SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 6348

VIZUALIZACIJA I INTERPRETACIJA KONVOLUCIJSKIH MODELA

Martin Sršen

Zagreb, lipanj 2019.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
ODBOR ZA ZAVRŠNI RAD MODULA

Zagreb, 15. ožujka 2019.

ZAVRŠNI ZADATAK br. 6348

Pristupnik: **Martin Sršen (0036502135)**
Studij: Računarstvo
Modul: Računarska znanost

Zadatak: **Vizualizacija i interpretacija konvolucijskih modela**

Opis zadatka:

Duboki konvolucijski modeli su glavni sastojak mnogih praktičnih primjena računalnog vida. Međutim, često se javlja kritika da takvi modeli nisu spremni za industrijsku upotrebu zbog loše interpretabilnosti, odnosno, nemogućnosti modela da svoju odluku obrazloži ljudima. Zbog toga postupci za interpretiranje odluka i vizualiziranje naučenih parametara konvolucijskih modela predstavljaju zanimljivo područje istraživanja.
U okviru rada, potrebno je odabrati okvir za automatsku diferencijaciju te upoznati biblioteke za rukovanje matricama i slikama. Proučiti i ukratko opisati postojeće pristupe za interpretiranje i vizualiziranje konvolucijskih modela. Uhodati jednostavne postupke vizualiziranja naučenih filtara te interpretiranja klasifikacijske odluke modela prednaučenog na ImageNetu. Prikazati i ocijeniti ostvarene rezultate. Predložiti pravce budućeg razvoja.
Radu priložiti izvorni i izvršni kod razvijenih postupaka, ispitne slijedove i rezultate, uz potrebna objašnjenja i dokumentaciju. Citirati korištenu literaturu i navesti dobivenu pomoć.

Zadatak uručen pristupniku: 15. ožujka 2019.
Rok za predaju rada: 14. lipnja 2019.

Mentor:
Prof. dr. sc. Siniša Šegvić

Djelovođa:
Izv. prof. dr. sc. Tomislav Hrkać

Predsjednik odbora za završni rad modula:
Marko Čupić
Doc. dr. sc. Marko Čupić

Zahvaljujem mentoru prof. dr. sc. Siniši Šegviću na stručnim savjetima i udjeljenoj pomoći pri izradi rada.

Zahvaljem svima koji su na bilo koji način pomogli u izradi rada.

Sadržaj

1. Uvod

Umjetna inteligencija (engl. artificial intelligence), ponekad nazvana strojna inteligencija je područje računalne znanosti koje pokušava kreirati stroj koji će oponašati kognitivne funkcije čovjeka, kao što su učenje i rješavanje problema. Ti problemi mogu biti vrlo jednostavni za čovjeka, dok su u istom trenutku za računalo vrlo teški i zahtjevni. Grana umjetne inteligencije koja proučava računalne algoritme koji se automatski poboljšavaju iskustvom bez uporabe eksplicitnih uputa, tj. uče na temelju datih podataka naziva se strojno učenje (engl. machine learning). Zanimljivo područje strojnog učenja kojeg ćemo proučavati u ovom radu je računalni vid, koji se bavi teorijom iza umjetnih sustava koji izvlače informacije iz slika. Ovo područje je inspirirano ljudskim vidom koji je najvažnije osjetilo kod ljudi. Neke od pod-domena računalnog vida su rekonstrukcija prizora (utvrđivanje oblika i izgleda stvarnih objekata), otkrivanje događaja, praćenje objekata u videu, prepoznavanje objekata (utvrđuje dali slika sadrži određeni object, značajku ili aktivnost), procjena 3D pozicije, učenje, indeksiranje, procjena pokreta te restauracija slika (uklanjanje smetnji na slikama).

Velika prekretnica u računalnom vidu se dogodila 2012. kada je konvolucijska neuronska mreža (tip duboke neuronske mreže) po prvi put producirala značajno bolje rezultate od metoda s ručno oblikovanim značajkama na glavnom natjecanju računalnog vida, ImageNet natjecanje u detekciji i prepoznavanju 1000 različitih objekata na slikama. Umjetne neuronske mreže imaju začetke u 40-tim godinama 20. Stoljeća, ali nisu doživjele procvat zbog slabe razvijenosti računala te nemogućnosti izvođenja većih i kompleksnijih izračuna. Razvojem grafičkih kartica umjetne neuronske mreže doživljavaju procvat jer omogućuju veliku količinu paralelnih izračuna. Prednost korištenja umjetne neuronske mreže je što se ne trebaju koristiti kompleksni matematički modeli te ručno upisivanje filtara već ih mreža sama uči na osnovu danih podataka.

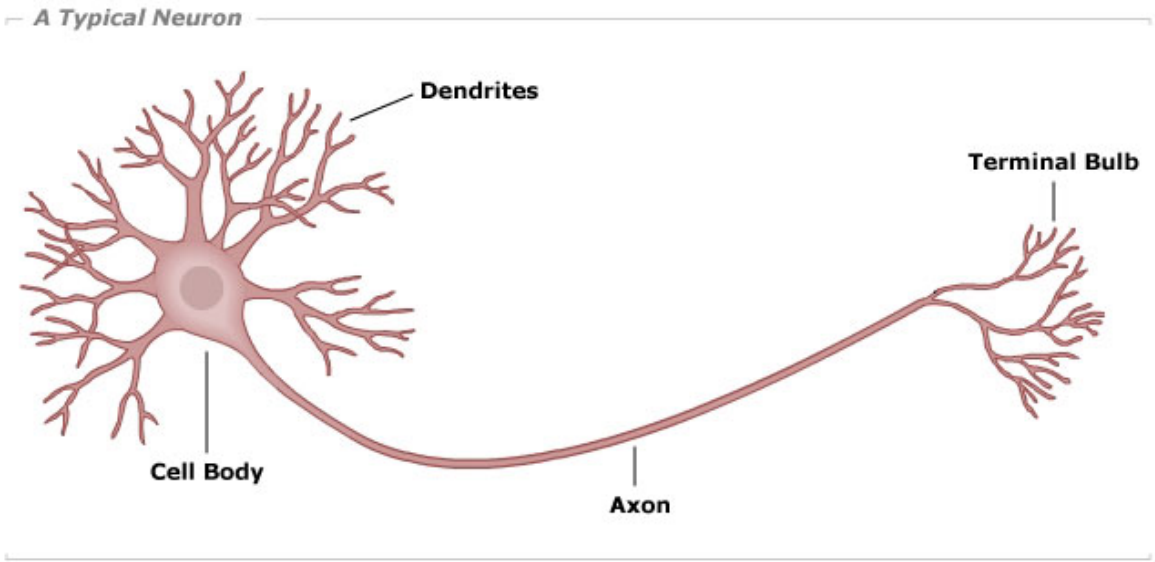
Cilj ovog rada je upoznati se s metodama vizualizacije te interpretacije konvolucijskih modela te njihovih parametara, proučiti metode vizualizacije, tj. vidjeti na koji način umjetna neuronska mreža prepoznaje objekt na slici te pokazati koji dijelovi slike igraju ključnu ulogu u klasifikaciji.

2. Umjetna neuronska mreža

Umjetna neuronska mreža jedan je od važnijih algoritama strojnog učenja koji je motiviran na principu rada biološkog mozga. Mozak je središnji organ svakog bića koji mu omogućuje učenje, pamćenje te rješavanje raznih problema. Sastoji se od oko 86 milijuna neurona međusobno povezanih s 10^{14} - 10^{15} sinapsi. Cilj umjetne inteligencije je upravo reproducirati mogućnosti mozga te ih poboljšati, pa je zbog toga razumno krenuti od mozga te reproduciranju funkcioniranja istog.

2.1. Neuron

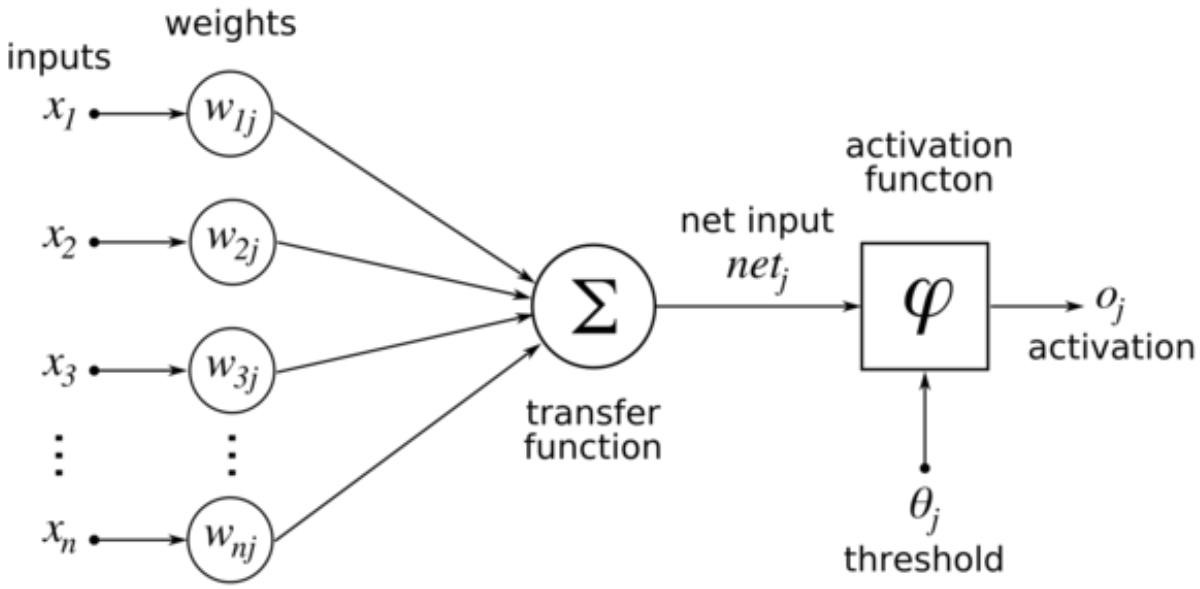
Neuron je osnovna građevna jedinica svake umjetne neuronske mreže. Inspiriran biološkim neuronom koji funkcionira tako da prikuplja informacije iz susjednih, povezanih neurona kroz dendrite, obrađuje signal te prosljeđuje kroz aksone preko sinaptičkih veza u slučaju da je signal dovoljno jak.



Slika 2.1: Neuron živčanog sustava. (Izvor: quora.com)

Umjetni neuron je elementarna jedinica u umjetnoj neuronskoj mreži koji je zapravo matematička funkcija koja pokušava imitirati rad biološkog neurona. Funkcionira tako da prima jedan ili više ulaza, množi ih s pripadajućim težinama, sumira ih te dodaje prag kako bi proizveo izlaz. Uloga praga je da se odredi

minimalna vrijednost koju neuron mora imati da bi postao aktivan. Obično se ulaz množi sa pripadajućim težinama koje predstavljaju jačinu veze između dva neurona te se suma prenosi kroz nelinearnu aktivacijsku/prijenosnu funkciju. Izlaz aktivacijske funkcije jednog neurona predstavlja ulaz idućeg neurona.



Slika 2.2: Model matematičkog neurona. (Izvor: m.tau.ac.il)

Funkcija koja predstavlja rad umjetnog neurona:

$$y = f\left(\sum_{i=0}^{n} x_i w_i + \theta\right) \tag{2.1}$$

gdje je značenje varijabli iduće:

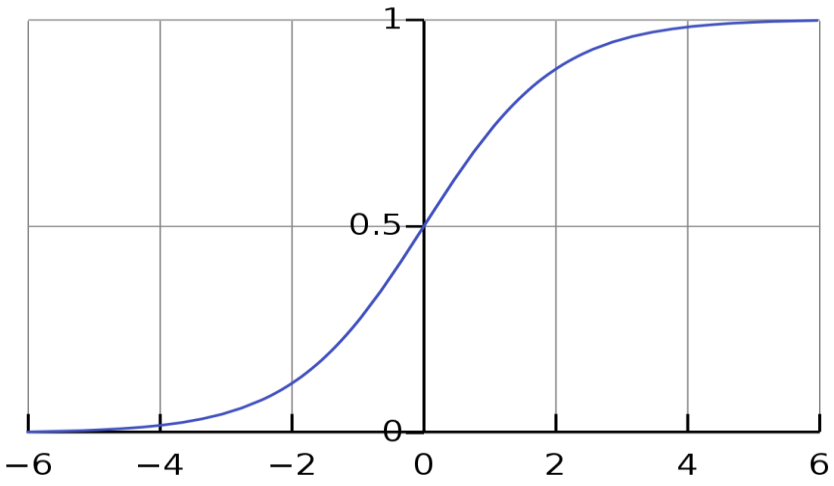
- y – izlaz neurona, tj. ulaz u idući neuron u nizu
- f – aktivacijska/prijenosna funkcija
- x – ulaz u neuron
- w – težina između prijašnjeg i trenutnog neurona
- θ – pomak

Umjetni neuroni se razlikuju na temelju aktivacijskih/prijenosnih funkcija. Ona može biti proizvoljna te se odabire ovisno o problemu koji rješava. Kada ne bi koristili aktivacijsku funkciju izlaz bi bio potpuno linearno ovisan o ulazu, one uvode potrebnu nelinearnost u neuronsku mrežu. Bez nelinearnosti neuronske mreže ne

bi mogle modelirati nelinearne pojave kao što su slike, zvuk, govor i slično. Važno svojstvo aktivacijske funkcije je da bude diferencijabilna da bi se mogli izračunati gradijenti pogreške u odnosu na težine, a zatim u skladu s time optimizirati težine pomoću gradijenta spusta ili bilo koje druge tehnike optimizacije kako bi se smanjila pogreška. Aktivacijske funkcije koje se najčešće koriste su:

- Sigmoidalna funkcija – glavno obilježje ove funkcije je da svaki ulaz skalira na interval između 0 i 1. Jako pozitivni brojevi poprimaju vrijednost približno 1, dok jako negativni približno 0. Sigmoidalna funkcija danas se koristi samo za predikciju kategoričkih slučajnih varijabli s binomnom razdiobom. Jedan od glavnih problema sigmoidalne fukcije je što kod povratne propagacije gradijent postaje sve manji kroz mrežu, te za duboke neuronske mreže može potpuno nestati.

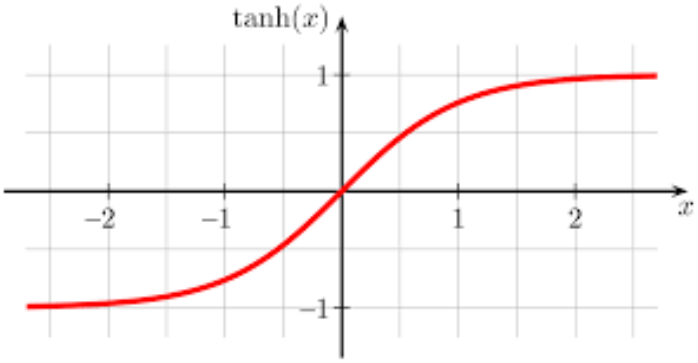
$$f(X) = \sigma(X) = \frac{1}{1 - e^{-X}} \tag{2.2}$$



Slika 2.3: Sigmoidalna funkcija. (Izvor: Wikipedia)

- Tanh – glavna razlika u odnosu na sigmoidalnu funkciju je da skalira ulaz na interval između -1 i 1. Ovo rješava problem centriranja vrijednosti oko 0, te je zbog toga za latentne aktivacije bolji izbor od sigmoidalne funkcije.

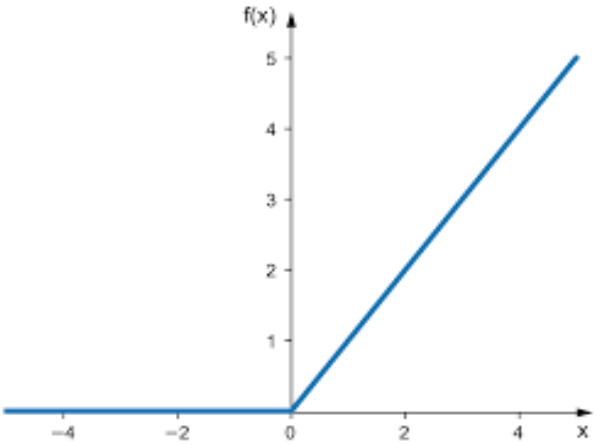
$$f(X) = \tanh(X) = \frac{\sinh(X)}{\cosh(X)} = \frac{e^{X} - e^{-X}}{e^{X} + e^{-X}} \tag{2.3}$$



Slika 2.4: Tangens hiperbolni. (Izvor: Wikipedia)

- ReLU (Rectified Linear Unit) – funkcionira tako da propušta ulaze koji su pozitivni, tj. veći od 0. Glavna prednost je brzina računanja što smanjiva vrijeme treniranja. Glavni problem je što neuroni mogu „odumrijeti" tako da preveliki gradijent izmjeni težine tako da se neuron više nikad ne aktivira. Problem se može rješiti tako da vrijednosti koje su manje od 0 pomnožimo s izabranom vrlo malom konstantom, najčešće 0.01, te tako dobijemo da gradijent ne bude 0, već jednak izabranoj konstanti.

$$\mathrm{f(X)} = \max(0, X) \tag{2.4}$$

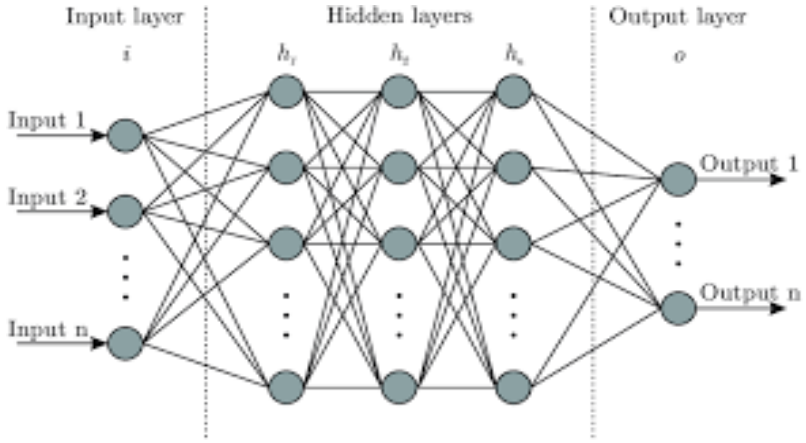


Slika 2.5: Linearna rektifikacijska funkcija. (Izvor: sebastianraschka.com)

2.2. Unaprijedna mreža s potpuno povezanim slojevima

Neuronska mreža nastaje povezivanjem većeg broja neurona. Osnovna arhitektura mreže je podijeljena na 3 sloja:

1. Ulazni sloj – predstavlja ulazne podatke u mrežu
2. Skriveni slojevi – ne moraju postojati, ali su upravo oni ključni u rješavanju težih problema
3. Izlazni sloj – izlaz mreže



Slika 2.6: Arhitektura neuronske mreže. (Izvor: researchgate.net)

Na primjer kod klasifikacijskog problema prepoznavanja rukom napisanih znamenki broj ulaznih neurona bit će jednak broju piksela u ulaznoj slici te će mreža imati 10 izlaznih neurona, po jedan za svaku znamenku koju prepoznajemo.

Svaki sloj neuronske mreže ima vezu sa susjednim tako da izlazi prethodnog sloja predstavljaju ulaze u idući sloj. Moguće su i arhitekture u kojem postoje veze na prijašnje slojeve, ali to nećemo razmatrati. U slučaju kada je svaki neuron prethodnog sloja povezan sa svakim neuronom idućeg sloja to nazivamo potpuno povezanom neuronskom mrežom. Neuronske mreže koje imaju 2 ili više skrivenih slojeva nazivamo dubokim neuronskim mrežama. Veći broj slojeva nam omogućuje preciznije dekomponirati problem na niz jednostavnijih dijelova te je time dekompozicija preciznija i rješenje je pouzdanije, ali je učenje dulje te složenije s povećanjem broja slojeva.

2.3. Učenje neuronske mreže

Učenje neuronske mreže predstavlja modifikaciju promjenjivih parametara mreže, težina u svrhu što boljih rezultata mreže. Razlikujemo 3 načina učenja s obzirom na oblik podataka:

1. nadzirano učenje (engl. supervised) – za svaki ulaz u mrežu znamo očekivani izlaz. Kao specijalni slučaj nadziranog možemo navesti polunadzirano učenje (engl. semi-supervised learning) koje koristi manju grupu podataka za koje znamo izlaz
2. nenadzirano učenje (engl. unsupervised) – znamo samo ulaze u mrežu
3. podržano učenje (engl. reinforcement) – koristi se sustav nagrađivanja, dajući mreži povratnu informaciju

Cilj neuronske mreže je minimizirati funkciju gubitka. Razlikujemo dvije grupe nadziranih modela s obzirom na problem:

1. klasifikacijski problem – tražimo kategoriju kojoj pripada dani ulaz
2. regresijski problem – tražimo kontinuirani izlaz u odnosu na ulaz

Najčešće korištena funkcija gubitka za problem regresije je funkcija srednje kvadratne pogreške (engl. mean square error) ili L2 gubitak, gdje je y očekivani izlaz te h dobiveni izlaz.

$$MSE = \frac{\sum_{i=1}^{n}(y_i - h_i)^2}{n} \tag{2.5}$$

Za problem binarne klasifikacije se najčešće koristi unakrsna entropija (engl. cross entropy loss)

$$L = -(y * \log(h) + (1 - y) * \log(1 - h)) \tag{2.6}$$

Kada kod klasifikacije imamo više od 2 izlazne klase, imamo neurona jednak broju klasa koje klasificiramo. Vrijednost neurona na i-toj poziciji izlaza predstavlja aktivaciju za i-tu klasu. Neuron sa najvećom aktivacijom smatramo klasom ulaza. Obično koristimo softmax funkciju na izlazne aktivacije da dobijemo vjerovatnosti pojedine klase.

$$P(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}} \tag{2.7}$$

Sada kada znamo kako izračunati grešku treba vidjeti kako promijeniti parametre mreže tako da u budućnosti greška bude što manja. Formula kojom ažuriramo parametre/težine neuronske mreže je sljedeća:

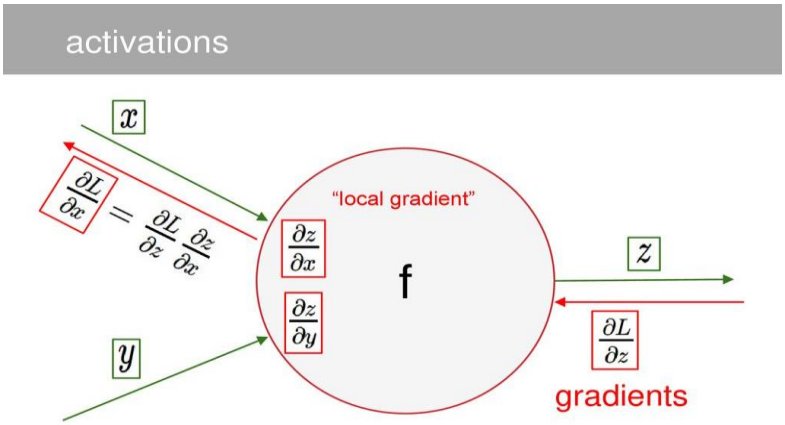
$$w_n = w_n - \alpha \cdot \frac{dL}{dw_n} \tag{2.8}$$

gdje je α faktor učenja, hiper-parametar koji kontrolira koliko prilagođavamo težine i pragove s obzirom na gradijent gubitka. Što veći faktor uzmemo, brže je učenje, ali riskiramo da algoritam učenja divergira te promašimo globalni minimum.

Sada još trebamo vidjeti kako dobiti gradijente po težinama, tj. vidjeti kako dobiti koliko koja težina te pomak utječu na promjenu gubitka. Za to se koristi algoritam propagacije pogreške unazad (engl. backpropagation), glavni i neizostavni algoritam za treniranje neuronske mreže. Upravo ovaj algoritam daje mogućnost mreži da uči na svojim pogreškama te postane bolja u rješavanju određenih problema. Koristimo ga da saznamo koliko je koja težina utjecala na grešku i nakon toga ažuriramo pripadajuće težine s obzirom na faktor učenja. Algoritam se temelji na sljedećem pravilu ulančavanja:

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx} \tag{2.9}$$

Algoritam propagacije pogreške unazad kreće od funkcije gubitka te pomoću pravila ulančavanja propagira pogrešku do prvog sloja.



Slika 2.7: Propagacija pogreške unatrag kroz neuron. (Izvor: slideplayer.com)

Cijeli proces učenja neuronske mreže možemo podijeliti u 5 koraka:

1. šaljemo ulazne podatke na ulaz neuronske mreže
2. računamo izlaz mreže na temelju datih ulaznih podataka
3. računamo grešku izlaza s obzirom na očekivani izlaz pomoću funkcije gubitka
4. propagiramo gradijent pogreške prema ulazu i ažuriramo parametre neuronske mreže tako da smanjimo ukupnu grešku
5. ponavljamo prethodno opisani postupak dok ne dobijemo očekivane rezultate ili prođe zadani broj iteracija

Veliki problem neuronske mreže je kada dođe do situacije da je mreža toliko dobro prilagođena skupu podataka na kojem je učena da je teško generalizirati i predvidjeti nove podatke. U praksi je ovaj problem teško otkriti pa je zbog toga dobra praksa podijeliti podatke za učenje na 3 cjeline:

1. skup za treniranje (za manji skup podataka obično 60% dok za i do 98%)
2. skup za validaciju (za manji skup 20%, za veći i do 1%)
3. skup za testiranje (za manji 20%, za veći 1%)

Ovo je važno zbog toga da lakše prepoznamo problem. Postoje razni načini regularizacije da izbjegnemo ovaj problem, ali nećemo ih ovdje razmatrati.

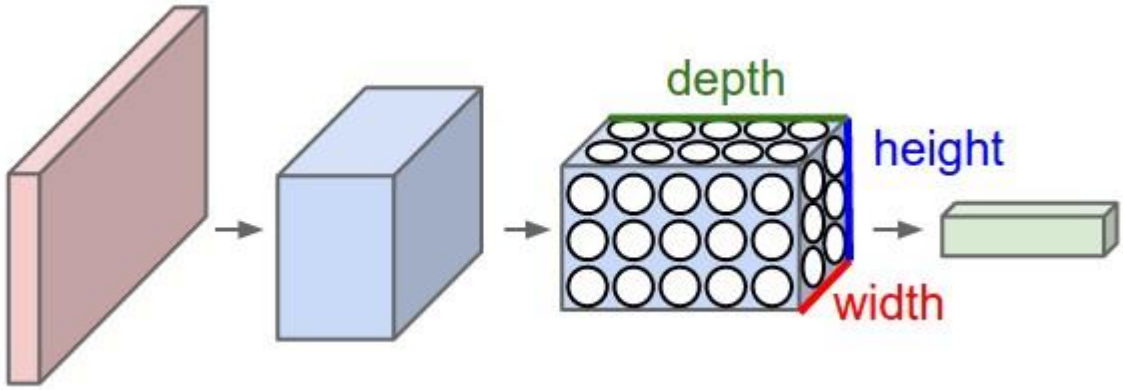
3. Konvolucijske neuronske mreže

Jedno od područja u kojima je neuronska mreža našla široku upotrebu je u području računalnog vida. Za konvolucijsku neuronsku mrežu možemo reći da je poboljšana verzija neuronske mreže specijalizirane za ulazne podatke koji imaju topološku strukturu, na primjer jezik, govor, vid, itd. Tipične konvolucijske neuronske mreže imaju 18-152 sloja čime ih možemo smjestiti u duboke neuronske mreže. Takve mreže dobile su ime po matematičkoj operaciji konvolucije koja je definirana nad dvije funkcije te daje treću funkciju koja označava količinu preklapanja prve te pomaknute druge funkcije. Veliki napredak u odnosu na mrežu s potpuno povezanim slojevima je što koristi konvolucijske filtare koji omogućuju prepoznavanje određenih karakteristika u slikama, kao npr. rubova. Kod klasičnih neuronskih mreža problematično je raditi sa slikama jer ako uzmemo u obzir da slika ima 200x200 pixela, te još ako dodamo boju taj broj pomnožimo s 3. To znači da bi ulaz u mrežu morao imati 120000 neurona te još ako koristimo potpuno povezanu mrežu problem se čini nemogućim. Konvolucijske mreže to rješavaju tako da koriste konvolucijske filtare koji traže određene karakteristike i oblike u slikama te ih spremaju u mape značajki koje se šalju dalje kroz mrežu. Kada slika prolazi kroz filtare, također joj se smanjuju dimenzije te se ubrzava daljnje izračunavanje kroz mrežu. Zbog kompleksnosti rada sa slikama u konvolucijskim neuronskim mrežama obično se koriste slojevi sažimanja (engl. pooling) koji smanjuju prostorne dimenzije latentnih reprezentacija, obično za faktor 2. Konvolucijske mreže kao ulaz primaju podatke u obliku višedimenzionalnog polja, u slučaju crno-bijelih slika 2D, te u boji 3D.

3.1. Arhitektura

Za razliku od klasičnih mreža koje imaju slojeve u jednoj dimenziji, konvolucijska neuronska mreža ima slojeve u tri dimenzije: širina, visina i broj mapa značajki. Širina i visina predstavljaju dimenzije mapa značajki. Svaki sloj ima funkciju da transformira ulazne trodimenzionalne podatke u izlazne trodimenzionalne podatke, osim u slučaju potpuno povezanih slojeva (eng. fully connected layers) koji ima

jednu dimenziju i osnovna svrha mu je da možemo svrstati ulaz u jednu od kategorija.



Slika 3.1: Konvolucijska neuronska mreža. (Izvor: css231n.github.io)

Konvolucijska neuronska mreža se sastoji od 3 glavne vrste slojeva: konvolucijskih slojeva (eng. convolutional layers), slojeva sažimanja (eng. pooling layers) i potpuno povezanih slojeva (eng. fully connected layers).

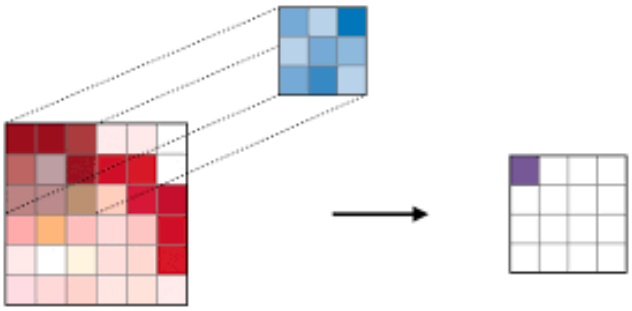
Konvolucijski sloj je najznačajniji dio konvolucijske neuronske mreže po kojoj je dobila i ime. Sastoji se od jednog ili više filtara koji sadrže težine koje se uče algoritmom propagacije pogreške unatrag. Konvolucijski filtar ili jezgra je zapravo tenzor trećeg reda (3D matrica) obično puno manje širine i visine od ulaza. Važno je napomenuti da dubina filtara mora biti jednaka dubini ulaza, tj. dubini ulaznih mapi značajki u konvolucijski sloj. Konvolucijski sloj je definiran s 5 osnovnih parametra:

1. Ulaz (eng. input) – oblik ulaznog tenzora (broj ulaznih kanala, visina, širina)
2. Težine (eng. weights) – filtari oblika (broj izlaznih kanala, visina, širina)
3. Pomak (eng. bias) – opcionalni tenzor (oblik jednak broju izlaznih kanala)
4. korak (eng. stride) – odgovara koraku pomaka filtara
5. nadopunjavanje (eng. padding) – koristi se najčešće kad zahtijevamo da izlazne dimenzije konvolucijskog sloja budu jednake ulaznim, obično se nadopunjava nulama.

Mape značajki (slike provučene kroz filtare) dobijemo tako da konvolucijski slojevi uzimaju mapu na ulazu konvolucijskog sloja te rade konvoluciju s filtarom. Veličina mape značajki u određenom sloju dana je izrazom:

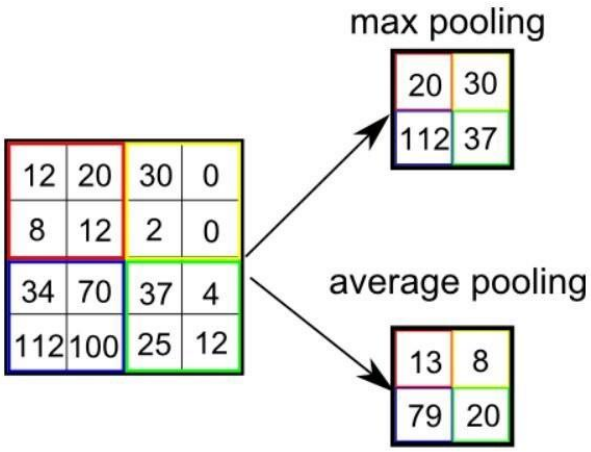
$$M^i = \frac{M^{i-1} - K^i}{S} + 1 \tag{3.1}$$

gdje M^i predstavlja dimenziju mape i-tog sloja, K^i dimenzije filtara koji povezuju mape prethodnog sloja s mapama trenutnog sloja, S predstavlja korak pomaka filtara po širini i visini prilikom konvolucije. Ovdje smo radi jednostavnosti koristili kvadratne mape značajki. Konvolucija se tvori prolazom prozora veličine jednake dimenziji filtara kroz ulaznu mapu te se množe vrijednosti filtara s preklopljenim vrijednostima ulazne mape. Dobiveni umnošci se sumiraju, dodaje se prag, računa izlaz aktivacijske funkcije te se rezultat zapisuje u određenu izlaznu mapu značajki. Dobivene mape značajki se slažu u dubine te prosljeđuju idućem sloju nizu. Dubina odgovara broju filtara u konvolucijskom sloju. Obično se nakon konvolucijskog sloja mape značajki provode kroz ReLU aktivacijsku funkciju čime se uvodi nelinearnost u mrežu.



Slika 3.2: Konvolucija ulazne mape značajki. (Izvor: stanford.edu)

Sloj sažimanja nije ključan sloj konvolucijske neuronske mreže, ali se često koristi kako bi smanjio broj parametara i operacija u mreži da bi se mreža mogla trenirati u realnom vremenu te time se štedi memorija. Ovaj sloj smanjuje rezoluciju ulaznih mapa tako da prozor veličine h x w ulazne mape mapira u jednu vrijednost. Najčešće se koristi prozor veličine 2 x 2 te tako širinu i visinu ulazne mape smanjimo na pola. Dva uglavnom korištena načina dobivanja vrijednosti iz prozora su preko maksimalne (eng. max pooling) ili srednje vrijednost (eng. average pooling) u prostoru sažimanja. Slojevi sažimanja ne smanjuju broj mapa i ne posjeduju nikakve težine koje je potrebno učiti.



Slika 3.3: Sloj sažimanja. (Izvor: quora.com)

Potpuno povezani sloj je ključan sloj u klasifikaciji ulazne slike jer upravo u ovom sloju možemo dobiti raspodjelu vjerojatnosti po klasama. Način na koji funkcionira je da određuje koje značajke iz prethodnog sloja najviše utječu na određenu klasu. Npr. ako program predviđa da je dana slika pas, imat će veliku vrijednost aktivacije značajki koje predstavljaju uši, oči i slično. Ovaj sloj daje potencijal za učenje prostornog rasporeda konvolucijskih značajki. Sloj je dobio ime po tome što je svaki neuron spojen sa svakim iz prethodnog sloja. Potpuno povezani sloj je jednodimenzionalan što znači da težine možemo prikazati vektorom. Računanje aktivacijske vrijednost se računa kao kod neuronskih mreža, množenjem ulaza s pripadnim težinama uz dodavanje konstante. Obično se izlazi zadnjeg sloja koji ima broj neurona jednak broju klasa u mreži šalje u softmax funkciju koja vraća distribuciju vjerojatnosti, tj. vidimo kolika je šansa da je upravo ta klasa na ulaznoj slici.

4. Programska podrška

4.1. Programski jezik Python

Python je interpretirani programski jezik visoke razine opće namjene. Objavljen je 1991. godine od strane Guida van Rossuma. Python naglašava čitljivost koda korištenjem razmaka te uvlačenja teksta. Njegovi jezični konstrukti te objektno-orijentirani pristup pomažu programerima da pišu jasan, logičan kod za projekte svih veličina. To je jezik koji sadrži dinamičko tipiziranje podataka te automatsko upravljanje memorijom. Od programskih paradigmi podržava proceduralnu, objektno orijentiranu, funkcionalnu i imperativnu. Python je jezik koji je interpretiran te stoga nema visoku efikasnost koda. Velika prednost pythona u odnosu na ostale programske jezike kod umjetne inteligencije općenito je što posjeduje veliki broj standardnih te neovisno razvijenih biblioteka te time omogućuje brzu i efikasnu implementaciju željenih rješenja. Korištena verzija za implementaciju programskog rješenja je Python 3.7.

4.2. Programska biblioteka PyTorch

PyTorch je programska biblioteka otvorenog koda (eng. open source code) namijenjena pretežno za implementiranje rješenja u strojnom učenju u pythonu. Vrlo mlada i nova biblioteka kreirana od strane istraživačke skupine umjetne inteligencije u Facebooku. PyTorch je zasnovan na Torch-u, paketu za strojno učenje otvorenog koda koji se temelji na programskom jeziku Lua. Pruža dvije ključne značajke visoke razine:

1. Tenzorsko računanje (matrično računanje). Ponaša se slično kao biblioteka numpy, ali omogućuje izvršavanje procesiranja operacija nad matricama na grafičkih procesorskih jedinica (GPU) koje su specijalizirane za izračunavanja jer imaju puno više jezgri od procesora te mogu paralelno izvoditi više operacija.

2. Duboke neuronske mreže sa mogućnošću automatskog praćenja gradijenta (Autograd) te tako omogućuje jednostavnu implementaciju algoritma propagacije pogreške unatrag

Verzija korištena za implementaciju rješenje je PyTorch 1.1.

```
1 import torch
2 a= torch.tensor([[1,2,3],[4,5,6]], requires_grad=True, dtype=torch.float)
3 b= a +2
4 c=2*b*b
5 out=c.mean()
6 out.backward()
7 print(a.grad)
```

```
tensor([[ 2.0000,  2.6667,  3.3333],
        [ 4.0000,  4.6667,  5.3333]])
```

5. Vizualizacija i interpretacija konvolucijskih modela

Područje računalnog vida sve brže napreduje otkako se koriste konvolucijske neuronske mreže. Brzina istraživanja u kombinaciji s ogromnom količinom slikovnih podataka na webu dala nam je nevjerojatne rezultate u posljednjih nekoliko godina. Duboki konvolucijski modeli su glavni sastojak mnogih praktičnih primjena računalnog vida, od prepoznavanja lica do automatizirane vožnje automobila. Međutim, često se javlja kritika da takvi modeli nisu spremni za industrijsku upotrebu zbog loše interpretabilnosti, odnosno, nemogućnosti modela da svoju odluku obrazloži ljudima. Zbog toga postupci za interpretiranje odluka i vizualiziranje naučenih parametara konvolucijskih modela predstavljaju zanimljivo područje istraživanja. Unatoč širokoj dostupnosti podataka, ponekad postaje vrlo teško razumjeti što i kako model uči. Vizualiiziranje i interpretiranje konvolucijskih modela nam može puno reći o tome gdje tražiti te kako ispraviti greške u modelu da bi u budućnosti dobili bolje rezultate. U ovom radu je odrađeno nekoliko jednostavnijih načina vizualizacije konvolucijske neuronske mreže:

- Optimalna pobuda razreda (eng. class model visualization)
- mapa značajnosti (eng. saliency map)
- vođeni backprop (eng. guided backpropagation)
- razredne aktivacijske mape (eng. class activation maps)
- optimalna pobuda latentnih aktivacija (eng. layer visualization)
- deep dream

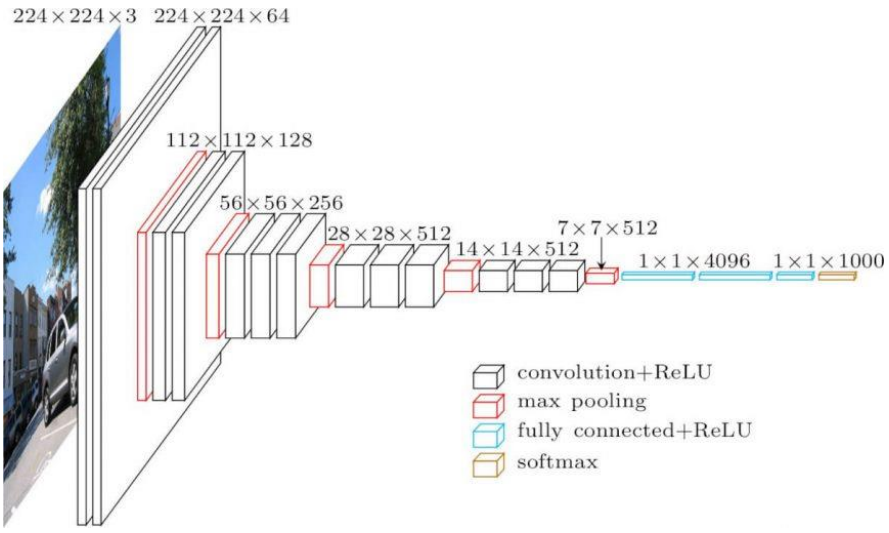
6. Implementacija i rezultati

6.1. Arhitektura mreže

Za model konvolucijske neuronske mreže koristi se VGG16. To je tip vrlo duboke konvolucijske neuronske mreže koja postiže rezultate od 92.7% top-5 točnost u klasifikaciju slika na ImageNetu. ImageNet je skup od 1.2 milijuna slika za treniranje koje pripadaju u 1000 klasa. Detaljna arhitektura VGG16:

1. ulazni sloj – slika dimenzija 224 x 224 x 3 – 224x224 piksela veličina slike te 3 kanala boje(RGB vrijednosti)
2. konvolucijski sloj – filtar 3x3, 3 ulazna kanala, 64 izlazna kanala što znači da imamo 64 filtara u ovom konvolucijskom sloju, pomak od 1 te nadopunjavanjem od 1. Izlaz prolazi kroz ReLU aktivacijsku funkciju
3. Konvolucijski sloj – filtar 3x3, 64 ulazna kanala, 64 izlaznih kanala, pomak od 1 te nadopunjavanje od 1. Izlaz prolazi kroz ReLU
4. Sloj sažimanja maksimalnom vrijednosti – dimenzije 2x2
5. Konvolucijski sloj – filtar 3x3, 64 ulazna kanala, 128 izlaznih kanala, pomak od 1 te nadopunjavanje od 1. Izlaz prolazi kroz ReLU
6. Konvolucijski sloj – filtar 3x3, 128 ulazna kanala, 128 izlaznih kanala, pomak od 1 te nadopunjavanje od 1. Izlaz prolazi kroz ReLU
7. Sloj sažimanja maksimalnom vrijednosti – dimenzije 2x2
8. Konvolucijski sloj – filtar 3x3, 128 ulazna kanala, 256 izlaznih kanala, pomak od 1 te nadopunjavanje od 1. Izlaz prolazi kroz ReLU
9. Konvolucijski sloj – filtar 3x3, 256 ulazna kanala, 256 izlaznih kanala, pomak od 1 te nadopunjavanje od 1. Izlaz prolazi kroz ReLU
10. Konvolucijski sloj – filtar 3x3, 256 ulazna kanala, 256 izlaznih kanala, pomak od 1 te nadopunjavanje od 1. Izlaz prolazi kroz ReLU
11. Sloj sažimanja maksimalnom vrijednosti – dimenzije 2x2
12. Konvolucijski sloj – filtar 3x3, 256 ulazna kanala, 512 izlaznih kanala, pomak od 1 te nadopunjavanje od 1. Izlaz prolazi kroz ReLU
13. Konvolucijski sloj – filtar 3x3, 512 ulazna kanala, 512 izlaznih kanala, pomak od 1 te nadopunjavanje od 1. Izlaz prolazi kroz ReLU

14. Konvolucijski sloj – filtar 3x3, 512 ulazna kanala, 512 izlaznih kanala, pomak od 1 te nadopunjavanje od 1. Izlaz prolazi kroz ReLU
15. Sloj sažimanja maksimalnom vrijednosti – dimenzije 2x2
16. Konvolucijski sloj – filtar 3x3, 512 ulazna kanala, 512 izlaznih kanala, pomak od 1 te nadopunjavanje od 1. Izlaz prolazi kroz ReLU
17. Konvolucijski sloj – filtar 3x3, 512 ulazna kanala, 512 izlaznih kanala, pomak od 1 te nadopunjavanje od 1. Izlaz prolazi kroz ReLU
18. Konvolucijski sloj – filtar 3x3, 512 ulazna kanala, 512 izlaznih kanala, pomak od 1 te nadopunjavanje od 1. Izlaz prolazi kroz ReLU
19. Sloj sažimanja maksimalnom vrijednosti – dimenzije 2x2
20. Sloj sažimanja srednjom vrijednost – izlaz za svaku jedinku ulaznog kanala 7x7. Izlaz vektor od 512x7x7 vrijednosti.
21. Potpuno povezani sloj – ulaz 25088 vrijednosti, izlaz 4096. Izlaz prolazi kroz ReLU aktivacijsku funkciju.
22. Sloj ignoriranja (eng. dropout layer) – regularizacijska tehnika koja sprječava prilagodljivost mreže na set za treniranje, djeluje tako da ignorira slučajno odabrane neurone. U ovom slučaju ignorira se neuron sa 50% šanse(p=0.5)
23. Potpuno povezani sloj – ulaz 4096 vrijednosti, izlaz 4096. Izlaz prolazi kroz ReLU aktivacijsku funkciju.
24. Sloj ignoriranja (eng. dropout layer) – ignorira neuron uz 50% šanse
25. Potpuno povezani sloj – ulaz 4096 vrijednost, izlaz 1000 vrijednosti koje predstavljaju aktivaciju za svaku klasu



Slika 6.1: Arhitektura konvolucijske neuronske mreže VGG16. (Izvor: neurohive.io)

Dvije najveće mane ove VGG16 konvolucijske neuronske mreže su:

1. treniranje mreže je jako sporo zbog velikog broja težina i velike dubine mreže
2. veliko memorijsko zauzeće zbog velikog broja težina u mreži

U ovom radu koristimo već treniranu mrežu te samo učitamo težine zajedno sa arhitekturom mreže iz torchvision paketa koji dolazi sa pytorchom.

```
self.model = models.vgg16(pretrained=True).to(self.device).eval()
```

Metoda to(self.device) definira na kojoj procesorskoj jedinici se obavljaju računanja u mreži. Definicija procesorske jedinice:

```
self.device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
```

6.2. Učitavanje te procesiranje ulaznih slika

Prvi korak nakon što učitamo arhitekturu mreže te njezine težine je učitati slike s računala ili koristeći neki udaljeni poslužitelj. U ovoj implementaciji se koristi dohvaćanje slike preko adrese slike s interneta.

```
image = Image.open(io.BytesIO(requests.get(image_url).content))
```

Requests je modul koji koristimo da dohvatimo podatke s udaljenog poslužitelja, Image je modul koji sadrži mnoštvo komponenti za rad sa slikama, io je glavni modul za rad sa tokovima podataka.

Nakon toga potrebno je transformirati ulaznu sliku u format ulaza mreže, tj. u tenzor dimenzija [3, 224, 224]. Kreiramo Variable klasu koja prima dobiveni tenzor, te tu klasu koristimo za ulaz u VGG16.

```
preprocess = transforms.Compose([
    transforms.Resize(224),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(
        mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225])
    ])
image = preprocess(image)
image = Variable(image.unsqueeze(0)).to(self.device)
```

Paket torchvision sadrži komponente pomoću kojih jednostavno možemo obrađivati ulazne slike. Tvornica transforms.Compose stvara funkciju koja provodi transformacije nad slikom u zadanom redoslijedu. Nakon što sliku transformiramo u dimenzije ulaza mreže te je pretvorimo u tenzor, trebamo normalizirati podatke na onaj način koji mreža zahtijeva, to zadajemo preko transforms.Normalize. Pomoću normaliziranja dobijemo vrijednosti piksela slike koje su u željenom rasponu preko sljedeće formule:

$$image = \frac{(\,image - mean\,)}{std} \tag{5.1}$$

Nakon što obradimo sliku, potrebno ju je vratiti u nenormalizirani oblik te pretvoriti u oblik koji možemo poslati biblioteci za prikazivanje slika.

```
image = image.cpu().detach().numpy().squeeze(0)
image = image.transpose((1, 2, 0))
mean = np.array([0.485, 0.456, 0.406])
std = np.array([0.229, 0.224, 0.225])
original = std * image + mean
```

U prvom koraku, cpu, prebacujemo računanja slike na procesor jer nije moguće daljnje radnje izvoditi na GPU, nakon toga zovemo detach što znači ako tenzor slike pamti gradijent prestaje ga pratiti. Transpose mijenja dimenzije slike sa [3, 224, 224] na [224, 224, 3] što je bitno kod prikazivanja slika. U ostalim koracima vraćamo sliku u nenormalizirani oblik. Za prikazivanje slika koristim matplotlib biblioteku koja prikazuje slike unutar numpy matrice s dimenzijama [224, 224, 3] za RGB slike, te [224, 224] za crno-bijele slike.

6.3. Izvedba vizualizacije konvolucijske neuronske mreže

6.3.1. Optimalna pobuda razreda (eng. class model visualization)

U ovom dijelu ćemo implementirati te pokazati rezultate optimalne pobude razreda. Cilj nam je za određenu klasu koju zadamo vidjeti kako konvolucijska neuronska mreža zapravo „vidi“ tu klasu. Ovom metodom mreža zapravo generira sliku iz ničega. Na ulaz predajemo generiranu sliku slučajnim vrijednostima koju dobijemo na sljedeći način:

```
image = np.uint8(np.random.uniform(150, 180, (1, 3, 224, 224)))/255
image = torch.tensor(image, dtype=torch.float32)
image = image.to(self.device)
image = Variable(image)
```

Optimalna pobuda razreda funkcionira tako da izračunamo aktivaciju mreže za danu klasu u izlaznom sloju te ažuriramo piksele tako da u idućoj iteraciji aktivacija za istu klasu u izlaznom sloju bude veća. To možemo učiniti tako da ažuriramo one piksele proporcionalno onome koliko su utjecali na aktivaciju klase za koju određujemo izlaznu sliku.

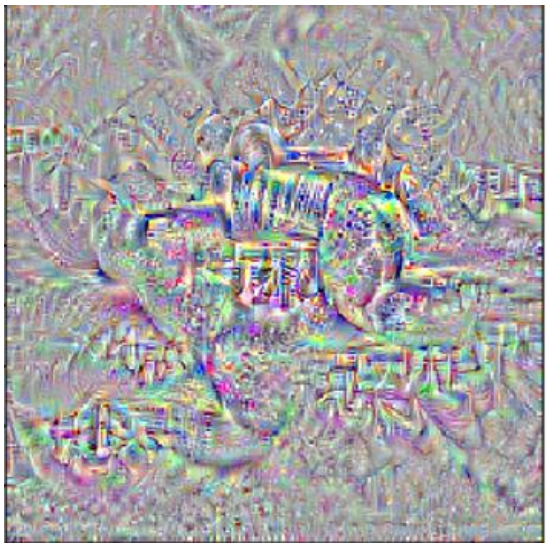
```
image = self.generateNoiseImage()
image.requires_grad = True

learning_rate = 0.05
optimizer = optim.SGD([image], lr=learning_rate, momentum=0.9)

iterations = 81
for i in range(iterations):
    optimizer.zero_grad()
    self.model.zero_grad()
    prediction = self.model(image)
    loss = -prediction[0][labelIndex]
    loss.backward()
    optimizer.step()
return image
```

Requires_grad služi da omogućimo praćenje gradijenta u odnosu na ulaznu sliku. To nam je važno da bi mogli izračunati koliko se zapravo mijenja aktivacija klase u odnosu na određeni piksel na ulaznoj slici. Torch.optim je paket koji sadrži razne implementacije optimizacijskih algoritama, algoritama pomoću kojih možemo

jednostavno ažurirati vrijednosti u odnosu na izračunati gradijent te dane parametre. Zero_grad moramo zvati nakon svake iteracije da postavimo gradijente na 0 nakon čega možemo ponovno računati gradijente. Kao funkciju gubitka koristimo negativnu aktivaciju klase zbog toga što kada minimiziramo gubitak povećavam istovremeno aktivaciju razreda, te će optimizacijska funkcija ažurirati piksele tako da smanji gubitak što znači povećanje aktivacije u idućoj iteraciji. Optimizer.step vrši ažuriranje težina na principu već poznate formule:

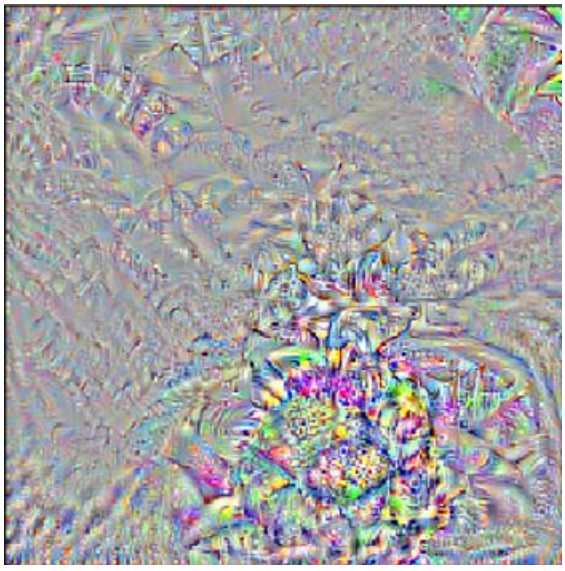
$$w_n = w_n - \alpha \cdot \frac{dL}{dw_n} \tag{5.2}$$

U nastavku ćemo prikazati nekoliko rezultata algoritma.



Slika 6.2, 6.3: Optimalna pobuda razreda, utezi za vježbanje, 2 pokretanja.

Slika 6.4: Optimalna pobuda razreda, jagoda .



Slika 6.5: Optimalna pobuda razreda, ananas.

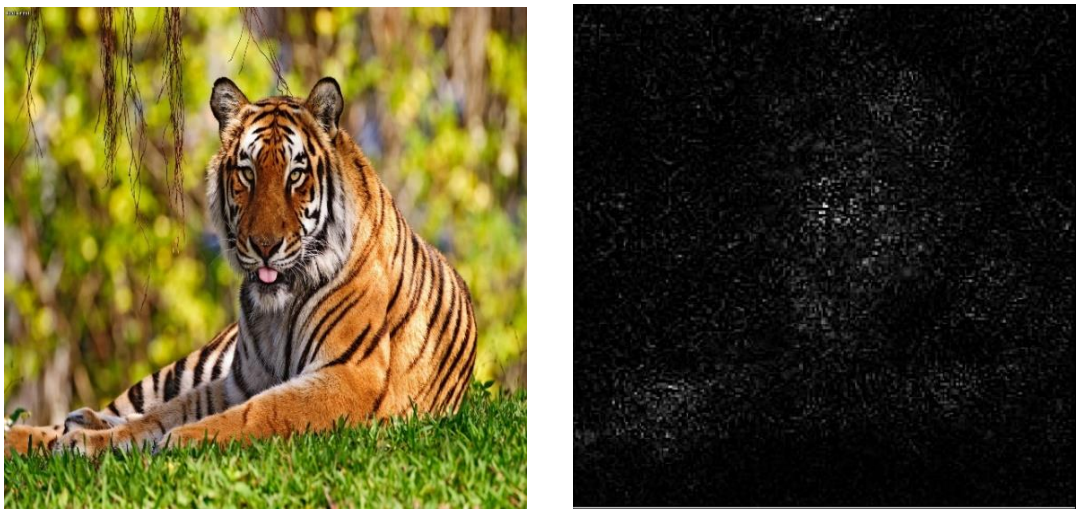
6.3.2. Mapa značajnosti (eng. saliency map)

Pretpostavimo da sve slike za vježbanje razreda točak sadrže u sebi također sliku vozila. Kako možemo znati da li CNN koristi piksele povezane s točkom ili piksele povezane s vozilom za klasifikaciju? Kod malih setova za vježbanje ovo je veliki problem te bi htjeli znati koje piksele mreža gleda kod klasifikacije. Ovom metodom možemo saznati upravo to. Ideja je prilično jednostavna. Izračunamo gradijent aktivacije izlazne klase u odnosu na ulaznu sliku. To nam zapravo kaže kako se mijenja vrijednost aktivacije izlazne klase u odnosu na malu promjenu piksela ulazne slike. Sve pozitivne vrijednosti u gradijentima govore nam da će mala promjena piksela povećati izlaznu vrijednost aktivacije klase. Stoga ideja algoritma je da se prikažu pozitivne vrijednosti gradijenta kao pikseli u slici, te tu sliku prikažemo.

```
image = self.getAndPreprocessImage(image_url)
image.requires_grad = True
self.model.zero_grad()

prediction = self.model(image)
percentages = F.softmax(prediction, dim=1)[0]
topres = torch.topk(prediction, n)
index = topres[1].cpu().detach().numpy()[0][n-1]

prediction[0][index].backward()
data = image.grad.data
data = self.convertToGrayByMax(data.squeeze(dim=0))
saliencyMap = data.clamp(min=0)
return saliencyMap
```

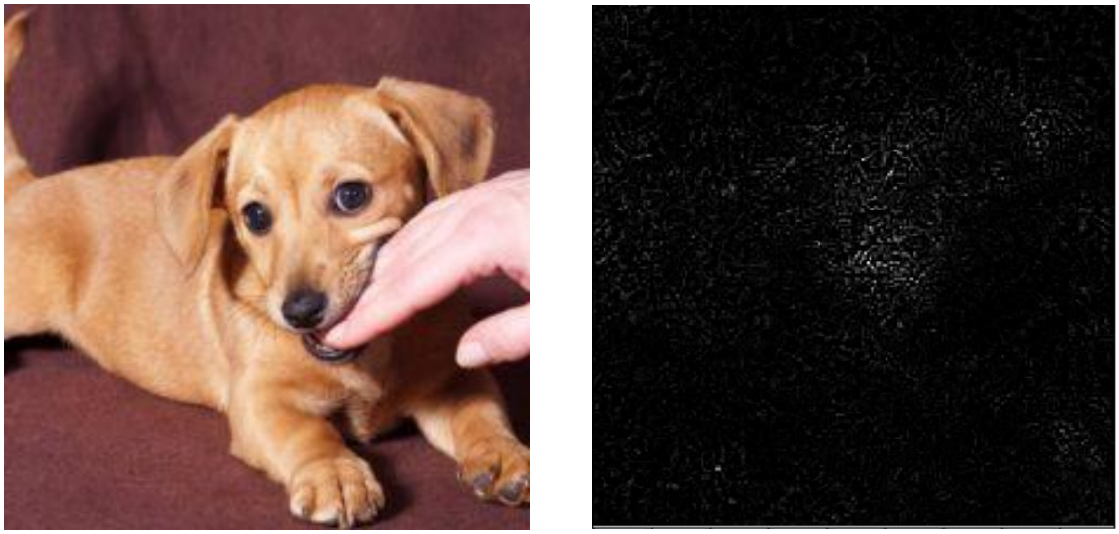
Postotke koliko je konvolucijska neuronska mreža sigurna da je to upravo taj objekt dobijemo pomoću funkcije softmax koja skalira ukupan izlaz da suma bude 1. Funkcija topk vraća aktivaciju te indekse n najboljih rezultata. N je broj koji predajemo funkciji i kaže koji po redu najbolji razred želimo vizualizirati. Image.grad.data je upravo tenzor koji predstavlja gradijent izlaza prema ulaznoj slici. Pomoću parametra n kojeg šaljemo kao parametar funkciji određujemo za koji najbolji rezultat po redu želimo prikazati mapu značajnosti. ConvertToGrayByMax prima RGB sliku te vraća crno-bijelu sliku koja je dobivena tako da uzmemo maksimalnu vrijednost piksela između tri boje. Funkcija clamp je funkcija koja u

ovom slučaju sve vrijednosti koje su manje od 0 zamijeni s 0. U nastavku ćemo prikazati nekoliko rezultata algoritma.



Slika 6.6: Mapa značajnosti, tigar.

Slika 6.7: Mapa značajnosti, leptir.



Slika 6.8: Mapa značajnosti, pas.

6.3.3. Vođeni backprop (eng. guided backpropagation)

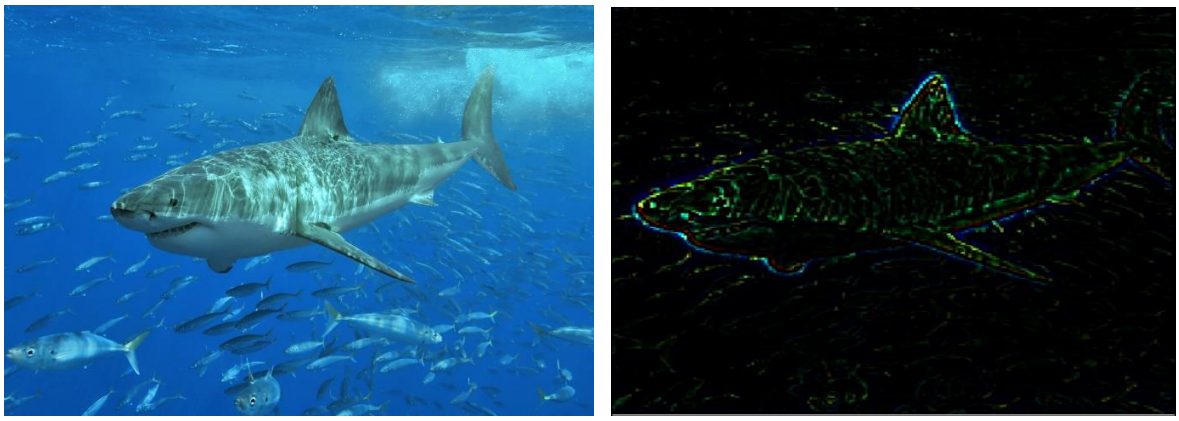
Algoritam se temelji na ideji da nas zanimaju samo uzorci na slici koje je mreža detektirala, te želimo sve gradijente koji negativno utječu na aktivaciju izlaznog sloja izjednačiti s 0. Algoritam zovemo vođen backprop zbog toga što ažuriramo, mijenjamo gradijente prije izlaza iz svakog sloja. Na ovaj način na ulazu, tj. gradijentu ulazne slike dobijemo samo one vrijednost koje pozitivno utječu na promjenu izlaznog sloja. Algoritam također zahtijeva da zadržimo samo one gradijente koji su imali pozitivnu aktivaciju u prolasku kroz mrežu u naprijed, ali se ne trebamo brinuti zbog toga što mreža koristi ReLU koji će automatski odraditi ovaj posao za nas. Način na koji možemo mijenjati gradijente kada prolaze kroz mrežu je preko ugrađenih funkcija register_backward_hook kojoj predajemo funkciju koja će obraditi te gradijente. Zakačku registriramo na module koji zapravo predstavljaju jedan sloj u mreži te tako pratimo gradijent kroz taj sloj.

```
#self.fmaps = []
#def forward_hook_fn(module, input, output):
    #self.fmaps.append(output)

def backward_hook_fn(module, grad_out, grad_in):
    new_grad_out = grad_out[0].clamp(min=0)
    #fgrad = self.fmaps[-1]
    #fgrad[fgrad > 0] = 1
    #new_grad_out = new_grad_out * fgrad
    #del self.fmaps[-1]
    return (new_grad_out,)
```

Komentirani dio programa je potrebno koristiti u slučaju da naša mreža ne koristi ReLU koji nam automatski sve gradijente negativnih aktivacija izjednači s 0. Dio algoritma koji računa izlaznu sliku:

```
modules = list(self.model.features._modules.items())
hooks = []
for name, module in modules:
    if isinstance(module, nn.ReLU):
        #hooks.append(module.register_forward_hook(forward_hook_fn))
        hooks.append(module.register_backward_hook(backward_hook_fn))

image = self.getAndPreprocessImage(image_url)
image.requires_grad = True
self.model.zero_grad()
output = self.model(image)
index = output.argmax().cpu().item()
output[0][index].backward()

for hook in hooks:
    hook.remove()
return image.grad.data
```

Modules je lista svih slojeva u mreži. U prvom koraku dodajemo zakačku na sve konvolucijske slojeve koja prati povratni gradijent te sve vrijednosti manje od 0 postavi na 0 te proslijedi dalje. Na prvi sloj postavljamo zakačku da možemo postaviti dobivene gradijente te iste vratiti nazad kao sliku. Na kraju uklanjamo zakačke te vraćamo sliku koja predstavlja gradijent aktivacije s obzirom na ulaznu sliku dobiven prethodno opisanim postupkom. U nastavku ćemo prikazati nekoliko primjera rada ovog algoritma.



Slika 6.9: Vođeni backprop, bijela morska psina.

Slika 6.10: Vođeni backprop, tigar.

Slika 6.11: Vođeni backprop, sportski auto.

6.3.4. Razredne aktivacijske mape (eng. class activation maps or CAM)

CAM je jednostavan algoritam pomoću kojega dobijemo regije na slici koje pozitivno doprinose klasifikaciji slike u zadani razred. CAM se također može koristiti da bi dobili lokalizaciju objekta na slici. Kako bi se stvorio CAM, mrežna arhitektura je ograničena da ima globalni sloj sažimanja srednjom vrijednošću nakon zadnjeg konvolucijskog sloja prije potpuno povezanog sloja. To znači da ne možemo primijeniti ovu tehniku na postojeće mreže koje nemaju tu strukturu. Ono što možemo učiniti je modificirati postojeće mreže i podesiti ih da omogućimo provođenje algoritma. U korištenoj mreži VGG16, posljednji konvolucijski sloj ima 512 filtara, a kako znamo što dublje u mrežu idemo, možemo pomoću filtara

detektirati kompleksnije uzorke, kao na primjer uši, oči, itd. Izlaz posljednjeg konvolucijskog sloja ima dimenziju [512, 14, 14]. Algoritam funkcionira na sljedeći način:

1. Nađemo aktivaciju posljednjeg konvolucijskog sloja, te dobijemo gradijent predviđene klase prema posljednjem konvolucijskom sloju, tj. koliko je koja težina u svakom filtaru utjecala na donošenje odluke.

```
self.activations = None
self.gradients = None
def hook_fn(module, input, output):
    self.activations = output
def bcw_hook(module, grad_out, grad_in):
    self.gradients = grad_in[0]
last_conv = self.model._modules['features'][29]
hook_fw = last_conv.register_forward_hook(hook_fn)
hook_bck = last_conv.register_backward_hook(bcw_hook)

image = self.getAndPreprocessImage(image_url)
output = self.model(image)
topres = torch.topk(output, n)
index = topres[1].cpu().detach().numpy()[0][n-1]

self.model.zero_grad()
output[:, index].backward()
```

2. Izračunamo srednju vrijednost gradijenta za svaki od filtara, tj. koliko je prosječno svaki filtar utjecao na donošenje odluke da je upravo to ta klasa na slici.

```
pooled_gradients = torch.mean(self.gradients, dim=[0, 2, 3])
```

3. Pomnožimo aktivaciju svakog filtara s prosječnom vrijednosti koliko je filtar utjecao na klasifikaciju. Važno je da množimo sa sažetim gradijentima umjesto s originalnim zbog toga što želimo naći ključne značajke na slici, a ne ključne aktivacije unutar određene značajke.

```
for j in range(512):
    self.activations[:, j, :, :] *= pooled_gradients[j]
```

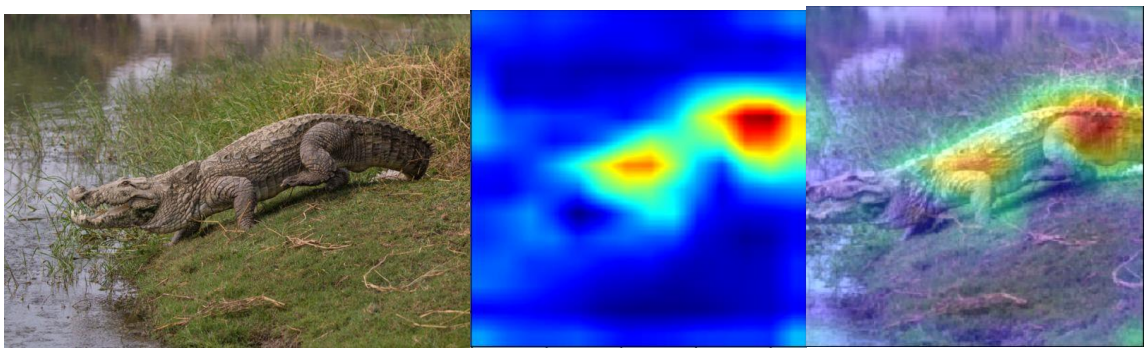
4. Nađemo srednju vrijednost koliko je koji neuron u filtarima imao utjecaja na izlaz klasifikacije tako da nađemo prosječnu vrijednost od 512 filtara za svaki od 14x14 neurona

```
heatmap = torch.mean(self.activations, dim=1).squeeze()
```

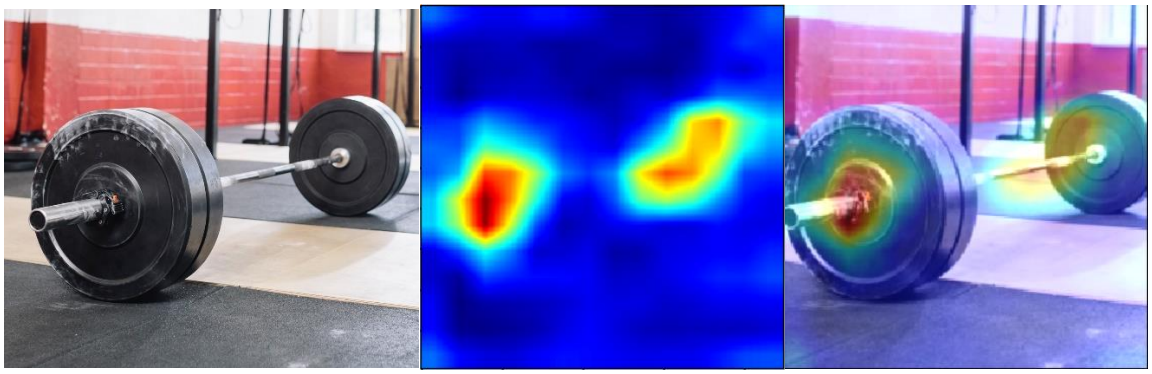
5. Nakon što smo dobili toplinsku kartu dimenzija 14x14 trebamo je još skalirati na veličinu ulazne slike te ih spojiti na sljedeći način:

```
new_img = heatmap*0.4 + imageAlt
```

U nastavku su prikazani rezultati primjene algoritma.



Slika 6.12: Razredne aktivacijske mape, krokodil.



Slika 6.13: Razredne aktivacijske mape, tegovi za vježbanje.

6.3.5. Optimalna pobuda latentnih aktivacija (eng. layer visualization)

Optimalna pobuda latentnih aktivacija je algoritam kojim možemo vizualizirati određene filtare unutar određenog sloja ili cijeli sloj skupa tako da odredimo srednju vrijednost svih filtara unutar sloja. Ideja je vrlo slična kao kod pronalaženja optimalne pobude razreda kod prve spomenute metode vizualizacije. Optimalnom pobudom latentnih aktivacija možemo primijetiti kako su filtari naučeni te koliko komplicirane uzorke mogu detektirati. Što mreža ide prema većoj dubini možemo primijetiti kompliciranije uzorke, dok su na početku samo određene boje i jednostavniji oblici poput linija, točki, itd. Cilj ovog načina vizualizacije je zapravo dobiti ulaznu sliku

koja što više aktivira pobudu latentnih aktivacija kojeg vizualiziramo. Algoritam možemo opisati u nekoliko koraka:

1. prvo odredimo mrežu kojoj je dani sloj koji želim vizualizirati zadnji sloj u mreži ili registriramo zakačku na taj sloj tako da možemo zapamtiti njegovu aktivaciju. Nakon toga inicijaliziramo sliku sa slučajno odabranim vrijednostima.

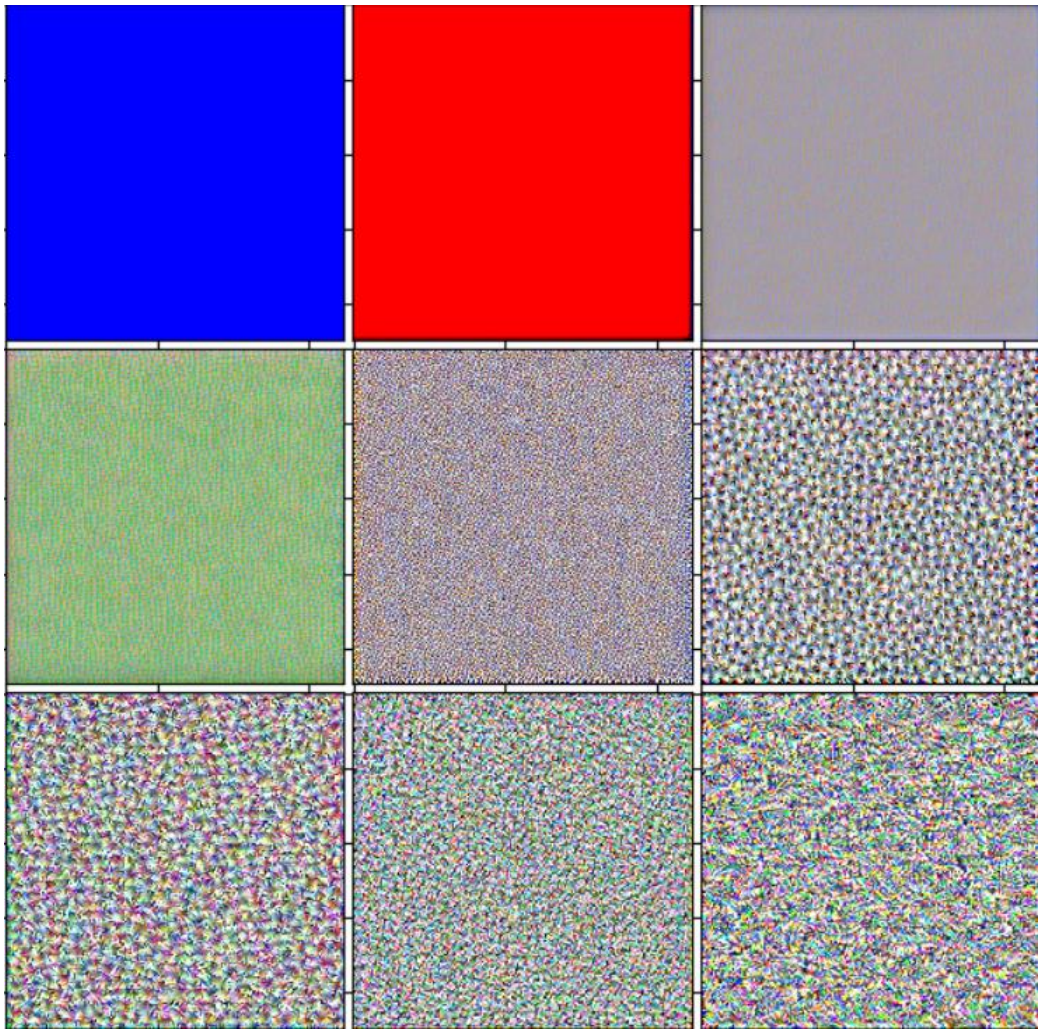
```
subnet = self.model._modules['features'][0:layer+1]
image = self.generateNoiseImage()
image.requires_grad = True;
```

2. Kroz svaku iteraciju računamo pobudu latentne aktivacije koju vizualiziramo za danu sliku sa slučajno odabranim vrijednostima. Izračunamo gradijent srednje vrijednosti(aktivacije) filtara kojeg vizualiziramo u odnosu na ulaznu sliku te je ažuriramo preko izračunatih vrijednosti. Ako vizualiziramo optimalne latentne aktivacije za cijeli sloj, samo nađemo srednju vrijednost(aktivaciju) svih filtara i gledamo gradijent dobivene vrijednosti.

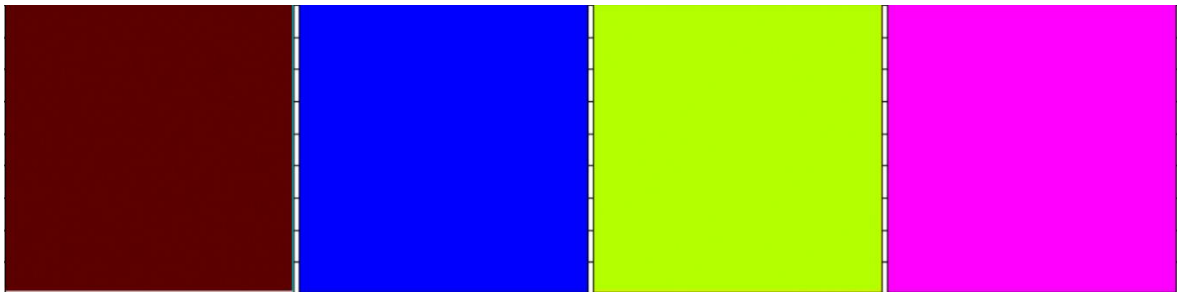
```
optimizer = optim.Adam([image], lr=lr, weight_decay=1e-6)
for i in range(iterations):
    optimizer.zero_grad()
    self.model.zero_grad()
    output = subnet(image)
    output = output[0][filter]
    #output = output[0]
    loss = -torch.mean(output)
    loss.backward()
    optimizer.step()
```

Dio koda u komentarima je način vizualizacije optimalne pobude aktivacija cijelog sloja umjesto jednog filtara.

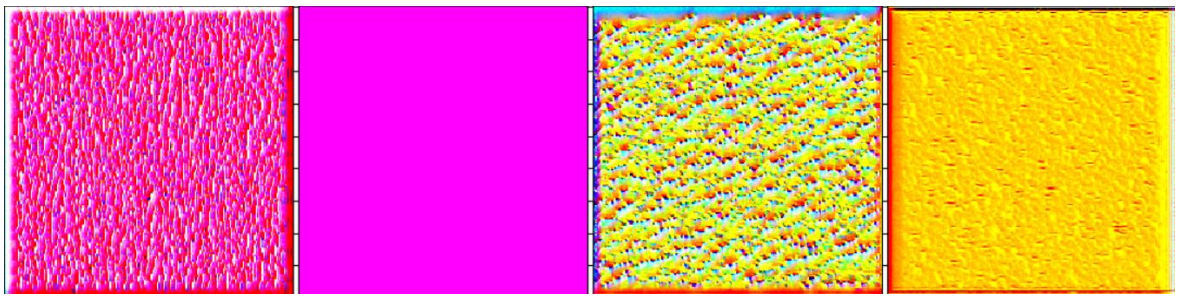
U nastavku ćemo pokazati rezultate algoritma, za vizualizaciju latentnih aktivacija cijelog sloja gledamo prvih 9 konvolucijskih slojeva, dok za vizualizaciju latentnih aktivacija filtara gledamo slučajno odabrana 4 filtara iz nekoliko konvolucijskih slojeva:



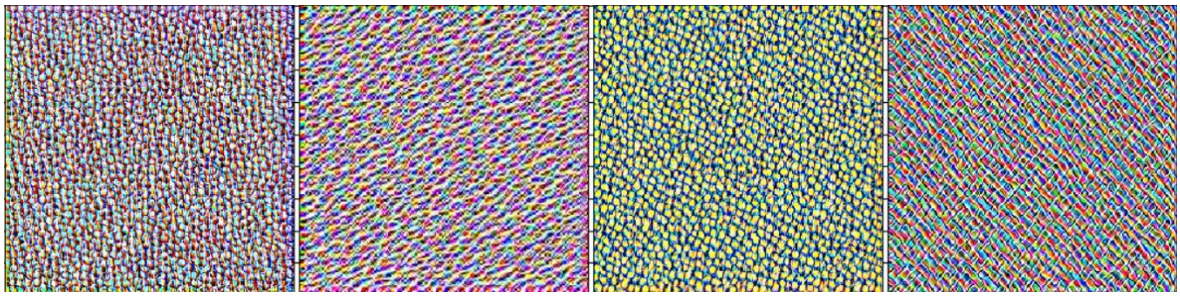
Slika 6.14: Vizualizacija optimalnih latentnih aktivacija konvolucijskih slojeva 1-9.



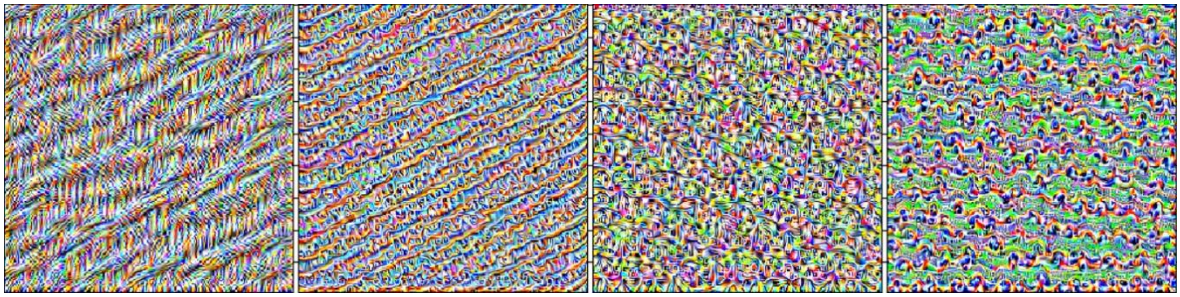
Slika 6.15: Vizualizacija optimalnih latentnih aktivacija filtara iz 1. konvolucijskog sloja.



Slika 6.16: Vizualizacija optimalnih latentnih aktivacija filtara iz 2. konvolucijskog sloja.



Slika 6.17: Vizualizacija optimalnih latentnih aktivacija filtara iz 5. konvolucijskog sloja.



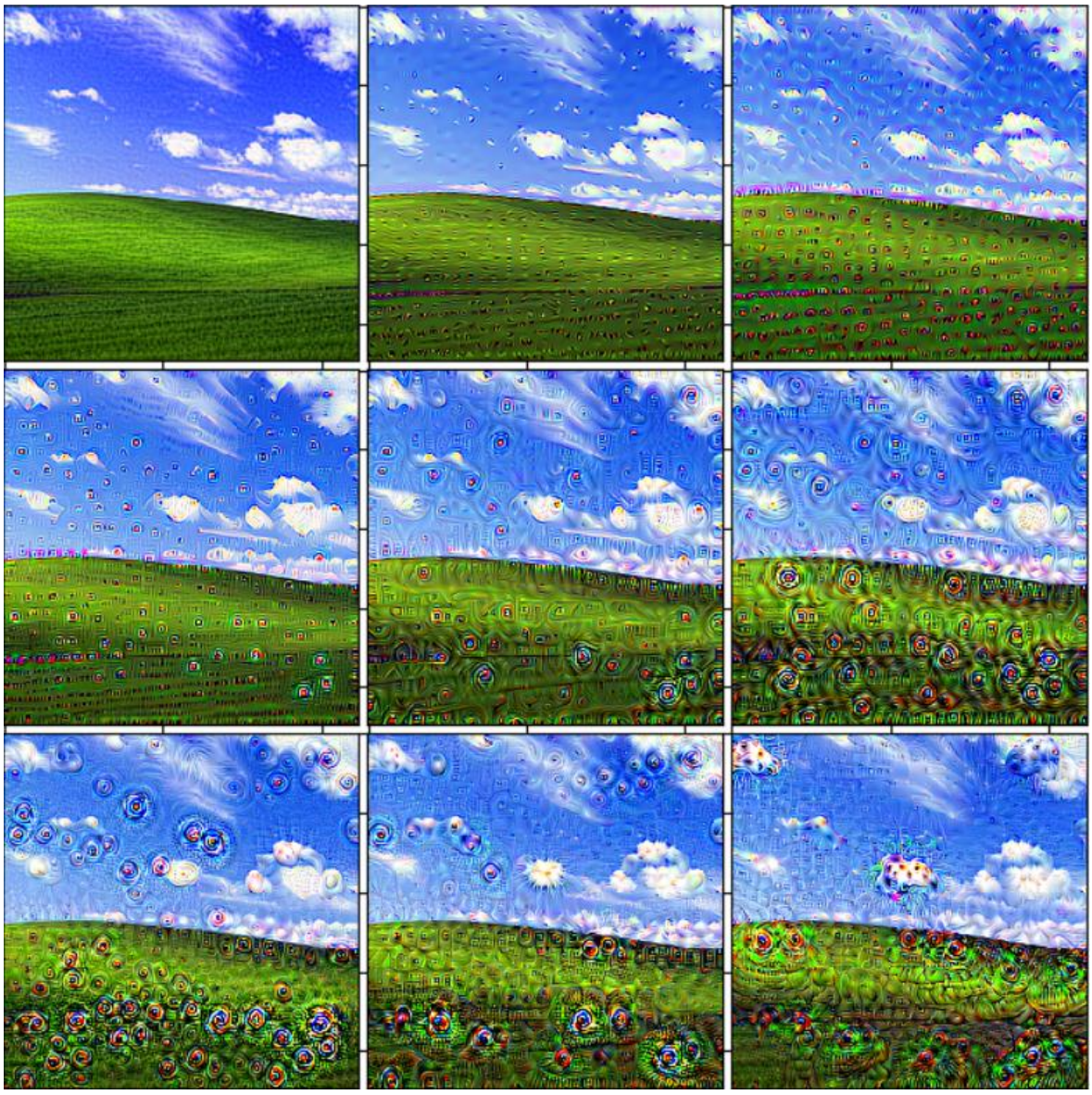
Slika 6.18: Vizualizacija optimalnih latentnih aktivacija filtara iz 8. konvolucijskog sloja.

6.3.6. Deep dream

Deep dream je program za računalni vid koji je osmislio Googleov inženjer Alex Mordvintsev. Program funkcionira tako da pronalazi i poboljšava uzorke u slici, stvarajući tako halucinogenu pojavu „kao u snu“ odakle i ime programa. To se dobije tako da ulaznu sliku pošaljemo kroz mrežu do sloja koji je zadužen za prepoznavanje uzoraka koje želimo dobiti na slici. Nakon što dobijemo aktivaciju željenog sloja računamo gradijent norme aktivacije po ulaznoj slici te ažuriramo piksele u slici kroz svaku iteraciju. Tako dobijemo da oni dijelovi slike koje mreža prepozna kao neki uzorak na kraju zapravo postanu taj uzorak. Možemo primijetiti sličnosti sa metodom optimalnom pobudom latentnih aktivacija po tome što iterativno modificiramo ulaznu sliku s ciljem maksimizacije izlaza, koji je kod metode optimalne pobude latentnih aktivacija aktivacija određenog filtara, dok je kod metode deep dream norma izlaza iz određenog sloja.

```
subnet = self.model._modules['features'][0:layer+1]
image = self.getAndPreprocessImage(image_url)
image.requires_grad = True

optimizer = optim.SGD([image], lr=lr,  weight_decay=1e-4)
for i in range(iterations):
    self.model.zero_grad()
    optimizer.zero_grad()
    output = subnet(image)
    loss = -output.norm()
    loss.backward()
    optimizer.step()
```

U nastavku je prikazano korištenje algoritma na konvolucijske slojeve 2 do 10.



Slika 6.19: Deep dream na konvolucijske slojeve 2-10.

7. Zaključak

Klasifikacija slika je problem računalnog vida s mnogim zanimljivim primjenama. Ponekad je teško shvatiti način na koji konvolucijska mreža dolazi do rezultata. U ovom radu smo se fokusirali na metode koje upravo rješavaju ovaj problem te nam daju uvid u to kako mreža funkcionira iznutra. Na početku smo pokazali što je to neuron, neuronska mreža te što su konvolucijske neuronske mreže koje predstavljaju poboljšanu verziju neuronske mreže specijalizirane za ulazne podatke koji imaju topološku strukturu. Upoznali smo se programskom podrškom koju smo koristili, konkretno bibliotekom otvorenog koda za python nazvanom PyTorch koja ima vrlo širok spektar funkcionalnosti pogodnih za implementiranje rješenja u strojno učenju. Na kraju smo prikazali neke od metoda pomoću kojih možemo vizualizirati funkcioniranje naučene konvolucijske neuronske mreže. Vizualizirali smo više načina na kojima dobijemo uvid u samu arhitekturu mreže kao na primjer optimalna pobuda razreda kojom maksimiziramo aktivaciju za određenu klasu te optimalna pobuda latentnih aktivacija kojom maksimiziramo aktivaciju određenog sloja. Prošli smo i nekoliko primjera u kojima vidimo kako mreža interpretira ulaznu sliku, koji dijelovi slike su ključni za dolaženje do samog rezultata klasifikacije.

Konvolucijske mreže su pokazale najbolje rezultate u primjeni te iz godine u godinu pokazuju sve bolje rezultate. Za očekivati je da će nadmašiti ljudsko oko jednog dana, a da bi to dostigli moramo shvatiti kako mreža uči, kako dolazi do željenih rezultata te to iskoristiti da bi dobili što bolje rezultate u budućnosti.

Literatura

[1] K. Simonyan, A. Vedaldi, A. Zisserman. Deep inside convolutional networks: visualising image classification models and saliency maps. Arxiv 2014
URL https://arxiv.org/pdf/1312.6034.pdf

[2] I. Sego. Klasifikacija slika kućnih brojeva dubokim konvolucijskim modelima. Zemris 2018
URL https://www.zemris.fer.hr/~ssegvic/project/pubs/sego18bs.pdf

[3] C. Olah, A. Merdvintsev, L. Schubert. Feature Visualization. Distill 2017
URL https://distill.pub/2017/feature-visualization/

[4] C. Olah, A. Satyanarayan, I. Johnson, S. Carter, L. Schubert, K. Ye, A. Merdvinstev. The building blocks of interpretability. Distill 2018
URL https://distill.pub/2018/building-blocks/

[5] Artificial neural network. Wikipedia
URL https://en.wikipedia.org/wiki/Artificial_neural_network

[6] Convolutional neural networks. CS231n
URL http://cs231n.github.io/convolutional-networks/

[7] N. Inkawhich. Adversarial example generation. PyTorch 2017
URL https://pytorch.org/tutorials/beginner/fgsm_tutorial.html

[8] A. Sharma. What are saliency maps in deep learning. Analytics India magazine 2018
URL https://www.analyticsindiamag.com/what-are-saliency-maps-in-deep-learning/

[9] S. Chilamkurthy. Dana loading and processing tutorial. PyTorch 2017
URL https://pytorch.org/tutorials/beginner/data_loading_tutorial.html

[10] NPTEL-NOC IITM. Deep learning(CS7015): Lec 12.5 Guided Backpropagation. Youtube 2018
URL https://www.youtube.com/watch?v=rcXOEzAuvu4

[11] Metalbubble. Class Activation Mapping. Github 2018
URL https://github.com/metalbubble/CAM

[12] J. Gildenblat. Class activation maps in Keras for visualizing where deep learning networks pay attention. Jacob's computer vision and machine larning blog 2016
URL https://jacobgil.github.io/deeplearning/class-activation-maps

[13] A. Paliwal. Understanding your convolution network with visualization. Towards dana science 2018
URL https://towardsdatascience.com/understanding-your-convolution-network-with-visualizations-a4883441533b

[14] N. Manwani. Deep dream with TensorFlow: a practical guide to build your first deep dream experience. Hackernoon 2018
URL https://hackernoon.com/deep-dream-with-tensorflow-a-practical-guide-to-build-your-first-deep-dream-experience-f91df601f479

Vizualizacija i interpretacija konvolucijskih modela

Sažetak

Duboki konvolucijski modeli su glavni sastojak mnogih praktičnih primjera računalnog vida. Međutim, često se javlja kritika da takvi modeli nisu spremni za industrijsku upotrebu zbog loše interpretabilnosti, odnosno, nemogućnost modela da svoju odluku obrazloži ljudima. Zbog toga postupci za interpretiranje odluka i vizualiziranje naučenih parametara konvolucijskih modela predstavljaju zanimljivo područje istraživanja.U okviru rada, odabran je okvir za automatsku diferencijaciju te upoznate biblioteke za rukovanje matricama i slikama. Ukratko su opisani postojeći postupci za interpretiranje i vizualiziranje konvolucijskih modela, njihovih naučenih filtara te načina na koji vide ulazne slike.

Ključne riječi: računalni vid, konvolucijske neuronske mreže, pytorch, strojno učenje, vizualizacija konvolucijske neuronske mreže

Visualization and interpretation of convolution models

Abstract

Deep convolution models are the main ingredient of many practical examples of computer vision. However, there is often criticism that such models are not ready for industrial use due to poor interpretability, that is, the inability of the model to explain its decision to people. Therefore, the procedures for interpretation of decisions and the visualization of the learned parameters of convection models represent an interesting field of research. Within this work, the framework for automatic differentiation and the familiar library for handling matrices and images

were selected. The existing procedures for interpreting and visualizing convolutional models, their learned filters, and the way in which they see input images are described briefly.

Keywords: computer vision, convolutional neural networks, pytorch, machine learning, visualization of convolutional neural network