Prepoznavanje rukom pisanih simbola primjenom konvolucijske neuronske mreže

Čivčija, Tomislav

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Tomislav Čivčija

PREPOZNAVANJE RUKOM PISANIH SIMBOLA PRIMJENOM KONVOLUCIJSKE NEURONSKE MREŽE

DIPLOMSKI RAD

Varaždin, 2018.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Tomislav Čivčija

Matični broj: 45253/16-R

Studij: Informacijsko i programsko inženjerstvo

PREPOZNAVANJE RUKOM PISANIH SIMBOLA PRIMJENOM KONVOLUCIJSKE NEURONSKE MREŽE

DIPLOMSKI RAD

Mentor:

Doc. dr. sc. Nikola Ivković

Varaždin, rujan 2018.

Tomislav Čivčija

Izjava o izvornosti

Izjavljujem da je moj 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 potvrdio prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

U radu su istraživane mogućnosti konvolucijskih neuronskih mreža za rješavanje problema rukom pisanih znakova. Prikazan je dosadašnji napredak u prepoznavanju znakova te su obrađene i analizirane najnovije metode i tehnike u tom području.

Konvolucijske neuronske mreže su posebno uspješne jer rješavaju probleme računalnog vida s prikladnim tehnikama. Problemi invarijantnosti se rješavaju primjenom lokaliziranog učenja nad svim dijelovima slike postupkom nazvan konvolucija (eng. *convolution*).

Ove mreže se mogu sastavljati sa različitim slojevima s ciljem rješavanja specifičnog problema. Ukoliko je uzorak preveliki može se koristiti sloj za smanjenje uzorka ili konvolucije 1x1. Kada se želi održati veličina uzorka može se koristiti dopuna, a u slučaju sporog konvergiranja, mreža se može inicijalizirati na različite načine. Za implementaciju neuronskih mreža u ovome radu je korišten *Python* dok je za treniranje i evaluaciju upotrjebljena MNIST-ova baza slika. Rezultati su pokazali da novije tehnike poput ispada i grupne normalizacije imaju velik utjecaj na preciznost mreže. Korištenjem tih novijih tehnika, mreža implementirana u radu postiže visoku preciznost (99.50%) u prepoznavanju rukom pisanih znamenki.

Ključne riječi: konvolucija; smanjenje uzorka; neuronske mreže; računalni vid; aktivacijske funkcije; mape značajki; invarijantnosti; treniranje sa GPU

Sadržaj

1. Prepoznavanje znakova računalnim vidom

Jedno od najstarijih aplikacija elektroničkih uređaja je bilo prepoznavanje znakova. Konkretno radi se o optičkom prepoznavanju karaktera (eng. *Optical character recognition, OCR*) na osnovu slika, bili to skenirani ili uslikani dokumenti, bilo kakve slike ručno napisanih simbola ili čak video isječaka titlova na televizijskim programima.

OCR je u svojem originalnom smislu od začetaka korišten za dokumente poslovne naravi, čekovi, osobne iskaznice, putovnice, bankovni računi, međutim u novije vrijeme počeo se primjenjivati i na šire područje poput pretraga uslikanih dokumenata (*Google Books*), pretvaranje ručno pisanih znakova u digitalne u realnom vremenu (eng. *Pen computing*) i čak kao pomoć osobama sa oslabljenim vidom.

1.1. Kategorizacija računalnog vida

OCR se može podijeliti dva glavna dijela - prepoznavanje tipkanih simbola ili ručno pisanih (eng. *Intelligent character recognition, ICR*). Također postoji još kategorija prepoznavanja pojedinačnog simbola ili skupa njih kao riječ. Problem se može podijeliti i na tzv. online i offline ručno pisane znakove (R. Plamondon i S. N. Srihari, 2000 [24]). Online prepoznavanje se izvodi u realnom vremenu dok sudionik piše nekom vrstom elektroničke olovke po uređaju koji registrira pokrete. Offline je slučaj kada su znakovi već ranije napisani i pohranjeni u digitalnom obliku. Prema A. Graves i J. Schmidhuber (2009) [25], online slučaj je općenito lakši problem zato što postoji dodatna informacija koja ne postoji u offline prepoznavanju - putanja olovke. Pomoću ove dodatne informacije, precizne online modele za prepoznavanje znaka je lakše ostvariti u odnosu na offline modele koji imaju samo sliku kao izvor informacija.

1.2. Izvlačenje značajki

Jedan od presudnih problema u prepoznavanju rukom pisanih simbola je izvlačenje značajki (eng. *Feature extraction*). Izvlačenje značajki predstavlja izvlačenje određenih uzoraka koji mogu biti font, nagnutost (kurziv, obrnuti kurziv), omjer širine i duljine znakova, razmak između znakova i sl. Učinkovitost prepoznavanja na širokom spektru rukopisa ponajviše ovisi o izvlačenju značajki. OCR i ICR algoritmi su sprva bili ručno podešavani od

strane programera, potrebno je bilo odabrati pravu vrijednost pojedinih parametara za najpreciznije prepoznavanje, parametara koji su direktno utjecali na izvlačenje različitih značajki. Ovakav pristup zahtjeva dug proces analiziranja utjecaja različitih postavki parametara na preciznost prepoznavanja i veliki je problem jer se proces ne može izvršavati automatski (Y. LeCun i sur., 1998 [1]).

1.3. Novije metode prepoznavanja

Y. LeCun i Y. Bengio (1995) [8] predlaže novu vrstu neuronskih mreža specifično prilagođenu za računalni vid i naziva ih konvolucijskim neuronskim mrežama (eng. *convolutional neural network*, CNN). Također je kreirao poznati MNIST skup podataka sa ručno pisanim brojkama. U to vrijeme je CNN bio najprecizniji algoritam nad MNIST skupom. Od ispitanih algoritama jako dobro su prošli K-najbližih susjeda (eng. *K-nearest neighbour*), metode potpornih vektora (*support vector machines*, SVM) te klasična neuronska mreža sa tzv. *deskewing* preprocesiranjem.

A. Graves i Jurgen Schmidhuber (2009) [25] su koristili višedimenzionalne neuronske mreže sa povratnim vezama (eng. *multidimensional recurrent neural network*, RNN) te duboke tzv. unaprijedne (eng. *feedforward*) neuronske mreže za offline prepoznavanje znakova. U istom radu, za online prepoznavanje su koristili tzv. algoritam dugo kratkoročne memorije (eng. *long short-term memory*, LSTM) čija arhitektura pogoduje konstantnom priteku podataka dok je znak u procesu pisanja. Razvojem ovih tehnika značajno su unaprijedili preciznost računalnog vida.

1.4. Utjecaj hardvera

R. Raina i sur. (2009) [26] su otkrili da korištenje grafičkih procesorskih jedinki (*graphics processing unit*, GPU) može dovesti više od 70x većih performansi što je efektivno značilo da se veličina neuronskih mreža može povećati za 70x i pritom trenirati jednako vremena. Ovo je bio presudan trenutak u računalnom vidu jer je omogućeno dizajniranje kompleksnih arhitektura neuronskih mreža koje imaju daleko veće performanse nego prethodne jednostavne i dimenzijski male mreže.

D. Ciresan i suradnici (2011) [27] trenirajući CNN pomoću GPU su ostvarili najbolje performanse u detektiranju kineskih znakova. Samo godinu dana poslije, D. Ciresan i suradnici

(2012) [28] navode da je njihova CNN arhitektura prva koja je ostvarila performanse na ljudskoj razini nad MNIST skupom podataka.

1.5. Cilj rada

Razvojem novih algoritama i pojačavanjem računalne moći modernih računala današnje se mogućnosti OCR-a dovode do novih granica i cilj ovog rada je istražiti te granice koristeći najmoderniji algoritam u računalnom vidu, konvolucijsku neuralnu mrežu (eng. *Convolutional neural network*, CNN), i utvrditi napredak CNN arhitekture kroz zadnje desetljeće.

Za implementaciju, istraživanje i iscrtavanje rezultata kroz rad je korišten programski jezik *Python* i biblioteke poput *NumPy*. Za konkretnu implementaciju konvolucijskih neuronskih mreža kroz rad su korištene biblioteke *Keras* i *TensorFlow*. Sve to je napisano u *Pycharm* uređivaču. Za usporedbu različitih utjecaja primjene određenih metoda i tehnika korišteni su vizualni prikazi poput tablica, grafova i slika.

2. Konvolucijske neuronske mreže

Predstavljene su prvi put 1990, tvorac metode je LeCun a dorađene i popularizirane 1998 od strane istog istraživača [1] primjenom učenja baziranog na gradijentu. Ova vrsta neuronske mreže je posebno prilagođena rješavanju problema računalnog vida i povlači paralelu sa prirodnim vidom životinja i ljudi.

2.1. Invarijantnosti

CNN kombinira tri arhitekturne ideje kako bi se osigurale invarijante nagnutosti, veličine i distorzije: lokalna receptivna polja, dijeljene težine i prostorno smanjenje uzorka. Invarijanta predstavlja sposobnost prepoznavanja objekta u drugačijim uvjetima npr.

- translatorna invarijantnost (još nazvana invarijantnost pomaka, eng. *Shift invariance*) (LeCun i sur 1998 [1]) - znači prepoznati objekt na različitim mjestima u slici
- Rotacijska invarijantnost (S. Li (2017) [7]) - znači prepoznati objekt u bilo kojem stanju rotacije, 2d ili 3d rotacija
- Invarijantnost veličine (LeCun i Bengio 1995 [8], Xu i sur 2014 [9]) - znači prepoznati objekt različitih veličina

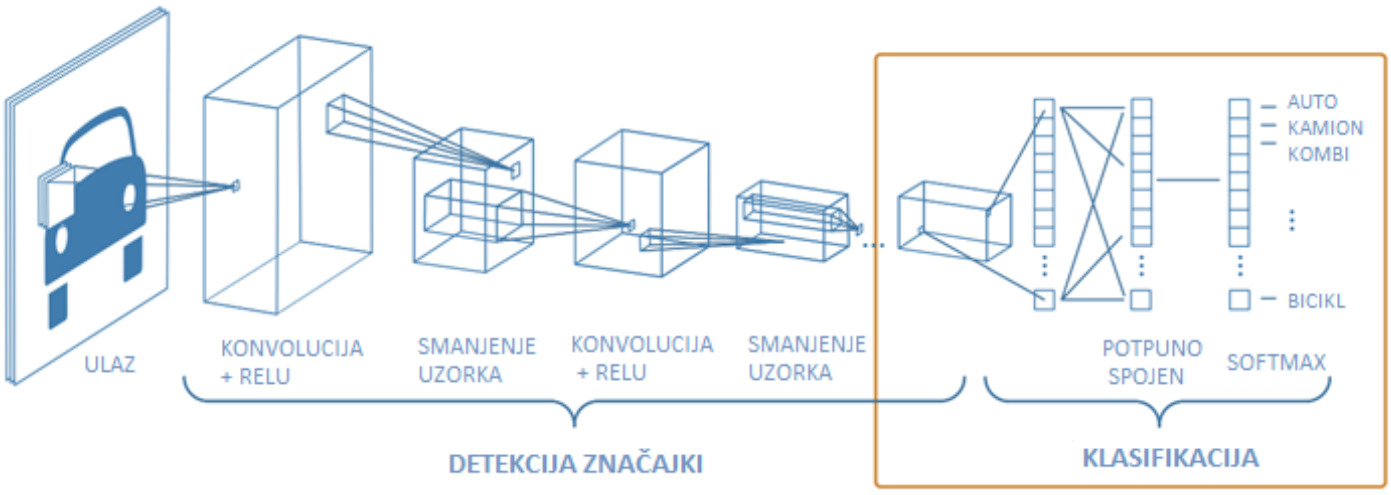
2.2. Klasična arhitektura

Klasična arhitektura se može vidjeti na slici 1. Slojevi CNN-a su

1. Konvolucijski (eng. *convolutional layer*, CONV)
2. Sloj aktivacije (eng. *activation layer*)
3. Sloj smanjenja uzorka (eng. *pooling, subsampling layer*, POOL)
4. Potpuno spojeni sloj (eng. *fully connected layer*, FC)

Konvolucijska mreža se može definirati kao niz tih slojeva koji svi imaju identičan tip ulaza i izlaza - 3D matrica, i svi vrše određenu transformacijsku funkciju. Promjenjive funkcije i funkcije koje se mogu upravljati parametrima su *CONV* i *FC* slojevi, a nepromjenjive funkcije su funkcije aktivacijskog sloja (najčešće korišten *ReLU*, druge opcije su *sigmoid*, *TanH*, *SoftPlus* i sl.) i *pooling* funkcije. U principu, konvolucijska mreža nastoji što bolje odrediti *CONV* i *FC* funkcije kako bi se minimizirala greška predikcije i to čini kroz unatražno rasprostiranje

(eng. *backpropagation*), više o tome kasnije. Također, može se reći da su neuroni u ovoj mreži zapravo *CONV* i *FC* slojevi. *FC* sloj je identičan kao u klasičnim neuronskim mrežama, svaki neuron je povezan sa svakim drugim iz prethodnog sloja te je smješten na kraju, na izlazu *CONV-ReLU-POOL* iteracija.



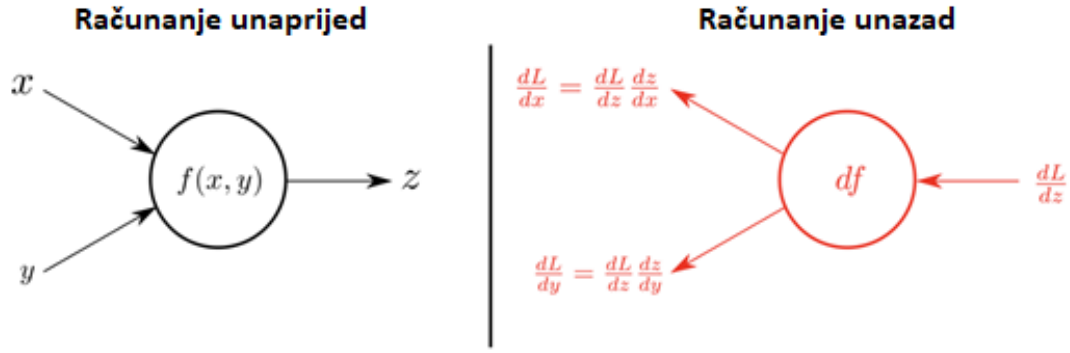
Slika 1 – Klasična arhitektura konvolucijske neuronske mreže (prevedeno sa S. Wadhwa, 2017 [3])

2.3. Unaprijedna priroda mreže i unatražno rasprostiranje

Unaprijedna priroda (eng. *feedforward*) konvolucijskih neuronskih mreža znači da konekcije između čvorova (eng. *node*) ne tvore petlju. Drugim riječima, čvorovi nikada ne idu unatrag niti u stranu unutar istog sloja već uvijek idu naprijed. *Feedforward* mreže se još nazivaju i aciklične dok se neuronske mreže sa povratnim vezama (eng. *recurrent neural networks*) nazivaju ciklične zbog petlji. Općenito gledajući za RNN mreže se kaže da su najdublje upravo zbog povratnih veza te da imaju širi spektar za računanje u odnosu na *feedforward* mreže. (J. Schmidhuber, 2014 [14])

Unaprijedna priroda konvolucijskih neuronskih mreža znači da je utjecaj neurona jednosmjeran. Međutim to vrijedi samo za postupak prepoznavanja, za treniranje mreže se koristi unatražno rasprostiranje, odnosno propagacija unatrag.

Propagacija unatrag je postupak kojim se korigiraju težine neurona i time zapravo postiže trening neuronske mreže. Unatražno rasprostiranje je metoda nastala iz dinamičkog programiranja. Metode nastale iz dinamičkog programiranja mogu pomoći da se višestruko smanji dubina problema. Algoritmi dinamičkog programiranja su nužni za sustave koji kombiniraju koncepte neuronskih mreža i grafičkih modela poput skrivenih markovljevih modela (eng. *hidden markov models*). (J. Schmidhuber, 2014 [14])



Slika 2 – Računanje unaprijed i unazad (prevedeno sa F. Kratzert, 2016 [4])

Učenje se postiže promjenom težina neurona nakon svakog procesiranja podataka, bazirano na količini pogreške izlaza uspoređenog sa stvarnim rezultatom. Ovo se još naziva i učenje sa nadgledanjem jer se evaluira performans na osnovu očekivanog rezultata. Kroz unatražno rasprostiranje, primjenom gradijenta pada (eng. *gradient descend*), težine neurona se mijenjaju na način da minimiziraju funkciju pogreške. Drugim riječima, na osnovu izračunate pogreške (razlika između očekivanog i dobivenog rezultata), težine se ažuriraju kako bi ta razlika bila manja u budućnosti. (S. Haykin, 1998 [15])

2.4. Lokalna receptivna polja

U računalu slika je predstavljena u 3 dimenzije - visina, širina i boja što daje matricu dimenzija *x,y,z*. Kako bi pojednostavili pristup, namjerno ćemo izostaviti treću dimenziju što efektivno znači da je dimenzija boje = 1, odnosno crno-bijela slika. Lokalna receptivna polja su dobivena konvolucijom, primjenom jezgre (često nazivani filteri , eng. *kernel)*, matrice težina koje vrše mapiranje ulaznog signala (u prvom koraku originalna slika, u ostalim koracima rezultat prethodnog ciklusa) u izlazni signal. *Kernel* je matrica koja definira veličinu receptivnog polja, svaka izlazna točka je dobivena primjenom *kernela* nad ulaznim signalom, *kernel*

veličine 3x3 znači da je receptivno polje točke veliko 3x3. Konvolucija se sastoji od iterativne primjene *kernela* nad ulaznim signalom nad različitim lokacijama. Jedna definirajuća karakteristika ovog procesa je preklapanje receptivnih polja točki u izlaznom signalu. Ako je npr. ulazni signal veličine 4x4 a *kernel* veličine 3x3, izlazni signal će biti veličine 2x2 i svaka točka će imati preklapanje receptivnih polja sa svakom drugom točkom u matrici.

Uloga receptivnih polja je višestruka. Kao prvo smanjuju veličinu kompletne mreže budući da svaki neuron nije povezan sa svakim iz sloja prije njega već samo sa lokalnim područjem - receptivnim poljem. Također primjenom konvolucije, pojedini *kernel* će biti konvoluiran nad cijelom slikom što znači pretraga nad cijelim područjem odnosno dijeljenje parametara (Y. LeCun i sur., 1998 [1])

2.5. Konvolucijski sloj

Konvolucija je postupak kojim se dva signala isprepleću u različitim intervalima kako bi proizveli treći signal koji je sačinjen od rezultata intervala (S.W. Smith, 1997 [30]). To je točno ono što se događa u konvoluciji slika odnosno dvodimenzionalih matrica piksela (ili trodimenzionalnih ukoliko je prisutna i informacija o boji), *kernel* se primjenjuje nad slikom koristeći konvoluciju i proizvodi treći signal - filtriranu sliku.

Kernel obično ima manje dimenzije od ulazne slike. Konvolucija se postiže time što se *kernel* primjenjuje nad slikom više puta na različitim mjestima prateći tzv. pomak (eng. *stride*). Pomak je hiper-parametar, predstavlja pomak *kernela* nad slikom, ukoliko je pomak jednak 2 *kernel* će se primijeniti 2 *mjesta* dalje u odnosu na prethodnu primjenu. Prednost ovog procesa je što pomak može biti manji od veličine *kernela* čime se postiže već ranije spomenuto preklapanje receptivnih polja izlaza. To nadalje pomaže pri otkrivanju značajki u slici iako se ne nalaze uvijek na istom području slike. Značajka može biti pomaknuta gore-dolje, lijevo-desno u odnosu na druge dijelove cjeline međutim preklapanje receptivnih polja znači da je više neurona “susrelo” tu značajku i veća je šansa da će biti prepoznata. Zbog toga je važno da pomak bude manji od veličine *kernela*, u suprotnom se gubi ova prednost.

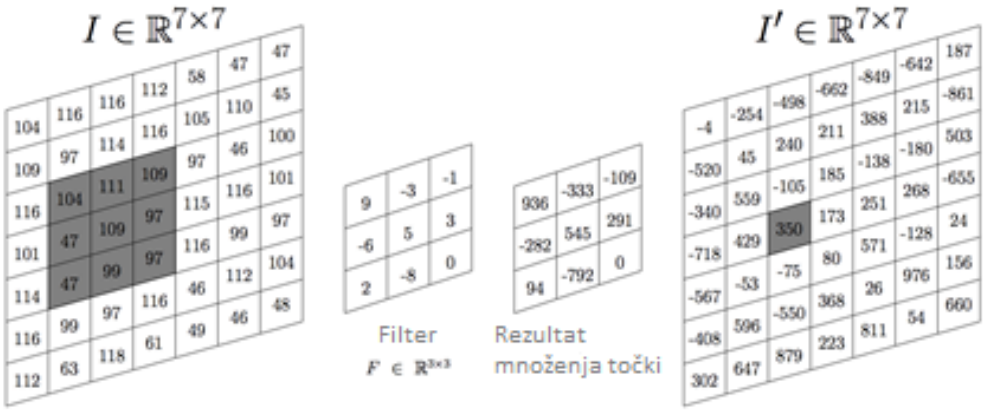
Usput, budući da se *kernel* primjeni na cijeloj slici dio po dio, konvolucija djeluje kao pretraga značajke na različitim mjestima, to omogućuje prepoznavanje značajke na drugačijim mjestima.

S tehničke strane da se zaključiti da je konvolucijski sloj posebna arhitektura neuronske mreže u kojoj je svaki neuron povezan sa lokalnom mrežom prethodnog sloja za razliku od

klasične arhitekture gdje je svaki neuron povezan sa svakim drugim iz prethodnog sloja. Ovaj pristup dodatno smanjuje kompleksnost izračuna i brzinu treniranja jer je manje parametara za izračunati.

2.5.1. Kerneli

Kerneli su srž konvolucijske neuronske mreže, njima se vrši konvolucija. To su matrice čija veličina je veličina receptivnih polja. *Kerneli* se kroz treniranje specijaliziraju za ekstrahiraje značajki koje se provlače kroz podatkovni skup treniranja.



Slika 3 – Primjena *kernela* nad ulaznim vrijednostima (prevedeno sa M. Thoma, 2017 [46])

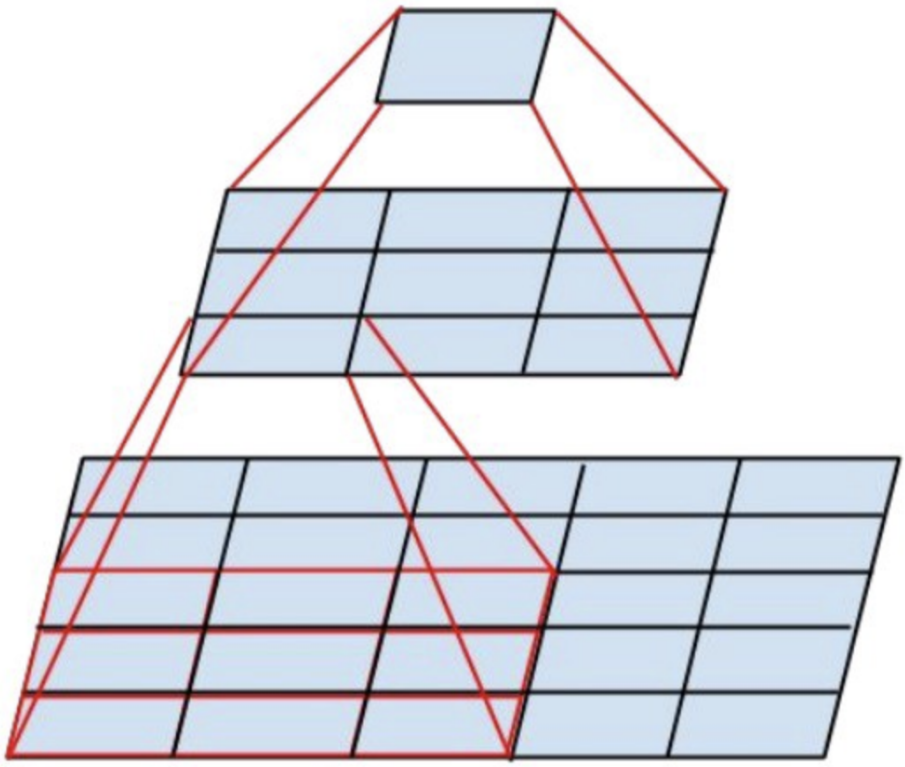
Primjena *kernela* na područje ulaza se sastoji od množenja elemenata matrice na istim koordinatama i zatim zbrajanja pojedinačnih elemenata u rezultatu. *Kernel* transformira ulaz i zbrajanjem se dobije kumulativni utjecaj *kernela* nad područjem ulaza.

2.5.1.1. Veličina kernela

Veći *kerneli* su sposobni prepoznati kompleksnije značajke međutim u isto vrijeme produžuju vrijeme treniranja mreže zbog povećanog broja parametara. Važno je naći ravnotežu između kvalitete i brzine kada se smišlja arhitektura mreže.

Pri otkrivanju konvolucijskih neuronskih mreža LeCun i sur (1998) su koristili 5x5 *kernele*. Budući da je primjena bila nad slikama veličine 32x32, veličina *kernela* je bila oko 5x5. Poslije toga kada se CNN počela primjenjivat nad većim slikama, i veličina *kernela* je sukladno bila povećavana na 9x9 ili 11x11 (A. Krizhevsky, 2012 [10]).

K. Simonyan i A. Zisserman (2014) [13] su uočili da višestruke konvolucije zaredom sa 3x3 *kernelom* mogu emulirati efekt većih receptivnih polja poput 5x5 ili 7x7. Slika ispod. Učinak 3x3 *kernela* primijenjenih zaredom je smanjenje parametara što vodi bržem treniranju mreže i omogućuje konstruiranje kompleksnijih i većih mreža.



Slika 4 – Vizualizacija primjene višestrukih 3x3 kernela zaredom (preuzeto sa Szegedy i sur, 2015 [20])

2.5.1.2. Inicijalizacija kernela

Različiti pristupi inicijalizaciji *kernela* mogu imati utjecaj na performanse i brzinu učenja mreže. Također, performanse nekih aktivacijskih funkcija mogu biti poboljšane ukoliko se koriste njima prilagođene inicijalizacije. Inicijalno, *kerneli* su se inicirali nasumično iz normalne Gaussove distribucije. X. Glorot i Y. Bengio (2010) [31] su probali riješiti problem - zašto standardni gradijent pada (eng. *standard gradient descent*) iz nasumične inicijalizacije ima loše performanse u dubokim neuronskim mrežama (eng. *deep neural network*, DNN). U radu su osmislili poboljšanu inicijalizaciju tzv. *normalizirana inicijalizacija* koja je poslije popularizirana

kroz razvojna okruženja *Caffe* i *Keras* kao *Xavier inicijalizacija*. Uz klasičnu Gaussovu distribuciju dodali su varijancu koja ovisi o broju ulaznih i izlaznih neurona:

$$Var(W) = \frac{2}{(n_{in} + n_{out})}$$

Pritom je n_{in} broj ulaznih neurona, n_{out} je broj izlaznih neurona. Ovo se dalje može pojednostaviti kao

$$Var(W) = \frac{1}{n_{in}}$$

U principu, ako su *kerneli* inicijalizirani prenisko onda se signal gubi kroz slojeve i naposljetku je premali da bi bio koristan. Također, ako je inicijalizacija previsoko onda je signal prejak i kroz slojeve se pojačava pa je naposljetku prevelik da bi bio koristan. *Xavierova inicijalizacija* podešava vrijednosti *kernela* na razinu koja čini signal niti previsokim niti preniskim.

K. He i sur. (2015a) [33] uvode promjene nad *Xavier* inicijalizacijom tako da je učinkovitija kada se primjenjuje *ReLU* aktivacijska funkcija. *ReLU* iznosi 0 za pola elemenata ulaza pa su stoga povećali varijancu za dva puta. Time se dobiva prilagođena varijanca koja drži signal konstantnim za slučaj *ReLU*.

$$Var(W) = \frac{2}{n_{in}}$$

2.5.2. Dopunjavanje

Dopunjavanje (eng. *padding*) je tehnika kojom se kontrolira dimenzija izlaza (najčešće CONV, ponekad i POOL sloja) (A. Karpathy, 2015 [32]). Dopunjavanje se postiže dodavanjem dodatnih redaka i stupaca na ulaz u CONV sloj. Metode dodavanja:

- *zero padding*: novi reci i stupci su popunjeni nulama
- *reflect*: refleksija slike na granici
- *nearest*: ponavljanje elementa koji je najbliži granici

41	21	-11	-96	52
6	6	-1	48	-92
-58	84	58	46	96
59	11	3	-6	-28
-50	42	59	71	26

0	0	0	0	0	0	0
0	41	21	-11	-96	52	0
0	6	6	-1	48	-92	0
0	-58	84	58	46	96	0
0	59	11	3	-6	-28	0
0	-50	42	59	71	26	0
0	0	0	0	0	0	0

Slika 5 – Primjena *Zero-Padding* tehnike

(lijevo mapa značajki, desno ista mapa značajki sa *Zero-Padding* dopunom)

Utjecaj dopunjavanja može biti različit:

A. Održava visina i širina izlaza prilikom konvolucije što može biti korisno jer pojednostavljuje proces dizajniranja arhitekture

B. Dopušta dizajniranje dubljih mreža jer sprječava ubrzano smanjenje dimenzija

C. Povećava performanse tako što čuva informacije na granici

D. Dopušta dizajniranje kompleksnijih mreža koje ulančavaju mape značajki različitih dimenzija kao izlaz CONV sloja (C. Szegedy i suradnici (2015) [17])

2.5.2.1. Održavanje dimenzija prilikom konvolucije

Korištenje *kernela* neizbježno smanjuje veličinu izlaza u odnosu na ulaz. Dimenzije izlaza Do pod pretpostavkom da su mape značajki i *kerneli* kvadratne matrice, su zadane formulom

$$D_o = \frac{D_i - D_k}{S} + 1$$

(Do = dimenzija izlaza, Di = dimenzija ulaza, Dk = dimenzija *kernela, S* = pomak)

U praksi vrijednosti su najčešće, $D_k = 3$ te $S = 1$ u kojem slučaju vrijedi $D_o = D_i - 2$. Tada se treba primijeniti dopunjavanje P veličine P = 1. Sa svih strana ulaza doda se po jedan niz elemenata čime se nakon konvolucije dobije izlaz jednake veličine. Općenito, ukoliko se žele održati iste dimenzije kroz konvoluciju za S = 1, dopunjavanje treba odgovarati sljedećoj vrijednosti:

$$P = \frac{D_k - 1}{2}$$

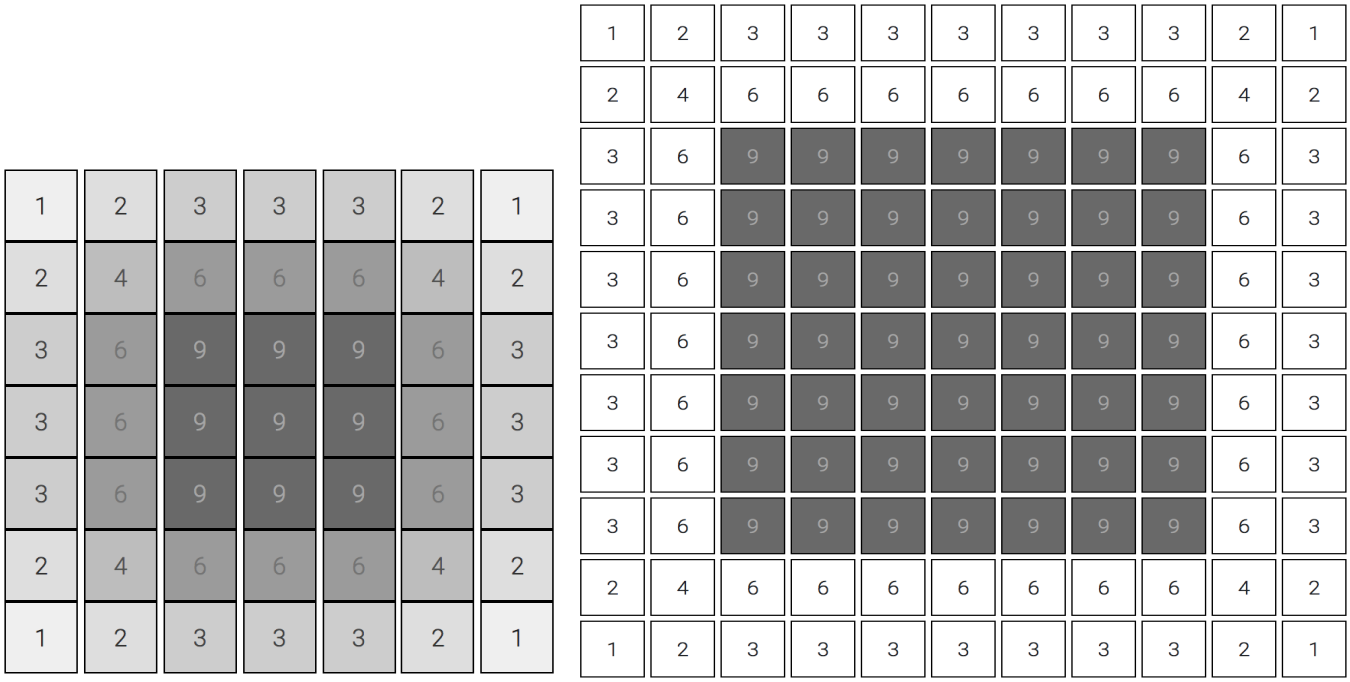
Ovo je jedan od razloga zašto su *kerneli* neparnih dimenzija, kada bi *kerneli* bili parni, dopunjavanje na jednoj strani bi morao biti veći što vodi do nekonzistentnosti.

2.5.2.2. Dopunjavanje u dizajnu dubljih mreža

Koristeći formulu ako je $D_i = 32$; $D_k = 5$; $S = 1$; D_o će biti $D_o = 28$ što je smanjenje za više od 10%. Sama ova činjenica postavlja grubi limit na iterativno primjenjivanje CONV slojeva. Koristeći dopunjavanje dimenzije ostaju iste pa stoga mreža može biti mnogo veća.

2.5.2.3. Čuvanje informacija na granici

Posljedica konvolucije je da su za $D_i > 5$, $D_k = 3$ te $S = 1$ granični elementi ulaza iskorišteni manje puta nego oni bliže centru. Drugim riječima, bez dopunjavanja elementi bliže granici imaju manje utjecaja na izlaz CONV sloja i njihov se utjecaj potpuno izgubi kroz više iteracija. Na figuri (ispod) se može vidjeti koliko je koji element ulaza uključen u kernel sa dopunjavanjem i bez dopunjavanja. Ukoliko se primjeni dopunjavanje granični elementi će imati jednak utjecaj kao i oni bliže centru. Dopunjavanje bi trebalo biti $P = D_k - 1$ ukoliko se informacije na granici žele čuvati.



Slika 6 – Prikaz uključenosti ulaznih vrijednosti kroz konvoluciju bez dopune (lijevo) i s dopunom (desno)

2.5.2.4. Ulančavanje mapa značajki

Najnovije arhitekture CNN-a poput onih iz [17] [20] [21] ulančavaju više odvojenih mapa značajki različitih dimenzija, načinjenih iz istog ulaza, u jedan izlaz. Budući da su mape različitih dimenzije, koristi se dopunjavanje kako bi se ujednačile dimenzije.

2.5.3. Pomak

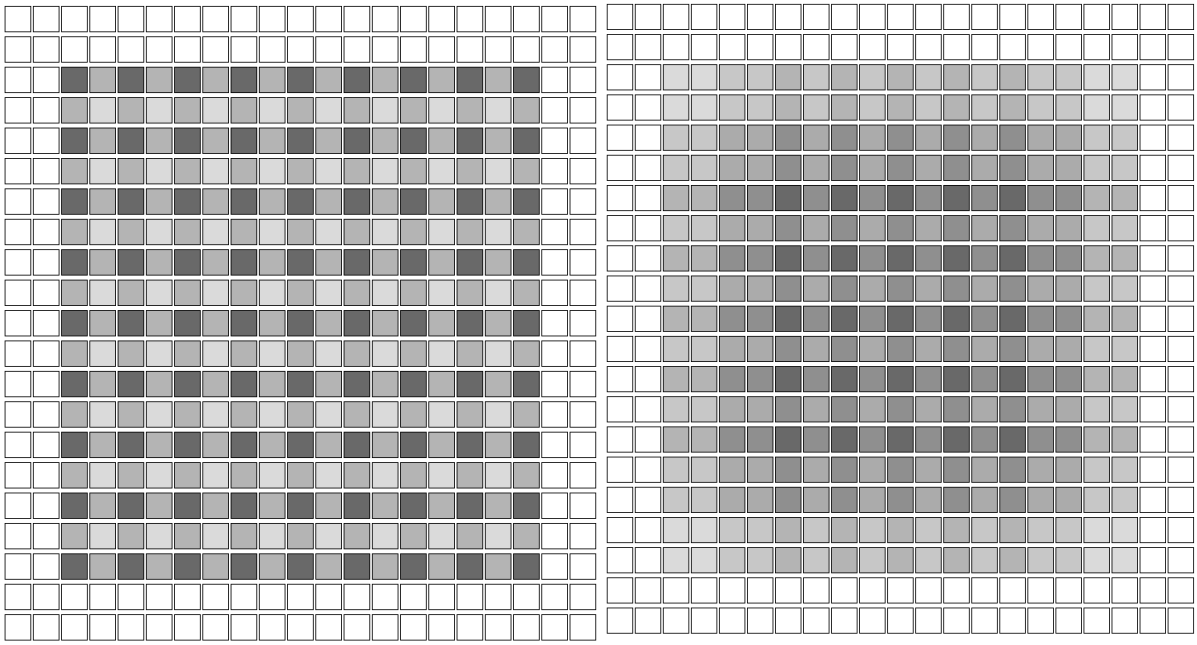
Najveću ulogu koju pomak ima je smanjenje dimenzije. Kako u CONV tako i u POOL sloju pomak odlučuje kolike će dimenzije biti na izlazi sloja. Međutim kako je u POOL sloju uvijek cilj postići smanjenje dimenzija i naglašenje *obrisa*, dimenzija *pooling* tehnike (D_p) je najčešće jednaka pomaku (S), najčešće iznosi $D_p = S = 2$. Tako su svi ulazi točno jednom obuhvaćeni a dimenzija izlaza je smanjena. Postoje i neke iznimke gdje je $D_p>2$ te $S != D_p$ poput nadograđene *Xception* mreže od L.C. Chen i sur (2018).

2.5.3.1. Pomak u konvolucijskom sloju

Pomak ima važnu ulogu u konvolucijskom sloju gdje ne utječe samo na dimenzije izlaza već koliko i u kojem rasporedu ulazi utječu na izlaz. Ako je S = 1, pomoću dopunjavanja se može postići slučaj da je utjecaj ulaza distribuiran jednako, odnosno da svaki element ulaza

jednako doprinosi izlazu. Ukoliko je S > 1, neizbježno dolazi do neravnomjerne distribucije koja se ne može popraviti dopunjavanjem. Na slici ispod vidljiv je nejednolika distribucija iskorištenosti ulaza.

Primjećujemo da što je *kernel* veći, distribucija je više normalizirana što je naravno i poželjno jer inače neki važni pikseli mogu biti izostavljeni pogotovo ako su slike manje veličine poput onih 32x32 iz MNIST baze. Zbog toga danas u praksi prevladava S = 1



Slika 7 – Distribucija iskorištenosti ulaza sa različitim veličinama kernela

3x3 *kernel* lijevo, 7x7 *kernel* desno; obje slike S = 2, D_i = 13, P = 2, bijeli kvadrati su dopunjavanje, iskorištenost ulaza 13x13 je predstavljena bojanjem od manje iskorištenosti (svijetlije) do više iskorištenosti (tamnije)

2.5.3.1.1. Prijedlog za jednoliko distribuiranu konvoluciju sa većim pomakom

Predložena su dva načina koji mogu održati približno jednoliku distribuciju. Oba načina su jako slična. U jednom konvolucijskom sloju obično postoji više *kernela*, često više od stotinu. Ključna ideja je da se *kerneli* ne primjenjuju uvijek na početku ulaza kao u običnom slučaju. Jedan način za održavanje jednolike distribucije iskorištenosti ulaza za S = 2 bi bio dati istup (eng. *offset*) veličine O = 1 svakom drugom *kernelu* za početni element primjene. Dakle svaki

neparni *kernel* bi bio primijenjen od početnog elementa a1,1 a svaki parni *kernel* bi bio primijenjen počevši od elementa a2,2. Rezultat je približno jednaka distribucija koja sadržava dva skupa elemenata koji tvore šahovnicu i razlika između njih je uvijek 1. Što je veći *kernel* relativna razlika se smanjuje (više puta su elementi iskorišteni a apsolutna razlika ostaje 1). U ovom rješenju, pojedini *kernel* bi ipak davao vlastiti izlaz koji nije pravilno distribuiran, kombinacijom izlaza dva *kernela* (parni i neparni) bi izlaz bio pravilno distribuiran.

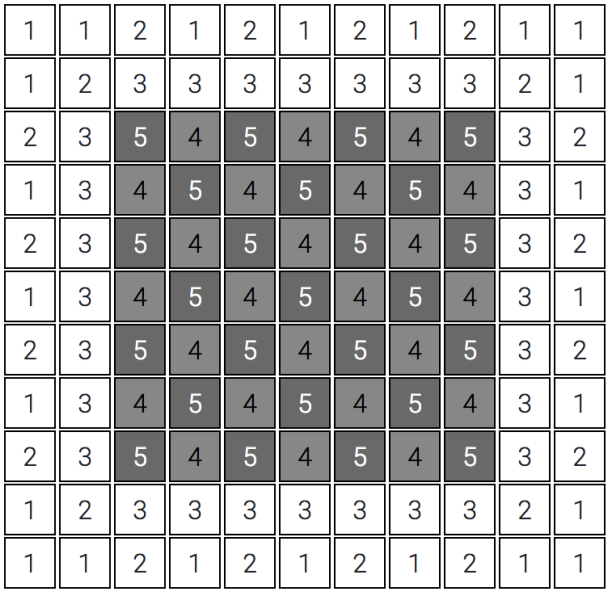
Hipotetski problem koji bi nastao je da bi isti *kernel* uvijek pridodavao višu vrijednost određenim pikselima što bi moglo negativno utjecati na performanse, pogotovo slika manjih dimenzija. To bi se moglo izbjeći oscilacijom primjene parnih-neparnih filtera na istup. U jednoj iteraciji parni filteri imaju istup a neparni nemaju, u drugoj iteraciji neparni imaju a parni nemaju. Drugi način za ostvarenje jednolike distribucije je da se isti filter primjeni dvaput nad ulazom, jednom sa istupom drugi puta bez.

Sa oba načina može se koristiti i pomak veličine S > 2. Kako bi se postignula ravnomjerna distribucija potrebno je tada koristiti dodatni istup za svaki dodatni pomak, odnosno generalno vrijedi da za broj istupa i skupova na koji se *kerneli* moraju podijeliti N_o vrijedi $N_o = S - 1$

Pretpostavljam da bi prednost ove tehnike bila dodatno smanjenje ulaza koje dolazi sa korištenjem pomaka S > 1 (korisno pri obradi većih slika) pritom zadržavajući performanse zbog istupa.

1	1	2	1	2	1	2	1	2	1	1
1	1	2	1	2	1	2	1	2	1	1
2	2	4	2	4	2	4	2	4	2	2
1	1	2	1	2	1	2	1	2	1	1
2	2	4	2	4	2	4	2	4	2	2
1	1	2	1	2	1	2	1	2	1	1
2	2	4	2	4	2	4	2	4	2	2
1	1	2	1	2	1	2	1	2	1	1
2	2	4	2	4	2	4	2	4	2	2
1	1	2	1	2	1	2	1	2	1	1
1	1	2	1	2	1	2	1	2	1	1

0	0	0	0	0	0	0	0	0	0	0
0	1	1	2	1	2	1	2	1	1	0
0	1	1	2	1	2	1	2	1	1	0
0	2	2	4	2	4	2	4	2	2	0
0	1	1	2	1	2	1	2	1	1	0
0	2	2	4	2	4	2	4	2	2	0
0	1	1	2	1	2	1	2	1	1	0
0	2	2	4	2	4	2	4	2	2	0
0	1	1	2	1	2	1	2	1	1	0
0	1	1	2	1	2	1	2	1	1	0
0	0	0	0	0	0	0	0	0	0	0



Slika 8 – Jednolika distribucija iskorištenosti ulaza pri konvoluciji (D_i = 7, P=2, D_k=3, S=2)

Lijevo $a_{start} = a_{1,1}$; Desno $a_{start} = a_{2,2}$; Sredina dolje kombinirani rezultat

2.5.4. Aktivacijske funkcije

Poznate i pod nazivom funkcije za gnječenje (eng. *squashing functions*) su funkcije koje mijenjaju ponašanje neuronskih mreža da budu nelinearne. Jedna od najjednostavnijih i najpopularnijih je *ReLU* popularizirana od strane Alex i sur. (2010). Novije funkcije su predložene i testirane poput *ELU*, parametarizirani ReLU (*PReLU*), *softplus* i *softsign* (Mishkin i sur. 2016 [11]).

Aktivacijske funkcije se razlikuju u dometu vrijednosti te njihovoj derivaciji. Teoretski razlozi zašto bi jedne funkcije bile bolje od drugih:

- *Umirući ReLU* - A. Maas i sur. (2013 [39]) opisuju ovaj problem u svom radu. *ReLU* gradijent je 0 za sve negativne vrijednosti. To znači da ako svi elementi skupa podataka za treniranje vode ka negativnom inputu za neki neuron kroz bilo koji dio procesa treniranja, taj neuron će ostati isti i stoga neće doprinositi treniranju mreže.

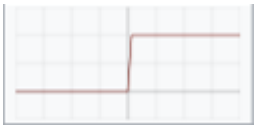
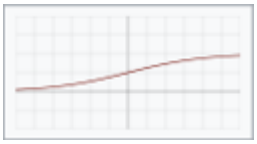
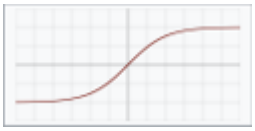
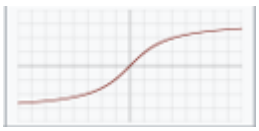
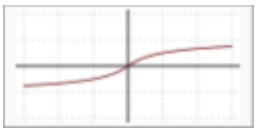
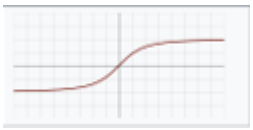
- Jedinke koje imaju aktivaciju čiji prosjek nije nula se ponašaju kao *bias* za sljedeći sloj. Ukoliko se takve jedinke ne ponište, učenje uzrokuje pomak *biasa* za jedinke sljedećeg sloja. Što su više jedinke povezane to je njihov pomak *biasa* veći. *ReLU* je funkcija koja prima pozitivne i negativne vrijednosti i pretvara ih u neutralne (nula) ili pozitivne. Stoga *ReLU* prosjek aktivacije nije blizu nuli, ne sadrži negativne vrijednosti. Pokazano je da

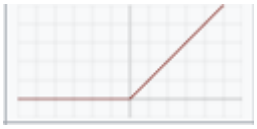
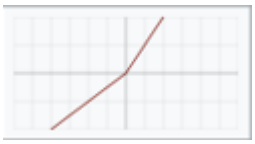
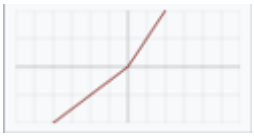
aktivacijske funkcije čiji je prosjek bliži nuli ubrzava učenje. Dakle cilj je koristiti funkciju čiji će prosjek biti bliže nuli, to će smanjiti pomak *biasa* što će naposljetku dati brže rezultate treniranja. Rješenje je koristiti ELU funkciju koju su specifično za ovu namjenu osmislili D.A. Clevert i sur. (2015 [34])

- Nestajući gradijent (eng. *vanishing gradient*) - aktivacijske funkcije poput *Tanh* i logističke funkcije su zasićene izvan intervala [-5, 5]. To znači da je ažuriranje težina vrlo nisko za nadolazeće neurone što je posebice problem za duboke ili mreže sa povratnim vezama (S. Hochreiter i sur., 2001 [42])

U nastavku predstavljamo najčešće aktivacijske funkcije

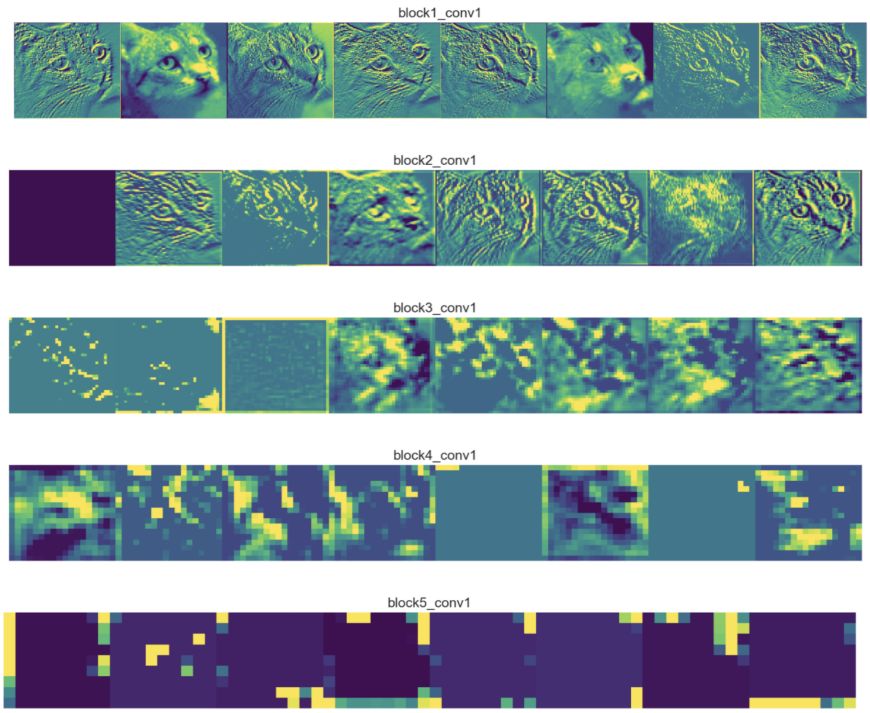
Tablica 1 – Najčešće aktivacijske funkcije

Identitet		$f(x) = x$
Binarni korak		$f(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases}$
Logistična (Sigmoid)		$f(x) = \sigma(x) = \frac{1}{1 + e^{-x}}$
TanH		$f(x) = \tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$
ArcTan		$f(x) = tan^{-1}(x)$
Softsign		$f(x) = \frac{x}{1 + \lvert x \rvert}$
Inverzna jedinka kvadratnog korijena		$f(x) = \frac{x}{\sqrt{1 + \alpha x^{2}}}$

Ispravljena linearna jedinka		$f(x) = \begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases}$
Leaky ispravljena linearna jedinka		$f(x) = \begin{cases} 0.01x, & x < 0 \\ x, & x \geq 0 \end{cases}$
Parametarska ispravljena linearna jedinka		$f(x, \alpha) = \begin{cases} \alpha x, & x < 0 \\ x, & x \geq 0 \end{cases}$
Nasumična leaky ispravljena linearna jedinka		$f(x, \alpha) = \begin{cases} \alpha x, & x < 0 \\ x, & x \geq 0 \end{cases}$

2.5.5. Mape značajki

Mape značajki (eng. *feature maps*) su izlaz *CONV* ili *POOL* sloja. U principu kada se govori o značajkama misli se na izlaz *CONV* sloja. Mape značajki su upravo kao što im ime implicira - mape koje sadrže značajke od prethodnog ulaza. Pojedini *kerneli* su trenirani da prepoznaju određene značajke, kada se ti *kerneli* primjene oni stvore mape značajki. Svaki *kernel* kreira po jednu mapu značajki. *RGB* slika je predočena kao tri odvojene mape, po jedna mapa za svaku boju. Mape značajki se mogu vizualizirati kako bi lakše dobili predodžbu koju značajku koji *kernel* detektira.



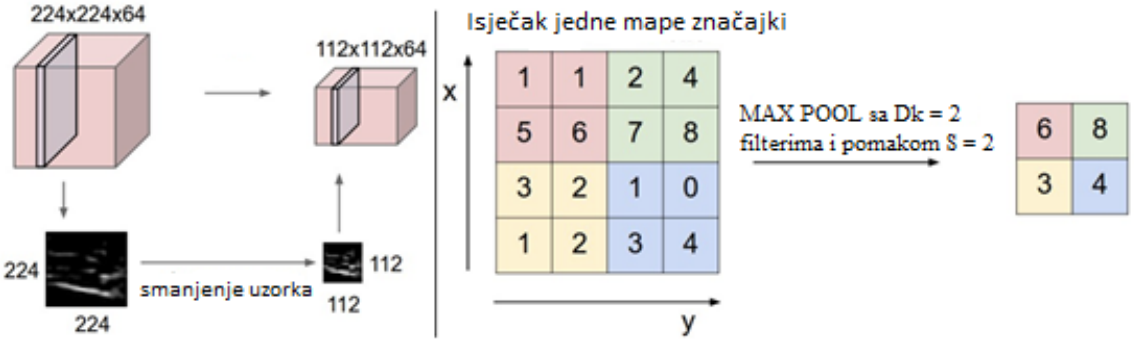
Slika 9 – Mape značajki kroz slojeve (Izvor A. Dertat, 2017. [44])

2.6. Sloj smanjenja uzorka

Sloj smanjenja uzorka (eng. *pooling layer, POOL*) se obično smješta nakon konvolucijskog sloja. Dodatno, moguće je *pooling* staviti na početak slike ukoliko se žele smanjiti dimenzije slike. Lecun i sur. (1998 [1]) naziva ovo *subsampling layer*, u novije vrijeme prihvaćena nomenklatura je *pooling layer*. Slično kao konvolucijski sloj koji koristi *kernele*, sloj smanjenja uzorka ima jedinku sa receptivno polje nad kojim se izvršavaju funkcije. Jedinka prima skup neurona (ulaznih elemenata) i daje samo jedan neuron na taj način smanjuje kompleksnost podataka.

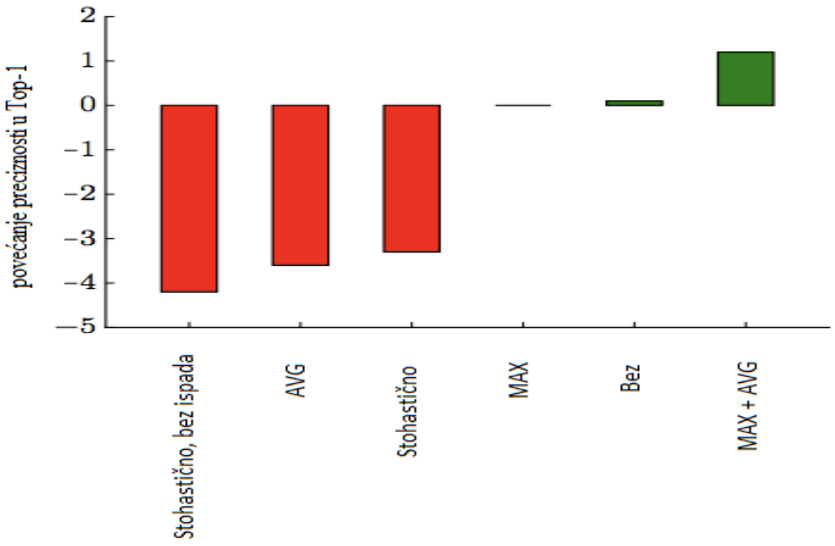
Uloga ovog sloja je smanjiti uzorak tako što se iterativno primjenjuje funkcija nad receptivnim poljem sa pomakom S > 1. Obično je veličina receptivnog polja jednaka pomaku što dovodi do maksimalnog smanjenja uzorka bez gubljenja informacija iz ulaza na način da su neki elementi preskočeni. Lecun i sur.(1998 [1]) koriste 2x2 veličinu receptivnog polja te pomak S = 2, time kroz kontinuiranu primjenu *poolinga* nema preklapanja informacija. Ukoliko se koriste ti parametri veličina izlaza je smanjena za točno pola redaka i stupaca. Također navode da prilikom računanja, na rezultat jedinke valja dodati multiplikativni koeficijent te dodati *bias*, koji se oboje mogu trenirati. Taj koeficijent i *bias* kontroliraju učinak aktivacijske funkcije (u njihovom slučaju, *sigmoidna* funkcija). Ako je koeficijent mali, onda jedinka

funkcionira linearno i pritom samo zamagli informacije. Nasuprot, ako je koeficijent velik, jedinke smanjenja uzorka se mogu promatrati kao da izvršavaju “ili” ili “i” operacije ovisno o vrijednosti *biasa*. (Lecun i sur., 1998 [1])



Slika 10 – Smanjenje uzorka (prevedeno sa A. Karpathy, 2011 [32])

Funkcije u ovom sloju su najčešće AVG ili MAX. Očigledno, AVG računa prosjek neurona iz receptivnog polja dok MAX traži najvišu vrijednost i nju daje kao rezultat. Postoje još *stohastički pooling* [35], *LP-Norm pooling* [36] te *Tree-Gated pooling* [37]. C.-Y. Lee i sur (2015) [37] preporučuju korisiti sumu *AVG i MAX* funkcija, a J.T. Springenberg i sur (2014) [38] preporuča kompletno preskočiti sloj smanjenja uzorka i zamijeniti ga sa korištenjem konvolucijskog sloja sa većim pomakom. Mishkin i sur (2016) [11] su testirali ove ideje i njihov rezultat se može vidjeti na grafu ispod (slika 10).



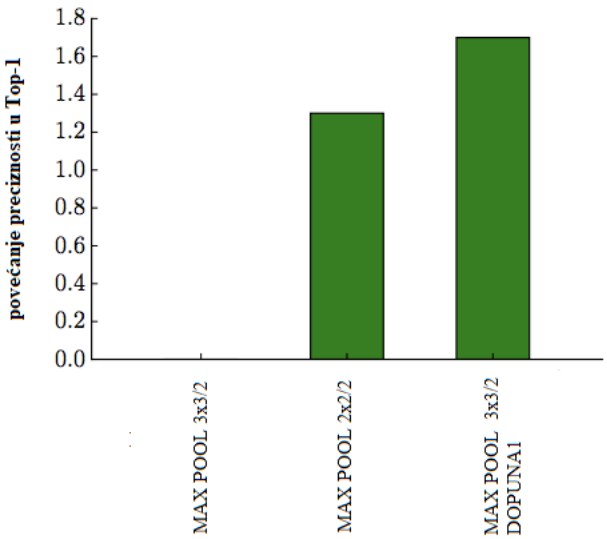
Slika 11 – Performanse metoda smanjenja uzorka (prevedeno sa Mishkin i sur., 2016 [11])

Njihovo objašnjenje je da suma AVG i MAX postiže najbolje performanse zbog toga što *Max pooling* donosi selektivnost i invarijantnost (nepromjenjivost u različitim uvjetima), također potvrđeno od strane M. Riesenhuber i T. Piggio (1999) [40], dok *AVG pooling* čuva gradijent ulaza tako što ne odbacuje *sve osim jednog ulaza* kao što to radi *Max pooling*.

2.6.1. Receptivno polje sloja smanjenja uzorka

Receptivno polje sloja smanjenja uzorka je još jedan dio slagalice. Naravno, za pomak S = D_p, veće receptivno polje znači veće gubljenje informacija. Sloj smanjenja uzorka je potreban kako bi se smanjile računalne potrebe treniranja, te također pomaže pri postizanju invarijantnosti (eng. *invariance*). Invarijantnost predstavlja veće šanse prepoznavanja objekta kroz različite uvjete (rotacija, pomak, veličina). Međutim, preveliko receptivno polje prirodno vodi do degradacije informacija. U praksi je najčešće korištena dimenzije D_p = 2 sa istim pomakom S = 2. Krizhevsky i sur (2012) [10] tvrde da je preklapajući D_p = 3 te S = 2 superioran nad D_p = 2.

Mishinkin i sur (2016) su testirali ove tvrdnje i zaključi su da D_p = 2 bez preklapanja (dakle S = 2) pruža bolje performanse od D_p = 3, međutim ukoliko se koristi dopunjavanje onda je D_p = 3 bolji. Navodno, 3x3 *pooling* sa dopunjavanjem vodi ka većem izlazu što znači da su performanse bolje zbog većeg izlaza međutim tada je kompleksnost računanja veća.



Slika 12 – Performanse receptivnih polja smanjenja uzorka (prevedeno sa Mishkin i sur., 2016 [11])

2.7. Potpuno spojeni sloj i kategorizacija

Potpuno spojeni sloj (eng. *fully connected layer*, FC) predstavlja stvarnu klasičnu neuronsku mrežu u kojoj su svi čvorovi povezani sa svima iz prethodnog sloja. Može se reći da je njihovo receptivno polje maksimalne moguće veličine, međutim u okviru potpuno spojenog sloja nema previše smisla pričati o receptivnim poljima. Receptivna polja su vezana uz vizualizaciju dok su neuroni potpuno spojenog sloja vezani uz *razumijevanje* krajnje konvoluirane i *poolirane* slike.

Izlaz konvolucijskih slojeva predstavlja značajke više razine (kompleksnije značajke) u podacima. Iako se izlaz mreže može povezati na izlazni sloj, dodavanje potpuno spojenog sloja je efikasan način učenja nelinearnih kombinacija tih značajki.

J. Wu (2017) [18], izlaz konvolucijskih slojeva sadrže distribuirane reprezentacije (moguće da se odnosi na brojne mape značajki) ulazne slike, i cilj je iskoristiti sve te značajke kako bi se izgradile nove značajke sa visokim sposobnostima, te potpuno spojeni sloj je koristan za ovu ulogu. Lecun i sur. (1998) objašnjava da se potpuno spojeni sloj koristi kao klasifikator. Kao takav, mreža se može sastojati samo od potpuno spojenog sloja i kao ulaz imati izvorni normaliziranu sliku, ali LeCun dalje navodi da postoje nedostaci poput toga što mreža ne bi sadržavala translatornu invarijantnost i broj parametara za treniranje bi bio preveliki za efikasno treniranje.

Kako bi se izlaz CONV/POOL sloja mogao spojiti na FC sloj, on se pretvara iz 3D dimenzija u niz vrijednosti zato što je FC jednodimenzionalna mreža. Zadnji sloj konvolucijske neuronske mreže je neka vrsta funkcije koja mapira FC izlaz u vjerojatnosti za kategorije. Stoga, veličina izlaza zadnjeg sloja je jednaka broju kategorija. U zadnje vrijeme, kao zadnji sloj se najčešće koristi *softmax* aktivacijska funkcija (AlexNET [10], Overfeat [12], VGG [13], GoogLeNet [15] itd.).

3. Pregled novijih arhitektura

Konvolucijske neuronske mreže postoje još od 90-ih godina prošlog stoljeća, međutim tek su u zadnje vrijeme dobili na značaju i postale tzv. *State of the art* algoritmi u računalnom vidu. U ovom odlomku pokrivamo razvoj arhitekture u zadnje vrijeme i razglabamo njihove doprinose i utjecaj na cijelo područje konvolucijskih neuronskih mreža. Prva mreža koja je poznata po imenu je bila LeNet iz LeCun i sur. (1998) [1]. U novije vrijeme s porastom popularnosti CNN tehnike, arhitekture su počele dobivati svoja imena. Neke od važnih arhitektura:

- 2012 - AlexNet
- 2014 - OverFeat
- 2014 - VGG
- 2014 - Network In Network
- 2015 - googLeNet (inception)
- 2015 - ResNet
- 2015 - Yolo
- 2016 - Squezze
- 2016 - Enet
- 2016 - FractalNet
- 2017 - Xception

3.1. AlexNET

AlexNet je naziv CNN mreže koja je popularizirala koncept konvolucijskih mreža. Tvorci *AlexNet*-a su A. Krizhevsky, I. Sutskever te G. E. Hinton (2012) [10]. Mreža je 2012. na natjecanju *ImageNet Large Scale Visual Recognition Challenge* imala performanse 15,3% pogreške u top-5 predikcija, bolje za 10,8% od sljedećeg najboljeg rješenja. Koristili su *kernele* veličine Dk = 9 i Dk = 11. Ovim djelom, *AlexNet* je označio početak nove ere za računalnu viziju. Glavne značajke koje su Krizhevsky i suradnici donijeli na stol su bile:

- Treniranje sa grafičkim procesorskim jedinkama
- Korištenje *ReLU* aktivacijske funkcije umjesto tada opće prihvaćenih *tanh* i *sigmoid*
- Preklapajući *max pooling*, D_p = 3, S = 2, povećava performanse top-1 pogreške za 0.4% te smanjuje šanse za preprilagođenost (eng. *overfitting*) odnosno prenaučenost
- Umnožavanje podataka prilikom treniranja
    - Translacija i horizontalna refleksija slika
    - Dok GPU trenira neuronsku mrežu, CPU umnožava podatke
    - Primjenjuju PCA na skup podataka za treniranje - analiza glavnih komponenti (eng. *principal component analysis*) čime smanjuju top-1 pogreške za više od 1%
- Implementiraju tzv. ispad (eng. *dropout*) na prva dva FC sloja koji su zaduženi za kategorizaciju slike, povećava kompleksnost treniranja bar za duplo jer se konvergiranje sporije događa međutim visoko smanjuje šanse za preprilagođenost

3.2. Overfeat

Mreža koja je derivacija *AlexNet*-a. Predstavlja novi model konvolucijskih neuronskih mreža koji podržavaju klasifikaciju, lokalizaciju i detekciju smještene u istu mrežu. Pruža posljednje dostignuće (eng. *state of the art*) u smislu performansi za detekciju objekata. Detalji mreže dostupni u radu P. Sermaneta i sur. (2014) [12].

3.3. VGG

K. Simonyan i A. Zisserman su krajem 2014-te razvili novu arhitekturu sa vrlo uspješnim rezultatima - VGG što znači Grupa Vizualne Geometrije (eng. *Visual Geometry Group)* [13]. Oni su bili prvi koji su počeli koristiti mnogo manje *kernele* veličine Dk = 3 u svakom konvolucijskom sloju i nizali više konvolucijskih slojeva zaredom bez primjene sloja za smanjenje uzorka. Ovo se čini da prkosi principima *LeNet*-a gdje su velike konvolucije korištene kao sredstva za hvatanje sličnih značajki u slici. Umjesto *kernela* dimenzija Dk = 9 i Dk = 11 u *AlexNet*-u, *kerneli* su postali manji.

Najveći doprinos VGG arhitekture je uvid da višestruke 3x3 konvolucije zaredom mogu simulirati ponašanje odnosno efekt većih receptivnih polja, npr 5x5 ili 7x7. Ove ideje su također iskorištene u još novijim arhitekturama poput *Inception* i *ResNet*.

VGG mreža koristi višestruke 3x3 konvolucijske slojeve kako bi prepoznala kompleksne značajke. Na slici ispod je vidljivo da blokovi 3, 4 i 5 sadržavaju tri puta konvolucijski sloj u nizu za C i D konfiguraciju, te čak četiri niza za konfiguraciju E. Ovo iznosi mnogo parametara za računanje te moći za učenje. Njihov zaključak je da dubina mreže može povoljno utjecati na performanse.

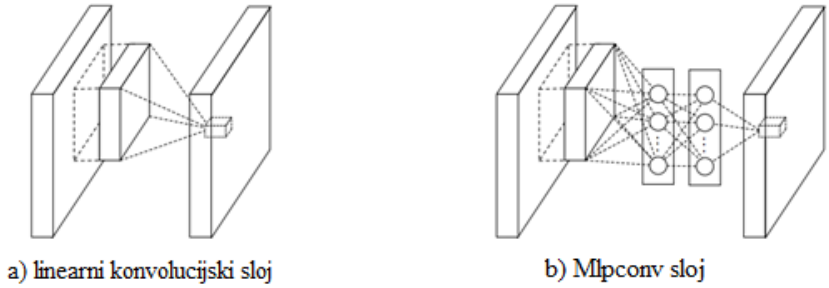
Tablica 2 – Arhitektura VGG mreže

ConvNet Konfiguracija					
A	A-LRN	B	C	D	E
11 slojeva	11 slojeva	13 slojeva	16 slojeva	16 slojeva	19 slojeva
Ulaz (224 X 224 RGB slika)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

(prevedeno sa K. Simonyan i A. Zisserman, 2014 [13])

3.4. Mreža u mreži - NiN

M. Lin i suradnici (2014) [16] su imali odličan i jednostavan uvid da korištenje konvolucije 1x1 može pružiti više *kombinatoričke* moći za značajke u konvolucijskim slojevima. Arhitekturu su nazvali „Mreža u mreži“ (eng. *Network In Network, NIN)*. U radu naglašavaju da konvolucijski sloj uobičajeno koristi linearne *kernele* koji su popraćeni sa nelinearnom aktivacijskom funkcijom koja skenira ulaz. Umjesto toga, oni su napravili mikro neuronske mreže sa kompleksnijim strukturama kako bi apstrahirali podatke kroz receptivna polja.



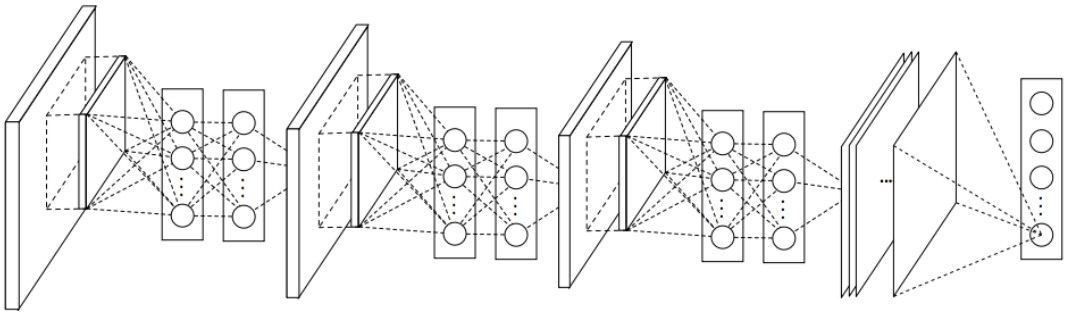
Slika 13 – Višeslojni *perceptron* kao konvolucijski sloj (prevedeno sa M. Lin i sur., 2014 [16])

Novu metoda koristeći mikro neuronsku mrežu nakon konvolucije su nazvali *Mlpconv* što označava višeslojni *perceptron*[1] + konvolucija. Oboje, i linearni konvolucijski sloj i *mlpconv* sloj mapiraju lokalno receptivno polje na izlaz gdje su značajke. Princip je isti, samo umjesto primjene *kernela* nad receptivnim poljem, oni predlažu primjenu mikro neuronske mreže (mreža je dijeljena za sva lokalna receptivna polja jedne mape značajki). Autori dalje tvrde da konvolucijski kernel pruža nisku razinu apstrakcije te da zamjena konvolucijskog *kernela* sa mikro neuronskom mrežom kao boljeg nelinearnog funkcijskog *aproksimatora* pruža višu razinu apstrakcije. Struktura NIN mreže je predstavljena na slici ispod. Dodatna razlika između klasične CNN mreže i NIN mreže je što NIN nema slojeve za smanjenje uzorka nakon konvolucijskih slojeva. Također, NIN nema FC slojeve već ih zamjenjuje sa novom strategijom,

[1] Višeslojni *perceptron* (eng. *multilayer perceptron*, MLP) je čest naziv za klasu *feedforward* neuralnih mreža sa minimalno tri sloja i koje koriste unatražno rasprostiranje za treniranje

tzv. globalni AVG *pooling*. Umjesto da dodaju FC slojeve na svaku mapu značajki iz zadnjeg CONV ili POOL sloja, oni uzimaju prosjek svake mape značajki i rezultirajuće podatke direktno prosljeđuju *softmax* sloju.

Prednost koju navode za globalni AVG *pooling* preko FC slojeva je da je više prirodna konvolucijskoj strukturi tako što prisiljava međusobnu spregu između mapa značajke i kategorija. Dodatno, ne postoje parametri koji se mogu učiti u ovom zadnjem sloju pa je spriječena preprilagođenost na ovom dijelu mreže. Također, globalni AVG *pooling* računa sumu iz prostorne informacije čime je više otporan na prostorne translacije (poboljšava translatornu invarijantnost iz poglavlja 3.1.) . Autori navode da je učinak *mlpconv* sloja ekvivalentan CONV sloju sa *kernelima* veličine Dk = 1.



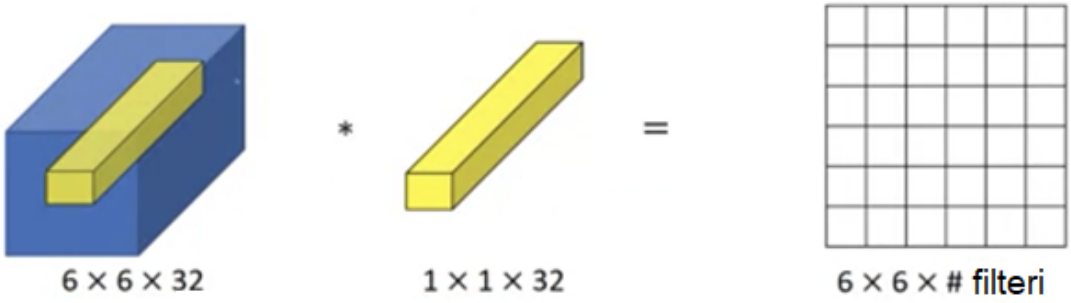
Slika 14 – Arhitektura konvolucijske neuronske mreže pri korištenju *NiN* pristupa (izvor M. Lin i sur., 2014 [16])

3.5. Inception i 1x1 konvolucije

C. Szegedy i suradnici (2015) [17] predlažu arhitekturu nazvanu *Inception* koja u vrijeme izlaska daje *state of the art* performanse za klasifikaciju i detekciju na već spomenutom *ImageNet* natjecanju, ovog puta 2014-te godine. U mreži su iskoristili saznanja iz NIN mreže (poglavlje 4.4), da je *mlpconv* jednak običnom CONV sa $D_k = 1$ *kernelom*.

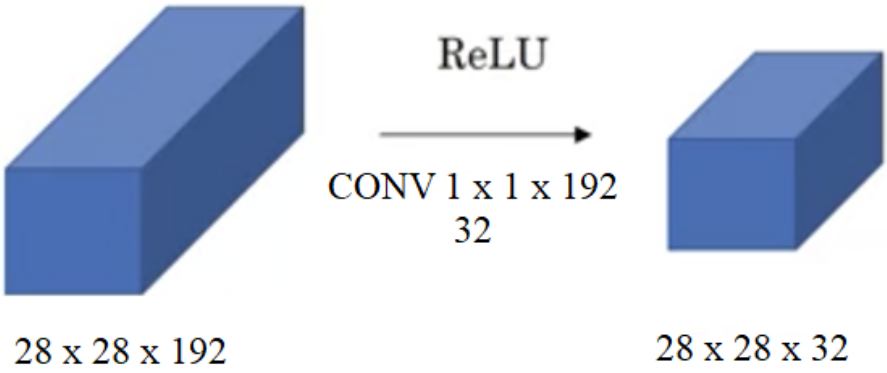
Isprva nije jasno kako konvolucije sa 1x1 mogu doprinijeti bilo čemu jer su to linearne funkcije a receptivno polje je veličine jednog elementa. Npr. za neku ulaznu matricu M_i dimenzija 16x16 i njen element $a_{i1,1}= 4$ te *kernel* K sa $a_{k1,1}= 2$ (jedini element matrice) dobit ćemo izlaznu matricu M_o i njen element $a_{o1,1} = 8$ ($a_i * a_k$). Međutim, u praksi inače nemamo samo

jednu mapu značajki (ulaz u sloj) već više njih u nizu, često čak stotine. To znači da su realne dimenzije ulazne matrice Mi, ukoliko imamo 256 mape značajki, 16x16x256 te dimenzije *kernela* 1x1x256! Kada se 1x1x256 *kernel* konvoluira kroz ulaz, za svaki korak primjene on koristi pojedinačni element $a_{x,x}$ ulaznog vektora ali od svih 256 mapa. Drugim riječima 1x1 *kernel* primijenjen na ulazni vektor kombinira svih 256 mapa značajki u jednom koraku i tako ekstrahira korisne podatke iz njih. Konvolucijom 1x1 *kernela* se kombinira 256 mapa značajki za svaki element ulaznog vektora. Izlaz takvog postupka je mapa značajki veličine 16x16 koja sadržava informacije iz svih mapa značajki vektora.



Slika 15 – Primjena 1x1 konvolucije (izvor – A. Ng, 2018 [43])

Očite prednosti nad korištenjem 1x1 konvolucija umjesto 3x3 konvolucija nema na prvi pogled. Međutim, bitno je zapaziti da svaka 1x1 konvolucija sadržava informacije od svih mapa značajki, te ako 1x1 sloj ima 64 kernela, rezultirajući izlaz će biti niz mapa značajki prikazan kao 16x16x64. To je glavna prednost 1x1 konvolucija, što mogu uspješno sažeti veličinu uzorka bez gubljenja preciznosti kao što to npr. čini *POOL* sloj. Razlika u smanjenju uzorka je što 1x1 konvolucija smanjuje uzorak u trećoj dimenziji dok *POOL* sloj smanjuje uzorak u prvoj i drugoj dimenziji. Također, s obzirom na malu veličinu *kernela* 1x1, cijena računanja je vrlo niska.



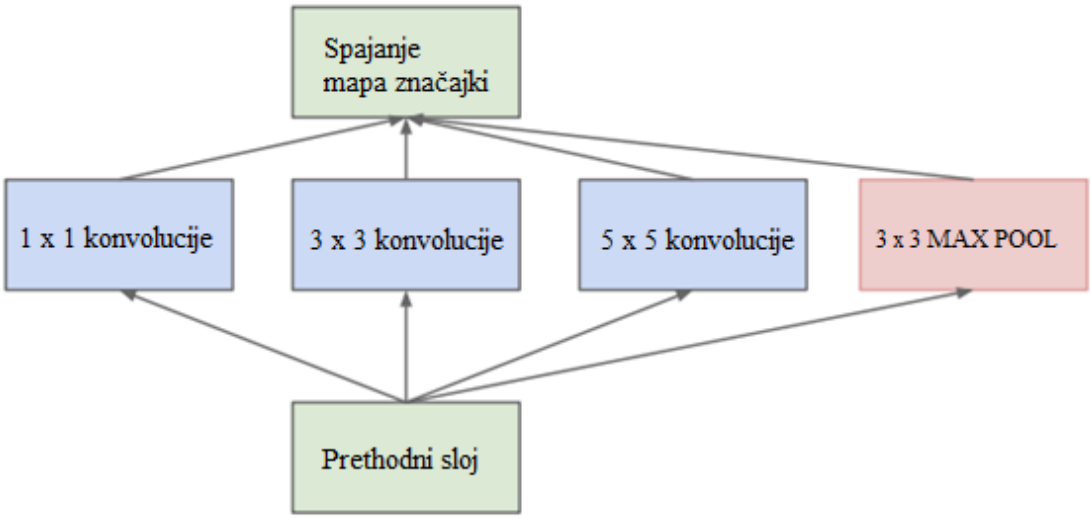
Slika 16 – Smanjenje uzorka sa 1x1 konvolucijom (izvor – A. Ng, 2018 [43])

3.5.1. GoogLeNet struktura

Glavno obilježje ove arhitekture je poboljšana iskorištenost računalnih resursa unutar mreže (misli se na bolje performanse sa jednakim brojem parametara za treniranje). Kroz oprezno izgrađen dizajn, povećali su dubinu i širinu mreže dok je u isto vrijeme količina računanja ostala ista. Kako bi optimizirali kvalitetu, odluke pri konstruiranju arhitekture su bile počinjene na osnovu *Hebbianovog*[2] principa. U radu su predložili više mreža, međutim najvažnija je bila *GoogleLeNet*, mreža duboka 22 sloja.

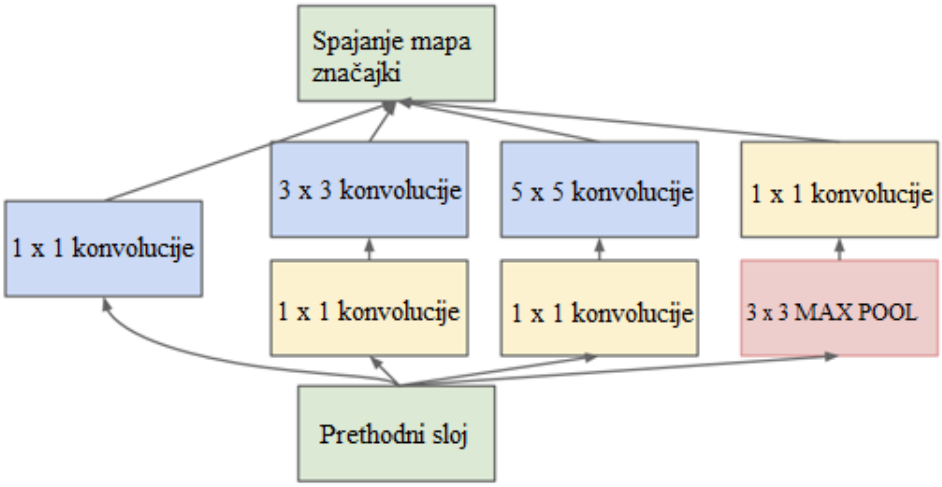
C. Szegedy i suradnici (2015) izgrađujući *Inception* model su htjeli naći lokalne značajke i ponoviti ih prostorno. To su postigli sa konstruiranjem konvolucijskog sloja koji ima više paralelnih operacija sa različitim veličinama filtera, čiji se mape značajki spajaju u jedan izlazni vektor formirajući ulaz za sljedeći sloj. Specifično, njihov konvolucijski odnosno *Inception* sloj je sadržavao 1x1, 3x3 te 5x5 *kernele* i dodatnu *MAX pooling* operaciju - vidljivo na slici ispod. Primjenjujući *kernele* različitih veličina na isti ulaz pronalazili su značajke međutim kroz različite veličine receptivnih polja.

[2] Hebbianova teorija - teorija o neuronima koja tvrdi da kada neuron A često uzrokuje da se neuron B okine, tijelo će kroz proces rasta ili metaboličku promjenu promijeniti jednu ili obe stanice na način da se efikasnost *A neurona da okine B* poveća



Slika 17 – Inception modul (preuzeto sa C. Szegedy i suradnici, 2015 [17])

Nadograđeni *Inception* model je takav da su prije 3x3 i 5x5 konvolucije koristili 1x1 konvoluciju kako bi postigli smanjenje veličine uzorka s čime je vrijeme treniranja mreže znatno smanjeno. Također su primijenili 1x1 konvoluciju nakon *MAX pooling* operacije kako bi smanjili broj mapa značajki sa npr. 512 na 64 ili 832 na 128.



Slika 18 – Inception modul sa smanjenjem dimenzija (preuzeto sa C. Szegedy i sur., 2015 [17])

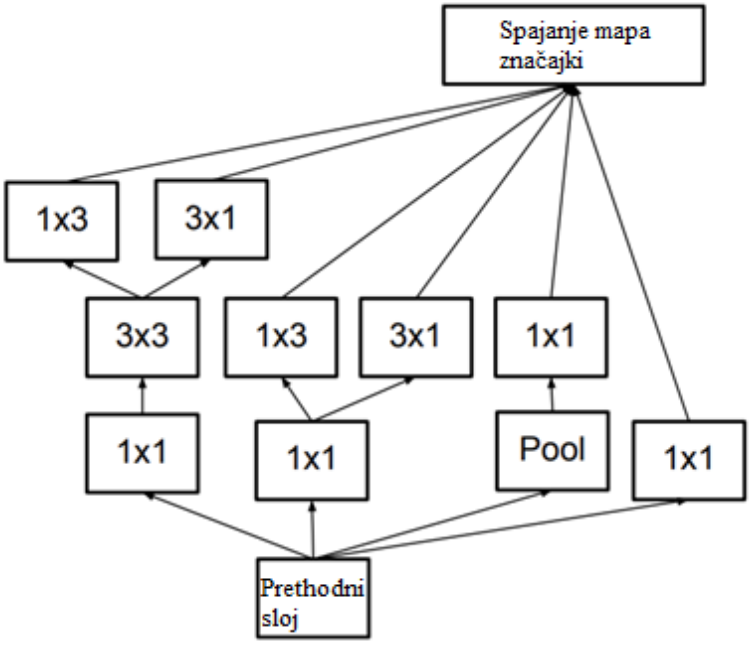
3.6. Inception v2 i v3

2015-te S. Ioffe i C. Szegedy [19] stvaraju novu verziju *GoogLeNet*-a koja je kasnije popularizirana kao *Inception v2*. U radu koriste tehniku tzv. grupne normalizacije (eng. *batch normalization*). Radi se o tehnici tzv. izbjeljivanja (eng. *whitening*) podataka kako bi se adresirao problem o *biasu* koji se perzistira i pojačava kroz slojeve ukoliko prosjek aktivacijskih vrijednosti značajno odstupa od nule.

Tehnika grupne normalizacije se sastoji od računanja prosjeka i standardne devijacija svih mapa značajki na izlazu nekog sloja i normaliziranja vrijednosti tih značajki s tim. Ovo postiže da neuronske mape imaju vrijednosti izlaza u istom rasponu te je prosjek aktivacijskih vrijednosti jednak nuli. Time pomaže pri treniranju budući da sljedeći sloj više ne uči utjecaj *biasa* na prethodne slojeve već je fokus potpuno na kombiniranju značajki. Također, za razliku od *Inception v1, Inception v2* umjesto 5x5 *kernela* koristi dva uzastopna 3x3 *kernela*.

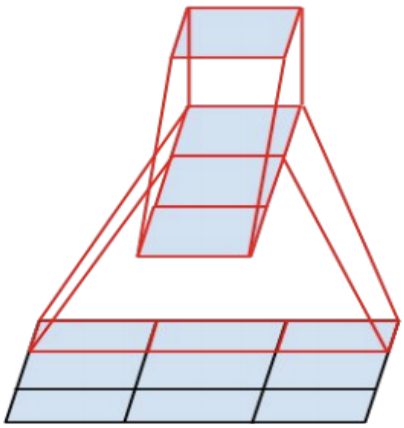
Krajem iste godine, Szegedy i sur (2015) [20] predstavljaju *Inception* v3 u kojoj se koncentriraju na arhitekturu u cijelosti. Neke od glavnih preporuka su

- Maksimizirati količinu informacija koje ulaze u CONV sloj, oprezno sastavljati dimenzije slojeva, balansirati dubinu i širinu, povećati količinu mapa značajki prije svakog sloja smanjenja uzorka
- Kada je dubina povećana, broj prepoznatih značajki je također sistemski povećana
- Koristiti *kernele* veličine Dk = 3, s obzirom na to da se ponašanje *kernela* Dk = 5 i Dk = 7 mogu simulirati primjenom višestrukih Dk = 3 *kernela*
- Izbjegavati preveliko smanjenje ulaznih podataka primjenom 1x1 konvolucija prerano u fazi treniranja mreže



Slika 19 – Arhitektura Inception v3 (prevedeno sa Szegedy i sur, 2015 [20])

Još jedna novina koju *Inception* v3 sa sobom donosi je produbljivanje ideje VGG tima. Originalnu ideju zamjene 5x5 ili 7x7 sa 3x3 *kernelima* proširili su na zamjenu 3x3 *kernela* sa dvoje uzastopnih *kernela* asimetričnih dimenzija 1x3 te 3x1, čime se dobiva ušteda vremena za računanje za 33%. Nadalje, pokazuju da se ovo može generalizirati na 1xn i nx1 slučaj. Rezultati koje su dobili pokazuju da korištenje takvih asimetričnih *kernela* nije preporučljivo pri početku CNN-a te daje dobre rezultate kada je broj mapa značajki između 12 i 20.



Slika 20 – Uzastopni *kerneli* asimetričnih dimenzija (preuzeto sa Szegedy i sur, 2015 [20])

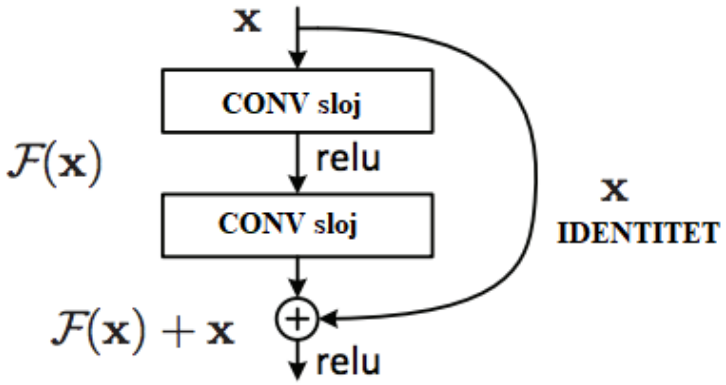
3.7. ResNet

U skoro isto vrijeme kada je *Inception* v3 napravljen, nova mreža je predstavljena od strane K. He i suradnika (2015) [21] nazvana *ResNet*. Autori u dubinu analiziraju pozitivan utjecaj dubine mreže na performanse i zaključuju da se taj trend ne može zadržati nakon određene razine dubine. U testovima su pokazali kako nakon određene granice, dodavanje više slojeva vodi ka višoj stopi greški prilikom treniranja. Kako bi izbjegli ovaj efekt degradacije autori predlažu novu tehniku.

Glavna tehnika primijenjena u ovoj mreži je tehnika rezidualnih blokova (eng. *residual blocks*). Tehnika je prilično jednostavna no njene implikacije su dosta široke. Najjednostavnije rečeno, kako bi stvorili rezidualni blok potrebno je na ulaz x+1 sloja, uz normalni izlaz sloja x, spojiti i ulaz sloja x. Formalno, rezidualni blok se može prikazati kroz formulu

$$y = F(x, \{Wi\}) + x.$$

Ovdje su x i y ulaz i izlaz promatranih slojeva respektivno. Funkcija F(x, {Wi}) predstavlja slojeve koji sudjeluju u treniranju, slojeva može biti više, najčešće i > 1. Ukoliko dimenzije F i x nisu iste, to se može riješiti primjenom dopunjavanja tako da je y uniforman izlaz iz bloka.



Slika 21 – Tehnika rezidualnih blokova (prevedeno sa K. He i sur., 2015 [21])

3.8. YOLO

J. Rendon i sur. (2015) [22] dovode novost u svijet računalnog vida. Njihova konvolucijska neuronska mreža je sposobna načiniti i detekciju i klasifikaciju objekta u isto vrijeme. Ovo je značajno iz razloga što se može primijeniti na realni problem prepoznavanja znakova koji nisu pojedinačni već više-riječni, ili potencijalno čak slika cijele stranice teksta. Prethodni sustavi na sličnu temu za detekciju objekata su koristili klasifikatore kako bi proveli detekciju. Kako bi prepoznali određeni objekt, sustavi uzmu klasifikator za taj objekt i provlače ga kroz dijelove ulazne slike, pritom povećavajući i smanjujući dimenzije pojedinog skeniranja. Najnoviji pristupi poput R-CNN (konvolucijska neuronska mreža sa povratnim vezama) koriste metode preporuke područja kako bi generirali granične kutije u slici i onda pokreću klasifikatore nad tim preporučenim kutijama. Nakon klasifikacije koriste procesiranje da rafiniraju granične kutije i eliminiraju dvostruke detekcije koje se nužno pojavljuju jer se granične kutije preklapaju. Ovakve sustave je teško optimizirati jer se svaka individualna komponenta mora trenirati zasebno.

YOLO rješava problem iz druge perspektive, detekciju objekata tretiraju kao regresijski problem na razdvojene granične kutije i uz njih vežu vjerojatnosti klasa. Jedna neuronska mreža predviđa granične kutije i vjerojatnosti klasa direktno iz cijele slike kroz samo jednu evaluaciju. Zbog toga je arhitektura nazvana YOLO - “samo gledaš jednom” (eng. *You Only Look Once)*. YOLO se trenira na cijelim slikama i direktno optimizira performanse detekcije jer postoji samo jedna komponenta u sustavu.

Postizanje YOLO sustava se izvodi u više koraka. Prvo se konvolucijski slojevi treniraju zasebno za klasifikaciju. Zatim se model mijenja kako bi izvršavao i detekcije. Dodaju četiri konvolucijska sloja i dva potpuno spojena sloja sa nasumično inicijaliziranim težinama. Posljednji sloj predviđa i vjerojatnosti klasa i koordinate graničnih kutija. Prednost YOLO i njegovih nasljednika nad ostalim sustavima je brzina predviđanja koja doseže do 67 puta po sekundi (J. Redmon and A. Farhadi, 2017 [23]).

3.9. Ostale arhitekture vrijedne spomena

Osim navedenih radova, postoji još mnoštvo njih koji nisu donijeli velike inovacije i nisu imali toliki utjecaj. Od njih izdvajam par najvrjednijih.

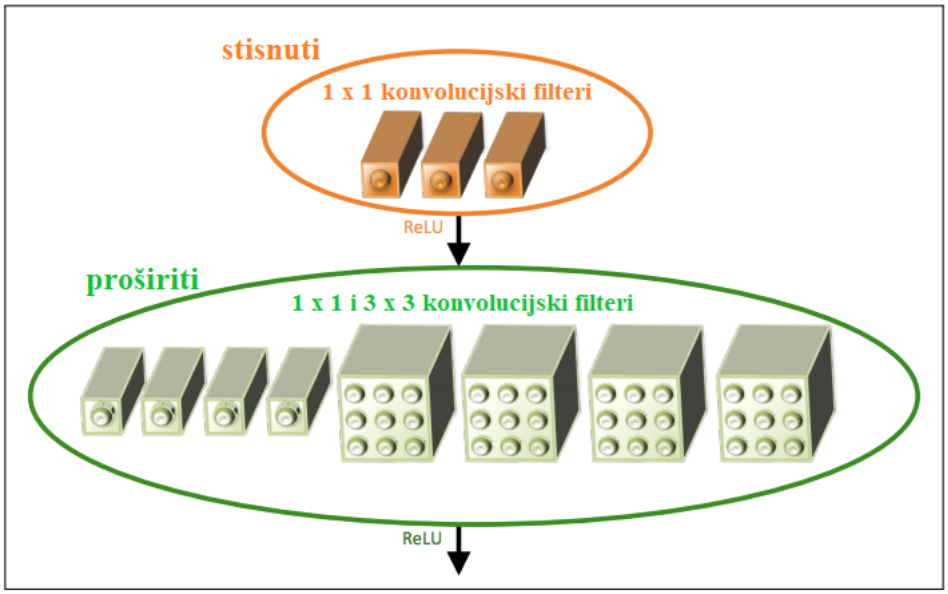
3.9.1. Squezze net

Iandola i sur (2016) [2] uzimaju mnoge koncepte predstavljene u *ResNet* i *Inception* arhitekturama i pokazuju da bolji dizajn arhitekture može rezultirati sa preciznom a vrlo kompaktnom mrežom bez upotrebe kompleksnih algoritama kompresije.

Tablica 3 – Usporedba *AlexNet* i *SquezzeNet*

Mreža	Top-5 preciznost ImageNet	Parametri modela	Veličina modela	Nakon duboke kompresije
***SqueezeNet* [2]**	80.3%	1.2M	4.8MB	0.47MB
***AlexNet* [10]**	80.3%	60M	243MB	6.9MB

Predstavljaju tzv. vatreni modul (eng. *fire modul*) koji se sastoji od dvije apstraktne operacije - stiskanje i proširenje. U operaciji stiskanja, podaci se smanjuju koristeći 1x1 konvolucije, zatim se to prosljeđuje operaciji proširenja koja je kombinacija 1x1 i 3x3 konvolucija. *SquezzeNet* se sastoji samo od ovih operacija koje su nizane i nakon kojih slijedi rezidualni blok iz 4.7.



Slika 22 – Vatreni modul *SquezzeNet* mreže (prevedeno sa Iandola i sur, 2016 [2])

3.9.2. Enet

A. Paszke i sur. (2016) [29] stvaraju *Enet*. Kombiniraju sve značajke novijih arhitektura u jednu efikasnu i laganu mrežu koja koristi vrlo malo parametara i ostvaruje *state of the art* rezultate. Mrežu su koristili kao *enkoder* i *dekoder*. *Enkoder* je obična CNN arhitektura za kategorizaciju dok je *dekoder* mreža dizajnirana za propagiranje kategorija natrag u originalnu sliku. Time postižu semantičku segmentaciju slike u realnom vremenu što ima brojne primjene u mobilnim aplikacijama.

3.9.3. Xception

F. Chollet (2017) [41] se nastavlja na rad *Inception* gdje generalizira pristup *Inception* arhitekture i predlaže promjene. Obični *Inception* modul prvo primjenjuje *kernele* 1x1 pa onda 3x3 međutim Chollet je obrnuo redoslijed i time postigao povećanje performanse. Razlika je u tome što primjenjuje *ReLU* samo jednom nakon zadnje konvolucije u modulu dok starija arhitektura primjenjuje *ReLU* u obe konvolucije.

4. Implementacija

Implementacija konvolucijske neuronske mreže u rješavanju problema prepoznavanja rukom pisanih znakova je izvršena u programskom jeziku Python. Pomoćne biblioteke su Keras i Tensorflow koje sadrže pojednostavljeno sučelje za rad sa neuronskim mrežama.

4.1. Upravljanje podacima za treniranje

Mreža je trenirana nad MNIST skupom podataka koji sadržava rukom pisane znamenke. Prije treniranja je bilo potrebno učitati, sortirati i podijeliti podatke u skupove. Ulazni podaci su organizirani u CSV formatu gdje svaki redak ima 785 stupaca. Prvi stupac predstavlja oznaku (eng. *label*) znamenke, a ostalih 784 stupaca predstavljaju piksele slike. Podaci se učitaju koristeći *pandas* biblioteku. Zatim se razdvoje oznake i pikseli u dva različita polja.

```
train = pd.read_csv("mnist.csv")
x_train = (train.ix[:, 1:].values).astype('float32')
y_train = train.ix[:, 0].values.astype('int32')
```

Pikseli se normaliziraju tako što se podijele sa najvećom mogućom vrijednošću, 255, te se naposljetku slijedni podaci 784 piksela konvertiraju u dvodimenzionalnu matricu veličine 28x28 što odgovara stvarnoj slici znamenke.

```
x_train = x_train / 255.0
X_train = x_train.reshape(x_train.shape[0], 28, 28, 1)
```

Ulazni podaci su podijeljeni u dva skupa, skup za treniranje (učenje), te skup za validaciju u omjeru 9:1 respektivno. Funkcija za podjelu podataka se zove *train_test_split*, korištenje vidljivo ispod:

```
train_test_split(X_train, y_train, test_size = 0.1)
```

4.2. Model mreže i treniranje

Nakon tog koraka, kreiran je model koristeći *Keras* biblioteku. *Keras* predstavlja pojednostavljeno sučelje za rad sa *TensorFlow*-om i time nudi moćnu biblioteku koja je laka za korištenje. Najveći problem pri implementiranju konvolucijske neuronske mreže je primijeniti

teoretsko znanje prilikom izgrađivanja arhitekture, sva programska podrška već postoji i na istraživaču je samo ponovno upotrijebiti već dostupne klase iz navedenih biblioteka.

```
epochs=30
act='elu'
init='he_normal'
learn_rate_perc=0.95
padd='same'

model = Sequential()
model.add(Conv2D(64, 3, 3), activation=act, padding=padd))
model.add(Conv2D(32, (1, 1), activation=act))
model.add(MaxPool2D((2, 2)))
model.add(Dropout(0.20))
model.add(Conv2D(128, (3, 3), activation=act,padding=padd))
model.add(Conv2D(48, (1, 1), activation=act,padding=padd))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Conv2D(256, (3, 3), activation=act,padding=padd))
model.add(Dropout(0.1))
model.add(Conv2D(64, (1, 1), activation=act,padding=padd))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation=act))
model.add(BatchNormalization())
model.add(Dropout(0.4))
model.add(Dense(10, activation='softmax'))
```

U *Keras* biblioteci, kreiran je tzv. Sekvencijalni model. Ta klasa se može tretirati kao lista slojeva, funkcijom *add* se dodaju različiti slojevi (*CONV, POOL, FC* i sl.). *Conv2D* je konvolucijski sloj, parametri uključuju količinu, veličinu, aktivacijsku funkciju te dopunu. Postoje još dodatni parametri koji nisu korišteni. *MaxPool2D* je sloj za smanjenje uzorka sa parametrom veličina receptivnog polja (dodatni parametar je *stride*), a *Dense* je potpuno spojeni sloj sa parametrima broj neurona i aktivacijska funkcija. Osim toga sloj za ispad *Dropout* prima samo postotak ispada, a sloj grupne normalizacije *BatchNormalization* ne sadržava parametre.

Globalni parametri (postavljeni od strane autora): *act* je aktivacijska funkcija koja je po pretpostavljenom podešena na *ELU*, *init* je vrsta inicijalizacije a odabrana je poboljšana *Xavier* inicijalizacija nazvana *he_normal*. Osim toga, naznačena je dopuna sa parametrom *padd* čija vrijednost *'same'* naznačuje modelu da dinamički primjeni dopunu na način da su dimenzije mapa značajki u ulazu i izlazi iz sloja jednake. Nakon što je model konstruiran i kompiliran koristeći `model.compile(…)`, korišteni su generatori slika iz *Keras* biblioteke za umjetno povećanje skupa za treniranje.

```
generator = ImageDataGenerator(
      rotation_range=15,
      zoom_range = 0.1,
      width_shift_range=0.1,
      height_shift_range=0.1
)
```

Taj generator je zatim proslijeđen funkciji koja pokreće treniranje. Postojeća slika iz originalnog skupa je promijenjena za svaki korak treniranja (slike se generiraju tijekom treniranja, a ne prije). Pritom je znamenka rotirana do 15 stupnjeva u nasumičnu stranu, približena ili udaljena do 10% te je cjelokupna slika pomaknuta do 10% horizontalno i vertikalno. Testirane su i više vrijednosti (25 stupnjeva rotacija, 15% pomak i približavanje) međutim budući da su tada slike previše odstupale od originalnih model nije mogao konvergirati ka globalnom maksimumu. U tom slučaju se također pojavio problem kod znamenki 6 i 9. Sa rotacijom do 25%, neke znamenke 6 i 9 koje su već napisane kurzivno su bile toliko rotirane da nije bilo razlike između njih što je vodilo do netočne klasifikacije i lošeg modela. Definirana je funkcija smanjenja razine učenja koja je zadana globalnim parametrom *learn_rate_perc* te je naposljetku započeto treniranje koristeći `model.fit_generator()`. Kada bi se treniranje odvijalo bez generatora slika, onda bi se koristio `model.fit()`.

```
smanjenje_razine_ucenja = LearningRateScheduler(lambda x: 1e-3 *
learn_rate_perc ** x, verbose=1)

model.fit_generator(
       generator.flow(X_train, Y_train, batch_size=batch_size),
       epochs=epochs, validation_data=(X_val, Y_val),
       verbose=1,
       steps_per_epoch=X_train.shape[0],
       callbacks=[smanjenje_razine_ucenja])
```

Treniranje se odvija u epohama. Jedna epoha znači jedan prolaz kroz cijeli skup za treniranje. Ovaj se postupak zatim ponavlja i u svakoj novoj epohi model postaje precizniji dok ne dostigne vrh performansi koji je omeđen arhitekturom. Treniranje je trajalo oko 120s po epohi, broj epoha je bio 30. Za treniranje je korištena osma generacija i5 procesora sa dvije jezgre i četiri dretve, ukoliko bi bila korištena grafička procesorska jedinka vrijeme trajanja bi se smanjilo do 30 puta ovisno o snazi. Na kraju treniranja model je testiran odnosno evaluiran koristeći `model.evaluate()` nad skupom za validaciju. Ova metoda evaluira model nad cijelim skupom i vraća ukupnu grešku i preciznost modela. Ukoliko se želi prepoznati pojedinačna znamenka, onda se treba koristiti `model.predict()`.

```
uk_greska, uk_preciznost = model.evaluate(X_val, Y_val, verbose=0)

predvidene_vrijednosti = model.predict(slike_znamenki)
```

Cijeli model je zatim zapisan na čvrsti disk, a uz to je također implementirano zapisivanje povijesti treniranja kako bi se kasnije mogli analizirati rezultati. Za automatizirano testiranje napravljena je funkcija koja izvršava sve prije nabrojene operacije (razdvajanje podataka na dva skupa, konstruiranje i kompiliranje modela, treniranje, validacija i zapis na čvrsti disk). Argumenti funkcije u *python* programskom jeziku mogu imati pretpostavljene vrijednosti koje su prikazane iznad zajedno sa kreiranjem modela. Funkcija je nazvana *process_m* što naznačuje procesiranje modela. Ispod je vidljivo korištenje funkcije za testiranje različitih elemenata arhitekture koje smo pokrili u poglavlju 5.

```
process_m(initXavier_', init='glorot_uniform')
process_m(initRandom_', init='random_uniform')
process_m(augmentNO_', ygen=False)
process_m(whiteningNO_', act='relu', ybn=False)
process_m(dropoutNO_', ydrop=False)
process_m(ninNO_', y1x1=False)
process_m(learn1_', learn_rate_perc=1)
process_m(learn098_', learn_rate_perc=0.98)
process_m(learn092_', learn_rate_perc=0.92)
process_m(learn080_', learn_rate_perc=0.80)
process_m(paddNO_', padd='valid')
```

4.3. Arhitektura

S obzirom da je veličina ulaznih slika jako mala, 28x28 piksela, prirodno treba izbjegavati česte operacije smanjenja uzorka. Poželjno je primijeniti višeslojne konvolucijske blokove prije smanjenja uzorka te držati broj mapa značajki ne prevelikim. Broj mapa značajki odnosno broj *kernela* u konvolucijskom sloju pri prepoznavanju velikih slika sa mnogo kategorija ima smisla jer svaki *kernel* prepoznaje određenu značajku. Međutim kada je broj kategorija malen (10 u slučaju znamenki, 0 do 9), broj različitih značajki je također malen pa povećavanje broja mapa značajki neće uroditi značajnim plodom.

U mrežu su dodani ispad količine d = 0.5 za FC sloj, te 0.1 za konvolucijske slojeve. Također, korištena je metoda 1x1 konvolucije za smanjenje dimenzija mreže što je omogućilo dublju mrežu sa razumnim vremenom računanja. Nakon svakog sloja primijenjena je metoda grupne normalizacije skupa sa ELU aktivacijskom funkcijom za sprječavanje pojačanja signala i preprilagođavanja.

Korištenjem uzastopnih 3x3 konvolucija smanjene su dimenzije pritom imajući sličan efekt kao korištenje 5x5 konvolucije. Kao optimizacijsku funkciju korištena je *Adam* funkcija po preporuci D. Kingma i J. Ba (2014) [45]. Nad početnim skupom podataka izvršena je

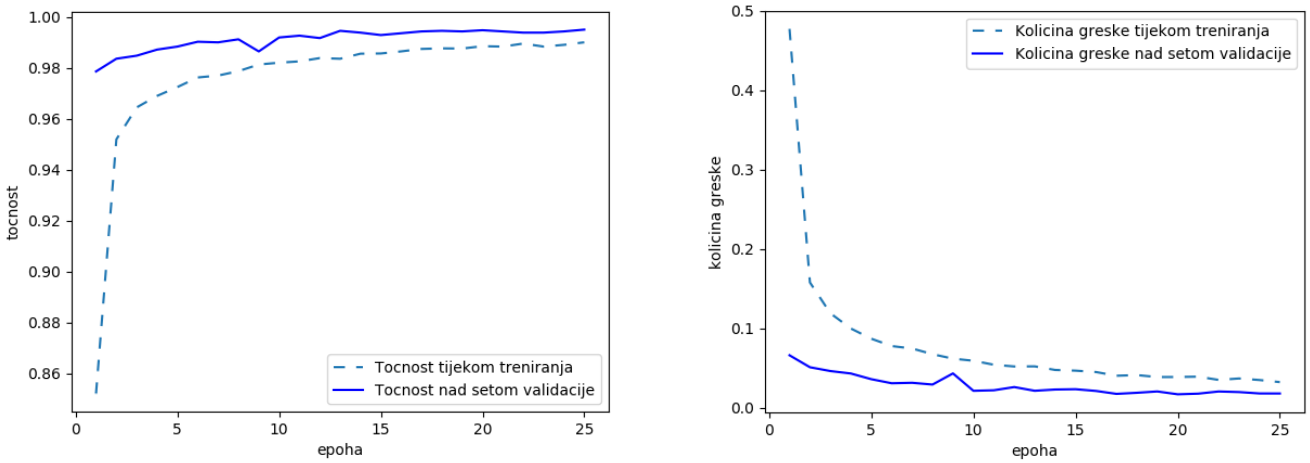
augmentacija preko vertikalnog i horizontalnog pomaka, rotacije te približavanja. Veličina grupe treniranja (eng. *batch*) je 96. Razina učenja je isprva jednaka 0.001 i podešena je da se smanjuje za 5% nakon svakog prijelaza podataka za treniranje. Ukupni broj parametara iznosi 576026, mreža je u nastavku nazvana ***EasyNet***.

Tablica 4 – Arhitektura implementirane mreže

Sloj	**#, Dimenzije / dopuna**	**F(x)**	**Pomak**	**Ispad**	**Grupna norm.**	**Veličina izlaza**	**Broj param.**
$\boldsymbol{CONV1}$	$64, 3x3/1$	ELU	1	$-$	$-$	$28x28x64$	640
$\boldsymbol{CONV2}$	$32, 1x1/0$	ELU	1	20%	$-$	$28x28x32$	2080
$\boldsymbol{POOL1}$	$32, 2x2/2$	max	2	$-$	$-$	$14x14x32$	$-$
$\boldsymbol{CONV3}$	$128, 3x3/1$	ELU	1	$-$	$-$	$14x14x128$	36992
$\boldsymbol{CONV4}$	$48, 1x1/0$	ELU	1	25%	$-$	$14x14x48$	6192
$\boldsymbol{POOL2}$	$48, 2x2/2$	max	2	$-$	$-$	$7x7x48$	$-$
$\boldsymbol{CONV5}$	$256, 3x3/1$	ELU	1	10%	$-$	$7x7x256$	110848
$\boldsymbol{CONV6}$	$64, 1x1/1$	ELU	1	25%	$-$	$7x7x64$	16448
$\boldsymbol{FC1}$	128, -/-	ELU	$-$	50%	DA	128	401536
$\boldsymbol{FC2}$	$10, -/-$	$softmax$	$-$	$-$	$-$	10	1290

Na grafovima ispod se može vidjeti preciznost mreže skupa sa relevantnim rezultatima.

Preciznost mreže iznosi 99.501%.



Slika 23 – Grafovi točnosti (lijevo) i količine greške (desno) za implementiranu arhitekturu

Usporedba *EasyNet* mreže sa ostalim poznatim MNIST mrežama:

Tablica 5 – Usporedba implementirane mreže sa ostalim poznatim MNIST mrežama

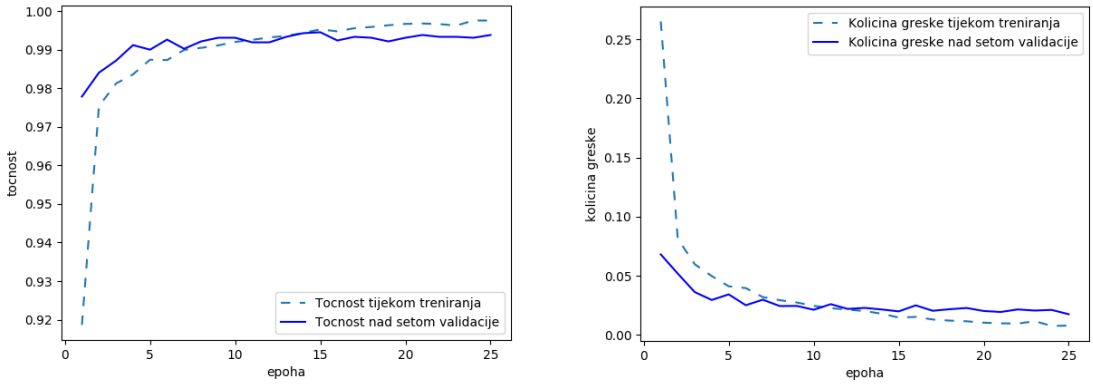
Mreža	**Točnost**	**Rad**
LeNet-5	99.20%	Lecun (1998)
Boosted LeNet-4	99.30%	Lecun (1998)
EasyNet	**99.50%**	**ovaj rad (2018)**
7 kombiniranih CNN	99.73% +-0.02%	Ciresan i sur (2011)
35 kombiniranih CNN	99.77%	Ciresan i sur (2012)

5. Eksperimenti

U ovom dijelu analiziramo utjecaj određenih dijelova strukture mreže i uspoređujemo performanse pri njihovoj promjeni ili uklanjanju.

5.1. Augmentacija skupa podataka za treniranje

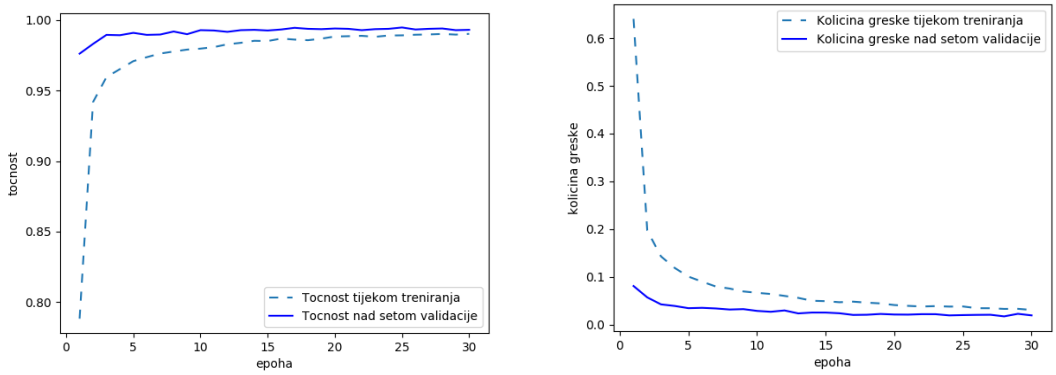
Izvorni skup slika iznosi 70000 slika. Kada se to podijeli na skup za treniranje i skup za validaciju (omjer 10:4 respektivno), ostane nam 42000 slika za treniranje. Zbog toga su primijenjene različite augmentacije nad tim slikama kako bi povećali skup za treniranje. Bez tog koraka skup za treniranje je premal za klasificiranje krajnjih slučajeva poput znamenki koje su rotirane na različit način, manje veličine i slično. Iz grafa ispod se da iščitati da je očito došlo do preprilagođavanja jer model ima veću preciznost nad skupom za treniranje.



Slika 24 – Grafovi točnosti (lijevo) i količine greške (desno) bez augmentacije podataka

5.2. ELU i grupna normalizacija

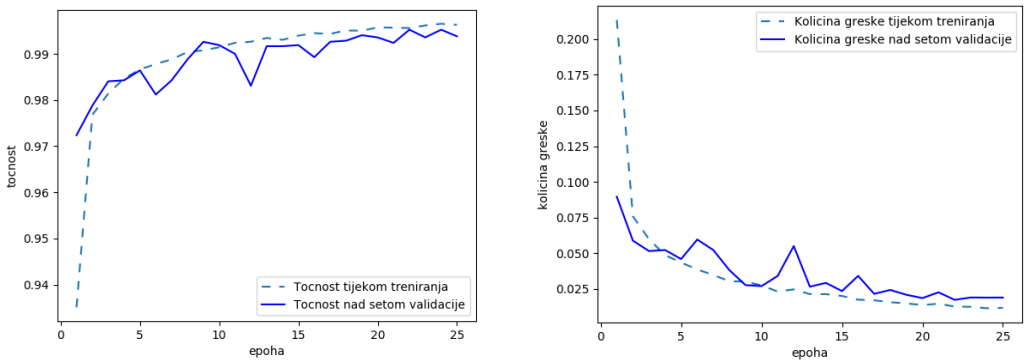
ELU i grupna normalizacija imaju dvije uloge – smanjenje preprilagođenosti i normalizacija signala kroz konvolucije. Ukoliko se grupna normalizacija ukloni i ELU se zamjeni sa ReLU, može doći do preprilagođenosti. Na grafu ispod nije došlo do velike preprilagođenosti, pretpostavljam da je to većinski zbog ispada i augmentacije podataka koji to sprječavaju. Međutim točnost nad validacijskim skupom ostaje ista od epohe 15 dok točnost tijekom treniranja raste i dalje što može naznačiti preprilagođavanje.



Slika 25 – Grafovi točnosti (lijevo) i količine greške (desno) bez izbjeljivanja podataka

5.3. Ispad

Korištenjem ispada neuroni se tjeraju da imaju *širi spektar razumijevanja*. Ako su im susjedni neuroni izbačeni iz treniranja, oni preuzimaju njihovu ulogu klasifikacije. Dvije su posljedice ovog pristupa – povećane performanse i sprječavanje preprilagođenosti. Bez primjene ispada preprilagođenost se događa gotovo odmah (vidljivo iz slike ispod).

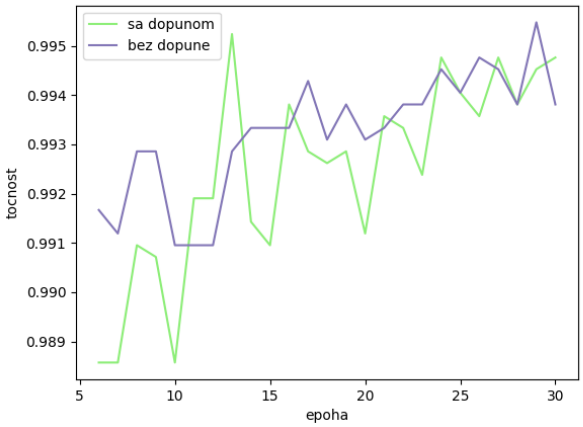


Slika 26 – Grafovi točnosti (lijevo) i količine greške (desno) bez ispada

5.4. Dopuna

Kroz rad smo koristili dopunu kako bi održali dimenzije mapa značajki konstantnim na dijelovima prije smanjenja uzorka. Bez dopune, pretpostavili smo da bi dimenzije bile premale da se iz njih može pravilno klasificirati znamenka. Testiranje je opovrgnulo prethodnu

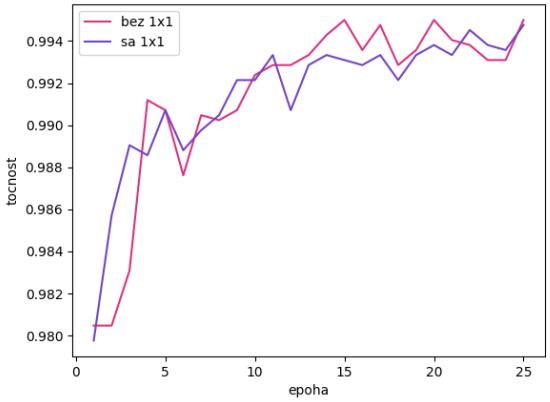
pretpostavku. Najvjerojatniji razlog je broj klasa: samo je 10 znamenki te iako je u zadnjem CONV6 sloju veličina pojedine mape značajki svega 3x3, preciznost je ostala slična.



Slika 27 – Graf točnosti sa i bez dopune (zadnjih 25 epoha)

5.5. Konvolucije 1x1

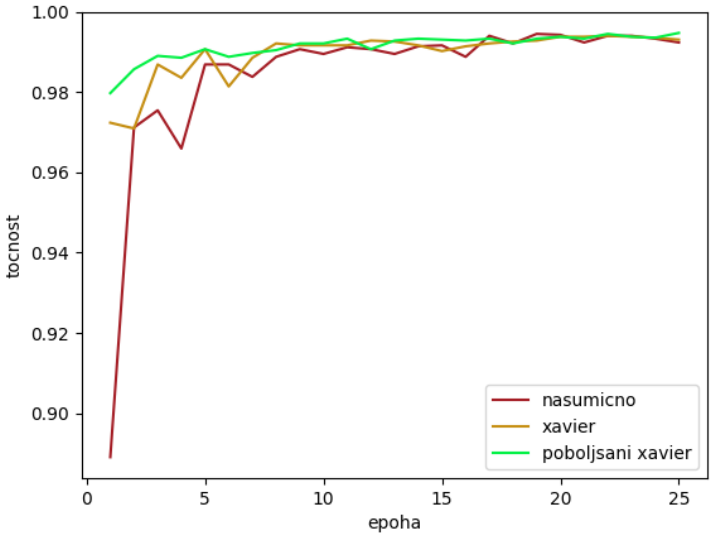
Konvolucije 1x1 su korisne za smanjenje dimenzija čime možemo produbiti model. U ovoj arhitekturi su korištene upravo za tu svrhu i ukoliko ih uklonimo broj parametara se poveća mnogostruko, sa 576026 na 1977226. Razlike u performansama su minorne (vidljivo na slici ispod) i idu u korist korištenja 1x1 konvolucija.



Slika 28 – Graf točnost sa i bez 1x1 konvolucija

5.6. Inicijalizacija kernela

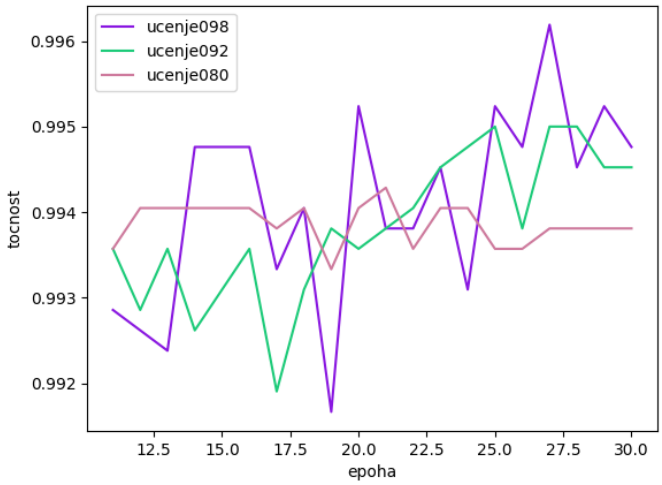
Korištena je unaprijeđena verzija *Xavier*-ove inivijalizacije po preporuci K. He i sur. (2015a) [33]. Testirali smo nasumično, *Xavier* i unaprijeđenu *Xavier* verziju. Vidljivo je iz prikaza da nasumično inicijalizirani kerneli imaju jako sporo konvergiranje pri početku treniranja za razliku od *xaviera*, no pri kraju ne ovisi koja je inicijalizacija korištena. Također, poboljšani *xavier* ima brže konvergiranje od običnog *xavier*.



Slika 29 – Točnost pri različitim inicijalizacijama kernela

5.7. Razina učenja

Razina učenja može utjecati na performanse mreže i njenu mogućnost da konvergira. Prevelika razina učenja vodi do rijetkog konvergiranja jer se funkcija previše mijenja, premala razina učenja vodi do sporijeg treniranja i može dovesti do konvergiranja ka lokalnom minimumu a cilj je konvergirati ka globalnom minimumu. Testirali smo 3 razine učenja, 0.98, 0.92 te 0.80, rezultati vidljivi na grafu ispod. Za razinu učenja 0.80 vidimo da ostaje u lokalnom minimumu kroz većinu treniranja, previsoka razina učenja 0.98 oscilira i ne može konvergirati.



Slika 30 – Graf točnosti za različite razine učenja (zadnjih 20 epoha)

5.8. Alternativna arhitektura

Jeftina mrežna arhitektura koja sadrži samo 143 890 parametara za treniranje i postiže preciznost do 99.4%. Glavna stvar u ovoj jeftinoj mreži je što nema sloja za smanjenje uzorka već koristi konvoluciju sa *kernelom* 3x3 ali pomakom S = 2 čime efektivno smanjuje dimenzije. Pritom, taj *kernel* uči kako smanjiti prostorne dimenzije a da pritom očuva najbitnije informacije. Sličan pristup su koristili A. Krizhevsky i sur (2012) [10]. Prije te konvolucije primijenjena je 1x1 konvolucija koja smanjuje dimenzije kroz smanjenje broja mapa značajki. Time se dobiva dvostruko smanjenje dimenzija, jedno smanjuje broj mapa značajki, drugo smanjuje prostorne dimenzije pojedine mape značajki. Programski kod ove arhitekture se nalazi ispod:

```
#blok1: dvije 3x3 konvolucije zaredom
model.add(ZeroPadding2D(padding=(1, 1)))
model.add(Conv2D(48, kernel_size=(3, 3),activation='elu'))
model.add(ZeroPadding2D(padding=(1, 1)))
model.add(Conv2D(48, kernel_size=(3, 3),activation='elu'))

#blok2: dvojako smanjenje dimenzija
model.add(Conv2D(24, kernel_size=(1, 1),activation='elu'))
model.add(Dropout(0.10))
model.add(ZeroPadding2D(padding=(1, 1)))
model.add(Conv2D(24, kernel_size=(3, 3), strides=(2,2), activation='elu'))
model.add(Dropout(0.10))
```

```
#blok3: dvije 3x3 konvolucije zaredom
model.add(Conv2D(72, kernel_size=(3, 3),activation='elu'))
model.add(ZeroPadding2D(padding=(1, 1)))
model.add(Conv2D(72, kernel_size=(3, 3),activation='elu'))

#blok4: dvojako smanjenje dimenzija
model.add(Conv2D(18, kernel_size=(1, 1),activation='elu'))
model.add(Dropout(0.10))
model.add(ZeroPadding2D(padding=(1, 1)))
model.add(Conv2D(18, kernel_size=(5, 5), strides = (2,2), activation='elu'))
model.add(Dropout(0.10))

#blok5: potpuno spojeni slojevi
model.add(Dropout(0.10))
model.add(Flatten())
model.add(Dense(96, activation='elu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))
```

5.9. Moguća unaprjeđenja

Za rad u budućnosti, kako bi mreža bila što preciznija, teoretski mogu se primijeniti sljedeća unaprjeđenja:

- Treniranje više nezavisnih mreža i kombiniranje njihovog predviđanja
- Treniranje više zavisnih mreža kao Lecun i sur. (1998). Trenirati prvu nad normalnim testnim podacima, u drugu mrežu ubaciti 50% točnih i 50% netočnih rezultata iz prve (miješani uzorci), a u treću ubaciti samo one koje su i prva i druga netočno klasificirale. Lecun ovime smanjuje LeNet-4 grešku sa 1.1% na 0.7%
- Primijeniti globalni AVG *pooling* koji je obrađen u poglavlju 4.4., time smanjiti kompleksnost i potencijalno poboljšati performanse
- Dodati grupnu normalizaciju na svaki konvolucijski sloj, mnogostruko sporije treniranje međutim moguće povećanje performansi

6. Zaključak

Danas su CNN *state of the art* algoritmi u računalnom vidu. Primijenjene su od prepoznavanja lica do autonomnih auta sa visokim uspjehom. Ništa od toga ne bi bilo moguće da treniranje pomoću grafičkih procesorskih jedinki nije uzelo zamaha. Cijela se industrija grafičkog hardvera prilagođava za treniranje neuronskih mreža.

U ovom radu su obrađene neuronske mreže i njihova primjena u problemu rukom pisanih znakova. Svaka stavka arhitekture CNN-a je obrađena te je prikazan njezin razvoj kroz vrijeme i najnovija poboljšanja. To uključuje sloj konvolucije, *kernele* i njihovu inicijalizaciju, pomak, dopuna, ispad, aktivacijske funkcije, sloj smanjenja uzorka, potpuno spojeni sloj, kategorizaciju i mape značajki. Uz pojedinačne stavke, kroz analizu slučaja uočavamo više općih pravila, povećavati broj mapa značajki kroz dubinu neuronske mreže, primijeniti neku vrstu izbjeljivanja podataka (ili ELU ili grupna normalizacija) i smanjivati razinu učenja kroz treniranje.

Provedeni eksperimenti su pokazali da pojedina tehnika nema uvijek jednak učinak, tehnike različito doprinose performansama ovisno o ostalim tehnikama koje se koriste u mreži i bitno je naći dobru kombinaciju tih parametara. Za moju odabranu arhitekturu to je značilo korištenje 1x1 konvolucija nakon 3x3 konvolucijskog sloja te ispad nakon 1x1 konvolucije. U nekoj drugoj arhitekturi ili problematici, drukčiji redoslijed konvolucija te primjena ispada na drugim mjestima bi možda dali bolji učinak.

Područje neuronskih mreža, pogotovo konvolucijskih se još uvijek intenzivno razvija što se vidi po brojnim doprinosima i novim metodama koje su predlagane kroz prethodnih 5 godina. Dogodilo se par mini-revolucija sa AlexNet, NiN te ResNet i za očekivati je slične događaje u bliskoj budućnosti.

7. Popis literature

[1] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, *Gradient-based learning applied to document recognition*, u: Proceedings of the IEEE, Vol. 86, 1998, str. 2278–2324.

[2] Ds F. N. Iandola i sur. "*SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 1MB model size*." arXiv preprint arXiv: 1602.07360, 2016

[3] S. Wadhwa, FloydHub Blog, *Building Your First ConvNet*, 2017, Dostupno: https://blog. floydhub.com/building-your-first-convnet/ [pristupljeno 4.9.2018].

[4] F. Kratzert, Github, *Understanding the backward pass through Batch Normalization Layer*, 2016, Dostupno: kratzert.github.io/2016/02/12/ understanding-the-gradient-flow-through-the-batch-normalization-layer.html [pristupljeno 4.9.2018].

[5] L.C. Chen i sur., „*Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation“*arXiv preprint arXiv: 1802.02611, 2018

[6] G. Larsson, M. Maire, i G. Shakhnarovich. *Fractalnet: Ultra-deep neural networks without residuals*. ICLR, 2017.

[7] S. Li, *Rotation Invariance Neural Network*, 2017

[8] Y. LeCun i Y. Bengio. *Convolutional networks for images, speech, and time series*. The handbook of brain theory and neural networks, 3361, 1995.

[9] Y. Xu i sur., *Scale-Invariant Convolutional Neural Network*, 2014

[10] A. Krizhevsky, I. Sutskever, G. E. Hinton, *Imagenet classification with deep convolutional neural networks*, u: F. Pereira, C. Burges, L. Bottou, K. Weinberger (Eds.), Advances in Neural Information Processing Systems 25, Curran Associates, Inc., 2012, str. 1097–1105.

[11] D. Mishkin, N. Sergievskiy, J. Matas, *Systematic evaluation of CNN advances on the ImageNet*, Center for Machine Perception, Faculty of Electrical Engineering, Czech Technical University in Prague, 2016

[12] P. Sermanet i sur.*, OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks*, Courant Institute of Mathematical Sciences, New York University, 2014

[13] K. Simonyan i A. Zisserman, *Very Deep Convolutional Networks For Large-Scale Image Recognition*, Department of Engineering Science, University of Oxford, 2014

[14] J. Schmidhuber, *Deep learning in neural networks: An overview*, University of Lugano & SUPSI, 2014

[15] S. Haykin, *Neural Networks: A Comprehensive Foundation* (2 ed.). Prentice Hall, University Hamilton Ontario, 1998

[16] M. Lin, Q. Chen, i S. Yan. *Network in network*. CoRR, abs/1312.4400, 2013.

[17] C. Szegedy i sur., *Going deeper with convolutions*. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, str. 1–9, 2015.

[18] J. Wu, *Introduction to Convolutional Neural Networks*. Nanjing University, China, 2017

[19] S. Ioffe and C. Szegedy. *Batch normalization: Accelerating deep network training by reducing internal covariate shift*. In Proceedings of The 32nd International Conference on Machine Learning, str. 448–456, 2015.

[20] C. Szegedy, V. Vanhoucke et al., "*Rethinking the inception architecture for computer vision*," arXiv preprint arXiv:1512.00567, Dec. 2015. [Online]. Available: https://arxiv.org/abs/1512.00567v3

[21] K. He i sur. *Deep residual learning for image recognition*. arXiv preprint arXiv:1512.03385, 2015

[22] J. Redmon i sur., *You only look once: Unified, real-time object detection*. arXiv preprint arXiv:1506.02640, 2015.

[23] J. Redmon and A. Farhadi. Yolo9000: *Better, faster, stronger*. In Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on, str. 6517–6525. IEEE, 2017

[24] R. Plamondon, S. N. Srihari. "*Online and offline handwriting recognition: a comprehensive survey*." Pattern Analysis and Machine Intelligence, IEEE Transactions on 22.1 (2000): str. 63-84., 2000

[25] A. Graves i J. Schmidhuber, "*Offline Handwriting Recognition with Multidimensional Recurrent Neural Networks*," in Advances in Neural Information Processing Systems 21, 2009, str. 545–552., 2009

[26] R. Raina i sur. "*Large-scale deep unsupervised learning using graphics processors*," u International Conference on Machine Learning (New York, NY: ACM Press), str. 1–8; 2009

[27] D. C. Ciresan i sur. *Flexible, high performance convolutional neural networks for image classification*. In International Joint Conference on Artificial Intelligence, str. 1237–1242, 2011

[28] D. C. Ciresan, U. Meier, J. Schmidhuber, *Multicolumn deep neural networks for image classification*. In Proceedings of CVPR, str. 3642–3649, 2012.

[29] A. Paszke i sur. "*Enet: A deep neural network architecture for real-time semantic segmentation*," arXiv preprint arXiv:1606.02147, 2016.

[30] S.W. Smith, "*The Scientist and Engineer's Guide to Digital Signal Processing*", California Technical Publishing, str. 243-260, 1997.

[31] X. Glorot i Y. Bengio, *Understanding the difficulty of training deep feedforward neural networks*. In Aistats, volume 9, str. 249–256, 2010.

[32] A. Karpathy, " Convolutional Neural Networks (CNNs / ConvNets)," veljača 2011. [Online]. Dostupno http://cs231n.github.io/convolutional-networks/

[33] K. H i sur.. "*Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*". ICCV 2015

[34] D.A. Clevert, T. Unterthiner, i S. Hochreiter. *Fast and accurate deep network learning by ´ exponential linear units (elus),* arXiv preprint arXiv:1511.07289, 2015.

[35] M. D. Zeiler, R. Fergus, Stochastic Pooling for Regularization of Deep Convolutional Neural Networks, arXiv e-printsarXiv:1301.3557.

[36] C. Gulcehre i sur., *Learned-Norm Pooling for Deep Feedforward and Recurrent Neural Networks*, arXiv e-prints arXiv: 1311.1780., 2013

[37] C.-Y. Lee, P. W. Gallagher, Z. Tu, *Generalizing Pooling Functions in Convolutional Neural Networks: Mixed, Gated, and Tree*, arXiv e-prints arXiv: 1509.08985., 2015

[38] J. T. Springenberg i sur., *Striving for Simplicity: The All Convolutional Net*, in: Proceedings of ICLR Workshop,. arXiv:1412.6806., 2014

[39] A. Maas, A. Hannun i A. Ng, *Rectifier nonlinearities improve neural network acoustic models*. Proceedings ICML 30, str. 1–6, 2013

[40] M. Riesenhuber i T. Poggio, *Hierarchical models of object recognition in cortex*, Center for Biological and Computational Learning and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, 1999

[41] F. Chollet. *Xception: Deep learning with depthwise separable convolutions*. arXiv preprint arXiv:1610.02357v2, 2016.

[42] S. Hochreiter, Y. Bengio, P. Frasconi i J. Schmidhuber, *Gradient flow in recurrent nets: the difficulty of learning long-term dependencies.* A field guide to dynamical recurrent neural networks. IEEE Press, 2001.

[43] A. Ng, Coursera, „*Networks in Networks and 1x1 Convolutions",* 2017 Dostupno https://www.coursera.org/lecture/convolutional-neural-networks/networks-in-networks-and-1x1-convolutions-ZTb8x [pristupljeno 9.9.2018]

[44] A. Dertat, Towards Data Science, *Applied Deep Learning - Part 4: Convolutional Neural Networks*, 2017, Dostupno https://towardsdatascience .com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2 [pristupljeno 9.9.2018]

[45] D. Kingma i J. Ba, *Adam: A method for stochastic optimization*, arXiv preprint arXiv:1412.6980, 2014

[46] M. Thoma, *Analysis and Optimization of Convolutional Neural Network Architectures,* arXiv preprint arXiv:1707, 2017

8. Popis slika

9. Popis tablica