Sveučilište J. J. Strossmayera u Osijeku
Odjel za matematiku
Sveučilišni preddiplomski studij matematike i računarstva

Filip Vuković

Josip Pavičić

Problem prepoznavanja oblika na fotografijama

Završni praktični projekt - dokumentacija

Osijek, 2021.

Sveučilište J. J. Strossmayera u Osijeku
Odjel za matematiku
Sveučilišni preddiplomski studij matematike i računarstva

Filip Vuković

Josip Pavičić

Problem prepoznavanja oblika na fotografijama

Završni praktični projekt - dokumentacija

Mentor: izv. prof. dr. sc. Domagoj Matijević

Osijek, 2021.

Sadržaj

Uvod

0.1 O problemu

Ovaj projekt se bavi problemom klasifikacije (lokalizacije) objekata na slikama.
Projekt kreće s kreiranjem vlastitog skupa podataka, odnosno slika, koje koristimo tijekom treniranja i testiranja vlastito konstruirane konvolucijske neuronske mreže. Skup podataka se sastoji od četiri klase slika. U sklopu ovog projekta, naša konvolucijska neuronska mreža doseže 88% točnosti prilikom klasifikacije.

0.2 Tehnologije

Implementacija projekta napisana je koristeći programski jezik Python i sljedeće biblioteke:

- NumPy[1]
  - open-source paket za računanje s multidimenzionalnim poljima
  - jako brzo računa zato što koristi vektorske operacije, dakle nemamo puno petlji i indeksa
  - radi na osnovi optimiziranog, pred-kompajliranog C koda
- PyTorch[2]
  - open-source framework koji koristimo za strojno učenje, baziran je na Torch biblioteci
  - sličan je NumPy-u, ali računanje provodi na grafičkoj kartici pomoću svojih tensor objekata
- cv2[3]
  - open-source biblioteka bazirana na OpenCV-u
  - uglavnom se fokusira na obradu slika, snimanje i analizu videa, uključujući značajke poput otkrivanja lica i otkrivanja objekata
- glob[4]
  - koristi se za dohvaćanje datoteka i/ili putanja koje odgovaraju navedenom uzorku

0.3 Organizacija koda

Kod[5] je strukturiran pomoću 3 glavne klase:

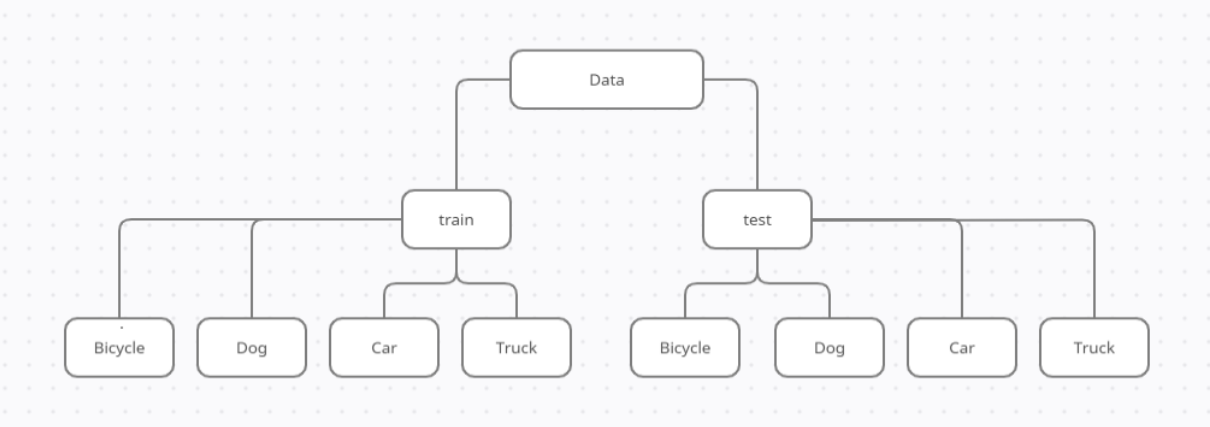
- *ImageAugmentation* - pretprocesiranje slika (mijenjamo veličinu, reflektiramo sliku s obzirom na x,y osi)
- *CDataset* - učitavanje i označavanje slika
- *CNN* - definiranje konvolucijske neuronske mreže

[1] https://numpy.org/
[2] https://pytorch.org/
[3] https://docs.opencv.org/4.5.2/d6/d00/tutorial_py_root.html
[4] https://docs.python.org/3/library/glob.html
[5] https://github.com/pavicicjosip/CNN/blob/main/Dataloader/Dataloader.ipynb

1 Pretprocesuiranje podataka

Pretprocesuiranje slika je niz koraka kojima mijenjamo originalne slike u njima slične koristeći rotacije, skaliranja, translacije, promjene boje, refleksije... Pomoću prethodnih radnji proširujemo naš skup podataka (slika) za treniranje modela.
Konvolucijske neuronske mreže imaju ulaz fiksne veličine (konstantne dimenzije). Stoga je potrebno pretprocesuirati slike, odnosno promijeniti na svima istu, unaprijed određenu veličinu. U našem radu koristimo fiksnu veličinu slike (64, 36).

1.1 Arhitektura podataka



Slika 1: Struktura mape *Data* (kvadratić predstavlja mapu)

Skup podataka se nalazi u mapi *Data*. U njoj su sadržane podmape *train* i *test* u kojima se nalaze skupovi podataka za treniranje i testiranje. Sastoje se od slika koje su podijeljene u mape, odnosno klase.

1.2 Implementacija

Za pretprocesiranje slika koristimo vlastito implementiranu Python klasu *ImageAugmentation*.

Sučelje klase *ImageAugmetation*		
ime varijable	tip	opis
path	string	putanja do mape s podacima

Enkapsulirani podaci klase *ImageAugmetation*		
ime varijable	tip	opis
dim	tuple	dimenzija na koju se resize-aju slike
folder_list	lista stringova	lista putanja mapa koje se nalaze u danoj putanji path
folders	lista stringova	lista koja sadrži imena mapa iz dane putanje path
files	lista stringova	putanje do slika

1.2.1 Kod i primjer rada

```
class ImageAugmentation:
    def __init__(self, path = "./"):
        self.dim = (64, 36)
        self.folder_list = glob.glob(path + "/*")
        self.folders = []
        for p in self.folder_list:
            self.folders.append(p.split("\\")[-1])



        for folder in self.folders:
            self.files = glob.glob(path + "/" + folder + "/*")

            for file in self.files:
                #resize
                file_name = file.split("\\")[-1]
                img = cv2.imread(path + "/" + folder + "/" + file_name)
                resized = cv2.resize(img, self.dim,interpolation=cv2.INTER_AREA)
                cv2.imwrite(path + "/" + folder + "/" + file_name,resized)


                constraint = ["rx_", "ry_"]
                if file_name[:3] not in constraint and path.split('/')[-1] != 'test':
                    rows, cols, dim = resized.shape
                    My = np.float32([[-1,  0, cols],
                                     [0 ,  1, 0   ],
                                     [0 ,  0, 1   ]])
                    Mx = np.float32([[1,  0, 0],
                                     [0 ,  -1, rows],
                                     [0 ,  0, 1   ]])
                    #resized img reflection on y axis
                    reflected_img = cv2.warpPerspective(resized,My,(int(cols),int(rows)))
                    cv2.imwrite(path + "/" + folder + "/" + "ry_"+ file_name,reflected_img)

                    #resized img reflection on x axis
                    reflected_img = cv2.warpPerspective(resized,Mx,(int(cols),int(rows)))
                    cv2.imwrite(path + "/" + folder + "/" + "rx_"+ file_name,reflected_img)
```

Slika 2: *ImageAugmentation* klasa

Inicijaliziramo klasu *ImageAugmentation* i njenom sučelju proslijedimo string *"./data/train"*. Pri inicijalizaciji klasa dohvaća putanje podmapa unutar mape *train* te izvlači njihova imena i sprema u varijablu *folders* (redci 3-7). Nadalje, petljom (red 11) prolazimo kroz svaku mapu i spremamo putanje svih slika u varijablu *files*. Zatim petljom (red 14) prolazimo kroz svaku prikupljenu putanju, odnosno sliku te joj mijenjamo dimenzije (redci 16-19). U grananju (red 23) provjeravamo jesmo li već trenutnu sliku reflektirali. Ukoliko nismo, reflektiramo te ju spremimo s odgovarajućim prefiksom u imenu (redci 24-37).

2 Učitavanje podataka

Pri učitavanju podataka koristimo implementiranu klasu *CDataset* koja nasljeđuje klasu *Dataset* iz *torch.utils* modula.
Trebamo implementirati konstruktor te prepisati (en. override) metodu *__getitem__*() tako da podražava dohvaćanje podataka za dani ključ. Opcionalno možemo prepisati (en. override) metodu *__len__*() koja vraća veličinu dataseta.

2.1 Implementacija

Sučelje klase *CDataset*		
ime varijable	tip	opis
train	boolean	boolean koji određuje učitavamo li podatke za treniranje
test	boolean	boolean koji određuje učitavamo li podatke za testiranje

Enkapsulirani podaci klase *CDataset*		
ime varijable	tip	opis
data_path	string	putanja do *train* ili *test* mape
folder_list	lista stringova	lista putanja do podmapa mape *train* ili *test*
data	lista listi stringova	lista koja sadrži par putanje do svake slike i njenu klasu
class_map	rječnik	rječnik koji svakoj klasi dodjeljuje odgovarajuću oznaku

2.1.1 Kod i opis rada

```
1  class CDataset(Dataset):
2      def __init__(self, train = False, test = False):
3          if(train):
4              self.data_path = "./data/train/"
5          if(test):
6              self.data_path = "./data/test/"
7          self.folder_list = glob.glob(self.data_path + "*")
8
9          self.data = []
10         for folder in self.folder_list:
11             folder_name = folder.split("\\")[-1]
12             for img_path in glob.glob(self.data_path + folder_name + "/*.jpg"):
13                 self.data.append([img_path, folder_name])
14                 print(self.data)
15         self.class_map = {"Car": 0, "Truck": 1, "Bicycle": 2, "Dog": 3}
16
17     def __len__(self):
18         return len(self.data)
19
20     def __getitem__(self, i):
21         img_path, class_name = self.data[i]
22         img = cv2.imread(img_path)
23         class_id = self.class_map[class_name]
24         img_tensor = torch.tensor(img, dtype=torch.float)
25         img_tensor = img_tensor.permute(2,0,1)
26         class_id = torch.tensor([class_id])
27         return img_tensor, class_id
28
```

Slika 3: *CDatabase* klasa

U konstruktoru, na temelju proslijeđenih vrijednosti, odlučujemo koji se skup podataka učitava te spremamo putanje do svake podmape (redci 3-7). Petljom (red 10) prolazimo kroz svaku podmapu te spremamo ime podmape u varijablu *folder_name*. Zatim, petljom (red 12) dohvaćamo putanju do svake slike te ju spremamo zajedno s imenom podmape u kojoj se nalazi, u listu *data*. Zatim pravimo bijekciju između imena klase i njene oznake (red 14).
Metoda *___len___*() vraća integer vrijednost, veličinu dataseta.
Metoda *___getitem___*() vraća sliku u obliku tenzora s njenom odgovarajućom oznakom.
Slijedi inicijalizacija *DataLoader* objekata pomoću kojeg dohvaćamo podatke za treniranje i testiranje.

```
batch_size = 5
train_dataset = CDataset(train=True)
data_loader_train = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)

test_dataset = CDataset(test=True)
data_loader_test = DataLoader(test_dataset, batch_size=batch_size, shuffle=True)
```

Slika 4: Inicijalizacija klase

U 2. i 3. retku učitavamo podatke za treniranje, a u 5. i 6. podatke za testiranje.

```
DataLoader(dataset, batch_size=1, shuffle=False, sampler=None,
           batch_sampler=None, num_workers=0, collate_fn=None,
           pin_memory=False, drop_last=False, timeout=0,
           worker_init_fn=None, *, prefetch_factor=2,
           persistent_workers=False)
```

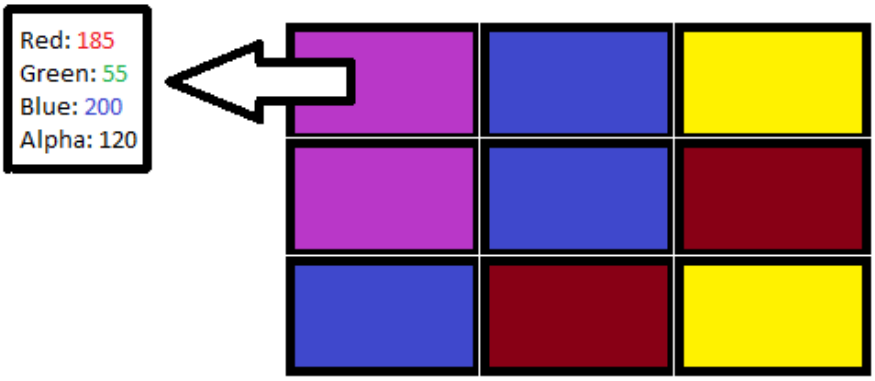
Slika 5: Sučelje klase *DataLoader*

Iz slike 5 vidljivo je da DataLoader-u [6] moramo proslijediti željeni skup podataka te možemo promijeniti ostale parametre. Mi smo izmijenili dva parametra, a to su:

- batch_size - koliko uzoraka po seriji (en. batch) učitati. U našem slučaju uzeli smo 5 jer je cjelobrojni djelitelj broja podataka u skupu za treniranje i testiranje. Time smo izbjegli da nam zadnja serija bude djelomično ispunjena. (zadana vrijednost je 1)
- shuffle - postavili smo na True tako da se skup podataka nasumično promiješa početkom svake epohe zato što su nam podaci učitani redom po klasama. Miješanje podataka služi u svrhu smanjenja varijance, osigurava generaliziranost i smanjuje prenaučenost (en. overfit) modela. (zadana vrijednost je False)

3 Konvolucijska neuronska mreža

Konvolucijska neuronska mreža, također poznata kao CNN ili *ConvNet* je klasa neuronskih mreža koja se bavi obradom podataka koji su *grid-like* topologije, na primjer slike. Digitalna slika sadrži niz piksela raspoređenih u obliku mreže. Svaki piksel se sastoji od tri vrijednosti koje reprezentiraju boju i četvrte vrijednosti koji reprezentira prozirnost.



Slika 6: Slika reprezentirana kao *grid* podatak

[6] https://pytorch.org/docs/stable/data.html

3.1 Rad na grafičkoj kartici

Grafička kartica je brža od procesora kad dođe u pitanje računanje problema koji se može računati paralelno. Paralelno računanje rastavlja originalan problem u manje i jednostavnije, neovisne probleme koji se mogu računati istovremeno. Rješenja tih manjih problema zatim kombiniramo kako bi dobili rezultat početnog problema.

U paralelnom računanju, sramotno paralelan (en. embarrassingly parallel) problem je onaj za kojeg treba jako malo, gotovo nikakav napor kako bi ga rastavi na male, neovisne probleme.

Upravo su takve neuronske mreže. Puno problema koje računamo možemo s lakoćom podijeliti u manje i jednostavnije probleme.

```
1 if torch.cuda.is_available():
2     device = torch.device("cuda:0")
3     print("Running on the GPU")
4 else:
5     device = torch.device("cpu")
6     print("Running on the CPU")
```

Slika 7: GPU konfiguracija, PyTorch

3.2 Implementacija konvolucijske neuronske mreže

```
class CNN(nn.Module):

    def __init__(self):
        super(CNN, self).__init__()

        self.conv_layer = nn.Sequential(
            nn.Conv2d(in_channels=3, out_channels=32, kernel_size=3, padding=1),
            nn.BatchNorm2d(32),
            nn.PReLU(),

            nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3, padding=1),
            nn.BatchNorm2d(64),
            nn.PReLU(),


            nn.MaxPool2d(kernel_size=2, stride=2),

            nn.Conv2d(in_channels=64, out_channels=64, kernel_size=3, padding=1),
            nn.BatchNorm2d(64),
            nn.PReLU(),

            nn.MaxPool2d(kernel_size=2, stride=2),
        )


        self.fc_layer = nn.Sequential(
            nn.Dropout(p=0.6),
            nn.Linear(9216, 1028),
            nn.PReLU(),
            nn.Dropout(p=0.6),
            nn.Linear(1028, 128),
            nn.PReLU(),
            nn.Dropout(p=0.6),
            nn.Linear(128, len(train_dataset.class_map)),
        )


    def forward(self, x):
        x = self.conv_layer(x)
        x = torch.flatten(x, 1)
        x = self.fc_layer(x)
        return x
```

Slika 8: Neuronska mreža

3.2.1 Arhitektura mreže

Naša konvolucijska mreža se sastoji od sljedećih slojeva:

- konvolucijskog
- pooling
- normalizacijskog
- potpuno povezanog

Nakon svakog konvolucijskog sloja slijedi ili normalizacijski[7] ili pooling[8] sloj.
Kao aktivacijska funkcija, najbolje nam se pokazala *Parametic ReLU (PReLU)*.
U potpuno povezanom sloju smo postavili *Dropout* od 60% ispred svakog linearnog sloja. Time mreža postaje manje osjetljiva na specifičnu težinu neurona, a time dobivamo bolju generalizaciju modela i spriječavamo pretreniranje.

3.2.2 Računanje ulazne dimenzije potpuno povezanog sloja

Neka imamo sljedeće oznake:

- o - broj izlaznih kanala posljednjeg konvolucijskog sloja (Conv2d) naše konvolucijske neuronske mreže
- w - širina slike
- h - visina slike
- n - broj MaxPool2d-a
- x - skup svih MaxPool2d-a(x_i) naše neuronske mreže koji je oblika $x_i = (k_i, s_i)$ gdje je k_i jednak kernel_size-u, a s_i stride-u i-tog MaxPool2d-a

$$f(o, w, h, x) = o \cdot w \cdot h \cdot \prod_{i=1}^{n} \frac{1}{k_i} \cdot \frac{1}{s_i}, \tag{1}$$

Jednadžbom (1) možemo izračunati ulaznu dimenziju potpuno povezanog sloja

Račun:
$o = 64$
$w = 64$
$h = 36$
x = (2,2),(2,2)

$$f(64, 64, 36, \{(2, 2), (2, 2)\}) = 64 \cdot 64 \cdot 36 \cdot \tfrac{1}{2} \cdot \tfrac{1}{2} \cdot \tfrac{1}{2} \cdot \tfrac{1}{2} = 9216$$

Za provjeru našeg računa, odlučili smo se napraviti "dummy" podatak, odnosno tensor dimenzija slike, ali ispunjen nulama (redak 3, slika 9). Jednim prolazom s tim tenzorom kroz konvolucijski sloj dobivamo podatak koji je rastegnut u 1-D polje (red 5, slika 9) te vraćamo njegovu duljinu (redak 7, slika 9), koja je ujedno i duljina ulaza potpuno povezanog sloja. Ovo vrijedi za svaku sliku jer su sve pretprocesirane na istu veličinu.

[7] https://pytorch.org/docs/stable/generated/torch.nn.BatchNorm2d.html
[8] https://pytorch.org/docs/stable/generated/torch.nn.MaxPool2d.html

```
net = CNN()
def calc_input_dims():
        batch_data = torch.zeros((1, 3, 64, 36))

        batch_data = net.conv_layer(batch_data)

        return int(np.prod(batch_data.size()))
calc_input_dims()
```

Slika 9: Inicijalizacija neuronske mreže i računanje ulazne dimenzije potpuno povezanog sloja

3.2.3 Računanje parametara mreže

- **Ulazni sloj:** u ulaznom sloju ništa ne učimo, dakle broj parametara je 0.
- **Konvolucijski sloj:** Neka je *kernel_size* dimenzija filtera prethodnog sloja kojih ima ukupno *in_channels* te je *out_channels* broj filtera trenutnog sloja. Broj parametara dan je izrazom (ne smijemo zaboraviti dodati jedan (bias))

$$((kernel\_size^2 \cdot in\_channels) + 1) \cdot out\_channels \tag{2}$$

- **pooling sloj:** ništa ne učimo, samo računamo određeni broj. Broj parametra je 0.
- **Potpuno povezani sloj:** potpuno povezani sloj ima najviše parametra zato što je svaki neuron povezan sa svim ostalim. Neka je *out_features* broj neurona u trenutnom sloju i neka je *in_features* broj neurona u prethodnom sloju. Broj parametara dan je izrazom (ne smijemo zaboraviti bias (1**in_features*))

$$((in\_features \cdot out\_features) + 1 \cdot out\_features) \tag{3}$$

Generalna formula
Neka je X skup svih konvolucijskih slojeva zapisanih u obliku uređenih trojki ($in\_channels$, $out\_channels, kernel\_size$) i x broj uređenih trojki skupa X. Neka je Y skup svih potpuno povezanih slojeva zapisanih u obliku uređenih parova ($in\_features, out\_features$) i y broj uređenih parova skupa Y. Tad možemo izračunati broj parametara izrazom:

$$\sum_{n=1}^{x}((X_{n,3} \cdot X_{n,3} \cdot X_{n,1} + 1) \cdot X_{n,2}) + \sum_{n=1}^{y}(Y_{n,1} \cdot Y_{n,2} + 1 \cdot Y_{n,2}) \tag{4}$$

PRIMJER:
Izračunajmo broj parametara mreže sa slike 8 pomoću (4).

$$X = \{(3, 32, 3),(32, 64, 3),(64, 64, 3)\},\ Y = \{(9216,1028),(1028,128),(128,4)\}$$

$$\sum_{n=1}^{x}((X_{n,3} \cdot X_{n,3} \cdot X_{n,1} + 1) \cdot X_{n,2}) + \sum_{n=1}^{y}(Y_{n,1} \cdot Y_{n,2} + 1 \cdot Y_{n,2})$$

$$= \{[(3 \cdot 3 \cdot 3 + 1) \cdot 32] + [(3 \cdot 3 \cdot 32 + 1) \cdot 64] + [(3 \cdot 3 \cdot 64 + 1) \cdot 64]\} + \{[9216 \cdot 1028 + 1 \cdot 1028] + [1028 \cdot 128 + 1 \cdot 128] + [128 \cdot 4 + 1 \cdot 4]\}$$

$$= \mathbf{9\ 663\ 624}$$

```
X = [(3, 32, 3),(32, 64, 3),(64, 64, 3)]
Y = [(9216,1028),(1028,128),(128,4)]
num_param = 0
for x in X:
    num_param += (x[2] * x[2] * x[0] + 1 )*x[1]
for y in Y:
    num_param += y[0] * y[1] + 1 * y[1]
num_param
```

```
9663624
```

Slika 10: Implementacija funkcije (4)

4 Treniranje

4.1 Funkcija gubitka i optimizator

```
lossF = nn.CrossEntropyLoss()
optimizer = optim.Adam(net.parameters(), lr=0.0001)
```

Slika 11: Funkcija gubitka i optimizator

Kao *loss* funkciju koristimo *CrossEntropyLoss*. Ona mjeri performanse klasifikacijskog modela čiji je izlaz vjerojatnosti između 0 i 1. *CrossEntropyLoss* raste logaritamski kako se predviđena vjerojatnost razlikuje od stvarne oznake.
Na primjer, predviđena vjerojatnost od 0,05, kad je stvarna oznaka 1, rezultira velikim gubitkom.

Kao optimizatora koristimo algoritam *Adam* jer kombinira najbolja svojstva algoritama *AdaGrad* i *RMSProp* te pruža optimizacijski algoritam koji radi jako dobro s aproksimiranim gradijentima.

4.2 Treniranje modela

```
1  net.to(device)
2  epochs = 50
3  train_loss = []
4  test_loss = []
5
6  for epoch in range(epochs):
7      print("Starting epoch #" + str(epoch))
8      running_loss_train = 0.0
9      running_loss_test = 0.0
10     net.to(torch.device("cuda:0"))
11     for i, data in enumerate(data_loader_train, 0):
12         # inputs, labels = data
13         inputs, labels = data[0].to(device), data[1].to(device)
14         optimizer.zero_grad()
15
16         outputs = net(inputs)
17         loss = lossF(outputs, labels.flatten())
18         loss.backward()
19         optimizer.step()
20
21         running_loss_train += loss.item()
22
23     net.to(torch.device("cpu"))
24     with torch.no_grad():
25         for data in data_loader_test:
26             images, labels = data
27             outputs = net(images)
28
29             loss = lossF(outputs, labels.flatten())
30             running_loss_test += loss.item()
31
32     train_loss.append(float(running_loss_train))
33     test_loss.append(float(running_loss_test))
34
35
36
37 print('Finished Training')
38
```

Slika 12: Treniranje modela

Prije početka treniranja prebacujemo našu mrežu (red 1), kasnije i slike s pripadajućim oznakama, na grafičku karticu. Odredimo broj epoha, tj ciklusa treniranja na trening podacima (red 2).Zatim petljom (red 11) prolazimo kroz sve trening podatke, definiramo optimizator i funkciju troška, vršimo propagaciju unaprijed, a potom i unazad. Propagacijom unazad računamo gradijente na osnovu kojih ažuriramo težine neuronske mreže s ciljem da smanjimo vrijednost funkcije gubitka.Propagacijom unazad naša mreža zapravo "uči". Petljom (red 24) testiramo naš model na test podacima nakon svakog ciklusa treniranja. Spremamo vrijednosti funkcije troška na trening i test podacima (red 32 i 33) koje nam kasnije reprezentiraju je li model pretreniran, podtreniran ili taman.

5 Evaluacija

Evaluacijom modela nam pomaže izabrati model koji će najbolje reprezentirati naše podatke.Za evaluaciju računamo matricu konfuzije te tijekom treniranja spremamo iznose funkcije gubitka za trening i test podatke te računamo točnost modela.

5.1 Matrica zabune

Matrica zabune[9] način je ocjenjivanja klasifikacijskog modela. To je usporedba između stvarne klase podatka i klase koju model predvidi za taj podatak. Dobivamo matricu zabune ispunjenu s točno pozitivnim(TP), točno negativnim (TN), netočno pozitivnim(FP) i netočno negativnim(FN) klasificiranim slikama za svaku klasu. Pomoću te matrice možemo računati:

- točnost (en. accuracy) (svi točni/svi)
  - $accuracy = \frac{TP+TN}{TP+TN+FP+FN}$
  - generalna točnost modela. Nije korisna kad broj podataka među klasama nije približno jednak.
- netočnost (en. misclassification) (svi netočni/svi)
  - $misclassification = \frac{FP+FN}{TP+TN+FP+FN}$
  - koliko podataka nismo točno klasificirali
- preciznost (en. precision) (točno pozitivni/predviđeni pozitivni)
  - $precision = \frac{TP}{TP+FP}$
  - od svih predikcija n-te klase, koliko je njih uistinu bilo te klase
- osjetljivost (en. sensitivity aka recall) (točno pozitivni/svi pozitivni)
  - $recall = \frac{TP}{TP+FN}$
  - od svih pravih podataka n-te klase, koliko je njih točno klasificirano
  - maksimiziranjem preciznosti možemo smanjiti osjetljivost modela. Vrijedi i obratno.

[9] https://blogs.oracle.com/aianddatascience/post/asimpleguide-tobuildingaconfusionmatrix

5.1.1 Implementacija

```python
conf_matrix = confusion_matrix(true, predict)

class Confusion_matrix():
    def __init__(self):
        self.M = []
        self.class_map = {"Car": 0, "Truck": 1, "Bicycle": 2, "Dog": 3}

    def extract_all(self, CM):
        for k in range(4):
            tp, fn, fp, tn = 0, 0, 0, 0
            for i in range(4):
                for j in range(4):
                    if i == j and i == k:
                        tp = CM[i][j]
                    elif i == k and (j > i or j < i):
                        fn += CM[i][j]
                    elif j == k and (i > j or i < j):
                        fp += CM[i][j]
                    else:
                        tn += CM[i][j]
            self.M.append([tn, fp, fn, tp])


        for i in range(4):
            position = list(self.class_map.values()).index(i)
            print("Class: ", list(self.class_map.keys())[position])
            print("[tn, fp, fn, tp]: ", self.M[i])
            print("Accuracy: ", self.accuracy(i))
            print("Misclassification: ", self.misclassification(i))
            print("Precision: ", self.precision(i))
            print("Recall: ", self.recall(i))
            print("*******************************************")

    def tn_fp_fn_tp(self, _class):
        return self.M[_class]

    def accuracy(self, _class):
        tn, fp, fn, tp = self.M[_class]
        return (tp+tn)/(tp+tn+fp+fn) * 100

    def misclassification(self, _class):
        tn, fp, fn, tp = self.M[_class]
        return (fp+fn)/(tp+tn+fp+fn) * 100

    def precision(self, _class):
        tn, fp, fn, tp = self.M[_class]
        return tp/(tp+fp) * 100

    def recall(self, _class):
        tn, fp, fn, tp = self.M[_class]
        return tp/(tp+fn) * 100
```

Slika 13: Klasa matrice zabune

Klasa *Confusion_matrix* sadrži nama važnu metodu *extract_all* kojoj prosljeđujemo matricu konfuzije. Pomoću nje dobivamo vrijednosti TP, TN, FP, FN, točnost, netočnost, preciznost i osjetljivost po svakoj klasi (slika 14).

```
Class:  Car
[tn, fp, fn, tp]:  [29, 1, 1, 9]
Accuracy:  95.0
Misclassification:  5.0
Precision:  90.0
Recall:  90.0
********************************************
Class:  Truck
[tn, fp, fn, tp]:  [29, 1, 2, 8]
Accuracy:  92.5
Misclassification:  7.5
Precision:  88.88888888888889
Recall:  80.0
********************************************
Class:  Bicycle
[tn, fp, fn, tp]:  [30, 0, 1, 9]
Accuracy:  97.5
Misclassification:  2.5
Precision:  100.0
Recall:  90.0
********************************************
Class:  Dog
[tn, fp, fn, tp]:  [28, 2, 0, 10]
Accuracy:  95.0
Misclassification:  5.0
Precision:  83.33333333333334
Recall:  100.0
********************************************
```

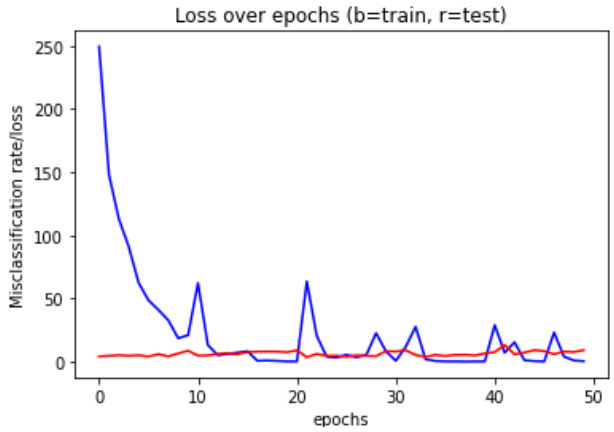
Slika 14: Metrike modela sa slike 8

Pošto imamo 40 trening podataka, a iz slike 14 vidljivo je da je točno pozitivnih sveukupno $9 + 8 + 9 + 10 = 36$. Iz navedenog slijedi da je točnost modela $\frac{36}{40} = 90\%$

5.2 Treniranost modela

Računanje funkcije gubitka na trening i test podacima u svakom ciklusu(epohi) treniranja nam daje važnu informaciju o treniranosti modela. Model može biti pretreniran, podtreniran i može biti "taman".

5.2.1 Pretrenirani model



Slika 15: Graf fukcija gubitka na pretreniranom modelu

Model je pretreniran u terminima funkcije gubitka kad je funkcija gubitka na trening podacima manja od funkcije gubitka na test podacima. Također funkcija gubitka na test podacima raste (na slici 15. je vidljiv blagi rast)

Model je pretreniran kad je toliko dobro natreniran na test podacima da nauči i njihov šum. Pretrenirani model nauči svaki podatak perfektno tako da kad mu dođe neviđeni ili novi podatak krivo ga klasificira (model nije generaliziran). Model koji je pretreniran ima odnos: $funkcija\_gubitka\_trening << funkcija\_gubitka\_test$

Načini za spriječiti pretreniranost:

- smanjiti kompleksnost modela (parametre)
- povećati skup trening podataka
- regularizacija (L1, lasso, dropout...) (koristimo kad ne znamo koje parametre maknuti)
- ranije zaustavljanje treniranja (zaustavimo treniranje kad funkcija gubitka na test podacima počne rasti)

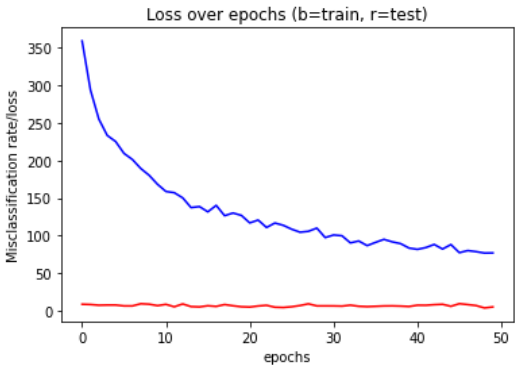
```
Class:  Car
[tn, fp, fn, tp]:  [29, 1, 1, 9]
Accuracy:  95.0
Misclassification:  5.0
Precision:  90.0
Recall:  90.0
*****************************************
Class:  Truck
[tn, fp, fn, tp]:  [29, 1, 1, 9]
Accuracy:  95.0
Misclassification:  5.0
Precision:  90.0
Recall:  90.0
*****************************************
Class:  Bicycle
[tn, fp, fn, tp]:  [30, 0, 1, 9]
Accuracy:  97.5
Misclassification:  2.5
Precision:  100.0
Recall:  90.0
*****************************************
Class:  Dog
[tn, fp, fn, tp]:  [28, 2, 1, 9]
Accuracy:  92.5
Misclassification:  7.5
Precision:  81.81818181818183
Recall:  90.0
*****************************************
```

Slika 16: Rezultati pretreniranog modela na test podacima

5.2.2 Podtrenirani model

Podtrenirani model općenito ima lošu matricu konfuzije i njene metrike. Ima premalo parametara, odnosno premalu kompleksnost za klasifikacijski problem. Funkcije gubitka na test i trening podacima su velike i imaju odnos: $funkcija\_gubitka\_trening >> funkcija\_gubitka\_test$ Kažemo da je model podtreniran ako ne može generalno naučiti uzorke podataka. Podtrenirani model ne nauči u potpunosti trening podatke. Načini za spriječiti podtreniranost:

- povećati kompleksnost modela (parametre)
- povećati broj ciklusa (epoha) treniranja
- smanjiti regularizaciju (L1, lasso, dropout...)

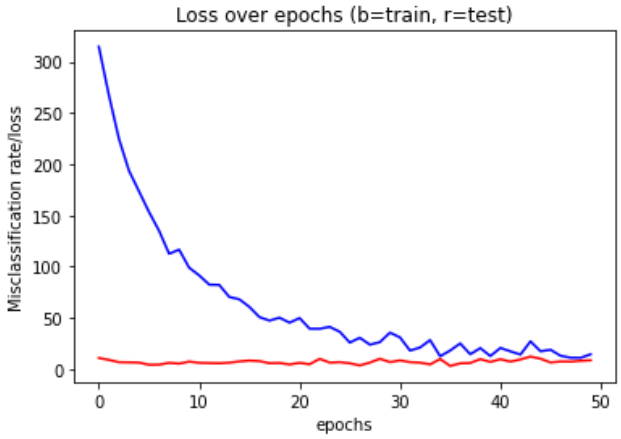


Slika 17: Graf funkcija gubitka na podtreniranom modelu

```
Class:  Car
[tn, fp, fn, tp]:  [26, 4, 1, 9]
Accuracy:  87.5
Misclassification:  12.5
Precision:  69.23076923076923
Recall:  90.0
********************************************
Class:  Truck
[tn, fp, fn, tp]:  [26, 4, 5, 5]
Accuracy:  77.5
Misclassification:  22.5
Precision:  55.55555555555556
Recall:  50.0
********************************************
Class:  Bicycle
[tn, fp, fn, tp]:  [28, 2, 4, 6]
Accuracy:  85.0
Misclassification:  15.0
Precision:  75.0
Recall:  60.0
********************************************
Class:  Dog
[tn, fp, fn, tp]:  [28, 2, 2, 8]
Accuracy:  90.0
Misclassification:  10.0
Precision:  80.0
Recall:  80.0
********************************************
```

Slika 18: Rezultati podtreniranog modela na test podacima

5.2.3 Optimalan model



Slika 19: Graf funkcija gubitka na optimalanom modelu

Nakon korekcija na neuronskoj mreži, popravaka pretreniranosti,podtreniranosti i arhitekture mreže, dobit ćemo model koji dobro klasificira neviđene podatke. Model je taman ako vrijedi: $funkcija\_gubitka\_trening \sim funkcija\_gubitka\_test$

Na slici 19 vidimo odnos trening i test funkcije gubitka po ciklusima (epohama) našeg modela sa slike 8 koji je zadovoljavajući.

6 Spremanje i učitavanje

PyTorch nam omogućuje spremanje istreniranog modela tako da ga kasnije možemo lakše učitati i koristiti bez ponovnog treniranja.

```
PATH = "./CNNmodule"
torch.save(net.state_dict(), PATH)
```

Slika 20: Spremanje

```
model = CNN()
model.load_state_dict(torch.load(PATH))
model.eval()
```

Slika 21: Učitavanje

Literatura

[1] https://pytorch.org/

[2] https://blogs.oracle.com/ai-and-datascience/post/a-simple-guide-to-building-a-confusion-matrix

[3] https://towardsdatascience.com/visual-guide-to-the-confusion-matrix-bb63730c8eba

[4] https://medium.com/mlearning-ai/training-convolutional-neural-network-convnet-cnn-on-gpu-from-scratch-439e9fdc13a5

[5] https://www.analyticsvidhya.com/blog/2020/09/overfitting-in-cnn-show-to-treat-overfitting-in-convolutional-neural-networks/

[6] https://www.ibm.com/cloud/learn/overfitting

[7] https://towardsdatascience.com/learning-curve-to-identify-overfitting-underfitting-problems-133177f38df5

[8] https://towardsdatascience.com/understanding-and-calculating-the-number-of-parameters-in-convolution-neural-networks-cnns-fc88790d530d

[9] https://pytorch.org/tutorials/beginner/data_loading_tutorial.html