

UNIVERSITÀ POLITECNICA DELLE MARCHE

INGEGNERIA INFORMATICA E DELL'AUTOMAZIONE

PROGETTO ADVANCED CYBERSECURITY 2021-2022

Analisi e Classificazione dei domini malevoli mediante Deep Learning



Professore

Prof. Luca Spalazzi

Gruppo 1

Michele Pasqualini
Denil Nicolosi
Massimo Ciaffoni
Francesco Zerbino Di Bernardo

Anno accademico 2021-2022

Indice

Elenco delle figure	3
Elenco delle tabelle	4
1 Introduzione	6
1.1 Tecnologie	7
2 Estrazione DNS e divisione in unigram, bigram e trigram	9
2.1 Divisione in 1-grams, 2-grams e 3-grams	12
3 Addestramento FastText	15
3.1 Introduzione	15
3.2 Addestramento	16
3.3 Tempi Computazionali	17
4 Training	18
4.1 Dataset UMUDGA	18
4.2 Architettura Stacked e scelta dei tagli	19
5 Risultati	21
5.1 Taglio 0.017	21
5.1.1 Matrice di Confusione	21
5.1.2 Risultati generali classificazione - Percentuale 0.017	23
5.2 Taglio 0.034	24
5.2.1 Matrice di Confusione	25
5.2.2 Risultati generali classificazione - Percentuale 0.034	27
5.3 Taglio 0.068	28
5.3.1 Matrice di Confusione	29
5.3.2 Risultati generali classificazione - Percentuale 0.068	31
5.4 Taglio 0.135	32
5.4.1 Matrice di Confusione	33
5.4.2 Risultati generali classificazione - Percentuale 0.135	35

5.5 Confronto risultati architettura	36
5.5.1 Risultati classificazione	36
6 Conclusioni e sviluppi futuri	37
Bibliografia	38

Elenco delle figure

1.1	DGA cyber threat	7
1.2	Linguaggio di programmazione python	8
1.3	Google Colab	8
3.1	Logo FastText	15
4.1	Esempio di contenuto del file " <i>umudga_1k.csv</i> "	18
4.2	Modello Stacked	19
5.1	Matrice di confusione per taglio di train 0.017	22
5.2	Matrice di confusione per taglio di train 0.034	26
5.3	Matrice di confusione per taglio di train 0.068	30
5.4	Matrice di confusione per taglio di train 0.135	34

Elenco delle tabelle

3.1	Tempi di addestramento FastText	17
4.1	Tempi di Addestramento architettura Stacked	20
5.1	Metriche per taglio di train di 0.017	21
5.2	Metriche di accuratezza per taglio 0.017	21
5.3	Metriche per taglio di train 0.017	24
5.4	Metriche per taglio di train di 0.034	24
5.5	Metriche di accuratezza per taglio 0.034	25
5.6	Metriche per taglio di train 0.034	28
5.7	Metriche per taglio di train di 0.068	28
5.8	Metriche di accuratezza per taglio 0.068	29
5.9	Metriche per taglio di train 0.068	32
5.10	Metriche per taglio di train di 0.135	32
5.11	Metriche di accuratezza per taglio 0.135	33
5.12	Metriche per taglio di train 0.135	36
5.13	Risultati metriche di classificazione	36

Listings

2.1	Script per l'estrazione dei log	9
2.2	Estrazione dei DNS	11
2.3	Suddivisione in unigrammi	12
2.4	Suddivisione in bigrammi	13
2.5	Suddivisione in trigrammi	13
3.1	Train del modello basato su unigrammi	16
3.2	Trasformazione del modello unigram da formato bin in formato vec	16

Capitolo 1

Introduzione

Gli attacchi informatici, nell'ultimo decennio, hanno registrato un incremento significativo sia sotto il profilo della frequenza che dell'impatto e delle dimensioni. Tra i più comuni troviamo phishing, trojan e worm, ovvero attacchi il cui scopo è quello di tentare di recuperare informazioni sensibili ai fini di azioni malevoli.

In questo progetto viene posta l'attenzione sui malware che permettono di creare le "*Botnet*", ovvero reti di macchine infette il quale vengono coordinate da un hacker attraverso un server di comando e controllo. Solitamente, per arginare questo tipo di infezione, si cerca di individuare e offuscare il server di comando e controllo. Questa operazione, però, non è molto semplice, perché come contromisura vengono adottate delle strategie che ne rendono difficile l'individuazione. Una di queste tecniche consiste nel modificare il nome di dominio dopo un certo intervallo di tempo, attraverso un DGA (Domain name Generation Algorithm).

Domain Generation Algorithm (DGA) è una tecnica che viene incorporata nel malware per generare periodicamente un gran numero di nomi di dominio pseudo-casuali inesistenti per il server di comando e controllo. Il malware tenta quindi di risolvere questi nomi di dominio generati inviando query DNS fino a quando uno dei domini non risolve l'indirizzo IP di un server di comando e controllo. I nomi di dominio generati da DGA funzionano come punti di connessione tra il malware e il suo server di comando e controllo. DGA viene utilizzato per impedire che il server di comando e controllo venga disattivato e ostacolare i tentativi di individuazione. Nella figura 1.1 viene mostrata una panoramica sul funzionamento di un DGA.

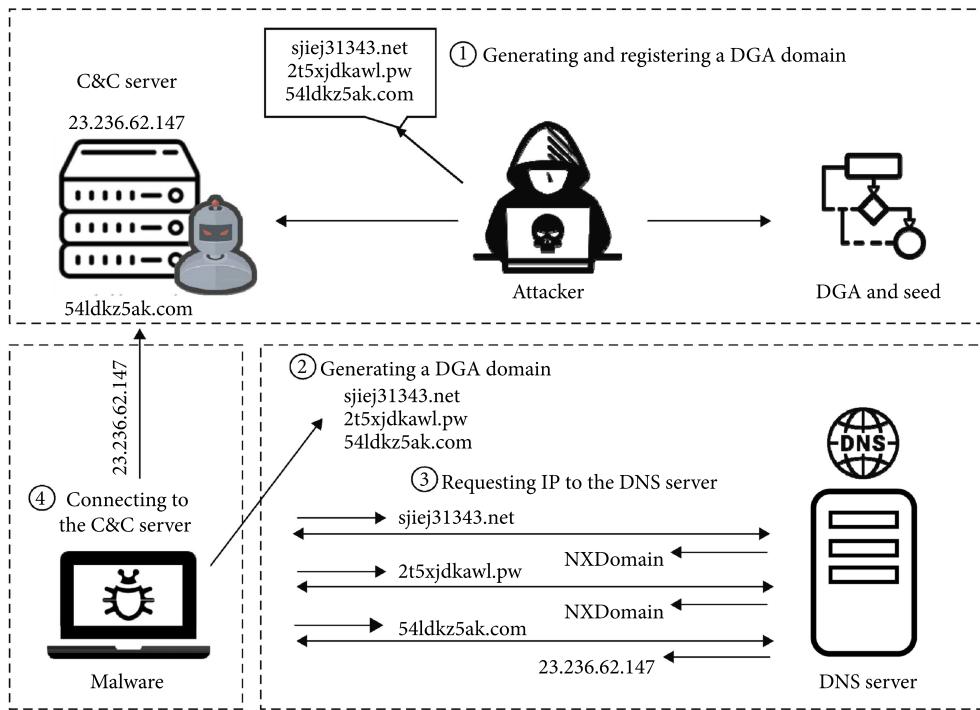


Figura 1.1: DGA cyber threat

Quindi, sfruttando questo meccanismo, i DGA sono in grado di creare nomi di dominio scambiabili per legittimi e simili a quelli reali, in modo tale da rendere molto oneroso il lavoro di analisi. La soluzione implementata in questo elaborato ha l’obiettivo di testare un’architettura basata su tecniche di Deep Learning che permetta di classificare i nomi di dominio in malevoli e non malevoli.

1.1 Tecnologie

Per lo sviluppo di questo progetto è stato utilizzato Google Colab insieme al linguaggio di programmazione python. Google Colab è un tool messo a disposizione da Google che offre la possibilità di scrivere notebook su un ambiente di sviluppo condiviso con all’interno presenti già diverse librerie di interesse. Altri vantaggi di Google Colab sono legati alla possibilità di lavorare direttamente su cloud ed interagire facilmente con Google Drive per l’importazione dei dati.



Figura 1.2: Linguaggio di programmazione python



Figura 1.3: Google Colab

Capitolo 2

Estrazione DNS e divisione in unigram, bigram e trigram

Per quanto riguarda i dataset utilizzati nella prima fase del progetto si è fatto uso di log scaricati dalla rete GARR. L'estrazione dei vari log è stata effettuata tramite uno script python il quale una volta presi in input l'anno di cui si vogliono estrarre i log e l'indirizzo IP (con relativa porta e password) del server va ad estrarre per ogni mese dei giorni in maniera casuale. Mandando in esecuzione molteplici volte lo script si sono estratti 10 log al mese per un periodo che va da aprile a luglio 2020 e da marzo ad ottobre 2021 per un totale di 120 log (la ragione di tale scelta è dovuta dall'assenza di dati in alcuni periodi). Inoltre si è prestata attenzione all'ora estratta in modo che tutte le ore del giorno fossero equamente rappresentate dal dataset.

```
1 import argparse
2 import paramiko
3 from random import randint
4 from paramiko.ssh_exception import SSHException
5 from scp import SCPClient, SCPException
6
7
8 #creo una connessione ssh al server
9 def createSSHClient(server, port, user, password):
10     client = paramiko.SSHClient()
11     client.load_system_host_keys()
12     client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
13     client.connect(server, port, user, password)
14     return client
15
16 #faccio uso del comando scp per scaricare i file di log
17 def scp_get(scp, dates, download_path):
18     for elem in dates:
19         scp.close()
```

```

20     try:
21         scp.get('VOL250GB/pdns_ ' + elem + '*', download_path)
22         print('Scarico --> ' + elem)
23     except SCPException as e:
24         print('Il log a data ' + elem + ' non esiste.')
25
26 def download(download_path, ip, port, host, psw, year):
27     ssh = createSSHClient(ip, port, host, psw)
28     scp = SCPClient(ssh.get_transport(), sanitize=lambda x: x)
29
30     year = year
31     dates = []
32
33     for month in range(1, 13):
34         month = str(month)
35         #estratto dei giorni da 1 a 31 a seconda del mese
36         if month in [4, 6, 9, 11]:
37             days = [randint(1, 30), randint(1, 30), randint(1, 30),
38 , randint(1, 30), randint(1, 30), randint(1, 30)]
39             elif int(month) == 2:
40                 days = [randint(1, 28), randint(1, 28), randint(1, 28),
41 , randint(1, 28), randint(1, 28), randint(1, 28)]
42             else:
43                 days = [randint(1, 31), randint(1, 31), randint(1, 31),
44 , randint(1, 31), randint(1, 31), randint(1, 31)]
45                 if int(month) < 10:
46                     month = '0' + str(month)
47                 for day in days:
48                     day = str(day)
49                     hour = str(randint(0, 23))
50                     if int(day) < 10:
51                         day = '0' + str(day)
52                     if int(hour) < 10:
53                         hour = '0' + str(hour)
54                     date = year + '_' + month + '_' + day + '_' + hour
55                     dates.append(date)
56
57     scp_get(scp, dates, download_path)
58
59 def main():
60     parser = argparse.ArgumentParser()
61     parser.add_argument('--ip-address', type=str, default='
62     193.205.129.120', help='Indirizzo IP dove scaricare i file GAR')
63     parser.add_argument('--port', type=str, default='63406', help=
64     'Porta indirizzo IP')

```

Listing 2.1: Script per l'estrazione dei log

Una volta estratti i log si è sfruttata la libreria Python *pandas* per effettuare ETL dei dati. In particolare per ogni file di log sono stati estratti tutti i DNS con un numero di caratteri superiore a 100 e tutti quei DNS che non fossero rappresentati da indirizzi IP. Per ognuno dei file dopo il filtraggio sono stati eliminati i possibili duplicati e si sono presi solo il 20% dei record in modo da avere un totale di 4198921 di DNS estratti.

```

1 def estrai_dns(dir_input, dir_output):
2     dns_estratti= 0
3     for filename in os.listdir(dir_input):
4         file = open(dir_input+'/'+filename, "r")
5         print("Estrazione DNS dal file di log:"+filename)
6         lines= file.readlines()
7         linee_ripulite = []
8         for line in lines:
9             elem= line.split(" ; ")
10            if(len(elem)<5):
11                #print("riga sbagliata")
12                continue
13            dns = elem[5]
14            #effettuo l'ETL dei log
15            if ripulisci_log(dns):
16                if (len(elem) == 10):
17                    linee_ripulite.append(line)
18        try:
19            df = pd.read_csv(io.StringIO('\n'.join(linee_ripulite)),
sep=' ; ', header=None, lineterminator='\n')
20            #per ogni riga prendo solo i DNS e ne estraggo il 20%
21            df = df.drop_duplicates(subset = [5])
22            df = df.sample(frac=0.2)
23            dns_estratti= dns_estratti+ len(df)

```

```

24     #scrivo i DNS estratti in un file di output
25     with open(dir_output + '/' + filename.split('.')[0] + ".log", "w") as out:
26         out.writelines('\n'.join(df.loc[:, 5].str[:-1].to_list()))
27         out.close()
28     except EmptyDataError:
29         print('File di log vuoto: ' + filename)
30     except Exception as e:
31         print("linea errata")
32     print("DNS estratti:" + str(dns_estratti))
33
34
35 def ripulisci_log(dns):
36     substring='arpa'
37     campi= dns.split('.')
38     #elimino tutti i DNS con numero di caratteri maggiore di 100 e
39     #quelli rappresentati da indirizzi IP
40     if((campi[0].isdigit() and campi[1].isdigit() and campi[2].isdigit() and campi[3].isdigit()) or (campi[0]=='.' and campi[1].isdigit() and campi[2].isdigit()) or len(dns)>100 or
41     substring in dns):
42         return False
43     else:
44         return True
45
46 estrai_dns(log_dir,dns_dir)

```

Listing 2.2: Estrazione dei DNS

2.1 Divisione in 1-grams, 2-grams e 3-grams

Una volta completata la fase di ETL, è stato implementato uno script per la creazione degli 1-grams, 2-grams e 3-grams, ovvero ciascun nome di dominio estratto viene suddiviso in sotto unità costituite rispettivamente da una, due e tre lettere. Nel listato seguente ne viene riportata l'implementazione. Per effettuare la divisione dei nomi di domini in unigrammi, bigrammi e trigrammi sono state realizzate tre funzioni diverse le quali salvano le divisioni in tre file distinti.

```

1 def dividi_dns_uni(input_dir, output_dir):
2     unigrams=[]
3     for filename in os.listdir(input_dir):
4         file = open(input_dir+'/'+filename, "r")
5         print("Divisione in unigram del file:" +filename)
6         lines= file.readlines()
7         for line in lines:

```

```

8         elem=line.replace(".", "")
9         unigrams.append(list(elem))
10        try:
11            with open(output_dir + '/' + "dns-unigram"+".log", "w") as
12                out:
13                    for elem in unigrams:
14                        out.writelines(''.join(elem))
15                out.close()
16            except Exception as e:
17                print("Errore"+e)
18
19 dividi_dns_uni(dns_dir,gram_dir)

```

Listing 2.3: Suddivisione in unigrammi

```

1 def dividi_dns_di(input_dir, output_dir):
2     digrams=[]
3     for filename in os.listdir(input_dir):
4         file = open(input_dir+'/'+filename, "r")
5         print("Divisione in digram del file:"+filename)
6         lines= file.readlines()
7         for line in lines:
8             elem=line.replace(".", "")
9             digrams.append(string_to_list_in_pairs(elem.replace("\n", ""))
10            try:
11                with open(output_dir + '/' + "dns-digram"+".log", "w") as out
12:
13                    for elem in digrams:
14                        out.writelines(''.join(elem)+'\n')
15                    out.close()
16                except Exception as e:
17                    print("Errore"+e)
18
19 def string_to_list_in_pairs (s):
20     return [ ''.join(pair) for pair in zip(s[:-1], s[1:])]
21
22 dividi_dns_di(dns_dir,gram_dir)

```

Listing 2.4: Suddivisione in bigrammi

```

1 def dividi_dns_tri(input_dir, output_dir):
2     trigrams=[]
3     for filename in os.listdir(input_dir):
4         file = open(input_dir+'/'+filename, "r")
5         print("Divisione in trigram del file:"+filename)
6         lines= file.readlines()
7         for line in lines:

```

```

8         elem=line.replace(".", "")
9         trigrams.append(string_to_list_in_tris(elem.replace("\n", ""))
10        )))
11        try:
12            with open(output_dir + '/' + "dns-trigram"+".log", "w") as
13                out:
14                    for elem in trigrams:
15                        out.writelines(''.join(elem)+'\n')
16                out.close()
17            except Exception as e:
18                print("Errore "+e)
19
20
21 dividi_dns_tri(dns_dir,gram_dir)

```

Listing 2.5: Suddivisone in trigrammi

Capitolo 3

Addestramento FastText

3.1 Introduzione

FastText è una libreria python open-source disponibile sul sito ufficiale fasttext.cc. Questa libreria è particolarmente indicata per la classificazione di testi mediante la rappresentazione di parole e frasi. Il modello di FastText può utilizzare un algoritmo di apprendimento supervisionato o non supervisionato, per ottenere rappresentazioni vettoriali delle parole. Nel modello vengono apprese le rappresentazioni delle parole per vettori di n-grammi. Questo aiuta a comprendere i suffissi e prefissi delle parole, addestrando un modello per apprendere gli incorporamenti.



Figura 3.1: Logo FastText

FastText offre due modelli per le rappresentazioni delle parole:

- **Skipgram:**
Il modello impara a prevedere una parola target grazie ad una parola vicina.
- **Cbow:**
Il modello predice la parola d'interesse in base al suo contesto (insieme di parole contenute in una finestra di dimensione fissa intorno alla parola target).

Per l'addestramento del modello utilizzato in questo progetto è stato scelto il modello skipgram non supervisionato.

3.2 Addestramento

A partire dai file generati precedentemente sugli unigrammi, bigrammi e trigrammi sono stati addestrati tre modelli di FastText, uno per ogni divisione. Per effettuare l'addestramento è sufficiente importare il rispettivo file n-grams e invocare la funzione *train_unsupervised*, specificando nei parametri, oltre al file, il tipo di modello (cbow o skipgram) e la dimensione di embedding. Viene riportato di seguito il codice utilizzato per il train del modello basato su unigrammi (per gli altri modelli basterà cambiare il file importato).

```
1 unigram="/content/drive/MyDrive/Progetto_Advanced_Cybersecurity/
           Divisione DNS/dns-unigram.log"
2 uni_model=fasttext.train_unsupervised(input=unigram, dim=128,
                                         model='skipgram')
3 uni_model.save_model("/content/drive/MyDrive/Progetto_Advanced
           Cybersecurity/unigram_model.bin")
```

Listing 3.1: Train del modello basato su unigrammi

Ottenuti i tre modelli i vari valori ottenuti sono stati salvati, utilizzando il codice fornito dagli sviluppatori di Fasttext, in formato *vec* il quale sarà successivamente utilizzato nell'addestramento della rete neurale. Di seguito riportiamo la trasformazione del modello basato su unigrammi (anche in questo caso per ottenere gli altri è sufficiente cambiare il modello importato e il nome del file vec da salvare).

```
1 # original BIN model loading
2 f = fasttext.load_model("/content/drive/MyDrive/
           Progetto_Advanced_Cybersecurity/unigram_model.bin")
3 lines=[]
4
5 # get all words from model
6 words = f.get_words()
7
8 with open("/content/drive/MyDrive/Progetto_Advanced_Cybersecurity/
           dga-mixed-embeddings-ensemble-incicco_emanuele/Code/vec/char.
           vec", 'w') as file_out:
9
10    # the first line must contain number of total words and vector
       dimension
11    file_out.write(str(len(words)) + " " + str(f.get_dimension()))
12    + "\n")
13
14    # line by line, you append vectors to VEC file
15    for w in words:
16        v = f.get_word_vector(w)
```

```

16         vstr = ""
17         for vi in v:
18             vstr += " " + str(vi)
19         try:
20             file_out.write(w + vstr + '\n')
21         except:
22             pass

```

Listing 3.2: Trasformazione del modello unigram da formato bin in formato vec

3.3 Tempi Computazionali

In questa sezione sono riportati i tempi computazionali di addestramento non supervisionato dei tre modelli Fasttext dei dati estratti dai file GARR. Non disponendo di una versione PRO di Colab i tempi di addestramento dei modelli bigram e trigram sono stati sufficientemente estesi.

Modello	Durata Esecuzione
1-gram	~ 15 minuti
2-gram	~ 2 ore
3-gram	~ 3 ore

Tabella 3.1: Tempi di addestramento FastText

Capitolo 4

Training

4.1 Dataset UMUDGA

Per la seconda parte del progetto si è utilizzato il dataset UMUDGA, differente rispetto a quello creato estraendo i log dalla rete GARR. Tale dataset è composto da 51 classi bilanciate di cui 50 rappresentano diverse tipologie di DGA mentre la restante rappresenta nomi di dominio legittimi. Al fine di utilizzare il dataset UMUDGA, l'addestramento dell'architettura è avvenuto mediante il file "*umudga_1k.csv*" presente nel repository il quale è composto da 1000 record per ciascuna classe rappresentata. 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.

```
1 dga,alureon,qkdchxxxxncom,qkdchxxxxn.com,q k d c h x x z n c o m,qk kd dc ch hx xx xx xz zn nc co om,qkd kdc dch chx hxx xxx xxz xzn znc nco com,q kdc h xxx zn com
2 dga,alureon,meojyutuspcom,meojyutusp.com,m e o j y t u s p s c o m,me eo oj jy yt tu us sp ps sc co om,meo eojo oyj jyt ytus usp sps psc sco com,me o jy t usps com
3 dga,alureon,hpsrtdskncom,hpsrtdskn.com,h p s r t d s k n c o m,hps ps sr utd tds dsk skn nco com,h psr utd skn com
4 dga,alureon,rvtajkslzcom,rvtajkslz.com,r v j t a j k s l z c o m,rvj vt jt a ej jk ks slz zc co om,rvj vtjt aej jkks slz lzc zco com,rvj t ajk ks lzc com
5 dga,alureon,nadytmalakecom,nadytmalake.com,a d y t m l a k e c o m,na ady yt tm ml ake ec co om,nad dyt ytm tlml ake kec eco com,nad y tm lake com
6 dga,alureon,pixmrgalvucom,pixmrgalvu.com,p x g q a l v u c o m,pxg xm grq qa al lv vu uc co om,pxg xgm mnrq rqa qal alv luv uvc uco com,px gmri q al vu com
7 dga,alureon,fdejqplmavcom,fdejqplmav.com,f d e j q l p m a v c o m,f d f d q e j q l p m pm ma av vc co om,fdej ejq jql qlp lpm pma mav avc vco com,fdej qjl pm av com
8 dga,alureon,lgqepktzvycom,lgqepktzvy.com,l g q e p k t z v y c o m,lg gq qe epk kt tz zv vy yc co om,lgq gae qep epk ktz tzv zvy vyc yco com,lg qep k tz vy com
9 dga,alureon,ulifmduiccom,ulifmduic.com,u l i f m x d u i c o m,uil li if fm mx xd du ui ic cc co om,ulif lif ifm fmxd mxdu dui uic icc cco com,ulif fm xd uic com
10 dga,alureon,dnkxqgjbfqcom,dnkxqgjbfq.com,d n k x q g j b f q c o m,dn k x q g j b f q c q co om,dnkx xqg qjg gjb bfq fqg qco com,d nk x qg j bf q com
11 dga,alureon,fsgrdrprsmcom,fsgrdrprsm.com,f s g r d r p r s m c o m,fs sg gn rd rp pr rs sm co om,fsgr grd rdn drp rpr rps rsm smc mco com,f sgn dr prs m com
12 dga,alureon,alififyhqilcom,a l i f i f y q h q l c o m,a l if fi if fy qh q l c o m,alif ifi fit ify fy qh qh qhl qlc lco com,a if if y qh ql com
13 dga,alureon,wapnzdolefcom,wapnzdolef.com,w q p n z d o l e f c o m,wq ap pn nz zd do l e ef fc co om,wap apr pnz zd do l oie lef efcc fco com,w p nz doi ef com
14 dga,alureon,wmncfyxcwccom,wmncfyxcwcc.com,w m n c f y x c w c c o m,wm mn nc cf fy yx xc cw wc cc co om,wmn mnc mfc cfy fyx xcw cwc cco com,wm nc fy x cwc com
15 dga,alureon,i j g h x l x r x u c o m,i j g j g h x l x r x u x u c o m,i j g j g h x l x r x u x u c o m,i j g j g h x l x r x u x u c o m,i j g h xl x r xu com
16 dga,alureon,wuejhyhzgcom,wuejhyhzg.com,w u e j h y h z h g c o m,wu ue ej hy yh zh hg gc co om,wuej ejh ejh hyh yhz hzh hg cco com,uej ejy hy hz hg com
17 dga,alureon,kkbqdrbkhcom,kkbqdrbkh.com,k k b q d r b k h c o m,kkk kb bq dr rbk bi in hc co om,kkb kbq bdq qdr drb rk bih lhc hco com,k kb qd rb kh com
18 dga,alureon,tgmwzfzgkdcncom,tgmwzfzgkdcn.com,t g m w z f z g k d c o m,tg gmw wz fz fg zgk gkd kdc dco com,tg mwz f z g kdc om
19 dga,alureon,hvxlylyhjcom,hvxlylyhj.com,h v x y l l y h r j c o m,hvx ly yl l l y ly yh rj jc co om,hvx vxy xyl lly lyh yhr hrj rjc jco com,hv xy ll y hr j com
20 dga,alureon,ijimsdyctwcom,ijimsdyctw.com,i j i m s d y c t w c o m,ij i ji im ms sd dy ct tw wc co om,iji jim ims msd sdy dyc ctw twc wco com,i j i m s d y c t w com
21 dga,alureon,aozmvpnfpfcom,aozmvpnfpf.com,a o z m v h p n f z c o m,ao oz zm vp hn fp nf fz zc co om,aoz ozm zm vhp hn fpnf nfz fzcc zco com,a oz mv hp nf z com
22 dga,alureon,msrpneurccom,msrpneurcc.com,m s r p x n e u r c c o m,m s r p x n e u r c c o m,msr srpx pxn xne neu eur urc rec cco com,m sr px ne urc com
23 dga,alureon,vrikphluaucom,vrikphluau.com,v r i k p h l u a u c o m,vn ri ik kp ph l u ua au uc co om,vri i kkp phl hilua uau auc uco com,vr i kph luau com
24 dga,alureon,vkvffgumcom,vkvffgum.com,v k v h f f g u m r c o m,vk kv h ff fg gu um mr co om,vkv kvh vhf hff ffg guu gum umm mrc rco com,v k vnf f gum r com
25 dga,alureon,uibwkgliecom,uibwkglie.com,u i b w k q l i o e c o m,uil i bw wk q l i l o oe ec co om,uib bwk wkq kq l i l o oe oec eco com,u i b w k q l i o com
26 dga,alureon,vqlwkcenrcscom,vqlwkcenrcs.com,v q l w k c e n r s c o m,vql wlk wkc kce cen enr nrs rsc sco com,v ql w kc en rs com
27 dga,alureon,cgvkfcsstacom,cgvkfcsstacom,c g v k f c s g t a c o m,cg gv vk kf fc cs sg gt ta ac co om,cgvk gvfk kfcc fcs csg sgt gta tac aco com,cg v kf c sgt a com
28 dga,alureon,ebedacsuylcom,ebedacsuyl.com,e b e d a c s u y l c o m,e b e d da ac cs su uy yl lc co om,ebe bed eda dac acs csu suy uyl ylc co com,e bed a csu yl com
29 dga,alureon,mtkvjenbljcom,mtkvjenblj.com,m t k v j e n b l j c o m,mt tkv vje en nb bl l j jc co om,mtk tkv kvj vje jen enb nbl blj ljc jco com,mt kv jen bl j com
30 dga,alureon,qmxlcymificom,qmxlcymificom,q m x l c y m i f l c o m,qm mx xl lc cy ym i f l flic lco com,qm mx xl lc cy ym i f l flic lco com,qm xl cy mi fl com
```

Figura 4.1: Esempio di contenuto del file "*umudga_1k.csv*"

4.2 Architettura Stacked e scelta dei tagli

Per il training si è utilizzata un’architettura di tipo Stacked e quattro diverse percentuali di training. Per come è stata definita, l’architettura Stacking è composta di due fasi di allenamento, una interna e una esterna.

Nella fase interna si addestrano i modelli individuali che compongono l’architettura Stacking, si effettuano poi le predizioni di questi modelli così da costruire un “dataset” su cui allenare il meta-modello dell’architettura Stacking (di default il meta-modello è una regressione logistica). Questo processo viene fatto per 5 volte.

Nella fase esterna si addestrano i modelli individuali (questa volta sull’insieme dei dati che costituiscono il fold “principale”), si effettuano poi le predizioni e si utilizza il meta-modello (addestrato precedentemente nella fase interna) per effettuare la predizione finale dell’architettura Stacked. Ovvero, si effettuano le predizioni con i modelli singoli, questi risultati vengono passati al meta-modello e la predizione di quest’ultimo è la predizione finale del modello Stacking.

Come si può osservare dalla figura 4.2, vengono utilizzati 5 modelli: i 3 modelli ottenuti con FastText, 1 modello random-multiclasse e 1 modello ELMo-multiclasse. Le predizioni ottenute da questi modelli vengono poi utilizzate dal modello di regressione logistica, determinando così l’output finale.

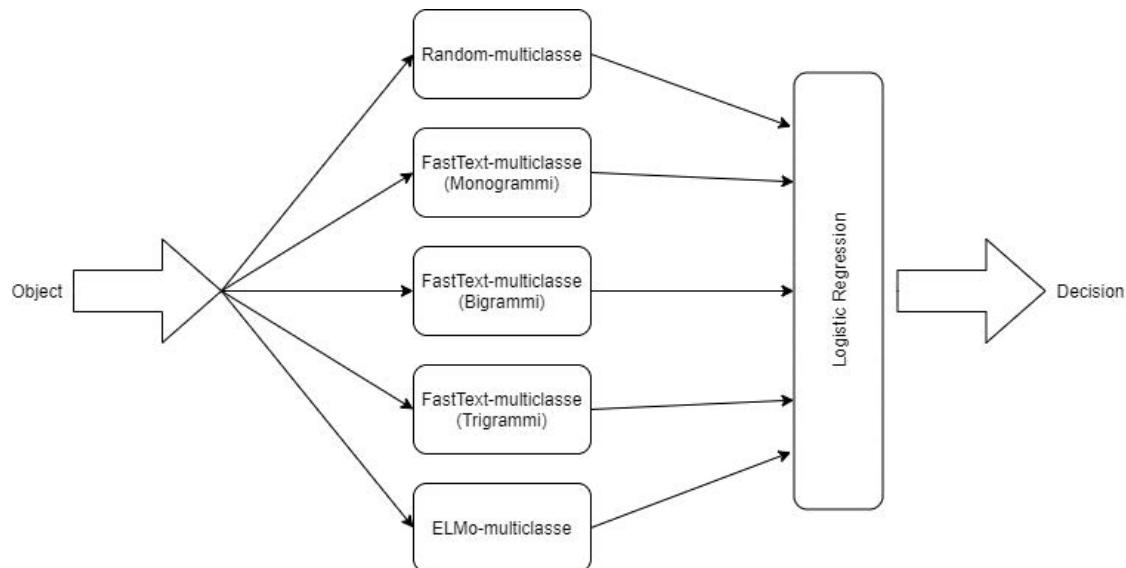


Figura 4.2: Modello Stacked

La rete può essere addestrata lanciando il seguente comando:

```
!python3 Main.py  
--dataset-path ../Dataset/umudga_1k.csv  
--output-path '/content/drive/MyDrive/  
Progetto_Advanced_Cybersecurity/Reports/Report/<taglio>'  
--model Stacking --epochs 20 --with-training --train-perc <taglio>
```

Dove <taglio> deve essere sostituito con il valore numerico della percentuale di training desiderata per la fase di addestramento (nel nostro caso sono stati utilizzati tagli crescenti da 0.017 a 0.135). Per quanto riguarda il tempo di esecuzione dei diversi addestramenti non disponendo di una versione PRO di Colab è stato necessario addestrare i modelli in più esecuzioni riprendendo l'addestramento da dove si era interrotto grazie ai due comandi `-starting-ext-fold-nr` e `-starting-int-fold-nr`. Si è comunque fatta una stima del tempo di esecuzione degli addestramenti riportata nella seguente tabella.

Percentuale di Train	Durata Esecuzione
0.017	~ 2,5 ore
0.034	~ 3,5 ore
0.068	~ 4,5 ore
0.135	~ 6 ore

Tabella 4.1: Tempi di Addestramento architettura Stacked

Capitolo 5

Risultati

5.1 Taglio 0.017

Una volta finito l'addestramento la fase di test utilizza i restanti campioni del dataset non utilizzati per effettuare delle predizioni e calcolare diverse metriche utili a valutare il corretto funzionamento della rete. I vari test sono effettuati per ognuno dei 5 fold e poi successivamente raggruppati in un unico report di cui riportiamo le metriche sia per quanto riguarda le micro che le macro (5.1). Inoltre in una seconda tabella riportiamo le metriche per quanto riguarda l'accuratezza del modello (5.2).

Metrica	Macro	Micro
Precision	0.586	0.595
Recall	0.595	0.595
F1-Score	0.560	0.595

Tabella 5.1: Metriche per taglio di train di 0.017

Accuracy	
Macro avg	0.586
Weighted avg	0.586
Overall	0.595

Tabella 5.2: Metriche di accuratezza per taglio 0.017

5.1.1 Matrice di Confusione

In questa sezione è riportata la matrice di confusione in modo da avere una visualizzazione migliore dei risultati della classificazione (Fig. 5.1).

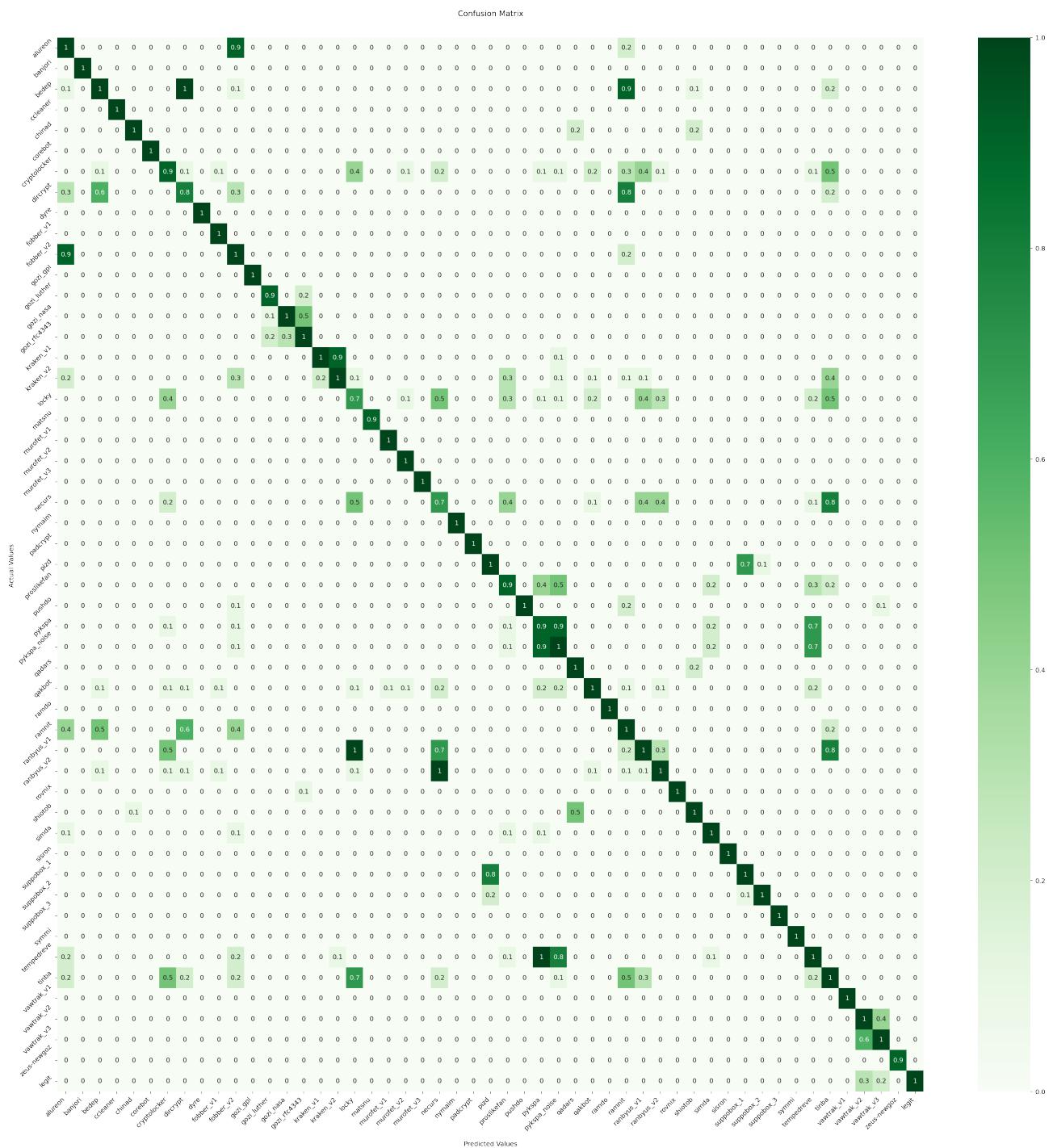


Figura 5.1: Matrice di confusione per taglio di train 0.017

5.1.2 Risultati generali classificazione - Percentuale 0.017

La seguente tabella (5.3) mostra le metriche di precision, recall e F1-score per ogni classe ed il relativo supporto utilizzato nella fase di test.

	Precision	Recall	F1-score	Support
alureon	0.23	0.61	0.33	983
banjori	1	1	1	983
bedep	0.34	0.53	0.38	983
ccleaner	0.99	1	0.99	983
chinad	0.84	0.65	0.73	983
corebot	1	0.97	0.98	983
cryptolocker	0.29	0.04	0.07	983
dircrypt	0.22	0.15	0.14	983
dyre	1	1	1	983
fobber_v1	0.54	0.96	0.69	983
fobber_v2	0.19	0.26	0.2	983
gozi_gpl	0.83	0.91	0.87	983
gozi_luther	0.69	0.77	0.72	983
gozi_nasa	0.61	0.35	0.43	983
gozi_rfc4343	0.49	0.54	0.51	983
kraken_v1	0.69	0.68	0.68	983
kraken_v2	0.37	0.14	0.19	983
locky	0.17	0.16	0.15	983
matsnu	0.77	0.95	0.85	983
murofet_v1	0.8	0.97	0.87	983
murofet_v2	0.58	0.98	0.73	983
murofet_v3	1	0.98	0.99	983
necurs	0.2	0.07	0.07	983
nymaim	0.83	0.72	0.77	983
padcrypt	0.89	0.96	0.92	983
pizd	0.46	0.58	0.51	983
proslikefan	0.35	0.33	0.33	983
pushdo	0.67	0.65	0.66	983
pykspa	0.19	0.12	0.13	983
pykspa_noise	0.23	0.22	0.17	983
qadars	0.54	0.81	0.64	983
qakbot	0.44	0.03	0.06	983
ramdo	0.8	0.98	0.88	983
ramnit	0.18	0.09	0.12	983

	Precision	Recall	F1-score	Support
ranbyus_v1	0.34	0.21	0.24	983
ranbyus_v2	0.41	0.45	0.38	983
rovnix	0.76	0.73	0.74	983
shiotob	0.56	0.24	0.3	983
simda	0.48	0.66	0.55	983
sisron	0.95	0.98	0.97	983
suppobox_1	0.51	0.43	0.45	983
suppobox_2	0.7	0.8	0.74	983
suppobox_3	0.91	0.97	0.94	983
symmi	0.96	1	0.98	983
tempedreve	0.25	0.16	0.17	983
tinba	0.1	0.04	0.05	983
vawtrak_v1	0.91	1	0.95	983
vawtrak_v2	0.42	0.73	0.5	983
vawtrak_v3	0.47	0.51	0.46	983
zeus-newgoz	0.99	0.98	0.98	983
legit	0.73	0.3	0.42	983

Tabella 5.3: Metriche per taglio di train 0.017

5.2 Taglio 0.034

Una volta finito l’addestramento la fase di test utilizza i restanti campioni del dataset non utilizzati per effettuare delle predizioni e calcolare diverse metriche utili a valutare il corretto funzionamento della rete. I vari test sono effettuati per ognuno dei 5 fold e poi successivamente raggruppati in un unico report di cui riportiamo le metriche sia per quanto riguarda le micro che le macro (5.4). Inoltre in una seconda tabella riportiamo le metriche per quanto riguarda l’accuratezza del modello (5.5).

Metrica	Macro	Micro
Precision	0.671	0.677
Recall	0.677	0.677
F1-Score	0.661	0.677

Tabella 5.4: Metriche per taglio di train di 0.034

Accuracy	
Macro avg	0.671
Weighted avg	0.671
Overall	0.677

Tabella 5.5: Metriche di accuratezza per taglio 0.034

5.2.1 Matrice di Confusione

In questa sezione è riportata la matrice di confusione in modo da avere una visualizzazione migliore dei risultati della classificazione (Fig. 5.2).

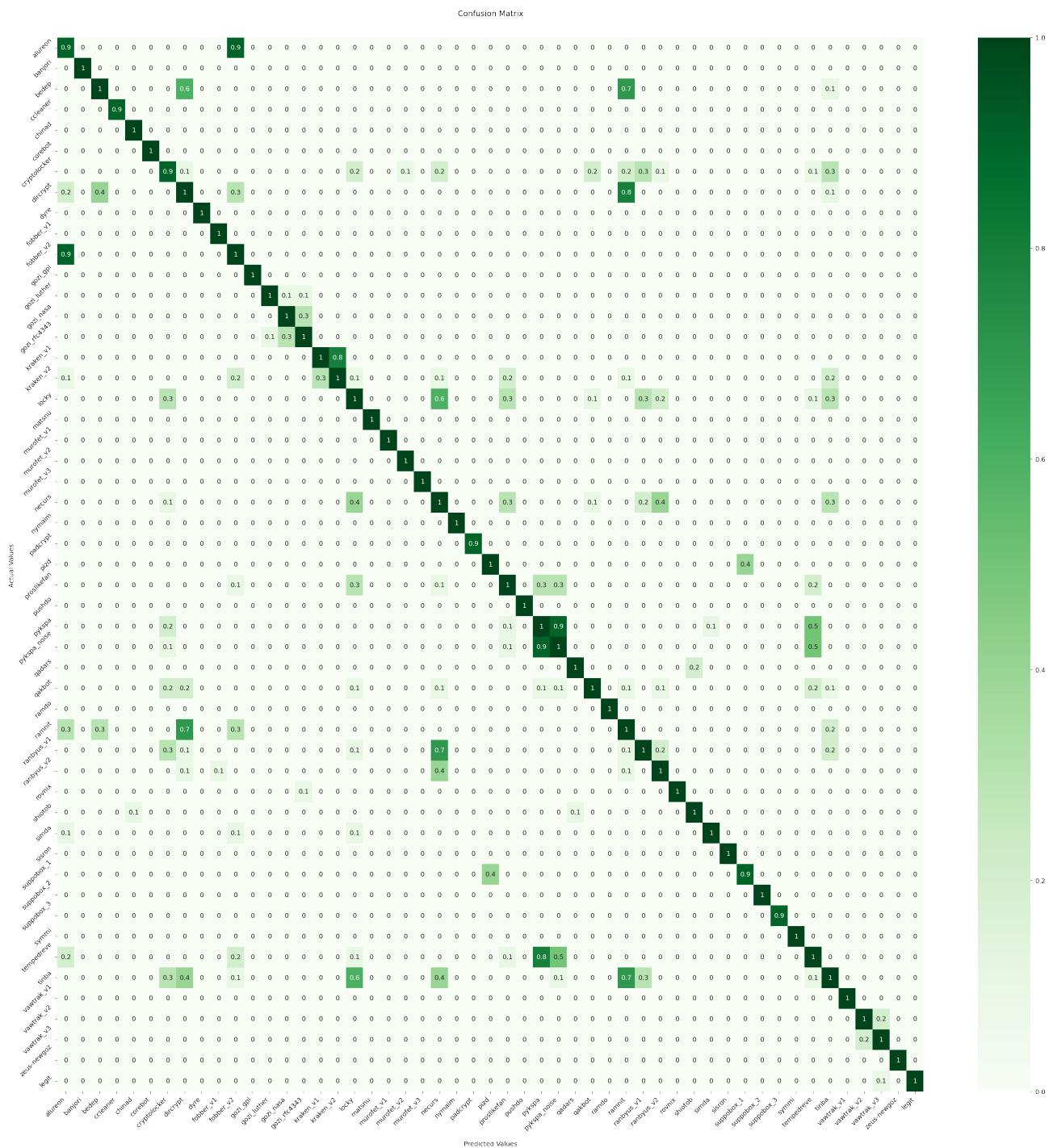


Figura 5.2: Matrice di confusione per taglio di train 0.034

5.2.2 Risultati generali classificazione - Percentuale 0.034

La seguente tabella (5.6) mostra le metriche di precision, recall e F1-score per ogni classe.

	Precision	Recall	F1-score	Support
alureon	0.28	0.54	0.37	966
banjori	1	1	1	966
bedep	0.47	0.63	0.54	966
ccleaner	0.99	1	1	966
chinad	0.86	0.91	0.88	966
corebot	1	0.98	0.99	966
cryptolocker	0.33	0.17	0.22	966
dircrypt	0.24	0.16	0.18	966
dyre	1	1	1	966
fobber_v1	0.71	0.96	0.82	966
fobber_v2	0.26	0.34	0.29	966
gozi_gpl	0.9	0.95	0.93	966
gozi_luther	0.81	0.8	0.8	966
gozi_nasa	0.65	0.67	0.65	966
gozi_rfc4343	0.6	0.64	0.61	966
kraken_v1	0.68	0.76	0.71	966
kraken_v2	0.36	0.22	0.27	966
locky	0.28	0.13	0.17	966
matsnu	0.92	0.93	0.93	966
murofet_v1	0.93	0.98	0.95	966
murofet_v2	0.73	0.98	0.84	966
murofet_v3	1	0.99	0.99	966
necurs	0.24	0.13	0.16	966
nymaim	0.89	0.84	0.86	966
padcrypt	0.95	0.98	0.96	966
pizd	0.66	0.6	0.63	966
proslikefan	0.4	0.48	0.43	966
pushdo	0.77	0.87	0.81	966
pykspa	0.25	0.21	0.19	966
pykspa_noise	0.29	0.2	0.22	966
qadars	0.88	0.73	0.79	966
qakbot	0.58	0.25	0.33	966
ramdo	0.91	0.99	0.95	966
ramnit	0.21	0.19	0.19	966

	Precision	Recall	F1-score	Support
ranbyus_v1	0.41	0.43	0.42	966
ranbyus_v2	0.42	0.64	0.49	966
rovnix	0.94	0.77	0.84	966
shiotob	0.71	0.61	0.65	966
simda	0.71	0.73	0.72	966
sisron	1	0.99	0.99	966
suppobox_1	0.67	0.7	0.68	966
suppobox_2	0.79	0.91	0.85	966
suppobox_3	0.95	0.99	0.97	966
symmi	0.99	1	0.99	966
tempedreve	0.32	0.32	0.3	966
tinba	0.3	0.12	0.15	966
vawtrak_v1	0.98	1	0.99	966
vawtrak_v2	0.69	0.72	0.7	966
vawtrak_v3	0.63	0.78	0.69	966
zeus-newgoz	1	0.99	0.99	966
legit	0.71	0.59	0.64	966

Tabella 5.6: Metriche per taglio di train 0.034

5.3 Taglio 0.068

Una volta finito l’addestramento la fase di test utilizza i restanti campioni del dataset non utilizzati per effettuare delle predizioni e calcolare diverse metriche utili a valutare il corretto funzionamento della rete. I vari test sono effettuati per ognuno dei 5 fold e poi successivamente raggruppati in un unico report di cui riportiamo le metriche sia per quanto riguarda le micro che le macro (5.7). Inoltre in una seconda tabella riportiamo le metriche per quanto riguarda l’accuratezza del modello (5.8).

Metrica	Macro	Micro
Precision	0.734	0.728
Recall	0.728	0.728
F1-Score	0.723	0.728

Tabella 5.7: Metriche per taglio di train di 0.068

Accuracy	
Macro avg	0.734
Weighted avg	0.734
Overall	0.728

Tabella 5.8: Metriche di accuratezza per taglio 0.068

5.3.1 Matrice di Confusione

In questa sezione è riportata la matrice di confusione in modo da avere una visualizzazione migliore dei risultati della classificazione (Fig. 5.3).

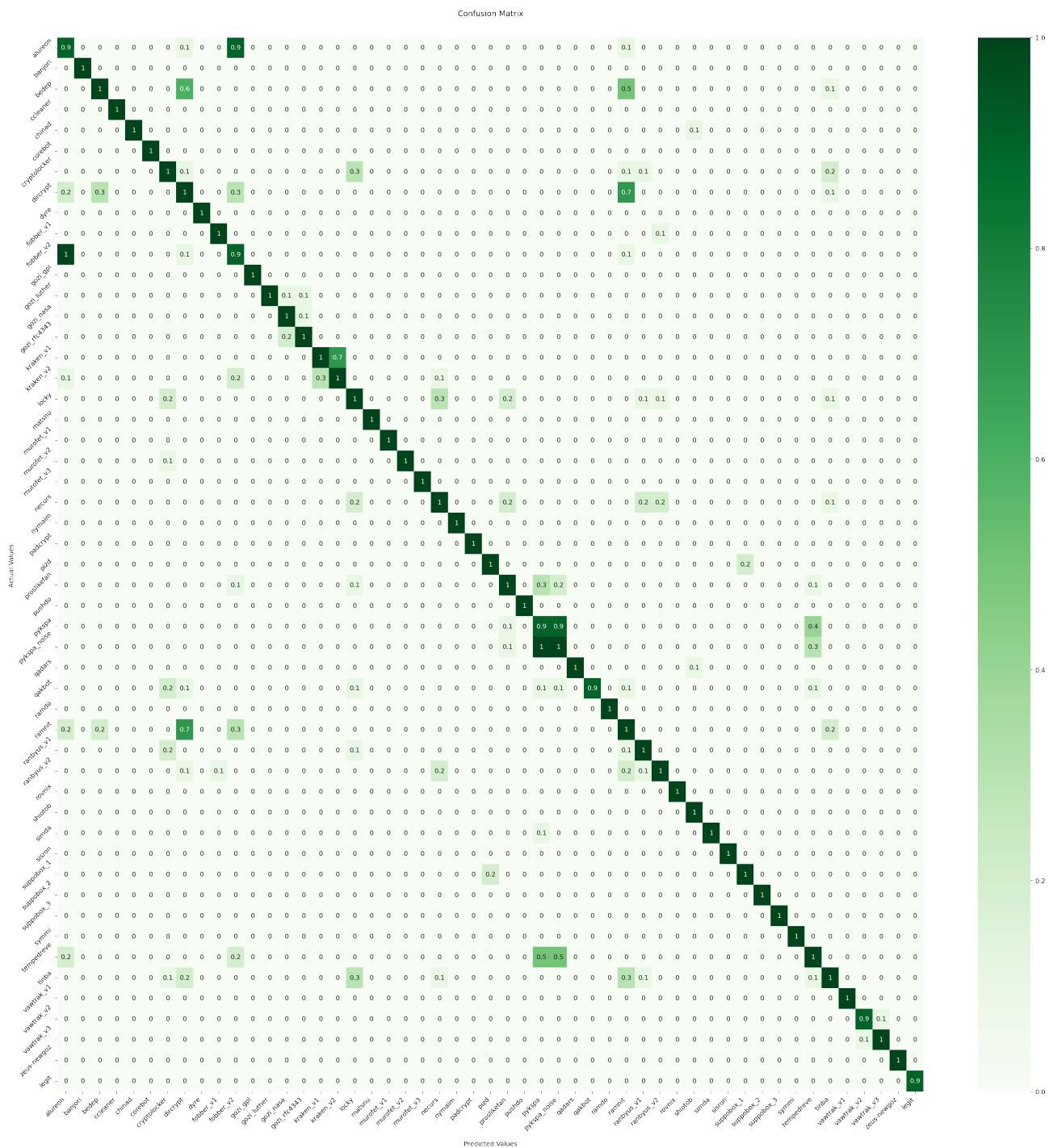


Figura 5.3: Matrice di confusione per taglio di train 0.068

5.3.2 Risultati generali classificazione - Percentuale 0.068

La seguente tabella (5.9) mostra le metriche di precision, recall e F1-score per ogni classe.

	Precision	Recall	F1-score	Support
alureon	0.29	0.43	0.33	932
banjori	1	1	1	932
bedep	0.54	0.62	0.57	932
ccleaner	0.99	1	1	932
chinad	0.95	0.88	0.91	932
corebot	1	0.99	1	932
cryptolocker	0.46	0.3	0.36	932
dircrypt	0.28	0.26	0.26	932
dyre	1	1	1	932
fobber_v1	0.8	0.88	0.83	932
fobber_v2	0.27	0.36	0.3	932
gozi_gpl	0.94	0.95	0.95	932
gozi_luther	0.89	0.79	0.83	932
gozi_nasa	0.69	0.81	0.74	932
gozi_rfc4343	0.71	0.71	0.71	932
kraken_v1	0.75	0.7	0.71	932
kraken_v2	0.46	0.4	0.42	932
locky	0.38	0.31	0.34	932
matsnu	0.94	0.94	0.94	932
murofet_v1	0.98	0.99	0.98	932
murofet_v2	0.83	0.94	0.88	932
murofet_v3	1	0.99	1	932
necurs	0.49	0.27	0.34	932
nymaim	0.91	0.86	0.88	932
padcrypt	0.98	0.98	0.98	932
pizd	0.72	0.79	0.75	932
proslikefan	0.48	0.55	0.51	932
pushdo	0.85	0.92	0.88	932
pykspa	0.26	0.28	0.27	932
pykspa_noise	0.3	0.18	0.22	932
qadars	0.91	0.83	0.87	932
qakbot	0.63	0.39	0.48	932
ramdo	0.96	0.99	0.97	932
ramnit	0.24	0.23	0.23	932

	Precision	Recall	F1-score	Support
ranbyus_v1	0.53	0.7	0.59	932
ranbyus_v2	0.62	0.56	0.57	932
rovnix	0.94	0.88	0.91	932
shiotob	0.77	0.76	0.76	932
simda	0.81	0.83	0.81	932
sisron	1	0.99	1	932
suppobox_1	0.8	0.76	0.77	932
suppobox_2	0.9	0.94	0.92	932
suppobox_3	0.99	0.99	0.99	932
symmi	0.99	1	1	932
tempedreve	0.39	0.45	0.41	932
tinba	0.43	0.33	0.35	932
vawtrak_v1	0.99	1	0.99	932
vawtrak_v2	0.83	0.83	0.82	932
vawtrak_v3	0.75	0.88	0.81	932
zeus-newgoz	1	0.99	1	932
legit	0.75	0.68	0.71	932

Tabella 5.9: Metriche per taglio di train 0.068

5.4 Taglio 0.135

Una volta finito l’addestramento la fase di test utilizza i restanti campioni del dataset non utilizzati per effettuare delle predizioni e calcolare diverse metriche utili a valutare il corretto funzionamento della rete. I vari test sono effettuati per ognuno dei 5 fold e poi successivamente raggruppati in un unico report di cui riportiamo le metriche sia per quanto riguarda le micro che le macro (5.10). Inoltre in una seconda tabella riportiamo le metriche per quanto riguarda l’accuratezza del modello (5.11).

Metrica	Macro	Micro
Precision	0.789	0.784
Recall	0.784	0.784
F1-Score	0.781	0.784

Tabella 5.10: Metriche per taglio di train di 0.135

Accuracy	
Macro avg	0.788
Weighted avg	0.788
Overall	0.784

Tabella 5.11: Metriche di accuratezza per taglio 0.135

5.4.1 Matrice di Confusione

In questa sezione è riportata la matrice di confusione in modo da avere una visualizzazione migliore dei risultati della classificazione (Fig. 5.4).

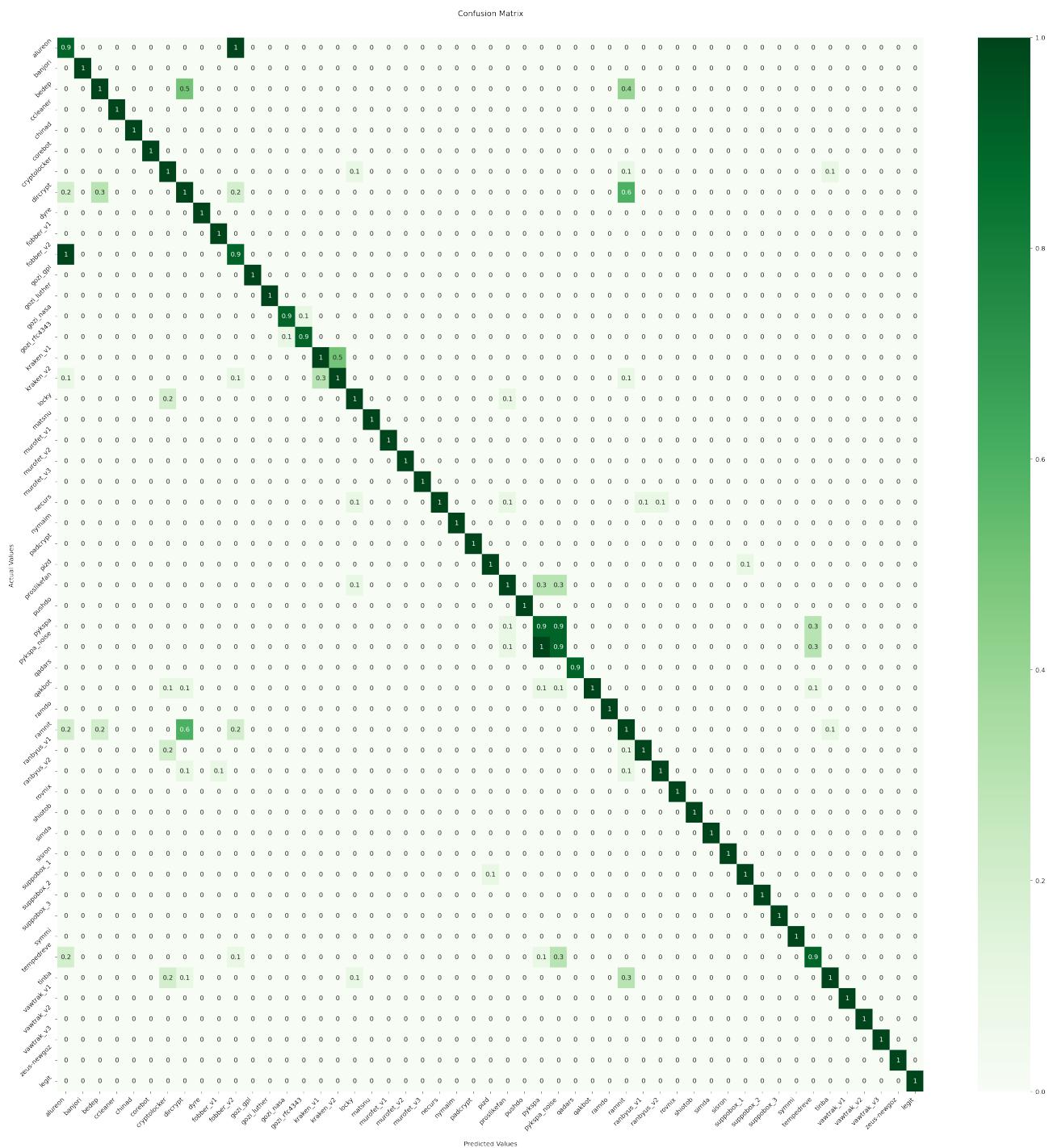


Figura 5.4: Matrice di confusione per taglio di train 0.135

5.4.2 Risultati generali classificazione - Percentuale 0.135

La seguente tabella (5.12) mostra le metriche di precision, recall e F1-score per ogni classe.

	Precision	Recall	F1-score	Support
alureon	0.32	0.49	0.38	865
banjori	1	1	1	865
bedep	0.56	0.66	0.6	865
ccleaner	1	1	1	865
chinad	0.96	0.95	0.96	865
corebot	1	0.99	1	865
cryptolocker	0.46	0.53	0.48	865
dircrypt	0.33	0.27	0.29	865
dyre	1	1	1	865
fobber_v1	0.82	0.98	0.89	865
fobber_v2	0.33	0.42	0.37	865
gozi_gpl	0.96	0.96	0.96	865
gozi_luther	0.9	0.9	0.9	865
gozi_nasa	0.81	0.78	0.79	865
gozi_rfc4343	0.74	0.83	0.78	865
kraken_v1	0.73	0.77	0.75	865
kraken_v2	0.57	0.44	0.49	865
locky	0.57	0.48	0.52	865
matsnu	0.95	0.95	0.95	865
murofet_v1	0.99	0.99	0.99	865
murofet_v2	0.85	0.97	0.91	865
murofet_v3	1	1	1	865
necurs	0.76	0.47	0.57	865
nymaim	0.94	0.89	0.91	865
padcrypt	0.99	0.99	0.99	865
pizd	0.85	0.89	0.87	865
proslikefan	0.58	0.52	0.55	865
pushdo	0.92	0.96	0.94	865
pykspa	0.32	0.25	0.28	865
pykspa_noise	0.31	0.27	0.28	865
qadars	0.93	0.91	0.92	865
qakbot	0.75	0.43	0.54	865
ramdo	0.98	1	0.99	865
ramnit	0.3	0.22	0.24	865

	Precision	Recall	F1-score	Support
ranbyus_v1	0.73	0.76	0.74	865
ranbyus_v2	0.77	0.71	0.74	865
rovnix	0.94	0.91	0.93	865
shiotob	0.88	0.79	0.83	865
simda	0.89	0.95	0.92	865
sisron	1	1	1	865
suppobox_1	0.89	0.89	0.89	865
suppobox_2	0.96	0.97	0.96	865
suppobox_3	0.99	1	0.99	865
symmi	0.99	1	1	865
tempedreve	0.5	0.62	0.55	865
tinba	0.57	0.56	0.54	865
vawtrak_v1	0.99	1	1	865
vawtrak_v2	0.95	0.96	0.96	865
vawtrak_v3	0.89	0.98	0.93	865
zeus-newgoz	1	1	1	865
legit	0.81	0.74	0.77	865

Tabella 5.12: Metriche per taglio di train 0.135

5.5 Confronto risultati architettura

5.5.1 Risultati classificazione

Nella tabella seguente (5.13) si possono confrontare i valori delle metriche (di cui riportiamo solo i macro e l'overall per l'accuratezza) ottenuti con l'architettura addestrata precedentemente su ogni singolo taglio.

Taglio	Accuracy	Precision	Recall	F1-score
0.017	0.595	0.586	0.595	0.560
0.034	0.677	0.671	0.677	0.661
0.068	0.728	0.734	0.728	0.723
0.135	0.784	0.789	0.784	0.781

Tabella 5.13: Risultati metriche di classificazione

Ponendo particolare attenzione sulle metriche riportate in tabella , è possibile osservare come, per i tagli più grandi (0.068 e 0.135) si ottengono risultati migliori con l'architettura Stacked. Invece, per le altre percentuali di train, si ottengono valori leggermente differenti.

Capitolo 6

Conclusioni e sviluppi futuri

In conclusione, grazie al lavoro realizzato è stato possibile addestrare e testare l'architettura Stacked al fine di effettuare una classificazione multi-classe per i vari nomi di dominio appartenenti al dataset UMUDGA. Dalla tabella 5.13 possiamo analizzare i risultati ottenuti e vediamo come l'architettura si comporta meglio nei casi in cui si hanno a disposizione più nomi di dominio, riuscendo ad ottenere risultati migliori utilizzando percentuali di training maggiori quindi in dei possibili test futuri si potrebbe addestrare l'architettura utilizzando un dataset più ampio come ad esempio il *umudga_5k.csv* (il quale è composto da 5000 record per classe) presente nel repository.

Inoltre, un' altro dei possibili sviluppi futuri potrebbe essere quello di adottare un'architettura differente, confrontando i risultati con quelli ottenuto in questo progetto.

Bibliografia

- [1] FastText. Word representations, <https://fasttext.cc/docs/en/unsupervised-tutorial.html>
- [2] Repository github del progetto:
github.com/MassimoCiaffoni/Progetto_Advanced_Cybersecurity
- [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.
<https://www.sciencedirect.com/science/article/pii/S0167404820300067>.