UNIVERZITET U NIŠU
ELEKTRONSKI FAKULTET
KATEDRA ZA RAČUNARSTVO

KLASIFIKACIJA SOFTVERA NA MALICIOZNI I BENIGNI

Predmet: Deep learning

Mentor:

Aleksandar Milosavljević

Studenti:

Kostić Jovana, 1436

Krstić Katarina, 1400

Sadržaj

1. Uvod

Veštačka inteligencija, deep learning i neuronske mreže grane su tehnologije, koje sa sobom nose jedan sasvim novi aspekt programiranja, kako samog procesa, tako i konačnih rezultata. Za razliku od klasičnog pristupa programiranju, u kojem se sastavljaju algoritmi po kojim će programi raditi, u ovom se slučaju gradi struktura koja će to sama da formira. Samim tim, kako zbog moći koju sa sobom nose, tako i zbog olakšanog procesa koji dovodi do konačnih rezultata, vremenom zaslužuju i sve više pažnje.

Na ovaj način, problemi neopisive kompleksnosti, za čije bi rešavanje klasičnim pristupom bilo neophodno, pre svega, obaviti ozbiljna istraživanja, konsultacije sa ekspertima za određene oblasti, a onda i samo kreiranje programa, oni dobijaju jednu sasvim novu formu. Tačnije, može se reći da se preterano i ne razlikuje postupak kojim bismo kompleksnije probleme ovim pristupom rešavali u odnosu na postupak kojim bismo rešavali one manjeg obima.

Tako ove tehnike zasigurno pružaju veliki značaj prvenstveno u polju medicine, biologije, psihologije, ali i bilo kojih drugih oblasti.

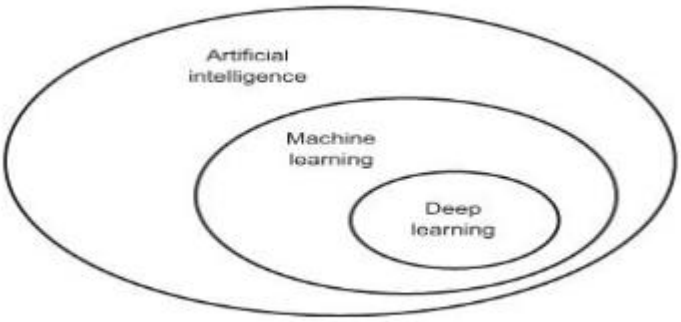
2. Veštačka inteligencija

Veštačka inteligencija pojavila se u prvoj polovini 20. veka kada počinje da se postavlja pitanje „Da li možemo da nateramo računare da razmišljaju?“, što je ujedno pitanje kojim se bavimo i dan danas kada je reč o ovoj oblasti. Najjasnija definicija ove oblasti bila bi sledeća: „**Napor za automatizaciju intelektualnih zadataka koje obično ljudi obavljanju**“.

Kao takva, veštačka inteligencija je osnovna oblast koja obuhvata mašinsko učenje i deep learning, ali koja, pored toga, uključuje još mnogo pristupa, koji ne obuhvataju nikakva učenja. Na primer, rani programi za šah su jedino uključivali hardcoded pravila pisana od strane programera i nisu se kvalifikovali kao mašinsko učenje.

Dugi niz godina, mnogi stručnjaci su verovali da bi veštačka inteligencija na ljudskom nivou mogla da se postigne pisanjem dovoljno velikog skupa eksplicitnih pravila, od strane programera, za manipulisanje znanjem. Ovaj pristup je poznat kao simbolična veštačka inteligencija, a bio je dominantna paradigma u veštačkoj inteligenciji od 1950. pa sve do kasnih 1980. Vrhunsku popularnost je ovaj pristup dostigao tokom ekspanzije ekspertskih sistema 1980.

Iako se simbolična veštačka inteligencija pokazala kao pogodna za rešavanje dobro definisanih, logičkih problema, kao što je igranje šaha, ispostavilo se da je nerešivo smisliti jasna pravila za rešavanje složenijih, nejasnih problema, kao što su klasifikacija slike, prepoznavanje govora, prevođenje jezika... Pojavio se novi pristup koji je zauzeo mesto simboličnoj veštačkoj inteligenciji, a to je mašinsko učenje.



Slika 1. – Veza između veštačke inteligencije, mašinskog učenja i deep learning-a

3. Mašinsko učenje

Mašinsko učenje je grana veštačke inteligencije i nastaje sa sledećim pitanjima: „Da li računar može da ide izvan onoga što znamo?“, „Kako da potražimo da radi i da samostalno nauči kako da obavlja određeni zadatak?“, „Umesto da programeri pišu pravila za obradu podataka, da li bi računar sam, automatski, mogao da nauči ova pravila gledajudi podatke?“

Ova pitanja otvaraju vrata novoj programskoj paradigmi. U klasičnom programiranju, paradigma simbolične veštačke inteligencije, ljudi unose pravila, odnosno program, i podatke koji se obrađuju po tim pravilima, a kao rezultat dobijaju odgovore. Sa mašinskim učenjem, ljudi unose podatke, kao i očekivane rezultate prilikom obrade datih podatka, s ciljem da ih sistem skupom pravila poveže. Ova pravila se zatim mogu primeniti nad novim podacima da bi se dobili jedinstveni odgovori.

Sistem mašinskog učenja se trenira, tj. obučava, a ne eksplicitno programira. Predstavljen je mnogim primerima relevantnim za zadatak i u tim primerima nalazi statističku strukturu, koja na kraju sistemu omogudava da smisli pravila za automatizaciju zadatka. Ovakav sistem zapravo sam sebe adaptira tokom perioda treninga, na osnovu primera sličnih problema, čak i kada ne postoji željeno rešenje za svaki problem. Nakon uspešnog treninga, ovaj sistem je sposoban da poveže podatke o problemu sa rešenjem, ulaze sa izlazima, a samim tim i da ponudi rešenje za neki nov problem.

Na primer, ako biste želeli da automatizujete zadatak za označavanje slika sa odmora, možemo predstaviti sistem mašinskog učenja sa mnogim primerima ved označenih slika od strane ljudi i sistem bi naučio pravila za povezivanje određenih slika sa određenim oznakama.

Dobar trening skup podataka bi morao da zadovolji sledede:

- Primeri moraju da reprezentuju generalnu populaciju
- Primeri moraju da sadrže članove svih klasa
- Primeri u svakoj klasi moraju da sadrže širok skup varijacija

Mašinsko učenje postaje trend vođen dostupnošdu bržih hardvera i vedih količina podataka. Mašinsko učenje je čvrsto povezano sa statistikom u matematici, ali se od nje razlikuje na nekoliko bitnih načina. Za razliku od statistike, mašinsko učenje teži da se nosi sa velikim i kompleksnim skupom podataka, kao što je skup podataka od milion slika, gde svaka od njih sadrži na desetine hiljada piksela, za šta bi klasišna statistička analiza, kao što je Bajesovo zaključivanje, bila nepraktična. Kao rezultat, mašinsko učenje, a posebno deep learning, izlaže relativno malo matematičke teorije – možda premalo – i inženjerski je orijentisano. To je praktična disciplina u kojoj su ideje dokazane više empirijski nego teoretski.

Da bismo definisali deep learning i razumeli razliku između deep learning-a i drugih pristupa mašinskog učenja, najpre moramo imati ideju o tome šta rade algoritmi mašinskog učenja. Do sada smo zaključili da mašinsko učenje otkriva pravila za izvršavanje zadatka, za obradu podataka, na osnovu datih primera onoga što se očekuje.

Dakle, za mašinsko učenje su nam neophodne tri stvari:

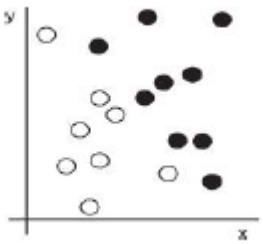
1) Ulazni podaci – Na primer, ako je zadatak prepoznavanje govora, ovi podaci bi mogli da budu zvučne datoteke ljudi koji govore. Ako je zadatak označavanje slika, mogli bi da budu slike...

2) Primeri očekivanog rezultata – U zadatku prepoznavanja govora, to bi mogli da budu prepisi zvučnih datoteka koje generišu ljudi. U zadatku sa slikom, očekivani rezultati mogu da budu oznake kao što su „pas“, „mačka“ itd.

3) Način za merenje da li algoritam dobro radi – Ovo je neophodno da bi se odredila udaljenost trenutnog izlaza algoritma od očekivanog rezultata. Merenje se koristi kao povratni signal za podešavanje načina rada algoritma. Ovaj korak podešavanja (prilagođavanja) je ono što i nazivamo učenjem.

Glavni problem u mašinskom učenju i deep learning-u jeste da se smisleno transformišu podaci: drugim rečima naučiti korisne prikaze ulaznih podataka – reprezentacije koje nas približavaju očekivanom rezultatu. Pre nego što odemo dalje, neophodno je i objasniti pojam reprezentacije. U svojoj srži, to je drugačiji način gledanja na podatke – za predstavljanje ili kodiranje podataka. Na primer, slika u boji može da se kodira ili u RGB formatu (red-green-blue) ili u HSV (hue-saturation-value) formatu. Ovo su dve različite reprezentacije istog podatka. Neki zadaci koji mogu da budu teški sa jednom reprezentacijom mogu da postanu lakši sa korišdenjem druge. Na primer, ako je zadatak „Izaberi sve crvene piksele na slici“ lakše je koristiti RGB format. Međutim, ako je zadatak „Učini sliku manje zasidenom“ lakše je koristiti HSV format. Modeli mašinskog učenja se zapravo bave nalaženjem odgovarajudih reprezentacija za njihove ulazne podatke – transformacije podataka koje ga čine pristupačnijim zadatku, kao što je zadatak klasifikacije.

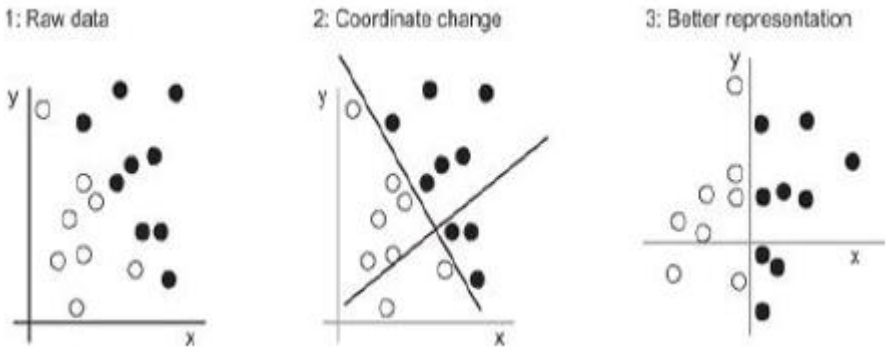
Kako bismo ovo učinilo što konkretnijim, posmatrademo sledede primere: Recimo da imamo primer dvodimenzionalnog koordinatnog sistema sa određenim tačkama u njemu predstavljenim (x, y) koordinatama. Kao što vidimo na slici ispod, određene tačkice su bele, dok su ostale crne boje. Recimo i da želimo da formiramo algoritam koji bi na osnovu koordinata tačke mogao da nam da odgovor na pitanje da li je tačka bele ili crne boje. U ovom slučaju, ulazi su koordinate tačaka, očekivani rezultat je odgovor na pitanje da li je konkretna tačka bele ili crne boje, dok bi mera ispravnosti algoritma bio procenat ispravno klasifikovanih tačaka.



Slika 2. – Primer dvodimenzionalnog koordinatnog sistema

Ono što je nama neophodno jeste nova reprezentacija koja bi jasno razdvojila crne od belih tačaka. Jedna od transformacija koju bismo mogli da iskoristimo jeste da promenimo položaj koordinatnog sistema, kao što je prikazano na slici 3. Možemo da zaključimo da su

koordinate tačaka u novom koordinatnom sistemu nova reprezentacija podataka. U ovakvoj reprezentaciji, problem klasifikacije na crne i bele tačke može biti predstavljen jednostavnim pravilom: „Crne tačke su one koje zadovoljavaju uslov x>0“ ili „Bele tačke su one koje zadovoljavaju uslov x<0.“ Samim tim, možemo redi i da novi način predstavljanja podataka rešava problem klasifikacije.



Slika 3. – Primer promene položaja koordinatnog sistema

U ovom slučaju je promena koordinatnog sistema učinjena ručno, ali, ukoliko bismo probali sistematsko traženje različitih mogudih načina za promenu koordinatnog sistema, a kao odgovor dobijali procenat ispravno klasifikovanih tačaka, to bi predstavljalo mašinsko učenje. Pojam „učenja“ u mašinskom učenju predstavlja proces automatskog traženja bolje reprezentacije podataka.

Dakle, svi algoritmi mašinskog učenja se sastoje od automatskog pronalaženja takvih transformacija, koje pretvaraju podatke u korisnije reprezentacije za dati zadatak. Ove operacije mogu da budu: menjanje koordinata, linearne projekcije (što može da uništi informacije), translacije, nelinearne operacije (kao što je „Označi sve tačke koje zadovoljavaju uslov x>0“)... Algoritmi mašinskog učenja obično i nisu kreativni u pronalaženju ovih transformacija, ved samo pretražuju unapred definisani skup operacija koji se naziva prostorom hipoteza.

To je ono što predstavlja mašinsko učenje, tehnički: traženje korisnih reprezentacija nekih ulaznih podataka, u okviru unapred definisanog prostora mogudnosti, korišćenjem uputstava iz povratnog signala.

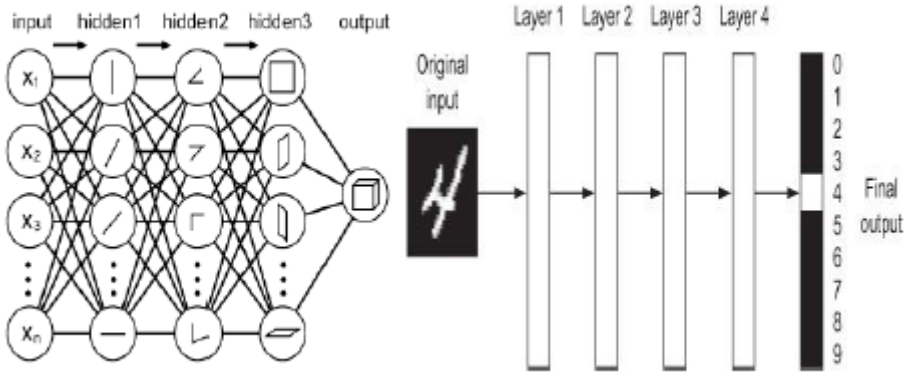
4. Deep learning

Deep learning je posebna podoblast mašinskog učenja: novo prenošenje reprezentacija o učenju iz podataka koje stavlja akcenat na učenje sukcesivnih slojeva sve smislenijoj reprezentaciji. „*Deep*“ u *deep learning*-u nije referenca na bilo koju vrstu dubljeg razumevanja pristupa, već se, umesto toga, odnosi na ideju o sukcesivnim slojevima reprezentacija. Druga prikladna imena za ovu oblast bila bi i učenje slojevitih reprezentacija i učenje hijerarhijskih reprezentacija. Koliko slojeva doprinosi modelu podataka naziva se dubinom modela. Savremeni *deep learning* često uključuje desetine, a nekad čak i stotine sukcesivnih slojeva reprezentacija – i svi oni su automatski naučeni za „*treniranje*“ podataka. U međuvremenu, drugi pristupi mašinskog učenja se fokusiraju na učenje samo jednog ili dva sloja reprezentacije podataka, pa se stoga ponekad nazivaju i plitkim učenjem.

5. Pojam neuronske mreže

U *deep learning*-u, ove slojevite reprezentacije se gotovo uvek uče iz modela zvanih neuronske mreže, *koje* su struktuirane u bukvalnim slojevima i složene jedna na drugu. Termin „*neuronske* mreže“ je referenca na neurobiologiju, ali, iako su neki od glavnih koncepata u deep learning-u razvijeni crpedi informacije od našeg razumevanja mozga, *deep learning* modeli nisu modeli mozga. Istorijski, inspiracija za celokupnu oblast potiče od želje da se proizvede veštački sistem koji bi bio sposoban za inteligentne proračune, slične onima koje rutinski obavlja sam čovek, odnosno ljudski mozak. Veštačka neuronska mreža sadrži određen broj veštačkih neurona koji su međusobno povezani, a pored toga, i organizovani po nivoima.

Osnovnu strukturu neuronske mreže čine čvorovi i potezi. Čvorovi u ulaznom sloju povezani su sa ostalim čvorovima narednog sloja preko potega. Poteg ima težinu, odnosno parametar koji predstavlja jačinu veze između dva čvora. Tipična neuronska mreža je sačinjena od jednog ulaznog, jednog izlaznog i promenljivog broja skrivenih slojeva između njih. Ako neuronska mreža ima nekoliko skrivenih slojeva, tada se ona naziva dubokom neuronskom mrežom. U osnovi, vrednosti u svakom čvoru ulaznog sloja množe se težinama i sumiraju se u čvorovima slededeg sloja. Na taj način, svaki čvor u prvom skrivenom sloju pripaja sve informacije iz ulaznih čvorova sa različitim težinama, generišući razne moguće pojednostavljene reprezentacije za razlikovanje u skupu podataka. Potom se informacije čvorova u prvom skrivenom sloju integrišu u čvorove sledećeg skrivenog sloja, a vrednosti svih čvorova prvog skrivenog sloja se ponovo množe različitim težinama kako bi se dobile različite vrednosti za sve čvorove narednog sloja. Kako se ovaj postupak ponavlja u više slojeva, mogu se uspostaviti napredniji kriterijumi, koji mogu razdvojiti razlike u skupu podataka, pošto je broj različitih kombinacija čvorova povećan.

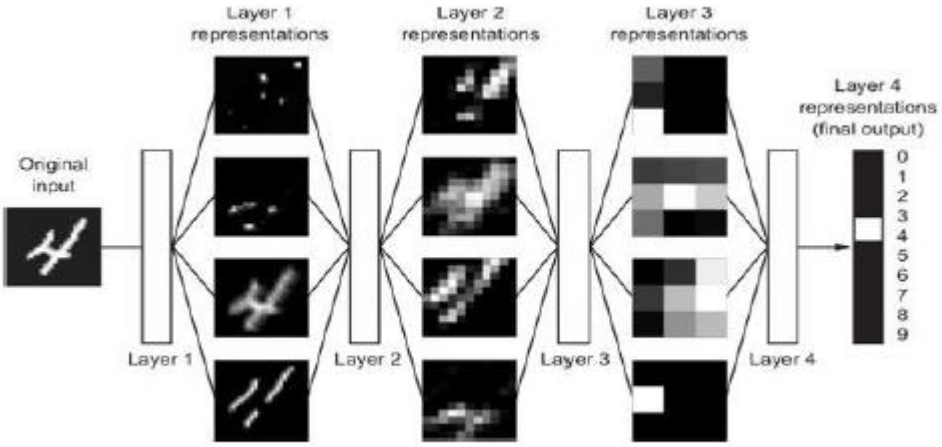


Slika 4. I 5. – Primeri izgleda reprezentacija učenih deep learning algoritmom

Na slici 5. Vidimo ulaz u neuronsku mrežu, koji predstavlja sliku na kojoj se nalazi broj, zatim neuronsku mrežu koja ima 4 sloja, i izlaz koji ona daje kao rezultat. A sada demo ispitati kako mreža duboka nekoliko sloja transformiše sliku broja sa ciljem da prepozna o kom broju je reč.

Kao što možemo videti na slici 6., mreža transformiše sliku broja u reprezentacije koje se sve više razlikuju od originalne slike i sadrže sve više informacija o krajnjem

rezultatu. Duboku mrežu možemo zapravo i da posmatramo kao operaciju višestepene destilacije informacije, gde informacija prolazi kroz sukcesivne filtere i izlazi sve više „pročišćena.“ Dakle, to je ono što predstavlja deep learning tehnički: višestepeni način učenja reprezentacija podataka.

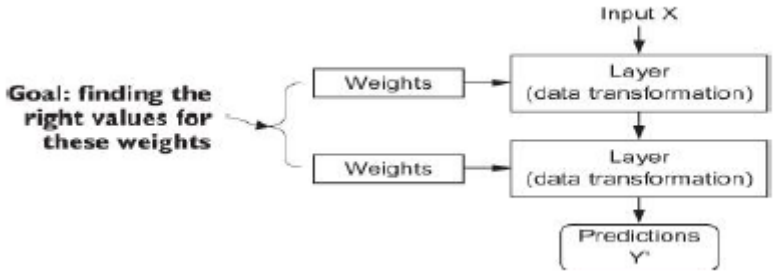


Slika 6 – Transformacija slike broja

5.1. Razumevanje rada deep learning-a

Do sada znamo da je mašinsko učenje u vezi sa povezivanjem ulaznih podataka (kao što su slike) sa ciljevima (kao što je oznaka „mačka“), što se postiže posmatranjem mnogo primera ulaznih podataka i ciljeva. Takođe znamo da duboke neuornske mreže to povezivanje ulaza sa ciljevima vrše dubokim nizom jednostavnih transformacija podataka (slojeva) i da se ove transformacije podataka uče izlaganjem primerima. Pogledajmo sada kako se to konkretno dešava.

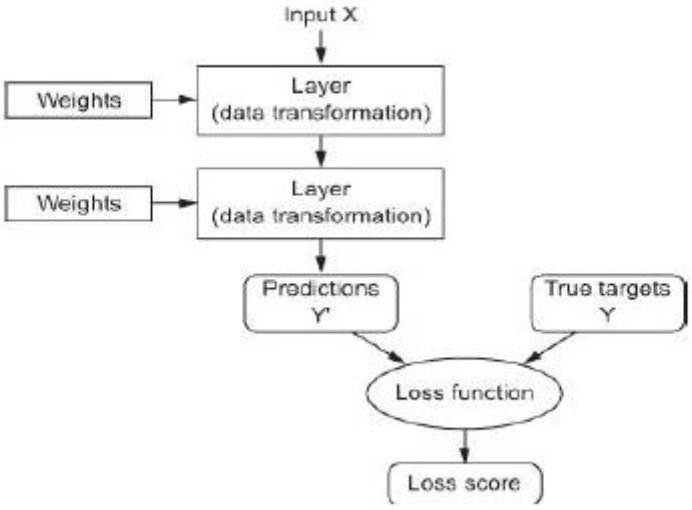
Specifikacija onoga što sloj čini sa ulaznim podatkom je skladištena u težini sloja, koja u suštini predstavlja gomilu brojeva. Tehnički gledano, rekli bismo da je transformacija koju implementira sloj parametrizovana težinom. Težine se, zato, još nazivaju i parametrima sloja. U ovom konktekstu, pod učenjem podrazumevamo nalaženje skupa vrednosti za sve težine slojeva u mreži, tako da mreža ispravno preslikava primere ulaza u njihove pridružene ciljeve. Ali, evo u čemu je stvar: duboka neuronska mreža može da sadrži na desetine miliona parametara. Nalaženje prave vrednosti za sve njih izgleda kao težak zadatak, pogotovu što menjanje vrednosti jednog parametra utiče na ponašanje svih ostalih.



Slika 7. - Prikaz cilja „učenja“ – nalaženje skupa vrednosti za sve težine slojeva u mreži

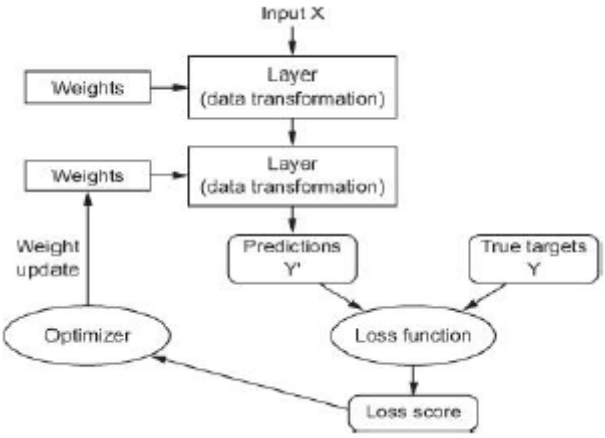
Da bismo nešto kontrolisali, najpre moramo biti u stanju da ga posmatramo. Da bismo kontrolisali izlaz neuronske mreže, treba da imamo mogućnost merenja koliko je ovaj izlaz udaljen od onoga što očekujemo kao rezultat. To je posao funkcije gubitka mreže.

Funkcija gubitka uzima predviđanja mreže i tačan cilj (ono što želimo da bude rezultat predviđanja mreže) i izračunava rezultat udaljenosti, snimajući koliko je mreža uradila na ovom konkretnom primeru.



Slika 8.-Prikaz funkcije gubitka mreže

Osnovni „trik“ u *deep learning*-u jeste da se ovaj rezultat udaljenosti koristi kao povratni signal, kako bi se malo prilagodila vrednost težine u onom smeru koji bi smanjio rezultat funkcije gubitka za dati primer. Ovo prilagođavanje je posao optimizatora koji implementira ono što se zove *Backpropagation* algoritam: centralni algoritam u *deep learning*-u. Na slici 8. je prikazano detaljnije objašnjenje kako on radi. Inicijalno, težinama su dodeljene random vrednosti, pa mreža samo sprovodi niz slučajnih transormacija. Prirodno je da njihovi rezultati tada budu jako daleko od očekivanih i da je, prema tome, i funkcija gubitka jako velika. Ali, sa svakim primerom koji mreža obrađuje, težine se malo po malo podešavaju u pravom smeru, što dovodi do opadanja funkcije gubitka.



Slika 8. Prikaz uloge optimizatora

To predstavlja trening petlju, koja, ponavljana dovoljan broj puta (obično desetine iteracija na hiljade primera) rezultira vrednostima težina koje smanjuju funkciju gubitka. Mreža sa minimalnim gubitkom je ona čiji su rezultati najbliže mogude ciljevima: utrenirana mreža.

5.2. Konvolucione neuronske mreže

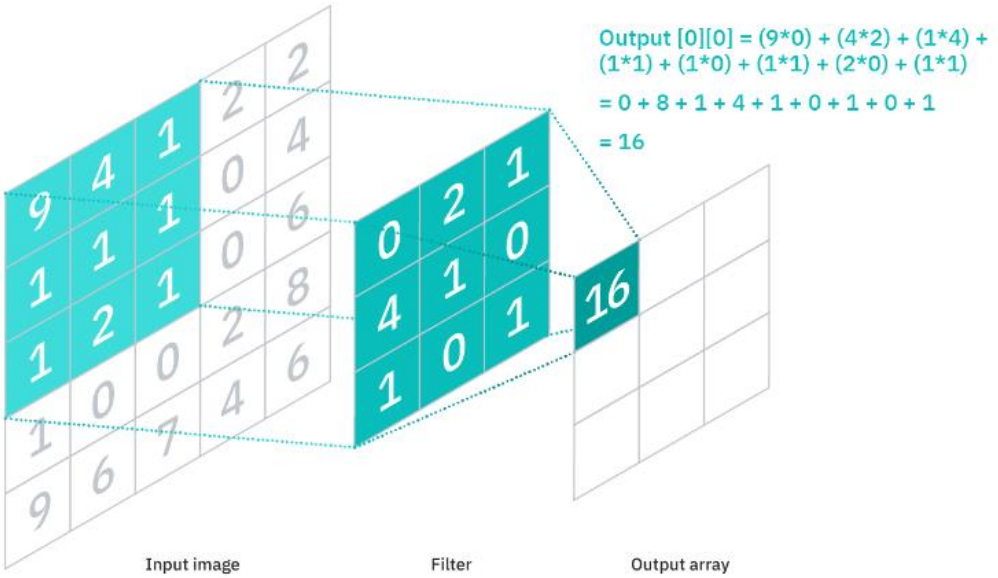
Srž snage konvolucionih neuronskih mreža ogleda se u načinu obrade slika, govornog jezika i audio signala. Poseduju tri osnovna tipa slojeva:

- *Convolutional layer*
- *Pooling layer*
- *Fully-conected(FC) layer*

Convolutional layer predstavlja prvi sloj konvolucione mreže. *Convolutional layer* može biti praćen dodatnim *convolutional layer*-ima ili *pooling layer*-ima, dok *fully-connected*(*FC*) predstavlja finalni, odnosno poslednji sloj. Svakim novim slojem konvoluciona neuronska mreža postaje kompleksnija, identifikujući sliku u sve većoj meri. Početni slojevi koncentrišu se na jednostavne elemente slike, poput njene boje i ivica, dok, kako proces obrade dalje odmiče, konvoluciona neuronska mreža prepoznaje sve konkretnije elemente i oblike objekta, dok ga konačno ne identifikuje.

Convolutional layer

Convolutional layer je osnovni gradivni element konvolucionih neuronskih mreže i u njemu se odvija najveći deo obrade. On zahteva nekoliko komponenti: ulazni podaci, filter i *feature map*. Ukoliko pretpostavimo da je ulaz boja slike, predstavljene u vidu 3D matrice piksela, to znači da će ulaz imati 3 dimenzije – visinu, širinu i dubinu. Filter, odnosno *kernel* ili *feature detector* kako se još naziva, vrši proveru postojanja konkretnog svojstva unutar dela slike koji se obrađuje. Taj proces predstavlja konvoluciju – *convolution*.



Slika 9 – Primer primene feature detector-a nad slikom

Feature detector je dvodimenzionalno polje težina, koje predstavlja deo slike. Iako mogu biti različitih veličina, matrica je najčešće dimenzija 3x3, što takođe utiče i na veličinu dela slike koji se posmatra. Rezultat primene filtera na ulazni deo slike smešta se u izlazni niz. Proces se ponavlja nad narednim područjem slike dokle god kompletna slika ne bude obrađena filterom. Konačni rezultat serijske obrade slike filterom predstavlja *feature map, activation map* ili *convolved feature*.

Težine unutar *feature detector*-a su fiksne i ne menjaju se ni za jedan deo slike na koji se primenjuju, a ova karaketeristika naziva se *parameter sharing*. Određeni parametri, poput težine, mogu se menjati u toku procesa treniranja, međutim postoje tri hiperparametra koji direktno utiču na veličinu izlaza i, kao takvi, moraju biti definisani pre nego sam proces treniranja neuronske mreže i počne, a oni su:

- Broj filtera
- *Stride*
- *Zero-padding*

S obzirom na to da izlaz ne mora da bude istih dimenzija kao ulaz, odnosno da ne moraju da se direktno mapiraju, *convolutional* i *pooling layer*-e nazivamo još i *partially connected layer*-ima.

Nakon svake operacije konvolucije, neuronska mreža primenjuje ReLU – *Rectified Linear Unit* transformacionu funkciju nad feature map-om, uvodeći nelinearnost u model.

Pooling layer

Pooling layer, takođe poznat pod nazivom *downsampling* ima vrlo sličan princip rada konvolucionom sloju. Međutim, u ovom slučaju se ne uzimaju u obzir težine, već se koriste agregacione funkcije, čija primena nad ulazom formira izlazni niz.

Postoje dve osnovne pooling funkcije:

- *Max pooling*
- *Average pooling*

Iako se primenom ovog sloja gubi velika količina informacija, on pomaže u pojednostavljenju, omogućava veću efikasnost i smanjuje rizik od *overfitting*-a.

Fully-connected layer

Za razliku od *partitially connected layer*-a, *fully connected layer* direktno povezuje svaki čvor izlaznog sloja sa čvorom prethodnog sloja. Dok konvolucioni i *pooling* slojevi teže korišćenju *ReLU* funkcija, *fully connected layer*-i najčešće koriste *softmax* aktivacionu funkciju da bi ispravno klasifikovali ulaze, generišući tako verovatnoću od 0 do 1.

6. Razvoj neuronske mreže

6.1. Keras

Za izgradnju i rad sa *deep learning*-om najčešće se koristi *Keras*. *Keras* je *deep learning framework* za *Python* koji je jako pogodan za korišdenje kada imamo zadatak da definišemo i obučavamo bilo koji *deep learning* model. *Keras* se može koristiti besplatno i kompatibilan je za gotovo sve verzije *Python*-a. *Keras* ima preko 200 000 korisnika, gde imamo i akademske istraživače, inženjere, kako početnike, tako i dugogodišnje radnike u velikim kompanijama, kao i one koji se upoznaju sa *deep learning*-om iz hobija. *Keras* se smatra najpopularnijim *framework*-om za *deep learning* i koriste ga *Google* i *Netflix*.

6.2. TensorFlow, Theano i CNTK

Keras je biblioteka za modele i nudi važne *high-level* gradivne elemente za razvoj *deep learning* modela. Međutim, *Keras* nije taj koji je zadužen za *low-level* operacije kao što su manipulacija nad ulaznim podacima i slično, pa se on oslanja na specijalizovane biblioteke koje se koriste kao backend podrška *Keras*-a. Ovi *backend*-i mogu biti korišdeni i pojedinačno i zajedno. Danas su primarne platforme za *deep learning* tri *backend*-a: *TensorFlow*, *Theano* i *Microsoft Cognitive Toolkit* (*CNTK*). *Theano* je razvijen od strane *MILA* laboratorije na Univerzitetu u *Montreal*-u, *TensorFlow* je razvio *Google*, a *CNTK Microsoft*. Neuronska mreža može biti građena i korišćenjem samo jednog od ovih *backend*-a, recimo *TensorFlow backend*-a, ali razlog zašto se uglavnom koristi *Keras* jeste taj što on nudi *high-level API* za neuronske mreže, koji može da bude pokrenut na bilo kojem od ovih *backend*-a. To znači da ćemo mnogo manje vremena da potrošimo na razmišljanje o tome šta će se dešavati u pozadini ukoliko koristimo *Keras*. Zato se *Keras* najčešće koristi kada je neophodno izvesti neki brzi eksperiment. Bilo koji deo koda, ili kompletan kod, može biti pokrenut korišćenjem bilo kojeg od ovih *backend*-a bez potrebe da se išta u kodu menja. Čak prilikom kreiranja nekog projekta možete jednom pokrenuti na jednom *backend*-u, zatim na slededem, što se često pokazuje kao korisna aktivnost jer se može pokazati da jedan od njih brže radi za neki konkretni problem. Uglavnom, preporučuje se korišćenje *TensorFlow backend*-a za većinu *deep learning* modela, jer se smatra da je najskalabilniji i najprilagođeniji potrebama.

6.3. Tok razvoja sa Keras-om

Tok rada koji je tipičan za *Keras* podrazumeva slededе korake:

1. Definisanje skupa trening podataka – ulazni podaci i njima pridruženi ciljevi

2. Definisanje mreže slojeva (modela) koji de da mapira zadate ulaze u zadate ciljeve

3. Konfigurisanje procesa učenja odabirom funkcije gubitka, optimizatora i određenih metrika za pradenje

4. Iteracija procesa nad trening podacima pozivom *fit()* metode

6.4. Instalacija

Keras je moguće instalirati izvršenjem sledeće komande u *command prompt*-u:

pip install keras

a *Tensorflow* komandom:

pip install tensorflow

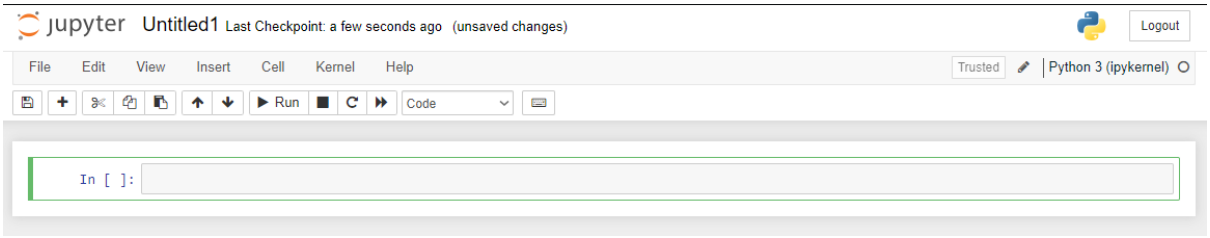
Pogodno grafičko okruženje za rad sa neuronskim mrežama je *Jupyter Notebook* i njega instaliramo na sledeći način:

pip install notebook

a pokrećemo ga naredbom:

jupyter notebook

Rezultat izvršenja ove naredbe je pokretanje editora na *localhost*-u, na *portu 8888*. Klikom na *New-Python3* otvara nam se novi tab u okviru kojeg možemo pisati i izvršavati naredbe i izgleda kao na sledećoj slici:



Slika 10 – Jupyter notebook

Keras može biti pokrenut i na *CPU* i na *GPU*, s tim što, da bi se pokrenuo korišćenjem *GPU*, ona mora biti *NVIDIA*.

7. Implementacija

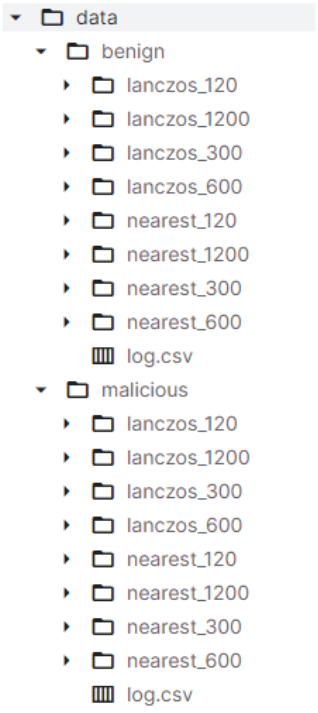
7.1. Definicija problema

U ovom radu bavimo se klasifikacijom softvera na maliciozni i benigni. Naime, *portable executables*[1] malizicioznih i benignih softvera u formi fotografija, koriste se kao ulazni podaci, dok izlaz mreže predstavlja rezultat klasifikacije, odnosno informaciju o tome da li se radi o benignom ili malicioznom softveru čija je fotografija obrađena. Rezultat rada je kreirana neuronska mreža koja vrši predikciju na osnovu ovakve fotografije.

7.2. Dataset

Datasetu koji je korišćen za implementaciju može se pristupiti na *Kaggle* zvaničnom sajtu https://www.kaggle.com/datasets/matthewfields/malware-as-images. Za potrebe ovog projekta kao ulazni podaci korišćen je skup slika koji je dobijen korišćenjem Python skripte Mallok - skripta koja *portable* executables transformiše u format slike.

Skup slika koji čini ovaj *dataset* organizovan je prvenstveno u dva direktorijuma: *benign* i *malicious*, što se odnosi na benigni i maliciozni softver. Zatim su grupisane prema *DPI* vezijama samih fotografija dobijenih korišćenjem dve različite metode interpolacije: *nearest* i *lanczos*.



Slika 11 – Organizacija dataset-a

Obradom podskupa slika ovog *dataset*-a koji se koristi za obučavanje mreže, formiraće se konvoluciona neuronska mreža koja će imati sposobnost zaključivanja da li se radi o benignom ili o malicioznom softveru. Ostatak slika biće iskorišćen u svrhe testiranja procenta uspešnosti klasifikacije softvera dobijenom neuronskom mrežom.

[1] *Portable executables* – Format izvršnih fajlova, objekatskih fajlova, DLL fajlova i mnogih drugih, koji se koriste na 32-bitnom i 64-bitnom Windows operativnom sistemu.

7.3. Implementacija neuronske mreže za klasifikaciju softvera na benigni i maliciozni

Biblioteke

Osim osnovnih tensorflow i keras biblioteka, kao osnovnih koje koristimo u radu sa neuronskim mrežama čiji je backend tensorflow, u ovom projektu koristimo još tri biblioteke: numpy, matplotlib i os.

Naime, numPy je biblioteka u python-u namenjena za obimnije proračune nad podacima. Osim toga, pruža mogućnost skladištenja podataka multidimenzionalnih struktura, pruža mnoštvo logičkih, matematičkih i statističkih operacija koje je moguće izvršavati nad podacima, kao i mogućnost manipulacija nad veličinama i dimenzijama tih podataka. Naravno, ovo je samo deo osnovnih karakteristika i sposobnosti NumPy biblioteke. Da bismo ovu biblioteku mogli da koristimo, neophodno je da je instaliramo na sličan način na koji smo to učinili i za keras i za tensorflow:

pip install numpy

Matplotlib predstavlja biblioteku namenjenu za vizuelizaciju podataka u python-u. Korišćenjem ove biblioteke imamo mogućnost formiranja grafikona na jednostavan način. Pritom, najveći deo mogućnosti i sposobnosti ove biblioteke sadržan je unutar pyplot podmodula, koji je neophodno importovati zasebno, ne bi li se koristio. Ipak, naravno, nije je neophodno instalirati nezavisno od matplotlib modula, s obzirom na to da je u njemu sadržan. Za instaliranje matplotlib biblioteke neophodno je izvršiti komandu:

pip install matplotlib

I, na kraju, os biblioteka je jedna od najmoćnijih biblioteka u python-u, namenjena da pruži mogućnost izvršavanja funkcionalnosti koje su zavisne od operativnog sistema. Ovu biblioteku nije neophodno prethodno instalirati da bi se ona mogla importovati i koristiti u programu.

```
In [1]: import tensorflow as tf
        import keras
        import numpy as np
        import matplotlib.pyplot as plt
        import os
```

Slika 12 – Učitavanje neophodnih biblioteka

Učitavanje podataka

Kada je reč o razvoju neuronskih mreža potrebna su nam tri podskupa podataka formirana od inicijalnog dataset-a koji koristimo za njeno kreiranje i rad: trening skup podataka, validacioni skup podataka i test skup podataka. U slučaju razvoja ove konkretne neuronske mreže, iz inicijalnog dataset-a odvojeno je po 5 fotografija iz skupa malicioznih i benignih softvera i one su iskorišćene za formiranje test skupa podataka. Preostali deo korišćen je za treniranje i validaciju mreže.

S obzirom na činjenicu da su ulazni podaci ove mreže fotografije, koristimo metodu koju nam nudi tensorflow biblioteka: *image_dataset_from_directory*, kojoj prosleđivanjem odgovarajućih parametara u *train_ds, val_ds* i *test_ds* smeštamo fotografije i njima pridružene labele trening, validacionog i test skupa podataka, respektivno.

```
In [2]: train_ds = tf.keras.utils.image_dataset_from_directory(
          "C:\\Users\\HP\\data\\",
          validation_split=0.2,
          subset="training",
          seed=123,
          image_size=(64, 64),
          batch_size=32)

        val_ds = tf.keras.utils.image_dataset_from_directory(
          "C:\\Users\\HP\\data\\",
          validation_split=0.2,
          subset="validation",
          seed=123,
          image_size=(64, 64),
          batch_size=32)

        test_ds = tf.keras.utils.image_dataset_from_directory(
          "C:\\Users\\HP\\test\\",
          image_size=(64, 64),
          batch_size=32)

Found 126 files belonging to 2 classes.
Using 101 files for training.
Found 126 files belonging to 2 classes.
Using 25 files for validation.
Found 10 files belonging to 2 classes.
```

Slika 13 – Učitavanje trening, validacionih i test podataka

Da bismo utvrdili validnost učitanih podataka, prikazujemo ih i proveravamo.

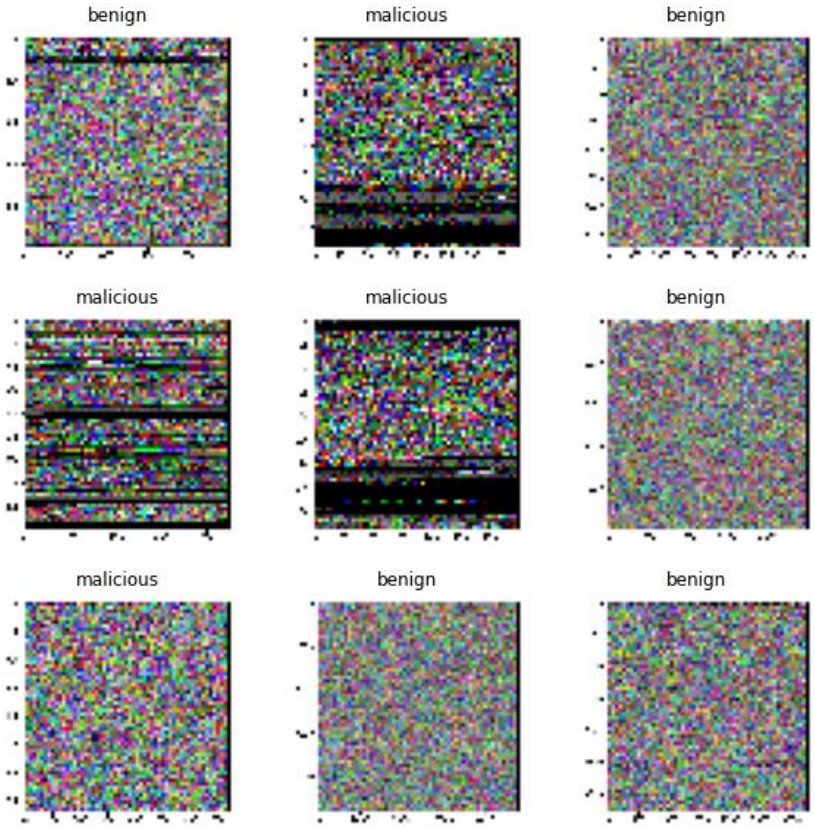
```
In [3]: class_names = train_ds.class_names
        print(class_names)

        plt.figure(figsize=(10, 10))
        for images, labels in train_ds.take(1):
            for i in range(9):
                ax = plt.subplot(3, 3, i + 1)
                plt.imshow(images[i].numpy().astype("uint8"))
                plt.title(class_names[labels[i]])
                plt.axis("off")

        for image_batch, labels_batch in train_ds:
            print(image_batch.shape)
            print(labels_batch.shape)
            break
```

Slika 14 – Naredbe za prikaz učitanih podataka

```
['benign', 'malicious']
(32, 64, 64, 3)
(32,)
```



Slika 15 – Rezultat izvršenja naredbi za prikaz učitanih podataka

Kreiranje modela

Korišćenje Sequential je jedan od mogućih načina za popunu modela neuronske mreže slojevima. Naime, u ovom slučaju lepimo slojeve jedan za drugim i na taj način formiramo i kompletnu mrežu. Dakle, kao što i sam naziv kaže – sekvencijalno navodimo slojeve mreže.

Odabir samih slojeva i broja slojeva je zapravo i najkompleksniji deo procesa formiranja neuronske mreže. Ove informacije se dobijaju testiranjem i traženjem rešenja koje bi dalo najbolje moguće rezultate.

Takođe, postoje dva načina na koje je moguće formirati slojeve: definisanjem aktivacione funkcije zajedno sa definisanjem samog sloja ili definisanjem ovih parametara nezavisno. Međutim, razlike u onome što se dobija ovim različitim pristupima nema.

Unutar modela koristi se pet različitih tipova slojeva:

- Rescaling
- Conv2D
- MaxPooling2D
- Flatten
- Dense

Rescaling je preprocessing sloj namenjen da izvrši reskaliranje ulaznih vrednosti i predstavi ih u novom opsegu. Pre svega, ovaj se tip sloja gotovo uvek koristi u slučaju kreiranja konvolucione neuronske mreže čiji su ulazni podaci fotografije, zato što je takve podatke neophodno normalizovati ne bi li dalji slojevi neuronske mreže mogli da ih obrađuju. Rescaling sloj radi tako što svaku ulaznu vrednost množi vrednošću koja je definisana kao scale parametar, a zatim dodaje vrednost definisanu kao offset parametar sloja. Međutim, offset u nekim slučajevima nije potreban, pa ga nije neophodno ni definisati, a, shodno tome, ukoliko se navede isključivo jedna vrednost kao parametar prilikom kreiranja ovakvog sloja, ona se odnosi na scale parametar, odnosno vrednost kojom se ulazna vrednost množi.

Na primer, ukoliko bismo želeli da ulazne vrednosti opsega [0, 255] pretvorimo u vrednosti opsega [0, 1], onda bismo kao scale parametar prosledili scale=1./255, dok, ukoliko bismo želeli da vrednosti istog ulaznog opsega pretvorimo u vrednosti opsega [-1, 1], kao scale parametar ovog sloja prosledili bismo scale=1./127.5, a kao offset parametar bismo prosledili -1.

Conv2D je 2D konvolucioni sloj koji kreira filtere i primenjuje ih nad ulaznim podacima, kreirajući tensor kao izlaz. Kako je ovo konvolucioni sloj, prlikom kreiranja duboke neuronske mreže, njega prate ili naredni konvolucioni ili pooling slojevi mreže.

MaxPooling2D je pooling sloj sa max pooling agregacionom funkcijom koju primenjuje nad svojim ulaznim podacima.

Flatten sloj vrši poravnanje nad podacima, dok Dense predstavlja fully-connected sloj.

Kada je reč o samim aktivacionim funkcijama u većini slojeva koriste se relu aktivacione funkcije, jer nije reč o značajno obimnom dataset-u. A, kako je reč o binarnoj

klasifikaciji, poslednji sloj koristi sigmoid aktivacionu funkciju, ne bi li generisao rezultat koji je 0 ili 1.

```
In [4]: num_classes = 2

model = tf.keras.Sequential([
  tf.keras.layers.Rescaling(1./255),
  tf.keras.layers.Conv2D(32, 3, activation='relu'),
  tf.keras.layers.MaxPooling2D(),
  tf.keras.layers.Conv2D(32, 3, activation='relu'),
  tf.keras.layers.MaxPooling2D(),
  tf.keras.layers.Conv2D(32, 3, activation='relu'),
  tf.keras.layers.MaxPooling2D(),
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(128, activation='softmax'),
  tf.keras.layers.Dense(1, activation='sigmoid'),
])
```

Slika 16 – Kreiranje modela

Kompajliranje modela

Tokom poziva compile metode nad kreiranim modelom neuronske mreže definiše se njen optimizator, funkcija gubitka i metrika. Najčešći tip optimizatora koji se koristi je adam.

Svrha funkcije gubitka jeste da mreža tokom procesa obučavanja prepozna na koji način treba da smanji gubitke i time poveća tačnost rezultata koje generiše. Dok je metrika zapravo parametar koji se posmatra za proračun mere performansi rada modela.

```
In [5]: model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

Slika 17 – Kompajliranje modela

Definisanje callback funkcija

Postoji veliki broj callback funkcija koje mogu biti iskorišćene u svrhe poboljšanja rada mreže tokom procesa njenog treniranja. Neke od njih su, između ostalog i early_stopping i reduce_lr. Naime, callback funkcije definišu se kao parametri prilikom poziva metode za obučavanje neuronske mreže.

early_stopping je callback funkcija koja treba da stopira proces obučavanja ukoliko se parametar definisan kao parametar metrike prestane da se poboljšava. Tako, ako je parametar metrike loss, odnosno gubitak, ukoliko se on ne smanji za patience definisan broj epoha, zaustaviće se proces obučavanja.

reduce_lr je callback funkcija koja bi trebalo da smanji learning rate ukoliko se on ne poboljša za definisani broj epoha.

```
In [6]:
early_stopping = keras.callbacks.EarlyStopping(monitor='val_accuracy', mode='max',
                                               patience=30, restore_best_weights=True, verbose=1)

reduce_lr = keras.callbacks.ReduceLROnPlateau(monitor='val_accuracy', mode='max',
                                              factor=0.1, patience=10, verbose=1)
```

Slika 18 – Definisanje callback funkcija

Obučavanje mreže

Da bismo proces obučavanja mreže otpočeli, neophodno je da nad kreiranim modelom pozovemo funkciju fit(). Njoj prosleđujemo trening podatke i očekivane rezultate, validacione podatke i očekivane rezultate, broj epoha, callback funkcije i još puno opcionih parametara.

```
In [8]: model.fit(
            train_ds,
            validation_data=val_ds,
            epochs=100,
            verbose=2,
            callbacks=[early_stopping, reduce_lr]
        )
```

Slika 19 – Obučavanje mreže

```
Epoch 1/100
4/4 - 11s - loss: 0.6415 - accuracy: 0.7525 - val_loss: 0.6329 - val_accuracy: 0.7600 - lr: 0.0010 - 11s/epoch - 3s/step
Epoch 2/100
4/4 - 11s - loss: 0.6204 - accuracy: 0.8515 - val_loss: 0.5949 - val_accuracy: 1.0000 - lr: 0.0010 - 11s/epoch - 3s/step
Epoch 3/100
4/4 - 12s - loss: 0.6166 - accuracy: 0.8614 - val_loss: 0.5915 - val_accuracy: 0.9600 - lr: 0.0010 - 12s/epoch - 3s/step
Epoch 4/100
4/4 - 11s - loss: 0.6150 - accuracy: 0.8416 - val_loss: 0.6015 - val_accuracy: 0.8800 - lr: 0.0010 - 11s/epoch - 3s/step
Epoch 5/100
4/4 - 12s - loss: 0.6154 - accuracy: 0.8317 - val_loss: 0.5892 - val_accuracy: 0.9200 - lr: 0.0010 - 12s/epoch - 3s/step
Epoch 6/100
4/4 - 11s - loss: 0.6072 - accuracy: 0.8713 - val_loss: 0.5972 - val_accuracy: 0.8800 - lr: 0.0010 - 11s/epoch - 3s/step
Epoch 7/100
4/4 - 11s - loss: 0.6080 - accuracy: 0.8416 - val_loss: 0.6001 - val_accuracy: 0.8800 - lr: 0.0010 - 11s/epoch - 3s/step
Epoch 8/100
4/4 - 11s - loss: 0.5998 - accuracy: 0.8812 - val_loss: 0.5960 - val_accuracy: 0.8800 - lr: 0.0010 - 11s/epoch - 3s/step
Epoch 9/100
4/4 - 11s - loss: 0.6238 - accuracy: 0.7822 - val_loss: 0.6005 - val_accuracy: 0.8800 - lr: 0.0010 - 11s/epoch - 3s/step
Epoch 10/100
```

Slika 20 – Proces obučavanja mreže

```
Epoch 10/100
4/4 - 12s - loss: 0.6054 - accuracy: 0.8515 - val_loss: 0.6093 - val_accuracy: 0.8000 - lr: 0.0010 - 12s/epoch - 3s/step
Epoch 11/100
4/4 - 11s - loss: 0.5968 - accuracy: 0.8911 - val_loss: 0.5745 - val_accuracy: 0.9600 - lr: 0.0010 - 11s/epoch - 3s/step
Epoch 12/100

Epoch 12: ReduceLROnPlateau reducing learning rate to 0.00010000000474974513.
4/4 - 11s - loss: 0.6011 - accuracy: 0.8614 - val_loss: 0.6196 - val_accuracy: 0.8000 - lr: 0.0010 - 11s/epoch - 3s/step
Epoch 13/100
4/4 - 11s - loss: 0.6111 - accuracy: 0.8218 - val_loss: 0.6187 - val_accuracy: 0.8000 - lr: 1.0000e-04 - 11s/epoch - 3s/step
Epoch 14/100
4/4 - 12s - loss: 0.6097 - accuracy: 0.8218 - val_loss: 0.6158 - val_accuracy: 0.8000 - lr: 1.0000e-04 - 12s/epoch - 3s/step
Epoch 15/100
4/4 - 12s - loss: 0.6074 - accuracy: 0.8218 - val_loss: 0.6125 - val_accuracy: 0.8000 - lr: 1.0000e-04 - 12s/epoch - 3s/step
Epoch 16/100
4/4 - 11s - loss: 0.6043 - accuracy: 0.8218 - val_loss: 0.6071 - val_accuracy: 0.8000 - lr: 1.0000e-04 - 11s/epoch - 3s/step
```

Slika 21 – Primer poziva reduce_lr callback funkcije u procesu obučavanja mreže

```
Epoch 20/100
4/4 - 12s - loss: 0.5967 - accuracy: 0.8614 - val_loss: 0.5793 - val_accuracy: 0.9200 - lr: 1.0000e-04 - 12s/epoch - 3s/step
Epoch 21/100
4/4 - 12s - loss: 0.5971 - accuracy: 0.8515 - val_loss: 0.5738 - val_accuracy: 0.9600 - lr: 1.0000e-04 - 12s/epoch - 3s/step
Epoch 22/100

Epoch 22: ReduceLROnPlateau reducing learning rate to 1.0000000474974514e-05.
4/4 - 11s - loss: 0.5971 - accuracy: 0.8713 - val_loss: 0.5720 - val_accuracy: 0.9600 - lr: 1.0000e-04 - 11s/epoch - 3s/step
Epoch 23/100
4/4 - 12s - loss: 0.5969 - accuracy: 0.8614 - val_loss: 0.5713 - val_accuracy: 0.9600 - lr: 1.0000e-05 - 12s/epoch - 3s/step
Epoch 24/100
4/4 - 12s - loss: 0.5970 - accuracy: 0.8614 - val_loss: 0.5711 - val_accuracy: 0.9600 - lr: 1.0000e-05 - 12s/epoch - 3s/step
Epoch 25/100
4/4 - 11s - loss: 0.5970 - accuracy: 0.8614 - val_loss: 0.5710 - val_accuracy: 0.9600 - lr: 1.0000e-05 - 11s/epoch - 3s/step
Epoch 26/100
```

Slika 22 - Primer poziva reduce_lr callback funkcije u procesu obučavanja mreže

```
Epoch 30/100
4/4 - 12s - loss: 0.5965 - accuracy: 0.8713 - val_loss: 0.5735 - val_accuracy: 0.9600 - lr: 1.0000e-05 - 12s/epoch - 3s/step
Epoch 31/100
4/4 - 12s - loss: 0.5963 - accuracy: 0.8713 - val_loss: 0.5739 - val_accuracy: 0.9600 - lr: 1.0000e-05 - 12s/epoch - 3s/step
Epoch 32/100
Restoring model weights from the end of the best epoch: 2.

Epoch 32: ReduceLROnPlateau reducing learning rate to 1.0000000656873453e-06.
4/4 - 12s - loss: 0.5962 - accuracy: 0.8713 - val_loss: 0.5741 - val_accuracy: 0.9600 - lr: 1.0000e-05 - 12s/epoch - 3s/step
Epoch 32: early stopping
Out[8]: <keras.callbacks.History at 0x2c802b5e580>
```

Slika 23 – Rezultat obučavanja mreže

Evaluacija mreže

Za evaluaciju rada neuronske mreže potrebno je da nad modelom izvršimo metodu evaluate. Njoj prosleđujemo test dataset i ona nam pruža rezultate rada mreže nad test podacima.

```
In [9]: res = model.evaluate(test_ds, batch_size=32)
        print(res)

        1/1 [==============================] - 2s 2s/step - loss: 0.6309 - accuracy: 0.8000
        [0.6308954358100891, 0.800000011920929]
```

Slika 24 – Evaluacija mreže

Testiranje rada mreže

Za testiranje mreže koristimo predict metodu koju pozivamo nad modelom i njoj prosleđujemo test podatke, dok ona nad njima izvršava predikciju da li se radi o benignom ili malicioznom softveru.

```
In [10]: y_out = model.predict(test_ds, batch_size=32)
```

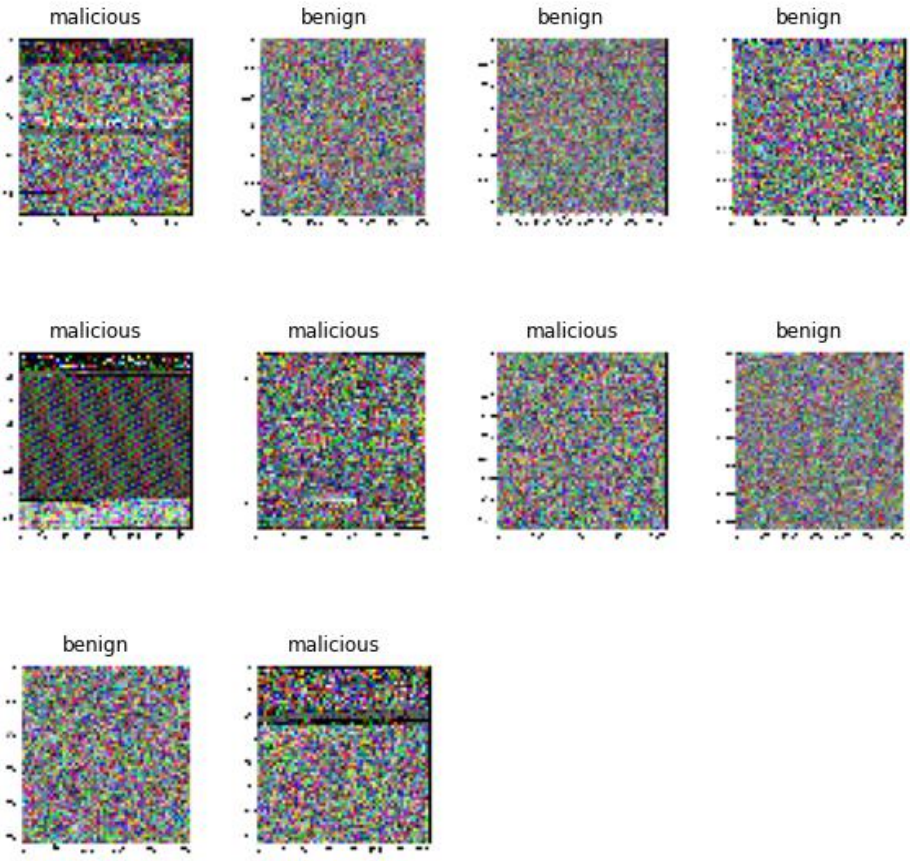
Slika 25 – Testiranje rada mreže

```
In [11]: plt.figure(figsize=(10, 10))
         for images, labels in test_ds.take(1):
             for i in range(10):
                 ax = plt.subplot(3, 4, i + 1)
                 plt.imshow(images[i].numpy().astype("uint8"))
                 plt.title(class_names[labels[i]])
                 plt.axis("off")


         print(y_out)

         for i in range(len(y_out)):
           if y_out[i] > 0.5:
             print('malicious')
           else:
             print('benign')
```

Slika 26 – Naredbe za prikaz rezultata rada mreže i test podataka



Slika 27 – Test podaci

```
[[0.53744644]
 [0.44192073]
 [0.49370524]
 [0.5628719 ]
 [0.44215605]
 [0.5624045 ]
 [0.44604355]
 [0.44190818]
 [0.5628491 ]
 [0.562793  ]]
malicious
benign
benign
malicious
benign
malicious
benign
benign
malicious
malicious
```

Slika 28 – Rezultat predikcije mreže

Zaključak

Konvolucione neuronske predstavljaju veoma efikasnu tehniku dubokog učenja za probleme vizuelnog prepoznavanja. Kao i svi alati koje duboko učenje omogućavava, sve neuronske mreže zavise od veličnike i kvaliteta podataka u samom procesu obučavanja. U slučaju kada imamo dobro pripremljen dataset, neuronske mreže imaju spospobnost da nadmaše mogućnost ljudskog perceptiranja kod problema vizuelnog prepoznavanja. Međutim, one još uvek nisu otporne na tzv. *side effects*, kao što je na primer odsjaj. Zato su i dalje u centru razmatranja, kako bi te nedostatke nadomestile.

Literatura

https://livebook.manning.com/book/deep-learning-with-python/chapter-3/125

https://www.kaggle.com

https://archive.ics.uci.edu/ml/index.php

https://www.ibm.com/cloud/learn/convolutional-neural-networks

https://numpy.org/doc/stable/index.html

https://www.tensorflow.org/api_docs/python/tf/keras/layers/Rescaling

https://keras.io/