UNIVERZITET CRNE GORE
ELEKTROTEHNIČKI FAKULTET PODGORICA

Nedeljko Nikolić

ANALIZA PERFORMANSI TEHNIKA KLASIFIKACIJE MREŽNOG SAOBRAĆAJA BAZIRANIH NA DUBOKOM UČENJU

MAGISTARSKI RAD

Podgorica, 2022.

PODACI I INFORMACIJE O MAGISTRANTU

Ime i prezime: **Nedeljko Nikolić**
Datum i mjesto rođenja: 26.02.1984, Podgorica
Prethodno završene studije:

Elektrotehnički fakultet, osnovne primijenjene studije (180 ECTS kredita), studijski program: Studije primijenjenog računarstva, 2011.

Elektrotehnički fakultet, specijalističke primijenjene studije (60 ECTS kredita), studijski program: Studije primijenjenog računarstva, 2012.

INFORMACIJE O MAGISTARSKOM RADU

Elektrotehnički fakultet
Studijski program: Studije primijenjenog računarstva
Naslov rada: ANALIZA PERFORMANSI TEHNIKA KLASIFIKACIJE MREŽNOG SAOBRAĆAJA BAZIRANIH NA DUBOKOM UČENJU
Mentor: Prof. dr Igor Radusinović

UDK, OCJENA I ODBRANA MAGISTARSKOG RADA

Datum prijave magistarskog rada: 21.01.2022.
Datum sjednice Vijeća na kojoj je prihvaćena tema: 05.05.2022.

Komisija za ocjenu teme i podobnosti magistranta:
1. Prof. dr Vesna Rubežić
2. Prof. dr Igor Radusinović
3. Prof. dr Žarko Zečević

Komisija za ocjenu rada:
1. Prof. dr Vesna Rubežić
2. Prof. dr Igor Radusinović
3. Prof. dr Žarko Zečević

Komisija za odbranu rada:
1. Prof. dr Vesna Rubežić
2. Prof. dr Igor Radusinović
3. Prof. dr Žarko Zečević

Datum odbrane: _____
Datum promocije:_____

Ime i prezime autora: Nedeljko Nikolić, Spec. App

E T I Č K A I Z J A V A

U skladu sa članom 22 Zakona o akademskom integritetu i članom 24 Pravila studiranja na postdiplomskim studijama, pod krivičnom i materijalnom odgovornošću, izjavljujem da je magistarski rad pod naslovom

" Analiza performansi tehnika klasifikacije mrežnog saobraćaja baziranih na dubokom učenju "

moje originalno djelo.

Podnosilac izjave,

Nedeljko Nikolić, Spec. App

Nedeljko N

U Podgorici, dana 16.12.2022. godine

Predgovor

U svijetu računarskih mreža konstantno se javljaju novi izazovi u vezi sa klasifikacijom i sigurnošću. Kao neko ko želi da bude u toku sa najnovijim trendovima želio sam da saznam više o ovoj temi. U ovom radu predstavio sam svoje istraživanje o primjeni tehnika dubokog učenja u klasifikaciji mrežnog saobraćaja i analizi performansi različitih modela.

Želim da se zahvalim svom mentoru, profesoru dr Igoru Radusinoviću, na ukazanoj prilici da proširim svoja znanja. Posebna zahvalnost ide dr Slavici Tomović bez čije pomoći i savjeta ovaj rad ne bi bio moguć.

Nadam se da će ovaj rad drugima pomoći u proširivanju znanja o primjeni dubokog učenja u mrežnoj sigurnosti i da će biti koristan za daljnja istraživanja u ovoj oblasti.

Tema: *Analiza performansi tehnika klasifikacije mrežnog saobraćaja baziranih na dubokom učenju*

Apstrakt:

Stalni porast Internet saobraćaja, razvoj novih Internet servisa i sve strožiji QoS (*Quality of Service*) zahtjevi ističu potrebu za razvojem naprednih alata za klasifikaciju mrežnog saobraćaja u realnom vremenu. Konvencionalne metode klasifikacije saobraćaja mrežnih aplikacija bazirane su na provjeri broja porta u zaglavlju paketa, klasifikaciji na osnovu statističkih karakteristika saobraćajnog toka ili prepoznavanju karakterističnih obrazaca (tzv. potpisa) u paketima. Međutim, mnoge aplikacije danas koriste neregistrovane ili nasumično generisane portove, statistička klasifikacija se ne može obavljati u realnom vremenu, a identifikacija i izvlačenje potpisa zahtijeva dosta vremena i radnog angažovanja. Kao rješenje za analizirani problem klasifikacije saobraćaja, u novijoj literaturi identifikovane su tehnike dubokog učenja i dosadašnja istraživanja su pokazala da ove tehnike ostvaruju obećavajuće rezultate, mogu se izvršavati u skoro realnom vremenu i ne zahtijevaju manuelnu ekstrakciju bitnih karakteristika paketa. U ovoj tezi predstavljena je implementacija i analizirane su performanse tri modela dubokog učenja za *online* klasifikaciju mrežnih aplikacija: višeslojni perceptron, složeni autoenkoder i konvolucione neuralne mreže. Za potrebe istraživanja prikupljen je *dataset* u lokalnoj računarskoj mreži pri uobičajenim operativnim uslovima. S obzirom na ograničenja u pogledu broja korisnika mreže, poseban akcenat je stavljen na fer evaluaciji modela adekvatnom predobradom *dataset*-a koja uključuje uklanjanje informacija iz zaglavlja paketa koje mogu drastično ograničiti sposobnost neuralne mreže da uči bitne obrasce, tj. mogućnost generalizacije. Rezultati istraživanja ukazali su na najperspektivnije modele neuralnih mreža i podešavanja parametara za identifikaciju saobraćaja, kao i poteškoće u identifikaciji određenih mrežnih aplikacija. Kao pomoćni alat za buduća istraživanja u ovoj oblasti razvijena je aplikacija za treniranje i testiranje modela dubokog učenja za klasifikaciju saobraćaja mrežnih aplikacija.

Ključne riječi: računarske mreže, duboko učenje, neuralne mreže, identifikacija mrežnog saobraćaja.

The theme: *Performance analysis of network traffic classification techniques based on Deep Learning*

Abstract:

The constant increase in Internet traffic, the development of new Internet services and increasingly strict QoS (Quality of Service) requirements emphasize the need for the development of advanced tools for the classification of network traffic in real-time. Conventional methods of classifying network applications are based on checking the port number in the packet header, classification based on statistical characteristics of the traffic flow, or recognizing characteristic patterns (so-called signatures) in packets. However, many applications today use unregistered or randomly generated ports, statistical classification cannot be performed in real-time, and signature identification and extraction require a lot of time and work engagement. As a solution to the analyzed problem of traffic classification, deep learning techniques have been identified in the recent literature and previous research has shown that these techniques achieve promising results, can be executed in near real-time, and do not require manual extraction of important packet characteristics. This thesis presents the implementation and analysis of the performance of three deep learning models for online classification of network applications: Multilayer Perceptron, Complex Autoencoder, and Convolutional Neural Networks. For research purposes, a dataset was collected in a local computer network under normal operating conditions. Given the limitations regarding the number of network users, special emphasis is placed on a fair evaluation of the model by adequate preprocessing of the dataset, which includes the removal of information from packet headers that can drastically limit the ability of the neural network to learn important patterns, i.e. possibility of generalization. The research results indicated the most promising neural network models and parameter settings for traffic identification, as well as difficulties in identifying certain network applications. As an auxiliary tool for future research in this area, an application was developed for training and testing deep learning models for the classification of network applications.

Keywords: computer networks, deep learning, neural network, network traffic identification.

Sadržaj:

UVOD

Internet saobraćaj se eksponencijalno povećava uslijed povećanja broja mobilnih uređaja koji koriste Internet konekciju i sve veće zastupljenosti *Internet of Things* aplikacija. Sa porastom Internet saobraćaja i razvojem Internet aplikacija različitih i sve strožijih QoS zahtjeva, raste i potreba za alatima za preciznu klasifikaciju saobraćaja u realnom vremenu, koji bi omogućili sprovođenje fleksibilnih QoS polisa i polisa rutiranja.

Prvobitne metode identifikacije mrežnog saobraćaja različitih aplikacija bile su bazirane na provjeri broja porta u zaglavlju paketa. Ove jednostavne metode bile su učinkovite dok su aplikacije koristile unaprijed definisane brojeve porta, registrovane od strane IANA (*Internet Assigned Numbers Authority*). Međutim, vremenom je ovaj pristup postao nepouzdan jer mnoge aplikacije danas koriste neregistrovane portove ili čak nasumično generisane portove koji se mogu preklapati sa portovima koje koriste druge aplikacije. Zastupljenost NAT (*Network Address Translation*) u mrežama je još jedan ograničavajući faktor. Prema nekim studijama, metode zasnovane na portovima mogu klasifikovati svega 30-70% trenutnog Internet saobraćaja [1].

Deep Packet Inspection (DPI) metode identifikacije mrežnog saobraćaja koriste se od 2002. godine i bazirane su na analizi potpisa (engl. *signature*). Potpis je dio korisnog dijela paketa koji je fiksan i razlikuje se od aplikacije do aplikacije. Može se opisati kao sekvenca stringova ili heksadecimalnih vrijednosti. Tehnike identifikacije saobraćaja različitih protokola nivoa aplikacije na osnovu potpisa su jednostavne i njihova efikasnost je relativno visoka. Nivo greške u klasifikaciji je teoretski manji od 10%. Međutim, kada se specifikacija protokola promijeni, ili se kreira novi protokol, moraju se ponovo tražiti bitni potpisi, što zahtijeva dosta vremena i radnog angažovanja. Takođe, u mrežama čiji je saobraćaj dominantno enkriptovan učinkovitost ovih metoda je mala.

Nedostaci DPI tehnika adresirani su statističkim pristupima koji vrše klasifikaciju na osnovu statističkih karakteristika saobraćajnog toka vezanih za dužinu paketa, intervale između uzastopnih dolazaka paketa, brzinu prenosa toka, sadržaj zaglavlja itd. Preduslov za dobro funkcionisanje ovih metoda je ekstrakcija relevantnih karakteristika saobraćajnih tokova, na osnovu kojih se kasnije vrši klasifikacija. Međutim, statistička klasifikacija se ne može obavljati u realnom vremenu, već je potrebno obraditi više paketa toka da bi se utvrdilo kojoj klasi saobraćajni tok pripada.

Kao efikasno rješenje za analizirani problem klasifikacije u novijoj literaturi identifikovane su tehnike dubokog učenja (engl. *deep learning*). Dosadašnja istraživanja su pokazala da ove tehnike ostvaruju obećavajuće rezultate, mogu se izvršavati u skoro realnom vremenu i, za razliku od konvencionalnih tehnika mašinskog učenja, ne zahtijevaju manuelnu ekstrakciju bitnih karakteristika paketa (engl. *features*). Međutim, *dataset*-ovi koji su korišćeni u različitim radovima razlikuju se i nisu javno dostupni, a pri tome se razlikuju i tehnike predobrade *dataset*-ova, što onemogućava rangiranje ostvarenih rezultata.

U ovoj tezi izvršena je sistematizacija i komparacija *state-of-the-art* tehnika za klasifikacija mrežnog saobraćaja baziranih na dubokom učenju. Za potrebe istraživanja prikupljen je *dataset* iz realne mreže. Posebna pažnja posvećena je izboru optimalne strukture i tipa neuralne mreže u cilju dobijanja što preciznijeg i računski jednostavnijeg modela klasifikacije, pritom

uvažavajući konkretnu primjenu za koju će se model koristiti. U sklopu teze razvijena je aplikacija za treniranje modela i testiranje treniranih modela.

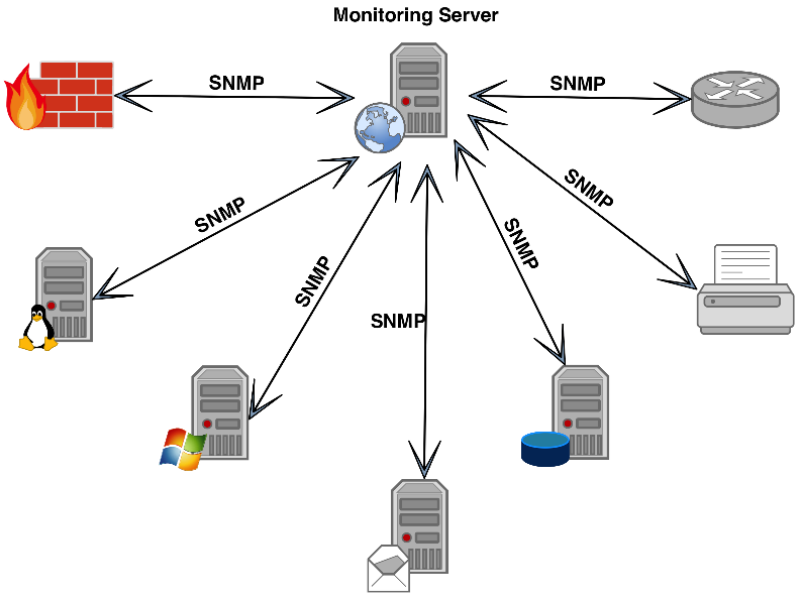
Značaj sprovedenog istraživanja ogleda se u činjenici da se tehnikama dubokog učenja mogu prevazići brojni nedostaci konvencionalnih tehnika za identifikaciju saobraćaja. Naime, ove tehnike mogu identifikovati saobraćaj mrežnih aplikacija koje koriste neregistrovane ili čak nasumične brojeve porta, a pri tome za identifikaciju karakterističnih obrazaca paketa nije potrebna manuelna ekspertska intervencija. U kontrolisanom mrežnom okruženju mrežni administratori mogu relativno jednostavno dinamički proširivati bazu aplikacija ili klasa saobraćaja koje model dubokog učenja treba da prepozna. Pri tome, primjenom koncepta federativnog učenja više različitih organizacija može kolaborativno trenirati model na svom saobraćaju, bez međusobnog ugrožavanja privatnosti.

1. KLASIFIKACIJA MREŽNOG SAOBRAĆAJA

1.1 Mjerenje mrežnog saobraćaja

U računarskim mrežama mjerenje mrežnog saobraćaja predstavlja proces mjerenja količine i vrste saobraćaja. Ovaj postupak je posebno važan za efektivno upravljanje propusnim opsegom mreže i može se sprovesti primjenom aktivnih ili pasivnih tehnika. Kod pasivnih tehnika podaci se sakupljaju pasivnim „osluškivanjem“ mrežnog saobraćaja, na primjer, koristeći (optičke) *link splitter*-e ili habove (engl. *hub)* za kopiranje saobraćaja sa linka, ili monitoringom bafera na ruterima. Mnogi moderni mrežni uređaji imaju neku vrstu ugrađenog mehanizma za pasivno mjerenje saobraćaja poput *Remote Monitoring* (RMON), koji može da se iskoristi da prikupi različite statističke podatke sa uređaja, kao što su broj poslatih bajtova, broj izgubljenih paketa i dr. Aktivne tehnike mjerenja kreiraju specijalne testne pakete koji se šalju putem mreže kako bi, na primjer, izmjerili vrijeme koje je potrebno da paket stigne do drugog kraja mreže (*one-way delay*), dostupni kapacitet mreže ili vrijeme odziva neke aplikacije. Suprotno pasivnim tehnikama, aktivne tehnike mjerenja generišu dodatni mrežni saobraćaj i to može da poremeti protok normalnog mrežnog saobraćaja [2]. Aktivne tehnike (poput *iPerf*-a [3]) su veoma zahtijevne ali neosporno i jako precizne. Pasivne tehnike manje opterećuju mrežu i zato se mogu koristiti kao *offline* aplikacije za upravljanje mrežom.

Postoje različiti softverski alati za mjerenje mrežnog saobraćaja. Neki uređaji mjere saobraćaj analizom ili „njuškanjem“ (engl. *sniffing*) paketa, a drugi korišćenjem SNMP [4] (engl. *Simple Network Management Protocol*, Slika 1), WMI [5] (engl. *Windows Management Instrumentation*) ili sličnih agenata, a za dobijene pakete podataka i informacije o tokovima podataka administratori mogu da razumiju ponašanje mreže, poput korišćenja aplikacija i korišćenja mrežnih resursa, kao i to koje su mrežne anomalije i sigurnosne slabosti mreže. Međutim, ovi drugi obično ne detektuju tipove saobraćaja, niti rade na uređajima koje nemaju na sebi aktivne kompatibilne agente, poput nepoznatih uređaja na mreži, ili uređaja na kojima kompatibilni agent nije prisutan/instaliran. U takvim slučajevima, potrebni su dodatni uređaji koji bi bili postavljeni između *Local Area Network*-a (LAN) i Internet rutera, i svi paketi koji izlaze i ulaze u mrežu moraju prolaziti kroz njih. Najčešće ovi uređaji funkcionišu poput mosta (engl. *bridge*) u mreži, i korisnicima su neprimjetni.



Slika 1. Simple Network Management Protocol [6]

Neki alati koji koriste za SNMP protokol za monitoring su *VistaInsight* [7], *Tivoli Netcool/OMNIbus* [8], *CA Performance Management* [9], *TotalView* [10], *Network Performance Monitor* [11], i NMIS [12].

1.2 Analiza paketa

Analizator paketa (poznat i kao *packet sniffer*) je kompjuterski program (poput *Wireshark*-a [13] i *SteelCentral Packet analyzer*-a), ili dio računarskog hardvera (uređaji za hvatanje paketa) koji presrijeće i loguje saobraćaj koji prolazi preko cijele mreže ili dijela mreže. Na Slici 2 je prikazan primjer hardverskog analizatora paketa.

Slika 2. Uređaj za hvatanje paketa IPCopper USC10G3 [14]

Hvatanje paketa je proces presrijetanja i logovanja saobraćaja. Kako tokovi podataka prolaze kroz mrežu, *sniffer* hvata svaki paket tokova, po potrebi dekodira „sirove“ podatke paketa, pokazujući vrijednosti njegovih polja, i analizira sadržaj prema RFC ili drugim specifikacijama.

Na žičnim mrežama poput *Ethernet*-a, *Token Ring*-a i FDDI (engl. *Fiber Distributed Data Interface*), može biti omogućeno hvatanje kompletnog saobraćaja mreže sa jednog

uređaja u mreži. U modernim mrežama saobraćaj može biti nagledan korišćenjem mrežnog sviča (engl. *switch)* sa takozvanim „portom za monitoring“, koji šalje kopiju svih paketa koji prolaze kroz određene portove na sviču. Mrežni TAP (engl. *Terminal Access Point,* Slika 3) je još efikasnija solucija od porta za monitoring jer ima manju šansu da ispusti pakete tokom visokog mrežnog opterećenja.

Slika 3. Dualcomm 10/100/1000Base-T Gigabit Ethernet Network TAP [15]

U *Wireless* LAN-ovima, saobraćaj može biti sniman na jednom kanalu, ili korišćenjem više mrežnih adaptera na više kanala istovremeno.

Kada se saobraćaj prikupi, snimaju se ili kompletni sadržaj paketa, ili samo zaglavlja (*header*, Slika 4). Ukoliko se čuvaju samo zaglavlja, smanjuje se potreban prostor za skladištenje, a i izbjegavaju se problemi vezani za zaštitu privatnosti podataka. Pri tome zaglavlja paketa obično sadrže dovoljno informacija za dijagnostiku problema. Prikupljeni paketi se dekodiraju iz sirove digitalne forme u čitljiv format koji omogućava korisnicima lak pregled razmijenjenih informacija.

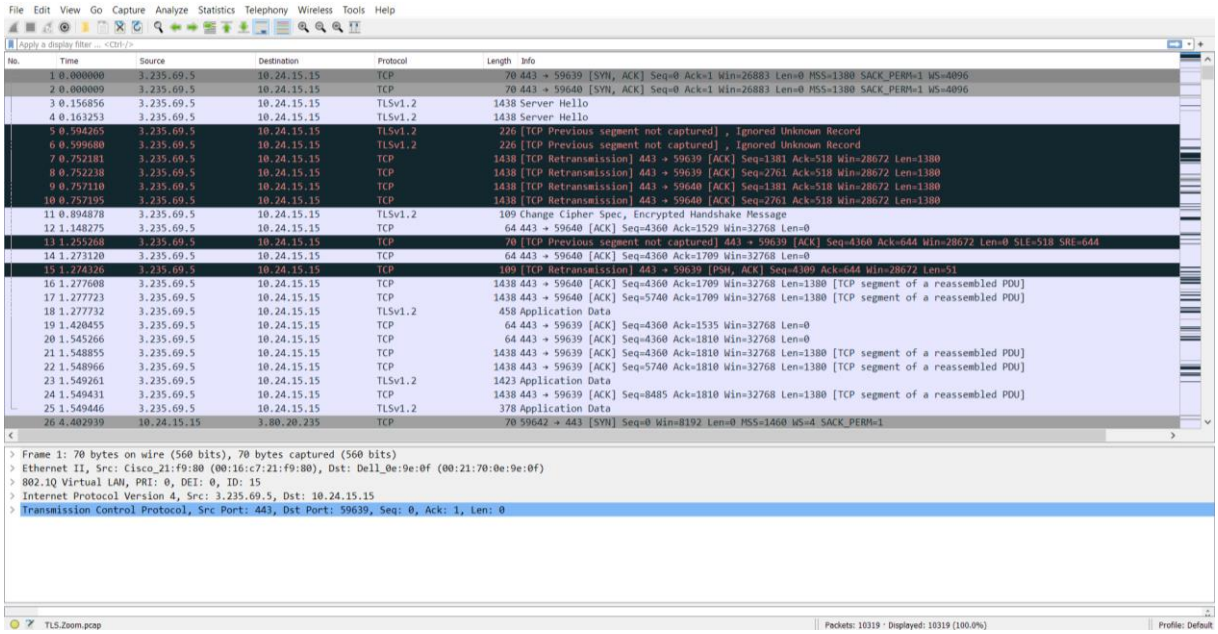
Slika 4. Zaglavlje paketa

Neki analizatori mreža takođe mogu da generišu saobraćaj. Mogu se koristiti kao testeri koji generišu ispravni saobraćaj nekog protokola, a pri tome mogu namjerno unositi greške kako bi testirali DUT (*Device Under Test*) sposobnost uređaja da se izbori sa greškama.

Analizatori paketa su od krucijalne važnosti za analizu mrežnih problema, otkrivanje pokušaja mrežnih upada, otkrivanje mrežnih zloupotreba od strane unutrašnjih ili spoljašnjih korisnika, izolaciju dijela mreže koji je pod sigurnosnim rizikom i dr.

Od softverskih alata za analizu saobraćaja danas je najšire zastupljen *Wireshark*, koji je podržan na Windows, Linux, macOS, BSD i drugim Unix operativnim sistemima. Uz *Wireshark* softverski paket dolazi instaliran i *TShark*, takođe program za analizu mrežnog saobraćaja, namijenjen za korišćenje bez grafičkog interfejsa preko komandne linije terminala. Koristi iste biblioteke kao i *Wireshark* za disekciju paketa, hvatanje i čitanje paketa, odnosno filtriranje. *Wireshark* razumije strukturu zaglavlja različitih mrežnih protokola. Može raščlaniti i prikazati sva polja u zaglavlju paketa, zajedno sa njihovim značenjima koja su definisana različitim mrežnim protokolima. *Wireshark* koristi fajlove formata *.pcap* [16] za čuvanje uhvaćenih paketa, pa može hvatati samo one pakete u mreži koje *.pcap* podržava.

Podaci za prikaz se mogu uhvatiti „sa žice“ tj. direktno sa mrežnog interfejsa, ili pročitati iz postojećeg *.pcap* fajla. Živi podaci se mogu čitati sa različitih tipova mrežnih interfejsa, uključujući *Ethernet, IEE 802.11, PPP,* i *loopback* interfejse. Uhvaćeni podaci se dalje mogu analizirati pomoću korisničkog interfejsa *Wireshark*-a (Slika 5) ili putem terminala *TShark*-a.



Slika 5. Grafički korisnički interfejs Wireshark-a

Uhvaćeni podaci se dalje mogu programski obraditi ili konvertovati putem CLI komandi programa *editcap*. Prikupljeni saobraćaj se filtrira korišćenjem ugrađenih filtera za prikaz. *Wireless* saobraćaj može biti takođe filtriran ukoliko prolazi kroz nadgledani *Ethernet*. *Wireshark* nudi i mogućnosti kreiranja novih *plugin*-ova za raščlanjivanje novih protokola. Može detektovati i *VoIP* pozive u uhvaćenom saobraćaju. Ako bi se pozivi kodirali podržanim enkoderom, njihov medijski sadržaj može biti čak i reprodukovan.

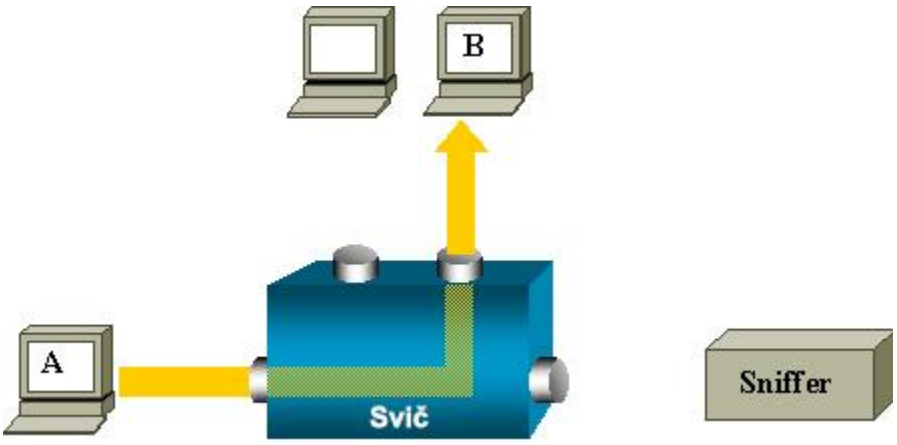
Wireshark-ov *default* fajl format za skladištenje mrežnog saobraćaja podržan je od *libpcap*-a [17] i *WinPcap*-a [18], pa se može koristiti sa drugim programima koji koriste isti format, uključujući *tcpdump* [17] i *CA NetMaster* [19]. *Wireshark* takođe može da čita uhvaćene pakete iz drugih programa za analizu mreža, kao što su *snoop* [20], *Network General's Sniffer* [21] i *Microsoft Network Monitor* [22].

1.3 SPAN

Switched Port Analyzer (SPAN) ili *Port Mirroring* funkcija pomaže pri analizi mrežnog saobraćaja koji prolazi kroz VLAN interfejse koristeći SPAN sesije. SPAN sesija šalje kopiju saobraćaja na interfejs sviča koji je povezan na mrežni analizator ili uređaj za monitoring. Pri tome ne utiče na tok mrežnog saobraćaja na izvorišnim interfejsima [23].

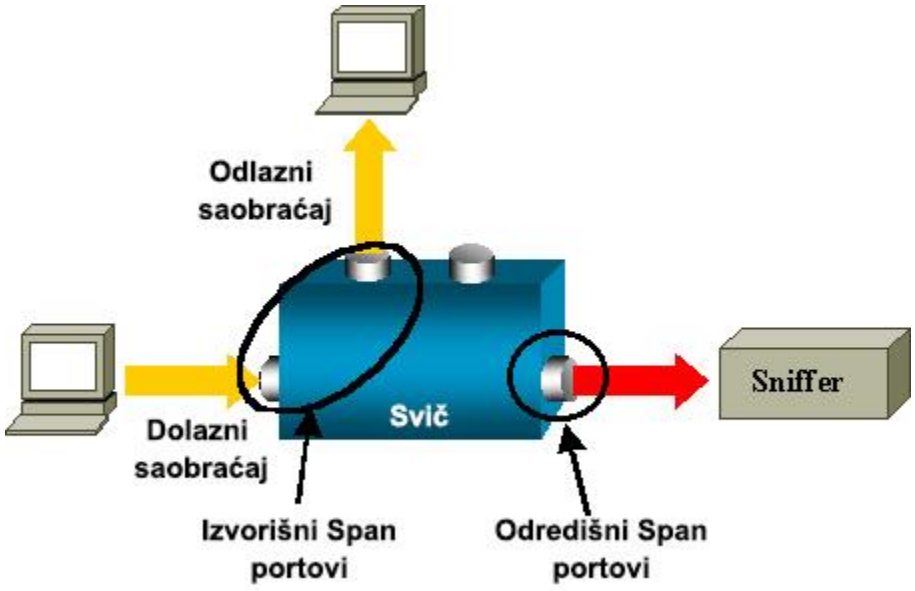
SPAN funkcionalnost se implementira na svičevima i jedna je od osnovnih funkcionalnih razlika između sviča i haba. Kada hab primi paket na neki od portova, on šalje kopiju tog paketa na sve portove osim na onaj sa kojeg je paket primljen. Stoga, ukoliko bi željeli da snimimo *Ethernet* saobraćaj koji razmjenjuju dva hosta povezana habom, dovoljno bi bilo da povežemo analizator paketa na hab.

Svič kreira tabelu prosleđivanja izvlačenjem informacije o izvorišnoj MAC adresi različitih paketa koje prima i mapiranjem tih adresa sa svojim interfejsima. Odluka o prosleđivanju paketa donosi se na osnovu odredišne MAC adrese. Stoga, pod pretpostavkom da je tabela prosleđivanja kompletna, *unicast* saobraćaj koji razmjenjuju hostovi A i B nije vidljiv analizatoru paketa i ostalim uređajima povezanim za svič (Slika 6).



Slika 6. Konfiguracija sviča koja nema mogućnost hvatanja saobraćaja [23]

Kako bi se vještački kopirali *unicast* paketi hostova A i B, potrebno je koristiti SPAN funkcionalnost na sviču. Elementi SPAN-a prikazani su na Slici 7.



Slika 7. Elementi SPAN-a prikazani na sviču [23]

Prilikom konfigurisanja SPAN-a prvo je potrebno definisati izvorišne SPAN portove (engl. *Source Span ports*) čiji će saobraćaj biti nadgledan pomoću SPAN sesije. Takođe, potrebno je jednom portu dodijeliti ulogu odredišnog (monitoring) SPAN porta (engl. *Destination Span ports*) i konfigurisati ga da nadgleda izvorišne portove.

1.4 Metode identifikacije mrežnog saobraćaja

Postojeći pristupi za klasifikaciju mrežnog saobraćaja mogu se kategorizovati na sledeći način:

- pristupi bazirani na analizi broja porta
- pristupi bazirani na analizi korisnog dijela paketa
- statistički pristupi

Klasifikacija mrežnog saobraćaja na osnovu analize portova je najstarija i najpoznatija metoda za ovu namjenu [24]. Ovaj tip klasifikatora ekstraktuje informaciju o broju porta iz TCP/UDP zaglavlja paketa i na osnovu njega povezuje paket sa određenom aplikacijom. Nakon ekstrakcije, broj porta upoređuje sa TCP/UDP brojevima portova koji su dodijeljeni od strane IANA za identifikaciju mrežnog saobraćaja.

Izvlačenje brojeva portova je jednostavan proces na koji ne utiču šeme enkripcije saobraćaja. Zbog toga se ovaj metod klasifikacije saobraćaja često koristi na *firewall* uređajima i prilikom provjere lista za kontrolu pristupa (engl. *Access Control Lists*) na različitim mrežnim uređajima [25].

Klasifikacija bazirana na portovima je poznata kao najjednostavnija i najbrža metoda za identifikaciju mrežnog saobraćaja. Međutim, velika zastupljenost NAT rutera (koji vrše modifikaciju broja porta u zaglavlju paketa radi uštede adresnog prostora) kao i sve veći broj aplikacija koji koriste neregistrovane ili naizmjenične brojeve porta, znatno su umanjili učinkovitost ovog pristupa. Prema [1], [26] samo 30% do 70% trenutnog Internet saobraćaja može biti klasifikovano korišćenjem klasifikacije koja se bazira na portovima. Zbog ovih razloga, potrebne su složenije metode kako bi se klasifikovao savremeni mrežni saobraćaj.

Tehnike za klasifikaciju mrežnog saobraćaja bazirane na analizi korisnog dijela paketa, poznate još pod nazivom DPI tehnike [27], koriste predefinisane šablone kao što su regularni izrazi i potpisi (engl. *signatures*) za različite aplikacione protokole [28], [29]. Na osnovu baze unaprijed identifikovanih šablona vrši se klasifikacija paketa. Potreba za dopunom baze šablona prilikom uvođenja novih protokola, kao i potencijalno ugrožavanje privatnosti korisnika su među najvažnijim nedostacima ovog pristupa. U [30] je predložen novi DPI sistem koji može ispitati enkriptovani sadržaj paketa bez dekripcije, i tako riješiti pitanje privatnosti korisnika, ali tim pristupom moguće je jedino obraditi *HTTP Secure* (HTTPS) saobraćaj [30].

Statistički pristupi klasifikacije saobraćaja bazirani su na pretpostavci da saobraćaj svake aplikacije ima neke statističke karakteristike koje su skoro jedinstvene. Različiti statistički metodi koriste različite funkcije i statistike. U [31] je predloženo kreiranje tzv. otiska protokola (engl. *protocol fingerprint*) na osnovu funkcije gustine vjerovatnoće intervala između uzastopnih dolazaka paketa i normalizovanih pragova. Ovim pristupom postignuta je preciznost od 91% za grupu protokola koja uključuje HTTP, *Post Office Protocol 3* (POP3) i *Simple Mail Transfer Protocol* (SMTP). U radu [32], uzeta je u obzir i funkcija gustine vjerovatnoće veličine paketa. Pokazalo se da predloženi pristup može da identifikuje veći opseg protokola, uključujući *File Transfer Protocol* (FTP), *Internet Message Access Protocol* (IMAP), SSH, i TELNET sa preciznošću do 87%.

U literaturi je predložen i veliki broj pristupa koji koriste prednosti mašinskog učenja za klasifikaciju mrežnog saobraćaja. U [33] je predložena *Bayes*-ova neuralna mreža koja uspješno klasifikuje brojne poznate P2P protokole, uključujući *Kazaa, BitTorrent, GnuTella*, postižući preciznost od 99%. Na istom setu aplikacija korišćen je i *Naive Bayes* klasifikator sa estimatorom gustine kernela [34]. Za identifikaciju mrežnog saobraćaja predlagani su i pristupi sa vještačkom neuralnom mrežom (engl. *Artificial neural network* - ANN) [35], [36]. U [36] je pokazano da pristup sa ANN može ostvariti bolje performanse od *Naive Bayes* metoda. U radu [37], predložen je pristup koji koristi karakteristike saobraćajnih tokova poput vremenskog trajanja tokova, broja bajtova u sekundi i broja paketa u sekundi, da klasifikuje mrežni saobraćaj *k-nearest neighbor* (k-NN) i *C4.5 decision tree* algoritmima. *C4.5 decision tree* algoritmom postignuta je preciznost od približno 90%, nad *dataset*-om koji uključuje šest ključnih klasa mrežnog saobraćaja, uključujući Web pretragu, email, *chat*, *streaming*, transfer fajlova i VoIP. Takođe postignuta je preciznost od približno 88% korišćenjem C4.5 algoritma na istom *dataset*-u koji je tunelovan kroz VPN. U radu [38] korišćeno je 111 manuelno odabranih karakteristika toka opisanih u [39] i primjenom k-NN algoritma postignuta je preciznost 94% za 14 klasa aplikacija. Glavni nedostatak svih ovih pristupa je što se izbor i ekstrakcija bitnih karakteristika saobraćaja vrše manuelno uz pomoć eksperta za analizu saobraćaja. To čini ovakve pristupe vremenski zahtijevnim, skupim i sklonim ljudskim greškama. Takođe, u radu [38], pokazano je da je vrijeme izvršavanja k-NN klasifikatora relativno veliko.

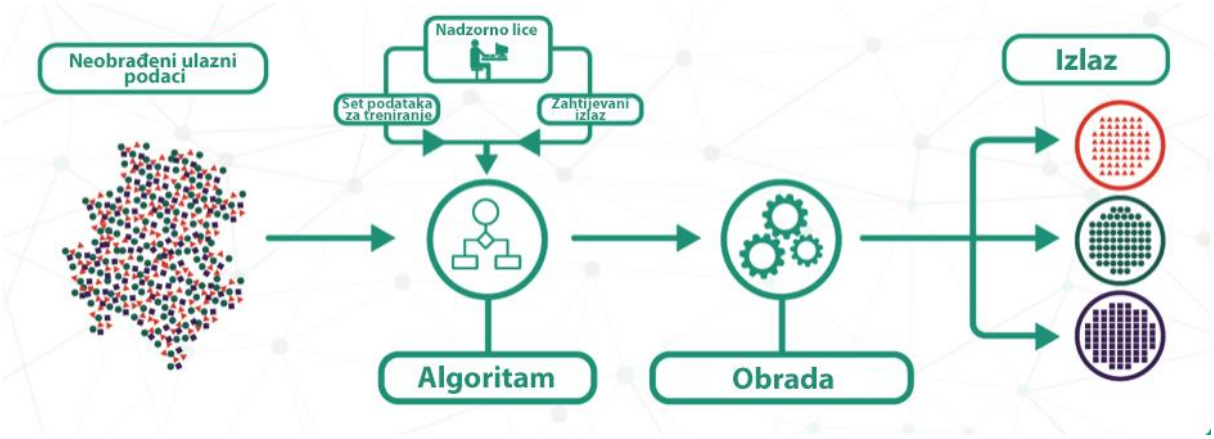
2. MODELI DUBOKOG UČENJA ZA KLASIFIKACIJU SAOBRAĆAJA

2.1 Duboko mašinsko učenje

Duboko učenje pripada domenu vještačke inteligencije (engl. *Artificial Intelegence* - AI) [40] i mašinskog učenja. Tokom posljednje decenije dubokom učenju i vještačkim neuralnim mrežama su posvećeni značajni naučni i tehnološko-razvojni resursi zbog potencijala da riješi široku klasu problema za koje konvencionalni metodi nisu dali adekvatan odgovor. Klasifikacija slika, prepoznavanje govora, prevođenje sa jednog na drugi jezik, medicinska dijagnostika, kontrola funkcija robota i vozila su samo neki od primjera primjene dubokog učenja i neuralnih mreža.

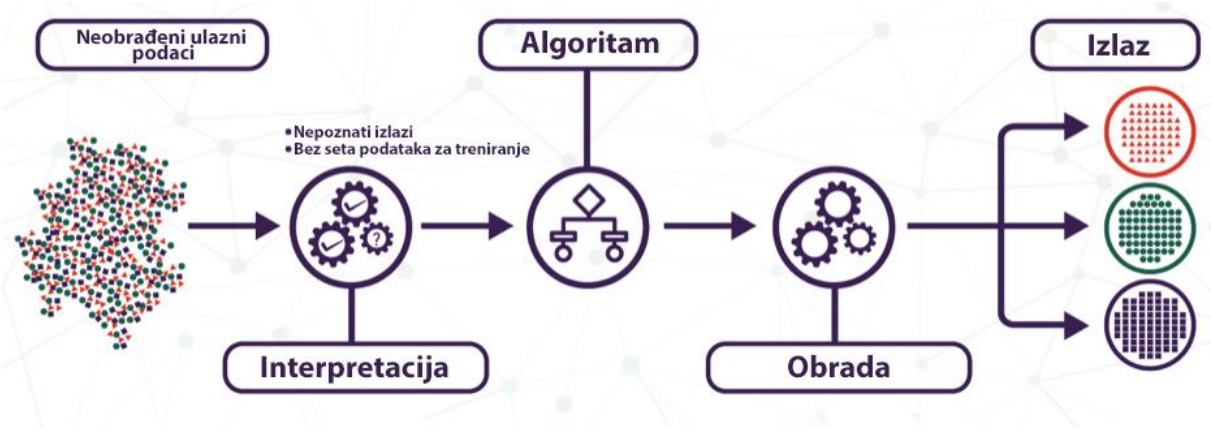
Tehnike dubokog učenja u novijoj literaturi primijenjene su i za klasifikaciju saobraćaja. Dosadašnja istraživanja su pokazala da ove tehnike ostvaruju obećavajuće rezultate, mogu se izvršavati u skoro realnom vremenu i da, za razliku od konvencionalnih tehnika mašinskog učenja, ne zahtijevaju manuelnu ekstrakciju bitnih karakteristika paketa.

Duboko mašinsko učenje može biti nadgledano, polu-nadgledano ili nenadgledano [41]. Kod nadgledanog mašinskog učenja (Slika 8) algoritmu se daju ulazni podaci iz kojih algoritam uči, kao i željeni izlazi. Algoritam učenja treba da „nauči“ funkciju koja mapira podatke sa odgovarajućim izlazima.



Slika 8. Nadgledano mašinsko učenje [42]

Suprotno nadgledanom mašinskom učenju, nenadgledano mašinsko učenje (Slika 9) upotrebljava se kada podaci koji se koriste za treniranje nijesu klasifikovani i označeni. Zadatak modela nenadgledanog učenja je da „nauči“ skrivene obrasce u neoznačenim podacima.



Slika 9. Nenadgledano mašinsko učenje [42]

Algoritmi polu-nadgledanog mašinskog učenja su kombinacija nadgledanog i nenadgledanog učenja, jer koriste i označene i neoznačene podatke za trening - uglavnom male količine označenih podataka i velike količine neoznačenih podataka. Cilj polu-nadgledanog sistema učenja jeste korišćenje neoznačenih podataka za efikasnije treniranje modela.

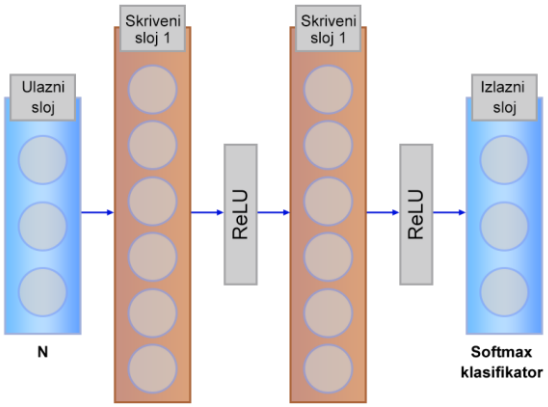
Algoritmi za pojačano mašinsko učenje (engl. *Reinforcement learning* – RL, Slika 10) su metodi učenja koji uče na osnovu interakcije sa okruženjem. Ispitivanje okruženja metodom pokušaja i greške i učenje na osnovu odložene nagrade su najvažnije karakteristike pojačanog mašinskog učenja. Na RL algoritmu učenja je da na osnovu iskustva u interakciji sa okruženjem utvrdi akciju ili koji skup akcija koje ostvaruju najveću nagradu, i da shodno tome koriguje svoje parametre za utvrđivanje akcije na osnovu ulaznog stanja.

Slika 10.Pojačano mašinsko učenje [42]

U ovom radu biće korišćena tri *Deep learning* metoda nadgledanog mašinskog učenja za *online* identifikaciju mrežnih aplikacija: višeslojni perceptron (engl. *Multilayer Perceptron* – MLP), složeni autoenkoder (engl. *Stacked Autoencoder* – SAE) i konvolucione neuralne mreže (engl. *Convolutional Neural Networks* - CNN ili ConvNets).

2.2 Višeslojni perceptron

Višeslojni perceptron je neuralna mreža sa *feed-forward* arhitekturom. MLP se sastoji od tri ili više slojeva. Prvi sloj prima vektore bajtova ulaznog paketa podatka koji treba da bude klasifikovan. Rezultati klasifikacije se nalaze u zadnjem sloju. Jedan ili više skrivenih slojeva koji se sastoje od većeg broja neurona se smještaju između ulaznog i izlaznog sloja, što je prikazano na Slici 11.



Slika 11.Arhitektura MLP

Skriveni slojevi predstavljaju računski mehanizam neuralne mreže. Računanje koje se izvršava u svakom skrivenom sloju i izlaznom sloju je predstavljeno formulom 1:

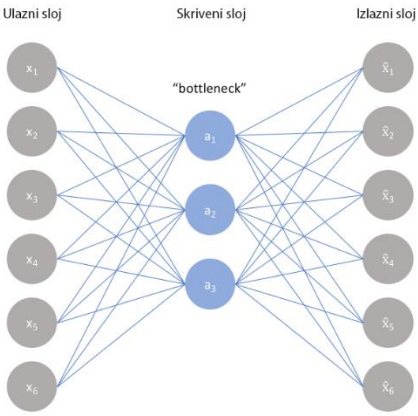
$$f(x) = \sigma\left(W^{(i)}x + b^{(i)}\right) \tag{1}$$

σ je aktivaciona funkcija, **x** je ulaz iz prethodnog sloja, $\mathbf{W^{(i)}}$ je matrica težine i $\mathbf{b^{(i)}}$ je vektor biasa za *i*-ti sloj. Posljednji sloj sadrži *softmax* aktivacionu funkciju. Izlazi *softmax* sloja su realni brojevi između 0 i 1.

2.1 Složeni autoenkoder

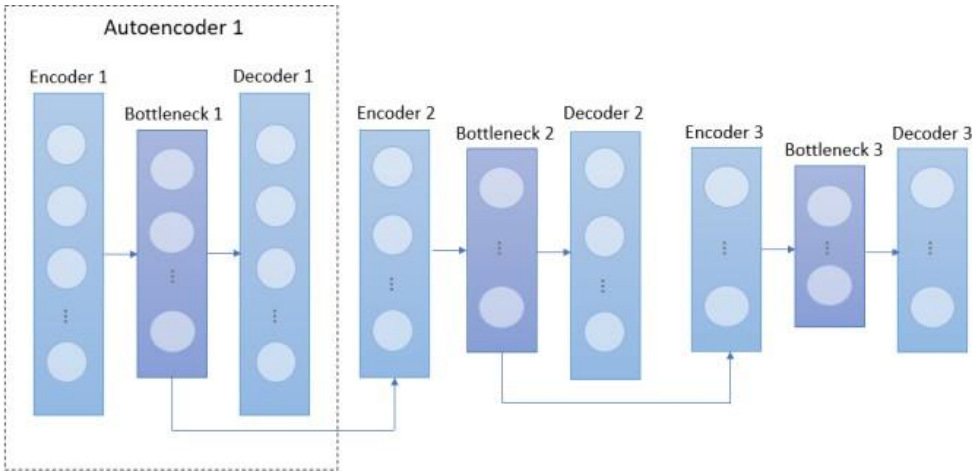
ANN (engl. *Autoencoder* - ANN) se tehnički sastoji od dvije osnovne komponente. Prva je enkoder koji je zapravo sloj potpuno povezan sa drugom komponentom, *bottleneck*-om. Iz *bottleneck*-a se dalje pokušava rekonstruisati ulazni podatak korišćenjem potpuno povezanih slojeva.

ANN je neuralna mreža sa nenadgledanim algoritmom za učenje koja koristi *backpropagation* da kreira izlazne vrijednosti koje su skoro slične ulaznim vrijednostima (Slika 12). Ulazni podaci su velikih dimenzija i provlačenjem kroz neuralnu mrežu vrši se njihova kompresija.



Slika 12.Slojevi u Autoencoder-u [43]

Stacked Autoencoder (SAE) je neuralna mreža koja se sastoji od više slojeva ANN, gdje su izlazi svakog skrivenog sloja povezani sa ulazima sledećeg skrivenog sloja (Slika 13).



Slika 13.Arhitektura Stacked Autoencoder-a [44]

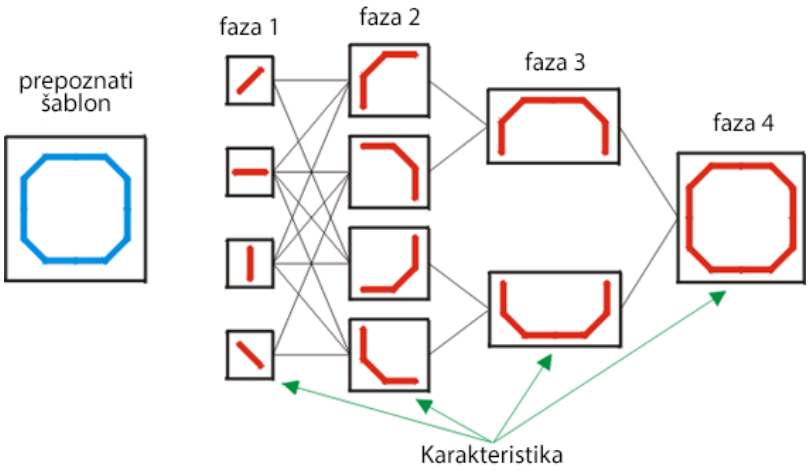
U praksi, korišćenje samo jednog autoenkodera obično nije dovoljno za dobro izvlačenje karakteristika. Zato se koristi više autoenkodera složenih jedan preko drugog, formiranjem arhitekture prikazane na Slici 13. U SAE arhitekturi, *bottleneck* sloj svakog autoenkodera je ulaz sledećeg autoenkodera u nizu. U prvoj fazi, slojevi autoenkodera se treniraju jedan po jedan, a onda da bi se postigli precizniji rezultati, primjenjuje se *backpropagation* algoritam na kompletan stek autoenkodera i svi parametri slojeva se fino podešavaju. Za potrebe klasifikacije se dodaje sloj sa N neurona i *softmax* aktivacionom funkcijom na posljednji *bottleneck* sloj.

2.2 Konvolucione neuralne mreže

Konvolucione neuralne mreže su popularna grupa neuralnih mreža koje pripadaju *Deep learning* metodama i koje su godinama specijalizovane za obradu vizuelnih podataka ili slika. Naziv „konvolucione neuralne mreže“ ukazuje da mreža koristi matematičku operaciju koja se

zove konvolucija umjesto standardnog množenja matrica u bar jednom od njihovih slojeva [45].

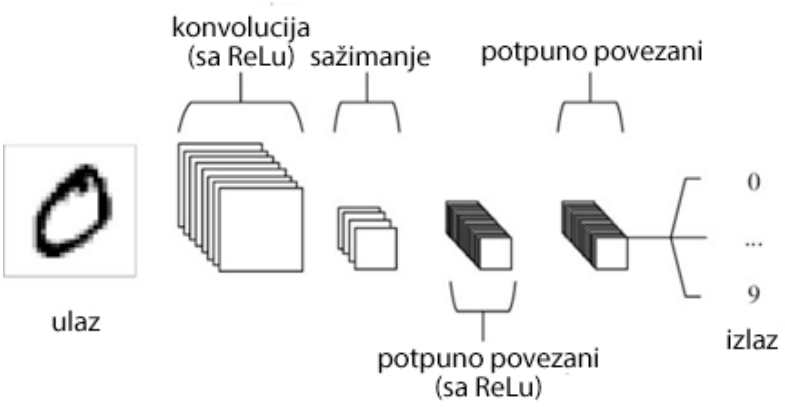
CNN dizajn je nastao po ugledu na funkcionisanje čula vida kod živih organizama. Istraživanje [46] je pokazalo je da vizuelni korteks mačke i majmuna sadrži neurone koji pojedinačno reaguju na male oblasti vizuelnog polja. Oblast vizuelnog prostora za koji se aktivira pojedinačni neuron poznat je kao receptivno polje. Susjedni neuroni imaju slična receptivna polja ili se ona preklapaju. Veličina receptivnog polja se mijenja sistematski kroz korteks kako bi se formirala kompletna mapa vizuelnog polja. Korteks na obije moždane hemisfere predstavlja kontra-lateralno vizuelno polje. Spomenuto istraživanje predstavilo je dva različita tipa receptivnih ćelija u mozgu: jednostavne ćelije, čiji je izlaz pojačan pravim ivicama, dajući im određenu orijentaciju unutar prijemnog polja, i kompleksne ćelije koje imaju veće receptivno polje, i koje su neosjetljive na tačnu poziciju ivica u vizuelnom polju. Na osnovama istraživanja vizuelnog korteksa nastala je *Neocognitron* neuralna mreža [47]. Osnovni princip rada ove neuralne mreže je hijerarhijsko izvlačenje karakteristika (engl. *features*). Izvlačenje karaktersitika se sprovodi u nekoliko faza. Najjednostavnije karakteristike (obično su to iskošene linije) izvlače se u prvoj fazi, dok se u svakoj sledećoj fazi izvlače složenije karakteristike. U ovom procesu važna je činjenica da se jedino informacije dobijene u prethodnim fazama koriste za izvlačenje karakteristika u određenoj fazi. Hijerarhija karakteristika koja se može koristiti za prepoznavanje broja 0 u *neocognitron*-u je predstavljena šematski na Slici 14. Vezama između karakteristika u susjednim fazama predstavljeno je koje su karakteristike iz prethodne faze korišćene tokom izvlačenja određene kompleksnije karakteristike.



Slika 14.Princip hijerarhijskog izvlačenja karakteristika [48]

CNN mreže se obično primjenjuju na slikama, koje se mogu posmatrati kao matrice piksela i okarakterisati širinom, visinom i vrijednostima piksela. Najčešće se vrši obrada RGB slika sa tri kanala. Vrijednosti RGB piksela u svakom kanalu su u granicama od 0 do 255. RGB slika se predstavlja kao trodimenzionalni niz brojeva, poznat i kao *tenzor trećeg ranga.* U procesu učenja karakteristika slika ključnu ulogu igraju konvolucioni slojevi CNN-a, koji ostvaruju osnovnu operaciju obučavanja neurona mreže.

Na Slici 15 prikazana je arhitektura jednostavnog CNN modela koja se sastoji od ulaznih podataka, konvolucionog sloja, sloja sažimanja, potpuno povezanog sloja i izlaznih podataka.



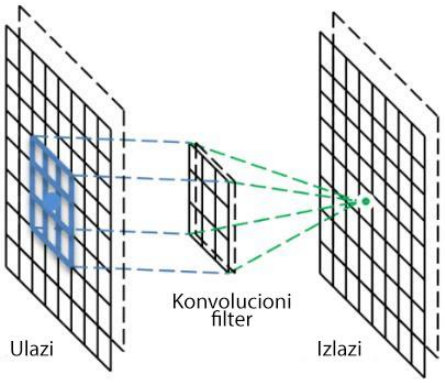
Slika 15. Jednostavna CNN arhitektura koja se sastoji od samo pet slojeva [49]

Ulazni sloj se ne odnosi samo na arhitekturu CNN-a i primjenljiv je na svaku neuralnu mrežu. To je sloj putem koga se podaci uvode u mrežu. Kod CNN-a ulazni podatak je najčešće slika ali u ovom radu, za potrebe klasifikacije saobraćaja, kao ulazni podatak korišćen je vektor bajtova paketa.

Osnovu CNN-a sačinjavaju tri sloja. To su konvolucioni, sloj sažimanja i potpuno povezani (engl. *fully connected*) sloj. Uz pomenuta tri sloja svake CNN, u mreži može biti prisutan i sloj aktivacione funkcije, koji može značajno poboljšati performanse CNN u obavljanju određenih zadataka.

Konvolucioni sloj realizuje osnovnu operaciju učenja neurona mreže primjenom konvolucionih filtera. Konvolucioni slojevi koriste konvolucione filtere za izvlačenje bitnih karakteristika iz ulaznog tenzora podataka. Filter se predstavlja dvodimenzionalnom matricom, manjih dimenzija u poređenju sa tenzorom na koji se primjenjuje. Sam postupak primjene filtera na tenzor zove se konvolucija. U zavisnosti od filtera koji se primjenjuje na ulaznom tenzoru, dobijaju se različiti rezultati. Filter se koristi na podskupu vrijednosti ulaznog tenzora koji je iste dimenzije. Svaka vrijednost tog podskupa se množi sa odgovarajućom vrijednošću u filteru, nakon čega se rezultat sumira u jednu vrijednost.

Princip rada konvolucionog filtera prikazan je na Slici 16. Na dijagramu je prikazan konvolucioni filter dimenzija 3x3 koji obrađuje dio slike iste dimenzije.



Slika 16. Prolaz kroz 3x3 konvolucioni filter sa ulaznom slikom 8x8 [50]

Neuroni CNN mreže raspoređeni su u 3D strukturu (širina×visina×dubina), pri čemu se svaki neuron povezuje sa određenim regionom (receptivnim poljem). Povezivanje svih neurona sa svim mogućim regionima ulaznih podataka nije praktično, jer može dovesti do prevelikog broja parametara za treniranje, što stvara preveliku kompleksnost izračunavanja. Pojam receptivnog polja u kontekstu CNN-a upućuje na dio ulaznih podataka koji je filteru vidljiv u datom trenutku.

Stride (kretanje/napredovanje) i *padding* (dopuna) su dva važna parametra konvolucionog sloja. *Stride* se odnosi na korak pomjeranja filtera preko tenzora sa ulaznog sloja. Na primjer, 3x3 filter sa *stride*-om veličine 2 bi počeo obradu kanala slike na poziciji (1,1), a onda prešao na (1,3), zatim (1,5), itd, čime prepolovljava veličinu izlaznog kanala/mape karakteristika, u poređenju sa konvolucionim filterom koji koristi *stride* od jednog piksela.

Generalno, prilikom prolaza konvolucionog filtera po tenzoru ulaznog podatka, pikseli koji se nalaze u sredini tenzora češće se koriste od onih koji se nalaze na ivicama tenzora. Ovo dovodi do gubitka informacija sa ivica ulazne matrice, čime i izlazni podatak iz konvolucionog sloja ima određeni gubitak informacija, što može praviti probleme u ostalim slojevima CNN-a. Kako bi se izbjegao ovaj problem uvodi se *padding* (ispunjavanje). *Padding* se odnosi na ivično dopunjavanje ulaznog podatka konstantnim vrijednostima. Najčešće se koriste sledeća dva načina dopune *padding*-om i to dopuna nulama i dopuna reflektivnim postavljanjem.

Bez upotrebe *padding*-a ivični elementi ulazne matrice podataka (npr. kanala slike) ne obrađuju se u istoj mjeri kao ostali podaci matrice, čime se pogoršavaju rezultati, posebno ukoliko mreža ima više konvolucionih slojeva. Najčešća vrijednost piksela kojima se vrši oivičavanje je nula, i odatle pojam dopune nulama (engl. *zero-padding*). Ako je, na primjer ulazni podatak konvolucionog sloja slika dimenzija 3x3, dopunom nulama postiže se dimenzija 5x5. Dalje, ako bi se na tako dopunjenu sliku u konvolucionom sloju primijenio filter 2×2 sa *stride*-om veličine 1, kao izlaz konvolucionog sloja dobila bi se struktura dimenzije 4×4. Reflektivno postavljanje je nešto efikasniji pristup koji dopunu vrši kopiranjem piksela sa ivice ulazne slike.

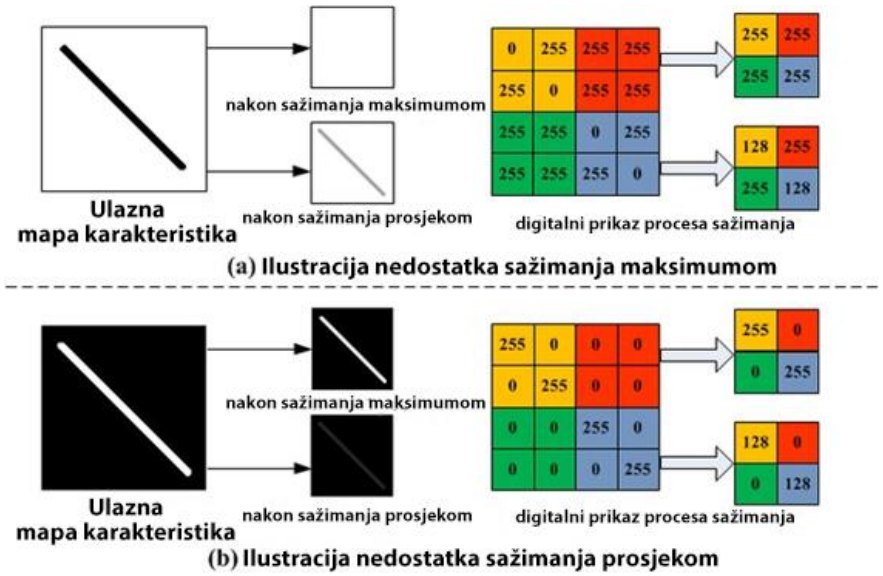
Uobičajeno je da se u arhitekturi CNN-a između sukcesivnih konvolucionih slojeva umetne sloj sažimanja čime se smanjuje broj parametara, a samim tim i izračunavanja unutar mreže. Smanjenjem broja parametara može se kontrolisati i varijansa CNN modela, tj. smanjiti rizik od *overfitting*-a [51]. Pored najčešće korišćenih sažimanja (prosječno i maksimumom) postoji veći broj metoda sažimanja, koji se primjenjuju u CNN-u, a to su: *Mixed Pooling* [52], *Spatial Pyramid Pooling* [53], *Stochastic Pooling* [54] i druge. Potpuno povezani sloj se tipično koristi kao posljednji sloj u mreži. Naziv sugeriše karakteristiku ovog sloja, a to je da su svi neuroni u ovom sloju su povezani sa svim izlazima prethodnog sloja.

Pravi izbor aktivacione funkcije (engl. *activation function*) može značajno poboljšati performanse CNN mreže koja se primjenjuje za određene zadatke [55]. Jedna od najčešće korišćenih aktivacionih funkcija je ReLU (engl. *Rectified Linear Unit*). Pored ReLU, neke od aktivacionih funkcija koje se često koriste su: *Leaky ReLU* (LReLU), *Parametric ReLU* (PReLU), *Randomized ReLU* (RReLU) i *Exponential Linear Unit* (ELU). Jedna od aktivacionih funkcija koji se može koristiti u CNN-u je i *softmax*. Stavlja se poslije izlaznog sloja, ima isti broj neurona kao izlazni sloj i dozvoljava neuralnoj mreži da vrši više-klasnu klasifikaciju ulaznih podataka.

Kako bi se izbjegao *overfitting* neuralne mreže, često se kao dio arhitekture koristi sloj ispuštanja (engl. *dropout layer*). Dio CNN mreže je često i sloj ravnanja (engl. *flatten layer*), koji oblikuje izlazne podatke konvolucionih slojeva u oblik pogodan za potpuno povezane slojeve.

Jedan od važnih koncepata CNN-a je sloj sažimanja (engl. *pooling layer*). Operacijom sažimanja se skup piksela u određenom regionu predstavlja određenom vrijednošću. Generalno, sloj sažimanja se koristi sa ciljem progresivnog smanjenja veličine slike, samim tim i broja karakteristika, što dovodi do smanjenja kompleksnosti izračunavanja. Najčešće korišćena operacija sažimanja je sažimanje maksimumom (engl. *max pooling*), a rjeđe se koristi i sažimanje prosjekom (engl. *average pooling*). Kod sažimanja maksimumom skup piksela u nekom regionu se predstavlja vrijednostima maksimalnog piksela u tom regionu. Sažimanje prosjekom bazira se na računanju srednje vrijednosti unutar određenog regiona. Ono se smatra korisnim kada treba uzeti u obzir sve vrijednosti unutar tog regiona. Međutim, ukoliko unutar regiona postoje ekvivalentne negativne i pozitivne vrijednosti, onda je rezultujuća vrijednost nula, što kasnije degradira proces učenja.

Sažimanje maksimumom je u kompjuterskoj viziji bitno jer se uklanjanjem vrijednosti koje nijesu maksimalne smanjuje računarska kompleksnost i pruža invarijantnost translacije. Prednost sažimanja maksimumom se obično bolje vidi kod izvlačenja važnih karakteristika kao što su ivice, koje maksimalno ističe, dok sažimanje prosjekom iste te karakteristike izvlači glatkije i detaljnije (Slika 17).



Slika 17. Sažimanje maksimumom i sažimanje prosjekom [56]

Nedostatak sažimanja maksimumom se vidi kod korišćenja vrijednosti piksela. Kako piksel sa vrijednošću 0 u kontekstu RGB vrijednosti (engl. *red, green* i *blue*, odnosi se na boje koje se koriste na displeju računara) predstavlja minimum osvjetljenja ili crnu boju, a 255 maksimum osvjetljenja ili bijelu boju, to kod sažimanja maksimumom uvijek daje vrijednost 255, čime se mogu izgubiti važni detalji. Sa druge strane, nedostaci sažimanja prosjekom su izraženi u situacijama kada su dominantne RGB vrijednosti 0 pa se uslijed izvlačenja prosječne vrijednosti detalji slabije ističu.

2.3 Metrike za procjenu performansi klasifikacije

Korišćenjem različitih metrika za procjenu performansi, može se poboljšati učinkovitost modela prije nego što se primijeni na nove podatke. Metrike koje se koriste za procjenu performansi klasifikacije su matrica zabune, tačnost, preciznost, odziv i F1 rezultat.

Izbor metrika performansi često zavisi od problema koji se rješava. Matrica zabune (engl. *Confusion Matrix*) služi kao osnova za računanje metrika performansi. Matrica se sastoji od četiri kategorije i to TP – *true positive*, FP – *false positive*, FN – *false negative* i TN – *true negative*

TP i TN kategorije se odnose na ispravno klasifikovane instance, dok se FP i FN odnose na neispravno klasifikovane instance iz *dataset*-a. Primjer matrice zabune za binarni klasifikator dat je u Tabeli 1.

Tabela 1. Primjer matrice zabune

Klasifikacija	Negativna (predviđena)	Pozitivna (predviđena)
Negativna (stvarna)	90 (TN)	1 (FP)
Pozitivna (stvarna)	8 (FN)	1 (TP)

Tačnost (engl. *Accuracy*) je metrika koja se definiše kao odnos ispravno klasifikovanih uzoraka *dataset*-a i ukupnog broj uzoraka:

$$Tačnost = \frac{TP + TN}{broj\ svih\ uzoraka} \quad (1)$$

Tačnost je očekivanje funkcije gubitaka i stoga je veoma bitan za analizu modela mašinskog učenja. Međutim, ova metrika nije relevantan pokazatelj stvarnih performansi ukoliko je *dataset* treniran na disbalansiranom *dataset*-u koji ne sadrži uzorke svake klase u podjednakoj proporciji. Na primjer, ukoliko je udio uzoraka negativne klase jako visok, tada će klasifikator koji svaki novi uzorak klasifikuje kao negativan imati visoku vrijednost tačnosti (iako se radi o veoma lošem modelu).

Preciznost (engl. *Precision*) je metrika koja indicira procenat relevantnih predikcija za neku klasu. Računa se prema formuli:

$$Preciznost = \frac{TP}{TP + FP} \quad (2)$$

Odziv (engl. *Recall*) se definiše kao odnos broja uzoraka koje je klasifikator označio kao pozitivne i ukupnog broja pozitivnih uzoraka:

$$Odziv = \frac{TP}{TP + FN} \quad (3)$$

U praksi je veoma teško postići da i preciznost i odziv imaju vrijednost 1. Neophodno je praviti kompromis između ove dvije mjere: povećanjem odziva smanjuje se preciznost, i obrnuto.

F1 rezultat kombinuje preciznost i *odziv* na sledeći način:

$$F1 = 2\,x\,\frac{preciznost\ x\ odziv}{preciznost + \ odziv} \quad (4)$$

U najboljem slučaju F1 rezultat može imati vrijednost jednaku 1, a u najgorem 0.

2.4 Klasifikacija mrežnog saobraćaja bazirana na dubokom mašinskom učenju

U literaturi je predloženo više pristupa klasifikacije mrežnog saobraćaja baziranih na dubokom mašinskom učenju [57], [58], [59], [60], [61], [62]. U radu [63] je korišćena SAE arhitektura neuralne mreže da klasifikuje mrežni saobraćaj za veću grupu protokola poput HTTP, SMTP itd. Međutim, u tehničkom izvještaju nijesu opisane karakteristike korišćenog *dataset*-a, korišćena metodologija i detalji implementacije.

U radu [62] predstavljani su modeli koji automatski izvlače karakteristike iz mrežnog saobraćaja, koristeći jedno-dimenzione konvolucione (1D-CNN) i SAE neuralne mreže za klasifikaciju i karakterizaciju mrežnog saobraćaja. Autori su postigli F1 rezultat od 0,98 za 1D-CNN i 0,95 za SAE kod klasifikacije mrežnog saobraćaja, a 0,93 za 1D-CNN i 0,92 za SAE kod karakterizacije mrežnog saobraćaja (identifikacije aplikacija).

U radu [64] korišćen je 1D-CNN model u kojem je za klasifikaciju mrežnog saobraćaja korišćen *nDPI* program. Model je treniran sa više različitih algoritama za optimizaciju. Rezultati pokazuju da najbolje rezultate daju algoritmi za optimizaciju koji se baziraju na metodi *Gradient Descent*. Međutim, tačnost prepoznavanja iznosi oko 0,78.

U ovom radu izvršena je analiza performansi MLP, SAE i CNN modela dubokog učenja u pogledu mogućnosti klasifikacije mrežnog saobraćaja. Modeli su međusobno upoređeni korišćenjem *dataset*-a prikupljenog u lokalnoj računarskoj mreži pri uobičajenim operativnim uslovima. Performanse modela analizirane su za različita ograničenja kompleksnosti u pogledu broja parametara u cilju identifikacije modela koji je najpogodniji za resursno-ograničene računarske platforme.

3. ANALIZA PERFORMANSI MODELA DUBOKOG UČENJA ZA KLASIFIKACIJU SAOBRAĆAJA

U radu je korišćen *dataset* kreiran od saobraćaja sakupljanog u periodu od nekoliko dana u računarskoj mreži Instituta za javno zdravlje Crne Gore, i za čije je prikupljanje korišćena *Port Mirroring* metoda.

Aplikacije u *dataset*-u definisane su pomoću *nfstream* [65] modula za Python. *Nfstream* provjera paketa oslanja sa na *nDPI* [66] biblioteku koja pripada grupi DPI tehnika. U *nDPI*-ju se aplikacija definiše jedinstvenim numeričkim *Id*-em i simboličkim imenom aplikacije (npr. *Skype*). Aplikacija se obično detektuje pomoću mrežnog *dissector*-a, pisanog u programskom jeziku C, a može biti definisana i pomoću porta, IP adrese, protokola i drugih karakteristika aplikacije.

NFStream inspekcija oslanja se na *nDPI* biblioteku koja omogućava *nfstream*-u da izvrši identifikaciju enkriptovanih aplikacija i tragova metapodataka (npr. TLS, SSH, DHCP, HTTP). *NFStream* obezbjeđuje *state of the art* ekstrakciju statističkih karakteristika zasnovanih na protoku. Uključuje *post mortem* statističke karakteristike (npr. minimum, srednja vrijednost, standardna devijacija, maksimalna veličina i vrijeme između dolazaka paketa) i *early flow* karakteristike (npr. sekvenca prvih n veličina paketa, vremena između dolazaka i pravce). Dizajniran je da radi sa *offline* saobraćajem u *.pcap* fajlovima ili živim saobraćajem sa mrežnog interfejsa.

Funkcionisanje *NFStream*-a se može vidjeti u sledećem kodu gdje se iz fajla *facebook.pcap* identifikuju krakteristike toka poput izvorišne adrese, mac adrese, porta i drugih karakteristika [67]:

```
from nfstream import NFStreamer
# We display all streamer parameters with their default values.
# See documentation for detailed information about each parameter.
# https://www.nfstream.org/docs/api#nfstreamer
my_streamer = NFStreamer(source="facebook.pcap", # or network interface
                         decode_tunnels=True,
                         bpf_filter=None,
                         promiscuous_mode=True,
                         snapshot_length=1536,
                         idle_timeout=120,
                         active_timeout=1800,
                         accounting_mode=0,
                         udps=None,
                         n_dissections=20,
                         statistical_analysis=False,
                         splt_analysis=0,
                         n_meters=0,
                         performance_report=0)

for flow in my_streamer:
    print(flow)  # print it.
# See documentation for each feature detailed description.
# https://www.nfstream.org/docs/api#nflow
NFlow(id=0,
      expiration_id=0,
      src_ip='192.168.43.18',
      src_mac='30:52:cb:6c:9c:1b',
```

```
    src_oui='30:52:cb',
    src_port=52066,
    dst_ip='66.220.156.68',
    dst_mac='98:0c:82:d3:3c:7c',
    dst_oui='98:0c:82',
    dst_port=443,
    protocol=6,
    ip_version=4,
    vlan_id=0,
    bidirectional_first_seen_ms=1472393122365,
    bidirectional_last_seen_ms=1472393123665,
    bidirectional_duration_ms=1300,
    bidirectional_packets=19,
    bidirectional_bytes=5745,
    src2dst_first_seen_ms=1472393122365,
    src2dst_last_seen_ms=1472393123408,
    src2dst_duration_ms=1043,
    src2dst_packets=9,
    src2dst_bytes=1345,
    dst2src_first_seen_ms=1472393122668,
    dst2src_last_seen_ms=1472393123665,
    dst2src_duration_ms=997,
    dst2src_packets=10,
    dst2src_bytes=4400,
    application_name='TLS.Facebook',
    application_category_name='SocialNetwork',
    application_is_guessed=0,
    requested_server_name='facebook.com',
    client_fingerprint='bfcc1a3891601edb4f137ab7ab25b840',
    server_fingerprint='2d1eb5817ece335c24904f516ad5da12',
    user_agent='',
    content_type='')
```

Kompletan kod programa korišćenih za izradu teze dostupan je na *Github*-u [68].

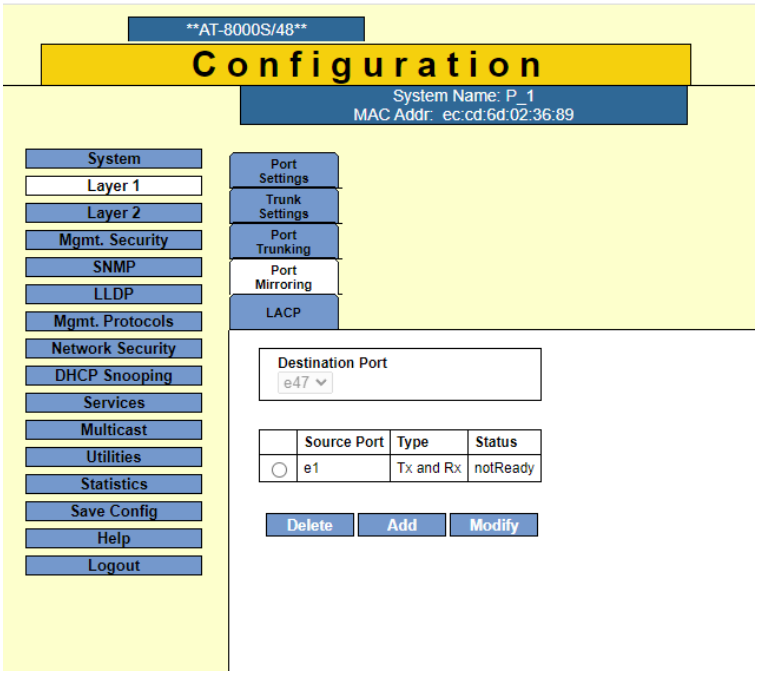
3.1 Kreiranje *dataset*-a

Mrežni saobraćaj je prikupljan *port mirroring* metodom. Kao što je objašnjeno u poglavlju 1, *mirroring* portovi sviča prosleđuju saobraćaj na povezani uređaj za analizu prikupljenog saobraćaja. Za potrebe ove teze, konfigurisan je *port mirror* za svič Allied Telesys AT8000S (Slika 18), koji je korišćen za prikupljanje mrežnog saobraćaja.

Slika 18.Allied Telesys 8000S/48 [69]

Allied Telesys AT8000S nudi mogućnost konfiguracije putem CLI komandi ili GUI *web* interfejsa. Koraci za konfigurisanje *port-mirroring* alata preko GUI interfejsa su sledeći:

1. Izbor opcije ***Layer 1* > *Port Mirroring***. Kao rezultat ovog koraka otvara se *The Port Mirroring* strana sa Slike 19, koja sadrži informacije o svim *port mirror*-ima koji su trenutno definisani na uređaju.

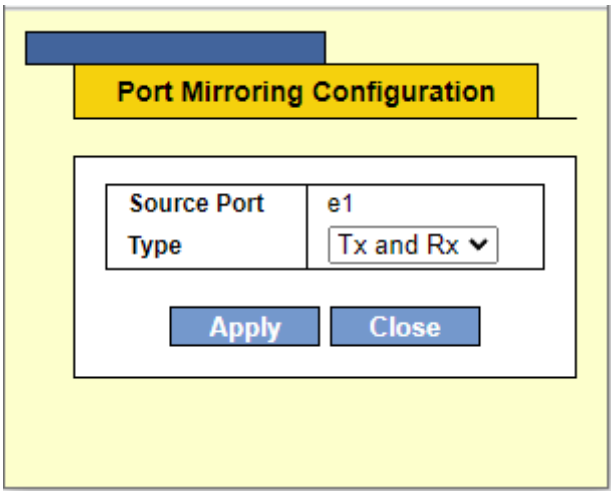


Slika 19.interfejs za konfiguraciju Port Mirroring-a

Opis *port-mirror*-a sadrži sledeće informacije:

- **Odredišni port (*Destination Port*)** — Definiše broj porta na koji se mrežni saobraćaj kopira. Taj port mora da bude nezavisan od VLAN-ova definisanih na sviču prije nego što se konfiguriše kao odredišni port za *mirroring*. Samo jedan odredišni port može biti definisan.
- **Izvorišni port (*Source Port*)** — Označava port sa kojeg se vrši nagledanje i kopiranje paketa.
- **Tip (*Type*)** — Opisuje način na koji je konfigurisan *port mirroring*. Moguće vrijednosti ovog polja su:
  - *RX* — Označava da je *port mirror* definisan za praćenje saobraćaja koji se prima na naznačenom portu.
  - *TX* — Označava da je *port mirror* definisan za praćenje saobraćaja koji se šalje sa naznačenog porta.
  - *RX and TX* — Označava da je *port mirror* definisan i za poslati i primljeni saobraćaj.
- **Status** — Pokazuje da li se *Source port* trenutno nadgleda. Moguće vrijednosti ovog polja su:
  - *Active* — Označava da se port trenutno nadgleda.
  - *Ready* — Označava da se port trenutno ne nadgleda.

2. Klikom na ***Add*** dugme otvara se prozor preko kojeg je moguće dodati novi *port mirror*, kao što je prikazano na Slici 20. U prozoru je potrebno je definisati *Destination Port*, *Source Port* i *Type* parametre za novi mirror.



Slika 20.Add Port Mirroring prozor

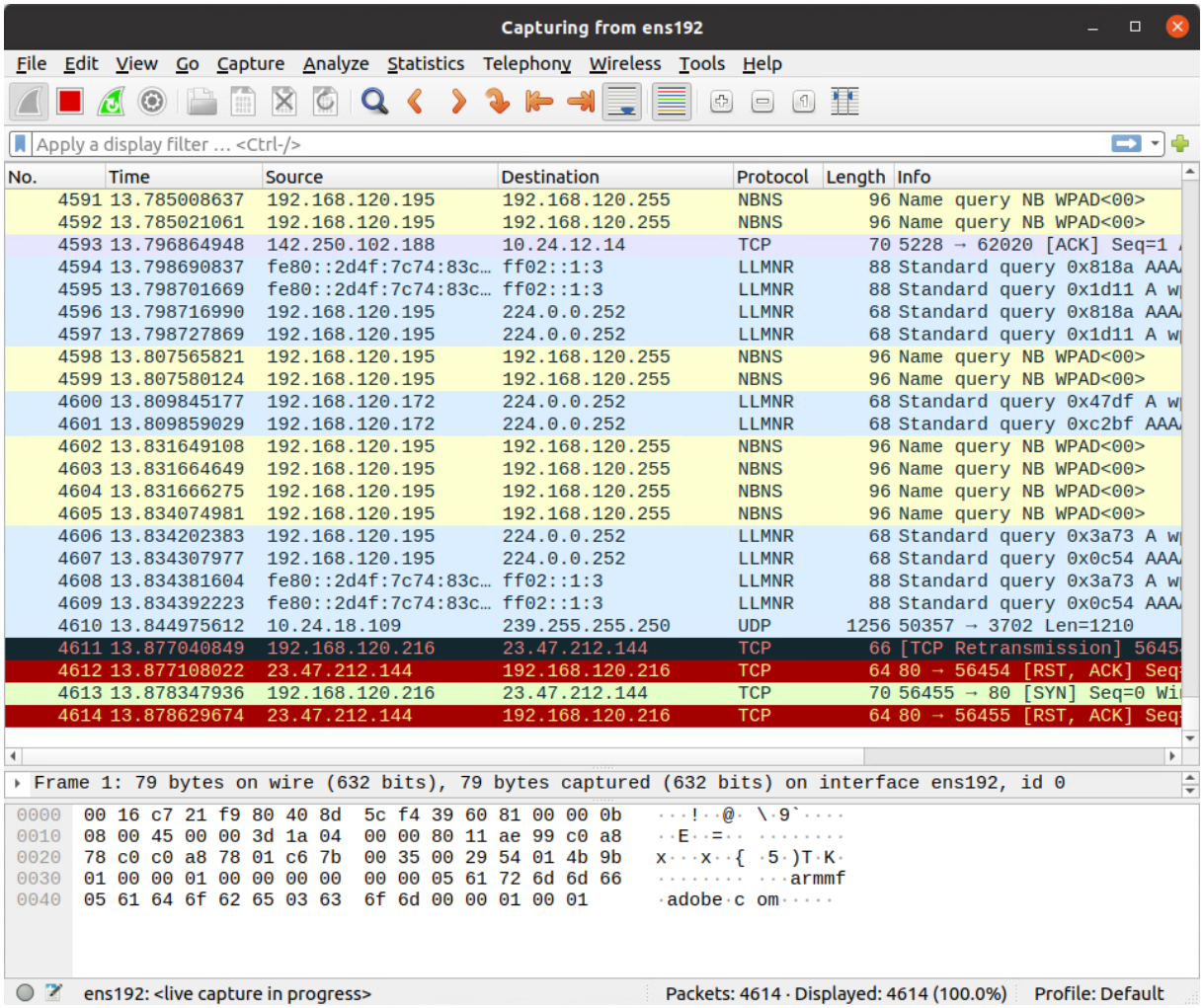
3. Potvrda izmjene klikom na ***Save Config*** opciju u meniju.

Alternativni način konfiguracije *port-mirroring*-a je direktan unos komandi u CLI. Sledećim komandama saobraćaj se kopira sa izvorišnog porta (1/e47 u primjeru) na određeni odredišni port (1/e47):

```
console(config)# interface ethernet 1/e47
console(config-if)# port monitor 1/e1
```

Sa komandom *port monitor source_port [rx | tx]* definiše se da li se nadgleda saobraćaj koji se šalje ili prima sa ovog porta. Ukoliko se *rx* ili *tx* ne definišu, onda se komanda odnosi i na primljeni i na poslati saobraćaj.

Za potrebe sakupljanja mrežnog saobraćaja *port mirroring* metodom korišćen je računar sa operativnim sistemom *Ubuntu 20.04 LTS*. Ubuntu je odabran zbog svoje fleksibilnosti i velikog broja programskih modula i biblioteka koje su na raspolaganju. Za potrebe *port mirroring*-a potrebno je konfigurisati mrežni interfejs računara koji je povezan na odredišni port sviča. Kako ovaj interfejs služi da nadgleda saobraćaj sa lokalno povezanog inerfejsa sviča, dovoljno je koristiti *link local* IPv4 metod povezivanja. Uvid u saobraćaj koji se prikuplja putem ovog interfejsa moguć je korišćenjem *Wireshark* softvera (Slika 21).



Slika 21.Prikaz port mirroring saobraćaja dobijen pomoću Wireshark-a

Da bi se izbjeglo sakupljanje saobraćaja sa pogrešnog interfejsa, ukoliko ih na računaru ima više, kreirana je funkcija za izbor interfejsa sa kojeg će biti pokrenuto skeniranje saobraćaja. Unix komandom *ip -o link show | awk -F': ' '{print $2}'* prikupljaju se informacije o interfejsima koji su na raspolaganju, a nakon toga se kreira njihova numerisana lista. Izborom broja sa liste bira se i interfejs sa kojeg će se saobraćaj preuzimati.

Nakon izbora interfejsa slijedi sakupljanje mrežnog saobraćaja. Sakupljeni mrežni saobraćaj čuva se u *.pcap* fajl, a to se vrši *Wireshark*-ovom komandom *dumpcap*.

Kako bi se mrežni saobraćaj mogao sakupiti predviđeno je tačno definisano vrijeme. To vrijeme je za ovaj rad definisano na period od jednog minuta.

Ovako obrađen saobraćaj analizira se *NFStream* modulom, i pri tome se kreira baza u *.csv* formatu. Ovakav *.csv* fajl sadrži metapodatke za prepoznate aplikacije (*src_ip*, *src_port*, *dst_ip*, *dst_port*, *protocol*...). Od dobijenih IP adresa i portova formira se jedan string, a koji će biti korišćen za kasnije upoređivanje.

Paketi još jednom prolaze identifikaciju koja se vrši pomoću *scapy* funkcija. *Scapy* je biblioteka koja se koristi se za interakciju sa paketima na mreži. Ima nekoliko funkcija kroz koje može lako da se upravlja i manipuliše paketima. Moguće je izvući izvorišnu i odredišnu IP adresu i port. Od ovih se podataka kreira string istog formata kao kod prethodne obrade sa *NFStream* modulom.

Vrši se poređenje paketa poređenjem stringova sa IP adresama i portovima dobijenih po analizi *scapy* funkcija sa stringovima kreiranih po *NFStream .csv* bazi. Ukoliko dođe do poklapanja stringova takvi paketi se smještaju u odvojene *.pcap* fajlove, čija se imena uzimaju iz odgovarajućeg *application_name* metaparametra iz *NFStream .csv* baze.

Upisivanje pronađenih aplikacija se takođe kontroliše. Kako se u mrežnom saobraćaju pojedine aplikacije češće pojavljuju od ostalih, tako se i pronađeni paketi u *.pcap* fajlovima tih aplikacija brže nagomilavaju. Nakon nekog vremena to bi dovelo i do pretrpavanja skladišnog prostora na *hard* disku računara, a i ne bi doprinijelo boljim rezultatima istraživanja. Ukratko, kontrolišu se ukupne veličine *.pcap* fajlova aplikacija i za rad je definisano da ne budu veće od jednog gigabita. Drugi način za ograničenje mogao je da bude definisan brojem paketa, ali je uzeta u obzir pretpostavka da ovakav način prikupljanja ne mora nužno obezbijediti dovoljan broj kvalitetnih paketa (sa *payload*-om) koji bi se mogli upotrijebiti za treniranje modela mašinskog učenja.

Aplikacije koje su količinski zadovoljile ograničenje programa stavljaju se na eliminacionu listu, kako se ne bi više uzimale u razmatranje tokom sledećih analiza novosakupljenog saobraćaja. Upis na ovu listu vrši se na kraju dodavanja paketa u fajl aplikacije, i to samo ako je taj fajl prešao predviđeno ograničenje od jednog gigabita. Ovaj sistem eliminacije u neku ruku ubrzava proces kojim se izvršavaju sledeće analize dobijenog saobraćaja, jer upravo eliminiše aplikacije koje se najčešće ponavljaju.

Fajlovi ulaznog saobraćaja se brišu na kraju svake izvršene analize i upisa pronađenih paketa, i kreiraju se novi. Za imenovanje novokreiranog *.pcap* fajla prikupljenog ulaznog saobraćaja, između ostalog, koristi se vrijeme u momentu početka njegovog kreiranja.

Po završenom sakupljanju saobraćaja, sortiranog u *.pcap* fajlove imenovane imenima aplikacija, vrši se dalja analiza. Odbacuju su aplikacije koje imaju protokol DNS u prefiksu imena, jer nijesu relevantne za klasifikaciju. Takođe, odbacuju su i aplikacije sa jako malom ukupnom količinom *stream*-ova, manjom od 1MB jer takvi podaci nijesu dovoljni za kvalitetno treniranje modela. Konačni spisak, koji uključuje 33 aplikacije raspoređene u 26 klasa, kreiran je u *Python dictionary* formi, i smješten u posebnom fajlu.

3.2 Predobrada *dataset*-a

Za potrebe predobrade podatke korišćen je pristup predložen u [62]. Paketi bez *payload*-a, kao što su TCP segmenti sa SYN, ANK ili FIN oznakama, koji su generisani prilikom uspostavljanja ili završetka TCP konekcije, se odbacuju. DNS paketi se takođe odbačeniodbacuju jer ne sadrže *payload* koji je koristan za ovo istraživanje.

Ethernet zaglavlje sa MAC adresama se uklanja, jer se MAC adrese razlikuju za različite mrežne interfejse računara, pa bi se tretirale kao posebna karakteristika paketa. Takva

karakteristika bi često bila različita kod paketa sa istim aplikacijama, time ne bi doprinosila učenju modela i zato je treba odbaciti. IP adrese su maskirane radi zaštite privatnosti podataka. Na ovaj način takođe je spriječeno da modeli mašinskog uče da klasifikuju mrežni saobraćaj na osnovu IP adrese u paketima. Kako standardni TCP paketi imaju dužinu od 20 bajta, a UDP 8 bajta u zaglavlju, UDP zaglavlja dopunjavana su sa 12 bajtova nula. Ovim korakom osigurano je da svi paketi pripremljeni za treniranje modela imaju jednake dužine.

3.3 Izrada modela mašinskog učenja i podešavanje parametara modela

Prilikom izrade modela korišćene su Python biblioteke *Pandas*, *NumPy*, *Tensorflow*, *i Keras i Scapy*. *Tensorflow* je modul koji služi za brza numerička računanja i izradu modela mašinskog učenja. *Keras* služi kao interfejs za funkcije *tensorflow*-a. Služi za jednostavnije konstruisanje arhitekture modela u *tensorflow*-u. Sadrži sve važne funkcije za kreiranje modela.

Ukupan broj paketa koji je korišćen za treniranje modela je 497296. Maksimalni broj paketa koji se koriste po klasi u modelu je 10000. Neke aplikacije su imale nešto manji broj paketa od zadatog maksimuma, a najmanji je bio 2487. Učitani podaci su podijeljeni u tri grupe, i to za treniranje, validaciju i testiranje, u odnosu 60:20:20, gdje se 60% koristi za treniranje. Veličina paketa ograničena je na 1500 bajta, što je standardna dužina MTU (engl. *Maximum Transmision Unit*). Manji paketi dopunjavani su nulama do te dužine. Nakon učitavanja podataka, a prije samog procesa treniranja, izvršena je konverzija paketa koji su u *scapy*-ju dobijeni kao nizovi u heksadecimalnom obliku u cjelobrojne nizove, u opsegu od 0 do 255. Takođe, vrijednosti bajtova paketa normalizovane su u opsegu od 0 do 1, radi bržeg treninga modela mašinskog učenja. U Tabeli 2 prikazane su detaljne informacije o klasama *dataset*-a.

Tabela 2. Klase koje su korišćene u *dataset*-u i broj njihovih paketa

	Aplikacija	**Za treniranje**	**Za validaciju**	**Za testiranje**	**Ukupno**
0	DCE_RPC	6000	2000	2000	10000
1	Dropbox	9731	3243	3245	16219
2	HTTP.Google	1492	497	498	2487
3	HTTP.Microsoft	7276	2425	2426	12127
4	HTTP.WindowsUpdate	6480	2160	2160	10800
5	Kerberos	6000	2000	2000	10000
6	LDAP	6000	2000	2000	10000
7	LLMNR/RTP	18000	6000	6000	30000
8	MongoDB	6000	2000	2000	10000
9	NetBIOS	12000	4000	4000	20000
10	NetBIOS.SMBv1	6000	2000	2000	10000
11	NetBIOS.SMBv23	12000	4000	4000	20000
12	QUIC/TLS.Cloudflare	19967	6655	6657	33279
13	QUIC/TLS.DoH_DoT	19984	6661	6663	33308
14	QUIC/TLS.Facebook	21105	7035	7036	35176
15	QUIC/TLS.YouTube	19148	6382	6384	31914
16	Radius	6000	2000	2000	10000
17	SMTP	6000	2000	2000	10000
18	SNMP	6000	2000	2000	10000
19	SSDP	6000	2000	2000	10000
20	TLS.Dropbox	12000	4000	4000	20000
21	TLS.Microsoft	34819	11606	11608	58033
22	TLS.POPS	6000	2000	2000	10000
23	TLS.Viber	7356	2452	2453	12261
24	UPnP	6000	2000	2000	10000
25	Viber	7740	2580	2581	12901
	Sve ukupno	298372	99453	99471	497296

Problem podešavanja modela svodi se na pronalaženje pravih parametara kojim bi se mogle postići optimalne performanse modela. Ovaj je proces obično dugotrajan i često se svodi na metodu pokušaja i greške. Kako bi se obezbijedilo ravnopravno poređenje, modeli su kreirani tako da imaju isti broj parametara. Konkretno su za sve uzete modele mašinskog učenja testirane tri različite konfiguracije sa 3, 6 i 9M (miliona) parametara respektivno. Za tu namjenu, podešavane su veličine dimenzije konvolucionih i *dense* slojeva.

Korišćeni MLP modele se sastoje od tri skrivena sloja iste veličine. Broj neurona u skrivenim slojevima biran je da približno zadovolji ciljani broj parametara, npr. 900 za 3M parametara, 1200 za 6M i 1800 za 9M parametara.

SAE modeli su implementirani po uzoru na model koji je predložen u [62]. Enkoder i dekoder imaju istu arhitekturu neuralnih mreža, sa jedinom razlikom da su skriveni slojevi obrnuto raspoređeni. U SAE konfiguraciji sa 3M parametara, Enkoder se sastoji od tri potpuno povezana sloja sa 600, 500, 400, i 300 parametara. Veličina *bottleneck* sloja je podešena na 200. U konfiguraciji sa većim brojem parametara veličina *bottleneck* sloja je duplirana, i broj i dimenzije slojeva enkodera i dekodera su povećane.

CNN modeli su generalno predviđeni za rad sa slikama odnosno podacima koje imaju više od jedne dimenzije. U ovom radu obrađuju se jednodimenzioni vektori podataka (u nastavku 1D-CNN modeli). 1D-CNN modeli se baziraju na arhitekturi predloženoj u [64]. Sastoje se od 2 konvoluciona sloja, praćena slojem sažimanja maksimumom, slojem ravnanja i potpuno povezanim slojevima sa 200, 100 i 50 neurona. Veličine kernela korišćene za 1D-CNN za 1D-CNN slojeve su 4 i 5, respektivno. Broj filtera (konvolucione matrice) u konfiguraciji sa 3M parametara je 21. Ovaj parametar je podešavan kako bi se postigla ciljana kompleksnost neuralne mreže. Kod korišćen za kreiranje ovakvog modela je dat u nastavku:

```
input_size = 1500
dropout = 0.05

# Izrada CNN modela sa 3M parametara
model = Sequential()
model.add(Conv1D(21, 5, input_shape = (input_size,1), activation = 'relu'))
model.add(Dropout(dropout))
model.add(Conv1D(21, 4, activation = 'relu'))
model.add(Dropout(dropout))
model.add(MaxPooling1D(2))
model.add(Flatten())
denses = [200, 100, 50]
for dense in denses:
    model.add(Dense(dense, activation = 'relu'))
    model.add(Dropout(dropout))
model.add(Dense(nb_classes, activation = 'softmax'))
```

Takođe je testiran i model bez potpuno povezanih slojeva (CNN-NFC), kojem su podešavane veličine filtera za dostizanje predviđene kompleksnosti neuralne mreže.

U svakom modelu, potpuno povezani slojevi i 1D-CNN (gdje ih ima) su regulisani sa vrijednošću *dropout*-a od 0,05, kako bi se izbjegao *overfitting*.

Modeli su trenirani 10 epoha, sa veličinom *batch*-a od 32 i *learning rate*-om 0,001. Parametri neuralne mreže sa najboljim rezultatima validacije tokom 10 epoha treniranja su

korišćena tokom faze testiranja. Trenirani model i njegovi tenzori čuvani su u posebnim fajlovima (*.h5* formata) na kraju svake epohe. Tenzori (engl. *tensors*) su strukture podataka koje koriste sistemi mašinskog učenja, i obično imaju definisani broj osa (engl. *rank*), oblik i tip podataka. Kod koji se odnosi na treniranje CNN modela je dat u nastavku:

```
batch_size = 32
nb_epochs = 10

model.compile(optimizer=optimizers.Adam(learning_rate=0.001), \
     loss='categorical_crossentropy', metrics=['accuracy'], run_eagerly='true')

# lokacija fajla modela
saved_model_file = 'models/cnn_model.h5'.format('conv1d-Adam')

# čuvanje modela u kontrolnim tačkama kada se funkcija loss poboljšava
checkpoint = ModelCheckpoint(saved_model_file, monitor='val_loss', save_best_only=True, \
verbose=1)
fit_history = model.fit(x_train, y_train, epochs=nb_epochs, batch_size=batch_size, \
     validation_data=(x_val,y_val), callbacks=[checkpoint])
```

Najefikasniji algoritmi optimizacije birani su za svaki DL model u skladu sa eksperimentalnim evaluacijama. Za 1D-CNN i CNN-NFC modele korišćen je algoritam optimizacije *Adam* [70]. SAE i MLP modeli su najbolje rezultate performanse imali sa *SGD-Momentum* algoritmom za optimizaciju.

3.4 Rezultati treniranja, validacije i testiranje modela

Rezultati analize predstavljeni su u Tabeli 3.

Tabela 3. Rezultati performansi

Algoritam	Precission	Odziv	F1 rezultat	Tačnost
CNN-NFC 3M	0,8099	0,8116	0,8094	0,8116
CNN-NFC 6M	0,8162	0,8145	0,8134	0,8145
CNN-NFC 9M	0,8067	0,8096	0,8054	0,8096
1D-CNN 3M	0,8661	0,8605	0,8584	0,8605
1D-CNN 6M	0,8671	0,8621	0,8608	0,8621
1D-CNN 9M	0,8642	0,8570	0,8547	0,8570
MLP 3M	0,8439	0,8365	0,8338	0,8364
MLP 6M	0,8464	0,8370	0,8314	0,8370
MLP 9M	0,8430	0,8358	0,8322	0,8358
SAE 3M	0,8428	0,8338	0,8284	0,8338
SAE 6M	0,8563	0,8485	0,8422	0,8485
SAE 9M	0,8450	0,8404	0,8348	0,8404

Najbolji rezultati postignuti su sa 1D-CNN modelom u konfiguraciji od 6M parametara. Ovaj model takođe postiže slične rezultate sa 3M parametara. Kompleksniji 1D-CNN modeli ne vode ka poboljšanju performansi. Zanimljivo je naznačiti da je CNN-NFC model imao najgore performanse od svih analiziranih modela, čak i u slučaju kada koristi dva konvoluciona sloja. Ovo sugeriše da je prisustvo potpuno povezanih slojeva ima mnogo bitniju ulogu od broja

konvolucionih filtera. MLP postiže pristojne performanse. Međutim, rezultati ostalih modela sugerišu da ostale neuralne mreže izvlače više korisnih obrazaca iz ulaznih podataka.

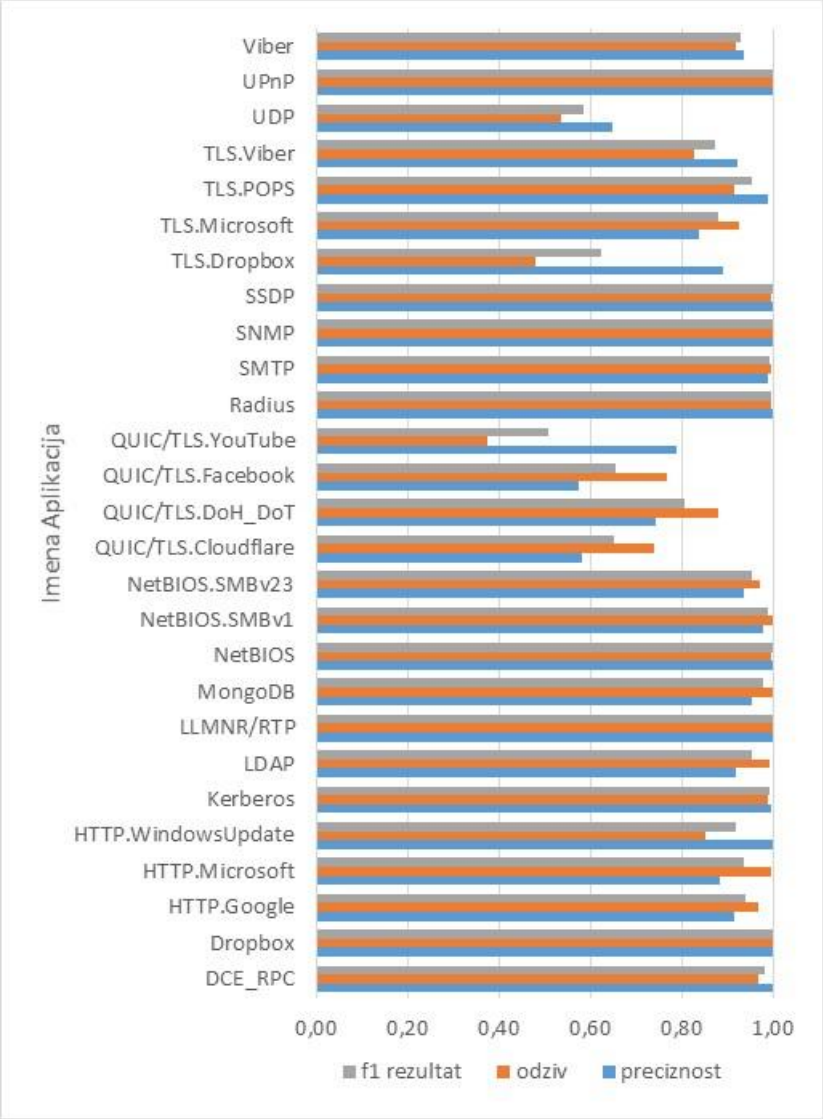
Kada se dobijeni uporede rezultati sa rezultatima iz rada [64], 1D-CNN postiže bolje performanse za *dataset* sa većom kompleksnošću. Iako je u [64] korišćen samo jedan konvolucioni sloj, ova analiza pokazuje da dodavanje dodatnih konvolucionih slojeva nije najvažniji faktor za unapređivanje performansi. Takođe treba naglasiti da je predobrada podataka *dataset-a* u ovom radu vršena na način koji eliminiše *bias* u rezultatima, što je dodatno uticalo da dobru generalizaciju modela.

U Tabeli 4 dat je izvještaj testiranja performansi 1D-CNN modela sa 6M parametara.

Tabela 4. Izvještaj performansi za 1D-CNN modela sa 6M parametara

	preciznost	**odziv**	**F1 rezultat**	**uzoraka za testiranje**
DCE_RPC	0,9979	0,9600	0,9786	2000
Dropbox	0,9997	1,0000	0,9998	3245
HTTP.Google	0,9175	0,9378	0,9275	498
HTTP.Microsoft	0,8841	0,9901	0,9341	2426
HTTP.WindowsUpdate	0,9857	0,8630	0,9203	2160
Kerberos	0,9955	0,9935	0,9945	2000
LDAP	0,9855	0,9850	0,9852	2000
LLMNR/RTP	0,9998	0,9993	0,9996	6000
MongoDB	0,9995	1,0000	0,9998	2000
NetBIOS	0,9990	0,9968	0,9979	4000
NetBIOS.SMBv1	0,9823	0,9965	0,9893	2000
NetBIOS.SMBv23	0,9328	0,9860	0,9587	4000
QUIC/TLS.Cloudflare	0,6566	0,7443	0,6977	6657
QUIC/TLS.DoH_DoT	0,8448	0,9060	0,8744	6663
QUIC/TLS.Facebook	0,6430	0,7838	0,7065	7036
QUIC/TLS.YouTube	0,7629	0,5031	0,6064	6384
Radius	1,0000	0,9960	0,9980	2000
SMTP	0,9990	0,9950	0,9970	2000
SNMP	1,0000	1,0000	1,0000	2000
SSDP	0,9990	0,9940	0,9965	2000
TLS.Dropbox	0,8802	0,5915	0,7075	4000
TLS.Microsoft	0,8929	0,9122	0,9025	11608
TLS.POPS	0,9850	0,9170	0,9498	2000
TLS.Viber	0,9328	0,8834	0,9075	2453
UPnP	1,0000	1,0000	1,0000	2000
Viber	0,9797	0,9733	0,9765	2581
aritmetička sredina	0,9202	0,9085	0,9115	
vagana aritmetička sredina	0,8671	0,8621	0,8608	
			Ukupno uzoraka:	99471
		tačnost	0,8621	

Detaljniji rezultati evaluacije za 1D-CNN 6M model su prikazani na Slici 22.



Slika 22. Pregled performansi 1D-CNN modela sa 6M parametara

Može se vidjeti da model postiže visoku preciznost, tačnost i F1 rezultat za aplikacije i protokole specifične za LAN (engl. *Local Area Network),* kao što su UPnP, SSDP, SNMP. Zadovoljavajući rezultati su takođe postignuti sa HTTP saobraćajem. Sa druge strane, pokazatelji performansi su relativno niži za veći dio saobraćaja koji koristi enkripciju (TLS i QUIC protokole). Generalno, enkripcija čini otežanom bilo kakvu tehniku klasifikaciju u realnom vremenu.

4. ANALIZA PERFORMANSI MODELA DUBOKOG UČENJA ZA KLASIFIKACIJU SAOBRAĆAJA NA ISCXVPN2016 *DATASET*-U

ISCXVPN2016 *dataset* izvučen je iz mrežnog saobraćaja [71], koji je korišćen u nekoliko dosadašnjih istraživanja za identifikaciju mrežnog saobraćaja na bazi mašinskog učenja [62], [37], [72], [73]. Saobraćaj je sakupljan na dva načina: putem regularne sesije (NON VPN) i korišćenjem VPN-a. Sakupljeni saobraćaj je organizovan u 14 kategorija, koje odgovaraju različitim tipovima saobraćaja poput *Browsing, Email, Chat, Streaming, File Transfer, VoIP i TraP2P*. U ovoj sekciji, za potrebe evaluacije performansi modela dubokog učenja za klasifikaciju saobraćaja, korišćen je samo NON VPN saobraćaj.

Labele za klasifikaciju (u nastavku NON VPN klasifikacija) ISCXVPN2016 saobraćaja korišćene u radu [62] su: *AIM Chat, Email, Facebook, FTPS, Gmail, Hangouts, ICQ, Netflix, SCP, SFTP, Skype, Spotify, Torrent, Tor, Vimeo, Voipbuster* i *Youtube*. Kreiran je *dataset* za ove labele i to direktno iz originalnih *.pcap* fajlova ISCXVPN2016 saobraćaja. Kako *Torrent* i *Tor* nisu dio NON VPN ISCXVPN2016 saobraćaja, ove klase nisu korišćene u analizi koja je predstavljena u nastavku. Dodatno je kreiran *dataset* labelama dobijenim korišćenjem *nDPI* metode (u nastavku *nDPI* klasifikacija).

Za pronalaženje optimalnih *hyper*-parametara modela dubokog učenja korišćen je *Keras Tuner* [74]. *Keras Tuner* je modul koji služi za olakšavanje pretrage optimalnih *hyper* parametara na način što se kreira konfiguracija sa vrijednostima *hyper*-parametara koji treba da budu testirani, pa se na tu konfiguraciju primjeni neki od algoritama za pretragu, poput *Bayesian Optimization*, *Hyperband* [75], and *Random Search*. U ovom radu, za optimizaciju modela korišćen je algoritam *Bayesian Optimization*. Ovaj algoritam naizmjenično bira parametre za testiranje, i nakon testiranja tih *hyper*-parametara, sledeće *hyper*-parametre bira upoređujući dobijene rezultate sa prethodnim *hyper*-parametrima.

Parametri koji su korišteni za optimizaciju CNN modela za NON VPN i *nDPI* klasifikaciju su *dropout*, *learning rate*, broj izlaznih filtera *dense* slojeva za MLP i SAE modele i broj izlaznih filtera za konvolucioni sloj kod CNN modela.

Za sve modele *dropout* parametar je testiran za veličine 0,2, 0,3 i 0,5 i testirane su vrijednosti *learning rate* algoritma za optimizaciju od 0,01, 0,001 i 0,0001.

Za MLP model predviđena su tri *dense* sloja sa istim brojem izlaznih parametara. Taj broj je tražen je u rasponu od 500 do 1500, sa korakom 100.

Kod SAE modela predviđeno je više *dense* slojeva za enkoder i dekoder. Ulazni parametar prvog *dense* sloja je 1500 a izlazni 1200. Broj izlaznih parametara sledećih slojeva se računa kreiranjem niza koji se dobija oduzimanjem broja 1200 sa testiranim brojem iz raspona od 300 do 700, sa korakom 50, i tako sve do pozitivnog broja najbližeg nuli. To bi u primjeru značilo da ako se iz raspona uzme broj 300, onda bi prvi, ulazni sloj imao izlazni parametar 1200, drugi 900, treći 600, četvrti 300, peti 300, šesti 600, sedmi 900 i osmi 1200.

Kod CNN modela broj izlaznih filtera za konvolucioni sloj je testiran u rasponu od 50 od 100 sa korakom 2. Posljednji sloj kod svih modela je *softmax* sloj.

Primjer konfiguracije *Keras Tuner*-a za CNN model je prikazan u sledećem kodu:

```
# Optimalni model će biti kreiran korišćenjem Keras Tunera
input_size = 1500
NUM_EPOCHS = 10

def build_model(hp):
    # Inicijalizacija sekvencijalnog API-ja i početak izrade modela

    model = Sequential()
    # Podešavanje droput parametra.
    # Odabir optimalne vrijednosti između 0.2, 0.3, 0.5
    dropout = hp.Choice('dropout', values=[0.2, 0.3, 0.5])

    # Podešavanje brojeva i jedinica u Conv1D sloju.
    # Broj jedinica: 10 - 50 sa korakom 10
    model.add(Conv1D(hp.Int("Conv1D_units_", min_value=50, max_value=100, step=2), 5, \
        input_shape = (input_size,1), activation = 'relu'))
    model.add(Dropout(dropout))
    model.add(MaxPooling1D(2))
    model.add(Flatten())

    # Dodavanje dense slojeva
    denses = [200, 100, 50]
    for dense in denses:
        model.add(Dense(dense, activation = 'relu'))
        model.add(Dropout(dropout))

    # Dodavanje izlaznog sloja.
    model.add(Dense(nb_classes, activation = 'softmax'))

    # Podešavanje learning rate
    # Odabir optimalne vrijednosti između 0.01, 0.001, or 0.0001
    hp_learning_rate = hp.Choice('learning_rate', values=[1e-2, 1e-3, 1e-4])
    model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=hp_learning_rate), \
        loss='categorical_crossentropy', metrics=['accuracy'], run_eagerly='true')

    return model

# Instanciranje tunera
tuner = kt.BayesianOptimization(build_model,
                     objective="val_accuracy",
                     directory="kt_dir",
                     project_name="kt_hyperband",
                     overwrite=True)

# Rezime konfiguracije za pretragu
Search space summary
Default search space size: 3
dropout (Choice)
{'default': 0.2, 'conditions': [], 'values': [0.2, 0.3, 0.5], 'ordered': True}
Conv1D_units_ (Int)
{'default': None, 'conditions': [], 'min_value': 50, 'max_value': 100, 'step': 2,
'sampling': None}
learning_rate (Choice)
{'default': 0.01, 'conditions': [], 'values': [0.01, 0.001, 0.0001], 'ordered': True}

stop_early = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5)

tuner.search(x_train, y_train, epochs=NUM_EPOCHS, validation_data=(x_val,y_val), \
    callbacks=[stop_early], verbose=0)
```

Nakon analize različitih konfiguracija modela pomoću *Keras Tuner*-a, pokazalo se da se za sve modele, MLP, SAE i CNN, ističu *dropout* vrijednosti 0,2 i *learning rate* od 0,0001.

Pri testiranju Keras Tuner-om za MLP modele najoptimalnije rezultate dala je konfiguracija sa izlaznim parametrom za *dense* slojeve od 800, tako da konačan MLP model ima tri *dense* sloja sa izlaznim parametrom vrijednosti 800.

Kod SAE modela optimalna testirana vrijednost osnove za izlazne parametre je 700, tako da konačan SAE model ima slojeve sa izlaznim parametrima 1200, 500, 500 i 1200.

Kod CNN modela, optimalni broj izlaznih filtera konvolucionog sloja je 60.

4.1 Kreiranje i treniranje modela za ISCXVPN2016 *dataset*

MLP model sadrži četiri *dense* sloja. Prvi *dense* sloj prihvata ulazni vektor ulaznih podataka veličine 1500 bajta. Izlazni parametri tri prva *dense* sloja su jednaki, i iznose 800. Posljednji *dense* sloj ima izlaznu veličinu jednaku broju klasa koje se koriste u *dataset*-u i koristi *softmax* aktivacionu funkciju. Između svih slojeva nalaze se *dropout* slojevi sa vrijednošću 0,2 i obezbjeđuju modelu izbjegavanje *overfitting*-a.

SAE model sadrži više *dense* slojeva. Prvi *dense* sloj prihvata vektor ulaznih podataka veličine 1500 bajta. Izlazni parametri *dense* slojeva se postepeno smanjuju ili povećavaju (1200, 500, 500 i 1200) kreirajući enkoder i dekoder. Posljednji *dense* sloj ima izlaznu veličinu jednaku broju klasa koje se koriste u *dataset*-u i koristi *softmax* aktivacionu funkciju. Između svih slojeva nalaze se *dropout* slojevi sa vrijednošću 0,2.

CNN model sadrži jedan konvolucioni sloj i četiri *dense* sloja. Konvolucioni sloj prihvata vektor ulaznih podataka veličine 1500 bajta, veličina kernela je 5 i koristi *ReLU* aktivacionu funkciju. Nakon ovoga podaci prolaze kroz slojeve sažimanja i ravnanja. Slijede tri *dense* sloja izvršavaju funkciju potpuno povezanih slojeva modela neuralne mreže, koristeći *ReLU* aktivacione funkcije. Njihove izlazne veličine su 200, 100 i 50. Posljednji *dense* sloj ima izlaznu veličinu jednaku broju klasa koje se koriste u *dataset*-u i koristi *softmax* aktivacionu funkciju. Između svih slojeva nalaze se *dropout* slojevi sa vrijednošću 0,2.

Modeli koriste *Adam* algoritam za optimizaciju. *Learning rate* kod svih modela je 0,0001, i svi se treniraju sa po 50 epoha. Trenirani modeli se čuvaju u posebnim fajlovima (*.h5* formata).

U tabeli 5 predstavljen je izvještaj performansi dobijen testiranjem CNN modela, po kojem se može vidjeti da se slabije performanse postižu samo za ICQ i AIM Chat klase. Prosječna tačnost je dosta visoka i iznosi oko 0,94.

Tabela 5. Rezultati CNN modela na NON VPN ISCXVPN2016 *dataset*-u.

	preciznost	**odziv**	**F1 rezultat**	**uzoraka za testiranje**
AIM Chat	0,6342	0,6037	0,6186	767
Email	0,7683	0,7216	0,7442	5917
Facebook	0,9744	0,9366	0,9551	21628
FTPS	0,9896	0,9998	0,9947	6084
Gmail	0,9283	0,8432	0,8837	1951
Hangouts	0,9436	0,9618	0,9526	20583
ICQ	0,5569	0,5527	0,5548	664
Netflix	0,9843	0,9646	0,9743	6158
SCP	0,9761	0,8150	0,8883	6724
SFTP	0,9988	0,9908	0,9948	12247
Skype	0,8949	0,9569	0,9249	33180
Spotify	0,8572	0,7812	0,8175	1568
Vimeo	0,9142	0,9665	0,9396	3736
Voipbuster	0,9993	0,9967	0,9980	10000
Youtube	0,9458	0,9380	0,9419	6550
aritmetička sredina	0,8911	0,8686	0,8789	
vagana aritmetička sredina	0,9381	0,9374	0,9369	
			Ukupno uzoraka:	137757
			tačnost 0,9374	

U tabeli 6 predstavljen je izvještaj performansi dobijen testiranjem modela na *dataset*-u kreiranom nDPI alatom. Iz tabele se može vidjeti da klasa *Facebook*, i klase sa prefiksom protokola TLS daju slabije rezultate. Ipak, prosječna tačnost je i ovdje visoka, i iznosi 0,93.

Tabela 6. Izvještaj performansi za CNN model sa nDPI klasifikacijom treniran ISCXVPN2016 *dataset*-om

	preciznost	**odziv**	**F1 rezultat**	**uzoraka za testiranje**
BitTorrent	0,9914	0,9966	0,9940	581
Dropbox	1,0000	1,0000	1,0000	2000
Facebook	0,4510	0,4600	0,4554	50
Google	0,8906	0,9521	0,9203	1274
HTTP.AmazonAWS	1,0000	0,7634	0,8659	93
HTTP.NetFlix	0,9699	0,9980	0,9837	2000
HTTP.Spotify	1,0000	1,0000	1,0000	27
HTTP.WindowsUpdate	0,9831	1,0000	0,9915	174
IMAPS.Google	0,9752	0,9387	0,9566	669
LLMNR	1,0000	1,0000	1,0000	2000
NTP	1,0000	0,9935	0,9967	461
NetBIOS	0,9990	0,9880	0,9935	2000

NetBIOS.SMBv1	0,9989	1,0000	0,9994	871
NetBIOS.SMBv23	0,7556	1,0000	0,8608	34
QUIC.Google	0,8671	0,6765	0,7601	405
RTP	0,9940	0,9925	0,9932	2000
SMTPS	0,9507	0,9060	0,9278	319
SNMP	1,0000	1,0000	1,0000	1492
SSDP	0,9990	1,0000	0,9995	2000
Skype_Teams	0,9975	1,0000	0,9988	2000
Skype_Teams.SkypeCall	0,9375	0,9836	0,9600	61
Spotify	0,9889	1,0000	0,9944	89
TLS.AmazonAWS	0,9773	0,7049	0,8190	61
TLS.Azure	0,3566	0,7667	0,4868	60
TLS.Facebook	0,7152	0,7236	0,7194	1527
TLS.Gmail	0,6000	0,3692	0,4571	130
TLS.Google	0,7578	0,6790	0,7162	2000
TLS.GooglePlus	0,8922	0,9705	0,9297	1390
TLS.Microsoft	0,4286	0,4500	0,4390	60
TLS.Skype_Teams	0,5000	0,3605	0,4190	147
TLS.Vimeo	0,5048	0,5638	0,5327	94
TLS.YouTube	0,2662	0,4066	0,3217	182
aritmetička sredina	0,8359	0,8326	0,8279	
vagana aritmetička sredina	0,9312	0,9312	0,9291	
			Ukupno uzoraka:	26251
			tačnost	0,9293

4.2 Analiza rezultata

Na osnovu rezultata dobijenih treniranjem različitih modela i konfiguracija kreirana je Tabela 7 sa uporednim rezultatima performansi testiranih modela za ISCXVPN2016 *dataset*:

Tabela 7. Performanse modela za ISCXVPN2016 *dataset*.

	ISCXVPN2016							
	NON VPN				**nDPI**			
	Preciznost	**Odziv**	**F1**	**Tačnost**	**Preciznost**	**Odziv**	**F1**	**Tačnost**
CNN	0,9381	0,9374	0,9370	0,9374	0,9312	0,9293	0,9291	0,9293
SAE	0,9412	0,9383	0,9389	0,9383	0,9247	0,9215	0,9216	0,9215
MLP	0,9314	0,9304	0,9298	0,9304	0,9237	0,9176	0,9190	0,9176

Ono što se u tabeli da primjetiti jeste da su rezultati svih modela približni. Nešto bolje performanse postignute su na NON VPN *dataset*-u u odnosu na *dataset* sa labelama aplikacija koji je generisan nDPI alatom. Rezultati su za nijansu slabiji od rezultata predstavljenih u [62].

SAE model mašinskog učenja se po performansama pokazao nešto bolji od CNN modela mašinskog učenja, što nije slučaj u radu [62]. Međutim, treba uzeti u obzir da postoje razlike u pogledu broja epoha treniranja i odabira određenih *hyper*-parametara.

U analizi je korišćen i MLP model mašinskog učenja. Dobijeni rezultati performansi pokazuju da i ovakvi modeli mašinskog učenja, iako relativno jednostavne strukture, mogu dati dobre rezultate (zavisno od kompleksnosti *dataset*-a), i da ih ne bi trebalo zanemarivati.

5. APLIKACIJE ZA TRENIRANJE MODELA MAŠINSKOG UČENJA I PREDIKCIJU

Za potrebe rada napravljene su dvije aplikacije. Namjena prve aplikacije je da omogući da se izvrši proces treniranja modela korišćenjem naizmjenično uzetog saobraćaja u *.pcap* formatu. Druga aplikacija može sa istreniranim modelima izvršiti predikciju naizmjenično uzetog saobraćaja, takođe u *.pcap* formatu, i predstaviti uspješnost predikcije.

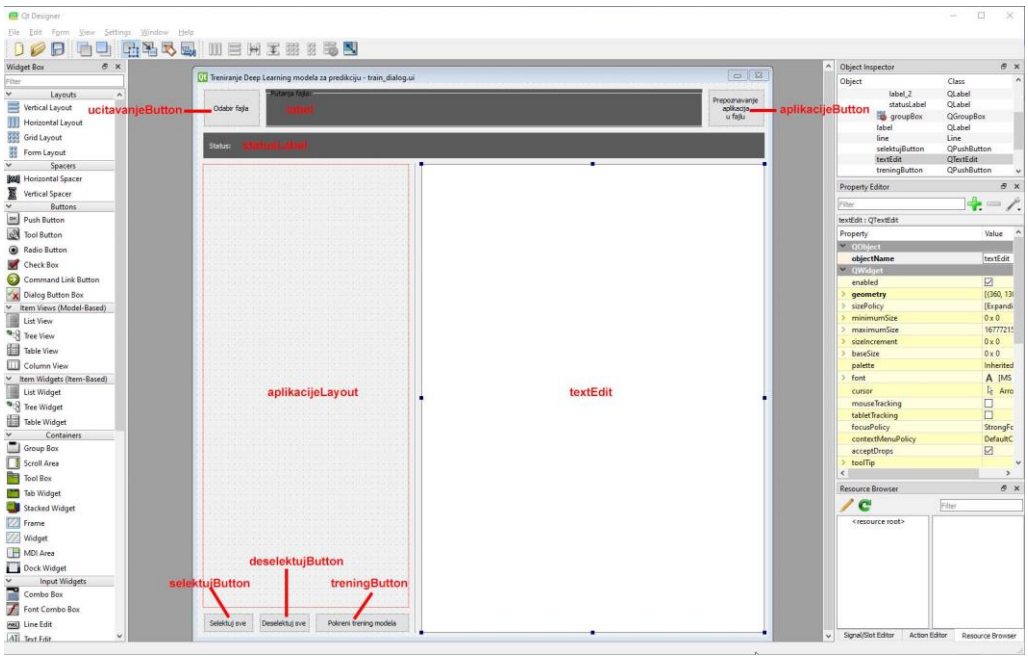
5.1 Grafički interfejs aplikacija

Za potrebe izrade grafičkog interfejsa aplikacija korišćen je GUI *framework PyQt*, verzija 6.3. *PyQt* je set modula za povezivanje *Python* programskog jezika sa *Qt* aplikativnim *framework*-om i moguće ga je koristiti na svim platformama koje podržavaju *Qt*, uključujući *Windows, macOS, Linux, iOS* i *Android*. Sami moduli za povezivanje sa *Python*-om sadrže preko 1000 klasa [76]. *Qt* takođe uključuje *Qt Designer*, grafičko dizajnersko okruženje. *PyQt* može da generiše kod za *Python* iz *Qt Designer*-a. Na taj kod je moguće dodati nove GUI kontrole pisane u *Python*-u ili dodatno kreirane u *Qt Designer*-u.

5.2 Aplikacija za treniranje modela

Aplikacija je zamišljena da za odabrani fajl u *.pcap* formatu izvršava proces prepoznavanja aplikacija koje fajl sadrži, prebrojava pakete po aplikacijama, bira željene aplikacije za treniranje, obrađuje podatke za treniranje i konačno, izvršava treniranje modela mašinskog učenja. Za početak, biće opisan korisnički interfejs aplikacije.

Aplikacija je dizajnirana u *Qt Designer*-u. Osnovi prozor grafičkog interfejsa prikazan je na Slici 23.



Slika 23.Dizajn prozora aplikacije za treniranje

Sa dugmetom *Odabir fajla* vrši se otvaranje forme za biranje fajla. Prilikom odabira prikazuje se putanja fajla u objektu *label*. Pritiskom dugmeta *Prepoznavanje aplikacija u fajlu* aktivira se funkcija koja vrši traženje aplikacija iz odabranog fajla. Prepoznate aplikacije se prikazuju u prozoru *aplikacijeLayout*.

Dugmad *Selektuj sve* i *Deselektuj sve* služe za izbor svih ponuđenih aplikacija ili za njihovo uklanjanje. Izabrane aplikacije se dalje prosleđuju procesu za treniranje modela pomoću dugmeta *Pokreni treniranje modela*. Proces obrade podataka i treniranja se prikazuju u objektu *textEdit*.

StatusLabel prikazuje poruke o fazama tokom pripreme, obrade i treniranja modela.

Od specifičnih biblioteka u kodu aplikacije korišćen je *PyQt6*, a od modula iz *qtWidgets* uzeti su *QApplication*, *QMainWindow*, *QLabel*, *QPushButton*, *QFileDialog*, *QVBoxLayout*, *QTextEdit*, *QCheckBox*, *QTreeWidget*, *QTreeWidgetItem*.

QApplication je klasa koja upravlja tokom GUI-ja aplikacije i njenim glavnim podešavanjima. Klasa *QMainWindow* obezbjeđuje glavni prozor aplikaciji. *QLabel* vidžet obezbjeđuje prikaz teksta ili slike. *QPushButton* vidžet obezbjeđuje komandno dugme. *QFileDialog* klasa obezbjeđuje dijalog koji omogućava korisnicima da odaberu fajlove ili direktorijume. *QVBoxLayout* klasa uređuje vidžete po vertikalnoj osi. *QTextEdit* klasa obezbjeđuje vidžet u kom se upisuje ili prikazuje tekst. *QCheckBox* vidžet obezbjeđuje *checkbox* sa ili bez pratećeg teksta. *QTreeWidget* klasa obezbjeđuje pregled koji koristi predefinisani tree model. *QTreeWidgetItem* klasa omogućava da se neki objekat integriše unutar *QTreeWidget*-a

Uic je kompajler korisničkog interfejsa (engl. *User Interface Compiler*) za *Qt Widgets*. *uic* isčitava *.xml* format definicija korisničkog interfejsa (koji se nalazi u *.ui* fajl) onako kako ga je generisao *Qt Designer*, i kreira odgovarajući C++ *header file*, tj. fajl predviđen da bude uključen na početku programa i sadrži tipove podataka i promjenljive koje će funkcije koristiti u programu.

```
from PyQt6 import QtGui, uic
from PyQt6.QtCore import Qt, QObject, QThread, pyqtSignal
```

Iz modula *QtCore* korišćene su klase *QObject*, *QThread* i *pyqtSignal*. *QObject* klasa je osnovna klasa svih *Qt* objekata. *QThread* (nit) klasa služi za upravlja nitima. Jedna od ključnih karakteristika *Qt*-a je korišćenje signala ili obavještenja za komunikaciju između objekata unutar programa. *pyqtSignal* je klasa koja može kreirati različite tipove signala, i između ostalih koristi *connect()*, *disconnect()* i *emit()* metode. Korišćenjem ovakvih signala jedan objekat može poslati poruku drugom da započne ili prekine izvršavanje neke funkcije, pošalje neku vrijednost, ispiše određeni tekst itd.

Klasa glavnog prozora aplikacije sadrži sledeće:

```
# Korisnički interfejs sa odabir fajla
class UI(QMainWindow):
    def __init__(self):
        super(UI, self).__init__()
```

```
        # Učitavanje ui fajla
        uic.loadUi("./Utilities/train_dialog.ui", self)

        # Define Widgets
        self.putanjaLabel = self.findChild(QLabel, "label")
        self.statusLabel = self.findChild(QLabel, "statusLabel")
        self.ucitavanje = self.findChild(QPushButton, "ucitavanjeButton")
        self.labelizacija = self.findChild(QPushButton, "aplikacijeButton")
        self.labelizacija.setDisabled(True)
        self.selektuj = self.findChild(QPushButton, "selektujButton")
        self.selektuj.setDisabled(True)
        self.deselektuj = self.findChild(QPushButton, "deselektujButton")
        self.deselektuj.setDisabled(True)
        self.treniraj = self.findChild(QPushButton, "treningButton")
        self.treniraj.setDisabled(True)
        self.statusLabel.setText("Potrebno je odabrati fajl, pa pokrenuti prepoznavanje
               aplikacija...")

        # Definisanje akcija dugmadi
        self.ucitavanje.clicked.connect(self.otvaranje)
        self.labelizacija.clicked.connect(self.preprocessing)
        self.treniraj.clicked.connect(self.odabrane_app)
        self.selektuj.clicked.connect(self.all_app)
        self.deselektuj.clicked.connect(self.none_app)

        # Definisanje okruženja aplikacije
        self.aplikacijeLayout = self.findChild(QVBoxLayout, "aplikacijeLayout")
        self.listA = None

        # Definisanje okruženja za ispis teksta sesije za treniranje
        self.treningText = self.findChild(QTextEdit, "textEdit")
        text=open('Utilities/Uputstvo.txt').read()
        self.treningText.setPlainText(text)

        # Definisanje varijabli
        self.dict_name2label = dict()
        self.counts = dict()
        self.checked_items = [] # lista za cekirane aplikacije

        # Tekst sa terminala
        sys.stdout = Stream(newText=self.onUpdateText)

        # Prikazivanje aplikacije
        self.show()
```

U kodu se može vidjeti da je pomoću *uic* učitan dizajn aplikacije iz *train_dialog.ui* fajla. Učitana su sva potrebna dugmad i drugi elementi.

Za definisanu dugmad povezane su funkcije koje trebaju da se izvršavaju.

U definisanom *QTextEdit* elementu ispisuje se tekst iz fajla *Uputstvo.txt*, gdje je dato malo uputstvo za korišćenje aplikacije i neke specifičnosti parametara koje se u aplikaciji koriste.

Posebno su definisana dva prazna *dictionary* elementa za smještanje naziva i broja paketa pronađenih aplikacija, i lista gdje će biti evidentirane odabrane aplikacije.

Akcija za dugme koje treba da otvori fajl nalazi se u sledećoj funkciji:

```
    # Otvaranje dijaloga za odabir .pcap fajla
    def otvaranje(self):
        clean_t()
        # Dijalog za otvaranje fajla
        fname, _ = QFileDialog.getOpenFileName(self, "Open File", "", \
               "PCAP files (*.pcap)")

        # Prikazivanje imena fajla i omogućavanje sledećeg dugmeta
        if fname:
            # Skrivanje dugmadi za kontrolu aplikacija
            self.selektuj.setDisabled(True)
            self.deselektuj.setDisabled(True)
            self.treniraj.setDisabled(True)
            # Ispis putanje i aktivacija sledećeg dugmeta
            self.putanjaLabel.setText(str(fname))
            self.labelizacija.setDisabled(False)
```

Sa *QFileDialog* se vrši odabir fajla, a ukoliko je neki fajl odabran omogućavaju se ili onemogućavaju kontrole drugih elemenata.

Akcija dugmeta za pokretanje učitavanja i predobrade fajla pokreće funkciju *predprocessing*:

```
  # Predobrada fajlova
    def preprocessing(self):
        self.treningText.clear()
        clean_t()

        # Dugmad koja treba ugasiti
        self.labelizacija.setDisabled(True)

        self.filepath = self.putanjaLabel.text()

        self.worker = PreprocessingThread(self.filepath, self.statusLabel, \
                self.dict_name2label)
        self.worker.finished.connect(self.odabir)
        self.worker.finished.connect(self.worker.quit)
        self.worker.finished.connect(self.worker.deleteLater)
        self.worker.start()
```

Osim manje promjene na elementima interfejsa aplikacije, ova funkcija pokreće *nit PreprocessingThread*. Signal *finished* poziva funkciju *odabir*.

Odabir aplikacija koje se pronađu tokom izvršavanja niti *PreprocessingThread*-a vrši se u funkciji *odabir*:

```
    # Funkcija koja kreira interfejs za odabir aplikacija za treniranje
    def odabir(self):
        self.treningText.clear()
        self.filter_liste()

        self.statusLabel.setStyleSheet('')

        # Otkrivanje dugmadi za kontrolu aplikacija
        self.selektuj.setDisabled(False)
        self.deselektuj.setDisabled(False)
```

```
        self.treniraj.setDisabled(False)

        self.statusLabel.setText("Prikazuju se pronađene aplikacije...")
        sleep(2)

        # QTreeWidget je fleksibilan za kreiranje liste aplikacija sa cekboksovima
        if not self.listA:
            self.listA = QTreeWidget()
            self.aplikacijeLayout.addWidget(self.listA)
            self.listA.setColumnCount(3)
            self.listA.setHeaderLabels(['Indeks','Aplikacija','Broj paketa'])
            self.listA.resizeColumnToContents(0)

        # Brisanje moguceg prethodnog sadrzaja QTreeWidget-a
        self.listA.clear()

        for key, val in self.counts.items():
            print("{} : {} paketa".format(key, val))
            item = QTreeWidgetItem()
            item.setCheckState(0, Qt.CheckState.Unchecked)
            item.setData(1, Qt.ItemDataRole.UserRole, id(item))
            item.setText(1, key)
            item.setText(2, str(val))
            self.listA.addTopLevelItem(item)

        if len(self.counts) == 0:
            self.statusLabel.setText("Aplikacije u .pcap fajlu nemaju dovoljno paketa za \
                   treniranje. Odaberite drugi .pcap fajl.")
            self.statusLabel.setStyleSheet('color: rgb(255, 0, 0);')
        else:
            self.statusLabel.setText("Odaberite aplikacije koje ce se koristiti za \
                   klasifikaciju i pokrenite trening modela")
```

Izbor se omogućava na način što se unutar objekta *QTreeWidget* generišu nizovi sa *checkbox*-ovima, imenima aplikacija i brojem paketa. Broj paketa je ostavljen kako bi korisnici aplikacije za treniranje mogli da imaju bolji uvid pri izboru. Minimum paketa koji je predviđen pri izboru aplikacije je 10. Aplikacije sa manjim brojem paketa neće biti prikazane na listi za *odabir*. Ukoliko nema aplikacija sa dovoljnim brojem paketa za treniranje, ispisuje se odgovarajuća poruka.

Funkcija *all_app* je povezana sa dugmetom koji vrši označavanje *check* opcije svih aplikacija na listi.

```
    # Odabira sve ponudjene aplikacije
    def all_app(self):
        def recurse(parent_item):
            for i in range(parent_item.childCount()):
                item = parent_item.child(i)
                item.setCheckState(0, Qt.CheckState.Checked)
         recurse(self.listA.invisibleRootItem())
```

Funkcija *none_app* vrši suprotnu operaciju prethodnoj funkciji, i uklanja *check* opciju sa svih aplikacija na listi.

```
    # Uklanja sve ponudjene aplikacije iz odabira
    def none_app(self):
        def recurse(parent_item):
            for i in range(parent_item.childCount()):
                item = parent_item.child(i)
                item.setCheckState(0, Qt.CheckState.Unchecked)
         recurse(self.listA.invisibleRootItem())
```

Sa funkcijom *odabrane_app* se uzima lista čekiranih aplikacija, data lista se sortira i upisuje u dictionary *dict_name2label*.

```
    # Funkcija koja provjerava koje su aplikacije odabrane, i kreira konacnu listu
    # aplikacija za treniranje
    def odabrane_app(self):
        # resetovanje potrebnih listi i interfejsa
        self.statusLabel.setStyleSheet('')
        self.checked_items = []
        self.dict_name2label = {}

        def recurse(parent_item):
            for i in range(parent_item.childCount()):
                item = parent_item.child(i)
                print (i, item.text(1))
                if item.checkState(0) == Qt.CheckState.Checked:
                    self.checked_items.append(item.text(1))

                self.checked_items.sort()

        recurse(self.listA.invisibleRootItem())
        for app in self.checked_items:
            self.dict_name2label[app + '.pcap'] = app

        if len(self.checked_items) > 4:
            # Skrivanje dugmadi, u daljem procesu nisu vise potrebna
            self.ucitavanje.setDisabled(True)
            self.selektuj.setDisabled(True)
            self.deselektuj.setDisabled(True)
            self.treniraj.setDisabled(True)

            # Redraw liste aplikacija QTreeWidget-a
            self.listA.clear()
            self.listA.setColumnCount(2)
            self.listA.setHeaderLabels(['Indeks','Aplikacija','Broj paketa'])
            for i, app in enumerate(self.checked_items):
                item = QTreeWidgetItem()
                item.setData(0, Qt.ItemDataRole.UserRole, id(item))
                item.setText(0, str(i))
                item.setText(1, app)
                item.setText(2, str(self.counts[app]))
                self.listA.addTopLevelItem(item)
            self.listA.resizeColumnToContents(0)

            # Sledeći korak je treniranje modela
            self.statusLabel.setText("Treniranje modela...")
            self.trening()

        elif len(self.checked_items) == 0:
            self.statusLabel.setText("Nije odabrana nijedna aplikacija za treniranje")
```

```
            self.statusLabel.setStyleSheet('color: rgb(255, 0, 0);')

        elif len(self.checked_items) <= 5:
            self.statusLabel.setText("Nije dabran dovoljan broj aplikacija za treniranje \
                      (Minimum je 5)")
            self.statusLabel.setStyleSheet('color: rgb(255, 0, 0);')
            self.none_app()

        print('Odabrano je', len(self.checked_items), 'aplikacija.')
        print('checked_items: ',self.checked_items)
```

Odabrane aplikacije se upisuju u *QTreeWidget* zajedno sa svojim indeksom i brojem paketa. Ukoliko nije odabrana nijedna aplikacija, ili nije odabran minimalno definisani broj aplikacija za treniranje (5), ispisuje se odgovarajuća poruka.

Funkcija *trening* prvo čuva *dictionary* sa odabranim aplikacijama u skripti *apps.py*. Ova skripta će se moći kasnije, sa treniranim modelima, koristiti za predikciju saobraćaja.

```
    # Funkcija koja poziva nit za trening modela
    def trening(self):

        # Konacna dict_name2label lista
        self.statusLabel.setText("Upis liste aplikacija u fajl za dalju upotrebu")
        with open('Utilities/apps.py','w') as data:
            data.write('# for app identification\n')
            data.write('## created from file: {}\n'.format(self.filepath))
            data.write('## date created: {}\n'.format(datetime.now()))
            data.write('dict_name2label = {\n')
            for key, val in self.dict_name2label.items():
                data.write("\t'{}': '{}',\n".format(key, val))
            data.write('\t}')

        self.treningText.clear()
        self.worker = TrainingThread(self.filepath, self.statusLabel, self.dict_name2label)
        self.worker.finished.connect(self.reset_ui)
        self.worker.finished.connect(self.worker.quit)
        self.worker.finished.connect(self.worker.deleteLater)
        self.worker.start()
```

Nakon ovog koraka poziva se nit *TrainingThread* koja izvršava glavni proces treniranja modela.

U niti *TrainingThread* se prvo vrši učitavanje *dataset*-ova u promjenljive za trening i validaciju. Kako je bitno prilagoditi dimenzije ulaznih nizova, vrši se *expand* dimenzija ulaznog niza, zavisno od tog da li je predviđen za treniranje sa CNN ili MLP/SAE modelima. Vrijednosti niza se postavljaju na *float32* numeričke vrijednosti. Vrši se i one-hot-encoding imena aplikacija kako bi se unaprijedila predikcija i klasifikacija modela. Posebno je dodata *EarlyStopping* [77] metoda sa periodom čekanja od 5 epoha, kako bi se izbjeglo dugo treniranje modela koje ne donosi rezultate. Treniranje je podešeno da se izvršava sa najviše 20 epoha, sa parametrima modela koji su definisani u fajlu *utils.py*. Tokom izvršavanja treniranja modela, nakon svake epohe, provjeravaju se gubici treninga i validacije, i model sa najboljim rezultatima čuva se u fajlu formata *.h5*, a po završetku treniranja ispisuje se i odgovarajuća poruka.

```
# Nit za treniranje modela
class TrainingThread(QThread):
    finished = pyqtSignal()
    def __init__(self, filepath, statusLabel, dict_name2label):
        QThread.__init__(self)
        self.filepath = filepath
        self.statusLabel = statusLabel
        self.dict_name2label = dict_name2label

    def run(self):
        # Odavde počinje proces treniranja
        ## Podaci će u procesu treniranja biti prilagođeni različitim modelima

        # učitavanje podataka
        x_train, y_train, x_val, y_val = load_data(self, 'training')

        # formatiranje nizova u skladu sa dimenzijama zahtjevanog prostora, i formatiranje
        # tipa varijable (za numpy)
        x_cnn = np.expand_dims(x_train, axis=2).astype(np.float32)
        x_val_cnn = np.expand_dims(x_val, axis=2).astype(np.float32)
        x_mlp_sae = np.array(x_train).astype(np.float32)
        x_val_mlp_sae = np.array(x_val).astype(np.float32)

        # one-hot-encoding imena aplikacija
        encoder = LabelEncoder()
        encoder.fit(y_train)
        class_labels = encoder.classes_

        # broj klasa u modelima
        self.nb_classes = len(class_labels)
        encoded_y_train = encoder.transform(y_train)
        y_train = np_utils.to_categorical(encoded_y_train)
        encoded_y_val = encoder.transform(y_val)
        y_val = np_utils.to_categorical(encoded_y_val)

        # Definisanje i treniranje modela masinskog ucenja
        batch_size = 32
        nb_epochs = 20

        def compiled_model(uncompiled_model):
            model = uncompiled_model
            model.summary()
            model.compile(
                optimizer = keras.optimizers.Adam(learning_rate=0.0001),
                loss = "categorical_crossentropy",
                metrics = ["accuracy"],
                run_eagerly = "true",)
            return model

        # EarlyStopping je postavljen kako bi se eliminisalo nepotrebno vrijeme dugog
        # treniranja
        es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=5)
        def ModelCheck(saved_model_file):
            checkpoint = ModelCheckpoint(saved_model_file, monitor='val_loss', \
                        save_best_only=True, verbose=1)
            return checkpoint

        ## prvo se trenira CNN model
        print("Treniranje CNN modela...")
        self.statusLabel.setText("Treniranje CNN modela...")
        model = compiled_model(cnn_model(self))

        # lokacija za čuvanje CNN modela
        saved_model_file = 'models/cnn_model.h5'.format('conv1d-cnn')
```

```
        # Keeping model in control points where function loss improves
        # Čuvanje modela u tačkama gdje se funkcija gubitaka poboljšava
        fit_history = model.fit(x_cnn, y_train, epochs=nb_epochs, batch_size=batch_size, \
                    validation_data=(x_val_cnn,y_val), \
                    callbacks=[ModelCheck(saved_model_file), es], verbose=2)

        print("Treniranje CNN modela je zavrseno.\n")
        self.statusLabel.setText("Treniranje CNN modela je zavrseno.")
        sleep(2)

        ## sledeci se trenira MLP model
        print("Treniranje MLP modela...")
        self.statusLabel.setText("Treniranje MLP modela...")
        model = compiled_model(mlp_model(self))

        # lokacija za čuvanje MLP modela
        saved_model_file = 'models/mlp_model.h5'.format('mlp')

        # Čuvanje modela u tačkama gdje se funkcija gubitaka poboljšava
        fit_history = model.fit(x_mlp_sae, y_train, epochs=nb_epochs, \
                    batch_size=batch_size, validation_data=(x_val_mlp_sae,y_val), \
                    callbacks=[ModelCheck(saved_model_file), es], verbose=2)
        print("Treniranje MLP modela je zavrseno.\n")
        self.statusLabel.setText("Treniranje MLP modela je zavrseno.")
        sleep(2)

        ## poslednji se trenira SAE model
        print("Treniranje SAE modela...")
        self.statusLabel.setText("Treniranje SAE modela...")
        model = compiled_model(sae_model(self))

        # lokacija za čuvanje SAE modela
        saved_model_file = 'models/sae_model.h5'.format('sae')

        # Čuvanje modela u tačkama gdje se funkcija gubitaka poboljšava
        fit_history = model.fit(x_mlp_sae, y_train, epochs=nb_epochs, \
                    batch_size=batch_size, validation_data=(x_val_mlp_sae,y_val), \
                    callbacks=[ModelCheck(saved_model_file), es], verbose=2)
        print("Treniranje SAE modela je zavrseno.\n")
        self.statusLabel.setText("Treniranje SAE modela je zavrseno.")
        sleep(2)
        print("Modeli su spremni za predikciju.\n")
        self.statusLabel.setText("Treniranje modela je zavrseno.")
        self.statusLabel.setStyleSheet('color: rgb(170, 255, 0);')
```

Po emitovanju signala *finished* iz date klase poziva se funkcija *reset_ui*, koja uklanja sve elemente iz elementa *QTreeWidget*. Takođe se deaktiviraju, odnosno, aktiviraju, potrebna dugmad.

```
    # Resetuje elemente grafickog prikaza na osnovne vrijednosti
    def reset_ui(self):
        # Brisanje sadržaja
        self.listA.clear()
        # Otkrivanje dugmadi
        self.ucitavanje.setDisabled(False)
        # Skrivanje dugmadi
        self.selektuj.setDisabled(True)
        self.deselektuj.setDisabled(True)
        self.treniraj.setDisabled(True)
```

Funkcija *filter_liste* služi za pozivanje niti *LabelingThread*, koja izvršava konačno kreiranje liste aplikacija koja će se koristiti za treniranje modela.

```
    # Funkcija koja poziva nit za filterisanje liste aplikacija
    def filter_liste(self):
        self.treningText.clear()
        self.worker = LabelingThread(self.filepath, self.statusLabel, \
                      self.dict_name2label, self.counts)
        self.worker.finished.connect(self.worker.quit)
        self.worker.finished.connect(self.worker.deleteLater)
        self.worker.start()

# Nit za kreiranje liste aplikacija za klasifikaciju
class LabelingThread(QThread):
    finished = pyqtSignal()
    def __init__(self, filepath, statusLabel, dict_name2label, counts):
        QThread.__init__(self)
        self.filepath = filepath
        self.statusLabel = statusLabel
        self.dict_name2label = dict_name2label
        self.counts = counts

    def run(self):
        # Konacno filterisanje i kreiranje liste aplikacija za klasifikaciju
        ## Bice odabrane samo one aplikacije koje su prosle preprocessing
        ## i imaju minimalno definisani broj paketa
        self.statusLabel.setText("Konačno filterisanje aplikacija za klasifikaciju")
        sleep(1)
        applications, packets = count_data(self)

        for i, app in enumerate(applications):
            self.counts[app] = packets[i]
        self.statusLabel.setText("Brojanje paketa za pronadjene aplikacije je zavrseno.")
        self.finished.emit()
```

Predobrada odabranog fajla vrši se unutar niti *PreprocessingThread*.

```
# Nit za proceduru predobrade odabranog fajla
class PreprocessingThread(QThread):
    finished = pyqtSignal()
    def __init__(self, filepath, label, dict_name2label):
        QThread.__init__(self)
        self.filepath = filepath
        self.statusLabel = label
        self.dict_name2label = dict_name2label

    def run(self):
        # NFStream priprema draft liste za aplikacije
        ndpi_dict(self)

        # NFStream prepoznavanje i predobrada .pcap fajlova
        ndpi_prepro(self)
        print(self.dict_name2label)
        self.finished.emit()
```

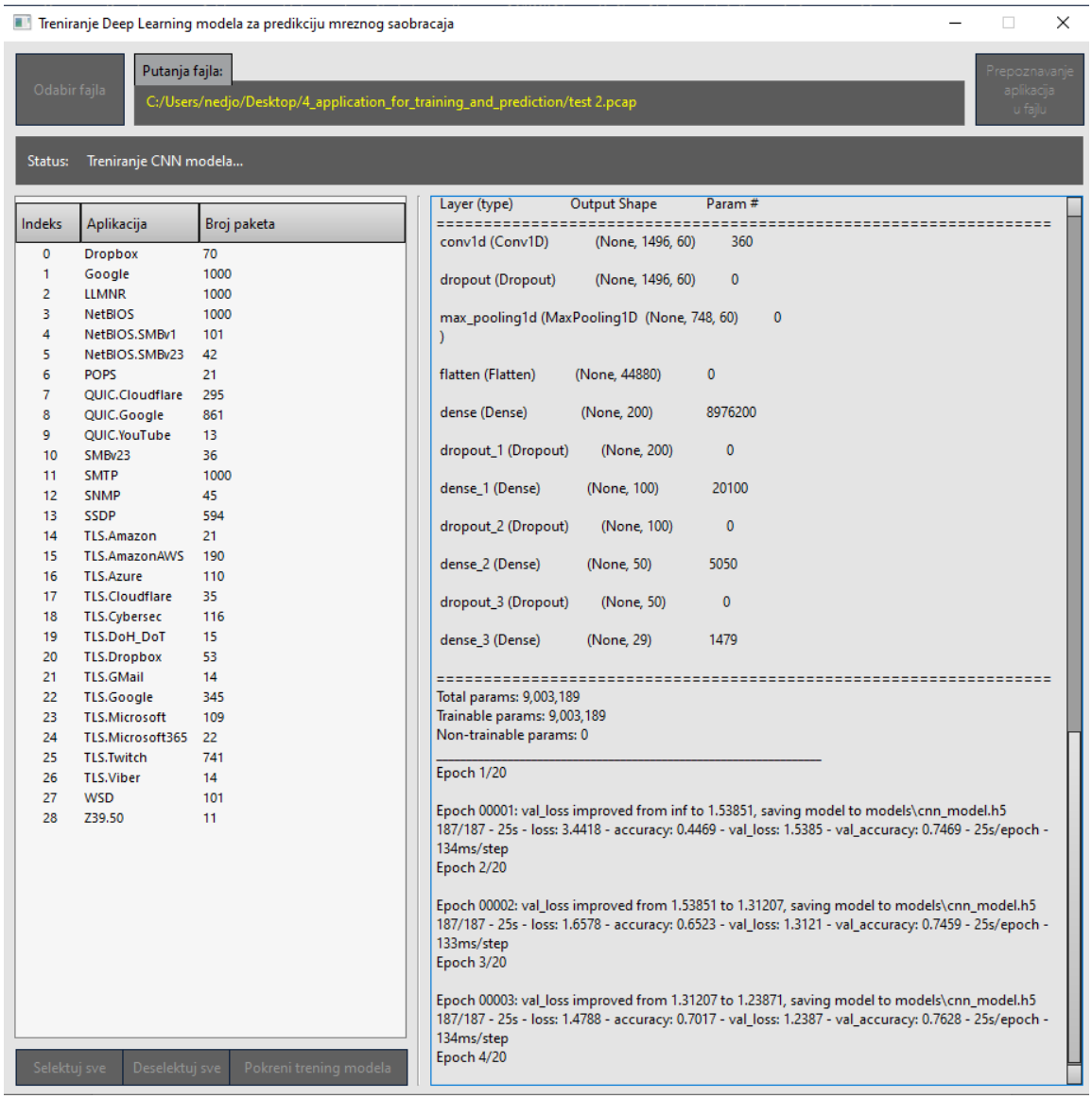
Za pokretanje aplikacije zadužen je sledeći dio koda.

```
# Inicijalizacija aplikacije
app = QApplication(sys.argv)
UIWindow = UI()

# otvaranje qss fajla
styleFile = open("./Utilities/Integrid.qss",'r')
with styleFile:
    qss = styleFile.read()
    app.setStyleSheet(qss)

# Pokretanje aplikacije
app.exec()
```

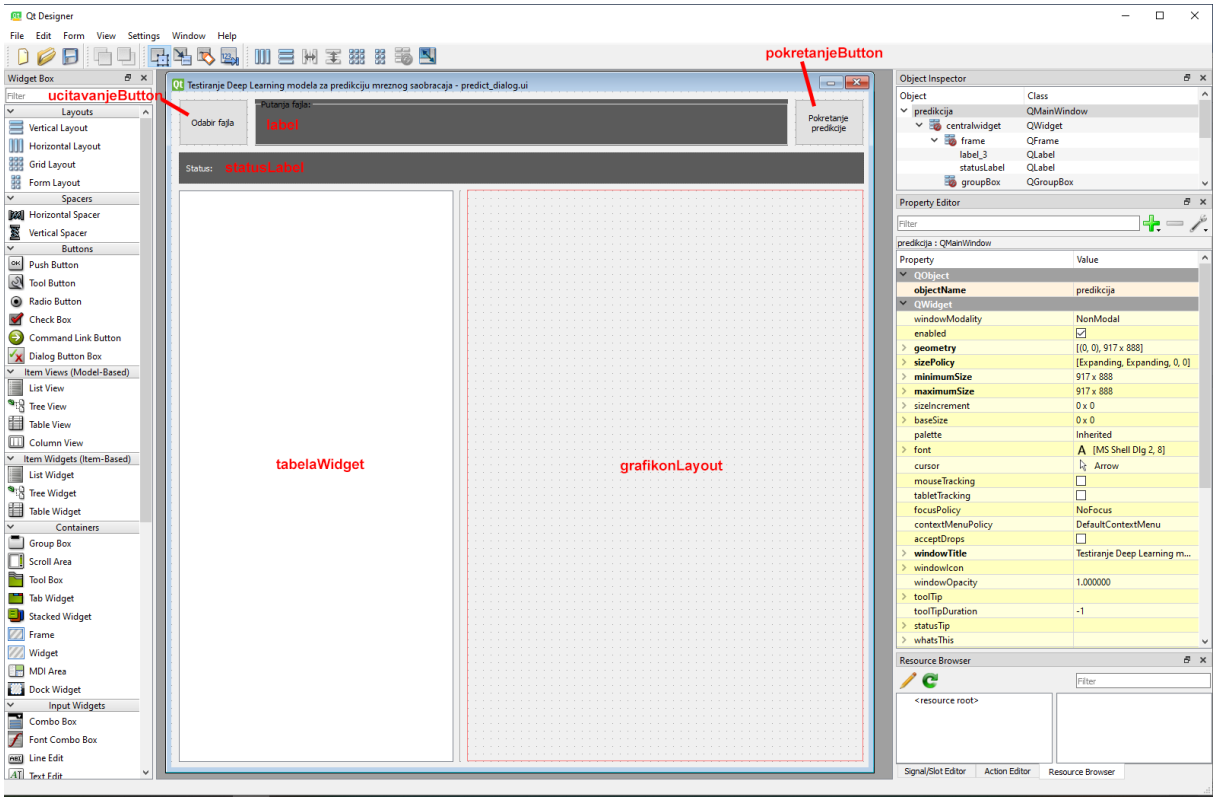
Posebno je definisan stil, odnosno tema aplikacije. Korišćena je tema *Integrid* [78]. Izgled glavnog prozora nakon pokretanja i izvršavanja aplikacije za treniranje prikazan je na Slici 24:



Slika 24.Aplikacija za treniranje modela

5.3 Aplikacija za predikciju

Aplikacija za predikciju je zamišljena tako da pomoću već istreniranih modela i unaprijed definisane liste aplikacija vrši predikciju klasa saobraćaja u zadatom *dataset*-u, a potom grafički predstavlja rezultate. Grafički interfejs aplikacije prikazan je na Slici 25.



Slika 25.Dizajn prozora aplikacije za predikciju

Sa dugmetom *Odabir fajla* vrši se otvaranje forme za biranje fajla. Putanja odabranog fajla se prikazuje u objektu *label*. Pritiskom na dugme *Pokretanje predikcije* pokreće se proces prepoznavanja aplikacija u fajlu. Prepoznate aplikacije se prikazuju u tabeli *tabelaWidget* i takođe grafički iscrtavaju u prozoru *grafikonLayout*.

StatusLabel prikazuje poruke o fazama tokom pripreme, obrade i treniranja modela.

Od biblioteke *PyQt6*, iz modula *qtWidgets* uzeti su *QTableWidget*, *QTableWidgetItem*. *QTableWidget* je klasa koja obezbjeđuje tabelarni pregled objekata. *QTableWidgetItem* klasa obezbjeđuje objekte koji treba da se koriste unutar *QTableWidget*. Iz biblioteke *matplotlib* korišćeni su sledeći vidžeti za kreiranje interaktivnih grafikona: *FigureCanvas* je oblast na kojoj se iscrtava grafikon, *NavigationToolbar2QT* kreira interfejs sa setom komandi koje se mogu koristiti za upravljanje grafikonom u *FigureCanvas*-u, a *Figure* je kreirana figura, u ovom slučaju grafikon.

Klasa glavnog prozora aplikacije sadrži sledeće:

```
# Korisnički interfejs sa odabir fajla
class UI(QMainWindow):
    def __init__(self):
        super(UI, self).__init__()

        # Load the ui file
        # Učitavanje ui fajlova
        uic.loadUi("./Utilities/predict_dialog.ui", self)

        # Define Widgets
        self.putanjaLabel = self.findChild(QLabel, "label")
        self.statusLabel = self.findChild(QLabel, "statusLabel")
        self.ucitavanje = self.findChild(QPushButton, "ucitavanjeButton")
        self.pokretanje = self.findChild(QPushButton, "pokretanjeButton")
        self.pokretanje.setDisabled(True)
        self.statusLabel.setText("Potrebno je odabrati fajl, pa pokrenuti predikciju...")

        # Definisanje tebele
        self.tabela = self.findChild(QTableWidget, "tabelaWidget")

        # Definisanje okruženja za grafikon
        self.layout = self.findChild(QVBoxLayout, "grafikonLayout")
        self.figure = plt.figure(figsize = (5, 25))
        self.figure.subplots_adjust(left=0.407,right=0.970,
                    bottom=0.091,top=0.977,
                    hspace=0.2,wspace=0.2)
        self.canvas = FigureCanvas(self.figure)
        self.layout.addWidget(NavigationToolbar(self.canvas, self))
        self.layout.addWidget(self.canvas)

        # Definisanje akcija dugmadi
        self.ucitavanje.clicked.connect(self.otvaranje)
        self.pokretanje.clicked.connect(self.predikcija)

        # Prikazivanje aplikacije
        self.show()
```

Funkcija *uic* učitava dizajn aplikacije iz *predict_dialog.ui* fajla. Za kreirane GUI dugmad objekte definisane su funkcije koje trebaju da se izvršavaju kao reakcija na klik događaj. Učitana je i klasa za tabelu, a za grafikon su definisani parametri koji bi trebali da ga adekvatno rasporede na zadatoj površini.

U funkciji *predikcija* kreira se glavna nit *MainThread* koja kao argumente uzima tekst atribut objekta *putanjaLabel* i objekte *statusLabel*, *tabela*, *figure* i *canvas*.

```
   # Funkcija koja poziva nit za predikciju mrežnog saobraćaja
    def predikcija(self):
        clean_t()
        self.worker = MainThread(self.putanjaLabel.text(), self.statusLabel, self.tabela, \
                    self.figure, self.canvas)
        self.worker.finished.connect(self.worker.quit)
        self.worker.finished.connect(self.worker.deleteLater)
        self.worker.start()
```

Sledećim kodom isprogramirana je funkcionalnost glavne niti:

```
# Glavni worker niti, ovdje se obavlja glavni dio programa
class MainThread(QThread):
    finished = pyqtSignal()
    def __init__(self, filepath, label, tabela, grafikon, grafikonCanvas):
        QThread.__init__(self)
        self.dict_name2label = dict_name2label
        self.filepath = filepath
        self.statusLabel = label
        self.tabela = tabela
        self.figure = grafikon
        self.canvas = grafikonCanvas

    def run(self):
        self.statusLabel.setText("Kreiranje praznih foldera.")
        create_folder(folders)
        sleep(1)
        self.statusLabel.setText("Brisanje sadrzaja foldera.")
        clean_folder(folders)
        sleep(1)

        # NFStream prepoznavanje i predobrada .pcap fajlova
        ndpi_prepro(self)

        # predikcija
        self.statusLabel.setText("Priprema paketa za predikciju...")
        sleep(1)
        x_test, y_test = load_data(self, 'prediction')

        # Provjera u slučaju da nema paketa za predikciju
        if x_test == None or y_test == None:
            self.statusLabel.setText("Nema dovoljno paketa za predikciju. Odaberite drugi \
                                     .pcap fajl")
            self.statusLabel.setStyleSheet('color: rgb(255, 0, 0);')
            finished = pyqtSignal()
        else:
            # formatiranje nizova u skladu sa dimenzijama zahtjevanog prostora
            x_cnn_test = np.expand_dims(x_test, axis=2).astype(np.float32)
            x_mlp_sae_test =  np.array(x_test).astype(np.float32)

            # sortirana lista klasa
            lista=[]
            [lista.append(x) for x in self.dict_name2label.values() if x not in lista]
            lista.sort()

            all_labels = []
            for value in self.dict_name2label.values():
                all_labels.append(value)

            # one-hot-encoding imena aplikacija
            encoder = LabelEncoder()
            encoder.fit(all_labels)
            class_labels = encoder.classes_

            # broj klasa u modelima
            nb_classes = len(class_labels)

            # broj testiranih uzoraka
            test_count = [y_test.count(i) for i in range(len(class_labels)) ]
            encoded_y_test = encoder.transform(y_test)
            y_test = np_utils.to_categorical(encoded_y_test, num_classes = nb_classes)
            in_classes = [item for item in range(0, nb_classes)] # lista indeksa labela
```

```
self.statusLabel.setText("Ucitavanje CNN modela...")
cnn_model = load_model('./models/cnn_model.h5')
print("CNN model.")
cnn_model.summary()
self.statusLabel.setText("Ucitavanje SAE modela...")
sae_model = load_model('./models/sae_model.h5')
print("SAE model.")
sae_model.summary()
self.statusLabel.setText("Ucitavanje MLP modela...")
mlp_model = load_model('./models/mlp_model.h5')
print("MLP model.")
mlp_model.summary()
self.statusLabel.setText("Racunanje performansi modela...")
cnn_preds = cnn_model.predict(x_cnn_test, batch_size=32,  verbose=0)
sae_preds = sae_model.predict(x_mlp_sae_test, batch_size=32,  verbose=0)
mlp_preds = mlp_model.predict(x_mlp_sae_test, batch_size=32,  verbose=0)

# pravi uzorci
y_true_labels = [np.argmax(t) for t in y_test]

# predicted samples
y_cnn_preds_labels = [np.argmax(t) for t in cnn_preds]
y_sae_preds_labels = [np.argmax(t) for t in sae_preds]
y_mlp_preds_labels = [np.argmax(t) for t in mlp_preds]

# broj testiranih uzoraka
test_count = [y_true_labels.count(i) for i in range(len(class_labels)) ]

# kreiranje formule za broja uzoraka po indeksima
cnn_count = [ 0 for _ in range(len(class_labels)) ]
sae_count = [ 0 for _ in range(len(class_labels)) ]
mlp_count = [ 0 for _ in range(len(class_labels)) ]

for i, j in zip(y_true_labels, y_cnn_preds_labels):
    if i == j:
        cnn_count[i] = cnn_count[i] + 1
for i, j in zip(y_true_labels, y_sae_preds_labels):
    if i == j:
        sae_count[i] = sae_count[i] + 1
for i, j in zip(y_true_labels, y_mlp_preds_labels):
    if i == j:
        mlp_count[i] = mlp_count[i] + 1

# normalizacija vrijednosti, lakši način za dobijanje grafikona u procentima
test_normal = [ 0 for _ in range(len(class_labels)) ]
cnn_normal = [ 0 for _ in range(len(class_labels)) ]
sae_normal = [ 0 for _ in range(len(class_labels)) ]
mlp_normal = [ 0 for _ in range(len(class_labels)) ]

def normalize(i):
    if test_count[i] > 0:
        raw = [test_count[i], cnn_count[i], sae_count[i], mlp_count[i]]
        return [float("{:.2f}".format(float(i)/max(raw))) for i in raw]
    else:
        return [0, 0, 0, 0]

for r in range(len(class_labels)):
    test_normal[r], cnn_normal[r], sae_normal[r], mlp_normal[r] = normalize(r)

print(test_normal)
print(cnn_normal)
print(sae_normal)
print(mlp_normal)
```

```
for i, item in enumerate(class_labels):
    a.add_row([i, item, cnn_count[i], sae_count[i], mlp_count[i], \
             test_count[i]])
    item_name = QTableWidgetItem(class_labels[i])
    item_cnn = QTableWidgetItem(str(cnn_count[i]))
    item_sae = QTableWidgetItem(str(sae_count[i]))
    item_mlp = QTableWidgetItem(str(mlp_count[i]))
    item_test = QTableWidgetItem(str(test_count[i]))

    # Elementi u tabeli su podešeni da budu poravnati na desno
    item_cnn.setTextAlignment(Qt.AlignmentFlag.AlignRight | \
                              Qt.AlignmentFlag.AlignVCenter)
    item_sae.setTextAlignment(Qt.AlignmentFlag.AlignRight | \
                              Qt.AlignmentFlag.AlignVCenter)
    item_mlp.setTextAlignment(Qt.AlignmentFlag.AlignRight | \
                              Qt.AlignmentFlag.AlignVCenter)
    item_test.setTextAlignment(Qt.AlignmentFlag.AlignRight | \
                               Qt.AlignmentFlag.AlignVCenter)
    self.tabela.setItem(i, 0, item_name)
    self.tabela.setItem(i, 1, item_cnn)
    self.tabela.setItem(i, 2, item_sae)
    self.tabela.setItem(i, 3, item_mlp)
    self.tabela.setItem(i, 4, item_test)
self.tabela.resizeColumnsToContents()
self.tabela.resizeRowsToContents()

# Crtanje grafikona
self.figure.clear()
ax = self.figure.add_subplot(111)

# Horizontal Bar Plot
barHeight = 0.25
barWidth = 0.25

# Postavljanje pozicije barova grafikona na X osi
br1 = np.arange(len(class_labels))
br2 = [y + barHeight for y in br1]
br3 = [y + barHeight for y in br2]
br4 = [y + barHeight for y in br3]

ax.barh(br1, cnn_normal, color ='r', height = barHeight, \
        edgecolor ='grey', label ='CNN')
ax.barh(br2, sae_normal, color ='g', height = barHeight, \
        edgecolor ='grey', label ='SAE')
ax.barh(br3, mlp_normal, color ='y', height = barHeight, \
        edgecolor ='grey', label ='MLP')
ax.barh(br4, test_normal, color ='b', height = barHeight, \
        edgecolor ='grey', label ='Broj uzoraka')

# Uklanjanje sploljnih ivica grafikona
for s in ['top', 'bottom', 'left', 'right']:
    ax.spines[s].set_visible(False)

# Dodavanje prostora između osa i labela
ax.xaxis.set_tick_params(pad = 5, labelsize = 8)
ax.yaxis.set_tick_params(pad = 10, labelsize = 8)

# Dodavanje ivica x i y ose
ax.grid(visible = True, color ='grey',
        linestyle ='-.', linewidth = 0.5,
        alpha = 0.2)

# Prikazivanje vrijednosti na vrhu
```

```
            ax.invert_yaxis()
            ax.set_ylabel('Aplikacija', fontweight ='bold', fontsize = 10)
            ax.set_xlabel('Predikcija (%)', fontweight ='bold', fontsize = 10)
            ax.yaxis.set_ticks([r + barWidth for r in range(len(class_labels))], \
                               class_labels)
            axfont = font_manager.FontProperties(style='normal', size=6)
            ax.legend(prop=axfont)
            ax.plot()
            self.canvas.draw()
            self.statusLabel.setText("Predikcija je zavrsena.")
            self.statusLabel.setStyleSheet('color: rgb(170, 255, 0);')
            self.finished.emit()
```

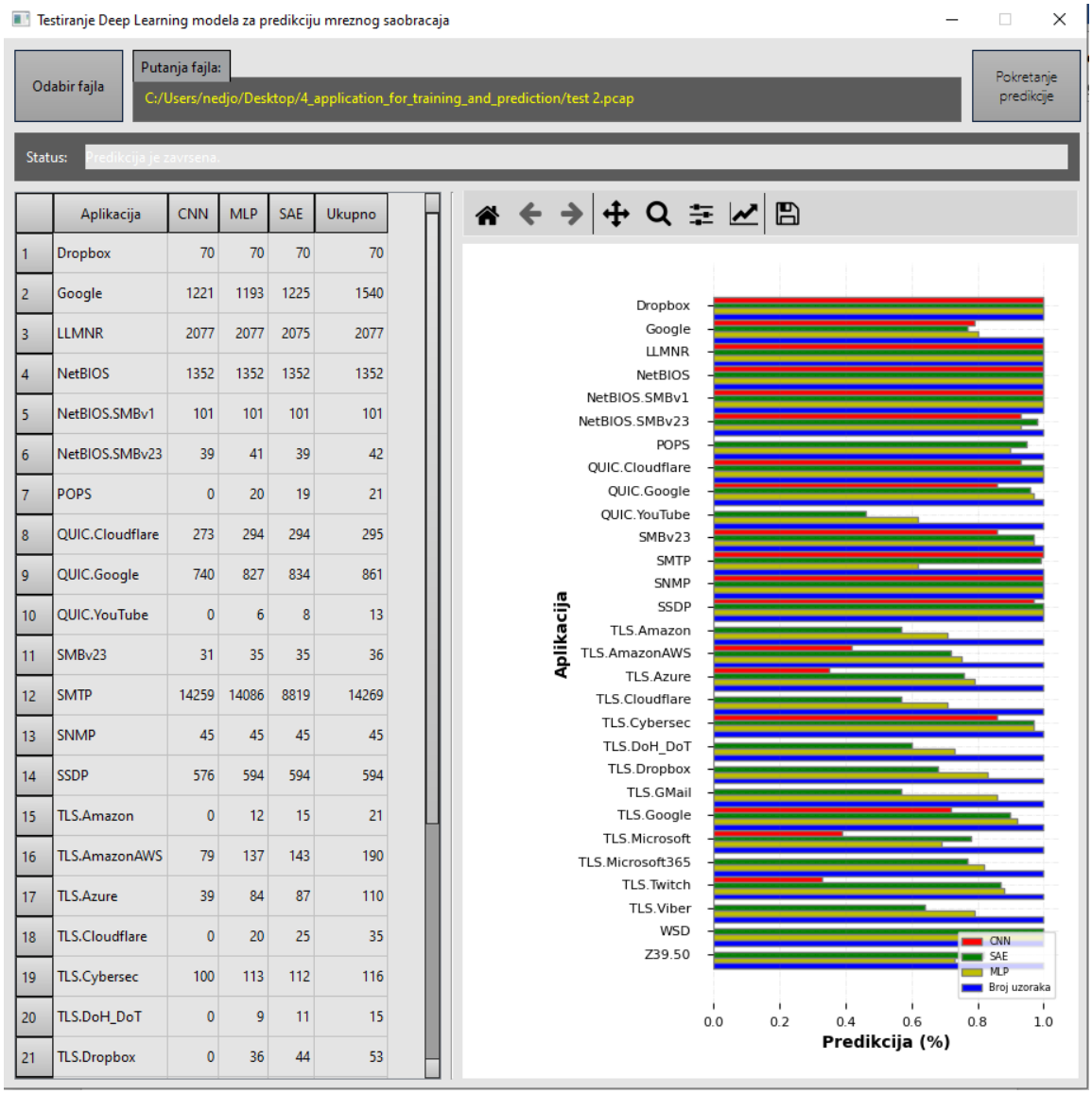
Nakon što se *clean_folder* funkcijom izvrši čišćenje sadržaja foldera koje GUI aplikacija koristi, vrši se prepoznavanje i predobrada *.pcap* fajlova.

Obrađeni fajlovi se učitavaju u *dataset* koji će se koristiti za predikciju. Najprije se provjerava se da li uopšte ima paketa za predikciju, i ukoliko ima, uzeti podaci se prilagođavaju dimenzijama ulaznog sloja za CNN, odnosno MLP i SAE modele.

Definišu se četiri prazne liste podataka i to: jedna za brojanje ukupnog broja paketa za testiranje, po aplikaciji (*test_count*), i tri za pronađeni broj paketa po aplikaciji i po modelima (*cnn_count*, *mlp_count*, *sae_count*). Ovi podaci se ispisuju uz ime aplikacije, po završetku predikcije.

Dodatno se definišu četiri prazne liste podataka (*test_normal*, *cnn_normal*, *sae_normal* i *mlp_normal*) u kojima će biti unesena vrijednost nakon normalizacije dobijenog broja paketa po modelima. Normalizacija se vrši u funkciji *normalize,* i to tako što se uzmu vrijednosti predikcije iz lista izbrojanih paketa po modelima, pa se konkretnu aplikaciju nađe procentualni odnos paketa ostalih modela u odnosu na model sa najvećom vrijednošću. Tako, recimo, ako za neku aplikaciju imamo vrijednost od 10 prepoznatih paketa za CNN, 5 paketa za SAE i 1 paket za MLP, procentualni odnos tih paketa po toj aplikaciji bio bi za CNN 100%, SAE 50%, a za MLP 10 % Normalizacija je praktična, jer se njome grafički mogu prikazati ujednačeni i pregledni odnosi predikcije.

Za grafikon je izabrano da bude tipa „*bar chart*“ sa četiri *bara*, za svaku od vrijednosti iz četiri normalizovane liste. Izgled glavnog prozora nakon pokretanja i izvršavanja aplikacije za predikciju prikazana je na Slici 26.



	Aplikacija	CNN	MLP	SAE	Ukupno
1	Dropbox	70	70	70	70
2	Google	1221	1193	1225	1540
3	LLMNR	2077	2077	2075	2077
4	NetBIOS	1352	1352	1352	1352
5	NetBIOS.SMBv1	101	101	101	101
6	NetBIOS.SMBv23	39	41	39	42
7	POPS	0	20	19	21
8	QUIC.Cloudflare	273	294	294	295
9	QUIC.Google	740	827	834	861
10	QUIC.YouTube	0	6	8	13
11	SMBv23	31	35	35	36
12	SMTP	14259	14086	8819	14269
13	SNMP	45	45	45	45
14	SSDP	576	594	594	594
15	TLS.Amazon	0	12	15	21
16	TLS.AmazonAWS	79	137	143	190
17	TLS.Azure	39	84	87	110
18	TLS.Cloudflare	0	20	25	35
19	TLS.Cybersec	100	113	112	116
20	TLS.DoH_DoT	0	9	11	15
21	TLS.Dropbox	0	36	44	53

Slika 26.Aplikacija za predikciju

5.4 Dodatne funkcije

Set funkcija koje su zajedničke za obije aplikacije smještene su u fajlovima *apps.py*, *nfs.py* i *utils.py*.

Fajl *apps.py* sadrži *dictionary* sa odabranim aplikacijama, dobijenim tokom procesa treniranja modela.

U fajlu *utils.py* nalazi se veći broj zajedničkih funkcija koje su objašnjene u prethodnim djelovima rada. Jedna od važnijih funkcija je *load_data*, koja služi za učitavanje paketa iz *.pickle* fajlova.

```
# Učitavanje svih fajlova iz .pickle foldera
def load_data(self, operacija):
    # ovdje se definiše minimalni broj paketa po aplikaciji
    max_data_nb = 1000
    train_rate = 0.75

    # todo_list se mora dodatno filterisati, kako bi bile obradjene samo odabrane
    # aplikacije
    todo_list = gen_todo_list(pickle_dir)
    tmp_todo = []
    print("\nPickle todo lista:\n")
    for file in todo_list:
        if re.split(pickle_dir + "|.pickle", file)[1] in self.dict_name2label.keys():
            tmp_todo.append(file)
            print(file)
    todo_list = tmp_todo
    sleep(10)

    X = []
    y = []
    X_val = []
    y_val = []

    if operacija == 'training':
        if len(todo_list) == 0:
            return None, None, None, None
        else:
            for counter, filename in enumerate(todo_list):
                (tmpX, tmpy) = load(filename)
                if len(tmpX) > 0:
                    tmpX , tmpy = tmpX[:max_data_nb], tmpy[:max_data_nb]
                    tmpX = processX(tmpX)
                    train_num = int(len(tmpX) * train_rate)
                    X.extend(tmpX[:train_num])
                    y.extend(tmpy[:train_num])
                    X_val.extend(tmpX[train_num:])
                    y_val.extend(tmpy[train_num:])
                    print('\rUčitavanje... {}/{}'.format(counter+1,len(todo_list)), \
                          end = '')
            print('\r{} Podaci su učitani.                '.format(len(todo_list)))
            return X, y, X_val, y_val

    elif operacija == 'prediction':
        if len(todo_list) == 0:
            return None, None
        else:
            for counter, filename in enumerate(todo_list):
                (tmpX, tmpy) = load(filename)
                if len(tmpX) > 0:
                    tmpX = processX(tmpX)
                    X.extend(tmpX)
                    y.extend(tmpy)
                    print('\rUčitavanje... {}/{}'.format(counter+1,len(todo_list)), \
                          end = '')
            print('\r{} \nPodaci su učitani.                '.format(len(todo_list)))
            return X, y
```

Ovdje se definiše i *max_data_nb* parametar, vrijednost koja ograničava maksimalni broj uzetih paketa za učitavanje. Kako je pri učitavanju podataka za treniranje potrebno odvojiti pakete za validaciju, definisana je posebna funkcija u odnosu na onu koja se koristi za predikciju, a koja sve pakete koristi za testiranje modela. Takođe, prilikom predikcije se ne ograničava broj uzetih paketa, već se uzimaju svi paketi.

Funkcija *count_data* ima ulogu da vrati broj paketa u *.pickle* fajlovima, i koristi je aplikacija za treniranje.

```
# brojanje paketa u .pickle fajlovima
def count_data(self):
    self.dict_name2label = dict()
    max_data_nb = 1000
    min_data_nb = 10
    todo_list = gen_todo_list(pickle_dir)
    applications = []
    packets = []

    for counter, filename in enumerate(todo_list):
        (tmpX, tmpy) = load(filename)
        tmpX , tmpy = tmpX[:max_data_nb], tmpy[:max_data_nb]
        # ovdje se definiše minimalni broj paketa po aplikaciji
        if len(tmpy) > min_data_nb:
            applications.append(tmpy[0])
            packets.append(len(tmpy))

    # paketi i aplikacije se moraju sortirati zajedno
    if len(applications) == 0:
        self.dict_name2label = dict()
        return applications, packets
    else:
        l = list(zip(applications, packets))
        l.sort()
        # 'unzip'
        applications, packets = zip(*l)
        for app in applications:
                self.dict_name2label[app + '.pcap'] = app
        return applications, packets
```

Parametri *max_data_nb* i *min_data_nb* se koriste da isfiltriraju listu aplikacija za koje će broj paketa biti ispisan. Funkcija vraća sortiranu listu aplikacija i broj paketa po aplikaciji.

Funkcija *ndpi_dict* služi za brojanje detekcija *nDPI* algoritma po aplikaciji i filtriranje aplikacija.

```
# Brojanje nDPI detekcija
def ndpi_dict(self):
    self.statusLabel.setStyleSheet('')
    self.statusLabel.setText("Kreiranje praznih foldera.")
    create_folder(folders)
    sleep(1)

    self.statusLabel.setText("Brisanje sadrzaja foldera.")
    clean_folder(folders)
    sleep(1)

    self.statusLabel.setText("Pokretanje modula NFStream...")
    sleep(1)
    filename = os.path.basename(self.filepath)
    # podjela .pcap fajlova korišćenjem nfstream application_name
    if self.filepath.endswith(".pcap"):
        # sprječava otvaranje više prozora u QT mainUI funkciji
        subprocess.run(['python', 'Utilities/nfs.py', self.filepath, nf_path])

        self.statusLabel.setText("NFStream prepoznavanje je zavrseno.")
        sleep(1)
```

```
    self.statusLabel.setText("Kreiranje draft liste za pronadjene aplikacije...")
    df = pd.read_csv(nf_path, header=0, sep=',')
    encoder = LabelEncoder()
    encoder.fit(df['application_name'])
    # lista svih nDPI prepoznatih aplikacija
    apps = encoder.classes_
    # lista nDPI aplikacija i protokola koji nece biti razmatrani
    blacklist = ['HTTP', 'TLS', 'QUIC', 'Unknown', 'DHCPV6']
    for app in apps:
        if app not in blacklist and app.startswith('DNS') == False: # za uklanjanje DNS-a
            self.dict_name2label[app + '.pcap'] = app
    self.statusLabel.setText("Kreiranje draft liste za pronadjene aplikacije je zavrseno.")
    sleep(1)
```

U listi *blacklist* nalaze se *nDPI* klase koje treba ukloniti prije treniranja. Ova lista se zapravo odnosi na kontrolne pakete različitih protokola koji nijesu pridruženi ni jednoj konkretnoj aplikaciji (npr. TLS). Uz ove pakete, iz *dataset*-a se takođe uklanjaju DNS paketi, koji nijesu relevantni za proces treniranja.

Funkcija *ndpi_prepro* se koristi u aplikaciji za treniranje, a služi za obradu paketa iz *.pcap* fajla i izvlačenje labela:

```
# obrada i labelovanje paketa sa nDPI
def ndpi_prepro(self):
    self.statusLabel.setStyleSheet('')

    self.statusLabel.setText("Pokretanje modula NFStream...")
    sleep(1)
    filename = os.path.basename(self.filepath)
    # podjela .pcap fajlova korišćenjem nfstream application_name
    if self.filepath.endswith(".pcap"):
        # sprječava otvaranje više prozora u QT mainUI funkciji
        subprocess.run(['python', 'Utilities/nfs.py', self.filepath, nf_path])

        self.statusLabel.setText("NFStream prepoznavanje zavrseno.")
        sleep(1)

        self.statusLabel.setText("Kreiranje .pcap fajlova sa nDPI labelama...")
        sleep(1)

        # sortirana lista klasa
        lista=[]
        [lista.append(x) for x in self.dict_name2label.values() if x not in lista]
        lista.sort()

        aplikacije = [ [] for _ in range(len(lista)) ]  # kreiranje prazne liste aplikacija
        x_nf, y_nf = dpi(nf_path)

        for i, packet in enumerate(read_pcap(self.filepath)):
            x_p = ip_port(packet)
            if(x_p != None and x_p in x_nf and y_nf[x_nf.index(x_p)] in lista):
                # pretraga app indeksa
                app_index = lista.index(y_nf[x_nf.index(x_p)])
                aplikacije[app_index].append(packet)
        write(aplikacije, lista)
        self.statusLabel.setText("Obrada " + filename + " je zavrsena.")
        sleep(1)
        with open('Utilities/apps.py','w') as data:
            data.write('# for app identification\n')
            data.write('## created from file {}\n'.format(self.filepath))
```

```
            data.write('## draft version for used for application filtering\n')
            data.write('dict_name2label = {\n')
            #for app in aplikacije:
            for key, val in self.dict_name2label.items():
                #if val == app:
                    data.write("\t'{}': '{}',\n".format(key, val))
            data.write('\t}')
    else:
        self.statusLabel.setText("Odabrani fajl nije u .pcap fajl formatu.")

    # predobrada .pcap fajlova
    self.statusLabel.setText("Predobrada .pcap fajlova...")
    subprocess.run(['python', 'Utilities/prepro.py'])
    self.statusLabel.setText("Predobrada je zavrsena.")
    sleep(1)
```

Modul *nfs.py* se pokreće kao *subprocess* i njime se izvršava proces prepoznavanja klasa sa *NFStream* bibliotekom. Iz kreiranog fajla *baza.csv* izvlače se i broje aplikacije, a sortirana lista aplikacija smješta u listu *dict_name2label*. Inicijalna lista aplikacija upisuje se u fajl *apps.py*, a tu će kasnije biti upisana i konačna lista aplikacija.

U *utils.py* takođe se definišu i parametri modela:

```
# definisanje CNN modela
def cnn_model(self):
    input_size = 1500
    dropout = 0.2

    # Build a model
    model = Sequential()
    model.add(Conv1D(60, 5, input_shape = (input_size,1), activation = 'relu'))
    model.add(Dropout(dropout))
    model.add(MaxPooling1D(2))
    model.add(Flatten())
    denses = [200, 100, 50]
    for dense in denses:
        model.add(Dense(dense, activation = 'relu'))
        model.add(Dropout(dropout))
    model.add(Dense(self.nb_classes, activation = 'softmax'))
    return model


# definisanje MLP modela
def mlp_model(self):
    input_size = 1500
    input_dense = 1500
    dropout = 0.2

    # Izrada modela
    model = Sequential()

    model.add(Dense(input_dense, batch_input_shape=(None, input_size), \
              activation = 'relu'))
    model.add(Dropout(dropout))

    for x in range(0, 2):
        model.add(Dense(units=input_dense, activation = 'relu'))
        model.add(Dropout(dropout))

    model.add(Dense(self.nb_classes, activation = 'softmax'))
```

```
    return model

# definisanje SAE modela
def sae_model(self):
    input_size = 1500
    input_dense = 600
    denses = np.arange(1200, 0, -input_dense).tolist()
    dropout = 0.2

    # Izrada modela
    model = Sequential()

    # enkoder
    model.add(Dense(denses[0], batch_input_shape=(None, input_size), activation = 'relu'))
    model.add(Dropout(dropout))
    for i in denses[1:]:
        model.add(Dense(i, activation = 'relu'))
        model.add(Dropout(dropout))

    # dekoder
    denses.reverse()
    for i in denses[1:]:
        model.add(Dense(i, activation = 'relu'))
        model.add(Dropout(dropout))
    model.add(Dense(input_size, activation = 'relu'))
    model.add(Dropout(dropout))

    model.add(Dense(self.nb_classes, activation = 'softmax'))
    return model
```

Skripta *nfs.py* ima sledeći kod:

```
import sys
from nfstream import NFStreamer

filepath = str(sys.argv[1])
nf_path = str(sys.argv[2])

if __name__ == '__main__':
        df = NFStreamer(source= filepath).to_pandas()[["src_ip", "src_port", "dst_ip", \
                        "dst_port", "application_name"]]
        df.to_csv(nf_path)
```

Ovdje je izvršena inicijalizacija i pokretanje *NFStream* modula koristeći argumenate *filepath* i *nf_path* iz glavne funkcije aplikacije. Argument *filepath* predstavlja putanju odabranog *.pcap* fajla dok je argument *nf_path* lokacija gdje će rezultati obrade tog fajla biti upisani. Traženi metapodaci su *src_ip*, *src_port*, *dst_ip*, *dst_port* i *application_name*. Fajl *nfs.py* je kreiran kao poseban program iz razloga što se pokretanjem modula *NFStream* direktno unutar objekata aplikacije za treniranje i testiranje usporava njen rad. Odvajanje i izvršavanje ove funkcije u posebnoj niti uklanja ovaj problem.

ZAKLJUČAK

U ovom radu diskutovane su neke od tradicionalnih metoda za klasifikaciju i identifikaciju mrežnog saobraćaja, i navedeni su njihovi nedostaci. Za klasifikaciju i identifikaciju mrežnog saobraćaja trenutno se najviše koriste DPI tehnike, a njihov nedostatak je stalna potreba za dopunjavanjem baza potpisa za različite aplikacije od strane specijalista za analizu saobraćaja.

Kako tehnike mašinskog učenja imaju mogućnost da automatski nauče složene šablone u podacima, za klasifikaciju i enkriptovanih i neenkriptovanih podataka predložene su različite duboke neuralne mreže [1] [58] [59] [62] [79] [80] [81]. Polazi se od pretpostavke da u kontrolisanom mrežnom okruženju administrator mreže može relativno jednostavno labelirati klase saobraćaja od interesa i trenirati model mašinskog učenja da kasnije automatski prepoznaje šablon karakterističan za te klase. Jednom kada se modeli mašinskog učenja istreniraju da prepoznaju predefinisane klase, relativno lako se mogu nadograditi i dotrenirati sa *transfer learning* metodama da prepoznaju nove klase/aplikacije, bez potrebe za učešćem DPI eksperata [82].

U radu su testirane tri modela dubokog učenja za klasifikaciju mrežnog saobraćaja: MLP, SAE i CNN. Svaka od njih koristi pristup obrade *payload*-a paketa. Takođe je testirana i varijacija CNN metode koja ne koristi potpuno povezane slojeve. Eksperimentalno je utvrđeno da svaka od ovih metoda daje obećavajuće rezultate u pogledu učinkovitosti klasifikacije mrežnog saobraćaja. Najbolje rezultate je dala standardna jednodimenzionalna CNN metoda (1D-CNN), dok je najlošije rezultate dala CNN metoda bez potpuno povezanih slojeva (CNN-NFC).

Za potrebe ovog istraživanja kreirane su dvije GUI aplikacije koje mogu da izvrše treniranje modela i predikciju mrežne aplikacije za zadati *dataset* saobraćaja. GUI aplikacije nude mogućnost podešavanja parametara modela, kao što su dimenzije *dense* i konvolucionih slojeva, *dropout*, *learning rate* itd.

Iz analiza performansi različitih modela dubokog učenja za klasifikaciju saobraća, može se jasno identifikovati problem identifikacije enkriptovanog saobraćaja, kao što je TLS ili QUIC. Problemi sa klasifikacijom enkriptovanog saobraćaja su i očekivani kada su u pitanju analizirani modeli, jer se klasifikacija može vršiti samo na osnovu limitirane količine informacija u zaglavlju paketa. Ostali obrasci karakteristični za aplikacije obično su efikasno prikriveni algoritmima enkripcije, pod pretpostavkom da oni koriste kvalitetne generatore slučajnih brojeva. Unaprijeđenje algoritama u pogledu veće preciznosti klasifikacije enkriptovanog saobraćaja moglo bi se u budućim istraživanjima postići na osnovu statističkih parametara toka paketa.

LITERATURA

[1] A. Moore / K. Papagiannaki, „Toward the accurate identification of network applications,“ *Springer, Berlin, Heidelberg,* t. 3431, br. 5, p. 41–54, 2005.

[2] V. Mohan, J. Y. R. Reddy / K. Kalpana, „Active and Passive Network Measurements: A Survey,“ *International Journal of Computer Science and Information Technologies,* t. 2, br. 4, pp. 1372-1385, 2011.

[3] „What is iPerf / iPerf3 ?,“ iperf.fr, [Na mreži]. Available: https://iperf.fr/.

[4] „IAB Minutes 1988-03-21,“ Internet Architecture Board, 21 03 1988. [Na mreži]. Available: https://www.iab.org/documents/minutes/minutes-1988/iab-minutes-1988-03-21/.

[5] „Windows Management Instrumentation (WMI),“ techtarget.com, [Na mreži]. Available: https://searchwindowsserver.techtarget.com/definition/Windows-Management-Instrumentation.

[6] „Simple Network Management Protocol (SNMP),“ Ozeki 10, [Na mreži]. Available: https://www.ozeki.hu/index.php?owpn=7614.

[7] „Service assurance for hybrid networks and services,“ [Na mreži]. Available: https://www.infovista.com/vistainsight/hybrid-network-solutions.

[8] „IBM Tivoli Netcool/OMNIbus V8.1 Documentation,“ IBM, [Na mreži]. Available: https://www.ibm.com/support/knowledgecenter/SSSHTQ_8.1.0/com.ibm.netcool_OMNIbus.doc_8.1.0/omnibus/wip/user/concept/omn_ovr_introtonetcoolomnibus.html.

[9] „CA Performance Management,“ BROADCOM, [Na mreži]. Available: https://docs.broadcom.com/doc/ca-performance-management.

[10] „TotalView®Automated Actionable Intelligence for IT Systems,“ PathSolutions, [Na mreži]. Available: https://www.pathsolutions.com/.

[11] „Network Performance Monitor,“ solarwinds, [Na mreži]. Available: https://www.solarwinds.com/.

[12] „NMIS Professional Intelligent Network Management Systems,“ Opmantek, [Na mreži]. Available: https://opmantek.com/nmis-professional-bundle/.

[13] „wireshark,“ [Na mreži]. Available: https://github.com/wireshark/wireshark.

[14] „IPCopper USC10G3,“ IPCopper, [Na mreži]. Available: http://www.ipcopper.com/product_usc10g3.htm.

[15] „10/100/1000Base-T Network TAP,“ Dualcomm, [Na mreži]. Available: https://www.dualcomm.com/products/usb-powered-10-100-1000base-t-network-tap.

[16] T. Keary, „PCAP: Packet Capture, what it is & what you need to know,“ comparitech, 19 May 2021. [Na mreži]. Available: https://www.comparitech.com/net-admin/pcap-guide/.

[17] „TCPDUMP & libpcap,“ The Tcpdump Group, [Na mreži]. Available: https://github.com/the-tcpdump-group/libpcap.

[18] „WinPcap,“ Riverbed Technology, [Na mreži]. Available: https://www.winpcap.org/default.htm.

[19] „NetMaster® Network Intelligence,“ BROADCOM, [Na mreži]. Available: https://www.broadcom.com/products/mainframe/operations-mgmt/netmaster.

[20] „solaris - snoop (1),“ [Na mreži]. Available: http://ibgwww.colorado.edu/~lessem/psyc5112/usail/man/solaris/snoop.1.html.

[21] K. Johanns , „Tech Time Warp: Network General — Maker of The Sniffer — Founded,“ 18 May 2018. [Na mreži]. Available: https://smartermsp.com/tech-time-warp-sniffer-network-general/.

[22] „Collect data using Network Monitor,“ 02 09 2022. [Na mreži]. Available: https://learn.microsoft.com/en-us/troubleshoot/windows-client/networking/collect-data-using-network-monitor.

[23] „Catalyst Switched Port Analyzer (SPAN) Configuration Example,“ CISCO, 22 January 2019. [Na mreži]. Available: https://www.cisco.com/c/en/us/support/docs/switches/catalyst-6500-series-switches/10570-41.html.

[24] A. Dainotti, A. Pescape / K. Claffy, „Issues and future directions in traffic classification,“ u *IEEE network 26*, 2012.

[25] Y. Qi, L. Xu, B. Yang, Y. Xue / J. Li, „Packet classification algorithms: From theory to practice,“ *INFOCOM 2009,* p. 648–656, 2009.

[26] A. Madhukar / C. Williamson, „A longitudinal study of p2p traffic classification. In: Modeling, Analysis, and Simulation of Computer and Telecommunication Systems,“ *14th IEEE International Symposium on, IEEE,* p. 179–188, 2006.

[27] J. Khalife, A. Hajjar / J. Diaz-Verdejo, „A multilevel taxonomy and requirements for an optimal traffic-classification model,“ *International Journal of Network Management 24,* p. 101–120, 2014.

[28] S. Yeganeh, M. Eftekhar, Y. Ganjali, R. Keralapura / A. Nucci, „Traffic classification using terms,“ *Computer Communications and Networks (ICCCN,* t. 21, p. 1–9, 2012.

[29] S. Sen, O. Spatscheck / D. Wang, „Accurate, scalable in-network identification of p2p traffic using application signatures,“ *Proceedings of the 13th International Conference on World Wide Web,* pp. 512-521, 2004.

[30] J. Sherry, C. Lan, R. Popa / S. Ratnasamy, „Blindbox: Deep packet inspection over encrypted traffic,“ *ACM SIGCOMM Computer Communication Review,* t. 45, p. 213–226, 2015.

[31] M. Crotti, M. Dusi, F. Gringoli / L. Salgarelli, „Traffic classification through simple statistical fingerprinting,“ *ACM SIGCOMM Computer Communication Review 37,* p. 5–16, 2007.

[32] X. Wang / D. Parish, „Optimised multi-stage tcp traffic classifier based on packet size distributions,“ *Communication Theory, Reliability, and Quality of Service (CTRQ),* p. 98–103, 2010.

[33] T. Auld, A. Moore / S. Gull, „Bayesian neural networks for internet traffic classification,“ *IEEE Transactions on neural networks 18,* p. 223–239, 2007.

[34] A. Moore / D. Zuev, „Internet traffic classification using bayesian analysis techniques,“ *ACM SIGMETRICS Performance Evaluation Review,* t. 33, p. 50–60, 2005.

[35] R. Sun, B. Yang, L. Peng, Z. Chen, L. Zhang / S. Jing, „Traffic classification using probabilistic neural networks,“ *Natural computation (ICNC),* t. 4, p. 1914–1919, 2010.

[36] H. Ting, W. Yong / T. Xiaoling, „Network traffic classification based on kernel self-organizing maps,“ *Intelligent Computing and Integrated Systems (ICISS),* p. 201, 2010.

[37] G. Gil, A. Lashkari, M. Mamun / A. Ghorbani, „Characterization of encrypted and vpn traffic using time-related features,“ *Proceedings of the 2nd International Conference on Information Systems Security and Privacy (ICISSP 2016),* p. 407–414, 2016.

[38] B. Yamansavascilar, M. Guvensan, A. Yavuz / M. Karsligil, „Application identification via network traffic classification,“ *Computing, Networking and Communications (ICNC),* p. 843–848, 2017.

[39] A. Moore, D. Zuev / M. Crogan, „Discriminators for use in flow-based classification,“ u *Tech. Rep*.

[40] D. Poole, A. Mackworth / R. Goebel, Computational Intelligence: A Logical Approach, Oxford University Press, 1998.

[41] „What is Machine Learning? A definition,“ 7 3 2017. [Na mreži]. Available: https://expertsystem.com/machine-learning-definition/.

[42] R. van Loon, „Machine learning explained: Understanding supervised, unsupervised, and reinforcement learning,“ [Na mreži]. Available: https://bigdata-madesimple.com/machine-learning-explained-understanding-supervised-unsupervised-and-reinforcement-learning/.

[43] F. V. Nelwamondo, D. Golding / T. Marwalab, „A dynamic programming approach to missing data estimation using neural networks,“ *Information Sciences,* t. 273, pp. 49-58, 2013.

[44] P. Prakash / A. S. K. Rao, R deep learning cookbook : solve complex neural net problems with TensorFlow, H2O and MXNet, Birmingham: Packt Publishing, 2017..

[45] I. Goodfellow, Y. Bengio / A. Courvil, „Deep Learning,“ MIT Press, [Na mreži]. Available: http://www.deeplearningbook.org/contents/convnets.html.

[46] D. H. HUBEL / T. N. WIESEL, „RECEPTIVE FIELDS AND FUNCTIONAL ARCHITECTURE OF MONKEY STRIATE CORTEX,“ *The Journal of Physiology,* t. 195, br. 1, pp. 215-243, 1968.

[47] K. Fukushima, „Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position,“ *Biological Cybernetics,* t. 36, pp. 193-202, 1980.

[48] „Basic principle of the neocognitron,“ [Na mreži]. Available: https://www.kiv.zcu.cz/studies/predmety/uir/NS/Neocognitron/en/hierarch-det.html.

[49] K. T. O'Shea / R. Nash, „An Introduction to Convolutional Neural Networks,“ 2015. [Na mreži]. Available: https://www.researchgate.net/publication/285164623_An_Introduction_to_Convolutional_Neural_Networks.

[50] K. S. Pervunin, „Artificial neural network for bubbles pattern recognition on the images,“ [Na mreži]. Available: https://www.researchgate.net/publication/309487032_Artificial_neural_network_for_bubbles_pattern_recognition_on_the_images/figures?lo=1.

[51] J. Johnson / A. Karpathy, CS231n: Convolutional Neural Networks for Visual Recognition, Stanford Computer Science, 2017..

[52] D. Yu, H. Wang, P. Chen / Z. Wei, „Mixed Pooling for Convolutional Neural Networks,“ u *Lecture Notes in Computer Science,* Shanghai, China, 2014.

[53] K. He, X. Zhang, S. Ren / J. Sun, „Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition,“ *IEEE Transactions on Pattern Analysis and Machine Intelligence,* t. 37, br. 9, pp. 1904-1916, 2015.

[54] M. D. Zeiler / R. Fergus, „Stochastic Pooling for Regularization of Deep Convolutional Neural Networks,“ u *1st International Conference on Learning Representations, ICLR 2013,* Scottsdale, United States, 2013.

[55] J. Gu, Z. Wang, J. Kuen, L. Ma, A. Shahroudy, B. Shuai, T. Liu, X. Wang, G. Wang, J. Cai / T. Chen, „Recent advances in convolutional neural networks,“ *Pattern Recognition,* t. 77, p. 354–377, 2015.

[56] H. Wang, P. Chen / Z. Wei, „Mixed Pooling for Convolutional Neural Networks,“ u *The 9th International Conference on Rough Sets and Knowledge Technology,* Shanghai, China, 2014.

[57] e. a. M. Lopez-Martin, „Network Traffic Classifier With Convolutional and Recurrent Neural Networks for Internet of Things,“ *IEEE Access,* t. 5, pp. 18042-18050, 2017.

[58] P. Wang, F. Ye, X. Chen / Y. Chen, „Datanet: Deep Learning Based Encrypted Network Traffic Classification in SDN Home Gateway,“ *IEEE Access,* t. 6, pp. 55380-55391, 2018.

[59] S. Rezaei, B. Kroencke / X. Liu, „Large-scale Mobile App Identification Using Deep Learning,“ *IEEE Access,* t. 8, 2020.

[60] V. Tong, H.-A. Tran, S. Souihi / A. Mellouk, „A Novel QUIC Traffic Classifier Based on Convolutional Neural Networks,“ pp. 1-6, 2018.

[61] R. Aljoufi / A. Lasebae, „Multi-task Learning for Intrusion Detection and Analysis of Computer Network Traffic,“ *E3S Web of Conferences,* p. 229, 2021.

[62] M. Lotfollahi, R. Shirali Hossein Zade, M. Jafari Siavoshani / M. Saberian, „Deep Packet: A Novel Approach For Encrypted Traffic Classification Using Deep Learning,“ *Soft Computing,* 2020.

[63] Z. Wang, „The applications of deep learning on traffic identification,“ u *BlackHat,* 2015.

[64] A. Jain, „Network Traffic Identification with Convolutional Neural Networks,“ 2018/08/01, pp. 1001-1007.

[65] „NFStream,“ [Na mreži]. Available: https://www.nfstream.org/.

[66] L. Deri, „nDPI,“ ntop, [Na mreži]. Available: https://www.ntop.org/products/deep-packet-inspection/ndpi/.

[67] „APIs Documentation,“ NFStream, [Na mreži]. Available: https://www.nfstream.org/docs/api.

[68] N. Nikolić, „Machine learning traffic identification,“ [Na mreži]. Available: https://github.com/nedjoni/mlnworkidentification.

[69] „8000S/48,“ Allied Telesys, [Na mreži]. Available: https://www.alliedtelesis.com/en/products/switches/8000s48.

[70] D. P. Kingma / J. Ba, „Adam: A Method for Stochastic Optimization,“ u *3rd International Conference for Learning Representations,* San Diego, 2015.

[71] „VPN-nonVPN dataset (ISCXVPN2016),“ Canadian Institute for Cybersecurity, [Na mreži]. Available: https://www.unb.ca/cic/datasets/vpn.html.

[72] M. Hou, „Pytorch Implementation of Deep Packet: A Novel Approach For Encrypted Traffic Classification Using Deep Learning,“ [Na mreži]. Available:

https://blog.munhou.com/2020/04/05/Pytorch-Implementation-of-Deep-Packet-A-Novel-Approach-For-Encrypted-Tra%EF%AC%83c-Classi%EF%AC%81cation-Using-Deep-Learning/#disqus_thread.

[73] I. Akbari / E. Tahoun, „PrivPkt: Privacy Preserving Collaborative Encrypted Traffic Classification“.

[74] „KerasTuner,“ [Na mreži]. Available: https://github.com/keras-team/keras-tuner.

[75] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh / A. Talwalkar, „Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization,“ *Journal of Machine Learning Research*, t. 18, pp. 1-52, 2018.

[76] „What is PyQt?,“ Riverbank Computing, [Na mreži]. Available: https://riverbankcomputing.com/software/pyqt/.

[77] „EarlyStopping,“ Keras, [Na mreži]. Available: https://keras.io/api/callbacks/early_stopping/.

[78] „TEMPLATES, Gallery Of Qt QSS Based Styles,“ QSS stock, [Na mreži]. Available: https://qss-stock.devsecstudio.com/templates.php.

[79] F. Pacheco, E. Exposito, M. Gineste, C. Baudoin / J. Aguilar, „Towards the Deployment of Machine Learning Solutions in Network Traffic Classification: A Systematic Survey,“ *IEEE Communications Surveys & Tutorials* , t. 21, br. 2, pp. 1988 - 2014, 2018.

[80] N. Cong Luong, D. Thai Hoang, S. Gong, D. Niyato, P. Wang, Y.-C. Liang / D. In Kim, „Applications of Deep Reinforcement Learning in Communications and Networking: A Survey,“ *IEEE Communications Surveys & Tutorials*, t. 21, br. 4, pp. 3133 - 3174, 2019.

[81] S. Tomović / I. Radusinović, „A Novel Deep Q-learning Method for Dynamic Spectrum Access,“ u *2020 28th Telecommunications Forum (TELFOR)*, Belgrade, Serbia, 2020.

[82] U. Trivedi / M. Patel, „A fully automated deep packet inspection,“ *2016 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS)*, pp. 1-6, 2016.