	0.864	0.719	0.773	983.000
padcrypt	0.958	0.928	0.942	983.000
pizd	0.545	0.410	0.436	983.000
proslikefan	0.414	0.427	0.398	983.000
pushdo	0.715	0.657	0.642	983.000
pykspa	0.264	0.174	0.182	983.000
pykspa_noise	0.251	0.153	0.184	983.000

Table B.1 continued from previous page

	precision	recall	f1-score	support
qadars	0.700	0.674	0.678	983.000
qakbot	0.323	0.193	0.214	983.000
ramdo	0.848	0.942	0.887	983.000
ramnit	0.174	0.099	0.124	983.000
ranbyus_v1	0.343	0.319	0.273	983.000
ranbyus_v2	0.405	0.394	0.354	983.000
rovnix	0.868	0.718	0.784	983.000
shiotob	0.660	0.398	0.459	983.000
simda	0.665	0.597	0.619	983.000
sisron	0.910	0.981	0.944	983.000
suppobox_1	0.503	0.727	0.588	983.000
suppobox_2	0.845	0.732	0.780	983.000
suppobox_3	0.962	0.988	0.975	983.000
symmi	0.949	0.997	0.972	983.000
tempedreve	0.286	0.201	0.202	983.000
tinba	0.227	0.145	0.162	983.000
vawtrak_v1	0.972	0.989	0.980	983.000
vawtrak_v2	0.504	0.584	0.533	983.000
vawtrak_v3	0.555	0.579	0.562	983.000
zeus-newgoz	0.922	0.969	0.940	983.000
legit	0.710	0.494	0.574	983.000
accuracy	0.606	0.606	0.606	0.606
macro avg	0.618	0.606	0.589	50 133.000
weighted avg	0.618	0.606	0.589	50 133.000

Tabella B.1: Multi Input With Random Embedding Class Report 0017

B.2 Multi Input with Random Embedding 0.034

	precision	recall	f1-score	support
alureon	0.273	0.152	0.165	966.000
banjori	0.998	1.000	0.999	966.000
bedep	0.633	0.436	0.490	966.000
ccleaner	0.983	1.000	0.991	966.000
chinad	0.927	0.855	0.885	966.000
corebot	0.999	0.986	0.992	966.000
cryptolocker	0.302	0.368	0.315	966.000
dircrypt	0.262	0.209	0.214	966.000
dyre	0.994	1.000	0.997	966.000
fobber_v1	0.769	0.889	0.813	966.000
fobber_v2	0.280	0.456	0.334	966.000
gozi_gpl	0.890	0.961	0.924	966.000
gozi_luther	0.819	0.855	0.833	966.000
gozi_nasa	0.536	0.843	0.650	966.000
gozi_rfc4343	0.776	0.389	0.488	966.000
kraken_v1	0.775	0.641	0.685	966.000

Table B.2 continued from previous page

	precision	recall	f1-score	support
kraken_v2	0.391	0.439	0.352	966.000
locky	0.370	0.255	0.252	966.000
matsnu	0.935	0.930	0.932	966.000
murofet_v1	0.947	0.986	0.965	966.000
murofet_v2	0.796	0.796	0.767	966.000
murofet_v3	1.000	0.993	0.997	966.000
necurs	0.373	0.260	0.282	966.000
nymaim	0.939	0.825	0.877	966.000
padcrypt	0.962	0.954	0.957	966.000
pizd	0.685	0.525	0.529	966.000
proslikefan	0.482	0.423	0.433	966.000
pushdo	0.845	0.760	0.787	966.000
pykspa	0.240	0.199	0.207	966.000
pykspa_noise	0.271	0.234	0.227	966.000
qadars	0.820	0.801	0.807	966.000
qakbot	0.610	0.343	0.403	966.000
ramdo	0.937	0.979	0.957	966.000
ramnit	0.208	0.337	0.199	966.000
ranbyus_v1	0.513	0.522	0.488	966.000
ranbyus_v2	0.573	0.442	0.484	966.000
rovnix	0.934	0.798	0.858	966.000
shiotob	0.757	0.570	0.636	966.000
simda	0.661	0.819	0.726	966.000
sisron	0.977	0.998	0.988	966.000
suppobox_1	0.596	0.742	0.636	966.000
suppobox_2	0.904	0.837	0.868	966.000
suppobox_3	0.975	0.989	0.982	966.000
symmi	0.982	1.000	0.991	966.000
tempedreve	0.413	0.273	0.294	966.000
tinba	0.407	0.200	0.253	966.000
vawtrak_v1	0.976	0.997	0.987	966.000
vawtrak_v2	0.657	0.738	0.689	966.000
vawtrak_v3	0.639	0.718	0.668	966.000
zeus-newgoz	1.000	0.992	0.996	966.000
legit	0.796	0.609	0.688	966.000
accuracy	0.673	0.673	0.673	0.673
macro avg	0.702	0.673	0.665	49 266.000
weighted avg	0.702	0.673	0.665	49 266.000

Tabella B.2: Multi Input With Random Embedding Class Report 0.034

B.3 Multi Input with Random Embedding 0.068

	precision	recall	f1-score	support
alureon	0.240	0.342	0.252	932.000
banjori	1.000	1.000	1.000	932.000

Table B.3 continued from previous page

	precision	recall	f1-score	support
bedep	0.607	0.567	0.558	932.000
ccleaner	0.986	1.000	0.993	932.000
chinad	0.907	0.930	0.917	932.000
corebot	0.999	0.987	0.993	932.000
cryptolocker	0.368	0.441	0.344	932.000
dircrypt	0.268	0.376	0.270	932.000
dyre	0.999	1.000	0.999	932.000
fobber_v1	0.778	0.956	0.856	932.000
fobber_v2	0.302	0.129	0.166	932.000
gozi_gpl	0.937	0.972	0.954	932.000
gozi_luther	0.752	0.927	0.824	932.000
gozi_nasa	0.730	0.762	0.741	932.000
gozi_rfc4343	0.772	0.619	0.681	932.000
kraken_v1	0.770	0.691	0.722	932.000
kraken_v2	0.483	0.515	0.433	932.000
locky	0.536	0.320	0.393	932.000
matsnu	0.910	0.952	0.930	932.000
murofet_v1	0.949	0.997	0.972	932.000
murofet_v2	0.787	0.926	0.843	932.000
murofet_v3	1.000	0.994	0.997	932.000
necurs	0.620	0.345	0.427	932.000
nymaim	0.929	0.838	0.880	932.000
padcrypt	0.988	0.952	0.970	932.000
pizd	0.799	0.488	0.541	932.000
proslikefan	0.563	0.468	0.510	932.000
pushdo	0.800	0.914	0.848	932.000
pykspa	0.299	0.167	0.192	932.000
pykspa_noise	0.336	0.164	0.202	932.000
qadars	0.908	0.815	0.853	932.000
qakbot	0.631	0.386	0.459	932.000
ramdo	0.931	0.971	0.949	932.000
ramnit	0.242	0.167	0.165	932.000
ranbyus_v1	0.590	0.600	0.555	932.000
ranbyus_v2	0.677	0.697	0.671	932.000
rovnix	0.938	0.907	0.922	932.000
shiotob	0.785	0.666	0.716	932.000
simda	0.788	0.836	0.805	932.000
sisron	0.989	0.998	0.993	932.000
suppobox_1	0.631	0.884	0.718	932.000
suppobox_2	0.932	0.883	0.906	932.000
suppobox_3	0.972	0.996	0.984	932.000
symmi	0.983	1.000	0.991	932.000
tempedreve	0.331	0.516	0.368	932.000
tinba	0.541	0.237	0.312	932.000
vawtrak_v1	0.985	1.000	0.993	932.000
vawtrak_v2	0.899	0.729	0.796	932.000
vawtrak_v3	0.725	0.933	0.813	932.000

Table B.3 continued from previous page

	precision	recall	f1-score	support
zeus-newgoz	1.000	0.998	0.999	932.000
legit	0.829	0.624	0.711	932.000
accuracy	0.717	0.717	0.717	0.717
macro avg	0.740	0.717	0.708	47532.000
weighted avg	0.740	0.717	0.708	47532.000

Tabella B.3: Multi Input With Random Embedding Class Report 0.068

B.4 Multi Input with Random Embedding 0.135

	precision	recall	f1-score	support
alureon	0.357	0.465	0.382	865.000
banjori	1.000	1.000	1.000	865.000
bedep	0.609	0.593	0.589	865.000
ccleaner	0.993	1.000	0.997	865.000
chinad	0.978	0.943	0.960	865.000
corebot	1.000	0.994	0.997	865.000
cryptolocker	0.498	0.514	0.481	865.000
dircrypt	0.303	0.335	0.309	865.000
dyre	0.999	1.000	1.000	865.000
fobber_v1	0.842	0.932	0.884	865.000
fobber_v2	0.381	0.293	0.308	865.000
gozi_gpl	0.954	0.972	0.963	865.000
gozi_luther	0.846	0.946	0.891	865.000
gozi_nasa	0.854	0.711	0.769	865.000
gozi_rfc4343	0.764	0.775	0.756	865.000
kraken_v1	0.786	0.714	0.745	865.000
kraken_v2	0.576	0.508	0.535	865.000
locky	0.654	0.477	0.542	865.000
matsnu	0.923	0.964	0.942	865.000
murofet_v1	0.989	0.993	0.991	865.000
murofet_v2	0.855	0.957	0.903	865.000
murofet_v3	1.000	0.997	0.999	865.000
necurs	0.753	0.579	0.650	865.000
nymaim	0.953	0.880	0.915	865.000
padcrypt	0.975	0.995	0.985	865.000
pizd	0.827	0.876	0.848	865.000
proslikefan	0.628	0.505	0.540	865.000
pushdo	0.926	0.955	0.939	865.000
pykspa	0.322	0.222	0.256	865.000
pykspa_noise	0.302	0.367	0.312	865.000
qadars	0.930	0.920	0.924	865.000
qakbot	0.597	0.479	0.523	865.000
ramdo	0.981	1.000	0.991	865.000
ramnit	0.288	0.297	0.253	865.000
ranbyus_v1	0.720	0.787	0.746	865.000

Table B.4 continued from previous page

	precision	recall	f1-score	support
ranbyus_v2	0.737	0.765	0.747	865.000
rovnix	0.943	0.927	0.934	865.000
shiotob	0.879	0.806	0.840	865.000
simda	0.856	0.956	0.902	865.000
sisron	0.998	1.000	0.999	865.000
suppobox_1	0.880	0.864	0.869	865.000
suppobox_2	0.953	0.963	0.958	865.000
suppobox_3	0.991	0.999	0.995	865.000
symmi	0.992	1.000	0.996	865.000
tempedreve	0.591	0.567	0.563	865.000
tinba	0.621	0.590	0.593	865.000
vawtrak_v1	0.993	1.000	0.996	865.000
vawtrak_v2	0.939	0.958	0.949	865.000
vawtrak_v3	0.922	0.962	0.941	865.000
zeus-newgoz	1.000	1.000	1.000	865.000
legit	0.818	0.760	0.785	865.000
accuracy	0.786	0.786	0.786	0.786
macro avg	0.794	0.786	0.782	44115.000
weighted avg	0.794	0.786	0.782	44115.000

Tabella B.4: Multi Input With Random Embedding Class Report 0.135

Appendice C

Ulteriori Informazioni

- Vista l'impossibilità di caricare il file relativo ai domini utilizzati (in quanto su supera la dimensione limite disponibile su GITHUB), le allegiamo il link ad un nostro repository privato: [Link](#).
- Sempre in ottica di una miglior visualizzazione all'interno del *repository* sono presenti anche i file *reports* considerati nell'appendice.