Klasifikacija objekata prikazanih na dubinskim slikama pomoću umjetne neuronske mreže

Križanac, Rea

SVEUČILIŠTE JOSIPA JURAJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studiji

KLASIFIKACIJA OBJEKATA PRIKAZANIH NA DUBINSKIM SLIKAMA POMOĆU UMJETNE NEURONSKE MREŽE

Završni rad

Rea Križanac

Osijek, 2021

Sadržaj:

1. UVOD

Čovjek u svakodnevnom životu procesuira veliki broj podataka u svom mozgu. Pogrešno je mišljenje da smo u današnje vrijeme digitalnim računalima prepustili obradu većine podataka. Digitalna računala uistinu obrađuju ogromne količine podataka, ali to je tek mali dio koji se svakodnevno obradi u mozgu čovjeka. Ljudski mozak je sposoban stalno učiti i na temelju logičkih zaključaka donositi ispravne odluke za razliku od digitalnih računala. Računalo za razliku od čovjeka u kratkom roku može riješiti komplicirane matematičke jednadžbe. Kada bi sposobnosti učenja koje ima čovjek imalo računalo omogućilo bi rasterećenje ljudskog mozga od velikog broja informacija.

Više od 50 godina umjetna inteligencija je prisutna u svakodnevnom životu kao što je pretraživanje interneta, automobilska industrija, zdravlje, promet, u poljoprivredi i drugim aspektima života i gospodarstva. Dostupnost velike količine podataka i napredak u računalnoj snazi doveli su do velikih otkrića na području umjetne inteligencije, te se ona smatra ključnom za digitalnu transformaciju društva. Područje umjetne inteligencije teži da se napraviti replika ljudskog mozga za rješavanje svakodnevnih problema. Cilj je, umjetnu inteligenciju naučiti da sama uči tj. da analizirajući svu okolinu sama pronalazi uzorke i odgovore na probleme.

U ovom završnom radu opisan je proces klasifikacije dubinskih slika pomoću umjetne neuronske mreže. U prvom djelu završnog rada obrađen je pojam umjetnih neuronskih mreža, vrste neurona i proces učenja. U drugom dijelu završnog rada obrađene su konvolucijske neuronske mreže, arhitektura i PyTorch programski jezik. U trećem dijelu završnog rada obrađuje se analiza rješenja i rezultati treniranja mreže.

1.1 Zadatak završnog rada

Zadatak završnog rada je korištenjem programskog jezika Python, biblioteke PyTorch i VGGNet konvolucijske neuronske mreže klasificirati dubinske slike. Ispitati učinkovitost programa na dubinskim slikama snimljenim 3D kamerom.

2. UMJETNE NEURONSKE MREŽE

Cijeli koncept umjetne inteligencije potaknut je promatranjem ljudskog mozga i živčanih stanica. Cilj je stvoriti umjetnu tvorevinu koja imitira proces obrade podataka u mozgu. Neuronske mreže prolaze proces učenja uzoraka gledajući na primjere podataka tako da mogu prepoznati te uzorke i primijeniti ih na nove stvari koje još nisu vidjele. Područje računarstva koje se bavi takvim aspektom obrade podataka zovemo neuro-računarstvo, a model obrade podataka umjetnom neuronskom mrežom (engl. Artifical Neural Network, ANN), te se takav model koristi u ovom radu.

2.1 Povijest umjetne neuronske mreže

Povijest neuronskih mreža potječe još iz 1943. kada su neurofiziolog Warren McCulloch i matematičar Walter Pitts istražujući princip rada neurona, objavili rad u kojem opisuju isti te predlažu matematički model za električni sklop koji modelira rad neurona. Nakon toga nastaje period stagnacije radi nemogućnosti implementacije neuronske mreže na računala. 1959. godine Bernard Widrow i Marcian Hoff sa Sveučilišta Stanford razvili su modele nazvane "ADALINE" (Adaptive Linear Network). Nakon njega slijedi "MADALINE“ (Multiple Adaptive Linear Network) koja je prva neuronska mreža primijenjena na stvarnom problemu. Nakon ovog otkrića opet dolazi do zatišaja u području neuro-računarstva sve do 80.-ih godina prošlog stoljeća kada dolazi do pojavljivanja višeslojnih mreža,… da bi od 90-tih godina do danas postale nezaobilazan aspekt svakodnevnog života.

2.2 Arhitektura neuronskih mreža

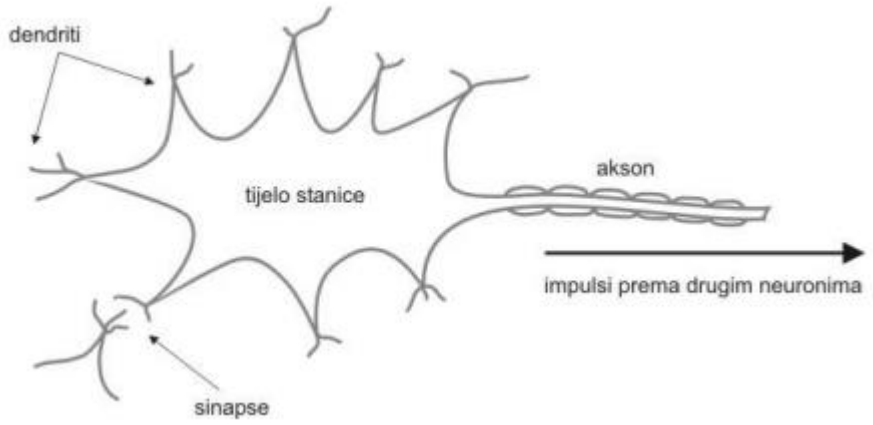
Arhitektura umjetnih neuronskih mreža je kopija arhitekture biološkog neuronskog sustava čija je osnova neuron. Neuron je osnovna i najsloženija jedinica živčanog sustava koji obrađuje i prenosi informacije drugim živčanim stanicama i mišićima. Pretpostavlja se da ljudski mozak ima oko 100 milijardi neurona, od kojih je svaki povezan s 1000 do 10 000 drugih neurona.[1] [1]

[1] https://enciklopedija.hr/natuknica.aspx?ID=43562

2.2.1 Biološki neuron

Biološki neuron je osnovna stanica živčanog sustava u svih živih bića. Više od 100 vrsta bioloških neurona sukladno svojoj funkciji raspoređeni su prema točno definiranom rasporedu. Osnovni dijelovi neurona su tijelo stanice, aksoni i dendriti. Slika 2.1 prikazuje građu biološkog neurona.

Stanično tijelo je poveznica između vanjskog i unutarnjeg dijela stanice. Iz staničnog tijela se proteže akson, dugačka cjevčica koja služi za prijenos električnih signala. Dendriti se protežu iz tijela stanice te služe za prijem podataka od drugih neurona. Oni su prekriveni sinapsama koje čine spojno sredstvo dvaju neurona zaslužno za komunikaciju s drugim neuronima. Prilikom slanja i primitka poruka neuroni prenose električne impulse duž svojih aksona. Prijenos informacija se odvija u jednom smjeru.



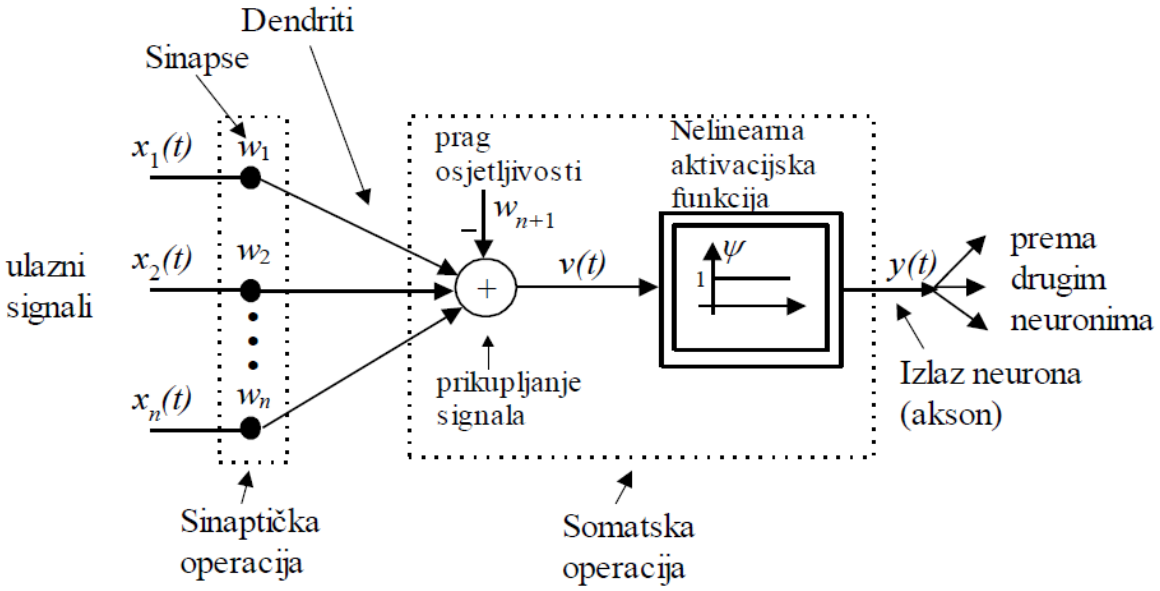
Slika 2.1 Građa biološkog neurona [1]

Jedan neuron sam po sebi relativno je beskoristan, ali u kombinaciji sa stotinama ili tisućama (ili mnogo više) drugih neurona, međusobna povezanost proizvodi odnose i rezultate koji često nadmašuju sve druge metode strojnog učenja.[2] [2]

2.2.2 Umjetni neuron

[2] Neural Networksfrom Scratch in Python; Harrison Kinsley & Daniel Kukieła

Inspiracija za razvoj umjetne neuronske mreže je ljudski mozak. To nije savršena usporedba, ali postoje mnoge sličnosti poput neurona, aktivacije i velikog broja međusobnih povezanosti iako su temeljni procesi veoma različiti. Glavna razlika je to što ljudski mozak nakon dvije slike može poprimiti novu informaciju i nastaviti je primjenjivati u tom kontekstu dok su neuronskoj mreži potrebne na stotine podataka kako bi mogle izvući željeni uzorak. S druge strane, sam čovjek ne može procesirati ogromne količine podataka u nekoliko minuta kao računalo. Primjer za to su društvene mreže gdje umjetna neuronska mreža obrađuje na milijune slika objavljenih taj dan, te ih prema prethodno prikupljenim podacima preporuča određenim korisnicima. Na slici 2.2 prikazan je model umjetnog neurona koji se također naziva i perceptron.



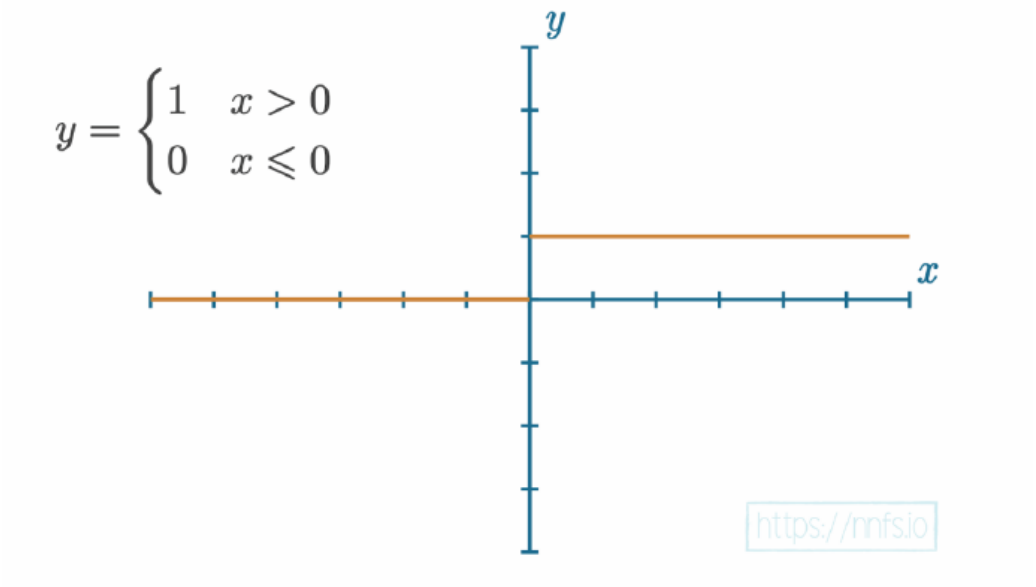
Slika 2.2 Građa umjetnog neurona[2]

Perceptron je osnovna jedinica umjetnih neuronskih mreža. Prijevod potiče od riječi percepcija čemu u suštini i odgovara, a to je kako se nešto opaža i procesira. Na slici 2.2 su vidljivi ulazni signali označeni sa $x_1(t)$, $x_2(t)$, …, $x_n(t)$. Težine su označene s w_1, w_2, …, w_n. One pomažu pri dodjeli važnosti svake ulazne varijable. Matematički zapis perceptron opisan je u navedenoj formuli:

$$y(t) = \sum_{i=1}^{n} w_i(t) * x_i(t) - w_{n+1} \quad (2.1)$$

Model koristi slijedeću analogiju: signali su opisani numeričkim iznosom i na ulazu u neuron množe se težinskim faktorom koji opisuje jakost sinapse; signali pomnoženi težinskim faktorima zatim se sumiraju analogno sumiranju potencijala u tijelu stanice; ako je dobiveni iznos iznad definirana praga, neuron daje izlazni signal.[3] [3] Taj model se koristi pri step aktivacijskoj funkciji. Pored nje se koriste i druge vrste aktivacijskih funkcija.

Perception je kreiran na modelu biološkog neurona, te u nekim slučajevima ima aktivacijsku step funkciju. Step funkcija oponaša neuron tako da, ako je rezultat dobiven koristeći formulu 2.1 veći od nula ona aktivira neuron. Ako je rezultat negativan onda ne dolazi ni do kakvog događaja. Navedeno je prikazano na slici 2.3

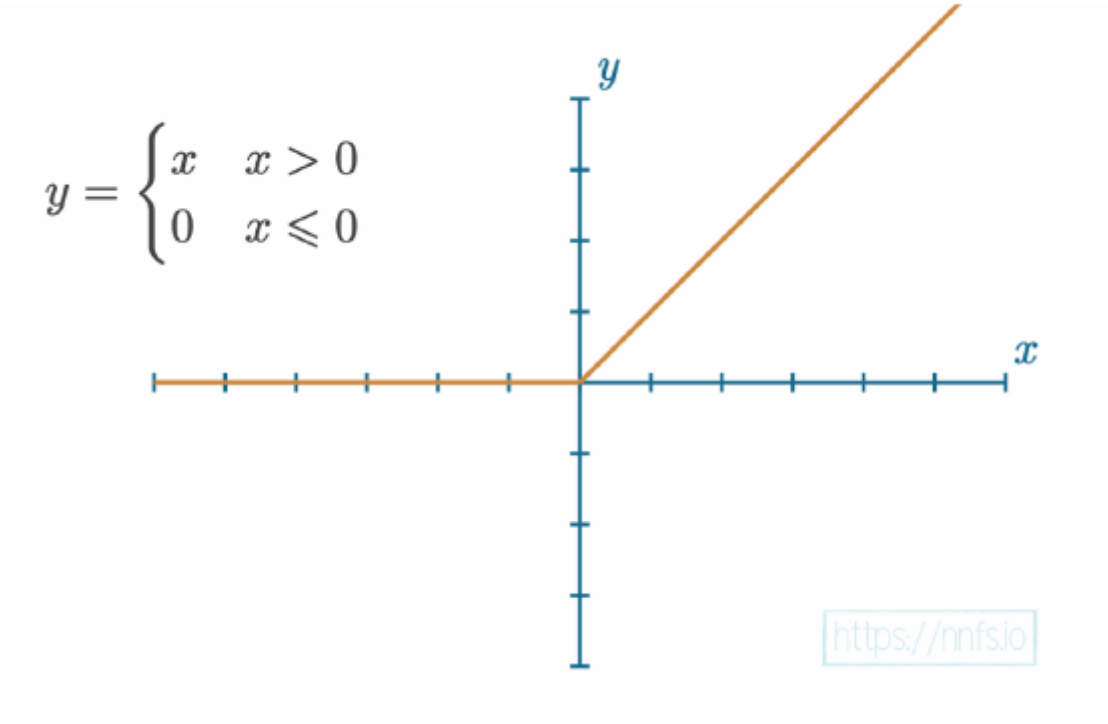


Slika 2.3Grafički prikaz step funkcije [3]

ReLu (Rectified Linear Unit) je jednostavna, ali vrlo važna aktivacijska funkcija. Ona se koristi iz više razloga dok bi glavni razlozi bili brzina i učinkovitost, te što ne dopušta da se većina neurona aktivira analogno. ReLU nije linearna funkcije te izbjegava problem prilikom slaganja više slojeva gdje se oni neće predstaviti kao jedan sloj. Na slici 2.4 vidljiv je njezin graf gdje vidimo da je funkcija zapravo najjednostavniji oblik y = x. Taj oblik je rezan na nula prema negativnoj strani. Ako je x manji ili jednak nuli onda je i sam y nula, a ako je x veći od nula onda

[3] Umjetne neuronske mreže (FER)

je y jednak x. Ova funkcija je bitna jer se koristi na nekim od najboljih konvolucijskih neuronskih mreža kao što su AlexNet i VGGNet.



Slika 2.4 Graf aktivacijske funkcije ReLU [4]

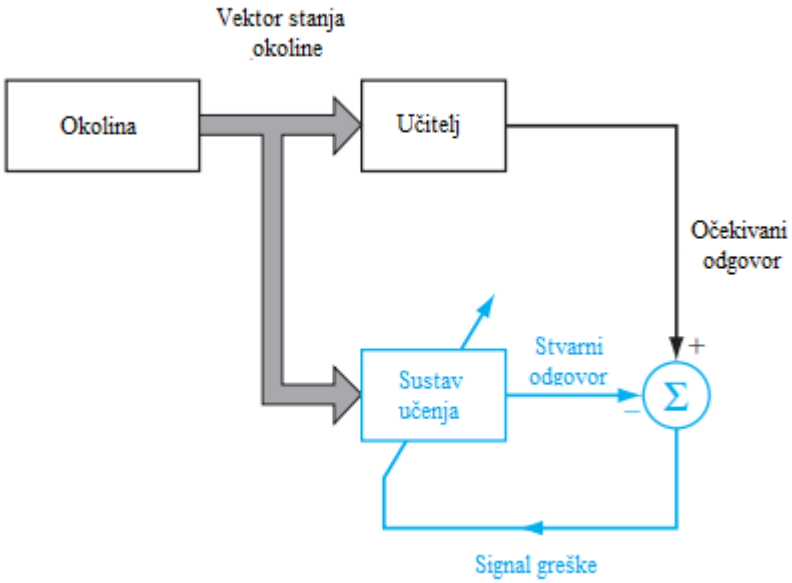
2.3 Vrste učenja neuronskih mreža

Čovjek svakodnevno iz svoje okoline prima razne informacije i uzorke iz kojih na dnevnoj bazi uči. Proces učenja neuronskih mreža imitira princip učenja čovjeka. Proces učenja neuronskih mreža podrazumijeva ponavljanje postupka u kojem se prikazuju ulazni podatci sa ili bez odgovarajućih izlaznih podataka. Iz toga možemo učenje neuronske mreže podijeliti na tri kategorije: nadzirano učenje, nenadzirano učenje i podržano učenje.

2.2.1 Nadzirano učenje

Nadzirano učenje također se naziva i učenje s učiteljem (eng. supervised learning) čiji je cijeli koncept baziran na tome da učitelj poznaje okruženje i zadaje neuronskoj mreži što da nauči.

Nadzirano učenje predstavlja osnovu za učenje korekcijom greške ili se također može reći da je to kontrolirani proces učenja na osnovu prethodnog iskustva. Na slici 2.4 prikazan je blok dijagram nadziranog učenja.



Slika 2.5 Nadzirano učenje [5]

Vektor stanja okoline poznat je učitelju i sustavu učenja. Neuronska mreža ima mogućnost naučiti vektor stanja okoline, ali mreža ne zna što taj vektor predstavlja. Učitelj iz informacija iz poznatog okruženja definira očekivane odgovore za željeni vektor. Sustav učenja iz vektora okoline stanja i skupa sinaptičkih vrijednosti daje stvarni odgovor. Ako se stvarni odgovor i očekivani odgovor ne poklapaju onda se šalje signal greške koji čini razliku između stvarnog i očekivanog odgovora. Na ovaj način učitelj prenosi svoje znanje iz okoline na mrežu, te se ovaj proces ponavlja dok se ne dođe do stanja gdje postoji mogućnost da se neuronska mreža sama, bez učitelja, daje odgovarajući odziv s obzirom na okolinu.

2.2.2 Nenadzirano učenje

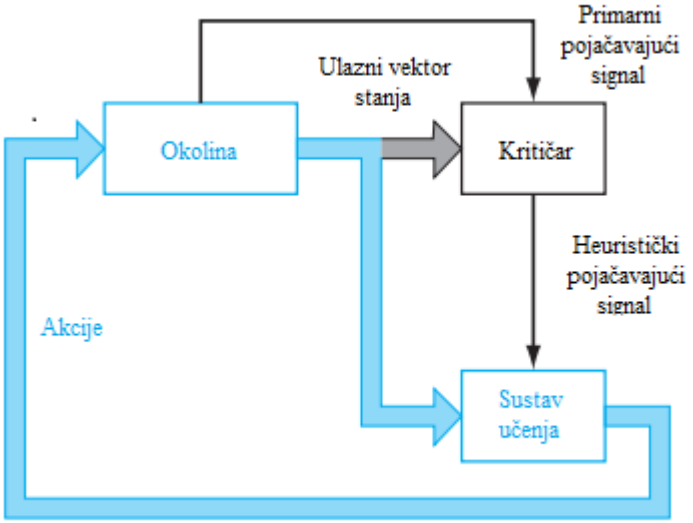
Nenadzirano učenje, kao što sam naziv sugerira, proces učenja koji se odvija bez učitelja. To znači da ne postoji definirana funkcija prema kojoj neuronska mreža treba učiti. Postoje dvije kategorije procesa učenja bez učitelja, a to je: pojačano i nenadzirano učenje.

2.2.2.1 Podržano učenje

Na slici 2.5 prikazan je blok dijagram pojačanog učenja gdje je vidljivo da ne postoji vektor stanja okoline s točno definiranim ulaznim i izlaznim veličinama. Iz slike je vidljiva zatvorena petlja čija komponenta je i okolina. Pošto je sustav učenja bez učitelja, on je izgrađen oko kritičara. On služi za pretvorbu primarnog signala pojačanja u kvalitetniji signal pojačanja, te se naziva signal heurističke armature, oba signala su skalarni ulazi. Sam sustav je osmišljen da uči pod odgođenim pojačanjem što znači da sustav promatra vremenski slijed podražaja također primljenih iz okoline, rezultat toga je generacija heurističkog signala za pojačanje. Cilj pojačanog učenja je minimiziranje troškovne funkcije definirane kao očekivanje kumulativnog troška radnje poduzete u slijedu koraka umjesto jednostavno neposrednog troška.[4][4] Funkcija ove vrste učenja je otkriti ranije određene radnje u slijedu vremenskih koraka koje su najbolji indikatori ove radnje, te ih vratiti u okoliš.

Cilj podržano učenja se ostvaruje tako što se izlazni signal dobiva kombinacijom heurističkog signala i ulaznog vektora stanja okoline. Taj generirani signal se vraća u okolinu te se tako stvara petlja procesa. Iako je ovaj sustav učenja naišao na određene poteškoće, ovakva vrsta učenja je i dalje zanimljiva korisnicima jer pruža mogućnost interakcije sustava za učenje sa vlastitom okolinom.

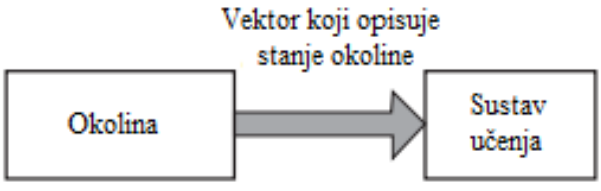
[4] (Haykin, 2009) 36.str



Slika 2.6 Podržano učenje [6]

2.2.2.2 Nenadzirano učenje

U ovom modelu učenja ne postoji vanjski utjecaj tj. nema učitelja niti kritičara koji nadziru proces učenja. Da bi se proces učenja na slici 2.6 mogao izvoditi koristi se natjecateljsko pravilo. Na primjer, moguće je koristiti neuronsku mrežu koja se sastoji od dva sloja: ulaznog i natjecateljskog. Ulazni sloj prima dostupne podatke. Natjecateljski sloj sastoji se od neurona koji se natječu jedni s drugima u skladu s pravilima učenja za odgovor na značajke sadržane u ulaznim podacima. U najjednostavnijem obliku mreža funkcionira u skladu sa strategijom „pobjednik-dobiva-sve“. U ovom načinu nenadziranog učenja, neuron s najvećim ukupnim brojem ulaza pobjeđuje konkurenciju i uključuje se, dok se svi ostali neuroni u mreži tada isključuju[5].[5]



Slika 2.7 Nenadzirano učenje [7]

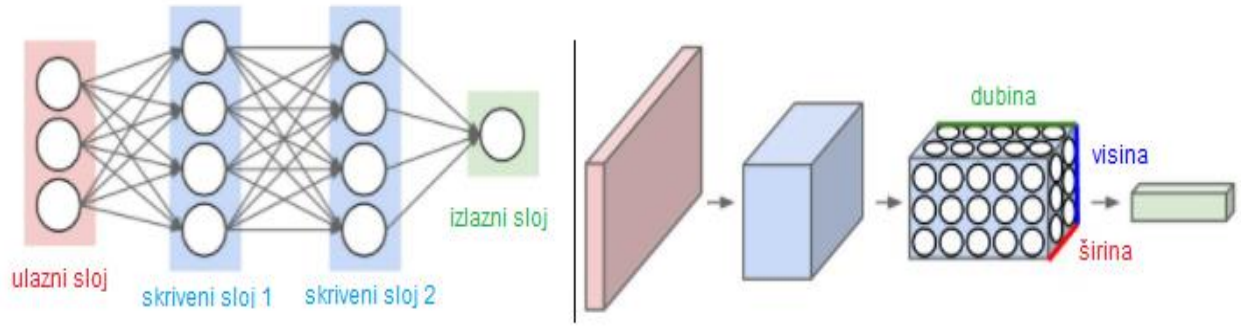
[5] (Haykin, 2009) 37-38.str

3. PRIJEDLOG RJEŠENJA

3.1 Konvolucijske neuronske mreže

Temelj za razumijevanje konvolucijskih neuronskih mreža je način na koji neuroni izvlače informacije potrebne za konstrukciju slike iz uzorka svjetlosti koji se nalazi u mrežnici oka. Profesori David Hubel i Torsten Wiesel su dokazali da neuroni u vidnom kontekstu raspoređeni po preciznoj arhitekturi. Navedena arhitektura je prikazana kao stupci u kojima su organizirane stanice sa sličnim funkcijama. To je ishod klasičnog eksperimenta koji se slučajno dogodio ranih 50-tih godina prošlog stoljeća.

Konvolucijske neuronske mreže (eng. Convolutional neural networks ili CNN) predstavljaju tip neuronskih mreža za obradu podataka koje se uspješno primjenjuju u analizi slika, teksta, zvuka i govora, a naziv potiče od matematičke operacije koja se zove konvolucija. Konvolucijska neuronska mreža po strukturi je jednaka običnoj neuronskoj mreži, te se sastoj od jednog ulaznog sloja, jednog izlaznog sloja kao i jednog ili više skrivenih slojeva, a razlikuje se po tome što joj slojevi imaju neurone raspoređene u 3 dimenzije: visina, širina i dubina. U obzir se uzima da su visina i širina ulaznog sloja mreže ovise o visini i širini slike dok se dubina referira na treću dimenziju volumena aktivacije, a ne na dubinu pune neuronske mreže. Dubina ulaznog sloja može varirati, a obično se može koristit 1 za crno-bijele slike ili 3 kao RGB komponenta boje, dubine skrivenih slojeva mogu biti različite. Navedeno je vidljivo na slici 3.1.

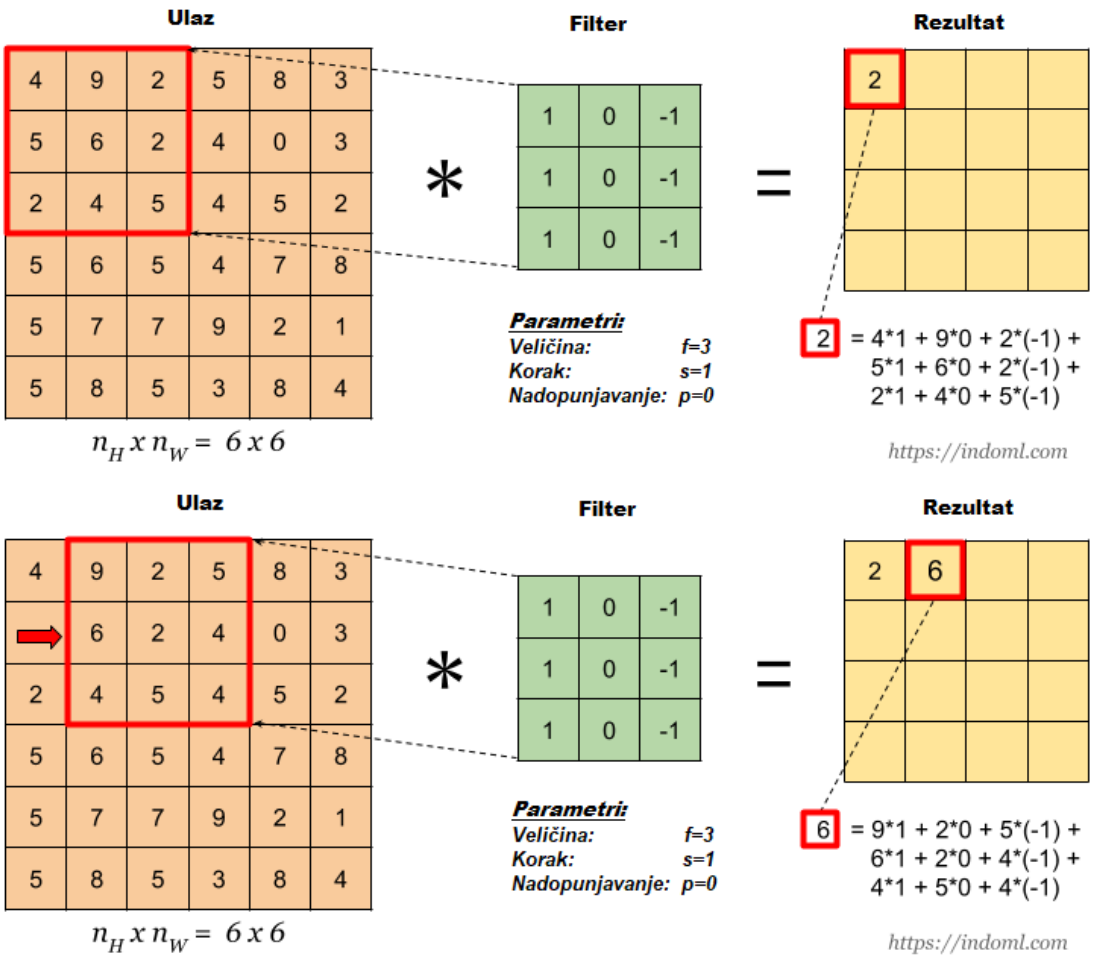


Slika 3.1 Lijevo- obična 3-slojna neuronska mreža; Desno- CNN raspodjela neurona na 3 dimenzije [8]

Najjednostavniji opis arhitekture konvolucijske neuronske mreže jest popis slojeva gdje svaki sloj na ulazu prihvaća 3D volumen i pretvara ga u izlazni 3D volumen pomoću funkcije. Općenito, prvi sloj po primitku slike prepoznaje osnovne elemnte slike kao što su rubovi, drugi sloj baziran na prvom može prepoznavati osnovne geometrijske oblike dok treći sloj pa na dalje, uz značajke iz prethodnih slojeva, mogu prepoznavati kompleksnije oblike. Arhitektura konvolucijskih neuronskih mreža prikazana je preko sljedećih komponenti: ulazni sloj, konvolucijski sloj, aktivacijska funkcija, sloj sažimanja i potpuno povezani slojevi. Temelj konvolucijskih neuronskih mreža je konvolucijski sloj.

Konvolucijski sloj je glavni dio konvolucijskih neuronske mreža jer se u njemu izvodi operacija konvolucije. Važna stavka svakog konvolucijskog sloja su filtri koji sadrže težine. Tokom procesa treniranja konvolucijskih neuronskih mreža, težine filtera se mijenjaju sukladno naučenom. Izlaz sloja će biti dvodimenzionalna matrica dok je točna veličina definirana veličina korakom pomaka filtra.

Prikaz rada konvolucijskog sloja je prikazan na slici 3.2. Ulaz i filter su prikazani matricama, te filter klizi matricom prema definiranom koraku pomaka. Na slici je vidljivo da je korak pomaka 1, filtar se pomiče za jedno mjesto u matrici ulaza. Polje [0,1] u matrici rezultata je 6, no da je korak pomaka bio 2 onda bi matrica rezultata bila 2x2 dimenzije, a polje [0,1] bi imalo vrijednost -4.



Slika 3.2 Način rada CNN [9]

3.1.1 VGGNet

Od početka istraživanja konvolucijskih neuronskih mreža pa do sada, uvođenjem novih skupova podataka, razvile su se brojne varijante CNN arhitekture. Neke od tih arhitektura su: ResNet, AlexNet, GoogLeNet, ZFNet, VGGNet, itd. Objavljivanjem AlexNet-a 2012. godine došlo je do revolucionarnog napretka u CNN. Ova mreža je pobijedila na ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) te je postala najsposobniji model za otkrivanje objekata. Ključna razlika je ta što se koristi ReLU umjesto THAN aktivacijske funkcije i optimizaciju na više grafičkih procesora.

2014. godine Karen Simonyan i Andrew Zisserman objavljuju svoj rad „Very Deep Convolutional Networks for Large-Scale Image Recognition“ u kojem predlažu model neurosnke mreže VGGNet. Model daje 92,7% točnosti što je među 5 najtočnijih testova na ImageNet-u. Prednosti nad AlexNet-om je ta što VGGNet koristi manje filtre 3x3, te uzima u obzir i dubinu

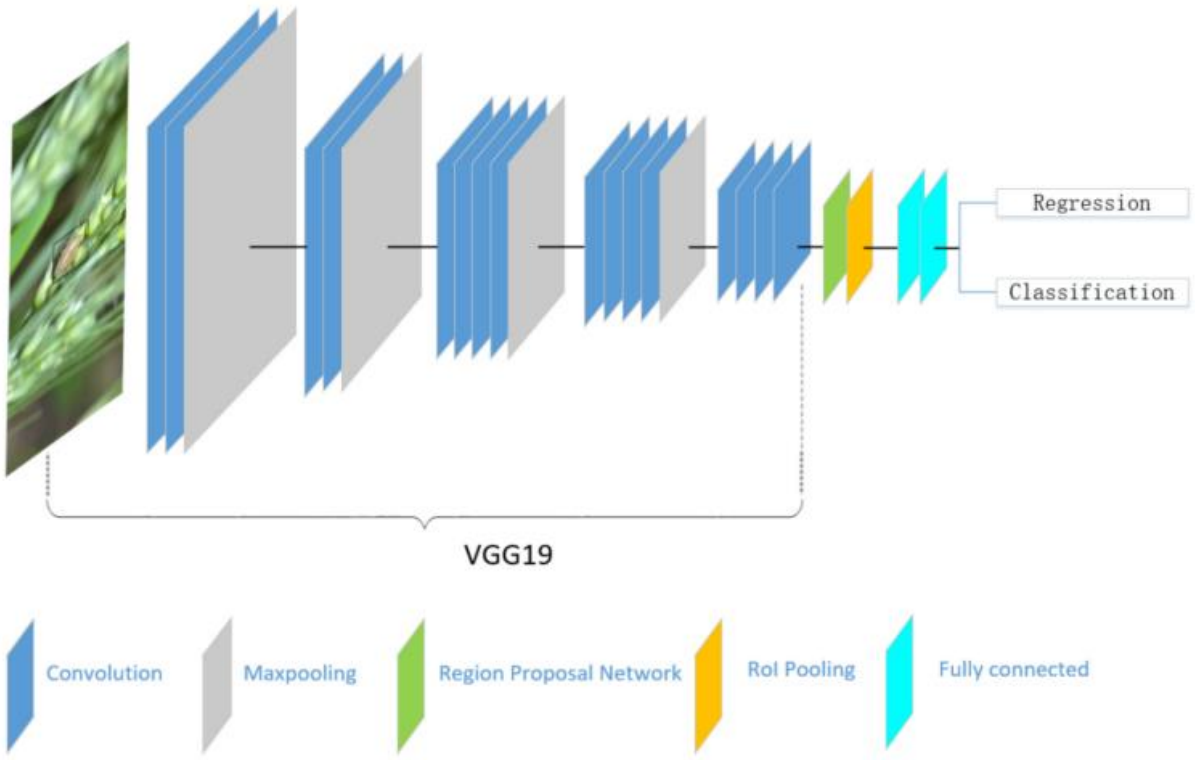
slike. Arhitektura VGG neuronske mreže je prikazana na slici 3.3. Slojevi VGGNet arhitekture su:

Ulazni sloj - VGG uzima RGB sliku dimenzija 224x224 piksela. Kako bi veličina ulazne slike bila dosljedna autori su za ILSVRC natjecanje izrezali središte slike tih dimenzija.

Konvolucijski sloj - korištenje manjeg filtra je moguće tako što se koriste dva 3x3 konvolucijska sloja u zamjenu za jedan sloj od 5x5 dimenzija. Za linearnu transformaciju se koristi konvolucijski filter veličine 1x1, nakon čega se koristi ReLU.

Potpuno povezani slojevi - oni dolaze na kraju mreže, a sastoje se od tri potpuno povezana sloja gdje prva dva imaju po 4096 kanala, a treći 1000 kanala.

Skriveni slojevi – svaki skriveni sloj koristi ReLU, te ne koristi lokalnu normalizaciju odziva (eng. Local Response Normalization) radi povećanja potrošnje memorije i vremena treninga bez većih prednosti.



Slika 3.3 Prikaz arhitekture modela VGG19 [10]

3.2 Pytorch

Pytorch autori za program tvrde „Okvir strojnog učenja otvorenog koda koji ubrzava put od prototipiranja istraživanja do implementacije proizvodnje.“[6][6]

Samo ime jezika inspirirano je popularnim znanstveni računalni okvir dubokog učenja Torch koji je napisan u Lua programskom jeziku. Autori PyTorcha su Adam Paszke, Sam Gross, Soumith Chintala i Gregory Chanan. Bili su inspirirani Pyton programskim jezikom te su napravili biblioteku za Python programe koja olakšava izgradnju mreža za duboko učenje. Pytorch je prikladan za iskusne kao i nove programere, te sve koji žele implementirati duboko učenje u svoje aplikacije. Pytorch gradi oko klase tenzora višedimenzionalni niz sličan onom u NumpPy-u. Pytorch pomaže nadopuniti razvoj strojnog učenja na nekoliko načina: ima vrlo visoke performanse na grafičkim procesorima s CUDA-om, opsežni blok neuronskih mreža i na kraju jednostavan intuitivan i stabilan API. Pytorch može besprijekorno skalirati do više grafičkih procesora koji rade na više strojeva pomoću izgrađenih i distribuiranih alata, što omogućuje primjenu ogromne količine računalnih resursa na najsloženije probleme. Prednosti Pytorch-a je to što jezgra C ++ uključuje JIT (eng. just-in-time) kompajler koji vam omogućuje da trenirate svoju mrežu na bilo kojem front-end jeziku kao što je Python ili C++. JIT prevoditelj koristi TorchScript za prijelaz koda između željnog načina rada za izradu prototipa i načina skripte za razvoj proizvodnje gdje izvođenje Python koda možda nije idealno. Jedna od najvećih prednosti je velika popularnost i samim time velika baza korisnika. Zahvaljujući aktivnom komuniciranju, istraživačima i pokretačima koji koriste Pytorch može se pristupiti bogatom ekosustavu alata, knjižnica, arhitekture modela i još mnogo toga što proširuje mogućnosti okvira. To dovodi do mogućnosti uključivanja u zajednicu kako bi se dobili odgovori na pitanja i probleme, te razvili svoje vještine kroz razne obrazovne resurse i internetske tečajeve. Pytorch je također integriran s popularnim cloud platformama.

[6] Pytorch

3.3 Programski kod

Ovdje je priložen kod koji je korišten za klasifikaciju slika. Kod u svojoj osnovi ostaje isti, a jedina stavka koda koja se mijenja su hiperparametri. U poglavlju 4.2 Opis rješenja ću prikazati hiperparametara i točnosti mreže.

```
!unzip depths.zip
```

```
[ ]  import torch
     import torch.nn as nn
     import torch.optim as optim
     from torch.optim import lr_scheduler
     import numpy as np
     import torchvision
     from torchvision import datasets, models, transforms
     import matplotlib.pyplot as plt
     import time
     import os
     import copy
     from sklearn.model_selection import train_test_split
```

```
[ ]  data_transforms = transforms.Compose([
         transforms.Resize(224),
         transforms.RandomVerticalFlip(p=1),
         transforms.ToTensor(),
     ])
     data_dir = './depths'
     image_datasets = datasets.ImageFolder(data_dir, data_transforms)
```

```
[ ]  indices = list(range(len(image_datasets)))
     targets = image_datasets.targets

     train_indices, test_indices = train_test_split(indices, test_size=0.2, stratify=targets)
```

```
[ ]  train_sampler = torch.utils.data.SubsetRandomSampler(train_indices)
     test_sampler = torch.utils.data.SubsetRandomSampler(test_indices)

     batch_size = 8

     train_loader = torch.utils.data.DataLoader(image_datasets, batch_size=batch_size, sampler=train_sampler, num_workers=2)
     test_loader = torch.utils.data.DataLoader(image_datasets, batch_size=batch_size, sampler=test_sampler, num_workers=2)
```

```
[ ]  device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
     print(device)
```

```
def train_model(model, criterion, optimizer, scheduler, num_epochs=25):
    start = time.time()

    best_model_wts = copy.deepcopy(model.state_dict())
    best_acc = 0.0

    for epoch in range(num_epochs):
        print('Epoch {}/{}'.format(epoch+1, num_epochs))
        print('-' * 10)

        for phase in ['train', 'validation']:
            if phase == 'train':
                model.train()
            else:
                model.eval()

            running_loss = 0.0
            running_corrects = 0

            if phase == 'train':
                for inputs, labels in train_loader:
                    inputs = inputs.to(device)
                    labels = labels.to(device)

                    optimizer.zero_grad()

                    with torch.set_grad_enabled(phase == 'train'):
                        outputs = model(inputs)
                        _, preds = torch.max(outputs, 1)
                        loss = criterion(outputs, labels)
                        if phase == 'train':
                            loss.backward()
                            optimizer.step()
                    running_loss += loss.item() * inputs.size(0)
                    running_corrects += torch.sum(preds == labels.data)
            elif phase == 'validation':
                for inputs, labels in test_loader:
                    inputs = inputs.to(device)
                    labels = labels.to(device)
```

```
                        optimizer.zero_grad()

                        with torch.set_grad_enabled(phase == 'train'):
                            outputs = model(inputs)
                            _, preds = torch.max(outputs, 1)
                            loss = criterion(outputs, labels)
                        running_loss += loss.item() * inputs.size(0)
                        running_corrects += torch.sum(preds == labels.data)

                if phase == 'train':
                    scheduler.step()

                if phase == 'train':
                  epoch_loss = running_loss / len(train_indices)
                  epoch_acc = running_corrects.double() / len(train_indices)
                elif phase == 'validation':
                  epoch_loss = running_loss / len(test_indices)
                  epoch_acc = running_corrects.double() / len(test_indices)

                print('{} Loss: {:.4f} -- Accuracy: {:.4f}'.format(phase, epoch_loss, epoch_acc))

                # deep copy the model
                if phase == 'validation' and epoch_acc > best_acc:
                    best_acc = epoch_acc
                    best_model_wts = copy.deepcopy(model.state_dict())

            print()

        time_elapsed = time.time() - start
        print('Training complete in {:.0f}m {:.0f}s'.format(
            time_elapsed // 60, time_elapsed % 60))
        print('Best validation accuracy: {:4f}'.format(best_acc))

        model.load_state_dict(best_model_wts)
        return model
```

```
[ ]  model_pt = models.vgg19_bn(pretrained=True) # with batch normalization
     num_ftrs = model_pt.classifier[6].in_features
     model_pt.classifier[6] = nn.Linear(num_ftrs, 10) # we are having 10 classes

     model_pt = model_pt.to(device)

     criterion = nn.CrossEntropyLoss()

     optimizer_ft = optim.SGD(model_pt.parameters(), lr=0.001, momentum=0.9)

     exp_lr_scheduler = lr_scheduler.StepLR(optimizer_ft, step_size=10, gamma=0.1)

     model_best_acc = train_model(model_pt, criterion, optimizer_ft, exp_lr_scheduler, num_epochs=50)
```

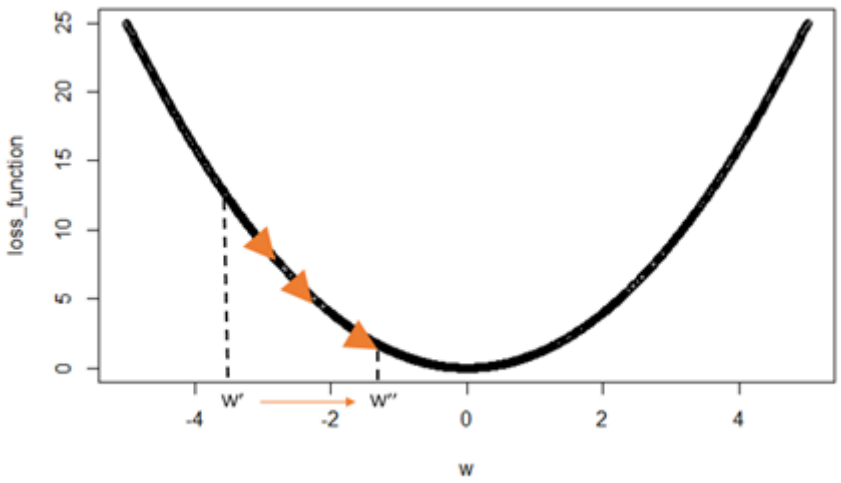
4. EVALUACIJA PREDLOŽENOG RJEŠENJA

4.1 Hiperparametri za optimizaciju koda

Prilikom same optimizacije možemo razmatrati parametre i hiperparametre. Da bi mreža bila točnija mora se izračunati ili pronaći najbolja konfiguracija tih parametara. Sama optimizacija se još uvijek proučava i čini kompleksan proces dok vrlo često stvara probleme prilikom treniranja mreža. Parametre model sam kroz učenje generira kao primjer možemo uzeti težine neurona spominjane u poglavlju 2.2.2 „Umjetni neuron“, i vrijednosti filtera u konvolucijskom sloju. Hiperparametri kroz sam proces učenja ne mijenjaju svoje početne vrijednosti. Neki od glavnih parametara koji su u ovom radu mijenjani su: broj epoha, stopa učenja, momentum, i veličina serije, .

Broj epoha (eng. number of epochs) u ovom kodu označeno je kao num_epochs. To je broj koji označava koliko će puta mreža učiti nad podatcima. Može se postaviti da broj epoha bude velik i staviti uvjet da se učenje prekine kod određenog postotka točnosti ili mrežu pokretati više puta i svaki put povećavati broj epoha dok se ne postigne određena točnost.

Stopa učenja (eng. learning rate) u ovom kodu označena kao lr. Ona služi za usporavanje procesa učenja dok se nesmetano konvergira. Prevelika stopa učenja uzrokuje promjene u točnosti i ograničava pronalazak odgovarajućeg minimuma funkcije greške, a premala stopa učenja usporava približavanje prema minimumu što je vidljivo iz slike 4.1. Cilj je da se w' što više približi minimumu.



Slika 4.1 Grafički prikaz stope učenja [11]

Momentum pomaže spoznati smjer sljedećeg koraka uz poznavanje prethodnih koraka. Pomaže u sprječavanju oscilacija. To je tehnika koja se koristi tijekom faze širenja unatrag. Kao što je rečeno u odlomku u vezi stope učenja cilj je da se parametri približe prema minimumu funkcije gubitaka. Sam proces može biti dug i utjecati na učinkovitost algoritama. Jedno od mogućih rješenja je praćenje prethodnih smjerova i njihovo držanje kao ugrađene informacije.

Veličina serije (eng. bach size) u ovom kodu označena je kao batch_size. To je broj poduzorka danih mreži nakon čega dolazi do ažuriranja parametara. Kada postoje milijarde podataka može doći do neučinkovitog hranjenja neuronske mreže sa svim tim podatcima. Dobra praksa je napuniti mrežu manjim uzorcima podataka. Taj broj ne smije biti premalen, ali niti pre velik jer tada dolazi do generaliziranja modela.

Prilikom pokušaja dobivanja što bolje točnosti mreže najviše sam mijenjala broj epoha i stopu učenja.

4.2 Opis rješenja

Kod sa najvećim postotkom točnosti naveden je u poglavlju 3.3 Programski kod. Parametri su navedeni na slici ispod.

```
[ ]  model_pt = models.vgg19_bn(pretrained=True) # with batch normalization
     num_ftrs = model_pt.classifier[6].in_features
     model_pt.classifier[6] = nn.Linear(num_ftrs, 10) # we are having 10 classes

     model_pt = model_pt.to(device)

     criterion = nn.CrossEntropyLoss()

     optimizer_ft = optim.SGD(model_pt.parameters(), lr=0.001, momentum=0.9)

     exp_lr_scheduler = lr_scheduler.StepLR(optimizer_ft, step_size=10, gamma=0.1)

     model_best_acc = train_model(model_pt, criterion, optimizer_ft, exp_lr_scheduler, num_epochs=50)
```

```
Training complete in 50m 20s
Best validation accuracy: 0.302895
```

Navođena činjenicom da su stopa učenja i broj epoha obrnuto proporcijalni tj. što je stopa učenja manja više epoha je potrebno, podesila sam hiperparametre. Smanjila sam stopu učenja kao i broj epoha.

```
model_pt = models.vgg19_bn(pretrained=True) # with batch normalization
num_ftrs = model_pt.classifier[6].in_features
model_pt.classifier[6] = nn.Linear(num_ftrs, 10) # we are having 10 classes

model_pt = model_pt.to(device)

criterion = nn.CrossEntropyLoss()

optimizer_ft = optim.SGD(model_pt.parameters(), lr=0.005, momentum=0.9)

exp_lr_scheduler = lr_scheduler.StepLR(optimizer_ft, step_size=10, gamma=0.1)

model_best_acc = train_model(model_pt, criterion, optimizer_ft, exp_lr_scheduler, num_epochs=30)
```

```
Training complete in 60m 26s
Best validation accuracy: 0.291759
```

Nakon ovoga dobila sam malo lošije rezultate nego prvi put, stoga zaključujem da je broj epoha od 50 možda bio prevelik. Sada povećavam stopu učenja te dodatno smanjujem broj epoha.

```
model_pt = models.vgg19_bn(pretrained=True) # with batch normalization
num_ftrs = model_pt.classifier[6].in_features
model_pt.classifier[6] = nn.Linear(num_ftrs, 10) # we are having 10 classes

model_pt = model_pt.to(device)

criterion = nn.CrossEntropyLoss()

optimizer_ft = optim.SGD(model_pt.parameters(), lr=0.1, momentum=0.9)

exp_lr_scheduler = lr_scheduler.StepLR(optimizer_ft, step_size=10, gamma=0.1)

model_best_acc = train_model(model_pt, criterion, optimizer_ft, exp_lr_scheduler, num_epochs=15)
```

```
Training complete in 80m 53s
Best validation accuracy: 0.280624
```

Rezultat je još lošiji naspram prijašnja dva testiranja mreže i iz ovoga ne mogu podržati izjavu o proporcionalnosti parametara. Sada pokušavam još više smanjiti stopu učenja, ali i smanjiti broj epoha naspram inicijalnog puta.

```
model_pt = models.vgg19_bn(pretrained=True) # with batch normalization
num_ftrs = model_pt.classifier[6].in_features
model_pt.classifier[6] = nn.Linear(num_ftrs, 10) # we are having 10 classes

model_pt = model_pt.to(device)

criterion = nn.CrossEntropyLoss()

optimizer_ft = optim.SGD(model_pt.parameters(), lr=0.0005, momentum=0.9)

exp_lr_scheduler = lr_scheduler.StepLR(optimizer_ft, step_size=10, gamma=0.1)

model_best_acc = train_model(model_pt, criterion, optimizer_ft, exp_lr_scheduler, num_epochs=40)
```

```
Training complete in 29m 11s
Best validation accuracy: 0.222717
```

Rezultat je se naglo pogoršao te povećavam stopu učenja, a smanjujem broj epoha.

```
model_pt = models.vgg19_bn(pretrained=True) # with batch normalization
num_ftrs = model_pt.classifier[6].in_features
model_pt.classifier[6] = nn.Linear(num_ftrs, 10) # we are having 10 classes

model_pt = model_pt.to(device)

criterion = nn.CrossEntropyLoss()

optimizer_ft = optim.SGD(model_pt.parameters(), lr=0.05, momentum=0.9)

exp_lr_scheduler = lr_scheduler.StepLR(optimizer_ft, step_size=10, gamma=0.1)

model_best_acc = train_model(model_pt, criterion, optimizer_ft, exp_lr_scheduler, num_epochs=25)
```

```
Training complete in 48m 4s
Best validation accuracy: 0.222717
```

Rezultat ostaje skoro nepromijenjen sa duplo većim vremenom potrebnim za treniranje mreže. Nakon čega pokušavam promijeniti i ostale hiperparametre. Postavljanjem momentuma na nula, ostavljanjem iste brzine učenja i povećavanjem brojem epoha rezultat se približava prvotnom, najboljem rezultatu.

```
model_pt = models.vgg19_bn(pretrained=True) # with batch normalization
num_ftrs = model_pt.classifier[6].in_features
model_pt.classifier[6] = nn.Linear(num_ftrs, 10) # we are having 10 classes

model_pt = model_pt.to(device)

criterion = nn.CrossEntropyLoss()

optimizer_ft = optim.SGD(model_pt.parameters(), lr=0.05, momentum=0)

exp_lr_scheduler = lr_scheduler.StepLR(optimizer_ft, step_size=10, gamma=0.1)

model_best_acc = train_model(model_pt, criterion, optimizer_ft, exp_lr_scheduler, num_epochs=50)
```

```
Training complete in 95m 46s
Best validation accuracy: 0.293769
```

Vođena time za sljedeće treniranje mreže smanjujem stopu učenja dok sve ostale parametre ostavljam istima čime rezultat opet pada na 27.89%.

```
model_pt = models.vgg19_bn(pretrained=True) # with batch normalization
num_ftrs = model_pt.classifier[6].in_features
model_pt.classifier[6] = nn.Linear(num_ftrs, 10) # we are having 10 classes

model_pt = model_pt.to(device)

criterion = nn.CrossEntropyLoss()

optimizer_ft = optim.SGD(model_pt.parameters(), lr=0.0005, momentum=0)

exp_lr_scheduler = lr_scheduler.StepLR(optimizer_ft, step_size=10, gamma=0.1)

model_best_acc = train_model(model_pt, criterion, optimizer_ft, exp_lr_scheduler, num_epochs=50)
```

```
Training complete in 95m 42s
Best validation accuracy: 0.278932
```

4.3 Usporedba slika

Na slikama iznad je vidljivo da slike na kojima je mreža testirana dosta odudaraju od slika snimanih ASUS Xtion PRO Live kamerom.

5. ZAKLJUČAK

Testiranjem koda napisanog koristeći VGGNet19 konvolucijsku neuronsku mrežu nisam dobila željene rezultate točnosti. Najveći rezultat je bio 30.29% dok su rezultati svih ostalih testiranja uz promjene parametara bili manji od navedenoga rezultata. Promatranjem rezultata i testiranjem mreže zaključila sam da kod iako funkcionalan ima grešku. VGGNet19 je jedna od najboljih konvolucijskih mreža današnjice, te se ona u potpunosti isključuje kao problem. Mijenjanjem hiperparametara ne dolazi ni do kakve velike razlike u točnosti mreže no njih ne možemo u potpunosti isključiti kao problem. Korištenjem hiperparametara također je postojala mogućnost korištenja neke od optimizacijskih metoda koje su izrazito kompleksne i sama ih nisam mogla primijeniti. Korištenjem ASUS Xtion PRO Live kamere dobivene slike nisu bile približne onima na kojima je mreža učena. Primjenjivanje takvih slika ne bi imalo svrhe uvrštavati u mrežu koja već daje izrazito loše rezultate.

6. LITERATURA

[1.] Leksikografski zavod Miroslav Krleža, **neuronska mreža**, https://enciklopedija.hr/natuknica.aspx?ID=43562

[2.] Kinsley H., Kukieła D., **Neural Networks from Scratch in Python,** 2020, Sentdex, Kinsley Enterprises

[3.] Dalbelo Bašić P., Čupić M. i Šnajder M., 2008. **Umjetne Neuronske Mreže**. 1. izdanje [elektronska knjiga] Zagreb, https://www.fer.hr/_download/repository/UmjetneNeuronskeMreze.pdf

[4.] Haykin, S. (2009). **Neural Networks and Learning Machines**. Third Edition. PEARSON, Prentice Hall

[5.] Haykin, S. (2009). **Neural Networks and Learning Machines**. Third Edition. PEARSON, Prentice Hall

[6.] Pytorch, https://pytorch.org/

[7.] Simonyan K. , Zisserman A., 2015. **Very deep convolutional networks for large-scale image recognition,** https://arxiv.org/pdf/1409.1556.pdf

[8.] IBM Cloud Education, **Convolutional Neural Network**, 20 listopada,2020., https://www.ibm.com/cloud/learn/convolutional-neural-networks

[9.] **CS231n Convonutional Neural Networks for Visual Recognition**, https://cs231n.github.io/convolutional-networks/

[10.] Fehlhaber, K., 2020. **Hubel And Wiesel & The Neural Basis Of Visual Perception - Knowing Neurons**. https://knowingneurons.com/2014/10/29/hubel-and-wiesel-the-neural-basis-of-visualperception/

[11.] IBM Cloud Education, **Neural Networks**, 17 kolovoza,2020., https://www.ibm.com/cloud/learn/neural-networks

[12.] **Student Notes: Convolutional Neural Networks (CNN) Introduction,** https://indoml.com/2018/03/07/student-notes-convolutional-neural-networks-cnn-introduction/

[13.] Wei J., **VGG Neural Networks: The Next Step After AlexNet**, 3 lipnja, 2019, https://towardsdatascience.com/vgg-neural-networks-the-next-step-after-alexnet-3f91fa9ffe2c

[14.] **VGG16 – Convolutional Network for Classification and Detection**, 20 studenoga, 2018, https://neurohive.io/en/popular-networks/vgg16/

[15.] Rosebrock A., **What is PyTorch**, 5 srpnja,2021., https://www.pyimagesearch.com/2021/07/05/what-is-pytorch/

[16.] Nyarko E.K., 2020. Meko računarstvo, FERIT predavanja

[17.]Xia D., Chen P., Wang B., Zhang J. , Xie C., **Insect Detection and Classification Based on an Improved Convolutional Neural Network**, 27 studenoga, 2018

POPIS SLIKA