PRIMJENA RAČUNALNOG VIDA U KONTROLI KVALITETE

Živkušić, Damir

Veleučilište u Karlovcu

Odjel Strojarstva

Stručni studij mehatronike

Damir Živkušić

PRIMJENA RAČUNALNOG VIDA U KONTROLI KVALITETE

ZAVRŠNI RAD

Mentor:

mr.sc. Vedran Vyroubal

Karlovac, 2020.

Karlovac University of Applied Sciences
Department of Mechanical Engineering

Professional study of Mechatronics

Damir Živkušić

APPLICATION OF COMPUTER VISION IN QUALITY CONTROL

FINAL PAPER

Mentor:
mr.sc. Vedran Vyroubal

Karlovac, 2020.

VELEUČILIŠTE U KARLOVCU

Trg J.J.Strossmayera 9
HR - 47000, Karlovac, Croatia
Tel. +385 - (0)47 – 843-500
Fax. +385 - (0)47 – 843-503
e-mail: dekanat @ vuka.hr

Klasa:
602-11/18-01/____

Ur.broj:
2133-61-04-18-01

Datum:

ZADATAK ZAVRŠNOG / DIPLOMSKOG RADA

Ime i prezime	Damir Živkušić		
OIB / JMBG			
Adresa			
Tel. / Mob./e-mail			
Matični broj studenta	0112612029		
JMBAG	0248035005		
Studij (staviti znak **X** ispred odgovarajućeg studija)	**X** preddiplomski		specijalistički diplomski
Naziv studija	Stručni studij mehatronike		
Godina upisa	2012		
Datum podnošenja molbe	1.6.2020.		
Vlastoručni potpis studenta/studentice			

.

Naslov teme na hrvatskom:

PRIMJENA RAČUNALNOG VIDA U KONTROLI KVALITETE

Naslov teme na engleskom:

APPLICATION OF COMPUTER VISION IN QUALITY CONTROL

Rad obuhvaća prikaz primjene računalnog vida i umjetne inteligencije za kontrolu kvalitete završnog proizvoda u industriji.
Potrebno je objasniti računalni vid i metode koje se mogu koristiti za automatsku kontrolu proizvoda na industrijskoj liniji.
Koristiti se stručnom literaturom, radnim materijalima, Zakonima i Pravilnicima, ostalom stručnom literaturom i konzultirati se s mentorom.
Završni rad izraditi sukladno Pravilniku VUKA.

Mentor: mr.sc. Vedran Vyroubal	Predsjednik Ispitnog povjerenstva: dr.sc. Adam Stančić

IZJAVA

Izjavljujem da sam svoj rad izradio samostalno pomoću stečenog znanja u Veleučilištu u Karlovcu, stručne literature i interneta te naravno uz pomoć mentora Vedrana Vyroubala.

ZAHVALA

Zahvaljujem mentoru na predloženoj temi, te na pomoći i na strpljenju kod same izrade rada i svim ostalim profesorima na podršci u studiranju te mojim kolegama za dodatnu pomoć kada mi je bila potrebna. Također se zahvaljujem kolegama firme Kontroltest International d.o.o. što su omogućili da se upoznam sa tehnologijama kontrole kvalitete kako bi uspješno odradio ovaj rad. Posebno se zahvaljujem supruzi i obitelji na konstantnoj podršci.

Karlovac, 2020. Damir Živkušić

_____

Sažetak

Cilj ovog rada je prikazati primjenu računalnog vida u procesu automatske kontrole kvalitete završnog proizvoda. Kontrola kvalitete je jako bitan faktor u industriji gdje se teži prema automatizaciji kako bi ubrzali procese te smanjili mogućnost ljudske pogreške. Računalni vid omogućuje prepoznavanje grešaka ili nepravilnosti u visokoj točnosti. U teorijskom dijelu rada opisana je povijest i određene tehnologije računalnog vida, također je opisana primjena računalnog vida u različitim dijelovima industrije. U prvom primjeru eksperimentalnog dijela prikazan je jedan od načina prepoznavanja razlika na dvije slike korištenjem standardnih alata za obradu slike. Umjetna inteligencija za potrebe računalnog vida razvijena je uz pomoć konvolucijskih neuronskih mreža a prikaza je u drugom primjeru eksperimentalnog dijela. Za potrebe ovog rada korišten je programski jezik Python te biblioteke kao što su OpenCV, NumPy i Keras za procesiranje, obradu slika i učenje neuronske mreže.

Ključne riječi: računalni vid, umjetna inteligencija, kontrola kvalitete

Summary

The aim of this paper is to present the application of computer vision in the process of automatic quality control of the final product. Quality control is a very important factor in an industry where automation is sought to speed up processes and reduce the possibility of human error. Computer vision allows the detection of errors or irregularities in high accuracy. The theoretical part of the paper describes the history and certain technologies of computer vision, it also describes the application of computer vision in different parts of the industry. In the first example of the experimental part, one of the ways of recognizing differences in two images using standard processing tools is demonstrated. Artificial intelligence for the needs of computer vision was developed with the help of convolutional neural networks and the presentation is in second example of the experimental part. For the purposes of this paper, the Python programming language and libraries such as OpenCV, Numpy and Keras were used for processing, image manipulation and learning of the neural network.

Keywords: computer vision, artificial intelligence, quality control

Popis slika

Slika 1 Psi graju poker 1
Slika 2 Viola/Jones prepoznavanje lica 4
Slika 3 Primjer neuronske mreže 5
Slika 4 Primjer oblaka točaka 6
Slika 5 Detektiranje pneumonije 8
Slika 6 Primjer AI sigurnosnog sustava 9
Slika 7 Prikaz piksela na slici 12
Slika 8 Pas 12
Slika 9 Slika otoka 13
Slika 10 Pravokutnik sa 3 regije 14
Slika 11 Harris detekcije kuteva 15
Slika 12 Shi-Tomasi detekcija kuteva 15
Slika 13 SIFT 16
Slika 14 ORB 17
Slika 15 Wathershed algoritam 18
Slika 16 Prikaz izvršenja 3. Koraka 20
Slika 17 RGB matrica 21
Slika 18 Grayscale matrica 21
Slika 19 Prikaz pretvorbe slike u nijanse sive 22
Slika 20 Rezultat indeksa strukturne sličnosti 23
Slika 21 Prikaz razlika između dvije slike 23
Slika 22 Binarizacija slike 24
Slika 23 Prikaz rezultata 26
Slika 24 Standarna CNN arhitektura 30
Slika 25 Prikaz ulaznog sloja 32
Slika 26 Filter u CNN 33
Slika 27 Primjer filtera 33
Slika 28 Mapa značajki 34
Slika 29 Impelementacija aktivacijske funkcije 35
Slika 30 Prikaz procesa Pooling 36
Slika 31 Proces poravnavanja (flattening) 36
Slika 32 Prikaz neuronske mreže za klasifikaciju 38
Slika 33Prikaz prilagođavanja neuronske mreže 39
Slika 34 Prikaz razlika u usporedbi 40
Slika 35 Učtavanje baze podataka 42
Slika 36 Učitavanja podataka 43
Slika 37 Rezultat poziva procesnih jedinica 43
Slika 38 Učenje neuronske mreže 45

Sadržaj

1 Uvod ... 1
2 Računalni Vid ... 2
2.1 Povijest računalnog vida ... 2
3 Tehnologije računalnog vida ... 5
3.1 Procesiranje slike ... 5
3.2 Machine Learning (Strojno učenje) ... 5
3.3 Deep Learning (Duboko učenje) ... 5
3.4 Prepoznavanje objekata ... 6
4 Primjena računalnog vida ... 7
4.1 Strojni Vid (Machine Vision) ... 7
4.2 Automobilska industrija ... 7
4.3 Medicina ... 8
4.4 Nadzor ... 8
5 Budućnost računalnog vida ... 9
6 OpenCV ... 10
6.1 Povijest ... 10
7 Upotreba OpenCV-a ... 11
7.1 Čitanje i prikazivanje slika ... 11
7.2 Rotacija slike ... 12
7.3 Otkrivanje značajki ... 13
7.3.1 Harris detekcija kuteva ... 15
7.3.2 Shi-Tomasi detekcija kuteva ... 15
7.3.3 SIFT ... 16
7.4 Segmentacija slike ... 17
8 Eksperimentalni dio ... 18
9 Jupyter Notebook ... 18
10 Primjer 1: Kontrola PCB ploče ... 19
10.1 Korak 1: Učitavanje biblioteka i funkcija ... 19
10.2 Korak 2: Učitavanje slika ... 19
10.3 Korak 3: Prikazivanje referentne slike ... 20
10.4 Korak 4: Pretvaranje slike u nijanse sive boje (Grayscaling) ... 20

10.5 Korak 5: Pronalaženje razlika u slikama 22
10.6 Korak 6: Pronalaženje obrisa 23
10.7 Korak 7: Prikazivanje razlika 25
10.8 Korak 8: Prikazivanje rezultata 26
11 Primjer 2: Umjetna inteligencija za kontrolu odljevaka 27
11.1 Korak 1: Učitavanje biblioteka: 28
11.2 Korak 2: Definiranje konstanti 29
11.3 Korak 3: Pokretanje CNN (Convolutional Neural Network) 29
11.4 Korak 4: Definiranje CNN 31
11.5 Korak 5: Kreiranje duboke neuronske mreže 37
11.6 Korak 6: Optimizacija učenja 39
11.7 Korak 7: Proširivanje baze podataka 41
11.8 Korak 8: Učitavanje podataka 41
11.9 Korak 9: Definiranje načina obrade podataka 43
11.10 Korak 10: Učenje modela 44
12 Zaključak 46
13 Literatura 48

1 Uvod

Računalni vid je trenutno jedno od najatraktivnijih područja istraživanja u informatici, to je područje informatike koje se fokusira na kopiranju kompleksnosti ljudskog vida tako da omogući računalima da procesiraju i identificiraju procese i objekte na isti način kao i ljudi. Istraživači koji se bave razvojem računalnog vida fotografije razmatraju uz procese odvajanja simboličkih informacija iz slikovnih podataka koristeći modele napravljene uz pomoć geometrije, fizike, statistike i teorije učenja.

Kada mi kao ljudi nešto promatramo ili pokušavamo razumjeti neku sliku kao npr. poznato djelo gdje psi igraju poker (slika 1), mi vidimo objekte, pse i prostoriju u kojoj se nalaze. Promatrajući sliku jednostavno možemo zaključiti da se tu radi o psima koji kartaju i uživaju s alkoholom i lulama. Vrlo lako uočimo i kakva je atmosfera u prostoriji te o kojim pasminama se radi. Kako gledamo sve više detalja donosimo više zaključaka i razvijamo neku priču iz te slike.

Slika 1 Psi graju poker

Za računalo ova slika je kao i sve druge slike, samo lista piksela, nekih numeričkih vrijednosti koji predstavljaju nijanse crvene, zelene i plave. Jedan od izazova znanstvenika od 1950-tih je da naprave uređaj koji će razumijevati vizualne informacije kao i ljudi, koji će uočavati detalje i raspoznavati različite elemente na slikama baš kao i ljudi.

2 Računalni Vid

Računalni vid se razvija više od 50 godina ali tek nedavno možemo primijetiti veliku zainteresiranost o tome kako uređaji „vide“ i kako se računalni vid može iskoristiti za izradu proizvoda i gradnju kompanija. Neke od takvih kompanija su Amazon Go, Google Lens, MicroBlink, Snapchat ili proizvodi kao što su autonomna vozila i prepoznavanje lica. Najjednostavnije, računalni vid možemo definirati kao disciplinu unutar umjetne inteligencije koja ima cilj razumijevati piksele ili skupove piksela.

2.1 Povijest računalnog vida

Prvi eksperimenti s računalnim vidom počeli su 1960-tih, a prva komercijalna upotreba je bila 1970. godine sa svrhom da se razlikuje tiskani i ručno pisani tekst. U kasnim 1960-tim godinama računalni vid nastaje na sveučilištima koji su postali pioniri umjetne inteligencije. Cilj je bio imitirati ljudski vizualni sistem, kao kamen temeljac uspostavljanja inteligentnog ponašanja kod robota [1]. Vjerovalo se da se to može ostvariti tako da postavimo kameru na računalo i da ga programiramo da nam objasni što vidi, to je bio cilj natjecanja u projektu „Summer Vision Project“. „Summer Vision Project“ su pokrenuli pioniri umjetne inteligencije Seymour Papert i Marvin Minsky, projekt je trajao 2 mjeseca i sudjelovalo je ukupno 10 ljudi. Zbog korištenja pristupa koji nije učinkovit i tehnoloških ograničenja u to vrijeme projekt je završio neuspješno ali svakako mnogi taj projekt označavaju kao početak računalnog vida kao znanstvenog područja. Istraživanja koja su se vršila početkom 1970-tih postavila su kamen temeljac za različite algoritme koje koristitimo i danas kao izvlačenje rubova sa slike, označavanje linija, modeliranje, prikazivanje objekata kao spoj različitih manjih struktura.

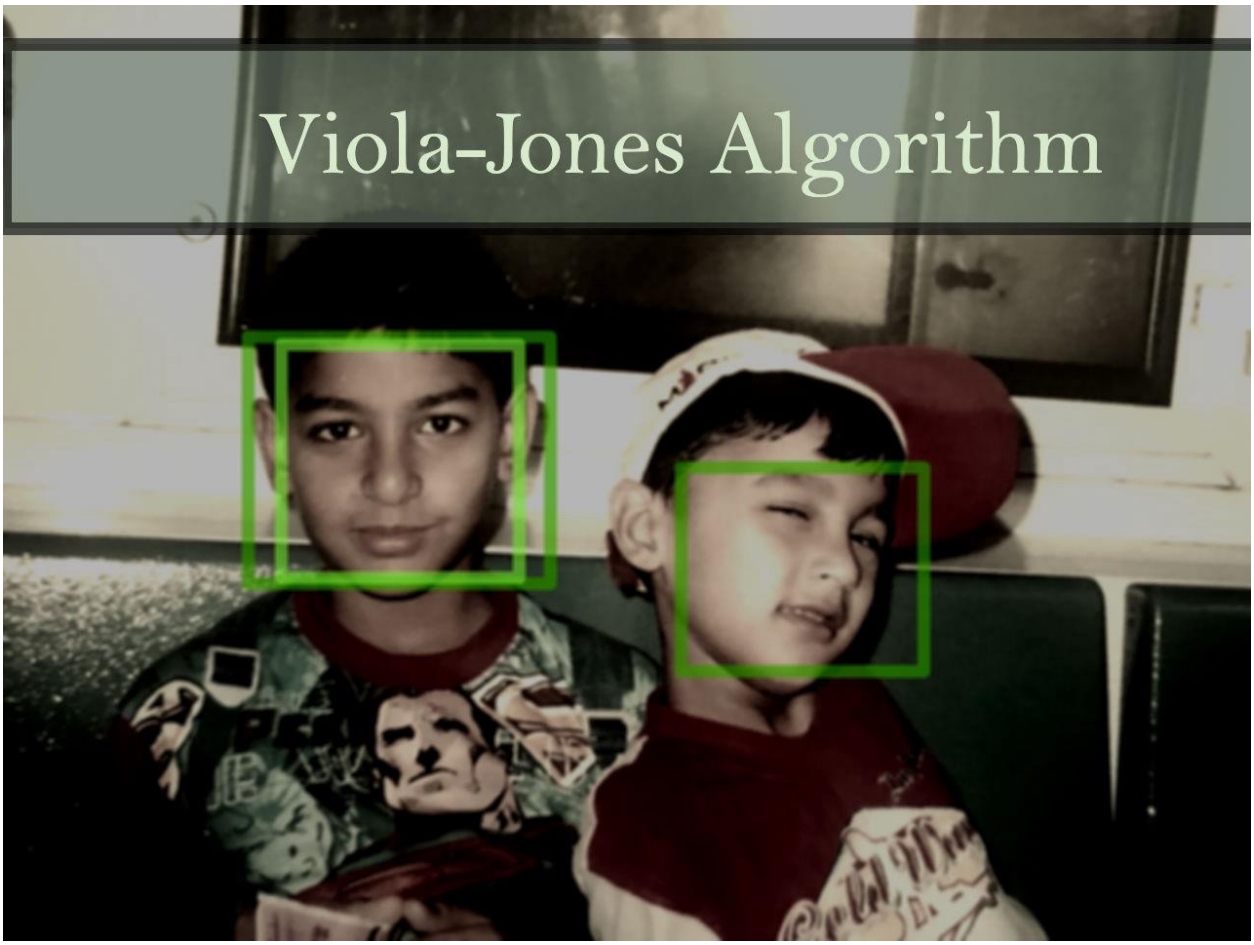
1979. godine japanski znanstvenik Kunihiko Fukushima predložio je korištenje „neocognitron“, metode računalnog vida koja se bazira na neurološkim istraživanjima koja su vršena na ljudskom vizualnom korteksu [2]. Iako metoda koju je Fukushima predložio nije bila uspješna da izvrši bilo

koji kompleksni vizualni zadatak, postavila je temelje za jedno od bitnijih razvoja u računalnom vidu.

Desetljeće kasnije, 1989. mladi francuski znanstvenik Yann LeCun primjenjuje algoritam učenja baziran na stilu povratne propagacije (backpropagation) na arhitekturu Fukushiminine konvolucijske neuronske mreže. Nakon nekoliko godina rada na projektu LeCun je objavio LeNet-5, prvu modernu konvolucijsku mrežu gdje je predstavio neke od bitnih koncepta koje i danas koristimo u računalnom vidu [3]. Konvolucijska mreža se sastoji od nekoliko slojeva umjetnih neurona koji su matematičke komponente koje ugrubo imitiraju biološke neurone. LeCun je iskoristio svoje inovacije kako bi izgradio proizvod koji iščitava poštanske brojeve te je napravio skup podataka ručno pisanih brojeva pod nazivom MNIST – jedan od najpopularnijih skupova podataka u strojnom učenju.

1990-tih računalni vid mijenja svoj fokus s pokušavanja rekonstruiranja objekata u 3D modele na kreiranju modela prepoznavanja objekta prema karakteristikama samog objekta. Najveći utjecaj na promjenu fokusa je imao znanstveni rad autora David Lowe, „Object Recognition from Local Scale-Invariant Features“. U njegovom radu opisan je sustav vizualnog prepoznavanja koji koristi lokalne karakteristike koje su neovisne o rotaciji, položaju i djelomičnim promjenama osvjetljenja. Prema Loweu ovaj način modeliranja je sličan funkcioniranju neurona koji su promatrani u korteksu primata a odgovorni su za detektiranje objekata.

2001. godine je kreiran prvi algoritam za otkrivanje lica koji je djelovao u stvarnom vremenu. Algoritam je razvije od strane Paul Viola i Michael Jones. Viola/Jones detektor lica je i dalje jako korišten. Danas se koristi u društvenim mrežama za otkrivanje lica i postavljanje raznih filtera te automatsko prepoznavanje i označavanje osoba. Također se jako često koristiti u automobilskoj industriji i osiguranju.



Slika 2 Viola/Jones prepoznavanje lica

Gledajući kroz povijest računalnog vida važno je napomenuti da zbog širokog spectra potencijalnih aplikacija čest trend je spajanje računalnog vida s drugim usko povezanim poljima. Tu spadaju obrada slike, fotogrametrija, određivanje poza objekata u 3D-u i računalna grafika. Neki od glavnih algoritama koji su rezultat istraživanja u području računalnog vida danas su dio biblioteka kao što su OpenCV ili Keras.

3 Tehnologije računalnog vida

3.1 Procesiranje slike

Računalna obrada slike je tehnologija koja se koristi u računalnom vidu kako bi prikazali određene podatke ili kako bi olakšali izvršenje određenih funkcija u procesu zadatka računalnog vida. Obrada slike odnosi se na analizu digitalnih slika ili implementaciju algoritma, uključujući klasifikaciju, ekstrakcije, uređivanje ili filtriranje. Obrada slike ističe tehnologije i metodologije koje se koriste za povećavanje informacija slike dok računalni vid ima za cilj izvršenje praktičnih akcija [4].

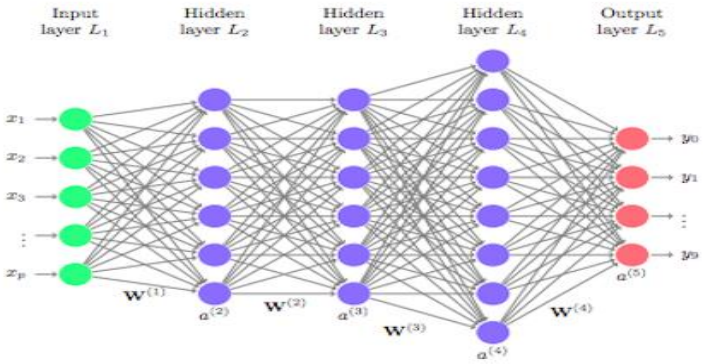
3.2 Machine Learning (Strojno učenje)

Strojno učenje je klasa algoritama koja pruža aplikacijama veću razinu točnosti. Algoritmi strojnog učenja ne moraju imati nužno jasan plan kako će zadatke izvršiti. Na temelju protoka podataka, statistike i napredne analitike oni mogu stalno poboljšavati vrijednosti rezultata. Strojno učenje oslanja se na veliki potencijal skupova podataka. Skup podataka je skup povezanih ulaznih podataka koji se kombiniraju kako bi se postigla bolja točnost [5].

3.3 Deep Learning (Duboko učenje)

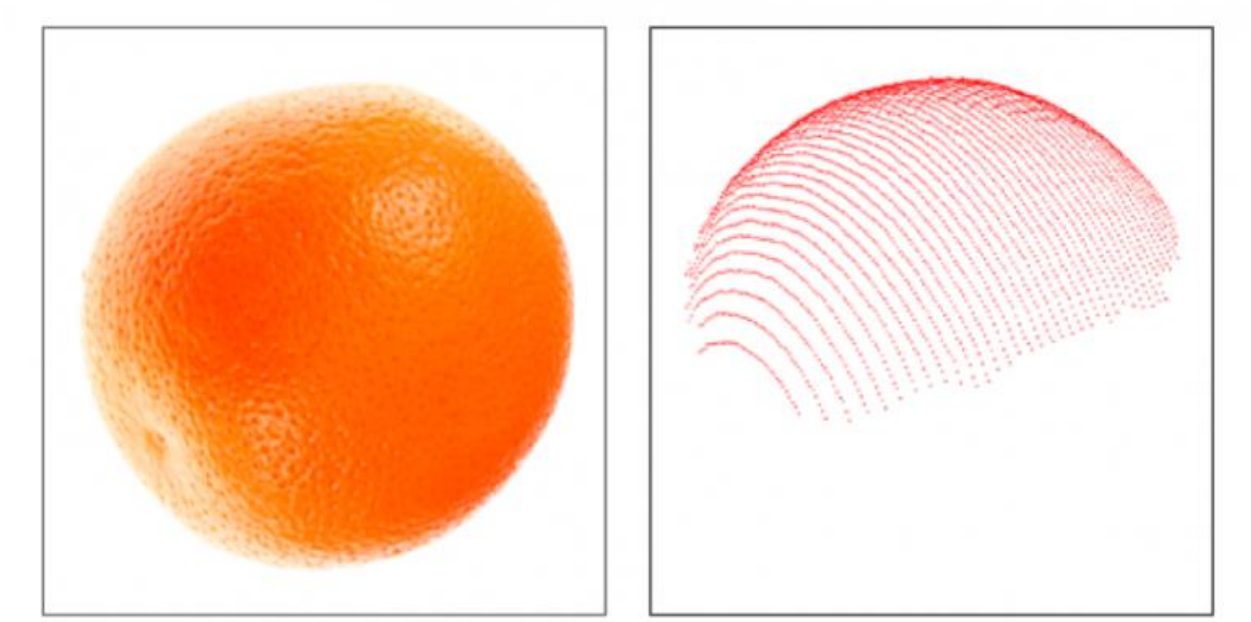
Deep Learning je dio umjetne inteligencije utemeljen na principima ljudskih načina učenja kako bi se postigla bolja razina znanja. Deep Learning pruža mogućnosti poboljšanja procesa, uključujući točnost rezultata računalnog vida. Algoritmi dubokog učenja oslanjaju se na neuronske mreže kako bi sortirali potprocese kao hijerarhiju koncepata [6].

Slika 3 Primjer neuronske mreže



3.4 Prepoznavanje objekata

Tehnologija koja se u posljednje vrijeme sve češće koristi u prepoznavanju i praćenju objekata je oblak točaka. Oblak točaka je zbirka podatkovnih točaka definiranih u trodimenzionalnom koordinatnom sustavu. Ova tehnologija obično se koristi u prostoru gdje su mjesto i oblik svakog objekta predstavljeni popisom koordinata (X,Y,Z). Popis koordinata naziva se oblakom točaka. Ova tehnologija omogućuje precizan prikaz mjesta gdje se neki objekt nalazi u prostoru i svaki se pokret može precizno pratiti [7].



Slika 4 Primjer oblaka točaka

4 Primjena računalnog vida

4.1 Strojni Vid (Machine Vision)

Iako se smatraju istom tehnologijom, računalni vid i strojni vid su različiti termini koji se koriste za tehnologije koje se poklapaju. Računalni vid u širokom pogledu se odnosi na snimanje i automatizaciju analize slike s naglaskom na funkciju analize slike u širokom rasponu teorijskih i praktičnih primjena. Strojni vid tradicionalno se odnosi na korištenje računalnog vida u industrijskoj ili praktičnoj primjeni gdje je potrebno izvršiti određene funkcije ovisno o analizi koja je odrađena od strane vizualnog sustava.

Kao primjer u pogonu za punjenje hrane i pića, vizualni sistem se može koristiti za identificiranje različitih parametara. Može provjeriti dali je prazna boca bez oštećenja i stranih predmeta, može provjeriti razinu ispunjavanja određenog proizvoda i može se koristiti za provjeru dali je ispravna etiketa i dali je naljepnica pravilno postavljena.

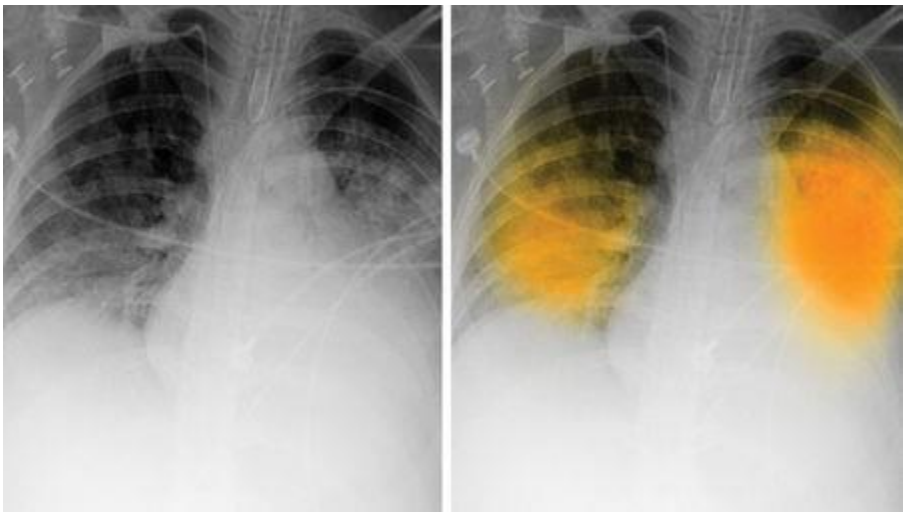
Linije između računalnog i strojnog vida se poklapaju tijekom godina jer nove tehnologije i brže obrade podataka omogućuju korištenje umjetne inteligencije u industrijskim procesima [8].

4.2 Automobilska industrija

Jedan od glavnih čimbenika u prometnim nesrećama je ljudska pogreška i nedostatak pozornosti. Računalni vid u automobilskoj industriji koristi se za rješavanje tih problema i eliminiranje ljudskog faktora. Računalni vid ima zadatak da prepozna pješake s određene udaljenosti te da raspoznaje linije na cesti i znakove pored ceste. Na primjer, recimo da se vozite i da biciklist ispred vas daje pruža ruku pokazujući da ima namjeru ući u vašu traku. Program baziran na računalnom vidu će detektirati pokrete ruke i ustanoviti dali je to bio znak koji upućuje da će biciklist skrenuti ili je to bio slučajan pokret ruke. Ako ustanovi da smo dobili signal namjere biciklista onda će automobil samostalno usporiti i pustiti biciklista da prođe. Računalo unutar automobila koristi duboko učenje kako bi napravio plan i simulirao razne situacije da se istrenira kako reagirati u određenim situacijama.

4.3 Medicina

Računalni vid koristi se za prepoznavanje i dijagnosticiranje stanja i bolesti i za izvršavanje medicinskih intervencija. U zdravstvu je bilo nekih argumenata tko je bolji: računalni vid ili senzori za pametnu zdravstvenu zaštitu. Nema potrebe za međusobno uspoređivanje jer se za postizanje boljih rezultata mora zajedno sa senzorima koristiti računalni vid. Na primjer, Gauss Surgical razvio je rješenje koje prati gubitak krvi u stvarnom vremenu. Senzori detektiraju količinu krvi koja se nalazi na kirurškim spužvama te se ti podaci zatim obrađuju algoritmima strojnog učenja koji određuju koliko je krvi izgubljeno. Tehnologija se trenutno koristi u raznim kirurškim operacijama i porođajima carskim rezom. Također imamo široku upotrebu korištenja računalnog vida za analizu X-ray slika kako bismo automatski odredili o kojoj se bolesti radi (slika 5).



Slika 5 Detektiranje pneumonije

4.4 Nadzor

Računalni vid ima široku primjenu u nadzoru javnih i privatnih mjesta. Najveći kupci tehnologije prepoznavanje lica su vladine agencije koje su zainteresirane za korištenje tehnologije za automatsko prepoznavanje kriminalaca u snimcima sigurnosnih kamera. Ipak, raširena upotreba prepoznavanja lica omogućuje vladama da pomno prate kretanja milijuna građana bili oni osumnjičeni za zločine ili ne, najbolji primjer za to imamo Kinu (slika 6). U Europi i SAD-u stvorio se otpor građana i zaposlenika pojedinih tvrtki. Uz pomoć aktivista za digitalna prava neke države i gradovi u SAD-u su zabranili javnu upotrebu prepoznavanja lica.

Slika 6 Primjer AI sigurnosnog sustava

5 Budućnost računalnog vida

Budući da računalni vid ima sve veći utjecaj na ljudski svijet moramo razmišljati o tome kako će nam ta tehnologija promijeniti život i sam pogled na svijet. Uz dodatna istraživanja i bolju prilagodbu tehnologije računalni vid će u budućnosti obavljati široku lepezu funkcija. Tehnologiju ne samo da će biti lakše istrenirati nego će i biti u mogućnosti obrađivati veću količinu slika nego što je to slučaj danas. Računalni vid će se također sve više koristiti u kombinaciji s drugim podskupovima umjetne inteligencije za izgradnju naprednijih aplikacija. Računalni vid će igrati vitalnu ulogu u razvoju umjetne superinteligencije tako da će omogućiti obradu informacija jednako dobro ili čak i bolje od ljudskog vidnog sustava.

Uređaji za snimanje bit će sve precizniji s više snage za analitiku i obradu slika. Algoritmi za obradu slika bit će slični onima koji se koriste danas, znanstvenici u tom području ne očekuju veće inovacije. Zajednica računalnog vida standardizirat će se na nekoliko arhitektura učenja što će omogućiti platformu za razvoj tržišta.

Budućnost računalnog vida otvorit će put sustavima umjetne inteligencije koji su jednako ljudski kao i mi. Međutim, prije nego što to učinimo treba riješiti nekoliko izazova od kojih je najveća demistifikacija crne kutije umjetne inteligencije. To je zato što je baš kao i kod drugih tehnologija za duboko učenje, računalni vid iako učinkovit još je uvijek nedovoljno jasan kada je u pitanju njegov unutarnji rad.

6 OpenCV

OpenCV je biblioteka funkcija koje se koriste za računalni vid, strojno učenje i obradu slika, a sada igra veliku ulogu u obradi u stvarnom vremenu, što je vrlo važno u današnjim sustavima. Njime se mogu obraditi slike i video za prepoznavanje predmeta, lica ili čak rukopisa čovjeka. Kad se integrira u razne biblioteke, poput Numpy, Python je sposoban obraditi OpenCV strukturu za analizu. Za prepoznavanje uzorka slike i njenih različitih značajki preko OpenCV koristimo vektorski prostor i izvodimo matematičke operacije na tim značajkama.

OpenCV je izdan pod BSD licencom i stoga je besplatan za akademsku i komercijalnu upotrebu. Postoje sučelja C++, C, Python, Java i podržava Windows, Linux, Mac OS, iOS i Android. Kada je dizajniran OpenCV, glavni fokus su bile aplikacije u realnom vremenu za računalnu učinkovitost. Kompletna arhitektura je napisana na optimiziranom C / C++ programskom jeziku kako bi se iskoristila prednost multi-core obrade [9].

6.1 Povijest

OpenCV je nastao ranih 2000-tih godina od strane inženjera računalnog vida Gary Bradsky. Bradsky je radio u Intelu i skupa sa timom inženjera, uglavnom iz Rusije razvili su prvu verziju OpenCV-a interno za Intel, nakon toga su objavili verziju V0.9 kao javno dostupnu 2002. godine. Bradsky je prešao u kompaniju Willow Garage, skupa sa ostalim osnivačima OpenCV-a Viktor Eurkhimov, Sergey Molinov, Alexander Shishkov i Vadim Pisarevsky (koji je osnovao kompaniju ItSeez, koja je kupljena od strane Intela 2016.) te su nastavili održavati biblioteku kao javno dostupan projekt.

Verzija 0.9 pretežito se oslanjala na C API (application programming interface) i već tada je OpenCV podržavao funkcije obrade podataka slike kao što su pristup pikselima, procesiranje slike, filtriranje, analiza geometrije i oblika, osnovne funkcije strojnog učenja. Mnoge od ovih funkcija su ostale uključene godinama a i neke se i danas koriste. Verzija 1.0 je izašla 2006 godine i označila je OpenCV kao dominantnu biblioteku za računalni vid. Bradsky i Adrian Kaehler 2008 godine objavili su knjigu „Learning OpenCV“ koja je postala svjetski hit i godinama služila kao vodilja za OpenCV API. OpenCV je postao veoma popularan u za akademske i industrijske aplikacije, pogotovo u domeni robotike iako je verzija 1.0 bila tek mali pomak od verzije 0.9. Nakon

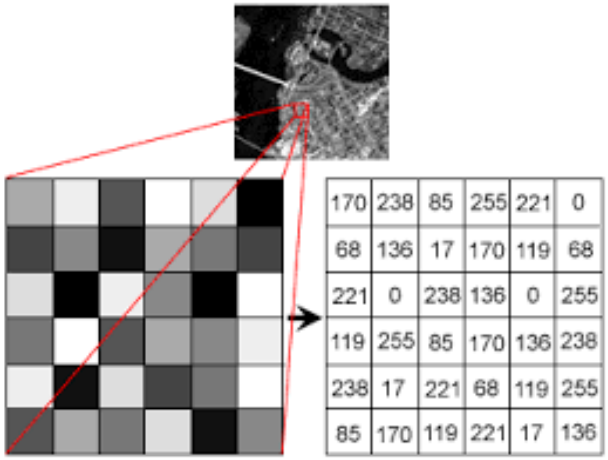
objavljivanja verzije 1.0 OpenCV je ušao u hibernaciju razvoja jer je tim radio na razvoju drugih projekata a javna zajednica još uvijek nije bila dobro formirana.Verzija 2.x je izašla 2009 i bila je korištena do 2015 godine. Verzija 2.x jer tijekom godina imala raznih podverzija a najznačajnije promjene bile su početak uporabe C++ API i donošenje koncepta modula koji se mogu graditi i povezivati odvojeno. Verzija 3.0 je 2015 godine bila u beta razvoju i dobro je prihvaćena od strane zajednice. Oni su tražili stabilniji API i bolju podršku obrade podataka preko procesora grafičke jedinice GPU. Također verzija 3.0 je imala bolju podršku za strojno učenje, upotrebu na Android aplikacijama i optimizaciju rada na mobilnim procesorima. Verzija 4.0 je izašla 2018 godine i danas predstavlja OpenCV koji je jedan od bitnijih javno dostupnih projekata za računalni vid. Stari C API je u potpunosti izbačen te se koristi C++11. Verzija 4.0 prati optimizaciju GPU i CPU a uveden je i GraphAPI (G-API) modul. G-API omogućuje heterogenu obradu podataka s GPU i CPU što omogućava korištenje moćnih alata kao što su TensorFlow biblioteka za duboko učenje i PyTorch razvijen od strane Facebook-a. Open CV je postavljen kao projekt s velikom podrškom javne zajednice i kao glavna biblioteka računalnog vida zahvaljujući stalnim razvojem u strojno i duboko učenje i integraciju s različitim programskim jezicima kao što je Python i ostali. [10]

7 Upotreba OpenCV-a

7.1 Čitanje i prikazivanje slika

Računala obrađuju podatke uz pomoć brojeva pa tako i slike. Slike pretvaramo u brojeve preko vrijednosti svakog piksela. Svaki broj prikazuje intenzitet određenog piksela na određenoj lokaciji. Slika 7 prikazuje vrijednosti piksela određenog područja gdje svaki piksel ima jedan broj koji predstavlja intenzitet crne boje na toj lokaciji. Slike koje su u boji će imati više od jedne vrijednosti za određeni piksel što prikazujemo u trodimenzionalnoj matrici. Ove vrijednosti prikazuju intenzitet određenog kanala – crvenog, zelenog, plavog kanala na RBG slici.

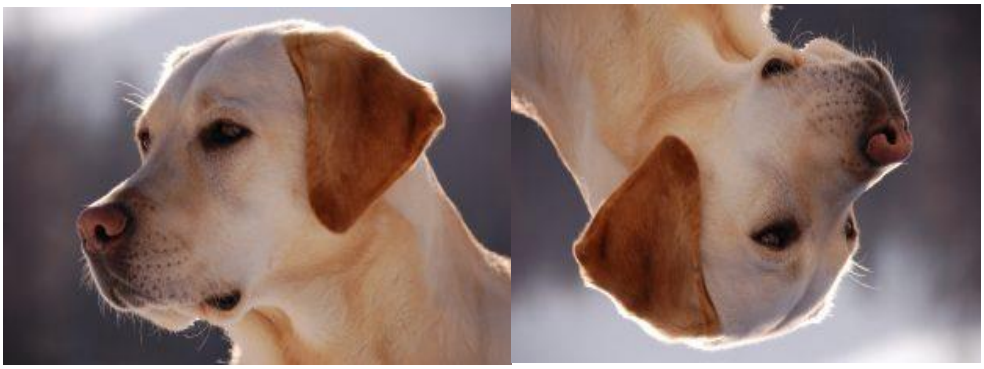
Očitavanje i razumijevanje slike je jako važno za računalni vid i to je ono što nam OpenCV olakšava.



Slika 7 Prikaz piksela na slici

7.2 Rotacija slike

Kako bi trenirali model za duboko učenje potrebna nam je velika količina podataka ali to nekada nije moguće jer nismo u stanju prikupiti i označiti slike koje su nam potrebne. U slučaju da radimo model klasifikacije da identificiramo životinju, u ovom slučaju psa (slika 8) obje slike moraju biti klasificirane kao pas.



Slika 8 Pas

Model može imati poteškoće da prepozna drugu sliku ako nije treniran na takvim slikama. U ovim slučajevima koristimo tehnike poput „data augmentation“ (povećanje podataka).

Ova metoda nam omogućuje stvaranje više uzoraka za obuku našeg modela dubokog učenja. Povećavanje podataka koristi dostupne uzorke podataka za proizvodnju novih primjenom operacija kao što su rotacija, povećanje, pomak itd. To čini naš model otpornijim na promjene i

dovodi do bolje generalizacije. OpenCV sadrži jednostavne funkcije kao je warpAffine() gdje definiramo oko koje osi ćemo raditi rotaciju.

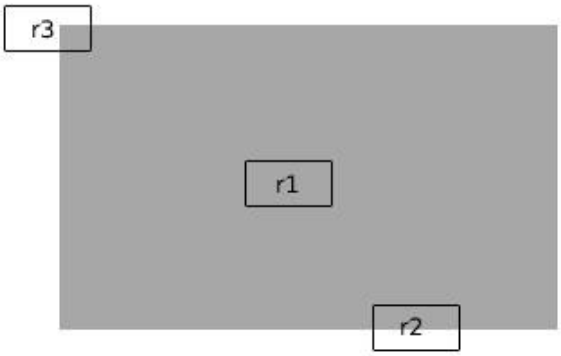
7.3 Otkrivanje značajki

Kada pogledamo sliku otoka (slika 9), naš mozak automatski registrira detalje koji se nalaze u središnjem ili desnom dijelu slike, dok lijevi dio ostane zanemaren. To je zbog toga što imamo veće promjene intenziteta više na tim dijelovima slike. Slično je i kod rješavanja problema u računalnom vidu, računalo mora prepoznati važne aspekte slike.

Slika 9 Slika otoka

Značajke su matematički prikazi ključnih područja sa slike. Značajke su vektorske reprezentacije vizualnog sadržaja sa slike kako bismo na njima mogli obavljati matematičke operacije.
Značajke slike igraju važnu ulogu u računalnom vidu za razne aplikacije kao što su otkrivanje objekata, segmentaciju, poravnanje slike [11].

Značajke mogu uključivati rubove, kuteve ili dijelove slike. Ako razmotrimo pravokutnik (slika 10) imamo tri regije r1, r2, r3.



Slika 10 Pravokutnik s 3 regije

Regija r1 je regija s jednakom površinom, okolinom i intenzitetom u pravokutniku, r2 je regija koja sadrži jedan od rubova pravokutnika do r3 sadrži jedan od kutova. R1 i r2 ne predstavljaju zanimljive značajke jer je mala vjerojatnost pronalaženja točnog podudaranja jer u pravokutniku postoje i druge slične regije. Regija r3 nam je zanimljiva jer predstavlja kut s istaknutom pozicijom i promjenom intenziteta. Budući da su kutevi zanimljive značajke, algoritmi za otkrivanje značajki su prvo krenuli s kutevima. U OpenCV-u postoji više tehnika za otkrivanje značajki:

- Harris detekcija kuteva
- Shi-Tomasi detekcija kuteva
- SIFT (Scale-Invariant Feature Transform)
- SRUF (Speeded-Up Robust Features)
- FAST detekcija kuteva
- ORB (Oriented FAST and Rotated Brief)

7.3.1 Harris detekcija kuteva

Pruža dobru ponovljivost pri promjeni osvjetljenja i rotacije, pa se stoga češće koristi u stereo podudaranju i pronalaženju slika iz baze podataka. Iako postoje nedostaci i ograničenja, Harris kutni detektor još uvijek je važna i temeljna tehnika za mnoge aplikacije računalnog vida (slika 11).

Slika 11 Harris detekcije kuteva

7.3.2 Shi-Tomasi detekcija kuteva

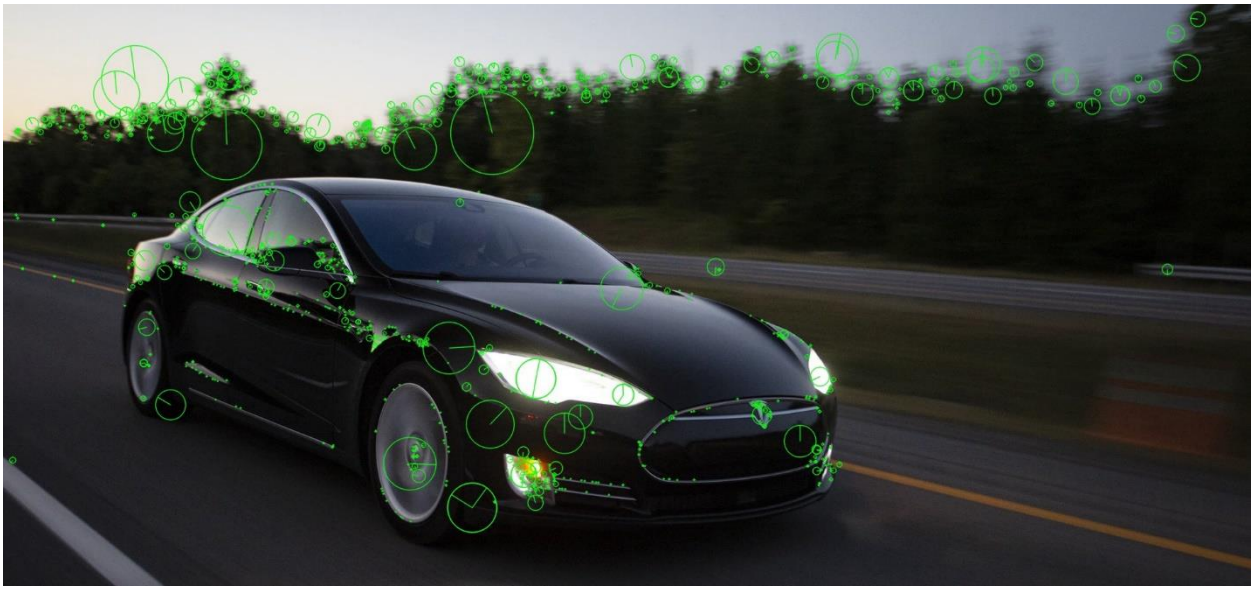
Shi i Tomasi smislili su drugačiju funkciju bodovanja od one koja se koristi u Harisovom detektoru uglova kako bi pronašli broj najjačih kutova na slici (slika 12).

Slika 12 Shi-Tomasi detekcija kuteva

Gornje dvije tehnike Haris Corner i Shi-Tomasi su rotacijske invarijantne, što znači da čak i ako su kutovi rotirani moći ćemo ih točno detektirati. Međutim oni su razmjerno promjenjivi, ako se kutevi povećaju izgubit ćemo oblik u odabranom području i detektori ih neće moći prepoznati.

7.3.3 SIFT

SIFT je neovisan o rotaciji i povećanju. SIFT pruža ključne točke i njihove opisnike gdje svaki opisuje svoju ključnu točku na odabranom mjerilu i rotaciji. Slika 13 prikazuje krugove koji opisuju ključne točke/značajke gdje veličina kruga predstavlja snagu ključne točke a linija unutar kruga označava njenu rotaciju.



Slika 13 SIFT

SURF je predstavljen kako bi omogućio brže vrijeme procesiranja sa svim prednostima SIFT-a. iako je SURF brz u usporedbi sa SIFT-om, nije tako brz da ga možete koristiti s uređajima u stvarnom vremenu poput mobilnih telefona. Tako je uveden algoritam FAST sa skraćenim vremenom obrade. Međutim, FAST nam daje samo ključne točke te trebamo izračunati deskriptore s drugim algoritmima poput SIFT i SURF.

ORB (slika 14) je učinkovita alternativa za SIFT i SURF. Iako izračunava manje ključnih bodova u usporedbi s SIFT i SURF, oni su ipak učinkoviti. Koristi FAST i BRIEF tehnike za otkrivanje ključnih točaka i izračunavanje deskriptora slike.

Slika 14 ORB

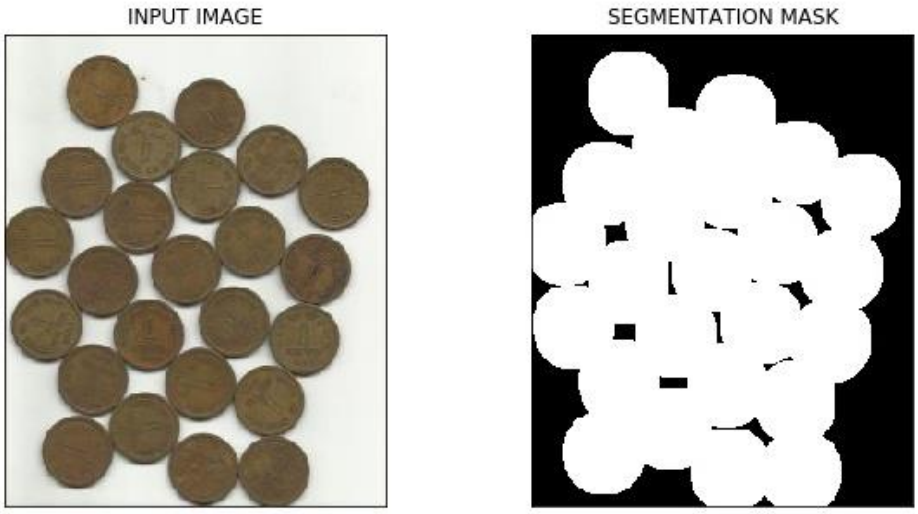
7.4 Segmentacija slike

U računalnom vidu izraz "segmentacija slike" ili jednostavno "segmentacija" odnosi se na podjelu slike u grupe piksela na temelju nekih kriterija.

Algoritam segmentacije uzima sliku kao ulazni podatak i kao rezultat daje kolekciju regija (ili segmenata) koji se mogu predstaviti kao:

- Zbirka kontura
- Maska u kojoj je svakom segmentu dodijeljena jedinstvena vrijednost sive boje ili boje kako bi je identificirao

Algoritam „Watershed“ (slika 15) je klasičan algoritam segmentacije slike. Vrijednosti piksela na slici smatra topografijom. Za pronalaženje granica objekta uzima početne markere kao ulaz. Tada algoritam započinje prelijevanje bazena s markera dok se markeri ne susretnu na granicama objekta.



Slika 15 Watershed algoritam

8 Eksperimentalni dio

U ovom dijelu rada bit će prikazana stvarna primjena računalnog vida u usporedbi dvije slike, pronalaženjem razlika između slika te automatsku klasifikaciju slike na temelju umjetne inteligencije. Eksperimentalni dio je podijeljen na dva primjera. U prvom primjeru koristimo OpenCV kako bi usporedili dvije slike i prikazali razlike na slikama. U drugom dijelu bit će prikazano stvaranje i učenje neuronske mreže kako bi klasificirali dali je završni proizvod dobar ili loš. Za oba primjera korištena je aplikacija Jupyter Notebook kako bi izvršili sve korake koji su potrebni u programskom kodu..

9 Jupyter Notebook

Jupyter Notebook je web-aplikacija otvorenog koda koja omogućuje stvaranje i razmjenu dokumenata koji sadrže kôd, jednadžbe, vizualizacije i narativni tekst. Ima široku primjenu a neke od mogućnosti su: čišćenje i transformacija podataka, numerička simulacija, statističko modeliranje, vizualizacija podataka, strojno učenje i još mnogo toga [13].

10 Primjer 1: Kontrola PCB ploče

Računalni vid ima široku primjenu u elektroničkoj industriji gdje je potrebna visoka preciznost proizvodnje koja se vrši u velikim serijama te je shodno tome potrebna i brza kontrola kvalitete proizvodnje i završnog proizvoda. Jako je česta primjena računalnog vida u kontroli postavljanja elektroničkih komponenti na PCB ploče kako bi utvrdili dali su sve komponente postavljene pravilno na odgovarajuće mjesto. Za svrhu ovog rada koristi će se OpenCV biblioteka u kombinaciji sa Scikit-Image bibliotekom funkcija za analizu slike i prikaz greške. Funkcija koja će izvršiti usporedbu između referentne slike i slike koju kontroliramo je compare_ssim koja je dio Scikit-Image biblioteke. Programski kod se odvija u 8 koraka gdje svaki korak izvršava određeni zadatak.

10.1 Korak 1: Učitavanje biblioteka i funkcija

```
from skimage.measure import compare_ssim
import imutils
import cv2
```

U prvom koraku moramo učitati biblioteke funkcija koje ćemo koristiti u našem programskom zadatku. U prvoj liniji koda učitavamo funkciju **compare_ssim** koja se nalazi u biblioteci Scikit-image. Skraćeni naziv biblioteke je **„skimage“,** nakon toga definiramo da tražimo funkciju **„compare_ssim“** koja se nalazi u modulu **„measure“**.

Učitavamo **„imutils“** skup funkcija koje nam omogućuju razne mogućnosti manipulacijom slike kao što su rotacija, translacija, prikazivanje rubova itd.. Također učitavamo i **„cv2“** što je skraćeni naziv biblioteke OpenCV.

10.2 Korak 2: Učitavanje slika

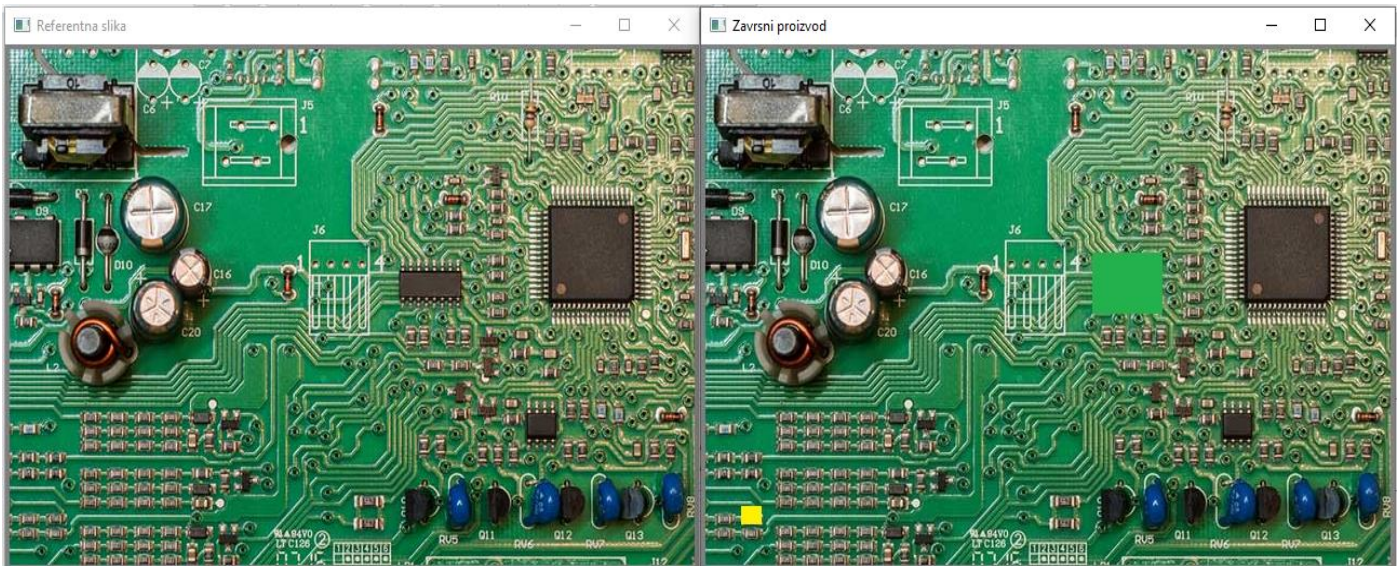
```
imageA = cv2.imread('./images/pcb_org.jpg')
imageB = cv2.imread('./images/pcb_mod.jpg')
```

U drugom koraku izvršavamo učitavanje referentne slike **„ImageA“** i slike koju ćemo kontrolirati prema referentnoj slici **„ImageB“**. Funkcija **„cv2.imread“** je dio OpenCV biblioteke i omogućava nam učitavanje slike i spremanje slike u memoriju za sljedeće operacije.

10.3 Korak 3: Prikazivanje referentne slike

cv2.imshow('Referentna slika', imageA)
cv2.imshow('Završni proizvod', imageB)
cv2.waitKey()
cv2.destroyAllWindows()

Treći korak nam daje prikaz referentne slike i slike završnog proizvoda na zaslon kako bi bili sigurni da smo očitali odgovarajuću sliku i kako bi bez greške mogli izvršiti sljedeće korake analize. **„cv2.imshow“** funkcija nam izvršava prikazivanje slika na zaslon, **„cv2.waitKey()“** sluša naredbu s tipkovnice što je u ovom slučaju pritisak bilo koje tipke dok ***cv2.destroyAllWindows()*** gasi sve prozore sa zaslona kada pritisnemo bilo koju tipku koju je prethodna funkcija memorirala.



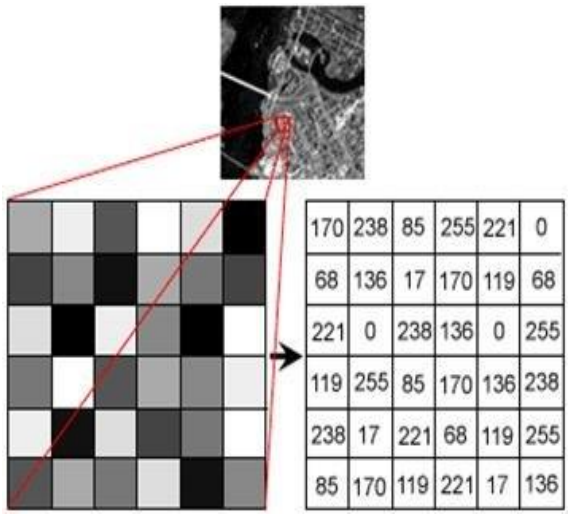
Slika 16 Prikaz izvršenja 3. Koraka

10.4 Korak 4: Pretvaranje slike u nijanse sive boje (Grayscaling)

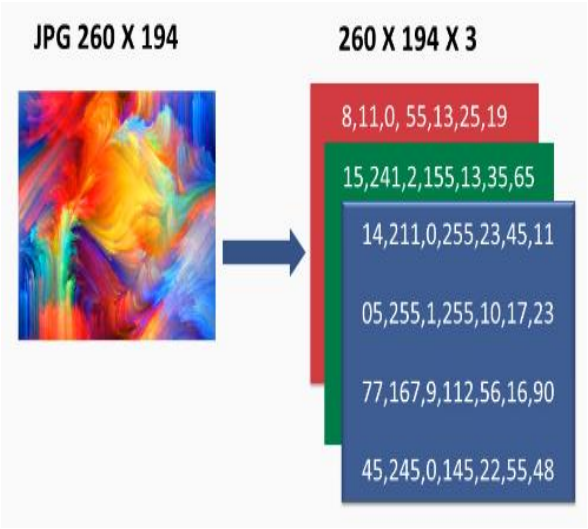
grayA = cv2.cvtColor(imageA, cv2.COLOR_BGR2GRAY)
grayB = cv2.cvtColor(imageB, cv2.COLOR_BGR2GRAY)
cv2.imshow('Gray ImageA', grayA)
cv2.imshow('Gray ImageB', grayB)
cv2.waitKey()
cv2.destroyAllWindows()

Korak 4 je ključni korak pripreme slike za analizu. U ovom koraku izvršavamo pretvaranje naših slika u boji u slike kojima su svi pikseli različitog intenziteta nijanse sive boje. Ovaj postupak nazivamo **Grayscaling.**

Razlog zašto je postupak pretvaranja slike u nijanse sive boje bitan je taj što u svakom pikselu moramo imati što manje ali kvalitetnijih informacija za daljnje procesiranje slike. Siva boja je boja u kojoj sve crvene, zelene i plave komponente imaju jednak intenzitet u RGB prostoru pa je potrebno odrediti samo jednu vrijednost intenziteta za svaki piksel. Kada radimo sa slikama koje su prošle postupak „grayscaling-a“, tj. koje imaju samo jednu vrijednost intenziteta onda se sve potrebne matematičke operacije izvršavaju u jednodimenzionalnoj matrici dok za slike u RGB nijansi sve matematičke operacije izvršavaju se u 3-dimenzionalnoj matrici što je zahtjevniji i dugotrajniji proces.



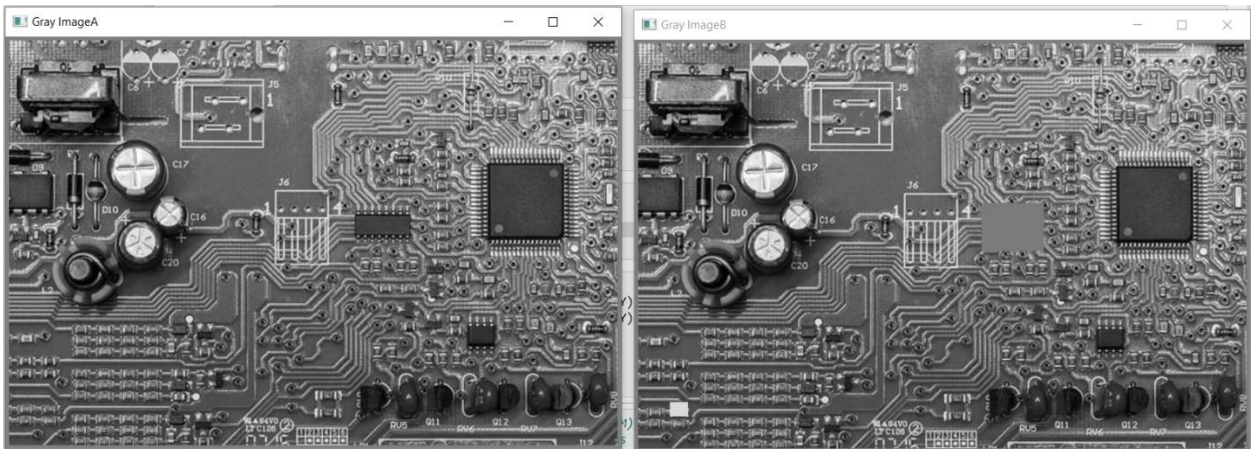
Slika 18 Grayscale matrica



Slika 17 RGB matrica

Za postupak pretvaranja slike u nijanse sive boje koristimo metodu **“cv2.cvtColor()”** koja je dio OpenCV biblioteke. **Cv2.cvtColor()** metodu koristimo za pretvorbu slike iz jednog spektra boja u drugi. Postoji više od 150 spektara boja u OpenCV biblioteci. Za našu svrhu koristimo parametar **“*cv2.COLOR_BGR2GRAY*“** koji nam omogućuje da sve piksele u slici pretvorimo u

nijanse sive boje tako da im na osnovu postojeće vrijednost RBG dodijeli jednaku vrijednost u domeni sive boje, 0-255.



Slika 19 Prikaz pretvorbe slike u nijanse sive

10.5 Korak 5: Pronalaženje razlika u slikama

```
(score, diff) = compare_ssim(grayA, grayB, full=True)
diff = (diff * 255).astype("uint8")
print("SSIM: {}".format(score))
```

Kako bismo ustanovili dali neke od elektroničkih komponenti nedostaju s PCB ploče potrebno je detektirati dali postoji razlika između referentne slike i slike završnog proizvoda. Kako bi detektirali razlike u ovom projektu koristimo metodu **„compare_ssim“**.

SSIM (Structural Similarity Index) u prijevodu Indeks Strukturne Sličnosti je metoda koja kvantificira gubitak kvalitete slike. Gubitak kvalitete nastaje obradom slike poput kompresije podataka ili gubitka podataka u prijenosu. Ova metoda zahtjeva dvije slike koje su napravljene iz istog kadra, iste veličine. Druga slika općenito je komprimirana ili na određenim mjestima sadrži drugačije informacije kao primjerice intenzitet piksela. SSIM zapravo mjeri percepcijsku razliku između dvije slične slike. SSIM ne može procijeniti koja je slika bolja, nego je potrebno znati koja je slika original ili koju sliku koristimo kao referentnu sliku.
Metoda **compare_ssim** kao rezultat daje dva parametra **score** i **diff.**

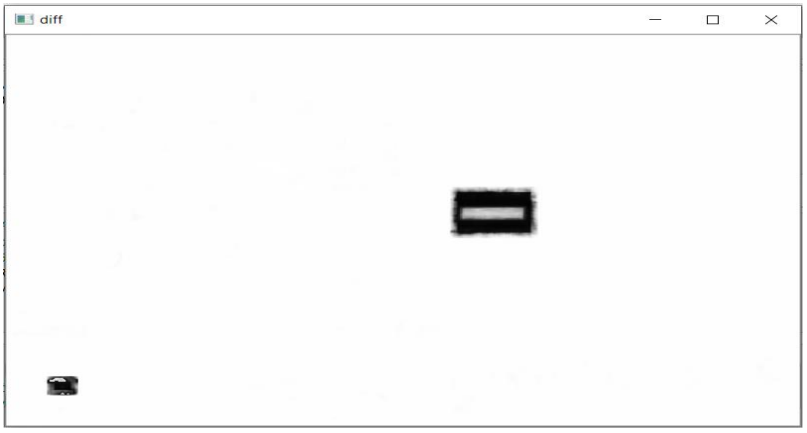
Parametar **score** je index strukturne sličnosti između dvije slike i može biti u rasponu od -1 do 1. U našem slučaju indeks strukturne sličnosti je 0.9881410968740406 (slika 20). Parametar **score** zapravo pokazuje kolika je sličnost između dvije slike, ako su slike identične onda je rezultat 1.

```
(score, diff) = compare_ssim(grayA, grayB, full=True)
diff = (diff * 255).astype("uint8")
print("SSIM: {}".format(score))
```

```
SSIM: 0.9881410968740406
```

Slika 20 Rezultat indeksa strukturne sličnosti

Parametar diff sadrži informacije o razlikama koje su pronađene između dvije slike (slika 21). Pošto su informacije spremljene matrici koja sadrži vrijednosti između 0 i 1 potrebno ih je pretvoriti u vrijednosti 0-255 kako bi ih mogu koristiti u sljedećim operacijama. Pretvorbu radimo da način da sve vrijednosti pomnožimo s 255 i niz svih brojeva pretvorimo u 8-bitne cijele brojeve.



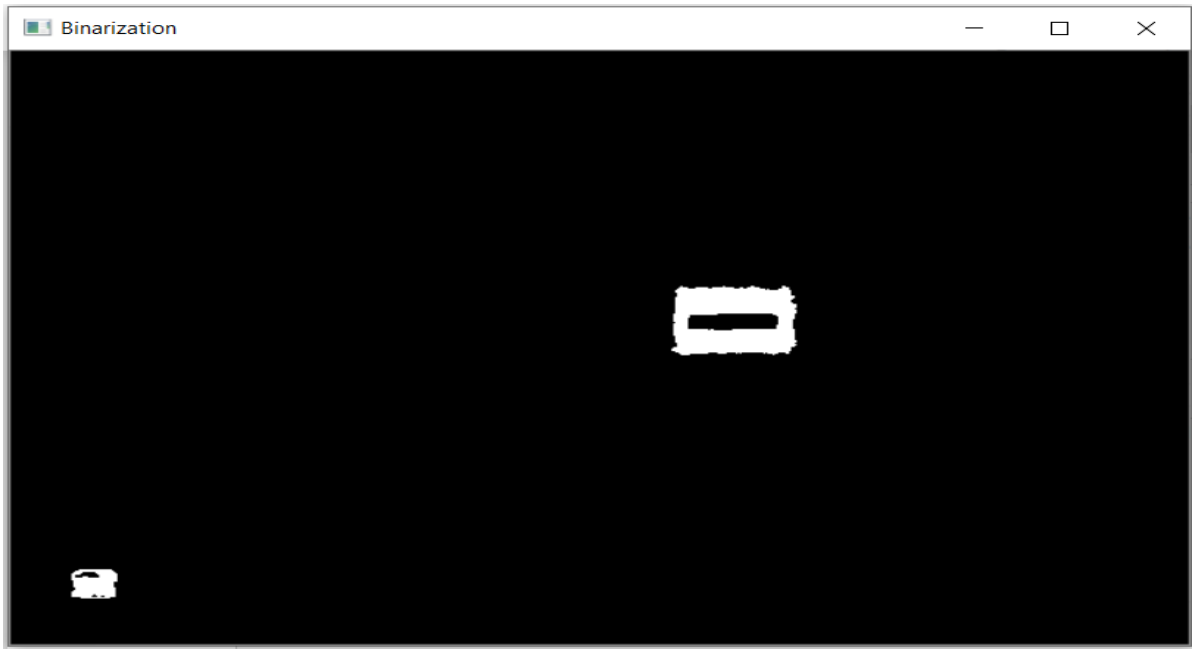
Slika 21 Prikaz razlika između dvije slike

10.6 Korak 6: Pronalaženje obrisa

```
thresh = cv2.threshold(diff, 0, 255,
        cv2.THRESH_BINARY_INV | cv2.THRESH_OTSU)[1]
cnts = cv2.findContours(thresh, cv2.RETR_EXTERNAL,
        cv2.CHAIN_APPROX_SIMPLE)
cnts = imutils.grab_contours(cnts)
```

Da bismo pravilo prikazali gdje se nalaze razlike na slikama potrebno je pronaći obrise objekata koji se razlikuju. Za pronalazak obrisa koristimo metodu **cv2.findContours()** iz biblioteke OpenCV. Obrise možemo definirati kao krivulje koje spajaju sve kontinuirane točke (duž neke granice) koje su istog intenziteta ili boje. U OpenCv biblioteci pronalaženje obrisa izjednačuje se kao traženje bijelih objekata na crnoj podlozi. Kako bismo pravilno koristili **cv2.findContours** metodu slika mora biti binarna (crno-bijela) gdje su vrijednosti 0 ili 255.

Metoda **cv2.threshold()** nam omogućuje da tražene razlike pretvorimo u objekte bijele boje vrijednosti 255 a pozadinu da pretvorimo u crnu boju vrijednosti 0. Kako bismo dobili pravilnu sliku moramo definirati određene parametre kao što je slika koju na kojoj radimo operaciju (diff), raspon vrijednosti u kojem želimo raditi (0, 255), parametri koji određuju način binarizacije (cv2.THRESH_BINARY_INV) i vrijednost koja će biti granica pretvorbe binarizacije (cv2.THRESH_OTSU).



Slika 22 Binarizacija slike

U trenutku kada imamo pravilo definiranu sliku gdje je objekt kojeg tražimo bijele boje na crnoj pozadini možemo krenuti s pronalaženjem kontura. Metoda **cv2.findContours()** zahtjeva 3 parametra. Za detektiranje obrisa moramo definirati na kojoj slici izvodimo operaciju (**thresh**), koje obrise želimo prikazati (**cv2.RETR_EXTERNAL**) i kako želimo prikazati obrise (**cv2.CHAIN_APPROX_SIMPLE**). Kada pronalazimo obrise OpenCv može kao obrise

označiti sve granice između crne i bijele boje, što nama u ovom slučaju nije potrebno. Parametrom **cv2.RETR_EXTERNAL** definiramo spremanje u memoriju koordinate onih obrisa koji su na vanjskim granicama. Pošto su obrisi krivulje one mogu imati razne oblike i razne točke spajanja. U slučaju kada koristimo ravne linije nije nam potrebno koristiti sve točke obrisa nego uz korištenje parametra **cv2.CHAIN_APPROX_SIMPLE** u memoriju spremamo samo točke koje su nam najpotrebnije za povezivanje linija.

10.7 Korak 7: Prikazivanje razlika

U prethodnim koracima smo detektirali razlike između referentne slike i slike završnog proizvoda, nakon toga smo pronašli gdje se razlike nalaze i definirali gdje su njihovi obrisi. U ovom koraku ćemo prikazati razlike kako bismo na obje slike točno vidjeli gdje se nalaze.

```
for c in cnts:
    (x, y, w, h) = cv2.boundingRect(c)
    cv2.rectangle(imageA, (x, y), (x + w, y + h), (0, 0, 255), 2)
    cv2.rectangle(imageB, (x, y), (x + w, y + h), (0, 0, 255), 2)
```

Za prikazivanje razlika slici koristimo **for** petlju. U prethodnom koraku koordinate obrisa smo spremili u varijablu **cnts** korištenjem metode **imutils.grab_contours**. Varijabla **cnts** sadrži sve koordinate koji su nam potrebni za prikazivanje obrisa na slikama. Korištenjem **for** petlje definiramo da svaku stavku u nizu podataka **cnts** spremi pod varijablom **c** i iskoristi je u izvršavanju određenih operacija. Petlja će nastaviti niz operacija za svaku stavku dok ne izvrši posljednju. Metodom **cv2.boundingRect()** definiramo da ćemo raditi pravilni pravokutnik oko naših obrisa gdje će **x** i **y** biti varijable za koordinate gornje lijeve točke, **w** varijabla za dužinu i **h** za visinu.

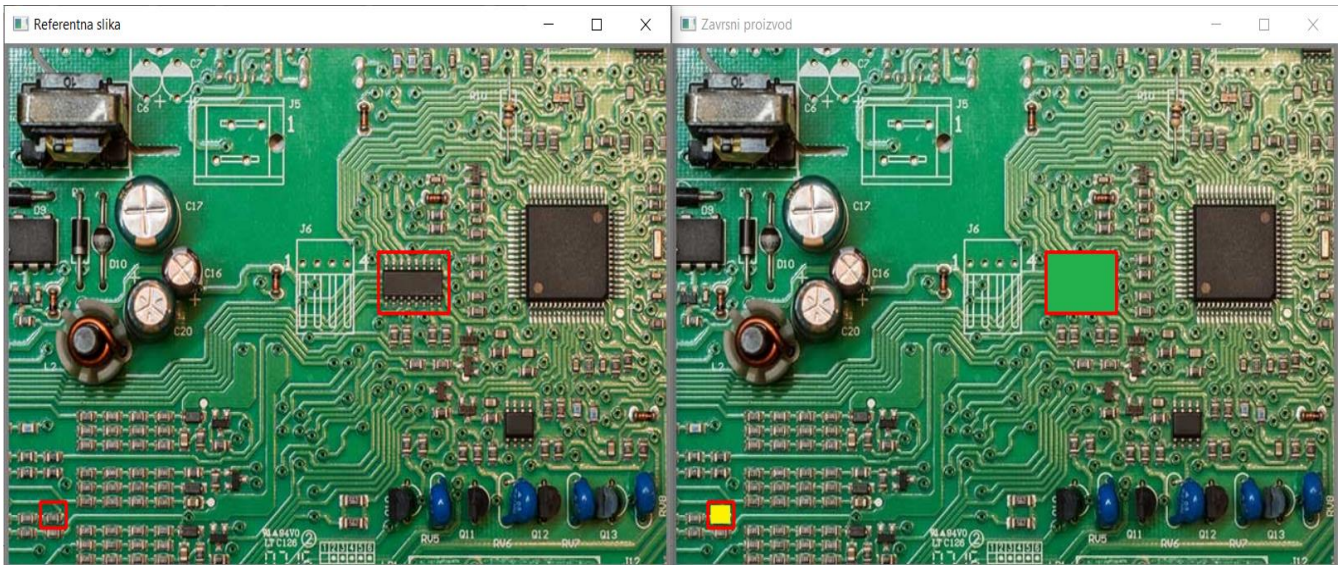
Kada smo definirali koordinate potrebno je ucrtati linije da bi bile vizualno prikazane na slikama. Za crtanje linija na slikama koristimo metodu **cv2.rectangle()**. Potrebno je definirati na kojoj slici ćemo izvršavati operaciju, u ovom slučaju to su **imageA** i **imageB** slike referentnog i završnog proizvoda. Početna točka je definirana u varijablama **x** i **y**, a oblik pravokutnika definiramo tako da točki **x** dodamo dužinu od **w** piksela u smjeru x pravca a točki **y** dodamo

dužinu od **h** piksela u smjeru y pravca **(x + w, y + h)**. Kao zadnje parametre definiramo da linije pravokutnika budu prikazane crvenom bojom **(0, 0, 255)** i da debljine linije bude 2 piksela

10.8 Korak 8: Prikazivanje rezultata

cv2.imshow("Referentna slika", imageA)
cv2.imshow("Zavrsni proizvod", imageB)
cv2.waitKey(0)
cv2.destroyAllWindows()

U završnom koraku prikazujemo rezultat usporedbe na zaslon. Na slici 23 se jasno vide lokacije elemenata slike koji su različiti. Razlike su označene u crvenoj boji kako bi se lakše uočili od strane operatora.



Slika 23 Prikaz rezultata

11 Primjer 2: Umjetna inteligencija za kontrolu odljevaka

U drugom primjeru primjene računalnog vida za kontrolu kvalitete korištena je umjetna inteligencija za razvoj sustava automatske kontrole u proizvodnji odljevaka. Tijekom procesa lijevanja dolazi do raznih nepravilnosti kao što su uključci ili poroznost na površini. Zadatak programa je automatski klasificirati prikazani odljevak kao ispravan ili neispravan. Ovaj sistem kontrole kvalitete može znatno ubrzati proizvodne procese jer se direktno na liniji proizvod može klasificirati kao škart. Trenutno kontrola odljevaka se vrši ručno i potreban je ljudski faktor kako bi klasificirali proizvod što zahtjeva mnogo vremena i ne pruža 100% sigurnost. U slučaju da kontrola nije dobro odrađena i neispravan proizvod ode u daljnju isporuku može doći do povrata čitave serije što uzrokuje velike gubitke kompanijama.

U ovom primjeru opisujemo korak po korak kako postaviti parametre za izradu umjetne neuronske mreže, kako naučiti model te kako testirati točnost modela neuronske mreže. Za izradu modela koristimo tip umjetne neuronske mreže pod nazivom Convolutional Neural Network (CNN). Za konfiguraciju neuronske mreže koristi se biblioteka Keras i razni alati iz biblioteke koji će biti opisani u daljnjem tekstu. Set podataka za učenje i treniranje modela pod nazivom „casting product image data for quality inspection“ [14] je preuzet sa Kaggle platforme koja pruža baze podataka za besplatno korištenje i istraživačke svrhe, autor je Ravirajsinh Dabhi. Model koji će biti opisan također se nalazi na platformi Kaggle pod nazivom „Casting Quality -> 98% training acc | 99,58% test“ [15] i napravljen je od strane korisnika GuillaumeSimler.

Skup podataka sastoji se od 7348 slika rotora podvodne pumpe i raspoređen je u dvije datoteke po nazivima „train“ i „test“ i svaka od datoteka sadrži dvije pod-datoteke pod nazivima „okfront“ i „deffront“

Struktura datoteka:
Train: - deffront ima 3758, okfront ima 2875 slika
Test: - deffront ima 453, okfront ima 262 slike

11.1 Korak 1: Učitavanje biblioteka:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Flatten
from keras.layers import Dense
from keras.layers import Dropout
```

U prvom koraku učitavamo potrebne biblioteke i alate koje ćemo koristiti u sljedećim koracima.

NumPy je Python biblioteka koja pruža operacije na višedimenzionalnim nizovima, raznim izvedenim objektima (poput matrica). NumPy nam pruža skup alata za brze operacije na nizovima, uključujući matematičke, logičke manipulacije oblicima, sortiranje, odabir, diskretne Fourierove transformacije, osnovnu linearnu algebru, osnovne statističke operacije i još mnogo toga [16]. **Pandas** je biblioteka pisana u Pythonu koja se koristi za manipulaciju i analizu podataka. Pandas predstavlja širok raspon alata od raščlanjivanja više formata datoteka do pretvaranja čitave tablice podataka u matrični niz što čini pandas pouzdanim saveznikom u strojnom učenju [17]. Za vizualno prikazivanje rezultata i statistike u grafovima koristimo **pyplot** skup funkcija.

Keras je knjižnica neuronske mreže otvorenog koda napisana u Pythonu. Može se izvoditi na TensorFlow, Microsoft Cognitive Toolkit, R, Theano ili PlaidML platformama. Dizajniran kako bi omogućio brzo eksperimentiranje s dubokim neuronskim mrežama, usredotočen je na prilagođenost korisnicima, modularnost i proširivost. Razvijen je kao dio istraživačkog napora projekta ONEIROS (Open-end Neuro-Electronic Intelligent Robot Operating System), a njegov glavni autor je François Chollet, Googleov inženjer [18].

Keras sadrži brojne implementacije najčešće korištenih blokova neuronske mreže kao što su slojevi, ciljevi, funkcije aktiviranja, optimizatori i niz alata koji olakšavaju rad sa slikovnim i tekstualnim podacima kako bi se pojednostavio kodiranje potreban za pisanje dubokog mrežnog koda mreže. Pored standardnih neuronskih mreža, Keras ima podršku za konvolucijsku i rekurentnu neuronsku mrežu [18].

11.2 Korak 2: Definiranje konstanti

FOLDER_ = 'Casting CNN'
BATCH_SIZE_ = 16
COLOR_SPECTRUM_ = (1)
IMG_SIZE_ = (300, 300)

U drugom koraku definiramo konstante koje će se koristiti u programu. Pod konstante podrazumijevamo nepromjenjive varijable koji će se koristiti u sljedećim koracima.
FOLDER označava lokaciju direktorija u kojem se nalazimo i koji sadrži sve što je potrebno za izradu modela neuronske mreže. **BATCH_SIZE** označava koliko će se uzoraka koristiti u jednom ciklusu treninga. Pošto u bazi podataka imamo 6633 uzoraka za treniranje, kompletan postupak treniranja trajat će 415 ciklusa.Varijabla **COLOR_SPECTRUM** označava koliko kanala boje ćemo koristiti u uzorcima za treniranje umjetne neuronske mreže. U ovom primjeru koristi se jedan kanal boja pošto su svi uzorci u nijansama sive. U slučaju da su slike u RBG spektru onda bi imali 3 kanala. **IMG_SIZE** varijabla označava koju veličinu slike koristimo što je u ovom primjeru 300x300 piksela.

11.3 Korak 3: Pokretanje CNN (Convolutional Neural Network)

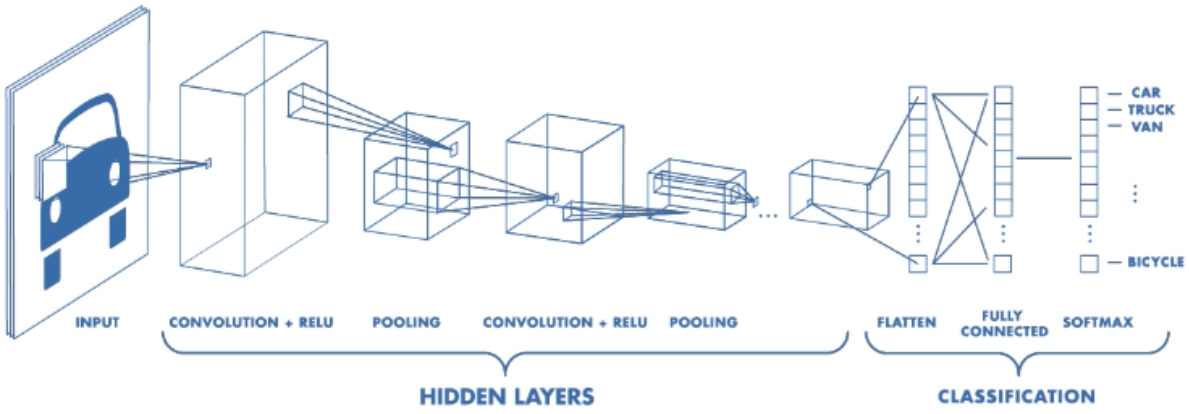
classifier = Sequential()

U dubokom učenju konvolucijske neuronske mreže (u sljedećem tekstu **CNN**) su vrste umjetnih neuronskih mreža koja se najčešće primjenjuje za vizualnu analizu slika i videa. CNN se pokazao kao odličan pristup pri rješavanju problema prepoznavanja objekata na slici ili klasifikaciji.
Tradicionalne neuronske mreže nisu idealne za obradu slike jer se moraju koristiti slike rastavljene u komade smanjene rezolucije. CNN radi na principu "neurona" koji su raspoređeni poput dijelova prednjeg režnja mozga, područja odgovornog za obradu vizualnih podražaja kod ljudi i kod životinja. Slojevi neurona raspoređeni su na takav način da pokrivaju cijelo vidno polje izbjegavajući problem obrade slike tradicionalnih neuronskih mreža[19].

Slojevi CNN-a sastoje se od ulaznog sloja, izlaznog sloja i skrivenog sloja koji uključuje više konvolucijskih slojeva[20]. Uklanjanje ograničenja i povećanje učinkovitosti za obradu slika

rezultira sustavom koji je daleko učinkovitiji i jednostavniji za učenje od tradicionalnih neuronskih mreža.

CNN obrađuje slike pojedinačno i dio po dio, dijelovi koje CNN traži zovu se značajke. Ako na dvije slike pronađe grubo podudaranje na približno istim položajima CCN dobiva puno bolji prikaz podudaranja. Svaka značajka je poput mini slike, malenog dvodimenzionalnog niza vrijednosti. Kada u CNN predstavimo novu sliku, CNN ne zna gdje će se značajke podudarati pa ih pokušava isprobati na svim mogućim položajima slike. Kada CNN traži podudaranje značajki kroz cijelu sliku prolazi filter koji je rezultat matematičke operacije konvolucije, te iz toga dolazi ime Konvolucijske Neuronske Mreže [21].



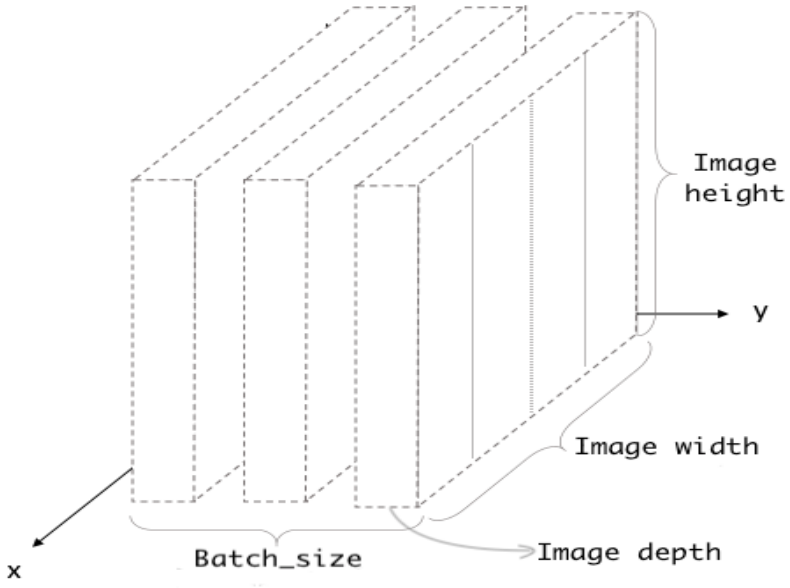
Slika 24 Standardna CNN arhitektura

U trećem koraku pod varijablom **classifier** pokrećemo skup funkcija **Sequential()** iz biblioteke Keras. Skupom funkcija **Sequential()** pokrećemo model koji će vršiti klasifikaciju slika iz baze podataka. Varijabla **classifier** će nam u budućnosti koristiti kako bi definirali kompletnu arhitekturu konvolucijske neuronske mreže. **Sequential** je jedan od mogućih alata iz biblioteke Keras i omogućuje nam da izgradimo neuronsku mrežu sloj po sloj gdje svi slojevi mogu biti međusobno povezani.

11.4 Korak 4: Definiranje CNN

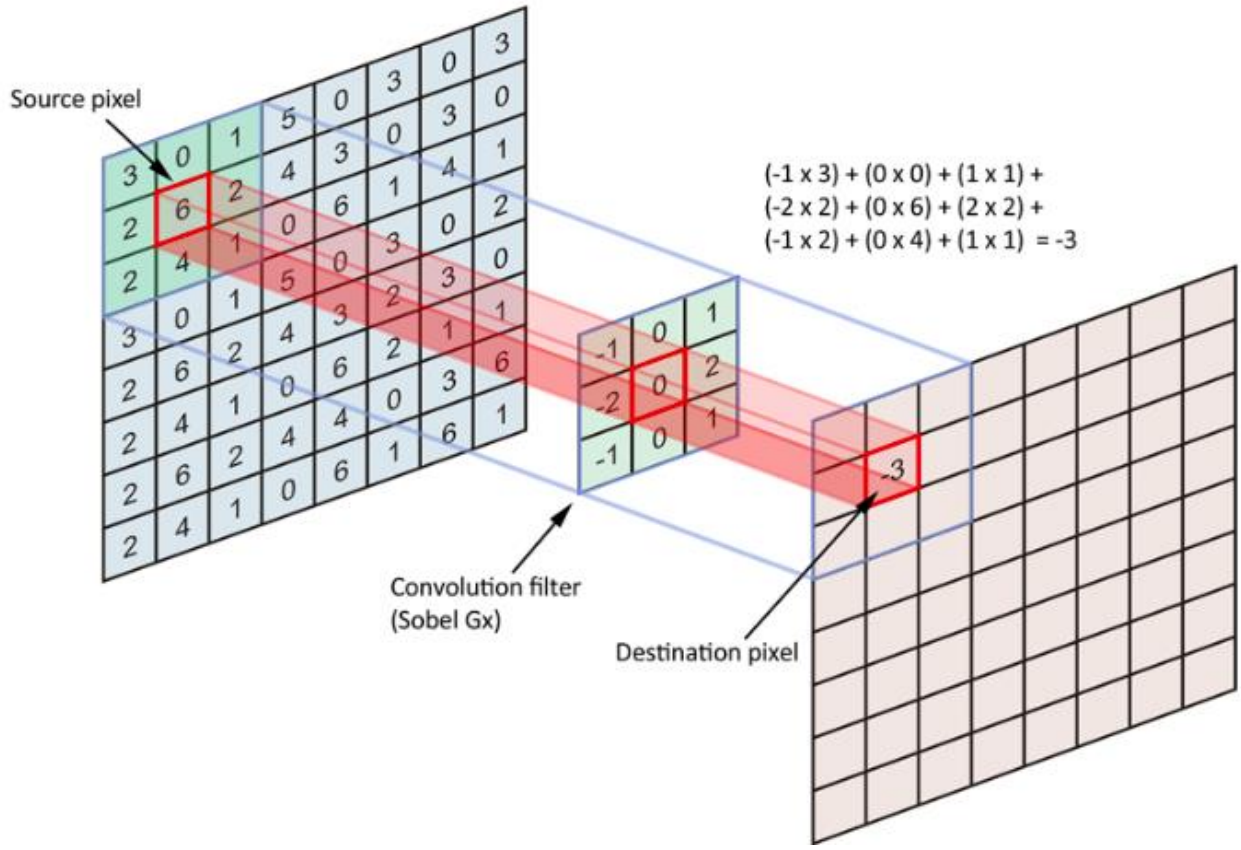
```
# Kreiranje slojeva
input_shape_ = (300,300) + (COLOR_SPECTRUM_, )
classifier.add(Conv2D(BATCH_SIZE_, (3,3), input_shape=input_shape_,
activation='relu'))
classifier.add(MaxPooling2D(pool_size=(2,2)))

# Kreiranje drugog sloja
classifier.add(Conv2D(BATCH_SIZE_, (3,3), activation='relu'))
classifier.add(MaxPooling2D(pool_size=(2,2)))

# Poravnavanje
classifier.add(Flatten())
```

U četvrtom koraku kreiramo konvolucijsku neuronsku mrežu. Keras nam omogućuje jednostavno kreiranje konvolucijske neuronske mreže tako da definiramo neuronske slojeve i aktivacijske funkcije. U dijelu ovog koraka moramo definirati ulazne parametre, koji će označavati početnu veličinu konvolucijske neuronske mreže. Varijabla **input_shape** označava karakteristiku uzoraka koji će se provoditi kroz neuronsku mrežu. Karakteristike uzoraka iz baze podataka su 300x300 piksela s jednim sivim spektrom boja. Kada definiramo ulazni sloj konvolucijske neuronske mreže kao ulazne podatke uvijek se mora postaviti 4D matrica. Ulazni podaci imaju oblik (veličina jedne serije (**BATCH_SIZE**), visina, širina, dubina) gdje prvi parametar označava veličinu serije a ostali predstavljaju dimenzije slike. Dubinu slike opisuje koliko spektara boje se nalazi na slici, ako je RBG onda je to 3 ili ako je u nijansi sive kao u ovom primjeru onda je 1.



Slika 25 Prikaz ulaznog sloja

U drugoj liniji koda kreiramo prvi sloj konvolucijske neuronske mreže. **Add** metoda označava da u naš model koji je aktiviran pozivom **classifier** funkcijom definiramo parametre neuronske mreže. Prvo definiramo da će neuronska mreža biti dvodimenzionalna konvolucijska neuronska mreža pozivom **Conv2D**. Conv2D je dio Keras biblioteke i zahtjeva parametre kao što su **filter, filter_size, input_shape, activation.** Pod **filter** u konvolucijskoj neuronskoj mreži podrazumijevamo matematičke operacije koje detektiraju značajke. Za svaki sloj CNN-a potrebno je definirati broj filtera, u ovom primjeru broj filtera je jednak broju ciklusa (64). Pošto značajke u jednoj slici mogu imati razne oblike filteri nam služe da otkrijemo razne značajke kao što su rubovi ili različite geometrije. CNN može imati nekoliko slojeva i sa svakim slojem filteri postaju napredniji u otkrivanju kompliciranijih značajki te imaju mogućnost otkrivanja objekata kao što su oči, usta ili uši na slici čovjeka. Drugi parametar koji moramo definirati je **filter_size** koji definira veličinu filtera, što je u ovom primjeru 3x3 matrica. Slika 26 prikazuje način rada filtera u CNN. Vrijednost filtera može biti jedna od mnogih operacija koje služe za procesiranje slike kao primjer detekcija rubova, binarizacija slike, zamagljivanje slike, oštrenje slike itd..

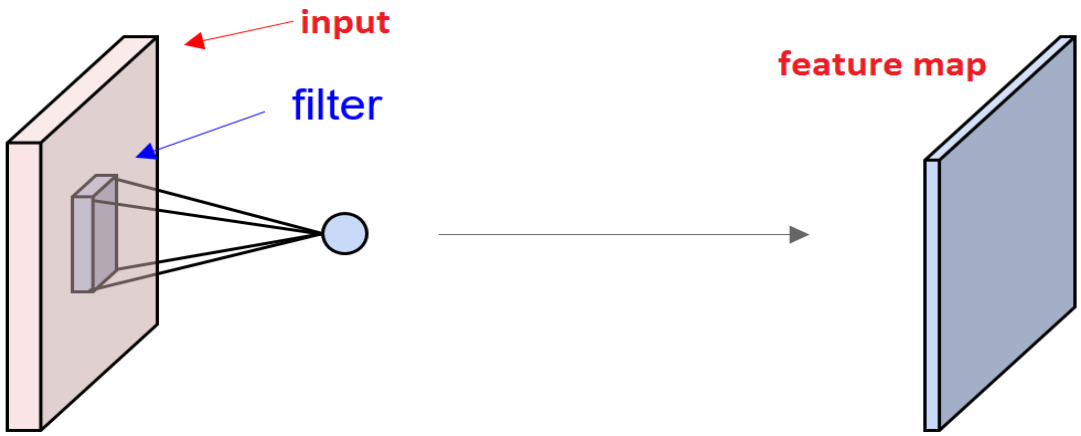


Slika 26 Filter u CNN

Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	

Slika 27 Primjer filtera

Tijekom prolaska filtera kroz ulaznu sliku kao rezultat dobije se velik broj mapa značajki (feature map) gdje svaka mapa sadrži određenu značajku koju je filter otkrio. Kroz sljedeće učenje određuje se koje značajke su nam bitne da dobijemo pravilnu klasifikaciju.

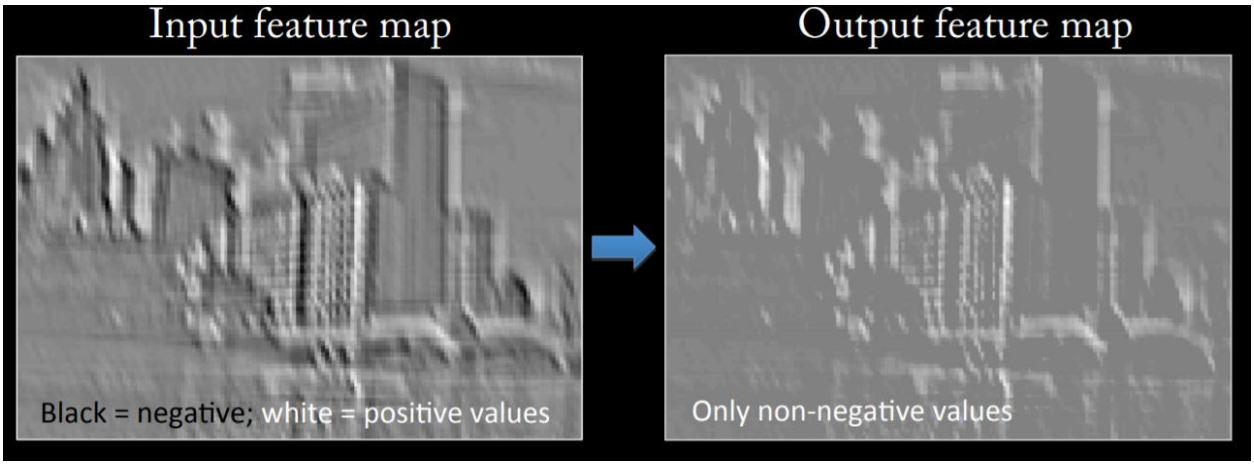


Slika 28 Mapa značajki

Skup mapa značajki prelazi u sljedeći sloj neuronske mreže ako smo ga definirali te se na njemu odrađuju sljedeće operacije konvolucije s filterima koje smo također na novo definirali. Kada prelazimo u sljedeći sloj neuronske mreže nije potrebno definirati ulazne veličine jer su one zapravo stvorene od mape značajki.

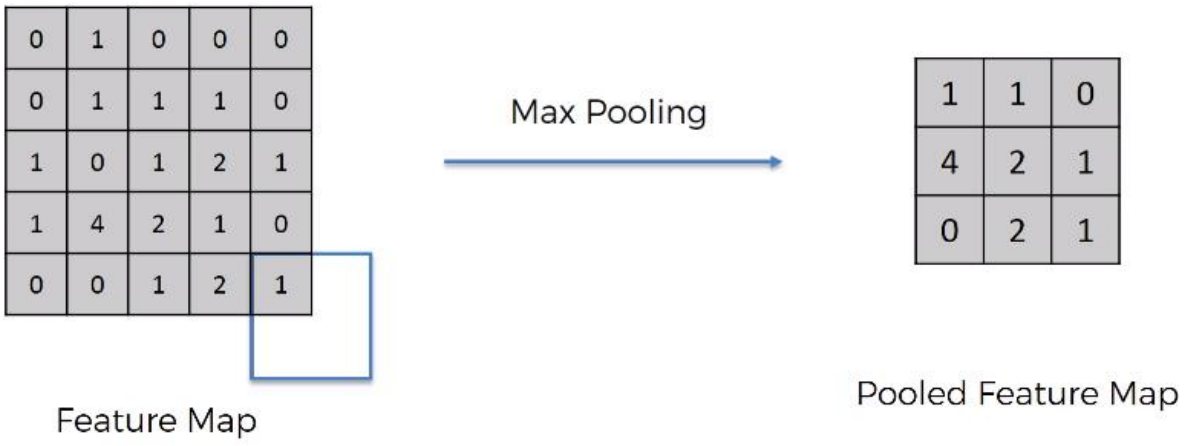
Za svaki sloj konvolucijske neuronske mreže potrebno je definirati i aktivacijsku funkciju (**activation function**). U umjetnim neuronskim mrežama **aktivacijska funkcija** čvora definira izlaz tog čvora s obzirom na ulaz ili skup ulaza [22]. U konvolucijskim neuronskim mrežama aktivacijske funkcije koristimo kako bi razbili linearnost u mapama značajki te tako dobili potpuno novi sloj neuronske mreže. Aktivacijske funkcije također koristimo kako bi neuronska mreža mogla odrediti koji neuroni će se aktivirati na određene signale, ona određuje koji raspon vrijednosti je odgovarajući da se određeni neuron uključi. Raspon aktivacije za svaki neuron je precizniji što mreža ide dublje. Neuronska mreža bez funkcije aktiviranja u osnovi je samo model linearne regresije. Funkcija aktiviranja vrši nelinearnu transformaciju na ulazne podatke što čini neurone sposobnijim za učenje i obavljanje složenijih zadataka [23]. Razbijanje linearnosti na slikama omogućuje bolje detektiranje značajki u sljedećim operacijama što ujedno dovodi i boljeg rezultata u učenju modela. U primjeru ovog projekta korištena je funkcija aktiviranja **relu. Relu** je najčešća funkcija aktiviranja koja se koristi u konvolucijskim neuronskim mrežama za uklanjanje linearnosti. Prolaskom filtera kroz sliku, linearnost se može uočiti najbolje na rubovima. Slika 29 prikazuje sliku sklopa zgrada koje su prošle proces konvolucije filtera te je prva slika zapravo rezultat konvolucije, mapa značajki. Ako se promatraju rubovi zgrade može se primijetiti kako postoji prijelaz s tamnih dijelova na svijetle, dolazi do postepene promjene kontrasta. Ta

promjena kontrasta je primjer linearnosti koja se uklanjanja funkcijom aktiviranja da bi dobili izraženije oblike ili rubove.



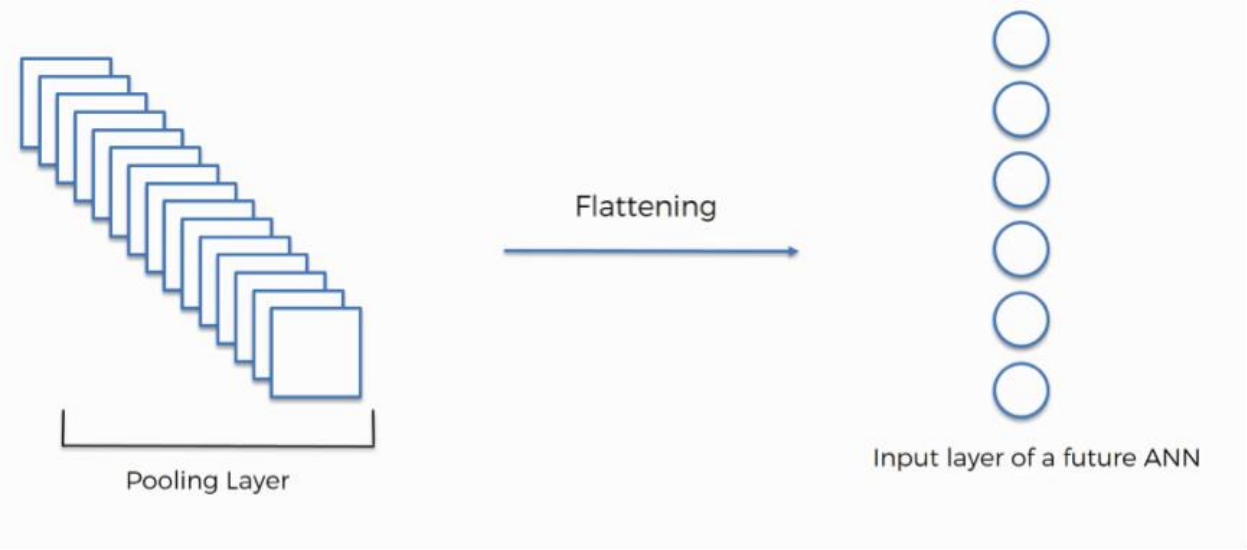
Slika 29 Implementacija aktivacijske funkcije

U 4. koraku ovog primjera do sada smo definirali parametar za kreiranje prvog konvolucijskog sloja. Pod parametrima definirali smo broj filtera koji će tražiti značajke slike za treniranje modela i njihovu veličinu, nakon toga smo definirali veličinu ulaznih parametra naše slika i aktivacijsku funkciju koja će djelovati na ulazne parametre mape značajki kako bi razbili linearnost. Nakon ovih procesa u umjetnoj neuronskoj mreži imamo značajke koje su potrebne da klasifikaciju, međutim značajke ne moraju na svim uzorcima biti jednake i nalaziti se na jednakom mjestu. Kako bi riješili taj problem moramo provesti još jedan proces pod nazivom **pooling**. Svrha **pooling-a** je postizanje prostorne invarijancije redukcijom mapa značajki. Svaka objedinjena mapa značajki odgovara jednoj mapi značajki iz prethodnog sloja [24]. **Max_Pooling** je jedna od mnogih funkcija poolinga koja se najčešće koristi u CNN. U korištenom primjeru koristi se veličina matrice 2x2, što znači da se iz svakog polja mape značajki veličine 2x2 uzima najveća vrijednost te se stvara nova mapa značajki koja sadrži puno manje informacija. Prednost ovog koraka je što imamo manje informacija s kojima treba u sljedećem sloju raditi a kvaliteta informacija je ostala ista pošto svakako tražimo najveće vrijednosti koje reprezentiraju značajke koje treba detektirati.



Slika 30 Prikaz procesa Pooling

U ovom primjeru imamo 2 sloja CNN-a te nakon što su svi procesi za oba sloja završeni potrebno je odraditi poravnavanje (**flattening**) podataka. Na kraju procesa drugog sloja CNN-a imamo puno manje podataka ali zato imamo dovoljno informacija za sve značajke koje su nam bitne kako bi radili klasifikaciju. Razlog zašto se mora odraditi poravnavanje je taj što podatke moramo ubaciti u umjetnu neuronsku mrežu kako bi naučili model koje značajke treba detektirati.



Slika 31 Proces poravnavanja (flattening)

11.5 Korak 5: Kreiranje duboke neuronske mreže

```
# Add a second hidden layer
classifier.add(Dense(units = 128, activation='relu'))
classifier.add(Dropout(rate=.2))

# Add a second hidden layer
classifier.add(Dense(units = 128, activation='relu'))
classifier.add(Dropout(rate=.2))

# Add a third hidden layer
classifier.add(Dense(units = 64, activation='relu'))
classifier.add(Dropout(rate=.2))

classifier.add(Dense(units = 1, activation='sigmoid'))
```

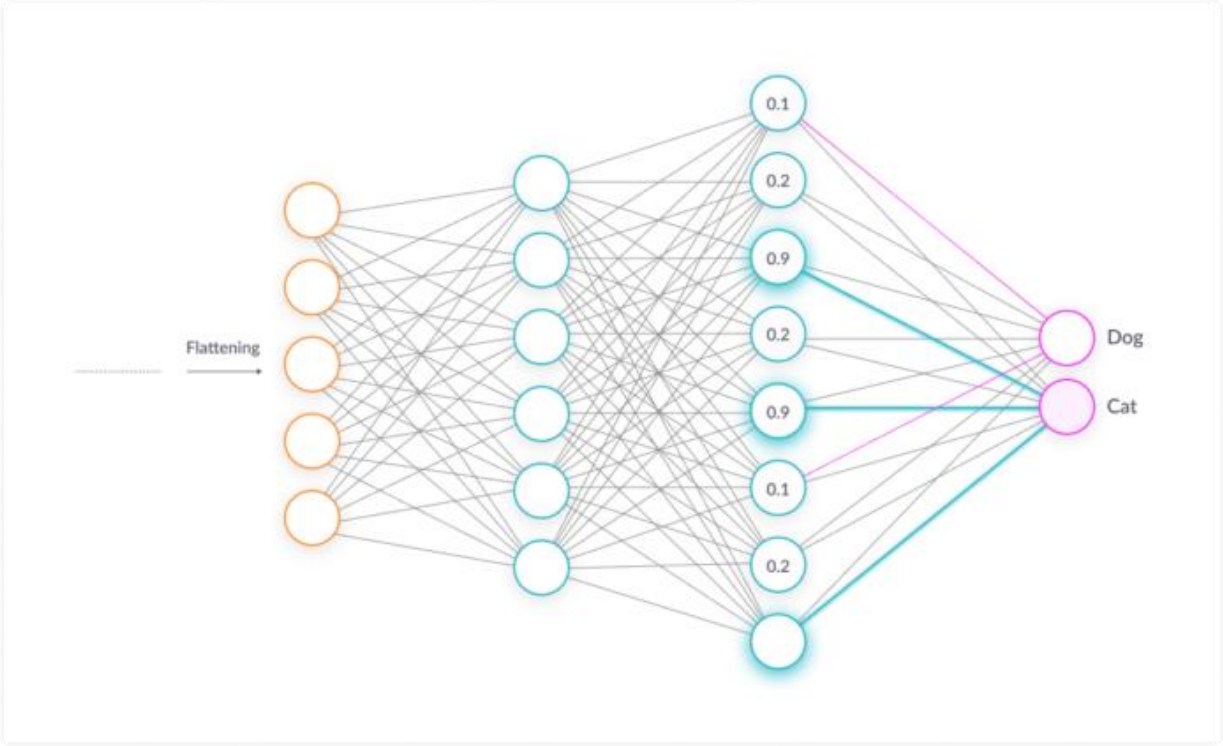
Kod stvaranja CNN mreža potrebno je napraviti kompleksnu neuronsku mreža gdje su svi neuroni međusobno povezani kako bi se moglo razlučiti koje značajke odgovaraju kojoj klasifikaciju. Keras nam omogućuje kreiranje neuronske mreže s potpuno povezanim slojevima.
Potpuno povezani slojevi bitna su sastavnica konvolucijskih neuronskih mreža (CNN-a), koje su se pokazale vrlo uspješnima u prepoznavanju i razvrstavanju slika za računalni vid. CNN postupak započinje konvolucijom i pooling-om, raščlanjivanjem slike na značajke i njihovom neovisnom analizom. Rezultat ovog procesa temelji se na potpuno povezanoj strukturi neuronske mreže koja donosi konačnu odluku o klasifikaciji [25]. U neuronskoj mreži neuroni su povezani sinapsama i međusobno razmjenjuju informacije, koliko je određena informacija bitna jednom od neurona se izražava preko težine (weights). U procesu učenja težina je faktor kojim se ulazne informacije multipliciraju te na osnovu rezultata neuroni uče i prilagođavaju težinu određenim ulaznim informacijama.

Rezultat konvolucije prolazi poravnavanje u jedan vektor vrijednosti koji predstavlja vjerojatnost da neko svojstvo pripada nekoj oznaci. Na primjer, ako je slika mačke, značajke koje predstavljaju stvari poput oštrica ili krzna treba imati veliku vjerojatnost da bude klasificirani kao "mačka". Slika 32 ilustrira kako ulazne vrijednosti ulaze u prvi sloj neurona. Oni se množe s težinama i prolaze kroz aktivacijsku funkciju (tipično ReLu) kao u klasičnoj umjetnoj neurološkoj mreži.

Zatim prelaze naprijed do izlaznog sloja u kojem svaki neuron predstavlja klasifikacijsku oznaku [26].

U slučaju neuronske mreže ovog koraka radi se o mreži s 4 sloja. Prva 2 sloja sadrže 128 neurona dok 3 sadrži 64 sloja. Aktivacijska funkcija u tri sloja je **relu** koja je opisana u prethodnom koraku dok je u četvrtom sloju aktivacijska funkcija **Sigmoid**. **Sigmoid** aktivacijska funkcija koristi se kada rezultat mora biti binaran, 0 ili 1. Prednost ove funkcije je korištenje u primjeni kada treba razlučiti dali nešto postoji ili ne, kao što je u ovom primjeru oštećenje odljevka.

Na osnovi baze podataka frontok i frontdef model će naučiti kako izgledaju ispravni i neispravni uzorci odljevka. Na temelju značajki prepoznati će da neispravni uzorci imaju određene piksele značajki koje se znatno razlikuju od uzoraka ispravnog proizvoda te će na osnovu toga odraditi klasifikaciju.



Slika 32 Prikaz neuronske mreže za klasifikaciju

Zadnjem sloju definiramo samo jedan izlaz zbog toga što radimo klasifikaciju i za svaki ciklus učenja imamo jedan rezultat, dali je ok ili def.

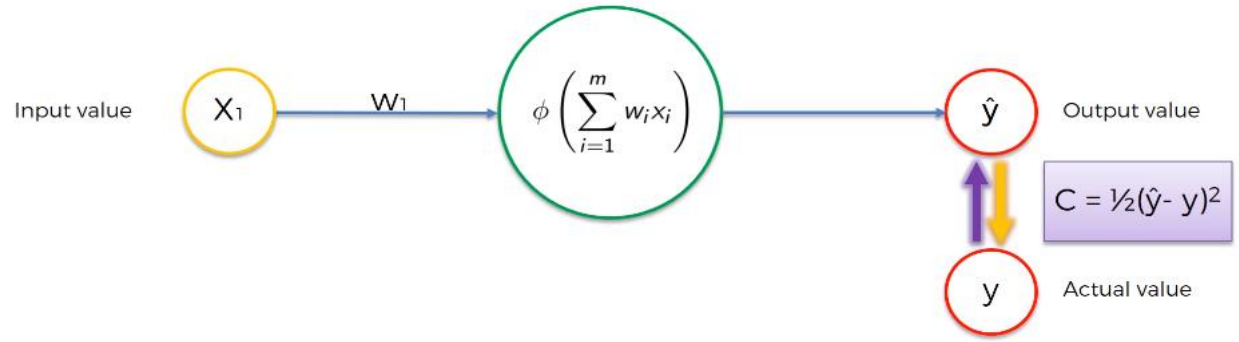
Kod trećeg sloja se radilo i izbacivanje neurona, **Dropout.** To je tehnika kod koje se nasumično odabrani neuroni zanemaruju tijekom treninga. Oni se nasumično "izbacuju", to znači da se njihov doprinos aktivaciji nizvodnih neurona vremenski uklanja na prednjem prolazu i nikakvo ažuriranje

težine se ne primjenjuje na neuron na stražnjem prolazu [27]. **Dropout** se provodi nasumičnim odabirom neurona koji se trebaju isključiti s danom vjerojatnošću (npr. 20%) za svaki ciklus ažuriranja težine. **Dropout** se koristi samo tijekom obuke modela i ne koristi se pri ocjenjivanju vještina modela.

11.6 Korak 6: Optimizacija učenja

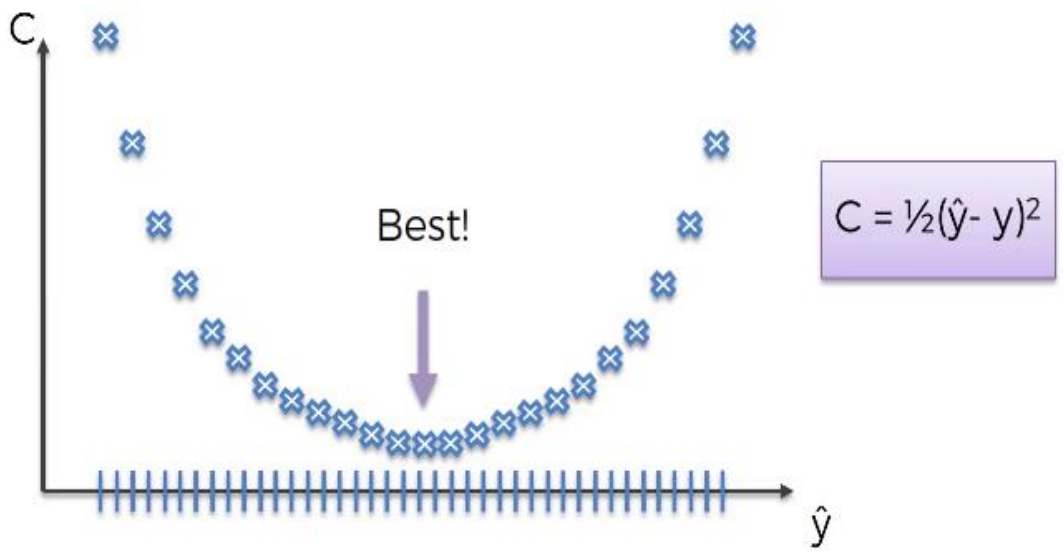
```
classifier.compile(optimizer= 'adam',
    loss='binary_crossentropy',
    metrics=['accuracy'])
```

Neuronske mreže općenito grade znanje iz skupova podataka u kojima je unaprijed zadan odgovor. Neuronske mreže uče prilagođavajući i podešavajući se da samostalno pronađu pravi odgovor povećavajući točnost svojih predviđanja. Neuronske mreže se prilagođavaju tako što uspoređuju početne izlaze s ponuđenim točnim odgovorom ili ciljem (slika 33). Način učenja u neuronskim mrežama se naziva pod terminom „Backprogagation“



Slika 33Prikaz prilagođavanja neuronske mreže

Kada promatramo primjer iz slike 33 onda imamo najjednostavniji primjer neuronske mreže s jednim neuronom. Neuron prima jedan ulazni podataka preko sinapse koja ima određenu težinu, ulazni podatak se obrađuje preko aktivacijske funkcije te imamo izlazni parametar. Kako bismo dobili točnost ove mreže uspoređujemo stvarni podatak y sa izlaznim podatkom. Razlika između njih se naziva “activation cost”. Optimizaciju koristimo kako bi pronašli težine gdje smo imali najmanju razliku između stvarne vrijednosti i dobivenog rezultata (slika 34).



Slika 34 Prikaz razlika u usporedbi

U tradicionalnom umjetnom neuronskom učenju najčešća metoda optimizacije učenja je gradijalni stohaistički spust. **Adam** je algoritam optimizacije koji se može koristiti umjesto klasičnog postupka stohastičkog spusta gradijenta za ažuriranje mrežne težine iterativno na temelju podataka o treningu. **Adam** se razlikuje od klasičnog stohastičkog gradijenta. Stohastički pad gradijenta održava jedinstvenu stopu učenja (koja se naziva alfa) za sva ažuriranja težine, a stopa učenja se ne mijenja tijekom treninga. Kod **Adam-a** stopa učenja održava se za svaku težinu mreže i zasebno se prilagođava kako se učenje odvija [28].

Kako bi znali dali je model točno odradio klasifikaciju okfront ili deffront koristimo funkciju **binary_crossentropy**. Binary_crossentropy mjeri koliko daleko predviđanje je bilo od stvarne vrijednosti za svaku stavku iz istog razreda te radi sveukupan prosjek da bi znali kolika je bila ukupna greška. Vrijednost točnosti predviđanja se kreće između 0 ili 1, što je niža vrijednost to je bolja točnost previđanja [29].

11.7 Korak 7: Proširivanje baze podataka

```
from keras.preprocessing.image import ImageDataGenerator
train_datagen = ImageDataGenerator(
        rescale=1./255,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True,
        vertical_flip=True,
        rotation_range=25)
test_datagen = ImageDataGenerator(rescale=1./255)
```

Povećanje slikovnih podataka tehnika je koja se može koristiti za umjetno proširivanje veličine skupa podataka za trening kreiranjem modificiranih verzija slika u skupu podataka.
Neuronske mreže dubokog učenja koje koriste više podataka mogu rezultirati vještijim modelima, a tehnike proširenja mogu stvoriti varijacije slika koje mogu poboljšati sposobnost odgovarajućih modela da generiraju ono što su naučile na novim slikama.

Kerasova knjižnica neuronske mreže dubokog učenja pruža mogućnost da se proširenje podataka uklopi u modele pomoću **ImageDataGenerator**. ImageDataGenerator nam omogućuje da slike iz baze podataka proširimo, povećamo, smanjimo, rotiramo i preokrećemo kako bi stvorili još veću količinu podataka da se model bolje nauči.

11.8 Korak 8: Učitavanje podataka

Kada smo kreirali neuronsku mrežu potrebno je učitati podatke na kojima će se mreža učiti. Podaci se nalaze u direktorijima za treniranje i testiranje koje je potrebno definirati kako bi ih Keras prepoznao.

```
from pathlib import Path
dataset_folder =  Path('/kaggle/input/real-life-industrial-dataset-of-casting-
product/casting_data')
if dataset_folder.exists():
   print(f'[-------]\nConnect the dataset folder at \n\t{str(dataset_folder)}\n[-------]')
else:
   print(f'[********]\nConnecting the dataset folder failed \n[********]')
```

rezultat:

```
[-------]
Connect the dataset folder at
        /kaggle/input/real-life-industrial-dataset-of-casting-product/casting_data
[-------]
```

Slika 35 Učtavanje baze podataka

Prvi dio je povezivanje sa glavnom bazom podataka u kojoj se ostali direktoriji nalaze. U ovom dijelu definiramo gdje se glavni direktorij nalazi te se koristi *if* uvjetna naredba da dobijemo obavijest ako smo učitali bazu podataka ili ako baza podataka nije učitana.

Nakon učitavanja glavne baze podataka imamo pristup direktorijima gdje se nalaze podaci za učenje i testiranje konvolucijske neuralne mreže „test“ i „train“.

```
training_files = dataset_folder / 'train/'
test_files = dataset_folder / 'test/'
training_set= train_datagen.flow_from_directory(
        training_files,
        target_size=(300,300),
        batch_size=BATCH_SIZE_,
        class_mode='binary',
        color_mode="grayscale")

test_set = test_datagen.flow_from_directory(
        test_files,
        target_size= (300,300),
        batch_size=BATCH_SIZE_,
        class_mode='binary',
        color_mode="grayscale")
        training_size = len(training_set)
        test_size = len(test_set)
```

Kada učitamo direktorije s podacima za testiranje i učenje potrebno je definirati parametre o podacima. Za setove treninga i učenja potrebno je upisati kolika je veličina pojedinačnih podataka, koliko serija podataka će se koristiti u jednom ciklusu učenja (**BATCH_SIZE**), koliko rezultata klasifikacije očekujemo (**class_mode**) i koji spektar boje koristimo (**color_mode**).

Varijable **training_size** i **test_size** sadrže veličine skupova podataka za učenje i testiranje modela.

rezultat

```
Found 6633 images belonging to 2 classes.
Found 715 images belonging to 2 classes.
```

Slika 36 Učitavanja podataka

11.9 Korak 9: Definiranje načina obrade podataka

```
import multiprocessing
import tensorflow as tf
def set_workers(local = False):
      catcha =''
      workers = multiprocessing.cpu_count()
      if local:
        workers -= 1
        catcha = 'locally '
  gpus = tf.config.experimental.list_physical_devices('GPU')
   print(f"Working with {workers} CPU threads {catcha}and with {len(gpus)} GPU" )
   return workers
workers_ = set_workers()
```

Za učenje umjetnih neuronskih mreža potrebno je mnogo procesne snage te što više tehnologija napreduje omogućeno nam je brže kreiranje modela. Za učenje modela moguće je koristiti CPU (central processing unit) ali ipak kada se radi s konvolucijskim neurološkim mrežama preporučuje se korištenje GPU (graphics processing unit). U ovom primjeru definirano je da učenje modela koristimo i GPU i CPU. U kodu prikazanom gore „workers“ označava skup procesora koje ćemo koristiti za izradu modelu. Kada u sljedećim koracima koristimo varijablu **workers_** onda zapravo pozivamo funkciju **set_workers()** koja pokreće korištenje 2 CPU jedinice i jedne GPU jedinice.

```
Working with 2 CPU threads and with 1 GPU
```

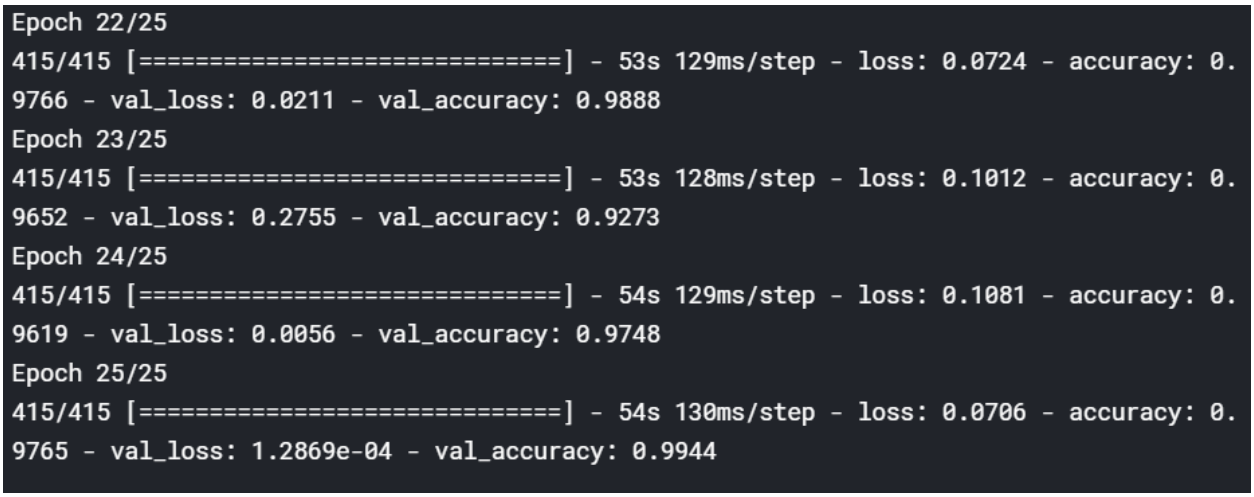
Slika 37 Rezultat poziva procesnih jedinica

11.10 Korak 10: Učenje modela

```
classifier.fit_generator(
    training_set,
    steps_per_epoch=training_size,
    epochs=25,
    validation_data=test_set,
    validation_steps=test_size,
    use_multiprocessing=True,
    workers=workers_)
```

Korak 10 je posljednji korak u konfiguraciji modela. **Fit_generator** funkcija nam daje mogućnost jednostavne konfiguracije svih parametara potrebnih za pokretanje učenja neuronske mreže. Moramo definirati koji skup podataka će se koristiti za treniranje modela, što je u ovom primjeru **training_set**. **Steps_per_epoch** varijabla je potrebna kako bi definirali koliko koraka učenja će se koristiti u svakoj epohi učenja. Epoha je pojam koji se označava broj prolaza čitavog skupa podataka u treningu koje je algoritam strojnog učenja odradio. Skupovi podataka obično se grupiraju u manje skupine (posebno ako je količina podataka vrlo velika) [30]. Ako je veličina serije cijeli skup podataka treninga, tada je broj epoha broj iteracija. Iz praktičnih razloga to obično nije slučaj. Mnogi modeli su stvoreni s više epoha. Broj epoha u ovom primjeru je 25 a broj koraka u svakoj epohi će biti veličina **training_size** koja je 415.

Kod testiranja točnosti neuronske mreže potrebno je također definirati skup podataka koji je u ovom primjeru **test_set**. Također je potrebno odrediti koliko koraka testiranja će se izvršiti, što je u ovom primjeru veličina **test_size** koja je 45. Da bi izvršili učenje i testiranje mreže potrebno je odrediti s kojim procesnim jedinicama raspolažemo i kako ćemo ih prenamijeniti. U prošlom koraku definirali smo **workers_** gdje smo odredili da koristimo dvije CPU i jednu GPU jedinicu.



Slika 38 Učenje neuronske mreže

Kada potvrdimo sve parametre pokrećemo učenje konvolucijske neuronske mreže, slika 38. Za svaku epohu učenja imamo prikaz vremena trajanja učenja, kolika je greška i točnost predviđanja za svaku epohu. Kako broj epoha raste tako imamo sve bolju točnost što znači da model napreduje iz epohe u epohu. U zadnjoj epohi vrijednost **accuracy** iznosi **0.9765** što daje točnost klasifikacije od **97,65%** kada koristimo skup podataka na kojima je model treniran dok kada koristimo skup podataka za testiranje onda točnost prikazuje varijabla **val_accuracy** koja iznosi **0.9944** što daje točnost od **99,44%** .

12 Zaključak

Trenutačno u svijetu sve kompanije teže prema automatizaciji i samostalnosti poslova vezanih za kontrolu kvalitete. Kontrola kvalitete je širok pojam i znatno se razlikuje od proizvoda do proizvoda. Računalni vid znatno ubrzava procese kontrole kvalitete i u nekim segmentima kontrole kvalitete omogućava kontrolu cjelokupne serije proizvoda. Uz pomoć Python programskog jezika omogućena je brza izrada programa za procesiranje slika te izvođenje brzih algoritama za detekciju nepravilnosti na proizvodu.

Unutar ovog rada opisane su mogućnosti Python programskog jezika, OpenCV, NumPy i Keras biblioteka za obradu slika i prikazivanje rezultata. U prvom primjeru eksperimentalnog dijela prikazana je primjena OpenCV biblioteke s alatom compare_ssim kako bi pronašli razlike između referentne slike i završnog proizvoda u kontroli PCB ploče. Iako compare_ssim brzo detektira razlike u pikselima i omogućuje jednostavno prikazivanje razlika na obje slike u stvarnoj primjeni ova metoda detekcije može imati značajne nedostatke. Za korištenje ovog alata, slika proizvoda kojeg kontroliramo mora imati slične karakteristike kao referentna slika. Ova metoda ne dopušta razlike u veličinama slika, čak i ako je ta razlika 1 piksel uzdužno ili poprečno. To znači da kada slikamo završni proizvod uvjeti kao što su kut slikanja, povećanje, jačina svjetlosti ili kontrast uvijek moraju biti jednaki. U industrijskim procesima jednake uvjete je jako teško ostvariti čak i kada je su u pitanju odvojeni laboratoriji jer može doći do pogreške u ljudskom faktoru.

Kako bi riješili problem standardne obrade slike gdje spomenuti uvjeti moraju biti jednaki, kontrola kvalitete uz korištenje računalnog vida sve više ide prema korištenju umjetne inteligencije. U drugom primjeru rada prikazana je primjena umjetne inteligencije za kontrolu odljevaka rotora za podvodne pumpe. Biblioteka Keras nam omogućuje jednostavno kreiranje neuronske mreže i jednostavnu pripremu podataka za učenje i testiranje modela. Učenje modela na skupu podataka zahtjeva mnogo procesne snage te je iz tog razloga opisan gotov primjera autora GuillaumeSimler pod nazivom „Casting Quality -> 98% training acc | 99,58% test“ . U ovom primjeru prikazana je primjerna CNN tipa neuronske mreže kako bi cjelokupni skup podataka pripremili u ulazne podatke te ih dalje procesirali u standardnoj neuronskoj mreži da naučimo model. Za učenje i testiranje modela korišten je skup podataka od 7348 slika. U procesu pripreme podataka, podaci koji su namijenjeni za učenje su dodatno manipulirani kako bi model što bolje prepoznao defekte neovisno o rotaciji i poziciji komada na slici, promjeni veličine slike ili o jačini

svjetlosti i kontrasta. U realnoj primjeni umjetne inteligencije za kontrolu kvalitete nedostatak je prikupljanje dovoljne količine podataka za učenje ili testiranje. Potrebno je dovoljna količina podataka kako bi model odrađivao klasifikaciju na pravilan način te je potrebna stručna osoba koja bi usavršavala model. Tehnologija konvolucijskih neuronskih mreža nije još napredovala da bi se sama učila ili usavršavala ali znanstvenici rade da i taj problem riješe. Jedno od trenutnih opcija je metoda „Transfer learning“ gdje je model već naučen na određenoj velikoj bazi podataka te se puno lakše prilagođava novom učenju. Modeli koju omogućavaju ovaj način rada su dostupni od velikih globalnih kompanija kao što su Google ili Amazon. Kamere koje su postavljaju na industrijske linije sve više koriste integrirane procesore za obradu podataka što znatno ubrzava vrijeme obrade slike. Napretkom obrade podataka u „oblaku“ bit će omogućeno učenje modela bez vlastite opreme te integracija modela u kamere omogućavat će sve veću primjenu umjetne inteligencije za prepoznavanje grešaka i klasifikaciju.

13 Literatura

1. [Richard Szeliski (30 September 2010). Computer Vision: Algorithms and Applications. Springer Science & Business Media. pp. 10–16. ISBN 978-1-84882-935-0.]
2. https://www.pcmag.com/news/what-is-computer-vision
3. https://hackernoon.com/a-brief-history-of-computer-vision-and-convolutional-neural-networks-8fe8aacc79f3
4. https://dzone.com/articles/computer-vision-overview-of-an-ai-cutting-edge-tec
5. https://dzone.com/articles/computer-vision-overview-of-an-ai-cutting-edge-tec
6. https://dzone.com/articles/computer-vision-overview-of-an-ai-cutting-edge-tec
7. https://www.hermary.com/learning/3d-vision-data-look-like/
8. https://www.visiononline.org/vision-resources-details.cfm/vision-resources/Computer-Vision-vs-Machine-Vision/content_id/4585
9. https://www.geeksforgeeks.org/opencv-overview/
10. Mastering OpenCV Third Edition, Roy Shilktor & David Millan Escriva
11. https://blog.francium.tech/feature-detection-and-matching-with-opencv-5fd2394a590
12. https://docs.anaconda.com/anaconda/navigator/
13. https://jupyter.org/
14. https://www.kaggle.com/ravirajsinh45/real-life-industrial-dataset-of-casting-product
15. https://www.kaggle.com/guillaumesimler/casting-quality-98-training-acc-99-58-test
16. https://numpy.org/doc/stable/user/whatisnumpy.html
17. https://www.educative.io/edpresso/what-is-pandas-in-python
18. Chollet, François (2016). "Xception: Deep Learning with Depthwise Separable Convolutions". arXiv:1610.02357.
19. https://en.wikipedia.org/wiki/Convolutional_neural_network
20. https://searchenterpriseai.techtarget.com/definition/convolutional-neural-network
21. rohrer.github.io/how_convolutional_neural_networks_work.html
22. Hinkelmann, Knut. "Neural Networks, p. 7" (PDF). University of Applied Sciences Northwestern Switzerland.
23. https://www.geeksforgeeks.org/activation-functions-neural-networks/
24. Evaluation of Pooling Operations in Convolutional Architectures for Object Recognition Dominik Scherer, Andreas Muller, Sven Behnke
25. https://missinglink.ai/guides/convolutional-neural-networks/fully-connected-layers-convolutional-neural-networks-complete-guide/
26. https://missinglink.ai/guides/convolutional-neural-networks/fully-connected-layers-convolutional-neural-networks-complete-guide/
27. https://machinelearningmastery.com/dropout-regularization-deep-learning-models-keras/
28. https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning
29. https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/loss-functions/binary-crossentropy
30. https://radiopaedia.org/articles/epoch-machine-learning