SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 935

KLASIFIKACIJA VRSTI PTICA IZ SLIKA TEMELJENA NA DUBOKOM UČENJU

Marela Arambašić

Zagreb, lipanj 2023.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 935

KLASIFIKACIJA VRSTI PTICA IZ SLIKA TEMELJENA NA DUBOKOM UČENJU

Marela Arambašić

Zagreb, lipanj 2023.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 10. ožujka 2023.

ZAVRŠNI ZADATAK br. 935

Pristupnica:	**Marela Arambašić (0036523688)**
Studij:	Elektrotehnika i informacijska tehnologija i Računarstvo
Modul:	Računarstvo
Mentor:	izv. prof. dr. sc. Alan Jović

Zadatak: **Klasifikacija vrsti ptica iz slika temeljena na dubokom učenju**

Opis zadatka:

Raspoznati vrstu ptice na temelju slike (fotografije) je izazovan zadatak zbog velikog broja vrsti ptica, kuta snimanja i individualnih značajki ptice. U ovom radu potrebno je opisati problem raspoznavanja ptica iz slika te predložiti programsko rješenje za klasifikaciju ptica iz slika. Rješenje se treba temeljiti na metodama dubokog učenja, pri čemu treba razmotriti postupke višeslojnog perceptrona i konvolucijske neuronske mreže uz korištenje prikladnih knjižnica za izgradnju modela (npr. Keras, PyTorch). Za vrednovanje modela treba razmotriti javno dostupne skupove podataka CUB-200-2011 i Birds 500 Species. U radu je također potrebno ostvariti jednostavnu web aplikaciju koja će omogućiti učitavanje slike ptice i njezinu klasifikaciju s naznakom pouzdanosti, a sve na temelju ugrađenog modela dubokog učenja.

Rok za predaju rada: 9. lipnja 2023.

STRANICA SA ZAHVALOM

Zahvaljujem roditeljima i obitelji te mentoru, izv. prof. dr. sc. Alan Joviću na pomoći pri izradu rada.

Sadržaj

Uvod 1

1. Skupovi podataka 2

2. Obični višeslojni perceptron 5

2.1. Neuronska mreža 5

2.2. Perceptron 6

2.3. Model 8

2.3.1. Inicijalizacija 8

2.3.2. Petlja učenja 10

2.3.3. Optimizacija 10

2.4. Učenje 11

2.5. Testiranje 12

3. Konvolucijske mreže 14

3.1. Konvolucijske mreže 14

3.2. Model 15

3.3. Priprema podataka 18

3.4. Učenje 20

3.5. Testiranje 21

4. Usporedba modela 23

5. Web aplikacija 24

5.1. Opis web aplikacije 24

5.2. Tehnologije i alati 25

5.2.1. Klijentska strana 25

5.2.2. Poslužiteljska strana 26

Zaključak 28

Literatura 29

Sažetak 30

Summary 31

Uvod

Na svijetu postoji između 9 i 10 tisuća vrsta ptica, od običnog goluba do šarenih žutoprsih tukana. Njihovo promatranje i klasificiranje je postao omiljeni hobi mnogima diljem svijeta. No, problem nastaje upravo u velikoj količini sličnih vrsta koje znatno otežavaju raspoznavanje bez pomoći računala.

Stoga se u ovom radu bavimo problematikom raspoznavanja ptica koristeći metode dubokog učenja. Prvo ćemo proučiti što su to višeslojni perceptroni i napraviti model sa što većom preciznošću. Zatim ćemo proučiti konvolucijske neuronske mreže, napraviti model te usporediti koeficijente preciznosti i točnosti tih dvaju modela. Konačno ćemo napraviti jednostavnu web stranica na kojoj možemo učitati sliku te dobiti njezinu klasifikaciju s naznakom pouzdanosti koristeći pouzdaniji od navedena dva modela.

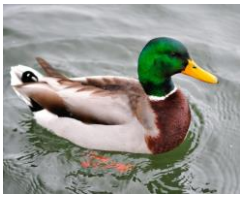
1. Skupovi podataka

Za potrebe učenja modela korišten je javno dostupan skup podataka CUB 200_2011. CUB 200_2011 sadrži 11788 slika raspoređenih u 200 kategorija, odnosno približno 59 slika u svakoj kategoriji.

U svrhu učenja korištene su 15 od 200 klasa dostupnih u skupu podataka zbog mogućnosti računala. Za učenje modela pomoću jednostavnog višeslojnog perceptrona podatci su u omjeru 9:1 podijeljeni u dvije grupe: podatci za učenje i podatci za testiranje. S druge strane, za učenje modela pomoću konvolucijskih neuronskih mreža podatci su podijeljeni u tri grupe: podatci za učenje, podatci za validaciju i podatci za testiranje. Za podatke za testiranje odvojeno je 10% ukupnih podataka, dok je ostatak podijeljen u preostale dvije grupe u omjeru 9:1. U tablici 1.1 navode se sve kategorije koje su korištene, broj dostupnih slika za svaku kategoriju te jedna slika iz CUB 200_2011 skupa podataka kao primjer kategorije.

Tablica 1.1 Opis kategorija

Kategorija (engl.)	**Broj dostupnih slika**	**Primjer kategorije**
Vermilion Flycatcher	60	
Evening Grosbeak	60	
Rose breasted Grosbeak	60	

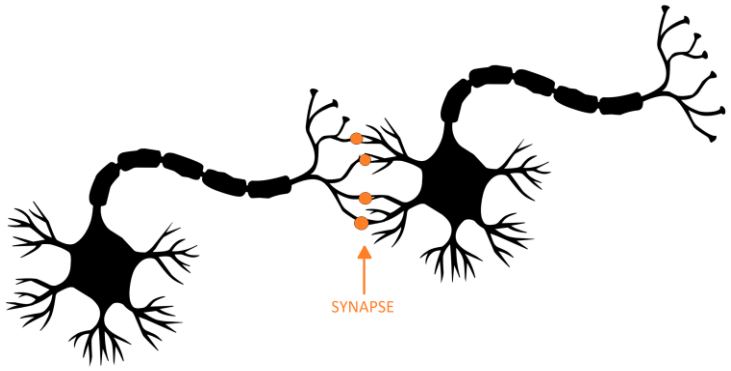
Slary backed Gull	50	
Rugous Hummingbird	60	
Green Violetear	60	
White breasted Kingfisher	60	
Mallard	60	
Brown Pelican	60	
White Pelican	50	
Horned Puffin	60	

White necked Raven	60	
Fox Sparrow	60	
Cape Glossy Starling	60	
Red cockaded Woodpecker	58	

2. Obični višeslojni perceptron

2.1. Neuronska mreža

Neuronske mreže su temelj dubokog učenja. Inspirirane su ljudskim mozgom koji sadrži oko 10 milijardi živčanih stanica ili neurona i oko 60000 milijardi međuspojeva. Uloga neurona je prihvaćanje, obrađivanje i odašiljanje podataka. Tijekom života uspostavljaju se veze među neuronima, sinapse kao što je prikazano na slici 2.1, koje omogućuju komunikaciju.



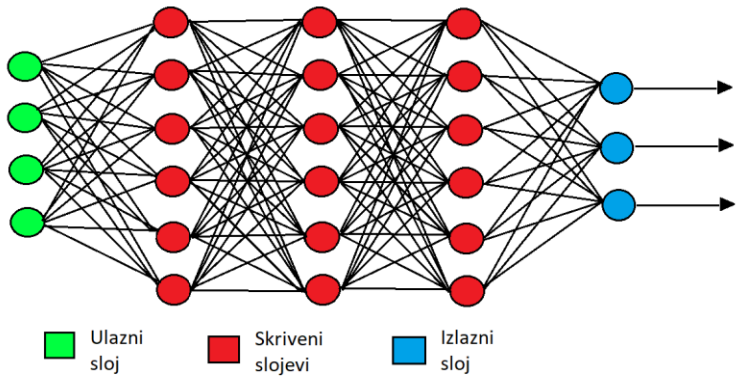
Slika 2.1 Veze između neurona [1]

Umjetne neuronske mreže pokušavaju imitirati način komunikacije između neurona. Mreža se sastoji od čvorova (neurona) podijeljenih u slojeve. Čvorovi se međusobno spajaju vezama pomoću težina i pragova koje se koriste za spremanje znanja. Težine predstavljaju važnost veza svakog ulaza u predviđanju konačnog izlaza dok prag označava vrijednost potrebnu za aktivaciju određenog čvora.

Svaki čvor je model linearne regresije koji sadrži ulaz, težinu, prag i izlaz. Za izračun izlaza ulazi se množe s pripadajućom težinom, potom se zbroje te propuste kroz nelinearnu aktivacijsku funkciju koja ograničava izlaz na vrijednosti između 0 i 1. Ako je izlaz veći od praga povezanog čvora u sljedećem sloju, taj se čvor aktivira i započinje se novi izračun izlaza.

Aktivacijske funkcije se koriste za sprječavanje linearnosti. Naprimjer, postoje sigmoida, tanh, softmax, ReLU, itd [2].

Slojevi se dijele na ulazni, izlazni i skrivene slojeve (slika 2.2). Ulazni sloj prima podatke i prenosi ga ostatku mreže. Izlazni sloj sadrži izlaz rješenja. Skriveni slojevi određuju kompleksnost mreže primjenom aktivacijskih funkcija na ulazne podatke. Mreža sadrži jedan ili više skrivenih slojeva.



Slika 2.2 Slojevi neuronskih mreža

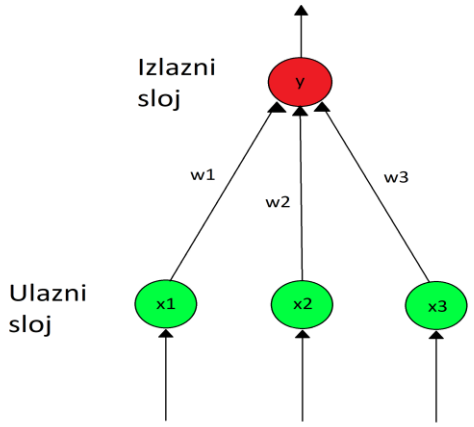
Neuronske mreže zahtijevaju učenje za poboljšanje putova i povećanje preciznosti. Pri učenju potrebno je paziti na probleme poput prenaučenosti (engl. *overfitting*), kada mreža točno predviđa podatke za učenje, no ne i nove podatke.

Većina neuronskih mreža su tzv. „feedforward“ mreže odnosno mreže čiji su putovi usmjereni u samo jednom smjeru, od ulaza do izlaza, bez petlji.

2.2. Perceptron

Perceptron je jednostavna *feedforward* neuronska mreža korištena za rješenje raznih problema kao što su klasifikacija uzoraka i interpolacija.

Jednoslojni perceptron (slika 2.3) služi za klasifikaciju ulaznih podataka u dvije klase. Pri klasifikaciji uzorci moraju biti linearno separabilni i ležati na suprotnim stranama hiperravnine. Ulaz je vektor, a izlaz označava pripadanje određenoj klasi. Izlaz se računa množenjem ulaznog podatka s težinom. Jednoslojni perceptron sadrži samo ulazni i izlazni sloj.



Slika 2.3 Skica jednoslojnog perceptrona

Višeslojni perceptron ima po jedan ulazni i izlazni sloj i jedan ili više skrivenih slojeva. Broj neurona može varirati u određenim slojevima. Učenje kod višeslojnog perceptrona je postupak pronalaženja težina tako da srednja pogreška bude minimalna. Takvo učenje pod nadzorom se odvija pomoću algoritma s povratnom propagacijom pogreške.

Pri inicijalizaciji mreže sve težine su postavljene na slučajne brojeve između -1 i 1. Nakon prolaza kroz mrežu dobiveni rezultat se usporedi s očekivanim izlazom. Zatim se započinje povratna propagacija pogreške. Postoje dva prolaza: prolaz unaprijed i povratni prolaz. Pri prolazu prema naprijed izlazi se računaju počevši od ulaznog prema izlaznom sloju, a signali se računaju prema izrazima za internu aktivnost i izlaz neurona kao što je prikazano na slici 2.4. Tijekom povratnom prolaza računaju se signali pogreške od izlaznog sloja prema ulazu rekurzivno te se radi korekcija težina. Proces se ponavlja određeni broj prolaza (epoha).

$$v_j(n) = \sum_{i=0}^{p} w_{ji}(n) y_i(n)$$

$$y_j(n) = \varphi_j(v_j(n))$$

Slika 2.4 Formula za računanje signala u prolazu unaprijed

Višeslojni perceptron ima razne primjene kao što su: analiza slika i signala, medicinska dijagnostika, prepoznavanje govora i sl.

2.3. Model

Model višeslojnog perceptrona gradimo pomoću modula Lightning iz knjižnice PyTorch.

Modul Lightning organizira PyTorchov kod u šest sekcija [3]:

- Inicijalizacija
- Petlja učenja
- Petlja validacije
- Petlja testiranja
- Petlja predviđanja
- Optimiranje

Na slici 2.5 vidljiv je prikaz svih uvezenih knjižnica.

```
import pytorch_lightning as pl
import torch
from PIL import Image
from torch.utils.data import DataLoader
from torchvision import datasets, transforms
import torch.nn as nn
```

Slika 2.5 Slika uvezenih knjižnica

2.3.1. Inicijalizacija

U funkciji „init“ (slika 2.6) definirani su slojevi koji se slaži uzastopno. Model se sastoji od 3 skrivena sloja, 1 ulaznog i 1 izlaznog sloja. U ulaznom sloju kao ulaz je definirana slika dimenzija 224 * 224 te izlaz kao 128 tenzora. Zatim dva skrivena sloja koja kao ulaz primaju određeni broj tenzora, a kao izlaz vraćaju upola manji broj tenzora. Konačno izlazni sloj prima 32 tenzora na ulazu i vraća 15 tenzora na izlazu odnosno broj različitih klasa koje pokušavamo klasificirati.

```
def __init__(self):
    super().__init__()
    pl.seed_everything(42)
    self.layers = nn.Sequential(
        nn.Linear(224 * 224 * 3, 128),
        nn.ReLU(),
        nn.Linear(128, 64),
        nn.ReLU(),
        nn.Linear(64, 32),
        nn.ReLU(),
        nn.Linear(32, 15)
    )
    self.ce = nn.CrossEntropyLoss()
```

Slika 2.6 Kod funkcije *__init__*

Kao aktivacijsku funkciju koristimo ReLU (slika 2.7). ReLU je nelinearna aktivacijska funkcija. Najčešće je korištena funkcija u dubokom učenju danas. Pozitivni brojevi se vraćaju kao pozitivni, a negativni se vraćaju kao 0.

$$\mathrm{ReLU}(x) = (x)^+ = \max(0, x)$$

Slika 2.7 Formula aktivacijske funkcije ReLU

Za definiranje gubitka koristimo funkciju unakrsne entropije (*CrossEntropyLoss*, CEL) prikazanu na slici 2.8. CEL se koristi pri rješavanju problema s klasifikacijom C klasa. Funkcija definira broj koji označava prosječne razlike između predviđenih izlaza i stvarnih izlaza. Cilj modela je minimizacija tog broja.

$$\ell(x, y) = L = \{l_1, \dots, l_N\}^\top, \quad l_n = -w_{y_n} \log \frac{\exp(x_{n,y_n})}{\sum_{c=1}^{C} \exp(x_{n,c})} \cdot 1$$

Slika 2.8 PyTorchova funkcija CEL

U funkciji „forward“ (slika 2.9) definiramo prolaz kroz neuronsku mrežu. Tenzor *x* pretvaramo u 2D tenzor kako bi omogućili rad s CEL-om.

```
def forward(self, x):
    x = x.view(x.size(0), -1)
    return self.layers(x)
```

Slika 2.9 Kod funkcije forward

2.3.2. Petlja učenja

U funkciji *training_step* (slika 2.10) poduzorak koji se daje na ulaz prilikom učenja, *batch*, razdvajamo na varijable *x* i *y*. Tenzor *x* predstavlja ulazne podatke u tom poduzorku te ga je potrebno, kao i u funkciji *forward,* pretvoriti u 2D tenzor. Funkcijom *self.layers* taj tenzor se propušta kroz sve slojeve modela i kao rezultat dobiva se izlaz. Gubitak izračunavamo pomoću prethodno definirane funkcije *CrossEntropyLoss* koristeći izlaz *y_hat* i oznake *y*.

```
def training_step(self, batch, batch_idx):
    x, y = batch
    y_hat = self.layers(x.view(x.size(0), -1))
    loss = self.ce(y_hat, y)
    self.log("train_loss", loss, on_step=True, on_epoch=True, prog_bar=True, logger=True)
    return loss
```

Slika 2.10 Kod funkcije koraka učenja

U svrhu praćenja podataka tijekom učenja koristimo metodu *self.log()* koja vraća gubitak tijekom svakog koraka, za svaku epohu. Korak simbolizira ažuriranje težina nakon što jedan *batch* podataka prođe kroz mrežu, dok je epoha potpuni prolaz svih podataka kroz mrežu. Odnosno, jedna epoha sadrži n koraka. Za vizualizaciju podataka koristi se traka za napredak dostupna u knjižnici PyTorch Lightning.

Funkcija „training_step“ vraća gubitak.

2.3.3. Optimizacija

Kao konačan korak u definiranju modela definira se optimizator u funkciji *configure_optimizers* (slika 2.11). Koristimo optimizator Adam sa stopom učenja (engl. *learning rate*) od 0.00005.

```
def configure_optimizers(self):
    optimizer = torch.optim.Adam(self.parameters(), lr=0.00005)
    return optimizer
```

Slika 2.11 Kod konfiguracije optimizatora

2.4. Učenje

Da bi se započelo učenje prvo je potrebno pristupiti podatcima za učenje kao što je prikazano na slici 2.12. Podatci se nalaze u odvojenim direktorijima. Put do glavnog direktorija definiran je u varijabli *directory*. Za stvaranje skupova podataka iz navedenih direktorija koristi se PyTorchova klasa *dataset*. Nad slikama se provodi transformacija. Slike se obrezuju na dimenzije 224x224 te se pretvaraju u tenzore. Zatim pomoću klase *dataloader* omogućava se iteracija nad navedenim skupovima gdje je veličina poduzorka (engl. *batch size*) jednaka 8. Naredbom *shuffle*=True osigurava se da su podatci izmiješani prilikom provedbe sljedeće epohe.

```
dataset = datasets.ImageFolder(directory, transform=transforms.Compose([transforms.CenterCrop(224),transforms.ToTensor()]))
dataloader = torch.utils.data.DataLoader(dataset, batch_size=8, shuffle=True)
```

Slika 2.12 Kod učitavanja podataka

Postavlja se učenje modela tako da se dinamično optimira korištenje memorije i da se provodi učenje na CPU-u. Maksimalni broj epoha je postavljen na 40. Postupak postavljanja prikazan je na slici 2.13.

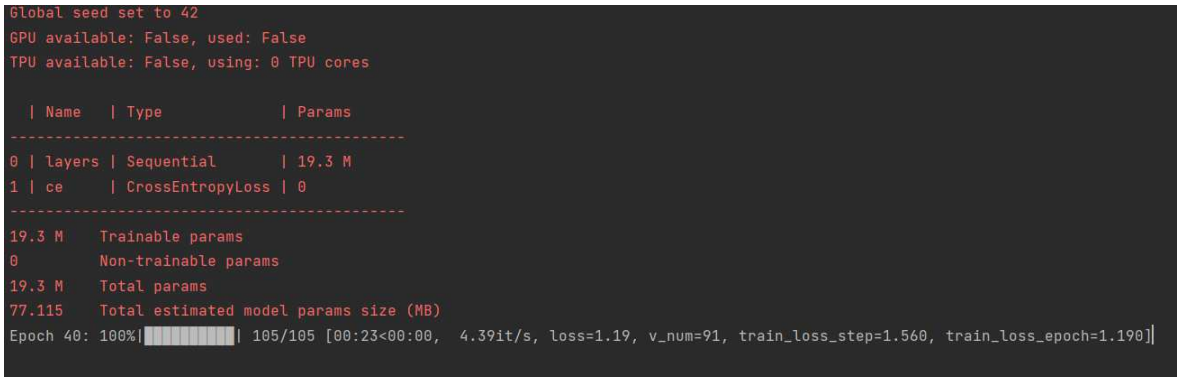
```
trainer = pl.Trainer(auto_scale_batch_size='power', gpus=0, deterministic=True, max_epochs=40)
trainer.fit(mlp, dataloader)

mlp.eval()
```

Slika 2.13 Kod učenja modela MLP-a

Nakon evaluacije model se sprema u .pth obliku pomoću naredbe *torch.save(mlp.state_dict(), 'model.pth').*

Pri provedbi koda prikazuje se opis modela te praćenje učenja gdje je naveden broj trenutne epohe, traka napretka, trenutna vrijednost funkcije gubitka i sl. Ispis je prikazan na slici 2.14.



Slika 2.14 Ispis pri pokretanju učenju modela

2.5. Testiranje

Skupovi podataka su odvojeni na skup za učenje i skup za testiranje. Skup za testiranje sadrži 10% originalnog skupa. Učitavanje se provodi na isti način kao i učitavanje skupa za učenje.

Testiranje se provodi nakon učenja tako što se iterira se skupom za testiranje (slika 2.15). Koristi se funkcija „torch.max()“, koja prima predviđanja trenutnog skupa slika, „mlp(images).data“.

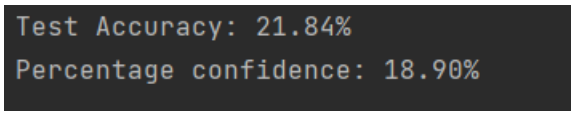
Cilj predviđanja je računanje koeficijenta pouzdanosti i točnosti, stoga u petlji pratimo ukupni broj oznaka u trenutnom skupu, prosječnu pouzdanost i broj točno izračunatih oznaka. Nakon provedbe petlje računamo točnost modela i koeficijent pouzdanosti.

```
with torch.no_grad():
    for images, labels in dataloader:
        confidence, predicted = torch.max(mlp(images).data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()
        percentage_confidence += confidence.mean() * 100

accuracy = correct / total
percentage_confidence_total = percentage_confidence / total
```

Slika 2.15 Kod za testiranje skupa podataka

Nakon provođenja učenja i testiranja tijekom 40 epoha dobivena je točnost od 21,84% i koeficijent pouzdanosti od 18,90% (slika 2.16). Kako bi se poboljšali postotci točnosti i pouzdanosti potrebno je više podataka u skupovima.



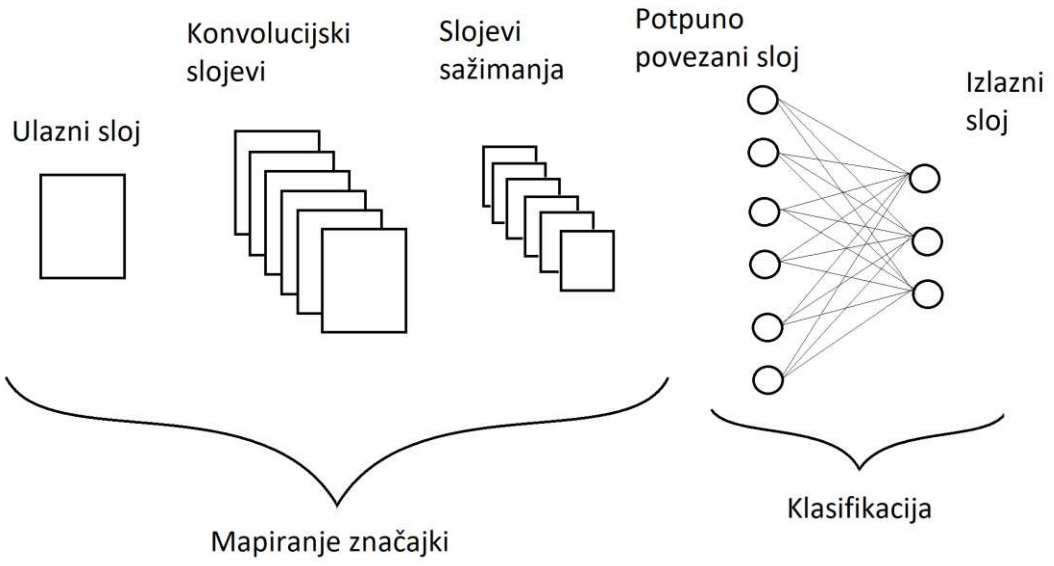
Slika 2.16 Rezultati testiranja

3. Konvolucijske mreže

3.1. Konvolucijske mreže

Konvolucijske mreže (slika 3.1) su vrste neuronskih mreža koje se često koriste pri klasifikaciji slika, prepoznavanju uzoraka, računalnom vidu i sl. Koriste principe množenja matrica za prepoznavanje uzoraka.

Slične su *feedforward* mrežama kao što su višeslojni perceptron. Poput perceptrona sastoje od po jednog ulaznog i izlaznog sloja te jednog ili više skrivenih slojeva. Međutim, razlikuju se u upotrebi konvolucijskih slojeva, slojeva sažimanja i potpuno povezanih slojeva. Konvolucijski slojevi i slojevi sažimanja provode funkciju mapiranja značajki, dok potpuno povezani slojevi pretvaraju te značajke u konačni izlaz poput klasifikacije. Mreže se najčešće sastoje od izmjeničnih konvolucijskih slojeva i slojeva sažimanja te na kraju imaju jedan ili više potpuno povezanih slojeva.



Slika 3.1 Primjer konvolucijske neuronske mreže

Učenje je proces pronalaženja filtra (kernel) i težina s ciljem minimiziranja razlika između previđenog i stvarnog izlaza. Za učenje najčešće se koristi algoritam propagacija pogreške unazad.

Konvolucijski sloj sastoji se od filtra istih dubina kao ulaz, no pretežito manjih dimenzija. Npr. na slici dimenzija 5x5x1 možemo primijeniti filtar dimenzija 3x3x1 da bismo dobili značajku dimenzija 3x3x1. Filtar se kreće po slici desno za određenu vrijednost dok ne prođe cijelu dužinu. Ta vrijednost se zove korak pomaka filtra (S u navedenoj formuli na slici 3.2). Zatim se spusti do početka s lijeve strane i proces se ponavlja. Prilikom pomicanja filtera dolazi do množenja matrica. Kao rezultat dobijemo značajku poput krajeva, boje, orijentacija gradijenta i sl. Svaki konvolucijski sloj može sadržavati više filtara.

$$W_{out} = \frac{W - F + 2P}{S} + 1$$

Slika 3.2 Formula za računanje širine izlaza

Sloj sažimanja je zaslužan za smanjivanje prostorne veličine značajke dobivene konvolucijskim slojem u svrhu smanjenja potrebne procesorske snage za obradu podataka. Postoje dvije uobičajene vrste sažimanja: maksimalno sažimanje i prosječno sažimanje. Maksimalno sažimanje vraća maksimalnu vrijednost dijela slike prekrivenog filterom, dok prosječno sažimanje vraća prosjek svih vrijednosti dijela slike prekrivenog filterom.

Neke od vrsta dostupnih CNN arhitektura su [4]:

-LeNet

-ResNet

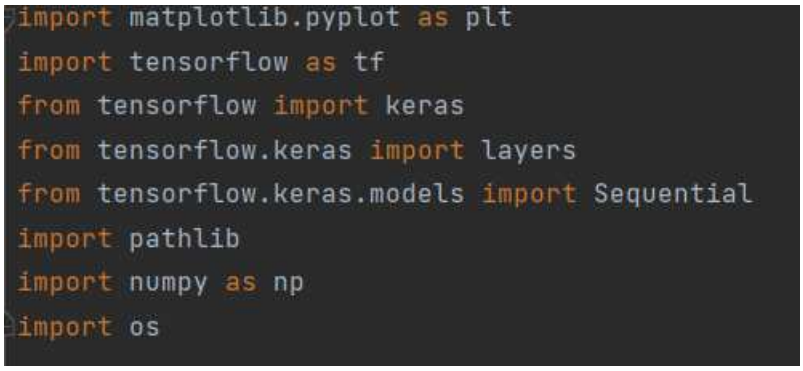
-ZFNet

-GoogLeNet

3.2. Model

Model konvolucijske mreže gradimo pomoću modula Keras iz knjižnice TensorFlow.

Na slici 3.3 vidljiv je prikaz svih uvezenih knjižnica.



Slika 3.3 Slika uvezenih knjižnica

Sekvencijalni Kerasov model se sastoji od 5 slojeva konvolucije (Conv2D), 5 slojeva sažimanja (MaxPooling2D) i jednog potpuno povezanog sloja (Dense) kao što je prikazano na slici 3.4.

Konvolucijski slojevi imaju po 16, 32, 64, 128 i 256 filtara veličine 3x3. Varijabla dopune (engl. *padding*) omogućava konzistentne dimenzije ulaza i izlaza. Kao aktivacijska funkcija ponovno se koristi ReLU.

Slojevi sažimanja smanjuju prostorne dimenzije.

Nakon prolaska kroz konvolucijske slojeve i slojeve sažimanja nastaju značajke. Funkcija izravnavanja (engl. *flatten*) pretvara 2D značajke u jednodimenzijske vektore.

Konačno, potpuno povezani sloj koristi značajke i proizvodi izlaze u smislu predviđenih klasa.

```
model = Sequential([
  data_augmentation,
  layers.Rescaling(1./255),
  layers.Conv2D(16, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(32, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(128, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(256, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Dropout(0.2),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(num_classes, name="outputs")
])
```

Slika 3.4 Kod sekvencijalnog modela

Funkcija isključivanja neurona (engl. *dropout*) se uvodi kao mjera za smanjenje prenaučenosti, tj. pretjeranog prilagođavanja. Ona nasumično isključuje 20% izlaznih jedinica, a iznos isključivanja može se i mijenjati. U ovom slučaju primijenjena je na zadnji sloj sažimanja.

Uz funkciju isključivanja postoji još metoda za smanjenje prenaučenosti. Jedna od tih metoda je poboljšavanje (engl. *augmentation*) podataka koji se postiže tehnikama povećanja količine podataka kao što su rotiranje, zrcaljenje i povećavanje (slika 3.5). Veći broj podataka najčešće osigurava bolju generalizaciju.

```
data_augmentation = keras.Sequential(
  [
    layers.RandomFlip("horizontal",
                      input_shape=(img_height,
                                   img_width,
                                   3)),
    layers.RandomRotation(0.1),
    layers.RandomZoom(0.1),
  ]
)
```

Slika 3.5 Kod za povećanje podataka

Pri kompajliranju (slika 3.6) kao optimizator navodi se Adam, a za funkciju gubitka upotrebljava se kategorijska unakrsna entropija za rijetke podatke (engl. *SparseCategoricalCrossentropy*, SCC).

```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

Slika 3.6 Kod sekvencijalnog modela

3.3. Priprema podataka

Podatci se učitavaju pomoću naredbe *pathlib.Path(directory)* gdje *directory* označava putanju do direktorija s kategorijama na računalu.

Pomoću Kerasove funkcije *image_dataset_from_directory* zadani podatci se pretvaraju u skupove podataka za daljnju obradu te se dijele na skupove za učenje i skupove za validaciju u omjeru 9:1 kao što je prikazano na slici 3.7. Varijabla *seed* osigurava miješanje podataka pokretanjem nove epohe. Dimenzije slika iznose 256x256. Konačno, veličina poduzorka u određenom skupu iznosi 16.

```
data_dir = pathlib.Path(directory)

batch_size = 16
img_height = 256
img_width = 256
validation_split = 0.1
seed = 234

train_ds = tf.keras.utils.image_dataset_from_directory(
  data_dir,
  validation_split=validation_split,
  subset="training",
  seed=seed,
  image_size=(img_height, img_width),
  batch_size=batch_size)

val_ds = tf.keras.utils.image_dataset_from_directory(
  data_dir,
  validation_split=validation_split,
  subset="validation",
  seed=seed,
  image_size=(img_height, img_width),
  batch_size=batch_size)
```

Slika 3.7 Kod za učitavanje i podjelu podataka

Zbog optimiranja podataka koristi se prethodno dohvaćanje nad skupovima podataka za učenje i validaciju (slika 3.8):

- *Dataset.cache* -> koristi se pri upotrebi manjih skupova podataka za čuvanje slika u memoriji nakon prolaska prve epohe.
- *Dataset.prefetch* -> preklapa učitavanje podataka i prolazak kroz model

Zatim se primjenjuje normalizacija podataka iz RGB vrijednosti u rasponu od 0-255 na raspon od 0-1 korištenjem naredbe *Rescaling*. Normalizacija se može primijeniti na skup podataka ili se može uključiti kao sloj unutar definicije modela.

```
train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=tf.data.AUTOTUNE)

normalization_layer = layers.Rescaling(1./255)

normalized_ds = train_ds.map(lambda x, y: (normalization_layer(x), y))
image_batch, labels_batch = next(iter(normalized_ds))
```

Slika 3.8 Kod za optimiranje i normalizaciju skupova podataka

3.4. Učenje

Model se uči pomoću prethodno definiranih skupova podataka za učenje i validacije (slika 3.9) te traje 30 epoha. Nakon evaluacije model se sprema u h5 obliku pomoću naredbe *model.save(„model.h5“).*

```
epochs = 30
history = model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=epochs
)
```

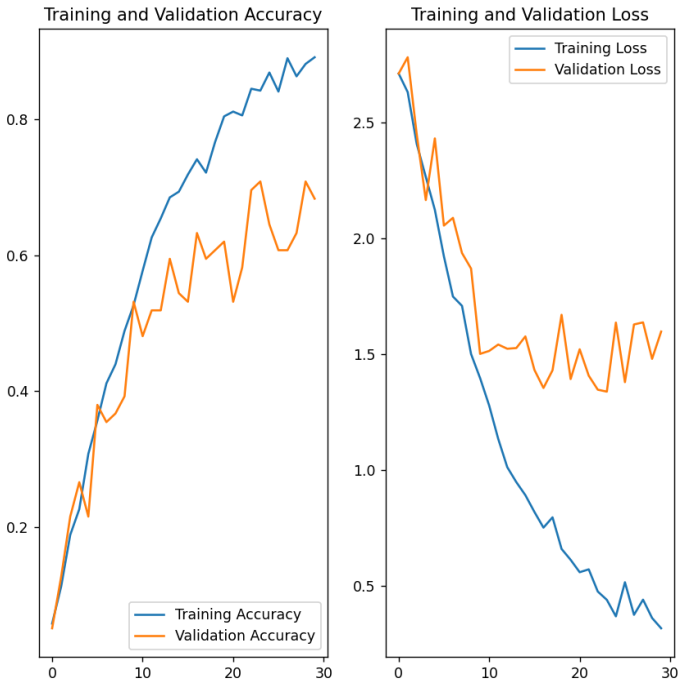
Slika 3.9 Kod za učenje modela

Tijekom učenja ispisuju se podatci o broju koraka, broju epohe, gubitku, preciznosti i sl. (slika 3.10).

```
Epoch 1/30
45/45 [==============================] - 39s 872ms/step - loss: 2.6653 - accuracy: 0.1053 - val_loss: 2.5228 - val_accuracy: 0.1646
Epoch 2/30
45/45 [==============================] - 38s 850ms/step - loss: 2.4180 - accuracy: 0.1770 - val_loss: 2.3974 - val_accuracy: 0.1519
Epoch 3/30
45/45 [==============================] - 41s 903ms/step - loss: 2.2335 - accuracy: 0.2458 - val_loss: 2.2232 - val_accuracy: 0.2532
Epoch 4/30
45/45 [==============================] - 37s 811ms/step - loss: 2.0808 - accuracy: 0.2921 - val_loss: 2.0772 - val_accuracy: 0.3038
Epoch 5/30
45/45 [==============================] - 37s 817ms/step - loss: 1.8855 - accuracy: 0.3483 - val_loss: 2.3879 - val_accuracy: 0.3038
Epoch 6/30
45/45 [==============================] - 38s 839ms/step - loss: 1.8591 - accuracy: 0.3904 - val_loss: 1.8248 - val_accuracy: 0.4051
```

Slika 3.10 Proces učenja modela

Na slici 3.11 prikazani su grafovi točnosti i gubitka. Na grafovima vidi se da je model najviše naučio u prvih 10 epoha. Nakon 10. epohe točnost validacije prestaje naglo rasti kao točnost treniranja i gubitak validacije stagnira na oko 1.5. Važno je pratiti razliku između treniranja i validacije zbog problema prenaučenosti. U slučaju kada bi točnost treniranja nastavila rasti ili ostala visoko, a točnosti validacije počela padati to bi značilo da model prepoznaje samo podatke na kojima je treniran.



Slika 3.11 Graf rezultata učenja

3.5. Testiranje

Varijabla *image_files* sadrži putanju do direktorija gdje se nalaze skupovi za testiranje odvojeni u datotekama. Iterira se po svim slikama (slika 3.12). Svaka slika se učitava, transformira u potrebne dimenzije, pretvara u *array* i dodaje joj se dimenzija, koja označava *batch* veličinu 1 potrebnu za pokretanje TensorFlow-a. Pozivom metode *predict* nad modelom dobije se tenzor s rezultatima predviđanja, no da bi se dobile vjerojatnosti potrebno je pozvati metodu *softmax.* Vjerojatnost dobivene klasifikacije se izlučuje metodom *max.* Pouzdanost se računa kao zbroj svih vjerojatnosti podijeljen s ukupnim brojem slika, dok se točnost računa kao broj točnih predviđanja podijeljen s ukupnim brojem slika.

```
for image_file in image_files.glob("*"):
    for filename in os.listdir(image_file):
        image_path = os.path.join(image_file, filename)
        img = tf.keras.utils.load_img(image_path, target_size=(img_height, img_width))
        img_array = tf.keras.utils.img_to_array(img)
        img_array = tf.expand_dims(img_array, 0)

        predictions = model.predict(img_array)
        score = tf.nn.softmax(predictions[0])
        total_score += 100 * np.max(score)
        total_images += 1

        if(current_number == np.argmax(score)): true += 1
    current_number += 1

confidence = total_score / total_images
accuracy = true / total_images * 100
```

Slika 3.12 Kod petlje za testiranje

Nakon provođenja učenja i testiranja tijekom 30 epoha dobivena je točnost od 66,67% i koeficijent pouzdanosti od 79,47 % (slika 3.13).

```
Confidence: 79.47%

Accuracy: 66.67%
```

Slika 3.13 Rezultati testiranja

4. Usporedba modela

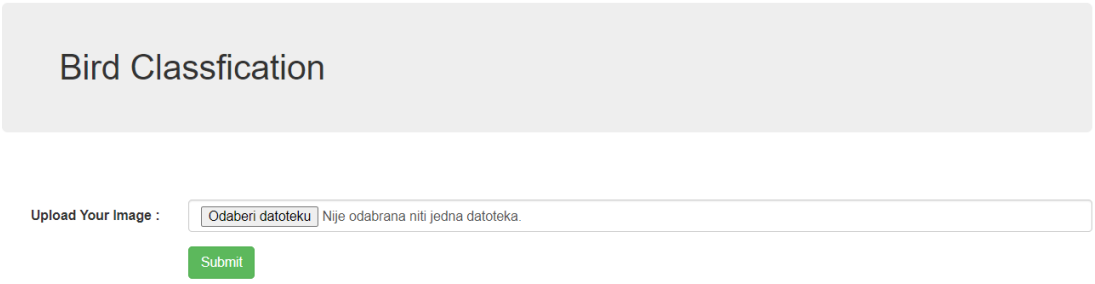
Modelom jednostavnog višeslojnog perceptrona predviđali smo 15 kategorija te kao rezultat dobili točnost od 21,84% i koeficijent pouzdanosti od 19,80%. S druge strane modelom konvolucijskih mreža predviđali smo 15 kategorija te kao rezultat dobili točnost od 66,67% i koeficijent pouzdanost od 79,47 %.

Veći koeficijent pouzdanosti i točnost smo dobili koristeći model konvolucijskih mreža stoga ćemo taj model implementirati u web stranicu.

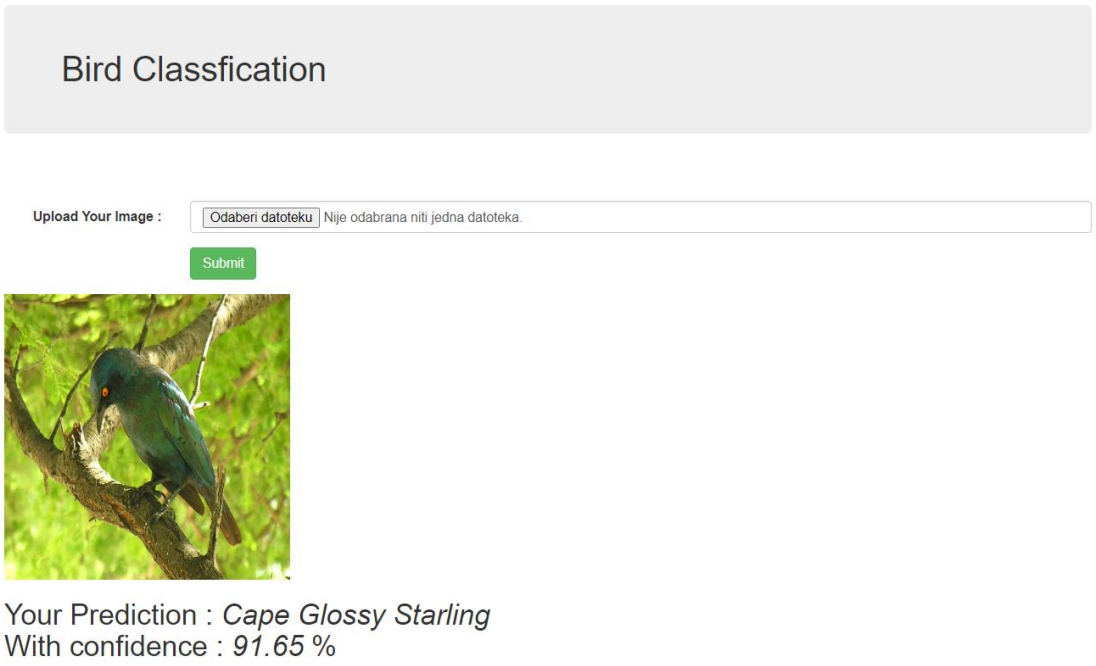
5. Web aplikacija

5.1. Opis web aplikacije

Web stranica sadrži klijentsku i korisničku stranu. Na stranici postoji mogućnost učitavanja slike ptice (slika 5.1). Pomoću ugrađenog modela za klasifikaciju ptica konvolucijskim mrežama, slika se klasificira te se kao povratna informacija korisniku prikazuje učitana slika, naziv vrste ptice te naznaka pouzdanosti (slika 5.2).



Slika 5.1 Izgled stranice prije učitavanja slike



Slika 5.2 Izgled stranice nakon obavljanja predviđanja

5.2. Tehnologije i alati

5.2.1. Klijentska strana

Za izgradnju klijentskog dijela web stranice korišten je prezentacijski jezik HTML. Definirane su forme za učitavanje slika i predaju informacije poslužiteljskoj strani aplikacije. Informacija se predaje pritiskom na gumb *Submit* te šalje POST metodom.

Klijentska strana također provjerava je li primila od poslužiteljske strane podatke o predviđanju te ih sukladno prikazuje korisniku (slika 5.3).

```
{% if prediction %}
<img src="{{img_path}}" height="300px" width="300px">
<h2> Your Prediction   : <i> {{prediction}} </i>
     <br>With confidence   : <i> {{confidence}} </i>%</h2>

{% endif %}
```

Slika 5.3 Kod za ispis rezultata predviđanja

5.2.2. Poslužiteljska strana

Za izgradnju poslužiteljskog dijela web stranice korišten je programski jezik Python i radni okvir za web stranice Flask.

Primanje slike, predviđanje i slanje rezultata predviđanja odvija se na ruti /submit (slika 5.4). Nakon što poslužiteljska strana primi POST zahtjev na ruti */submit* , pristupa se slici pomoću *request.files* metode.

```
@app.route("/submit", methods=['GET', 'POST'])
def get_output():
    if request.method == 'POST':
        img = request.files['my_image']
        img_path = "static/" + img.filename
        img.save(img_path)

        score = predict_label(img_path)
        prediction_name = dic[np.argmax(score)]
        formatted_confidence = "{:.2f}".format(100 * np.max(score))

        return render_template("index.html", prediction=prediction_name, confidence=formatted_confidence, img_path=img_path)
    return render_template("index.html")
```

Slika 5.4 Kod na ruti */submit*

Predviđanje se obavlja pomoću funkcije *predict_label* (slika 5.5). Slika se učitava iz putanje i mijenja joj se veličina jer nam je veličina slike 256x256 definirana u modelu. Kao i u poglavlju 3.5 poziva se metoda *predict* nad modelom i metoda *softmax* nad dobivenim rezultatom.

```
model = load_model('model.h5')

def predict_label(img_path):
    img = tf.keras.utils.load_img(
        img_path, target_size=(256, 256)
    )
    img_array = tf.keras.utils.img_to_array(img)
    img_array = tf.expand_dims(img_array, 0)
    predictions = model.predict(img_array)
    score = tf.nn.softmax(predictions[0])

    return score
```

Slika 5.5 Kod za funkciju *predict_label*

Nakon povratka iz funkcije *predict_label* iz rezultata se iščitava ime klase pomoću metode *argmax* i rječnika definiranog na slici 5.6 te naznaka pouzdanosti pomoću

metode *max*. Pri završetku funkcije *get_output* ponovno se r*ender-a* stranica *index.html* i klijentskoj strani se šalju podatci o imenu klase, naznaci pouzdanosti i putanji do slike.

```
dic = {0: 'Vermilion Flycatcher', 1: 'Evening Grosbeak', 2: 'Rose breasted Grosbeak', 3: 'Slaty backed Gull',
       4: 'Rufous Hummingbird', 5: 'Green Violetear', 6: 'White breasted Kingfisher', 7: 'Mallard',
       8: 'Brown Pelican', 9: 'White Pelican', 10: 'Horned Puffin', 11: 'White necked Raven',
       12: 'Fox Sparrow', 13: 'Cape Glossy Starling', 14: 'Red cockaded Woodpecker'}
```

Slika 5.6 Rječnik s imenima klasa

Pythonov poslužitelj pokrećemo na localhost-u naredbom *python app.py*.

Zaključak

U ovom radu opisan je problem raspoznavanja ptica iz slika. Kao rješenje problema ponuđene su dvije metode dubokog učenja, jednostavni višeslojni perceptron i konvolucijske mreže.

Svaka od metoda je pobliže opisana te je napravljen model koristeći programski jezik Python. Konačno metoda s najvećom naznakom pouzdanosti implementirana je u jednostavnoj web stranici.

Metoda konvolucijskih mreža se pokazala boljom za rješenje problema.

Kroz rad naučila sam principe dubokog učenja te optimiranje modela za zadani zadatak.

Modeli bi se mogli poboljšati korištenjem većih skupova podataka, računala s većom procesnom snagom te još boljim optimiranjem hiperparametara.

Literatura

[1] Free Image on Pixabay - Nerve Cell, Neuron, Brain, Neurons, Poveznica: https://pixabay.com/vectors/nerve-cell-neuron-brain-neurons-3759541/, Pristupan 6.6.2023.

[2] Activation functions in Neural Networks, Poveznica: https://www.geeksforgeeks.org/activation-functions-neural-networks/, Pristupano: 20.5.2023.

[3] LightningModule, LightningModule - PyTorch Lightning 2.0.2 documentation, Poveznica: https://lightning.ai/docs/pytorch/stable/common/lightning_module.html#training, Pristupano 20.5.2023.

[4] A Comprehensive Guide to Convolutional Neural Networks - the ELI5 way, Saturn Cloud Blog, Poveznica: https://saturncloud.io/blog/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way/, Pristupano 20.5.2023.

Sažetak

KLASIFIKACIJA VRSTI PTICA IZ SLIKA TEMELJENA NA DUBOKOM UČENJU

U ovom radu je opisan problem klasifikacije vrsta ptica. Pri izradu modela korištene su dvije metode strojnog učenja: jednostavni višeslojni perceptron i konvolucijske mreže. Za učenje, validaciju i testiranje korišteno je 15 kategorija iz javno dostupnog skupa podataka CUB 200_2011. Testiranjem modela jednostavnog perceptrona dobivena je točnost od 21,84% i koeficijent pouzdanosti od 19,80%, dok je testiranjem modela konvolucijskih mreža dobivena točnost od 66,67% i koeficijent pouzdanosti od 79,4 %. Model konvolucijskih mreža iskazao se kao točniji i pouzdaniji te je zato ugrađen u jednostavnu web aplikaciju za klasifikaciju ptica.

Ključne riječi: jednostavni višeslojni perceptron, konvolucijska mreža, klasifikacija ptica.

Summary

CLASSIFICATION OF BIRD SPECIES FROM IMAGES BASED ON DEEP LEARNING

This paper describes the problem of classification of bird species. Two machine learning methods were used to create models: simple multilayer perceptron and convolutional networks. There were 15 categories from the publicly available dataset CUB 200_2011 that were used for learning, validation and testing. Testing the simple perceptron model resulted in an accuracy of 21,84% and a reliability coefficient of 19,18%, while testing the convolutional network model resulted in an accuracy of 66,67% and a reliability coefficient of 79,47%. The convolutional network model proved to be more accurate and reliable, and was thus incorporated into a simple web application for bird classification.

Keywords: simple multilayered perceptron, convolutional network, bird classification.