# Konvolucijska neuronska mreža za klasifikaciju objekata temeljem boje i oblika

**Kos, Matija**




SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

# ZAVRŠNI RAD

**Matija Kos**

Zagreb, 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

# ZAVRŠNI RAD

Mentor: Student:

Izv. prof. dr. sc. Tomislav Stipančić, dipl. ing. Matija Kos

Zagreb, 2024.

Izjavljujem da sam ovaj rad izradio samostalno koristeći znanja stečena tijekom studija i navedenu literaturu.

Zahvaljujem se mentoru izv. prof. dr. sc. Tomislavu Stipančiću na pruženoj podršci i pomoći tijekom izrade ovog završnog rada.

Također se posebno zahvaljujem svojoj obitelji, prijateljima i kolegama na neizmjernoj podršci koju su mi pružili tijekom preddiplomskog studija.

Matija Kos

SVEUČILIŠTE U ZAGREBU
**FAKULTET STROJARSTVA I BRODOGRADNJE**
Središnje povjerenstvo za završne i diplomske ispite
Povjerenstvo za završne i diplomske ispite studija strojarstva za smjerove:
proizvodno inženjerstvo, računalno inženjerstvo, industrijsko inženjerstvo i menadžment, inženjerstvo materijala i mehatronika i robotika

| Sveučilište u Zagrebu<br>Fakultet strojarstva i brodogradnje | |
|---|---|
| Datum | Prilog |
| Klasa: **602 – 04 / 24 – 06 / 1** | |
| Ur.broj: **15 – 24 –** | |

# ZAVRŠNI ZADATAK

Student: **Matija Kos** JMBAG: **0035225994**

Naslov rada na hrvatskom jeziku: **Konvolucijska neuronska mreža za klasifikaciju objekata temeljem boje i oblika**

Naslov rada na engleskom jeziku: **Convolutional neural network for object classification based on color and shape**

Opis zadatka:

Zadnjih godina tehnike dubokog učenja postižu zapažene rezultate kod rješavanja problema detekcije i klasifikacije objekata na slikama. Detekcija objekata je zadatak iz domene računalnog vida koji uključuje određivanje prisutnosti, lokalizacije te identifikacije jednog ili više objekata na slici od interesa. Klasifikaciju je moguće temeljiti na različitim značajkama objekata te na različitim metodama.
U sklopu rada potrebno je:

- generirati bazu slika koje sadrže objekte različitih oblika i boja za trening i testiranje koristeći Python programski jezik
- osmisliti te izraditi model konvolucijske neuronske mreže za prepoznavanje objekata na temelju boje i oblika
- evaluirati model neuronske mreže koristeći standardne evaluacijske metrike
- implementirati model konvolucijske neuronske mreže na stvarnom vizijskom sustavu u sklopu Laboratorija za projektiranje izradbenih i montažnih sustava te dati kritički osvrt.

U radu je potrebno navesti korištenu literaturu i eventualno dobivenu pomoć.

Zadatak zadan: 30. 11. 2023.

Zadatak zadao: Izv. prof. dr. sc. Tomislav Stipančić

Datum predaje rada:
**1. rok:** 22. i 23. 2. 2024.
**2. rok (izvanredni):** 11. 7. 2024.
**3. rok:** 19. i 20. 9. 2024.

Predviđeni datumi obrane:
**1. rok:** 26. 2. – 1. 3. 2024.
**2. rok (izvanredni):** 15. 7. 2024.
**3. rok:** 23. 9. – 27. 9. 2024.

Predsjednik Povjerenstva: Prof. dr. sc. Damir Godec

# SADRŽAJ

# POPIS SLIKA

# SAŽETAK

U području umjetne inteligencije, posebno se ističe računalni vid, gdje se duboko učenje koristi kako bi se omogućilo računalnim sustavima razumijevanje vizualnih informacija. Konvolucijske neuronske mreže igraju ključnu ulogu u ovom kontekstu, omogućujući sustavima analizu slika i videa te izvršavanje različitih zadataka poput prepoznavanja objekata, klasifikacije elemenata i praćenja kretanja unutar scene.

U radu je napravljen teorijski pregled dubokog učenja, konvolucijskih neuronskih mreža i računalnog vida. Također se programskim paketom Python, odrađuje praktični dio generiranja podataka, izrada modela konvolucijske neuronske mreže i implementiranje na stvarni vizijski sustav.

Ključne riječi: duboko učenje, računalni vid, Python, konvolucijske neuronske mreže, klasifikacija, detekcija objekata

# SUMMARY

In the field of artificial intelligence, computer vision stands out, where deep learning is used to enable computer systems to understand visual information. Convolutional neural networks play a key role in this context, allowing systems to analyze images and videos and perform various tasks such as object recognition, element classification, and motion tracking within a scene.

The paper provides a theoretical overview of deep learning, convolutional neural networks, and computer vision. Additionally, the practical part involves data generation, building a convolutional neural network model, and implementing it on a real vision system using the Python programming package.

Key words: deep learning, computer vision, Python, convolutional neural networks, classification, object detection

# 1. UVOD

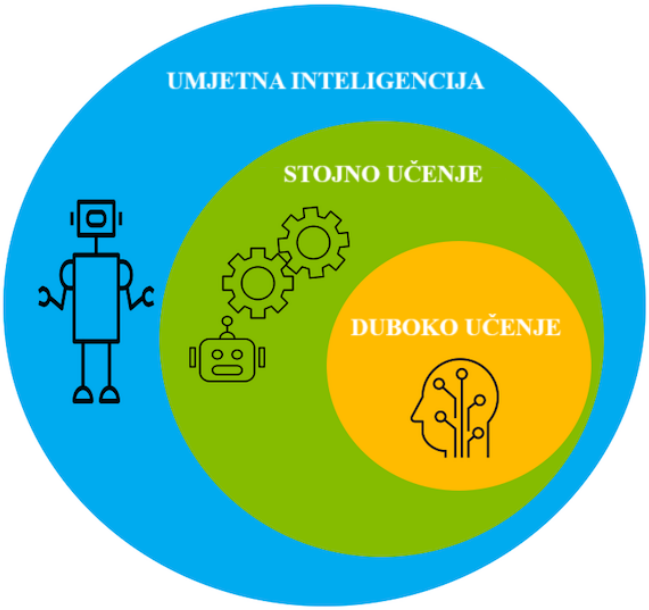
Umjetna inteligencija predstavlja fascinantan segment tehnologije koji se kontinuirano razvija s ciljem da omogući računalima da izvode zadatke koji zahtijevaju ljudsku inteligenciju. Jedna od najmoćnijih grana umjetne inteligencije je duboko učenje, koje se temelji na modelima umjetnih neuronskih mreža inspiriranih biološkim neuronima. Ova tehnika omogućuje računalima da nauče složene obrasce i reprezentacije iz velikih skupova podataka, što je dovelo do revolucije u različitim područjima, uključujući strojni i računalni vid.

Računalni vid je područje umjetne inteligencije koje se bavi razumijevanjem vizualnih informacija. Konvolucijske neuronske mreže su ključni alat u računalnom vidu koji omogućuje računalima da automatski izdvoje značajke iz slika, što je esencijalno za zadatke kao što su klasifikacija, detekcija objekata i segmentacija slika.

Tema ovog rada fokusirana je na primjenu konvolucijskih neuronskih mreža za klasifikaciju objekata prema boji i obliku. Kroz pristup dubokog učenja, istražujemo kako konvolucijska neuronska mreža može naučiti prepoznavati obrasce boja i oblika te ih klasificirati s visokom točnošću. Ovakva primjena ne samo da demonstrira moć dubokog učenja u domeni strojnog vida, već također ima praktičnu primjenu u različitim područjima kao što su računalni sustavi za nadzor, robotika i obrada slika.

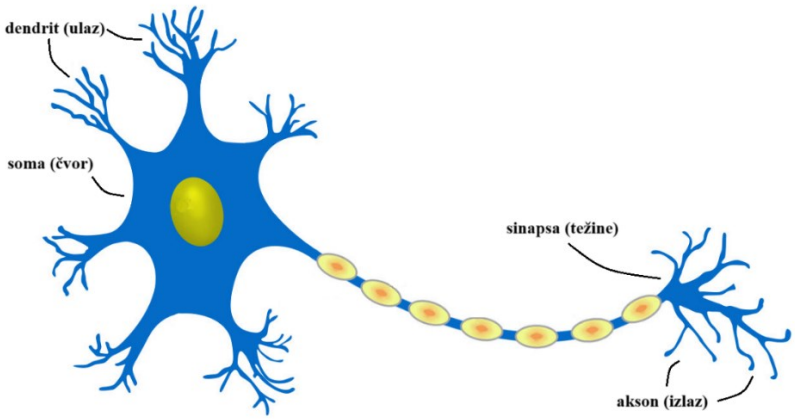
# 2. DUBOKO UČENJE

Duboko učenje je podskup strojnog učenja, ono pokreće mnoge aplikacije umjetne inteligencije koje poboljšavaju automatizaciju, obavljajući analitičke i fizičke zadatke bez ljudske intervencije.



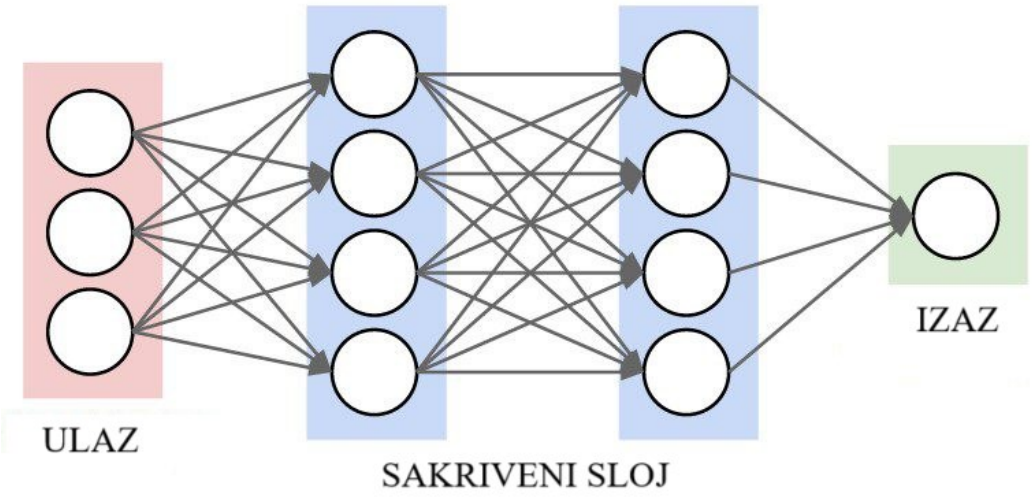

**Slika 1. Podgrupe umjetne inteligencije**

Duboko učenje je u osnovi neuronska mreža s tri ili više sloja. Te neuronske mreže pokušavaju simulirati ponašanje ljudskog mozga (iako su daleko od podudaranja s njegovom sposobnošću) omogućujući mu da "uči" iz velikih količina podataka. Iako neuronska mreža s jednim slojem i dalje može praviti približne prognoze, dodatni skriveni slojevi mogu pomoći u optimizaciji i poboljšanju preciznosti.[1]




**Slika 2. Biološki neuron**

Neuronske mreže sastoje se od više slojeva međusobno povezanih čvorova, pri čemu svaki sloj nadograđuje prethodni sloj kako bi fino podešavao i optimizirao predikciju ili kategorizaciju. Ovaj napredak računanja kroz mrežu naziva se progresija prema naprijed. Ulazni i izlazni slojevi duboke neuronske mreže nazivaju se vidljivi slojevi. Ulazni sloj je mjesto gdje duboki model učenja prima podatke za obradu, dok je izlazni sloj mjesto gdje se konačna predikcija ili klasifikacija obavlja.




**Slika 3. Umjetni neuron**

Drugi proces nazvan povratna propagacija koristi algoritme poput gradijentnog spusta kako bi izračunao pogreške u predikcijama, a zatim prilagodio težine i pristranost funkcije pomicanjem unatrag kroz slojeve u nastojanju da obuči model. Zajedno, progresija prema naprijed i povratna propagacija omogućuju neuronskoj mreži da donosi predikcije i ispravlja bilo kakve pogreške. Tijekom vremena, algoritam postaje postupno precizniji.

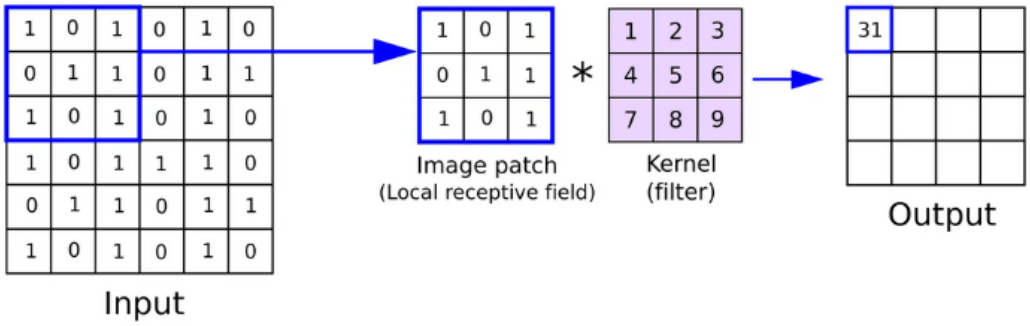
Postoje različite vrste neuronskih mreža koje se koriste za rješavanje određenih problema ili skupova podataka. Na primjer, Konvolucijske neuronske mreže, korištene prvenstveno u aplikacijama računalnog vida i klasifikaciji slika, mogu detektirati značajke i obrasce unutar slike, omogućujući rješavanje zadatka poput detekcije ili prepoznavanja objekata. [2]

# 3. KONVOLUCIJSKE NEURONSKE MREŽE

Konvolucijske neuronske mreže (*eng. convolutional neural networks*), odnosno CNN, su vrsta neuronskih mreža koja se specijalizira u procesiranju podataka kao što su slike.

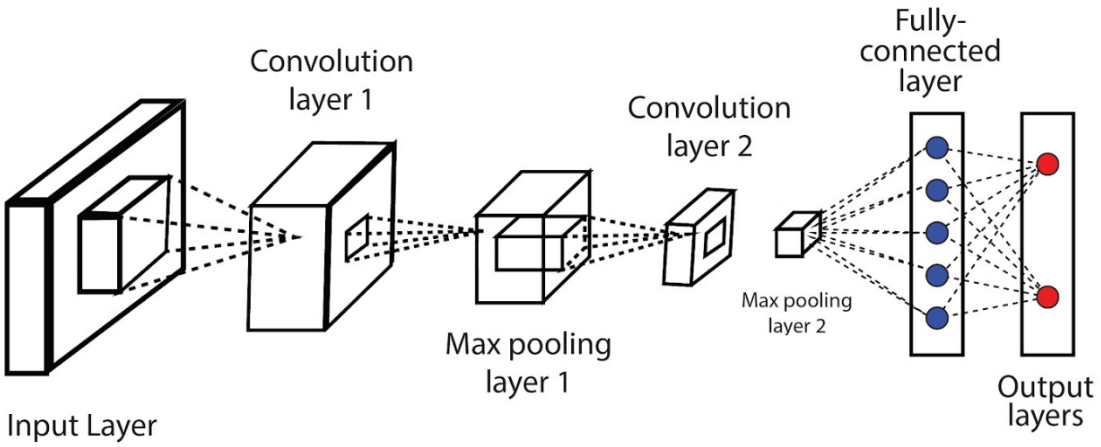
U konvolucijskoj neuronskoj mreži (CNN), konvolucije su osnovne matematičke operacije koje stvaraju skup težina nazvan kernel ili filter. Ovaj filter, manji od cijele ulazne slike, se pomiče po slici, a vrijednosti unutar filtera množe se s vrijednostima na slici. Proces stvara "kartu značajki", dvodimenzionalni niz koji predstavlja cijelu sliku.

Proces stvaranja karte značajki prikazan je na slici 4.




**Slika 4. Konvolucija**

Općenito, konvolucijska neuronska mreža se sastoji od konvolucijskog sloja, sloja sažimanja te potpuno povezanih slojeva. Na slici 5 je prikazana arhitektura konvolucijske neuronske mreže sa svim njezinim dijelovima.



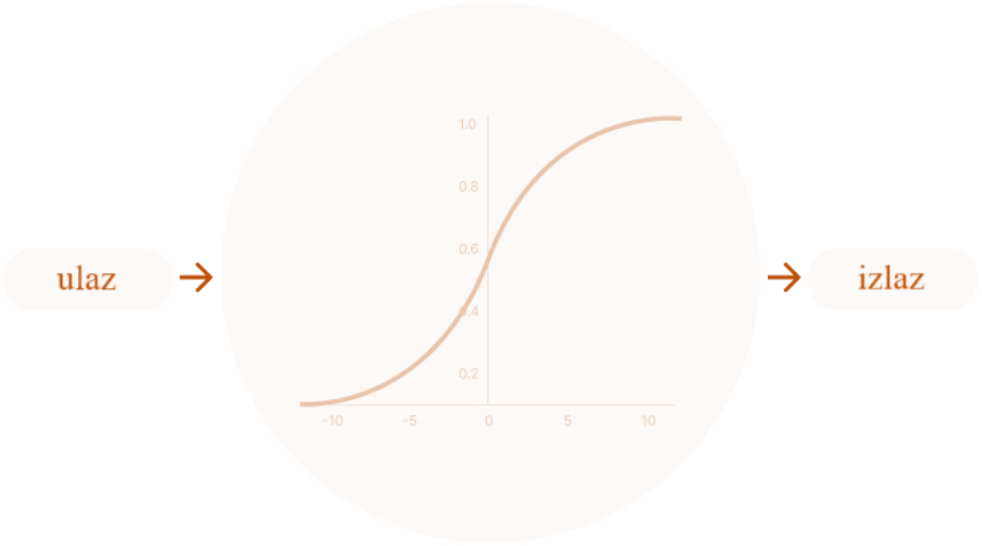

**Slika 5. Konvolucijska neuronska mreža**

## 3.1. Konvolucijski sloj

Konvolucijski sloj pretvara slikovne podatke u numeričke nizove. Ovi slojevi mogu se slagati jedan na drugi, omogućujući višestruke konvolucije i grupiranje značajki slike. Rani konvolucijski slojevi izdvajaju osnovne značajke poput linija, dok kasniji slojevi spajaju te značajke u složenije obrasce poput životinja ili ljudskih lica.[2]

## 3.2. Aktivacijske funkcije

Aktivacijska funkcija odlučuje treba li se neuron aktivirati na temelju svojeg ulaza, pomažući u predikciji putem matematičkih operacija. Njezina glavna svrha je transformirati ulazne vrijednosti u izlaz, olakšavajući protok informacija kroz slojeve u neuronskoj mreži. Zapravo one dodaju nelinearnost neuronskoj mreži kako bi učenje složenog zadatka bilo moguće.




**Slika 6. Aktivacijske funkcije**

### 3.2.1. Linearna funkcija

Linearna aktivacijska funkcija je funkcija gdje je aktivacija proporcionalna ulazu. Ona ne mijenja ništa na ulazu, jednostavno vraća vrijednost koju je dobila. Problem je da će se svi slojevi neuronske mreže sažeti u jedan, bez obzira na broj slojeva u neuronskoj mreži (zadnji sloj će biti linearna funkcija prvog sloja).[3]

Matematička reprezentacija linearne aktivacijske funkcije (1)

$$f(x) = x \tag{1}$$

**Slika 7. Linearna funkcija**

### 3.2.2. Sigmoid

Sigmoidna aktivacijska funkcija uzima bilo koju stvarnu vrijednost kao ulaz, a daje izlazne vrijednosti između 1 i 0. Što je veći ulaz (pozitivniji), to će izlazna vrijednost biti bliža 1.0, dok će za što manji ulaz (negativniji), izlaz biti bliži 0.0. Funkcija sprječava skokove u izlaznim vrijednostima, ali kako se približava nuli, mreža prestaje učiti što je još poznato kao problem nestajućeg gradijenta. Izraz funkcije nije simetričan oko nule, dakle izlaz svih neurona bit će isti što čini treniranje neuronske mreže težim i nestabilnim. [3]

Matematička reprezentacija Sigmoid aktivacijske funkcije (2)

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2}$$




**Slika 8. Sigmoid**

### 3.2.3. Tanh

Tanh aktivacijska funkcija je vrlo slična sigmoidnoj, ali s razlikom u rasponu izlaznih vrijednosti. Što je veći ulaz (pozitivniji), to će izlazna vrijednost biti bliža 1.0, dok će za što manji ulaz (negativniji), izlaz biti bliži -1.0. Funkcija je centrirana oko nule, stoga možemo lako mapirati izlazne vrijednosti kao jako negativne, neutralne ili jako pozitivne. Tanh funkcija se također suočava s problemom nestajućih gradijenata. [3]

Matematička reprezentacija Tanh aktivacijske funkcije (3)

$$f(x) = \frac{(e^{x} - e^{-x})}{(e^{x} + e^{-x})} \tag{3}$$




**Slika 9. Tanh**

### 3.2.4. ReLU

ReLU (*eng. Rectified Linear Unit*) aktivacijska funkcija ne aktivira sve neurone istovremeno. Neuroni će biti deaktivirani samo ako je izlaz linearne transformacije manji od 0. Budući da su aktivirani samo određeni broj neurona, ReLU funkcija je znatno računalno učinkovitija u usporedbi sa sigmoid i TanH funkcijama.[3] ReLU može biti osjetljiv tijekom treniranja jer ga veliki gradijent koji prolazi kroz njega može ažurirati na tako da neuron više nikada neće biti dalje ažuriran.

Matematička reprezentacija ReLU aktivacijske funkcije (4)

$$f(x) = \max(0, x) \tag{4}$$

**Slika 10. ReLU**

### 3.2.5. Softmax

Softmax aktivacijska funkcija se opisuje kao kombinacija više sigmoida. Ona izračunava relativne vrijednosti i vraća vjerojatnost svake klase. Najčešće se koristi kao aktivacijska funkcija za posljednji sloj neuronske mreže u slučaju višeklasne klasifikacije.[3]

Matematička reprezentacija Softmax aktivacijske funkcije (5)

$$f(x) = \frac{e^{x_i}}{\sum_j e^{x_j}} \tag{5}$$



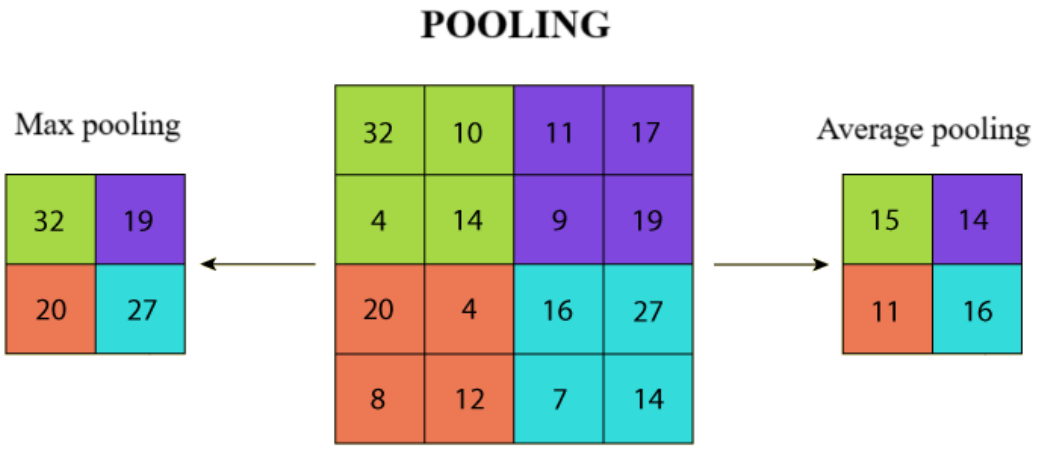

**Slika 11. Softmax**

### 3.3. Potpuno povezani sloj

Neuroni u ovom sloju imaju potpunu povezanost sa svim neuronima u prethodnom i narednom sloju. Potpuno povezani sloj pomaže mapirati reprezentaciju između ulaza i izlaza.

### 3.4. Sloj sažimanja

Sloj sažimanja zamjenjuje izlaz mreže na određenim lokacijama izvlačeći sažetak statistike bliskih izlaza. To pomaže u smanjenju prostornih dimenzija, što smanjuje potrebnu količinu računanja. Operacija grupiranja se obrađuje na svakom rezu pojedinačno. Najpopularniji proces grupiranja je maksimalno grupiranje, koje daje maksimalni izlaz iz susjedstva.[2]




**Slika 12. Sloj sažimanja**

# 4. MODEL KONVOLUCIJSKE NEURONSKE MREŽE

Praktični dio zadatka odnosi se na konstruiranje modela konvolucijske neuronske mreže, generiranju podataka za učenje i prepoznavanje objekata s pomoću vizijskog sustava. Za to je korišten programski jezik Python s određenim bibliotekama.

## 4.1. Python

Python je jedan od najpopularnijih programskih jezika. Ima jednostavnu sintaksu koja omogućuje programerima da pišu programe s manje linija koda nego kod ostalih programskih jezika. Isto tako kod se može izvesti čim se napiše, što omogućuje brzu izradu i testiranje prototipa. [4] U nastavku su opisane neke korištene biblioteke.

**Slika 13. Python logo**

### 4.1.1. TensorFlow

TensorFlow je besplatna, open-source biblioteka za strojno učenje i umjetnu inteligenciju. Može se koristiti za različite zadatke, ali je posebno usredotočena na treniranje dubokih neuronskih mreža. [5]

### 4.1.2. OpenCV

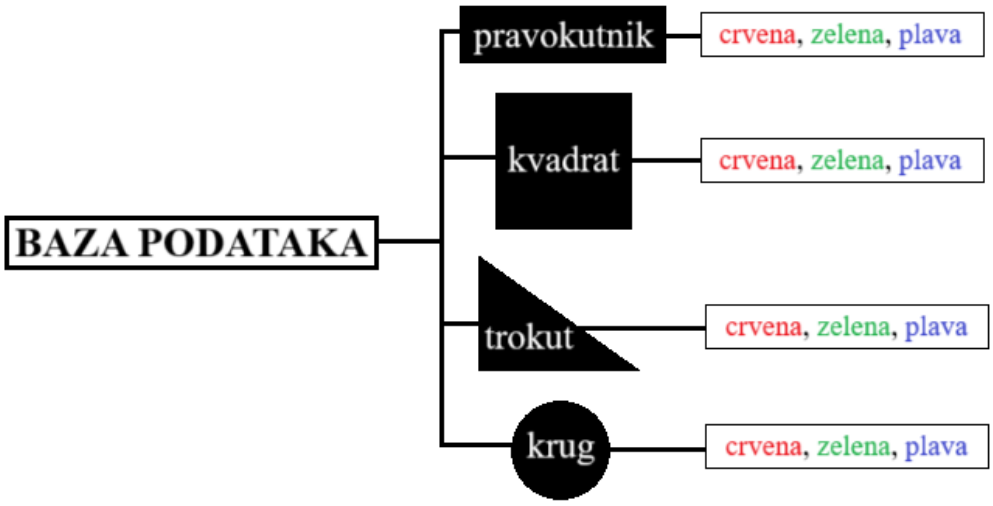
OpenCV (eng. *Open Source Computer Vision*) je besplatna biblioteka za računalni vid koja je prvotno razvijena od strane tvrtke Intel. Danas je OpenCV postao standardni alat za sve što se tiče računalnog vida. Ova biblioteka nudi više od 2500 algoritama, obimnu dokumentaciju, izvorni kod i primjere za obradu slika u stvarnom vremenu. [6]

### 4.1.3. NumPy

NumPy (eng. *Numerical Python*) je temeljna biblioteka za znanstveno računanje s Pythonom. Ona dodaje podršku Pythonu za velike, višedimenzionalne nizove i matrice, s velikom kolekcijom visoko razinskih matematičkih funkcija za rad s tim nizovima. [7]

## 4.2. Baza podataka

Prije izrade modela konvolucijske neuronske mreže, mora se pripremiti baza podataka. Baza podataka je zapravo skup slika koje konvolucijska neuronska mreža koristi za učenje. Sastoji se od 12 mapa. Mape su sve moguće kombinacije oblika i boje. Klase u ovom skupu su: krug, kvadrat, pravokutnik, trokut i boje: plava, zelena, crvena.




**Slika 14. Struktura baze podataka**

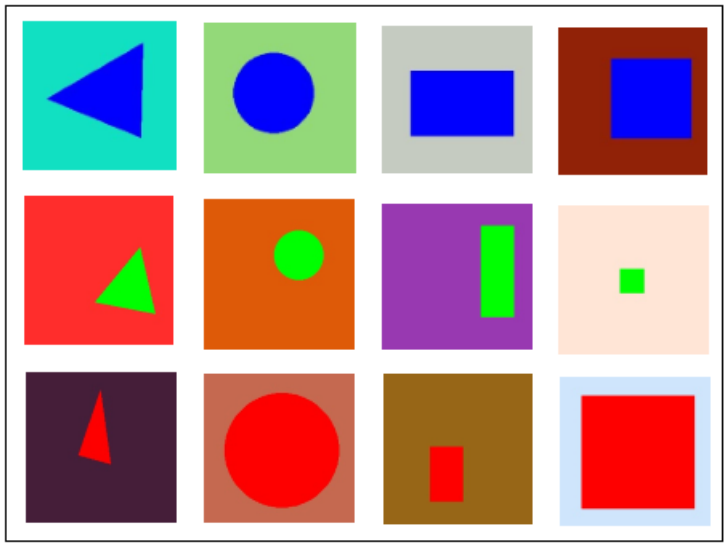
### 4.2.1. Generiranje podataka

Generiranje podataka odvija se s pomoću funkcije *'generate_shape'*. Funkcija uzima parametre oblika, veličinu slike i nasumično bira boju specifično za svaki oblik;

- za krug nasumično uzima središte i radijus kružnice pa popuni oblik,
- za kvadrat nasumično uzima dužinu stranice i odredi gornji lijevi kut pa popuni,
- za pravokutnik nasumično uzima visinu, širinu i gornji lijevi kut pa popuni,
- za trokut odredi minimalni površinu tako da nasumično odredi 3 točke u zamišljenoj granici pa popuni.

Boja oblika i pozadine se uvijek razlikuju radi lakšeg učenja konvolucijske neuronske mreže. Generiranje podataka se izvodi u 5000 uzoraka po klasi, što za ukupno 12 klasa (4 klase oblika i svaka 3 klase boja) iznosi 60000 slika.

Na slici 15 prikazano jer par primjera slika iz baze podataka.



**Slika 15. Generirane slike**

## 4.3. Procesiranje podataka

Prije nego se krene s učenjem, podatci se moraju procesirati. Uz pomoć *tf.keras* generirani podatci (slike) se učitavaju u model. Redoslijed slika se prilikom učitavanja miješa, te se njihova veličina podešava na 64x64 piksela. Generirana baza se dijeli na 2 dijela. Dio za treniranje koji iznosi 40 % od ukupnih slika (24000 slika) i dio za testiranje koji iznosi 60 % (36000 slika). Prije nastavka, veličina Batcha se podešava na 32 (to je broj uzoraka koji se koristi za jedan prolaz kroz mrežu, dakle 32 uzoraka po prolasku).

## 4.4. Arhitektura modela

Model je sekvencijalni model, što znači da je to linearni skup slojeva. Sastoji se od nekoliko slojeva uključujući 4 konvolucijska i 4 potpuno povezana slojeva i 1 sloj sažimanja za klasifikaciju. Svaki sloj transformira ulazne podatke u određeni oblik izlaza. Broj parametara (težina) varira za svaki sloj i označen je s Param. Ukupno, model ima 653,900 parametara, od kojih su svi obučivi. To uključuje parametre koji će se ažurirati tijekom obuke (Trainable params), ukupno 653,900. Nema parametara koji se ne treniraju, što znači da će svi biti prilagođeni tijekom procesa obuke. Na slici 16 je prikazana arhitektura modela konvolucijske neuronske mreže zajedno s njezinim parametrima u svakom sloju.

| Layer (type) | Output Shape | Param # |
|---|---|---|
| conv2d (Conv2D) | (None, 62, 62, 32) | 896 |
| max_pooling2d (MaxPooling2D) | (None, 31, 31, 32) | 0 |
| conv2d_1 (Conv2D) | (None, 29, 29, 64) | 18496 |
| max_pooling2d_1 (MaxPooling2D) | (None, 14, 14, 64) | 0 |
| conv2d_2 (Conv2D) | (None, 12, 12, 128) | 73856 |
| max_pooling2d_2 (MaxPooling2D) | (None, 6, 6, 128) | 0 |
| conv2d_3 (Conv2D) | (None, 4, 4, 256) | 295168 |
| max_pooling2d_3 (MaxPooling2D | (None, 2, 2, 256) | 0 |
| flatten (Flatten) | (None, 1024) | 0 |
| dropout (Dropout) | (None, 1024) | 0 |
| dense (Dense) | (None, 256) | 262400 |
| dense_1 (Dense) | (None, 12) | 3084 |

Total params: 653,900

Trainable params: 653,900

Non-trainable params: 0

**Slika 16. Arhitektura modela**

### 4.4.1. Konvolucijski slojevi

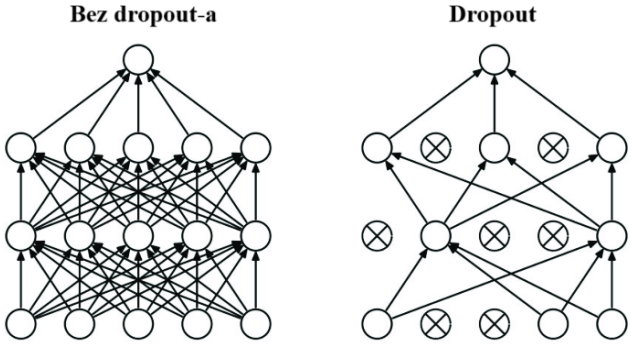
Konvolucijski sloj se dodaje uz pomoć naredbe *Conv2D*. Definira se broj filtera. Filtri služe za otkrivanje uzoraka poput rubova ili tekstura. Prvi konvolucijski sloj sastoji se od 32, drugi od 64, treći od 128 i četvrti od 256 filtra. Za svaki konvolucijski sloj koristi se ReLu aktivacijska funkcija.

### 4.4.2. Potpuno povezani slojevi i sloj sažimanja

Potpuno povezani slojevi dodaju se nakon svakog Konvolucijskog sloja uz pomoć naredbe *MaxPooling2D*. Sloj sažimanja dodaje se uz pomoć naredbe *Flatten*. Dok se kao aktivacijska funkcija sloja sažimanja koristi Softmax.

### 4.4.3. Dropout

Dropout je tehnika regularizacije koja uključuje nasumično postavljanje dijela neurona modela na nulu tijekom treninga. To pomaže u sprječavanju prevelike ovisnosti modela o bilo kojem neuronu, što može pomoći u smanjenju prenaučenosti. [8]




**Slika 17. Dropout**

## 4.5. Optimiziranje modela

Model je optimiziran uz pomoć Adam algoritma. Adam je optimizacijski algoritam koji se koristi umjesto klasične stohastičke gradijentne metode za iterativno ažuriranje težina mreže na temelju podataka za obučavanje. Ta metoda je računalno učinkovita, ima malo zahtjeva za memorijom i dobro je prilagođena problemima koji su veliki (u smislu velikog broja podataka i parametara).[9]

## 4.6. Treniranje modela

Za treniranje modela korištena je naredba *model.fit*. Ona u sebe uzima parametar broj epoha. Broj epoha definira koliko će puta algoritam učenja proći preko baze podataka za treniranje. Za potrebe ovog zadatka, uzima se 10 epoha. Prilikom treniranja se uz pomoć naredbe *model.evaluate* za svaku epohu prikazuju; vrijeme trajanja epohe, gubitak i preciznost kod treniranja, te gubitak i preciznost kod testiranja. Na slici 18 prikazane su epohe učenja neuronske mreže. Vidi se vrijeme trajanje učenja svake epohe i vrijeme po svakom koraku. Također su prikazani gubitci i preciznost predviđanja kod podataka za treniranje (*loss*, accuracy) kao i gubitci i preciznost predviđanja kod podatka za testiranje koje mreža još nikad nije „vidjela“ (*val_loss*, *val_accuracy*). Iz slike vidimo kako se sa svakom sljedećom epohom, gubitci smanjuju, a preciznost povećava što ukazuje na dobro i ispravno funkcioniranje učenja

modela konvolucijske neuronske mreže. Na kraju se vidi kako je preciznost klasifikacije kod podataka za testiranje ≈ 93 % što pokazuje na odličnu istreniranost mreže.

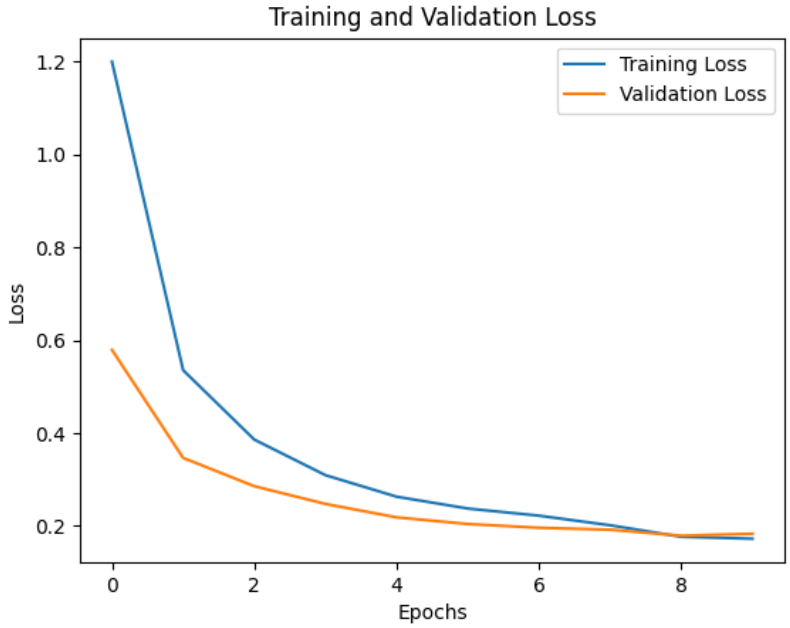
```
Epoch 1/10
750/750 [==============================] - 201s 260ms/step - loss: 1.2001 - accuracy: 0.5615 - val_loss: 0.5794 - val_accuracy: 0.7594
Epoch 2/10
750/750 [==============================] - 181s 241ms/step - loss: 0.5357 - accuracy: 0.7691 - val_loss: 0.3468 - val_accuracy: 0.8529
Epoch 3/10
750/750 [==============================] - 124s 165ms/step - loss: 0.3862 - accuracy: 0.8354 - val_loss: 0.2858 - val_accuracy: 0.8775
Epoch 4/10
750/750 [==============================] - 125s 167ms/step - loss: 0.3096 - accuracy: 0.8684 - val_loss: 0.2476 - val_accuracy: 0.8985
Epoch 5/10
750/750 [==============================] - 129s 172ms/step - loss: 0.2632 - accuracy: 0.8890 - val_loss: 0.2187 - val_accuracy: 0.9107
Epoch 6/10
750/750 [==============================] - 116s 155ms/step - loss: 0.2377 - accuracy: 0.9005 - val_loss: 0.2044 - val_accuracy: 0.9161
Epoch 7/10
750/750 [==============================] - 123s 164ms/step - loss: 0.2224 - accuracy: 0.9081 - val_loss: 0.1963 - val_accuracy: 0.9219
Epoch 8/10
750/750 [==============================] - 118s 158ms/step - loss: 0.2017 - accuracy: 0.9182 - val_loss: 0.1921 - val_accuracy: 0.9236
Epoch 9/10
750/750 [==============================] - 118s 157ms/step - loss: 0.1769 - accuracy: 0.9271 - val_loss: 0.1792 - val_accuracy: 0.9283
Epoch 10/10
750/750 [==============================] - 119s 158ms/step - loss: 0.1727 - accuracy: 0.9317 - val_loss: 0.1834 - val_accuracy: 0.9262
1125/1125 [==============================] - 69s 62ms/step - loss: 0.1834 - accuracy: 0.9262
Validation Loss: 0.183426633477211
Validation Accuracy: 0.926194429397583
```

**Slika 18. Treniranje modela**

## 4.7. Evaluacijske metrike

Evaluiranje algoritma strojnog učenja je bitan dio svakog projekta. U ovom radu koristile su se sljedeće evaluacijske metrike; dijagram gubitka, dijagram preciznosti i matrica konfuzije.

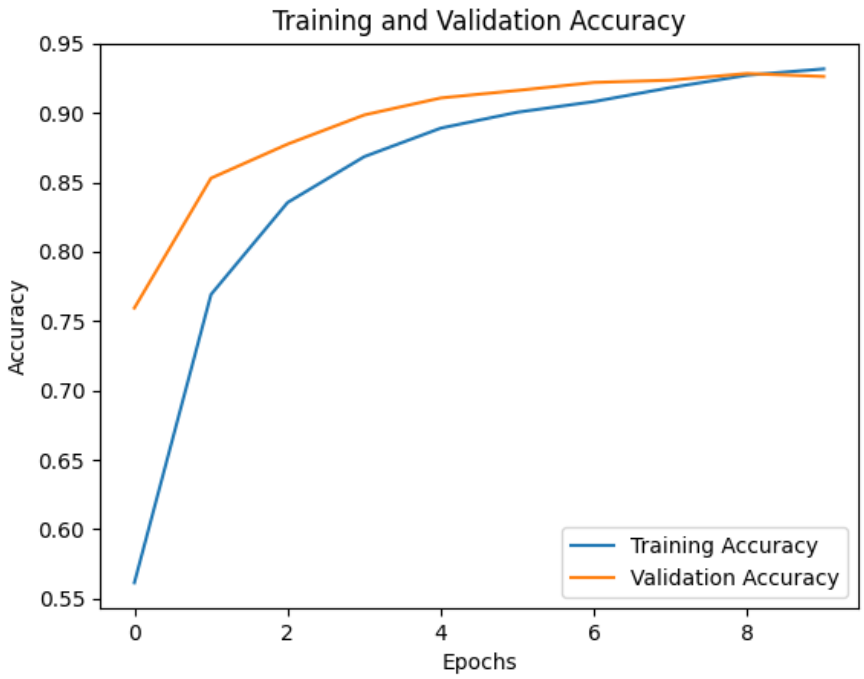
Dijagram gubitaka prikazuje kako su se gubitci na podatcima za treniranje i na podatcima za testiranje mijenjali kroz svaku epohu, što je prikazano na slici 19.




**Slika 19. Dijagram gubitka**

Iz slike je vidljivo kako se gubitci na podacima za treniranje (*loss*) približavaju nuli kako epohe prolaze. Gubitci na podatcima za testiranje (*val_loss*) dobro prate liniju loss što ukazuje na dobar rad mreže odnosno učenja.

Dijagram preciznosti prikazuje se preciznost pogađanja oblika i boje na podatcima za treniranje i na podatcima za testiranje mijenja kroz svaku epohu, što je prikazano na slici 20.

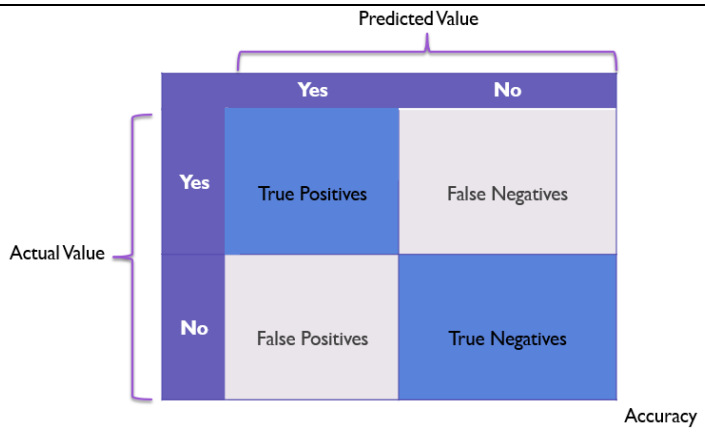



**Slika 20. Dijagram preciznosti**

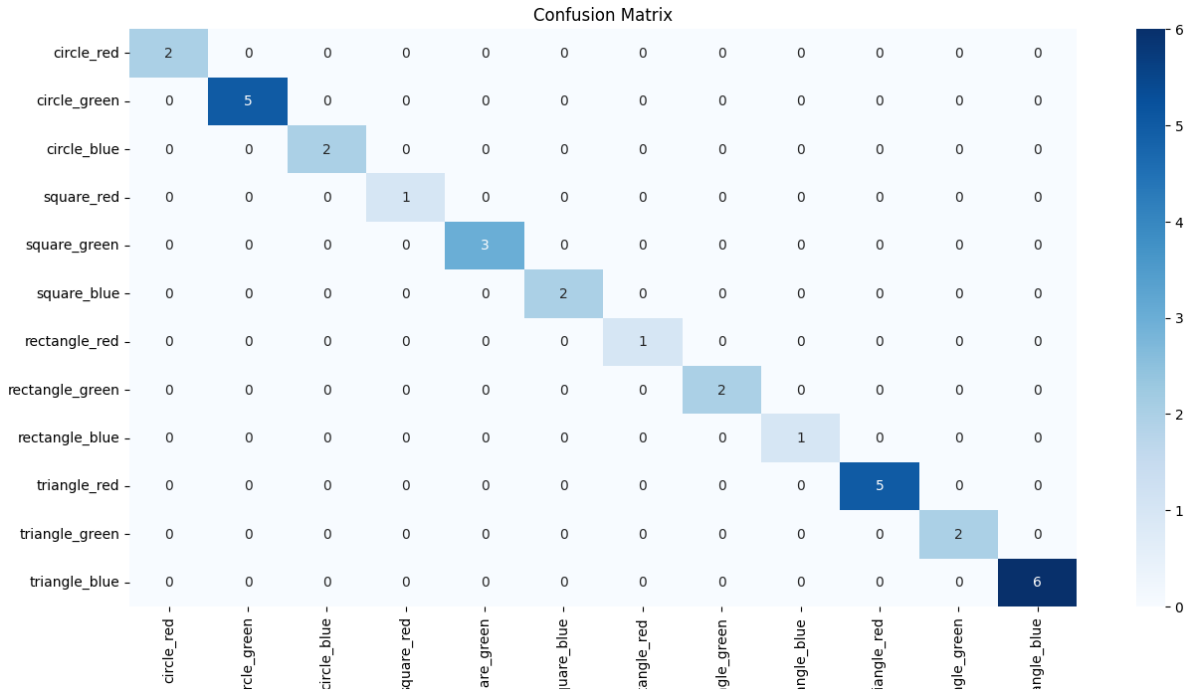
Iz slike je vidljivo kako preciznost predviđanja na podatcima za treniranje (*accuracy*) raste kako epohe prolaze i to do ≈ 93 %. Preciznost predviđanja na podatcima za testiranje (*val_accuracy*) dobro prati liniju accuracy što također ukazuje na dobar rad mreže.

### 4.7.1. Matrica konfuzije

Matrica konfuzije je matrica koja sažima performanse modela strojnog učenja na skupu testnih podataka. To je način prikazivanja broja točnih i netočnih instanci na temelju predikcija modela. Često se koristi za mjerenje performansi klasifikacijskih modela, koji ciljaju predvidjeti kategoričku oznaku za svaku ulaznu instancu. Matrica prikazuje broj instanci koje je model proizveo na testnim podacima.[10]




**Slika 21. Matrica konfuzije – općenito**

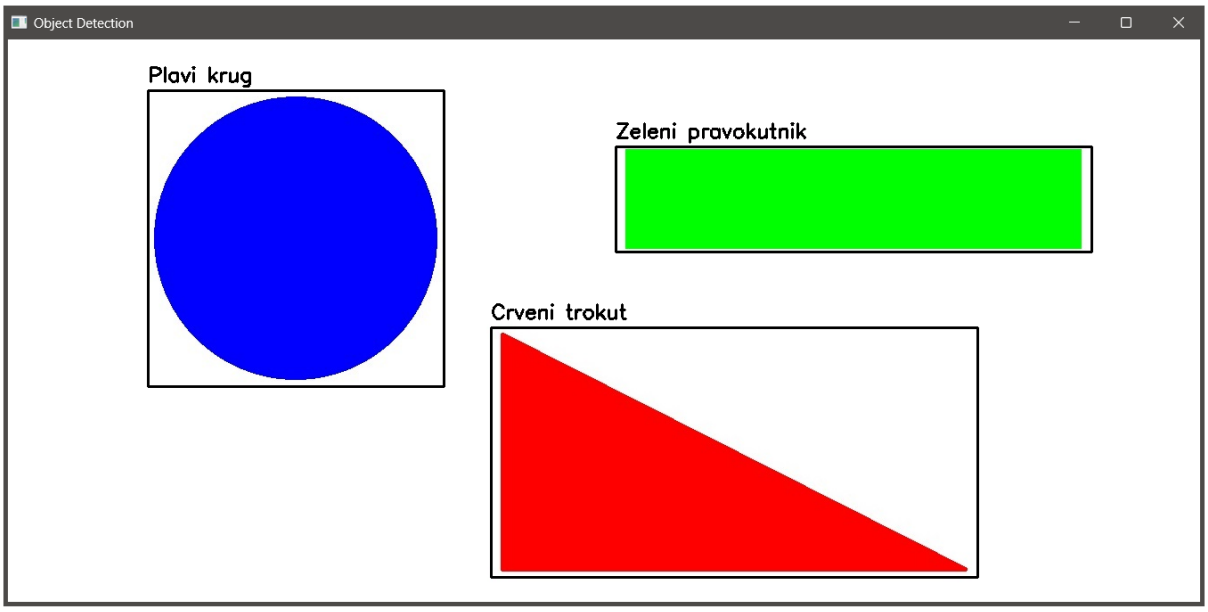



**Slika 22. Matrica konfuzije**

U ovome radu matrica konfuzija je matrica 12x12. Izgled matrice konfuzije prikazan je na slici 22. U matrici konfuzije je vidljivo kako su postignuti vrlo dobri rezultati klasifikacije konvolucijske neuronske mreže. Visoke vrijednosti duž dijagonale upućuju na snažnu prisutnost pravih pozitiva i pravih negativa što znači da model pokazuje izvrsnu točnost i preciznost u skoro svim klasama.

## 4.8. Testiranje modela

Prije implementacije konvolucijske neuronske mreže na vizijski sustav, mora se provjerit ispravnost klasifikacije pomoću testiranja. Testiranje se provodi na slici s više objekata u različitim bojama. Time dodatno provjeravamo ispravnost rada pošto su pri učenju bile samo slike sa jednim objektom po slici.




**Slika 23. Rezultati testiranja**

Na slici za testiranje se nalaze 3 različita oblika, različitih boja. To je slika koju mreža prvi put vidi. Objekti se prepoznaju uz pomoć Canny algoritma i kreiranja regija interesa (eng. *Region of interest, ROI*) što je detaljnije opisano kasnije u radu. Iz slike vidimo kako je mreža točno klasificirala sva 3 objekta te time potvrđujemo uspješnost učenja odnosno ispravnost rada mreže.

# 5. IMPLEMENTACIJA NA VIZIJSKOM SUSTAVU

## 5.1. Računalni vid

Računalni vid je grana umjetne inteligencije koja omogućuje računalima i sustavima da analiziraju digitalne slike, videozapise i druge vizualne ulaze kako bi iz njih izvukli važne informacije. Na temelju tih informacija, ti sustavi mogu donositi odluke ili davati preporuke. Računalni vid trenira strojeve da obavljaju funkcije koje moraju učiniti puno brže od ljudi, koristeći kamere, podatke i algoritme umjesto mrežnice, optičkih živaca i vizualnog korteksa. Računalni vid koristi se u industrijama koje se protežu od energetike i javnih usluga do proizvodnje i automobilske industrije. [11]


**Slika 24. Računalni vid**

## 5.2. Vizijski sustav u labosu

Model konvolucijske neuronske mreže implementiran je u postojeći vizijski sustav u sklopu laboratorija za projektiranje izradbenih i montažnih sustava. Postojeći vizijski sustav sastoji se od kamere Logitech StreamCam i postolja. Kamera snima i prijenosi video u full HD 1080p rezoluciji na 60 fps (eng. *frames per second*).

**Slika 25. Logitech StreamCam**


**Slika 26. Vizijski sustav u labosu**

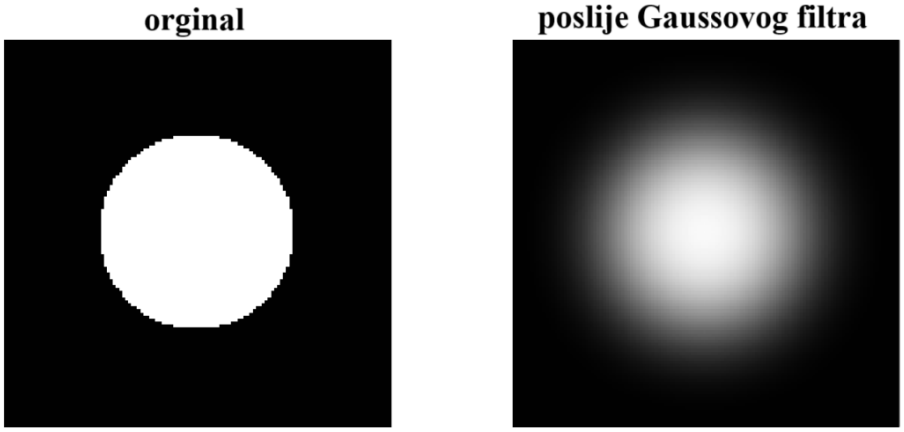
## 5.3. Način rada

Program za detekciju objekata napravljen je uz pomoć Canny algoritma za prepoznavanje rubova. Slika se prebaci u sive nijanse koristeći OpenCV te se provede Canny algoritam. Pronalaženje rubova na slici odvija se uz pomoć funkcije *cv.findCountours()*. To se iterira po svakom detektiranom rubu i filtrira na temelju ukupne površine oblika. Oko svakog detektiranog oblika crta se pravokutnik koji označuje taj oblik čime se stvaraju tzv. regije interesa (eng. *Region of interest, ROI*). Svaki ROI se izvlači iz slike na temelju koordinata pravokutnika koji ga okružuje te on predstavlja jednu zasebnu sliku. Svaka ta slika se provlači kroz prije trenirani model konvolucijske neuronske mreže i tako provodi klasifikacija oblika prema boji i obliku.

### 5.3.1. Canny algoritam

Canny detekcija rubova popularni je višestupanjski algoritam za detekciju rubova. Sastoji se od sljedećih koraka[12];

1. Smanjenje šuma: budući da je detekcija rubova osjetljiva na šum u slici, prvi korak je uklanjanje šuma u slici uz pomoć Gaussovog filtra.



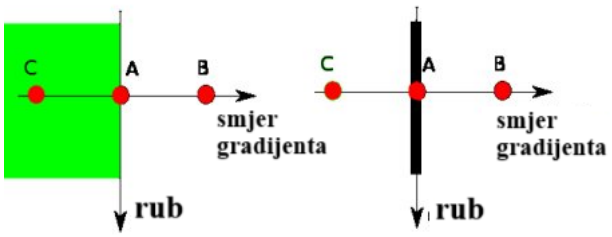

**Slika 27. Smanjenje šuma**

2. Pronalaženje intenziteta gradijenta slike: slika se dalje filtrira Sobelovim filtrom u horizontalnom i vertikalnom smjeru kako bi se dobile derivacije $G_x$ i $G_y$. Iz te dvije slike nalazimo rubove i smjer za svaki pixel. Smjer gradijenta uvijek je okomit na rubove i zaokružuje se na jedan od četiri kuta koji predstavljaju vertikalne, horizontalne i dijagonalne smjerove.

$$\text{Rubovi (G)} = \sqrt{G_x^2 + G_y^2}$$

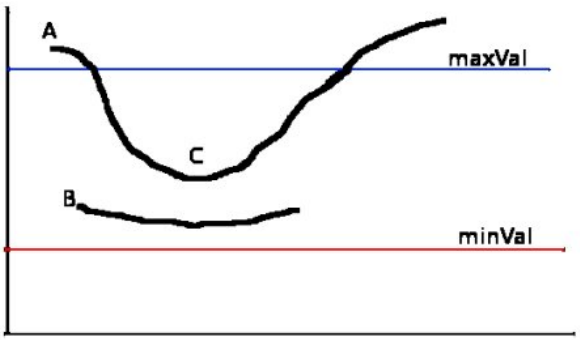
$$\text{Kut } (\theta) = tan^{-1}\left(\frac{G_y}{G_x}\right)$$

3. Supresija nepropusnosti maksimuma: provodi se pregledavanje cijele slike te se na svakom pikselu provjerava je li piksel lokalni maksimum u svom susjedstvu. Ako nije lokalni maksimum on se potiskuje ili uklanja.

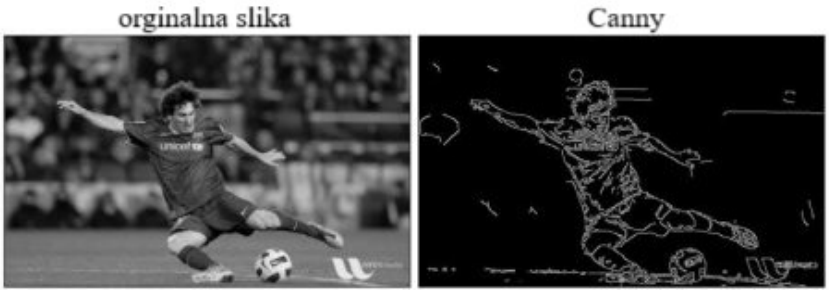



**Slika 28. Supresija**

4. Prag histereze: ova faza odlučuje koji su rubovi stvarni rubovi, a koji nisu. Za to se koriste dvije vrijednosti; minimalna (minVal) i maksimalna (maxVal). Svi rubovi s intenzitetom gradijenta većim od maksimalne vrijednosti sigurno su rubovi, dok oni ispod minimalne vrijednosti sigurno nisu rubovi i odbacuju se. Za one koji su između, gleda se kako su povezani. Ako su povezani s pikselima 'sigurnih rubova' smatraju se dijelom ruba, a ako nisu se odbacuju.




**Slika 29. Prag histereze**



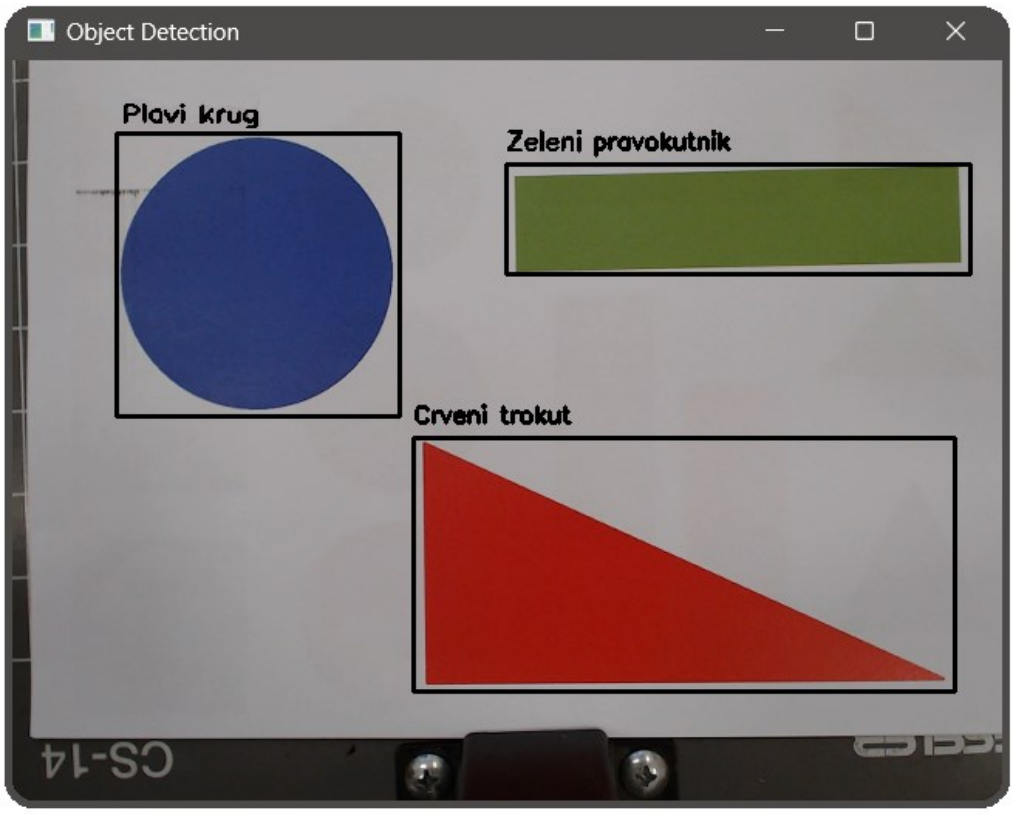

**Slika 30. Canny**

### 5.3.2. Regija interesa (ROI)

Regija interesa (eng. *Region of interest, ROI*) odnosi se na relevantni dio krivulje mjerenja ili slike na kojem je fokusirana statistička analiza. Često se koristi u različitim područjima, procese snimanja slika te medicinske primjene poput nuklearne medicine. ROI-ovi se mogu odrediti ručno ili uz pomoć softvera za evaluaciju, bilo poluautomatski gdje korisnik definira dio područja, ili potpuno automatski gdje softver određuje ROI bez intervencije korisnika. ROI omogućuje izračune poput maksimalne vrijednosti, prosjeka, širine vrha i površine ispod krivulje, olakšavajući analizu i interpretaciju podataka. [13]

## 5.4. Rezultati

Na slici 31 prikazan je rad detekcije objekata i klasifikacije.




**Slika 31. Primjer klasifikacije**

Na ovom primjeru se vidi kako detekcija objekata radi odlično kao i izoliranje ROI. Klasifikacija se provela uspješno pošto su sva 3 oblika točno klasificirani što znači da konvolucijska neuronska mreža radi ispravno.

### 5.4.1. Problemi

Kroz testiranje na vizijskom sustavu može se vidjeti koliko je utjecaj svjetlosti bitan kod detekcije i klasifikacije oblika. Dobro osvjetljenje potrebno je za ispravnu klasifikaciju, ali to ne mora biti pravilo. U par primjera je došlo do krive klasifikacije baš zbog odbijanja svjetlosti od boje na papiru. Odbijanje svjetlosti otežava mreži da prepozna boje i oblike što je i prikazano na slici 32.


**Slika 32. Primjer krive klasifikacije**

Rješenje tog problema moguće je optimiziranjem kamere i osvjetljenja, kao i stvaranjem složenije baze podataka, s više različitih parametara oblika i više različitih boja što sa sobom povlači kompliciraniju i zahtjevniju arhitekturu konvolucijske neuronske mreže.

# 6. ZAKLJUČAK

U završnom radu je obrađen teorijski pregled ključnih koncepta umjetne inteligencije, dubokog učenja, konvolucijskih neuronskih mreža i računalnog vida. Posebna pažnja posvećena je konvolucijskim neuronskim mrežama, ključnom alatu u računalnom vidu. Ove mreže sastoje se od slojeva konvolucije, aktivacije i poolinga, omogućujući modelu da automatski nauči značajke iz podataka (slika).

Osim toga, praktično je izveden eksperimentalni dio, koji obuhvaća stvaranje modela konvolucijske neuronske mreže, evaluaciju njegove učinkovitosti te testiranje na stvarnim podacima. Model konvolucijske neuronske mreže pokazuje odlične rezultate što se vidi kroz dijagrame gubitaka i preciznosti i matrice konfuzije. Nadalje, implementacija razvijenog modela na stvarnom vizijskom sustavu pružila je mogućnost provjere njegove funkcionalnosti u realnom okruženju.

Kroz ovaj proces, provjerena je sposobnost modela za klasifikaciju objekata putem kamere, što predstavlja važan korak u primjeni konvolucijskih neuronskih mreža u stvarnim sustavima. Može se vidjeti značaj osvjetljenja za ispravnu klasifikaciju kao i potrebnost za robusnosti modela i složenije baze podataka.

Ovaj rad pruža temeljnu teorijsku osnovu i praktične primjene u području dubokog učenja i računalnog vida, otvarajući put za daljnja istraživanja u području umjetne inteligencije.

# LITERATURA

[1] What is deep learning, https://www.ibm.com/topics/deep-learning, pristupljeno 10.2.2024.

[2] Convolutional Neural Networks, https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939, pristupljeno 10.2.2024.

[3] Activation Functions in Neural Networks, https://www.v7labs.com/blog/neural-networks-activation-functions, pristupljeno 10.2.2024.

[4] Python introduction, https://www.w3schools.com/python/python_intro.asp, pristupljeno 10.2.2024.

[5] TensorFlow. https://www.tensorflow.org/, pristupljeno 10.2.2024.

[6] OpenCV, https://opencv.org/, pristupljeno 10.2.2024.

[7] NumPy, https://numpy.org/, pristupljeno 10.2.2024.

[8] Overfitting and Regularization in CNN: https://dotnettutorials.net/lesson/overfitting-and-regularization-in-cnn/, pristupljeno 15.2.2024.

[9] Adam, https://keras.io/api/optimizers/adam/, pristpuljeno 10.2.2024.

[10] Confusion Matrix in Machine Learning, https://www.geeksforgeeks.org/confusion-matrix-machine-learning/, pristupljeno 10.2.2024.

[11] What is computer vision, https://www.ibm.com/topics/computer-vision, pristupljeno 10.2.2024.

[12] Canny Edge detection, https://docs.opencv.org/3.4/da/d22/tutorial_py_canny.html, pristupljeno 15.2.2024.

[13] Region of interest (ROI), https://www.smartray.com/glossary/region-of-interest-roi/, pristupljeno 15.2.204.

# PRILOZI

I. DataGenerator.py

```
import cv2
import numpy as np
import os
import random

# definiranje direktorija i labela
root_dir = 'Data'
shape_labels = ['circle', 'square', 'rectangle', 'triangle']
color_labels = ['red', 'green', 'blue']

# kreiranje direktorija
for shape in shape_labels:
    for color in color_labels:
        os.makedirs(os.path.join(root_dir, f'{shape}_{color}'), exist_ok=True)

for shape in shape_labels:
    shape_dir = os.path.join(root_dir, f'{shape}_red')
    num_classes = len(os.listdir(shape_dir))
    print(f"Classes for {shape}: {num_classes}")

print(f"Directories in {root_dir}: {os.listdir(root_dir)}")

# definiranje boja (BGR)
color_mapping = {
    'red': (0, 0, 255),      # BGR za crveno
    'green': (0, 255, 0),    # BGR za zeleno
    'blue': (255, 0, 0),     # BGR za plavo
}

# kreiranje oblika
```

```
def generate_shape(shape, image_size, border_size=40, color='red'):
    background_color = (random.randint(0, 255), random.randint(0, 255), random.randint(0, 255))
    image = np.ones((image_size[0], image_size[1], 3), dtype=np.uint8) * background_color
    bgr_color = color_mapping[color]

    if shape == 'circle':
        max_radius = min((image_size[0] - 2 * border_size) // 2, (image_size[1] - 2 * border_size) // 2)
        radius = random.randint(10, max_radius)
        center_x = random.randint(border_size + radius, image_size[0] - border_size - radius)
        center_y = random.randint(border_size + radius, image_size[1] - border_size - radius)
        points = cv2.ellipse2Poly((center_x, center_y), (radius, radius), 0, 0, 360, 1)
        cv2.fillPoly(image, [points], bgr_color)

    elif shape == 'square':
        max_side_length = min(image_size[0] - 2 * border_size, image_size[1] - 2 * border_size)
        side_length = random.randint(10, max_side_length)
        top_x = random.randint(border_size, image_size[0] - border_size - side_length)
        top_y = random.randint(border_size, image_size[1] - border_size - side_length)
        points = np.array([[top_x, top_y], [top_x + side_length, top_y], [top_x + side_length, top_y + side_length], [top_x, top_y + side_length]], np.int32)
        cv2.fillPoly(image, [points], bgr_color)

    elif shape == 'rectangle':
        max_width = image_size[0] - 2 * border_size
        max_height = image_size[1] - 2 * border_size
        width = random.randint(20, max_width)
        height = random.randint(20, max_height)
        top_x = random.randint(border_size, image_size[0] - border_size - width)
        top_y = random.randint(border_size, image_size[1] - border_size - height)
        points = np.array([[top_x, top_y], [top_x + width, top_y], [top_x + width, top_y +
```

```
height], [top_x, top_y + height]], np.int32)
        cv2.fillPoly(image, [points], bgr_color)

    elif shape == 'triangle':
        min_area = 2000
        while True:
            point1 = (random.randint(border_size, image_size[0] - border_size),
                      random.randint(border_size, image_size[1] - border_size))
            point2 = (random.randint(border_size, image_size[0] - border_size),
                      random.randint(border_size, image_size[1] - border_size))
            point3 = (random.randint(border_size, image_size[0] - border_size),
                      random.randint(border_size, image_size[1] - border_size))
            points = np.array([point1, point2, point3], np.int32)
            area = cv2.contourArea(points)
            if area >= min_area:
                cv2.fillPoly(image, [points], bgr_color)
                break
    return image
def generate_and_save_image(shape, color, image_id):
    image_size = (255, 255)
    image = generate_shape(shape, image_size, border_size=40, color=color)
    image_filename = os.path.join(root_dir, f"{shape}_{color}", f"{image_id}.jpg")
    cv2.imwrite(image_filename, image)

# generiranje slika
num_samples_per_class = 5000
for i in range(num_samples_per_class):
    for shape in shape_labels:
        for color in color_labels:
            generate_and_save_image(shape, color, i)

class_indices = {}
for i, shape in enumerate(shape_labels):
```

```
    for color in color_labels:
        class_label = f"{shape}_{color}"
        class_indices[class_label] = i * len(color_labels) + color_labels.index(color)
print("Class Indices:", class_indices)

num_classes = len(class_indices)
print("Number of Classes:", num_classes)
```

II. CNN.py

```
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.layers import BatchNormalization, Dropout
from tensorflow.keras.optimizers import Adam
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix

# korištenje GPU
physical_devices = tf.config.list_physical_devices('GPU')
if len(physical_devices) > 0:
    tf.config.experimental.set_memory_growth(physical_devices[0], True)

# definiranje konstanti
DATA_DIR = 'Data'
IMAGE_SIZE = (64, 64)
BATCH_SIZE = 32
NUM_CLASSES = 12
EPOCHS = 10

# funkcija za učitavanje i procesiranje slika
```

```
def load_images_with_generator(data_dir, image_size):
    datagen = tf.keras.preprocessing.image.ImageDataGenerator(
        rescale=1./255,
        validation_split=0.6  # podjela na 40% podataka za treniranje i 60% za testiranje
    )
    train_generator = datagen.flow_from_directory(
        data_dir,
        target_size=image_size,
        batch_size=BATCH_SIZE,
        class_mode='categorical',
        subset='training',
        shuffle=True,
    )
    validation_generator = datagen.flow_from_directory(
        data_dir,
        target_size=image_size,
        batch_size=BATCH_SIZE,
        class_mode='categorical',
        subset='validation',
        shuffle=True,
    )
    return train_generator, validation_generator

train_generator, validation_generator = load_images_with_generator(DATA_DIR,
IMAGE_SIZE)

# arhitektura konvolucijske neuronske mreže
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(IMAGE_SIZE[0],
IMAGE_SIZE[1], 3)),
    BatchNormalization(),
    layers.MaxPooling2D((2, 2)),
```

```
    layers.Conv2D(64, (3, 3), activation='relu'),
    BatchNormalization(),
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(128, (3, 3), activation='relu'),
    BatchNormalization(),
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(256, (3, 3), activation='relu'),
    BatchNormalization(),
    layers.MaxPooling2D((2, 2)),

    layers.Flatten(),
    layers.Dense(256, activation='relu'),
    Dropout(0.5),
    layers.Dense(NUM_CLASSES, activation='softmax')
])

# optimiziranje modela
optimizer = Adam(learning_rate=0.0001)
model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'])

# treniranje modela
history = model.fit(train_generator, epochs=EPOCHS, validation_data=validation_generator)

# evaluacija modela
loss, accuracy = model.evaluate(validation_generator)
print("Validation Loss:", loss)
print("Validation Accuracy:", accuracy)

# dijagram gubitaka
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
```

```
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training and Validation Loss')
plt.legend()
plt.show()

# dijagram preciznosti
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()
plt.show()

# imena klasa
class_names = ['circle_red', 'circle_green', 'circle_blue', 'square_red', 'square_green',
'square_blue', 'rectangle_red', 'rectangle_green', 'rectangle_blue', 'triangle_red',
'triangle_green', 'triangle_blue']

validation_generator.reset()
validation_images, validation_labels = next(validation_generator)
validation_predictions = np.argmax(model.predict(validation_images), axis=-1)
cm = confusion_matrix(np.argmax(validation_labels, axis=1), validation_predictions)

# kreiranje matrice konfuzije
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=class_names,
yticklabels=class_names)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()
```

```
# spremanje modela
model.save("cnn1.h5")
print("Model spremljen kao 'cnn1.h5'")
```

III. Testing.py

```
import cv2
import numpy as np
import os
import random

# Definiranje direktorija i labela
root_dir = 'Data'
shape_labels = ['circle', 'square', 'rectangle', 'triangle']
color_labels = ['red', 'green', 'blue']

# Kreiranje direktorija
for shape in shape_labels:
    for color in color_labels:
        os.makedirs(os.path.join(root_dir, f'{shape}_{color}'), exist_ok=True)

for shape in shape_labels:
    shape_dir = os.path.join(root_dir, f'{shape}_red')
    num_classes = len(os.listdir(shape_dir))
    print(f"Classes for {shape}: {num_classes}")

print(f"Directories in {root_dir}: {os.listdir(root_dir)}")

# definiranje boja (BGR)
color_mapping = {
    'red': (0, 0, 255),      # BGR za crveno
    'green': (0, 255, 0),    # BGR za zeleno
```

```
    'blue': (255, 0, 0),     # BGR za plavo
}

# kreiranje oblika
def generate_shape(shape, image_size, border_size=40, color='red'):
    background_color = (random.randint(0, 255), random.randint(0, 255), random.randint(0, 255))
    image = np.ones((image_size[0], image_size[1], 3), dtype=np.uint8) * background_color
    bgr_color = color_mapping[color]

    if shape == 'circle':
        max_radius = min((image_size[0] - 2 * border_size) // 2, (image_size[1] - 2 * border_size) // 2)
        radius = random.randint(10, max_radius)
        center_x = random.randint(border_size + radius, image_size[0] - border_size - radius)
        center_y = random.randint(border_size + radius, image_size[1] - border_size - radius)
        points = cv2.ellipse2Poly((center_x, center_y), (radius, radius), 0, 0, 360, 1)
        cv2.fillPoly(image, [points], bgr_color)

    elif shape == 'square':
        max_side_length = min(image_size[0] - 2 * border_size, image_size[1] - 2 * border_size)
        side_length = random.randint(10, max_side_length)
        top_x = random.randint(border_size, image_size[0] - border_size - side_length)
        top_y = random.randint(border_size, image_size[1] - border_size - side_length)
        points = np.array([[top_x, top_y], [top_x + side_length, top_y], [top_x + side_length, top_y + side_length], [top_x, top_y + side_length]], np.int32)
        cv2.fillPoly(image, [points], bgr_color)

    elif shape == 'rectangle':
        max_width = image_size[0] - 2 * border_size
        max_height = image_size[1] - 2 * border_size
        width = random.randint(20, max_width)
```

```
        height = random.randint(20, max_height)
        top_x = random.randint(border_size, image_size[0] - border_size - width)
        top_y = random.randint(border_size, image_size[1] - border_size - height)
        points = np.array([[top_x, top_y], [top_x + width, top_y], [top_x + width, top_y +
height], [top_x, top_y + height]], np.int32)
        cv2.fillPoly(image, [points], bgr_color)

    elif shape == 'triangle':
        min_area = 2000
        while True:
            point1 = (random.randint(border_size, image_size[0] - border_size),
                      random.randint(border_size, image_size[1] - border_size))
            point2 = (random.randint(border_size, image_size[0] - border_size),
                      random.randint(border_size, image_size[1] - border_size))
            point3 = (random.randint(border_size, image_size[0] - border_size),
                      random.randint(border_size, image_size[1] - border_size))
            points = np.array([point1, point2, point3], np.int32)
            area = cv2.contourArea(points)
            if area >= min_area:
                cv2.fillPoly(image, [points], bgr_color)
                break
    return image
def generate_and_save_image(shape, color, image_id):
    image_size = (255, 255)
    image = generate_shape(shape, image_size, border_size=40, color=color)
    image_filename = os.path.join(root_dir, f"{shape}_{color}", f"{image_id}.jpg")
    cv2.imwrite(image_filename, image)

# generiranje slika
num_samples_per_class = 5000
for i in range(num_samples_per_class):
    for shape in shape_labels:
        for color in color_labels:
```

```
            generate_and_save_image(shape, color, i)

class_indices = {}
for i, shape in enumerate(shape_labels):
    for color in color_labels:
        class_label = f"{shape}_{color}"
        class_indices[class_label] = i * len(color_labels) + color_labels.index(color)
print("Class Indices:", class_indices)

num_classes = len(class_indices)
print("Number of Classes:", num_classes)
```

IV. ObjectRecognition_cam.py

```
import cv2
import numpy as np
from keras.models import load_model
from PIL import Image

# učitavanje modela konvolucijske neuronske mreže
model = load_model('cnn1.h5')

# definiranje klasa
class_indices = {
    'Plavi krug': 0, 'Zeleni krug': 1, 'Crveni krug': 2,
    'Plavi kvadrat': 3, 'zeleni kvadrat': 4, 'Crveni kvadrat': 5,
    'Plavi pravokutnik': 6, 'Zeleni pravokutnik': 7, 'Crveni pravokutnik': 8,
    'Plavi trokut': 9, 'Zeleni trokut': 10, 'Crveni trokut': 11
}

# funkcija za predviđanje oblika
def predict_image_class(image):
```

```
    img_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    img_pil = Image.fromarray(img_rgb)
    img_resized = img_pil.resize((64, 64))
    img_array = np.array(img_resized)
    img_array = np.expand_dims(img_array, axis=0)
    img_array = img_array.astype('float32') / 255.0
    prediction = model.predict(img_array)
    predicted_class_index = np.argmax(prediction)
    predicted_class =
list(class_indices.keys())[list(class_indices.values()).index(predicted_class_index)]
    return predicted_class

# funkcija za detekciju oblika
def detect_objects(frame):
    # u sivo
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Canny detekcija rubova
    edges = cv2.Canny(gray, 50, 150)
    contours, _ = cv2.findContours(edges, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

    # lista za ROI
    ROIs = []
    ROI_coords = []

    # označivanje oblika
    for contour in contours:
        area = cv2.contourArea(contour)
        if area > 500:
            x, y, w, h = cv2.boundingRect(contour)

            # proširivanje bounding box-a
```

```
            expand_factor = 0.02
            new_x = max(0, x - int(w * expand_factor))
            new_y = max(0, y - int(h * expand_factor))
            new_w = min(frame.shape[1], x + w + int(w * expand_factor)) - new_x
            new_h = min(frame.shape[0], y + h + int(h * expand_factor)) - new_y

            cv2.rectangle(frame, (new_x, new_y), (new_x + new_w, new_y + new_h), (0, 255, 0),
2)

            # izuzimanje ROI sa slike
            roi = frame[new_y:new_y + new_h, new_x:new_x + new_w]
            ROIs.append(roi)
            ROI_coords.append((new_x, new_y, new_w, new_h))

    return frame, ROIs, ROI_coords

# inicijalizacija kamere
cap = cv2.VideoCapture(1)

while True:
    ret, frame = cap.read()

    # Ddetekcija oblika
    frame_with_boxes, ROIs, ROI_coords = detect_objects(frame)

    if ROIs:
        # klasifikacija za svaki ROI
        for i, (roi, (x, y, w, h)) in enumerate(zip(ROIs, ROI_coords)):
            predicted_label = predict_image_class(roi)

            # crtanje oznake oko oblika
            cv2.rectangle(frame_with_boxes, (x, y), (x + w, y + h), (0, 0, 0), 2)
```

```
            text_x = x
            text_y = y - 10 if y - 10 > 10 else y + 20

            # ispisivanje teksta klase
            cv2.putText(frame_with_boxes, predicted_label, (text_x, text_y),
            cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0), 2)

    # prikaz kamere
    cv2.imshow('Object Detection', frame_with_boxes)

    # gašenje kamere
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```