Programska aplikacija za otkrivanje značajki na licu

Bračun, Juraj

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Juraj Bračun

Zagreb, 2021. godina.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Mentor: Student:

doc. dr. sc. Tomislav Stipančić, dipl. ing. Juraj Bračun

Zagreb, 2021.godina.

Izjavljeum da sam ovaj rad radio samostalno koristeći znanja stečena tijekom studija i navedenu literaturu.

Zahvaljujem se mentoru doc. dr. sc. Tomislavu Stipančiću na pruženoj pomoći i utrošenom vremenu kako bih uspješno napisao ovaj rad. Hvala svim kolegicama i kolegama te prijateljima i prijateljicama na dobro provedenom vremenu tijekom dosadašnjeg studiranja te na svim lijepim riječima potpore kada je bilo najteže. Nadam se da je najteži dio iza mene i da u nadolazećim semestrima neće biti nikakvih poteškoća prilikom studiranja.

Juraj Bračun

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE
Središnje povjerenstvo za završne i diplomske ispite
Povjerenstvo za završne ispite studija strojarstva za smjerove:
proizvodno inženjerstvo, računalno inženjerstvo, industrijsko inženjerstvo i menadžment, inženjerstvo materijala i mehatronika i robotika

Sveučilište u Zagrebu Fakultet strojarstva i brodogradnje	
Datum	Prilog
Klasa: **602 - 04 / 21 - 6 / 1**	
Ur.broj: **15 - 1703 - 21 -**	

ZAVRŠNI ZADATAK

Student: **Juraj Bračun** Mat. br.: 0035210477

Naslov rada na hrvatskom jeziku: **Programska aplikacija za otkrivanje značajki na licu**

Naslov rada na engleskom jeziku: **Facial landmark detection application**

Opis zadatka:

Tehnike umjetne inteligencije i strojnog vida koriste se kod različitih primjena u okviru metoda za ostvarivanje robotske percepcije radnog prostora. Kao osnovu za prepoznavanje kontekstualnih informacija na ljudskom licu, u radu je potrebno primijeniti metodu za definiranje i prepoznavanje karakterističnih točaka čiji se pomaci analiziraju te dovode u vezu s čitanjem emocija ili tumačenjem izražaja lica, uključujući:

- proučiti metodologiju za prepoznavanje i otkrivanje osnovnih značajki na ljudskom licu,
- odrediti prikladni skup podataka za učenje konvolucijske neuronske mreže,
- razviti model konvolucijske neuronske mreže kroz fazu treninga koristeći odabrani skup podataka za učenje,
- koristiti razvijeni model za pronalazak karakterističnih točaka na slikama koje nisu bile korištene u fazi učenja.

Razvijeni algoritam za prepoznavanje i praćenje karakterističnih točaka na ljudskom licu potrebno je eksperimentalno verificirati u laboratorijskim uvjetima te analizirati njegovu pouzdanost.

U radu je potrebno navesti korištenu literaturu te eventualno dobivenu pomoć.

Zadatak zadan:
30. studenoga 2020.

Zadatak zadao:

Doc. dr. sc. Tomislav Stipančić

Datum predaje rada:
1. rok: 18 veljače 2021.
2. rok (izvanredni): 5. srpnja 2021.
3. rok: 23. rujna 2021.

Predviđeni datumi obrane:
1. rok: 22.2. – 26.2.2021.
2. rok (izvanredni): 9.7.2021.
3. rok: 27.9. – 1.10.2021.

Predsjednik Povjerenstva:

Prof. dr. sc. Branko Bauer

SADRŽAJ

POPIS SLIKA

POPIS OZNAKA

AdaGrad	- optimizacijski algoritam,
ch	- kosinus hiperbolni,
CSV	- *engl. Column Separated Vlaues,* datoteka u kojoj seu pohranjene vrijednosti odvojene zarezom,
e	- baza prirodnog logaritma,
FPS	- *engl. Frmaes Per Second*, broj sličica u jednoj sekundi,
MSELoss	- engl. Mean Square Error Loss, greška pri korištenju metode najmanjih kvadrata,
R	- ReL funkcija,
ReL	- *engl. Rectified Linear activation function*, aktivacijska funkcija,
RMSProp	- optimizacijski algoritam,
S	- Sigmoid funkcija,
sh	- sinus hiperbolni,
tnah,th	- tangens hiperbolni,

SAŽETAK

Umjetna inteligencija u zadnje je vrijeme neizostavan dio skoro svakog računalnog programa, web aplikacije ili robotskog sustava. Sposobnosti računala i robota uvelike su se proširile tijekom posljednjih nekoliko godina što je rezultiralo sve većem i sve prisutnijem korištenju umjetne inteligencije kao alata za rješavanje različitih problema u širokom spektru tehničkih područja. Jedno od područja primjene je i mogućnost sustava da procesuira vizualne informacije te iz njih nešto spoznaje ili zaključuje. Pojava strojnog vida omogućila je bolju povezanost i lakšu komunikaciju računala sa vanjskim svijetom te otvorila novi niz potencijalnih rješenja kompleksnih tehničkih problema. Tako je sposobnost prepoznavanja ljudkog lica i njegovih značajki poprimila jednu rasprostranjenu i široku primjenu u svakodnevnom životu.

Ljudsko lice jedan je od kompleksnijih dijelova našeg tijela. Sastoji se od nekoliko organa te vrlo složene geometrije. Potreba za njegovim prepoznavanjem te određivanjem pojedinosti na svakom individualnom licu bit će sve prisutnija u budućnosti. Stoga je vrlo važno primjeniti ispravne metode kojima se te potrebe mogu zadovoljiti kako bi se što lakše i što preciznije postigao ispravan odziv sustava.

Ključne riječi: umjetna inteligencija, strojni vid, ljudsko lice

SUMMARY

Artificial intelligence has become unavoidable part of almost every computer program, web application or robot system in recent history. Over the past few years, robot and computer abilities have grown significantly, which resulted in an increasing number of artificial intelligence usage as a tool for solving many kinds of problems in wide variety of technical areas. One such usage is an ability of system to process visual information and make some decisions and conclusions based on those information. The occurrrnce of machine vision has provided computers with better connection and easier communication with outside world which opened whole new range of potential solutions to ever existing complex technical problems. Thus the ability to recognize human face and it's characteristics has become widely used in everyday life.

Human face is one of the most complex part of our body. It consists of few organs and has very complicated geometry. Need for it's recognition and determining characteristics of each and every individual face will be even more present as the time goes by. Hence, it's very important to apply correct methods which can satisfy those needs so that computer response would be as easy and as precise as possible.

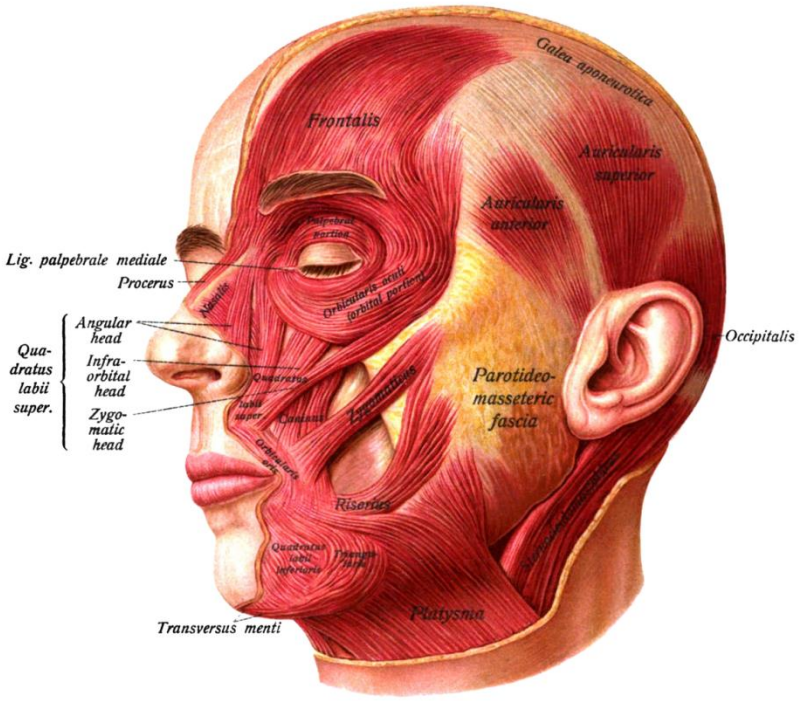
Key words: artificial intelligence, machine vision, human face

1. UVOD

Sve većom prisutnošću novijih tehnologija u svakodnevnom životu razvijaju se neke nove mogućnosti obavljanja pojedinih radnji na neki drugačiji način. Primjerice otključavanje mobilnog telefona nekad je zahtijevalo ručno upisivanje lozinke kako bi dobili pristup vlastitom uređaju. Danas se koriste neke nove metode kojima se rješio taj problem neželjenog pristupa drugih osoba mobilnom uređaju koji korisnik posjeduje putem biometrije (npr. otisak prsta) ili putem preopznavanja lica (engl. *face recognition*) osobe čiji je sami uređaj. Iz ovoga je primjera vidljiva važnost i tražena preciznost samog sustava kojim je potrebno utvrditi radi li se o vlasniku mobilnog uređaja ili nekoj drugoj osobi. Sličnih je primjera mnogo, a u budućnosti će ih vjerovatno biti i sve više što samo potiče dalji razvoj ovog područja tehnike. Potreba za kreiranjem računalnog sustava koji može komunicirati putem vizualnih informacija sa vanjskom okolinom te joj se sukladno dobivenim informacijama prilagoditi, javlja se u skoro svakoj grani tehnike koju moderno društvo koristi. Umjetna inteligencija igra važnu ulogu u kreiranju jednog takvog sustava pri čemu do izraženosti dolaze neuronske mreže kao srž cijelog sustava. Neuronske mreže postoje u raznim oblicima i veličinama, no svima je zajednički cilj predvidjeti neke rezultate na temelju zadanih ulaznih veličina tj. procesom učenja naučiti kako poznate veličine iz poznatog skupa podataka reproducirati na neki novi, nepoznati skup podataka. Jedan takav proces kreiranja, treniranja i testiranja neuronske mreže biti će glavna tema ovoga rada. No prije same mreže, potrebno je objasniti nekoliko stvari stvari vezanih uz sami problem određivanja karakterističnih točaka na ljudskom licu.

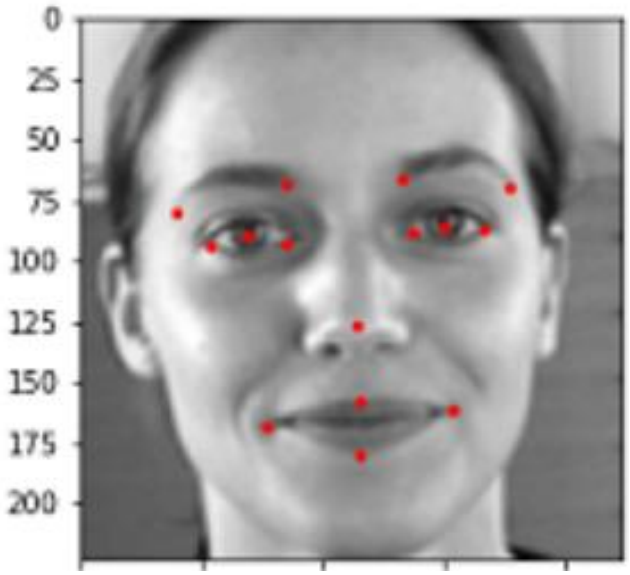
2. ZNAČAJKE LJUDSKOG LICA

Ljudsko je lice vrlo složen i geometrijski nepravilan dio ljudskoga tijela. Pomoću njega izražavamo vlastite emocije te nam ono omogućuje neverbalno komuniciranje s drugim ljudima. Sastoji se od nekoliko manjih dijelova tijela od kojih su među važnijima: čelo, oči, nos, obrazi, brada i usta. Svaki od navedenih dijelova lica ima svoju ulogu te je odgovoran za pojedine funkcije nužne za pravilan rad organizma. Lice je također prepuno mišićima i živacima pomoću kojih se neki dijelovi mogu pomicati i dovoditi u razne položaje što je najviše primjetljivo prilikom izražavanja neke emocije. Većina mišića lica reagira refleksno ovisno o tome kako se osjećamo tj. koje emocije prevladavaju u nama. Naravno, svaki se čovjek razlikuje od drugoga, pa će se tako i lica razlikovati od čovjeka do čovjeka, odnosno neće svi mišići lica reagirati jednako kod svakoga. Sve te gore navedene stvari treba uzeti u obzir kada se kreira program vezan uz ljudska lica ili njihovo prepoznavanje. Također, proporcije i oblik lica mogu se jako razlikovati ovisno o dobi, spolu ili trenutnom raspoloženju čovjeka zbog čega se treba osigurati relativna fleksibilnost programa kojem je zadaća uočavanje i određivanje karakterističnih točaka na licu.



Slika 1. Ljudsko lice

U ovom će se programu koristiti 15 točaka kako bi se definiralo lice tj. njegovi pojedini dijelovi. Točke prikazuju sljedeće dijelove lica: centar lijevog oka, centar desnog oka, unutarnji kut lijevog oka, vanjski kut lijevog oka, unutarnji kut desnog oka, vanjski kut desnog oka, unutarnji kraj lijeve obrve, vanjski kraj lijeve obrve, unutarnji kraj desne obrve, vanjski kraj desne obrve, vrh nosa, lijevi kraj usta, desni kraj usta, sredina gornje usne, sredina donje usne. Položaj točaka te njihove međusobne udaljenosti ovise o velikom broju parametara tj. različite su za svako pojedino lice. Iako je broj izabranih karakterističnih točaka relativno mali, dovoljan je kako bi se lice moglo prepoznati te odrediti njegov položaj na slici. Slika 2. prikazuje jedan primjer gdje bi se navedene točke nalazile u odnosu na sliku lica. Karakteristične točake označene su crvenim kružićima te svaka jednoznačno predstavlja jedan dio od gore navadenih dijelova lica.



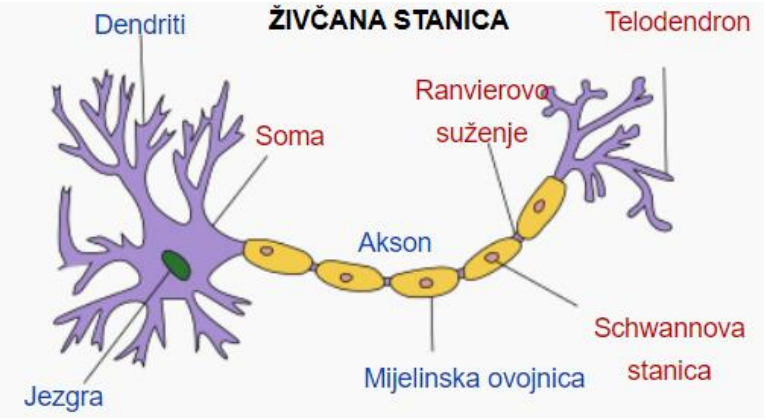
Slika 2. Primjer karakterističnih točaka na licu

3. UMJETNA INTELIGENCIJA U PODRUČJU OTKRIVANJA ZNAČAJKI LJUDSKOG LICA

Problem otkrivanja značajki na ljudskome licu najjednostavnije i najpreciznije se može riješiti primjenom umjetne inteligencije, konkretno uporabom neuronskih mreža. Neuronske su mreže dosegle vrlo visoke mogućnosti u poslijednjim desetljećima te se sve više koriste za rješavanje sličnih problema u tehnici.

3.1 Struktura i rad neuronskih mreža

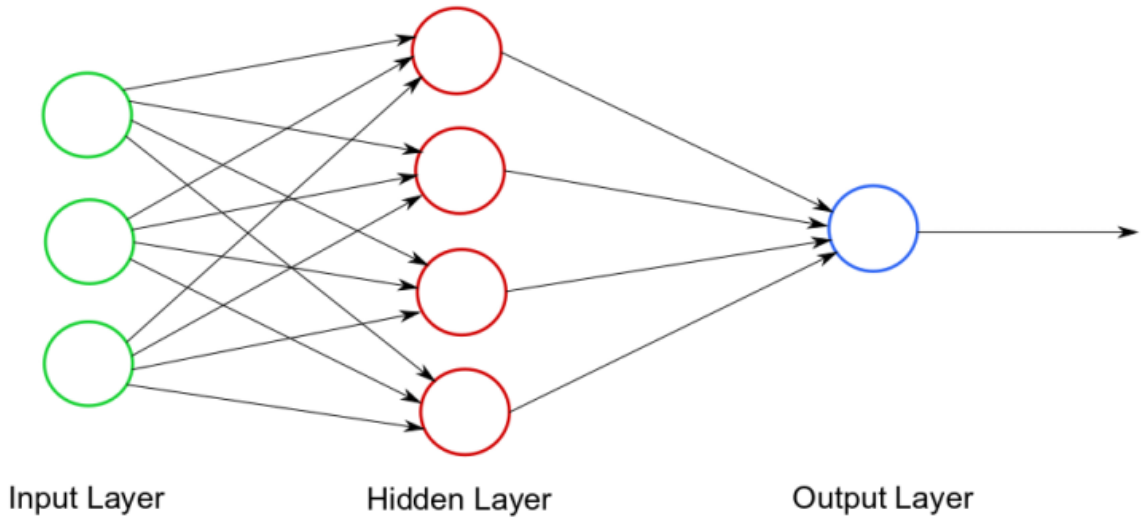
Neuronske su mreže napravljene po uzoru na sklop neurona u ljudskome mozgu. Neuroni se smatraju osnovne jedinice živčanog sustava te su jedan od najkompleksnijih dijelova organizma. Živčani sustav judskog tijela u prosjeku se sastoji od oko 86 milijardi međusobno povezanih neurona[1]. Neuroni međusobno komuniciraju pomoću sinapsi koje omogućuju prijenos električnog ili kemijskog signala sa jednog neurona na drugi ili do neke kranje stanice do koje se signal treba prenijeti. Tako se prenose informacije od mozga do bilo kojeg dijela tijela upravljanog živčanim sustavom. Svi dijelovi neurona vidljivi su na slici 3.



Slika 3. Neuron

Na sličan se način prenose informacije unutar umjetnih neuronskih mreža gdje se proces predaje informacija tj. signala između neurona u ljudskome mozgu pokušava što približnije rekreirati. U umjetnim neuronskim mrežama neuroni su zamijenjeni umjetnim neuronima koji

su povezani pomoću veza te tako tvore povezani skup neurona tj. neuronsku mrežu. Ulazni signali umjetnih neurona obično su realni brojevi koji prilikom dolaska sa jednog neurona na drugi mijenjaju svoj iznos. Na ulazu u neuron signali se množe težinama koje opisuju jakost sinapse. Potom se umnošci signala i težina zbrajaju te se dovode do ulaza aktivacijske funkcije. Nakon aktivacijske funkcije iz neurona izlazi promijenjeni signal koji se predaje sljedećem povezanom neuronu. Neuroni su u mreži podijeljeni u slojeve od kojih svaki sloj transformira iznos ulaznih signala te predaje takav promijenjeni signal sljedećem sloju u mreži sve dok signali ne dođu do posljednjeg sloja mreže tj. poprime konačnu vrijednost. Ovisno o broju slojeva, mreže mogu biti jednoslojne i višeslojne. Na slici 4. prikazana je jednostavna jednoslojna neuronska mreža sa jednim skrivenim slojem.



Slika 4. Primjer jednoslojne neuronske mreže

Kako bi neuronska mreža imala neku svrhu, odnosno davala neke rezultate, potrebno je prvo proći proces treniranja.. Neuronske mreže procesom treniranja mijenjaju iznose potrebnih težina kako bi minimalizirale razliku između izlaza iz posljednjeg sloja mreže i željenog izlaza kojem mreža treba težiti. Na taj se način postiže sličan efekt kao kod učenja u ljudskom mozgu. Vrijeme treniranja mreže može se razlikovati ovisno o ulaznom broju podataka, zadanom broju epoha kroz koje mreža mora proći te o samoj snazi računala koje simulira neuronsku mrežu.

Ukoliko želimo što bolje rezultate prilikom testiranja već istrenirane neuronske mreže, trebali bismo omogućiti što veći broj ulaznih podataka koji pokrivaju različite parametre koji bi

mogli utjecati na konačan rezultat testiranja. Jedini problem velikog broja ulaznih podataka te samim time većeg i dugotrajnijeg učenja mreže, može se javiti u obliku *Overfitting*-a. Overfitting ili "pretreniranje mreže" javlja se kada model mreže vrlo precizno obrađuje podatke prilikom treniranja mreže dok se prilikom testiranja na nepoznatim podacima ta preciznost vrlo bitno smanjuje te su rezultati lošiji od očekivanih. Tako na primjeru određivanja značajki na licu treba voditi računa kakvi su ulazni podaci kojima treniramo mrežu. Preveliki skup podataka dovest će do overfitting-a, dok s druge starne nije dovoljno uzeti malen broj slika na kojima ljudi izgledaju slično bez ikakvih pozadinskih smetnji ili nečega drugoga što bi moglo predstavljati problem. Ispravan odabir skupa podataka može se pratiti pomoću grafa koji pokazuje ovisnost greške o broju prijeđenih epoha u fazi učenja i validacije mreže. Takav je graf za ovaj primjer prikazan u poglavlju 5. Iapk, veći se problemi javljaju kada je skup ulaznih podataka vrlo skroman te ne predstavlja raznovrsnost podataka kako bi mreža mogla biti što fleksibilnija. Naravno, mreža trenirana na takvom skupu podatka davat će ispravne rezultate samo ako je nepoznati ulaz sličan onima korištenim za treniranje, gdje će, u ovom slučaju, pri drugačijim položajima lica ili udaljenostima od kamere rezultati biti lošiji od očekivanih. Nisu svi sustavi koji se bave ovim problemom jednako kompleksni, tako da se negdje možda neće nužno tražiti tolika preciznost mreže, ali kod većine sustava cilj je detektirati točke na licu bez obzira na položaj lica ili njegovu udaljenost.

3.2 Primjeri sustava detekcije lica korištenjem neuronskih mreža

Sveprisutnost tehnologije u svakodnevnom životu omogućila je korištenje neuronskih mreža kako bi se poboljšale ili olakšale pojedine radnje. Sustavi detekcije točaka na licu ili prepoznavanju pojedinih lica sve su zastupljeniji u okolini u kojoj boravimo ili uređajima koje koristimo. U slijedećem dijelu ovoga poglavlja biti će navedeni samo neki od primjera koji koriste takve sustave za raznu uporabu.

3.2.1 Sigurnosni sustav prilikom prelaska granice

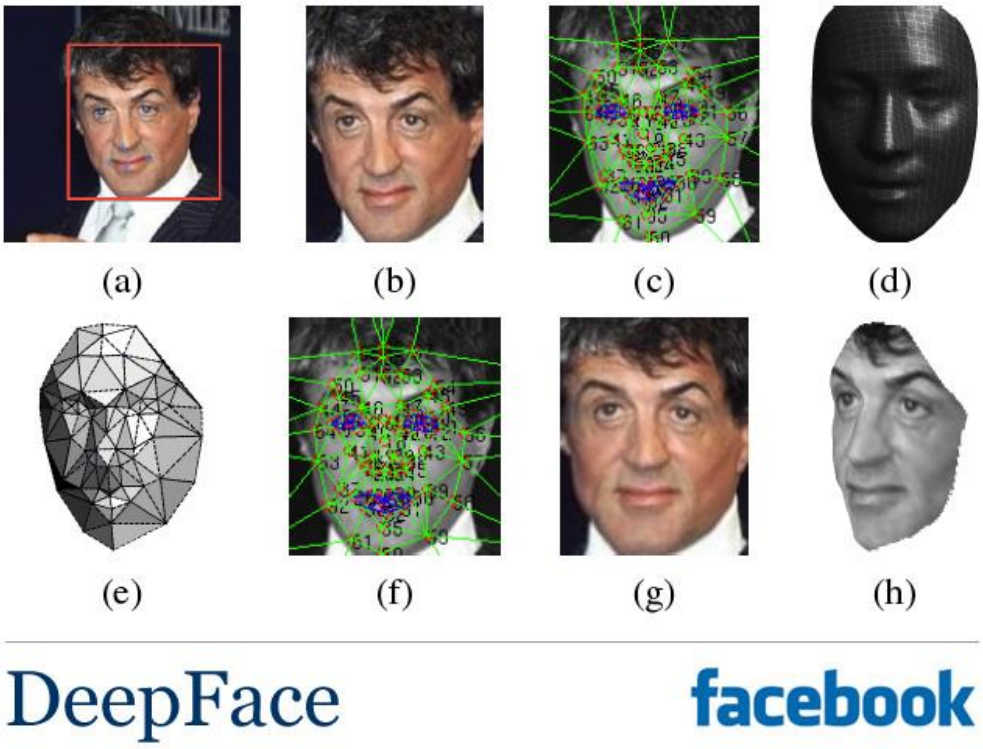
Australske pogranične službe i carina Novog Zelanda uveli su sustav SmartGate na nekoliko svojih aerodroma i lučnih prijelaza. SmartGate je sustav koji koristi tehnologiju

prepoznavanja lica kako bi verificirao osobu koja prelazi granicu te ju usporedio sa podacima vezanim uz čip na biometrijskoj putovnici te osobe [2]. Također, sustav kontrolira ilegalne pokušaje prelaska granice tj. osobe koje nemaju pravo prijeći granicu neće proći kroz ovaj sustav. Sličan je sustav implementiran u još nekoliko država, primjerice Kanadi, gdje je svaki aerodrom opremljen sustavom koji prepoznaje lica putnika te ih uspoređuje sa slikama na njihovim putovnicama. Taj je program prvi put postavljen u Vancouveru 2017. godine, a kroz 2018. i 2019. godinu i u ostatku zemlje.

Slika 5. SmartGate sustav u Australiji

3.2.2 Sustav DeepFace

DeepFace je sustav temeljen na dubokom učenju koji služi za prepoznavanje lica napravljen u istraživačkom centru Facebooka. Sustav propoznaje ljudska lica na slikama korištenjem neuronske mreže sa 9 slojeva i 120 milijuna veza između neurona [3]. DeepFace mreža je trenirana sa 4 milijuna slika koje su objavili korisnici Facebooka te pokazuje rezultate slične onima koje mogu dati ljudi. Točnost sustava da prepozna lice je 97, 35% $\pm$ 0, 25%, dok je točnost ljudi pri rješavanju istog problema 97, 53% čime je moguće da DeepFace ponekad radi bolje od čovjeka.



Slika 6. DeepFace sustav

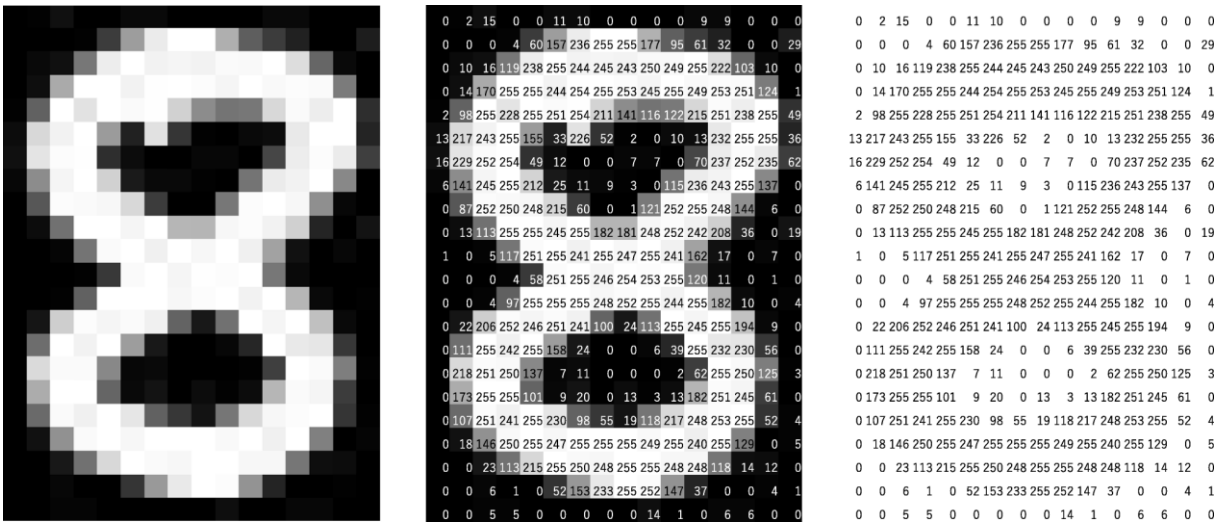
4. APLIKACIJA ZA OTKRIVANJE ZNAČAJKI NA LICU

Za izradu aplikacije koja prepoznaje i otkriva značajke na ljudskom licu koristio se programski jezik Python koji je postao vrlo jednostavan i moćan alat za kreiranje raznih programa i aplikacija. Pored samog Pythona, važno je i spomenuti PyTorch koji je knjižnica otvorenog koda te se koristi za svrhu izrade programa koji se bave dubokim učenjem te područjem računalnog vida i obrade prirodnog jezika [4]. PyTorch je primarno razvijen od strane Facebook-ovog laboratorija za istraživanje umjetne inteligencije te se koristi u raznim aplikacijama poput Tesla Autopilot, PyTorch Lightning, Catalyst i drugi, kao temeljni dio softvera zaduženog za umjetnu inteligenciju.

4.1 Skup podataka za treniranje neuronske mreže

Kako bi se neuronska mreža, koja je pokretač i centar cijele aplikacije, mogla istrenirati te naučiti prepoznavati značajke na ljudskom licu, potrebno je odabrati ispravan skup podataka. Ovdje se koristila zadana baza podataka koje je spremljena kao CSV datoteka koja sadrži vrijednosti odvojene zarezom za prikaz podataka. U datoteci se nalazi 7048 različitih podataka, redova, od kojih 4909 nije u potpunosti definirano tj. sadrže barem jednu neispravnu vrijednost u jednom ili više stupaca. Samo 2139 redaka sadrži sve potrebne i ispravne vrijednosti u svim stupcima, što se mora uzeti u obrzir prilikom unošenja podataka u neuronsku mrežu. Dakle, u realnosti na raspolaganju je 2139 slika na temelju kojih možemo istrenirati mrežu i napraviti aplikaciju za prepoznavanje značajki na licu. Svaka slika u CSV datoteci zapisana je pomoću vrijednosti svakog piksela na slici.

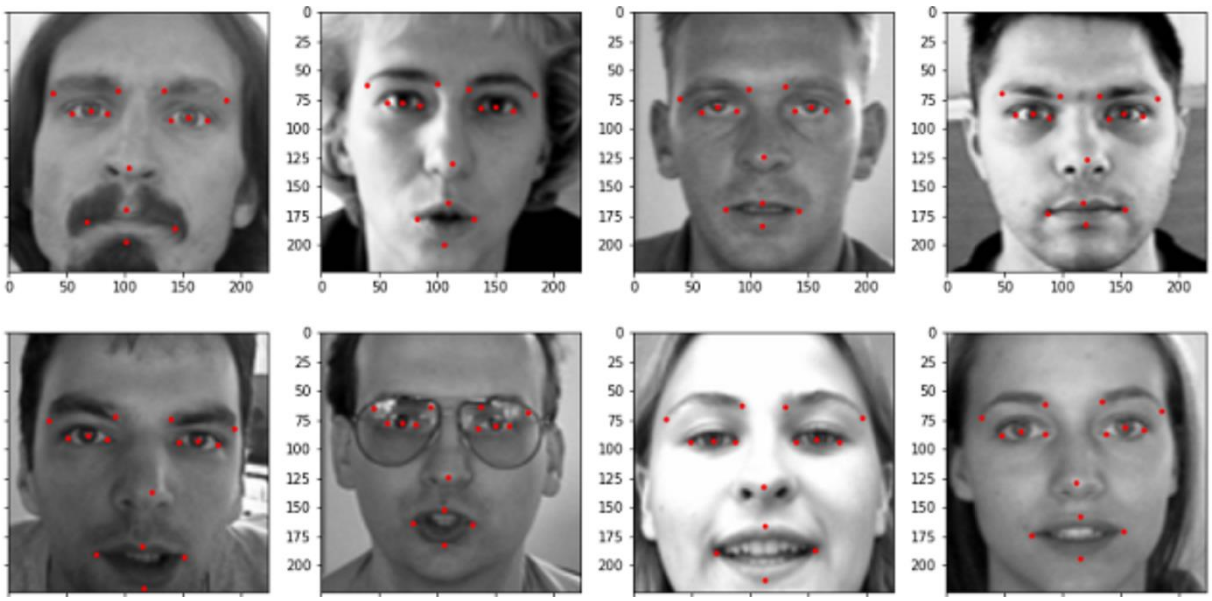
Slike su u nijansama sive boje, *engl. grayscale*, i dimenzija 96×96 piksela. Vrijednosti piksela na slikama zapisane su u obliku brojeva od 0-255 gdje broj 0 odgovara potpuno crnoj boji a 255 potpuno bijeloj. Svaka vrijednost piksela predstavljat će jedan broj u matrici čije su dimenzije jednake dimenzijama slike.



Slika 7. Primjer zapisivanja slike pomoću vrijednosti piksela

Na slici 7. prikazan je jedan jednostavan primjer kako se svaka slika može zapisati pomoću vrijednosti pojedinog piksela te tako spremiti u odgovarajuću matricu. Na isti su način spremljene slike u CSV datoteci s jedinom razlikom gdje vrijednosti piksela nisu raspoređene u matrici nego su zapisane kao niz piksela gdje svakih 96 piksela čini jedan redak matrice.

Pored samih slika potrebno je još i pridodati odgovarajuće kritične točke na svakoj slici. Kao što je spomenuto u poglavlju 2. ovdje se koristi 15 različitih točaka kako bi se definiralo lice. Uz svaku sliku u skupu podataka stoje i vrijednosti tih točaka. Točke su označene brojevima od 0 do 14 i svaka točka zapisana je kao x i y koordinata na slici. Za ulazni skup podataka kojim se trenira mreža te točke morale su biti ručno ucrtane tj. njihove koordinate je odredio čovjek što dovodi do problema kako se skup podataka uvećava jer je posao čovjeka vremenski sve zahtijevniji. Idealno je dati mreži što je više moguće podataka kako bi radila što bolje i preciznije, ali pitanje je isplati li se to vremenski za ljude koji tu mrežu kreiraju. Također, i sam proces treniranja vremenski može biti zahtijevan, pa stoga treba naći neku idealnu granicu gdje je ljudsko utrošeno vrijeme isplativo za svrhu za koju se neuronska mreža koristi. No, kako je ovdje baza podataka zadana o tome nije bilo prevelikih dilema, pa ulazi skup podataka izgleda ovako:



Slika 8. Primjer ulaznog skupa podataka

Kao što je vidljivo sa slike 8, primjeri slika kojima se trenira mreža mogu se dosta razlikovati. Ljudi na slikama raznih su dobnih skupina te se njihovi izrazi lica kao i proporcije razlikuju što će sigurno pospiješiti preciznost mreže prilikom testiranja na do sada ne viđenom skupu podataka. Također, neki ljudi na slikama nose naočale ili imaju brkove ili bradu što isto tako može predstavljati izazov za neuronsku mrežu ali s druge strane doprinosi raznovrsnosti skupa ulaznih podataka.

Neke od negativnih strana ovakvoga ulaznog skupa su okrenutost prema kameri na svakoj slici i relativno mala udaljenost lica od kamere. Problem kod toga što svi ljudi gledaju izravno u kamreu je takav da će prilikom okretanja glave mreža raditi vrlo loše te će vjerojatno pokazivati neispravne koordinate točaka. Što se tiče udaljenosti lica od kamere, na sličan će način rezultati biti lošiji od očekivanih jer će puno veći dio slike biti neka pozadina ili drugi dijelovi tijela dok će se sve točke morati dosta približiti jedne drugima ukoliko se udaljimo od kamere. Ako se pak čovjek približi kameri tako da mu se dio lica ne vidi ili je sakriven, neke točke ne bi se trebale uopće prikazivati. No mreža je trenirana tako da se na slici uvijek nalazi jednak broj točaka što će dovesti do situacije gdje će neke točke biti prikazane na mjestu gdje uopće ne bi trebale postojati. Još je jedan problem koji se može pojaviti a taj je naginjanje glave u jednu stranu. Sve su slike prikazane uspravno tj. ljudi na slikama stoje uspravno, no što ako je nečija glava nagnuta na jedno rame ili ako je slika možda naopačke? Sve te stvari

mogu doprinjeti smanjenju preciznosti mreže prilikom testiranja te bi ih trebalo uzeti u obzir ukoliko je zahtijevana preciznost programa velika.

4.2 Struktura aplikacije u Pythonu

Da bi se lakše mogao pratiti tok razvijanja aplikacije u Pythonu, poželjno je napraviti strukturu koja prikazuje na koji način aplikacija funkcionira i koje su sve datoteke potrebne za rad jedne takve aplikacije. Osnovni direktorij aplikacije podijeljen je u tri datoteke. To su redom: *Input* datoteka, u kojoj se nalaze svi podaci koji će biti korišteni kao ulaz u aplikaciju tj. neuronsku mrežu, *Outputs* datoteka, koja će prikazivati rezultate testiranja i treniranja mreže te sam napredak mreže tokom svoga rada i *Src* datoteka u kojoj se nalazi 6 Python skripti koje će upravljati cijelom aplikacijom. Shematski prikaz gore navedenih datoteka nalazi se na slici 9.

```
1.  ├──input
2.  │   └──facial-keypoints-detection
3.  │       │   IdLookupTable.csv
4.  │       │   SampleSubmission.csv
5.  │       │
6.  │       ├──test
7.  │       │       test.csv
8.  │       │
9.  │       └──training
10. │               training.csv
11. │
12. ├──outputs
13. │       loss.png
14. │        ...
15. │
16. └──src
17.     │   config.py
18.     │   dataset.py
19.     │   model.py
20.     │   test.py
21.     │   train.py
22.     │   utils.py
```

Slika 9. Shematski prikaz datoteka

Kao što je prikazano na slici 9, svaka od navedenih datoteka sadrži neki dio aplikacije potreban za funkcioniranje sustava u cjelosti ili praćenje rada sustava. Tako se u datoteci *Input* nalaze CSV podaci spomenuti u poglavlju 4.1 gdje je *test.csv* skup podataka kojim će se aplikacija moći testriati na nepoznatim ulaznim podacima tj. slikama. *training.csv* skup je podataka kojim će se mreža trenirati te su njegova svojstva, način zapisivanja te grafički prikaz detaljnije razrađeni u poglavlju 4.1. Pored podataka za treniranje i testiranje, nalaze se još dvije CSV datoteke, *IdLookupTable.csv* i *SampleSubmission.csv* u kojima su jednostavno zapisani nazivi koordinata točaka koje će se promatrati na licu te njima pridruženi jedinstveni redni brojevi.

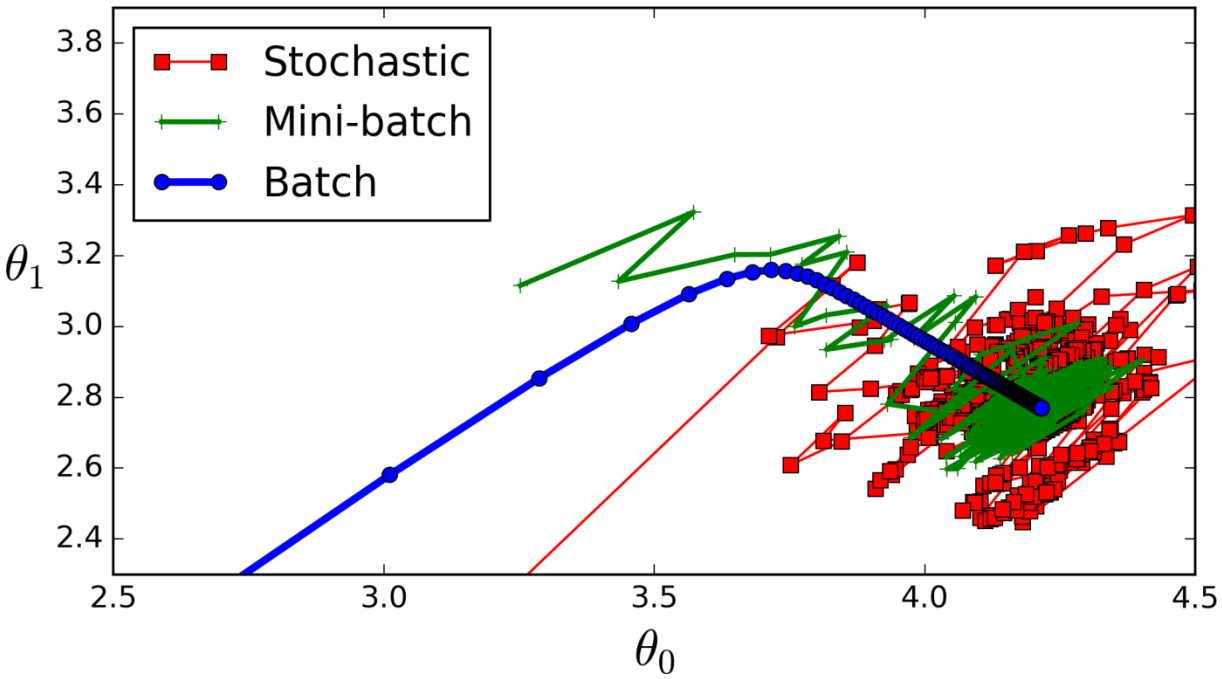
Datoteka *Outputs* sadržavat će nekoliko različitih stvari. U ovu će datoteku biti redom pohranjivane slike kako će mreža prolaziti epohama tijekom procesa treniranja. Na taj će se način moći pratiti kako mreža uči te koliko je zapravo precizna. Pored slika u fazi treniranja, također će se pohraniti graf sa prikazom smanjenja greške rada mreže tj. koliko je odtupanje pretpostavljenih točaka na licu od onih realnih napravljenih od strane čovjeka. O ovim će se pojmovima više govoriti u poglavlju 4.2.5.

Na kraju dolazimo do takozvane srži sustava odnosno do *Src* datoteke koja sadrži pokretače aplikacije. Kako je gore spomenuto, a i vidljivo sa slike 9, ovdje se nalaze Python skripte od kojih svaka ima vlastitu ulogu u radu aplikacije. Svaka od pojedinih skripti bit će detaljno objašnjena i razrađena u narednim poglavljima kako bi se stekao bolji uvid u tematiku.

4.2.1 Config.py skripta za kofiguraciju aplikacije

Ova je skritpa dosta kratka te sadrži malen dio sveukupnog koda, ali tu se također nalaze neki vrlo bitni parametri koji definiraju neuronsku mrežu. Prije svega, potrebno je zapisati gdje se na računalu nalazi datoteka koja sadrži ulazne podatke i datoteka gdje će se izlazni podaci pohranjivati. To je učinjeno vrlo jednostavno tako da se putanja direktorija gore spomenute *Input* datoteke i *Outputs* datoteke pohrani kao varijabla koja će kasnije biti korištena u nekim drugim funkcijama. Nakon toga, definirani su slijedeći parametri: veličina serije, *engl. batch size*, stopa učenja, broj epoha, uređaj koji će obrađivati podatke te omjer podataka koji će se koristiti za treniranje odnosno validaciju mreže.

Veličina serije u području neuronskih mreža odnosi se na broj ulaznih podataka koji prolaze kroz mrežu u procesu treniranja prilikom čega se parametri težina u mreži ne mijenjaju. To znači da će mreža učiti na određenom broju podataka nakon čijeg se prolaska kroz mrežu, vrijednosti težina mijenjaju te se mreža prilagođava sljedećoj seriji podataka tj. uči. Ovaj je pristup koristan prilikom postupka treniranja mreža sa velikim brojem ulaznih podataka. Poželjno je podijeliti te podatke na manje serije kako bi se smanjio intenzitet korištenja memorije u računalu. Manje serije znače manji broj ulaznih podataka pa je prosječna iskorištenost memorije manja nego kod treniranja mreže svim podacima odjednom. Negativna strana ovoga pristupa može se pojaviti u slučaju premale veličine serije gdje će gradijent funkcije greške biti neprecizan tj. greška se neće kontinuirano smanjivati.

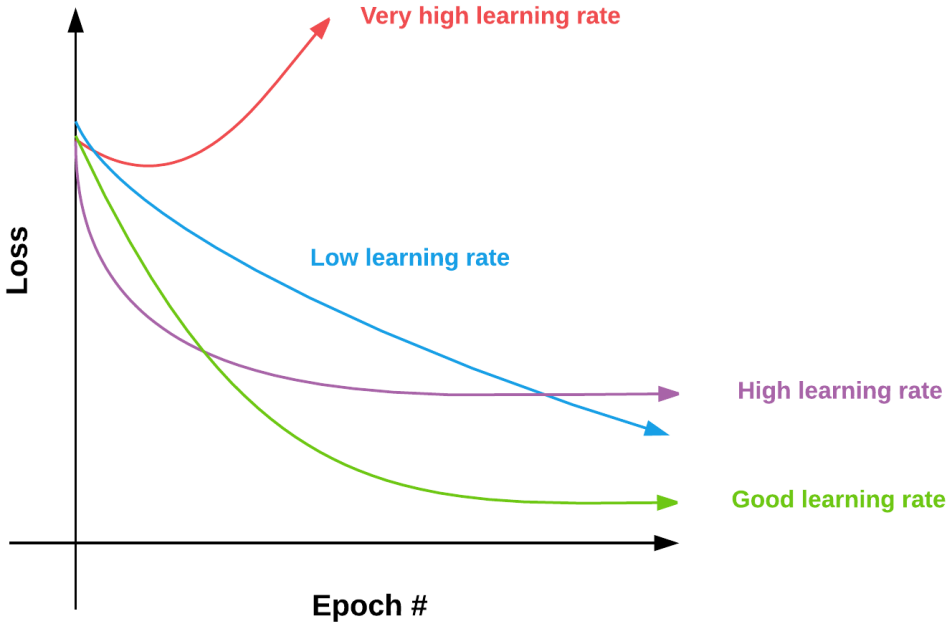


Slika 10. Gradijent funkcije greške ovisno o veličini serije

Na slici 10. prikazane su tri različite veličine serije te kako one utječu na gradijent. Plavom je bojom označena veličina serije u kojoj svi podaci za treniranje prolaze kroz mrežu u jednom koraku, odnosno veličina serije jednaka je veličini ulaznih podataka. Ovakav pristup zahtijeva manji broj iteracije prilikom računanja gradijenta pa je sustav efikasniji, također frekvencija kojom se težine ažuriraju često dovodi do stabilne i brze konvergencije u nekim problemima. Brza konvergencija nije nužno dobra u svakom slučaju jer može nametnuti parametre sustava koji možda nisu optimalni tj. ako gradijent prerano krene konvergirati može se naći u nekom lokalnom minimumu gdje će greška biti veća nego je očekivano. Zelenom je bojom na slici 10. pak označena veličina serije gdje su ulazni podaci podijeljeni na više serija. Takav je

slučaj i u ovoj aplikaciji, gdje je 2139 slika podijeljeno u serije veličine 256 slika. Vidljivo je kako gradijent više fluktuira te nema tako stabilnu i brzu konvergenciju kao kod krivulje označene plavom bojom. Na taj se način izbjegavaju mogućnosti ulaska u lokalne minimume pa je greška najčešće dobro minimizirana. Treća je mogućnost podijeliti ulazne podatke tako da je veličina serije jednaka 1 tj. da se nakon svakog ulaznog podatka ažuriraju vrijednosti težina. Takva se metoda zove stohastički gradijentni spust. Na slici 10. označena je crvenom bojom. Neke prednosti korištenja ove metode mogu biti jednostavnost primjene i razumijevanja na koji se način proces učenja odvija u mreži, brza frekvencija mijenjanja težina može dovesti do prilično brzog učenja u nekim problemima i mogućnost konvergiranja u lokalni minimum jako je mala. Problemi su pak velika računalna inzenzivnost koja dovodi do duljeg vremena potrebnog u procesu učenja, velika fluktuacija gradijenta može rezultirati velikoj varijaciji pogreške tokom cijelog procesa učenja ili čak dovesti gradijent u ne optimalan položaj zbog vrlo skokovite promjene parametara.

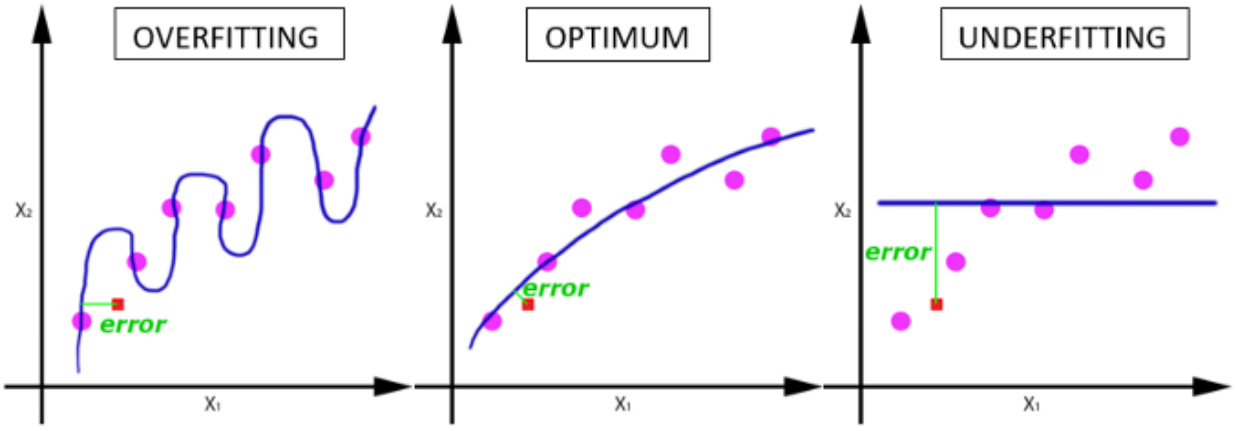
Stopa učenja, *engl. learning rate*, je parametar koji kontrolira koliko će se model mreže promijeniti ovisno o procjenjenoj grešci svkai put kada se vrijednosti težina ažuriraju. U ovoj je aplikaciji stopa učenja jednaka 0,0001. Povećanjem stope učenja omogućuje se brže učenje modela mreže ali s mogućnošću postizanja nedovoljno optimalnih težina. Smanjenjem stope učenja postižu se optimalnije vrijednosti težina ali je proces učenja sporiji i zahtijeva više epoha kako bi se te težine postigle.



Slika 11. Ovisnost funkcije greške o stopi učenja

Kao što slika 11. prikazuje, odabir stope učenja imat će velik utjecaj na rad mreže te ponašanje funkcije greške. Nažalost, ne postoji neki postupak kojim bi se dobar iznos stope učenja mogao odmah odrediti već je potrebno provesti nekoliko testiranja sa različitim vrijednostima te izabrati najbolju.

Broj epoha je broj koliko puta cijelokupni skup podataka prođe jednom kroz mrežu. U ovoj je aplikaciji broj epoha odabran kao vrijednost 300, što znači da svih 2139 slika 300 puta prolazi kroz mrežu tijekom faze učenja. Različiti brojevi epoha također imaju za posljedicu drugačije ponašanje mreže. Na slici 12. prikazane su moguće varijante odabira broja epoha. Vidljivo je kako će prevelik broj epoha (lijevi graf na slici 12.) uzrokovati mala odstupanja na nekim mjestima gdje su vrijednosti poznate odnosno na vrijednostima na kojima je mreža trenirana ali će ta odstupanja biti puno veća na mjestima gdje te vrijednosti nisu poznate tj. prilikom procesa testiranja na neviđenim podacima. Premali će pak broj epoha biti nedovoljan da minimizira odstupanja tj. iznosi pogrešaka bit će vrlo veliki na pojedinim mjestima.



Slika 12. Utjecaj broja epoha na učenje neuronske mreže

Omjer podataka kojim se mreža trenira i koji se koristi za validaciju uzet je kao 0,2, što znači da se 80% slika koristi za treniranje a ostatak za validaciju. Uređaj koji će obrađivati podatke te računati potrebne vrijednosti je grafička kartica jer se aplikacija većinom bavi vizualnim podacima tj. slikama. Ukoliko to nije moguće, procesor će preuzeti tu ulogu.

4.2.2 Utils.py skripta

U ovoj su skripti napravljene tri funkcije koje će omogućiti lakšu vizualizaciju i iscrtavanje kritičnih točaka na licu u pojedinim fazama aplikacije. Prva je funkcija zadužena za crtanje predviđenih točaka koje je mreža dala kao izlaz nakon određenog broja epoha u fazi treniranja i zadanih točaka kojima mreža treba težiti. Dakle, pomoću ove funkcije, svakih će nekoliko epoha slika na kojoj su ucrtane predviđene točke i zadane točke biti pohranjena u *Outputs* datoteku kako bi se moglo pratiti ponašanje mreže tijekom faze treniranja. Funkcija ovisi o četiri parametra. To su lista slika iz jedne serije, od kojih će se uzeti samo prva slika kao primjer, koordinate predviđenih točaka, koordinate zadanih točaka te broj epoha tj. koliko se često sprema zadana slika u datoteku. Na taj se način omogućuje praćenje rada mreže kako ona prolazi kroz epohe te se može vidjeti jesu li neki od odabranih parametara iz prethodnog poglavlja ispravni ili ne.

Druga je funkcija slična prvoj sa razlikom u tome što će ova funkcija iscrtavati predviđene točke na licu u procesu testiranja mreže na nepoznatim slikama. Ovdje su potrebna samo dva parametra kao ulaz u funkciju. To su lista slika koje želimo testirati i lista koordinata točaka koje je mreža predvidjela kao odgovarajuće točke na licu. Izlaz iz ove funkcije također će biti spremljen u *Outputs* datoteku kao slika, samo što će se ovdje na istoj slici prikazati 9 slika lica sa predviđenim točkama.

Posljednja funkcija služi kao provjera jesu li podaci kojima treniramo mrežu uistinu ispravni. Jedni parametar o kojem funkcija ovisi je cjelokupni skup podataka kojim se mreža trenira. Ova će funkcija prije početka treniranja prikazati nekoliko slika sa zadanim točkama na licu što predstavlja podatke koje će sami model mreže vidjeti. To je dobar indikator jesu li ulazni podaci ispravno napravljeni te hoće li se mreža zaista trenirati na podacima koje očekujemo.

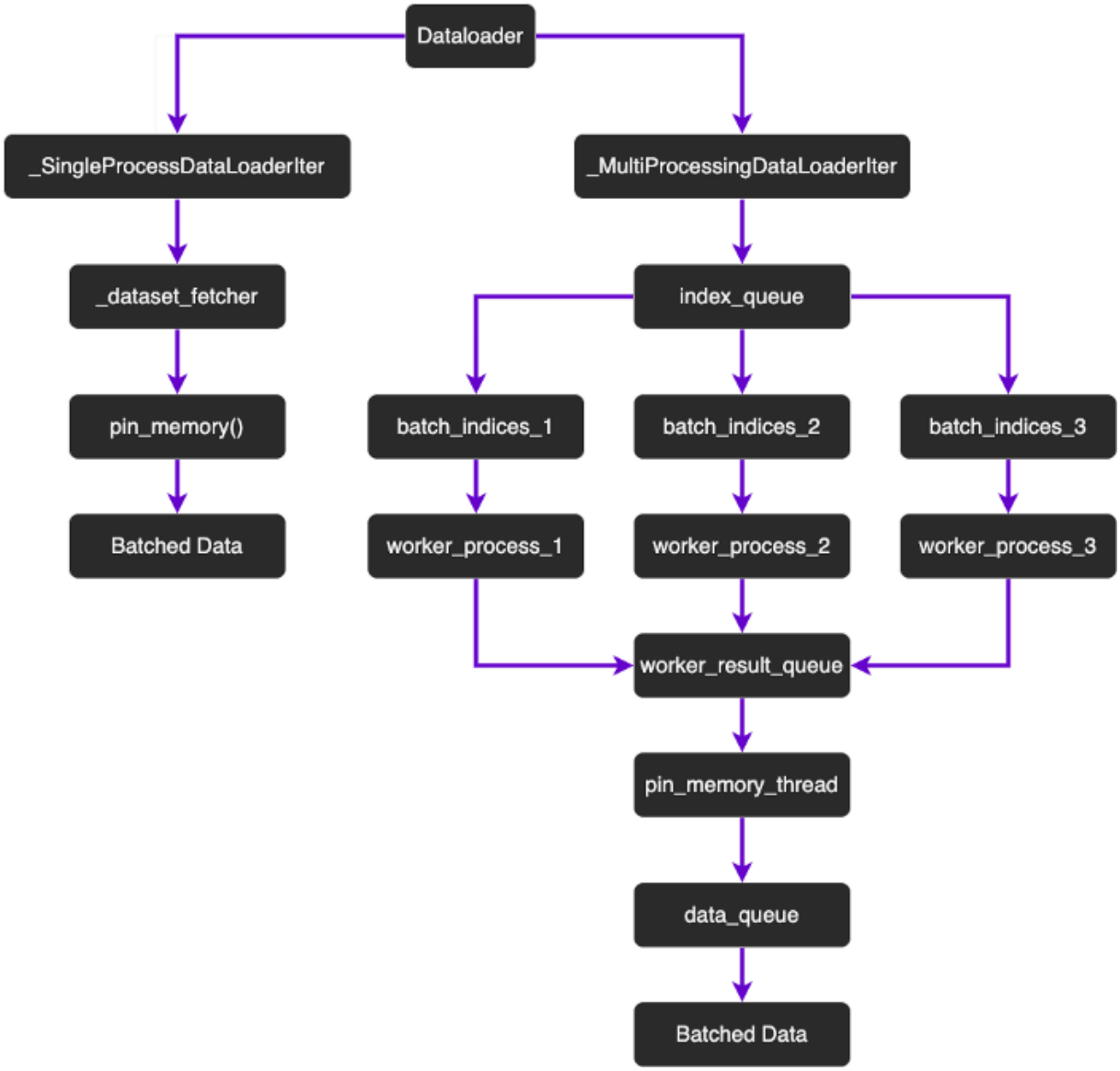
Sve su gore navedene funkcije relativno jednostavne te lagane za razumijeti. Iako niti jedna od njih nije nužna za rad aplikacije, poželjno ih je napraviti jer su dobar pokazatelj što se događa ispod površine. Ljudi kao vizualna bića puno jednostavnije razumiju stvari putem slika nego samo čitajući kod programa i pokušavajući zaključiti što se dešava.

4.2.3 Dataset.py skripta za pripremu ulaznih podataka

Ova je skripta zadužena za osigurano i ispravno postavljanje ulaznih podataka prilikom treniranja mreže. Prvi je korak raspodijeliti podatke koji se koriste za treniranje i one koji se koriste za validaciju. Kao što je navedeno u poglavlju 4.2.1, 80% podataka koristimo za treniranje mreže. Na početku je napravljena funkcija koja će te podatke raspodijeliti prikladno odabranom omjeru te ih pohraniti u dvije varijable kao liste slika sa odgovarajućim koordinatama točaka. Funkcija, dakle, ovisi o dva parametra tj. o omjeru podatka za treniranje i validaciju te o datoteci u kojoj su ulazni podaci spremljeni, u ovom slučaju *training.csv*. Nakon što su podaci pohranjeni u varijable, treba ih pretvoriti u odgovarajuće PyTorch tenzore kako bi odgovarali zahtijevanoj vrsti ulaza u model mreže. Ovdje će se to napraviti pomoću kreiranja klase unutar koje se nalaze funkcije zadužene za pretvorbu slika i koordinata točaka u takve tenzore.

Prvo je potrebno izvući i pohraniti vrijednosti piksela za svaku sliku te ih zapisati kao listu tekstualnih veličina, *engl. string*, unutar kojih se nalaze svi pikseli pojedine slike. Nakon toga, za svaku je sliku potrebno redimenzionirati odgovarajući *string* u format 96×96, odnosno 96 brojeva u 96 redaka, kako bi odgovarao dimenzijama slike. Zatim se taj string transformira u matricu piksela te se prilagodi vrijednostima *grayscale* formata slike. Pored toga potrebno je još izvući koordinate točaka za svaku sliku. Koordinate se pohranjuju u matricu gdje prvih 15 redaka odgovara koordinatama točaka za prvu sliku i tako redom. Na kraju se obje matrice pretvore u PyTorch tenzore te su spremne za sljedeće procese.

Posljednji je korak postaviti odgovarajuće parametre u gore navedenu funkciju te izlaze iz te funkcije postaviti kao vrijednosti spomenute klase čime se kreira novi objekt sa potrebnim vrijednostima varijabli spremnim za ulaz u neuronsku mrežu. Također, potrebno je još i postaviti dobivene varijable u ugrađenu PyTorch funkciju *DataLoader* koja omogućuje postupak iteracije unutar mreže. Na slici 13. prikazan je primjer kako funkcija *DataLoader* priprema ulazne podatke ako je broj radnika koji paralelno učitavaju te podatke jednak 3.

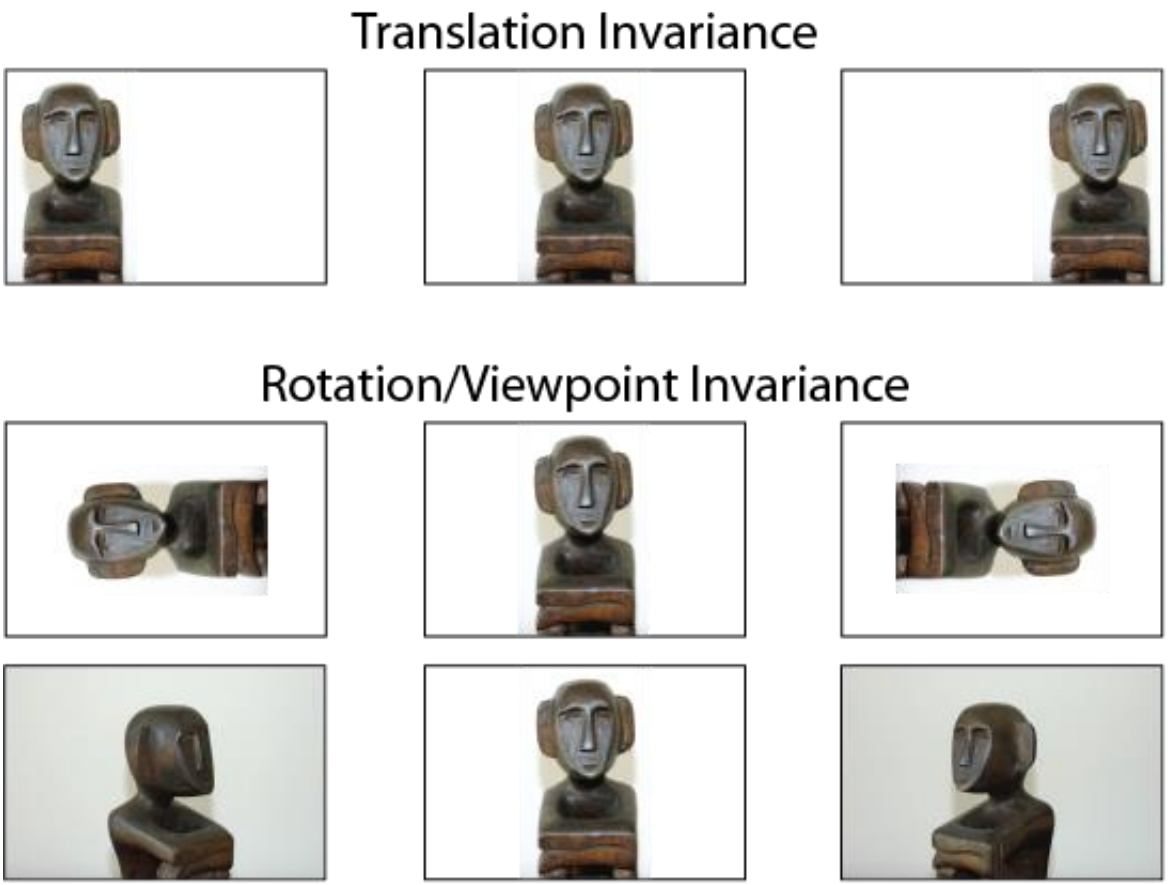


Slika 13. Primjer *DataLoader* funkcije

4.2.4 Model.py skripta za kreiranje neuronske mreže

U ovoj se skripti definira potpuna struktrua mreže. Mreža će se sastojati od tri konvolucijska sloja i jednog potpuno povezanog sloja. Konvolucijske su mreže vrlo pogodne za rješavanje vizualnih problema zbog primjene konvolucije. Konvolucija, u području neuronskih mreža, je linearna operacija koja uključuje množenje skupa težina sa ulaznim vrijednostima, slično kao kod tradicionalnih neuronskih mreža. Razlika je u tome što se u konvolucijskim mrežama množe matrice ulaznih varijabli i dvodimenzionalne matrice težina koje se nazivaju *filteri*. Rezultat množenja ulaza i filtera je skalarni produkt koji se dobiva zbrajanjem po svakom

elementu kako bi se dobila jedinstvena vrijednost. Filteri su uvijek manji nego ulazne vrijednosti kako bi se omogućilo množenje istih filtera ulaznim veličinama više puta, na različitim dijelovima ulaza. Filteri se sistematski primjenjuju na svakom dijelu ulaza koji se preklapa. Takva sistematska primjena istih filtera na cijeloj slici vrlo je korisna. Ako su filteri napravljeni kako bi prepoznali određenu značajku na slici, tada će sistematizacija primjene filtera omogućiti filterima pronalazak te značajke bilo gdje na slici. Ta se mogućnost često naziva invarijanca translacije [5].

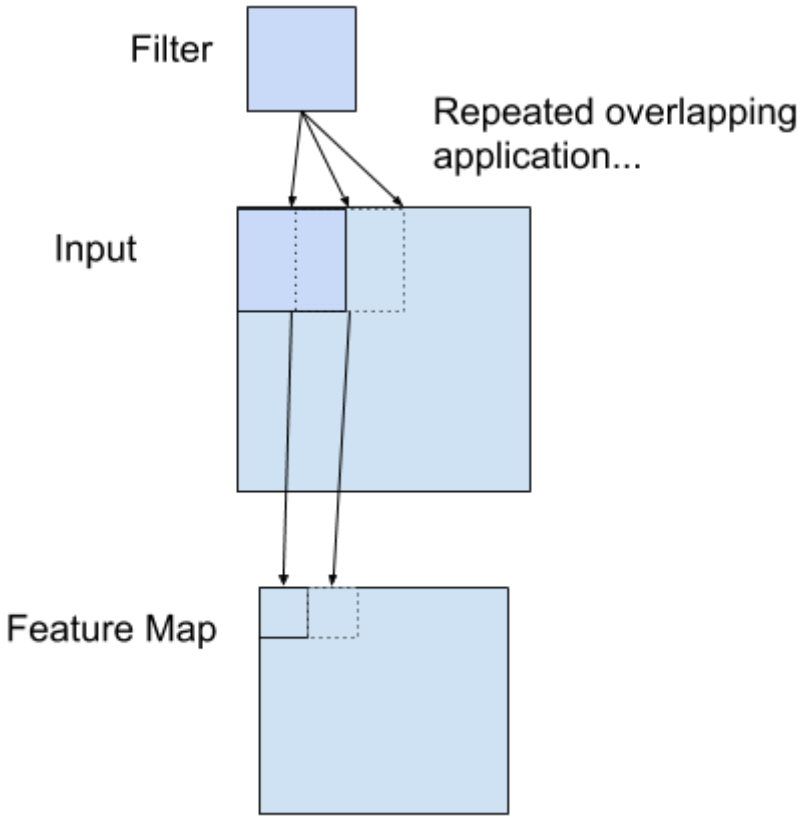


Slika 14. Invarijanca translacije i rotacije

Na slici 14. prikazane su invarijance translacije i rotacije kako bi jasnije predočili na što se ti pojmovi odnose. Invarijanca translacije, u ovome kontekstu, tako označava translaciju iste slike za određeni broj piksela u nekom smjeru. Kod aplikacija koje koriste gore navedene filtere, moguće je prepoznati određenu značajku bez obzira na kojem se položaju ona nalazi na slici, sve dok značajka zadrži iste proprocije te nije okrenuta oko neke osi.

Kao što je spomenuto, sklarani produkt dat će jednu vrijednost kada se pomnože ulazne vrijednosti i filteri. Kako su filteri primjenjeni više puta na ulaznim vrijednostima, rezultat

svakog množenja dat će novu vrijednost, te će se sve vrijednosti moći zapisati kao dvodimenzionalna matrica koja sadrži sve dobivene vrijednosti nastale primjenom filtera te predstavlja filtraciju ulaznih vrijednosti. Dobivena matrica naziva se ″mapa značajki″. Postupak dobivanja mape značajki prikazan je na slici 15.

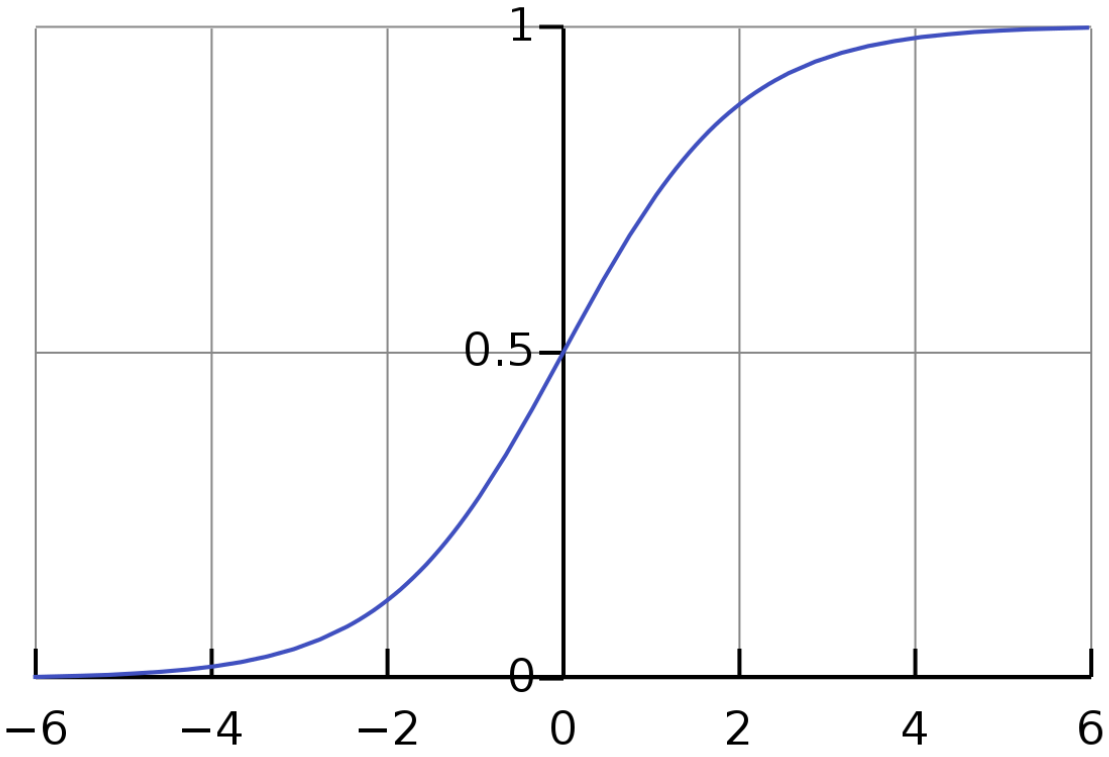


Slika 15. Prikaz ″mape značajki″

Nakon što je mapa značajki kreirana, njene se vrijednosti mogu provući kroz aktivacijsku funkciju sloja. U ovoj je aplikaciji odabrana aktivacijska funkcija svakog konvolucijskog sloja ReL, *engl. rectified linear function*. ReL je linearna funkcija koja će vratiti ulazne vrijednosti iste onima prije aktivacije, ukoliko su te vrijednosti veće od 0. U suprotnome, ReL funkcija kao rezultat dat će 0. Ova je funkcija vrlo česta u mnogim neuronskim mrežama radi jednostavnosti i lakoće treniranja mreže uz što se postižu i bolje performanse. Za razliku od nekih nelinearnih funkcija koje su se koristile 90-ih godina prošlog stoljeća poput *Sigmoid* i *Tanh* funkcija, ova funkcija smanjuje mogućnost zasićenja i nestajanja gradijenta. Sigmoid i Tanh funkcije davale su izlazne vrijednosti jednake 1 ako su ulazi veći ili jednaki 1, te vrijednosti -1 kod Tanh odnosno 0 kod Sigmoid, ako su vrijednosti ulaza manje od -1 tj. 0.

Na taj je način puno vrijednosti poprimilo izlaz 1, 0 ili -1 što je dovelo do zasićenja mreže. Te funkcije su osjetljive na promjene samo u područjima između graničnih vrijednosti pa se učenje mreže dosta otežalo. Pored mogućnosti zasićenja, javile su se i poteškoće pri dobivanju korisnih informacija o gradijentu. Kako greška prolazi mrežom te mijenja iznose težina, ona postaje sve manja i manja prolazeći kroz svaki sloj ovisno o aktivacijskoj funkciji. Kod Sigmoid i Tanh funkcija iznos greške vrlo brzo pada te sprječava mrežu u efektivnom učenju. Ti su se problemi smanjili uporabom ReL aktivacijske funkcije [6].

Prikaz gore navedenih aktivacijskih funkcija kao i njihove pripadajuće jednadžbe prikazane su na slikama 16., 17. i 18.

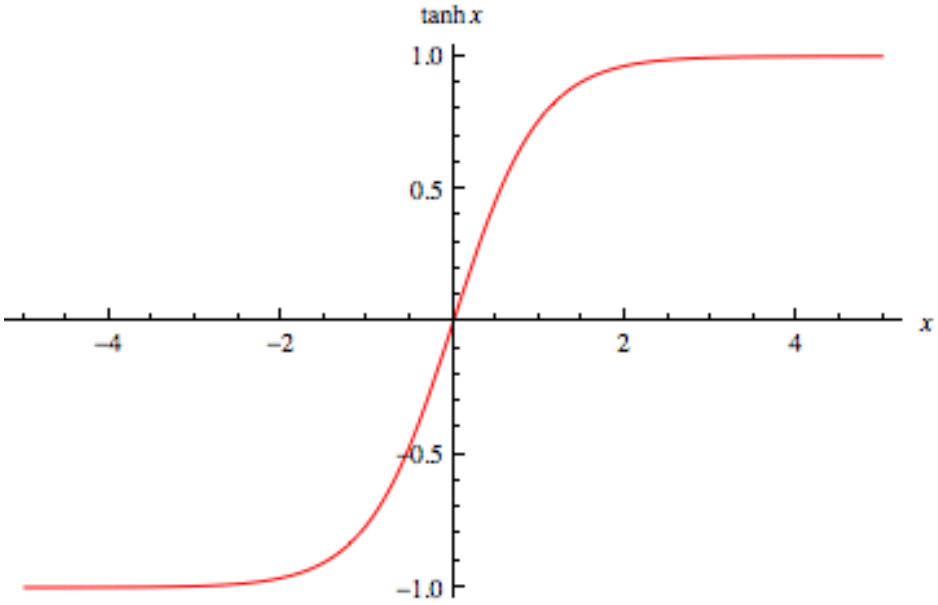


Slika 16. Sigmoid aktivacijska funkcija

Sigmoid funkcija često se označava slovom S radi njenog karakterističnog izgleda koji podjeća na to slovo. Jednadžba ove funkcije glasi:

$$S(x) = \frac{1}{1+e^{-x}} = \frac{e^{x}}{e^{x}+1}, \tag{1}$$

gdje je e baza prirodnog logaritma, a S vrijednost Sigmoid funkcije.

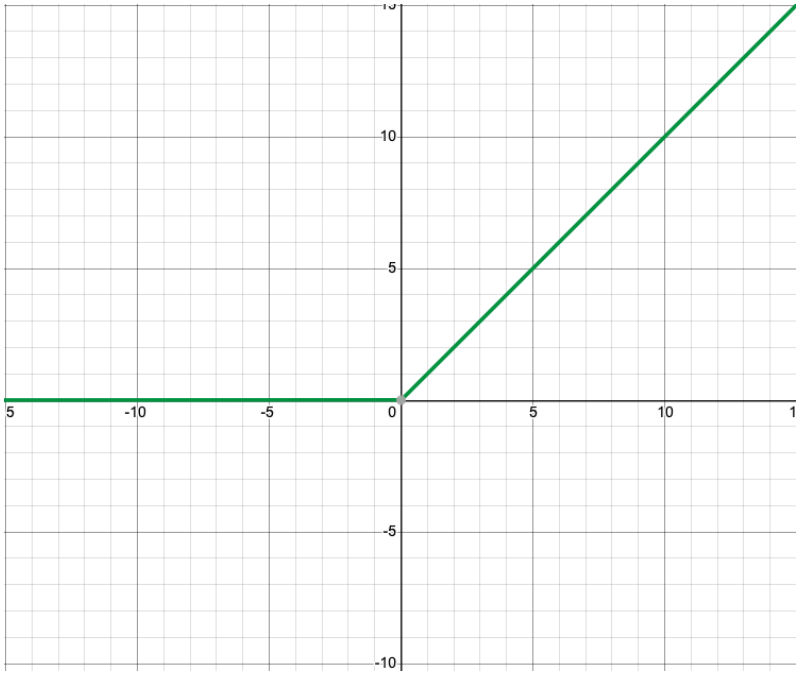


Slika 17. Tanh aktivacijska funkcija

Tanh, odnosno tangens hiperbolni, funkcija je koja priapada hiperboličkim funkcijama. Ona odgovara trigonometrijskoj funkciji tangens, samo definiranoj na hiperboli umjesto na kružnici. Jednadžba ove funkcije često označavane i kao $th(x)$ glasi:

$$\tanh(x) = th(x) = \frac{sh(x)}{ch(x)} = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}, \tag{2}$$

gdje su $sh(x)$ sinus hiperbolni a $ch(x)$ kosinus hiperbolni.

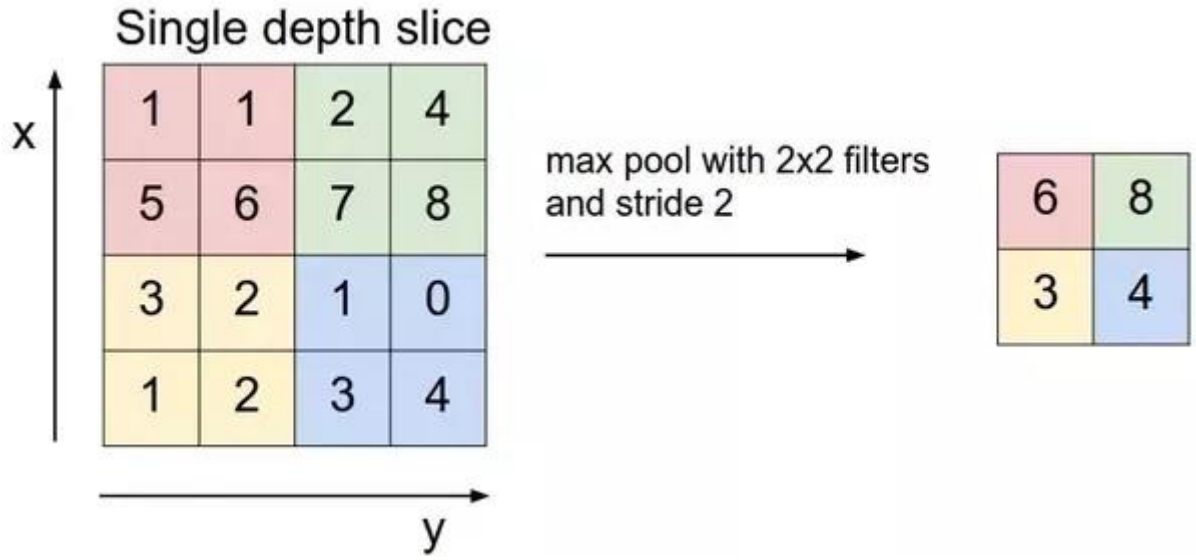


Slika 18. ReL aktivacijska funkcija

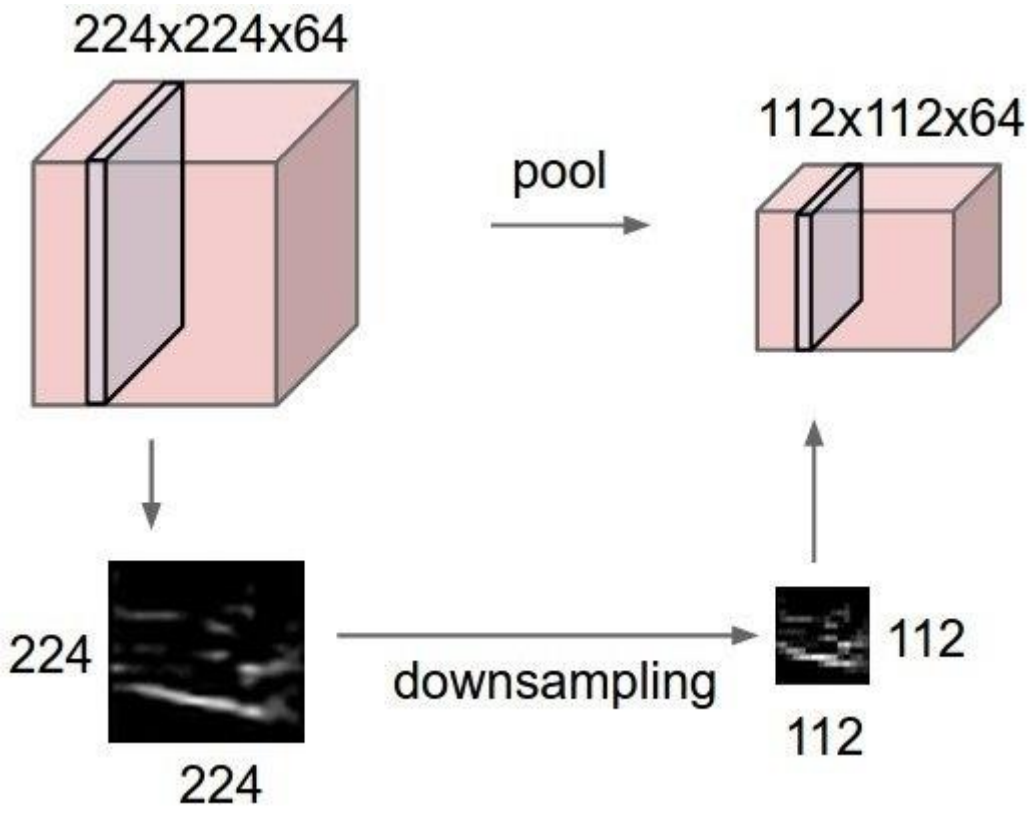
ReL funkcija najčešće se označava sa velikim slovom R. Iako postoji više varijanti ove funkcije, ova je najčešća te se koristi u zadanoj aplikaciji. Njena jednadžba glasi:

$$R(x) = \begin{cases} x, & x > 0 \\ 0, & x \leq 0 \end{cases}. \tag{3}$$

Problem koji se može javiti koristeći ovakvu shemu aplikacije tj. koristeći mapu značajki, je velika osjetljivost na lokaciju značajki u ulaznim podacima. To znači da će mali pomaci značajki na ulaznim slikama davati drugačije mape značajki. Kako bi se ta osjetljivost smanjila, koristi se *Max Pooling* operacija poslije svakog konvolucijskog sloja. Ta operacija reducira dimenzije ulaznih slika te uvodi pretpostavke o značajkama sadržanim u pojedinim dijelovima slike. Cilj ove operacije je sažeti ulazne podatke (*engl. down-sample*) na način da se pomoću filtera, najčešćih dimenzija 2×2 pikslea, cijela slika podijeli na dijelove. Svaki dio je veličine zadanog filtera te se iz svakog dijela uzima najveća vrijednost piksela koji je sadržan u tom dijelu [7]. Tako se reducira vrijeme računanja potrebnih parametara te se omogućava veća fleksibilnost mreže. Na slici 19. prikazan je način rada *Max Pooling* operacije dok je na slici 20. prikaza primjer *down-sample* slike.



Slika 19. Max Pooling operacija



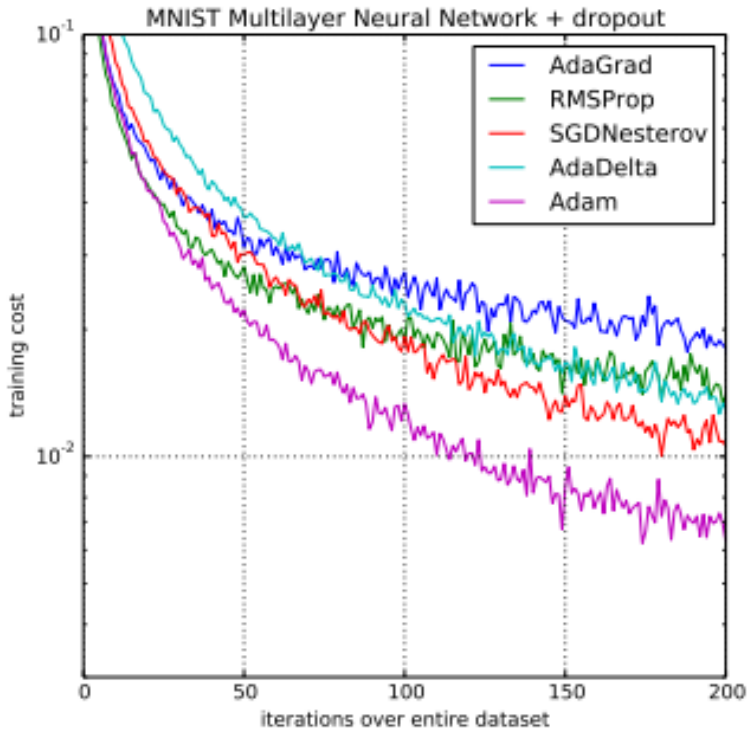
Slika 20. Primjer down-sample slike

Nakon što podaci prođu kroz sve konvolucijske slojeve, dolaze do potpuno povezanog sloja. Prije nego dosegnu taj sloj, potrebno je riješiti problem mogućeg *overfitting*-a. Overfitting se može javiti prilikom treniranja mreže na malom broju podataka gdje će mreža naučiti statističke šumove u tim podacima što dovodi do loših rezultata prilikom testiranja. Kako bi se to izbjeglo, koristi se metoda izbacivanja čvorova (*engl. dorpout*). Dropout je regulacijska metoda koja aproksimira treniranje velikog broja neuronskih mreža različitih građa. Tijekom treniranja mreže, određeni broj izlaznih vrijednosti iz prethodnih slojeva je ignoriran tj. privremeno izbačen iz mreže. Na taj se način postiže više šumova u procesu treniranja što prisiljava preostale čvorove u sloju da manje ili više preuzmu odgovornost o ulaznim podacima. Takav koncept gdje se slojevi mreže moraju prilagođavati zbog izbacivanja pojedinih čvorova, čini model mreže puno robusniji.

Kada su podaci prošli sve konvolucijske slojeve i Dropout metodu, dolaze do posljednjeg, potpuno povezanog, sloja mreže. Ovaj sloj služi kao klasifikacija rezultata proizašlog iz prethodnih slojeva. Ovdje se ″izravnavaju″ ulazne vrijednosti u vektor koji sadrži vjerojatnosti da se pojedina značajka nalazi negdje na slici. U ovoj se aplikaciji aktivacijska funkcija ovoga sloja ne koristi jer su potrebne sve koordinate svih 15 točaka za svaku ulaznu sliku.

4.2.5 Train.py skripta za treniranje mreže

Nakon što se kreirao model mreže, potrebno je omogućiti njegovo treniranje. Ovdje će se postaviti osnovne funkcije i zadati potrebni parametri kako bi se taj proces treniranja mogao kasnije provesti. Najprije se iz *model.py* te *dataset.py* skripti implementiraju model mreže te podaci pripremljeni za treniranje mreže. Zatim se model mreže inicijalizira te se postave parametri optimizacije i funckije greške. Za optimizaciju će se koristiti *Adam optimizer*. Adam je optimizacijski algoritam koji se koristi umjesto klasičnog stohastičkog gradijentnog spusta kako bi se težine ažurirale iterativno tokom treniranja [8]. Adam koristi prednosti dva poznata optimizacijska algoritma *AdaGrad* te *RMSProp*. AdaGrad je algoritam koji održava postojeću stopu učenja po svakoj težini, što poboljšava performanse u problemima tipa prirodnog govora i računalnog vida. RMSProp koristi stopu učenja koja se prilagođava na temelju prosjeka veličina gradijenta za svaku težinu te se tako postižu bolja rješenja za probleme sa šumovima na ulazu. Kombinirajući ove dvije prednosti, Adam daje bolje rezultate od navedenih optimizacijskih algoritama što je prikazano na slici 21.



Slika 21. Usporedba raznih optimizacijskih algoritama

Kao funkciju greške tj. funkciju koja će minimizirati odstupanja modela mreže od zadanih vrijednosti, korstit će se MSELoss funkcija. MSELoss (*engl. Mean Squared Error Loss*) je standardna funkcija koja se koristi kod ovakvih problema. Ona računa prosječnu vrijednost kvadrata odstupanja predviđenih točaka od zadanih, pri čemu se teži da ta vrijednost bude što bliža 0. Kvadriranjem odstupanja velike će greške imati puno veću vrijednost MSELoss funkcije nego manje greške.

Nakon što su definirani optimizacijski algoritam te funkcija greške, potrebno je napraviti još dvije funkcije unutar PyCharm programa kojima će se omogućiti proces treniranja i validacije mreže. Funkcija treniranja mreže ovist će o tri parametra: modelu mreže koji se implementirao iz *model.py* skripte, izlaznim podacima *Dataloader* funkcije dobivenim u *dataset.py* skripti te o samim ulaznim podacima. Izlazne vrijednosti ove funkcije bit će odstupanja predviđenih točaka na licu od onih zadanih nakon svake epohe. Tako će se proces treniranje moći postepeno pratiti.

Pored funkcije kojom će se model trenirati, treba kreirati funkciju koja će evaluirati koliko dobro model mreže uči. Taj se postupak naziva validacija te predstavlja nepristranu procjenu modela na temelju izdvojenog skupa ulaznih podataka na kojima mreža nije trenirala. Za funkciju kojom će se provesti validacija, potrebna su četiri parametra. Tri parametra ista su kao kod funkcije treniranja mreže s razlikom gdje su ulazni podaci sada drugačiji od onih na kojima se trenira odnosno, kao što je i ranije navedeno, oni čine 20% od cjelokupnog ulaznog skupa podataka. Pored ova tri parametra, koristi se i broj epoha koji je definiran u *config.py* skripti. Taj se broj ovdje koristi kako bi se svakih nekoliko epoha grafički prikazao proces validacije tj. odabrano je da se svakih 25 epoha kroz koje mreža prođe tokom treniranja, spremi po jedna slika na kojoj su prikazane zadane točke na licu te one koje je mreža izbacila kao izlaz. Ovaj se proces radi pomoću prve funkcije spomenute u *utils.py* skripti odnosno poglavlju 4.2.2. Izlazne vrijednosti funkcije validacije su slične onima u gornjoj funkciji.

Na kraju je još napisan dio koda kojim se pokreću ove dvije funkcije te se izlazne vrijednosti pohranjuju u liste te se grefički prikazuju u grafu. Graf predstavlja funkciju greške modela mreže te krivulju validacije mreže.

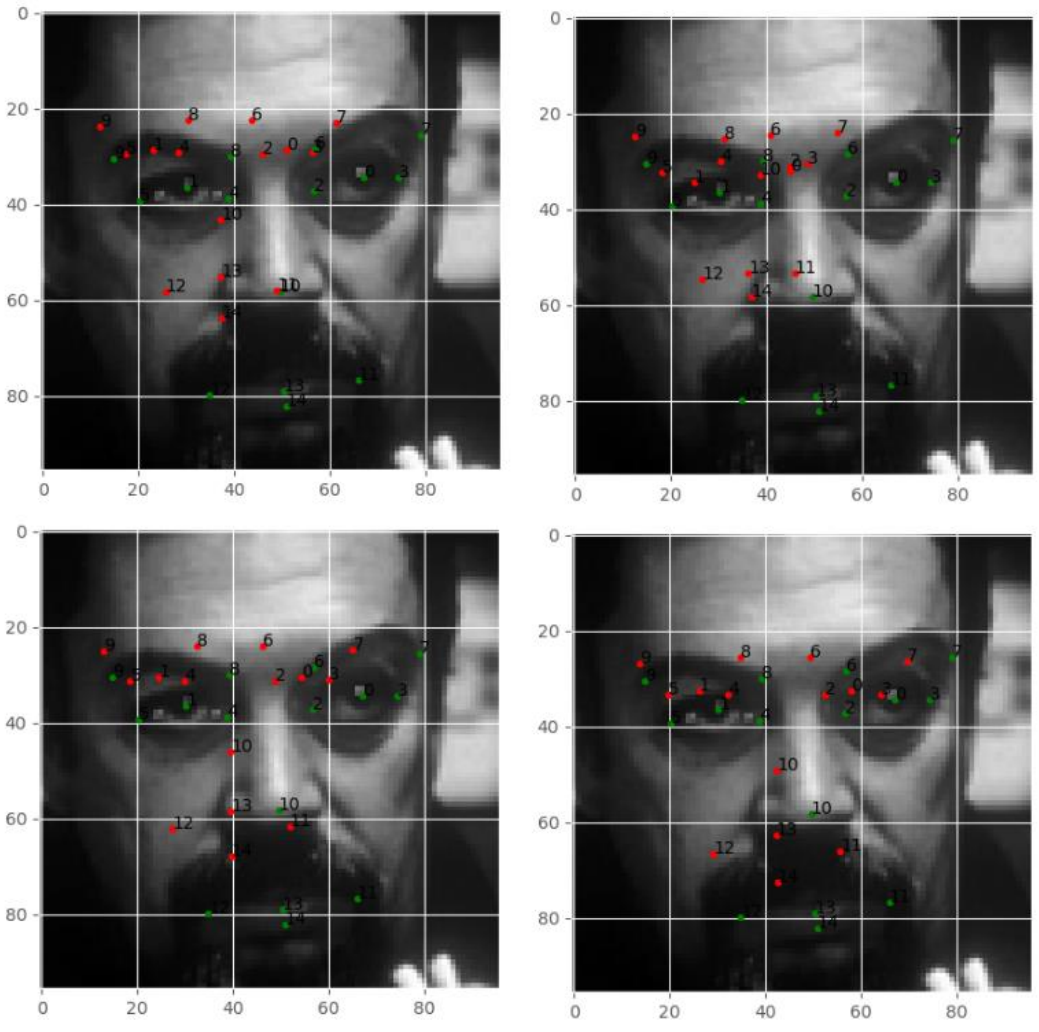
4.2.6 Test.py skripta za testiranje mreže

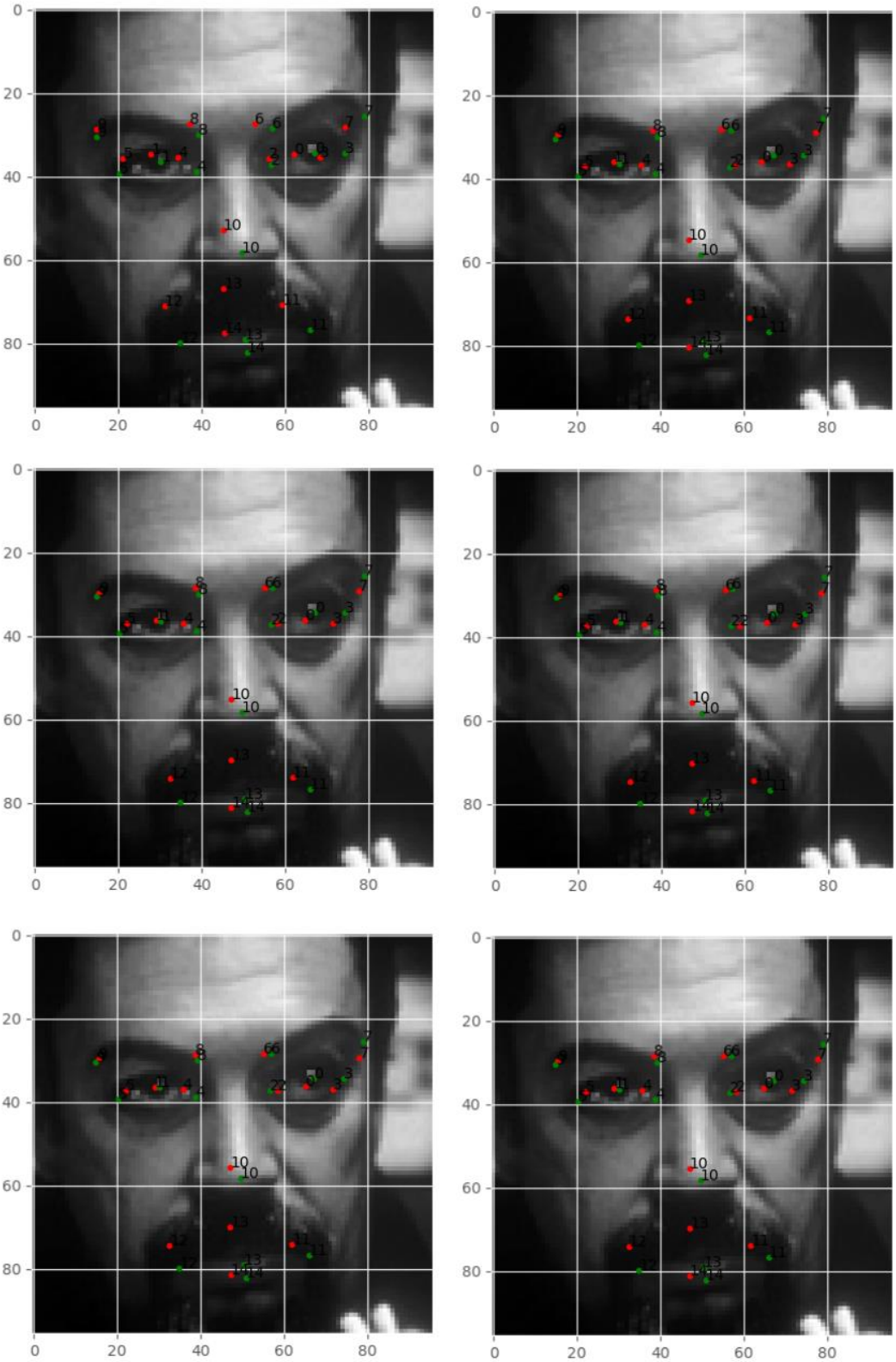
Posljednja skripta cijele aplikacije služi za testiranje mreže na nepoznatim ulaznim podacima. Ova je skripta relativno jednostavna jer jedino što treba napraviti je provući nepoznate slike kroz mrežu te prikazati točke koje je mreža izbacila kao izlaz. Za početak potrebno je inicijalizirati model mreže te učitai vrijednosti istreniranih težina. Zatim se podaci iz *test.csv* datoteke trebaju pripremiti tj. pohraniti u obliku matrice koja sadrži sve vrijednosti piksela na slikama slično kao u poglavlju 4.2.3. Nakon toga, slike se postave na ulaz mreže te mreža izbaci koordinate točaka koje pretpostavlja da odgovaraju definiranim točkama na licu. Te se točke na kraju prikažu pomoću druge funkcije definirane u poglavlju 4.2.2.

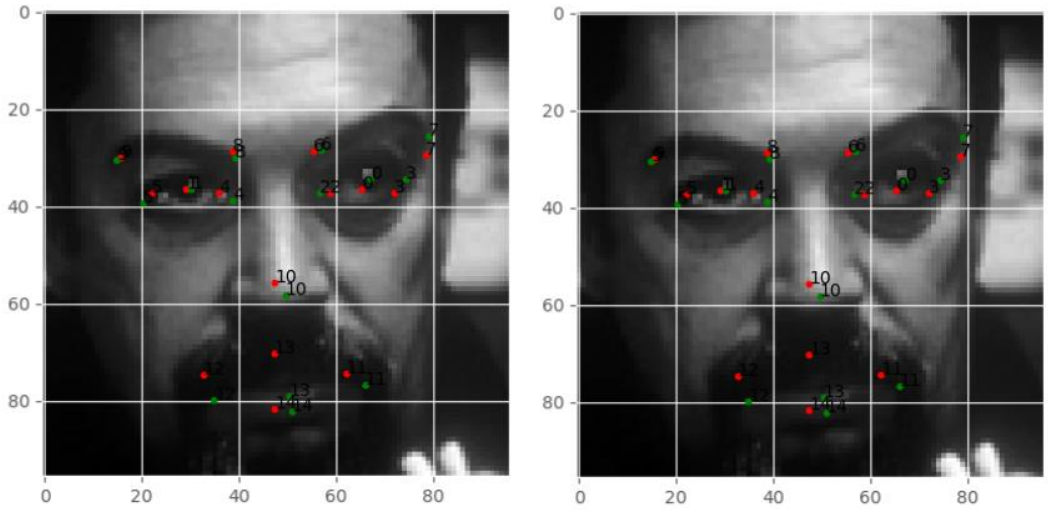
Pored testiranja mreže na slikama, mreža će biti testirana putem web kamere. Kako bi se to omogućilo potrebno je modificirati dio koda ove skripte. Najprije se spoji uređaj web kamere sa Python porgramom te se on učita u skriptu. Zatim se dimenzije slike sa kamere reduciraju na potrebne dimenzije na kojima je mreža trenirala što je u ovome slučaju 96×96 pikslea. Nakon toga slika se još pretvori u *grayscale* format te se pripremi za ulaz u mrežu. Svaka sličica (*engl. frame*), sprema se kao varijabla te se postavlja na ulaz mreže. Mreža u realnom vremenu prepoznaje značajke lica te prikazuje točke za koje pretpostavlja da odgovaraju tim značajkama. Usporedba ova dva načina testiranja mreže bit će prikazana u poglavlju 6.

5. TRENIRANJE NEURONSKE MREŽE

Nakon što je cijela aplikacija napisana u Python programu, protreno je pokrenuti odgovarajuće skripte i započeti rad aplikacije. Prvi je korak treniranje neuronske mreže koja će biti mozak aplikacije te će predviđati kritične točke na licu. Kako bi proces treniranja započeo, pokreće se *train.py* skripta. Prije nego samo treniranje započne, aktivira se treća funkcija iz *utils.py* skripte koje je opisana u poglavlju 4.2.2. Na ekranu će se dakle, prikazati slike onakve kakve će ih mreža vidjeti što služi kao posljednja provjera je li sve napravljeno kako treba. Kada se izađe iz prikaza slika, mreža kreće u fazu treniranja. Treniranje se vrši kroz 300 epoha kao što je zadano u *config.py* skripti te se svakih 25 epoha pohranjuje jedna slika na kojoj je vidljiva usporedba predviđenih točaka i onih zadanih, što je objašnjeno u poglavlju 4.2.2. Na slici 22. prikazani su rezultati treniranja mreže u obliku navedenih slika.

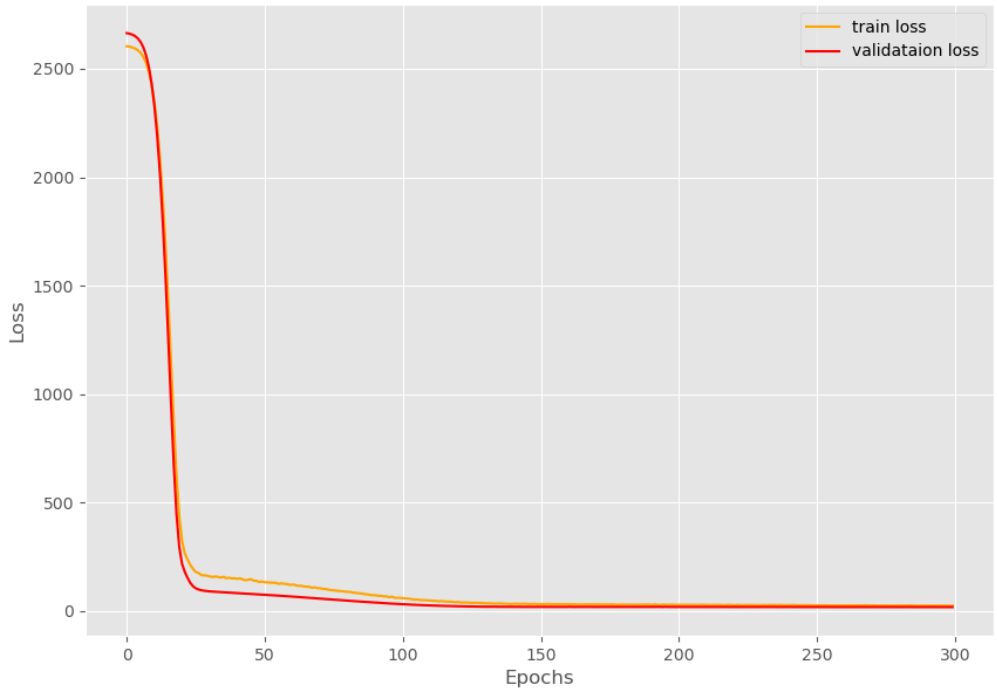






Slika 22. Prikaz treniranja neuronske mreže pomoću slika

Na gore prikazanoj slici crvenom su bojom označene predviđene točke, dok su zelenom bojom označene zadane točke kojima mreža teži. Vidljiv je dobar napredak mreže kroz epohe te se većina točaka nalazi vrlo blizu zadanih. Neke pak točke čine iznimku na ovoj fotografiji, npr. točka broj 13 koja označava sredinu gornje usne ne poklapa se sa zadanom točkom. Razlozi ovom odstupanju mogu biti razni, a vjerovatno je najveći utjecaj imao cjelokupni skup ulaznih podataka koji bi trebao biti raznovrsnijeg karaktera. Pored slike 22., proces treniranje mreže kroz epohe može se i grafički prikazati.



Slika 23. Ovisnost greške o broju epoha

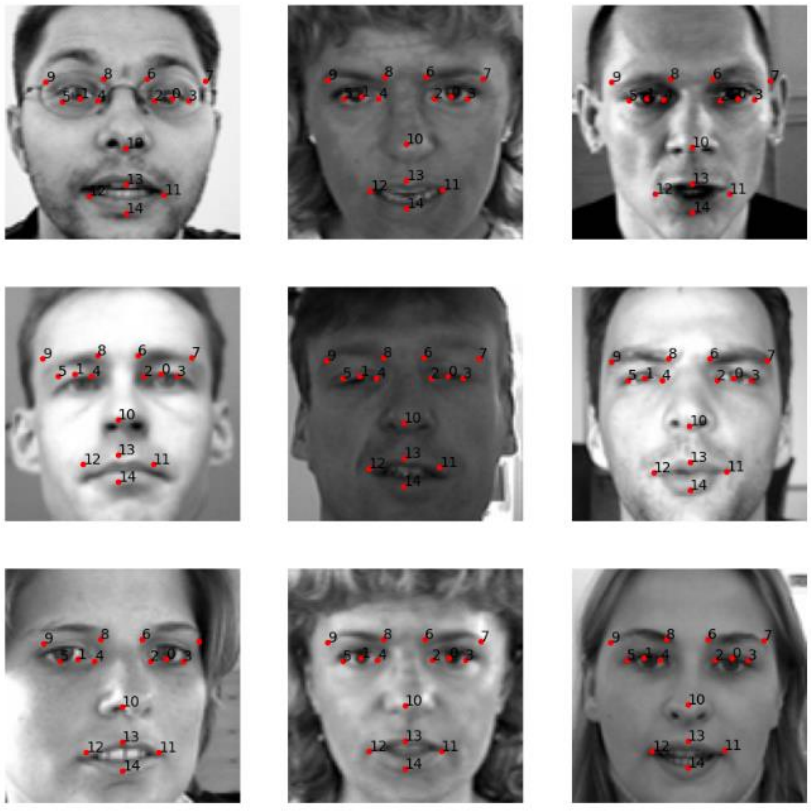
Na slici 23. prikazana je ovisnost veličine greške o broju epoha. Taj se prikaz vrlo često koristi u području neuronskih mreža te daje dobar uvid kako je mreža reagirala na zadane parametre treniranja. Vidljivo je kako ponašanje mreže nije bilo nepredvidljivo što znači da su nametnuti parametri relativno dobro pogođeni te nisu doveli do nekih od problema spomenutih u poglavlju 4.2.1. Također je prikazan veliki pad iznosa greške u prvih 25 epoha nakon čega on počinje sporije padati. Tu se očituje MSELoss funkcija greške koja kvadriranjem velikog iznosa greške brzo mijenja težine u mreži te se tako mreža u početku vrlo brzo mijenja. Nakon inicijalne brze promjene, greška nastavlja padati ali sporijim gradijentom. Nakon 300 epoha, razina je greške vrlo mala te bi mreža trebala prikazivati relativno prcizne točke na nepoznatim slikama sve dok su one slične ulaznom skupu.

6. TESTIRANJE NEURONSKE MREŽE

Nakon što je faza treniranja završila, aplikacija je spremna za rad. Koliko dobro aplikacija radi tj. koliko dobro neuronska mreža može prepoznati značajke na ljudskome licu, provjerit će se kroz dva testiranja. Kao što je navedeno u poglavlju 4.2.6., jedan postupak testiranja bit će pomoću nepoznatih slika sličnima onima koje su se koristile za treniranje, dok se u drugom postupku koristi web kamera kao ulaz. Rezultati oba postupka te njihova usporedba prikazani su u sljedećim poglavljima.

6.1. Testiranje na nepoznatim slikama

Pokretanjem skripte *test.py* model mreže počinje učitavati slike iz *test.csv* datoteke gdje se nalaze slike kojima će se mreža testirati. Slike su sličnog oblika onima koje je mreža koristila za treniranje pa bi rezultati ovoga testa trebali očekivano biti dobri. Kada se proces testiranja završi, druga funkcija u utils.py skripti pohranjuje sliku na kojoj se nalazi 9 rezultata ovoga testa.

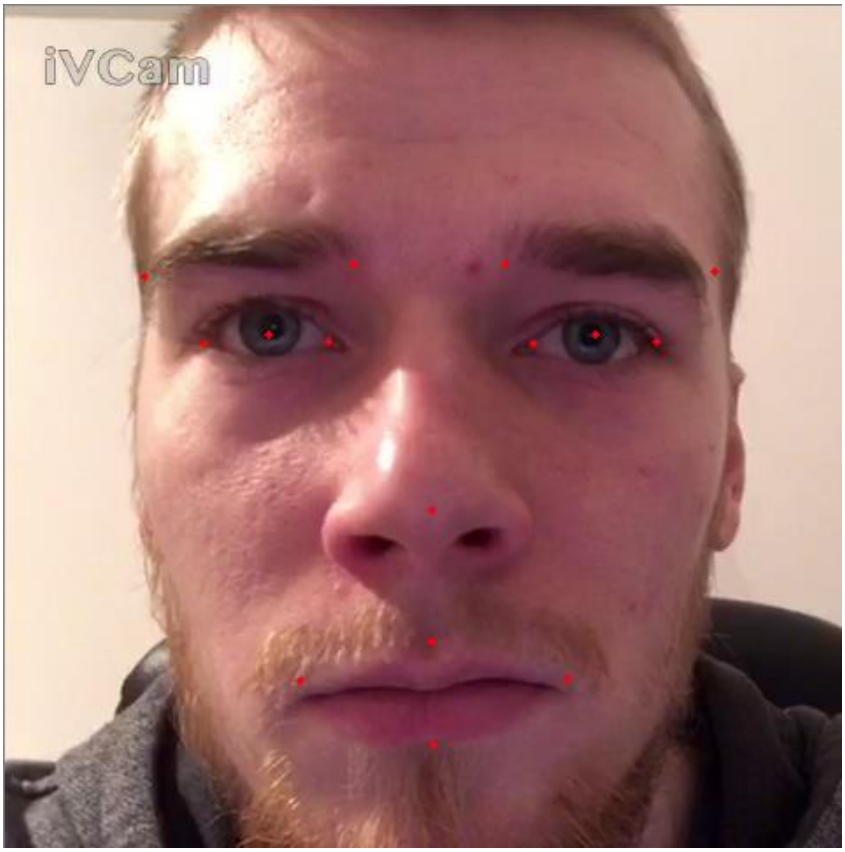


Slika 24. Rezultati testiranja na nepoznatim slikama

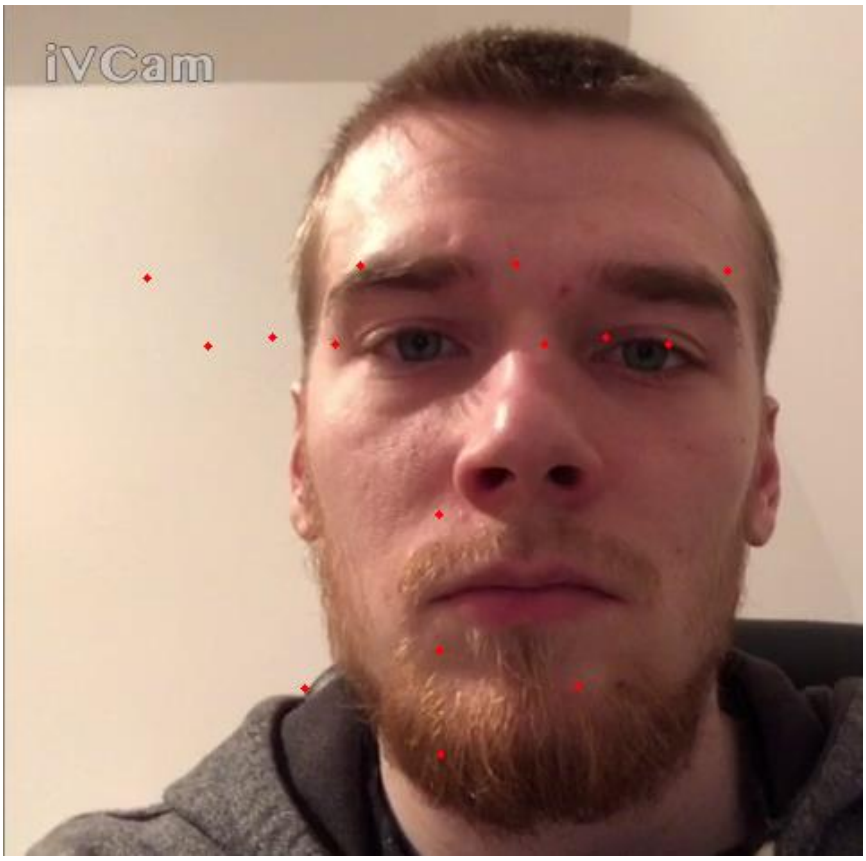
Slika 24. prikazuje pretpostavljene točke označene crvenom bojom na 9 različitih slika. Vidljivo je kako su rezultati ovoga testa vrlo zadovoljavajući te točke dobro predstavljaju karakteristike svakog lica. Naravno, postoje odstupanja nekih točaka na pojedinm slikama što ipak pokazuje neku mjeru greške ovako postavljenog i treniranog modela mreže. Unatoč tome, za ovakav tip slika ipak se može reći kako radi sa relativno visokom preciznosti te je prepoznavanje karakteristika lica uspješno savladano.

6.2. Testrianje pomoću web kamere

Kada je riječ o ovakvom načinu testiranja, nekoliko se stvari mora napomenuti prije predstavljanja samih rezultata. Najprije je potrebno istaknuti kako prilikom testiranja može doći do nekoliko problema. Prvi je problem mogućnost kašnjenja izlaznih podataka u odnosu na ulazne podatke. Kako se svaka sličica tokom rada web kamere sprema te ubacuje u model mreže, prilikom prevelikog broj sličica u sekundi *engl. Frames Per Second (FPS)*, može se dogoditi situacija da mreža ne stigne provući toliki broj sličica u sekundi pa dolazi do kašnjenja gdje se neka promjena na ulazu očituje tek nekoliko sekundi kasnije na izlazu. Taj bi se problem mogao riješiti na način da se npr. tek svaka druga ili svaka treća sličica uzmu kao ulaz u mrežu što će smanjiti opterećenost mreže. Pošto je web kamera koja se koristila u ovom testiranju namještena tako da vrijednost FPS-a bude jednaka 30, taj se problem izbjegao jer je računalo dovoljno brzo moglo obrađivati sve podatke. Pored toga, problem može predstavljati i način na koji je sama mreža trenirana. Sve su slike u ulaznom skupu podataka slike lica nekih osoba koje stoje ispred kamere tj. na svim slikama postoje sve kritične točke koje mreža treba označiti. Kako je mreža trenirana da uvijek izbacuje koordinate svih 15 točaka, ona će to uvijek i napraviti makar na slici neće biti ničega. Za rješavanje tog problema potrebno je postaviti neke aktivacijske funkcije na zadnjem sloju mreže koje bi ukoliko neka od točaka nije prepoznata, mogle jednostavno maknuti tu točku iz krajnjeg rezultata. Ovdje to nije napravljeno pa će se sve točke prikazivati neovisno o slici na web kameri. Još problema koji se mogu javiti, spomenuto je u poglavlju 4.1. pa i njih treba imati na umu prilikom tumačenja rezultata. Na slikma 25. i 26. prikazani su rezultati testiranja pomoću web kamere.



Slika 25. Testrianje web kamerom 1



Slika 26. Testiranje web kamerom 2

Na slici 25. vidljivo je kako prilikom imitiranja ulaznog skupa podataka za treniranje, aplikacija relativno dobro radi. Pretpostavljene točke većim dijelom odgovaraju karakteristikama lica te su otprilike pogodile gdje se lice nalazi. To je i očekivano jer se prilikom testiranja na sličnim slikama pokazalo da će mreža dati zadovoljavajuće rezultate kao u prethodnom poglavlju. No kod slike 26. počinju se isticati većina navedenih problema s kojima se ova aplikacija može susresti. Prvo je vidljivo da gotovo nijedna točka ne predstavlja karakteristiku lica koju bi trebala predstavljati. Udaljenost lica od kamere veoma je utjecala na rezultat koji mreža daje. Niti jedna od slika u skupu koji je služio za treniranje nije prikazivala ovakav položaj i udaljenost lica što je imalo za posljedicu vrlo malu fleksibilnost same mreže tj. koordinate točaka se ne mijenjaju za neki značajan iznos. Uz to, vidi se i problem gdje mreža prikazuje pretpostavljene točke na potpuno jednobojnom dijelu slike (lijevo od lica) gdje se ne nalazi ništa. Važno je i otkloniti sumnje utjecaja boje i dimenzija slika jer se prije ulaska u model mreže svaka slika smanji na dimenziju 96×96 piksela te pretvori u *grayscale* format kako bi odgovarala slikama u fazi treniranja. Na ovim je slikama prikazan ulaz onakav kakav izgleda prije tih operacija radi lakše vizualizacije rezultata. Na kraju su dodane koordinate točaka pomnožene sa potrebnim koeficijentom ovisnim o ulaznim dimenzijama web kamere te tako prikazane slike odgovaraju onome što mreža pretpostavlja.

6.3. Usporedba rezultata

Sada kada su predstavljeni rezultati oba načina testiranja, jasno se vide mane ovakvog načina treniranja mreže. Prije svega, skup podataka trebao bi biti puno veći i raznovrsniji kako bi se povećala fleksibilnost aplikacije. Mreža će prikazivati dobre rezultate samo kada joj se predstave slike slične onima na kakvima je trenirala, dok prilikom značajnih promjena na slikama, mreža će zapravo biti dosta loš način da se prikažu karakteristike na licu. Što se tiče same podjele načina na koji se mreža testira, zapravo je svejedno između slika i video formata jer je na kraju krajeva svaki video samo skup više slika pa to nema znatnijeg utjecaja na rad mreže.

7. ZAKLJUČAK

Gledajući na sveukupnu strukturu i rad aplikacije u cjelosti, može se zaključiti kako je inicijalna namjena i potreba za izradom jedne ovakve aplikacije zaista bila opravdana. Naravno, proučavajući i testirajući sve veći broj mogućnosti ovakvoga sustava, samo je pitanje vremena kada će se granice područja rada prijeći te sustav više neće pouzdano raditi. Ovdje se takav slučaj dogodio prilikom testiranja mreže na podacima koji su značajno odstupali od onih naučenih što je dovelo do neželjenih rezultata. S time na umu, treba istaknuti kako primjena ovakvog sustava neće pokriti široki spektar problema gdje se koriste sustavi prepoznavanja ljudskih lica, ali će biti dovoljno dobra kao rješenje nekih jednostavnijih primjera. Daljnjom nadogradnjom, moguće je postići puno preciznije i bolje rezultate koji bi proširili područja primjene neuronskih mreža u vizijskim sustavima. No, rekonstrukcija strukture mreže i ulaznih podatka te optimizacija pojednih parametara u aplikaciji tema su nekog drugog rada, koji će moći koristiti ovaj sustav kao dobru podlogu za kreiranje neke nove, bolje i preciznije aplikacije.

LITERATURA

[1] Neuron, *https://hr.wikipedia.org/wiki/Neuron,* Pristupljeno 30. prosinca, 2020.

[2] SmartGate, *https://en.wikipedia.org/wiki/SmartGate,* Pristupljeno 4. siječnja, 2021.

[3] DeepFace, *https://en.wikipedia.org/wiki/DeepFace,* Pristupljeno 4. siječnja, 2021.

[4] PyTorch, *https://en.wikipedia.org/wiki/PyTorch,* Pristupljeno 10. siječnja, 2021.

[5] Brownlee Jason, Deep Learning for comupter vision, 17. travanj, 2019. How Do Convolutional Layers Work in Deep Learning Neural Networks?, *https://machinelearning mastery.com/convolutional-layers-for-deep-learning-neural-networks/,* Pristupljeno 19. siječnja, 2021.

[6] Brownlee Jason, Deep Learning Performance, 9. siječanj, 2019. A Gentle Introduction to the Rectified Linear Unit (ReLU), *https://machinelearningmastery.com /rectified-linear-activation-function-for-deep-learning-neural-networks/,* Pristupljeno 19. siječnja, 2021.

[7] What is max pooling in convolutional neural networks?, *https://www.quora.com/ What-is-max-pooling-in-convolutional-neural-networks,* Pristupljeno 21. siječnja, 2021.

[8] Brownlee Jason, Deep Learning Performance, 3. srpanj, 2017. Gentle Introduction to the Adam Optimization Algorithm for Deep Learning, *https://machinelearningmastery.com /adam-optimization-algorithm-for-deep-learning/,* Pristupljeno 22. siječnja, 2021.

PRILOZI

1. CD-R disk