SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni diplomski studij računarstva

DETEKCIJA PUKOTINA NA SLIKAMA NEURNSKOM MERŽOM

Projektni zadatak

Bruno Šimunović

Osijek, 2022.

Sadržaj

1. UVOD 3
1.1. Cilj projektnog zadataka 3
2. KORIŠTENE TEHNOLOGIJE 4
2.1. OpenCV biblioteka 4
2.2. NumPy 4
2.3. Tensorflow 4
2.4. Pandas 5
3. DUBOKO UČENJE 6
3.1. PROPAGACIJA UNAPRIJED I UNAZAD 7
3.2. KONVOLUCIJSKA NEURONSKA MREŽA 8
3.2.1. ResNet 9
4. IZRADA PROJEKTNOG ZADATKA 11
4.1. Prikupljanje podataka 11
4.2. Segmentacija slika 12
4.3. Treniranje modela 14
4.4. Evaluacija modela 15
4.4.1. Rezultati testiranja 18
5. ZAKLJUČAK 20
6. LITERATURA 21

1. UVOD

U današnje vrijeme većina objekata koja nas okružuje je napravljena od betona, te ti objekti su podložni velikom naporu iz čega proizlaze pukotine u betonu. Do pucanja betona dolazi i pri drugim čimbenicima kao što su staranje i vremenski uvjeti. Kako bi se izbjegla velika katastrofa potrebno je u pravo vrijeme promaći pukotinu te proglasiti objekt nesigurnim za korištenje, a za to su potrebni istrenirani inženjeri. Ovo nije jednostavno te lako može doći do pogreške. Izbjegavanje ovakve pogreške skoro u potpunosti omogućavaju nam današnje tehnologije poput računalnog vida i neuronskih mreža.

1.1. Cilj projektnog zadataka

Cilj ovoga projekta je rješavanje problema detekcije pukotina tako da se umanji utjecaj čovjeka te time smanji mogućnost pogreške. Problem bi se trebao riješiti pomoću uporabe neuronske mreže. Rješenje mora prepoznati dali se na priloženoj slici s kamere nalazi pukotina ili ne.

2. KORIŠTENE TEHNOLOGIJE

Za rješavanje problema projektnog zadatka potrebno je koristiti biblioteke koje su podržane od strane programskog jezika Pythona. Za rješavanje i prepoznavanje znakovnog jezika u stvarnom vremenu koriste se biblioteke OpenCV, NumPy i Tensorflow, Mediapipe.

2.1. OpenCV biblioteka

OpenCV [1] (engl. Open Source Computer Vision) je biblioteka otvorenog koda koja sadrži programske funkcije za računalni vid i strojno učenje. Izgrađen je kako bi osigurao zajedničku infrastrukturu za aplikacije računalnog vida i ubrzao korištenje strojne percepcije u komercijalnim proizvodima. Biblioteka sadrži više od 2500 optimiziranih algoritama, među kojima je i mnogo najsuvremenijih algoritama u području računalnog vida i strojnog učenja. OpenCV se primjenjuje u prepoznavanju osoba preko kamere, praćenja ljudskih pokreta, prepoznavanja krajolika, praćenja objekata u pokretu, obavljanju određenih operacija na računalu uz pomoć kamere, itd.. Postoje implementacije za C++, Python i Java programske jezike uz podršku za Windows, Linux, Android i MacOS operacijske sustave. OpenCV se najviše orijentira prema aplikacijama koje informacije primaju u stvarnom vremenu.

2.2. NumPy

NumPy [4] je biblioteka otvorenog koda za znanstveno računanje u programskom jeziku Python. Sadrži podršku za velike, višedimenzionalne nizove i matrice s velikom zbirkom matematičkih funkcija visoke razine za rad na tim nizovima. Korištenje NumPy u Python-u omogućuje funkcionalnosti koje su usporedive s korištenjem MATLAB programa. NumPy se može koristiti kao višedimenzionalni spremnik generalnih podataka.

2.3. Tensorflow

TensorFlow [3] je besplatna softverska knjižnica otvorenog koda za strojno učenje i umjetnu inteligenciju. Može se koristiti u nizu zadataka, ali se posebno fokusira na obuku i zaključivanje dubokih neuronskih mreža. TensorFlow je razvio tim Google Brain za internu Googleovu upotrebu u istraživanju i proizvodnji. Treniranje modela može se izvesti preko procesorske ili grafičke

jedinice. Mogućnost integriranja u Python programski jezik čini ga pogodnim za korištenje prilikom kreiranja i evaluacije modela.

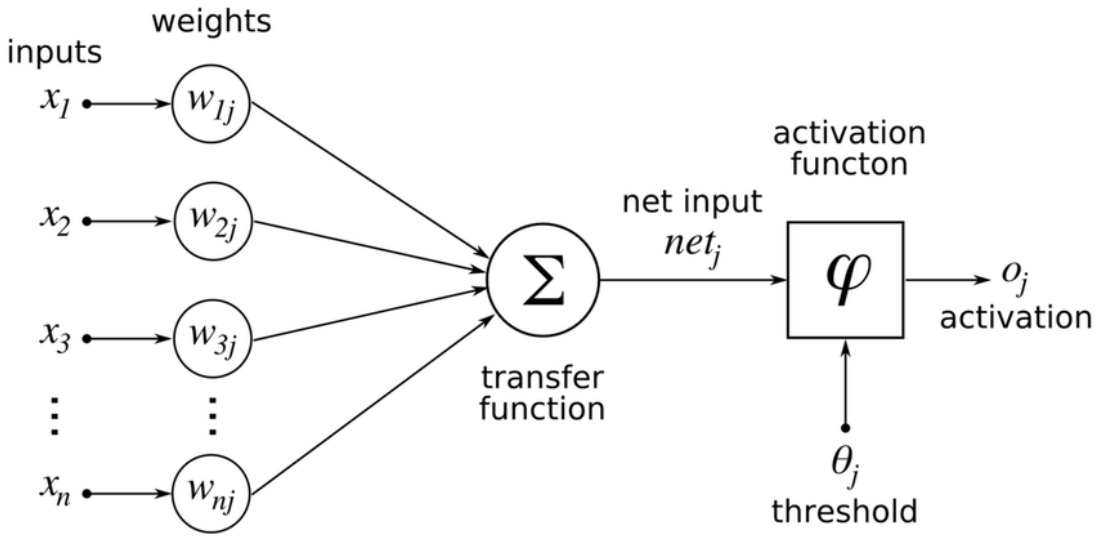
2.4. Pandas

Pandas [2] predstavlja brzu i fleksibilnu open source biblioteku za Python programski jezik koja pruža visoke performanse za cjeli proces analize podataka na jednostavan i intuitivan način. Ima cilj da postane jedna od najmoćnijih i najfleksibilnijih open source alata za analizu i manipulaciju podataka. Namenjena je za sve Python korisnike koji žele da rade sa podacima i koriste moćan alat za to.Pored biblioteka Matplotlib i NumPy, Pandas je jedna od najkorištenijih kada je u pitanju „data science“. Originalni autor je Wes McKinney a posljednja stabilna verzija je 1.5.0 koja je izašla ove godine.

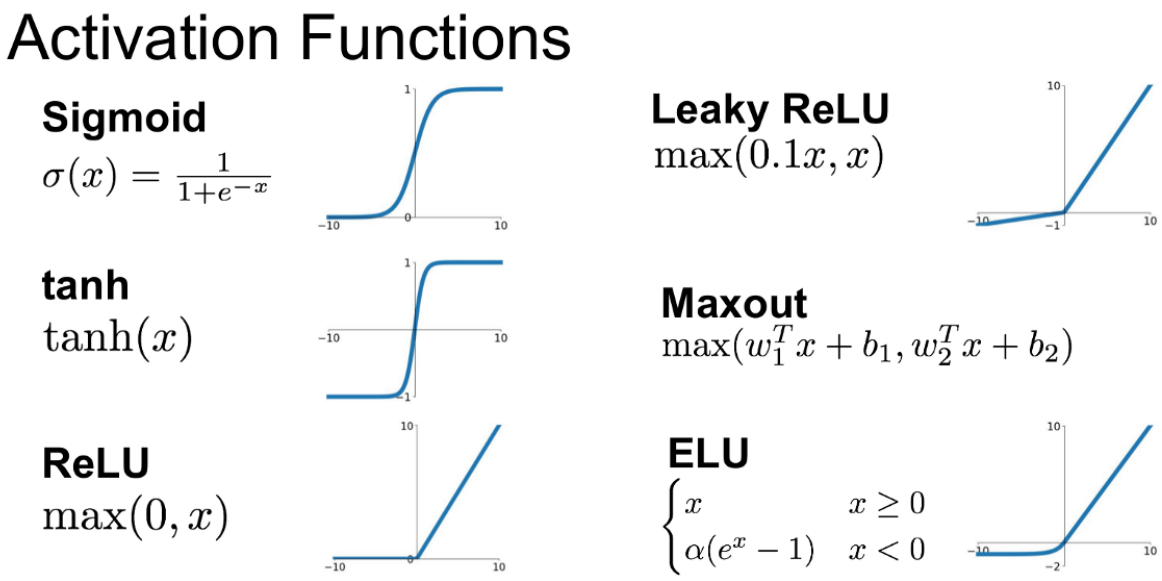
3. DUBOKO UČENJE

Duboko učenje [5] je grana strojnog učenja koja za složenu vrstu podataka nudi rješenja do kojih se dolazi slijedom istreniranih nelinearnih transformacija. Sastoji se od velikog broja procesora zvanih neuroni koji pokušavaju imitirati ljudski mozak. Trenutno se koristi u raznim stvarnim problemima kao npr. Automobilska industrija, predviđanje rasta/pada dionica, kamerama, medicini itd. Potreba za dubokim učenjem stvara se kada je potrebno procesirati veliki broj značajki gdje su podaci nestrukturirani.

Duboko učenje se često naziva i duboka neuronska mreža zbog korištenja arhitekture neuronske mreže. Takva mreža se sastoji od brojnih neurona (Slika 3.1.) koji sadrže vrijednost koja se mjenja tokom učenja zvana težina. Dodatno, pri učenju neuronima aktivacijska funkcija omogućava korištenje korisnih i odbacivanje nerelativnih informacija, tj. onda omogućava učenje. Postoje razne aktivacijske funkcije [7] (Slika 3.2.) , a među najpoznatijima su reLU, Sigmoid, Leaky reLu, Tanh. Ovisno o vrsti našeg problema biramo tip aktivacijske funkcije. Neki neuroni mogu aktivirati niz drugih neurona u ovisnosti kao je neuronska mreža istrenirana i time se dolazi do predviđenog rezultata.



Slika 3.1. Neuron neuronske mreže

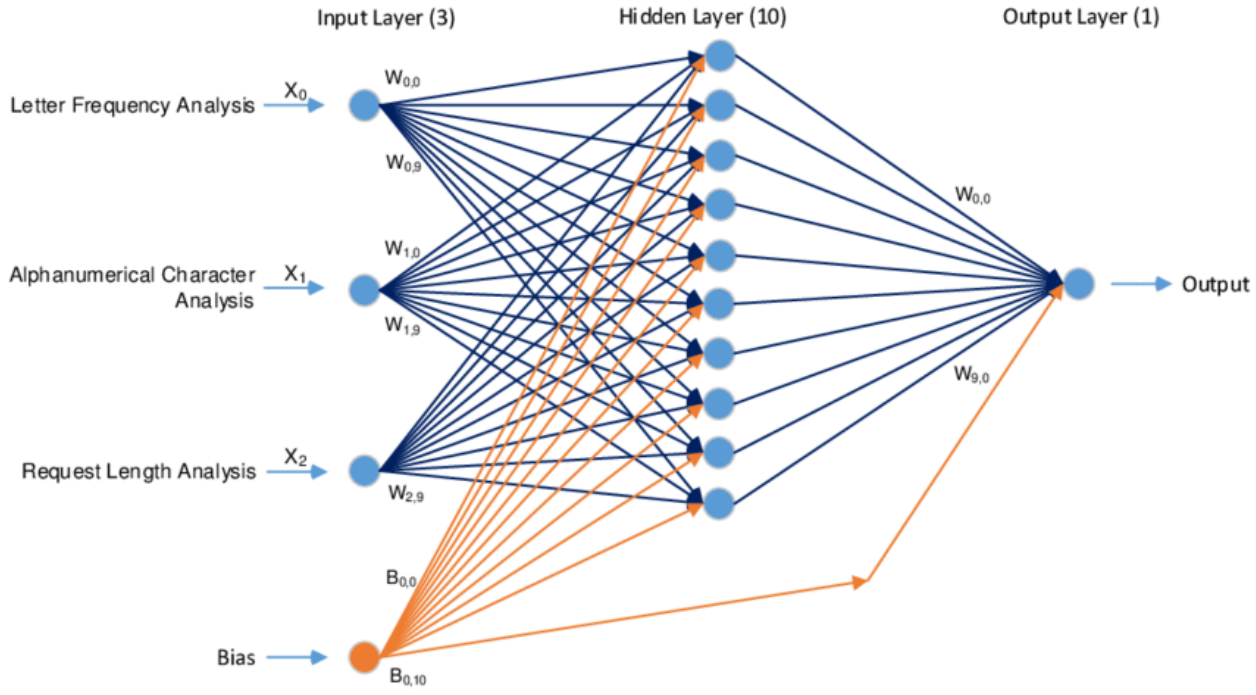


Slika 3.2. Aktivacijske funkcije

Neuronska mreža (Slika 3.3.) sastoji se od ulaznog sloja, srednjih slojeva i izlaznog sloja. Unutar ulaznog sloja možemo imati više ulaza sa ulaznim vrijednostima koje se prosljeđuju srednjem sloju. Srednji sloj se još naziva i skriveni sloj, a može ih biti više. Što je više skrivenih slojeva to je mreža kompleksnija, te ima mogućnosti rješavanja kompleksnijih problema. I na kraju imamo izlazni sloj gdje dobivamo rješenje neuronske mreže. Neuronska mreža ima poseban proces učenja a to je propagacija unaprijed i unazad. Postoji par vrsta neuronskih mreža poput konvolucijske neuronske mreže,neuronske mreže s povratnom vezom, kohonelove samo organizirajuče mreže itd.

3.1. PROPAGACIJA UNAPRIJED I UNAZAD

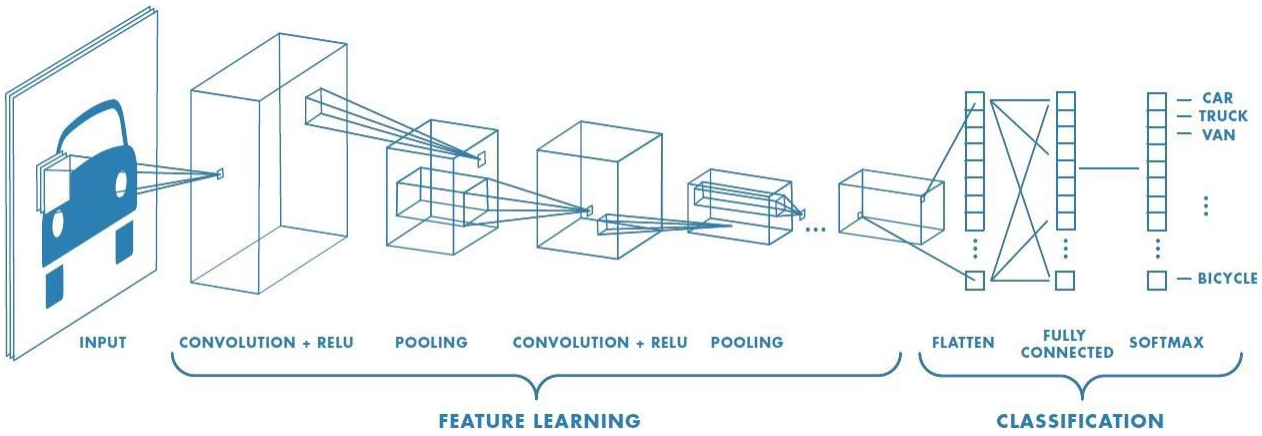
Propagiranje naprijed je način prelaska s ulaznog sloja (lijevo) na izlazni sloj (desno) u neuronskoj mreži. Proces pomicanja s desna na lijevo tj. unatrag od izlaznog prema ulaznom sloju naziva se propagacija unatrag. Propagacija unatrag je poželjna metoda za prilagodbu težina budući da se brže približava kako se krećemo od izlaza do skrivenog sloja. Ovdje mijenjamo težine skrivenog sloja koji je najbliži izlaznom sloju, ponovno izračunavamo gubitak i ako je potrebno dodatno smanjiti grešku onda ponavljamo cijeli proces i tim redoslijedom idemo prema ulaznom sloju.



Slika 3.3. Neuronska mreža

3.2. KONVOLUCIJSKA NEURONSKA MREŽA

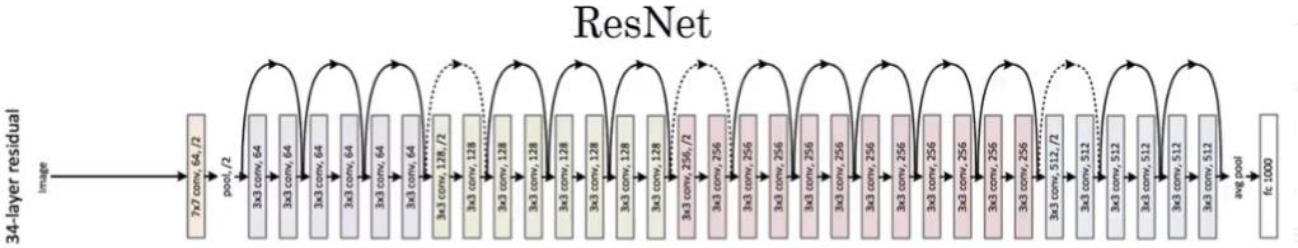
Konvolucijske neuronske mreže (Slika 3.4.) razlikuju se od ostalih neuronskih mreža po svojoj superiornoj izvedbi sa ulazima slike, govora ili audio signala. Imaju tri glavne vrste slojeva, a to su: Konvolucijski sloj, Pooling sloj i potpuno povezani sloj. Konvolucijski sloj je prvi sloj konvolucijske mreže. Dok konvolucijski slojevi mogu biti praćeni dodatnim konvolucijskim slojevima ili pooling slojevima, potpuno povezani sloj je završni sloj. Sa svakim slojem, konvolucijska neuronska mreža povećava svoju složenost, identificirajući veće dijelove slike. Raniji slojevi fokusiraju se na jednostavne značajke, kao što su boje i rubovi. Kako slike napreduju kroz slojeve konvolucijske neuronske mreže, on počinje prepoznavati neke uzorke dok konačno ne identificira željeni objekt.



Slika 3.5. Konvolucijska neuronska mreža

3.2.1. ResNet

Preostala neuronska mreža (ResNet) [8] je umjetna neuronska mreža (ANN). To je varijanta funkcionalne vrlo duboke neuronske mreže sa stotinama slojeva. Preskačući veze ili prečaci se koriste za preskakanje nekih slojeva. Tipični ResNet modeli implementirani su s dvostrukim ili troslojnim preskakanjem koje sadrže nelinearnosti (ReLU) i batch normalizaciju između. Modeli s nekoliko paralelnih preskakanja nazivaju se DenseNets. U kontekstu zaostalih neuronskih mreža, nerezidualna mreža može se opisati kao obična mreža. Postoje dva glavna razloga za dodavanje preskakanja veza: izbjegavanje problema nestajanja gradijenta, što dovodi do lakše optimizacije neuronskih mreža, gdje mehanizmi zatvaranja olakšavaju protok informacija kroz mnoge slojeve ("informacijske autoceste") ili ublažavaju problem degradacije (zasićenja točnosti); gdje dodavanje više slojeva prikladno dubokom modelu dovodi do veće pogreške u treningu. Tijekom treninga, utezi se prilagođavaju kako bi utišali gornji sloj i pojačali prethodno preskočeni sloj. U najjednostavnijem slučaju, prilagođavaju se samo težine za vezu susjednog sloja, bez eksplicitnih pondera za gornji sloj. Ovo najbolje funkcionira kada se prekorači jedan nelinearni sloj ili kada su svi međuslojevi linearni. Preskakanje učinkovito pojednostavljuje mrežu, koristeći manje slojeva u početnim fazama obuke. To ubrzava učenje smanjenjem utjecaja nestajućeg gradijenta, budući da postoji manje slojeva za širenje. Mreža zatim postupno obnavlja preskočene slojeve dok uči prostor značajki. Pred kraj treninga, kada su svi slojevi prošireni, ostaje bliže razdjelniku i time brže uči. Neuronska mreža bez rezidualnih dijelova istražuje više prostora značajki.



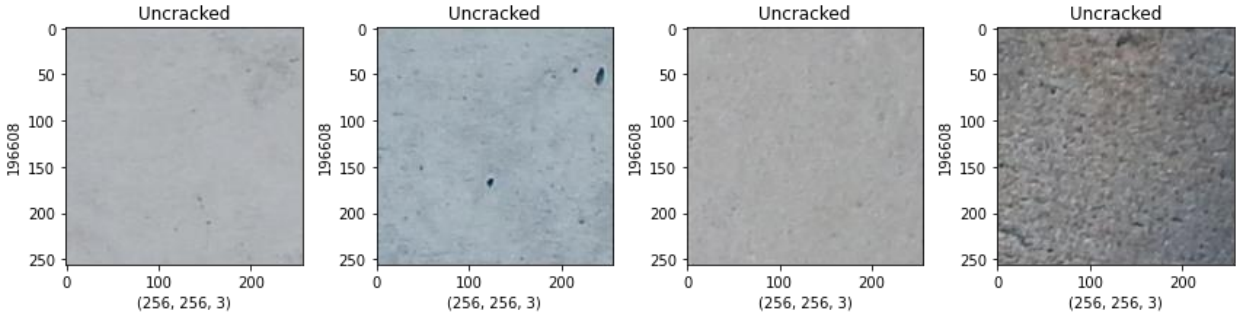
Slika 3.6. Arhitektura ResNeta od 34 sloja

4. IZRADA PROJEKTNOG ZADATKA

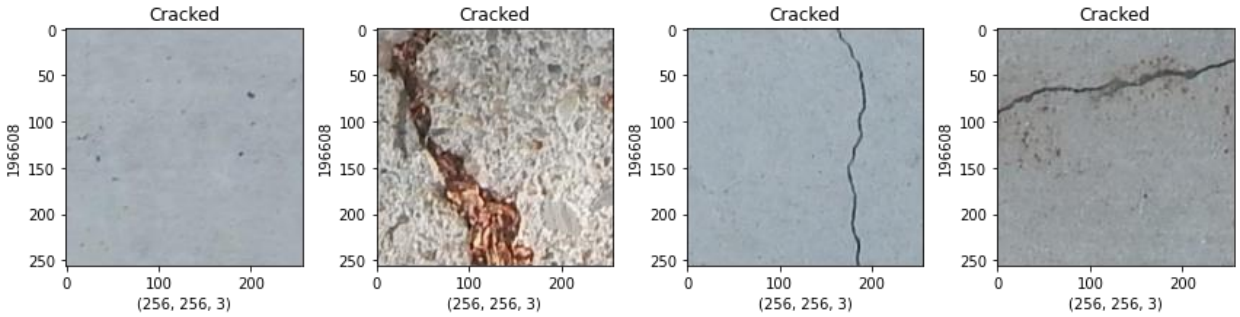
Projekt je izrađen u programskom jeziku Python, koristeći biblioteke OpenCV, Tensorflow, Pandas i Numpy. Podaci koji se koriste su prikupljeni sa stranice kaggle. Kreirani su različiti modeli konvolucijske mreže za riješavanje problema te najbolji je korišten za testiranje. Problem detekcije pukotina predstavlja klasifikacijski problem.

4.1. Prikupljanje podataka

Za treniranje mreže korišteni su podaci koji sadrže 7051 slika sa pukotinama, 7467 bez pukotina i 2000 ne sortiranih. Podaci sa pukotinama i bez se nalaze u datoteci *train* pod imenom *cracked* i *uncracked*, a nesortirani podaci su u *test* datoteci. Na slikama 4.1 i 4.2 mogu se vidjeti neki od uzoraka iz tih datoteka.



Slika 4.1. Bez pukotina u betonu



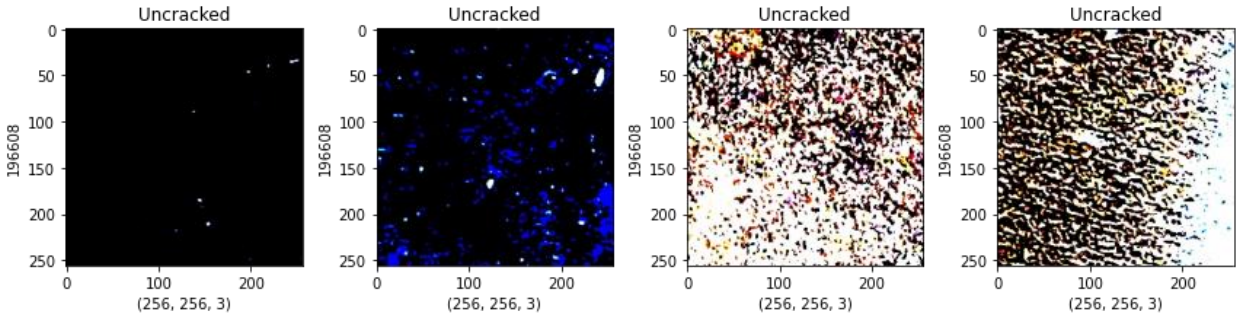
Slika 4.2. Sa pukotinama u betonu

Prije kreiranja modela mreže prvo je definirana veličina slike, broj slika koji će se trenirati po koraku, ukupni broj klasa, optimizator, funkcija gubitka, te se učitavaju slike pomoću

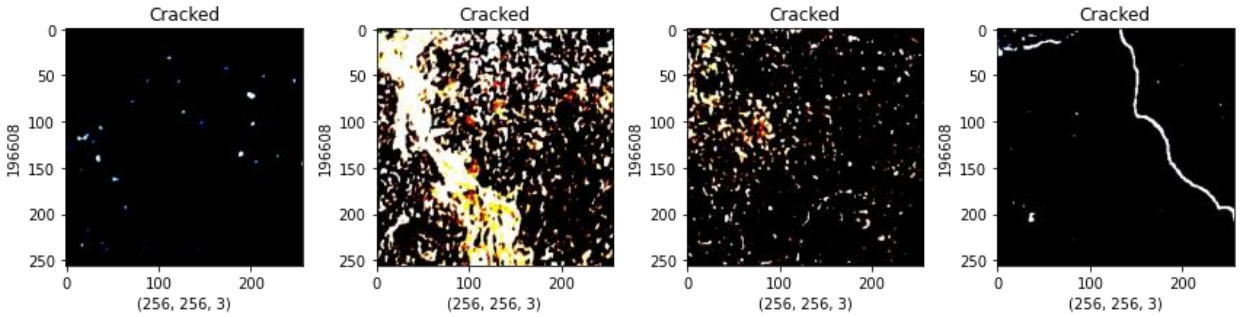
funkcije *flow_from_directory( )* i prireduju za treniranje *train_ds* i za testiranje *test_ds* pomoću *tensorflow* funkcije *ImageDataGenerator()*. Funkciji prosljeđujemo veličinu seta za validaciju što je u ovom slučaju 10% i dodatno parametre za uređivanje slika (npr. Rotacija slike, zumiranje, svjetlina, pomicanje itd.) kako bi se mreža susrela sa različitim slučajevima. Bitno je napomenuti da *ImageDataGenerator()* mijenja vrijednosti piksela slika na vrijednosti između 0 i 1 tako što dijeli svaki piksel sa vrijednosti od 255. To se odrađuje kako bi se merža normalizirala te ubrzala.

4.2. Segmentacija slika

Prije samog treniranja, primjenjene su različite metode obrade slike kako bi što bolje stekli znanje o podacima s kojima se raspolaže. Jedna od metoda se zove thresholding ili postavljanje praga koja se koristi iz biblioteke računalnog vida kako bi se prikazali pikseli koji nas „zanimaju“. Te piksele dobivamo tako da odredimo granice praga funkcije cv2.threshold(slika, minimalnaVrijednostPraga, maximalnaVrijednostPraga, metodaThersholdinga). Rezultat možemo vidjeti na Slici 4.3 i 4.4.

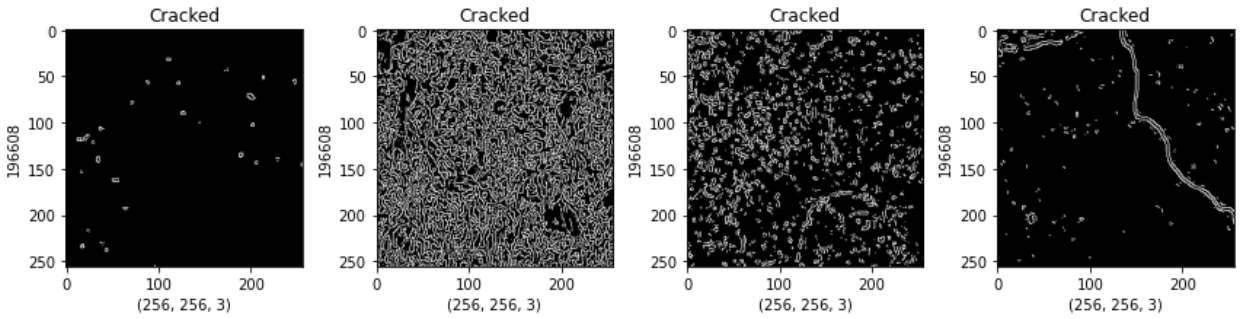


Slika 4.3. Bez pukotina u betonu pomoću funkcije „*threshold“*

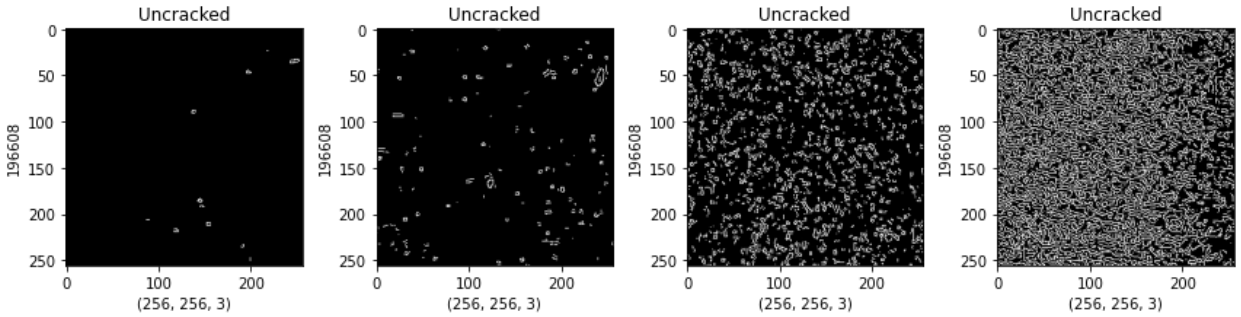


Slika 4.4. Sa pukotinama u betonu pomoću funkcije „*threshold“*

Kako bi dalje obradili sliku i otkrili rubove pukotina primjenjuje se Canny algoritam (engl. *Canny edge detection*) [9]. Canny detekcija rubova nam omogućava uklanjanje rubova koji nas „ne zanimaju“ kako bi uspjeli dobiti što bolji prikaz konture promatranog objekta. Uz njega se primjenjuje i Gaussov filter [10] koji nam omogućava reduciranje smetnji te rezultat primjene tih algoritama su Slike 4.5 i 4.6.

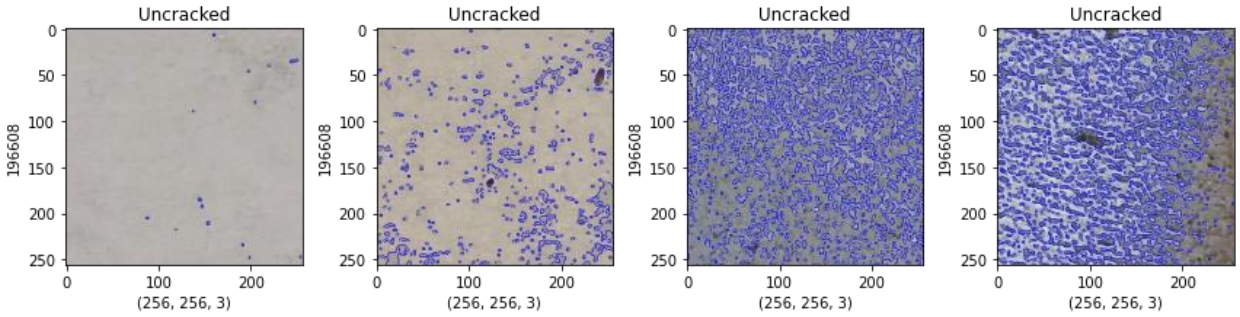


Slika 4.5. Bez pukotina u betonu pomoću Canny algoritma

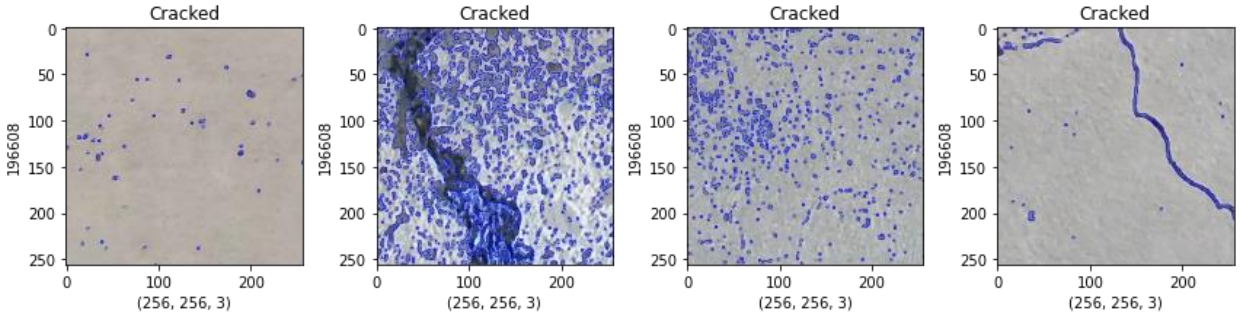


Slika 4.6. Sa pukotinama u betonu pomoću Canny algoritma

Možemo primijetiti da na nekim od slika se teško ne vidi razlika dali je ona s pukotinom ili bez, kako bi se ova analiza slike poboljšala primjenjuju se dodatno funkcije *cv2.findCounturs()* i *cv2.drawConturs().* One nam omogućavaju da eliminiramo kraće beznačajne linije te samo ostavimo duže te ih nacrtamo na originalnoj slici. Rezultat je vidljiv na Slikama 4.7 i 4.8.



Slika 4.7. Bez pukotina u betonu pomoću *findCounturs*() i *drawConturs*() funkcija



Slika 4.8. Sa pukotinama u betonu pomoću *findCounturs*() i *drawConturs*() funkcija

Iz priloženog da se zaključiti da i nakon primjene ovih funkcija na nekim slikama može biti teško odlučiti dali se radi o slici s pukotinom ili bez, te to može buniti neuronsku mrežu pri treniranju, jer neki podaci nisi ni nama najjasniji.

4.3. Treniranje modela

Kako bi se što bolje istrenirala mreža primijenjeno je više modela mreža sa različitim slojevima, različite veličine slika, različitom agmentacijom slika. Prilikom treniranja mreže korišten je *batchSize* od 20, 32, 64, 128 što utječe na broj podataka koji se treniraju po koraku. Korištena je veličina slike od 256x256, 126x126, 64x64 piksela. Od slojeva je korišten ulazni, konvolucijski, maxpooling, potpuno povezani sloj. Konvolucijski sloj *Conv2D()* kreira konvolucijsku jezgru i time obrađuje slike u ovisnosti o veličini jezgre i broju filtera. Dodatno, koristi se reLU aktivacijska funkcija zbog brzine i jednostavnosti, te se smatra boljom od sigmoid funkcije za neuronske mreže. Nakon konvolucijskg sloja koristi se Maxpooling sloj koji smanjuje broj dobivenih parametara, uz to se koristi i *Dropout()* sloj. Takav sloj nam služi da smanjimo „over-fitting“ mreže tako što ignoriramo određene neurone. Nakon svih tih slojeva korišten je

BatchNormalization() sloj radi stabilizacije modela. Na kraju ide potpuno povezani sloj u koji sastoji se od više slojeva kako bi postigli jednodimenzionalni vektor te dobili željene vrijednosti vektora. Za sloj koristimo *Flatten()* sloj i *Dense()* sloj sa softmax aktivacijskom funkcijom. U početku su testirani proizvoljni modeli. Takvi modeli su imali loše rezultate, pa su korišteni već dobro poznati modeli kao ResNet i Unet koji imaju dobre rezultati za ovakve probleme. Testiran je ResNet model od 18, 34, 50 slojeva te mreže i U net. Arhitektura modela se nalazi na slici 4.9.

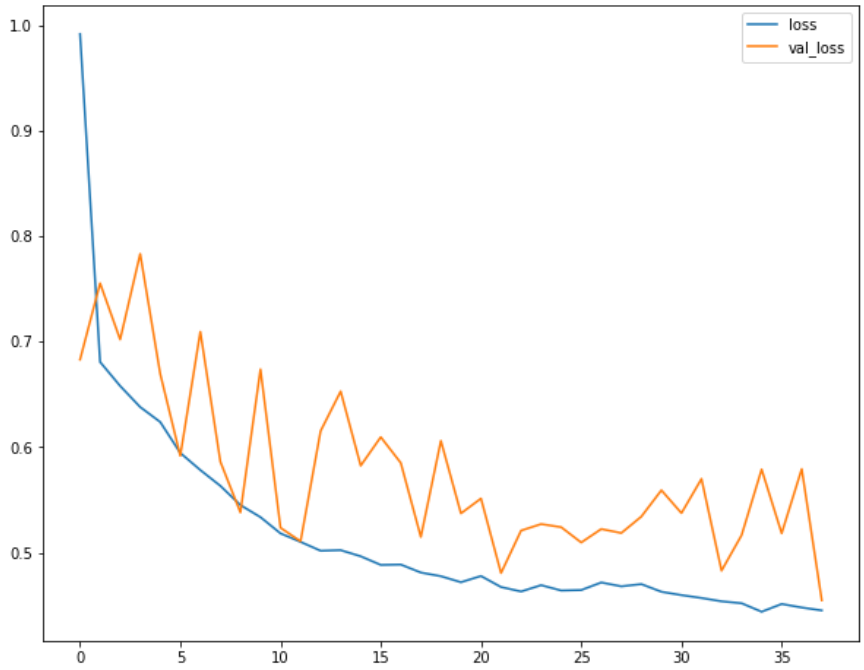
```
Model: "ResNet34"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 256, 256, 3) 0
__________________________________________________________________________________________________
zero_padding2d (ZeroPadding2D)  (None, 262, 262, 3)  0           input_1[0][0]
__________________________________________________________________________________________________
conv2d (Conv2D)                 (None, 131, 131, 32) 4736        zero_padding2d[0][0]
__________________________________________________________________________________________________
batch_normalization (BatchNorma (None, 131, 131, 32) 128         conv2d[0][0]
__________________________________________________________________________________________________
activation (Activation)         (None, 131, 131, 32) 0           batch_normalization[0][0]
__________________________________________________________________________________________________
max_pooling2d (MaxPooling2D)    (None, 66, 66, 32)   0           activation[0][0]
__________________________________________________________________________________________________
conv2d_1 (Conv2D)               (None, 66, 66, 32)   9248        max_pooling2d[0][0]
__________________________________________________________________________________________________
batch_normalization_1 (BatchNor (None, 66, 66, 32)   128         conv2d_1[0][0]
__________________________________________________________________________________________________
activation_1 (Activation)       (None, 66, 66, 32)   0           batch_normalization_1[0][0]
__________________________________________________________________________________________________
conv2d_2 (Conv2D)               (None, 66, 66, 32)   9248        activation_1[0][0]
__________________________________________________________________________________________________
batch_normalization_2 (BatchNor (None, 66, 66, 32)   128         conv2d_2[0][0]
...
Total params: 8,616,577
Trainable params: 8,608,961
Non-trainable params: 7,616
__________________________________________________________________________________________________
```

Slika 4.10. Kratki opis arhitekture ResNet 34

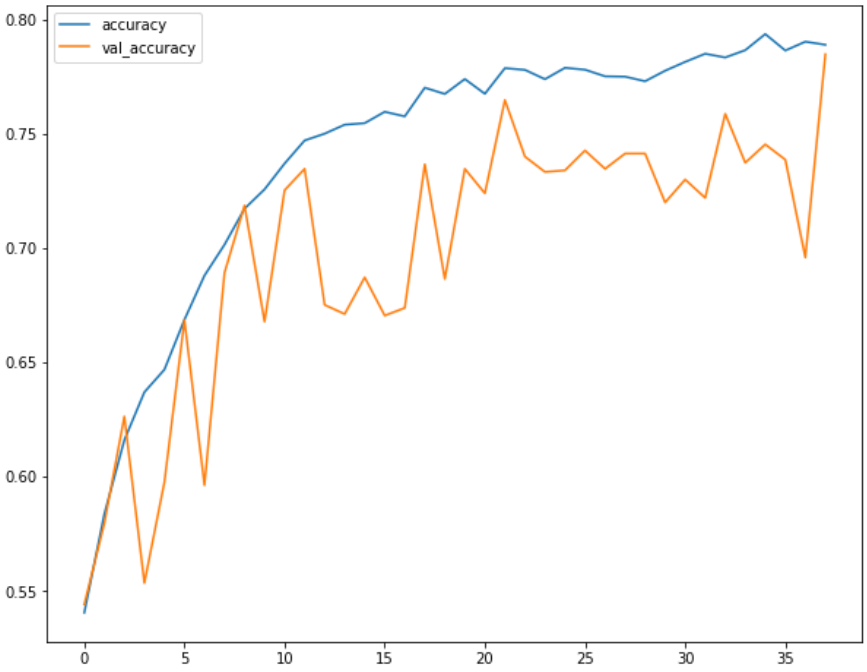
4.4. Evaluacija modela

Nakon što smo model trenirali slijedi evalucija modela. Modeli su testirani na 1496 slika. Nakon toga testiranja pokrenuta je i predikcija 2000 slika koje nisu bile označene. Iz navedenog,

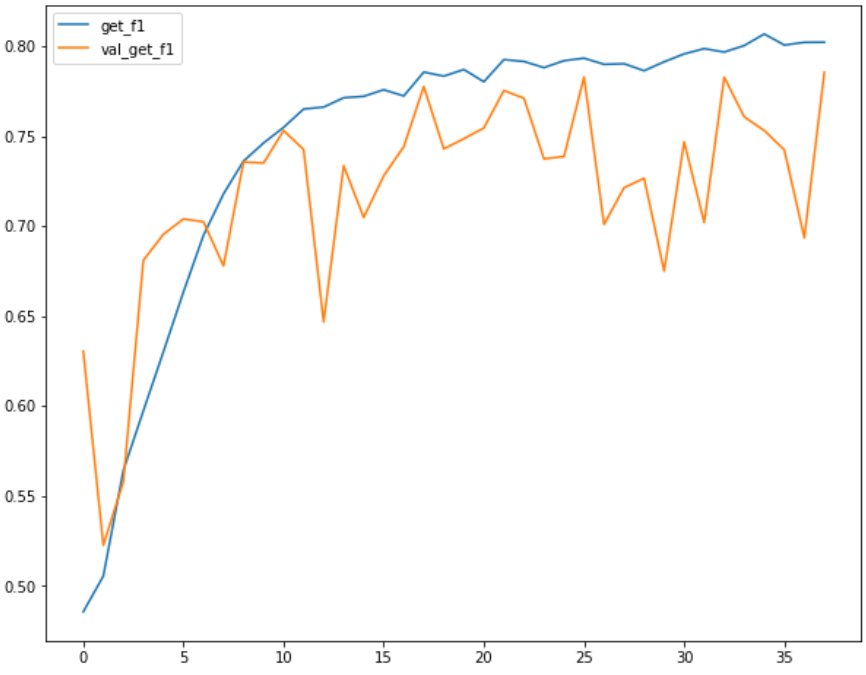
korišteni modeli su ResNet 18, 34, 50 i U net. Pri prvim testiranjima ovih modela bilo je vidljivo da će U Net i ResNet 50 biti brzo odbačeni radi zastarjele tehnologije u laptopu kao što je CPU Intel Core i5 7300HQ i GPU Nvidia GTX 1050Ti. Nakon više obavljenih testova najbolji rezultat je imala ResNet 34 mreža, s stopom učenja od 0.001, veličinom slike 256x256 piksela, optimizatorom „Adam“, funkcijom gubitka *binary_crossentropy*, batch veličinom od 32. Rezultat je bila točnost nad testiranim setom od 77%, a tjek treniranja je prikazan na Slici 4.11, 4,12, 4.13.



Slika 4.11. Tijek funkcije gubitka u odnosu na korake treniranja

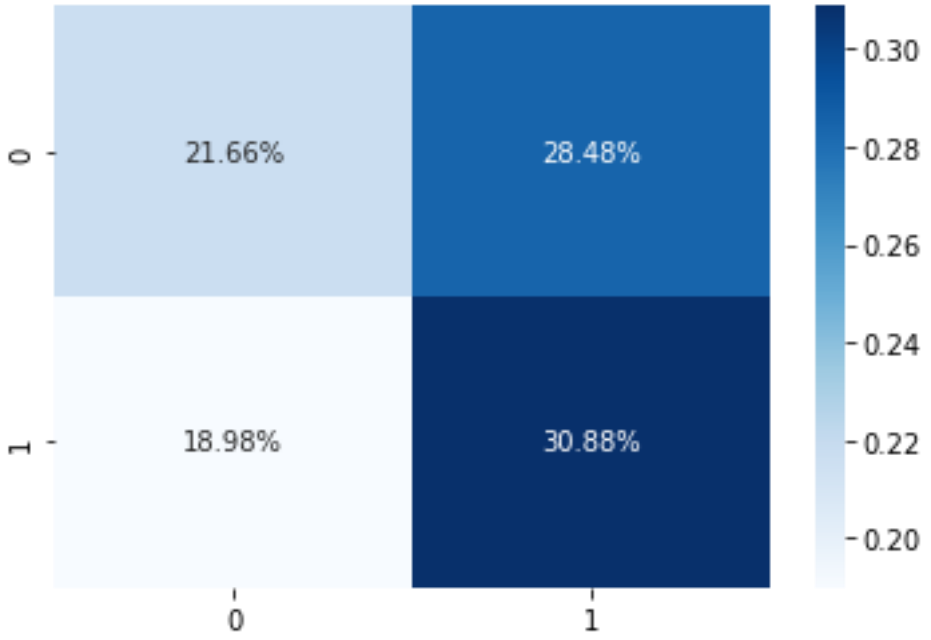


Slika 4.12. Tijek točnosti u odnosu na korake treniranja



Slika 4.13. Tijek funkcije f1 u odnosu na korake treniranja

Navedeni grafovi prikazuju točnost, funkciju gubitka i f1 score. Može se primjetiti da rezultati testnog set imaju veća odstupanja i moguće da dolazi do „overfittinga“. Ovom problem nastaje jer imamo mali broj podataka na kojima testirati mrežu, ali i najvjerovatnije do onoga i ranije spomenutog kao nejasni podaci. Dodatno je moguće vidjeti i na slici 4.14 gdje je prikazana matrica zabune.

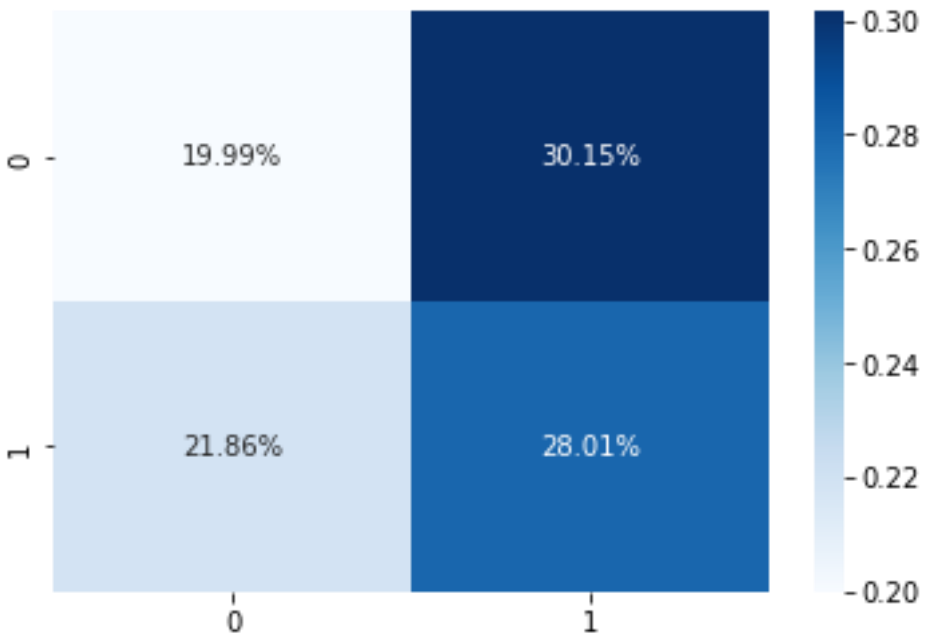


Slika 4.14. Matrica zabune

Iz ove matrice vidljivo je da ima podjednak broj dobro ocjenjenih (mjesta gdje se sjeku isti brojevi) i loše ocjenjenih (mjesta gdje se sjeku različiti brojevi) što ukazuje da model nije savršen te treba dorade.

4.4.1. Rezultati testiranja

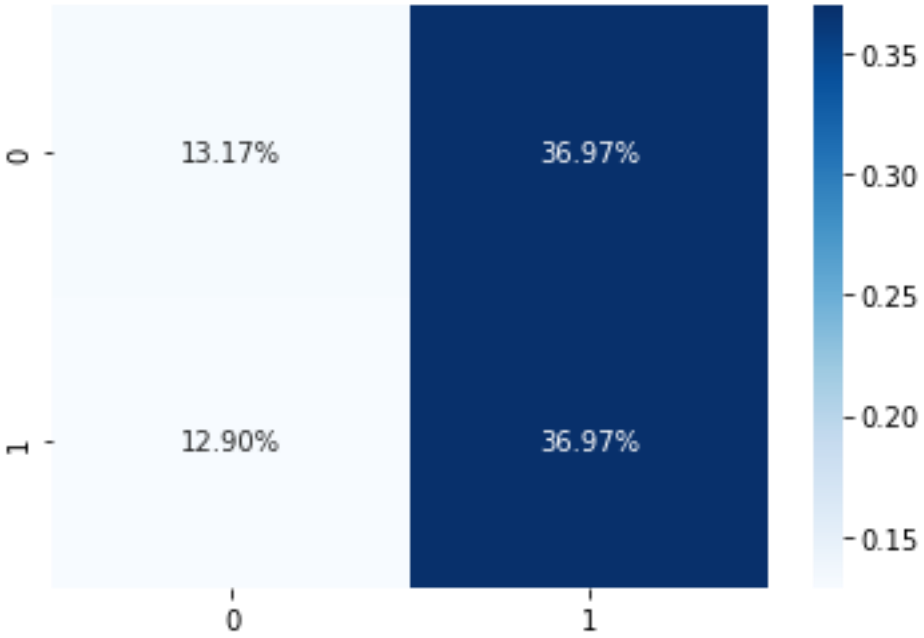
Matrica zabune modela koji je ostavljen da duže ući nad podacima bez ikakvoga nadzora je prikazana na slici 4.15.



Slika 4.15. Matrica zabune ne kontroliranog modela

Ovaj model iako je davao dobre rezultate na treniranim podacima prilikom testrianja je došlo do pada u preformansama jer je došlo do „overfittinga“.

I još na kraju jeda od modela gdje je stopa učenja bila jako velika u iznosu od 0.01 gdje nije došlo do napretka, te njegovi rezultati su vidljivi na slici 4.16.



Slika 4.16. Matrica zabune modela sa stopom učenja 0.01

5. ZAKLJUČAK

U ovome projektnom zadatku cilj je bio proučiti neuronske mreže te napraviti algoritam koji će naučiti konvolucijsku neuronsku mrežu prepoznavanju pukotina. U projektu je određena i obrada slika kako bi se steklo bolje informacije o slikama s kojim radimo. Na testiranim podacima je došlo do uspjeha gdje je dobro izražena pukotina. U dobroj većini slučajeva je mreža točna i ima dobru pretpostavku o čemu se radi na slici. Veliki problem je nastao u pribavljenim podacima gdje ih je dosta bilo nejasno što smo mogli uočiti pri obradi slika i to je doprinijelo lošijem radu mreže. Kako bi se ovaj projekt uspješnije izvršio, potreban je veći skup jasnijih podataka, ali i naprednije računalo koje bi moglo izvršiti bolji model mreže poput U-Neta. Ovaj problem je vrlo značajan za opću sigurnost, te iako nije jako precizan mogao bi se koristiti ako su priloženi kvalitetni podaci.

6. LITERATURA

[1] OpenCV https://opencv.org/about/

[2] Pandas, https://dusanmilosevic.com/uvod-u-python-pandas/

[3] Tensorflow, https://www.tensorflow.org/

[4] NumPy https://numpy.org/about/

[5] Neural Network Full Course https://www.youtube.com/watch?v=ob1yS9g-Zcs

[6] Dropout layer https://machinelearningmastery.com/dropout-regularization-deep-learning-models-keras/

[7] Aktivacijske funkcije https://www.v7labs.com/blog/neural-networks-activation-functions

[8] ResNet https://hhr.wiki/detial/Residual_neural_network

[9] Canny edge detection https://docs.opencv.org/3.4/da/d22/tutorial_py_canny.htm

[10] Gaussian blur https://www.sciencedirect.com/topics/engineering/gaussian-blur