Primjena konvolucijskih neuronskih mreža za detekciju upale pluća na slikama magnetske rezonance

Krnetić, Luka

SVEUČILIŠTE U DUBROVNIKU
ODJEL ZA ELEKTROTEHNIKU I RAČUNARSTVO

LUKA KRNETIĆ

PRIMJENA KONVOLUCIJSKIH NEURONSKIH MREŽA ZA DETEKCIJU UPALE PLUĆA NA SLIKAMA MAGNETSKE REZONANCE

DIPLOMSKI RAD

Dubrovnik, rujan, 2021.

SVEUČILIŠTE U DUBROVNIKU
ODJEL ZA ELEKTROTEHNIKU I RAČUNARSTVO

PRIMJENA KONVOLUCIJSKIH NEURONSKIH MREŽA ZA DETEKCIJU UPALE PLUĆA NA SLIKAMA MAGNETSKE REZONANCE

DIPLOMSKI RAD

Studij: Primijenjeno/poslovno računarstvo
Kolegij: Uvod u znanost o podacima
Mentor: izv.prof.dr.sc. Mario Miličević
Student: Luka Krnetić

Dubrovnik, rujan, 2021.

SAŽETAK

Upala pluća je zarazna bolest donjeg dijela dišnog sustava koja nastaje kada bakterije, virusi ili gljivice napadaju pluća te dovode do upalnog odgovora kojim se ljudsko tijelo brani od infekcija. U najgorem slučaju rezultat upale pluća može biti smrt osobe. Nije rijetkost da netko oboli od upale pluća, osobito u zemljama u razvoju gdje se velik broj ljudi suočava s energetskim siromaštvom i oslanja na zagađujuće oblike energije. Ovaj rad se bavi primjenom konvolucijskih neuronskih mreža za detekciju upale pluća iz slika magnetske rezonance. U teoretskom dijelu rada objašnjene su umjetne neuronske mreže, tehnike koje se koriste kako bi se poboljšao rad neuronskih mreža, konvolucijske neuronske mreže, GradCAM tehnika te dobiveni rezultati. U praktičnom dijelu rada, za klasifikaciju slika magnetske rezonance, korištene su različite arhitekture konvolucijskih neuronskih mreža, kao i tehnike za unaprjeđenje rada neuronskih mreža. Korištena je GradCAM tehnika kako bi se vizualiziralo na što konvolucijska neuronska mreža obraća pažnja kada klasificira određenu sliku. Također je napravljena web aplikacija putem koje se na temelju učitane slike može doći do dijagnoze upale pluća. Programski kod je napisan u programskom jeziku Python, a također je korišten programski okvir Tensorflow.

Ključne riječi: konvolucijske neuronske mreže, detekcija upale pluća, slike magnetske rezonance

ABSTRACT

Pneumonia is a contagious disease of the lower respiratory tract that occurs when bacteria, viruses or fungi attack the lungs and lead to an inflammatory response that protects the human body from infections. In the worst case, the result of pneumonia can be the death of a person. It is not uncommon for someone to get pneumonia, especially in developing countries where large numbers of people face energy poverty and rely on polluting forms of energy. This paper deals with the application of convolutional neural networks for the detection of pneumonia from magnetic resonance imaging images. The theoretical part of the paper explains artificial neural networks, techniques used to improve the operation of neural networks, convolutional neural networks, GradCAM techniques and the obtained results. In the practical part of the paper, different architectures of convolutional neural networks as well as techniques for improving the operation of neural networks were used to classify magnetic resonance images. The GradCAM technique was used to visualize what the convolutional neural network pays attention to when classifying a particular image. A web application was also created through which the diagnosis of pneumonia can be made on the basis of the uploaded image. The programming code is written in the Python programming language, and the Tensorflow programming framework is also used.

Key words: convolutional neural networks, detection of pneumonia, magnetic resonance imaging

Sadržaj:

1. UVOD

Budući da se ovaj rad temelji na dubokom učenja, onda će područje dubokog učenja posebno biti objašnjeno u daljnjem dijelu rada. U uvodu će biti objašnjeno strojno učenje, jer se duboko učenje temelji na strojnom učenju.

Strojno učenje prema Tom Michael Mitchellu (američkom računalnom znanstveniku) je „Mogućnost računalnog programa da poboljša zadatak T u odnosu na metriku performanse P temeljeno na iskustvu E“. Rad strojnog učenja se temelji na generiranju hipoteze, automatizaciji otkrivanja i korištenja pravilnosti u velikim skupovima podataka te istraživanja što se može naučiti, pod kojim uvjetima i koja je kvaliteta naučenih modela. Model u strojnom učenju predstavlja matematički izraz koji rješava problem iz stvarnog svijeta. Da bi model naučio određene pravilnosti potrebno je imati podatke, hipotezu i određene evaluacijske metrike.

Strojno i duboko učenje se sve više koristi u raznim industrijama pa tako i u medicini. Konvolucijske neuronske mreže su vrsta neuronskih mreža koja se obično koristi kada je ulazni podatak slika ili video. Budući da je u ovom radu ulazni podatak slika magnetske rezonance, onda su korištene konvolucijske neuronske mreže.

1.1. Definicija rada

Ovaj rad se bavi primjenom konvolucijskih neuronskih mreža za detekciju upale pluća. Korištena je GradCAM tehnika da bi se uvidjelo na koje značajke konvolucijska neuronska mreža obraća pažnju kada je u pitanju klasifikacija slike. Napravljena je i web aplikacija unutar koje se može učitati slika te na temelju učitane slike dobiti dijagnoza upale pluća.

1.2. Svrha i ciljevi rada

Kada je u pitanju klasifikacija upale pluća prema slikama magnetske rezonance, cilj rada je stvoriti modele visoke točnosti, dobiti interpretaciju konvolucijske neuronske mreže koristeći

GradCAM tehniku te stvoriti web aplikaciju putem koje je moguće doći do dijagnoze bolesti upale pluća na temelju učitane slike magnetske rezonance.

1.3. Metodologija rada

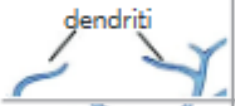
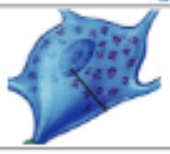
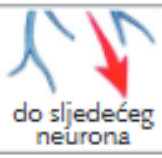
U radu su primijenjene standardne metode ispitivanja klasifikacije pojedinačnog modela prateći neke od metrika kao što su točnost, odziv, preciznost, stvarno pozitivni, stvarno negativni, lažno pozitivni i lažno negativni. Na osnovu teoretskog znanja o neuronskim mrežama, u praktičnom djelu rada, koristeći programsku biblioteku Tensorflow, konstruirani su modeli te je korištena GradCAM tehnika za vizualizaciju na što neuronska mreža obraća pažnju klada klasificira određenu sliku. Za izradu web aplikacije korištena je programska biblioteka TensorflowJS.

1.4. Struktura rada

Rad se sastoji od 7 poglavlja koji međusobno čine zaokruženu cjelinu. U prvom poglavlju su navedeni struktura, svrha i cilj rada. Zatim su navedene metrike koje se prate pri evaluaciji modela. U drugom poglavlju objašnjene su umjetne neuronske mreže, dok su u trećem poglavlju objašnjene konvolucijske neuronske mreže. U četvrtom poglavlju su objašnjene arhitekture koje su se koristile za stvaranje modela. GradCAM tehnika je objašnjena u petom poglavlju, dok je programski kod objašnjen u šestom poglavlju. U zadnjem dijelu rada navedeni su rezultati rada.

2. UMJETNE NEURONSKE MREŽE

Spoznaja o načinu funkcioniranja ljudskog mozga potaknula je istraživanje i razvoj umjetnih neuronskih mreža. Analogija umjetnih neuronskih mreža s biološkim neuronskim mrežama prikazana je na sljedećoj slici.

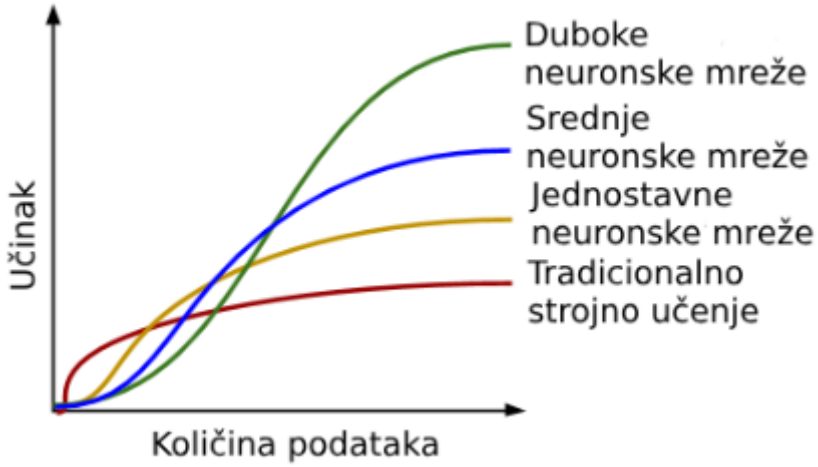
Biološki neuron		Umjetni neuron
	Prima ulazni signal putem dendrida (sinaptičke veze)	Prima ulaze (*i*) koji su određeni težinskim koeficijentima (*w*)
	Obrada signala u somi	Obrada ulaza, unutarnji prag – bias (*b*)
	Pretvara obrađeni ulaz u izlaz putem aksona	Pretvara ulaze u izlaz (prijenosna funkcija)
	Šalje informacije putem sinapsi do svih neurona s kojima je neuron povezan	Šalje informaciju prema izlazu i sljedećim neuronima

Slika 1 - Biološki i umjetni neuron [1]

Prvi rad o umjetnim neuronskim mrežama objavili su McCulloh i Pitts (1943.). Oni su koristili vrlo jednostavan model neurona koji, kao i biološki neuron, obrađuje signale putem sinaptičke i somatske operacije [1]. Taj vrlo jednostavan model neurona nazvan je perceptron.

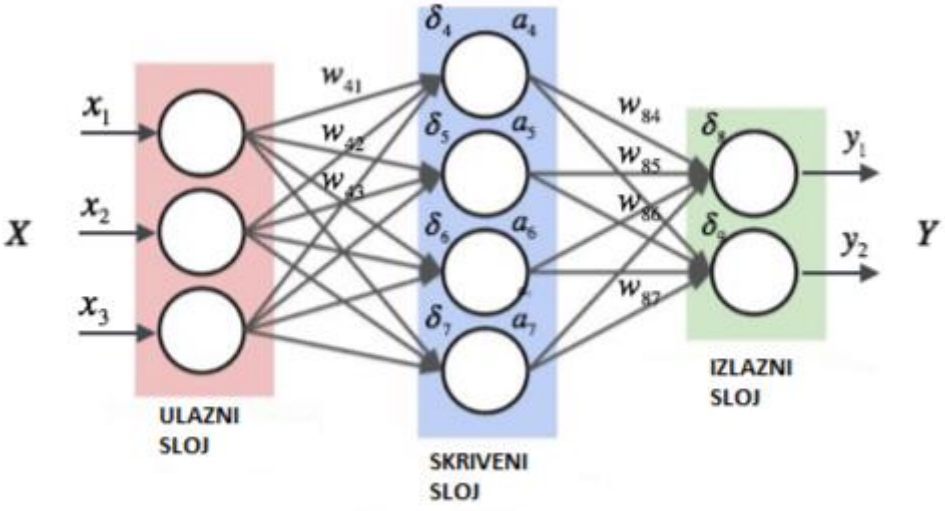
Iako su koncept umjetnih neuronskih mreža Warren McCulloch i Walter Pitts predstavili 1943. godine, postavlja se pitanje zašto su se neuronske mreže popularizirale tek prije nekoliko godina. Glavni razlog tome je što su računalno skupe, računala tada nisu bila dovoljno jakih performansi da podrže rad neuronskih mreža. Drugi razlog je količina podataka.

Na slici ispod prikazan je odnos količine podataka prema učinku neuronskih mreža.



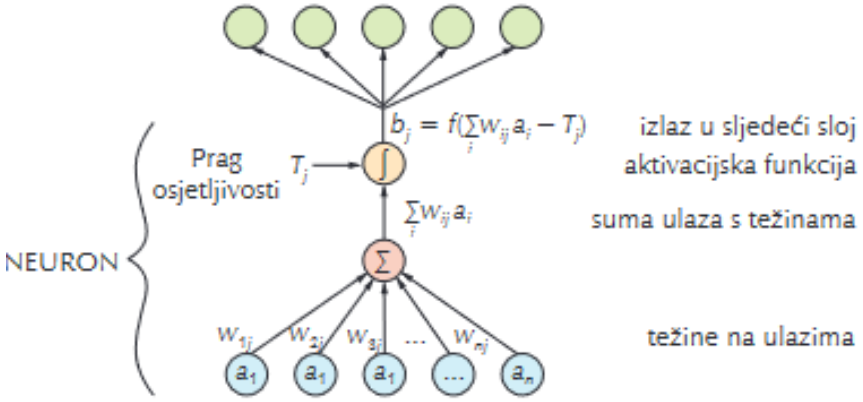
Slika 2 - Odnos količine podataka i učinka [2]

Svaka neuronska mreža sastoji se od ulaznog, skrivenog i izlaznog sloja.



Slika 3 - Primjer strukture neuronske mreže [2]

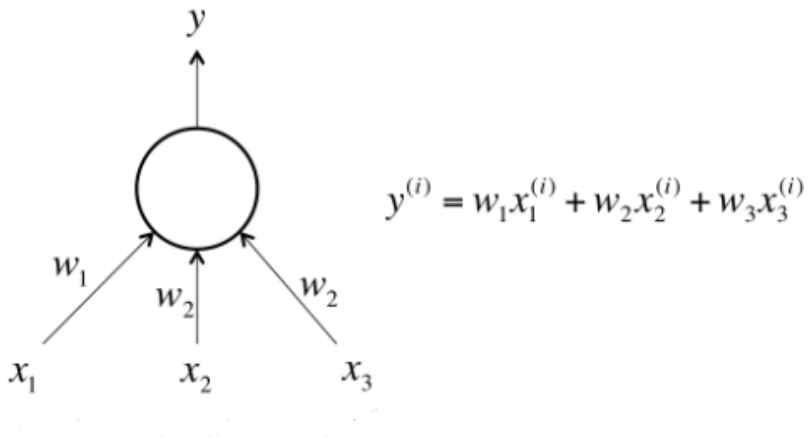
Ulazni sloj poprima vrijednosti ulaznih veličina. Sinaptička operacija predstavlja množenje svakog ulaznog signala s težinskim koeficijentom, w_i. Tako otežani ulazni signali se zbrajaju, a njihov zbroj uspoređuje se s pragom osjetljivosti neurona (engl. Threshold). Težinski faktori analogni su dendritima biološkog neurona. Skriveni sloj zbraja otežane ulaze pomoću neke funkcije sumiranja i tako stvara vlastitu internu aktivaciju. Ako je zbroj otežanih signala veći od praga osjetljivosti neurona, nelinearna aktivacijska funkcija *f* generira izlazni signal neurona iznosa b_j.



Slika 4 - Prikaz perceptrona [1]

2.1. Aktivacijske funkcije

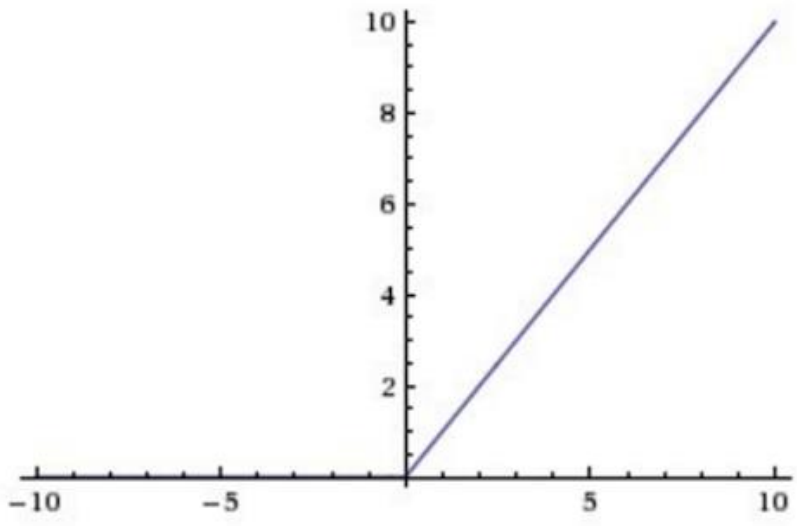
Odabir aktivacijske funkcije ovisi o tipu problema koji se rješava, kao i o arhitekturi same neuronske mreže.



Slika 5 - Primjer linearne aktivacijske funkcije [3]

Linearni neuroni su jednostavni za izračunati, ali imaju svoje nedostatke. Svaka unaprijedno povezana neuronska mreža koja se sastoji od linearnih neurona može biti prikazana bez skrivenih slojeva. To je problematično jer skriveni slojevi služe za prepoznavanje određenih značajki [1]. Da bi se naučile određene kompleksne veze između podataka, potrebno je koristiti nelinearne funkcije. Neke od najpoznatijih nelinearnih aktivacijskih funkcija su ReLU, sigmoida i tangens hiperbolni.

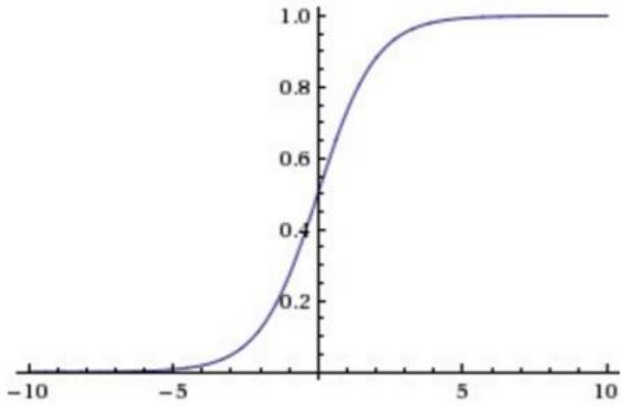
Na sljedećoj slici je prikazana ReLU (engl. Rectified Linear Unit) aktivacijska funkcija.



Slika 6 - ReLU aktivacijska funkcija [5]

Prema slici je vidljivo da je za ulaznu vrijednost x > 0 izlazna vrijednost x, a za ulaznu vrijednost x < 0, izlazna vrijednost 0. ReLU aktivacijska funkcija je dosta popularna u kontekstu konvolucijskih neuronskih mreža. Izbjegava i ispravlja problem nestajanja gradijenata.

Sigmoidna funkcija je oblika $f(x) = \frac{1}{1+\epsilon^{-x}}$

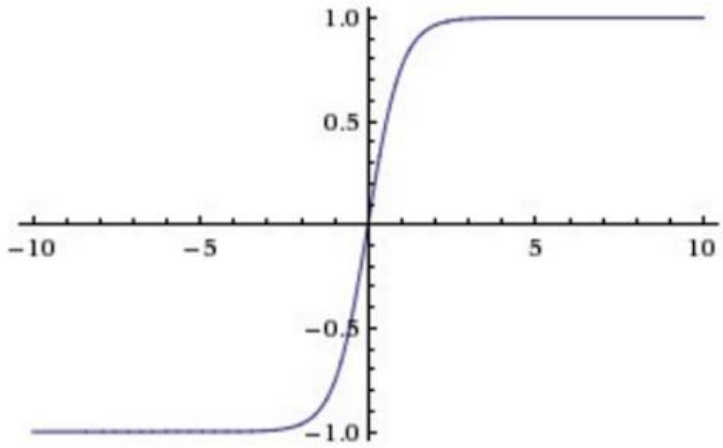


Slika 7 – Sigmoida [5]

Na slici je vidljivo da je izlazna vrijednost funkcije uvijek u intervalu [0, 1]. Zbog svoje glatkoće omogućuje svojstvo nelinearnosti – njezine su kompozicije također nelinearne.

Obično se označava s σ i koristi za binarnu klasifikaciju, jer su izlazne vrijednosti vjerojatnosti [0,1].

Tangens hiperbolni je funkcija koja je prikazana na sljedećoj slici.



Slika 8 - Tangens hiperbolni [5]

Prema slici je vidljivo da su izlazne vrijednosti tangens hiperbolne funkcije na intervalu [-1, 1].

Tangens hiperbolna funkcija je zadana jednadžbom $tanx(x) = \frac{\epsilon^x - \epsilon^{-x}}{\epsilon^x + \epsilon^{-x}}$

Prednost tangens hiperbolne funkcije je da su negativne ulazne vrijednosti mapirane strogo negativno, dok su 0 ulazi mapirani blizu nule u tangens hiperbolnom grafu. Jedan od problema tangens hiperbolne funkcije, kao i sigmoidne funkcije, je u tome što za dosta visoke, ili dosta niske vrijednosti, rezultat derivacije je nizak broj, pa se onda i učenje odvija dosta sporije [2].

2.2 Neuronske mreže, nadgledano učenje i aplikacije

Kada se radi o neuronskim mrežama, česti oblik učenja koji se koristi je **nadgledano učenje**. U nadgledanom učenju uvijek postoji određena izlazna vrijednost Y za određene ulazne vrijednosti X, pa se pokušava na temelju ulazne vrijednosti aproksimirati izlazna vrijednost.

Tablica 1 - Primjeri nadgledanog učenja

Ulaz (x)	Izlaz (y)	Aplikacija
Značajke stana	Cijena	Određivanje cijene nekretnine
Reklama, informacije o korisniku	Hoće li korisnik kliknuti na reklamu? (0/1)	Online oglašavanje
Slika	Objekt	Označavanje slike
Tekst na engleskom jeziku	Tekst na kineskom jeziku	Strojno prevođenje
Zvuk	Transkript tekst	Raspoznavanje govora
Slika, Informacije od radara	Pozicija ostalih automobila	Autonomna vožnja

Najpoznatiji predstavnici neuronskih mreža su umjetne neuronske mreže (engl. Artificial Neural Networks), konvolucijske neuronske mreže (engl. Convolutional Neural Networks) i povratne neuronske mreže (engl. Recurrent Neural Network). Na primjer, za određivanje cijene nekretnine koriste se umjetne neuronske mreže, za određene probleme klasifikacije i detekcije objekta na slici obično se koriste konvolucijske neuronske mreže, a za rad sa sekvencijalnim podacima, kao na primjer tekstom i zvukom, koriste se povratne neuronske mreže.

U nadgledanom učenju podaci mogu biti strukturirani i nestrukturirani. Strukturirani podaci su podaci koji za svaku od određenih značajka imaju određenu vrijednost, kao na primjer strukturirani podaci o stanu bi bili veličina stana 60 m^2, 3 sobe, 1 zahod... Nestrukturirani podaci su podaci koji nemaju strukturu. Algoritmi koji se koriste za rad nad nestrukturiranim podacima sami nalaze strukturu unutar podataka.

2.3 Logistička regresija za neuronske mreže

Logistička regresija je algoritam koji služi za binarnu klasifikaciju. Binarna klasifikacija podrazumijeva klasifikaciju ulazne varijable na dvije izlazne varijable.

Na slici je vidljivo da za svaku od tri boje, crvenu, zelenu i plavu, postoji određena matrica s vrijednostima koji predstavljaju piksele.

Slika 9 - Kako računalo vidi sliku [6]

U ovom slučaju, ulazni sloj neuronske mreže je vektor *X*. Za sliku vektor *X* je dimenzija širina slike * visina slike * dubina slike. Na primjer, za sliku visine i širine 64 piksela koja je prikazana pomoću tri boje, crvene, zelene i plave, računalo će stvoriti vektor *X* dimenzije n_x 12 288. Dakle, u slučaju binarne klasifikacije, za problem određivanja mačke na slici, ulazna vrijednost je vektor *X*, slika, a izlazna vrijednost je *Y*. U ovom slučaju, izlazna vrijednost je vjerojatnost prisutnosti mačke na slici.

Za sljedeće primjere koji će se koristiti za objašnjavanje primjene logističke regresije u slučaju neuronskih mreža, notacija koja će se koristiti je sljedeća:

- (x, y) $x \in \mathbb{R}^{n_x}, y \in \{0, 1\}$ – trening podaci
- m trening slučajeva $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}),... (x^{(m)}, y^{(m)})\}$
- skup svih trening podataka je matrica *X* gdje je svaki stupac transponirani redak jedan trening slučaj. Visina te matrice je n_x, a širina m
- skup svih izlaza je matrica *Y*, $Y \in \mathbb{R}^{1xm}$
- problem logističke regresije je za dani *x* odrediti $\hat{y}$, gdje je $\hat{y} = P(y = 1|x)$
- koriste se parametri $w \in \mathbb{R}^{n_x}$ i $b \in \mathbb{R}$ tako da je $\hat{y} = \sigma(w^t x + b)$, gdje je σ sigmoidna funkcija
- $z^{[l]} = w^T x^{(i)} + b$
- $a^{[l]}$ = aktivacija i-tog sloja, vektor s vrijednostima aktivacija svakog čvora u sloju l
- [i] podrazumijeva veličinu i-tog sloja, na primjer $a^{[L]}$ je aktivacija L-tog sloja
- (i) podrazumijeva vrijednost i-tog primjera
- $a^{[L]}_i$ podrazumijeva i-tu vrijednost aktivacije u L-tom sloju

2.3.1 Logistička regresija u kontekstu neuronskih mreža

Ako je funkcija konveksna, onda postoji jedan globalni minimum [4]. Konveksna funkcija gubitka za logističku regresiju je:

$$L(\hat{y}, y) = -\big(y log\hat{y} + (1 - y) log(1 - \hat{y})\big) \quad (1)$$

Intuicija zašto je gornja jednadžba dobra za funkciju gubitka može se sagledati s dva slučaja. Ako je y = 1, onda je vrijednost jednadžbe $L(\hat{y}, y) = -log\hat{y}$, cilj je imati vrijednost $-log\hat{y}$ što manju što je moguće, a to će biti kada je $\hat{y}$ velik broj. Ako je y = 0, onda je vrijednost jednadžbe $L(\hat{y}, y) = -\log(1 - \hat{y})$, cilj je imati vrijednost što manju, a to će biti kada je $\hat{y}$ što bliže 0.

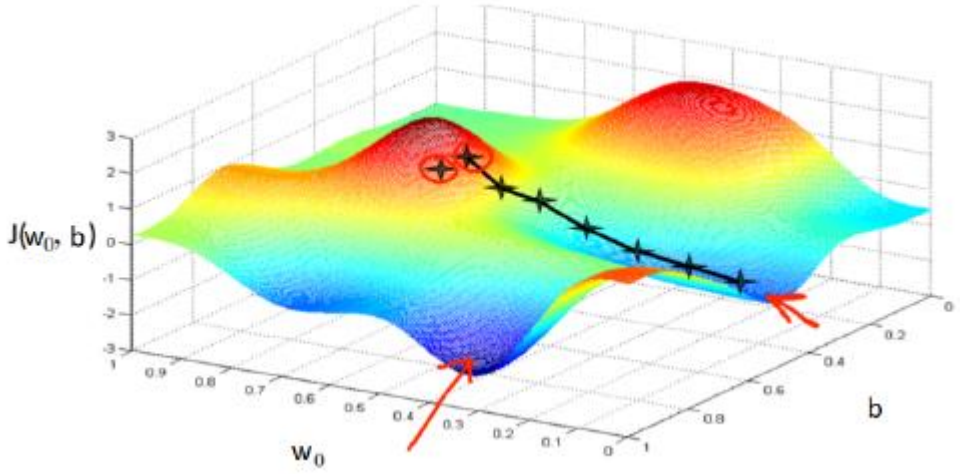
Funkcija koja mjeri gubitak na cijelom trening skupu definirana je sljedećim izrazom:

$$J(w,b) = \frac{1}{m}\sum_1^m L\left(\hat{y}^{(i)}, y^{(i)}\right) \tag{2}$$

to jest: $$J(w,b) = \frac{1}{m}\sum_1^m -(y^{(i)} log\hat{y}^{(i)} + \left(1-y^{(i)}\right)\left(1-\hat{y}^{(i)}\right)) \tag{3}$$

2.3.2 Gradijentni spust

Budući da je cilj imati nisku vrijednost funkcije gubitka, onda ima smisla tražiti parametre *w* i *b* koji minimiziraju funkciju gubitka.



Slika 10 - Gradijentni spust [2]

Gornja slika je ilustracija gradijentnog spusta. Gornja slika se može objasniti sljedećom pričom: zamislimo da se nalazimo na samom vrhu brežuljka te nam je cilj spustiti se u najnižu točku. Zamislimo da se oko nas nalazi magla te da vidimo samo 5 metara oko sebe. Moramo odlučiti u kojem smjeru se moramo kretati kako bi se spustili s brežuljka. To možemo napraviti tako što pogledamo oko sebe te odredimo u kojem smjeru je najstrmiji pad, pa se onda počnemo kretati u tom smjeru. Nakon što dođemo u novu točku, postupak ponavljamo, sve dok se ne možemo više spuštati nizbrdo već se samo penjati uzbrdo.

U gradijentnom spustu početne vrijednosti parametara su 0 ili neke nasumične vrijednosti pa je onda obično vrijednost funkcije gubitka jako velika (vrh brežuljka). Bitno je naglasiti da početne vrijednosti za logističku regresiju mogu biti 0. Ako su početne vrijednosti

inicijalizirane na 0, a algoritam koji se koristi za rad na podacima su neuronske mreže, onda skriveni slojevi imaju iste vrijednosti i uče iste značajke [8]. Strmi pad je metafora za gradijent spusta, dakle gradijent spusta govori u kojem smjeru se moramo kretati kako bi vrijednost funkcije gubitka bila što manja. Pomak gradijentnog spusta zadan je parametrom α (spust od 5 metara).

Gradijentni spust se matematički može objasniti sljedećim postupcima:

W ←Zadati slučajnim odabirom na određeni broj

Ponavljati sve dok konvergira {

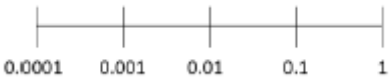
$$W_j \leftarrow W_j - \alpha \frac{\partial J(w)}{\partial W_j} \tag{4}$$

}

Gradijentni spust za *m* trening podataka definiran je na sljedeći način:

$$\frac{\partial}{\partial w} J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \frac{\partial}{\partial w} L(a^{(i)}, y^{(i)}) \tag{5}$$

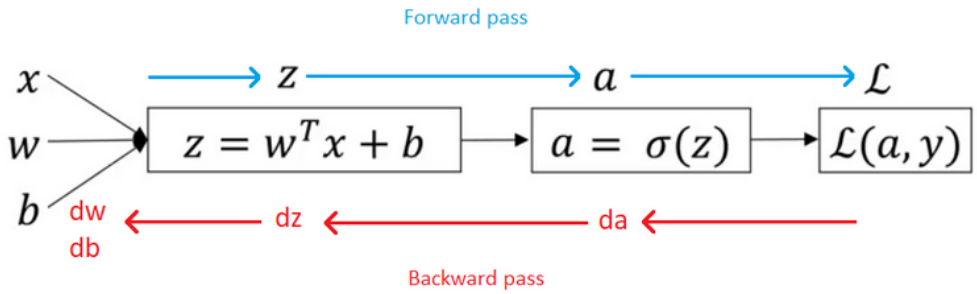
Koeficijent učenja (engl. Learning rate) α je koeficijent brzine konvergiranja. Ako je α prevelik, onda dolazi do divergencije, a ako je α premalen, onda dolazi do spore konvergencije. Kada je u pitanju određivanje vrijednosti koeficijenta učenja, onda se može koristiti skalom koja je prikazana na sljedećoj slici.



Slika 11 - Skala za koeficijent učenja [2]

2.4 Unaprijedna i unazadna propagacija

Sljedeća slika prikazuje **unaprijednu propagaciju** (engl. Forward propagation), kao i **unazadnu propagaciju** (engl. Backward propagation). U sljedećoj slici slovo *d* podrazumijeva parcijalnu derivaciju funkcije gubitka za određenu vrijednost. Na primjer, *dw* podrazumijeva parcijalnu derivaciju funkcije gubitka s obzirom na koeficijent *w*.



Slika 12 - Primjer unaprijedne i unazadne propagacije [9]

U ovom jednostavnom primjeru vektor *x* zajedno s parametrima *w* i *b* čini ulazni sloj. Skriveni sloj čini izlaz sigmoidne funkcije za matrični produkt koeficijenta *w* i *x*, zajedno s zbrajanjem vektora otklona *b*, $(z = w^T x + b)$. Izlazni sloj čini računanje funkcije gubitka *L*. Unaprijednu propagaciju čini računanje $z = w^T x + b$, apliciranje nelinearne aktivacijske funkcije, u ovom slučaju sigmoidne aktivacijske funkcije, te na kraju računanje gubitka. Unazadnu propagaciju čini računanje parcijalne derivacije funkcije gubitka za *a*, *z*, *w* i *b*.

Propagaciju pogreške unatrag, od izlaznog sloja neuronske mreže pa sve do ulaznog sloja moguće je izračunati primjenom lančanog pravila deriviranja. Neka su $g : \mathbb{R}^m \to \mathbb{R}^n$ *i* $f: \mathbb{R}^n \to \mathbb{R}$ realne funkcije, ako $y = g(x)$ i $z = f(x)$, tada vrijedi lančano pravilo deriviranja:

$$\frac{\partial y}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j} \frac{\partial y_j}{\partial x_i} \tag{6}$$

Dakle, ako bi se radilo o dubljoj neuronskoj mreži, u onoj mreži zadanoj s *L* razina dubine, gdje postoji *L* slojeva, gdje je ulazni sloj vektor x, težine l-tog sloja $w^{[l]}$ te vektor otklona $b^{[l]}$, onda bi se za svaki $w^{[1]}$, $w^{[2]}$, ... $w^{[L]}$, $b^{[1]}$, $b^{[2]}$, ... $b^{[L]}$, $z^{[1]}$, $z^{[2]}$, ... $z^{[L]}$, $a^{[1]}$, $a^{[2]}$, ... $a^{[L]}$ računala parcijalna derivacija funkcije gubitka te bi se vrijednost za svaki od elemenata $w^{[1]}$, $w^{[2]}$, ... $w^{[L]}$

i $b^{[1]}$, $b^{[2]}$, ... $b^{[L]}$ promijenila kao što je prikazano u jednadžbi (4). Opisana metoda računanja parcijalnih derivacija za svaki od navedenih elemenata kao i oduzimanje gradijenata za članove $w^{[1]}$, $w^{[2]}$, ... $w^{[L]}$ i $b^{[1]}$, $b^{[2]}$, ... $b^{[L]}$ se naziva **algoritam propagacije pogreške unatrag.**

2.5 Unaprjeđivanje neuronske mreže

U ovom poglavlju opisano je kako se neuronske mreže mogu unaprijediti, a također su objašnjene i određene optimizacijske nadogradnje gradijentnog spusta, kao i optimizacijski problemi.

2.5.1 Trening, validacijski i testni podaci

Odabir kako postaviti trening, validacijski i testni skup podataka može uvelike utjecati na performanse neuronske mreže. Trening skup podataka je onaj skup podataka na kojem algoritam obavlja fazu učenja. Validacijski skup podataka je onaj skup koji algoritam koristi kako bi unaprijedio hiperparametre. Testni skup podataka služi tome da se provjeri točnost algoritma na novim podacima.

Neki od hiperparametara koje algoritam podešava na validacijskom skupu je broj slojeva, broj čvorova skrivenog sloja, stopa za učenje i aktivacijske funkcije. U praksi ne postoji neki definirani odabir hiperparametara - strojno i duboko učenje su iterativan proces koji se sastoji od ideje, programiranja i eksperimentiranja.

U prošloj eri dubokog učenja običaj je bio koristiti 70% podataka za učenje te 30% podataka za testiranje, ili 60% podataka za učenje te 20% podataka za validaciju i testiranje – u praksi se pokazalo da je na navedeni način najbolje raspodijeliti podatke te se tako obično raspodjeljivalo podatke. U novoj eri dubokog učenja, podataka ima dosta više, pa se podaci gledaju raspodijeliti na drukčiji način. Danas je običaj da podatkovni skup sadrži par milijuna podataka – u tom slučaju podaci se obično raspodijele na 98% podataka za treniranje, 1% za validaciju te 1% za testiranje [9].

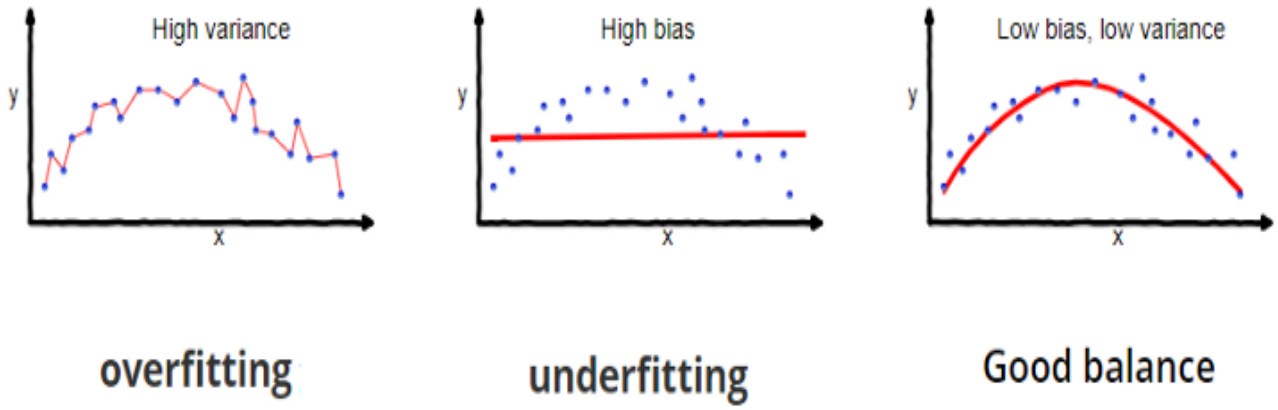
Pri odabiru podataka je bitno da podaci dolaze iz iste distribucije. Na primjer, podaci za učenje mogu biti slike s interneta, a podaci za validaciju i test mogu biti slike s mobilne aplikacije – to nije dobro, jer bi slike trebale biti iz iste distribucije.

2.5.2 Podučenost i prenaučenosti

Konačni cilj je stvoriti model koji ostvaruje dobre rezultate na novim podacima, dakle dobiti visoku točnost na testnim podacima.

Ako model ne ostvaruje dobre rezultate u fazi učenja, onda se smatra da je model *podučen* (engl. Underfitting), ili da ima visoku pristranost (engl. Bias). Ako model ostvaruje dosta visoke rezultate u fazi učenja, a niske u fazi testiranja, onda se smatra da model ima visoku varijancu te da je model *prenaučen* (engl. Overfitting).

Prva slika s lijeve strane pokazuje model visoke varijance, druga slika s lijeve strane visoku pristranost, dok slika skroz desno pokazuje nisku varijancu i nisku pristranost.



Slika 13 - Visoka varijanca, visoka pristranost i dobar balans [11]

Kada se gleda točnost modela bitno je gledati kolika je ljudska pogreška u odnosu na pogrešku modela. Na primjer, model koji ostvaruje 15% pogreške na trening podacima u odnosu na pogrešku čovjeka od 1% je podučen i ima visoku pristranost. Najbolja stvar koja se može dogoditi je niska varijanca i niska pristranost. Na primjer, za ljudsku pogrešku od 1%, 0.5% pogreške na trening podacima i 1% na test podacima je jako dobar rezultat. Najgora stvar koja se može dogoditi je visoka pristranost i visoka varijanca, na primjer na trening podacima greška od 15% te na test podacima greška od 30%.

Neke od stvari koje se mogu probati napraviti kako bi se riješio problem visoke podučenosti su povećavanje mreže, dakle veći broj hiperparametara, trenirati duže ili promijeniti arhitekturu mreže. Visoka prenaučenost se rješava s prikupljanjem više podataka, regularizacijom ili promjenom arhitekture.

2.5.3 Regularizacija

Kada postoji problem visoke varijance, korisno je koristiti metode **regularizacije**. Metode regularizacije će biti objašnjene pomoću ideje logističke regresije. Formula funkcije gubitka za logističku regresiju u kojoj je uključena regularizacija je:

$$J(w,b) = \frac{1}{m}\sum_1^m L\left(\hat{y}^{(i)}, y^{(i)}\right) + \frac{\lambda}{2m}\|w\|^2 \tag{7}$$

$\|w\|^2$ je zadano s:

$$\|w\|^2 = \sum_{j=1}^{n_x} w_j^2 = w^t w \tag{8}$$

Gore navedena regularizacija se zove **L2** regularizacija. **L1** regularizacija bi bila zadana sa sljedećom jednadžbom:

$$J(w,b) = \frac{1}{m}\sum_1^m L\left(\hat{y}^{(i)}, y^{(i)}\right) + \frac{\lambda}{2m}\|w\| \tag{9}$$

Funkcija gubitka neuronske mreže u kojoj je uključena regularizacija je:

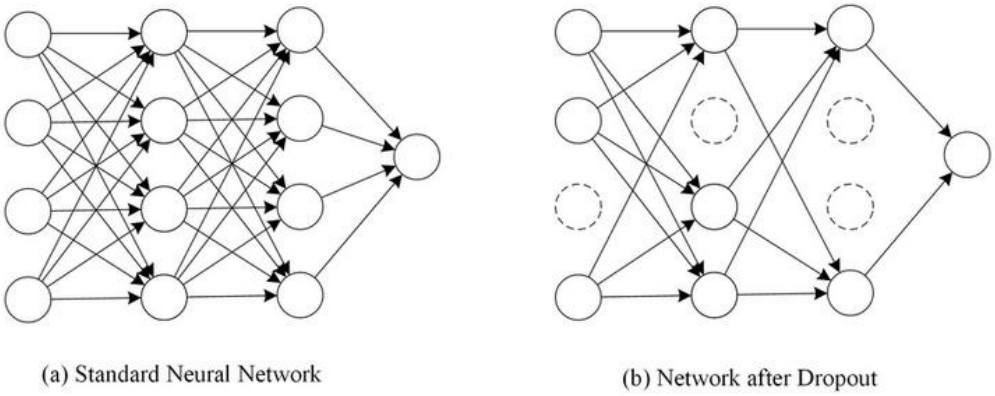
$$J\left(w^{[1]}, b^{[1]} \dots w^{[l]}, b^{[l]}\right) = \frac{1}{m}\sum_1^m L\left(\hat{y}^{(i)}, y^{(i)}\right) + \frac{\lambda}{2m}\sum_{l=1}^{L}\left\|w^{[l]}\right\|^2 \tag{10}$$

$\left\|w^{[l]}\right\|^2$ je izraz koja se zove **Frobeniusova norma**. Frobeniusova norma je zadana sljedećim izrazom:

$$\left\|w^{[l]}\right\|^2 = \sum_{i=1}^{n^l}\sum_{j=1}^{n^{[l-1]}}(w_{i,j}^{[l]})^2 \tag{11}$$

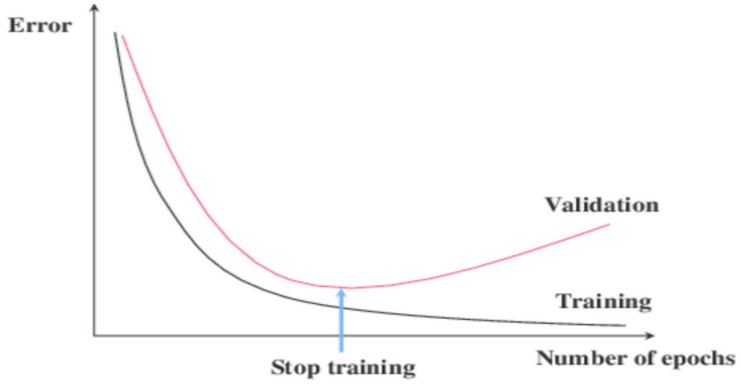
Metoda koja se često koristi kada se hoće ostvariti regularizacija je **nestajanje neurona** (engl. Dropout). *Dropout* je metoda gdje se hiperparametrom $p \in [0,1]$ zadaje vjerojatnost hoće li se odbaciti ili zadržati određeni neuron.

Sljedeća slika je ilustracija nestajanja neurona.



Slika 14 - Neuronska mreža prije i nakon nestajanje neurona [10]

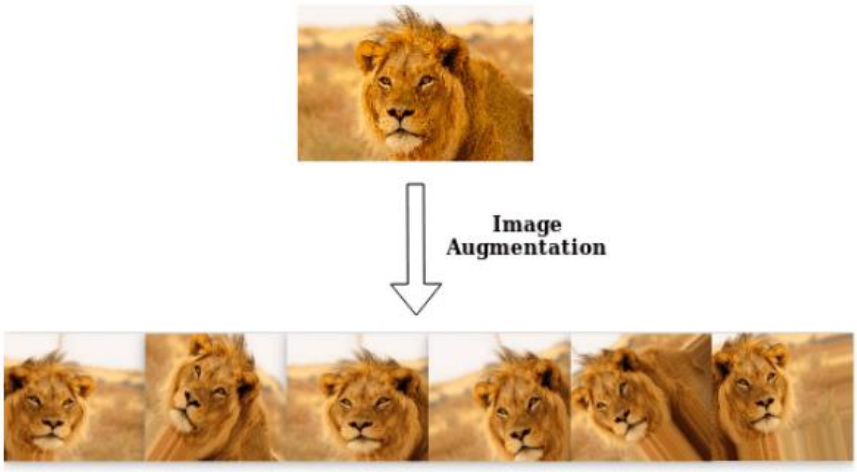
Rano zaustavljanje (engl. Early Stopping) je metoda regularizacije kod koje se nakon svake epohe radi validacija, pa ako se ustvrdi da se učenje pogoršava, onda se staje s učenjem. Sljedeća slika je ilustracija ranog zaustavljanja. Nakon određenih epoha kada se pogoršava učenje na validacijskom skupu, onda se zaustavlja s treniranjem.



Slika 15 - Ilustracija ranog zaustavljanja [11]

Ostale metode kojima se može postići regularizacija su **augmentacija podataka** i **normalizacija grupe** (engl. Batch Normalization). Augmentacija podataka podrazumijeva smisleno kreiranje novih podataka. Na primjer, rotacijom ili zrcaljenjem slike se dobiva nova slika, ne utječe se na semantiku, a generira se novi podatak.

Sljedeća slika prikazuje razne tehnike augmentacije slike.



Slika 16 - Primjeri augmentacije slike [12]

Normalizacija grupe ima više prednosti. Normalizacijom grupe se prikuplja statistika, prosjek i standardna devijacija značajka, te potom normalizira značajke tako da značajke, to jest vrijednosti slike, budu s prosjekom nula i standardnom devijacijom jedan. Korisno je koristiti normalizaciju grupe jer se onda izbjegava dominantnost jedne značajke nad drugom značajkom, treniranje se obavlja brže te se izbjegava da se određeni sloj ne određuje uvelike promjenom vrijednosti nekog drugog sloja.

2.5.4 Optimizacijski problemi

Jedna od tehnika koja može ubrzati treniranje neuronske mreže je normaliziranje ulaznih veličina. Normaliziranje ulaznih veličina je slično normalizaciji grupe, oduzima se prosjek značajka tako da prosjek značajka bude nula te se vrijednosti značajka dijele sa standardnom devijacijom značajka kako bi standardna devijacija značajka bila 1. Osim navedenog, korisno je i značajke postaviti na vrijednosti 0 – 1, jer se onda postiže brže treniranje te krivulje funkcije gubitka postaju simetričnije [13].

Slika 17 - Normaliziranje funkcije dovodi do simetričnosti kontura funkcije gubitka [13]

Jedan od problema treniranja neuronskih mreža, posebno dubokih neuronskih mreža, su eksplodirajući gradijenti. Uzrok nastajanja eksplodirajućih gradijenata je dijeljenje parametara kroz velik broj uzastopnih slojeva [14]. Jedno od rješenja problema eksplodirajućih gradijenata jest da se gradijentima doda vrijednost iz normalne distribucije $N(0, \sigma^2)$ pri svakoj promjeni težina.

Kada se implementira unazadna propagacija, može se dogoditi da je krivo korištena određena jednadžba. Da bi se provjerila implementacija unazadne propagacije, onda se može koristiti metoda provjere gradijenata. Da bi se objasnila metoda provjere gradijenata, prvo je potrebno objasniti numeričku aproksimaciju gradijenta.

Bolja aproksimacija derivacije je da se osim $\theta + \varepsilon$ uzme i udaljenost $\theta - e$ [16], dakle koristi se formula:

$$\lim_{\epsilon \to 0} \frac{f(\theta+e) - f(\theta-e)}{2e} \tag{12}$$

Aproksimacija greške koristeći jednadžbu (12) je O(ε^3) [16].

Da bi se odradila provjera gradijenata potrebno je parametre $w^{[1]}$, $b^{[1]}$,...,$w^{[L]}$,$b^{[L]}$ združiti u jedan veliki vektor θ. Također, potrebno je istu stvar napraviti s gradijentima, združiti gradijente $dw^{[1]}$, $db^{[1]}$,...,$dw^{[L]}$, $db^{[L]}$ i pretvoriti ih u jedan veliki vektor $d\theta$. Pitanje koje se postavlja jest je li $d\theta$ gradijent funkcije gubitka. Izračuna se aproksimacija gradijenta kao što je prikazano pod jednadžbom (12) za svaki parametar koristeći sljedeću formulu:

$$d\theta_{aproksimacija\_i} = \frac{J(\theta_1,\theta_2,\ldots,\theta_i+e,\ldots) - J(\theta_1,\theta_2,\ldots,\theta_i+e,\ldots)}{2e} \tag{13}$$

Ako je euklidska vrijednost razlike vektora $d\theta_{aproksimacija\_i}$ i $d\theta_{\_i}$ otprilike 10^{-7}, onda je to odraz dobre unazadne propagacije, ako nije, onda se vjerojatno dogodila greška pri implementaciji unazadne propagacije [16].

Problem lokalnog minimuma je problem koji se javlja kod nekonveksnih funkcija. Konveksne funkcije imaju svojstvo koje problem optimizacije svodi na problem pronalaska lokalnog minimuma, jer svaki lokalni minimum ujedno predstavlja globalni minimum. Ako se radi o nekonveksnim funkcijama, kao što je na primjer funkcija gubitka, vrlo je lako moguće da postoji više lokalnih minimuma. Problem nastaje kada lokalni minimumi imaju dosta veći gubitak u odnosu na globalni minimum. Ovaj problem je vrijedan proučavanja i danas još uvijek ne postoji neko definirano rješenje kako doći do globalnog minimuma. Neovisno o navedenom problemu, pri korištenju neuronskih mreža se postižu dosta dobri rezultati, jer većina neuronskih mreža ima male vrijednosti funkcije gubitka u lokalnim minimumima.

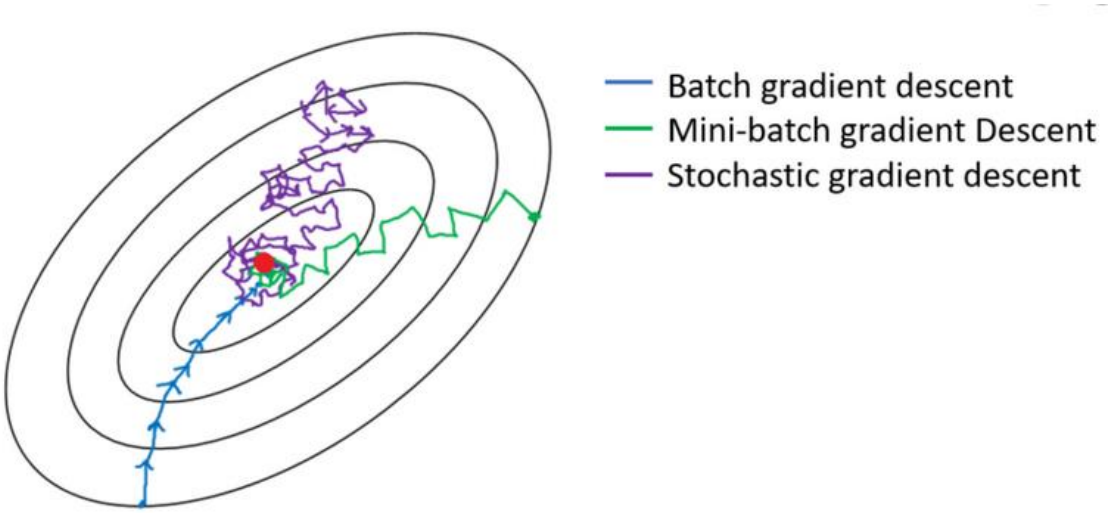
Problem sedlaste točke je problem koji se javlja rjeđe u odnosu na problem lokalnog minimuma. U sedlastim točkama gradijenti funkcije gubitka su nula pa se učenje neuronske mreže značajno usporava. S porastom dimenzionalnosti broj sedlastih točka se povećava.

2.5.5 Optimizacijski algoritmi

Gradijentni spust kojim se postiže minimizacija funkcije gubitka nije jedini algoritam koji se može koristiti za minimizaciju funkcije gubitka. Navedeni algoritam koji je spomenut na engleskom jeziku se zove *Batch Gradient Descent*, a pojam *batch* podrazumijeva da se gradijent funkcije gubitka s obzirom na određene parametre računa za cijeli set podataka. Dakle, ako se koristi *batch* gradijentni spust, onda se za jedno ažuriranje parametara koristi cijeli set podataka. Kako se svaki put koristi cijeli set podataka, onda je navedeni algoritam dosta spor te zahtijeva ogromne količine radne memorije, jer cijeli podatkovni skup treba stati u memoriju prilikom računanja gradijentnog spusta.

Kako bi se uštedjelo na memoriji, ali i ubrzalo pronalaženje globalnog minimuma, onda se može koristiti *Mini-batch* gradijentni spust. Kod *Mini-batch* gradijentnog spusta podatkovni skup se rasporedi u dijelove (engl. Batch), te se onda na svakom manjem dijelu, dakle ne cijelom trening skupu, računa gradijent vrijednosti parametara.

Stohastički gradijentni spust je optimizacijski algoritam kod kojeg se računa gradijent na samo jednom primjeru skupa podataka. Ako je na raspolaganju dosta memorije, onda nije dobro koristiti stohastički gradijentni spust, jer se zbog vektorizacije može skratiti vrijeme treniranja modela.



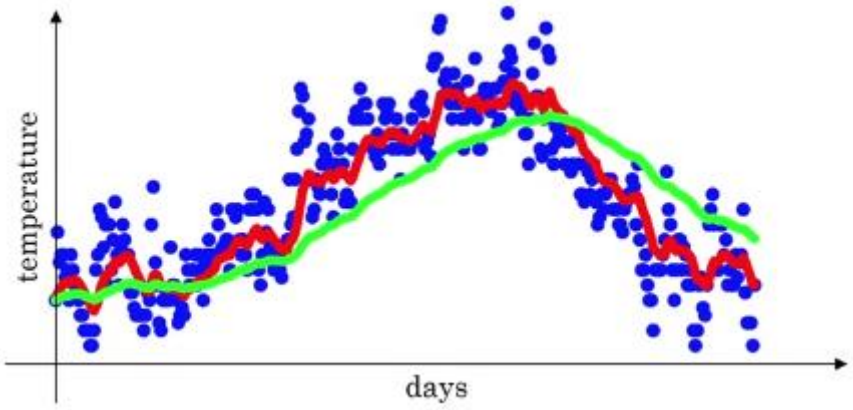
Slika 18 - Različite trajektorije optimizacijskih algoritama [17]

Da bi se objasnili sljedeći optimizacijski algoritmi, najprije je potrebno objasniti pojam eksponencijalnog pomičnog prosjeka (engl. Exponential Moving Average), jer se sljedeći algoritmi temelje na eksponencijalnom pomičnom prosjeku.

Ako se uzme za primjer određivanje temperature za određene dane, lokalni trend temperature može se definirati formulom:

$$v_t = \beta v_{t-1} + (1 - \beta)\theta_t \tag{14}$$

Na sljedećoj slici prikazan je pomični prosjek za vrijednost β = 0.9 (crvena krivulja) i β = 0.98 (zelena krivulja). Što je veća vrijednost za β, to je onda krivulja glađa i manje osjetljiva na granične vrijednosti.



Slika 19 - Eksponencijalni pomični prosjek temperature [18]

Otprilike, za vrijednost $\frac{1}{1-\beta}$ se uzima prosjek dnevne temperature. Na primjer, ako je β = 0.9, onda se uzima prosjek za 10 dana dnevne temperature, ako je β = 0.98, onda se uzima prosjek od 50 dana dnevne temperature.

Gradijentni spust s momentom je tip gradijentnog spusta koji je gotovo uvijek brži od klasičnog gradijentnog spusta, a temelji se na ideji računanja eksponencijalnih pomičnih prosjeka.

Ako se uzmu u obzir trajektorije sa slike 18, može se vidjeti da se gotovo uvijek hoće smanjiti pomak vertikalno, a povećati pomak horizontalno. Jedna od ideja kako se to može ostvariti je da se koristi gradijentni spust s momentom. Gradijentni spust s momentom se izvodi na sljedeći način: pri svakoj iteraciji *t* se računaju derivacije koeficijenata za trenutni *mini-batch* te eksponencijalne pomične prosjeke za svaki od koeficijenata koristeći jednadžbu (14), na kraju se promijene vrijednosti parametara tako da se vrijednost parametra oduzme s produktom faktora za učenje i eksponencijalnog pomičnog prosjeka za dani koeficijent.

Drugi dosta korišten algoritam je **RMSprop** (engl. Root Mean Square Prop).

Koristeći jednadžbu (14), kvadrirani parametri *w* i *b* ažuriraju se na sljedeći način:

$$S_{dw} = \beta\, S_{dw} + (1-\beta)dw^2 \tag{15}$$

$$S_{db} = \beta\, S_{db} + (1-\beta)db^2 \tag{16}$$

Ažuriranje koeficijenata *w* i *b* se vrši na način sljedeći način:

$$w = w- \propto \frac{dw}{\sqrt{S_{dw}}} \tag{17}$$

$$b = b- \propto \frac{db}{\sqrt{S_{db}}} \tag{18}$$

Adam je uz RMSprop jedan od često korištenih algoritama. Adam kombinira gradijentni spust s momentom zajedno s RMSprop-om. Pri svakoj iteraciji *t* treba izračunati derivacije *dw* i *db* na trenutnom *mini-batchu*, nakon toga treba izračunati eksponencijalni pomični prosjek parametara *w* i *b* s momentom:

$$V_{dw} = \beta_1\, V_{dw} + (1-\beta_1)dw \tag{19}$$

$$V_{db} = \beta_1\, V_{db} + (1-\beta_1)db \tag{20}$$

Zatim se izračunava eksponencijalni pomični prosjek kao kod RMSprop algoritma, dakle:

$$S_{dw} = \beta_2\, S_{dw} + (1-\beta_2)dw^2 \tag{21}$$

$$S_{db} = \beta_2\, S_{db} + (1-\beta_2)db^2 \tag{22}$$

Kako bi se riješio problem greške pristranosti [17] koristi se sljedeći račun:

$$V_{dw}^{ispravljeno} = \frac{V_{dw}}{(1-{\beta_1}^t)} \tag{23}$$

$$V_{db}^{ispravljeno} = \frac{V_{db}}{(1-{\beta_1}^t)} \tag{24}$$

$$S_{dw}^{ispravljeno} = \frac{S_{dw}}{(1-{\beta_2}^t)} \tag{25}$$

$$S_{db}^{ispravljeno} = \frac{S_{db}}{(1-\beta_2{}^t)} \tag{26}$$

Ažuriranje parametara se vrši na sljedeći način:

$$w = w - \propto \frac{V_{dw}^{ispravljeno}}{\sqrt{S_{dw}^{ispravljeno}}} \tag{27}$$

$$b = b - \propto \frac{V_{db}^{ispravljeno}}{\sqrt{S_{db}^{ispravljeno}}} \tag{28}$$

Jedna od tehnika koja se može koristiti za ubrzavanje učenja jest **smanjenje koeficijenta učenja** (engl. Learning Rate Decay). Dakle, kako vrijeme prolazi, kako algoritam uči kroz razne epohe, tako se koeficijent za učenje s vremenom smanjuje. Prednost korištenja smanjenja koeficijenta za učenje je veća vjerojatnost konvergiranja ka globalnom minimumu [18].

Formula koja se može koristiti za smanjenje stope učenja α je:

$$\propto = \frac{1}{1+brzina_{smanjenja} * broj_{epoha}} \propto_0 \tag{29}$$

Eksponencijalno smanjenje stope za učenje je definirano formulom:

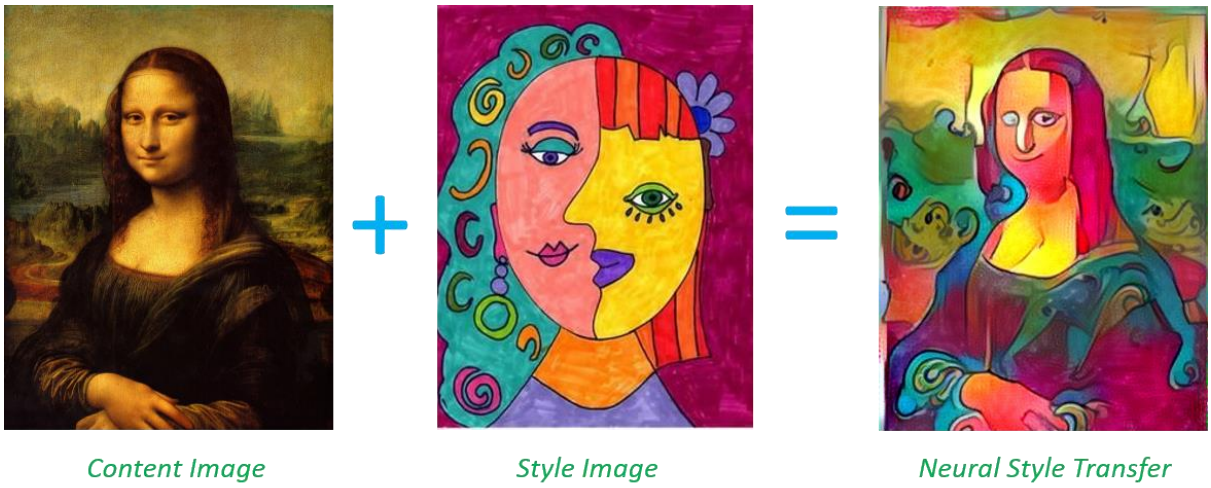
$$\propto = a^{broj\_epoha} \propto_0 , \qquad a < 1 \tag{29}$$

Jedna od metoda smanjenja koeficijenta za učenje je i diskretno smanjenje stope za učenje – za određene epohe postoji diskretna vrijednost stope za učenje.

3. KONVOLUCIJSKE NEURONSKE MREŽE

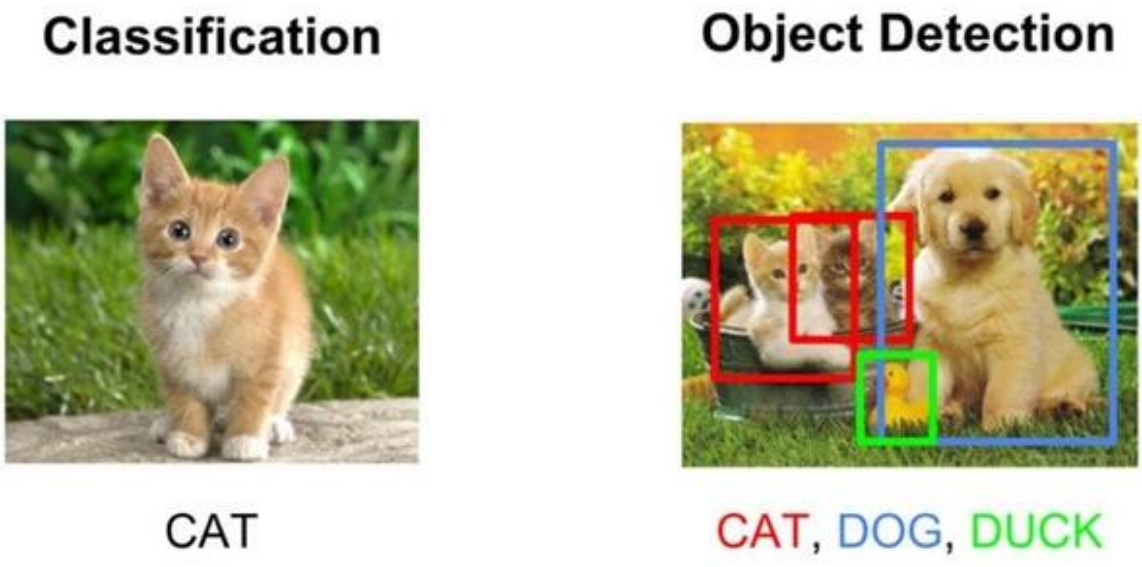
3.1. Računalni vid

Računalni vid je tehnologija koja se rapidno unaprijedila zahvaljujući neuronskim mrežama. Duboko učenje omogućuje računalnom vidu rješavanje određenih problema kao što je na primjer autonomna vožnja i prepoznavanje lica. U današnje vrijeme već se pomoću prepoznavanja lica može otključati pametni mobitel ili ulazna vrata kuće. Duboko učenje čak omogućava stvaranje nove vrste umjetnosti – na temelju slike sadržaja i slike stila nastaje nova slika koja sadrži kombinaciju obje navedene slike – navedena tehnika se zove neuronski prijenos stila. Sljedeća slika prikazuje neuronski prijenos stila.



Slika 20 - Primjer neuronskog prijenosa stila [22]

Klasifikacija slike i objektna detekcija su problemi koji su najviše zastupljeni u području računalnog vida. Klasifikacija slike podrazumijeva određivanje koja klasa je prisutna na slici, a objektna detekcija lociranje određene klase na slici.

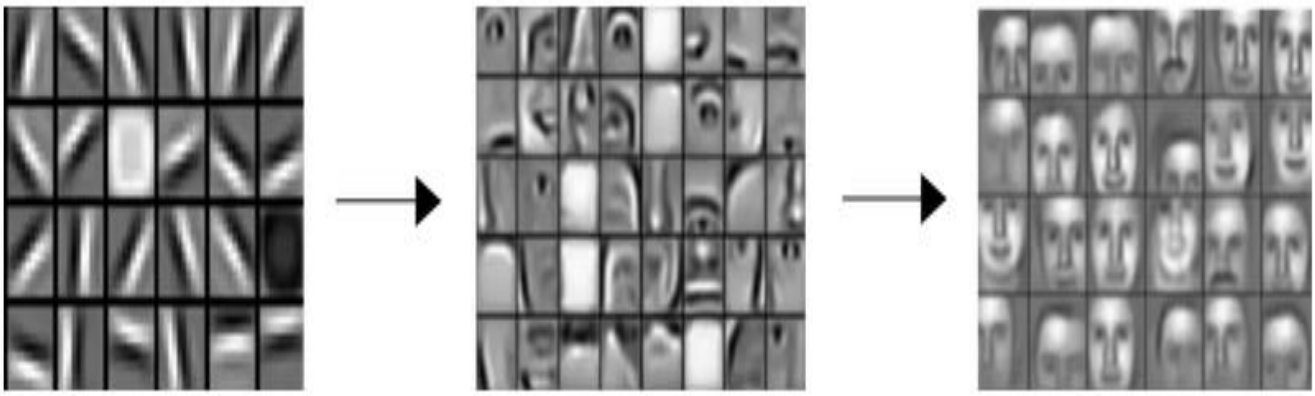


Slika 21 - Primjer klasifikacije slike i objektne detekcije [23]

Jedan od izazova računalnog vida je da podaci ulaznog sloja mogu zauzimati dosta memorije. Potrebno je pronaći određena rješenja kako bi se radilo sa slikama većih dimenzija – jedno od rješenja su konvolucijske neuronske mreže.

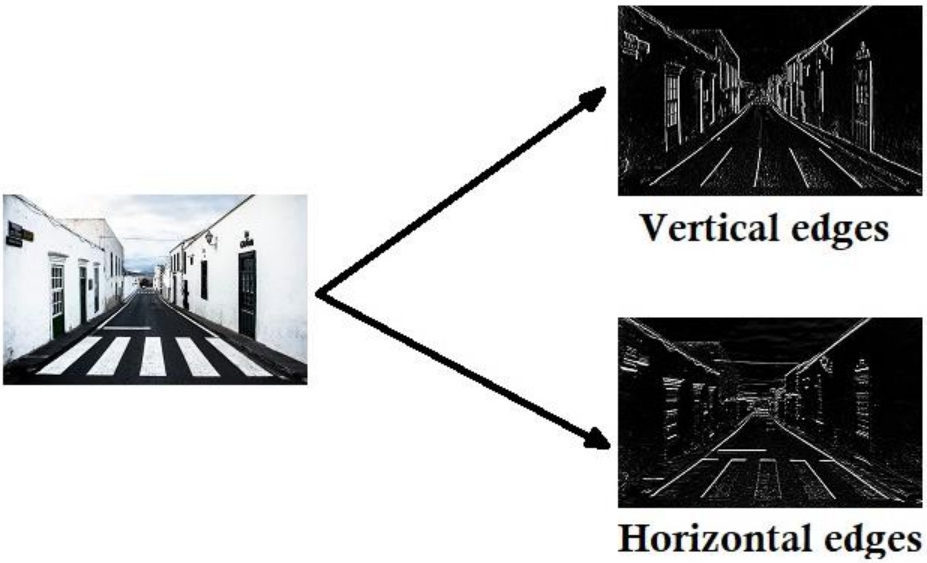
3.2 Konvolucija i problem određivanja ruba

Operacija konvolucije je temelj konvolucijskih neuronskih mreža. Matematički gledano, konvolucija je integral koji mjeri koliko se dvije funkcije podudaraju dok jedna funkcija prolazi kroz drugu. Koristeći detekciju vertikalnog ruba kao primjer, objasnit će se kako funkcionira operator konvolucije. Obično raniji slojevi neuronske mreže otkrivaju jednostavnije značajke, kao na primjer rubove. Kasniji slojevi otkrivaju kompleksnije značajke, kao na primjer lice.



Slika 22 - Primjer što raniji, srednji i kasniji slojevi uče [24]

Sljedeća slika prikazuje ulaznu sliku te dvije izlazne slike. Gonja izlazna slika je prikaz detekcije vertikalnih rubova, dok je donja izlazna slika prikaz detekcije horizontalnih rubova.



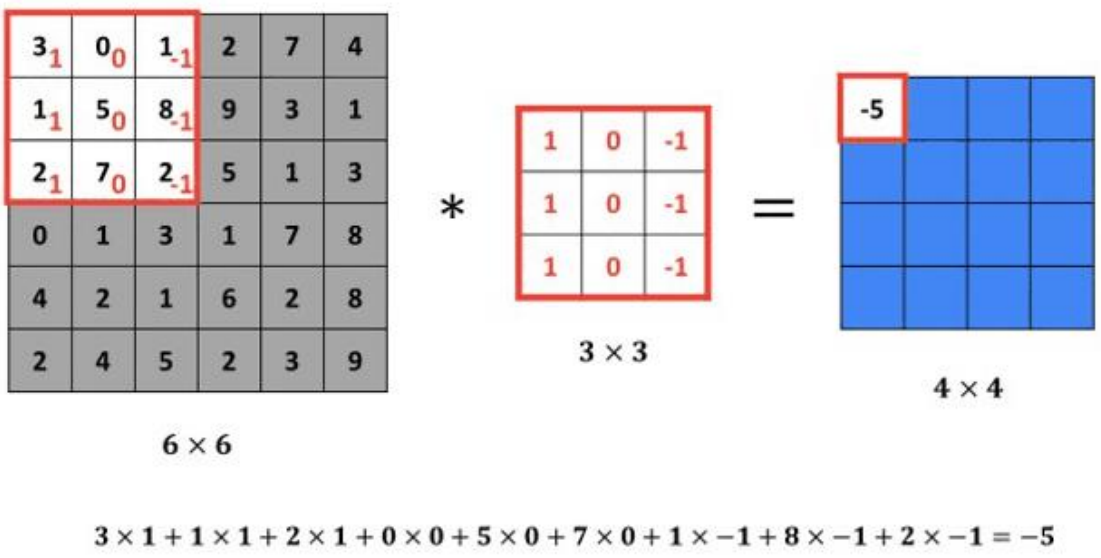
Slika 23 - Primjer određivanja vertikalnih i horizontalnih rubova [24]

Sljedeća slika prikazuje matricu koja sadrži vrijednosti određene slike. U ovom primjeru koristi se jedna dimenzije dubine. Slika je dimenzija 6 x 6 x 1.

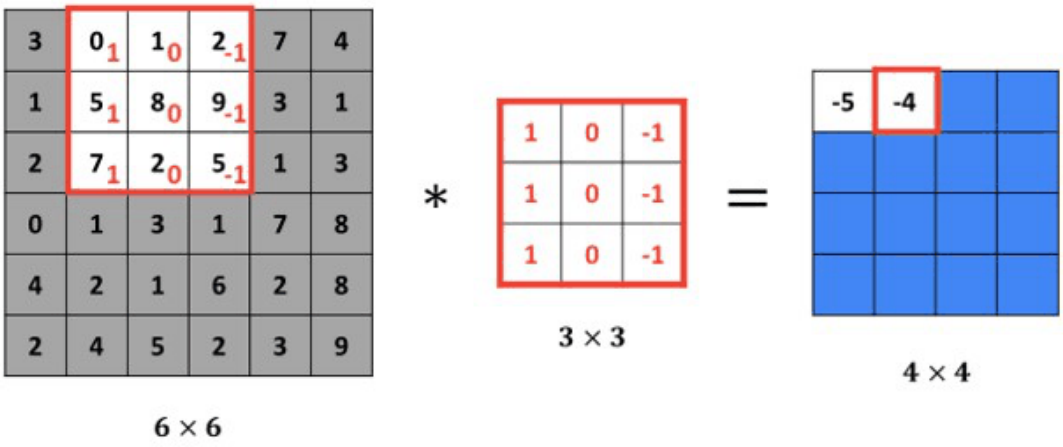
3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

Slika 24 - Primjer slike dimenzija 6 x 6 x 1 [24]

U danom primjeru, na slici 25., matrica 3 x 3 je **filter**. U literaturi se nekad umjesto riječi filter koristi riječ jezgra. Rezultat konvoluiranja ulazne slike dimenzija 6 x 6 s filterom 3 x 3 je izlazna slika dimenzija 4 x 4. Način kako se dobije izlazna slika 4 x 4 je sljedeći: da bi se izračunao prvi element, dakle gornji lijevi kut 4 x 4 matrice, uzima se 3 x 3 dio slike gornjeg lijevog kuta, a zatim se matričnim produktom množi odsječeni dio slike s filterom. Postupak se ponavlja sve dok se filter ne zalijepi na zadnju poziciju na koju se može zalijepiti.

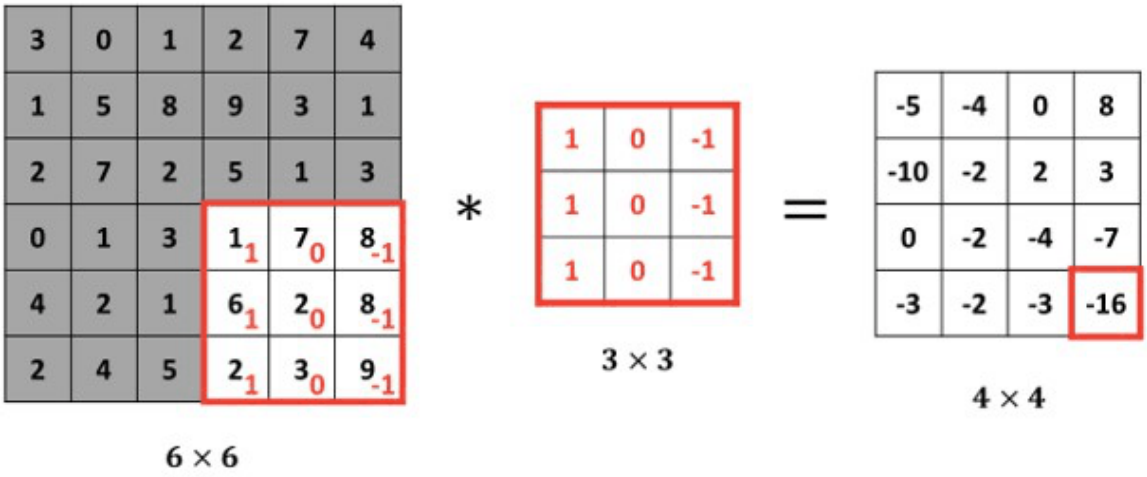


Slika 25 - Primjer dobivanja prve vrijednosti konvolucije [24]



$$0 \times 1 + 5 \times 1 + 7 \times 1 + 1 \times 0 + 8 \times 0 + 2 \times 0 + 2 \times -1 + 9 \times -1 + 5 \times -1 = -4$$

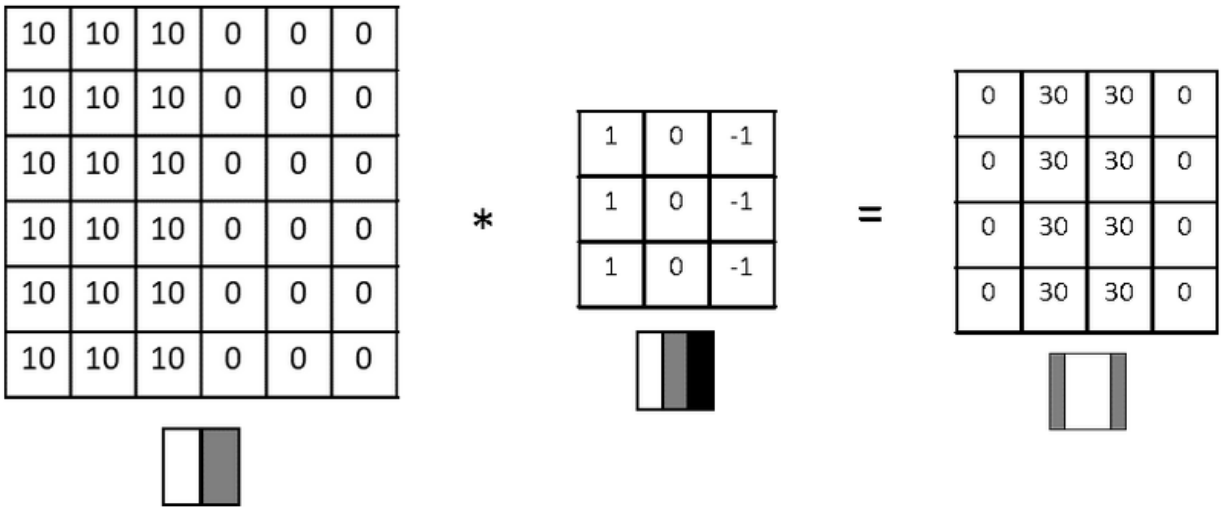
Slika 26 - Primjer dobivanja druge vrijednosti konvolucije [24]



$$1 \times 1 + 6 \times 1 + 2 \times 1 + 7 \times 0 + 2 \times 0 + 3 \times 0 + 8 \times -1 + 8 \times -1 + 9 \times -1 = -16$$

Slika 27 - Primjer dobivanja zadnje vrijednosti konvolucije [24]

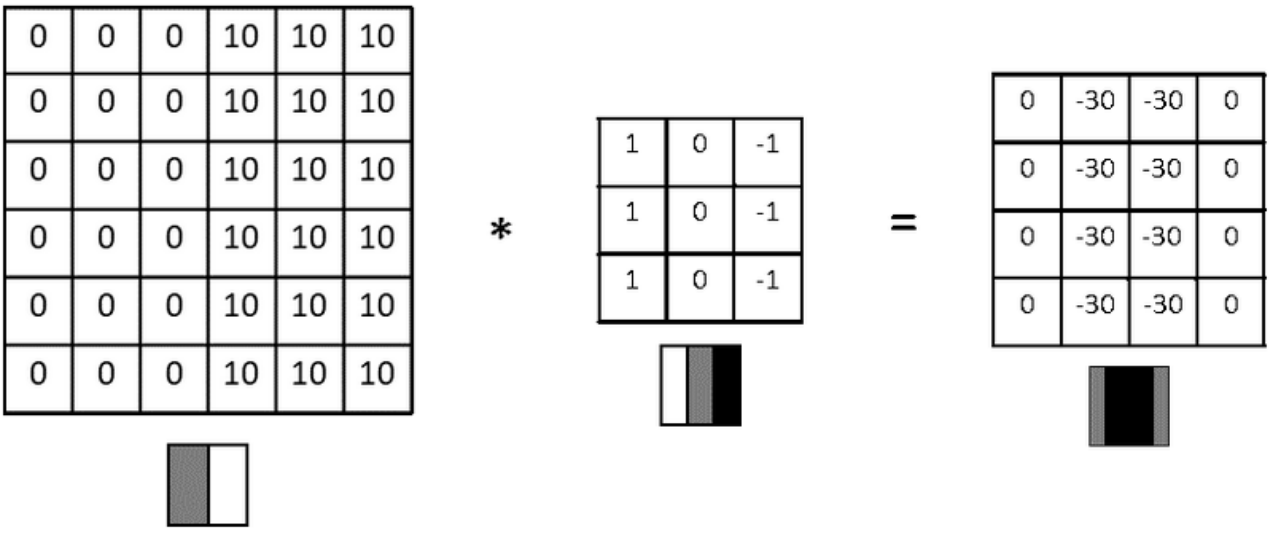
Navedeni postupak predstavlja određivanje vertikalnih rubova. Dakle, izlazna slika 4 x 4 pokazuje vertikalne rubove ulazne slike dimenzija 6 x 6.



Slika 28 - Primjer slike koja je na lijevoj strani bijele, a na desnoj sive boje [25]

Na slici 28, lijevi dio slike prikazuje sliku koja je s lijeve strane obojena bijelom, a s desne strane sivom bojom. Ako se slika s lijeve strane konvoluira s prikazanim filterom u sredini, onda je izlazna slika koja sa strana ima sivu, a u sredini bijelu boju.

Sljedeća slika s lijeve strane prikazuje sliku koja je s lijeve strane sive, a s desne strane bijele boje.



Slika 29 - Primjer slike koja je na lijevoj strani sive, a na desnoj bijele boje [25]

Na prethodnoj slici može se vidjeti da su vrijednosti izlazne slike osim 0 sada -30, dakle ne 30 kao u prethodnom primjeru. Izlazna slika je prikazana u dvije nijanse sive – u sredini je crna, a na desnoj siva boja.

Na sljedećim slikama su prikazani vertikalni i horizontalni filter.

Slika 30 - Vertikalni i horizontalni filter [25]

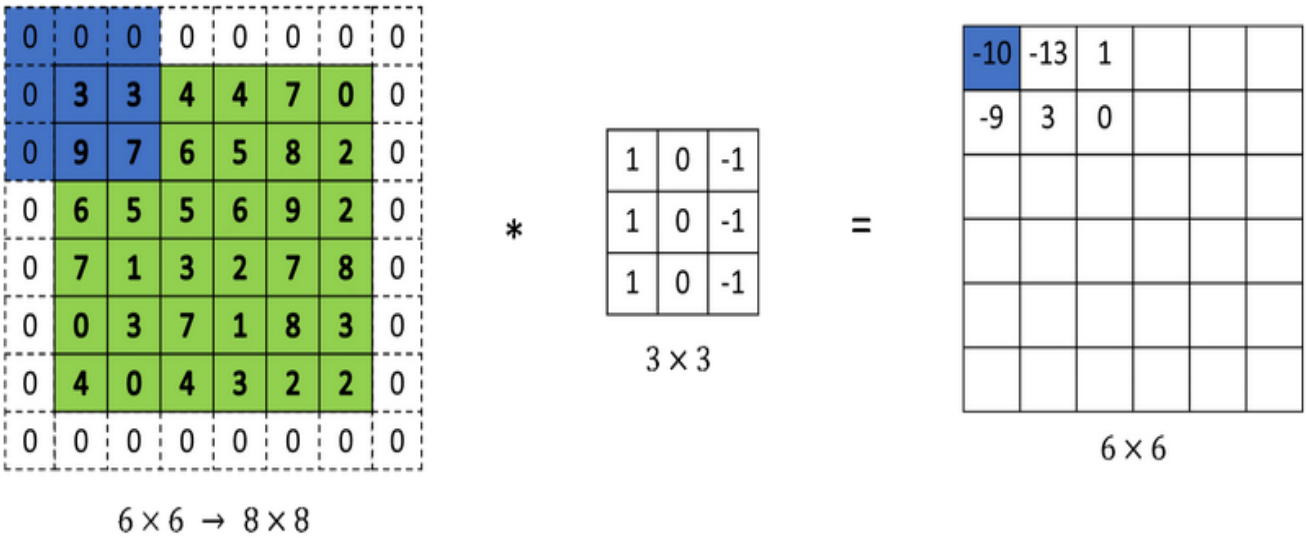
Ostali dosta korišteni filteri su Sobelov i Scharrov filter. Na sljedećoj slici su prikazani vrijednosti navedena dva filtera.

Slika 31 - Sobelov i Scharrov filter [25]

S razvojem dubokog učenja, jedna od stvari koja je naučena jest da kada je cilj detektirati rubove na slici koja posjeduje kompleksne značajke, onda se ne mora uzeti 9 vrijednosti koji su istraživači iz područja računalnog vida već odredili, već se može pomoću unazadne propagacije doći do novih filtera [25].

3.3 Nadopunjavanje

Nadopunjavanje (engl. Padding) je često korištena tehnika koja se primjenjuje kod konvolucijskih neuronskih mreža. U prethodnim primjera vidljivo je da ako je ulazna slika dimenzija 6 x 6, te se konvoluira s filterom 3 x 3, onda je izlazna slika dimenzija 4 x 4. Generalno, ako se radi o slici dimenzija *n x n* koja se konvoluira s *f x f* filterom, dimenzije izlazne slike su *(n – f + 1) x (n – f +1)*. Prema navedenom, mogu se uvidjeti dvije „mane“ konvolucija: (1) svaki put kada se obavi konvolucija slika se smanjuje, (2) ako se uzme u obzir piksel na rubu slike, taj se piksel samo jednom koristi u izlaznim podacima, to jest izlaznoj slici. Ako se uzme u obzir piksel u sredini, onda se taj piksel dosta puta koristi u izlaznim podacima. Iz navedenog se može zaključiti da se na izlaznoj slici gubi informacija o pikselima koji se nalaze na rubovima. Kako se ulazna slika ne bi smanjila, i kako se ne bi izgubila informacija o podacima koji se nalaze na rubovima slike, onda se koristi nadopunjavanje. Operacija nadopunjavanja nadopunjuje slici vrijednosti 0 izvan rubova slike. Na sljedećoj slici je prikazana operacija nadopunjavanja.



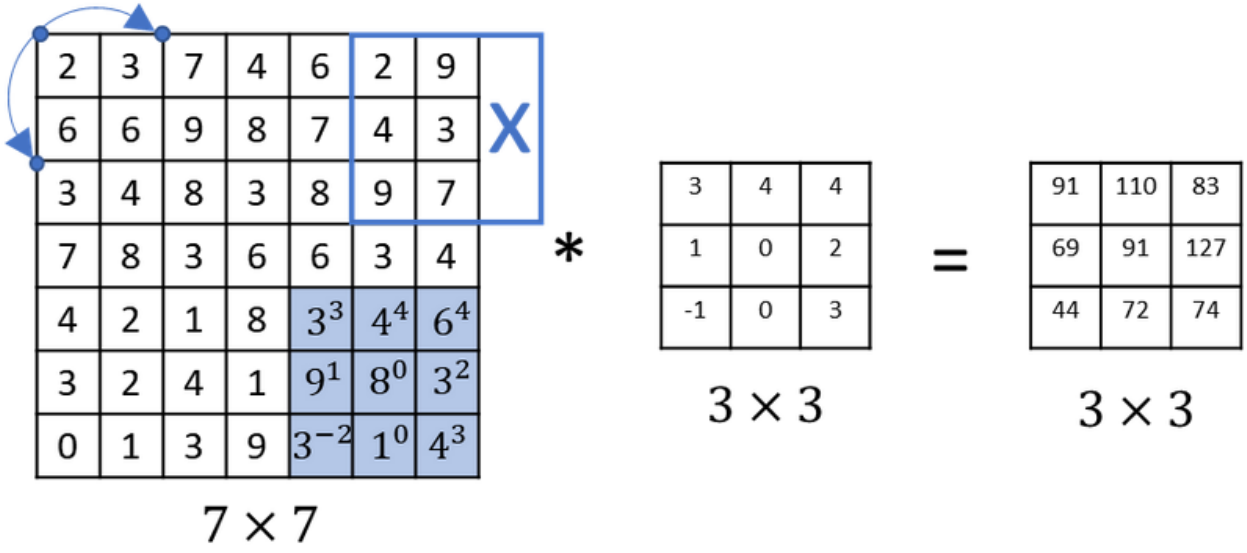
Slika 32 - Primjer nadopunjavanja [26]

Može se generalno zaključiti da kada se koristi nadopunjavanje (*p*), onda će dimenzije izlazne slike biti *(n + 2p - f + 1) x (n + 2p – f + 1)*.

Najčešće dvije korištene metode nadopunjava su *valid* i *same*. Ako se koristi *valid* nadopunjavanje, onda se slika ne nadopunjuje, dakle izlazna dimenzija slika će biti dimenzija

(n – f + 1) x (n – f + 1). Ako se koristi *same* nadopunjavanje, onda će dimenzije izlazne slike biti jednake dimenzijama ulazne.

U prethodnim primjerima vezanim za konvoluiranje, može se vidjeti da se filter pomaknuo za jedan, to jest, kada je prvotno filter „prilijepljen“ na gornji lijevi dio slike, onda je ponovno filter za jednom u desno prilijepljen na drugo mjesto. U slučaju da se filter pomaknuo za 2 mjesta, onda bi se moglo reći da se napravio korak (engl. Stride) 2. Sljedeća slika prikazuje konvoluciju s pomakom 2.



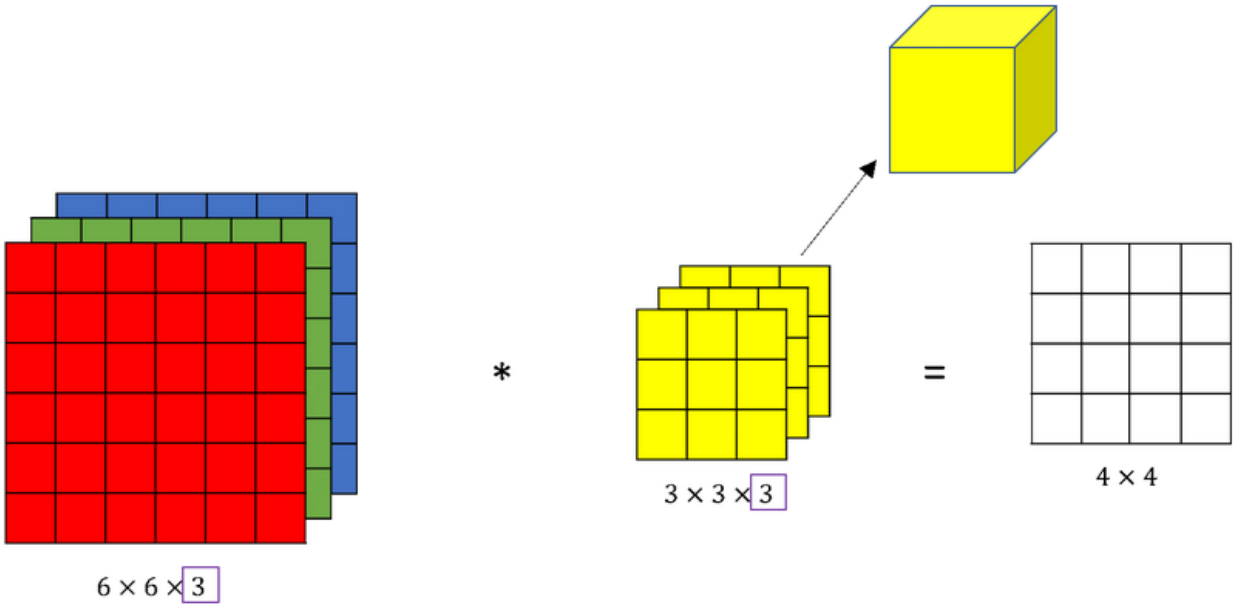
Slika 33 - Konvolucija s pomakom 2 [27]

Ako se slika dimenzija n x n, konvoluira s filterom f x f, koristi nadopunjavanje *p* i pomak *s*, onda će dimenzije izlazne slike biti

$$\left[\frac{n-f+2p}{s}\right]+1 \; x \left[\frac{n-f+2p}{s}\right]+1$$

3.4 Konvolucija na slikama s tri dimenzije

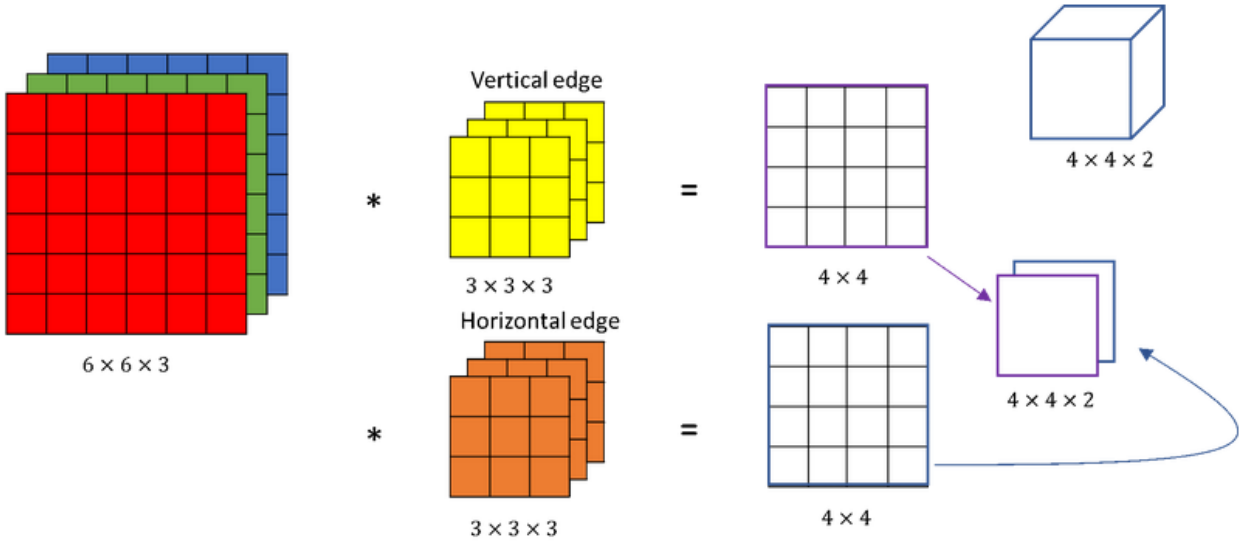
Da bi se 3D slika konvoluirala, onda se mora koristiti filter dubine 3 [28]. Na sljedećoj slici prikazana je ulazna slika dimenzija 6 x 6 x 3, filter dimenzija 3 x 3 x 3 te izlazna slika 4 x 4 koja je rezultat konvolucije.



Slika 34 - Primjer konvoluiranja 3D slike [28]

Da bi se izračunala prva vrijednost izlazne slike, uzima se filter dimenzija 3 x 3 x 3 te prisloni u gornji lijevi rub slike dimenzija 6 x 6 x 3. Svaki od 27 brojeva filtera (9 vrijednosti crvenog, zelenog i plavog dijela slike) se pomnoži s brojevima koji se nalaze na istim područjima slike – na kraju se zbroje svi umnošci te se time dobije prva vrijednost izlazne slike. Da bi se dobila sljedeća vrijednost, kocku se pomakne jednom u desno – tako sve do kraja dok se kocka ne postavi na svaku moguću poziciju.

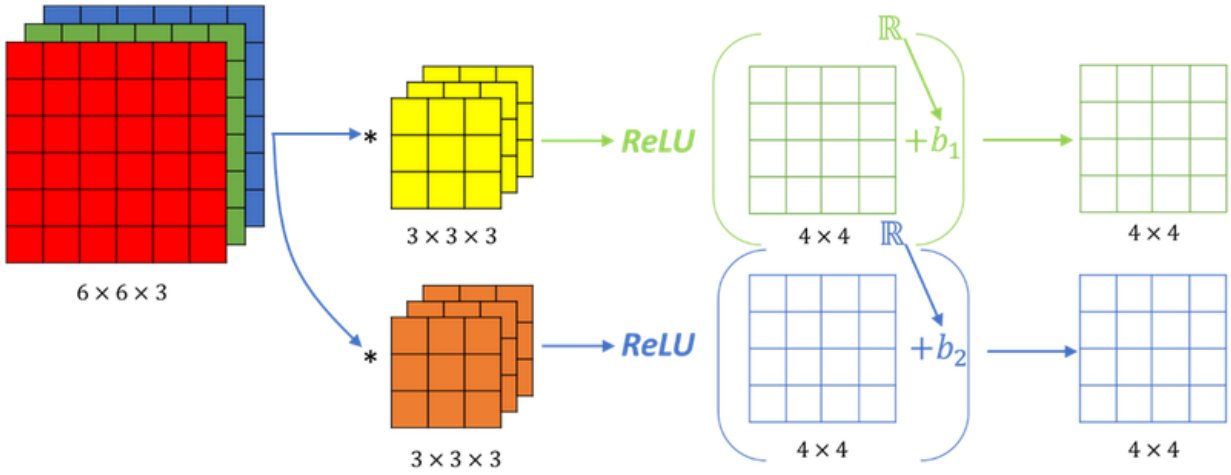
Postavlja se pitanje što ako se hoće koristiti više filtera, na primjer filter za detekciju horizontalnih rubova i filter za detekciju vertikalnih rubova. U navedenom slučaju za dva filtera, dubina izlazne slike je 2, jedna dimenzija predstavlja horizontalne, a druga vertikalne rubove.



Slika 35 - Primjer konvoluiranja 3D slike s horizontalnim i vertikalnim filterima [28]

3.5 Jedan sloj konvolucijske neuronske mreže

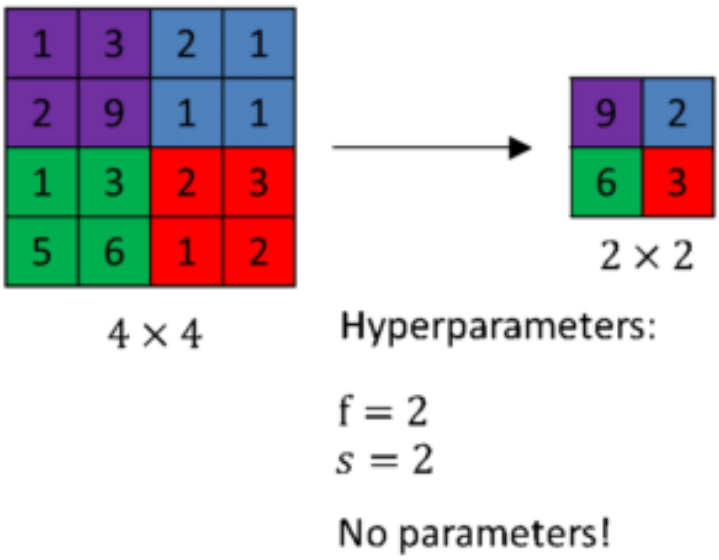
U prethodnim primjerima može se vidjeti da su dvije izlazne slike rezultat konvoluiranja s dva filtera. Za dobiti konvolucijski sloj, dobivenim vrijednostima slike se doda vrijednost otklona te se aplicira aktivacijska funkcija, na primjer ReLU.



Slika 36 - Jedan sloj konvolucijske neuronske mreže [29]

3.6 Slojevi sažimanja

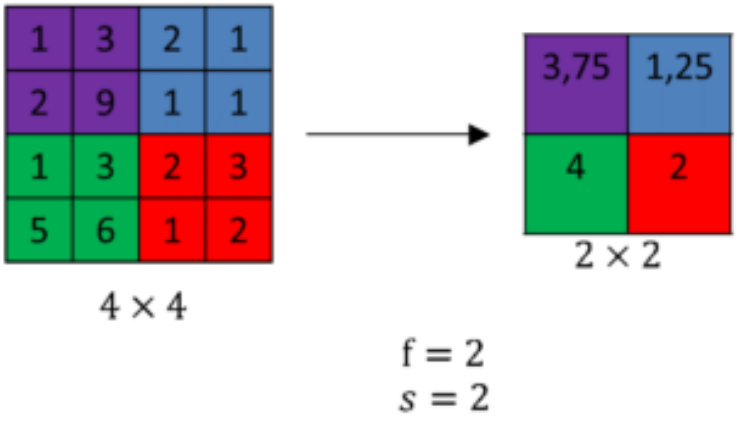
Slojevi sažimanja se koriste kako bi se dimenzije značajka smanjile. Smanjenjem dimenzija značajka se dobiva na brizni. Dvije vrste sažimanja koje se često koriste u praksi su *max* i *average.* Sažimanje se može koristiti s pomakom, kao i konvolucija. Sljedeća slika pokazuje maksimalno sažimanje dimenzija 2 x 2 s pomakom 2.



Slika 37 - Primjer sloja sažimanja [30]

Procedura maksimalnog sažimanja je sljedeća: određuje se veličina filtera te se filter priljepljuje na sliku, uzima se najveća vrijednost na prilijepljenom djelu slike te se ta vrijednost koristi u izlaznoj vrijednosti slike. Na primjer, na gornjoj slici je vidljivo da je prvi filter prilijepljen na gornji lijevi dio slike (ljubičasta boja). Od mogućih vrijednosti 1, 3, 2 i 9 vrijednost 9 se nalazi na ljubičastom djelu izlazne slike, jer je najveća vrijednost na ljubičastom dijelu ulazne slike 9. Dva glavna razloga zašto se koristi maksimalno sažimanje su: (1) u dosta eksperimenata je otkriveno da dobro funkcionira, (2) ne postoje parametri koje je potrebno naučiti [30]. Ako se radi maksimalno sažimanje na sliku s tri razine dubine, onda se obavlja sažimanje na svakoj dimenziji.

Na sljedećoj slici je prikazano prosječno sažimanje.

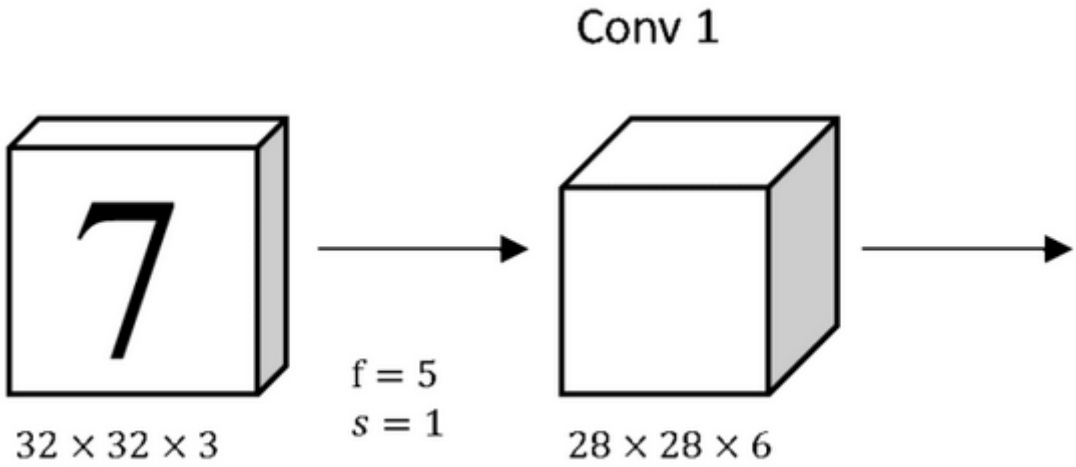


Slika 38 - Primjer prosječnog sažimanja [30]

Formula za kalkuliranje dimenzija značajki nakon sažimanja je $\left[\frac{n - f + 2p}{s}\right] + 1.$

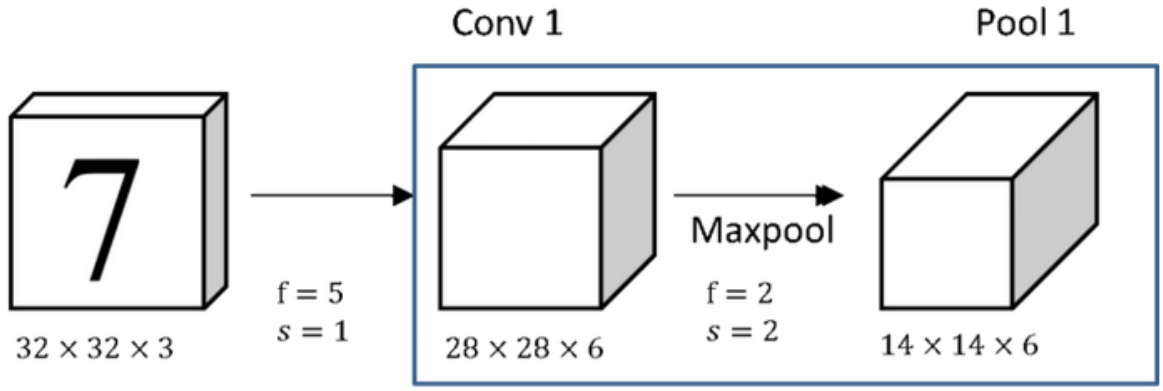
3.7 Primjer konvolucijske neuronske mreže

U ovom primjeru će za prikaz konvolucijske neuronske mreže biti korištena LeNet-5 arhitektura. Slika je dimenzija 32 x 32 x 3, te je na ulaznoj slici jedan od brojeva od 0 do 9. Dakle, radi se o problemu klasifikacije broja od 0 do 9.



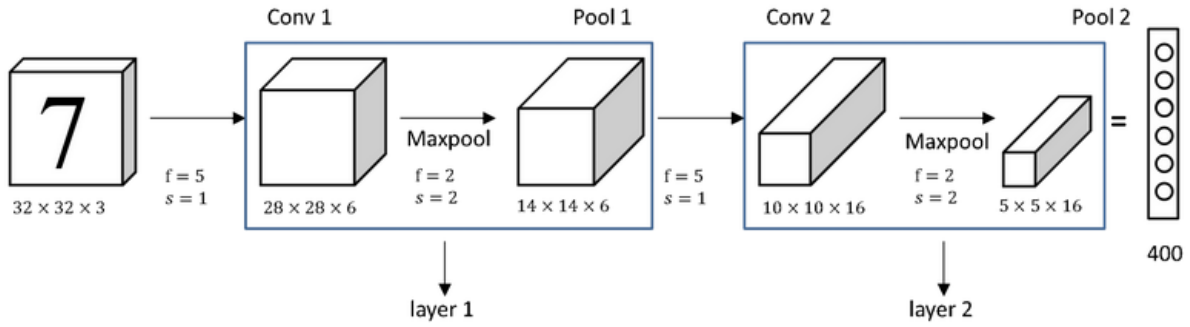
Slika 39 - Prvi sloj LeNet-5 konvolucijske neuronske mreže [31]

Ulazni sloj čini 32 x 32 x 3 slika, za prvi sloj je korišteno 6 5 x 5 filtera s pomakom 1 bez nadopunjavanja. Izlazna slika je dimenzija 28 x 28 x 6. Najprije je vršena konvolucija, zatim je dodan vektor otklona te aplicirana neka nelinearna aktivacijska funkcija, na primjer ReLU. Korišten je maksimalni sloj sažimanja s filterom 2 x 2 i pomakom 2 – rezultat toga je volumen, to jest izlazna slika dimenzija 14 x 14 x 6.



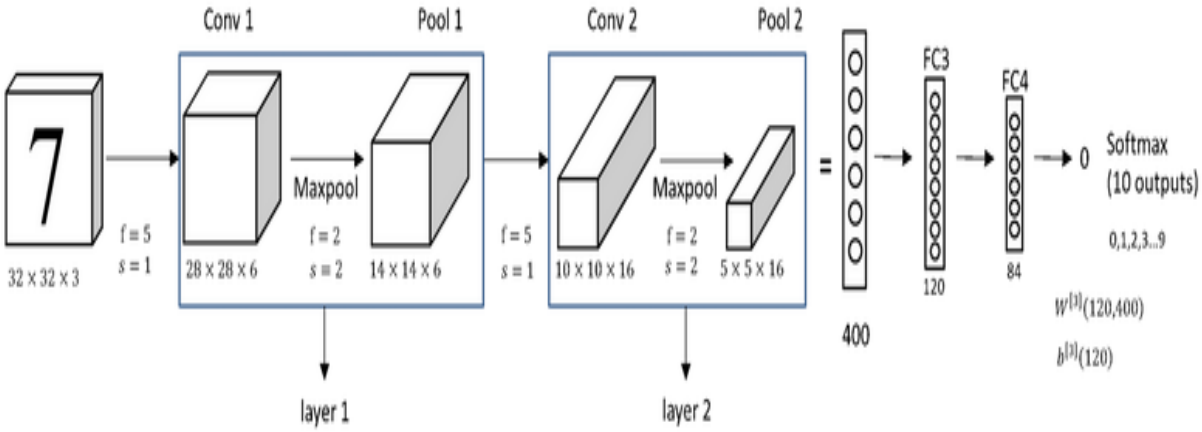
Slika 40 – Sažimanje dodano u prvi sloj LeNet-5 konvolucijske neuronske mreže [31]

Sljedeći korak je korištenje 16 5 x 5 filtera s pomakom 1 bez nadopunjavanja – izlazna slika je onda dimenzija 10 x 10 x 16. Korišteno je maksimalno sažimanje s okvirom 2 i pomakom 2, pa je time dobiven volumen 5 x 5 x 16. Volumen dimenzija 5 x 5 x 16 je pretvoren u vektor 400 x 1.



Slika 41 - Prva dva sloja LeNet-5 konvolucijske neuronske mreže [31]

Nadalje, korišten je prvi potpuno povezani sloj od 120 članova, a zatim potpuno povezani sloj od 84 člana. Na kraju je korišten potpuno povezani sloj s 10 članova, jer toliko ima i znamenaka za klasifikaciju. Korištena je aktivacijska funkcija *softmax*. Aktivacijska funkcija *softmax* se koristi kada je potrebno klasificirati više od dvije klase. Na sljedećoj slici je prikazana potpuna LeNet5 konvolucijska neuronska mreža.

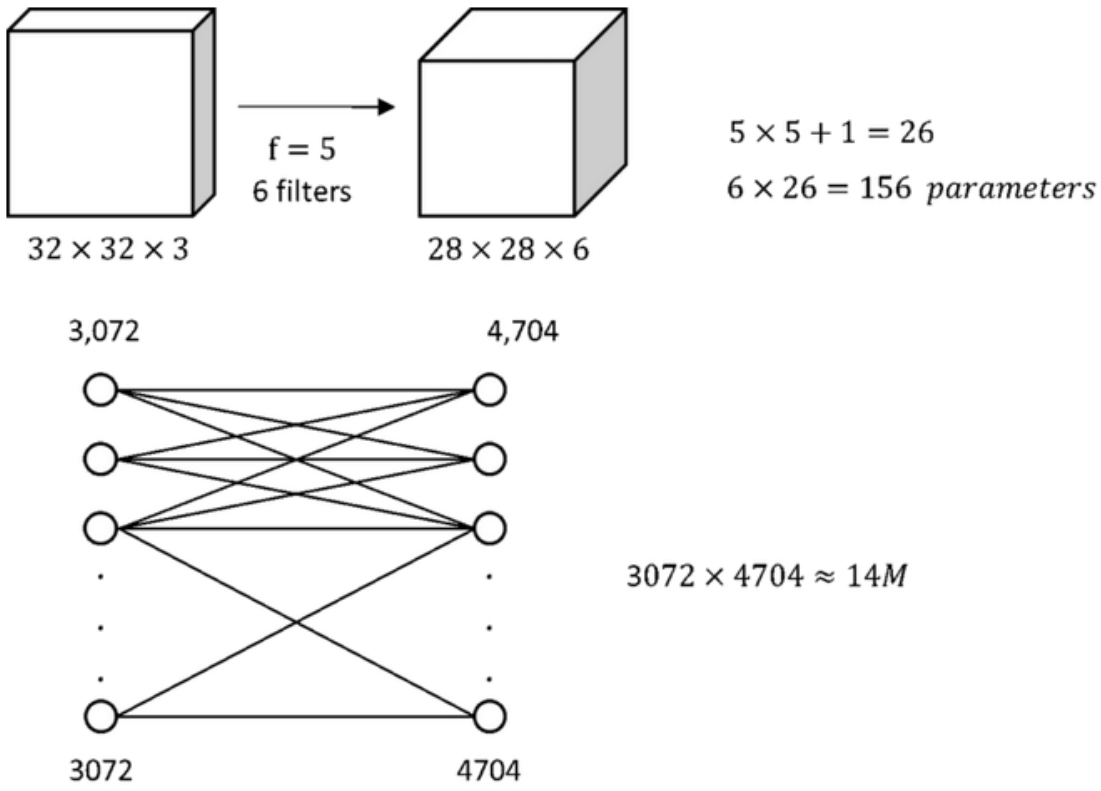


Slika 42 - Potpuna LeNet-5 konvolucijska neuronska mreža [31]

3.8 Zašto konvolucije?

Postoje dvije glavne prednosti konvolucijskih nad potpuno povezanim slojevima: (1) **dijeljenje parametara** i (2) **oskudnost veza** [32]**.**

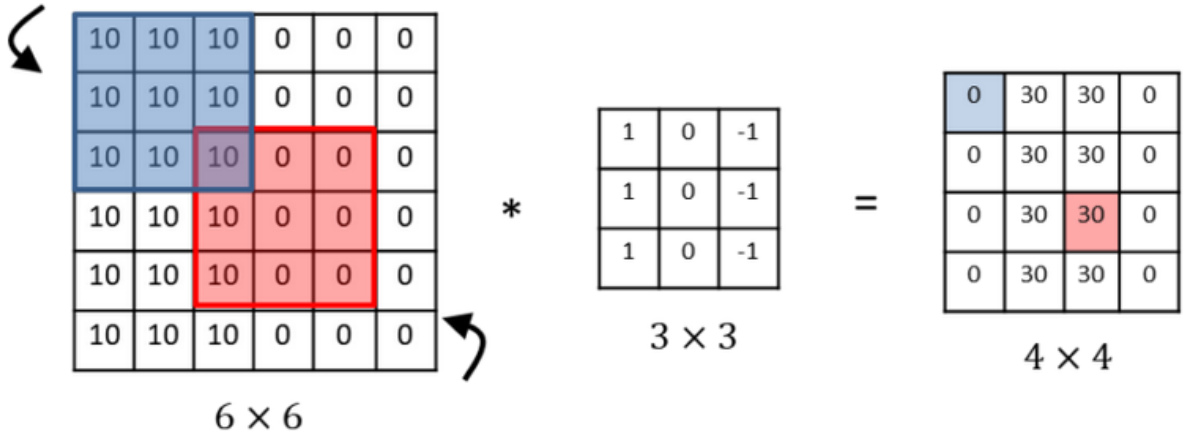
Ako se uzme u obzir ulazna slika s dimenzijama 32 x 32 x 3, slika se konvoluira s 6 5 x 5 filtera, onda je izlazna slika dimenzija 28 x 28 x 6.



Slika 43 - Usporedba potpuno povezanih i konvolucijskih slojeva [32]

Rezultat produkta 32 x 32 x 3 je 3072. Rezultat produkta 28 x 28 x 6 je 4704. Mreža s 3072 članova u jednom sloju i 4704 članova u drugom sloju, ako se radi o potpuno povezanim slojevima, onda bi matrica težina bila dimenzija 3072 x 4704, što je onda približno 14.000.000 parametara za naučiti. Uzimajući obzir da se radi o maloj ulaznoj slici, to je onda dosta parametara za naučiti. Ako se uzme u obzir broj parametara u konvolucijskoj neuronskoj mreži, svaki filter je 5 x 5, dakle 25 parametara s vektorom otklona čini 26 parametara. 26 parametara u 6 filtera podrazumijeva ukupni broj od 156 parametara. Prema navedenim primjerima može se zaključiti da je broj parametara u konvolucijskom sloju, u usporedbi s brojem parametara u potpuno povezanim slojem, znatno manji.

Detektor značajki, na primjer detektor značajki vertikalnih rubova, je koristan u svakom djelu slike. To je primjer dijeljenja parametara.



Slika 44 - Oskudnost veza u konvolucijskim neuronskim mrežama [32]

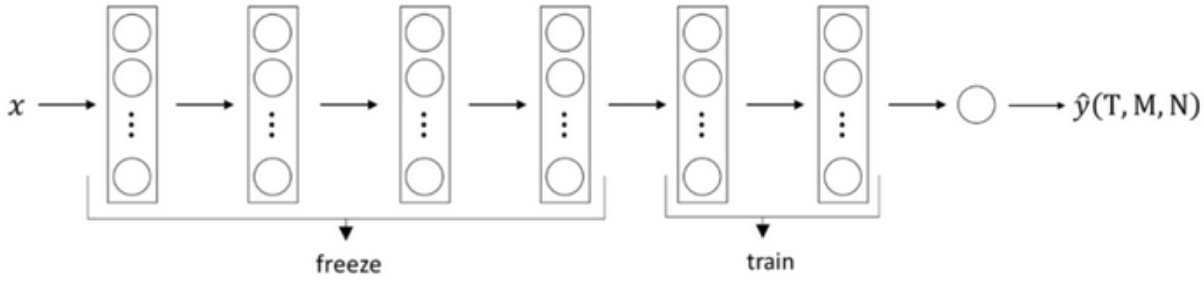
Na gornji lijevi (plavi) dio slike je preslikan konvolucijski filter 3 x 3. Izlazna vrijednost 0 obojena plavom bojom je povezana samo na plavi dio, dakle 9 od 36 ulaznih značajki. Ostale značajke nemaju nikakav utjecaj na izlaznu vrijednost 0. Na primjer vrijednost 30 izlazne značajki obojena crvenom bojom nije povezana s vrijednosti 0 obojenom plavom bojom. To je primjer oskudnosti veza.

Dodatna prednost konvolucije je **translacijska invarijantnost** [32]. Ilustrirano primjerom, ako se uzme za primjer klasifikacija automobila, translacijska invarijantnost govori da automobil pomaknut par piksela u određenom smjeru je i dalje automobil.

3.9 Prijenosno učenje

Jedna od tehnika koja uvelike olakšava stvaranje aplikacija u području računalnog vida je **prijenosno učenje**. *Imagenet* je poznati podatkovni skup s 1.281.167 slika za trening, 50.000 slika za validaciju i 100.000 testnih slika. Dosta arhitektura je trenirano na *Imagenet* podatkovnom skupu pa da se ne gubi vrijeme na treniranje, onda je moguće preuzeti naučene parametre. Dakle, prijenosno učenje omogućuje da se iskoristi znanje s nekih drugih arhitektura na osobnu arhitekturu, a to se može napraviti zbog sličnosti značajki koje se nalaze na slikama. Prijenosno učenje se koristi tako da se ukloni završni sloj koji se koristi za određivanje klase,

jer se obično radi o drukčijem broju klasa koje je potrebno odrediti. Ostali prijašnji slojevi koji su već naučili određene značajke se zamrznu te se trenira onaj dio modela koji se nalazi nakon zamrznutih značajki.



Slika 45 - Prijenosno učenje [33]

4. KORIŠTENE ARHITEKTURE

4.1 InceptionV3

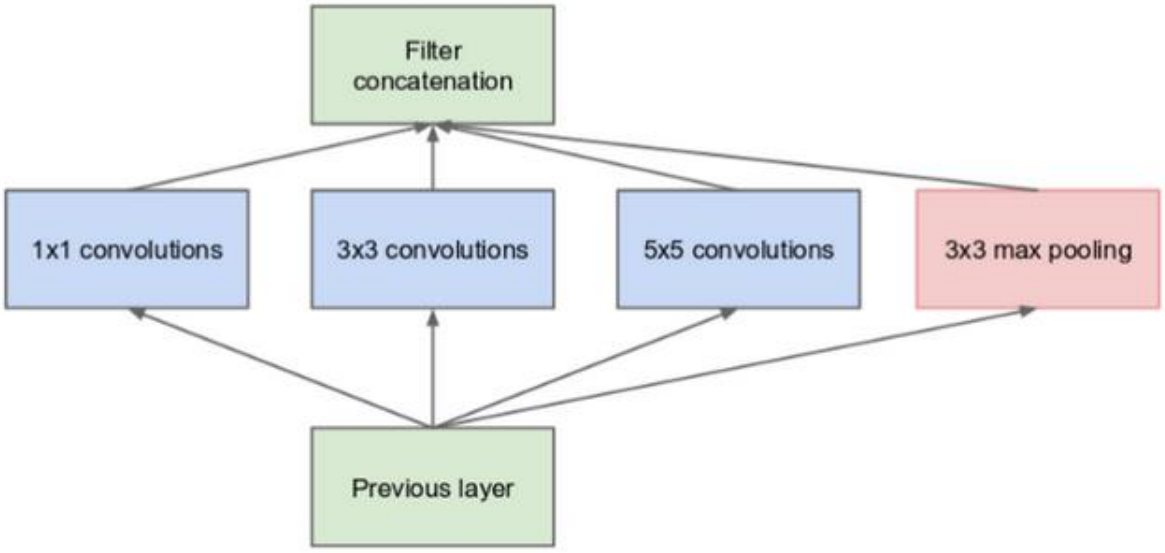
Inception arhitektura je prekretnica po pitanju toga kako su se arhitekture konvolucijskih neuronskih mreža konstruirale – većina arhitektura konvolucijskih neuronskih mreža se sastojala od duboke strukture konvolucijskih slojeva. Inception arhitektura je napravila promjenu tako da su se arhitekture neuronskih mreža počele konstruirati u širinu, a ne samo u dubinu.

Glavni dijelovi slike imaju visoku varijaciju veličine kroz različite slike. Na primjer, na donjim slikama se vidi kako veličina psa varira ovisno o slici.

Slika 46 - Prikaz varijacije glavnog dijela na slikama [34]

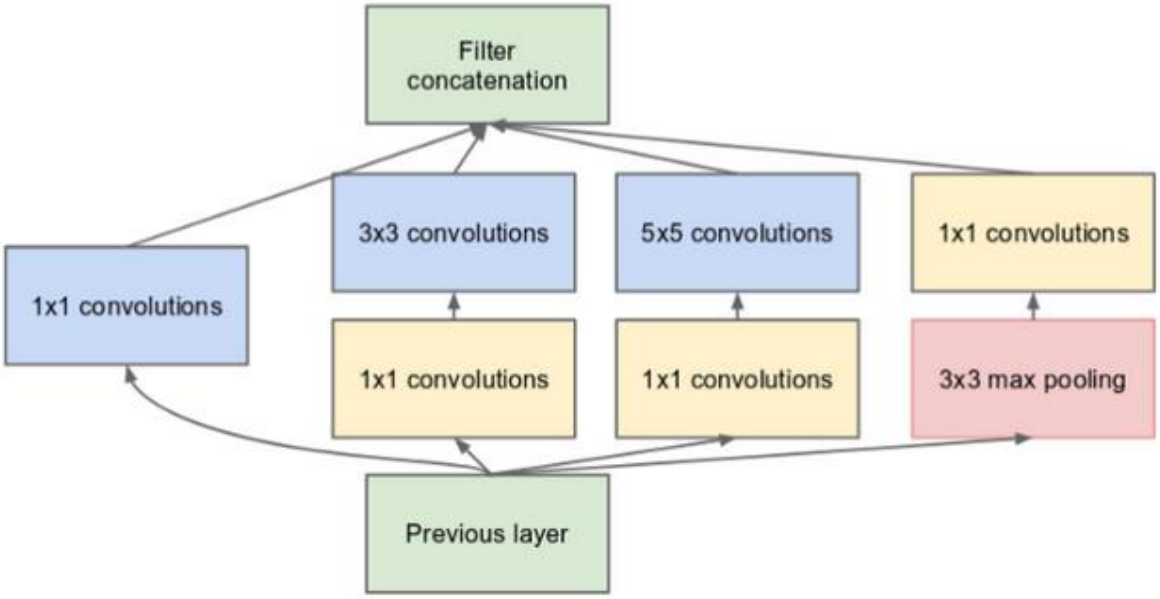
Zbog visoke varijacije u lokaciji informacije, odabiranje ispravnog filtera za konvolucijsku operaciju nije lak posao. Veći filter je dobar za informaciju koja je distribuirana globalno, a manji filter za informaciju koja je distribuirana lokalno [34]. Solucija koja se koristila u verziji InceptionV1 jest da se koriste filteri različite veličine na istoj razini.

Slika ispod je "naivni" Inception modul. Izvode se konvolucije na ulazu, s 3 različite veličine filtera (1 x 1, 3 x 3, 5 x 5), a uz to se izvodi i maksimalno sažimanje. Izlazi se spajaju i šalju sljedećem početnom modulu.



Slika 47 - "Naivni" Inception modul [34]

Kao što je rečeno, duboke neuronske mreže zahtijevaju dosta memorije – kako bi se smanjili zahtjevi na memoriju, korištene su dodatne 1 x 1 konvolucije prije 3 x 3 i 5 x 5 konvolucija.



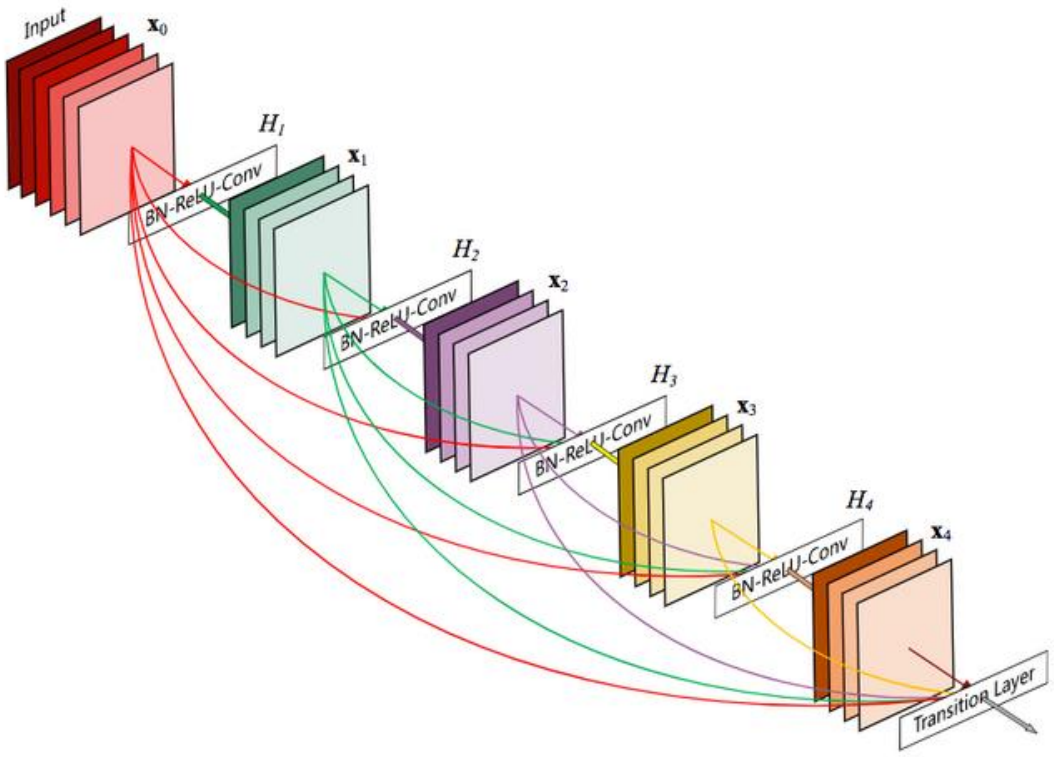
Slika 48 - Inception modul s 1x1 konvolucijama [34]

Kao i većina dubokih arhitektura, InceptionV1 arhitektura je imala problema s nestajanjem gradijenata. Da bi se riješio problem nestajanja gradijenata, u sredini arhitekture su uvedena dva dodatna klasifikatora. Ukupni gubitak je težinska suma dodatnih klasifikatora i stvarnog gubitka. U InceptionV2 verziji, da bi se dobilo na brzini, a i smanjilo na gubitku informacije, 5 x 5 konvolucije su faktorizirane s dvije 3 x 3 konvolucije.

S vremenom se uvidjelo da dodatni klasifikatori ne doprinose sve dok se ne približi kraju treniranja modela. Autori su argumentirali da dodatni klasifikatori funkcioniraju kao regularizatori [34]. Da bi se riješio navedeni problem, uveo se InceptionV3. I dalje su 5 x 5 konvolucije faktorizirane na 3 x 3 konvolucije, kao optimizator je korišten RMSProp, faktorizirane su 7 x 7 konvolucije, a Batch normalizacija je korištena na dodatnim klasifikatorima. Kako bi se riješio problem prenaučenosti modela, korištena je metoda regularizacije pri računanju gubitka.

4.2 DenseNet121

Arhitektura DenseNet se prvi put pojavila u radu „Densely Connected Convolutional Networks“ od strane „Facebook AI Research“ odjela. U navedenom radu se koristila arhitektura koja se sastojala od 121 sloj, pa je zato ime arhitekture DenseNet121. Važni dio arhitekture, onaj po čemu je DenseNet arhitektura prepoznatljiva, su DenseNet blokovi. Prednost DenseNet arhitekture je u tome da je svaki sloj povezan sa sljedećim slojevima - to omogućava onda da svaki sloj prima značajke prethodnih slojeva, pa se tako pokušava riješiti problem nestajanja gradijenata.

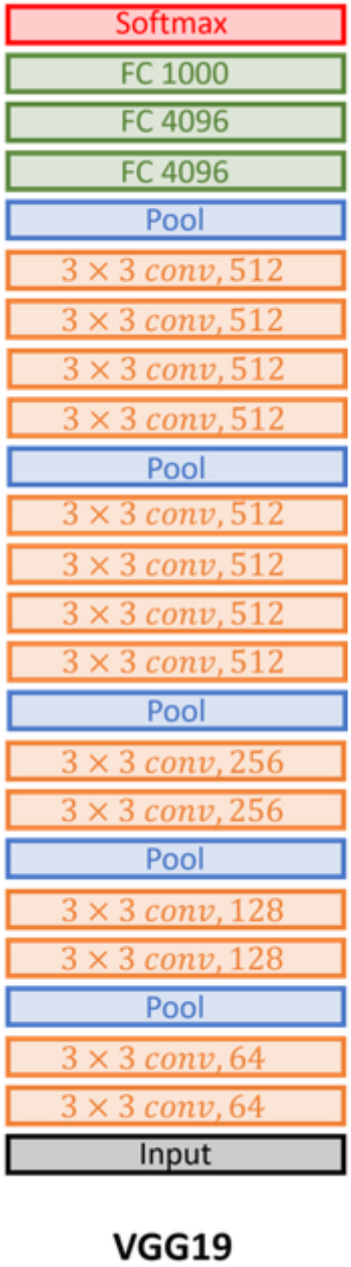


Slika 49 - Primjer DenseNet bloka [35]

Dense blok se sastoji od Batch normalizacije, ReLU aktivacijske funkcije i 3 x 3 konvolucije.

4.3 VGG19

VGG19 nosi ime po tome što se arhitektura VGG19 sastoji od 19 konvolucijskih slojeva. Na slici ispod je prikazana kompletna VGG19 arhitektura.



Slika 50 - VGG 19 arhitektura [36]

Dimenzije prije *conv* označavaju dimenzije konvolucijskog filtera, *conv* označava kovolucijski sloj, a broj iza *conv* označava broj filtera. *Pool* označava sloj sažimanja, *FC* označava potpuno povezani sloj, dok broj iza *FC* označava broj neurona potpuno povezanog sloja.

Vidljivo je da arhitektura VGG19 nije po ničemu specifična, osim što se sastoji od velikoj broja konvolucijskih slojeva.

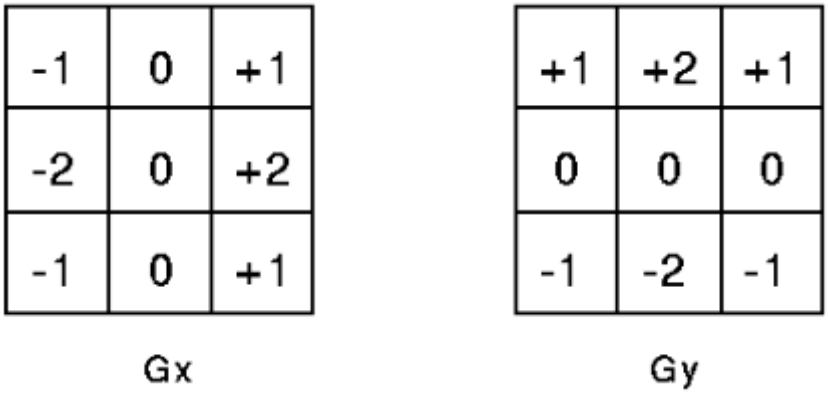
4.4 MobileNet

MobileNet je konvolucijska neuronska mreža namijenjena za mobilne aplikacije. Cilj stvaranja MobileNet arhitekture jest da se stvori konvolucijska neuronska mreža s manjim brojem parametara. Da bi se smanjio broj parametara koriste se **dubinski odvojive konvolucije** (engl. Depthwise separable convolutions). Dubinske odvojive konvolucije se sastoje od dvije operacije: (1) dubinske konvolucije (engl. Depthwise convolution) i (2) smjerne konvolucije (engl. Pointwise convolution).

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

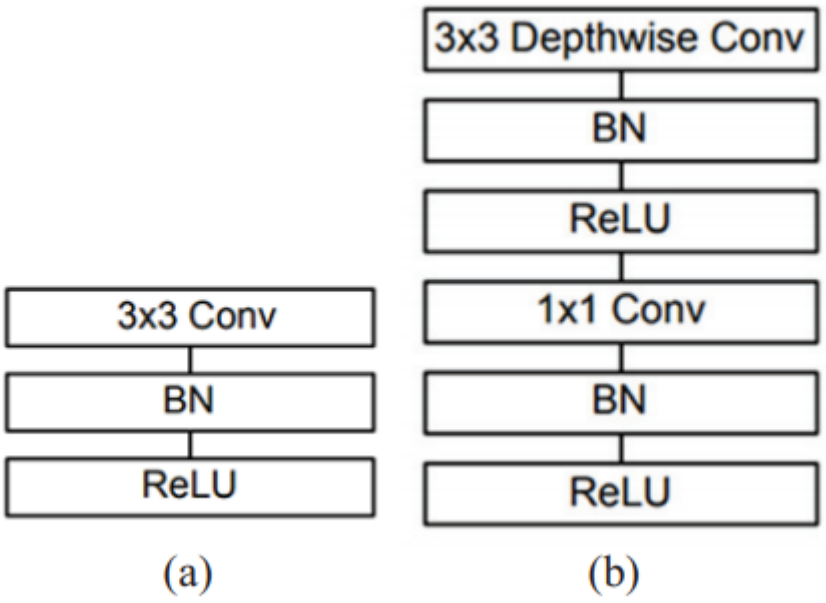
Slika 51 - MobileNet arhitektura [37]

Dubinski odvojiva konvolucija je potekla od ideje da se dubina filtera i prostorna dimenzija mogu odvojiti. Za objasniti dubinsku odvojivu konvoluciju, uzet je za primjer Sobelov filter.



Slika 52 - Sobelov filter [37]

Gx filter se može dobiti kao matrični produkt [1 2 1] transponirano i [-1 0 1]. U tom slučaju umjesto 9 postoji 6 parametara. Ista ideja aplicirana na *Gy* podrazumijeva dubinski odvojivu konvoluciju. Smjerna konvolucija podrazumijeva konvoluiranje s 1 x 1 filterom kako bi se promijenila dimenzija.



Slika 53 - (a) Tradicionalna arhitektura konvolucijske neuronske mreže b) MobileNet arhitektura [37]

4.5 Osobna arhitektura

Pri stvaranju osobne arhitekture, to jest osobnog modela, korišteno je teoretsko znanje opisano u prijašnjim dijelovima rada. Više o funkcionalnostima određenih slojeva se može doznati u dijelu gdje je objašnjen programski kod osobnog modela. Na sljedećoj slici je prikazana arhitektura osobnog modela.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 180, 180, 16)      160
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 90, 90, 16)        0
_________________________________________________________________
batch_normalization (BatchNo (None, 90, 90, 16)        64
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 90, 90, 32)        4640
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 45, 45, 32)        0
_________________________________________________________________
batch_normalization_1 (Batch (None, 45, 45, 32)        128
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 45, 45, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 22, 22, 64)        0
_________________________________________________________________
batch_normalization_2 (Batch (None, 22, 22, 64)        256
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 22, 22, 64)        36928
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 11, 11, 64)        0
_________________________________________________________________
batch_normalization_3 (Batch (None, 11, 11, 64)        256
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 11, 11, 128)       73856
_________________________________________________________________
batch_normalization_4 (Batch (None, 11, 11, 128)       512
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 11, 11, 128)       147584
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 5, 5, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 3200)              0
_________________________________________________________________
dense (Dense)                (None, 512)               1638912
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 513
=================================================================
Total params: 1,922,305
Trainable params: 1,921,697
Non-trainable params: 608
```

Slika 54 - Osobna arhitektura

4.6 Evaluacijske metrike

Evaluacijske metrike se koriste za evaluaciju modela. **Točnost** je zadana formulom:

$$Točnost = \frac{Broj\ točnih\ predikcija}{Broj\ netočnih\ predikcija} \tag{30}$$

Za odziv i preciznost bitno je znati broj stvarno pozitivnih, stvarno negativnih, lažno pozitivnih i lažno negativnih.

- **Stvarno pozitivni** – slučaj kada model točno klasificira pozitivnu klasu. Na primjer, na slici magnetske rezonance je prisutna upala pluća, a izlazna vrijednost modela je prisutnost upale pluća.
- **Stvarno negativni** – slučaj kada model točno klasificira negativnu klasu. Na primjer, na slici magnetske rezonance nije prisutna upala pluća, a izlazna vrijednost modela je neprisutnost upale pluća.
- **Lažno pozitivni** – slučaj kada model netočno klasificira pozitivnu klasu. Na primjer, na slici magnetske rezonance nije prisutna upala pluća, a izlazna vrijednost modela je prisutnost upale pluća.
- **Lažno negativni** – slučaj kada model netočno klasificira negativnu klasu. Na primjer, na slici magnetske rezonance je prisutna upala pluća, a izlazna vrijednost modela je neprisutnost upale pluća.

Odziv (eng. Recall) je zadan sljedećom formulom:

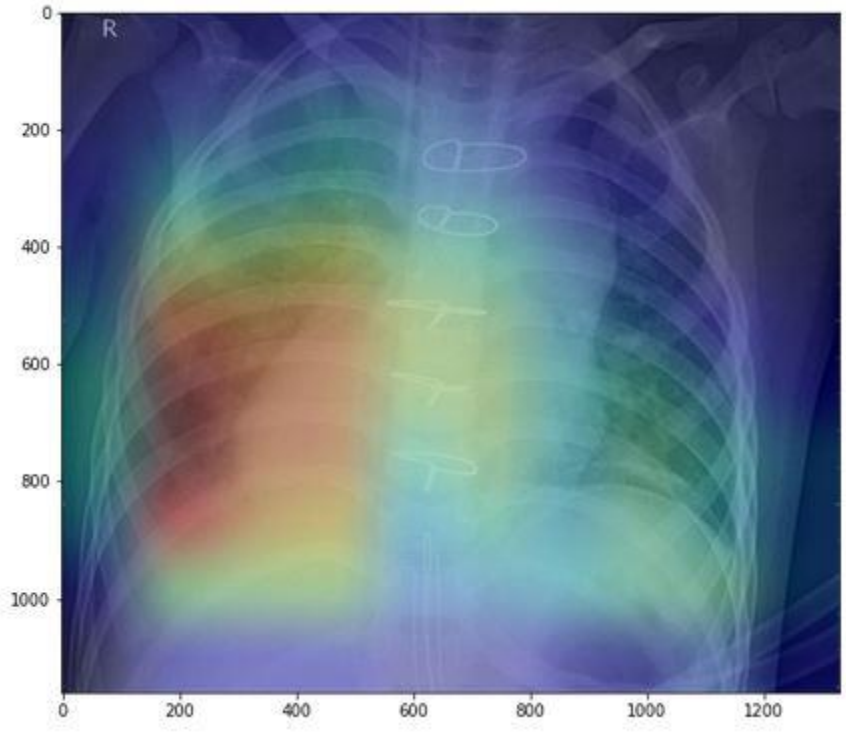
$$Odziv = \frac{Stvarno\ pozitivni}{Stvarno\ pozitivni + Lažno\ negativni} \tag{31}$$

Preciznost (engl. Precision) je zadana sljedećom formulom:

$$Preciznost = \frac{Stvarno\ pozitivni}{Stvarno\ pozitivni + Lažno\ pozitivni} \tag{32}$$

5. GRADCAM

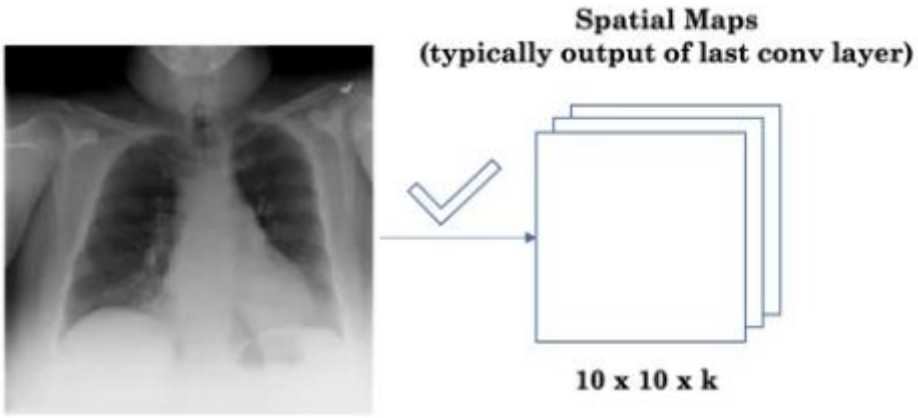
GradCAM je tehnika pomoću koje se može doznati na što konvolucijska neuronska mreža obraća pažnju kada klasificira određenu sliku. Postoji priča za koju se ne zna je li istinita, ali svejedno se često prepričava kada je u pitanju interpretacija neuronskih mreža. Prema priči je američka vojska htjela razviti uređaj za detekciju kamuflažnih tenkova, pa je za razviti model bilo potrebno skupiti slike kamuflažnih i nekamuflažnih tenkova. Potrebne slike su se skupile i postigla se visoka točnost na testnim podacima. Kada se model isprobao u stvarnosti uočeno je da model konstantno griješi. Da bi se riješio problem, prvo se obratilo pažnju na podatkovni skup – uočeno je da je na fotografijama kamuflažnih tenkova vrijeme oblačno, a na fotografijama nekamuflažnih tenkova vrijeme sunčano. Umjesto detektora za kamuflažne tenkove napravljen je detektor za oblačno vrijeme. U slučaju navedene priče, prema slikama podatkovnog skupa bilo je moguće otkriti koji je uzrok niske točnosti kada je u pitanju klasifikacija nad stvarnim podacima, ali što ako se na slikama podatkovnog skupa ne može uvidjeti zašto neuronska mreža ima nisku točnost na testnim podacima? Onda je za to dobro koristiti GradCAM tehniku.



Slika 55 - Rezultat korištenja GradCAM tehnike

Kada konvolucijska neuronska mreža obrađuje sliku, podaci slike su proslijeđeni kroz razne slojeve. Kao što je rečeno, generalno se zna da početni slojevi detektiraju značajke niske

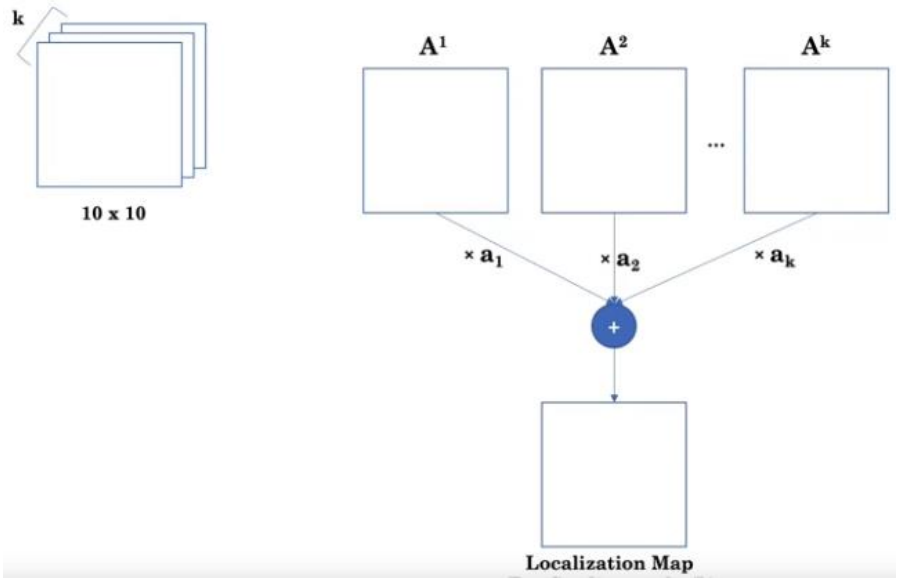
kompleksnosti, dok krajnji slojevi detektiraju kompleksnije značajke. Kraj konvolucijske neuronske mreže su potpuno povezani slojevi, a prije nego se informacija iz konvolucijskih slojeva proslijedi potpuno povezanim slojevima, potrebno je „spljoštiti“ podatke – pretvoriti podatke u jednodimenzionalni stupac, to jest, pretvoriti podatke u izravnan sloj (engl. Flatten). Jednom kada su značajke pretvorene u izravnan sloj, prostorna informacija o podacima je izgubljena – dakle, ako se hoće vizualizirati određene značajke na koje model obraća pažnju kada je u pitanju klasifikacija određene klase, onda se moraju uzeti značajke prije nego se podaci pretvore u izravnan sloj. Najkompleksnije značajke koje je konvolucijska neuronska mreža naučila detektirati se nalaze u zadnjem konvolucijskom sloju. U zadnjem konvolucijskom sloju se nalazi prostorna mapa značajki (engl. Spatial maps).



Slika 56 - Prostorna mapa značajki [38]

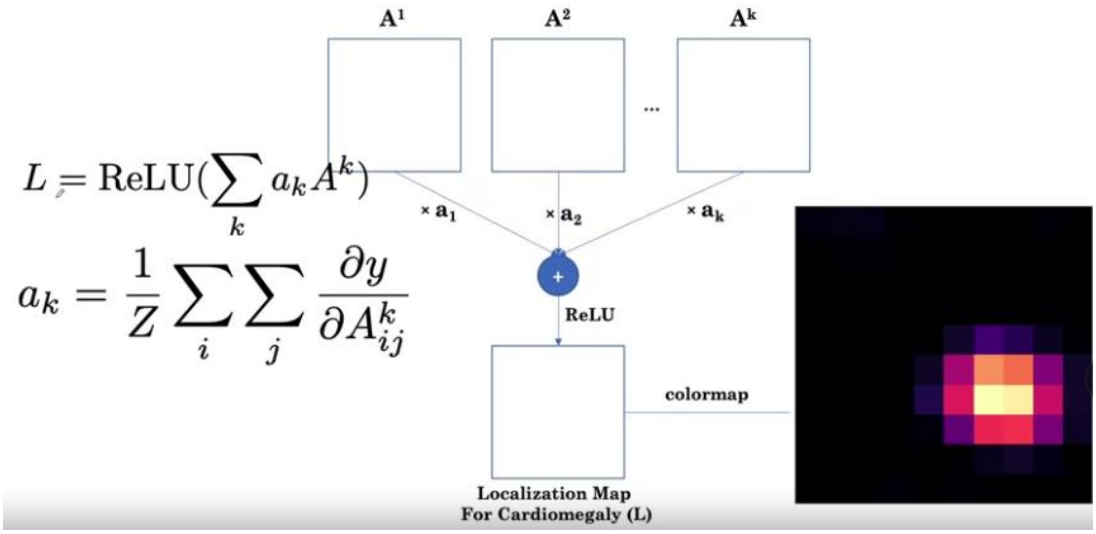
Na slici iznad broj *k* označava broj prostornih značajki, dok prva dva broja (10) označavaju širinu i visinu prostorne mape. Lokalizacijska mapa se dobije kao suma reda umnoška utjecaja određene prostorne mape i prostorne mape. Matematički to je izraženo sljedećom jednadžbom:

$$L = \sum_k a_k A^k \tag{33}$$



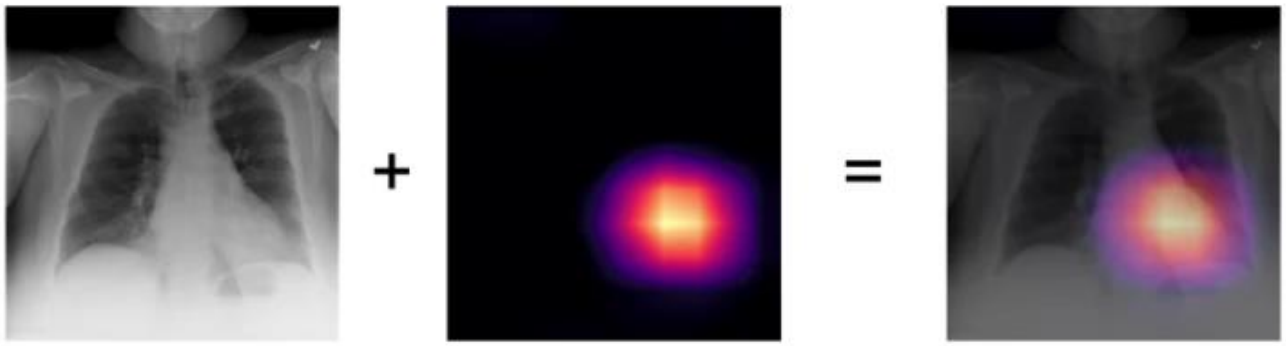
Slika 57 - Lokalizacijska mapa [38]

U gornjoj slici a_k označava utjecaj određene prostorne mape na krajnju klasu. Računajući parcijalnu derivaciju krajnje klase prema prostornoj značajki dobije se utjecaj značajke na krajnju klasu. Težina prostorne mape se dobije kao srednja suma prosjeka parcijalnih derivacija svih značajki prema određenoj klasi. Budući da je u interesu doznati samo utjecaj pozitivnih značajki, aplicira se ReLU aktivacijska funkcija. Dobivena lokalizacijska mapa se može iskoristiti za stvaranje toplinske mape – dobiveni brojevi se pretvaraju u boje.



Slika 58 - Pretvaranje lokalizacijske mape u toplinsku mapu [39]

Toplinska mapa je obično male dimenzije zbog slojeva konvolucija i sažimanja pa je potrebno interpolirati mapu u sliku izvorne veličine. Na kraju, pomoću izvorne slike i toplinske mape nastaje izlazna slika.



Slika 59 - Izlazna slika kao suma izvorne slike i toplinske mape [39]

6. PODATKOVNI SKUP I IMPLEMENTACIJA PROGRAMSKOG KODA

Podatkovni skup koji je korišten za treniranje modela preuzet je s web sjedišta *Kaggle.com*. U preuzetom podatkovnom skupu je 5856 podataka – iz tog razloga trening, validacijski i testni skup podataka su raspoređeni na 70% podataka za trening, 15% podataka za validaciju i 15% za testiranje. Kako bi se radilo s jednim od *Tensoflow* modula, *ImageDataGeneratorom*, onda su podaci raspoređeni u tri mape: *train*, *test* i *val*. Svaka od mapa sadrži dvije podmape: *Pneumonia* i *Normal*. U mapi *Pneumonia* smještene su slike s klasom upale pluća, dok su u mapi *Normal* smještene slike bez klase upale pluća.

Program se izvodio u programskom okruženju *Google Colaboratory*. Na slici ispod prikazan je programski kod koji se koristio za učitavanje programskih biblioteka. Prema slici je vidljivo učitavanje programskog okvira *Tensorflow*, kao i modula koji *Tensorflow-u* pripadaju.

```
import numpy as np
import tensorflow as tf
import os
from glob import glob
import matplotlib.pyplot as plt
from tensorflow import keras
from keras.preprocessing import image
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import RMSprop,Adam
import tensorflow as tf
from keras.layers import Conv2D, MaxPooling2D, Dense, Dropout, Input, Flatten, SeparableConv2D, BatchNormalization
from tensorflow.keras.callbacks import TensorBoard, EarlyStopping, LearningRateScheduler, ModelCheckpoint, CSVLogger, ReduceLROnPlateau

print(tf.__version__)
```

```
2.4.1
```

Slika 60 - Učitavanje programskih biblioteka

Budući da je programski kod izvođen u okruženju *Google Colaboratory*, onda je bilo potrebno smjestiti odgovarajuće slike koje čine podatkovni skup na *Google disk*. Na sljedećoj slici je prikazan programski kod koji omogućuje učitavanje slika s *Google diska*.

```
from google.colab import drive
```

+ Code + Text

```
[ ] from google.colab import drive
    drive.mount('/content/gdrive')
```

Slika 61 - Učitavanje i korištenje Google diska

Za ispisivanje broja slika u odgovarajućoj mapi korišten je sljedeći programski kod.

```
train_dir = os.path.join(BASE_DIR,'train/')
val_dir = os.path.join(BASE_DIR,'val/')
test_dir = os.path.join(BASE_DIR,'test/')

print('Number of images in training set = ',str(len(glob(train_dir+'*/*'))))
print('Number of images in validation set = ',str(len(glob(val_dir+'*/*'))))
print('Number of images in testing set = ',str(len(glob(test_dir+'*/*'))))
```

```
Number of images in training set =  4335
Number of images in validation set =  782
Number of images in testing set =  782
```

Slika 62 - Ispis broja slika u odgovarajućim folderima

6.1 Osobni model

Na sljedećoj slici je prikazan programski kod koji je korišten za stvaranje vlastitog modela.

```
model=tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(16, (3,3), activation='relu',padding='same', input_shape=(180, 180, 1)),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Conv2D(32, (3,3), activation='relu',padding='same'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Conv2D(64, (3,3), activation='relu',padding='same'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Conv2D(64, (3,3), activation='relu',padding='same'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Conv2D(128, (3,3), activation='relu',padding='same'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Conv2D(128, (3,3), activation='relu',padding='same'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

Korištenjem *Sequential* modula stvoren je sekvencijalni model. Korišten je *Conv2D* sloj, to jest konvolucija *Conv2D* je funkcija kojoj prvi parametar u listi argumenata predstavlja broj filtera, drugi parametar veličinu okvira, pod *activation* je definirana aktivacijska funkcija, a *padding = 'same'* znači da će slika biti iste veličine kao što je bila prije konvoluiranja. U prvoj konvoluciji pod *input_shape* je definirana veličina slike, u ovom slučaju to je veličina širine i visine koja iznosi 180, a broj dimenzije je 1, jer su slike crno bijele. Također je korištena funkcija *MaxPooling2D* koja sažima sliku. U ovom slučaju je za smanjivanje slike korišten okvir dimenzija 2 x 2. Osim navedenih funkcija korištena je i funkcija *Batchnormalization*. Da bi se *Dense* funkcija mogla koristiti, korištena je funkcija *Flatten* kako bi se vrijednosti slike pretvorile u jednodimenzionalno polje. Prvi parametar funkcije *Dense* je broj neurona koji sloj sadrži, dok je pod *activation* definirana odgovarajuća aktivacijska funkcija. Na kraju je korišten jedan neuron jer je potrebna jedna vrijednost koja pokazuje kolika je vjerojatnost da je na slici prisutna upala pluća.

Da bi se model mogao trenirati potrebno ga je kompajlirati, odrediti optimizator s određenom stopom za učenje, odrediti funkciju gubitka te specificirati određene metrike. Na slici ispod, za optimizator je korišten *RMSprop* sa stopom za učenje 0.001, za funkciju gubitka je korištena

binary_crossentropy funkcija, dok su za evaluaciju korištene metrike *točnost* (engl. Accuracy), *preciznost* (engl. Precision), *odziv* (engl. Recall), *točno pozitivan* (engl. True positive), *točno negativan* (engl. True negative), *netočno negativan* (engl. False negative) i *netočno pozitivan* (engl. False positive).

```
model.compile(optimizer=RMSprop(lr=0.001),
              loss='binary_crossentropy',
              metrics=['accuracy','Precision','Recall', tf.keras.metrics.TruePositives(), tf.keras.metrics.TrueNegatives(),
                       tf.keras.metrics.FalseNegatives(), tf.keras.metrics.FalsePositives()])
```

Slika 63 - Kompajliranje modela

Preuređivanja slika koja su vršena su vidljiva na slici ispod. Slika je normalizirana tako da sadrži vrijednosti između nula i jedan, također su korištene i augmentacijske tehnike, kao na primjer rotacijska promjena za deset stupnjeva, promjena veličine i širine za deset posto te približavanje za deset posto.

```
train_datagen=ImageDataGenerator(rescale=1.0/255,
                                 rotation_range=10,
                                 width_shift_range=0.1,
                                 height_shift_range=0.1,
                                 zoom_range=0.1,
                                 )
```

Slika 64 - Data augmentacijske tehnike za uređivanje slike

Za učitavanje slika su korišteni generatori koji učitavaju slike iz mapa. Također je specificirana veličina slike, kao i *batch size*. Korištenjem *batch size-a* omogućeno je treniranje na više slika odjednom, u ovom slučaju na 128 slika.

```
val_datagen=ImageDataGenerator(rescale=1.0/255)

test_datagen=ImageDataGenerator(rescale=1.0/255)

train_generator=train_datagen.flow_from_directory(train_dir, color_mode="grayscale",target_size=(180,180),batch_size=128,class_mode='binary')

val_generator=val_datagen.flow_from_directory(val_dir,color_mode="grayscale",target_size=(180,180),batch_size=128,class_mode='binary')

test_generator=test_datagen.flow_from_directory(test_dir,color_mode="grayscale", target_size=(180,180),batch_size=128,class_mode='binary')
```

Slika 65 - Korištenje generatora za treniranje

Za spremanje modela korištena je funkcija *ModelCheckpoint*. Model je spremljen prema najboljem iznosu validacijskog gubitka.

```
model_checkpoint = ModelCheckpoint('osobni_model.h5', monitor='val_loss', save_best_only=True, verbose=1)
```

Slika 66 - Korištenje ModelCheckpoint funkcije

Treniranje modela je obavljeno pokretanjem metode *fit*. Prvi argument metode *fit* je generator koji je učitao trening podatke, drugi argument je validacijski generator koji je učitao

validacijske podatke, treći argument je broj epoha, *verbose=2* znači da se ne ispisuju dijelovi epoha pri učenju te se koristi *CSVLogger* za spremanje vrijednosti pri učenju.

```
history=model.fit(train_generator,validation_data=val_generator,epochs=50,
                  verbose=2, callbacks=[CSVLogger("OsobniModel.csv"), model_checkpoint])
```

Slika 67 - Treniranje modela

Sljedećim kodom se crtaju određeni grafovi kojima *x* os predstavlja broj epoha, a *y* os određenu metriku koja je definirana u *compile* metodi.

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

train_precision=history.history['precision']
val_precision=history.history['val_precision']

train_recall=history.history['recall']
val_recall=history.history['val_recall']

loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(acc))

plt.plot(epochs, acc, 'r', label='Training accuracy')
plt.plot(epochs, val_acc, 'b', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()

plt.plot(epochs, train_precision, 'r', label='Training precision')
plt.plot(epochs, val_precision, 'b', label='Validation precision')
plt.title('Training and validation precision')
plt.legend()
plt.figure()

plt.plot(epochs, train_recall, 'r', label='Training recall')
plt.plot(epochs, val_recall, 'b', label='Validation recall')
plt.title('Training and validation recall')
plt.legend()
plt.figure()

plt.plot(epochs, loss, 'r', label='Training Loss')
plt.plot(epochs, val_loss, 'b', label='Validation Loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```

Slika 68 - Programski kod za stvaranje odgovarajućih grafova

Programski kod za evaluaciju modela je prikazan na slici ispod.

```
evaluation = model.evaluate(test_generator)
```

Slika 69 - Evaluacija modela

6.2 InceptionV3

U sljedećem programskom dijelu je objašnjeno korištenje modela InceptionV3. Najprije su učitane programske biblioteke za korištenje modela InceptionV3.

```
from tensorflow.keras.applications.inception_v3 import InceptionV3
```

Slika 70 - Učitavanje programskog koda za InceptionV3 model

Korištene su težine s podatkovnog skupa *Imagenet*, isključeno je uključivanje zadnjeg dijela modela, dok su pod *input_shape* specificirane dimenzije slike. Također su zaključeni slojevi za treniranje.

```
pretrained_model=InceptionV3(input_shape=(180,180,3),
                             include_top=False,
                             weights='imagenet')
for layers in pretrained_model.layers:
    layers.trainable=False
```

Slika 71 - Postavljanje InceptionV3 modela

Preuzet je izlaz sloja *mixed10*.

```
last_layer=pretrained_model.get_layer('mixed10')
last_output = last_layer.output
```

Slika 72 - Preuzimanje sloja mixed10 za InceptionV3 model

Na sloj *mixed10* nadopunjen je model slojevima koji su prikazani na slici ispod. Korištena je i funkcija *Dropout*. Rezultat korištenja funkcije *Dropout* je nestajanje određenih neurona što je korisno jer se nestajanjem određenih neurona osigurava regularizacija.

```
x=tf.keras.layers.Flatten()(last_output)
x=tf.keras.layers.Dense(1024,activation='relu')(x)
x=tf.keras.layers.Dropout(0.2)(x)
x=tf.keras.layers.Dense(512,activation='relu')(x)
x=tf.keras.layers.Dropout(0.2)(x)
x=tf.keras.layers.Dense(1,activation='sigmoid')(x)

model=tf.keras.Model(pretrained_model.input,x)

model.compile(optimizer=RMSprop(lr=0.001),
              loss='binary_crossentropy',
              metrics=['accuracy','Precision','Recall', tf.keras.metrics.TruePositives(),
                       tf.keras.metrics.TrueNegatives(), tf.keras.metrics.FalseNegatives(), tf.keras.metrics.FalsePositives()])
```

Slika 73 - Dorađivanje InceptionV3 modela

6.3 DenseNet121

U sljedećem programskom dijelu objašnjeno je korištenje modela DenseNet121. Najprije su učitane programske biblioteke za korištenje modela DenseNet121.

```
from keras.applications import DenseNet121
```

Slika 74 - Učitavanje programskog koda za DenseNet121 model

Kao i prije kod modela InceptionV3, korištene su težine s podatkovnog skupa *Imagenet*, isključeno je uključivanje zadnjeg dijela modela, dok su pod *input_shape* specificirane dimenzije slike. Također su zaključeni slojevi za treniranje.

```
pretrained_model=DenseNet121(input_shape=(180,180,3),
                             include_top=False,
                             weights='imagenet')
for layers in pretrained_model.layers:
    layers.trainable=False
```

Slika 75 - Postavljanje DenseNet121 modela

Preuzet je izlaz sloja *conv5_block16_0_relu* te je stvoren ostatak modela.

```
last_layer=pretrained_model.get_layer('conv5_block16_0_relu')
last_output = last_layer.output

x=tf.keras.layers.Flatten()(last_output)
x=tf.keras.layers.Dense(1024,activation='relu')(x)
x=tf.keras.layers.Dropout(0.3)(x)
x=tf.keras.layers.Dense(256,activation='relu')(x)
x=tf.keras.layers.Dropout(0.3)(x)
x=tf.keras.layers.Dense(1,activation='sigmoid')(x)

model=tf.keras.Model(pretrained_model.input,x)

model.compile(optimizer=RMSprop(lr=0.001),
              loss='binary_crossentropy',
              metrics=['accuracy','Precision','Recall', tf.keras.metrics.TruePositives(),
                       tf.keras.metrics.TrueNegatives(), tf.keras.metrics.FalseNegatives(), tf.keras.metrics.FalsePositives()])
```

Slika 76 - Dorađivanje DenseNet121 modela

6.4 VGG19

U sljedećem programskom dijelu objašnjeno je korištenje VGG19 modela. Najprije su učitane programske biblioteke za korištenje modela VGG19.

```
from keras.applications import VGG19
```

Slika 77 - Učitavanje programskog koda za VGG19 model

Kao i kod prethodne dvije arhitekture, korištene su težine s podatkovnog skupa *Imagenet*, isključeno je uključivanje zadnjeg dijela modela, dok su pod *input_shape* specificirane dimenzije slike. Također su zaključeni slojevi za treniranje.

```
pretrained_model = VGG19(input_shape=(180,180,3),
                                include_top=False,
                                weights='imagenet')

for layers in pretrained_model.layers:
    layers.trainable=False
```

Slika 78 - Postavljanje VGG19 modela

Preuzet je izlaz sloja *block5_conv4* te je stvoren ostatak modela.

```
last_layer = pretrained_model.get_layer('block5_conv4')
last_output = last_layer.output

x = tf.keras.layers.Flatten()(last_output)
x = tf.keras.layers.Dense(1024,activation='relu')(x)
x = tf.keras.layers.Dropout(0.2)(x)
x = tf.keras.layers.Dense(512,activation='relu')(x)
x = tf.keras.layers.Dropout(0.2)(x)
x = tf.keras.layers.Dense(1,activation='sigmoid')(x)

model = tf.keras.Model(pretrained_model.input, x)

model.compile(optimizer=RMSprop(lr=0.001),
              loss='binary_crossentropy',
              metrics=['accuracy','Precision','Recall', tf.keras.metrics.TruePositives(),
                       tf.keras.metrics.TrueNegatives(), tf.keras.metrics.FalseNegatives(), tf.keras.metrics.FalsePositives()])
```

Slika 79 - Dorađivanje VGG19 modela

6.5 MobileNet

U sljedećem programskom dijelu je objašnjeno korištenje modela MobileNet. Najp rije su učitane programske biblioteke za korištenje modela MobileNet.

```
from keras.applications import MobileNet
```

Slika 80 - Učitavanje programskog koda za MobileNet model

Kao i kod prethodne tri arhitekture korištene su težine s podatkovnog skupa *Imagenet*, isključeno je uključivanje zadnjeg dijela modela, dok su pod *input_shape* specificirane dimenzije slike. Također su zaključeni slojevi za treniranje.

```
pretrained_model = MobileNet(input_shape=(180,180,3),
                                include_top=False,
                                weights='imagenet')

for layers in pretrained_model.layers:
    layers.trainable=False
```

Slika 81 - Postavljanje MobileNet modela

Preuzet je izlaz sloja *conv_pw_13_relu* te je stvoren ostatak modela.

```
last_layer = pretrained_model.get_layer('conv_pw_13_relu')
last_output = last_layer.output

x = tf.keras.layers.Flatten()(last_output)
x = tf.keras.layers.Dense(1024,activation='relu')(x)
x = tf.keras.layers.Dropout(0.2)(x)
x = tf.keras.layers.Dense(512,activation='relu')(x)
x = tf.keras.layers.Dropout(0.2)(x)
x = tf.keras.layers.Dense(2,activation='softmax')(x)

model = tf.keras.Model(pretrained_model.input, x)

model.compile(optimizer=RMSprop(lr=0.001),
              loss='binary_crossentropy',
              metrics=['accuracy','Precision','Recall', tf.keras.metrics.TruePositives(),
                       tf.keras.metrics.TrueNegatives(), tf.keras.metrics.FalseNegatives(), tf.keras.metrics.FalsePositives()])
```

Slika 82 - Doradivanje MobileNet modela

6.6 GradCAM

Za korištenje GradCAM tehnike korišten je model Xception i težine s podatkovnog skupa *Imagenet*.

```
model_builder = keras.applications.xception.Xception
model = model_builder(weights="imagenet")
```

Slika 83 - Učitavanje Xception modela

Preuzet je sloj *block14_sepcon2_act* te je izgrađen ostatak modela.

```
last_layer=model.get_layer('block14_sepconv2_act')
last_output = last_layer.output
x = tf.keras.layers.GlobalAveragePooling2D(name="avg_pool")(last_output)
x=tf.keras.layers.Dense(1,activation='sigmoid', name = "predictions")(x)
model=tf.keras.Model(model.input,x)

model.compile(optimizer=RMSprop(lr=0.001),
              loss='binary_crossentropy',
               metrics=['accuracy','Precision','Recall'])
```

Učitana je slika te su definirani slojevi koji se koriste unutar *make_gradcam_heatmap* funkcije.

```
last_conv_layer_name = "block14_sepconv2_act"
classifier_layer_names = [
    "avg_pool",
    "predictions",
]

# The local path to our target image
img_path = keras.utils.get_file(
    "pneumonia_1.jpeg", "https://i.imgur.com/YCGfMk9.jpg"
)
```

Slika 84 - Preuzimanje slike i definiranje slojeva koje se koriste za GradCAM tehniku

Definirana je funkcija *get_img_array* unutar koje je slika procesuirana te joj dodana *batch* dimenzija.

```
def get_img_array(img_path, size):
    # `img` is a PIL image of size 299x299
    img = keras.preprocessing.image.load_img(img_path, target_size=size)
    # `array` is a float32 Numpy array of shape (299, 299, 3)
    array = keras.preprocessing.image.img_to_array(img)
    # We add a dimension to transform our array into a "batch"
    # of size (1, 299, 299, 3)
    array = np.expand_dims(array, axis=0)
    return array
```

Slika 85 - get_img_array funkcija

Unutar funkcije *make_gradcam_heatmap* najprije je stvoren model kojem je ulaz slika, a izlaz zadnji konvolucijski sloj, kao i model kojem je ulaz zadnji konvolucijski sloj, a izlaz povezani dio slojeva koji su specificirani u varijabli *classifier_layer_names*.

Izračunati su gradijenti klasificirane klase za ulaznu sliku s obzirom na aktivacije zadnjeg konvolucijskog sloja. Gradijenti su bitni jer govore koliko određena značajka utječe na klasifikaciju sliku. Gradijentima je oduzet prosjek po svakoj od tri dimenzije.

Kako bi bile naglašene značajke koje imaju veći utjecaj na klasifikaciju sliku, dobiveni gradijenti su korišteni tako da je izlazni dio zadnjeg konvolucijskog sloja pomnožen s gradijentima. Stvorena je toplinska mapa kao prosjek vrijednosti po kanalima. Toplinska mapa je normalizirana tako da vrijednosti toplinske mape budu između 0 i 1.

```python
def make_gradcam_heatmap(
    img_array, model, last_conv_layer_name, classifier_layer_names
):
    # First, we create a model that maps the input image to the activations
    # of the last conv layer
    last_conv_layer = model.get_layer(last_conv_layer_name)
    last_conv_layer_model = keras.Model(model.inputs, last_conv_layer.output)

    # Second, we create a model that maps the activations of the last conv
    # layer to the final class predictions
    classifier_input = keras.Input(shape=last_conv_layer.output.shape[1:])
    x = classifier_input
    for layer_name in classifier_layer_names:
        x = model.get_layer(layer_name)(x)
    classifier_model = keras.Model(classifier_input, x)

    # Then, we compute the gradient of the top predicted class for our input image
    # with respect to the activations of the last conv layer
    with tf.GradientTape() as tape:
        # Compute activations of the last conv layer and make the tape watch it
        last_conv_layer_output = last_conv_layer_model(img_array)
        tape.watch(last_conv_layer_output)
        # Compute class predictions
        preds = classifier_model(last_conv_layer_output)
        top_pred_index = tf.argmax(preds[0])
        top_class_channel = preds[:, top_pred_index]

    # This is the gradient of the top predicted class with regard to
    # the output feature map of the last conv layer
    grads = tape.gradient(top_class_channel, last_conv_layer_output)

    # This is a vector where each entry is the mean intensity of the gradient
    # over a specific feature map channel
    pooled_grads = tf.reduce_mean(grads, axis=(0, 1, 2))

    # We multiply each channel in the feature map array
    # by "how important this channel is" with regard to the top predicted class
    last_conv_layer_output = last_conv_layer_output.numpy()[0]
    pooled_grads = pooled_grads.numpy()
    for i in range(pooled_grads.shape[-1]):
        last_conv_layer_output[:, :, i] *= pooled_grads[i]

    # The channel-wise mean of the resulting feature map
    # is our heatmap of class activation
    heatmap = np.mean(last_conv_layer_output, axis=-1)

    # For visualization purpose, we will also normalize the heatmap between 0 & 1
    heatmap = np.maximum(heatmap, 0) / np.max(heatmap)
    return heatmap
```

Slika 86 - make_gradcam_heatmap funkcija

Na sljedećoj slici je prikazan ostatak programskog koda koji je korišten za GradCAM tehniku. Najprije je procesuirana slika, kao i toplinska mapa tako da vrijednosti budu u intervalu 0-255, jer je tih vrijednosti i slika. Korištena je superpozicija toplinske mape na stvarnu sliku kako bi bilo vidljivo na koje značajke neuronska mreža obraća pažnju kada klasificira sliku.

```
# We load the original image
img = keras.preprocessing.image.load_img(img_path)
img = keras.preprocessing.image.img_to_array(img)

# We rescale heatmap to a range 0-255
heatmap = np.uint8(255 * heatmap)

# We use jet colormap to colorize heatmap
jet = cm.get_cmap("jet")

# We use RGB values of the colormap
jet_colors = jet(np.arange(256))[:, :3]
jet_heatmap = jet_colors[heatmap]

# We create an image with RGB colorized heatmap
jet_heatmap = keras.preprocessing.image.array_to_img(jet_heatmap)
jet_heatmap = jet_heatmap.resize((img.shape[1], img.shape[0]))
jet_heatmap = keras.preprocessing.image.img_to_array(jet_heatmap)

# Superimpose the heatmap on original image
superimposed_img = jet_heatmap * 0.4 + img
superimposed_img = keras.preprocessing.image.array_to_img(superimposed_img)

# Save the superimposed image
save_path = "pneumonia.jpg"
superimposed_img.save(save_path)

# Display Grad CAM
#display(Image(save_path))
%matplotlib inline

import matplotlib.pyplot as plt
import matplotlib.image as mpimg

no_cols=4
no_rows=4

pic_index=0
fig=plt.gcf()
fig.set_size_inches(no_cols*10,no_rows*10)

sp=plt.subplot(no_rows,no_cols,1)
sp.axis()
img=mpimg.imread(save_path)
plt.imshow(img,cmap='gray')

plt.show()
```

Slika 87 - Ostatak programskog koda za korištenje GradCAM tehnike

6.7 Web aplikacija

Za koristiti web aplikaciju, najprije je bilo potrebno instalirati *TensorflowJS* modul.

```
!pip install tensorflowjs
```

Slika 88 - Instaliranje TensorflowJS modula

Nadalje, potrebno je bilo spremiti korišteni model. Zbog toga što MobileNet sadrži mali broj parametara, onda je navedeni model idealan za korištenje u web pregledniku.

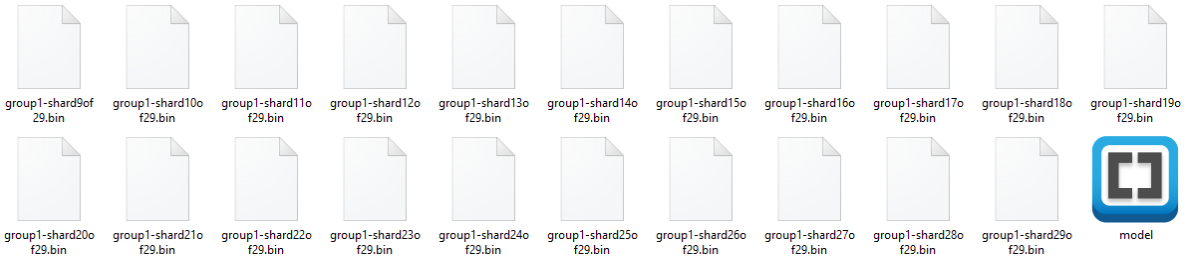
```
import time
saved_model_path = "/tmp/saved_models/{}".format(int(time.time()))
model.save(saved_model_path)
```

Slika 89 - Spremanje MobileNet modela

Potrebno je konvertirati spremljeni model kako bi se instance konvertiranog modela mogle koristiti u web pregledniku.

```
!tensorflowjs_converter \
  --input_format=keras_saved_model \
  /tmp/saved_models/1612132490 \
  /tmp/linear
```

Slika 90 - Konvertiranje modela



Slika 91 - Konvertirani model

Na sljedećoj slici prikazano je učitavanje *TensorflowJS* programske biblioteke i *jQuery* programske biblioteke. Također je definiran stil za određene *HTML* elemente.

```
<html lang="en">
<head>
    <title>Pneumonia detection</title>
<script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs@latest"> </script>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
</head>
<body>
<style>
    img {
        display: block;
        margin-left: auto;
        margin-right: auto;
    }

    p {
        display: block;
        margin-left: auto;
        margin-right: auto;
        width:400px;
    }
    .wrapper {
        height: 300px;
        display: flex;
        align-items: center;
        justify-content: center;
    }
</style>
```

Slika 92 - Učitavanje Javascript programskih biblioteka i stila

Pomoću sljedećeg *PHP* koda provjereno je učitavanje slike, provjerena je vrsta učitane datoteke, kao i veličina slike. Također je provjereno je li slika pod učitanim imenom već postoji.

```php
if(isset($_POST["submit"])) {
    $target_dir = "uploads/";
    $target_file = $target_dir . basename($_FILES["fileToUpload"]["name"]);
    echo '<img id="img" src="' . $target_dir . basename($_FILES["fileToUpload"]["name"]) . '"
        style="width:400px;height:400px;margin:0 auto;"/>';
    $uploadOk = 1;
    $imageFileType = strtolower(pathinfo($target_file, options: PATHINFO_EXTENSION));
    $check = getimagesize($_FILES["fileToUpload"]["tmp_name"]);
    if($check !== false) {
        //echo "<p>File is an image - " . $check["mime"] . ".</p>";
        $uploadOk = 1;
    } else {
        echo "<p>File is not an image.</p>";
        $uploadOk = 0;
    }
    // Check if file already exists
    if (file_exists($target_file)) {
        echo "<p>File already exists.</p>";
        $uploadOk = 0;
    }
// Check file size
    if ($_FILES["fileToUpload"]["size"] > 50000000) {
        echo "<p>Sorry, your file is too large.</p>";
        $uploadOk = 0;
    }
```

Slika 93 - PHP programski dio za učitavanje slike i provjeru veličine slike

Pomoću sljedećeg *PHP* programskog koda provjeren je tip slike te je provjereno je li slika spremljena.

```php
// Allow certain file formats
    if($imageFileType != "jpg" && $imageFileType != "png" && $imageFileType != "jpeg"
        && $imageFileType != "gif" ) {
        echo "<p>Sorry, only JPG, JPEG, PNG & GIF files are allowed.</p>";
        $uploadOk = 0;
    }

// Check if $uploadOk is set to 0 by an error
    if ($uploadOk == 0) {
        echo "<p>Sorry, your file was not uploaded.</p>";
// if everything is ok, try to upload file
    } else {
        if (move_uploaded_file($_FILES["fileToUpload"]["tmp_name"], $target_file)) {
            echo "<p>The file ". htmlspecialchars( basename( $_FILES["fileToUpload"]["name"])). " has been uploaded.</p>";
        } else {
            echo "<p>Sorry, there was an error uploading your file.</p>";
        }
    }
```

Slika 94 - PHP programski kod za provjeru tipa slike te za spremanje slike

Pomoću sljedećeg programskog koda prikazana je forma za unos slike.

```html
<div class="wrapper">
    <form action="" method="post" enctype="multipart/form-data"
          style="width: 400px;margin: 0 auto;">
        Select image to upload:
        <input type="file" name="fileToUpload" id="fileToUpload">
        <input type="submit" value="Upload Image" name="submit">
    </form>
</div>
```

Slika 95 - HTML i CSS kod za formu za unos slike

U sljedećem *Javascript* kodu najprije je učitan model, dohvaćena slika, pa je slika zatim skalirana i normalizirana kako bi se obavila klasifikacija slike.

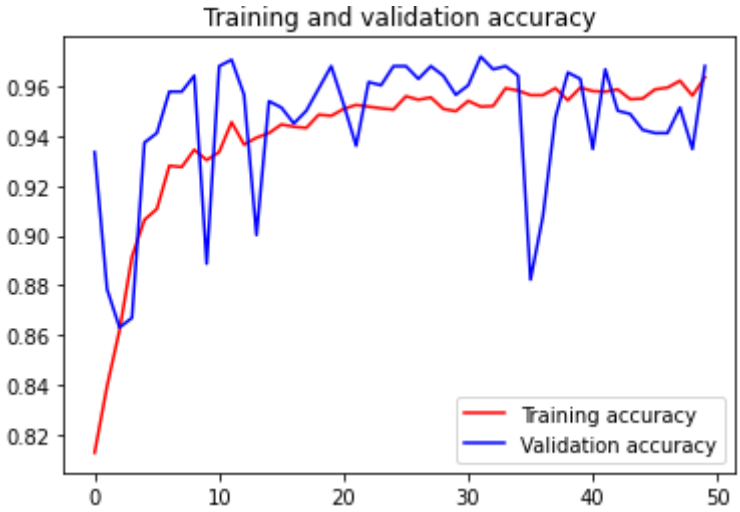
```
<script>
<?php
if(isset($_POST["submit"])) {
    echo "async function run(){
        const MODEL_URL = 'http://localhost/upala_pluca_detekcija2/model.json';
        const model = await tf.loadLayersModel(MODEL_URL);
        console.log(model.summary());
        const image = document.getElementById('img');
        var img = tf.browser.fromPixels(image, 3).resizeNearestNeighbor([180,180]).toFloat();
        const offset = tf.scalar(255.0);
        var normalized  = img.div(offset);
        const axis = 0;
        normalized = normalized.expandDims(axis);
        console.log(normalized.shape);
        prediction = model.predict(normalized);
        console.log(prediction.dataSync());
        var pIndex = tf.argMax(prediction, 1).dataSync();
        var classNames = ['Normal', 'Pneumonia'];
        alert(classNames[pIndex]);
    }
    run();";
    }
?>
</script>
```

Slika 96 - Javascript dio koda za korištenje modela

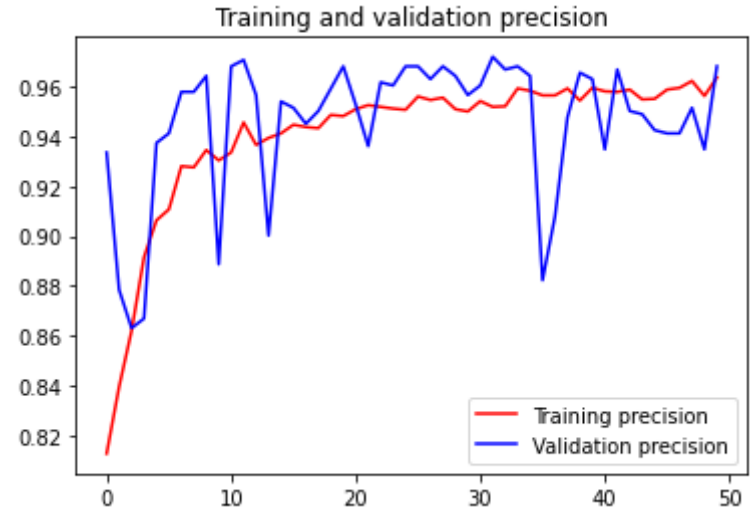
7. REZULTATI

7.1 Osobni model

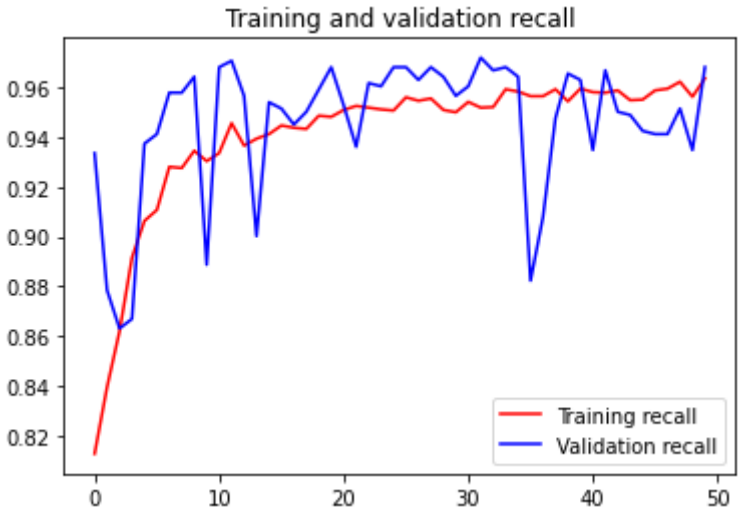
Pomoću sljedećih slika prikazane su točnost, preciznost, odziv i gubitak osobnog modela na trening i validacijskom skupu. Na *Y* osi prikazana je neka od navedenih metrika, dok je na *X* osi prikazan broj epoha.



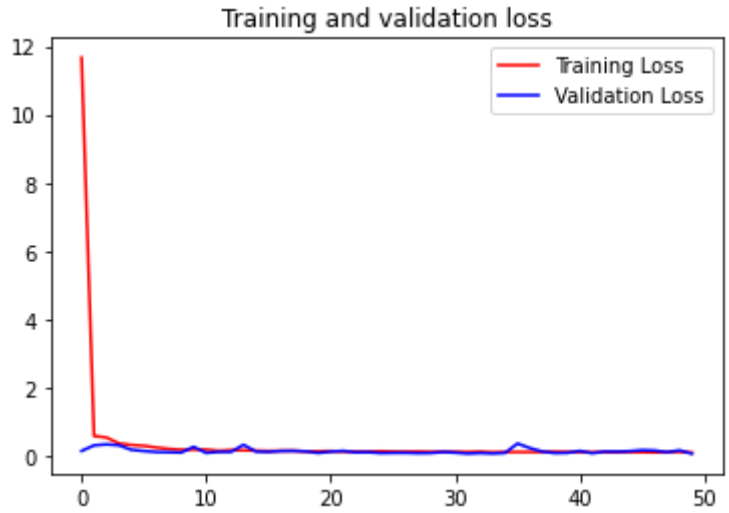
Slika 97 - Trening i validacijska točnost osobnog modela



Slika 98 - Trening i validacijska preciznost osobnog modela



Slika 99 - Trening i validacijski odziv osobnog modela



Slika 100 - Trening i validacijski gubitak osobnog modela

Na sljedećoj slici je prikazana evaluacija osobnog modela.

```
evaluation = model.evaluate(test_generator)
print('Loss  :', evaluation[0])
print('Accuracy  :', evaluation[1])
print('Precision :', evaluation[2])
print('Recall :', evaluation[3])
print('True positives :', evaluation[4])
print('True negatives :', evaluation[5])
print('False negatives :', evaluation[6])
print('False positives :', evaluation[7])
```

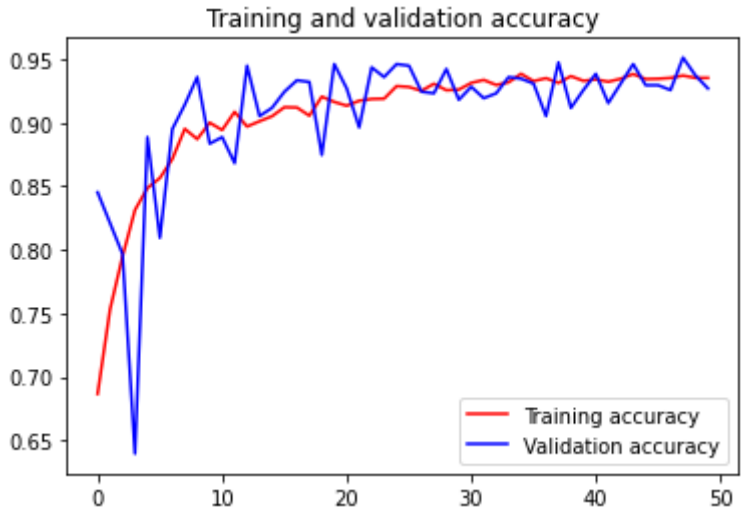
```
7/7 [==============================] - 144s
Loss  : 0.10018554329872131
Accuracy  : 0.9731457829475403
Precision : 0.9946902394294739
Recall : 0.9689655303955078
True positives : 562.0
True negatives : 199.0
False negatives : 18.0
False positives : 3.0
```

Slika 101 - Evaluacija osobnog modela

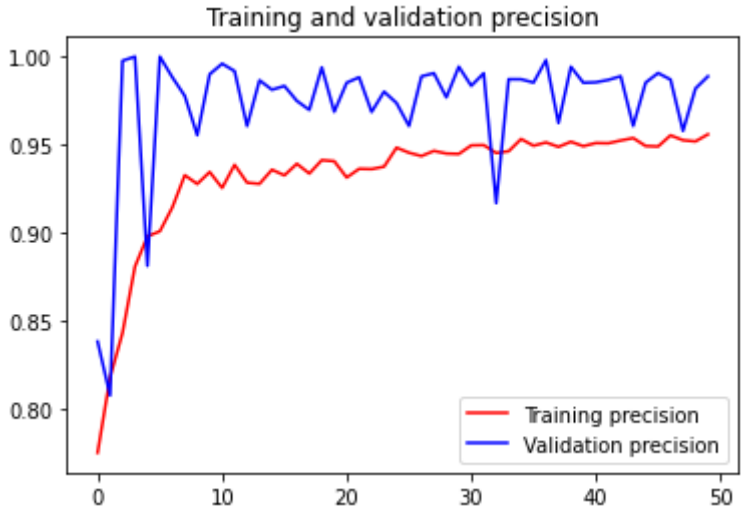
Vidljiva je visoka točnost modela, ali je i vidljiv relativno visok broj lažno negativnih, kao i odziv, što korelira s višim brojem lažno negativnih – to je u slučaju medicine najbitnija metrika – naravno, bilo bi što bolje kada bi navedena metrika pokazivala što manji broj. Ista stvar se može zaključiti za ostale modele.

7.2 InceptionV3

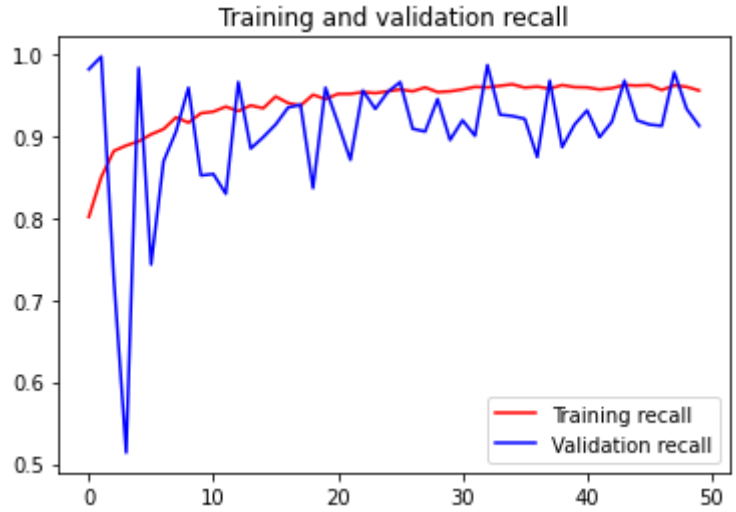
Pomoću sljedećih slika prikazane su točnost, preciznost, odziv i gubitak InceptionV3 modela na trening i validacijskom skupu. Na *Y* osi prikazana je neka od navedenih metrika, dok je na *X* osi prikazan broj epoha.



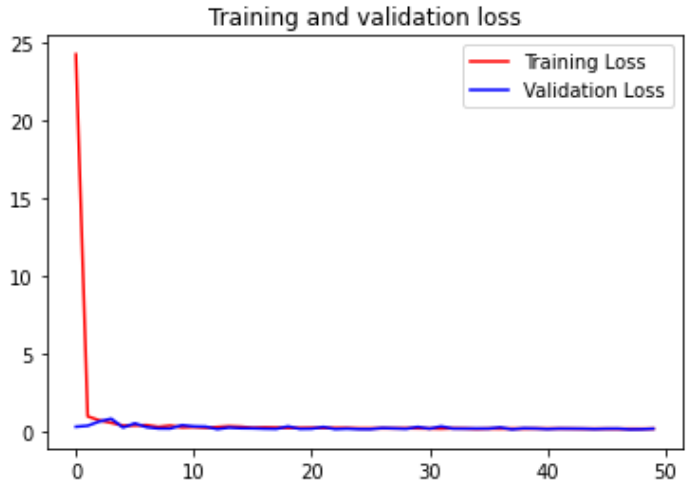
Slika 102 - Trening i validacijska točnost InceptionV3 modela



Slika 103 - Trening i validacijska preciznost InceptionV3 modela



Slika 104 - Trening i validacijska odziv InceptionV3 modela



Slika 105 - Trening i validacijski gubitak InceptionV3 modela

Na sljedećoj slici je prikazana evaluacija InceptionV3 modela.

```
evaluation = model.evaluate(test_generator)
print('Loss   :', evaluation[0])
print('Accuracy  :', evaluation[1])
print('Precision :', evaluation[2])
print('Recall :', evaluation[3])
print('True positives :', evaluation[4])
print('True negatives :', evaluation[5])
print('False negatives :', evaluation[6])
print('False positives :', evaluation[7])
```

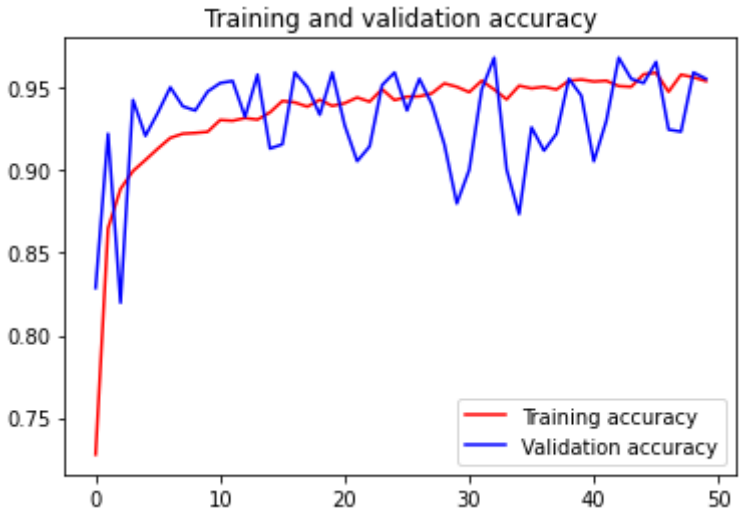
```
7/7 [==============================] - 133s
Loss   : 0.17190653085708618
Accuracy  : 0.9386188983917236
Precision : 0.9925925731658936
Recall : 0.9241379499435425
True positives : 536.0
True negatives : 198.0
False negatives : 44.0
False positives : 4.0
```

Slika 106 - Evaluacija InceptionV3 modela

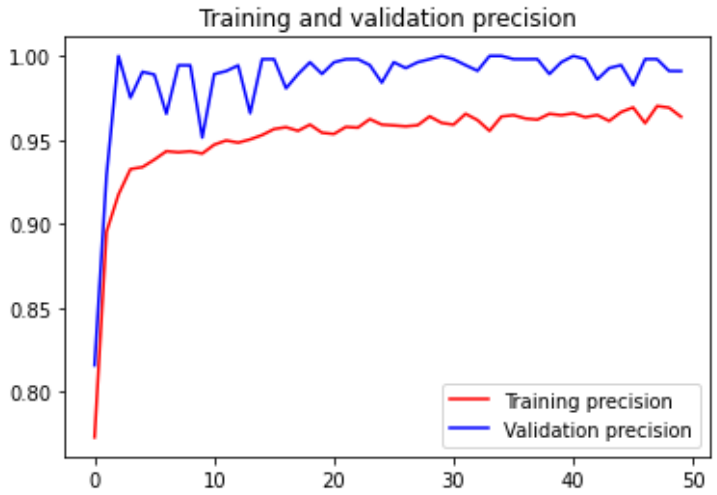
Uočljiva je relativno visoka točnost modela. Može se primijetiti dosta veći broj lažno negativnih u odnosu na osobni model.

7.3 DenseNet121

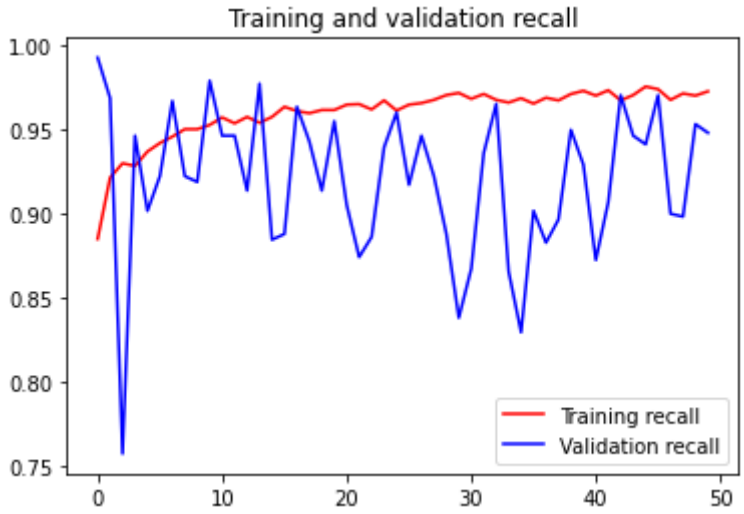
Pomoću sljedećih slika prikazane su točnost, preciznost, odziv i gubitak DenseNet121 modela na trening i validacijskom skupu. Na *Y* osi prikazana je neka od navedenih metrika, dok je na *X* osi prikazan broj epoha.



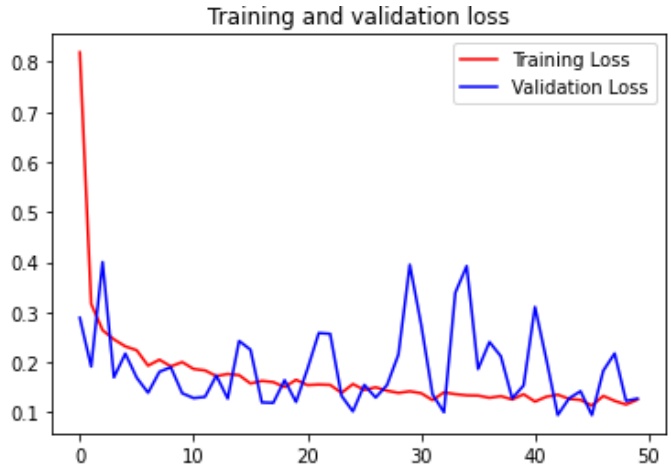
Slika 107 - Trening i validacijska točnost DenseNet121 modela



Slika 108 - Trening i validacijska preciznost DenseNet121 modela



Slika 109 - Trening i validacijski odziv DenseNet121 modela



Slika 110 - Trening i validacijski gubitak DenseNet121 modela

Na sljedećoj slici je prikazana evaluacija DenseNet121 modela.

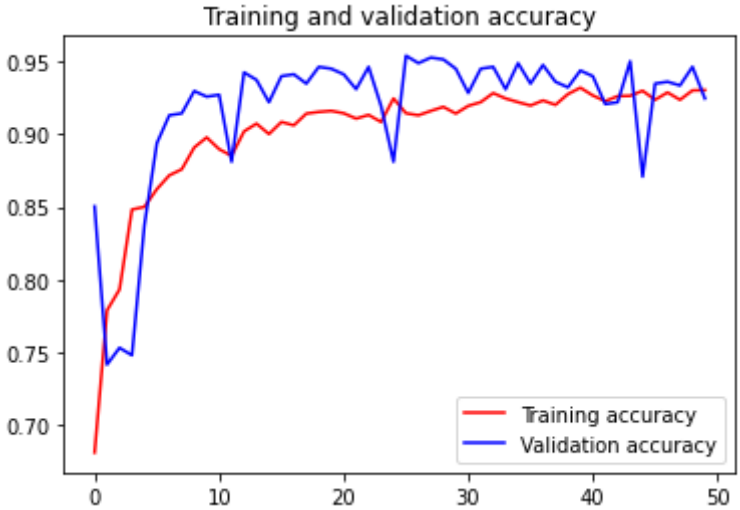
```
evaluation = model.evaluate(test_generator)
print('Loss   :', evaluation[0])
print('Accuracy  :', evaluation[1])
print('Precision :', evaluation[2])
print('Recall :', evaluation[3])
print('True positives :', evaluation[4])
print('True negatives :', evaluation[5])
print('False negatives :', evaluation[6])
print('False positives :', evaluation[7])
```

```
7/7 [==============================] - 144s
Loss   : 0.10018554329872131
Accuracy  : 0.9731457829475403
Precision : 0.9946902394294739
Recall : 0.9689655303955078
True positives : 562.0
True negatives : 199.0
False negatives : 18.0
False positives : 3.0
```

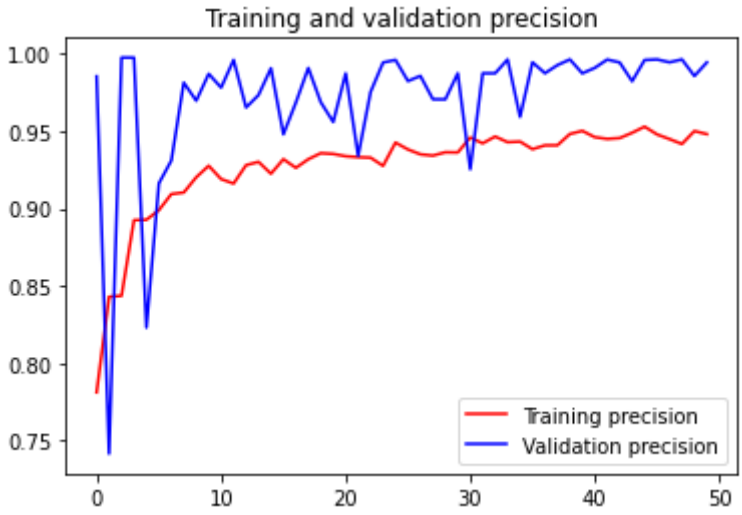
Prema rezultatima je vidljivo da je DenseNet121 model po evaluaciji dosta sličan osobnom modelu.

7.4 VGG19

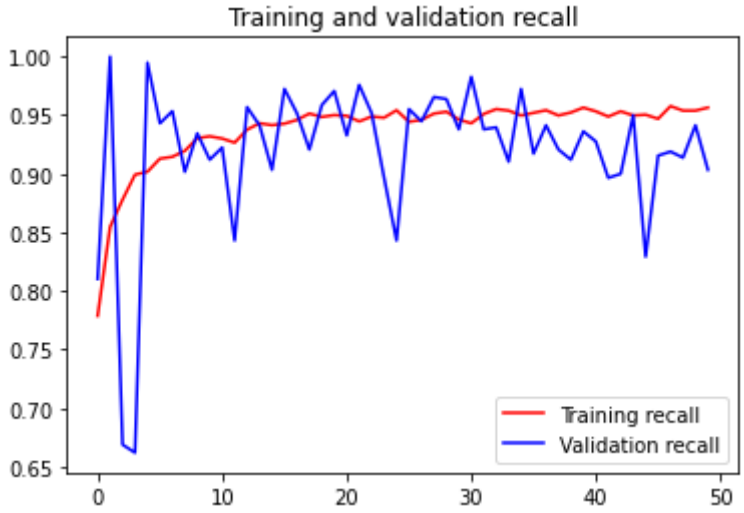
Pomoću sljedećih slika prikazane su točnost, preciznost, odziv i gubitak VGG19 modela na trening i validacijskom skupu. Na *Y* osi prikazana je neka od navedenih metrika, dok je na *X* osi prikazan broj epoha.



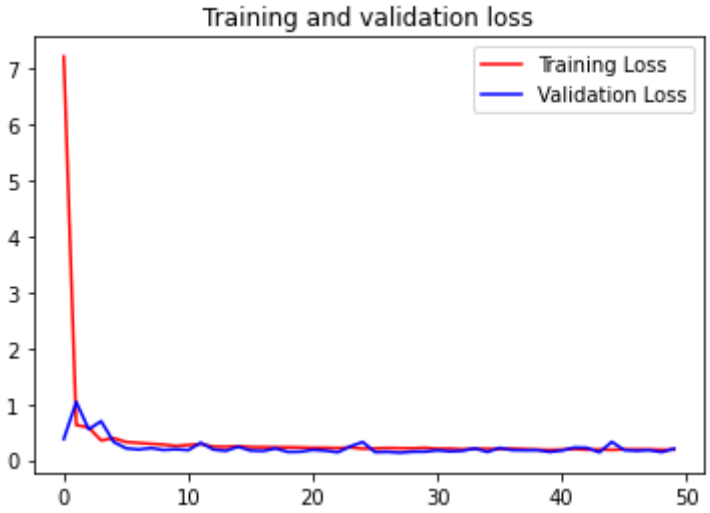
Slika 111 - Trening i validacijska točnost VGG19 modela



Slika 112- Trening i validacijska preciznost VGG19 modela



Slika 113 - Trening i validacijski odziv VGG19 modela



Slika 114 - Trening i validacijski gubitak VGG19 modela

Na sljedećoj slici je prikazana evaluacija VGG19 modela.

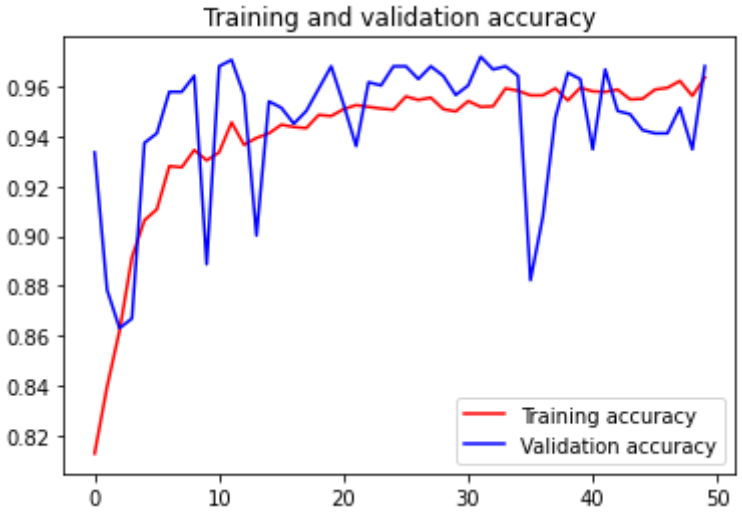
```
evaluation = model.evaluate(test_generator)
print('Loss   :', evaluation[0])
print('Accuracy  :', evaluation[1])
print('Precision :', evaluation[2])
print('Recall :', evaluation[3])
print('True positives :', evaluation[4])
print('True negatives :', evaluation[5])
print('False negatives :', evaluation[6])
print('False positives :', evaluation[7])
```

```
7/7 [==============================] - 332s
Loss   : 0.19576068222522736
Accuracy  : 0.9296675324440002
Precision : 0.996219277381897
Recall : 0.9086207151412964
True positives : 527.0
True negatives : 200.0
False negatives : 53.0
False positives : 2.0
```

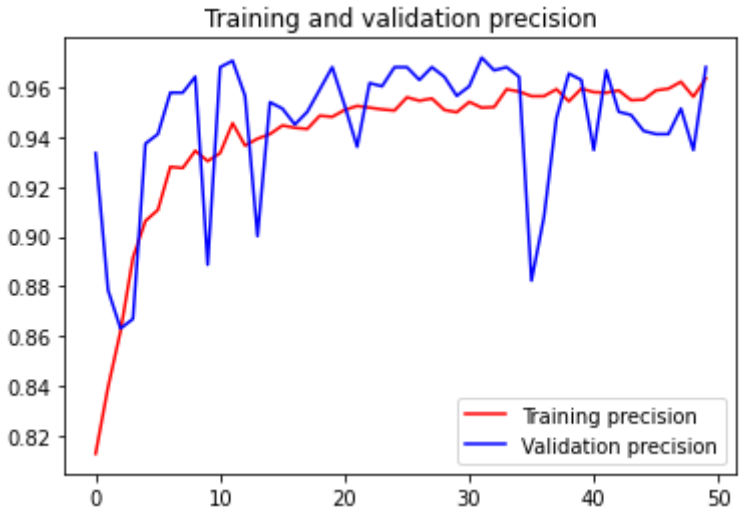
Prema rezultatima evaluacije VGG19 modela vidljiv je dosta velik broj lažno negativnih.

7.5 MobileNet

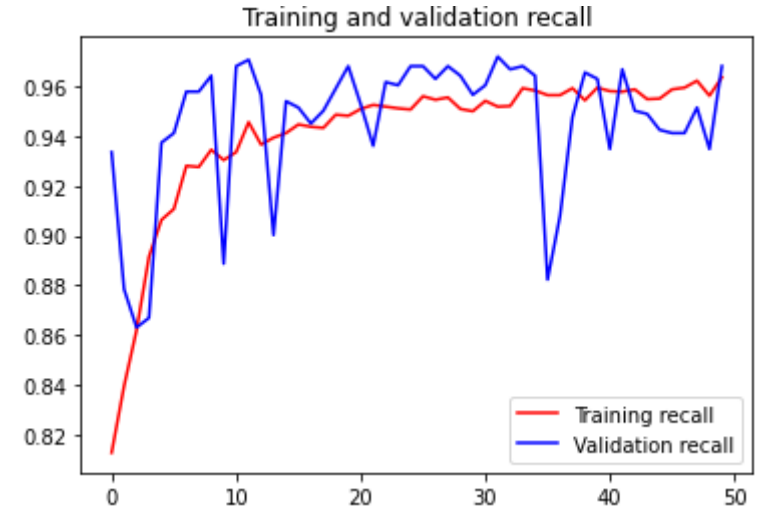
Pomoću sljedećih slika prikazane su točnost, preciznost, odziv i gubitak MobileNet modela na trening i validacijskom skupu. Na *Y* osi prikazana je neka od navedenih metrika, dok je na *X* osi prikazan broj epoha.



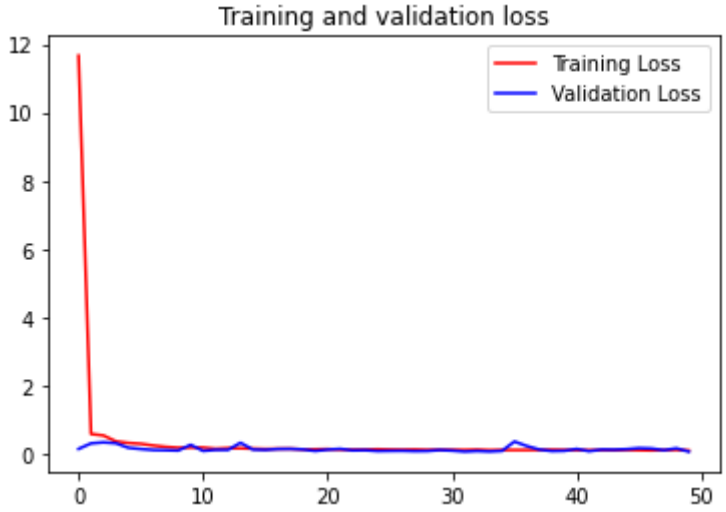
Slika 115 - Trening i validacijska točnost MobileNet modela



Slika 116 - Trening i validacijska preciznost MobileNet modela



Slika 117 - Trening i validacijski odziv MobileNet modela



Slika 118 - Trening i validacijski gubitak MobileNet modela

Na sljedećoj slici je prikazana evaluacija MobileNet modela.

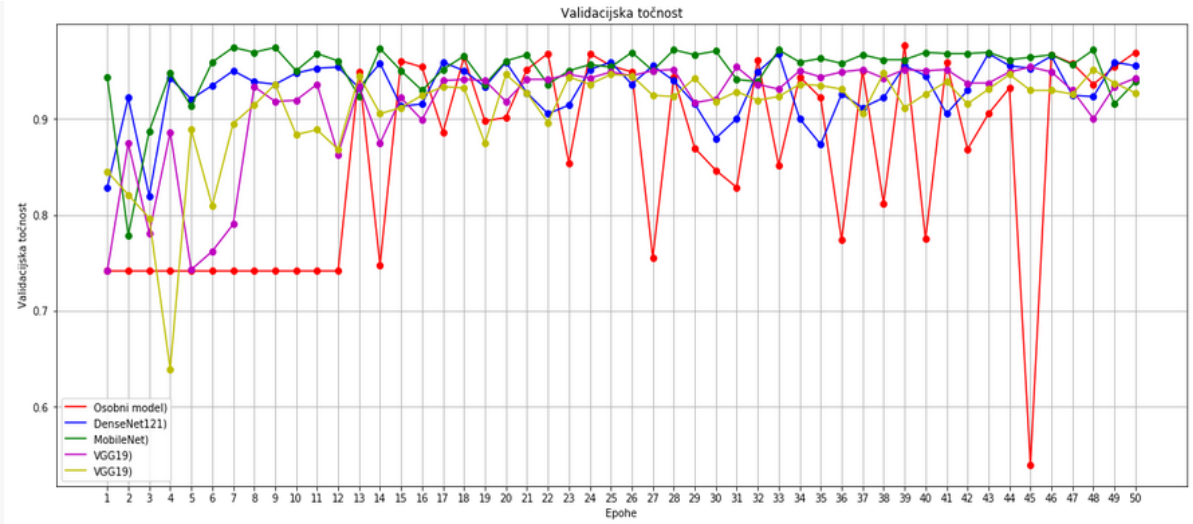
```
evaluation = model.evaluate(test_generator)
print('Loss   :', evaluation[0])
print('Accuracy  :', evaluation[1])
print('Precision :', evaluation[2])
print('Recall :', evaluation[3])
print('True positives :', evaluation[4])
print('True negatives :', evaluation[5])
print('False negatives :', evaluation[6])
print('False positives :', evaluation[7])
```

```
7/7 [==============================] - 7s 1s/
Loss   : 0.09993147104978561
Accuracy  : 0.97826087474823
Precision : 0.97826087474823
Recall : 0.97826087474823
True positives : 765.0
True negatives : 765.0
False negatives : 17.0
False positives : 17.0
```

Vidljivo je da je u slučaju MobileNet modela najviša točnost, broj lažno negativnih za samo jedan je manji od osobnog modela i DenseNet121 modela, ali u slučaju lažno pozitivnih, uspoređujući s osobnim modelom i DenseNet121 modelom, vidljivo je da se radi o relativno velikom broju lažno pozitivnih.

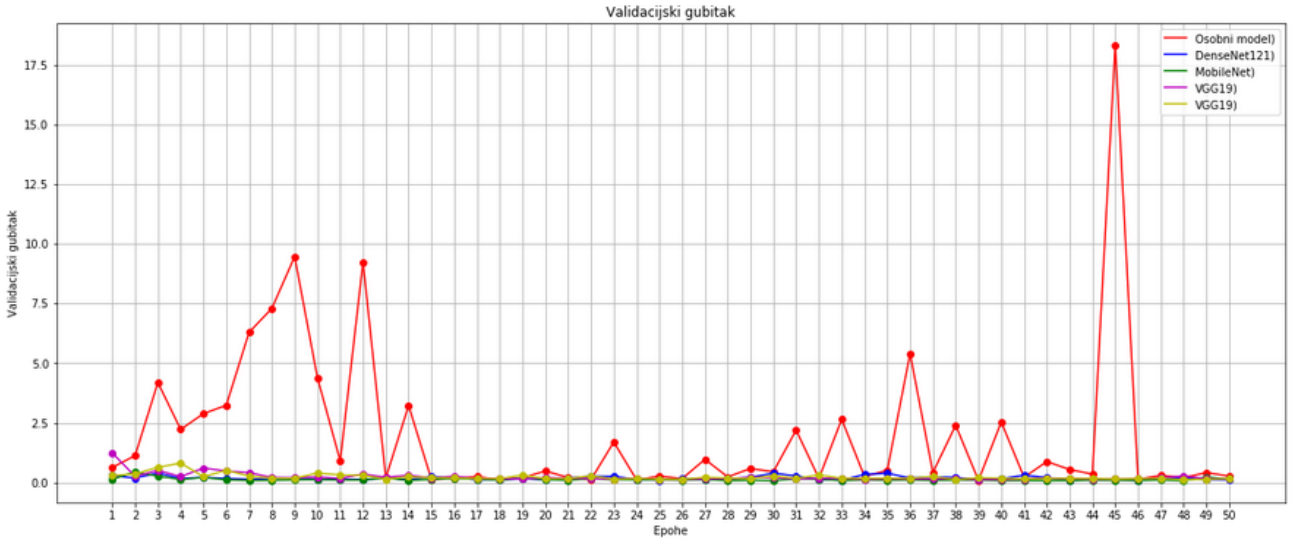
7.6 Svi modeli zajedno

Na sljedećim slikama je prikazana validacijska točnost kao i validacijski gubitak navedenih modela. Na prvom grafu *X* os predstavlja broj epoha, dok *Y* os predstavlja validacijsku točnost. Na drugom grafu *X* os predstavlja broj epoha, dok *Y* os predstavlja validacijski gubitak.



Slika 119 - Graf validacijske točnosti svih modela zajedno

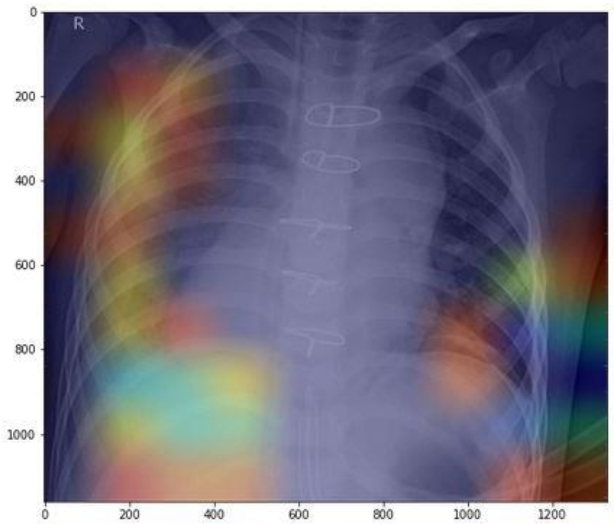
Fluktuacija točnosti se može primijetiti kod svakog modela, baš kao što je prikazana na prethodnim grafovima. Može se primijetiti kako je u slučaju svakog modela, osim u slučaju osobnog modela, malena oscilacija validacijskog gubitka.



Slika 120 - Graf gubitka svih modela zajedno

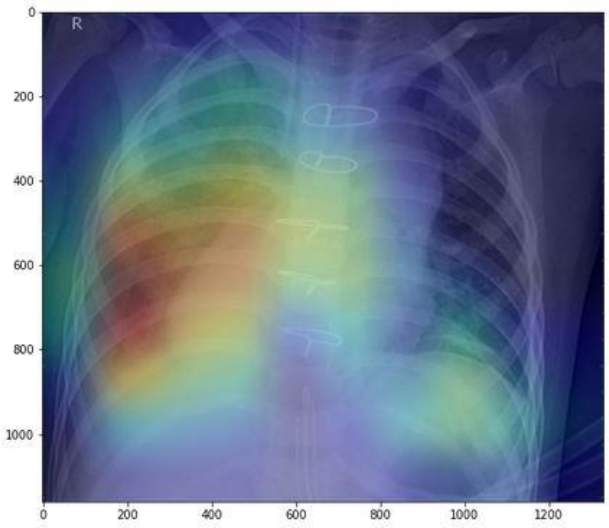
7.7 GradCAM

Na sljedećoj slici je prikazan rezultat korištenja GradCAM tehnike bez treniranja modela.



Slika 121 - Korištenje GradCAM tehnike bez treniranja modela

Na sljedećoj slici je prikazan rezultat korištenja GradCAM tehnike s treniranjem modela.



Slika 122 - Korištenje GradCAM tehnike s treniranjem modela

ZAKLJUČAK

Računalni vid je područje umjetne inteligencije koje zahvaljujući konvolucijskim neuronskim mrežama bilježi sve bolje rezultate. Budući da se u ovom radu radi o problemu klasifikacije slike magnetske rezonance, onda su korištene konvolucijske neuronske mreže.

Provjerena je funkcionalnost GradCAM tehnike tako da se pokušala dobiti slika bez i s treniranjem modela. Kada se dobila slika s treniranjem modela, tada je prikazana crvena boja na području pluća, dok su na ostalim dijelovima slike hladne boje koje su odraz manje važnosti značajki na klasifikaciju slike. U slučaju kada model nije treniran, položaj boja je nasumičan. Budući da je crvena boja odraz veće povezanosti značajke s klasifikacijom, a crvena boja se nalazi na području pluća, onda se može zaključiti da model pri klasifikaciji slike obraća pažnju na ispravne značajke.

Proces klasifikacije na web aplikaciji je relativno spor pa se za problem predikcije modela predlaže korištenje *dekstop* aplikacije. U slučaju korištenja *dekstop* aplikacije, zbog dosta veće brzine rada modela u odnosu na web aplikaciju, bilo bi dobro koristiti više modela kako bi se smanjila mogućnost greške.

Za bolje rezultate predlaže se pojačana podatkovna augmentacija, kao i prikupljanje većeg broja slika.

Budući da je korišteno besplatno okruženje *Google Colaboratory*, eksperimentiranje metoda je ograničeno memorijom i vremenom korištenja. Da bi se dodatno poboljšali rezultati, predlaže se treniranje modela u nekom drugom okruženju s većom količinom memorije i većim vremenskim pristupom.

Što se tiče uporabe dobivenih modela u praksi trebalo bi se voditi logikom da korišteni model bude onaj s najmanjom količinom lažno negativnih. Broj lažno negativnih je najvažnija metrika za medicinu jer govori o tome koliko je model puta pogriješio kada je izlazna vrijednost bila predikcija da čovjek nema, a zapravo u stvarnosti ima bolest. Ako bismo pratili metriku lažno negativnih, onda bi se MobileNet model koristio u praksi. Problem MobileNet modela, ako se usporedi s osobnim modelom i DenseNet121 modelom, je relativno visok broj lažno pozitivnih.

Dodatna prednost MobileNet modela i osobnog modela su manji broj parametara, što znači brže izvođenje modela u stvarnosti.

LITERATURA

[1] Ž. U. Andrijić, »https://hrcak.srce.hr,« [Mrežno] https://hrcak.srce.hr/file/322233.

[2] I. Židov, »https://repozitorij.mathos.hr,« [Mrežno] https://repozitorij.mathos.hr/islandora/object/mathos%3A256/datastream/PDF/view.

[3] S. Pattanayak, »https://github.com,« [Mrežno]. https://github.com/aniruddhachoudhury/Data-Science-Books/blob/master/Pro%20Deep%20Learning%20with%20TensorFlow%20-%20A%20Mathematical%20Approach%20to%20Advanced%20Artificial%20Intelligence%20in%20Python.pdf. Pristupljeno 9. travnja 2021.

[4] N. Buduma, »Fundametals of Deep learning,« [Mrežno] : http://perso.ens-lyon.fr/jacques.jayez/Cours/Implicite/Fundamentals_of_Deep_Learning.pdf.

[5] »https://line.17qq.com,« [Mrežno] https://line.17qq.com/articles/ssnnnknky.html. Pristupljeno 9. travnja 2021.

[6] S. Sharma, »https://towardsdatascience.com,« 2021. [Mrežno].https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6.

[7] A. Ng, »https://www.coursera.org,« [Mrežno]. https://www.coursera.org/learn/neural-networks-deep-learning/lecture/yWaRd/logistic-regression-cost-function. Pristupljeno 8. travnja 2021.

[8] A. Ng, »https://www.coursera.org,« [Mrežno] https://www.coursera.org/learn/neural-networks-deep-learning/lecture/XtFPI/random-initialization. Pristupljeno 15. travnja 2021.

[9] [Mrežno] http://datahacker.rs/gradient-descent-neural-networks/. Pristupljeno 15. travnja 2015.

[10] A. Ng, »https://www.coursera.org,« [Mrežno] https://www.coursera.org/learn/deep-neural-network/lecture/cxG1s/train-dev-test-sets Pristupljeno 17. travnja 2021.

[11] »https://www.researchgate.net,« [Mrežno] https://www.researchgate.net/figure/Dropout-neural-network-model-a-is-a-standard-neural-network-b-is-the-same-network_fig3_309206911 Pristupljeno 18. travnja 2021.

[12] B. Chen, »https://laptrinhx.com,« [Mrežno] https://laptrinhx.com/a-practical-introduction-to-early-stopping-in-machine-learning-2456645136/. Pristupljeno 18. travnja 2021.

[13] S. Goyal, »https://towardsdatascience.com,« [Mrežno] https://towardsdatascience.com/machinex-image-data-augmentation-using-keras-b459ef87cd22. Pristupljeno 18. travnja 2021.

[14] A. Ng, »https://www.coursera.org,« [Mrežno] https://www.coursera.org/learn/deep-neural-network/lecture/lXv6U/normalizing-inputs. Pristupljeno 18. travnja 2021.

[15] S. Segvić, »http://www.zemris.fer.hr,« [Mrežno] http://www.zemris.fer.hr/~ssegvic/du/du6recurrent2.pdf. Pristupljeno 18. travnja 2021.

[16] A. Ng, »https://www.coursera.org/,« [Mrežno] https://www.coursera.org/learn/deep-neural-network/lecture/RwqYe/weight-initialization-for-deep-networks. Pristupljeno 20. travnja 2021.

[17] A. Ng, »https://www.coursera.org,« [Mrežno] https://www.coursera.org/learn/deep-neural-network/lecture/XzSSa/numerical-approximation-of-gradients. Pristupljeno 22. travnja 2021.

[18] E. Irby, »https://medium.com,« [Mrežno] https://medium.com/analytics-vidhya/gradient-descent-vs-stochastic-gd-vs-mini-batch-sgd-fbd3a2cb4ba4. Pristupljeno 1. svibnja 2021.

[19] A. Ng, »https://www.coursera.org,« [Mrežno] https://www.coursera.org/learn/deep-neural-network/lecture/duStO/exponentially-weighted-averages. Pristupljeno 1. svibnja 2021.

[20] A. Ng. [Mrežno] https://www.coursera.org/learn/deep-neural-network/lecture/w9VCZ/adam-optimization-algorithm. Pristupljeno 1. svibnja 2021.

[21] A. Ng, »https://www.coursera.org,« [Mrežno] https://www.coursera.org/learn/deep-neural-network/lecture/hjgIA/learning-rate-decay. Pristupljeno 6. svibnja 2021.

[22] N. Mohan, »https://www.linkedin.com,« [Mrežno]. https://www.linkedin.com/pulse/artificial-intelligence-applications-neural-style-transfer-mohan/.

[23] [Mrežno] https://serengetitech.com/tech/deep-learning-object-detection/.

[24] [Mrežno] http://datahacker.rs/edge-detection/.

[25] [Mrežno] http://datahacker.rs/edge-detection-extended/.

[26] [Mrežno] http://datahacker.rs/what-is-stride-cnn/.

[27] [Mrežno] http://datahacker.rs/what-is-padding-cnn/.

[28] http://datahacker.rs/convolution-rgb-image/. [Mrežno].

[29] [Mrežno] http://datahacker.rs/one-layer-covolutional-neural-network/.

[30] [Mrežno] http://datahacker.rs/pooling-layers-cnn/.

[31] [Mrežno] http://datahacker.rs/convolutional-neural-network-begineers/.

[32] [Mrežno] http://datahacker.rs/why-convolution/.

[33] [Mrežno] http://datahacker.rs/transfer-learning-deep-learning/.

[34] [Mrežno] https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202.

[35] [Mrežno] https://arthurdouillard.com/post/densenet/.

[36] [Mrežno] http://datahacker.rs/deep-learning-vgg-16-vs-vgg-19/.

[37] [Mrežno] https://medium.com/analytics-vidhya/image-classification-with-mobilenet-cc6fbb2cd470.

[38] [Mrežno] https://www.coursera.org/learn/ai-for-medical-treatment/lecture/qoD4p/localization-maps.

[39] [Mrežno] https://www.coursera.org/learn/ai-for-medical-treatment/lecture/mofKv/heat-maps.

PRILOZI

Popis tablica

Popis slika

IZJAVA

Izjavljujem pod punom moralnom odgovornošću da sam diplomski rad izradio samostalno, isključivo znanjem stečenim na Odjelu za elektrotehniku i računarstvo, služeći se navedenim izvorima podataka i uz stručno vodstvo mentora izv.prof.dr.sc. Maria Miličevića, kome se još jednom srdačno zahvaljujem.

Luka Krnetić