Klasifikacija slika korištenjem konvolucijskih neuronskih mreža

Milaković, Roberto

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Roberto Milaković

KLASIFIKACIJA SLIKA KORIŠTENJEM KONVOLUCIJSKIH NEURONSKIH MREŽA

DIPLOMSKI RAD

Varaždin, 2024.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Roberto Milaković

Matični broj: 0016137990

Studij: Informacijsko i programsko inženjerstvo

KLASIFIKACIJA SLIKA KORIŠTENJEM KONVOLUCIJSKIH NEURONSKIH MREŽA

DIPLOMSKI RAD

Mentorica:

Prof. dr. sc. Jasminka Dobša

Varaždin, lipanj 2024.

Roberto Milaković

Izjava o izvornosti

Izjavljujem da je moj završni/diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor/Autorica potvrdio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

U ovome radu bavit ćemo se konvolucijskim neuronskim mrežama, Prvo ćemo napraviti kratki uvod u područje neuronskih mreža te ćemo objasniti što su to neuronske mreže i proći ćemo način rada klasičnih jednoslojnih mreža i način rada višeslojnih mreža. Proučit ćemo kako klasične višeslojne mreže uče i kako se treniraju. Zatim ćemo napraviti kratki uvod u konvolucijske neuronske mreže. Proći ćemo strukturu i slojeve od kojih su te mreže napravljene. Objasnit ćemo što je to konvolucija, a što sažimanje. Pošto se konvolucijske mreže također same uče objasnit ćemo propagaciju unazad kod navedenih slojeva. Zatim ćemo proći kroz nekoliko primjera konvolucijskih mreža i objasniti ih detaljno. Na kraju nam preostaje implementirati vlastitu konvolucijsku mrežu, provesti ju nad skupom podataka i objasniti kako ona radi.

Ključne riječi: perceptron; aktivacijska funkcija; višeslojna neuronska mreža; gradijentni spust; propagacija unazad; konvolucija; sažimanje; konvolucijska neuronska mreža

Sadržaj

1. Uvod

Mozak je najsloženiji organ u cijelome ljudskom tijelu, a također i jedna od najsloženijih stvari na svijetu. Mnogi znanstvenici već duže vrijeme istražuju kako tako mali organ može toliko puno stvari postići. Istražujući ljudski mozak možemo raspoznati kako ljudi uče pomoću raznih osjetila.

Kako je računalna znanost jako napredovala u zadnjih nekoliko desetaka godina tako je popularnost strojnog učenja porasla. Danas se sve više i više kompleksnijih zadataka obavlja pomoću algoritama koji primjenjuju nekakvu vrstu učenja kako bi ih brzo riješili. Jedan od većih problema s kojima se računalna znanost susreće je prepoznavanje slika računalnim vidom. Dok računala jako lagano mogu rješavati matematičke zadatke, ona imaju problem sa prepoznavanjem slika jer su za računala slike nule i jedinice pa ne znaju šta predstavljaju.

Premda ljudski mozak jako dobro uz pomoć očiju može prepoznati objekte, znanstvenici su došli do zaključka da primjene način učenja kod ljudskog mozga tako da i računala mogu prepoznavati slike. Znamo da mozak u sebi sadrži preko 80 milijardi neurona (živčanih stanica) koje čine neuronsku mrežu te su na sličan način napravljeni algoritmi umjetnih neuronskih mreža koje ćemo kasnije objasniti.

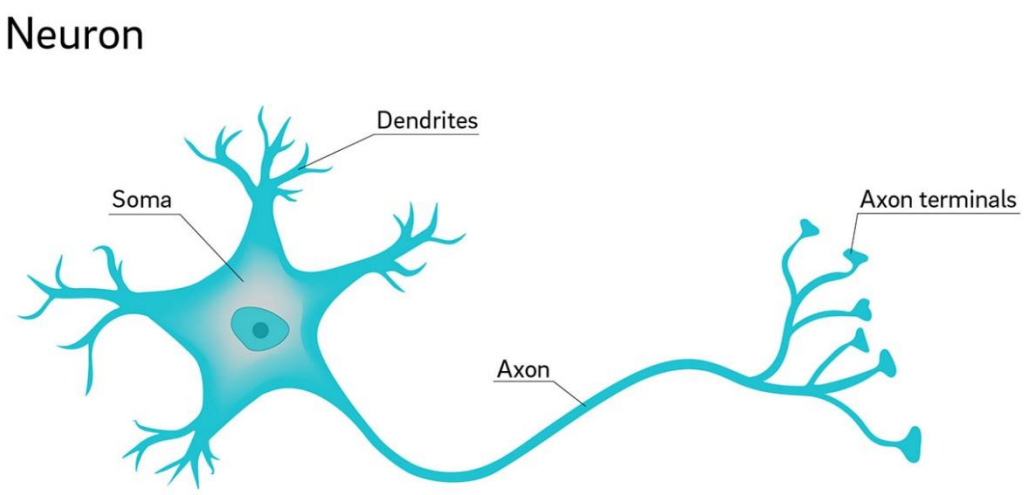
U ovome radu ćemo objasniti strukturu klasičnih i konvolucijskih neuronskih mreža, te ćemo objasniti njihov proces učenja i kako funkcioniraju, a što se praktičnog dijela rada tiče prikazat ćemo na odabranome skupu slika vlastiti primjer klasifikacije slika pomoću konvolucijskih neuronskih mreža.

2. Neuronske mreže

Biološke neuronske mreže su građene od neurona odnosno živčanih stanica. Oni kao i svaka druga stanica imaju DNK kod te su generirani na isti način kao i sve druge stanice [1]. Međutim neuroni se razlikuju prema tome šta prenose informacije. Neuron je napravljen od 3 dijela:

- **soma** – tijelo stanice,
- **dendriti** – grane raspršene uokolo koje primaju signale od drugih neurona,
- **akson** – grana koja prenosi signal na druge neurone [1].

Prijenos signala se ostvaruje putem sinapse gdje akson jednog neurona šalje signal na dendrite drugog njemu bliskog neurona. Slika 1. nam prikazuje sve dijelove biološkog neurona.

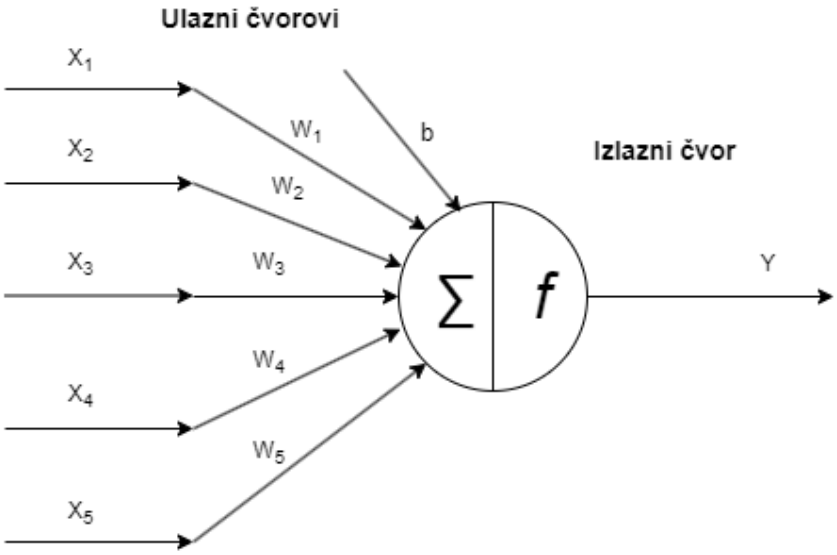


Slika 1: Prikaz biološkog neurona i njegovih dijelova. Preuzeto sa [2]

Gledajući princip rada bioloških neuronskih mreža može nam pomoći razumjeti kako su znanstvenici izradili umjetne neuronske mreže koje se danas u svakakvim poslovima koriste kako bi brzo i efikasno sa velikim postotkom točnosti odradili zadatke.

2.1. Jednoslojne umjetne neuronske mreže

Jedan od prvih modela umjetne neuronske mreže implementirao je Frank Rosenblatt 1950-ih godina, a taj model se zove perceptron. Osim šta je primjer jedne od najjednostavnijih umjetnih neuronskih mreža perceptron je također model umjetnog neurona. Slika ispod prikazuje perceptron.



Slika 2: Model perceptrona (vlastita izrada)

Perceptron radi na jako sličan način kao i neuron, odnosno ima ulazne čvorove i jedan izlazni čvor. Sastoji se od sljedećih dijelova:

- $X_1, X_2, \ldots, X_n$ – set ulaznih vrijednosti za predviđanje izlazne vrijednosti
- $W_1, W_2, \ldots, W_n$ – težine koje dodaju važnost ulaznim vrijednostima kod predviđanja izlazne vrijednosti
- $\sum$ – funkcija zbrajanja, zbraja sve ulazne vrijednosti pomnožene sa njihovim težinama
- f – aktivacijska funkcija koja transformira rezultat zbroja u određene vrijednosti, možemo gledati to kao funkciju koja određuje, hoće li se perceptron odnosno umjetni neuron aktivirati ili ne s obzirom na rezultat zbroja
- b – pristranost (eng. bias), broj koji se dodaje sumi tako da pomakne rezultat aktivacijske funkcije prema jednoj ili drugoj vrijednosti
- Y – izlazna vrijednost koja je dobivena primjenom aktivacijske funkcije [3].

Primjenjujući sve navedene dijelove perceptrona izlaznu vrijednost Y dobivamo pomoću sljedeće jednadžbe:

$$Y = f\left(\sum_{i=1}^{n} X_i W_i + b\right). \tag{2.1}$$

Kao što smo već rekli f predstavlja funkciju aktivacije, međutim postoji više funkcija aktivacije (linearna, sigmoidna, Tanh, ReLU, itd.) koje se koriste u primjeni. U većini slučajeva se koriste nelinearne aktivacijske funkcije. Za jednostavni primjer ćemo uzeti sigmoidnu aktivacijsku funkciju kako bi pokazali rad aktivacijske funkcije, a njezina formula izgleda ovako:

$$f(x) = \frac{1}{1 + e^{-x}}\ . \tag{2.2}$$

Sigmoidna funkcija daje vrijednosti od 0 do 1 koje predstavljaju vjerojatnost te je dobra u slučajevima kod modela gdje se treba predvidjeti vjerojatnost [4]. Ovdje nam x predstavlja zbroj ulaza pomnoženih s njihovim težinama te ako je x jako velik onda se izlaz bliži prema vrijednosti 1, međutim ako je x jako nizak onda dobivamo izlaznu vrijednost koja se približava 0. Aktivacijske funkcije se odabiru ovisno o tome o kakvoj se klasifikaciji radi.

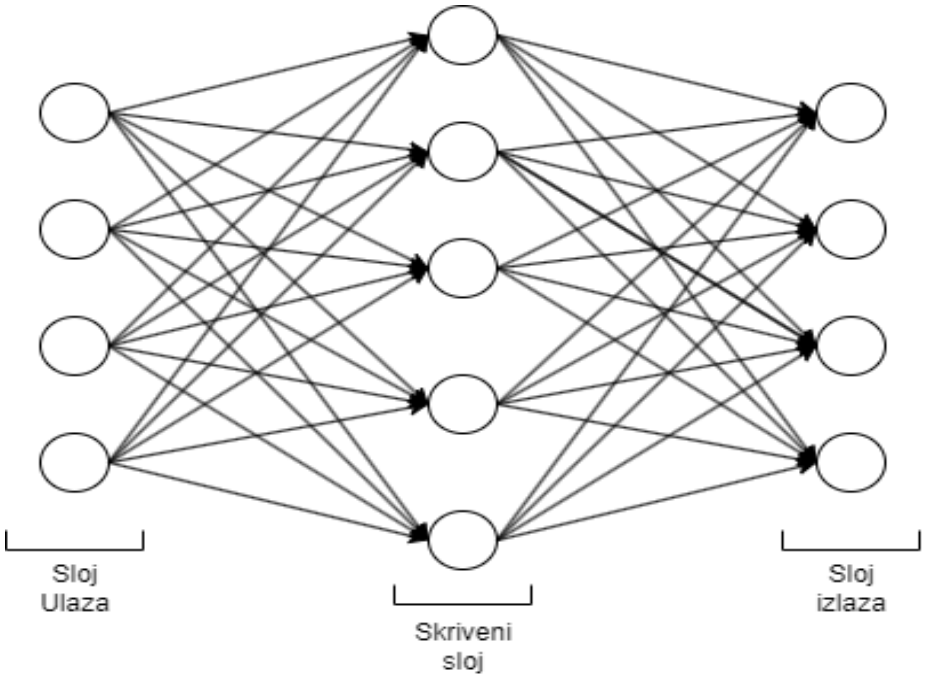
Jednoslojne neuronske mreže su građene tako da imaju samo jedan sloj ulaza, i jedan izlaz odnosno te mreže koriste samo jedan perceptron dok su višeslojne neuronske mreže građene od više perceptrona. Nadalje ćemo objasniti rad višeslojnih neuronskih mreža.

2.2. Višeslojne umjetne neuronske mreže

Kao što vidimo iz samoga naziva višeslojne neuronske mreže su građene od više slojeva, a svaki sloj se sastoji od mnogo perceptrona. Kada govorimo o višeslojnim neuronskim mrežama obično govorimo o mrežama sa najmanje tri ili više slojeva perceptrona. Takve neuronske mreže sastoje se od jednog sloja ulaza, jednog ili više skrivenih slojeva i jednog sloja izlaza [5]. Svrha sloja ulaza je da proslijedi vrijednosti koristeći linearne funkcije skrivenim slojevima koji koriste nelinearne ili linearne funkcije za učenje i predviđanje, a sloj izlaza prima obrađene podatke od skrivenih slojeva i proizvodi konačne odluke i rezultate [5], [6], [7].

U suštini višeslojne umjetne neuronske mreže koriste *„Feedforward“* arhitekturu gdje svi slojevi mreže prosljeđuju podatke unaprijed. Osim toga postoje i višeslojne neuronske

mreže sa arhitekturom u oba smjera međutim nećemo se trenutno fokusirati na njih. Slika 3 nam prikazuje primjer višeslojne neuronske mreže sa jednim skrivenim slojem.



Slika 3: Prikaz jednostavne višeslojne neuronske mreže (vlastita izrada)

Kako višeslojne neuronske mreže katkad stvaraju pogreške pri odlučivanju i prikazivanju rezultata potrebno ih je trenirati odnosno naučiti da prikazuju točne rezultate. Za učenje višeslojne neuronske mreže koriste se algoritam gradijentnog spusta i propagacije unazad.

2.2.1. Algoritam gradijentnog spusta

Algoritam gradijentnog spusta minimizira funkciju gubitka odnosno greške koja služi za određivanje koliko dobro se rezultati neuronske mreže poklapaju sa stvarnim podacima. Za optimizaciju funkcije greške tražimo minimum te funkcije. Ako nam je funkcija greške konveksna onda tražimo globalni minimum, ali kako u većini slučajeva kod neuronskih mreža ta funkcija nažalost nije konveksna tražimo lokalni minimum [8]. Kako nam derivacija funkcije $f'(x)$ daje nagib funkcije $f(x)$ u vrijednosti x, to će nam poslužiti kao način kako da napravimo mali korak prema minimiziranju funkcije [9]. Algoritam osim toga ovisno o primjeni koristi ili konstantu ili varijablu koja naznačuje stopu učenja koja će biti označena sa c. Jednadžba gradijentnog spusta izgleda ovako:

$$k_{i+1} = k_i - c\nabla f(k_i), \tag{2.3}$$

$$\nabla f(k_i) = \begin{bmatrix} \frac{\partial f}{\partial x_1}(k_i) \\ \vdots \\ \frac{\partial f}{\partial x_n}(k_i) \end{bmatrix}. \tag{2.4}$$

- $\nabla f(k_i)$ – naznačuje gradijent funkcije greške sa n dimenzija,
- k_i – trenutna pozicija (korak) u koordinatnom sustavu koja naznačuje kolika je vrijednost funkcije greške,
- k_{i+1} – buduća pozicija (korak) u koordinatnom sustavu,
- c – stopa učenja određuje veličinu koraka.

Koraci algoritma gradijentnog spusta su sljedeći:

- odabrati neku nasumičnu vrijednost za prvi korak,
- primijeniti jednadžbu gradijentnog spusta (jednadžba 2.3),
- ponavljati drugi korak sve dok se ne dođe do stabilizacije funkcije greške ili dok se ne izvrši maksimalni broj iteracija [9].

2.2.2. Algoritam propagacije unazad

Kod neuronske mreže sloj ulaza prenosi vrijednosti na skrivene slojeve. Nakon toga prvi skriveni sloj uzima te vrijednosti, množi ih dodijeljenim težinama i računa sumu tih umnožaka i provodi aktivacijsku funkciju nad vrijednostima. Rezultat aktivacijske funkcije prosljeđuje se idućem skrivenom sloju i tako sve dok se ne prođu svi skriveni slojevi. Nakon toga isti postupak radi sloj izlaza koji dobiva kao rezultat vjerojatnosti s kojima program proizvodi odluke. Cijeli taj proces se naziva propagacija unaprijed.

Nakon dobivenih rezultata računa se funkcija greške (gubitka) gdje se uspoređuje rezultat neuronske mreže sa stvarnim vrijednostima. Preostaje nam računanje gradijenta funkcije greške kako bi se mogao primijeniti algoritam gradijentnog spusta. Za računanje gradijenta funkcije greške potreban nam je algoritam propagacije unazad. Gledajući elemente neuronske mreže možemo primijetiti da jedine elemente kojima vrijednosti možemo mijenjati, a da utječu na rezultat točnosti neuronske mreže, su težine w i pristranost b (engl. bias). Prolaskom kroz višeslojnu neuronsku mrežu unazad od sloja izlaza do sloja ulaza i mijenjanjem vrijednosti svih težina (i pristranosti) dobivamo algoritam propagacije unazad [10].

Prije nego što započnemo sa objašnjenjem propagacije unazad, prvo trebamo razumjeti što je to lančano pravilo infinitezimalnog računa (eng. calculus). Pomoću lančanog pravila možemo izračunati derivacije kompozicija funkcija na sljedeći način:

- Ako uzmemo u obzir da je: $z = f(y), \quad y = g(x)$
- Onda ćemo prema lančanome pravilu moći dobiti derivaciju:

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} \cdot \frac{\partial y}{\partial x} \tag{2.5}$$

Propagacija unazad koristi upravo to lančano pravilo kako bi se izračunao gradijent funkcije greške. Da bi bolje razumjeli kako propagacija unazad radi krećemo od posljednjeg sloja neurona (sloj izlaza). Cilj nam je izračunati gradijent funkcije greške, a njega ćemo dobiti tako da izračunamo sve parcijalne derivacije greške u odnosu na težine i pristranosti.

Radi lakšeg razumijevanja oznaku L ćemo dati za trenutni sloj, a j za indeks perceptrona u tome sloju, dok će n označavati broj perceptrona (n_L označava broj perceptrona sloja L). Prvo što se trebamo prisjetiti je način rada jednog neurona (perceptrona). U slučaju višeslojne mreže perceptron uzima sumu svih izlaza $a_k^{(L-1)}$ iz prethodnih slojeva pomnoženih sa njihovim dodijeljenim težinama $w_{jk}^{(L)}$ i pribraja pristranost $b_j^{(L)}$ [11]. Taj zbroj ćemo označiti na sljedeći način:

$$z_j^{(L)} = \sum_{k=1}^{n_{L-1}} w_{jk}^{(L)} a_k^{(L-1)} + b_j^{(L)} \tag{2.6}$$

Izlaz perceptrona ćemo dobiti primjenom aktivacijske funkcije f na navedeni zbroj:

$$a_j^{(L)} = f\left(z_j^{(L)}\right), \tag{2.7}$$

a grešku ćemo izračunati pomoću sljedeće formule:

$$E = \sum_{j=1}^{n_L} \left(a_j^{(L)} - y_j\right)^2. \tag{2.8}$$

Pošto znamo formule za izlaz perceptrona i za izračun greške, trebat će nam parcijalne derivacije greške u odnosu na težine (2.9) i pristranosti (2.10) trenutnog sloja te izlaze perceptrona prethodnog sloja (2.11):

$$\frac{\partial E}{\partial w_{jk}^{(L)}} = \frac{\partial z_j^{(L)}}{\partial w_{jk}^{(L)}} \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \frac{\partial E}{\partial a_j^{(L)}}, \tag{2.9}$$

$$\frac{\partial E}{\partial b_j^{(L)}} = \frac{\partial z_j^{(L)}}{\partial b_j^{(L)}} \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \frac{\partial E}{\partial a_j^{(L)}}\,, \tag{2.10}$$

$$\frac{\partial E}{\partial a_k^{(L-1)}} = \sum_{j=1}^{n_L} \frac{\partial z_j^{(L)}}{\partial a_k^{(L-1)}} \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \frac{\partial E}{\partial a_j^{(L)}} \tag{2.11}$$

Vidimo da je parcijalna derivacija greške u odnosu na izlaz perceptrona prethodnog sloja suma (jednadžba 2.11). Razlog je u tome što ta izlazna vrijednost utječe na sve perceptrone trenutnog sloja. Iz prikazanih formula parcijalnih derivacija dobiti ćemo sljedeće:

$$\nabla E \leftarrow \begin{cases} \dfrac{\partial E}{\partial w_{jk}^{(L)}} = a_k^{(L-1)} f'\left(z_j^{(L)}\right) \dfrac{\partial E}{\partial a_j^{(L)}} \\ \dfrac{\partial E}{\partial a_j^{(L)}} = \displaystyle\sum_{j=1}^{n_{L+1}} w_{jk}^{(L+1)} f'\left(z_j^{(L+1)}\right) \dfrac{\partial E}{\partial a_j^{(L+1)}} \\ \quad ili\ za\ sloj\ izlaza: \\ \dfrac{\partial E}{\partial a_j^{(L)}} = 2\left(a_j^{(L)} - y_j\right) \end{cases} \tag{2.12}$$

Iz jednadžbe iznad (2.12) možemo vidjeti kako lagano doći do gradijenta greške i kako dobiti vrijednosti za svaki sloj. Iako su jednadžbe dosta kompliciranе na prvi pogled, jednadžbe se uglavnom ponavljaju i vidimo da koriste princip lančanog pravila kod parcijalnih derivacija. Pošto smo sada objasnili sve bitne stvari vezane za višeslojne neuronske mreže, možemo krenuti na objašnjenje konvolucijskih neuronskih mreža.

3. Konvolucijske neuronske mreže (CNN)

3.1. Kratki uvod

Znanstvenici su oduvijek bili zainteresirani za razvoj računalnih programa koji bi mogli prepoznavati i klasificirati slike. Međutim, dugo vremena to nije bilo moguće jer su računala bila dosta loša u prepoznavanju slika jer ne mogu prepoznati što slika prikazuje. Tek s razvojem umjetnih neuronskih mreža, ovo područje doživjelo je značajan napredak. Nakon što su Hubel i Wiesel 1959. godine otkrili da životinje pomoću stanica u vidnome korteksu prepoznaju svjetlo unutar receptivnih polja, Kunihiko je u svome radu 1980. godine predstavio „neocognitron“ koji bi se mogao uzeti kao prethodnik konvolucijskim neuronskim mrežama [12].

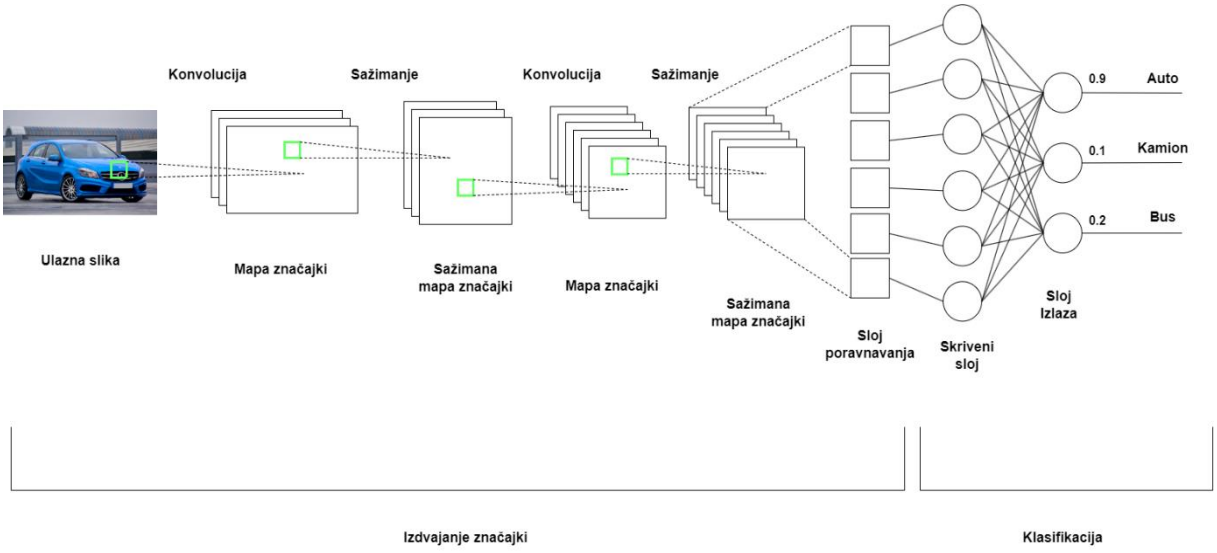
Skoro 10 godina kasnije su Yan LeCun i njegovi suradnici u svome radu objavili razvojni okvir (eng. framework) konvolucijske neuronske mreže i kasnije su ga poboljšali [13]. Zatim su napravili jednu od prvih pravih konvolucijskih neuronskih mreža ***LeNet-5*** kojoj je svrha bila klasifikacija brojeva napisanih rukom na čekovima i koristile su ju banke.

Konvolucijske neuronske mreže (eng. Convolutional Neural Networks, CNN) su višeslojne neuronske mreže napravljene za rad sa podacima strukturiranim u obliku dvodimenzionalnih polja ili mreže (eng. grid). Znamo da je jedan od najpoznatijih primjera takvih podataka slika, prema tome konvolucijske neuronske mreže imaju najširu primjenu kod klasifikacije odnosno prepoznavanja slika, međutim nešto o drugim primjenama ćemo reći kasnije. Što se slika tiče, one za razliku od drugih podataka takvog tipa imaju jedno dosta dobro svojstvo koje im omogućava da se pomiču i okreću, a da sustav (u ovom slučaju konvolucijska neuronska mreža) daje isti rezultat neovisno o tome. Takvo svojstvo se zove translacijska invarijantnost.

U narednim poglavljima detaljno će biti objašnjen princip rada konvolucijske neuronske mreže i njezina struktura, uključujući opis svakog tipa sloja koji je čini. Također, razmotriti ćemo postupak učenja ove vrste neuronske mreže, a na kraju poglavlja prikazat ćemo dodatne primjere primjene konvolucijskih neuronskih mreža.

3.2. Struktura konvolucijske neuronske mreže

Kao što smo već rekli Konvolcuijske neuronske mreže su građene od više različitih tipova slojeva, te možemo reći da je to jedna vrsta višeslojnih neuronskih mreža. Slično kao i kod takvih neuronskih mreža, konvolucijske mreže imaju sloj ulaza i sloj izlaza. Međutim sloj ulaza prima slike kao podatke, dok sloj izlaza klasificira slike. Na ostalim slojevima se konvolucijske mreže najviše razlikuju od uobičajenih višeslojnih mreža. Samu strukturu i slojeve možemo vidjeti na slici br. 4.



Slika 4: Struktura Konvolucijske neuronske mreže (vlastita izrada)

Također iz slike možemo vidjeti proces rada konvolucijske mreže. Prvo kao ulaz se dobije slika te se iz nje pomoću više slojeva konvolucije i sažimanja izdvoje značajke. Naravno moguće je napraviti konvolucijsku mrežu koja ima samo jedan sloj konvolucije i sažimanja, ali je puno efikasnije i bolje u praksi imati više takvih slojeva. Nakon što su značajke izdvojene, one prolaze kroz sloj poravnavanja koji pretvara značajke u jednodimenzionalno polje kako bi konvolucijska mreža mogla raditi klasifikaciju nad njima. Tu vidimo slojeve koji su slični normalnim višeslojnim neuronskim mrežama jer rade isti posao, a to je klasifikacija odnosno odlučivanje. Iako su na slici prikazana samo 2 sloja, konvolucijska neuronska mreža ima 3 tipa nove vrste slojeva, a to su: konvolucija, sažimanje i aktivacijska funkcija ReLU.

.

Kod konvolucijskih mreža stanja unutar slojeva su postavljena pomoću prostornih struktura mreže, a svi ti prostorni odnosi se nasljeđuju od jednog sloja prema drugome jer svaka vrijednost značajke ovisi o malome prostoru na prethodnome sloju [14]. Također treba održavati te odnose pošto konvolucija i transformacija ovise o njima. Bitno je naznačiti da je svaki sloj trodimenzionalna struktura mreže, a te dimenzije su visina, širina i dubina (nije isto kao dubina neuronske mreže). Dubina se ovdje odnosi na broj kanala boja unutar slike kao ulaza ili na broj mapa značajki u drugim slojevima.

Prve dvije dimenzije predstavljaju prostorne odnose dok treća dimenzija označava nezavisna svojstva. U ulaznom sloju konvolcuijske mreže ta nezavisna svojstva se odnose na intenzitete boja (crveno, zeleno, plavo), dok se kod ostalih skrivenih slojeva odnose na svakakve razne oblike i značajke koje su izvađene iz lokalnih dijelova slike. Kod ulaznog sloja dimenzije visine i širine uzimaju broj piksela slike, dok je dubina poprimila vrijednost 3 zbog već navedena 3 različita kanala boja. Međutim vrijednost dubine je puno veća kod skrivenih slojeva jer broj nezavisnih svojstava koji su bitni kod klasifikacije može biti znatno velik. Ubuduće radi lakšeg razumijevanja koristit ćemo iduće oznake:

- visina – L, širina – B, dubina – d, broj sloja – q
- veličina ulazne vrijednosti q-tog sloja – $L_q \times B_q \times d_q$

3.2.1. Operacija konvolucije

Za razumijevanje operacije konvolucije potrebno nam je znati kako filter radi. Filter (eng. kernel) je skup parametara u obliku trodimenzionalnih strukturnih jedinica te je dosta manji naspram sloja na kojem je primijenjen [14]. Što se dubine filtera tiče, ona uvijek mora biti ista kao i dubina sloja. U većini slučajeva filter je u obliku nekakvog kvadrata prema prostornim dimenzijama. Zato dimenzije filtera za q-ti sloj možemo označiti ovako:

$$F_q \times F_q \times d_q$$

Za dimenziju F_q se obično uzima nekakav mali broj koji je neparan.

Tijekom operacije konvolucije filter se postavlja na svaki mogući dio sloja s time da se potpuno preklapa sa slojem i vrši skalarni umnožak vektora između parametara $F_q \times F_q \times d_q$ u filtru i odgovarajućem dijelu sloja iste veličine [14]. Kako u operaciji konvolucije filter prolazi kroz cijeli sloj na kojemu je primijenjen, nas zanima koliko mogućih pozicija ima taj filter pošto svaka pozicija računa piksel odnosno značajku za idući sloj konvolucijske mreže. Što bi značilo da nam broj tih pozicija filtra nad slojem određuje visinu i širinu za idući sloj. Ako uzmemo u obzir nekakav q-ti sloj dobiti ćemo sljedeće jednadžbe:

- visina idućeg sloja – $L_{q+1} = L_q - F_q + 1$
- širina idućeg sloja – $B_{q+1} = B_q - F_q + 1$

Ovo nam govori da je ukupni broj mogućih skalarnih umnožaka vektora jednak $L_{q+1} \times B_{q+1}$, a to nam također definira veličinu sljedećeg sloja. Preostaje nam jedino saznati koja je dubina idućeg sloja. Dubinu određujemo prema broju filtera primijenjenih na prethodnome sloju, te dobivene značajke primjenom svih korištenih filtera zovemo mapom značajki [14]. Ako imamo 5 filtera primijenjenih na trenutnom sloju to znači da ćemo imati dubinu $d_{q+1} = 5$ za idući sloj. Također, svaki sloj može imati drugačiji broj filtera i prema tome imati drugačije dubine. Glavni smisao iza svega toga je da svaki filter prepozna nekakav prostorni uzorak i zbog toga kada imamo mnogo filtera, možemo uzeti mnogo raznih oblika koje kombiniramo da bi napravili finalnu sliku [14]. U sljedećem poglavlju upoznat ćemo se s nekoliko najpoznatijih filtera.

3.2.1.1. Filteri za pronalazak rubova

Sobel filter

Široko korišten u obradama slika, računa aproksimaciju gradijenta za funkciju intenziteta slike. Za dimenzije 3×3 filteri izgledaju ovako:

$$T_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad T_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$ [15].

T_x predstavlja filtar za vertikalne rubove, a T_y filtar za horizontalne rubove.

Prewitt filter

Radi po istom principu kao i Sobel filter, ali ima malo drugačije vrijednosti filtera. Možemo vidjeti ispod filtere dimenzija 3×3:

$$T_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \quad T_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$ [16].

Isto kao i kod Sobel filtera, T_x se koristi za vertikalne rubove, a T_y za horizontalne.

Laplacian filter

Laplacian filter je također korišten za pronalazak rubova, te aproksimira drugu derivaciju koja daje rubnu veličinu. Obično se koristi jedan od ova dva filtera za dimenzije 3×3:

$$T = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \; ili \; T = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -8 & 1 \\ 0 & 1 & 0 \end{bmatrix}.$$

Mala greška koju Laplacian filter može uzrokovati je da katkad na neke rubove dvaput reagira.

3.2.1.2. Filteri za izoštravanje

Ovi filteri služe za povećavanje oštrine slike tako da rubovi i detalji slike budu bolje definirani. Osim toga ti filteri smanjuju šum slike. Jedan od filtera za izoštravanje je Unmask filter. On radi na način da se prvo napravi zamućivanje (izglađivanje) slike, te se oduzimanjem zamućene slike od originalne slike dobiju samo izoštreni dijelovi slike. Zatim se ti dijelovi kao maska stave na originalnu i dobije se izoštrena slika.

3.2.1.3. Filteri za zamućivanje (izglađivanje)

Svrha filtera za zamućivanje je postići suprotno od filtera za izoštravanje, odnosno oni brišu detalje iz slike ali im je i također cilj smanjiti šum slike.

Guassov filter

Linearni filter koji radi prema Gaussovoj distribuciji, te određuje težinu svih susjednih piksela [17]. Primjer Gaussovog filtera dimenzije 5×5 je prikazan na sljedeći način:

$$T = \frac{1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix} \qquad [17].$$

Box filter (prosječni filter)

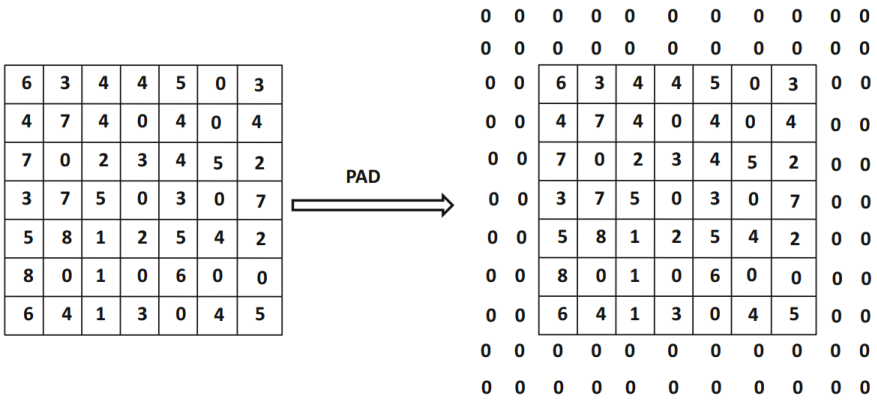
Linearni filter koji uprosječuje vrijednosti u susjedstvu oko svakog piksela, primjer tog filtera za dimenzije 3×3 izgleda ovako:

$$T = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}.$$

Postoji još mnogo vrsta filtera kod konvolucije međutim nećemo ih prolaziti. Uostalom pošto su to konvolucijske neuronske mreže, u praksi se također vidi velika primjena filtera koji se sami uče.

3.2.2. Nadopunjavanje

Možemo primijetiti da će visina L_{q+1} i širina B_{q+1} za idući sloj biti manje od visine i širine trenutnog sloja nakon operacije konvolucije. Takvo smanjenje u veličini nam predstavlja problem jer se na rubovima mape značajki (odnosno slike za prvi sloj) gube dio informacija. Nadopunjavanjem (eng. padding) možemo riješiti takav problem. Trenutni sloj nadopunimo na sve strane sa $(F_q - 1)/2$ piksela koji imaju vrijednost 0 [14]. Zašto baš toliko piksela? To će nam pomoći da nakon konvolucije idući sloj (mapa značajki) bude iste veličine kao i prethodni sloj bez nadopunjavanja. Međutim zbog samog nadopunjavanja prethodnom sloju će se visina i širina povećati za $F_q - 1$ piksela. Pošto su vrijednosti tih piksela 0 one nikako ne pridonose skalarnom umnošku vektora. Ovakav tip nadopunjavanja zove se polu-nadopunjavanje (eng. half-padding) jer kada je filter postavljen na rubnim dijelovima sloja, skoro pola filtera izlazi van. Primjer polu-nadopunjavanja za filter dimenzija 5×5 odnosno $F_q = 5$ prikazan je na slici broj 5.



Slika 5: Prikaz polu-nadopunjavanja. Na lijevoj strani je prikazana mapa značajki (ili slika), a na desnoj je ta mapa značajki nadopunjena nulama u svim smjerovima. Preuzeto sa [14].

Cilj polu-nadopunjavanja je da se zadrži prostorna vrijednost, tj. da se ne izgube informacije kod konvolucije. Postoje i druge vrste nadopunjavanja, jednu vrstu smo već prošli, a to je ne nadopunjavanje (drugi naziv je važeće nadopunjavanje, eng. valid padding). Kao što smo već rekli ako se ne napravi nadopunjavanje, onda imamo gubitak informacija na rubovima. Još jednu vrstu nadopunjavanja koju ćemo proći je potpuno nadopunjavanje (eng. full-padding). Kao što naziv govori mi kod ovog nadopunjavanja dopuštamo filteru da skoro potpuno izađe izvan kod svih rubova sloja. Prema tome sloj prije konvolucije je nadopunjen sa $F_q - 1$ piksela (kojima je vrijednost 0) sa svake strane što znači da su širina i visina tog sloja

povećane za $2(F_q - 1)$ [14]. Nakon operacije konvolucije idući sloj poprima sljedeću visinu i širinu:

$$L_{q+1} = L_q + F_q - 1, \tag{3.1}$$

$$B_{q+1} = B_q + F_q - 1. \tag{3.2}$$

Vidimo da ovakav tip nadopunjavanja ustvari poveća širinu i visinu idućeg sloja nakon provedene konvolucije. Pomoću tog potpunog tipa nadopunjavanja može se napraviti obrnuta konvolucija ako se uz prikladno definirani filter istih dimenzija primjeni na sloj na kojemu je već normalna konvolucija bila primijenjena [14]. Obrnute konvolucije mogu se vidjeti u primjeni kod propagacije unazad i kod autoenkodera za konvolucijske mreže.

3.2.3. Korak

Postoji i drugi način smanjivanja prostorne veličine sloja osim konvolucije svake pozicije. Taj način postižemo pomoću koraka (eng. stride). Prvo dodijelimo neku vrijednost za korak S_q i pomoću te vrijednosti možemo smanjiti granularnost konvolucije tako da izvršavamo konvoluciju na lokacijama $1, S_q + 1, 2S_q + 1$ ... po širini i visini sloja. Prema [14] idući sloj nakon konvolucije sa koracima ima sljedeću visinu i širinu:

$$L_{q+1} = (L_q - F_q) / S_q + 1, \tag{3.3}$$

$$B_{q+1} = (B_q - F_q) / S_q + 1\,. \tag{3.4}$$

Koristeći korake obje prostorne dimenzije idućeg sloja otprilike će se smanjiti za faktor od S_q, a prostor za S_q^2. Ako nam je $S_q = 1$ dobiti ćemo isti efekt kao da smo konvoluciju proveli na svakoj poziciji jer prolazimo kroz sve vrijednosti po visini i širini. U mnogim slučajevima koriste se koraci veličina $S_q = 1$ i $S_q = 2$. Glavna prednost koraka je to šta mogu povećati receptivno polje svake značajke i istovremeno smanjiti veličinu cijeloga sloja.

3.2.4. Postavke operacije konvolucije

Uobičajeno je da slike koje unosimo budu u obliku kvadrata, odnosno da njihova širina i visina budu iste, ako se ne koriste takve slike onda se u pretprocesiranju dolazi do tog svojstva. Za filtere koriste se veličine $F_q = 3$ ili $F_q = 5$ jer manji filteri obično daju bolje rezultate i dolazi do dubljih konvolucijskih mreža. Što se tiče broja filtera u svakome sloju, bilo bi poželjno da se koristi 2^n filtera jer dobivamo dvije prednosti od toga, prva je bolje procesiranje, a druga su dubine koje će također bit potencija od 2 [14].

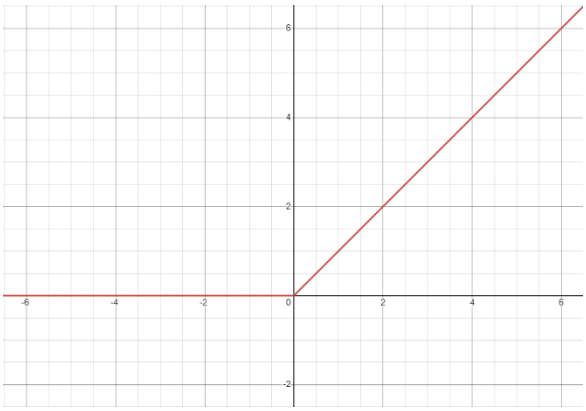
Dosad nismo spomenuli pristranost u konvolucijskim mrežama, međutim nju nije teško nadodati. Pristranost dodajemo svakome filteru na svim slojevima pa prema tome ako imamo filter k u q-tom sloju, onda taj filter ima pristranost $b^{(k,q)}$. Ta pristranost se nadodaje skalarnom umnošku vektora kada se provodi konvolucija sa tim slojem i filterom.

3.2.5. ReLU aktivacijska funkcija

ReLU (eng. rectified linear unit) je aktivacijska funkcija kao i sigmoidna funkcija koju smo već objasnili. Za razliku od sigmoidne funkcije ReLU se pokazala kao bolja aktivacijska funkcija kod konvolucijske neuronske mreže. Princip rada ReLU kod konvolucijske mreže je skoro isti kao i na normalnim višeslojnim mrežama, te se primjenjuje nakon operacije konvolucije. Prema [18] ReLU funkcija je definirana na sljedeći način:

$$f(x) = \max(x, 0) \tag{3.5}$$

Funkcija ReLU radi na način da za sve negativne vrijednosti x poprima vrijednost 0 dok je za sve pozitivne vrijednosti x ista vrijednost, odnosno kod pozitivnih vrijednosti funkcija je linearna. Graf funkcije prikazan je na slici 6.



Slika 6: graf ReLU aktivacijske funkcije (vlastita izrada).

Kod konvolucijskih mreža ReLU funkcija se provodi nakon svake operacije konvolucije na sloju q za svaku vrijednost $L_q \times B_q \times d_q$. Prema tome kakva je ReLU funkcija, računala ju mogu dosta efikasno računati. Također je prikazano da primjena ReLU funkcije ima velike prednosti u brzini i preciznosti naspram drugih aktivacijskih funkcija [19].

3.2.6. Sažimanje (eng. pooling)

Naspram operacije konvolucije, operacija sažimanja je dosta drugačija jer radi pomoću male mreže dimenzija $P_q \times P_q$ na q-tom sloju i stvara sloj $q + 1$ koji ima istu dubinu kao i sloj q

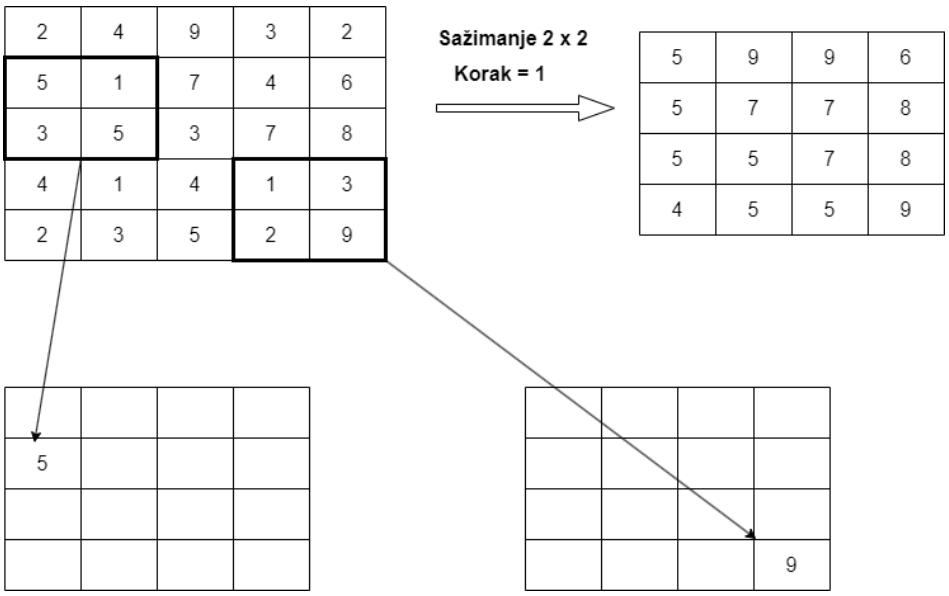
dok kod filtera to nije tako [14]. Najveća razlika je u tome što operacija konvolucije filterom prolazi kroz sve mape značajki istodobno jer filter ima dubinu d_q, dok se operacija sažimanja vrši na svakoj mapi značajki zasebno. Zbog toga se nakon sažimanja ne povećava dubina d_q odnosno broj mapi značajki.

Sažimanje za svaku malu mrežu $P_q \times P_q$ nad svakom mapom značajki d_q vraća maksimalnu vrijednost i to zovemo maksimalno sažimanje (eng. max-pooling). To je ujedno i najkorišteniji oblik sažimanja. Postoje i druga sažimanja ali se jako rijetko koriste. Kao i kod konvolucije, kod sažimanja se također koriste koraci. Razlika je u tome što kod sažimanja uobičajeno je koristiti korake veličine $S_q > 1$. Prema tome visina i širina novonastalih mapi značajki za sloj $q + 1$ je:

$$L_{q+1} = \left(L_q - P_q\right)/S_q + 1, \tag{3.6}$$

$$B_{q+1} = \left(B_q - P_q\right)/S_q + 1 \tag{3.7}$$

Iz jednadžbi 3.6 i 3.7 možemo vidjeti da sažimanje dosta smanjuje visinu i širinu svake mape značajki, ali zato povećava receptivno polje u isto vrijeme. Ta povećanja u receptivnim poljima nam pomažu da se uhvate veće regije slika unutar kompleksnih značajki u kasnijim slojevima [14]. Uobičajeno je koristiti korake veličine 2 i filter dimenzija 2×2, ako se želi smanjiti prostorna veličina svake mape značajki. Slika 7. nam prikazuje operaciju sažimanja na određenoj mapi značajki.



Slika 7: Operacija sažimanja napravljena na mapi značajki 5×5, veličina filtera je 2×2, a veličina koraka 1. Dobiveni rezultat je mapa značajki veličine 4×4, te se također se vide 2 koraka sažimanja (vlastita izrada).

3.2.7. Potpuno povezani slojevi

Potpuno povezane slojeve možemo vidjeti na slici 4 na desnome kraju konvolucijske neuronske mreže. Oni rade na isti način kao i kod normalnih višeslojnih mreža. Svaka značajka od prostornih slojeva konvolucijske mreže povezana je sa svakim stanjem prvoga potpuno povezanog sloja [14]. Uobičajeno je koristiti više potpuno povezanih slojeva na kraju tako da se dobiju bolje odluke i rezultati. Zanimljiva stvar je da konvolucijski slojevi mreže iako zauzimaju više prostora imaju manje parametara, dok potpuno povezani slojevi imaju puno više parametara zbog ogromne količine veza između tih slojeva.

Izlazni sloj je obično dizajniran ovisno o primjeni konvolucijske neuronske mreže. Gledajući za svrhu klasifikacije izlazni sloj je povezan sa svim neuronima iz predzadnjeg sloja. Također postoji alternativa za korištenje potpuno povezanih mreža, a to je da se koristi jedan tip sažimanja (prosječno sažimanje) preko cijelog prostora posljednjih mapa značajki koji će dati jednu vrijednost [14]. Zbog toga bi broj značajki u posljednjem prostornom sloju bio jednak broju filtera. Takva alternativa smanjuje broj parametara potpuno povezanih mreža, te nam daje neke prednosti za generalizaciju.

3.2.8. Povezivanje slojeva mreže

Kao što smo prikazali, kod konvolucijske neuronske mreže imamo 3 tipa slojeva: konvolucijski slojevi, slojevi sažimanja i ReLU slojevi (ne računajući potpuno povezane slojeve na kraju). ReLU aktivacijski slojevi obično idu odmah nakon konvolucijskih slojeva i prema tome su ta dva sloja većinom jedan uz drugoga. Uzevši u obzir ta dva sloja kao par, nakon nekoliko parova tih slojeva imamo sloj sažimanja. Za razliku od konvolucijskih i ReLU slojeva, broj slojeva sažimanja je dosta manji jer nam je dovoljno malo operacija sažimanja kako bi se prostor mapa značajki znatno smanjio. Redoslijed povezanih slojeva je većinom ovakav:

$$Conv \rightarrow ReLU \rightarrow Conv \rightarrow ReLU \rightarrow Pool$$

$$Conv \rightarrow ReLU \rightarrow Conv \rightarrow ReLU \rightarrow Conv \rightarrow ReLU \rightarrow Pool \quad [14].$$

Iznad vidimo primjer povezivanja gdje nakon dva ili tri para konvolucijskog sloja ($Conv$) i ReLU sloja ($ReLU$) imamo jedan sloj sažimanja ($Pool$). Ako takve uzorke ponovimo nekoliko puta i dodamo im nekoliko potpuno povezanih slojeva (FCL) dobit ćemo sljedeću konvolucijsku neuronsku mrežu:

$$Conv \rightarrow ReLU \rightarrow Conv \rightarrow ReLU \rightarrow Pool \rightarrow Conv \rightarrow ReLU \rightarrow Conv \rightarrow ReLU \rightarrow Pool \rightarrow Conv \rightarrow ReLU \rightarrow Conv \rightarrow ReLU \rightarrow Pool \rightarrow FCL \rightarrow FCL \rightarrow FCL\ .$$

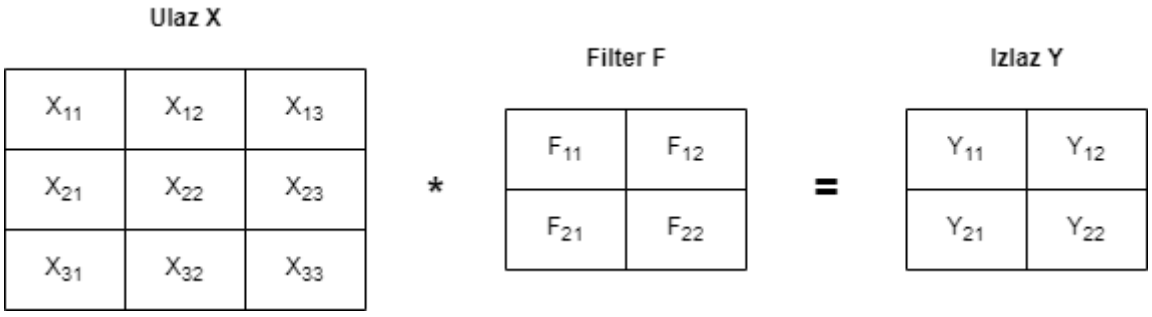
Naravno takav prikaz konvolucijske mreže nije upotpunjen sa brojem, veličinama i nadopunjavanjem kod filtera i sažimanja, ali nam zato otprilike točno prikazuje kakav je redoslijed slojeva i koliko slojeva se nalazi u konvolucijskim mrežama, te kolika je dubina konvolucijske mreže. Za razliku od današnjih konvolucijskih mreža početne mreže su bile dosta plitke (manje slojeva) što je i za očekivati pošto su tadašnja računala bila znatno slabija, međutim ako uspoređujemo osnovne principe rada, oni su otprilike ostali jednaki. Jedna od većih razlika je to što su tadašnje konvolucijske mreže koristile sigmoidnu aktivacijsku funkciju, a ne ReLU funkciju.

3.3. Učenje konvolucijske neuronske mreže

Konvolucijske neuronske mreže se kao i druge višeslojne mreže uče i treniraju pomoću propagacije unazad. Pošto kod propagacije unazad krećemo od kraja konvolucijske mreže, prvi slojevi će nam biti potpuno povezani slojevi. Propagaciju unazad za potpuno povezane slojeve smo već prošli u poglavlju 2.2.2. pa nema potrebe za ponavljanjem. Što se ReLU aktivacijskog sloja tiče on je isti kao i kod normalnih višeslojnih neuronskih mreža pa ga je dosta jednostavno propagirati. Bitno nam je objasniti propagaciju unazad kod konvolucijskog sloja i kod sloja sažimanja.

3.3.1. Propagacija unazad kod konvolucije

Radi lakšeg objašnjenja propagacije unazad kod konvolucije koristit će se ulaz sa dubinom $d = 1$ i koraci veličine $S = 1$. Ulaznu sliku (ili mapu značajki) označit ćemo sa X, filter sa F i izlaznu mapu značajki sa Y. Operacija konvolucije je prikazana na sljedeći način:



Slika 8: Operacija konvolucije (vlastita izrada)

Uzeli smo mapu značajki X (ulaz) veličine 3×3, filtar F veličine 2×2 i pošto nemamo nikakve korake niti nadopunjavanja dobili smo mapu značajki Y (izlaz) veličine 2×2.

Prema [20] operaciju konvolucije možemo prikazati i na sljedeći način:

$$\begin{bmatrix} Y_{11} & Y_{12} \\ Y_{21} & Y_{22} \end{bmatrix} = konvolucija \left(\begin{bmatrix} X_{11} & X_{12} & X_{13} \\ X_{21} & X_{22} & X_{23} \\ X_{31} & X_{32} & X_{33} \end{bmatrix}, \begin{bmatrix} F_{11} & F_{12} \\ F_{21} & F_{22} \end{bmatrix} \right). \tag{3.8}$$

Gledajući operaciju konvolucije vidimo da utjecaj na izlaznu mapu značajki ima ulazna mapa značajki i filter. Prema tome trebamo izračunati parcijalnu derivaciju greške u odnosu na filter i parcijalnu derivaciju greške u odnosu na ulaznu mapu značajki [20]. Uzevši u obzir lančano pravilo, dobijemo sljedeće parcijalne derivacije:

$$\frac{\partial E}{\partial F_i} = \sum_{k=1}^{n} \frac{\partial E}{\partial Y_k} \frac{\partial Y_k}{\partial F_i}, \qquad \frac{\partial E}{\partial X_i} = \sum_{k=1}^{n} \frac{\partial E}{\partial Y_k} \frac{\partial Y_k}{\partial X_i}. \tag{3.9}$$

Ako primijenimo ove parcijalne derivacije za svaku od vrijednosti filtera F i za svaku od vrijednosti ulaza X dobivamo idući niz jednadžbi:

$$za\ svaki\ F_i \begin{cases} \frac{\partial E}{\partial F_{11}} = \frac{\partial E}{\partial Y_{11}} \cdot X_{11} + \frac{\partial E}{\partial Y_{12}} \cdot X_{12} + \frac{\partial E}{\partial Y_{21}} \cdot X_{21} + \frac{\partial E}{\partial Y_{22}} \cdot X_{22} \\ \frac{\partial E}{\partial F_{12}} = \frac{\partial E}{\partial Y_{11}} \cdot X_{12} + \frac{\partial E}{\partial Y_{12}} \cdot X_{13} + \frac{\partial E}{\partial Y_{21}} \cdot X_{22} + \frac{\partial E}{\partial Y_{22}} \cdot X_{23} \\ \frac{\partial E}{\partial F_{21}} = \frac{\partial E}{\partial Y_{11}} \cdot X_{21} + \frac{\partial E}{\partial Y_{12}} \cdot X_{22} + \frac{\partial E}{\partial Y_{21}} \cdot X_{31} + \frac{\partial E}{\partial Y_{22}} \cdot X_{32} \\ \frac{\partial E}{\partial F_{22}} = \frac{\partial E}{\partial Y_{11}} \cdot X_{22} + \frac{\partial E}{\partial Y_{12}} \cdot X_{23} + \frac{\partial E}{\partial Y_{21}} \cdot X_{32} + \frac{\partial E}{\partial Y_{22}} \cdot X_{33} \end{cases}, \tag{3.10}$$

$$za\ svaki\ X_i \begin{cases} \frac{\partial E}{\partial X_{11}} = \frac{\partial E}{\partial Y_{11}} \cdot F_{11} \\ \frac{\partial E}{\partial X_{12}} = \frac{\partial E}{\partial Y_{11}} \cdot F_{12} + \frac{\partial E}{\partial Y_{12}} \cdot F_{11} \\ \frac{\partial E}{\partial X_{13}} = \frac{\partial E}{\partial Y_{12}} \cdot F_{12} \\ \frac{\partial E}{\partial X_{21}} = \frac{\partial E}{\partial Y_{11}} \cdot F_{21} + \frac{\partial E}{\partial Y_{21}} \cdot F_{11} \\ \frac{\partial E}{\partial X_{22}} = \frac{\partial E}{\partial Y_{11}} \cdot F_{22} + \frac{\partial E}{\partial Y_{12}} \cdot F_{21} + \frac{\partial E}{\partial Y_{21}} \cdot F_{12} + \frac{\partial E}{\partial Y_{22}} \cdot F_{11} \\ \frac{\partial E}{\partial X_{23}} = \frac{\partial E}{\partial Y_{12}} \cdot F_{22} + \frac{\partial E}{\partial Y_{22}} \cdot F_{12} \\ \frac{\partial E}{\partial X_{31}} = \frac{\partial E}{\partial Y_{21}} \cdot F_{21} \\ \frac{\partial E}{\partial X_{32}} = \frac{\partial E}{\partial Y_{21}} \cdot F_{22} + \frac{\partial E}{\partial Y_{22}} \cdot F_{21} \\ \frac{\partial E}{\partial X_{33}} = \frac{\partial E}{\partial Y_{22}} \cdot F_{22} \end{cases}. \tag{3.11}$$

Prema [20] dobivene parcijalne derivacije za svaku vrijednost filtera F (jednadžbe 3.10) mogu se prikazati kao konvolucija ulaza X i gradijenta greške $\frac{\partial E}{\partial Y}$:

$$\begin{bmatrix} \frac{\partial E}{\partial F_{11}} & \frac{\partial E}{\partial F_{12}} \\ \frac{\partial E}{\partial F_{21}} & \frac{\partial E}{\partial F_{22}} \end{bmatrix} = konvolucija \left(\begin{bmatrix} X_{11} & X_{12} & X_{13} \\ X_{21} & X_{22} & X_{23} \\ X_{31} & X_{32} & X_{33} \end{bmatrix}, \begin{bmatrix} \frac{\partial E}{\partial Y_{11}} & \frac{\partial E}{\partial Y_{12}} \\ \frac{\partial E}{\partial Y_{21}} & \frac{\partial E}{\partial Y_{22}} \end{bmatrix} \right). \tag{3.12}$$

Što se parcijalnih derivacija za svaku vrijednost ulaza X (jednadžbe 3.11) tiče, njih dobijemo tako da obrnemo filter F vertikalno i horizontalno, zatim napravimo konvoluciju sa obrnutim filterom F i potpuno nadopunjenim gradijentom greške $\frac{\partial E}{\partial Y}$. Konvoluciju gdje imamo potpuno nadopunjenu matricu na koju se primjenjuje filter možemo još zvati potpunom konvolucijom (eng. full convolution). Parcijalne derivacije svake vrijednosti ulaza X su prema tome prikazane ovako:

$$\begin{bmatrix} \frac{\partial E}{\partial X_{11}} & \frac{\partial E}{\partial X_{12}} & \frac{\partial E}{\partial X_{13}} \\ \frac{\partial E}{\partial X_{21}} & \frac{\partial E}{\partial X_{22}} & \frac{\partial E}{\partial X_{23}} \\ \frac{\partial E}{\partial X_{31}} & \frac{\partial E}{\partial X_{32}} & \frac{\partial E}{\partial X_{33}} \end{bmatrix} = potpuna\ konvolucija \left(\begin{bmatrix} \frac{\partial E}{\partial Y_{11}} & \frac{\partial E}{\partial Y_{12}} \\ \frac{\partial E}{\partial Y_{21}} & \frac{\partial E}{\partial Y_{22}} \end{bmatrix}, \begin{bmatrix} F_{22} & F_{21} \\ F_{12} & F_{11} \end{bmatrix} \right). \tag{3.13}$$

Kao zaključak dobivamo jednadžbe za parcijalne derivacije greške u odnosu na filter F (3.14) i u odnosu na ulaznu mapu značajki X (3.15):

$$\frac{\partial E}{\partial F} = konvolucija \left(X, \frac{\partial E}{\partial Y} \right), \tag{3.14}$$

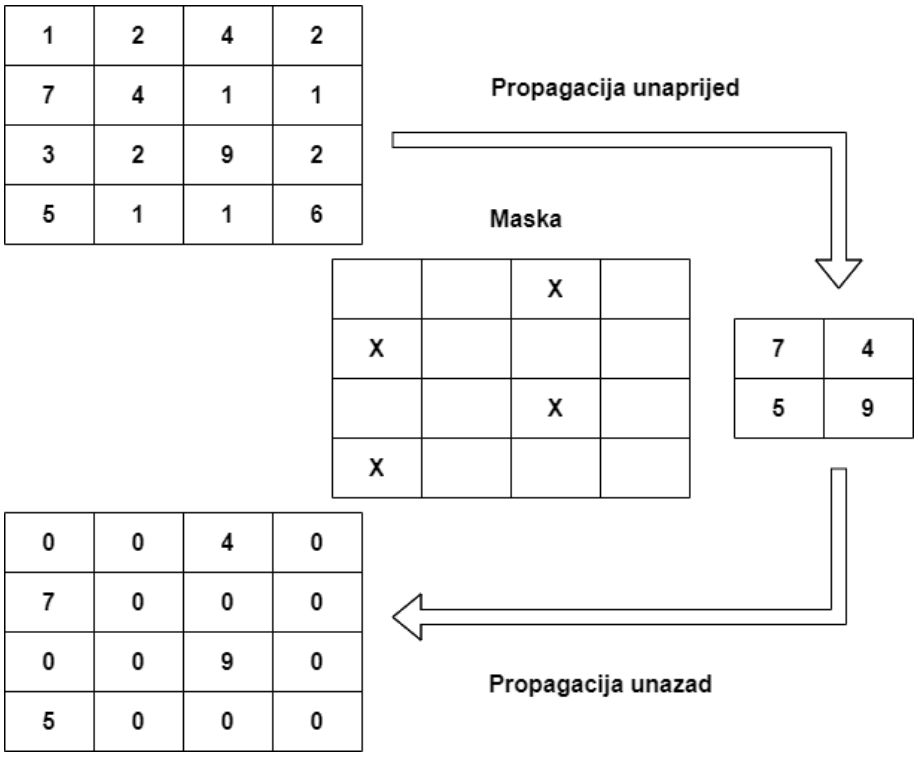
$$\frac{\partial E}{\partial X} = potpuna\ konvolucija \left(\frac{\partial E}{\partial Y}, obrnuti\ F \right). \tag{3.15}$$

Vidimo da su oba izračuna ustvari operacije konvolucije i zbog toga su računalima poprilično efikasni za riješiti.

3.3.2. Propagacija unazad kod sažimanja

Kod sažimanja propagacija unazad ne računa nikakve parcijalne derivacije, nego se prosljeđuje gradijent. Glavno pitanje je kako proslijediti gradijent. Gledajući način rada sažimanja bilo kakvo pomicanje kod vrijednosti koje nisu maksimalne neće utjecati na izlazne vrijednosti prema tome se njima dodjeljuje gradijent 0. Pošto se kod sažimanja prenose maksimalne vrijednosti, znači da one imaju utjecaj, te njima postavljamo gradijent 1 [21], [22]. Imamo jedan problem kod toga, a to je da u koraku kod propagacije unazad ne znamo gdje su

bile maksimalne vrijednosti postavljene u ulaznom sloju prije sažimanja. Taj problem možemo riješiti na način da kod propagacije unaprijed tijekom sažimanja spremimo masku koja ima naznačene pozicije maksimalnih vrijednosti. Prema toj maski onda dodjeljujemo gradijente. Na slici broj 9 prikazan je proces propagacije unazad kod sažimanja.



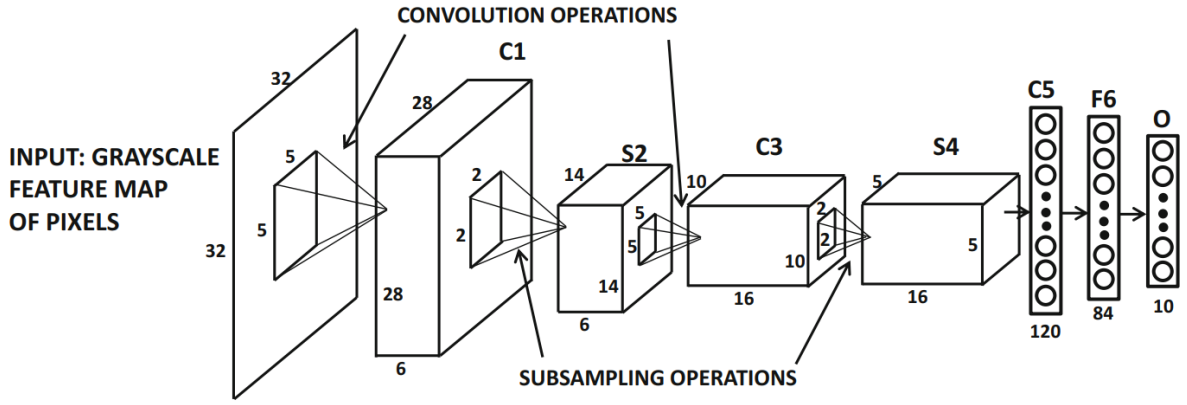
Slika 9: Propagacija unazad kod sažimanja. Tijekom propagacije unaprijed stvara se maska, koja se koristi za propagaciju unazad kako bi se mogle maksimalne vrijednosti vratiti nazad na svoje pozicije (vlastita izrada).

4. Primjeri slučajeva konvolucijske neuronske mreže

U ovome poglavlju proći ćemo kroz nekoliko studija slučajeva konvolucijskih neuronskih mreža koje su bile vrlo uspješne. Ti primjeri će nam pružit bolji uvid i razumijevanje o konvolucijskim mrežama i njihovom dizajnu arhitekture, te zašto takve konvolucijske mreže rade jako dobro.

4.1. LeNet-5

Jedan od najranijih primjera konvolucijske neuronske mreže je bio LeNet-5. Izradili su ga 1998. godine Yan LeCun i njegovi suradnici te se ta konvolucijska mreža smatra kao jako bitan i veliki utjecaj na današnje konvolucijske mreže. LeNet-5 je za ulazne podatke koristio slike sive nijanse i samo jedan kanal boja, što je značilo da su ulazni podaci bili dvodimenzionalni jer su imali samo visinu i širinu. Ta mreža se sastojala od 2 sloja konvolucije, 2 sloja sažimanja i 3 potpuno povezana sloja. U dubljim slojevima mreža je imala više mapi značajki zbog toga što se koristilo više filtera na svim slojevima. Arhitekturu mreže LeNet-5 možemo vidjeti na slici 10.



Slika 10: Arhitektura konvolucijske mreže LeNet-5. Preuzeto sa [14].

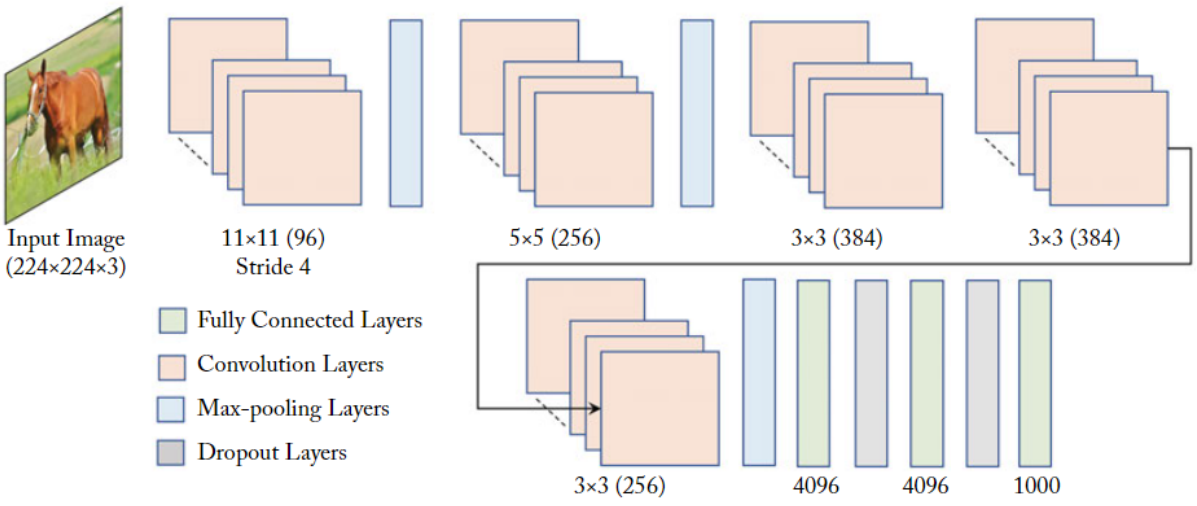
Zanimljiva činjenica je da se u njihovome originalnome radu za prvi potpuno povezani sloj C-5 govorilo da je to konvolucijski sloj jer je postojala mogućnost da se generalizira za prostorne značajke kod većih ulaznih slika. Međutim u njihovoj implementaciji LeNet-5 je koristio taj sloj kao potpuno povezani sloj jer je prostorna veličina filtera bila jednaka prostornoj veličini ulazne slike [14]. Osim toga u originalnome radu se aktivacijska funkcija izvršava tek nakon poduzorkovanja (eng. subsampling). To nam se može činiti malo neuobičajeno za

današnje konvolucijske mreže jer je poduzorkovanje danas zamijenjeno sa sažimanjem te se slojevi sažimanja dosta rjeđe pojavljuju za razliku od tih slojeva poduzorkovanja. Nadalje aktivacijske funkcije se danas većinom koriste nakon slojeva konvolucije.

Kod poduzorkovanja su se koristila prostorna područja veličine 2×2 sa korakom veličine 2. Za razliku od sažimanja tražile su se prosječne vrijednosti, zatim su se skalirale sa težinama i bila je dodana pristranost [14]. Današnje arhitekture su većinom izbacile skaliranje i pristranost. Ova konvolucijska mreža je dosta plitka i zastarjela, ali su principi ostali slični, te je doprinijela jako puno za razvoj konvolucijskih neuronskih mreža.

4.2. AlexNet

AlexNet su 2012. godine predložili Alex i njegovi suradnici [19] te su iste godine na natjecanju ImageNet 2012. osvojili prvo mjesto. To je bio prvi jako veliki model konvolucijske neuronske mreže koji je doveo do ponovnog oživljavanja konvolucijskih neuronskih mreža. Za razliku od AlexNet-a drugi tadašnji modeli nisu bili toliko veliki niti su bili trenirani na jako velikom skupu podataka kao što je bio skup podataka za ImageNet natjecanje [23]. AlexNet je također imao dosta dublju mrežu koja mu je omogućavala da se puno više parametara izmjenjuje. AlexNet konvolucijska mreža se sastoji od 8 slojeva ukupno, od kojih je 5 slojeva konvolucije, a posljednja 3 sloja su potpuno povezani slojevi. Slika 11 nam prikazuje kako je AlexNet mreža građena.



Slika 11: Arhitektura konvolucijske mreže AlexNet. Preuzeto sa [23].

Na slici se mogu vidjeti i drugi slojevi poput slojeva sažimanja i slojeva izbacivanja (eng. dropout). Ako pribrojimo te slojeve vidimo da mreža ima više od 8 slojeva, međutim običaj kod konvolucijskih mreža je da se samo broje slojevi koji imaju prostorne filtere sa težinama, i potpuno povezani slojevi. Prema tome se slojevi sažimanja i drugi takvi slojevi ne broje. Na slici možemo osim toga vidjeti i dimenzije svakog sloja. AlexNet mreža po prvi put koristi ReLU aktivacijsku funkciju iako je ta ReLU funkcija bila prvi put predložena dosta prije AlexNet mreže, ona se tek počela koristiti kod te mreže [24].

U AlexNet-u se koristi maksimalno sažimanje sa preklapanjem kako bi zamijenilo tadašnje popularno prosječno sažimanje i poduzorkovanje. Takvo sažimanje je obogatilo značajke i izbjeglo zamućivanje rezultata koje je uzrokovalo prosječno sažimanje. Slojevi izbacivanja se koriste nakon prva dva potpuno povezana sloja kako bi se smanjilo pretreniranje (eng. overfitting) i kako bi bolja generalizacija bila primijenjena na nove primjere slika. Posljednji potpuno povezani sloj je klasificirao slike u jednu od 1000 klasa iz ImageNet skupa podataka.

Normalizacija lokalnog odgovora (LRN) je prvi put bila predložena u AlexNet mreži kako bi neurone sa malim vrijednostima prigušila, dok bi neurone sa velikim vrijednostima ostavila aktivnima [24]. Prema tome LRN služi za poboljšanje generalizacije kod mreža. AlexNet mreža koristi i dvije metode augmentacije podataka. Jednom metodom izvlači dijelove veličine 224×224 piksela iz slika veličine 256×256 piksela i njihove horizontalne refleksije kako bi se povećao broj podataka za treniranje konvolucijske mreže. Za drugu metodu koristi analizu glavnih komponenti (PCA) kako bi se promijenile vrijednosti boja RGB nad skupom slika za treniranje [24].

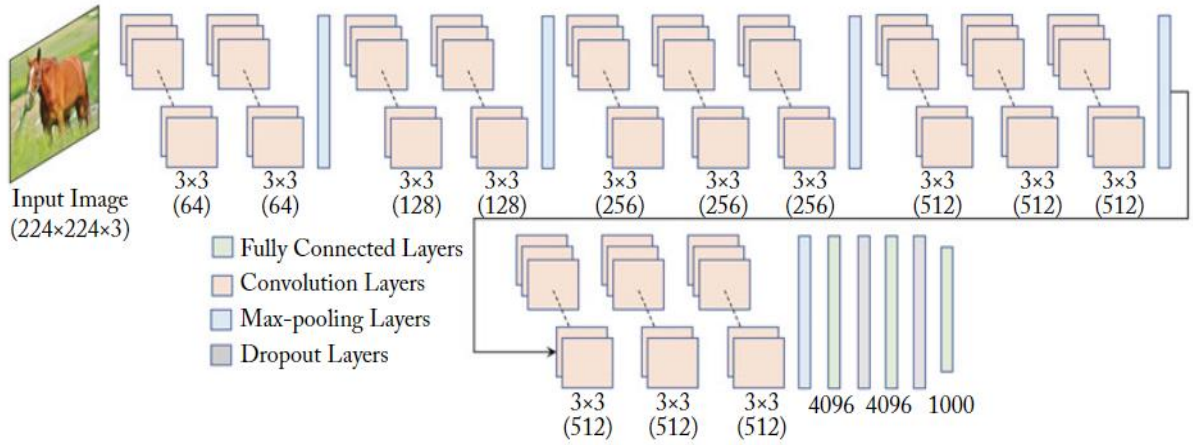
Kod svoje prve implementacije AlexNet mreže, treniranje se trebalo odvojiti na dvije grafičke procesorske jedinice (GPU) jer se na jednoj nije mogla postaviti cijela neuronska mreža pošto je imala oko 62 milijuna parametara [23]. Samo treniranje je trajalo oko 6 dana jer se skup podataka od ImageNet-a sastojao od oko 1.2 milijuna slika. AlexNet konvolucijska mreža je pridonijela nove metode i tehnologije te je prikazala svoj uspjeh nad ogromnim skupom podataka.

4.3. VGGnet

Od svog samog početka (2014.g,) VGGnet arhitekture konvolucijskih mreža su postale popularne, iako nisu osvojile prvo mjesto na ILSVRC (ImageNet) natjecanju tadašnje godine. Razlog njihove popularnosti je korištenje malih filtera kod konvolucije koje dovode do vrlo duboke mreže, a uz to VGGnet arhitekture imaju i jednostavnu građu. Te arhitekture je

predstavila VGG grupa (eng. Visual Geometry Group) sa Oxforda [24]. Autori su prikazali skup arhitektura od kojih su dva skupa bila uspješnija od drugih, a to su VGGnet-16 i VGGnet-19 arhitekture konvolucijskih mreža. Njihove arhitekture koriste filtere veličine 3×3 kod slojeva konvolucije pošto filteri veličine 3×3 i 5×5 imaju receptivna polja iste veličine, ali zato imaju manji broj parametara za oko 45 posto [24].

Čim se koriste manji filteri može se više slojeva posložiti što rezultira povećanjem dubine mreže. Glavna ideja iza toga je da korištenjem dubljih mreža dobivamo bolje učenje značajki koje nam omogućava bolje klasificiranje slika. Dublje mreže također stvaraju više nelinearnosti zbog toga što koriste više ReLU aktivacijskih funkcija. Kod VGGnet arhitektura normalizacija lokalnog odgovora je bila izbačena pošto autori nisu smatrali da njihov efekt na duboke konvolucijske mreže ima vidljiv utjecaj [24]. Još jedna stvar na koju su se autori odlučili je da nakon svakog maksimalnog sažimanja broj filtera povećaju za faktor 2. To omogućuje nekakvu razinu balansa kod računalnog napora i neke od novijih arhitektura konvolucijskih mreža su naslijedile to svojstvo. Arhitekturu VGGnet-16 mreže i dimenzije njezinih slojeva možemo vidjeti na slici broj 12.



Slika 12: Arhitektura VGGnet-16 mreže. Preuzeto sa [23].

Slično kao i kod AlexNet mreže VGGnet-16 mreža ima slojeve izbacivanja nakon prva dva potpuno povezana sloja. Glavna prednost ove mreže je njezina dubina i manji broj parametara koje omogućavaju filteri dimenzija 3×3.

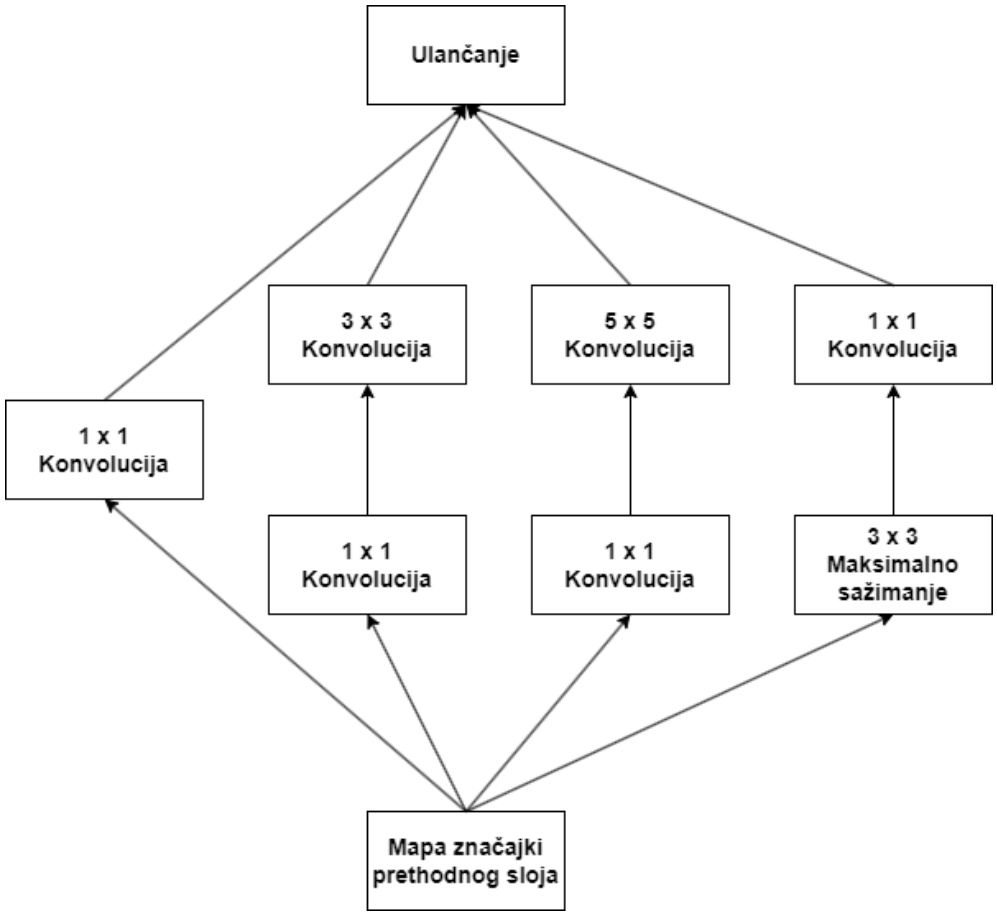
4.4. GoogleNet

GoogleNet konvolucijska mreža se također pojavila 2014. godine. na ILSVRC natjecanju i osvojila je prvo mjesto. Drugi naziv pod kojim je poznata GoogleNet mreža je Inception V1. Kod prije navedenih arhitektura vidimo slijednu arhitekturu samo sa jednim putem unutar kojeg su naslagane razne vrste slojeva kako bi sačinile konvolucijsku mrežu. GoogleNet arhitektura nam predstavlja alternativu za slijedni tip arhitekture implementirajući novi koncept zvan modul začetka (eng. inception module). Procesiranje tog modula se izvodi paralelno [23].

Ideja iza modula začetka je u tome što su informacije slika dostupne na različitim razinama detalja, na primjer, ako koristimo veći filter kod konvolucije moći ćemo uzeti veće informacije sa manje varijacija, dok kod malih filtera ćemo imati detaljnije informacije unutar manjeg područja. Jedan od problema je u tome što ne znamo unaprijed koja razina detalja je potrebna za svaki dio slike. Zbog toga modul začetka koristi paralelno postavljene filtere konvolucije veličina 1×1, 3×3 i 5×5. Ako ulančamo te sve pojedinačne značajke duž dimenzije dubine, dobiti ćemo značajke sa prevelikim brojem dimenzija. GoogleNet je riješio taj problem tako da moduli začetka izvršavaju smanjenje dimenzije prije nego što se proslijedi ulazna slika filterima veličina 3×3 i 5×5 [23]. Smanjenje dimenzije postiže se konvolucijom sa filterom veličine 1×1. Te konvolucije se još zovu operacije uskog grla (eng. bottleneck operations). Smanjenje dimenzije pomoću filtera 1×1 omogućuje povećanu efikasnost kod većih konvolucija. Osim toga filteri 1×1 nam također pomažu i kod smanjenja dubine nakon slojeva sažimanja.

Kod GoogleNet konvolucijske mreže, moduli začetka su naslagani jedan iza drugoga te mreža ukupno ima 22 sloja. Znamo da je uobičajen pristup kod konvolucijskih mreža korištenje nekoliko potpuno povezanih slojeva na kraju mreže. Međutim GoogleNet mreža koristi drugačiji pristup, ona koristi prosječno sažimanje preko cijelog prostora od posljednjih mapi značajki nakon kojeg slijedi jedan potpuno povezani sloj. Sloj prosječnog sažimanja omogućava mreži brže izračune sa boljim točnošću kod klasifikacije i puno manji broj parametara. Iako GoogleNet konvolucijska mreža izgleda znatno kompleksnije nego ostale mreže, ako gledamo broj parametara GoogleNet ima negdje oko 6 milijuna parametara za razliku od AlexNet mreže sa 62 milijuna parametara i VGGnet mreže sa 138 milijuna parametara što je znatno manje [23]. Glavni cilj GoogleNet mreže bio je povećanje točnosti sa

smanjenjem cijene izračuna, osim toga vidimo što znači dobar dizajn arhitekture konvolucijske mreže. Modul začetka do kojeg je sačinjena GoogleNet mreža prikazan je na slici broj 13.



Slika 13: Modul začetka unutar GoogleNet mreže (vlastita izrada).

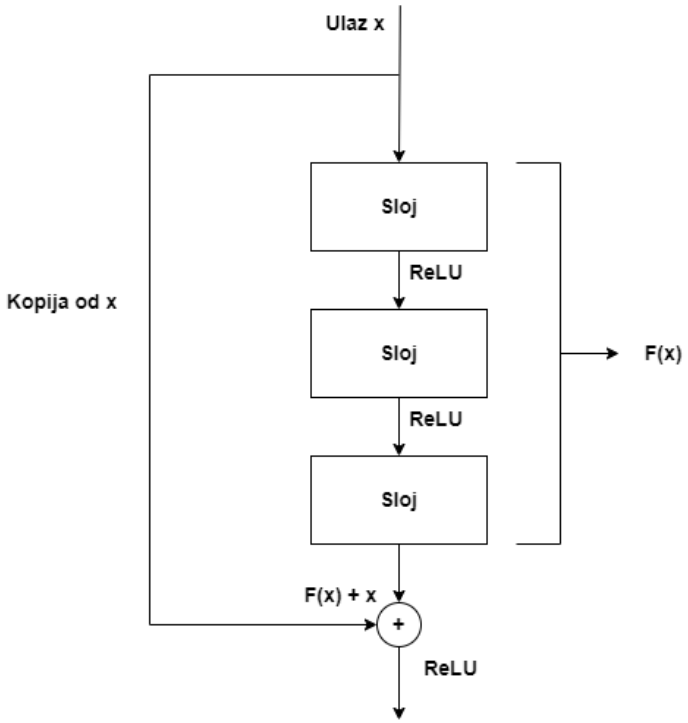
4.5. ResNet

ResNet konvolucijska mreža nastala je 2015. godine te je također pobijedila na ILSVRC natjecanju iste godine. ResNet mrežu su izradili Kaiming He i njegovi suradnici jer su htjeli primijeniti novi način učenja kod dubokih mreža sa mnogo slojeva [25]. ResNet mreža koristi 152 sloja što je znatno više nego što ostale navedene mreže imaju. Treniranje toliko duboke mreže tada nije bilo moguće bez nekakvih znatnih inovacija. Glavni problem kod treniranja takvih mreža bio je u tome što se ne dobiva efikasna konvergencija zbog kompleksnosti površina greške odnosno gubitka i zbog nedovoljnog napretka u procesu optimizacije [14].

Ako gledamo dotadašnje konvolucijske mreže, one imaju uvijek fiksan broj slojeva za klasifikaciju slika. Možemo to gledati na način da ako imamo sliku koja ima dosta kompleksne značajke i jednostavne značajke, kod kompleksnijih značajki konvolucijskoj mreži bi trebalo

znatno više slojeva da se prepoznaju i nauče te značajke dok bi kod jednostavnih značajki trebalo znatno manje slojeva. Prema tome ako imamo konvolucijsku mrežu s velikim brojem slojeva konvergencija će biti jako spora zbog učenja velikog broja značajki koje se mogu učiti sa znatno manjim brojem slojeva [14]. Postavlja se pitanje zašto onda ne damo konvolucijskoj mreži da sama odluči koliko joj je slojeva potrebno da se nauči svako svojstvo slike.

ResNet konvolucijska mreža nam pruža odgovor na to pitanje koristeći rezidualne blokove (od tuda joj i naziv ResNet odnosno Residual Network). Rezidualni blokovi omogućuju mreži da koristi veze za preskakanje slojeva. Gledajući prijašnje konvolucijske mreže, one imaju veze između slojeva k i $k+1$. Za razliku od njih ResNet mreža ima veze između slojeva k i $k+r$ za $r > 1$ [14]. Takav pristup pomoću rezidualnih blokova omogućava da se uzme ulaz od sloja k i kopira te nadoda izlazu sloja $k+r$. Princip rada rezidualnog bloka prikazan je na slici broj 14.



Slika 14: Princip rada rezidualnog bloka sa vrijednosti $r = 3$ (vlastita izrada).

Na slici možemo vidjeti vezu koja kopira ulaz x i preskače 3 sloja mreže te pribraja kopirani ulaz x sa izlaznim rezultatom od preskočena 3 sloja. Rezidualni blokovi omogućavaju efektivan tok gradijenta jer algoritam propagacije unazad može propagirati gradijente unazad koristeći veze za preskakanje. Cijela ResNet mreža se sastoji se od mnogo naslaganih rezidualnih blokova. Također se koristi određen tip nadopunjavanja i korak od 1 kako se ne bi

mijenjala prostoran veličina ulaza od sloja do sloja. Radi toga se ulaz sloja k može pribrojiti izlazu sloja $k + r$. Međutim postoje slojevi koji koriste veće korake kako bi smanjili prostorne dimenzije mapi značajki za faktor 2. Kod takvog slučaja se ne mogu koristiti kopije ulaza sloja kod veze za preskakanje, nego se treba koristiti linearna matrica projekcije kako bi se prilagodile dimenzije. Matrica projekcije koristi skup 1×1 konvolucija sa korakom 2 kako bi se prostorna veličina smanjila za faktor od 2 [14].

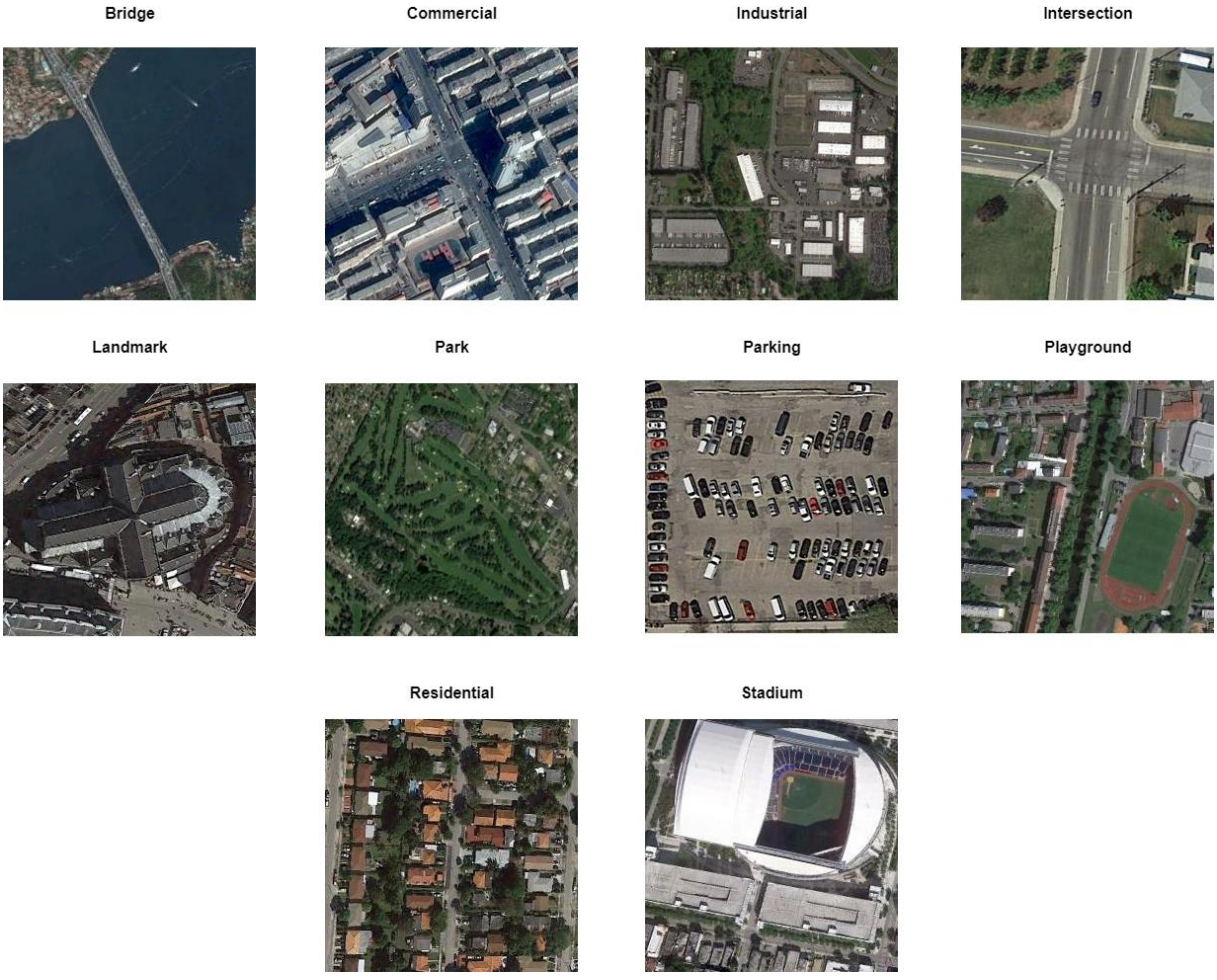
Kod originalne implementacije ResNet mreže jednom se definira veličina skoka (vrijednost r) i rade se veze za preskakanje između slojeva k i $k + r$, međutim buduće nadogradnje ResNet mreže koriste u sebi veze za preskakanje između svakakvih mogućih parova slojeva. ResNet mreže su pokazale manju računalnu kompleksnost u usporedbi sa drugim navedenim mrežama [26]. Njihova primjena je mnogo pridonijela i pokazala kako je bitno imati mogućnost varijabilne dubine.

5. Implementacija konvolucijske neuronske mreže

U ovome poglavlju ćemo implementirati konvolucijsku neuronsku mrežu nad izabranim skupom podataka. Ova implementacija će nam pomoći lakše razumjeti rad konvolucijskih mreža i pokazati kako se one treniraju i testiraju.

5.1. Skup podataka

Za skup podataka nad kojim će se vršiti klasifikacija odabran je skup slika gradova iz zraka. Skup podataka je preuzet sa najveće platforme za podatkovnu znanost Kaggle te se može preuzeti ovdje [27]. Taj skup podataka je sastavljen od slika koje su uzete iz druga dva javno dostupna skupa podataka AID [28] i NWPU-Resisc45 [29]. Skup se sastoji od 8000 slika koje su klasificirane u 10 klasa podijeljenih prema različitim tipovima objekata u gradovima.



Slika 15: Prikaz slika i njihovih klasa (vlastita izrada).

Klase skupa podataka su sljedeće: most (eng. Bridge), poslovne zgrade (eng. Commercial), industrijsko područje (eng. Industrial), raskrižje (eng. Intersection), znamenitost (eng. Landmark), park, parking, igralište (eng. Playground), stambeno područje (eng. Residential) i stadion (eng. Stadium). Primjer slike za svaku klasu se može vidjeti na slici broj 15. Klase ćemo ostaviti na engleskom jeziku pošto su tako preuzete. Za svaku klasu odabrano je 800 slika veličine 256×256 piksela. Pošto nam trebaju slike za učenje i slike za provjeru bit odvojene napravljena je mala promjena. Od svake klase je uzeto 200 slika (2000 ukupno) i stavljenu u mapu za provjeru, a ostalih 6000 slika je stavljeno u mapu za treniranje.

5.2. Implementacija

Za implementaciju konvolucijske mreže korišten je Python programski jezik [30] i PyTorch biblioteka [31]. PyTorch biblioteka je dosta popularna biblioteka za strojno učenje te za rad sa neuronskim mrežama. Jedna od velikih prednosti te biblioteke je što ima dosta intuitivan način za implementaciju neuronskih mreža. Za razvojno okruženje je korišten alat Visual Studio Code [32]. Kako bi se ubrzao rad neuronskih mreža korišten je alat CUDA [33]. Biblioteka Matplotlib [34] je korištena za izradu grafova, a ostale slike u radu su napravljene pomoću alata draw.io [35] i Desmos [36]. Sada ćemo objasniti samu implementaciju konvolucijske mreže.

5.2.1. Priprema

Prvo što trebamo napraviti je uvesti sve biblioteke koje su nam potrebne za rad programa i za implementaciju konvolucijske mreže (prikazano na isječku koda broj 1).

```
import os
import torch
import torch.nn
import torch.nn.functional as func
from torchvision import datasets, transforms
from torch.utils.data import DataLoader
import matplotlib.pyplot as plt
import numpy as np
```

Isječak koda 1: Uvoz potrebnih biblioteka.

Nakon toga trebamo učitati skupove podataka za treniranje i za testiranje slika. Isječak koda broj 2 nam prikazuje spomenuto učitavanje.

```
#putanja do datoteke s podacima
notebookPutanja = os.path.abspath("convolutional_network.ipynb")
datasetPutanja = os.path.dirname(os.path.dirname(notebookPutanja)) +
("\\Cityscape_Dataset\\")


#učitavanje skupova podataka
setZaTreniranje = datasets.ImageFolder(root=datasetPutanja +
"Treniranje", transform=transforms.ToTensor())
setZaTestiranje = datasets.ImageFolder(root=datasetPutanja +
"Testiranje", transform=transforms.ToTensor())


#stvaranje utovarivača podataka
loaderZaTreniranje = DataLoader(dataset=setZaTreniranje, batch_size=50,
shuffle=True)
loaderZaTestiranje = DataLoader(dataset=setZaTestiranje, batch_size=50,
shuffle=True)
```

Isječak koda 2: Učitavanje skupova podataka.

Prvo što dohvaćamo je putanja od Python datoteke u kojoj implementiramo program (*notebookPutanja*). Zatim spremamo u varijablu *datasetPutanja* putanju do datoteke sa skupom podataka koju smo dobili manipulacijom *notebookPutanja* putanje i nadodavanjem naziva datoteke *Cityscape_Dataset*. Pomoću dobivene putanje dohvaćamo skupove podataka i spremamo ih u varijable *setZaTreniranje* (skup za treniranje) i *setZaTestiranje* (skup za testiranje). Metoda *ImageFolder* za parametre uzima putanju i potrebnu transformaciju. Posao te metode je učitavanje slika iz datoteka gdje su nazivi datoteka klase dodijeljene slikama, a pomoću transformacije se pretvaraju slike u tenzore (algebarske objekte) s kojima PyTorch može raditi.

Preostaje nam još stvaranje *DataLoadera* (utovarivača podataka) za treniranje (varijabla *loaderZaTreniranje*) i za testiranje (varijabla *loaderZaTestiranje*). *Dataloaderi* uzimaju napravljene skupove tenzora *setZaTreniranje* i *setZaTestiranje* i u hrpama (*batch_size*) ih utrpavaju u konvolucijsku mrežu. Postavili smo veličinu hrpa na 50, te smo također dopustili *Dataloaderima* da izmiješaju dobivene skupove tenzora. Slijedi nam izrada konvolucijske mreže.

5.2.2. Izrada konvolucijske mreže

Konvolucijske mreže u PyTorch-u rade se pomoću klasa na vrlo intuitivan i jednostavan način. Prvi korak je izrada klase, zatim treba inicijalizirati sve slojeve koje će konvolucijska mreža imati pomoću *init* metode. Nakon inicijaliziranih slojeva pomoću metode *forward*

povezuju se slojevi i stvara se struktura konvolucijske mreže. Samo ime *forward* naznačuje unaprijed u smislu propagacije unaprijed kroz mrežu pomoću koje konvolucijske mreže klasificiraju slike.

Konvolucijske mreže u PyTorch-u imaju mnogo više metoda ali za jednostavan princip rada potrebne su nam ove dvije. Isječak koda broj 3 nam prikazuje samu izradu konvolucijske mreže.

```
#izrada konvolucijske mreže

class konvMreza(torch.nn.Module):

    #inicijalizacija slojeva mreže

    def __init__(self):

        super().__init__()

        #slojevi konvolucije

        self.konvolucija1 = torch.nn.Conv2d(in_channels=3,
out_channels=9, kernel_size=3, stride=1, padding=1)

        self.konvolucija2 = torch.nn.Conv2d(in_channels=9,
out_channels=18, kernel_size=3, stride=1, padding=1)

        self.konvolucija3 = torch.nn.Conv2d(in_channels=18,
out_channels=36, kernel_size=3, stride=1, padding=1)



        #slojevi sažimanja

        self.sazimanje = torch.nn.MaxPool2d(kernel_size=2, stride=2,
padding=0)



        #potpuno povezani slojevi

        self.potpuni1 = torch.nn.Linear(in_features=36*32*32,
out_features=192)

        self.potpuni2 = torch.nn.Linear(in_features=192,
out_features=64)

        self.potpuni3 = torch.nn.Linear(in_features=64, out_features=10)
```

Isječak koda 3: Definiranje klase konvolucijske mreže i inicijalizacija slojeva.

Klasu naše konvolucijske mreže smo nazvali *konvMreza* i ona poprima parametar modula za neuronske mreže. Inicijalizirali smo 3 sloja 2D konvolucije. Pitanje je zašto baš 2D konvolucija ako slike imaju 3 kanala boja te im je treća dimenzija $d = 3$. U PyTorch biblioteci

2D konvolucija radi na način da se izvršava konvolucija po visini i širini što znači da će i filteru bit treća dimenzija $d = 3$ odnosno konvolucija se neće računati po trećoj dimenziji. Slojeve smo definirali na sljedeći način:

Konvolucija

- prvi sloj konvolucije (*konvolucija1*) – ulazna dubina $d_x = 3$, izlazna dubina $d_y = 9$, filter 3×3, korak $s = 1$, nadopunjavanje $p = 1$
- drugi sloj konvolucije (*konvolucija2*) – ulazna dubina $d_x = 9$, izlazna dubina $d_y = 18$, filter 3×3, korak $s = 1$, nadopunjavanje $p = 1$
- treći sloj konvolucije (*konvolucija3*) – ulazna dubina $d_x = 18$, izlazna dubina $d_y = 36$, filter 3×3, korak $s = 1$, nadopunjavanje $p = 1$

Sažimanje

- sloj maksimalnog sažimanja (*sazimanje*) – filter 2×2, korak $s = 2$, nadopunjavanje $p = 0$

Potpuno povezani slojevi

- prvi sloj (*potpuni1*) – broj ulaznih značajki (broj ulaznih perceptrona) je $36 \times 32 \times 32 = 36864$, a broj izlaznih značajki (broj izlaznih perceptrona) je 192
- drugi sloj (*potpuni2*) – broj ulaznih značajki je 192, a broj izlaznih značajki je 64
- treći ili posljednji sloj (*potpuni3*) – broj ulaznih značajki je 64, a broj izlaznih značajki je 10 jer ima toliko klasa za klasifikaciju.

Kod slojeva konvolucija koristimo nadopunjavanje $p = 1$, kako ne bi narušili rubne značajke slike i kako bi nam ostala ista prostorna veličina nakon konvolucije. Što se maksimalnog sažimanja tiče tu je prikazano kao da imamo jedan sloj, međutim koristit ćemo ga više puta jer nam nije potreban drugačiji sloj sažimanja. Razlog zašto sloj sažimanja ima navedenu veličinu filtera i koraka je u tome što s takvim maksimalnim sažimanjem mi smanjujemo prostorne dimenzije širine i visine upola kod mapi značajki. Idući isječak koda se nastavlja na izradu konvolucijske mreže te nam prikazuje definiranje redoslijeda slojeva.

```
#definiranje redoslijeda slojeva
def forward(self, podaci):
```

```
            podaci = func.relu(self.konvolucija1(podaci))
            podaci = self.sazimanje(podaci)
            podaci = func.relu(self.konvolucija2(podaci))
            podaci = self.sazimanje(podaci)
            podaci = func.relu(self.konvolucija3(podaci))
            podaci = self.sazimanje(podaci)
            podaci = torch.flatten(podaci, 1)
            podaci = func.relu(self.potpuni1(podaci))
            podaci = func.relu(self.potpuni2(podaci))
            rezultat = self.potpuni3(podaci)
            return rezultat


    #Postavljanje mreže na GPU
    procesorskaJedinica = torch.device("cuda" if torch.cuda.is_available()
else "cpu")
    konvolucijskaMreza = konvMreza().to(procesorskaJedinica)
```

Isječak koda 4: Definiranje redoslijeda slojeva.

Kao što smo već rekli pomoću metode *forward* se definira redoslijed slojeva mreže. Prema isječku koda 4 možemo vidjeti da se taj redoslijed definira tako da nad varijablom *x* koja nam predstavlja ulaznu mapu značajki izvršavamo metode koje smo definirali kao slojeve u isječku koda 3. Prema tome redoslijed slojeva naše konvolucijse mreže *konvMreza* izgleda ovako:

$$konvolucija1 \rightarrow ReLU \rightarrow sazimanje \rightarrow konvolucija2 \rightarrow ReLU \rightarrow sazimanje \rightarrow konvolucija3 \rightarrow ReLU \rightarrow sazimanje \rightarrow flatten \rightarrow potpuni1 \rightarrow ReLU \rightarrow potpuni2 \rightarrow ReLU \rightarrow potpuni3.$$

Iz redoslijeda se vidi da nakon svakog sloja konvolucije dolazi sloj aktivacije ReLU i sloj maksimalnog sažimanja. Sloj *flatten* nam služi kao sloj poravnavanja koji pretvara značajke u jednodimenzionalno polje radi prijenosa značajki na potpuno povezane slojeve. Nakon prva dva potpuno povezana sloja se također koristi aktivacijska funkcija ReLU. Što se zadnjeg potpuno povezanog sloja tiče, on klasificira slike u jednu od 10 klasa.

Nismo još zadnje dvije linije koda u isječku objasnili. Pomoću njih provjeravamo ima li grafička procesorska jedinca (GPU) paralelnu računalnu platformu zvanu CUDA koju je napravila NVIDIA [33] kako bi se poboljšale i ubrzale računalne aplikacije koristeći GPU za izvršavanje umjesto CPU-a. U suštini sa te dvije linije koda prebacujemo našu *konvMreza* mrežu na GPU radi boljeg i bržeg izvršavanja.

5.2.3. Treniranje konvolucijske mreže

Nakon implementirane konvolucijske mreže slijedi nam treniranje mreže nad skupom od 6000 slika. Prvi korak koji prije treniranja trebamo napraviti je definirati funkciju gubitka i gradijentni spust. To možemo vidjeti u isječku koda broj 5.

```
#funkcija gubitka/greške

funkcijaGubitka = torch.nn.CrossEntropyLoss()

#stohastički graadijent spusta

gradijent = torch.optim.SGD(params=konvolucijskaMreza.parameters(),
lr=0.001, momentum=0.9)
```

Isječak koda 5: Funkcija gubitka i gradijent spusta.

Za funkciju greške (*funkcijaGubitka*) koristi se funkcija gubitka unakrsne entropije (eng. cross-entropy loss function). Cilj te funkcije gubitka je pronaći optimalno rješenje tako da se težine računaju unutar neuronske mreže tijekom treniranja. Varijabla *gradijent* će sadržavati metodu stohastičkog gradijenta spusta. Kao prvi parametar metoda prima parametre naše konvolucijske mreže. Parametar *lr* naznačava stopu učenja (eng. learning rate), a parametar momentum se koristi zbog toga što PyTorch koristi implementaciju stohastičkog gradijenta spusta sa Nesterovim momentumom (prema dokumentaciji [37]) kojoj jednadžbe glase ovako:

$$v_{k+1} = \mu \cdot v_k + g_{k+1} \tag{5.1}$$

$$p_{k+1} = p_t - lr \cdot v_{k+1} \tag{5.2}$$

U jednadžbama 5.1 i 5.2 v predstavlja brzinu, p predstavlja parametre, g gradijent i μ naznačava momentum, k naznačuje indeks sloja. Slijedi nam implementacija treniranja odnosno učenja konvolucijske mreže, a to možemo vidjeti u isječku koda 6.

```
#treniranje konvolucijske mreže

epohe = 50
```

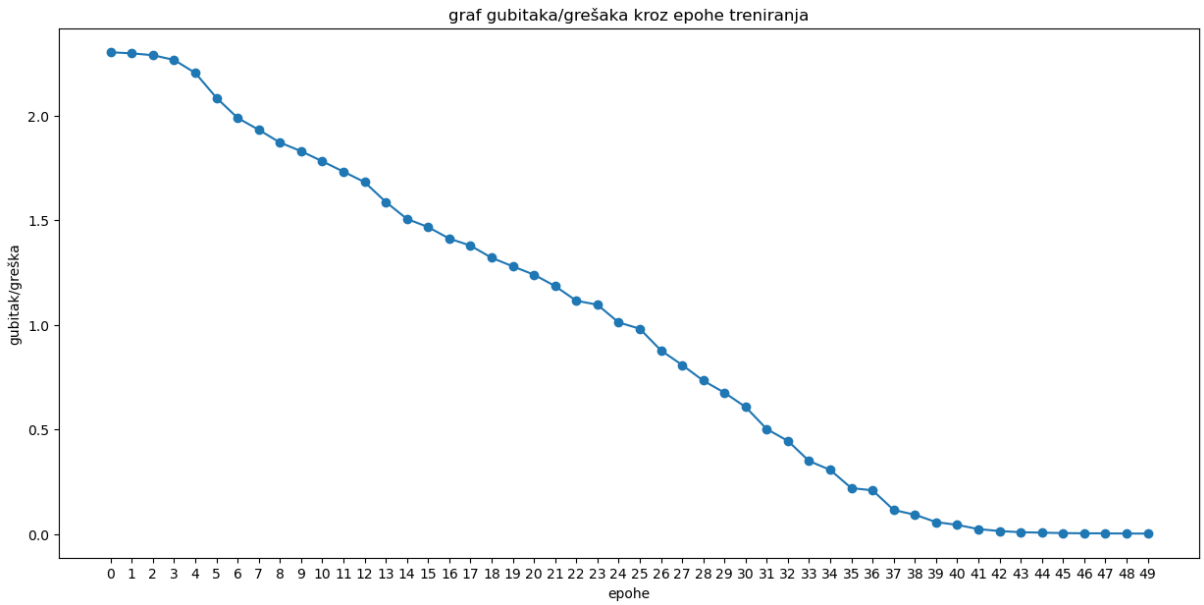
```
gubitci = []


#iteracije (epohe) treniranja mreže
for epoha in range(epohe):
    konvolucijskaMreza.train()


    privGubitak = 0


    for i, (ulazi, oznake) in enumerate(loaderZaTreniranje):
        ulazi, oznake = ulazi.cuda(), oznake.cuda()
        gradijent.zero_grad()
        izlazi = konvolucijskaMreza(ulazi)
        gubitak = funkcijaGubitka(izlazi, oznake)
        gubitak.backward()
        gradijent.step()
        privGubitak += gubitak.item()


    gubitci.append(privGubitak / len(loaderZaTreniranje))
    print(f"Epoha {epoha+1}: Gubitak: {privGubitak / 
len(loaderZaTreniranje):.5f}")
```

Isječak koda 6: treniranje implementirane konvolucijske mreže.

Prvi korak kod treniranja je odrediti broj epoha (eng. epoch). Epohe kod neuronskih mreža znače broj puta koliko sve ulazne slike prođu kroz algoritam, odnosno epoha bi značila jedan ciklus treniranja konvolucijske mreže. Mi smo postavili 50 epoha kako bi se dobro istrenirala konvolucijska mreža.

Da bi trenirali konvolucijsku mrežu iteriramo kroz tih 50 epoha gdje unosimo slike u hrpama od 50 (isječak koda 2) i računamo njihove izlazne vrijednosti (varijabla *izlazi*). Nakon toga određujemo vrijednost gubitka pomoću funkcije gubitka (definirana u isječku koda 5). dobivamo *gubitak* i pozivamo metodu *backward* koja izvršava propagaciju unazad. PyTorch ima u sebi implementiranu propagaciju unazad, pa ju ne moramo implementirati. Naša *gradijent* varijabla odnosno stohastički gradijent spusta izvršava korak pomoću metode *step*. Zatim spremamo gubitak u privremenu *privGubitak* varijablu koja se resetira tijekom početka svake epohe. Na kraju svake epohe spremamo vrijednosti gubitaka u jedno polje radi interpretacije.

Treniranje obično najduže traje zbog prevelikog broja operacija i izvršavanja. Treniranje ove implementirane konvolucijske mreže nad skupom od 6000 slika je trajalo 12 minuta i 7 sekundi iako je sve implementirano na GPU. Rezultat treniranja je na slici broj 16 prikazan kao graf grešaka u odnosu na broj epohe.



Slika 16: Graf gubitaka/grešaka u odnosu na epohe (vlastita izrada).

Na početku je iznos greške bio oko 2.3, te se smanjio nakon 50 epoha na 0.003 što je dosta dobro, ali naravno može biti bolje sa većim brojem epoha. Vidimo da se pred sami kraj (zadnjih 10-tak epoha) znatno smanjila razlika između vrijednosti grešaka, te je prema tome konvolucijska mreža najviše naučila od 5. do 35. epohe. Preostaje nam testiranje nad ostalim slikama.

5.3. Testiranje konvolucijske mreže

Prije samog testiranja spremili smo stanje istrenirane mreže, te smo učitali to stanje kako bi mogli započeti testiranjem. Taj postupak možemo vidjeti u isječku koda 7.

```
#spremanje stanja istrenirane mreže
torch.save(konvolucijskaMreza.state_dict(), "trained_cnn.pth")
konvolucijskaMreza = konvMreza().to(procesorskaJedinica)
#dohvaćanje stanja istrenirane mreže
```

```
konvolucijskaMreza.load_state_dict(torch.load("trained_cnn.pth"))
```

Isječak koda 7: Spremanje i dohvaćanje stanja istrenirane mreže.

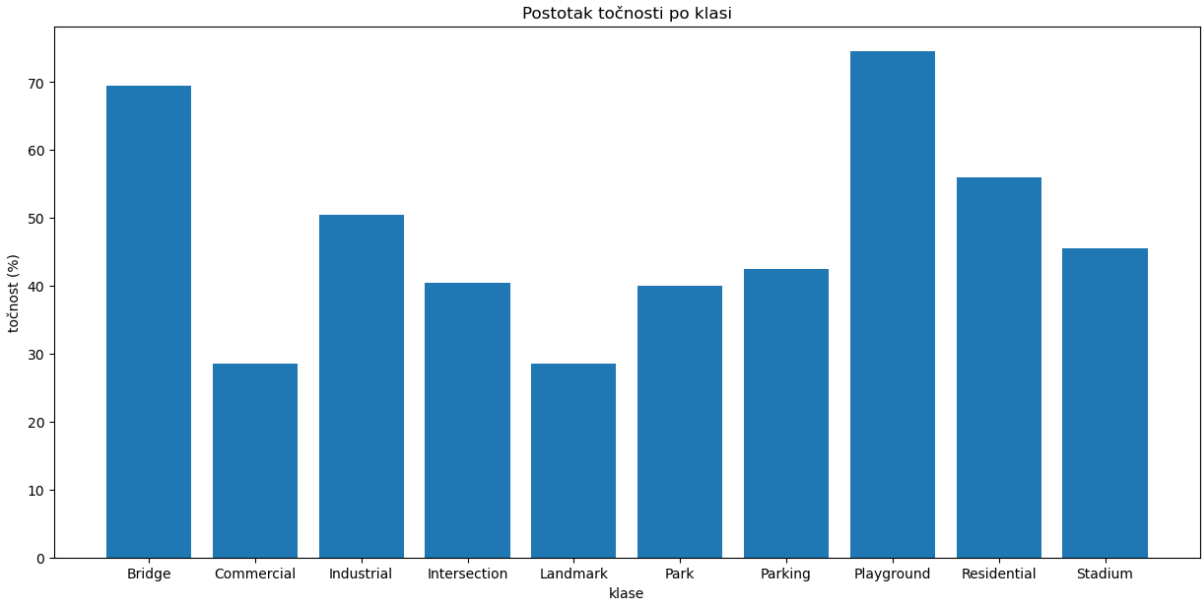
Testiranje će se izvršiti nad preostalim skupom sa 2000 slika. Slike će se unositi u neuronsku mrežu u hrpama od 50 isto kao i kod treniranja. Implementaciju testiranja možemo vidjeti u isječku koda 8.

```
#testiranje konvolucijske mreže
konvolucijskaMreza.eval()

svaPred = []
sveOznake = []
tocnaPred = torch.zeros(10)
totalPred = torch.zeros(10)

with torch.no_grad():
    for ulazi, oznake in loaderZaTestiranje:
        ulazi, oznake = ulazi.to(procesorskaJedinica),
oznake.to(procesorskaJedinica)
        izlazi = konvolucijskaMreza(ulazi)
        _, predviđanja = torch.max(izlazi, 1)
        svaPred.append(predviđanja)
        sveOznake.append(oznake)

        #ažuriranje točnih pogodaka i ukupnih pokušaja
        for oznaka, predviđanje in zip(oznake, predviđanja):
            if oznaka == predviđanje:
                tocnaPred[oznaka] += 1
            totalPred[oznaka] += 1

klase = []

#izračun postotka točnosti za sve klase
tocnosti = 100 * tocnaPred.float() / totalPred
for idKlase, nazivKlase in enumerate(setZaTestiranje.classes):
    print(f'Postotak tocnosti za klasu {nazivKlase:5s} je:
{tocnosti[idKlase]:.1f} %')
    klase.append(nazivKlase)
```

Isječak koda 8: Testiranje konvolucijske mreže nad preostalim skupm slika.

Prvo se postavi konvolucijska mreža u način testiranja pomoću *eval* metode. Zatim odredimo polja za broj točnih pogodaka (*tocnaPred*) i za broj ukupnih pokušaja (*totalPred*). Krećemo prolaziti kroz sve slike ali tijekom testiranja pozovemo *no_grad* metodu da se mreža ne uči nad ovim skupom podataka. Idući korak je izvući predviđanja i postaviti vrijednost za koju računalo najviše „sigurno“ u polje *predviđanja*.

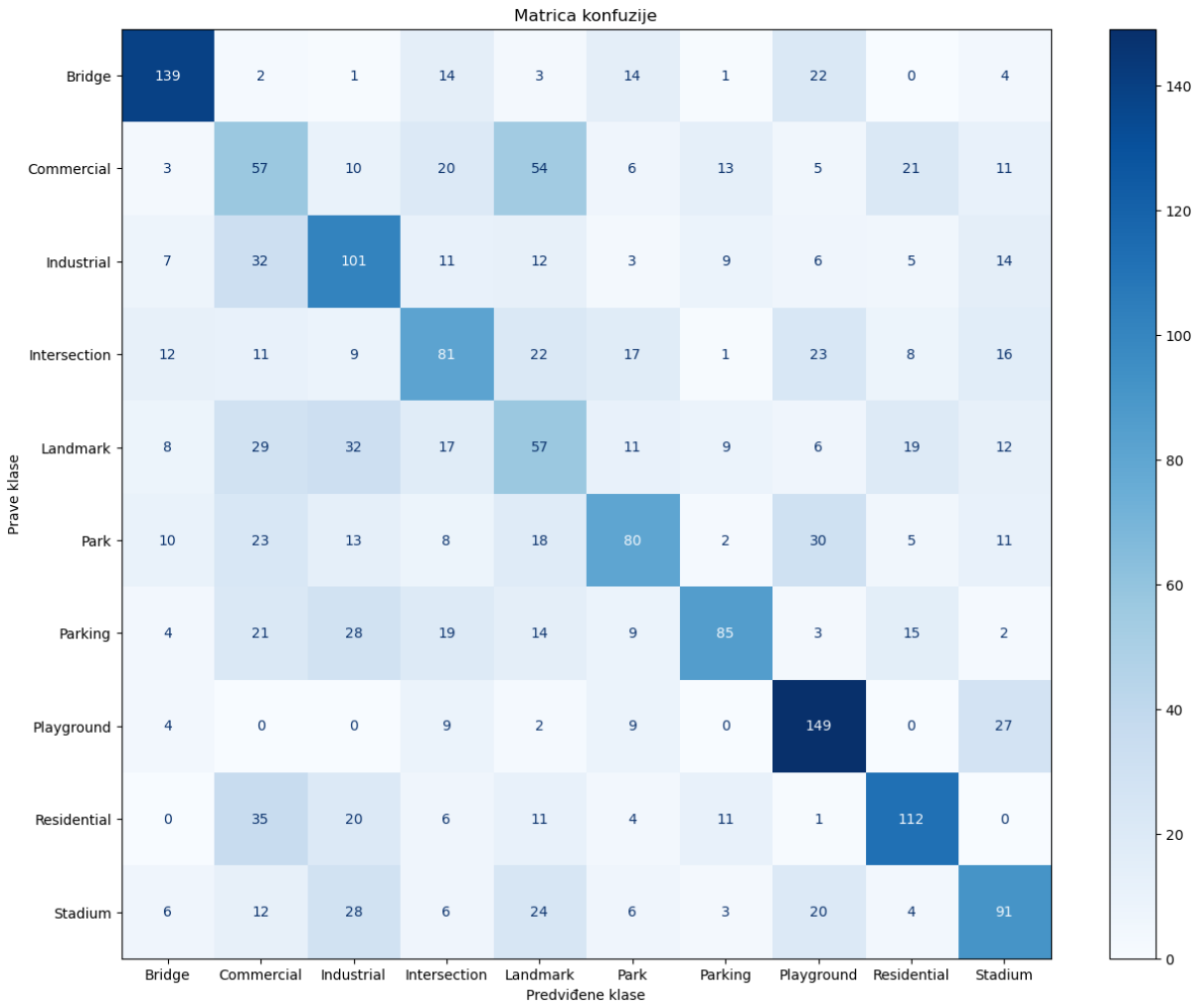
Zatim se prolazi kroz to polje i oznake klasa (polje *oznake*) te se određuje je li vrijednost pogođena, ako je povećava za tu klasu se povećava broj točnih pogodaka. Neovisno o tome ako je vrijednost pogođena broj ukupnih pokušaja se povećava za jedan. Za kraj se u polje *tocnosti* zapisuje postotak točnosti za sve klase. Rezultat tih postotaka točnosti je prikazan na slici broj 17.



Slika 17: Postotak točnosti po klasi (vlastita izrada).:

Najveći postotak točnosti su imale klase: most (69.5%) i igralište (74.5%). Srednji postotak točnosti su imale klase: industrijsko područje (50.5%), raskrižje (40.5%), park (40%), parking (42.5%), stambeno područje (56%) i stadion (45.5%). Najmanji postotak točnosti imaju 2 klase, a to su poslovne zgrade (28.5%) i znamenitost (28.5%). Sve u svemu ovaj rezultat je dosta dobar pošto je to klasifikacija slika nad 10 klasa. Ako bi ovo bila binarna klasifikacija (2 klase) onda ovo ne bi bio dobar rezultat.

Prema postotcima točnosti vidimo da neke klase slika naša konvolucijska neuronska mreža teže prepoznaje. Napravit ćemo matricu konfuzije (eng. Confusion Matrix) kako bi saznali koje klase slika se lakše prepoznaju, a koje klase se miješaju tijekom klasifikacije. Matricu konfuzije možemo vidjeti na slici 18.



Slika 18: Matrica konfuzije (vlastita izrada)

U matrici konfuzije stupci predstavljaju vrijednosti klasa koje je naša konvolucijska mreža predvidjela, dok redovi predstavljaju stvarne klase slika. Na dijagonali se nalaze sve točno klasificirane slike, te ako napravimo sumu po dijagonali dobit ćemo ukupan broj točno klasificiranih slika.

Za klasu most mreža je točno klasificirala 139 od 200 slika, te nije imala nekakav problem kod klasifikacije slika mosta.

Za klasu poslovne zgrade mreža je točno klasificirala samo 57 od 200 slika. Mreža je tu klasu najviše miješala sa klasom znamenitosti (54 slike poslovnih zgrada su klasificirane kao znamenitosti).

Za klasu industrijsko područje 101 slika je točno klasificirana dok je mreža 32 slike klasificirala pod poslovne zgrade. Preostale slike je mreža podjednako raspodijelila po ostalim klasama.

Za klasu raskrižje mreža je točno klasificirala 81 od 200 slika. Naša mreža je klasu raskrižja dosta miješala sa klasom znamenitosti (22 slike) i klasom igralište (23 slike).

Kod klase znamenitosti mreža je također vrlo loše klasificirala slike (samo 57 slika). Klasu znamenitosti mreža je većinom miješala sa klasom poslovne zgrade (29 slika), klasom industrijsko područje (32 slike) i malo manje sa klasama raskrižje (17 slika) i stambeno područje (19 slika).

Kod klase park mreža je točno klasificirala 80 od 200 slika, a najviše je miješala sa klasama igralište (30 slika) i poslovne zgrade (23 slike).

Za klasu parking točno je klasificirano 85 od 200 slika. Tu klasu je mreža malo miješala sa klasama poslovne zgrade (21 slika) i industrijsko područje (28 slika).

Kod klase igralište mreža je imala veliki uspjeh u klasifikaciji (149 točno klasificiranih slika), a mali problem joj je stvarala klasa stadion (27 slika).

Za klasu stambeno područje mreža je točno klasificirala 112 od 200 slika te je dosta miješala stambeno područje sa poslovnim zgradama (35 slika).

Na kraju nam je ostala klasa stadion kod koje je mreža točno klasificirala 91 sliku. Mreža je stadione dosta miješala sa industrijskim područjima (28 slika), znamenitostima (24 slike) i igralištima (20 slika).

Iz matrice konfuzije možemo zaključiti da su kod mnogih klasa problem u klasifikaciji stvarale klase poslovne zgrade i industrijsko područje, te imamo dvije klase koje su vrlo uspješno klasificirane, a to su most i igralište. Naša mreža je imala najmanji uspjeh kod poslovnih zgrada i znamenitosti.

6. Zaključak

Konvolucijske mreže s u jedna velika ali dosta mlada grana strojnog učenja. One se u današnjem svijetu sve više i više koriste kod nekakve klasifikacije slika, segmentacije slika, klasifkacije zvučnih zapisa, analize videa, robotike, autonomnih vozila, medicine, prepoznavanja lica. Njihova primjena je ogromna u današnjem svijetu, a s obzirom kako tehnologija napreduje sigurno će se pronaći još mnogo područja primjene.

Njihova arhitektura, koja uključuje slojeve konvolucije, aktivacije i sažimanja, omogućava efikasnu analizu složenih tipova podataka kroz veliki broj mapi značajki. Ova sposobnost prepoznavanja uzoraka u podacima čini ih izuzetno efikasnim u prepoznavanju i klasifikaciji složenih vizualnih informacija. Mnoge industrije su doživjele značajan napredak zbog primjene konvolucijskih mreža. Te mreže su temeljna tehnologija u modernom dubokom učenju te se nastavljaju razvijati i unaprjeđivati te pružati nove inovacije u procesiranju podataka i rješavati probleme koji su prije bili nezamislivi. Njihova prilagodljivost i snaga učenja iz podataka osigurava im da će ostati ključna tehnologija u razvijanju naprednih sustava umjetne inteligencije u budućnosti. Ni sami mi kao ljudi ne znamo kuda vodi taj napredak umjetne inteligencije jer je toliko nepredviđena ta tehnologija pa zbog toga izaziva dosta negativnih reakcija. Međutim vjerujem da treba ostati pozitivan i razmišljati više o boljoj budućnosti te kako koristiti tu tehnologiju za našu dobrobit.

Popis literature

[1] „Biological Neural Network: Importance, Components & Comparison“, upGrad blog. Pristupljeno: 26. lipanj 2024. [Na internetu]. Dostupno na: https://www.upgrad.com/blog/biological-neural-network/

[2] „Why are Neuron Axons Long and Spindly?“ Pristupljeno: 26. lipanj 2024. [Na internetu]. Dostupno na: https://today.ucsd.edu/story/why_are_neuron_axons_long_and_spindly

[3] A. Tripathi, „What Is Perceptron? Introduction, Definition & More“, Blogs & Updates on Data Science, Business Analytics, AI Machine Learning. Pristupljeno: 30. lipanj 2024. [Na internetu]. Dostupno na: https://www.analytixlabs.co.in/blog/what-is-perceptron/

[4] S. SHARMA, „Activation Functions in Neural Networks“, Medium. Pristupljeno: 30. lipanj 2024. [Na internetu]. Dostupno na: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

[5] „Recent_developments_in_multilayer_percep20161014-4696-1dwci62-libre.pdf“. Pristupljeno: 01. kolovoz 2024. [Na internetu]. Dostupno na: https://d1wqtxts1xzle7.cloudfront.net/49599119/Recent_developments_in_multilayer_percep20161014-4696-1dwci62-libre.pdf?1476458891=&response-content-disposition=inline%3B+filename%3DRecent_Developments_in_Multilayer_Percep.pdf&Expires=1722525458&Signature=Y2DYLbKwdzxLBT9NK7pVMtTrte-fusgHchHJ1vobIui~TWneXIAfttCJdaclyk3bHqHgLmuCX6qK5KPfLJ9LJbUGKrXuAVV2i80fUYONxED0iu5nfvzRXByGMRuOHjJlLUdQDFZFazMQUgXgXdHB0zWq5PbAxMEQXVjlA9WyY2NRTa5FPoONIKZ8tgk7J4WPFOh4ZuW4cKqap5~EdN252y2YHiEZEdsZtSweaqj2PJ0Epi9XvroUbGn-G4fc5S7UHmwtAkQ3w0yvkTYha1TwzaLrIy0LFRCB5tqbzcNOlxa2Y7E3n~d0Ozm4wteKhV7gfXplHiakwhhMlFsLXW0wcQ__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA

[6] M.-C. Popescu, V. Balas, L. Perescu-Popescu, i N. Mastorakis, „Multilayer perceptron and neural networks“, *WSEAS Trans. Circuits Syst.*, sv. 8, srp. 2009.

[7] M. T. Camacho Olmedo, M. Paegelow, J. Mas, i F. Escobar, „Geomatic Approaches for Modeling Land Change Scenarios. An Introduction“, 2018, str. 1–8. doi: 10.1007/978-3-319-60801-3_1.

[8] D. Trehan, „Gradient Descent Explained“, Medium. Pristupljeno: 06. kolovoz 2024. [Na internetu]. Dostupno na: https://towardsdatascience.com/gradient-descent-explained-9b953fc0d2c

[9] I. Goodfellow, Y. Bengio, i A. Courville, *Deep Learning*. MIT Press, 2016.

[10] „The Backpropagation Algorithm“. Pristupljeno: 07. kolovoz 2024. [Na internetu]. Dostupno na: https://eclass.uoa.gr/modules/document/file.php/MATH728/%CE%91%CE%BA%CE%B1%CE%B4%CE%B7%CE%BC%CE%B1%CF%8A%CE%BA%CF%8C%20%CE%88%CF%84%CE%BF%CF%82%202021-22/Projects/pr4.pdf

[11] „3Blue1Brown - Backpropagation calculus“. Pristupljeno: 08. kolovoz 2024. [Na internetu]. Dostupno na: https://www.3blue1brown.com/lessons/backpropagation-calculus

[12] K. Fukushima, „Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position“, *Biol. Cybern.*, sv. 36, izd. 4, str. 193–202, tra. 1980, doi: 10.1007/BF00344251.

[13] Y. LeCun *i ostali*, „Handwritten Digit Recognition with a Back-Propagation Network“, u *Advances in Neural Information Processing Systems*, Morgan-Kaufmann, 1989. Pristupljeno: 09. kolovoz 2024. [Na internetu]. Dostupno na: https://proceedings.neurips.cc/paper_files/paper/1989/hash/53c3bce66e43be4f209556518c2fcb54-Abstract.html

[14] C. C. Aggarwal, *Neural Networks and Deep Learning: A Textbook*. Cham: Springer International Publishing, 2018. doi: 10.1007/978-3-319-94463-0.

[15] Zhang Jin-Yu, Chen Yan, i Huang Xian-Xiang, „Edge detection of images based on improved Sobel operator and genetic algorithms“, u *2009 International Conference on Image Analysis and Signal Processing*, Linhai, China: IEEE, 2009, str. 31–35. doi: 10.1109/IASP.2009.5054605.

[16] G. Amer, M. Ahmed, i A. Abushaala, *Edge Detection Methods*. 2020.

[17] Dinesh, „Smoothing in Image Processing“, Scaler Topics. Pristupljeno: 11. kolovoz 2024. [Na internetu]. Dostupno na: https://www.scaler.com/topics/smoothing-in-image-processing/

[18] J. Gu *i ostali*, „Recent Advances in Convolutional Neural Networks“, 19. listopad 2017., *arXiv*: arXiv:1512.07108. Pristupljeno: 13. kolovoz 2024. [Na internetu]. Dostupno na: http://arxiv.org/abs/1512.07108

[19] A. Krizhevsky, I. Sutskever, i G. E. Hinton, „ImageNet Classification with Deep Convolutional Neural Networks“, u *Advances in Neural Information Processing Systems*, Curran Associates, Inc., 2012. Pristupljeno: 13. kolovoz 2024. [Na internetu]. Dostupno na: https://proceedings.neurips.cc/paper_files/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html

[20] „CNN_Backprop_Recitation_5_F21.pdf“. Pristupljeno: 15. kolovoz 2024. [Na internetu]. Dostupno na: https://deeplearning.cs.cmu.edu/F21/document/recitation/Recitation5/CNN_Backprop_Recitation_5_F21.pdf

[21] „Derivation of Backpropagation in Convolutional Neural Network (CNN)“, PyCodeMates. Pristupljeno: 16. kolovoz 2024. [Na internetu]. Dostupno na: https://www.pycodemates.com/2023/07/backward-pass-in-convolutional-neural-network-explained.html
[22] M. Rathi, „Backpropagation in a Convolutional Neural Network“. Pristupljeno: 16. kolovoz 2024. [Na internetu]. Dostupno na: https://mukulrathi.com/demystifying-deep-learning/conv-net-backpropagation-maths-intuition-derivation/
[23] „A Guide To Convolutional Neural Networks For Computer Vision“, UserManual.wiki. Pristupljeno: 16. kolovoz 2024. [Na internetu]. Dostupno na: https://usermanual.wiki/Document/A20Guide20to20Convolutional20Neural20Networks20for20Computer20Vision.938259461/view
[24] Z. Li, F. Liu, W. Yang, S. Peng, i J. Zhou, „A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects“, *IEEE Trans. Neural Netw. Learn. Syst.*, sv. 33, izd. 12, str. 6999–7019, pros. 2022, doi: 10.1109/TNNLS.2021.3084827.
[25] K. He, X. Zhang, S. Ren, i J. Sun, „Deep Residual Learning for Image Recognition“, 10. prosinac 2015., *arXiv*: arXiv:1512.03385. Pristupljeno: 18. kolovoz 2024. [Na internetu]. Dostupno na: http://arxiv.org/abs/1512.03385
[26] A. Khan, A. Sohail, U. Zahoora, i A. S. Qureshi, „A Survey of the Recent Architectures of Deep Convolutional Neural Networks“, *Artif. Intell. Rev.*, sv. 53, izd. 8, str. 5455–5516, pros. 2020, doi: 10.1007/s10462-020-09825-6.
[27] „Aerial Images of Cities“. Pristupljeno: 20. kolovoz 2024. [Na internetu]. Dostupno na: https://www.kaggle.com/datasets/yessicatuteja/skycity-the-city-landscape-dataset
[28] „AID: A Benchmark Dataset for Performance Evaluation of Aerial Scene Classification“. Pristupljeno: 20. kolovoz 2024. [Na internetu]. Dostupno na: https://captain-whu.github.io/AID/
[29] „Papers with Code - RESISC45 Dataset“. Pristupljeno: 20. kolovoz 2024. [Na internetu]. Dostupno na: https://paperswithcode.com/dataset/resisc45
[30] „Welcome to Python.org“, Python.org. Pristupljeno: 21. kolovoz 2024. [Na internetu]. Dostupno na: https://www.python.org/
[31] „PyTorch“, PyTorch. Pristupljeno: 21. kolovoz 2024. [Na internetu]. Dostupno na: https://pytorch.org/
[32] „Visual Studio Code - Code Editing. Redefined“. Pristupljeno: 21. kolovoz 2024. [Na internetu]. Dostupno na: https://code.visualstudio.com/
[33] „CUDA Toolkit - Free Tools and Training“, NVIDIA Developer. Pristupljeno: 21. kolovoz 2024. [Na internetu]. Dostupno na: https://developer.nvidia.com/cuda-toolkit
[34] „Matplotlib — Visualization with Python“. Pristupljeno: 21. kolovoz 2024. [Na internetu]. Dostupno na: https://matplotlib.org/

[35] „draw.io“. Pristupljeno: 21. kolovoz 2024. [Na internetu]. Dostupno na: https://app.diagrams.net/
[36] „Desmos | Beautiful free math.“ Pristupljeno: 21. kolovoz 2024. [Na internetu]. Dostupno na: https://www.desmos.com/
[37] „SGD — PyTorch 2.4 documentation“. Pristupljeno: 21. kolovoz 2024. [Na internetu]. Dostupno na: https://pytorch.org/docs/stable/generated/torch.optim.SGD.html

Popis slika

Popis isječaka koda