SVEUČILIŠTE U ZAGREBU

**FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA**

ZAVRŠNI RAD br. 6348

**VIZUALIZACIJA I INTERPRETACIJA KONVOLUCIJSKIH MODELA**

Martin Sršen

Zagreb, lipanj 2019.

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 problem 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. Područje inspirirano na temelju ljudskog vida koji je najvažnije osjetilo kod ljudi. Neke od pod-domena računalnog vida su rekonstrukcija prizora (proces hvatanja oblika I izgleda stvarnih objekata), otkrivanje događaja, video praćenje, 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) predstavljena svijetu 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 umejtne neuronske mreže je što se ne trabaju koristiti kompleksni matematički modeli te ručno upisivanje filtera već ih mreža sama uči na osnovu danih podataka.

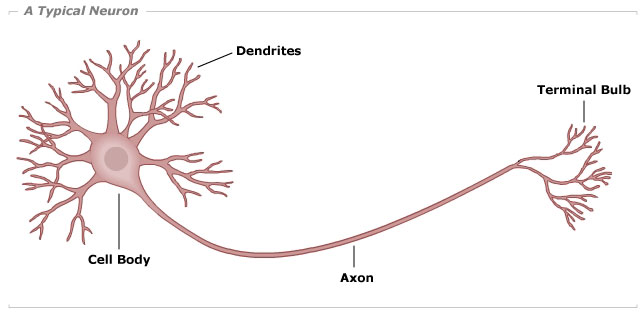
Cilj ovog rada je upoznati se s metodama vizualizacije te interpretacije konvolucijskih modela te njihovih parametara, proučiti metode vizualizacije, tj. vidjeti na koji način umjetna neuronska mreža prepoznaje objekt na slici, koji dijelovi slike igraju ključnu ulogu u klasifikaciji.

Umjetna neuronska mreža

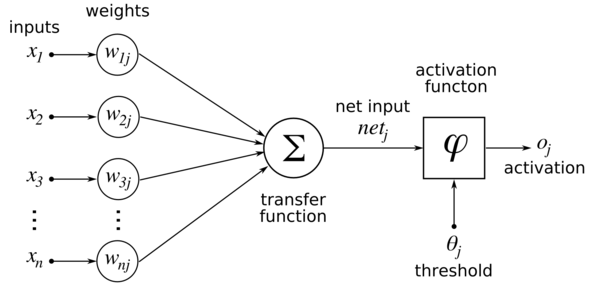
Umjetna neuronska mreža, jedan od važnijih algoritama strojnog učenja koji je motiviran na principu rada biološkog mozga. Mozak je središnji organ svakog živog bića koji mu omogućuje učenje, pamćenje te rješavanje raznih problema. Cilj umjetne inteligencije je upravo reproducirati mogućnosti mozga te ih poboljšati, pa je zbog toga razumno krenuti od funkcioniranja mozga te reporoduciranju funkcioniranja istog. Ljudski mozak se sastoji od oko 86 milijuna neurona međusobno povezanih sa 1014 - 1015 sinapsi.

Neuron

Osnovna građevna jedinica svake umjetne neuronske mreže. Inspiriran biološkim neuronom koji funkcionira na način 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.



Umjetni neuron je elementarna jedinica u umjetnoj neuronskoj mreži koji je zapravo matematička funkcija koja pokušava imitirati rad biološkog neurona. Funkcionira na način da prima jedan ili više ulaza ,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 pripadajučim težinama koje predstavljaju jačinu veze između dva neurona te se suma prenosi kroz nelinearnu aktivacijsku/prijenosnu funkciju. Izlaz aktivacijske funcije jednog neurona predstavlja ulaz idućeg neurona.



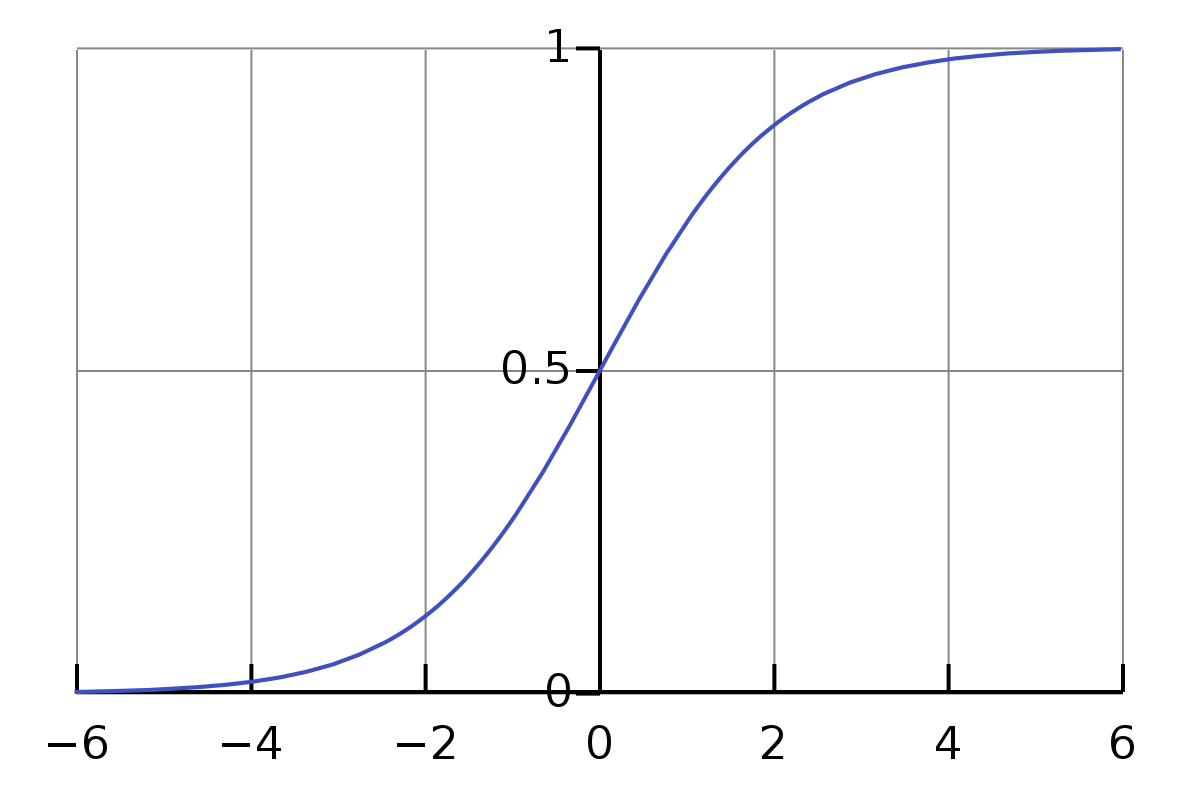
Funkcija koja predstavlja rad umjetnog neurona:

gdje je značenje varijabli iduće:

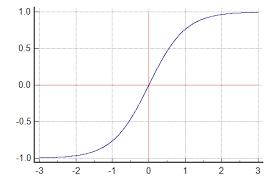
* y – izlaz neurona, tj. ulaz u idući neuron u nizu
* f – aktivacijska/prijenosna funkcija
* x – ulaz u neuron
* w – težina između prijašnjeg i trenutnog neurona
* – prag

Umjetni neuroni se razlikuju na temelju aktivacijskih/prijenosnih funkcija. Ona može biti proizvoljna te se odabire ovisno o problemu koji rješava. Kada nebi koristili aktivacijsku funkciju, mala promjena ulaza bi mogla potpuno promjeniti izlaz, te bi izlaz bio potpuno linearno ovisan o ulazu. Aktivacijske funkcije uvode nelinearnost u neuronsku mrežu. Bez nelinearnosti neuronske mreže nebi mogle modelirati komplicirane stvari kao 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 spuštanja ili bilo koje druge tehnike optimizacije kako bi se smanjila pogreška. Aktivacijske funkcije koje se najčešće koriste su:

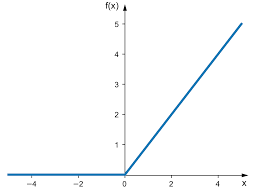
* Sigmoidalna funkcija – glavno obilježje ove funkcije je da svaki ulaz skalira na interval između 0 i 1. Jako pozitivni brojevi poprimaju vrijednost približno 1, dok jako negativni približno 0. Sigmoidalna funkcija se koristi sve manje zbog niza razloga, jedan od glavnih je što kod povratne propagacije gradijent postaje sve manji kroz mrežu, te za duboke neuronske mreže može potpuno nestati.



* 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 preporućeno koristiti nju u odnosu na sigmoidalnu funkciju.



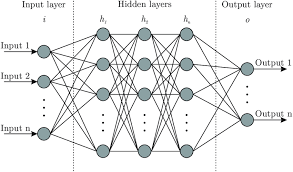
* ReLU (Rectified Linear Unit) – funkcionira na način da propušta ulaze koji su pozitivni, tj. veći od 0. Glavna prednost je brzina računanja što smanjiva vrijeme treniranja. Glavni problem je što neuroni mogu „odumrijeti” na način da preveliki gradijent izmjeni težine na način da se neuron više nikad ne aktivira. Problem se može rješiti na način da gradijent ne bude 0 nego neka mala vrijednost.



Arhitektura neuronske mreže

Neuronska mreža nastaje povezivanjem većeg broja neurona. Osnovna arhitektura mreže je podjeljena 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



Primjer kod klasifikacijskog problema prepoznavanja rukom pisanih znamenki imat ćemo ulaznih neurona jednako broju pixela u ulaznoj slici te 10 izlaznih neurona, po jedan za svaku znamenku koju prepoznajemo.

Svaki sloj neuronske mreže ima vezu sa susjednim na način 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 skojeva 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.

Učenje neuronske mreže

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

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

Cilj neuronske mreže kod nadziranog učenja je minimizirati funkciju gubitka. Razlikujemo dvije grupe funkcija gubitka 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,

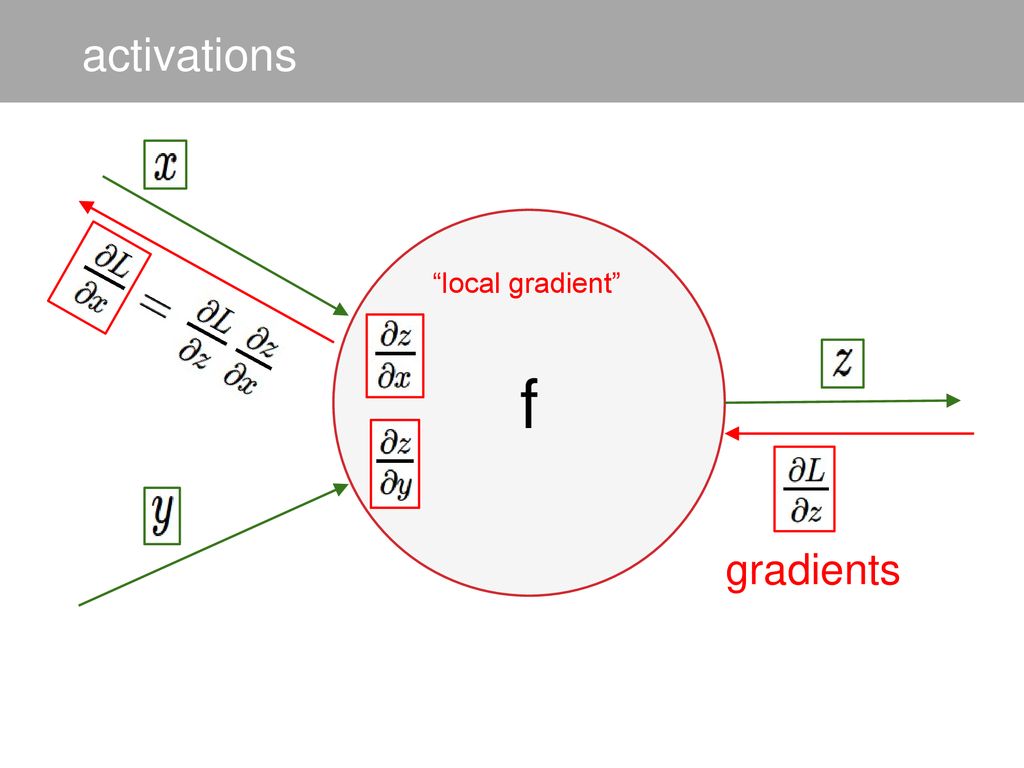
dok za problem klasifikacije se najčešće koristi unakrsna entropija (engl. cross entropy loss)

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

gdje je 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 prag 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 pravilu sljedećem pravilu ulančavanja:

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



Cijeli proces učenja neuronske mreže možemo podjeliti 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 grešku 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 podjeliti podatke za učenje na 3 cjeline:

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

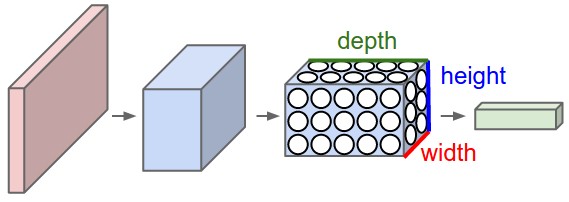
Ovo je važno zbog toga da lakše prepoznamo problem. Postoje razni načini regularizacije da izbjegnemo ovaj problem, ali nećemo ih ovdje razmatrati.

**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 računalni vid. Tipične konvolucijske neuronske mreže imaju 10-ak slojeva čime ih možemo smjestiti u duboke neuronske mreže. Dobila je 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 neuronsku mrežu je što koristi konvolucijske filtere 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 na način da koriste konvolucijske filtere koji traže određene karakteristike i oblike u slikama te ih spremaju u mape značajki koje se šalju dalje kroz mrežu. Kada slika prolazi kroz filtere, također joj se smanjuju dimenzije te se ubrzava daljnje izračunavanje kroz mrežu. Zbog kompleksnosti rada sa slikama u konvolucijske neuronske mreže obično se koriste slojevi sažimanja (engl. pooling) koji smanjivaju dimanzije slike, obično za faktor 2. Konvolucijske mreže kao ulaz primaju podatke u obliku višedimenzionalnog polja, u slučaju crno-bijelih slika 2D, te u boji 3D.

Arhitektura

Za raliku od neuronskih mreža koja ima slojeve u jednoj dimenziji, konvolucijska neuronska mreža ima slojeve u tri dimenzije: širina, visina i dužina. Širina i visina predstavljaju dimenzije mapi značajki te dužina broj ulaznih mapi 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 dimenzija i svrha mu je da možemo svrstati ulaz u jednu od kategorija.



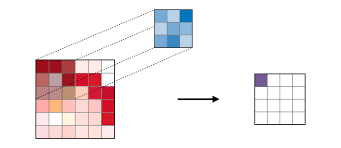
Konvolucijska neuronska mreža se sastoji od 3 glavna sloja: konvolucijskih slojeva (eng. convolutional layers), slojeva sažimanja (eng. pooling layers) i potpuno povezanih slojeva (eng. fully connected layers).

Konvolucijski sloj je najznačajniji dio konvolucijske neuronske mreže po kojoj je dobila i ime. Sastoji se od jednog ili više filtera koji sadrže težine koje se uče algoritmom propagacije pogreške unatrag. Filter ili jezgra, je zapravo matrica obično puno manje širine i visine od ulaza, ali pokrivaju čitavu dubinu ulaza čija je uloga pronalaženje određenih karakteristika u slici. Konvolucijski sloj je definiran s 3 glavna parametra:

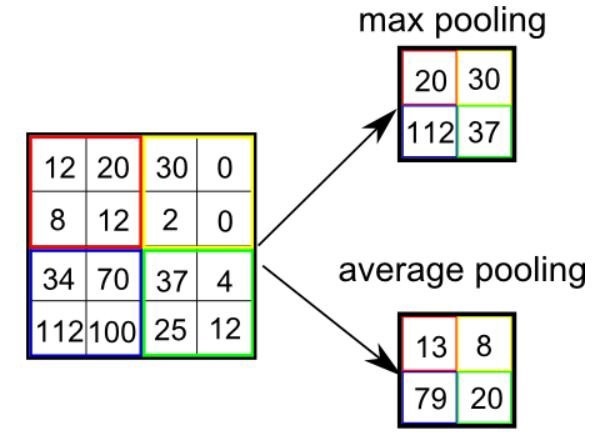
1. dubina (eng. depth) – odgovara broju filtera sadržanih u konvolucijskom sloju
2. korak (eng. stride) – odgovara koraku pomaka filtera
3. nadopunjavanje nulama (eng. zero-padding) – koristi se najčešće kad zahtjevamo da izlazne dimenzije konvolucijskog sloja budu jednake ulaznim

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

gdje predstavlja dimenziju mape i-tog sloja, dimenzije filtera koji povezuju mape prethodnog sloja s mapama trenutnog sloja, predstavlja korak pomaka filtera po širini i visini prilikom konvolucije. Ovdje smo radi jednostavnosti koristili kvadratne mape značajki. Konvolucija se tvori prolazom prozora veličine jednake dimenziji filtera kroz ulaznu mapu te se množe vrijednosti filtera 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 filtera u konvolucijskom sloju. Obično se nakon konvolucijskog sloja mape značajki provode kroz ReLU aktivacijsku funkciju čime se uvodi nelinearnost u mrežu.



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 na način da prozor veličine h x w ulazne mape mapira u jednu vrijednost. Najčešće se koristi prozor veličine 2 x 2 te na taj način širinu i visinu ulazne mape smanjimo na pola. Dva uglavnom korištena načina dobivanja vrjednosti iz prozora su preko maksimalne (eng. max pooling) ili srednje vrijednost (eng. average pooling) u bloku. Slojevi sažimanja ne smanjuju broj mapa i ne posjeduju nikakve težine koje je potrebno učiti.



Potpuno povezani sloj je ključan sloj u klasifikaciji ulazne slike jer upravo u ovom sloju možemo dobiti raspodjelu vjerovatnosti 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 velike vrjednost 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 jednodimenzijalan što znači da težine možemo prikazati vektorom. Računanje aktivacijske vrijednost se računa kao kod neuronskih mreža, množenjem ulaza sa 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.

Programska podrška

Programski jezik Python

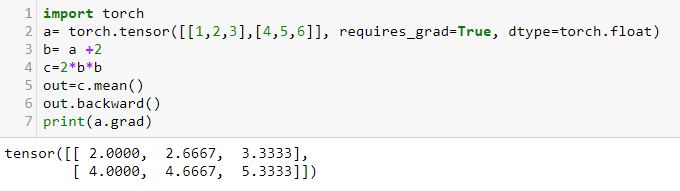
Python je interpretirani programski jezik visoke razine opće namjene. Objavljen je 1991. godine od strane Guida van Rossuma. Python naglašava čitljivost koda korištenjem razmaka te uvlačenja teksta. Njegovi jezični konstrukti te objektno-orijentirani pristup pomažu programerima da pišu jasan, logičan 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.

Programska biblioteka PyTorch

PyTorch je programska biblioteka otvorenog koda (eng. open source code) namjenjena pretežno za implementiranje rješenja u strojnom učenju u pythonu. Vrlo mlada i nova biblioteka kreirana od strane instraživačke skupine umjetne inteligencije u Facebooku. Jezik je baziran na biblioteci Torch koja je implementirana u jeziku LuaJIT, koji ne zna mnogo ljudi stoga je library implementiran u pythonu te nazvan PyTorch. 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 mogučnošću automatskog praćenja gradijenta (Autograd) te tako omogućuje jednostavnu implementaciju algoritma propagacije pogreške unatrag

Verzija korištena za implementaciju rješenje je PyTorch 1.1.



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. To nam može puno reći o tome gdje tražiti i kako ispraviti greške u modelu da bi u budućnosti dobili bolje rezultate. U ovom radu je odrađeno par jednostavnijih načina vizualizacije konvolucijske neuronske mreže:

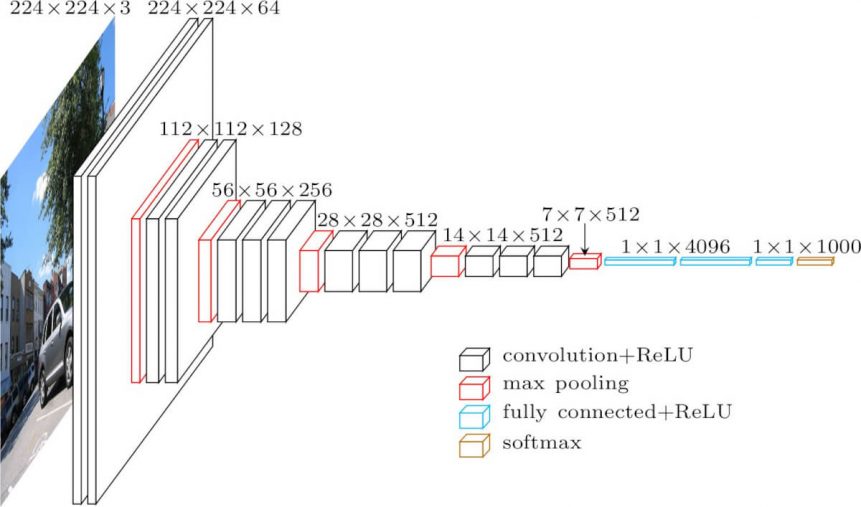
* vizualizacija klase modela (eng. class model visualization)
* mapa prepoznatljivosti (eng. saliency map)
* vođeno provođenje gradijenta unatrag (eng. guided backpropagation)
* aktivacijske mape klase (eng. class activation maps)
* vizualizacija sloja (eng. layer visualization)
* deep dream

Implementacija i rezultati

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. To 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 – filter 3x3, 3 ulazna kanala, 64 izlazna kanala što znači da imamo 64 filtera u ovom konvolucijskom sloju, pomak od 1 te nadopunjavanjem od 1. Izlaz prolazi kroz ReLU aktivacijsku funkciju
3. Konvolucijski sloj – filter 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 – filter 3x3, 64 ulazna kanala, 128 izlaznih kanala, pomak od 1 te nadopunjavanje od 1. Izlaz prolazi kroz ReLU
6. Konvolucijski sloj – filter 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 – filter 3x3, 128 ulazna kanala, 256 izlaznih kanala, pomak od 1 te nadopunjavanje od 1. Izlaz prolazi kroz ReLU
9. Konvolucijski sloj – filter 3x3, 256 ulazna kanala, 256 izlaznih kanala, pomak od 1 te nadopunjavanje od 1. Izlaz prolazi kroz ReLU
10. Konvolucijski sloj – filter 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 – filter 3x3, 256 ulazna kanala, 512 izlaznih kanala, pomak od 1 te nadopunjavanje od 1. Izlaz prolazi kroz ReLU
13. Konvolucijski sloj – filter 3x3, 512 ulazna kanala, 512 izlaznih kanala, pomak od 1 te nadopunjavanje od 1. Izlaz prolazi kroz ReLU
14. Konvolucijski sloj – filter 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 – filter 3x3, 512 ulazna kanala, 512 izlaznih kanala, pomak od 1 te nadopunjavanje od 1. Izlaz prolazi kroz ReLU
17. Konvolucijski sloj – filter 3x3, 512 ulazna kanala, 512 izlaznih kanala, pomak od 1 te nadopunjavanje od 1. Izlaz prolazi kroz ReLU
18. Konvolucijski sloj – filter 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



Dvije najveće mane ove VGG16 konvolucijske neuronske mreže su:

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

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



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

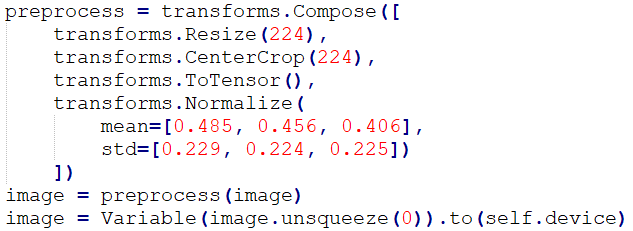


Učitavanje te procesiranje ulaznih slika

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

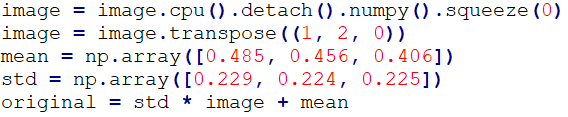


Nakon toga potrebno je transformirati ulaznu sliku u format ulaza mreže, tj. u tensor dimenzija [3, 224, 224]. Kreiramo Variable klasu koja prima dobiveni tensor, te tu klasu koristimo za ulaz u VGG16.



Torchvision paket sadrži metode pomoću kojih jednostavno možemo obrađivati ulazne slike. Pomoću transforms.Compose dobijemo metodu koja čini transformacije nad slikom u zadanom redoslijedu. Nakon što sliku transformiramo u dimenzije ulaza mreže te je pretvorimo u tensor, trebamo normalizirati podatke na onaj način koji mreža zahtjeva, to zadajemo preko transforms.Normalize.

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

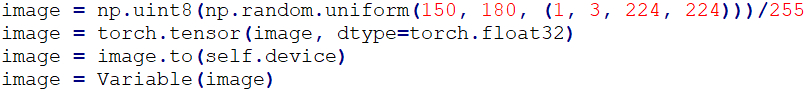


U prvom koraku, cpu, prebacujemo računanja slike na procesor jer nije moguće daljnje radnje izvoditi na GPU, nakon toga zovemo detach što znači ako tenzor slike pamti gradijent prestaje ga pratiti. Transpose mijenja dimenzije slike sa [3, 224, 224] na [224, 224, 3] što je bitno kod prikazivanja slika.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

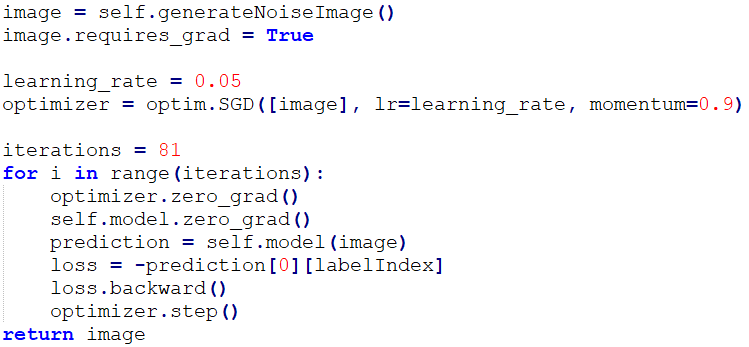
**Izvedba vizualizacije konvolucijske neuronske mreže**

Vizualizacija klase modela (eng. class model visualization)

U ovom dijelu ćemo implementirati te pokazati rezultate vizualizacije klase modela. 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:

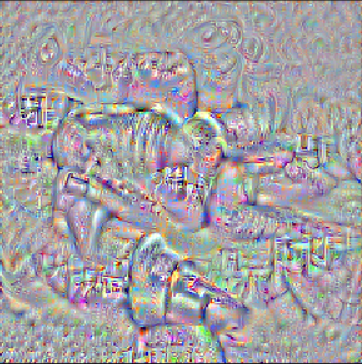


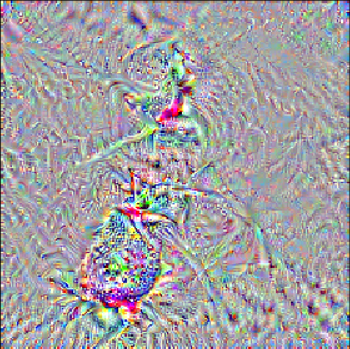
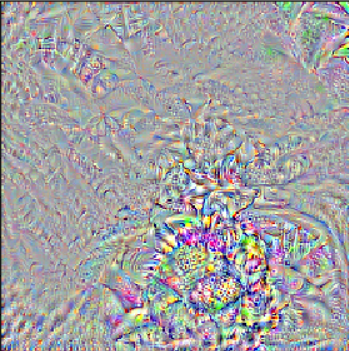
Vizualizacija klase modela funkcionira na način da izračunamo aktivaciju mreže za danu klasu u izlaznom sloju te ažuriramo piksele na taj način da u idućoj iteraciji aktivacija za istu klasu u izlaznom sloju bude veća. To možemo učiniti na način da ažuriramo one piksele proporcionalno onome koliko su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



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 težine na 0 nakon čega možemo ponovno računati gradijente. Kao funkciju gubitka koristimo negativnu aktivaciju klase, te će optimizacijska funkcija ažurirati piksele na način da smanji grešku što znači povećanje aktivacije u idućoj iteraciji. Optimizer.step vrši ažuriranje težina na principu već poznate formule:

, s tim da je modificirana na način da vrši učenje na optimalniji način. U nastavku ćemo prikazati par rezultata algoritm.

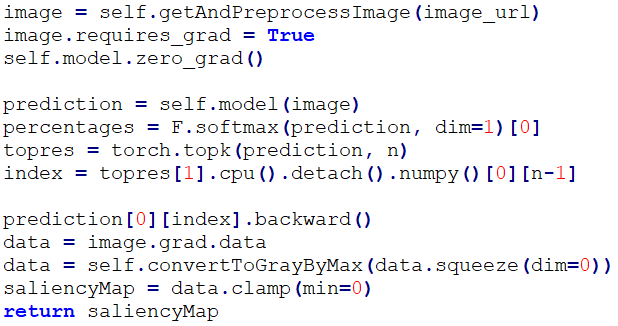
  
dumbell

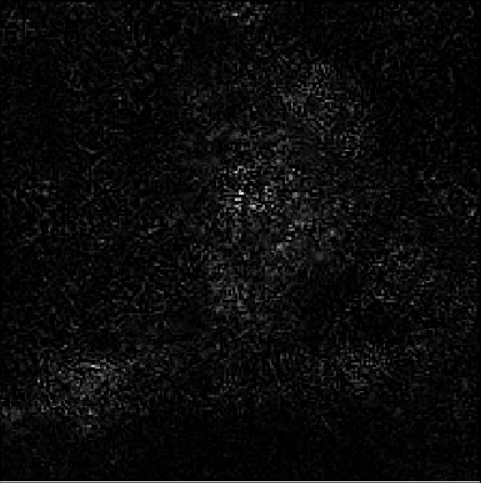
Strawberry pineapple

mapa prepoznatljivosti (eng. saliency map)

Pretpostavimo da sve slike za vježbanje klase točak sadrže u sebi također sliku vozila. Kako možemo znati da li CNN koristi piksele povezane sa 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.



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 indeks n najboljih rezultata. 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 prepoznatljivosti. 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 sa 0. U nastavku ćemo prikazati par rezultata algoritma.

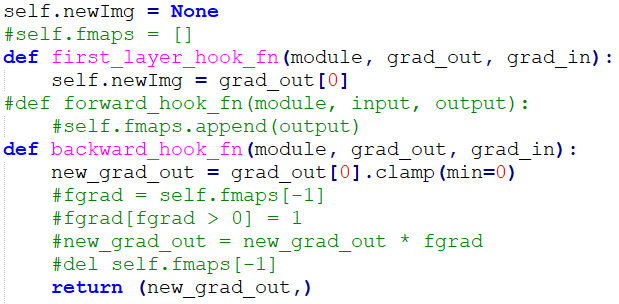
 

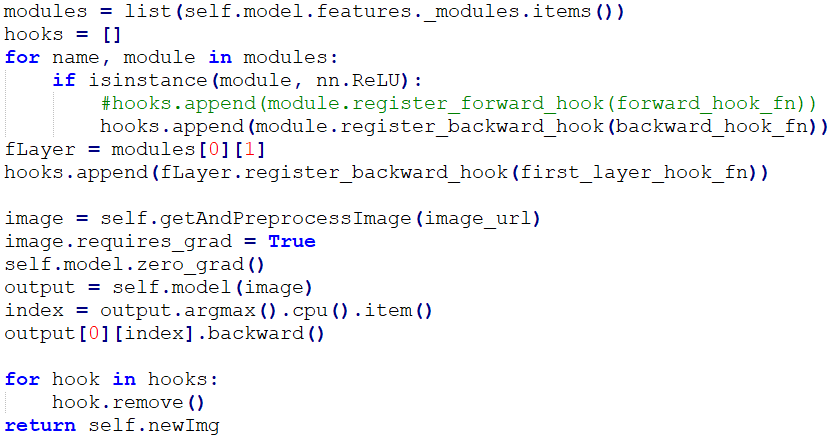
 

Vođeno provođenje gradijenta unatrag (eng. guided backpropagation)

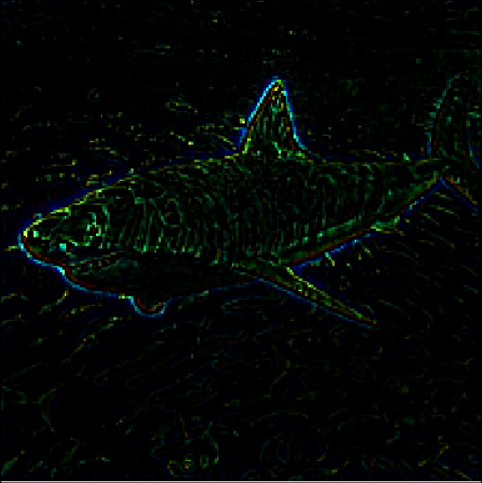
Algoritam se temelji na ideji da nas zanimaju samo uzorci na slici koje je neuron detektirao, te želimo sve gradijente koji negativno utječu na aktivaciju izlaznog sloja izjednačiti s 0. Algoritam zovemo vođeno provođenje gradijenta unatrag zbog toga što ažuriramo, mijenjamo gradijente prije izlaza iz svakog sloja. Na ovaj način na ulazu, tj. gradijentu ulazne slike dobijemo samo one vrijednost koje pozitivno utječu na promjenu izlaznog sloja. Algoritam također zahtjeva da zadržimo samo one gradijente koji su imali pozitivnu aktivaciju u prolasku kroz mrežu u naprijed, ali se ne trebamo brinuti zbog toga jer 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.



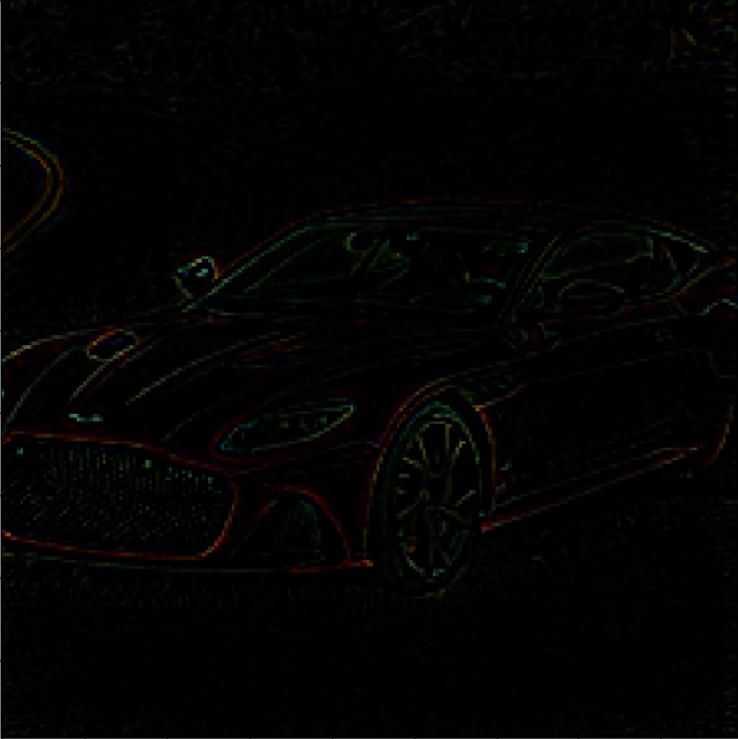
Komentirani dio programa je potrebno koristiti u slučaju da naša mreža ne koristi ReLU koji nam automatski sve gradijente negativnih aktivacija izjednači s 0. Dio algoritma koji računa izlaznu sliku:



Modules 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 vračamo sliku. U nastavku ćemo prikazati par primjera rada ovog algoritma.

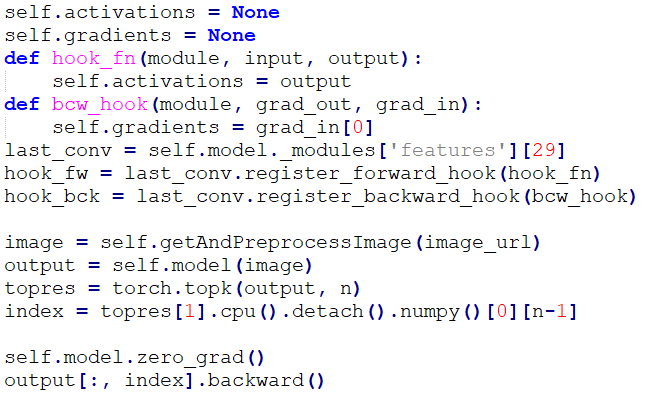
 

aktivacijske mape klase (eng. class activation maps or CAM)

CAM je jednostavan algoritam pomoću kojega dobijemo regije na slici koje konvolucijska neuronska mreža koristi da bi odredila klasu ulazne slike. 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 filtera, a kako znamo što dublje u mrežu idemo, možemo pomoću filtera 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 filteru utjecala na donošenje odluke.



1. Izračunamo srednju vrijednost gradijenta za svaki od filtera, tj. koliko je prosječno svaki filter utjecao na donošenje odluke da je upravo to ta klasa na slici.



1. Pomnožimo aktivaciju svakog filtera sa prosječnom vrijednosti koliko je filter utjecao na klasifikaciju.



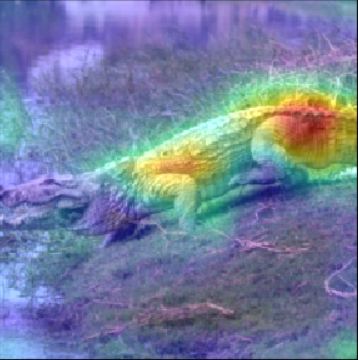
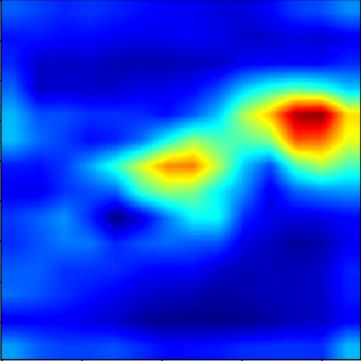
1. Nađemo srednju vrijednost koliko je koji neuron u filterima imao utjecaja na izlaz klasifikacije na način da nađemo prosječnu vrijednost od 512 filtera za svaki od 14x14 neurona

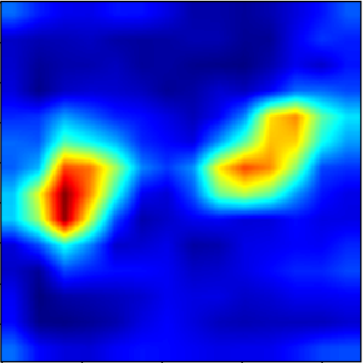


1. Nakon što smo dobili toplinsku kartu dimenzija 14x14 trebamo je još skalirati na veličinu ulazne slike te ih spojiti na sljedeći način:



U nastavku su prikazani rezultati primjene algoritma.

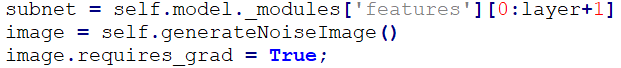




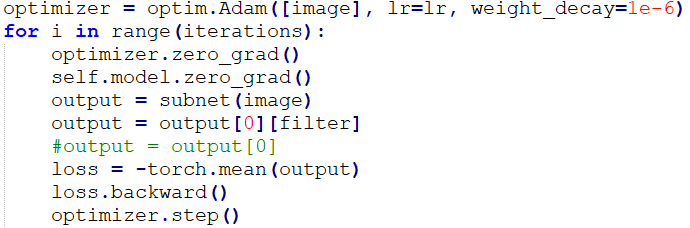
vizualizacija sloja (eng. layer visualization)

Vizualizacija sloja je algoritam kojim možemo vizualizirati određene filtere unutar određenog sloja ili cijeli sloj skupa tako da odredimo srednju vrijednost svih filtera unutar sloja. U mogućnosti smo primijetiti kako su filteri 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 sloj ili filter kojeg vizualiziramo. Algoritam možemo opisati u par 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.

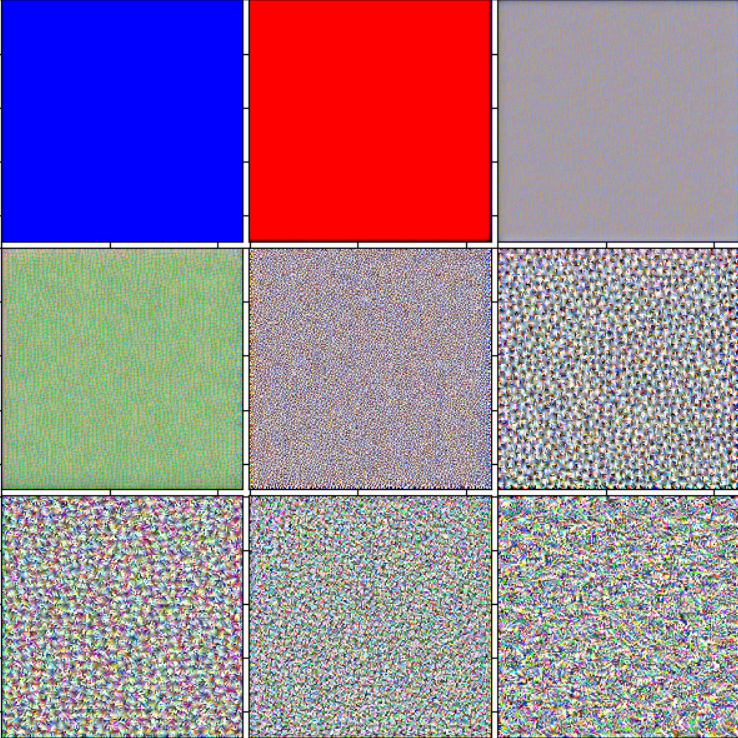


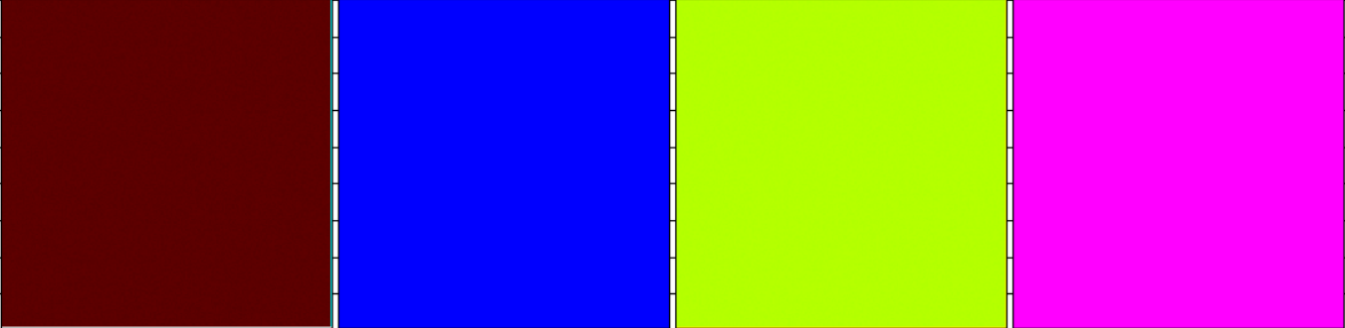
1. Kroz svaku iteraciju računamo aktivaciju sloja kojeg vizualiziramo za danu sliku sa slučajno odabranim vrijednostima. Izračunamo gradijent srednje vrijednosti(aktivacije) filtera kojeg vizualiziramo u odnosu na ulaznu sliku te je ažuriramo preko izračunatih vrijednosti. Ako vizualiziramo cijeli sloj,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

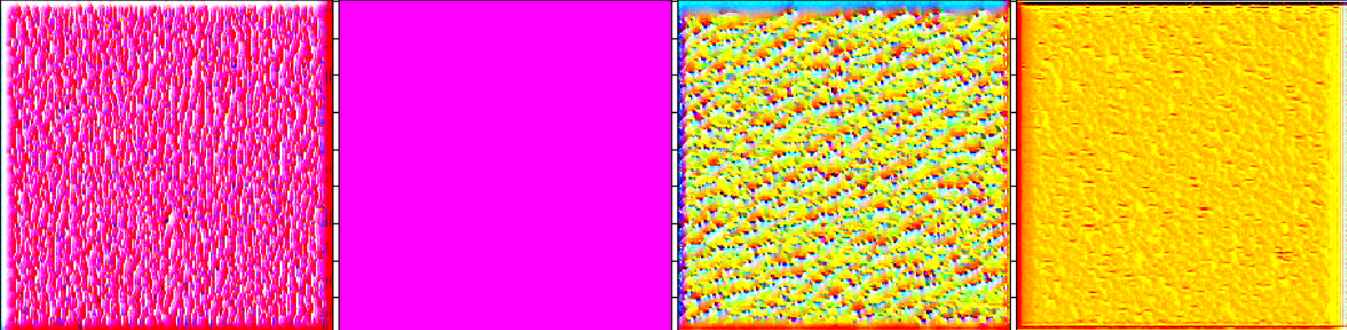


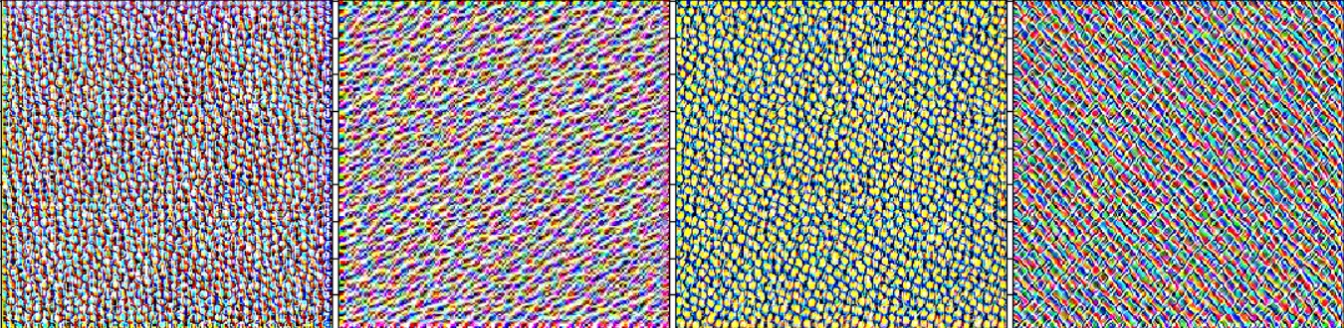
Dio koda u komentarima je način vizualizacije cijelog sloja umjesto jednog filtera.

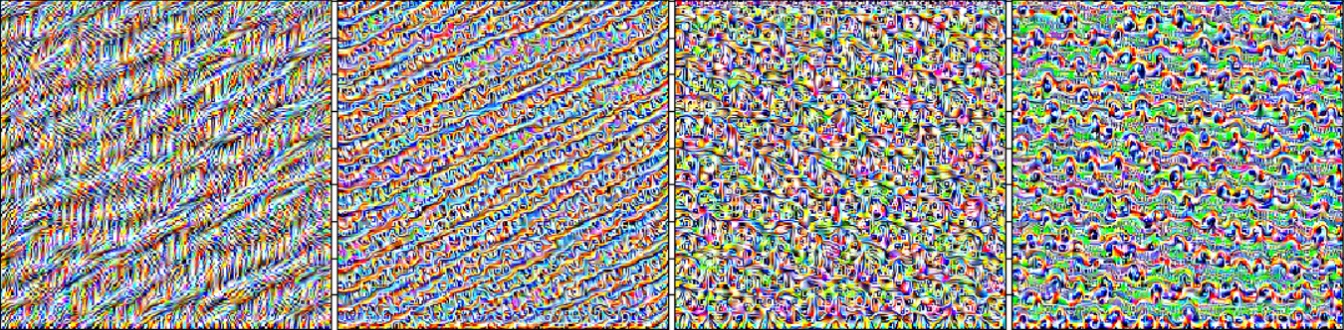
U nastavku ćemo pokazati rezultate algoritma, za vizualizaciju cijelog sloja gledamo prvih 9 konvolucijskih slojeva, dok za vizualizaciju filtera gledamo slučajno odabrana 4 filtera iz par konvolucijskih slojeva:

Viz konv1-9

1.

2.

5.

8.

deep dream