NEURALNE MREZE

Projekat iz dubokog učenja

NATALIJA GVOZDENOVIĆ 2021/0294 I MINJA BABIĆ 2021/0228

Sadržaj

TABELA SADRŽANIH SLIKA

1. Uvod

U sklopu ovog projektnog zadatka, istraživano je i primenjeno duboko učenje na dataset-u namenjenom klasifikaciji slika. Cilj je razviti model koji će moći da klasifikuje geometrijske oblike, koji su crtani rukom. Model će klasifikovati slike geomatrijskih oblika u određene klase, a u ovom izveštaju detaljno će se opisati sam dataset i pristup koji je korišćen za rešavanje ovog problema. Važno je napomenuti da su se Konvolucione neuralne mreže (CNN) pokazale izuzetno efikasno u zadacima obrade slika, što je i korišćeno u ovom projektu. Ove mreže su projektovane za rad s vizuelnim podacima, kao što su slike, i omogućavaju modelima da efikasno uče obeležja i obrazce prisutne u slikama. CNN su važne u dubokom učenju jer poseduju sposobnost automatskog ekstrahovanja obeležja iz slika, što je ključno za uspešnu klasifikaciju. Ove mreže koriste konvolucione slojeve za detektovanje lokalnih oblika i obrazaca, a slojevi sažimanja za smanjenje dimenzionalnosti, čineći ih idealnim za zadatke klasifikacije slika.

2. Opis problema

2.1 Opis dataset-a

Dataset koji se koristi za ovaj projekat obuhvata raznovrsne slike koje prikazuju različite geometrijske oblike koji su crtani rukom. Sve slike su podeljene u jasno definisane klase, a svaka klasa predstavlja određenu vrstu geometrijskog objekta. Sve slike se nalaze u jednom direktorijumu “dataset”, grupisane po direktorijumima koji nose naziv kao nazivi klasa (naziv vrste geometrijskog objekta) koje imamo u dataset-u. Postoji 8 klasa u našem dataset-u čiji nazivi su: **circle** (krug), **kite** (deltoid), **parallelogram** (paralelogram), **rectangle** (pravugaonik), **rhombus** (romb), **square** (kvadrat), **trapezoid** (trapez), **triangle** (trougao)

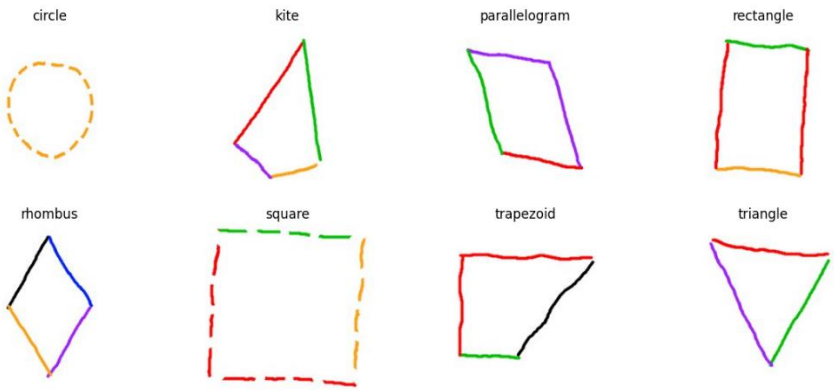

Figure 1 Prikaz jednog primera svake klase

Ukupan broj slika u datasetu iznosi 20 000, a raspoređene su u 8 klasa. Svaka klasa sadrži 2500 slika. Slike su dimenzije 224 x 224 I formata .jpg.

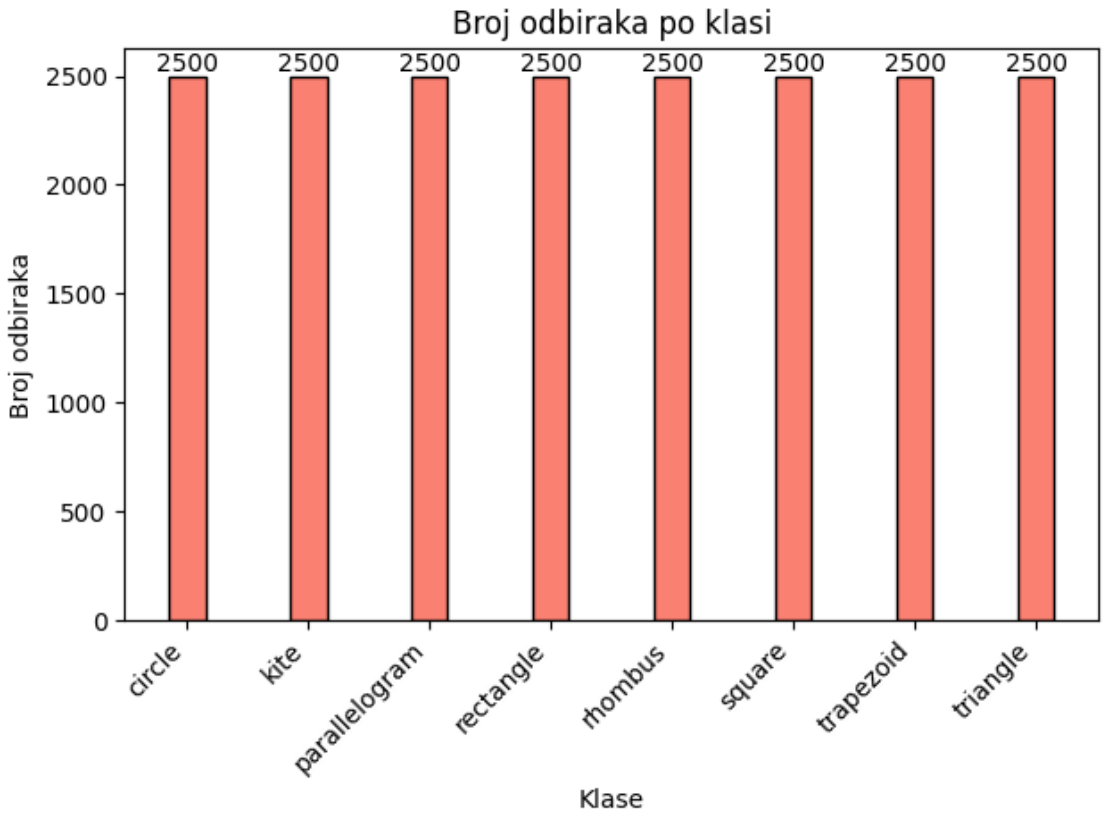

Figure 2 Prikaz broja odbiraka po klasi

2.1.1 Balansiranost podataka

Balansiranost podataka igra ključnu ulogu u efikasnom treniranju modela dubokog učenja. Kada je reč o klasifikaciji slika, važno je da svaka klasa ima približno jednak broj uzoraka u skupu podataka. Međutim, u našem konkretnom slučaju, srećom, već posedujemo dataset koji je sam po sebi balansiran. Svaka od osam klasa ima tačno 2500 uzoraka, obezbeđujući jednak broj odbiraka za svaku geometrijsku figuru.

Nije bilo potrebe za dodatnim koracima balansiranja podataka, poput oversampling-a ili undersampling-a, jer je distribucija uzoraka već uniformna. Ova jednakost u broju uzoraka omogućava modelu da nauči reprezentacije svake klase podjednako, čime se postiže pravednija i pouzdanija klasifikacija.

2.2 Cilj rešavanja problema

Cilj ovog projekta je iskoristiti duboko učenje za rešavanje problema klasifikacije geometrijskih figura na osnovu slika. Glavni fokus je na stvaranju modela koji može precizno identifikovati različite oblike, uključujući krug, romb, kvadrat, trougao i ostale geometrijske figure, u skladu sa specifičnostima dataset-a. Korišćenjem dubokih neuralnih mreža, posebno konvolucionih neuralnih mreža (CNN), želimo postići visok stepen tačnosti u klasifikaciji, omogućavajući modelu da prepozna obrasce i karakteristike koje definišu svaku pojedinačnu klasu.

3. Postupak reševanje problema

Prilikom rešavanja problema klasifikacije slika kroz duboko učenje, primenili smo postupan i sistematičan pristup kako bismo obezbedili bolje razumevanje i implementaciju modela. Korak po korak, kroz nekoliko ključnih faza, dolazili smo do zadovoljavajuće tačnosti.

3.1 Podela podataka na odgovarajuće skupove

Nakon analize našeg dataset-a i zaključka da ne moramo da primenjujemo metode kojima ćemo da balansiramo podatke, prva važna stvar, pre nego što se počne sa pravljenjem modela, je podela podatka na odgovarajuće skupove.

Podela podataka na odgovarajuće skupove ključan je korak u procesu razvoja i evaluacije modela dubokog učenja. Ovaj korak omogućava precizno ocenjivanje performansi modela, kao i sprečavanje pojave problema poput preobučavanja modela. U kontekstu klasifikacije geometrijskih figura, podatke smo podelili na 3 skupa: trening, validacioni i test skup. Odnos odbiraka u trening, validacionom I test skupu je 60% - 20% - 20%. Važnost podele na ove skupove se ogleda u sledećem:

- **Trening skup** koristi se za obučavanje modela, gde model uči prepoznavanje obrazaca i obeležja slika. On je osnova na kojoj model uči i prilagodjava svoje parameter kako bi minimizovao grešku u predviđanju.Model tokom više epoha učenja “vidi” podatke za trening I tako prilagođava svoje težine I gradi sposobnost generalizacije.
- **Validacioni skup** koristi se za podešavanje hiperparametara i praćenje performansi modela tokom obučavanja. Bitan je prilikom primene tehnika za sprečavanje od preobučavanja mreže. Takođe praćenjem performansi modela na validacionom skupu tokom obučavanja omogućava nam da identifikujemo trendove, promene u tačnosti I gubitke , što može biti korisno za dijagnostiku I prilagođavanje modela kako bi se postigao što bolji rezultat.
- **Test skup** se koristi za evaluaciju sposobnosti modela da generalizuje na nepoznate podatke. Model koji dobro radi na test skupu pokazuje visoku sposobnost generalizacije i bolje će se ponašati u stvarnim situacijama. Ovo je ključni skup za procenu performansi modela i njegovu primenu u praktičnim scenarijima.

Ako ne bi bilo ovakve podele I pravilnog korišćenja ovih skupova ne bismo mogli da dobijemo tačne I dobre rezultate. Kada bi se umesto test skupa koristi trening skup, tačnost bi bila skoro stoprocentna što ne bi imalo smisla, cilj je da model radi za bilo koje podatke uključujći I one na kojima nije trenirao.

3.1.1 Način podele podataka po skupovima

Ispod možemo da vidimo deo koda koji deli podatke na 3 skupa. Važna funkcija koja je korišćena je `image_dataset_from_directory`, služi za učitavanje podataka iz direktorijuma (na putanji main_path). Koristeći dva puta ovu funkciju delimo dataset na dva skupa, jedan je trening skup a drugi je skup koji sadrži podatke za test I za validacioni skup. Podaci su izmešani jer je shuffle postavljen na True, a odnos broja odbiraka u skupovima je definisan sa tim što smo obeležili da je validation size 40% (u ovom slučaju i val i test skup), a onda je ostatak – 60% rezervisan za trening skup. Ova podela je odrađena jer je

eksplicitno navedeno o kom skupu podataka se radi, subset=" training " ili subset="validation". Nakon toga Xvaltest skup smo podelili na pola kako bismo dobili dva skupa od po 20% podataka. Korišćene funkcije su take i skip , koje uzimaju to jest preskaču prvih size podatak.

```
Xtrain= image_dataset_from_directory(main_path,
                                     subset='training',validation_split=0.4,
                                    image_size=img_size,
                                    batch_size=batch_size,
                                    seed=123, shuffle=True
                                     )

Xvaltest = image_dataset_from_directory(main_path,
                                    subset = 'validation',
                                    validation_split=0.4,
                                    image_size=img_size,
                                    batch_size=batch_size,
                                    seed=123, shuffle = True)

validation_size = int(0.5 * len(Xvaltest))
test_size = len(Xvaltest) - validation_size

Xval = Xvaltest.take(validation_size)
Xtest = Xvaltest.skip(validation_size)
```

Kako bismo proverili da li smo dobro podelili podatke napisali smo sledeći deo koda:

```
def count_images_per_class(dataset):
    class_counts = {}
    for _, labels in dataset:
        for label in labels.numpy():
            if label not in class_counts:
                class_counts[label] = 1
            else:
                class_counts[label] += 1
    return class_counts

# Brojanje slika za skupove
train_class_counts = count_images_per_class(Xtrain)
print(f"\nUkupan broj slika u trening skupu: {sum(train_class_counts.values())}")
print("Broj slika po klasama u trening skupu:")
print(train_class_counts)

val_class_counts = count_images_per_class(Xval)
print(f"\nUkupan broj slika u validacionom skupu: {sum(val_class_counts.values())}")
print("Broj slika po klasama u validacionom skupu:")
print(val_class_counts)

test_class_counts = count_images_per_class(Xtest)
print(f"\nUkupan broj slika u test skupu: {sum(test_class_counts.values())}")
print("Broj slika po klasama u test skupu:")
print(test_class_counts)
```

Ispis u komandnoj liniji:

Ukupan broj slika u trening skupu: 12000

Broj slika po klasama u trening skupu:
{4: 1513, 3: 1495, 1: 1474, 2: 1505, 5: 1505, 7: 1463, 0: 1549, 6: 1496}

Ukupan broj slika u validacionom skupu: 3968
Broj slika po klasama u validacionom skupu:
{5: 482, 1: 506, 0: 486, 4: 492, 7: 500, 6: 495, 2: 500, 3: 507}

Ukupan broj slika u test skupu: 4032
Broj slika po klasama u test skupu:
{3: 506, 2: 503, 4: 503, 6: 506, 1: 520, 0: 461, 5: 495, 7: 538}

Kao što vidimo broj odbiraka svake klase je približno isti za svaku klasu. Kada pogledamo broj slika u trening skupu on je tačno 60% od ukupnog broja slika (12 000/20 000= 0.7). Međutim broj slika u test I validacionom skupu nije isti, tačnije razlikuje se za 64 slike. To se desilo zbog batch_size argumenta u `image_dataset_from_directory` funkciji.

```
batch_size = 64
```

Kako je batch_size postavljen na 64, to znači da će u svakoj iteraciji treniranja naš model koristiti podatke sa 64 slike. Ovo se koristi kako bi se ubrzao proces treniranja i smanjila potreba za memorijom. Podaci su a taj način podeljeni na manje grupe podataka koje se nazivaju serije ili paketi (engl. batches) koji sadrže batch_size podatajka. Uobičajene vrednosti batch_size mogu biti 32, 64, 128 ili neke druge broj koji je stepen 2.

3.2 Predprocesiranje podataka

Predprocesiranje podataka je ključan korak u pripremi podataka za treniranje modela Ovaj korak ima za cilj poboljšanje performansi modela i olakšavanje procesa učenja. Narednih nekoliko koraka smo uključili u naše predprocesiranje podataka.

3.2.1 Skaliranje podataka

Skaliranje slika na određenu veličinu može olakšati rad modela, posebno ako slike nisu istih dimenzija ili su povećih dimenzija. Modeli uglavnom zahtevaju da se slike budu istih dimanzija. Velicina slike može da utiče na brzinu obučavanja modela, jer veće slike zauzimaju više memorijskog prostora. Podaci su u našem projektu skalirani prilikom učitavanja, u funkciji `image_dataset_from_directory`, tako što je postavljen argument funkcije image_size= img_size.

```
img_size = (64,64)
```

Kao što vidimo naše slike koje su 224 x 224 su smanjene na dimenziju 64 x 64.

3.2.2 Normalizacija podataka

Normalizacija podataka je proces skaliranja vrednosti podataka na određeni opseg, često na interval od 0 do 1. Ovo je posebno bitno kada se radi sa slikama, gde pikseli imaju vrednosti u opsegu od 0 do 255. Normalizacija pomaže stabilizaciji procesa učenja i ubrzava konvergenciju modela. Koristili smo

normalizaciju kako bismo skalirali vrednosti piksela na opseg od 0 do 1 radi lakšeg procesiranja podataka prilikom obučavanja mreže I boljih performansi obučavanja modela .

Normalizaciju smo dodavali prilikom formiranja modela kao sloj poseban sloj u modelu:

```
layers.Rescaling(1./255, input_shape=(64, 64, 3))
```

- 1./255: Ovaj deo označava faktor skaliranja. Vrednosti piksela su u opsegu [0, 255], gde 0 predstavlja crnu, 255 belu, a vrednosti između odgovaraju nijansama sive. Deljenjem svih vrednosti piksela sa 255, postiže se normalizacija vrednosti u opseg [0, 1], što često može poboljšati performanse obučavanja modela.
- input_shape=(64, 64, 3): Ovde se postavlja očekivana veličina ulaznih slika. U ovom slučaju, očekuje se da slike imaju dimenzije 64x64 piksela i tri kanala boje (R, G, B). To se koristi kako bi se definisao oblik ulaznih podataka koji će se očekivati od ovog sloja u modelu.

Ovaj sloj smo koristili kao drugi sloj u neuralnoj mreži kako bi se normalizovali pikseli slika pre nego što prođu kroz ostale slojeve mreže tokom procesa obučavanja. Prvi sloj se odnosi na augmentaciju podataka.

3.2.3 Augmentacija podataka

Augmentacija skupa podataka predstavlja tehniku proširivanja trening skupa podataka sa sintetičkim podacima. Tehnike augmentacije podataka su sledeće:

- pomeranje trening slika za nekoliko piksela u svakom pravcu
- rotiranje slike
- menjanje svetlosti i kontrasta
- skaliranje slike
- ubacivanje šuma u ulaz

```
data_augmentation = Sequential()
data_augmentation.add(
    layers.RandomFlip("horizontal_and_vertical", input_shape=(64,64, 3) ) )
data_augmentation.add(layers.GaussianNoise(0.5, input_shape=(64, 64, 3)))
data_augmentation.add(layers.RandomRotation(0.2))
data_augmentation.add( layers.RandomZoom(0.1))
```

Kako bismo što bolje obučili naš model, napravili smo “novi model” koji sadrži slojeve koji menjanju naše ulazne podatke. Ovi slojevi obavaljaju različite tehnike augmentacije podataka. Prvo, primenili smo horizontalno i vertikalno okretanje pomoću RandomFlip sloja kako bih dodala više perspektiva na geometrijske oblike. Ovo je posebno korisno jer geometrijski oblici, kao što su krugovi i trouglovi, mogu imati različite orijentacije u stvarnom svetu.

Zatim, dodali smo GaussianNoise kako bih simulirali sitne varijacije koje mogu nastati tokom procesa crtanja geometrijskih oblika. Ova tehnika pomaže modelu da nauči invarijantnost na šum i bolje se prilagodi stvarnim uslovima.

RandomRotation i RandomZoom su bili od značaja za rukovanje različitim veličinama i orijentacijama geometrijskih oblika. Uvođenjem nasumičnih rotacija i uvećanja, model postaje otporniji na promene u perspektivi i veličini slike.

3.3 Neuralna mreza

Ovde cemo formirati I obuciti mrezu...

3.3.1 Formiranje neuralne mreže

Formiranje neuralne mreže dosta utiče na rezultate vašeg projekta. Kako bismo dobro obučili model I napravili ga da što manje greši moramo dobro formirati model. Formiranje modela se sastoji iz nekoliko koraka koje smo uključili u naš model. Neki od mogućih koraka su definisanje arhitekture mreže, izbor slojeva, postavljanje ulaznih dimenzija, izbor aktivacione funkcije, izbor kriterijumske funkcije (loss funkcija) i optimizatora nakon čega sledi kompilacija modela i samo obučavanje modela.

3.3.1.1 Kriterijumska funkcija

Kriterijumska funkcija koja je korišćena je Sparse Categorical Crossentropy.

$$L(y_{\text{true}}, y_{\text{pred}}) = -\frac{1}{N}\sum_{i=1}^{N} \log\left(\frac{e^{y_{\text{pred}_i}[y_{\text{true}_i}]}}{\sum_j e^{y_{\text{pred}_i}[j]}}\right)$$

Gde su:

- N broj uzoraka.
- y_{true_i} indeks stvarne klase za i-ti uzorak.
- y_{pred_i} vektor predviđenih logita za i-ti uzorak (izlazi pre aktivacije modela).

Figure 3 formula Sparse Categorical Crossentropy kriterijumske funkcije

Ova funkcija omogućava efikasno kodiranje ciljnih podataka i dobra je kada se radi klasifikacija više od dve klase.

3.3.1.2 Funkcija aktivacije neurona

Korišćena funkcija aktivacije u svim slojevima je ReLU. ReLU je jednostavan matematički izraz koji se brzo i lako izračunava.

$$f(x) = \max(0, x)$$

ReLU pomaže u smanjenju problema s nestajanjem gradijenta tako što sa svojim lineralnim delom ojačava gradijent i tako ubrzava učenje.

3.3.1.3 Metoda optimizacije kriterijumske funkciju

Adamov optimizator kombinuje ideje iz drugih optimizatora, kao što su momentum i RMSprop, kako bi obezbedio efikasno i brzo optimizovanje modela. Adam automatski prilagođava stopu učenja za svaki

parametar na osnovu istorije gradijenata. Ovo pomaže u efikasnom treniranju modela sa različitim brzinama konvergencije za različite parametre.

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$$

3.3.1.4 Arhitektura kreiranog modela

Model: "sequential_1"

Layer (type)	Output Shape	Param#
sequential (Sequential	(None, 64, 64, 3)	0
rescaling (Rescaling)	(None, 64, 64, 3)	0
conv2d (Conv2D)	(None, 64, 64, 32)	896
max_pooling2d (MaxPooling2	(None, 32, 32, 32)	0 D)
conv2d_1 (Conv2D)	(None, 32, 32, 64)	18496
max_pooling2d_1 (MaxPoolin	None, 16, 16, 64)	(0 g2D)
conv2d_2 (Conv2D)	(None, 16, 16, 128)	73856
max_pooling2d_2 (MaxPoolin	(None, 8, 8, 128)	0 g2D)
conv2d_3 (Conv2D)	(None, 8, 8, 256)	295168
max_pooling2d_3 (MaxPoolin	(None, 4, 4, 256)	0 g2D)
flatten (Flatten)	(None, 4096)	0
dense (Dense)	(None, 512)	2097664
dense_1 (Dense)	(None, 256)	131328
dense_2 (Dense)	(None, 8)	2056

==========

Total params: 2619464 (9.99 MB)

Trainable params: 2619464 (9.99 MB)

Non-trainable params: 0 (0.00 Byte)

3.3.2 Obucavanje neuralne mreže

```
model.compile(optimizer = opt,
                  loss=SparseCategoricalCrossentropy(),
                  metrics='accuracy')

    return model

es = EarlyStopping(monitor='val_loss', mode='min', patience=20, verbose=1,
restore_best_weights=True)

tuner = kt.RandomSearch(make_model,
                        objective='val_loss',
                        overwrite = True,
                        max_trials=10)

tuner.search(Xtrain,
             epochs=10,
             validation_data=Xval,
             callbacks = [es],
             verbose=1)

best_model = tuner.get_best_models()
best_hyperparam = tuner.get_best_hyperparameters(num_trials=1)[0]

print('Optimalna konstanta obučavanja: ', best_hyperparam['learning_rate'])

model = tuner.hypermodel.build(best_hyperparam)

N = 8
history = model.fit(Xtrain,
                    epochs=50,
                    validation_data=Xval,
                    callbacks=[es],
                    verbose=1)
```

model.fit pokreće obučavanje

3.4 Preobucavanje mreze

Preobučavanje neuralne mreže je pojava kada mreža uči dobro na trening skupu podataka, ali gubi sposobnost generalizacije na novim podacima. Ako model ima previše parametara ili je prekomerno složen u odnosu na trening skup, on može memorisati podatke umesto da nauči njihove generalne karakteristike. To znači da će se za malu promenu ulaznih podataka postići veliki stepen greške. Medjutim, ukoliko se mreži da premalo parametara za obuku, ona ne može da uspe da se obuči. Zbog toga je važno postići balans između dovoljne složenosti modela kako bi se uspešno nosio s problemom preobučavanja

Da bi se sprečilo preobučavanje, mogu se primeniti različite tehnike kao što su regularizacija, potkresivanje mreže(prunning neural networks), dropout, rano zaustavljanje... Kao metodu zaštite od preobučvanja koristili smo

3.4.1 Rano zaustavljanje

Na naš model je primenjena tehnika ranog zaustavljanja(early stopping). Osnovna ideja ove tehnike je praćenje performansi modela na validacionom skupu podataka tokom treninga i prekidanje treninga kada se primeti da se performanse na validacionom skupu prestaju poboljšavati, čime se sprečava dalje preobučavanje.

3.5 Hiperparametri

U našem modelu smo pronašli optimalnu vrednost konstante obučavanja. Konstanta obučavanja se koristi kako bi se optimizovala konvergencija i performanse modela tokom treniranja neuralnih mreža. Pronalaženje optimalne vrednosti za ovaj hiperparametar ključno je za postizanje bržeg i efikasnijeg učenja.

```
lr = hp.Choice('learning_rate', values=[1e-2, 1e-3, 1e-4])
    opt = Adam(learning_rate=lr)

    model.compile(optimizer = opt,
                  loss=SparseCategoricalCrossentropy(),
                  metrics='accuracy')

    return model

es = EarlyStopping(monitor='val_loss', mode='min', patience=20, verbose=1,
restore_best_weights=True)

tuner = kt.RandomSearch(make_model,
                        objective='val_loss',
                        overwrite = True,
                        max_trials=10)

tuner.search(Xtrain,
             epochs=10,
             validation_data=Xval,
             callbacks = [es],
             verbose=1)

best_model = tuner.get_best_models()
best_hyperparam = tuner.get_best_hyperparameters(num_trials=1)[0]

print('Optimalna konstanta obučavanja: ', best_hyperparam['learning_rate'])

model = tuner.hypermodel.build(best_hyperparam)
```

Na osnovu ponovnog obučavanja mreže za svaku vrednost mogućeg hiperparametra konstante obučavanja, dobili smo vrednost konstante obučavanja preko najboljeg modela. Kasnije smo koristili istu tu konstantu u optimizatoru I poboljšali tačnost klasifikacije.

4. Finalno obučen model

4.1 Performanse neuralne mreže kroz epohe obučavanje

Narandzasta boja predstavlja performanse mreže na validacionom skupu, a plava boja na trening skupu podataka.

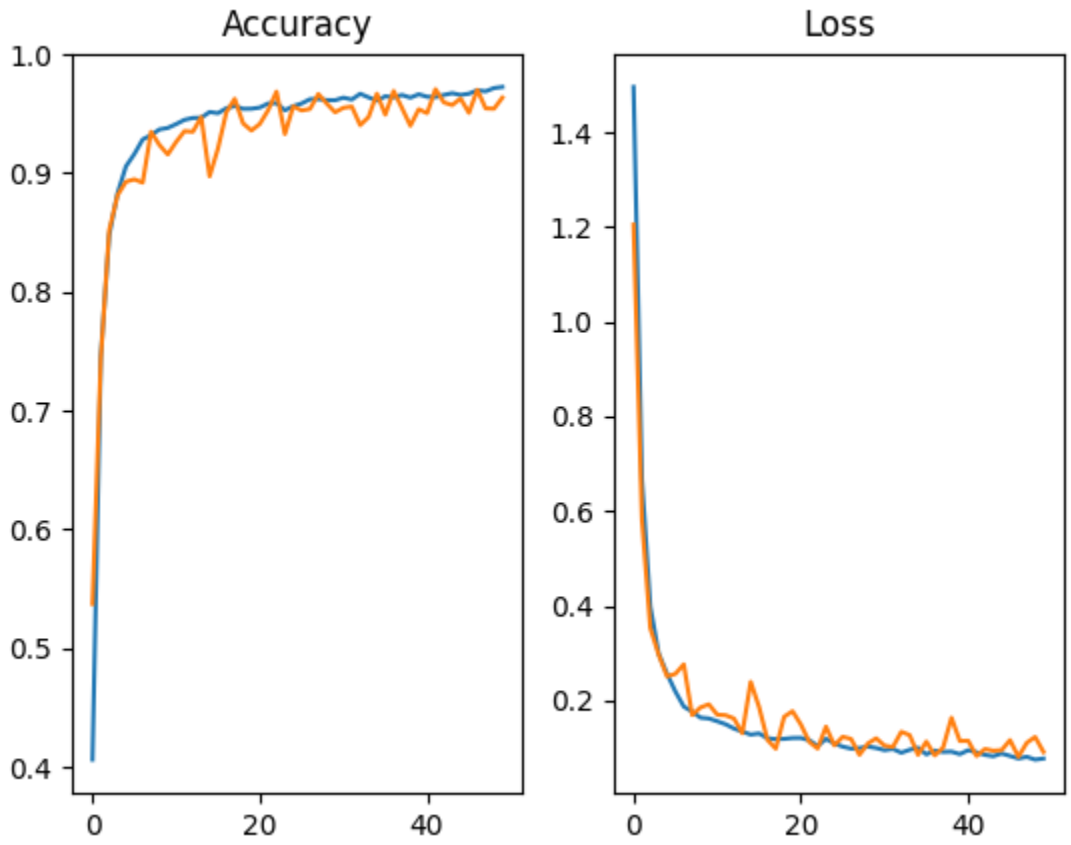

Figure 4 Performanse neuralne mreže kroz epohu obučavanja

4.2 Konfuzione matrice na trening I test skupu

Konfuziona matrica služi za evaluaciju performansi klasifikacionih modela na test skupu podataka. Elementi konfuzione matrice su:
TP: Podaci koji pripadaju pozitivnoj klasi I tacno su klasifikovan
TN: Podaci koji pripadaju negativnoj klasi I tačno su klasifikovani
FP: Podaci koji pripadaju pozitivnoj klasi I pogrešno su klasifikovani
FN: Podaci koji pripadaju negativnoj klasi I pogrešno su klasifikovani

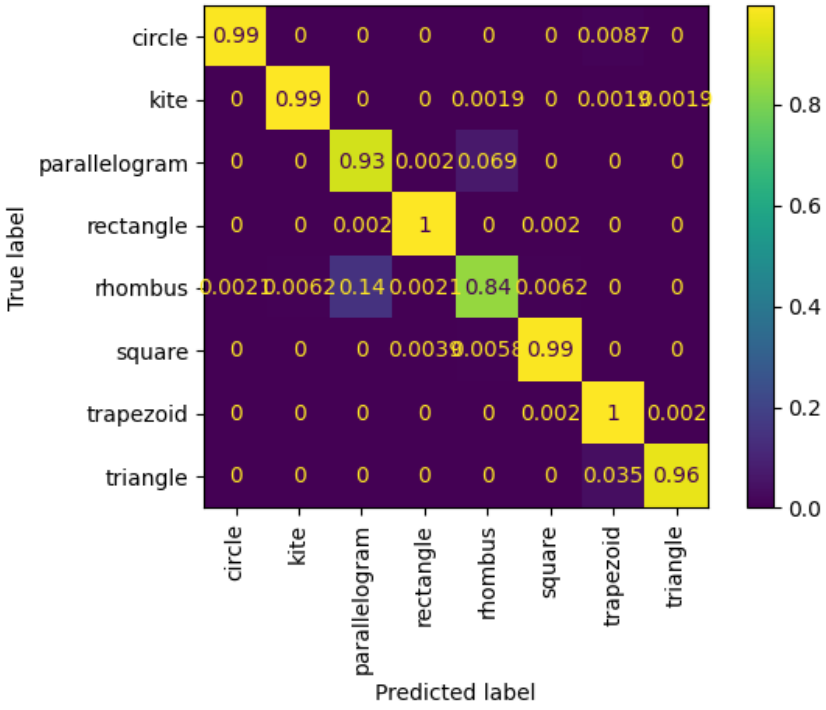

Figure 5 Konfuziona matrica test skupa

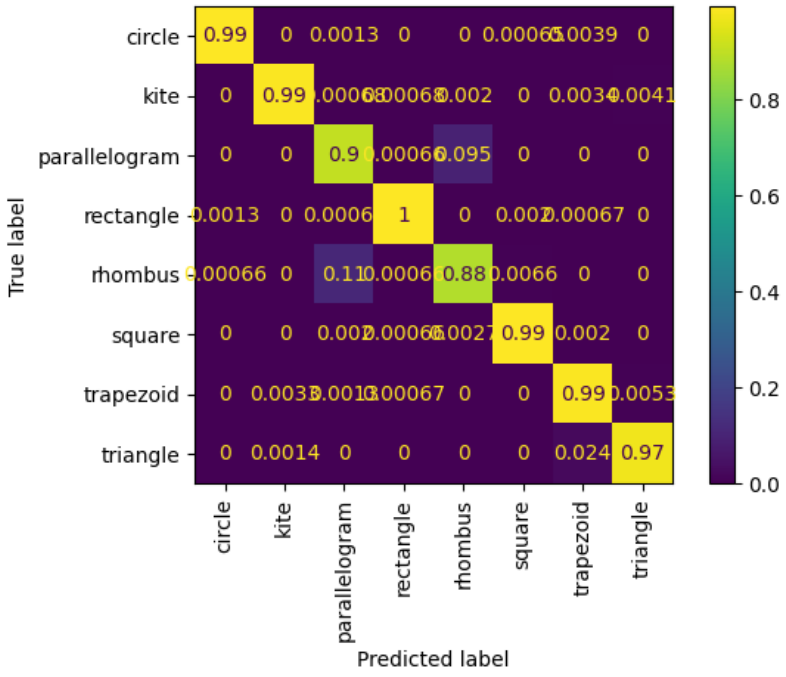

Figure 6 Konfuyiona matrica trening skupa

4.3 Primeri dobro I lose klasifikovanih primera dataset-a

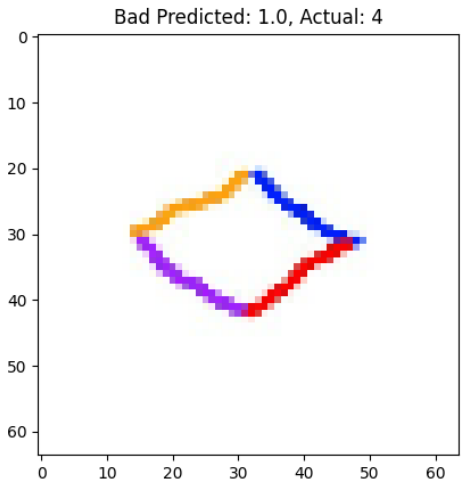

Figure 7 Primer loše klasifikovanog odbirka

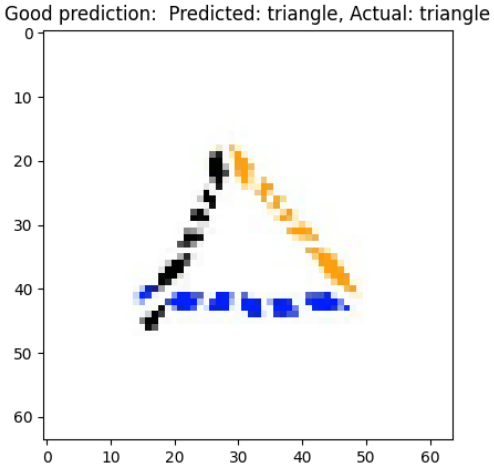

Figure 8 Primer dobro klasifikovanog odbirka

5. Zaključak

U našem istraživanju postigli smo uspešne rezultate u treniranju neuralne mreže primenom augmentacije podataka, skaliranja, normalizacije i tehnike ranog zaustavljanja. Dodatno, pronašli smo optimalnu vrednost hiperparametra za konstantu obučavanja, što je doprinelo postizanju optimalnih performansi i stabilnosti tokom treninga. Tačnost posle 50 epoha koju smo dobili testirajući primerke iz test skupa je