Zaključivanje afektivnog virtualnog agenta temeljeno na višemodalnoj interakciji

Korade, Dinko

SVEUČILIŠTE U ZAGREBU

FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Dinko Korade

Zagreb, 2020.

SVEUČILIŠTE U ZAGREBU

FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Mentor:

Doc.dr.sc. Tomislav Stipančić, dipl. ing.

Student:

Dinko Korade

Zagreb, 2020.

Izjavljujem da sam ovaj rad izradio samostalno koristeći stečena znanja tijekom studija i navedenu literaturu.

Posebno bih se zahvalio mentoru doc. dr. sc. Tomislavu Stipančiću na stručnim savjetima, pristupačnosti i pruženoj pomoći pri izradi ovoga rada.

Od srca se zahvaljujem svojim roditeljima i sestri što su izdržali sve moje mušice i bili mi najveća podrška tijekom ovog studija.

Dinko Korade

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

Sveučilište u Zagrebu Fakultet strojarstva i brodogradnje	
Datum:	Prilog:
Klasa:	
Ur. broj:	

DIPLOMSKI ZADATAK

Student: **DINKO KORADE** Mat. br.: 0035199393

Naslov rada na hrvatskom jeziku: **Zaključivanje afektivnog virtualnog agenta temeljeno na višemodalnoj interakciji**

Naslov rada na engleskom jeziku: **Reasoning of an affective virtual agent based on multimodal interaction**

Opis zadatka:

Osjetilne modalnosti omogućuju prikupljanje podataka različitim osjetilima kao što su osjetila vida, zvuka, okusa, njuha i dodira. Da bi prikupili informacije iz realnog svijeta tehnički sustavi koriste senzore.
Upravljački mehanizam afektivnog virtualnog agenta omogućuje analizu i procjenu emocionalnog stanja osobe od interesa. U tu svrhu se često koriste vizijski sustavi pomoću kojih se vrši akvizicija lica i/ili poze osobe od interesa kako bi određene karakteristične točke bile prepoznate te dovedene u vezu s trenutnim emocionalnim stanjem.
Višemodalni pristup povećava percepcijski potencijal virtualnog agenta te omogućava pouzdaniju procjenu. Tako parcijalna informacija koja trenutno stigne u sustav može promijeniti perspektivu zaključivanja agenta.
U radu je potrebno izraditi cjelovito softversko rješenje afektivnog virtualnog agenta koji zaključak o trenutnom emocionalnom stanju osobe od interesa donosi temeljem:

1. analize mimike i izražaja lica osobe,
2. analize intenziteta pokreta tijela osobe,
3. informacije o razini buke u prostoriji.

Dobiveno softversko rješenje je potrebno eksperimentalno evaluirati uključivši ljudske subjekte.
U radu je potrebno navesti korištenu literaturu te eventualno dobivenu pomoć.

Zadatak zadan:	Rok predaje rada:	Predviđeni datum obrane:
14. studenog 2019.	16. siječnja 2020.	20. siječnja do 24. siječnja 2020.

Zadatak zadao: Predsjednica Povjerenstva:

doc. dr. sc. Tomislav Stipančić prof. dr. sc. Biserka Runje

SADRŽAJ

POPIS SLIKA

POPIS TABLICA

POPIS OZNAKA

Oznaka	Jedinica	Opis
a		brojčana vrijednost neurona
B		bias
I_0	W/m^2	prag čujnosti
I_1	W/m^2	jakost zvuka
L	dB	glasnoća
n		broj povezanih neurona
net		ukupna vrijednost svih ulaza u neuron

SAŽETAK

Tema ovog rada je prepoznavanje emocija pomoću afektivnog virtualnog agenta. Emocije su veoma kompleksan pojam i kao takve su dio mnogih istraživanja. Prvi dio rada se bavi teorijom i objašnjava pojmove koji će se koristiti. Zatim slijedi poglavlje o izvedbi zadatka gdje se objašnjava procedura i postupak izrade. Na kraju dolazi najzanimljiviji dio u obliku dobivenih rezultata. Analizirat će se trenirani model neuronske mreže kao i razlike između jednomodalnog i multimodalnog pristupa procjeni.

Ključne riječi: afektivna robotika, neuronske mreže, Python, emocije.

SUMMARY

The main focus of this thesis is emotion recognition with affective virtual agent. Emotions are very complex, and they have been the subject matter of many researchers and papers. The first part of this thesis deals with theory and introduces the key terms. This is followed by explanatious of procedures and tools used to conduct this task. Finally, the last part of the thesis, and the most interesting one, shows the project results; it analyses the neural network model as esll as monomodal and multimodal interaction.

Key words: affective robotics, neural network, Python, emotions.

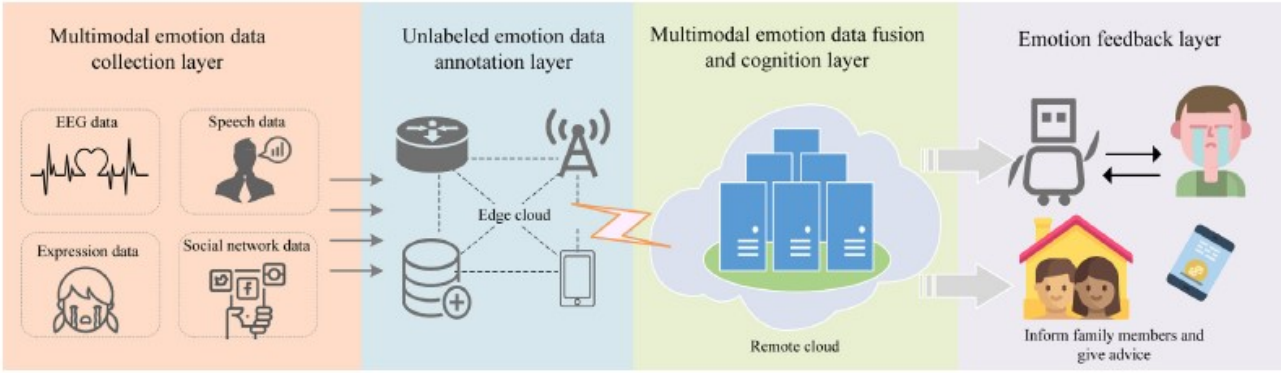
1. UVOD

U današnje vrijeme svjedoci smo kako je tehnologija postala veliki dio naše svakodnevnice. Više se „družimo“ s njom nego s ljudima. Mlađe generacije često ispituju roditelje kako su oni provodili svoje vrijeme kada nisu imali mobitele ili robote da ih zabavljaju. Danas su ti mobitel i roboti normalna stvar i iz godine u godinu ih poboljšavamo. Trenutno veliku popularnost stječe afektivna robotika. Afektivna robotika je grana robotike koja prepoznaje i uzvraća osjećaje i emocije u interakciji s ljudima. U Japanu su razvili robota kojeg su nazvali *Pepper* [Slika 1]. Pepper je humanoidni robot koji može prepoznati emociju čovjeka na temelju njegovog izraza lica i pomoću te informacije određuje kako će stupiti u interakciju s tom osobom.

Slika 1. Robot Pepper

Problem kod emocija je što ih nije lagano odrediti. Mnogi sociolozi i psiholozi dan danas vode debate o tome što su emocije i kako ih definirati. Među poznatijim teorijama su Russelova teorija emocija, Ekmanova, Schachterova teorija i mnoge druge. Sada se postavlja pitanje, ako se mi ljudi ne možemo dogovoriti oko definicije emocija i imamo problema pri njihovom određivanju, kako onda očekujemo da roboti to bolje rade od nas? Razvoj interneta i umjetne inteligencije nam je u tome pomogao. Prvi sustavi koji su se bavili procjenom emocija uglavnom su bili jednomodalni. Procjenjivali su emocije na temelju izraza lica, analize glasa, analize pokreta ili neke druge metode. Takav pristup nije pokazivao veliku točnost zbog nedostatka informacija koje su mogle utjecati na procjenu. Zato se okrenulo multimodalnom pristupu [1] kako bi se povećala količina informacija na temelju kojih se dolazi do procjene emocije.

Prvi korak u procjeni emocija je svakako izraz lica. Iz njega možemo prikupiti najviše informacija. Nakon toga slijedi analiza glasa osobe i analiza pokreta, odnosno položaj u kojem se tijelo nalazi prilikom procjene. To su osnovne metode koje su prisutne u većini algoritama, a nova istraživanja idu dalje i analiziraju podatke poput brzine otkucaja srca, temperature, EEG-a, rukopisa, pa čak i podatke s društvenih mreža.



Slika 2. Multimodalna procjena

Postupak procjene emocije u stvarnom vremenu se sastoji od nekoliko koraka [Slika 2], [1] . Prvo je potrebno odrediti koji se podaci prikupljaju. Na slici vidimo da se ovdje radi o analizi izraza lica, analizi govora, EEG-a i podataka s društvenih mreža. Nakon što su ti podaci prikupljeni, kreće se s njihovom obradom. Nakon obrade i fuzije rezultata dobije se povratna informacija koja govori o kojoj se emociji radi.

Svrha procjene emocije u afektivnoj robotici je da nam robot može prikazati empatiju i pozitivno utjecati na naše raspoloženje. Primjerice, ako smo tužni, pustiti će nam neku veselu pjesmu kako bi nas oraspoložio i slično. Osim prikazivanja empatije, želi se postići i procjena budućeg ponašanja osobe čije se emocije procjenjuju, na primjer predviđanje kako će neki film utjecati na tu osobu, a pokušavaju se detektirati počeci raznih bolesti poput depresije.

U ovom će se radu primjeniti neke od navedenih načina za procjenu emocija. Neke metode zahtjevaju skupu opremu, stoga će u ovom radu fokus biti na prepoznavanju emocija pomoću izraza lica, intenziteta pokreta tijela i razine buke u prostoriji. Cilj je multimodalnim postupkom prikupiti što više informacija i dati povratnu informaciju o tome koje su emocije prisutne kod osobe koja se analizira. Softversko rješenje je izvedeno u programskom jeziku *Python* koji posjeduje programske biblioteke pomoću kojih se mogu sprovesti navedene metode za procjenu emocija.

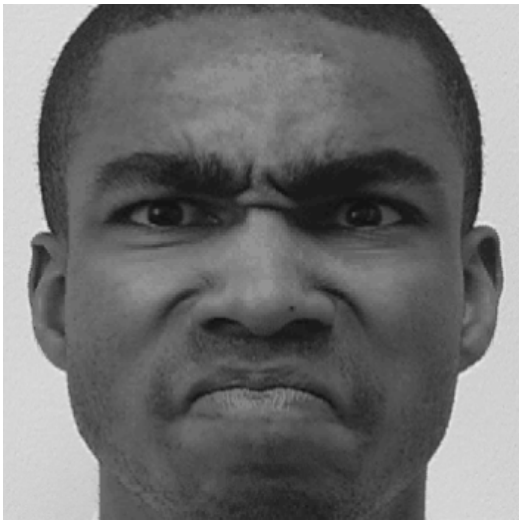
2. RAZRADA ZADATKA

U prethodnom poglavlju smo dobili kratak uvod u temu ovog rada. Želja je pomoću programskog jezika *Python* procjeniti emocije prisutne kod testirane osobe. Zato prije teorije i same izvedbe zadatka treba bolje upoznati koncept emocija kako bi se lakše shvatile neke pretpostavke donesene u radu.

2.1. EMOCIJE

Emocije su kao i većina drugih fenomena u psihologiji, poput inteligencije i pamćenja, latentan pojam. To znači da ih nije moguće izravno promatrati. Emocije izražavaju naš stav i odnos prema nekom objektu (nešto nas veseli, a nešto ljuti). Zato su one subjektivnog karaktera i postoji više čimbenika koji na njih utječu. Baš zbog te subjektivnosti se u struci vode brojne debate o njima. Pionir u opisivanju emocija je svakako bio Charlse Darwin, otac moderne biologije. On je emocijama dodjelio ulogu u našem preživljavanju. Povezao ih je sa zaostalim oblicima poašanja naših predaka [2]. Tako je povezao podsmijeh s režanjem, a plakanje beba s vrištanjem. Ipak, modernije vrijeme je donijelo i nove teorije. Neke od njih su Jamesova i Schachterova teorija emocija [3]. Jamesov se pristup može nazvati i "tjelesni pristup" jer ističe da emocije definiraju tjelesne promjene poput ubrzanog rada srca, dubokog disanja, širenja očiju, glasnog govora... Schachterova teorija se djelomično slaže s Jamesovom, ali on utjecaj fizičkog pristupa proširuje još i s kognicijom. Smatra kako prijašnja iskustva itekako imaju utjecaj na ponovni susret s emocijom. Dobar primjer je naveden u knjizi *Klasične teorije emocija u svjetlu suvremenih empirijskih spoznaja* [3], gdje se navodi primjer terorističkog napada na Pariz. Slučajni prolaznici će reagirati panično, dok će vojnik ili policajac sigurno reagirati staloženije. To ne znači da vojnik ili policajac ne osjeća strah, što prema Jamesu dovodi do ubrzanja otkucaja srca ili veće glasnoće govora, već da imaju iskustva s takvim događajima i spremni su bolje reagirati. Zanimljiva su istraživanja Paula Ekmana koji tvrdi da se emocije mogu prepoznati na temelju izraza lica. On je jedan od rijetkih kojih je ponudio svoje mišljenje o osnovnim emocijama, a prema njemu to su:

- Ljutnja
- Strah
- Gađenje
- Sreća
- Tuga
- Iznenađenje
- Prezir



Slika 3. Mimika emocije ljutnje

Na slici vidimo izraz lica koji predstavlja ljutnju [Slika 3]. Možemo primijetiti skupljenje usne, naboranu kožu iznad nosa i spuštene obrvne. Prema Ekmanu određeni izraz lica koji prestavlja neku emociju povlači za sobom neke fizičke pojave koji dodatno potvrđuju tu emociju. Tako je za ljutnju osim prikazanog izraza lica, još je karakterističan glasan govor i često pomicanje tijela.
Iz priloženog vidimo da su emocije dosta kompleksan pojam oko kojeg se vode mnoga istraživanja i debate, a ovdje su opisane neke od njih s ciljem lakšeg shvaćanja tematike i metoda koje će se koristiti u ovom radu.

2.2. PROGRAMSKI JEZIK PYTHON

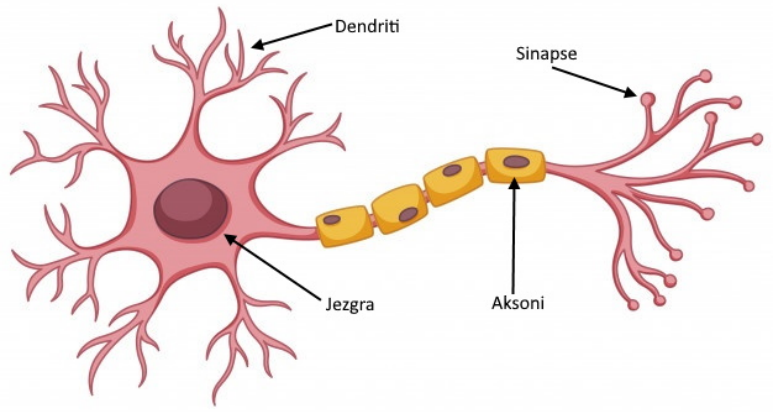
Python je jedan od najpoznatiji svjetskih programskih jezika. Stvorio ga je Guido Van Rossum 1990. godine, a prva javna verzija bila je dostupna u veljači 1991. godine [4]. Ime je dobio po popularnoj televizijskoj seriji *Monty Python's Flying Circus*. Veoma je jednostavan i fleksibilan, te omogućuje programerima da koriste više stilova programiranja poput: objektno orijentiranog, funkcionalnog ili proceduralnog programiranja. Interpreterski je jezik i zbog toga je sporiji u odnosnu na *C* i *C++*. Razlog zbog kojeg se koristi u ovom radu je veliki broj dodatnih biblioteka koje olakšavaju rad u njemu. Najbitniji za ovaj rad su *TensorFlow* (neuronske mreže), *PyAudio* (mjerenje glasnoće) i *OpenCv* (intenzitet gibanja). Korišteni su i mnogi drugi paketi, a svi će biti detaljnije objašnjeni kod analize samog koda.

3. TEORIJSKA OSNOVA RADA

U zadatku je navedeno da će se procjena emocija izvoditi na temelju analize izraza lica, analize inteziteta pokreta i informacije o razini buke u prostoriji. Kako bi sve podatke mogli prikupiti i analizirati ih, moramo znati nešto više u njima. Zato je u ovom poglavlju predstavljena teorija koje će nam u tome pomoći.

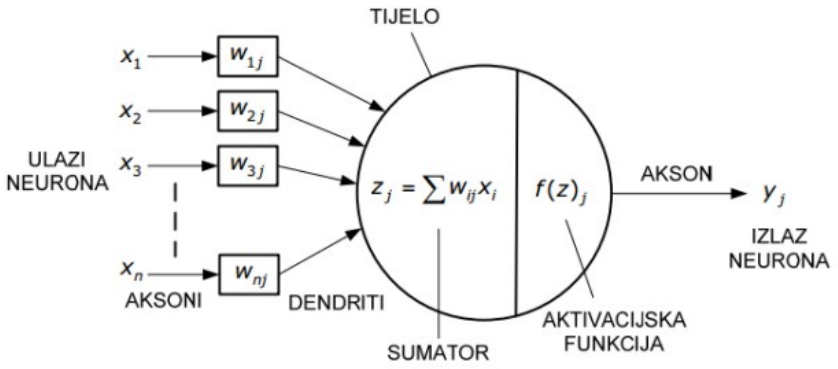
3.1. NEURONSKA MREŽA

Ideja za umjetnu neuronsku mrežu generirana je iz pokušaja modeliranja biofiziologije ljudskog mozga. Zato su rani radovi u ovom području dolazili od znanstvenika, biofizičara i psihologa, a inženjeri su se tek kasnije uključili. 1980. godine, William James postavlja sljedeću tvrdnju: „Aktivnost bilo koje točke mozga čovjeka predstavlja zbroj tedencija svih ostali točaka da se prazne (ispaljuju) u nju“[5]. Ova je tvrdnja poslužila 1943. godine McCullochu i Pittsu da predlože jednostavan model umjetnog neurona koji se koristi i danas. Da bismo lakše shvatili umjetni neuron, prvo moramo objasniti biološki.



Slika 4. Pojednostavljeni biološki neuron

Pojednostavljeni biološki neuron se sastoji od jezgre, aksona i dendrita [Slika 4]. Sinapsa je mjesto komunikacije između dva neurona. Na tom se mjestu prenosi impuls s jednog neurona na drugi. Akson jednog neurona formira sinaptičke veze s drugim neuronima. Impulsi su različitog intenziteta i time je određena efikasnost sinaptičkog prijenosa. Neuron će poslati impuls kroz akson ako je doveden u stanje dovoljne uzbude. Iz ovog vidimo ideju koju je William James opisao. Na sličan način funkcioniraju i umjetni neuroni [Slika 5].

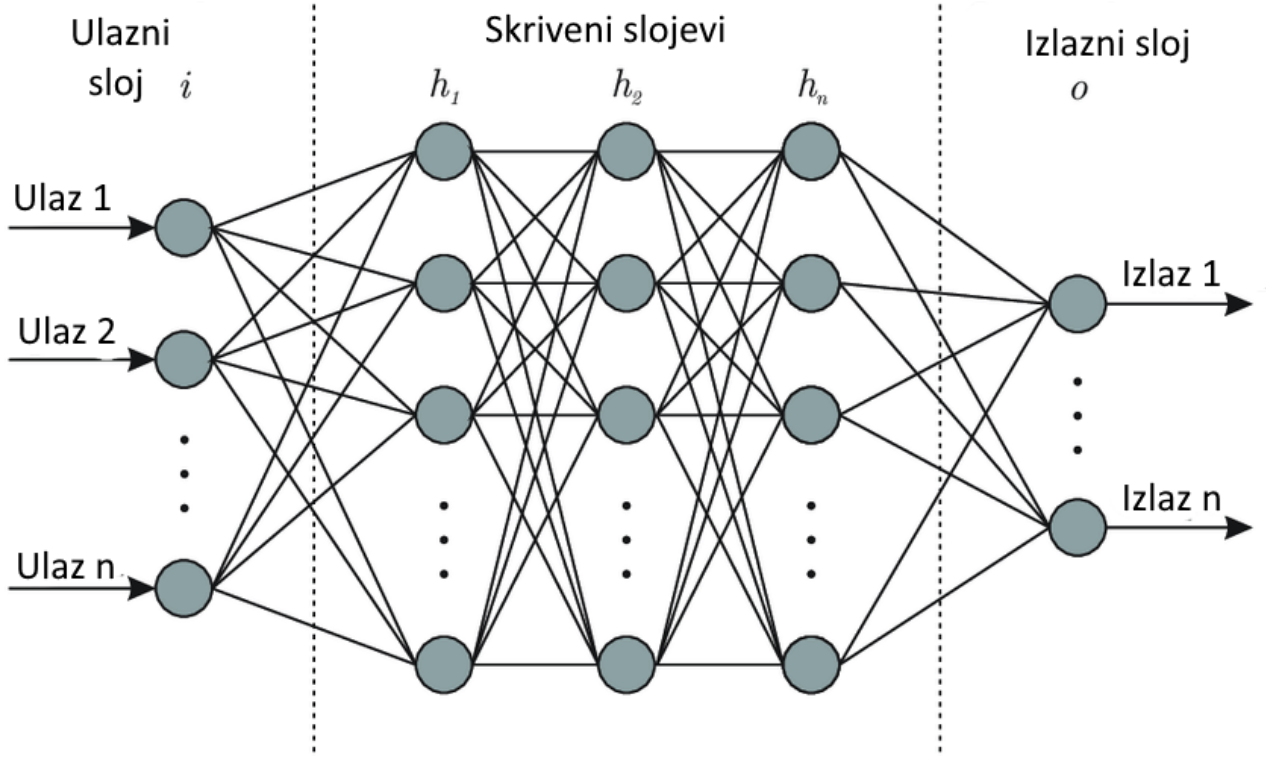


Slika 5. Umjetni neuron

Umjetni neuron je dizajniran da oponaša osnovne funkcije biološkog neurona. Tijelo biološkog neurona zamjenjuje se sumatorom, ulogu dendrita preuzimaju ulazi u sumator, izlaz sumatora je akson umjetnog neurona, a uloga praga osjetljivosti bioloških neurona preslikava se na aktivacijske funkcije. Funkcije sinaptičke veze biološkog neurona s njegovom okolinom preslikavaju se na težinske faktore, preko kojih se i ostvaruje veza umjetnog neurona s njegovom okolinom [5].
Postoje mnoge usporedbe između mozga i računala. Ipak, mozak se sastoji od tolikog broja neurona da se ni najmodernije računalo ne može mjeriti s njim. Unatoč tome što računala ne mogu doseći razinu mozga, neuronske mreže su pronašle široku primjenu te se koriste za: obradu slika, prepoznavanje oblika, analizu signala, razna predviđanja, dijagnostiku i slično.

3.1.1. OSNOVNI OBLIK NEURONSKE MREŽE

Neuronske mreže imaju više podjela. Mogu se dijeliti prema broju slojeva (jednoslojne, dvoslojne, troslojne...), prema načinu učenja (jedan korak ili iterativno), prema protoku signala (unaprijedne ili povratne) i mnoge druge. Za obradu slika najbolje rezultate je pokazala konvolucijska neuronska mreža (eng. *convolutional neural network*). Kako bismo je lakše objasnili, prvo ćemo prikazati princip rada osnovnog oblika neuronske mreže.

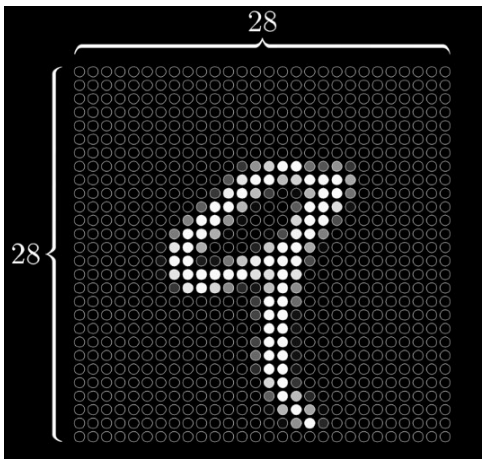


Slika 6. Osnovni oblik neuronske mreže

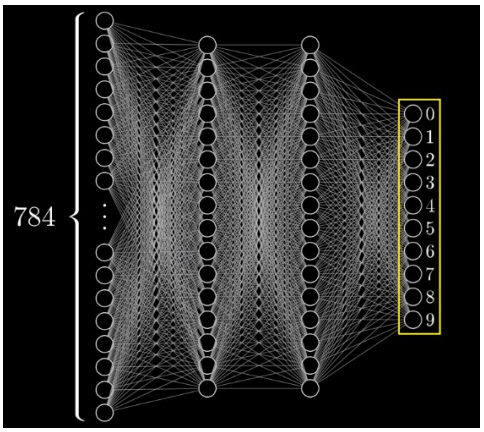
Neuronska mreža ima poprilično jednostavan izgled [Slika 6]. Ona se sastoji od ulaznog sloja i izlaznog sloja, te skrivenih slojeva čiji je broj proizvoljan. Krugovi na slici predstavljaju pojedinačne neurone i svaki sadrži brojčanu vrijednost koja će se u nastavku označavati slovom *a*. Svaki neuron povezan je sa svim neuronima u sljedećem sloju. Konekcije prestavljaju sinaptičke veze i manifestiraju se preko težinskih faktora koji će se u nastavku označavati s *w*. Težinski faktor može biti negativan ili pozitivan broj.

Jedan od najpopularnih problema koji je riješen pomoću neuronskih mreža je prepoznavanje brojeva od 0 do 9. Naš mozak vrlo lako prepoznaje brojeve bez obzira na rukopis kojim su napisani. Ipak, računalu to predstavlja problem i potrebna mu je neuronska mreža kako bi to naučio. U nastavku će uz pomoć slika biti objašnjen postupak koji opisuje kako točno neuronska mreža uči.

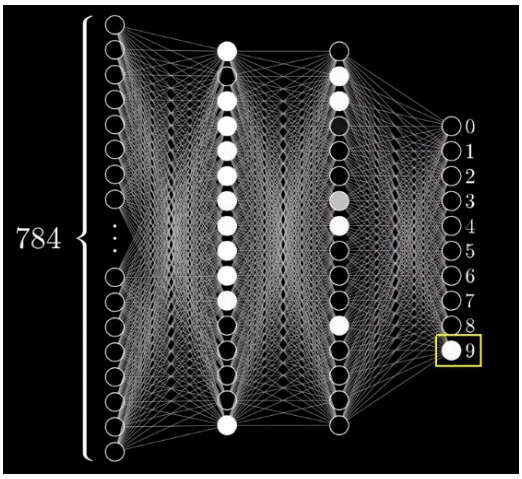
Za početak, potrebna je velika baza slika (eng. *database*) različito napisanih brojeva pomoću kojih će mreža učiti. Iz te baze, svaki član ulazi u mrežu preko ulaznog sloja. Na slici vidimo kako to izgleda na primjeru broja 9 [Slika 7]. Slika broja se sastoji od 784 (28x28) piksela. Svaki piksel će predstavljati po jedan neuron u ulaznom sloju mreže [Slika 8]. Svaki neuron ima svoju vrijednost koja se kreće na skali od 0 do 1. U ovom slučaju 0 prestavlja crni piksel, a 1 bijeli.



Slika 7. Rastavljanje broja 9 na piksele



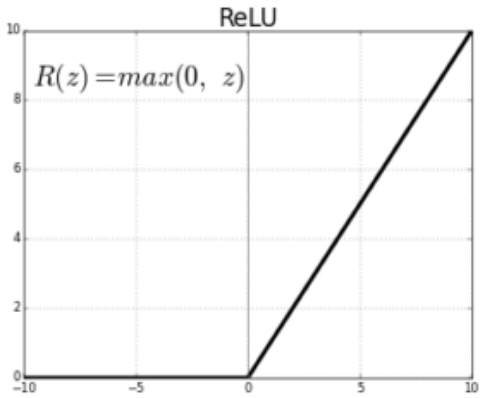
Slika 8. Ulazni sloj mreže



Slika 9. Aktivna neuronska mreža

Nakon ulaznog sloja slijede skriveni slojevi. Ne postoji pravilo koje određuje točan broj skrivenih slojeva i njihovih neurona, već je to više iskustveni parametar, a njihov je broj veći što je kompleksniji problem koji rješavamo. Na kraju mreže je izlazni sloj koji se sastoji od onoliko neurona koliko ima potencijalnih rješenja problema. Na [Slika 9] vidimo da su aktivni neuroni obojani u bijelo, a oni koji nisu aktivni u crno. Hoće li neuron biti aktivan ovisi o aktivacijskoj funkciji.

Postoji nekoliko vrsta aktivacijskih funkcija, a najpoznatije su sigmoidna i relu funkcija. Za dobiveni ulaz one daju izlaz koji određuje aktivnost neurona. Na [Slika 10] vidimo kako izgleda relu aktivacijska funkcija, a vrijednost neurona se računa prema izrazu (3.1).

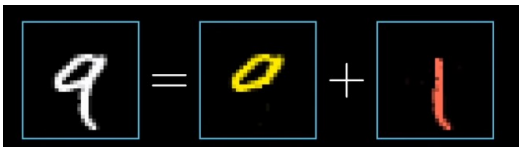


Slika 10. Relu aktivacijska funkcija

$$net = \sum_{i=1}^{n} a_i w_1 \pm B \tag{3.1}$$

- net - ukupna vrijednost
- a - veličina *i*-tog neurona iz prethodnog sloja
- w - težinski faktor *i*-tog neurona iz prethodnog sloja
- B - Bias
- n - broj povezanih neurona

Iz izraza (3.1) vidimo da se ukupna vrijednost dobije kao zbroj vrijednosti (a) svih povezanih neurona iz prethodnog sloja pomnoženih s odgovarajućim težinskim faktorom (w). Bias je fiksni broj koji označava neku vrijednost, pozitivinu ili negativnu koju suma mora prijeći. Ukupna suma onda ulazi u aktivacijsku funkciju i njezin izlaz daje veličinu samog neurona, odnosno njegov izlaz koji onda pomnožen drugim težinskim faktorima putuje dalje prema drugim neuronima.
Bitna je uloga skrivenih slojeva jer oni nastoje ulaz povezati s odgovarajućim izlazom. Ulazni sloj sadrži onoliko neurona koliko slika ima piksela. Onda za neurone u srednjem sloju možemo reći da te piksele pokušavaju spojiti u smislene cjeline koje kad se povežu daju željeni izlaz. Za ilustraciju toga će nam najbolje pomoći sljedeća slika [Slika 11].



Slika 11. Podijela broja 9 na dijelove

Na slici je broj devet prikazan iz dva dijela. Naša mreža sadrži dva skrivena sloja i u nekoj pojednostavljenoj verziji mogli bismo reći kako jedan neuron u prvom skrivenom sloju predstavlja točno prvi dio broja, a drugi dio broja predstavlja neuron u sljedećem sloju. Kad se ta dva neurona spoje dobije se željeni izlaz. Takve kombinacije postoje i za sve ostale brojeve što dodatno komplicira učenje. Cilj je kroz učenje postići vrijednosti aktivacijskih funkcija i težinskih faktora tako da za dani ulaz dobijemo željeni izlaz. No što ako na izlazu dobijemo krivi rezultat? Zato postoji ocjenjivanje neuronske mreže koje mijenja težinske faktore i tako uči mrežu. Učenje ovisi o vrsti mreže, a ovdje je izvedeno pomoću mjere točnosti (eng. *cost or loss function*) i metode povratnog rasprostriranja (eng. *backpropagation*). Mjera točnosti je funkcija koja ocjenjuje izvedbu neuronske mreže. Ona određuje grešku između dobivene vrijednosti i očekivane vrijednosti. Cilj joj je tu grešku minimizirati ili maksimizirati, ovisno o vrsti zadatka. Na temelju nje radi metoda povratnog rasprostriranja koja mijenja težinske faktore s ciljem smanjenja greške [5], [6]. *Youtube* kanal 3BLUE1BROWN [7] sadrži brojne primjere neuronskih mreža i svakako je preporučljiv za dodatne informacije.

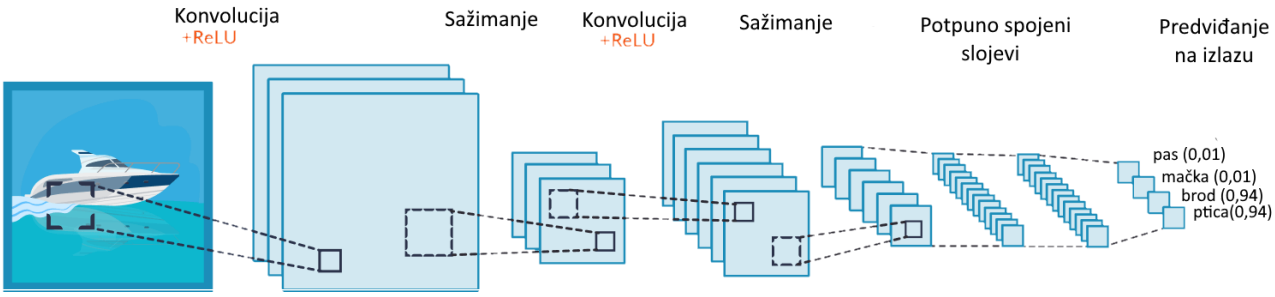
3.2. KONVOLUCIJSKE NEURONSKE MREŽE

Konvolucijska neuronska mreža se može smatrati nadogradnjom na višeslojne mreže. Ona u skrivenim slojevima sadrži konvolucijske (eng. *convolution*) slojeve i slojeve sažimanja (eng. *pooling*). Dobra je u prepoznavanju obrazaca i uzoraka, zbog čega se koristi za analizu slika. Pod obrasce i uzorke se misli na rubove, oblike, teksture i objekte. Za prepoznavanje obrasca se koriste filtri, a što mreža ide dublje u analizi slike, to joj je lakše prepoznati detalje poput očiju, obrva, usana i sl. Filter je ništa drugo nego obična matrica kvadratnih dimenzija koja prolazi svaki dio slike u koracima (eng. *stride*) i tako detektira razne objekte [Slika 12]. U običnim neuronskim mrežama učenje mreže se postizalo izmjenom težinskih faktora, dok se kod konvolucijskih mreža uče težine unutar filtra.

1	2	-1
0	1	2
-2	-1	1

Slika 12. Konvolucijski filter

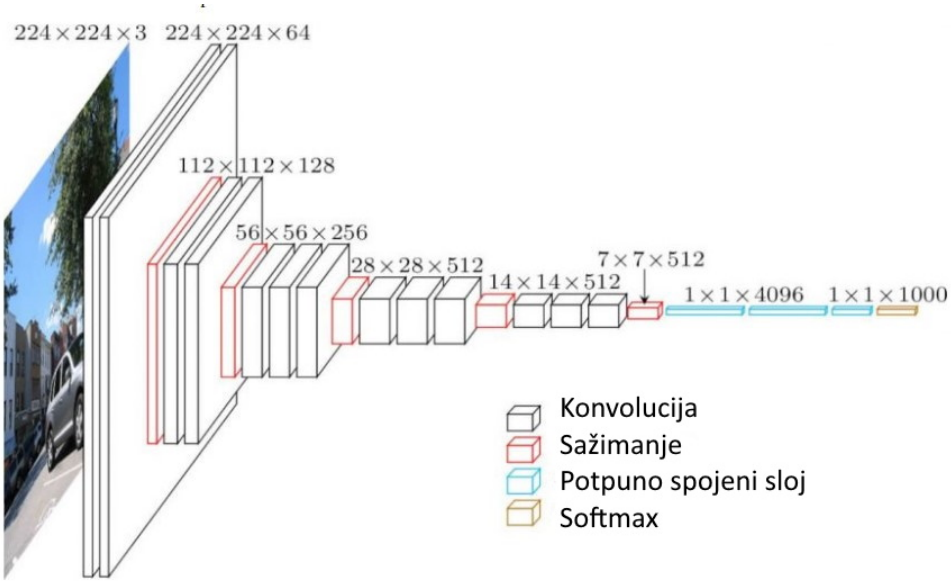
Nakon jednog ili više konvolucijskih slojeva, dolazi do sloja sažimanja. Oni se koriste s ciljem smanjivanja rezolucije. Na taj način smanjuju osjetljivosti na sitne pomake značajki na slici. Na [Slika 13] vidimo primjer konvolucijske neuronske mreže. Ulaz u mrežu je slika i nakon nje slijedi niz konvolucijskih slojeva i slojeva sažimanja [8]. Zatim slijede potpuno spojeni slojevi i na kraju izlazni sloj koji pokazuje procjenu mreže.



Slika 13. Konvolucijska neuronska mreža

3.2.1. PRETRENIRANA NEURONSKA MREŽA VGG16

Kako su konvolucijske neuronske mreže najbolje za analizu slika, tako postoje razni modeli koji se koriste u tu svrhu. Jedna od njih je i model VGG16. VGG16 je konvolucijska neuronska mreža čiji su model predložili K. Simonyan i A. Zisserman sa sveučilišta *Oxford* u svom radu *Very Deep Convolutional Networks for Large-Scale Image Recognition* [9]. Postigli su točnost od 92,7 %, a njihova baza se sastojala od preko 14 milijuna slika raspodjeljenih u 1000 klasa. Mreža je posebna jer sadrži veliki broj hiper-parametara i u konvolucijskim slojevima koristi filtere dimenzija 3x3 s korakom 1. Slojevi sažimanja koriste filtere dimenzija 2x2 s korakom 2. Na [Slika 14] je vidljiva njezina konstrukcija koja završava s dva potpuno spojena sloja i izlaznim slojem koji predstavlja ”softmax” funkciju. Softmax funkcija daje predviđanje u postocima za svako od ponuđenih rješenja.



Slika 14. VGG16

VGG16 ima preko 138 milijuna parametara što ju čini veoma sporom, a naučene težine zauzimaju puno memorije. Ipak, odlična je za učenje zato što se vrlo lako može implementirati pomoću *PyTorcha*, *TensorFlowa* i *Kerasa*. Iz tog razloga je ona korištena u ovom radu, a njezina implementacija će biti pokazana u sljedećem poglavlju.

3.3. ZVUK

Zvuk je mehanički val frekvencija. Ljudsko uho čuje frekvencije u rasponu od 16 Hz do 20 kHz. Frekvencije ispod 16Hz se nazivaju infrazvukom, a više od 20 kHz ultrazvukom. Te se frekvencije često koriste u raznim znanostima kao na primjer u medicini. Ako je frekvencija viša od 1 GHz onda već govorimo o hiperzvuku. Zvuk nastaje periodičnim titranjem izvora zvuka koji u neposrednoj okolini mijenja tlak medija u kojem se širi. Poremećaj tlaka se prenosi na susjedne čestice i tako se širi u obliku longitudinalnih i transverzalnih valova. Brzina zvuka ovisi o gustoći, temperaturi i tlaku, a njegova brzina u zraku iznosi oko 343 m/s. Zvuk se širi bez prijenosa mase, ali se zvukom prenose impuls sile i energija pomoću koje se računaju jakost i glasnoća. Jakost zvuka je fizikalna veličina koja opisuje energiju zvučnog vala u vremenskom razdoblju kroz površinu okomitu na smjer širenja vala. Upravo će nam jakost zvuka trebati da izračunamo glasnoću zvuka pomoću decibela. Decibel je jedinica razine neke fizikalne veličine (snage, jakosti zvuka i slično) i računa se prema sljedećem izrazu:

$$L = 10 \cdot log_{10}(\frac{I_1}{I_0}) \tag{3.2}$$

- L - glasnoća u decibelima (dB)
- I_1 - jakost zvuka (W/m^2)
- I_0 - referentna vrijednost ili prag čujnosti

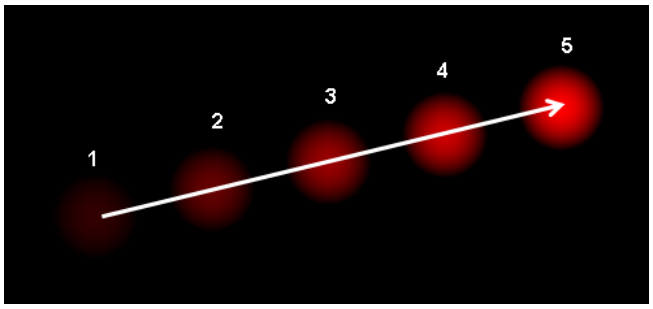
Iz jednadžbe (3.2) vidimo da je usporedba određena logaritamskim mjerilom i kao što je prikazano I_1 označava razinu jakosti zvuka koja je izmjerena, dok I_0 označava prag čujnosti. Prag čujnosti je najmanja razina jakosti zvuka koju čovjek može čuti. U [Tablica 1] možemo vidjeti raspon decibela i intenziteta zvuka za određeni izvor. Upravo će nam ta tablica trebati u ovom radu kako bi na temelju izmjerene glasnoće mogli odrediti o kojoj se razini zvuka radi. Ove tablice nisu univerzalne već različiti izvori navode različite intenzitete, ali odstupanja ne bi trebala biti veća od $\pm$ 10 dB. U ovom radu zvuk će se mjeriti pomoću *Pythonove* biblioteke *PyAudio*, a kôd i njegovo objašnjenje će biti prikazano u sljedećem poglavlju.

Izvori zvuka	Nivo intenziteta dB	Intenzitet W/m^2
Prag čujnosti	0	10^{-12}
Tihi razgovor	40	10^{-8}
Glasni razgovor	60	10^{-6}
Gust ulični saobraćaj	80	10^{-4}
Zakivanje	100	10^{-2}
Granica bola	120	1

Tablica 1. Raspon i intenzitet

3.4. INTENZITET GIBANJA (eng. OPTICAL FLOW)

Intenzitet gibanja (eng. *optical flow*) je vidljivo gibanje površina ili objekata na nekom videu ili prizoru (eng. *scene*). Prikazuje razliku gibanja između promatrača i prizora. Često se koristi u vizijskim sustavima za detektiranje i praćenje objekata, detektiranje gibanja ili robotsku navigaciju. Postoji više vrsta algoritama za optical flow, a jedni od napoznatijih su Lucas-Kanade metoda i Gunnar Farnebackov algoritam. U ovom je radu pomoću *Pythonove* biblioteke *OpenCv* korišten Farnebackov algoritam. On procjenjuje gibanje koristeći polinomske izraze [10]. Na [Slika 15] vidimo detekciju gibanja i smjer gibanja loptice koji su se odredili pomoću niza slika (eng. *frames*). U radu smjer gibanja nije potreban, već će se algoritam koristiti kako bi se odredio intenzitet gibanja osobe čije se emocije procjenjuju.



Slika 15. Detekcija gibanja loptice

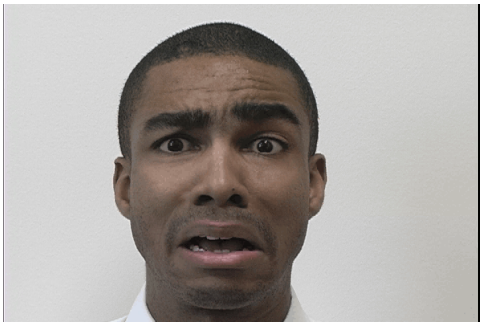
4. IZVEDBA ZADATKA

Prepoznavanje lica osobe i njezinih emocija postala je popularna tema u području strojnog učenja i neuronskih mreža. Na internetu postoji veliki broj članaka [11], [12] koji se bave tim problemom i veliki broj različitih izvedbi. U ovom poglavlju će biti objašnjen svaki dio ovog rada i način na koji funkcionira. Kao što je prije navedeno, zadatak se izvodi u programskom jeziku *Python*. Detaljno će biti objašnjeni samo najbitniji dijelovi kôda, dok će cijeli kôd biti prikazan u Prilogu zajedno s komentarima zbog boljeg razumijevanja.

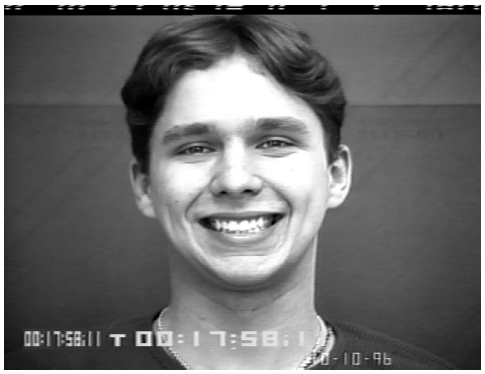
4.1. BAZA PODATAKA

Kako bi neuronska mreža mogla učiti, potrebna joj je odgovarajuća baza podataka. U primjeru s brojevima iz prošlog poglavlja korištena ja baza brojeva *MINST-a*. Ona je javna i služi uglavnom kao pokazni primjer za učenje neuronskih mreža. Sa slikama s emocijama ljudi je stvar malo drugačija. Takva istraživanja i radovi postoje, ali nitko ne želi javno objaviti kompletne baze slika. Zato se baza slika prikupljala iz različitih izvora [13], [14], [15].

Slika 16. Emocija ljutnje



Slika 17. Emocija straha



Slika 18. Emocija sreće

Slika 19. Emocija neutralno

Slika 20. Emocija tuge

Na [Slika 16, 17, 18, 19 i 20] vidimo emocije koje će neuronska mreža učiti prepoznati. Izbor je pao na ljutnju, strah, sreću, neutralno i tugu iz razloga što su to najčešće emocije i svakodnevno se susrećemo s njima. Nakon prikupljenih slika pojavio se problem. Ukupan je broj slika iznosio 695. To je premali broj za trenirnje neuronske mreža. Usporedbe radi, baza podataka *MINST-a* sadrži oko 70000 slika. Kako bi povećali broj slika baze, pronađen je novi izvor [16]. Radi se o animiranim likovima koji oponašaju ljudske emocije.

Slika 21. Emocija ljutnje animirana

Slika 22. Emocija straha animirana

Slika 23. Emocija srece animirana

Slika 24. Emocija neutralno animirana

Slika 25. Emocija tuge animirana

Na slikama [Slika 21, 22, 23, 24 i 25] vidimo prikaz emocija pomoću animiranih likova. Baza animiranih slika sadrži preko 50 000 slika, što je ipak previše jer zauzima ogroman memorijskih prostor. Zato će za svaku emociju biti odabrano po 1000 slika. Baza sada ukupno sadrži 5695 slika. S toliko slika se može trenirati model koji će dati zadovoljavajuće rezultate prilikom testiranja.

4.2. PRIPREMA BAZE LJUDSKIH LICA

U prethodnom potpoglavlju je pokazano kako izgleda baza slika pomoću koje je trenirana neuronska mreža. Baza je podijeljena na 5 kategorija zato što želimo prepoznati 5 različitih emocija (ljutnja, strah, sreća, neutralno i tuga). S obzirom na to da su slike iz različitih izvora, one nisu istih dimenzija i nije im svima lice u krupnom planu. Zato je prije nego se započne s treningom, potrebno pripremiti bazu lica. Prvi korak je pripremiti biblioteke koji će se koristiti u programu [Slika 26].

```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from PIL import Image
import glob
import cv2
from sklearn.model_selection import train_test_split
from keras.layers import Dropout, Dense
from keras.layers.normalization import BatchNormalization
from keras.models import Sequential, load_model
from keras.applications import VGG16
from sklearn.metrics import accuracy_score, confusion_matrix
```

Slika 26. Potrebne Python biblioteke

```
human_angry = glob.glob("Human/anger/*")
print ("Number of images in Angry emotion = "+str(len(human_angry)))

human_angry_folderName = [str(i.split("\\")[0])+"/" for i in human_angry]
human_angry_imageName = [str(i.split("\\")[1])+"/" for i in human_angry]
human_angry_emotion = [["Angry"]*len(human_angry)][0]
human_angry_label = [1]*len(human_angry)

print(len(human_angry_folderName))
print(len(human_angry_imageName))
print(len(human_angry_emotion))
print(len(human_angry_label))

df_angry = pd.DataFrame()
df_angry["folderName"] = human_angry_folderName
df_angry["imageName"] = human_angry_imageName
df_angry["Emotion"] = human_angry_emotion
df_angry["Labels"] = human_angry_label
df_angry.head()
```

Slika 27. Sortiranje slika emocije ”Ljutnja”

S obzirom na to da se baza podataka sastoji od slika ljudi i animiranih likova, prije spajanja će se svaka baza zasebno obraditi. Na [Slika 27] vidimo kako izgleda dio programa za sortiranje emocije ljutnje. Linija 16 sadrži varijablu *human_angry* koja sadrži sve slike s putanje *„Human/anger“*. Treba napomenuti kako se radi o relativnoj putanji i da su nazivi datoteka dio osobnog odabira. Zatim se pomoću paketa *pandas* stvara tablica koja sadrži četiri stupca. Prvi stupac sadrži ime foldera, drugi sadrži ime slike, treći ime emocije i četvrti sadrži oznaku u obliku rednog broja koja će kasnije predstavljati tu emociju. Za emociju ljutnje to je broj 1, za emociju straha broj 2 i tako redom za ostale emocije. Ako je sve izvedeno kako treba, dobije se ispis [Slika 28].

```
Number of images in Angry emotion = 127
127
127
127
127
     folderName             imageName Emotion  Labels
0  Human/anger/  00_005_00000023.png/   Angry       1
1  Human/anger/  01_001_00000067.png/   Angry       1
2  Human/anger/  02_001_00000016.png/   Angry       1
3  Human/anger/  03_001_00000071.png/   Angry       1
4  Human/anger/  04_001_00000022.png/   Angry       1
```

Slika 28. Ispis tablice emocije ljutnje

Isti postupak se provodi provodi i za ostale emocije. Kada se za svaku emociju kreira njezina tablica, potrebno ih je sve spojiti u jednu i naizmjenično sortitati. Razlog naizmjeničnog sortiranja je taj što ako mreži na ulaz stavimo 10 puta istu emociju, ona svaku sljedeću može početi napamet pogađati. Ovakvi naizmjenični unosi daju kvalitetnije rezultate. Kada se sve tablice spoje i promiješaju, novonastala tablica treba izgledati kao na [Slika 29].

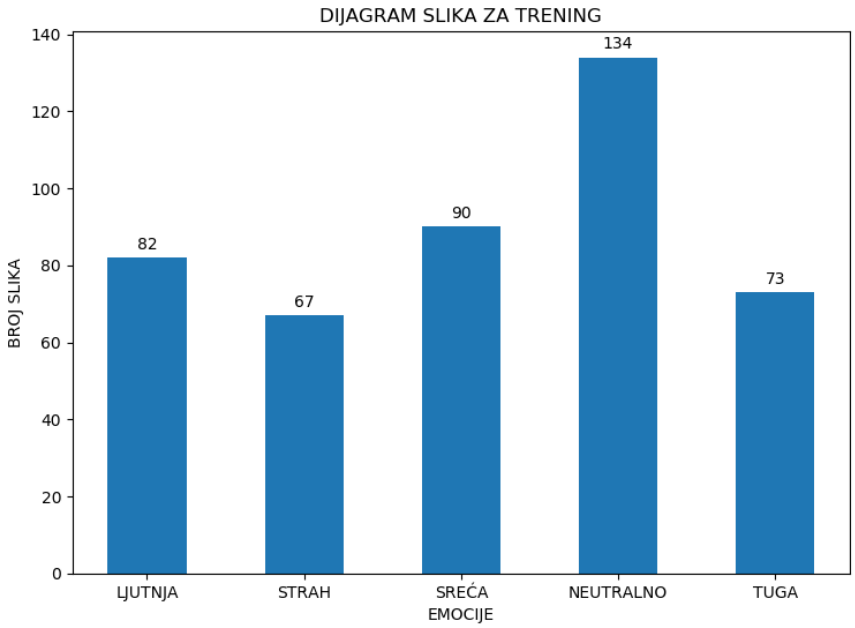
```
       folderName             imageName  Emotion  Labels
0  Human/neutral/       google_315.jpg/  Neutral       4
1    Human/happy/  28_011_00000016.png/    Happy       3
2  Human/sadness/       google_086.jpg/      Sad       5
3    Human/happy/  52_004_00000033.png/    Happy       3
4    Human/anger/       google_011.jpg/    Angry       1
```

Slika 29. Ispis konačne tablice za ljude

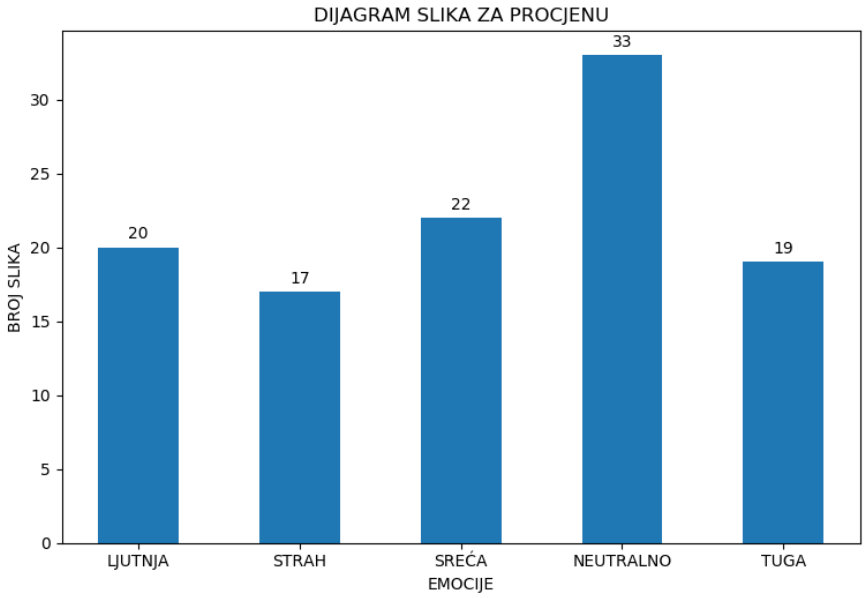
Sada imamo tablicu pomoću koje možemo pristupiti svim slikama baze zajedno s oznakom emocije koju prikazuju. Sljedeći korak je bazu podijeliti u 3 dijela. Prvi i najveći dio su slike koje će se koristiti za trening mreže (eng. *data_train*). Drugi dio čine slike koje služe za testiranje točnosti mreže nakon treninga (eng. *data_test*) i treći dio koji služi za procjenu mreže tijekom treninga (eng. *data_validation*). Taj se dio odvija u kôdu pomoću funkcije *train_test_split* [Slika 30].

```
139  df_human_train_data, df_human_test = train_test_split(Final_human, stratify=Final_human["Labels"], test_size = 0.197860)
140  df_human_train, df_human_cv = train_test_split(df_human_train_data, stratify=df_human_train_data["Labels"], test_size= 0.197860)
141  print(df_human_train.shape)
142  print(df_human_cv.shape)
143  print(df_human_test.shape)
144
145  df_human_train.reset_index(inplace = True, drop = True)
146  df_human_train.to_pickle("HumanData/df_human_train.pkl")
147
148  df_human_cv.reset_index(inplace = True, drop = True)
149  df_human_cv.to_pickle("HumanData/df_human_cv.pkl")
150
151  df_human_test.reset_index(inplace = True, drop = True)
152  df_human_test.to_pickle("HumanData/df_human_test.pkl")
```

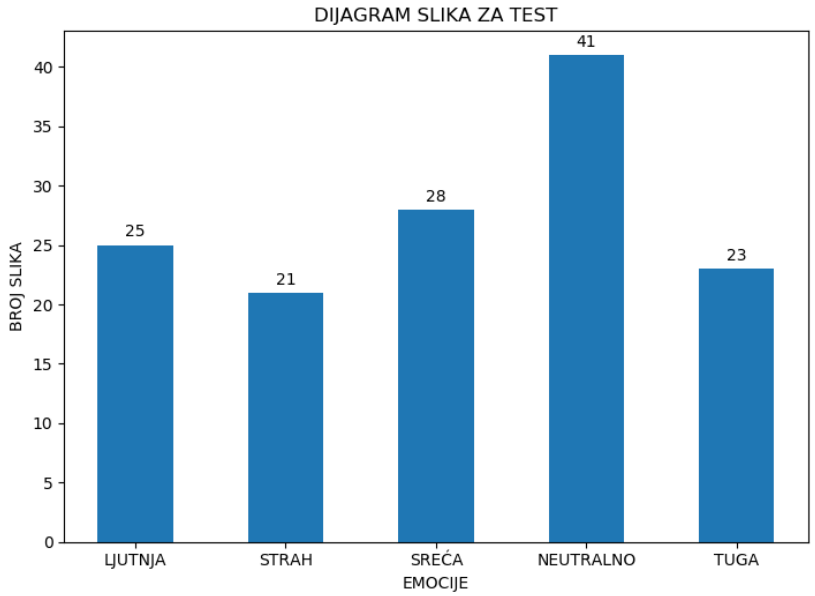
Slika 30. Podijela baze na tri dijela



Slika 31. Dijagram broja slika za trening



Slika 32. Dijagram broja slika za procjenu



Slika 33. Dijagram broja slika za test

Na [Slika 31, 32 i 33], vidimo grafički prikaz raspodjele slika. Također je vidljiv i broj slika za svaku emociju u određenoj bazi. Praksa je da baze za test i procjenu sadrže 5-20% veličine ukupne baze. Kvalitetnije bi bilo da svaka emocija ima jednak broj slika, ali izvori baze nisu napravljeni s jednakim brojem i zato postoji određena razlika.

4.3. PROCESIRANJE SLIKA LJUDSKIH LICA

Sada kada je baza ljudskih lica podijeljena na tri dijela, potrebno je pripremiti slike za trening. Kao što je i prije napomenuto, slike su prikupljane iz različitih izvora i nisu snimane na isti način i nisu istih dimenzija. Druga bitna stvar je da se svaka slika prebaci u sivu skalu (eng. *gray scale*). Na taj se način anulira utjecaj boja na procjenu emocija, a također se smanji veličina slika i time se ubrza proces treninga.

```
96  def convt_to_gray(df):
97      count = 0
98      for i in range(len(df)):
99          path1 = df["folderName"][i]
100         path2 = df["imageName"][i]
101         path2 = path2[:-1]
102         img = cv2.imread(os.path.join(path1, path2))
103         gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
104         cv2.imwrite(os.path.join(path1,path2), gray)
105         count += 1
106
107     print("Total number of images converted and saved = "+str(count))
108
109 convt_to_gray(df_human_train)
110 convt_to_gray(df_human_cv)
111 convt_to_gray(df_human_test)
```

Slika 34. Pretvaranje slika u sivu skalu

Na [Slika 34] vidimo dio programa koji je zadužen da sve slike prebaci u sivu skalu. Za to je zadužena funkcija koja se zove *convt_to_gray*. Ulaz u funkciju je prethodno definirana tablica slika. Unutar funkcije se pomoću programske strukture for petlje iterira kroz svaku sliku. Ovdje se koristi *Pythonova* biblioteka *OpenCv*, čija je kratica cv2. U liniji 102 se pomoću metode *imread* učitava slika u program. Zatim se u sljedećoj liniji koristi metoda *cvtColor*, koja sliku prebacuje iz BGR modela u sivu skalu. Na kraju se u liniji 104 promijenjena slika ponovno zapisuje na istu putanju s koje je uzeta. *OpenCv* učitava sliku u BGR modelu boja, a ne RGB. Zato se pretvorba izvodi na prikazani način. Sljedeći postupak na svakoj slici je detektiranje lica osobe i izoliranje istog. Zatim sve slike pohranjuju tako da imaju jednake dimenzije. Na [Slika 35] vidimo dio programa koji rješava navedeni problem. Prvo je potrebno učitati klasifikator pomoću kojeg će se detektirati lice na slici. U liniji 114 vidimo da se koristi *haarcascade* [17]. Funkcija *face_det_crop_resize* za ulazni parametar ima putanju do željene slike, a te se putanje opet iteriraju pomoću for petlji. Unutar funkcije se slika prvo učita, a zatim u liniji 118 se pomoću metode *detectMultiScale* dobiju koordinate detektiranog lica. Potrebno je provjeriti je li sustav uopće detektirao neko lice. Ako nije, onda se slika ponovno zapisuje na isto mjesto samo s dimenzijom 350x350, a ako je lice detektirano ono se prvo na slici izolira (linija 124), a nakon toga se ponovno zapisuje na isto mjesto s dimenzijama 350x350 (linija 125). [Slika 36 i 37] prikazuju razliku između slika prije i nakon obrade. Na [Slika 37] vidimo izolirano lice s emocijom sreće. S takvom slikom će trening mreže biti puno bolji jer su detalji izraženiji i gubi se utjecaj pozadine.

```
face_cascade = cv2.CascadeClassifier("haarcascade_frontalface_default.xml")
def face_det_crop_resize(img_path):
    img = cv2.imread(img_path)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray, 1.3, 5)

    if faces == ():
        cv2.imwrite(img_path, cv2.resize(gray, (350,350)))
    else:
        for (x,y,w,h) in faces:
            face_clip = img[y:y+h, x:x+w]
            cv2.imwrite(img_path, cv2.resize(face_clip, (350,350)))

for i, d in df_human_train.iterrows():
    img_path = os.path.join(d["folderName"], d["imageName"])
    img_path = img_path[:-1]
    face_det_crop_resize(img_path)


for i, d in df_human_cv.iterrows():
    img_path = os.path.join(d["folderName"], d["imageName"])
    img_path = img_path[:-1]
    face_det_crop_resize(img_path)


for i, d in df_human_test.iterrows():
    img_path = os.path.join(d["folderName"], d["imageName"])
    img_path = img_path[:-1]
    face_det_crop_resize(img_path)
```

Slika 35. Detekcija lica

Slika 36. Slika prije obrade

Slika 37. Slika nakon obrade

4.4. PRIPREMA BAZE LICA ANMIRANIH LIKOVA

Kao što je prethodno obrađena baza slika s ljudskim slikama, tako će biti obrađena i baza s animiranim likovima uz jednu razliku. Baza s animiranim likovima sadrži preko 50 000 slika što zauzima oko 3 GB memorije. Zato je u programu zadano da se slučajnim odabirom odabere 1000 slika od svake emocije. Na [Slika 38] vidimo da je kôd sličan kao i na [Slika 27]. Razlika je u liniji 39 koja reducira bazu. Odabire 1000 slika slučajnim odabirom, dok ostatak obriše. Na [Slika 39] dobijemo ispis kreirane tablice koji smo već vidjeli na [Slika 28]. Isti postupak se provodi za sve ostale emocije. Na kraju se opet sve spaja u jednu tablicu s naizmjeničnim rasporedom emocija [Slika 40].

```
16  anime_angry = glob.glob("Animated/Angry/*.png")
17  print("Number of images in Angry emotion = "+str(len(anime_angry)))
18
19  anime_angry_folderName = [str(i.split("\\")[0])+"/" for i in anime_angry]
20  anime_angry_imageName = [str(i.split("\\")[1]) for i in anime_angry]
21  anime_angry_emotion = [["Angry"]*len(anime_angry)][0]
22  anime_angry_labels = [1]*len(anime_angry)
23
24  print(len(anime_angry_folderName))
25  print(len(anime_angry_imageName))
26  print(len(anime_angry_emotion))
27  print(len(anime_angry_labels))
28
29  df_angry = pd.DataFrame()
30  df_angry["folderName"] = anime_angry_folderName
31  df_angry["imageName"] = anime_angry_imageName
32  df_angry["Emotion"] = anime_angry_emotion
33  df_angry["Labels"] = anime_angry_labels
34  df_angry.head()
35
36  print(df_angry.head())
37
38  df_angry = df_angry.sample(frac = 1.0)
39  df_angry_reduced = df_angry.sample(n = 1000)
40  df_angry_reduced.shape
```

Slika 38. Sortiranje animirani slika

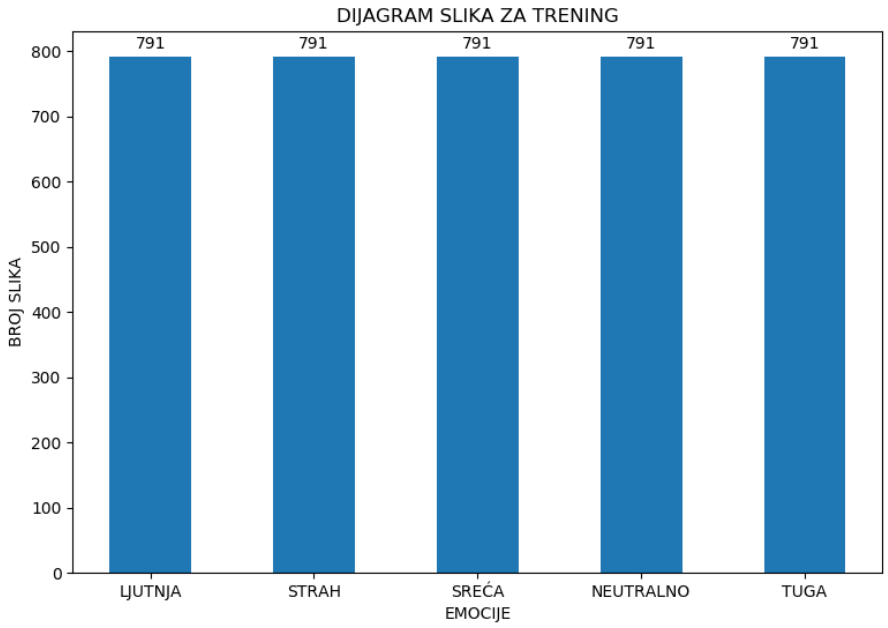
```
Number of images in Angry emotion = 1000
1000
1000
1000
1000
        folderName           imageName Emotion  Labels
0  Animated/Angry/  aia_anger_1001.png   Angry       1
1  Animated/Angry/  aia_anger_1025.png   Angry       1
2  Animated/Angry/  aia_anger_1027.png   Angry       1
3  Animated/Angry/  aia_anger_1031.png   Angry       1
4  Animated/Angry/  aia_anger_1040.png   Angry       1
```

Slika 39. Tablica animiranih slika ljutnje

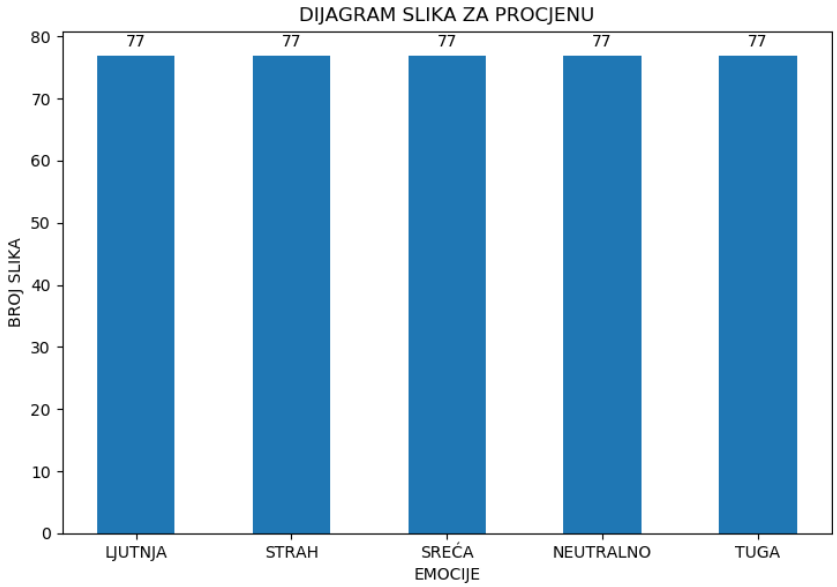
	folderName	imageName	Emotion	Labels
0	Animated/Happy/	bonnie_joy_364.png	Happy	3
1	Animated/Fear/	bonnie_fear_419.png	Fear	2
2	Animated/Happy/	ray_joy_711.png	Happy	3
3	Animated/Neutral/	jules_neutral_996.png	Neutral	4
4	Animated/Angry/	jules_anger_1080.png	Angry	1

Slika 40. Sortirane animirane slike

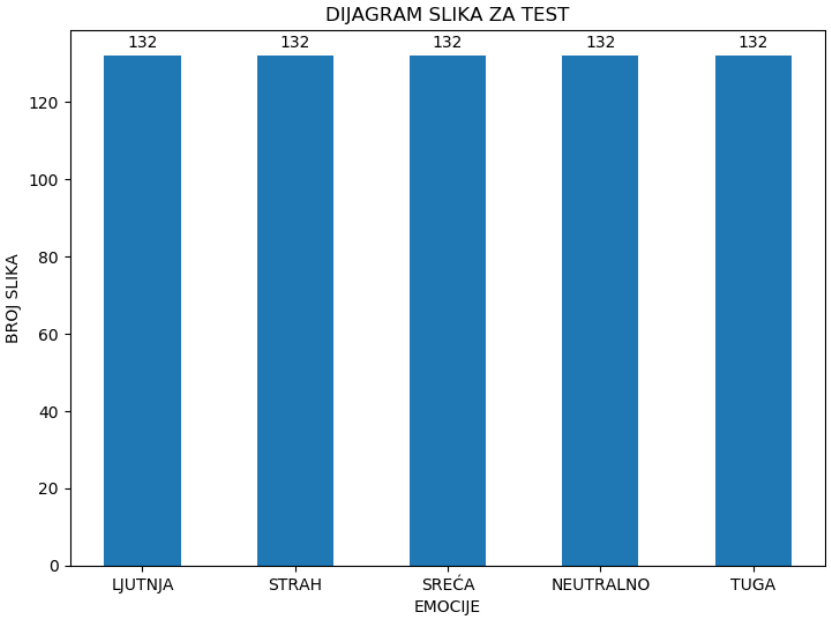
Nakon što je kreirana tablica slika, potrebno ju je podijeliti u 3 kategorije. Slike za trening, slike za procjenu i slike za test. Dijagrami na [Slika 41, 42 i 43] pokazuju tu podjelu i ovaj put primjećujemo da svaka emocija sadrži isti broj slika. To je željeno svojstvo baza i ovdje je uspješno ostvareno. Razlog je veći broj slika u bazi koji olakšava odabir.



Slika 41. Dijagram animiranih slika za trening



Slika 42. Dijagram animiranih slika za procjenu



Slika 43. Dijagram animiranih slika za test

4.5. PROCESIRANJE ANIMIRANIH SLIKA

Isto kao i za prvu bazu, animirane slike je potrebno pretvoriti u sivu skalu, izolirati lice i postaviti iste dimenzije. Kôd je sličan kao na [Slika 34 i 35]. Jedina je razlika u putanjama koje se moraju promijeniti. Rezultat je vidljiv na [Slika 44 i 45] gdje vidimo kako je slika izgledala prije i nakon obrade. Opet je važno uočiti kako je lice sada u krupnom planu i lakše je uočiti detalje.

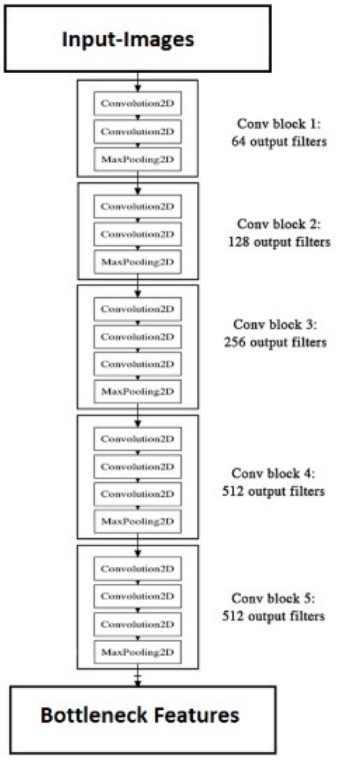
Slika 44. Animirana slika prije obrade

Slika 45. Animirana slika nakon obrade

Nakon procesiranja slika imamo dvije baze slika. Svaka baza je podijeljena u tri dijela: baza za trening, baza za procjenu i baza za testiranje. Baze za trening ćemo spojiti kako bi dobili kvalitetniji model, a baze za procjenu i test ćemo ostaviti podijeljene na bazu slika ljudi i bazu slika animiranih likova. Na taj način će se model moći raznovrsnije testirati. Sljedeći korak je konstrukcija neuronske mreže i njezin trening. Detaljan opis tog postupka je pokazan u sljedećem potpoglavlju.

4.6. KREIRANJE NEURONSKE MREŽE VGG16

Nakon prikupljene i obrađene baze podataka, možemo započeti s konstrukcijom neuronske mreže. Konstruiranje mreže je jako složen zadatak, a problem postaje još i veći ako nemamo dovoljno kompjuterske moći za izvršavanje tog zadatka. Zato se koristi koncept prijenosnog učenja (eng. *transfer learning*) gdje se koriste popularne pretrenirane mreže koje će prepoznati ključne značajke i detalje na slikama. U poglavlju 3.2.1. smo spominjali mrežu VGG16 koju ćemo koristiti pomoću *Pythonovih* biblioteka *TensorFlow* i *Keras*. *TensorFlow* je javno dostupna (eng. *open source*) platforma za strojno učenje. Sadrži fleksibilan sistem alata, biblioteka i javnih resursa koji omogućuju ljudima izradu svojih aplikacija [18]. Dostupan je za nekoliko programskih jezika, pa tako i za *Python*. U sklopu *TensorFlow* koristi se i *Keras* [19]. Keras je API (Application Programing Inteface) napisan u *Pythonu*. Kompatibilan je s *TensorFlowom* i pomoću njega ćemo implementirati već spomenuti pretrenirani model VGG16. Na [Slika 46] vidimo izgled VGG16 modela koji ćemo za potrebe naše mreže malo izmjeniti.



Slika 46. Bottleneck feature

Sa slike vidimo da izlaz iz mreže nisu potpuno spojeni slojevi nego *Bottleneck Features*. U doslovnom prijevodu to bi značilo značajke uskog grla. One sadrže značajke i detalje koji su prepoznati na slici. Zato ćemo svaku sliku provući kroz model u serijama (eng. *batch*) po 10 slika i dobivene podatke pohraniti u memoriju računala. Na [Slika 47] je prikazan dio kôda koji kreira i pohranjuje *Bottleneck Feature*. U liniji 78 je kreiran model VGG16. Parametar *weights='imagenet'* određuje da model ne uči nove težine, već da ih ostavi ovakvima kakve su u originalnom modelu. Drugi parametar *include_top=False* označava brisanje potpuno spojenih slojeve kako bismo kasnije mogli kreirati svoje. Nakon modela, potrebno je definirati mape u koje će se značajke pohraniti.

```
78  model = VGG16(weights='imagenet', include_top=False)
79  SAVEDIR = "Bottleneck_Features/Bottleneck_CombinedTrain/"
80  SAVEDIR_LABELS = "Bottleneck_Features/CombinedTrain_Labels/"
81  batch_size = 10
82  for i in range(int(len(Train_Combined)/batch_size)):
83      x,y = loadCombinedTrainBatch(batch_size)
84      print("Batch {} loaded".format(i+1))
85
86      np.save(os.path.join(SAVEDIR_LABELS, "bottleneck_labels_{}".format(i+1)), y)
87
88      print("Creating bottleneck features for batch {}".format(i+1))
89      bottleneck_features = model.predict(x)
90      np.save(os.path.join(SAVEDIR, "bottleneck_{}".format(i+1)), bottleneck_features)
91      print("Bottleneck features for batch {} created and saved\n".format(i+1))
```

Slika 47. Kôd za kreiranje Bottleneck Featurea

Varijabla *batch_size* određuje koliko će biti velike serije slika. Iz [Slika 47] vidimo da će svaka serija imati po 10 slika. Sljedeći korak je korištenje programske strukture *For petlje*. Ona će ponavljati postupak dok sve slike ne prođu kroz model. Varijabla *Train_Combined* sadrži sve slike namijenjene za trening. U liniji 83 poziva se funkcija *loadCombinedTrainBatch*. Njezin ulazni parametar je varijabla *batch_size*. Njezin kôd je prikazan na [Slika 48]. Ona kreira dvije liste: *batch_images* i *batch_labels*. U prvu listu zapisuje učitane slike u obliku matrice, a u drugu listu se zapisuje oznaka za emociju koju ta slika predstavlja. Na izlazu u liniji 74 funkcija vraća dva niza koji sadrže slike i njihove oznake. Sljedeći korak je linija 89 gdje slike prolaze kroz model i zapisuju pronađene značajke u varijabli *bottleneck_features*. Na kraju se značajke slika i njihove oznake spremaju u prethodno odabrane mape pod određenim rednim brojem (linije 86 i 90). Procedura se provodi sve dok sve slike ne prođu kroz model, a postupak je isti i za slike koje se koriste za procjenu i test modela.

```
57  def loadCombinedTrainBatch(batch_size):
58      global TrainCombined_batch_pointer
59      batch_images = []
60      batch_labels = []
61      for i in range(batch_size):
62          path1 = Train_Combined.iloc[TrainCombined_batch_pointer + i]["folderName"]
63          path2 = Train_Combined.iloc[TrainCombined_batch_pointer + i]["imageName"]
64          if path2[-1] == "/":
65              path2 = path2[:-1]
66          read_image = cv2.imread(os.path.join(path1,path2))
67          read_image_final = read_image/255.0
68          batch_images.append(read_image_final)
69
70          batch_labels.append(TrainCombined_Labels[TrainCombined_batch_pointer + i])
71
72      TrainCombined_batch_pointer += batch_size
73
74      return np.array(batch_images), np.array(batch_labels)
```

Slika 48. Funkcija loadCombinedTrainBatch

Nakon što su kreirane i pohranjene značajke za svaku sliku iz baze podataka, možemo trenirati novu mrežu koja će sadržavati potpuno spojene slojeve. Na njezinom će ulazu biti matrica određenih dimenzija koja sadrži pronađene značajke, a na izlazu će biti procjena emocije. Na ovaj način su iskorištene prednosti *transfer learninga* [12], [20] i mreža postaje jednostavnija i brža za korištenje.

4.7. TRENING MREŽE

Prije završne faze treninga moramo kreirati mrežu koja će učiti razliku između emocija. Na [Slika 49] vidimo kako izgleda model. Model je sekvencijalan što znači da se kreira sloj po sloj. Prvi sloj je kreiran u liniji 27. *Dense* označava potpuno spojeni sloj. Aktivacijska funkcija mu je *relu*, a ulazne dimenzije ovise o dimenzijama ulaznih matrica. U liniji 28 je dodana opcija *Dropout*. Ona se koristi kako bi se izbjeglo preklapanje (eng. *overfitting*) tako što odbacuje određen postotak naučenih težina i ponovno ih uči. Preklapanje je greška u učenju neuronskih mreža koja nastaje kada mreža „napamet" nauči primjere. Kada se pojave novi primjeri, mreža ih pogađa i nastaju velike greške u rezultatima. *Dropout* je jedan od načina rješavanja preklapanja. Nakon njega slijede još dva sloja od 256 i 128 neurona i nakon njih dolazimo do linije 33 i *BatchNormalizationa*. *BatchNormalization* se koristi zbog mogućnosti da su velike razlike između iznosa aktivacijskih funkcija. *BatchNormalization* te vrijednosti skalira na manju vrijednost (na primjer između 0 i 1) i na taj način stabilizira mrežu i ubrza njezin trening. Zadnja dva sloja su opet *Dense* slojevi. Zadnji sloj se razlikuje od prethodnih. Prva razlika je što mu izlazna dimenzija mora biti jednaka broju mogućnosti koje mreža prepoznaje. S obzirom na to da želimo da mreža prepozna razliku između 5 emocija, izlazna dimenzija mora biti 5. Druga razlika je aktivacijska funkcija koja je u ovom slučaju *softmax*. Ona određuje u kojem postotku mreža povezuje pojedinu sliku s određenom emocijom.

```
24  def model (input_shape):
25      model = Sequential()
26
27      model.add(Dense(512, activation = 'relu', input_dim = input_shape))
28      model.add(Dropout(0.1))
29
30      model.add(Dense(256, activation = 'relu'))
31
32      model.add(Dense(128, activation = 'relu'))
33      model.add(BatchNormalization())
34
35      model.add(Dense(64, activation = 'relu'))
36      model.add(Dense(output_dim = no_of_classes, activation = 'softmax'))
37
38      return model
```

Slika 49. Model neuronske mreže

Sada kada je kreiran model neuronske mreže, dolazimo do dijela kôda gdje se izvršava trening. Kôd se nalazi na [Slika 50]. Prvi korak je *For* petlja koja vrti krugove treninga (eng. *epoch*). Zatim imamo varijable *avg_epoch_CombTr_acc*, *avg_epoch_CombTr_loss* i druge, pomoću kojih ćemo računati iznose točnosti i grešaka. U liniji 70 se nalazi još jedna *For* petlja koja će osigurati da u svakom krugu treninga svaka serija slika prođe kroz model. Imamo tri skupine podataka. Prva koja služi za trening i sadrži miješane slike ljudi i anmiranih likova. Druga koja se sastoji od slika ljudi i služi za procjenu mreže i treća koja se sastoji od animiranih slika i također služi za procjenu mreže. U liniji 74 se učitava prva serija slika. Te se slike redimenzioniraju u oblik koji zahtijeva ulaz u mrežu i spremaju se u varijablu *X_CombTrain* (linija 75). Varijabla *Y_CombTrain* učitava oznake koje određuju o kojoj se emociji radi. Isti potupak se provodi od linije 78 do linije 84 s podacima za procjenu. Nakon toga je sve spremno za trening. U liniji 86 se koristi napredba *model.train_on_batch* čiji su parametri *X_CombTrain* i *Y_CombTrain*. Na izlazu dobijamo varijable *CombTrain_Loss* i *CombTrain_Accuracy* pomoću kojih možemo analizirati točnost i grešku treninga našeg modela. U linijama 87 i 88 se koristi naredba *model.test_on_batch* koja za ulazne parametre ima *X_CVHuman*, *Y_CVHuman*, odnosno *X_CVAnime* i *Y_CVAnime*. To su slike i njihove oznake namijenjene za procjenu mreže. Izlazne varijable su *CVHuman_Loss*, *CVHuman_Accuracy*, *CVAnime_Loss* i *CVAnime_Accuracy*. Pomoću tih varijabla će se izvući dodatni podaci za analizu dobivenog modela. Ovaj se postupak provodi za svaku seriju spremljenih slika. Kada su sve serije slika prođu mrežu, gotov je jedan krug treninga mreže i slijedi novi. Za ovakve primjere je dovoljno da se model trenira u 10 do 20 krugova, što u ovisnosti o jačini kompjutera može potrajati 3-6 sati.

```
66  for epoch in range(epochs):
67      avg_epoch_CombTr_loss, avg_epoch_CombTr_acc, avg_epoch_CVHum_loss, avg_epoch_CVHum_acc, avg_epoch_CVAnime_loss, avg_epoch_CVAnime_acc = 0,0,0,0,0,0
68      epoch_number.append(epoch+1)
69
70      for i in range(combTrain_bottleneck_files):
71
72          step += 1
73
74          X_CombTrain_load = np.load(os.path.join(SAVEDIR_COMB_TRAIN, "bottleneck_{}.npy".format(i+1)))
75          X_CombTrain = X_CombTrain_load.reshape(X_CombTrain_load.shape[0], X_CombTrain_load.shape[1]*X_CombTrain_load.shape[2]*X_CombTrain_load.shape[3])
76          Y_CombTrain = np.load(os.path.join(SAVEDIR_COMB_TRAIN_LABELS, "bottleneck_labels_{}.npy".format(i+1)))
77
78          X_CVHuman_load = np.load(os.path.join(SAVEDIR_CV_HUMANS, "bottleneck_{}.npy".format((i % CVHuman_bottleneck_files)+1)))
79          X_CVHuman = X_CVHuman_load.reshape(X_CVHuman_load.shape[0], X_CVHuman_load.shape[1]*X_CVHuman_load.shape[2]*X_CVHuman_load.shape[3])
80          Y_CVHuman = np.load(os.path.join(SAVEDIR_CV_HUMANS_LABELS, "bottleneck_labels_{}.npy".format((i % CVHuman_bottleneck_files)+1)))
81
82          X_CVAnime_load = np.load(os.path.join(SAVEDIR_CV_ANIME, "bottleneck_{}.npy".format((i % CVAnime_bottleneck_files)+1)))
83          X_CVAnime = X_CVAnime_load.reshape(X_CVAnime_load.shape[0], X_CVAnime_load.shape[1]*X_CVAnime_load.shape[2]*X_CVAnime_load.shape[3])
84          Y_CVAnime = np.load(os.path.join(SAVEDIR_CV_ANIME_LABELS, "bottleneck_labels_{}.npy".format((i % CVAnime_bottleneck_files)+1)))
85
86          CombTrain_Loss, CombTrain_Accuracy = model.train_on_batch(X_CombTrain, Y_CombTrain)
87          CVHuman_Loss, CVHuman_Accuracy = model.test_on_batch(X_CVHuman, Y_CVHuman)
88          CVAnime_Loss, CVAnime_Accuracy = model.test_on_batch(X_CVAnime, Y_CVAnime)
89
90          print("Epoch: {}, Step: {}, CombTr_Loss: {}, CombTr_Acc: {}, CVHum_Loss: {}, CVHum_Acc: {}, CVAnime_Loss: {}, CVAnime_Acc: {}"
91          .format(epoch, step, CombTrain_Loss, CombTrain_Accuracy, CVHuman_Loss, CVHuman_Accuracy, CVAnime_Loss, CVAnime_Accuracy))
92
93          avg_epoch_CombTr_loss += CombTrain_Loss / combTrain_bottleneck_files
94          avg_epoch_CombTr_acc += CombTrain_Accuracy / combTrain_bottleneck_files
95          avg_epoch_CVHum_loss += CVHuman_Loss / combTrain_bottleneck_files
96          avg_epoch_CVHum_acc += CVHuman_Accuracy / combTrain_bottleneck_files
97          avg_epoch_CVAnime_loss += CVAnime_Loss / combTrain_bottleneck_files
98          avg_epoch_CVAnime_acc += CVAnime_Accuracy / combTrain_bottleneck_files
99
100     print("Avg_CombTrain_Loss: {}, Avg_CombTrain_Acc: {}, Avg_CVHum_Loss: {}, Avg_CVHum_Acc: {}, Avg_CVAnime_Loss: {}, Avg_CVAnime_Acc: {}"
101     .format(avg_epoch_CombTr_loss, avg_epoch_CombTr_acc, avg_epoch_CVHum_loss, avg_epoch_CVHum_acc, avg_epoch_CVAnime_loss, avg_epoch_CVAnime_acc))
```

Slika 50. Trening mreže

4.8. BIBLIOTEKA PYAUDIO

Kada je kreiran model neuronske neuronske mreže (potpoglavlje 4.7.), imamo alat za prepoznavanje emocija pomoću mimike lica. Sljedeća stvar koja je potrebna je mjerenje razine buke u prostoriji. Za to će se koristiti *Pythonova* biblioteka *PyAudio* [21]. Pomoću *PyAudia* će se mjeriti razina buke u prostoriji preko decibela. Na [Slika 51] je kôd koji pokazuje kako namjestiti snimanje zvuka. Linija 27 sadrži varijablu *ref* koja će biti referentna vrijednost za prikazivanje decibela. Varijabla *FORMAT* određuje u kojem će se formatu snimati zvuk. *CHANNELS* određuje da će se snimati monokormatski odnosno preko ugrađenog mikrofona. Još su bitne varijable *RATE* i *CHUNK* koje određuju koliko se snima uzoraka po sekundi, odnosno uzoraka po okviru (eng. *frame*). U liniji 35 se kreira objekt *audio*, čije se postavke namještaju u linijama 38-45 pomoću već spomenutih varijabli. Važan je parametar *stream_callback* koji poziva funkciju *callback*. Ona vrši obradu snimljenog zvuka. Linija 48 pokreće snimanje.

```
27  ref = 32768
28  FORMAT = pyaudio.paInt16
29  CHANNELS = 1
30  RATE = 16000
31  CHUNK = 2**10
32  DEVICE_ID = 0
33
34  frames = []
35  audio = pyaudio.PyAudio()
36  ham = np.hamming(CHUNK)
37
38  stream = audio.open(
39      format=FORMAT,
40      channels=CHANNELS,
41      rate=RATE,
42      input=True,
43      input_device_index=DEVICE_ID,
44      frames_per_buffer=CHUNK,
45      stream_callback=callback
46  )
47  print ("start...")
48  stream.start_stream()
```

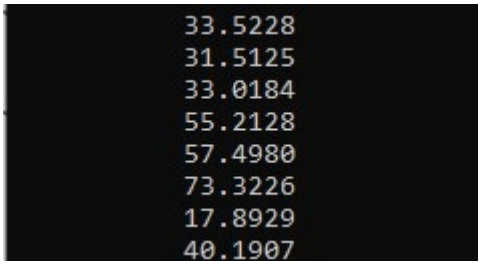
Slika 51. PyAudio postavke

Na [Slika 52] vidimo *callback* funkciju koja vrši obradu snimljenog zvuka. Od ulaznih parametara nam je bitan *in_data* jer sadrži snimljene podatke. Varijabla *frames* je globalna i u nju se zapisuju dobiveni podaci. Dobiveni podaci su u obliku stringa i potrebno ih je prvo pretvoriti u brojeve (linija 12). Nakon toga se nad tim podacima vrši Fourierova transformacija. Sljedeći dio kôda se bavi pretvorbom dobivenih rezultata u decibele [22]. Na kraju je ispis glasnoće u realnom vremenu koji traje sve dok se program ne prekine.

```
9   def callback(in_data, frame_count, time_info, status):
10      global frames
11
12      frames = np.fromstring(in_data, dtype=np.int16)
13      data_f = np.fft.rfft(frames)
14      s_mag = np.abs(data_f) * 2/np.sum(ham)
15      s_dbfs = 20*np.log10(s_mag/ref)
16      K = 120
17      s_db = s_dbfs + K
18
19      db = np.mean(s_db)
20
21      sys.stdout.write("\r%18.4f" % db)
22      sys.stdout.flush()
23
24      return (None, pyaudio.paContinue)
```

Slika 52. PyAudio postavke

Na [Slika 54] vidimo nekoliko ispisa glasnoće u prostoriji. Rezultati su prikazani u decibelima, a provjera točnosti očitanja je rađena pomoću mobilne aplikacije. Na [Slika 15] smo vidjeli tablicu koja povezuje glasnoću s određenim načinom govora, pa će nam u nastavku ta tablica i ovdje navedena očitanja pomoći u kreiranju procjene emocija.



Slika 53. Očitanja glasnoće

4.9. CALC_OPTICAL_FLOW_FARNEBACK

U potpoglavlju 3.4. su objašnjeni neki detalji vezani za računanje inteziteta gibanja (eng. *optical flow*). Pythonova biblioteka *OpenCv* [23] sadrži funkciju *calcOpticalFlowFarneback*. Ona koristi Gunnar Farnebackov algoritam [10] za određivanje gibanja. Na [Slika 54] vidimo jednostavan kôd koji opisuje način rada funkcije. U liniji 12 vidimo funckiju *calcOpticalFlowFarneback* koja ima deset ulaznih parametara. Video možemo opisati kao niz slika, pa algoritam radi tako da uspoređuje dvije susjedne slike i na temelju te usporedbe određuje intenzitet gibanja. Prva dva parametra *prevgray* i *gray* predstavljaju prethodnu i trenutnu sliku. Ostali parametri služe za podešavanje algoritma i preuzeti su iz primjera koje je dala službena stranica *OpenCv-a*. Izlaz funkcije je matrica iste veličine kao i ulazne slike, gdje svaki element matrice predstavlja promjenu intenziteta svakog piksela na slici. Ta se matrica sprema u varijablu *flow*. U liniji 14 se sve vrijednosi matrice pretvaraju u apsolutnu vrijednost, a u liniji 15 se računa suma svih članova te matrice. U liniji 16 se ispisuju vrijednosti intenziteta gibanja, a na [Slika 55] se vidi kako izgleda ispis.

```
5   cam = cv2.VideoCapture(cv2.CAP_DSHOW + 0)
6   ret, prev = cam.read()
7   prevgray = cv2.cvtColor(prev, cv2.COLOR_BGR2GRAY)
8
9   while True:
10      ret, img = cam.read()
11      gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
12      flow = cv2.calcOpticalFlowFarneback(prevgray, gray, None, 0.5, 3, 15, 3, 5, 1.2, 0)
13      prevgray = gray
14      a = np.abs(flow)
15      a = np.sum(a)
16      print("\r%10.1f"  % a)
```

Slika 54. calcOpticalFlowFarneback

```
 80603.3
 88188.9
225557.9
 73722.6
234448.2
248461.6
103350.2
466711.3
103699.6
430229.6
168204.4
292186.9
107220.2
238876.1
155661.8
220816.0
```

Slika 55. Ispis inteziteta gibanja

[Slika 55] pokazuje nekoliko ispisa inteziteta gibanja. Što je gibanje intenzivnije to je iznosi veći. Važno je napomenuti da je algoritam osjetljiv na svjetlost i pozadinska gibanja. Zato je prije upotrebe algoritma potrebno kalibrirati vrijednosti. Iz navedenog primjera sa slike vidimo da slučaj u kojem nema gibanja predstavljaju brojevi između 70 000-80 000. Kada se gibanje pojavi, te vrijednosti se povećavaju.

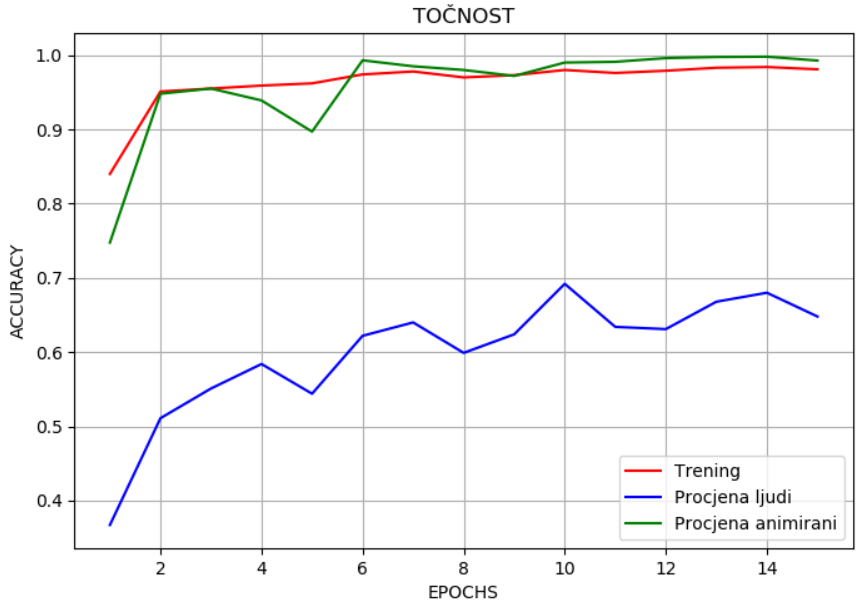
U ovom poglavlju je prikazana izvedba zadatka. Objašnjeni su najbitniji dijelovi kôda i način na koji oni funkcioniraju. U prilogu ovog rada će biti napisan cijeli kôd s komentarima kako bismo razjasnili sve nejasnoće koje su se možda pojavile u ovom poglavlju. Sljedeće poglavlje pokazuje dobivene rezultate.

5. REZULTATI

U ovom poglavlju će biti prikazani dobiveni rezultati. Podijeljeni su u tri dijela. Prvi dio će prikazati grafove i matrice dobivene na temelju testnih slika. Drugi dio će testirati jednomodalnih pristup, odnosno procjenu emocija temeljenu samo na mimici lica. Treći dio koji će prikazati rezultate multimodalnog pristupa.

5.1. PROCJENA I TEST MREŽE

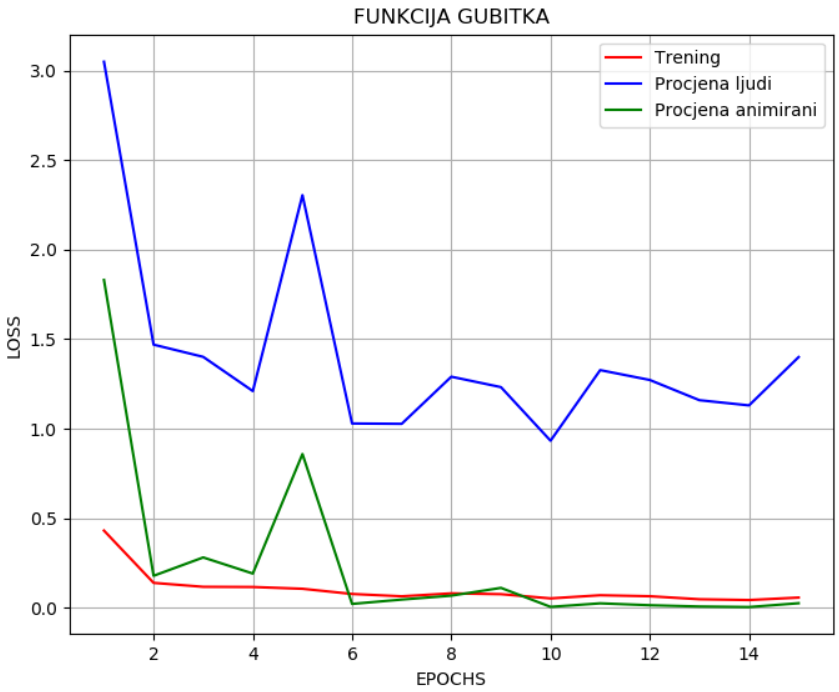
Kao što je navedeno u potpoglavlju 4.2. baza slika je podijeljena u 3 kategorije. Prvi i najveći dio baze slika se koristi za trening mreže, dok se drugi i treći dio baze slika koristi za procjenu i test modela. Baza slika za procjenu i test obično čini 5-20% ukupne veličine baze i sve slike u njima su autentične, odnosno ista slika se nigdje ne pojavljuje dva puta. Na [Slika 56] vidimo graf točnosti. Na osi apcisa se nalazi broj kruga treninga (eng. *epochs*), a na osi ordinata dobivena točnost (*eng. accuracy*). Crvena linija predstavlja podatke dobivene od baze za trening, dok plava i zelena linija predstavljaju podatke dobivene sa slikama namijenjenim za procjenu mreže. Cilj je postići što veću točnost. Podaci za trening vrlo brzo konvergiraju prema konačnih 98% uspješnosti, dok je kôd slika za procjenu primjetna velika razlika između slika ljudi i slika animiranih likova. Procjena animiranih likova je nakon 15 krugova treninga stala na odličnih 99%, a procjena slika ljudi nakon 10. kruga varira između 65 i 70%. To je vrlo dobar rezultat iz dva razloga. Problem nedostatka slika ljudi riješen je uvrštavanjem slika animiranih likova koji na kraju čine veći dio baze. Drugi razlog je to što su animirane slike bolje kvalitete od slika ljudi. Kada se slikamo, uvijek postoje određene smetnje na slikama i iz tog razloga su slike ljudi lošije. Ipak, rezultati su zadovoljavajući i graf uz nekoliko iznimki pokazuje kontinuirani rast točnosti modela.



Slika 56. Graf točnosti

Na [Slika 57] je prikazan graf koji pokazuje funkciju gubitka. Na osi ordinata se ovoga puta nalazi funkcija gubitka (eng.*loss*). Razlika između ovog i prethodnog grafa je u tome što je cilj funkcije gubitka da je svakim krugom njezina vrijednost sve manja i manja. Zaključci doneseni za [Slika 56] mogu se primjetiti i ovdje. Vidljiva je razlika između slika ljudi i slika animiranih likova. Plava linija bi sigurno imala manji iznosi kada bi se baza proširila s dodatnim slikama ljudi. Unatoč tom problemu, rezultati su zadovoljavajući, a iz grafa se vidi da podaci za trening i procjenu animiranih likova konvergiraju prema nuli. Razlike u vrijednosti funkcije između prvog i zadnjih kruga iznose:

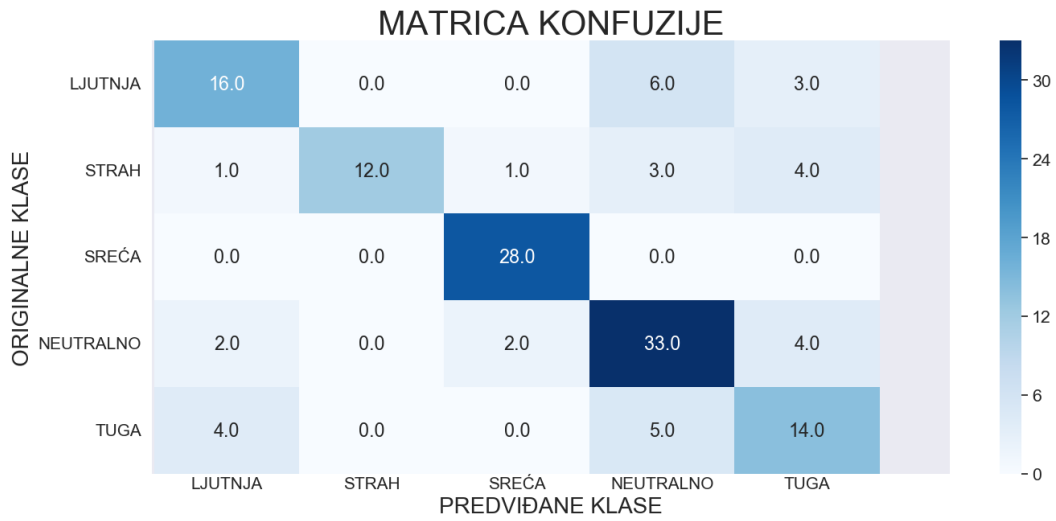
- Trening: 0,43 - 0,056
- Procjena ljudi: 3,04 - 1,4
- Procjena animirani: 1,83 - 0,025



Slika 57. Funkcija gubitka

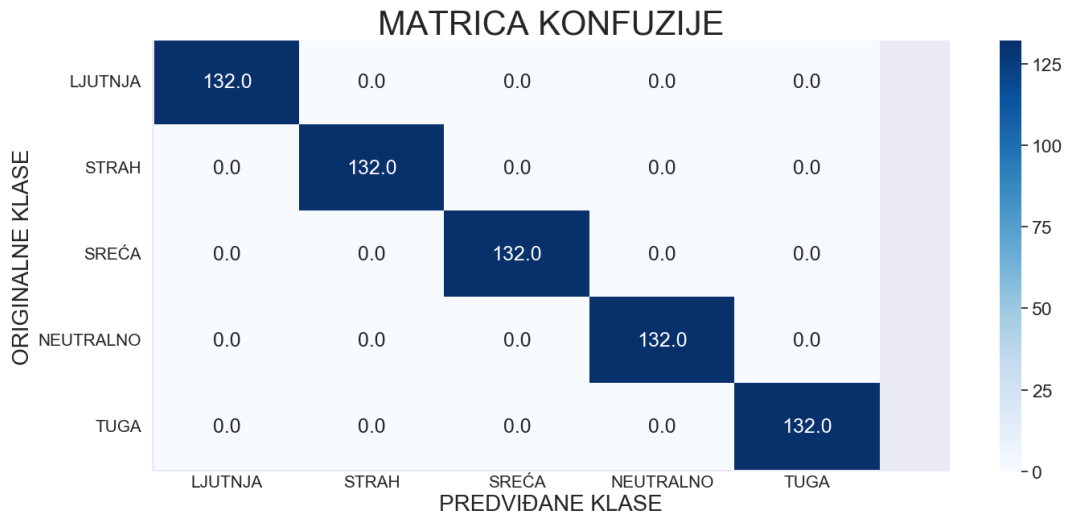
Kako bi se dobili prethodno prikazani grafovi koristila se baza slika za procjenu. Sljedeći podaci dobiveni su pomoću baze slika namjenjene za test. Broj slika u bazi za test prikazan je u 4. poglavlju [Slika 33 i 43], a [Slika 58 i 59] prikazuju matricu konfuzije (eng. *confusion matrix*). Matrica konfuzije se često naziva i matrica pogreške. Ona se često koristi u strojnom učenju i neuronskim mrežama kada je prisutna klasifikacija. Pomaže u procjeni ako mreža ima problem razlikovati dvije klase. U ovom radu imamo 5 emocija koje predstavljaju 5 klasa. Redovi matrice predstavljaju originalne klase, a stupci predviđane. Na taj se način može ocjeniti kako mreža predviđa klase u usporedbi s originalnim. Na dijagonali matrice je prikazan broj točnih predviđanja.

[Slika 58] prikazuje konfuzijsku matricu kreiranu sa slikama ljudi. Dobivena je točnost od 74,64%. Najbolji postotak predviđanja imaju emocija sreće i neutralno. Sljedeća emocija koja se najbolje predviđa je emocija ljutnje. Ipak, matrica pokazuje da se emocija ljutnje često zna zamijeniti s emocijama za neutralno i tugu, dok emocije straha i tuge imaju najveći broj pogrešaka. Iako imaju veći broj točnih odgovora, vidimo da se emocija straha često miješa s emocijama tuge i neutralno, dok se tuga miješa s emocijom ljutnje i neutralno. Razlog tome je kompleksnost same emocije kada se radi o mimici lica. Emocija sreće i neutralno su prilično jednostavne, dok ostale emocije imaju kompleksniju mimiku, pa je samim time i veći broj pogrešaka. Najbitnije je da mreža za svaku emociju ima više točnih nego netočnih odgovora.



Slika 58. Matrica konfuzije (slike ljudi)

Na [Slika 59] se nalazi matrica konfuzije dobivena od testnih slika za animirane likove. Iz slike vidimo da je točnost 100%. Razlog tome je puno veći broj slika za trening i test mreže kao i sama kvaliteta slika. Slike su puno jasnije, nemaju šuma i to olakšava posao neuronskoj mreži. Zato matrica konfuzije za ovaj slučaj pokazuje savršene rezultate.



Slika 59. Matrica konfuzije (animirane slike)

5.2. JEDNOMODALNI PRISTUP

U prethodnom potpoglavlju su prikazani statistički podaci kako bi dobili uvid u kvalitetu neuronske mreže. Vidjeli smo da su rezultati poprilično zadovoljavajući, ali da mreža sadrži određene nedostatke. U ovom potpoglavlju će se mreža testirati s nekoliko slika odabranih na internetu.

Slika 60. Ljutnja

```
PREDVIĐANJE
LJUTNJA: 0.8156912326812744
STRAH: 0.02042628452181816
SREĆA: 0.08656582981348038
NEUTRALNO: 0.02720615081489086
TUGA: 0.05011054500937462

DOMINANTNA EMOCIJA = LJUTNJA: 0.81569123
```

Slika 61. Procjena emocije ljutnje

[Slika 60] prikazuje čovjeka koji pokazuje emociju ljutnje. Namrgođeno lice i agresivan stav to potvrđuju. Procjena modela prikazana je na [Slika 61] i vidimo da mreža tvrdi da se radi o emociji ljutnje s 81,5%. Zatim slijeda sreća s 8% i tuga s 5%. U ovom slučaju je mreža poprilično sigurna u svoju procjenu i rezultat je točan.

Slika 62. Neutralno

```
LJUTNJA: 0.035222675651311874
STRAH: 0.0033992312382906675
SREĆA: 0.30816298723220825
NEUTRALNO: 0.5730711221694946
TUGA: 0.08014395087957382


DOMINANTNA EMOCIJA = NEUTRALNO: 0.5730711
```

Slika 63. Procjena emocije neutralno

Primjer na [Slika 62] je ipak malo kompleksniji. Osoba na slici ima blagi smiješak i zapravo je stvar subjektivnog dojma o kojoj se emociji radi. Na [Slika 63] je prikazana procjena modela. Model tvrdi s 57,3% da je dominantna emocija neutralno, dok je druga dominantna emocija sreća s 30,8%. Iz ovog primjera vidimo da rješenje ne mora biti jednolično, odnosno da je moguća kombinacija više emocija koje predstavlja konačno rješenje.

Slika 64. Strah

```
Predicted Expression Probabilities
ANGRY: 0.012713826261460781
FEAR: 0.6794090270996094
HAPPY: 0.0048047942109405994
NEUTRAL: 0.15049020946025848
SAD: 0.15258218348026276


Dominant Probability = FEAR: 0.679409
```

Slika 65. Procjena emocije strah

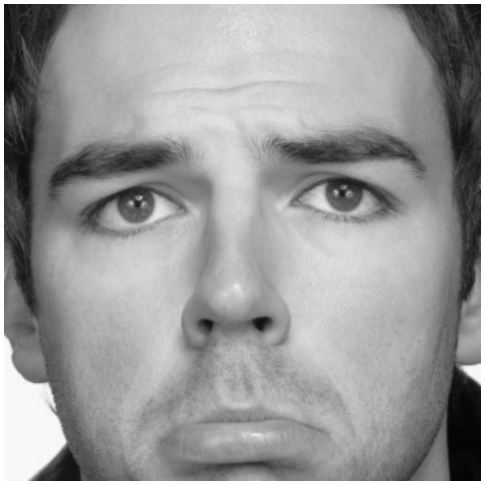
[Slika 64] prikazuje osobu koja simulira emociju straha. Procjena modela na [Slika 65] to i potvrđuje sa 67,9% uspješnosti. Ono što je zanimljivo u ovom primjeru jest što je model dao 15% vjerojatnosti da su druga i treća dominantna emocija tuga i neutralno. U matrici konfuzije na [Slika 59] smo vidjeli kako model zna imati problema kada procjenjuje emociju straha i pomiješati je s emocijama tuge i neutralno. Ovaj primjer to potvrđuje, ali je na kraju model ipak ispravno procjenio.

Slika 66. Tuga

```
Predicted Expression Probabilities
ANGRY: 0.006992314010858536
FEAR: 0.00050005799985789955
HAPPY: 2.720404700085055e-05
NEUTRAL: 0.00033500586869195104
SAD: 0.9921453595161438


Dominant Probability = SAD: 0.99214536
```

Slika 67. Procjena emocije tuga

[Slika 66] prikazuje čovjeka s emocijom tuge. Procjena modela je na [Slika 67] i za ovaj primjer model tvrdi s 99,2 % da je dominatna emocija tuga. U ovom primjeru vidimo kako model nema dilema i da je sa skoro stopostotnom sigurnošću točno procijenio o kojoj se emociji radi.



Slika 68. Tuga2

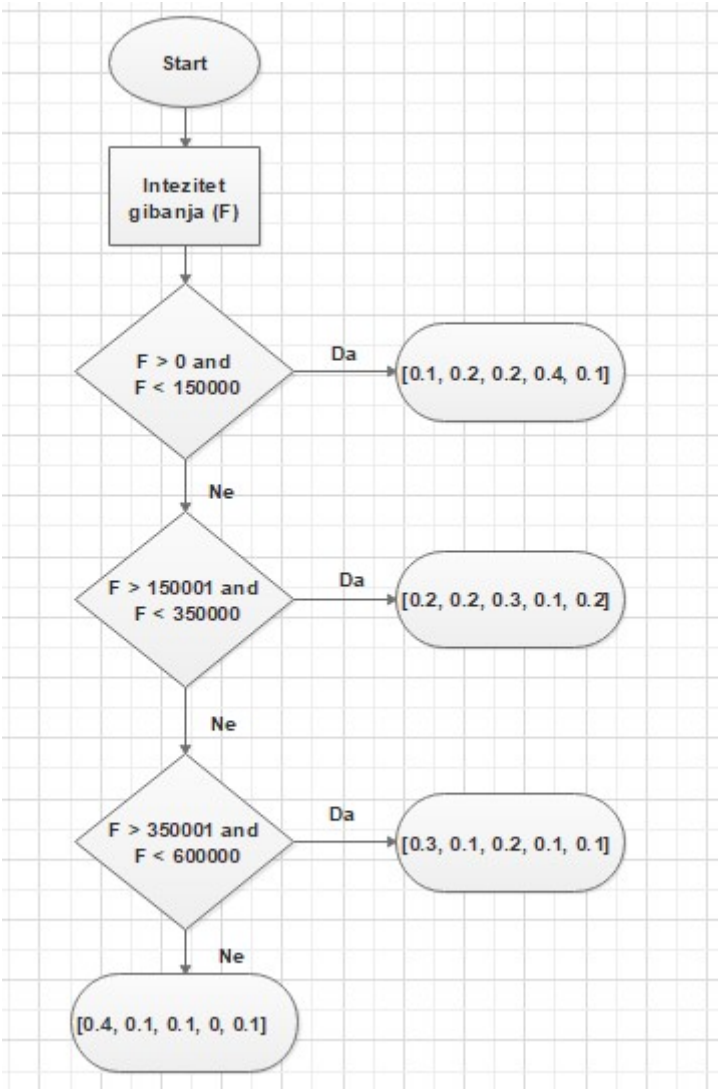
```
PREDVIĐANJE
LJUTNJA: 0.8115992546081543
STRAH: 0.00027745560510084033
SREĆA: 0.0011022905819118023
NEUTRALNO: 0.04094374552369118
TUGA: 0.1460772454738617


DOMINANTNA EMOCIJA = LJUTNJA: 0.81159925
```

Slika 69. Kriva procjena emocije tuga

Da nije sve idealno pokazuje [Slika 68]. Na slici imamo emociju tuge, ali ovaj puta njezina procjena nije dobra [Slika 69]. Model tvrdi s 81,1 % da je riječ o emociji ljutnje, a druga dominantna emocija je tuga s 14,6 %. Iako je druga emocija tuga, ovdje se radi o dosta velikoj pogrešci modela.
U ovom potpoglavlju je prikazano nekoliko primjera kako bismo testirali model. Model uglavnom daje rezultate koji favoriziraju jednu emociju. Prisutne su i neke iznimke, ali je model uglavnom siguran u svoju procjenu. Slike koje su korištene u ovim primjerima su visoke kvalitete što je svakako pomoglo u dobivanju boljih rezultata.

5.3. MULTIMODALNI PRISTUP

U prethodnim potpoglavljima smo vidjeli kako funkcionira jednomodalni pristup ovom problemu. Procjene su dobre, ali želimo da rezultat uključuje više emocija. Takav pristup je poželjan iz razloga što jako rijetko osjećamo samo jednu emociju. Zato postoji multimodalni pristup koji će na temelju dodatnih informacija drugačije vrednovati različite emocije. U ovom radu, multimodalni pristup se sastoji od tri segmenta na temelju kojih se donosi konačni zaključak. Radi se o analizi mimike, analizi inteziteta pokreta i o razini buke u prostoriji. Analiza mimike je objašnjena u prethodnom potpoglavlju, dok se analiza pokreta i buke prvi puta koriste. Zato je prije prikaza rezultata objašnjen princip na kojem oni rade.

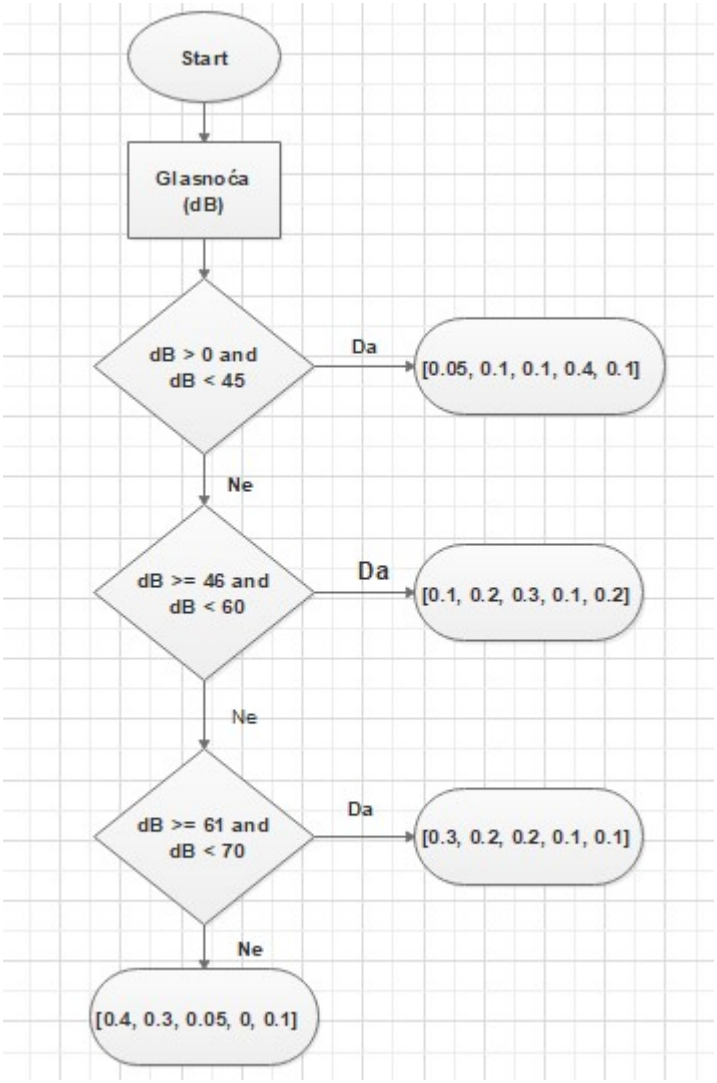


Slika 70. Dijagram toka za intezitet gibanja

```
64  def flow_det(flow_a):
65      flow_a = round(flow_a)
66      if flow_a>0 and flow_a<=150000:
67          flow_scale = np.array([0.1, 0.2, 0.2, 0.4, 0.1])
68      elif flow_a>=150001 and flow_a<=350000:
69          flow_scale = np.array([0.2, 0.2, 0.3, 0.1, 0.2])
70      elif flow_a>=350001 and flow_a<=600000:
71          flow_scale = np.array([0.3, 0.1, 0.2, 0.1, 0.1])
72      elif flow_a>=600001 and flow_a<=15000000:
73          flow_scale = np.array([0.4, 0.1, 0.1, 0, 0.1])
74      else:
75          print("Something is wrong")
76
77      return flow_scale
```

Slika 71. Funkcija za skaliranje inteziteta gibanja

[Slika 70] prikazuje dijagram toka koji opisuje način vrednovanja dobivenog inteziteta gibanja s favoriziranjem određene emocije. [Slika 71] prikazuje kôd koji izvršava prikazani tok. Funkcija *flow_det* čija je ulazna vrijednost detektirani intenzitet gibanja dobivena je od funkcije *calcOpticalFlowFarneback*. Nakon učitanja ulazne vrijednosti, funkcija *flow_det* pomoću if programske strukture skalira tu vrijednost i na temelju dobivenog rezultata favorizira određenu emociju. Emocije se favoriziraju putem niza brojeva kao što je prikazano u linijama 67, 69, 71 i 73. Svaki broj u nizu predstavlja jednu emociju, a to su redom: ljutnja, strah, sreća, neutralno i tuga. Prvi slučaj u liniji 66, kada je dobivena vrijednost između 0 i 150 000 daje sljedeći niz: [0,1 0,2, 0,2 0,4 0,1]. Vidimo da ovaj niz kao dominantnu emociju ističe neutralno. Za različite intenzitete, različite emocije dolaze do izražaja. Veći intenzitet povezujemo s emocijom ljutnje i što je on veći, time ljutnja postaje izraženija. Emocije poput sreće, straha i tuge su karakteristične za srednje vrijednosti inteziteta gibanja. Skala kreirana na slici je plod testiranja i proučavanja raznih stručnih radova. Nigdje ne postoji rad koji tvrdi da određeni intenzitet treba povezati s nekom od emocija. Zato je ova skala osobni odabir. Također treba spomenuti kako je skala podložna promjenama ovisno o mjestu snimanja rezultata. Razlog tome su pozadinske kretnje i svjetlost koji utječu na očitavanje rezultata.
[Slika 72] prikazuje dijagram toka koji prikazuje vrednovanje izračunate glasnoće. [Slika 73] prikazuje kôd koji izvršava dijagram toka za glasnoću. Funkcija *zvuk_scale* kao ulazni parametar prima očitanu glasnoću u decibelima dobivenu pomoću biblioteke *PyAudio*. Zatim dobivenu vrijednost skalira pomoću *if* programske strukture. Na [Slika 15] smo vidjeli kako se različiti iznosi decibela povezuju s određenim načinom govora. Tako se recimo 40 dB povezuje s tihim govorom. Ti su podaci pomogli kôd kreiranja ove skale. Na [Slika 73] vidimo da je za niže iznose glasnoće dominantna emocija neutralno. Kod srednjih vrijednosti to je sreća jer se u tim granicama kreće glasnoća normalnog smijeha. U tim granicama je malo manje dominantna emocija tuge, koja zna biti popraćena plakanjem, koje je ipak nešto tiše. Što je glasnoća veća, to je dominantnija emocija ljutnje koja se povezuje s vikanjem. Također, favorizira se i emocija straha koja zna biti popraćena urlikom ili vrištanjem.



Slika 72. Dijagram toka za glasnoću zvuka

```
49  def zvuk_scale(db):
50      if db>0 and db<=45:
51          db_scale = np.array([0.05, 0.1, 0.1, 0.4, 0.1])
52      elif db>=46 and db<=60:
53          db_scale = np.array([0.1, 0.2, 0.3, 0.1, 0.2])
54      elif db>=61 and db<=70:
55          db_scale = np.array([0.3, 0.2, 0.2, 0.1, 0.1])
56      elif db>=71 and db<=100:
57          db_scale = np.array([0.4, 0.3, 0.05, 0, 0.1])
58      else:
59          print("Something is wrong")
60
61      return db_scale
```

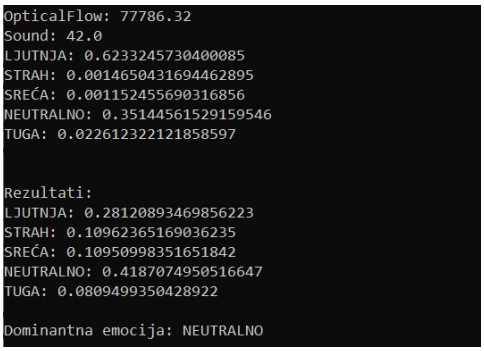
Slika 73. Funkcija za skaliranje glasnoće zvuka

Pomoću prethodnih slika vidjeli smo na koji način funkcioniraju i kako su kreirane skale koje ocjenjuju emocije na temelju intenziteta gibanja i razine buke u prostoriji. Dobiveni rezultati sve tri metode se zbrajaju i iz njih se dobija konačni zaključak. Kao što je i prije navedeno, ove tablice su kreirane na temelju testiranja i stvar su osobnog odabira. Potrebno je uzeti u obzir i kontekst okruženja u kojem se testiranje odvija. Skale su testirane u laboratoriju i u stanu. Kada bi se testiranje provelo na stadionu ili nekom trgu, one bi sigurno drugačije izgledale.
Sada kada su objašnjene i druge dvije metode ocjenjivanja emocija, možemo preći na dobivene rezultate.

5.3.1. REZULTATI MULTIMODALNOG PRISTUPA

Problem pri prikazivanju rezultata multimodalnog pristupa je što zahtijeva snimanje u realnom vremenu. Kako se to ne može prikazati na komadu papira, u prilogu se nalazi CD s nekoliko video snimki na kojima će biti prikazani rezultati. Ipak, u sklopu ovog poglavlja će pojedini rezultati biti prikazani pomoću slika.
Na [Slika 74] je prikazana slika koju je program koristio kako bi procjenio emociju. Na [Slika 75] vidimo očitane vrijednosti u trenutku kada je uhvaćena navedena slika. *OpticalFlow* očitava vrijednost od oko 77 000 što je prilično mirno po definiranoj skali [Slika 71]. Razina buke iznosi 42 dB, a model je donio procjenu da je najdominantnija emocija ljutnja, a druga najdominantnija neutralno. Kada se ti rezultati sumiraju, konačan zaključak pokazuje da je dominantna emocija ipak neutralno. Sa [Slika 74] se može zaključiti da je više prisutna emocija neutralno, ali sigurno postoje naznake ljutnje. Multimodalni pristup je uspio ispraviti grešku modela i donijeti točan zaključak. Treba primijetiti kako je multimodalni pristup pojačao utjecaj i drugih emocija. To je iz razloga što buka i intenzitet gibanja favoriziraju emocije drugačije u odnosu na izraz lica.

Slika 74. Multimodalni pristup Neutralno



Slika 75. Multimodalni pristup procjena Neutralno

[Slika 76] prikazuje emociju tuge, a [Slika 77] procjenu multimodalnog pristupa. Vidimo da je intezitet gibanja veliki, dok je razina buke 67 dB. Analizom izraza lica mreža je zaključila da se radi o emociji tuge. Ukupan rezultat potvrđuje tu procjenu i kao drugu dominantnu emociju navodi neutralno. Multimodalni pristup nam je ponudio više dominantnih procjena i svakako se može govoriti o njihovoj kombinaciji.

Slika 76. Multimodalni pristup emocija Tuga

```
OpticalFlow: 261568.61
Sound: 67.0
LJUTNJA: 0.02099660225212574
STRAH: 0.00013733014930039644
SREĆA: 0.0054107606410980225
NEUTRALNO: 0.35100623965263367
TUGA: 0.6224491000175476


Rezultati:
LJUTNJA: 0.17965399875005428
STRAH: 0.13797838815022553
SREĆA: 0.17427957066894545
NEUTRALNO: 0.19000214946108596
TUGA: 0.3180858929696888

Dominantna emocija: TUGA
```

Slika 77. Multimodalni pristup procjena Tuga

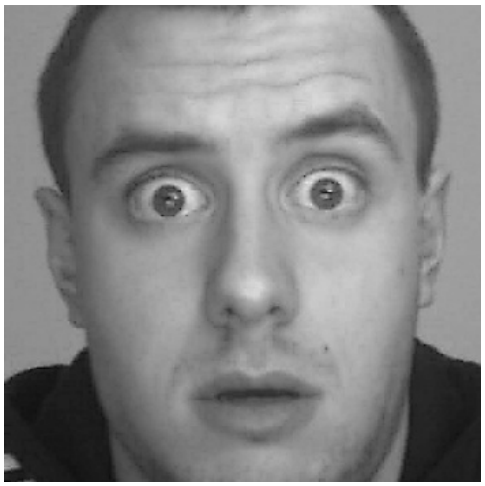
Sljedeći primjer prikazuje novu emociju i njezinu procjenu [Slika 78 i 79]. Intenzitet gibanja je velik i izmjerena je glasnoća od 71 dB. Mreža je procijenila da se radi o emociji neutralno, a ukupan rezultat kao dominantnu emociju postavlja ljutnju. Razlog je visoki intezitet gibanja i visoka glasnoća koji su emociju ljutnje stavili u prvi plan. Multimodalni pristup je ispravio grešku i kao drugu dominantnu emociju stavio neutralno.

Slika 78. Multimodalni pristup emocija Ljutnje

```
OpticalFlow: 214534.81
Sound: 71.0
LJUTNJA: 0.3356504440307617
STRAH: 0.0016771581722423434
SREĆA: 0.0033348260913044214
NEUTRALNO: 0.6214775443077087
TUGA: 0.037860024720430374


Rezultati:
LJUTNJA: 0.32829840172273606
STRAH: 0.17602707320826805
SREĆA: 0.1239771320783367
NEUTRALNO: 0.25315001578439683
TUGA: 0.11854737720626217

Dominantna emocija: LJUTNJA
```

Slika 79. Multimodalni pristup procjena Ljutnje

[Slika 80 i 81] prikazuju emociju straha i njezinu procjenu. Ukupan rezultat pokazuje da se radi o emociji straha zajedno s emocijom neutralno. Multimodalni pristup je opet dao na značaju i drugim emocijama u odnosu na samu procjenu neuronske mreže. Zato i ovu procjenu možemo smatrati točnom.



Slika 80. Multimodalni pristup emocija Straha

```
OpticalFlow: 276677.97
Sound: 38.0
LJUTNJA: 1.1743607331027306e-07
STRAH: 0.6159652471542358
SREĆA: 1.5766029264341341e-06
NEUTRALNO: 0.38402965664863586
TUGA: 3.448823463259032e-06


Rezultati:
LJUTNJA: 0.09090913207046118
STRAH: 0.3330782660403735
SREĆA: 0.14545511629644173
NEUTRALNO: 0.3214653242353996
TUGA: 0.10909216135732402

Dominantna emocija: STRAH
```

Slika 81. Multimodalni pristup procjena Straha

Zadnji primjer prikazuje emociju sreće i njezinu procjenu [Slika 82 i 83]. Iako su neke emocije dobile višu procjenu, ovaj primjer je poprilično jasan i dominantna emocija je sreća.

Slika 82. Multimodalni pristup emocija Sreće

```
OpticalFlow: 42170.78
Sound: 69.0
LJUTNJA: 1.0887671351156314e-06
STRAH: 1.6731637515476905e-06
SREĆA: 0.9995245933532715
NEUTRALNO: 0.00045942820725031197
TUGA: 1.3256702914077323e-05


Rezultati:
LJUTNJA: 0.1379314080079571
STRAH: 0.13793160952402894
SREĆA: 0.48259468067438105
NEUTRALNO: 0.17257221423131966
TUGA: 0.06897008756231314

Dominantna emocija: SRECA
```

Slika 83. Multimodalni pristup procjena Sreće

Na primjerima u ovom poglavlju mogli su se vidjeti rezultati multimodalnog pristupa. Njegova primarna zadaća je procjena emocija u realnom vremenu što analizu putem slika čini težom za objasniti. Ono što se može zaključiti na temelju dobivenih rezultata jest da multimodalni pristup često može ispraviti rezultate neuronske mreže i kao procjenu ponuditi više od jedne emocije. To je zato što uzima u obzir širi kontekst i više informacija prilikom procjenjivanja.

6. ZAKLJUČAK

Afektivna robotika je doživjela veliki uspon u zadnjih nekoliko godina. Njezina ideja je pokazivanje suosjećanja i empatije robota prema ljudima. Kako bi to bila u stanju, prvo mora razlikovati ljudske emocije. Emocije su veoma kompleksan i nemjerljiv pojma. Postoje tisuće raznih teorija o njima, a psiholozi i sociolozi dan danas vode debate o njihovoj podijeli. Emocije su stvar subjektivnog dojma i mi kao ljudi često imamo problem razlikovati ih. Razvoj procesora, senzora i neuronskih mreža je omogućio robotima da procjenjuju emocije na ljudima. Namjena takvih robota je raznolika. Od uporabe u pametnim kućama do predviđanja ponašanja ili otkrivanja ranog stadija bolesti poput depresije. U sklopu ovog rada su obrađeni jednomodalni i multimodalni pristupi pri procjeni emocija. Oba pristupa kao osnovu koriste neuronske mreže i raspoznavanje emocija na temelju mimike lica. Nedostatak jednomodalnog pristupa je manjak prikupljenih informacija i njegove procjene često vežu samo jednu emociju kao konačno predviđanje. Multimodalni pristup uzima kontekst okoline prilikom procjene. Intezitet pokreta tijela i razina buke u prostoriji favoriziraju emocije drugačije u odnosu mimiku. Zato multimodalni pristup kao rezultat procjene daje dvije ili tri emocije. Emocije nisu jednolične i prednost takvog pristupa je šira lepeza rješenja i manja mogućnost pogreške. To ne znači da program ne može krivo zaključiti, ali uzimanje u obzir kontekst okoline u kojoj se procjena odvija dovodi do više mogućih rješenja. Rad je osmišljen i kreiran u laboratorijskim uvjetima i kao takav se pokazao veoma uspješan u procjenama i fleksibilan za nastavak razvoja. Sljedeći koraci za unapređenje su uvođenje novih emocija i povećanje broja slika u bazi podataka, dorada već postojećih medota procijenjivanja emocija i uvođenje nekih novih poput EEG-a ili mjerenja broja otkucaja srca.

Literatura

[1] Yingying Jiang, M. Shamim Hossain, Min Chen, Abdulhameed Alelaiwi, Muneer Al-Hammadi. *A snapshot research and implementation of multimodal information fusion for data-driven emotion recognition*. School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, China. Department of Software Engineering, College of Computer and Information Sciences, King Saud University, Riyadh 11543, Saudi Arabia. Department of Computer Engineering, College of Computer and Information Sciences, King Saud University, Riyadh 11543, Saudi Arabia.

[2] Pristup shvaćanju emocija: *https://www.nakladaslap.com/public/docs/knjige/Razumijevanje%20emocija%20-%20pog.pdf*, Pristupljeno: 10. siječnja, 2020.

[3] Saša Drače, Jadranka Kolenović-Đapo. *Klasične teorije emocija u svjetlu suvremenih empirijskih spoznaja*. Sarajevo: Filozofski fakultet Univerziteta u Sarajevu; 2017.

[4] *Python*: *https://en.wikipedia.org/wiki/Python_(programming_language)*, Pristupljeno: 10. siječnja, 2020.

[5] Branko Novaković, Dubravko Majetić, Mladen Široki. *Umjetne neuronske mreže*. Zagreb: Fakultet strojarstva i brodogradnje; 1998.

[6] Metoda povratnog rasprostiranja: *https://en.wikipedia.org/wiki/Backpropagation*, Pristupljeno: 10. siječnja 2020.

[7] 3BLUE1BROWN: *https://www.youtube.com/channel/UCYO_jab_esuFRV4b17AJtAw*, Pristupljeno 10. siječnja 2020.

[8] Damir Kopljar. *Konvolucijske neuronske mreže* [Završni zadatak]. Zagreb: Fakultet elektrotehnike i računarstva; 2016.

[9] VGG neuronska mreža: *https://neurohive.io/en/popular-networks/vgg16/*, Pristupljeno: 10. siječnja 2020.

[10] Gunnar Farneback. *Two-Frame Motion Estimation Based on Polynomial Expansion*. Computer Vision Laboratory, Linkoping University, SE-581 83 Linkoping, Sweden.

[11] Prepoznavanje emocija: *http://www.paulvangent.com/2016/04/01/emotion-recognition-with-python-opencv-and-a-face-dataset/*, Pristupljeno: 10. siječnja 2020.

[12] Trening TensorFlow: *https://medium.com/@jsflo.dev/training-a-tensorflow-model-to-recognize-emotions-a20c3bcd6468*, Pristupljeno: 10. siječnja 2020.

[13] Baza slika: *http://www.consortium.ri.cmu.edu/ckagree/*, Pristupljeno: 10. siječnja 2020.

[14] Baza slika: *http://app.visgraf.impa.br/database/faces/*, Pristupljeno: 10. siječnja 2020.

[15] Baza slika: *https://zenodo.org/record/3451524#.XgDudGRKiM8*, Pristupljeno: 10. siječnja 2020.

[16] *https://grail.cs.washington.edu/projects/deepexpr/ferg-db.html*, Pristupljeno: 10. siječnja 2020.

[17] Detekcija lica: *https://towardsdatascience.com/a-guide-to-face-detection-in-python-3eab0f6b9fc1*, Pristupljeno: 10. siječnja 2020.

[18] TensorFlow: *https://www.tensorflow.org/*, Pristupljeno: 10. siječnja 2020.

[19] Keras: *https://keras.io/*, Pristupljeno: 10. siječnja 2020.

[20] Predviđanje u realnom vremenu: *https://medium.com/datadriveninvestor/real-time-facial-expression-recognition-f860dacfeb6a*, Pristupljeno: 10. siječnja 2020.

[21] PyAudio: *https://people.csail.mit.edu/hubert/pyaudio/docs/#pyaudio.PyAudio*, Pristupljeno: 10 siječnja 2020.

[22] Decibeli: *https://dsp.stackexchange.com/questions/32076/fft-to-spectrum-in-decibel*, Pristupljeno: 10. siječnja 2020.

[23] OpenCv: *https://docs.opencv.org/master/dc/d6b/group__video__track.html#ga5d10ebbd59fe09c5f650289ec0ece5af*, Pristupljeno: 10. siječnja 2020.

PRILOZI

I. CD-R disk

II. Python kôd

SORTIRANJE SLIKA LJUDI

```
#Ubacivanje potrebnih paketa
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from PIL import Image
import glob
import cv2
from sklearn.model_selection import train_test_split
from keras.layers import Dropout, Dense
from keras.layers.normalization import BatchNormalization
from keras.models import Sequential, load_model
from keras.applications import VGG16
from sklearn.metrics import accuracy_score, confusion_matrix

#Ocitavanje putanje svih slika emocije ljutnje iz lokalnog file-a
human_angry = glob.glob("Human/anger/*")
print ("Number of images in Angry emotion = "+str(len(human_angry)))

#Raspodjela putanje i oznaka za slike s emocijom ljutnje
human_angry_folderName = [str(i.split("\\")[0])+"/" for i in human_angry]
human_angry_imageName = [str(i.split("\\")[1])+"/" for i in human_angry]
human_angry_emotion = [["Angry"]*len(human_angry)][0]
human_angry_label = [1]*len(human_angry)
print(len(human_angry_folderName))
print(len(human_angry_imageName))
print(len(human_angry_emotion))
print(len(human_angry_label))

#Kreiranje tablice podataka pomocu paketa pandas
df_angry = pd.DataFrame()
df_angry["folderName"] = human_angry_folderName
df_angry["imageName"] = human_angry_imageName
df_angry["Emotion"] = human_angry_emotion
df_angry["Labels"] = human_angry_label
df_angry.head()
print(df_angry.head())

#Ocitavanje putanje svih slika emocije straha iz lokalnog file-a
human_fear = glob.glob("Human/fear/*")
print ("Number of images in Fear emotion = "+str(len(human_fear)))
```

```
#Raspodjela putanje i oznaka za slike s emocijom straha
human_fear_folderName = [str(i.split("\\")[0])+"/" for i in human_fear]
human_fear_imageName = [str(i.split("\\")[1])+"/" for i in human_fear]
human_fear_emotion = [["Fear"]*len(human_fear)][0]
human_fear_label = [2]*len(human_fear)
print(len(human_fear_folderName))
print(len(human_fear_imageName))
print(len(human_fear_emotion))
print(len(human_fear_label))

#Kreiranje tablice podataka pomocu paketa pandas
df_fear = pd.DataFrame()
df_fear["folderName"] = human_fear_folderName
df_fear["imageName"] = human_fear_imageName
df_fear["Emotion"] = human_fear_emotion
df_fear["Labels"] = human_fear_label
df_fear.head()
print(df_fear.head())

#Ocitavanje putnje svih slika emocije srece iz lokalnog file-a
human_happy = glob.glob("Human/happy/*")
print ("Number of images in Happy emotion = "+str(len(human_happy)))

#Raspodjela putanje i oznaka za slike s emocijom srece
human_happy_folderName = [str(i.split("\\")[0])+"/" for i in human_happy]
human_happy_imageName = [str(i.split("\\")[1])+"/" for i in human_happy]
human_happy_emotion = [["Happy"]*len(human_happy)][0]
human_happy_label = [3]*len(human_happy)
print(len(human_happy_folderName))
print(len(human_happy_imageName))
print(len(human_happy_emotion))
print(len(human_happy_label))

#Kreiranje tablice podataka pomocu paketa pandas
df_happy = pd.DataFrame()
df_happy["folderName"] = human_happy_folderName
df_happy["imageName"] = human_happy_imageName
df_happy["Emotion"] = human_happy_emotion
df_happy["Labels"] = human_happy_label
df_happy.head()
print(df_happy.head())

#Ocitavanje putanje svih slika emocije neutralno iz lokalnog file-a
```

```
human_neutral = glob.glob("Human/neutral/*")
print ("Number of images in Neutral emotion = "+str(len(human_neutral)))

#Raspodjela putanje i oznaka za slike s emocijom neutralno
human_neutral_folderName = [str(i.split("\\")[0])+"/" for i in human_neutral]
human_neutral_imageName = [str(i.split("\\")[1])+"/" for i in human_neutral]
human_neutral_emotion = [["Neutral"]*len(human_neutral)][0]
human_neutral_label = [4]*len(human_neutral)
print(len(human_neutral_folderName))
print(len(human_neutral_imageName))
print(len(human_neutral_emotion))
print(len(human_neutral_label))

#Kreiranje tablice podataka pomocu paketa pandas
df_neutral = pd.DataFrame()
df_neutral["folderName"] = human_neutral_folderName
df_neutral["imageName"] = human_neutral_imageName
df_neutral["Emotion"] = human_neutral_emotion
df_neutral["Labels"] = human_neutral_label
df_neutral.head()
print(df_neutral.head())

#Ocitavanje putanje svih slika emocije tuga iz lokalnog file-a
human_sad = glob.glob("Human/sadness/*")
print ("Number of images in Sad emotion = "+str(len(human_sad)))

#Raspodjela putanje i oznaka za slike s emocijom tuge
human_sad_folderName = [str(i.split("\\")[0])+"/" for i in human_sad]
human_sad_imageName = [str(i.split("\\")[1])+"/" for i in human_sad]
human_sad_emotion = [["Sad"]*len(human_sad)][0]
human_sad_label = [5]*len(human_sad)
print(len(human_sad_folderName))
print(len(human_sad_imageName))
print(len(human_sad_emotion))
print(len(human_sad_label))

#Kreitanje tablice podataka pomocu paketa pandas
df_sad = pd.DataFrame()
df_sad["folderName"] = human_sad_folderName
df_sad["imageName"] = human_sad_imageName
df_sad["Emotion"] = human_sad_emotion
df_sad["Labels"] = human_sad_label
df_sad.head()
```

```
print(df_sad.head())

#Spajanje svih tablica u jednu
frames = [df_angry, df_fear, df_happy, df_neutral, df_sad]
Final_human = pd.concat(frames)
print(Final_human.shape)

#Promjena redosljeda slika u tablici
Final_human.reset_index(inplace = True, drop = True)
Final_human = Final_human.sample(frac = 1.0)
Final_human.reset_index(inplace = True, drop = True)
Final_human.head()
print(Final_human.head())

#Podijela baze na bazu za trening, procjenu i test
df_human_train_data, df_human_test = train_test_split(Final_human, stratify=Final_human|
df_human_train, df_human_cv = train_test_split(df_human_train_data, stratify=df_human_tr
print(df_human_train.shape)
print(df_human_cv.shape)
print(df_human_test.shape)

#Spremanje dobivenih baza
df_human_train.reset_index(inplace = True, drop = True)
df_human_train.to_pickle("HumanData/df_human_train.pk1")

df_human_cv.reset_index(inplace = True, drop = True)
df_human_cv.to_pickle("HumanData/df_human_cv.pk1")

df_human_test.reset_index(inplace = True, drop = True)
df_human_test.to_pickle("HumanData/df_human_test.pk1")
```

PROCESIRANJE SLIKA LJUDI

```
#Ubacivanje potrebnih paketa
import os
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from PIL import Image
import glob
import cv2
from sklearn.model_selection import train_test_split
from keras.layers import Dropout, Dense
from keras.layers.normalization import BatchNormalization
from keras.models import Sequential, load_model
from keras.applications import VGG16
from sklearn.metrics import accuracy_score, confusion_matrix

#Ucitavanje prethodno kreiranih baza za trening, predvidanje i test
df_human_train = pd.read_pickle("HumanData/df_human_train.pk1")
print(df_human_train.head())
print(df_human_train.shape)

df_human_cv = pd.read_pickle("HumanData/df_human_cv.pk1")
print(df_human_cv.head())
print(df_human_cv.shape)

df_human_test = pd.read_pickle("HumanData/df_human_test.pk1")
print(df_human_test.head())
print(df_human_test.shape)


#Funkcija za pretvaranje slika u sivu skalu
def convt_to_gray(df):
    count = 0
    for i in range(len(df)):
        path1 = df["folderName"][i]
        path2 = df["imageName"][i]
        path2 = path2[:-1]
        img = cv2.imread(os.path.join(path1, path2))
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        cv2.imwrite(os.path.join(path1,path2), gray)
        count += 1
```

```
    print("Total number of images converted and saved = "+str(count))

#Izvršavanje prethodne funkcije
convt_to_gray(df_human_train)
convt_to_gray(df_human_cv)
convt_to_gray(df_human_test)

#Očitavanje klasifikatora za detekciju lica
face_cascade = cv2.CascadeClassifier("haarcascade_frontalface_default.xml")

#Funkcija za detekciju lica i promjenu dimenzije
def face_det_crop_resize(img_path):
    img = cv2.imread(img_path)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray, 1.3, 5)

    if faces == ():
        cv2.imwrite(img_path, cv2.resize(gray, (350,350)))
    else:
        for (x,y,w,h) in faces:
            face_clip = img[y:y+h, x:x+w]
            cv2.imwrite(img_path, cv2.resize(face_clip, (350,350)))

#For petlje za iteraciju kroz svaku sliku baze
for i, d in df_human_train.iterrows():
    img_path = os.path.join(d["folderName"], d["imageName"])
    img_path = img_path[:-1]
    face_det_crop_resize(img_path)


for i, d in df_human_cv.iterrows():
    img_path = os.path.join(d["folderName"], d["imageName"])
    img_path = img_path[:-1]
    face_det_crop_resize(img_path)


for i, d in df_human_test.iterrows():
    img_path = os.path.join(d["folderName"], d["imageName"])
    img_path = img_path[:-1]
    face_det_crop_resize(img_path)


print("Done")
```

SORTIRANJE SLIKA ANIMIRANIH LIKOVA

```
#Ubacivanje potrebnih paketa
import os
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from PIL import Image
import glob
import cv2
from sklearn.model_selection import train_test_split
from keras.layers import Dropout, Dense
from keras.layers.normalization import BatchNormalization
from keras.models import Sequential, load_model
from keras.applications import VGG16
from sklearn.metrics import accuracy_score, confusion_matrix

#Ocitavanje putanje svih slika emocije ljutnje iz lokalnog file-a
anime_angry = glob.glob("Animated/Angry/*.png")
print("Number of images in Angry emotion = "+str(len(anime_angry)))

#Raspodjela putanje i oznaka za slike s emocijom ljutnje
anime_angry_folderName = [str(i.split("\\")[0])+"/" for i in anime_angry]
anime_angry_imageName = [str(i.split("\\")[1]) for i in anime_angry]
anime_angry_emotion = [["Angry"]*len(anime_angry)][0]
anime_angry_labels = [1]*len(anime_angry)
print(len(anime_angry_folderName))
print(len(anime_angry_imageName))
print(len(anime_angry_emotion))
print(len(anime_angry_labels))

#Kreiranje tablice podataka pomocu paketa pandas
df_angry = pd.DataFrame()
df_angry["folderName"] = anime_angry_folderName
df_angry["imageName"] = anime_angry_imageName
df_angry["Emotion"] = anime_angry_emotion
df_angry["Labels"] = anime_angry_labels
df_angry.head()
print(df_angry.head())

#Slucajni odabir 1000 slika emocije ljutnje
df_angry = df_angry.sample(frac = 1.0)
df_angry_reduced = df_angry.sample(n = 1000)
```

```
df_angry_reduced.shape
print(df_angry_reduced.shape)

df_angry_reducedIndx = df_angry_reduced.index
count = 0
for i, d in df_angry.iterrows():
    if i not in df_angry_reducedIndx:
        os.remove(os.path.join(d["folderName"], d["imageName"]))
        count += 1
print("Total number of images removed = "+str(count))

#Ocitavanje putanje svih slika emocije straha iz lokalnog file-a
anime_fear = glob.glob("Animated/Fear/*.png")
print("Number of images in Fear emotion = "+str(len(anime_fear)))

#Raspodjela putanje i oznaka za slike s emocijom straha
anime_fear_folderName = [str(i.split("\\")[0])+"/" for i in anime_fear]
anime_fear_imageName = [str(i.split("\\")[1]) for i in anime_fear]
anime_fear_emotion = [["Fear"]*len(anime_fear)][0]
anime_fear_labels = [2]*len(anime_fear)
print(len(anime_fear_folderName))
print(len(anime_fear_imageName))
print(len(anime_fear_emotion))
print(len(anime_fear_labels))

#Kreiranje tablice podataka pomocu paketa pandas
df_fear = pd.DataFrame()
df_fear["folderName"] = anime_fear_folderName
df_fear["imageName"] = anime_fear_imageName
df_fear["Emotion"] = anime_fear_emotion
df_fear["Labels"] = anime_fear_labels
df_fear.head()
print(df_fear.head())

#Slucajni odabir 1000 slika emocije straha
df_fear = df_fear.sample(frac = 1.0)
df_fear_reduced = df_fear.sample(n = 1000)
df_fear_reduced.shape
print(df_fear_reduced.shape)

df_fear_reducedIndx = df_fear_reduced.index
count = 0
for i, d in df_fear.iterrows():
```

```
    if i not in df_fear_reducedIndx:
        os.remove(os.path.join(d["folderName"], d["imageName"]))
        count += 1
print("Total number of images removed = "+str(count))

#Ocitavanje putanje svih slika emocije srece iz lokalnog file-a
anime_happy = glob.glob("Animated/Happy/*.png")
print("Number of images in Happy emotion = "+str(len(anime_happy)))

#Raspodjela putanje i oznaka za slike s emocijom srece
anime_happy_folderName = [str(i.split("\\")[0])+"/" for i in anime_happy]
anime_happy_imageName = [str(i.split("\\")[1]) for i in anime_happy]
anime_happy_emotion = [["Happy"]*len(anime_happy)][0]
anime_happy_labels = [3]*len(anime_happy)
print(len(anime_happy_folderName))
print(len(anime_happy_imageName))
print(len(anime_happy_emotion))
print(len(anime_happy_labels))

#Kreiranje tablice podataka pomocu paketa pandas
df_happy = pd.DataFrame()
df_happy["folderName"] = anime_happy_folderName
df_happy["imageName"] = anime_happy_imageName
df_happy["Emotion"] = anime_happy_emotion
df_happy["Labels"] = anime_happy_labels
df_happy.head()
print(df_happy.head())

#Slucajni odabir 1000 slika emocije srece
df_happy = df_happy.sample(frac = 1.0)
df_happy_reduced = df_happy.sample(n = 1000)
df_happy_reduced.shape
print(df_happy_reduced.shape)

df_happy_reducedIndx = df_happy_reduced.index
count = 0
for i, d in df_happy.iterrows():
    if i not in df_happy_reducedIndx:
        os.remove(os.path.join(d["folderName"], d["imageName"]))
        count += 1
print("Total number of images removed = "+str(count))

#Ocitavanje putanje svih slika emocije neutralno iz lokalnog file-a
```

```
anime_neutral = glob.glob("Animated/Neutral/*.png")
print("Number of images in Neutral emotion = "+str(len(anime_neutral)))

#Raspodjela putanje i oznaka za slike s emocijom neutralno
anime_neutral_folderName = [str(i.split("\\")[0])+"/" for i in anime_neutral]
anime_neutral_imageName = [str(i.split("\\")[1]) for i in anime_neutral]
anime_neutral_emotion = [["Neutral"]*len(anime_neutral)][0]
anime_neutral_labels = [4]*len(anime_neutral)
print(len(anime_neutral_folderName))
print(len(anime_neutral_imageName))
print(len(anime_neutral_emotion))
print(len(anime_neutral_labels))

#Kreiranje tablice podataka pomocu paketa pandas
df_neutral = pd.DataFrame()
df_neutral["folderName"] = anime_neutral_folderName
df_neutral["imageName"] = anime_neutral_imageName
df_neutral["Emotion"] = anime_neutral_emotion
df_neutral["Labels"] = anime_neutral_labels
df_neutral.head()
print(df_neutral.head())

#Slucajni odabir 1000 slika emocije neutralno
df_neutral = df_neutral.sample(frac = 1.0)
df_neutral_reduced = df_neutral.sample(n = 1000)
df_neutral_reduced.shape
print(df_neutral_reduced.shape)

df_neutral_reducedIndx = df_neutral_reduced.index
count = 0
for i, d in df_neutral.iterrows():
    if i not in df_neutral_reducedIndx:
        os.remove(os.path.join(d["folderName"], d["imageName"]))
        count += 1
print("Total number of images removed = "+str(count))

#Ocitavanje putanje svih slika emocije tuge iz lokalnog file-a
anime_sad = glob.glob("Animated/Sad/*.png")
print("Number of images in Sad emotion = "+str(len(anime_sad)))

#Raspodjela putanje i oznaka za slike s emocijom tuge
anime_sad_folderName = [str(i.split("\\")[0])+"/" for i in anime_sad]
anime_sad_imageName = [str(i.split("\\")[1]) for i in anime_sad]
```

```
anime_sad_emotion = [["Sad"]*len(anime_sad)][0]
anime_sad_labels = [5]*len(anime_sad)
print(len(anime_sad_folderName))
print(len(anime_sad_imageName))
print(len(anime_sad_emotion))
print(len(anime_sad_labels))

#Kreiranje tablice podataka pomocu paketa pandas
df_sad = pd.DataFrame()
df_sad["folderName"] = anime_sad_folderName
df_sad["imageName"] = anime_sad_imageName
df_sad["Emotion"] = anime_sad_emotion
df_sad["Labels"] = anime_sad_labels
df_sad.head()
print(df_sad.head())

#Slucajni odabir 1000 slika emocije tuga
df_sad = df_sad.sample(frac = 1.0)
df_sad_reduced = df_sad.sample(n = 1000)
df_sad_reduced.shape
print(df_sad_reduced.shape)

df_sad_reducedIndx = df_sad_reduced.index
count = 0
for i, d in df_sad.iterrows():
    if i not in df_sad_reducedIndx:
        os.remove(os.path.join(d["folderName"], d["imageName"]))
        count += 1
print("Total number of images removed = "+str(count))

#Spajanje svih tablica u jednu
frames = [df_angry_reduced, df_fear_reduced, df_happy_reduced, df_neutral_reduced, df_sa
Final_Animated = pd.concat(frames)
Final_Animated.shape
print(Final_Animated.shape)

#Promjena redosljeda slika u tablici
Final_Animated.reset_index(inplace = True, drop = True)
Final_Animated = Final_Animated.sample(frac = 1.0)
Final_Animated.reset_index(inplace = True, drop = True)
Final_Animated.head()
print(Final_Animated.head())
```

```
#Podijela baze na bazu za trening, procjenu i test
df_anime_train_data, df_anime_test = train_test_split(Final_Animated, stratify=Final_Ani
df_anime_train, df_anime_cv = train_test_split(df_anime_train_data, stratify=df_anime_tr
df_anime_train.shape, df_anime_cv.shape, df_anime_test.shape

print(df_anime_train.shape)
print(df_anime_cv.shape)
print(df_anime_test.shape)

#Spremanje dobivenih baza
df_anime_train.reset_index(inplace = True, drop = True)
df_anime_train.to_pickle("AnimatedData/df_anime_train.pk1")

df_anime_cv.reset_index(inplace = True, drop = True)
df_anime_cv.to_pickle("AnimatedData/df_anime_cv.pk1")

df_anime_test.reset_index(inplace = True, drop = True)
df_anime_test.to_pickle("AnimatedData/df_anime_test.pk1")

print("Done")
```

PROCESIRANJE SLIKA ANIMIRANIH LIKOVA

```
#Ubacivanje potrebnih paketa
import os
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from PIL import Image
import glob
import cv2
from sklearn.model_selection import train_test_split
from keras.layers import Dropout, Dense
from keras.layers.normalization import BatchNormalization
from keras.models import Sequential, load_model
from keras.applications import VGG16
from sklearn.metrics import accuracy_score, confusion_matrix

#Ucitavanje prethodno kreiranih baza za trening, predvidanje i test
df_anime_train = pd.read_pickle("AnimatedData/df_anime_train.pk1")
df_anime_train.head()

print(df_anime_train.head())
print(df_anime_train.shape)

df_anime_cv = pd.read_pickle("AnimatedData/df_anime_cv.pk1")
df_anime_train.head()

print(df_anime_cv.head())
print(df_anime_cv.shape)

df_anime_test = pd.read_pickle("AnimatedData/df_anime_test.pk1")
df_anime_test.head()

print(df_anime_test.head())
print(df_anime_test.shape)

#Funkcija za pretvaranje slika u sivu skalu
def convt_to_gray(df):
    count = 0
    for i in range(len(df)):
        path1 = df["folderName"][i]
        path2 = df["imageName"][i]
        img = cv2.imread(os.path.join(path1, path2))
```

```
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        cv2.imwrite(os.path.join(path1,path2), gray)
        count += 1

    print("Total number of images converted and saved = "+str(count))

#Izvrsavanje funkcije pretvorbe u sivu skalu
convt_to_gray(df_anime_train)
convt_to_gray(df_anime_cv)
convt_to_gray(df_anime_test)

#Funkcija za detekciju lica i promjenu dimenzija
def change_image(df):
    count = 0
    for i, d in df.iterrows():
        img = cv2.imread(os.path.join(d["folderName"], d["imageName"]))
        face_clip = img[40:240, 35:225]
        face_resized = cv2.resize(face_clip, (350,350))
        cv2.imwrite(os.path.join(d["folderName"], d["imageName"]), face_resized)
        count += 1
    print("Total number of images cropped and resized = {}".format(count))

#Izvrsavanje funkcije za detekciju lica
change_image(df_anime_train)
change_image(df_anime_cv)
change_image(df_anime_test)
```

KREIRANJE BOTTLENECK_FEATURESA POMOĆU MODELA VGG16

```
#Ubacivanje potrebnih paketa
import os
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from PIL import Image
import glob
import cv2
from sklearn.model_selection import train_test_split
from keras.layers import Dropout, Dense
from keras.layers.normalization import BatchNormalization
from keras.models import Sequential, load_model
from keras.applications import VGG16
from sklearn.metrics import accuracy_score, confusion_matrix

#Ucitavanje baza za procjenu
df_anime_train = pd.read_pickle("AnimatedData/df_anime_train.pk1")
print(df_anime_train.shape)
print(df_anime_train.head())

df_human_train = pd.read_pickle("HumanData/df_human_train.pk1")
print(df_human_train.head())
print(df_human_train.shape)

#Spajanje baza ljudi i animiranih likova za trening
frames = [df_human_train, df_anime_train]
combined_train = pd.concat(frames)
print(combined_train.shape)

combined_train = combined_train.sample(frac = 1.0)
combined_train.reset_index(inplace = True, drop = True)
combined_train.to_pickle("CombinedData/combined_train.pk1")

Train_Combined = pd.read_pickle("CombinedData/combined_train.pk1")
CV_Humans = pd.read_pickle("HumanData/df_human_cv.pk1")
Test_Humans = pd.read_pickle("HumanData/df_human_test.pk1")
CV_Animated = pd.read_pickle("AnimatedData/df_anime_cv.pk1")
Test_Animated = pd.read_pickle("AnimatedData/df_anime_test.pk1")
print(Train_Combined.shape)
print(CV_Humans.shape)
print(CV_Animated.shape)
```

```
print(Test_Humans.shape)
print(Test_Animated.shape)

TrainCombined_batch_pointer = 0
CVHumans_batch_pointer = 0
CVAnimated_batch_pointer = 0
TestHumans_batch_pointer = 0
TestAnimated_batch_pointer = 0

TrainCombined_Labels = pd.get_dummies(Train_Combined["Labels"]).as_matrix()
print(TrainCombined_Labels.shape)

#Funkcija za kreiranje batch_size slika i njihovih oznaka baze za trening
def loadCombinedTrainBatch(batch_size):
    global TrainCombined_batch_pointer
    batch_images = []
    batch_labels = []
    for i in range(batch_size):
        path1 = Train_Combined.iloc[TrainCombined_batch_pointer + i]["folderName"]
        path2 = Train_Combined.iloc[TrainCombined_batch_pointer + i]["imageName"]
        if path2[-1] == "/":
            path2 = path2[:-1]
        read_image = cv2.imread(os.path.join(path1,path2))
        read_image_final = read_image/255.0
        batch_images.append(read_image_final)
        batch_labels.append(TrainCombined_Labels[TrainCombined_batch_pointer + i])

    TrainCombined_batch_pointer += batch_size

    return np.array(batch_images), np.array(batch_labels)

#Ucitavanje VGG16 modela i mapa u koje ce se spremati bottleneck featuresi
model = VGG16(weights='imagenet', include_top=False)
SAVEDIR = "Bottleneck_Features/Bottleneck_CombinedTrain/"
SAVEDIR_LABELS = "Bottleneck_Features/CombinedTrain_Labels/"
batch_size = 10
#For petlja za iteraciju kroz cijelu bazu za trening
for i in range(int(len(Train_Combined)/batch_size)):
    x,y = loadCombinedTrainBatch(batch_size)
    print("Batch {} loaded".format(i+1))
    np.save(os.path.join(SAVEDIR_LABELS, "bottleneck_labels_{}".format(i+1)), y)
    print("Creating bottleneck features for batch {}".format(i+1))
    bottleneck_features = model.predict(x)
```

```
    np.save(os.path.join(SAVEDIR, "bottleneck_{}".format(i+1)), bottleneck_features)
    print("Bottleneck features for batch {} created and saved\n".format(i+1))

CVHumans_Labels = pd.get_dummies(CV_Humans["Labels"]).as_matrix()
print(CVHumans_Labels.shape)

#Funkcija za kreiranje batch_size slika i njihovih oznaka baze za procjenu (slike ljud
def loadCVHumanBatch(batch_size):
    global CVHumans_batch_pointer
    batch_images = []
    batch_labels = []
    for i in range(batch_size):
        path1 = CV_Humans.iloc[CVHumans_batch_pointer + i]["folderName"]
        path2 = CV_Humans.iloc[CVHumans_batch_pointer + i]["imageName"]
        if path2[-1] == "/":
            path2 = path2[:-1]
        read_image = cv2.imread(os.path.join(path1,path2))
        read_image_final = read_image/255.0
        batch_images.append(read_image_final)
        batch_labels.append(CVHumans_Labels[CVHumans_batch_pointer + i])

    CVHumans_batch_pointer += batch_size

    return np.array(batch_images), np.array(batch_labels)

#Ucitavanje VGG16 modela i mapa u koje ce se spremati bottleneck featuresi
model = VGG16(weights='imagenet', include_top=False)
SAVEDIR = "Bottleneck_Features/Bottleneck_CVHumans/"
SAVEDIR_LABELS = "Bottleneck_Features/CVHumans_Labels/"
batch_size = 10
#For petlja za iteraciju kroz cijelu bazu za procjenu
for i in range(int(len(CV_Humans)/batch_size)):
    x,y = loadCVHumanBatch(batch_size)
    print("Batch {} loaded".format(i+1))
    np.save(os.path.join(SAVEDIR_LABELS, "bottleneck_labels_{}".format(i+1)), y)
    print("Creating bottleneck features for batch {}".format(i+1))
    bottleneck_features = model.predict(x)
    np.save(os.path.join(SAVEDIR, "bottleneck_{}".format(i+1)), bottleneck_features)
    print("Bottleneck features for batch {} created and saved\n".format(i+1))

CVAnimated_Labels = pd.get_dummies(CV_Animated["Labels"]).as_matrix()
print(CVAnimated_Labels.shape)
```

```
#Funkcija za kreiranje batch_size slika i njihovih oznaka baze za procjenu (slike anim
def loadCVAnimatedBatch(batch_size):
    global CVAnimated_batch_pointer
    batch_images = []
    batch_labels = []
    for i in range(batch_size):
        path1 = CV_Animated.iloc[CVAnimated_batch_pointer + i]["folderName"]
        path2 = CV_Animated.iloc[CVAnimated_batch_pointer + i]["imageName"]
        if path2[-1] == "/":
            path2 = path2[:-1]
        read_image = cv2.imread(os.path.join(path1,path2))
        read_image_final = read_image/255.0
        batch_images.append(read_image_final)
        batch_labels.append(CVAnimated_Labels[CVAnimated_batch_pointer + i])

    CVAnimated_batch_pointer += batch_size

    return np.array(batch_images), np.array(batch_labels)

#Ucitavanje VGG16 modela i mapa u koje ce se spremati bottleneck featuresi
model = VGG16(weights='imagenet', include_top=False)
SAVEDIR = "Bottleneck_Features/Bottleneck_CVAnimated/"
SAVEDIR_LABELS = "Bottleneck_Features/CVAnimated_Labels/"
batch_size = 10
#For petlja za iteraciju kroz cijelu bazu za procjenu
for i in range(int(len(CV_Animated)/batch_size)):
    x,y = loadCVAnimatedBatch(batch_size)
    print("Batch {} loaded".format(i+1))
    np.save(os.path.join(SAVEDIR_LABELS, "bottleneck_labels_{}".format(i+1)), y)
    print("Creating bottleneck features for batch {}".format(i+1))
    bottleneck_features = model.predict(x)
    np.save(os.path.join(SAVEDIR, "bottleneck_{}".format(i+1)), bottleneck_features)
    print("Bottleneck features for batch {} created and saved\n".format(i+1))

TestHumans_Labels = pd.get_dummies(Test_Humans["Labels"]).as_matrix()
print(TestHumans_Labels.shape)

#Funkcija za kreiranje batch_size slika i njihovih oznaka baze za test (slike ljudi)
def loadTestHumansBatch(batch_size):
    global TestHumans_batch_pointer
    batch_images = []
    batch_labels = []
    for i in range(batch_size):
```

```
        path1 = Test_Humans.iloc[TestHumans_batch_pointer + i]["folderName"]
        path2 = Test_Humans.iloc[TestHumans_batch_pointer + i]["imageName"]
        if path2[-1] == "/":
            path2 = path2[:-1]
        read_image = cv2.imread(os.path.join(path1,path2))
        read_image_final = read_image/255.0
        batch_images.append(read_image_final)
        batch_labels.append(TestHumans_Labels[TestHumans_batch_pointer + i])

    TestHumans_batch_pointer += batch_size

    return np.array(batch_images), np.array(batch_labels)

#Ucitavanje VGG16 modela i mapa u koje ce se spremati bottleneck featuresi
model = VGG16(weights='imagenet', include_top=False)
SAVEDIR = "Bottleneck_Features/Bottleneck_TestHumans/"
SAVEDIR_LABELS = "Bottleneck_Features/TestHumans_Labels/"
batch_size = 10
#For petlja za iteraciju kroz cijelu bazu za test
for i in range(int(len(Test_Humans)/batch_size)):
    x,y = loadTestHumansBatch(batch_size)
    print("Batch {} loaded".format(i+1))
    np.save(os.path.join(SAVEDIR_LABELS, "bottleneck_labels_{}".format(i+1)), y)
    print("Creating bottleneck features for batch {}".format(i+1))
    bottleneck_features = model.predict(x)
    np.save(os.path.join(SAVEDIR, "bottleneck_{}".format(i+1)), bottleneck_features)
    print("Bottleneck features for batch {} created and saved\n".format(i+1))

leftover_points = len(Test_Humans) - TestHumans_batch_pointer
x,y = loadTestHumansBatch(leftover_points)
np.save(os.path.join(SAVEDIR_LABELS, "bottleneck_labels_{}".format(int(len(Test_Humans)
bottleneck_features = model.predict(x)
np.save(os.path.join(SAVEDIR, "bottleneck_{}".format(int(len(Test_Humans)/batch_size) +

TestAnimated_Labels = pd.get_dummies(Test_Animated["Labels"]).as_matrix()
print(TestAnimated_Labels.shape)

#Funkcija za kreiranje batch_size slika i njihovih oznaka baze za test (slike animiran
def loadTestAnimatedBatch(batch_size):
    global TestAnimated_batch_pointer
    batch_images = []
    batch_labels = []
    for i in range(batch_size):
```

```
        path1 = Test_Animated.iloc[TestAnimated_batch_pointer + i]["folderName"]
        path2 = Test_Animated.iloc[TestAnimated_batch_pointer + i]["imageName"]
        if path2[-1] == "/":
            path2 = path2[:-1]
        read_image = cv2.imread(os.path.join(path1,path2))
        read_image_final = read_image/255.0
        batch_images.append(read_image_final)
        batch_labels.append(TestAnimated_Labels[TestAnimated_batch_pointer + i])

    TestAnimated_batch_pointer += batch_size

    return np.array(batch_images), np.array(batch_labels)

#Ucitavanje VGG16 modela i mapa u koje ce se spremati bottleneck featuresi
model = VGG16(weights='imagenet', include_top=False)
SAVEDIR = "Bottleneck_Features/Bottleneck_TestAnimated/"
SAVEDIR_LABELS = "Bottleneck_Features/TestAnimated_Labels/"
batch_size = 10
#For petlja za iteraciju kroz cijelu bazu za test
for i in range(int(len(Test_Animated)/batch_size)):
    x,y = loadTestAnimatedBatch(batch_size)
    print("Batch {} loaded".format(i+1))
    np.save(os.path.join(SAVEDIR_LABELS, "bottleneck_labels_{}".format(i+1)), y)
    print("Creating bottleneck features for batch {}".format(i+1))
    bottleneck_features = model.predict(x)
    np.save(os.path.join(SAVEDIR, "bottleneck_{}".format(i+1)), bottleneck_features)
    print("Bottleneck features for batch {} created and saved\n".format(i+1))
```

TRENIRANJE MREŽE

```
#Ubacivanje potrebnih paketa
import os
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from PIL import Image
import glob
import cv2
from sklearn.model_selection import train_test_split
from keras.layers import Dropout, Dense
from keras.layers.normalization import BatchNormalization
from keras.models import Sequential, load_model
from keras.applications import VGG16
from sklearn.metrics import accuracy_score, confusion_matrix

#Učitavanje baza za trening, procjenu i test
no_of_classes = 5
Train_Combined = pd.read_pickle("CombinedData/combined_train.pk1")
CV_Humans = pd.read_pickle("HumanData/df_human_cv.pk1")
Test_Humans = pd.read_pickle("HumanData/df_human_test.pk1")
CV_Animated = pd.read_pickle("AnimatedData/df_anime_cv.pk1")
Test_Animated = pd.read_pickle("AnimatedData/df_anime_test.pk1")

#Kreiranje modela za trening (potpuno spojeni slojevi)
def model (input_shape):
    model = Sequential()

    model.add(Dense(512, activation = 'relu', input_dim = input_shape))
    model.add(Dropout(0.1))

    model.add(Dense(256, activation = 'relu'))

    model.add(Dense(128, activation = 'relu'))
    model.add(BatchNormalization())

    model.add(Dense(64, activation = 'relu'))
    model.add(Dense(output_dim = no_of_classes, activation = 'softmax'))

    return model

#Mape koje sadrže bottleneck featurse svih slika
```

```
SAVEDIR_COMB_TRAIN = "Bottleneck_Features/Bottleneck_CombinedTrain/"
SAVEDIR_COMB_TRAIN_LABELS = "Bottleneck_Features/CombinedTrain_Labels/"

SAVEDIR_CV_HUMANS = "Bottleneck_Features/Bottleneck_CVHumans/"
SAVEDIR_CV_HUMANS_LABELS = "Bottleneck_Features/CVHumans_Labels/"

SAVEDIR_CV_ANIME = "Bottleneck_Features/Bottleneck_CVAnimated/"
SAVEDIR_CV_ANIME_LABELS = "Bottleneck_Features/CVAnimated_Labels/"

SAVER = "Model_Save/"

input_shape = 10*10*512

#Compile modela
model = model(input_shape)
model.summary()
model.compile(loss = 'categorical_crossentropy', optimizer = 'adam', metrics = ['accura

#Odredivanje krugova treninga i batch_sizea
epochs = 15
batch_size = 10
step = 0
combTrain_bottleneck_files = int(len(Train_Combined) / batch_size)
CVHuman_bottleneck_files = int(len(CV_Humans) / batch_size)
CVAnime_bottleneck_files = int(len(CV_Animated) / batch_size)
epoch_number, CombTrain_loss, CombTrain_acc, CVHuman_loss, CVHuman_acc, CVAnime_loss, CV

#Trening modela pomocu for petlje i iteracija kroz sve bottleneck featurese
for epoch in range(epochs):
    avg_epoch_CombTr_loss, avg_epoch_CombTr_acc, avg_epoch_CVHum_loss, avg_epoch_CVHum_a
    epoch_number.append(epoch+1)

    for i in range(combTrain_bottleneck_files):

        step += 1

        X_CombTrain_load = np.load(os.path.join(SAVEDIR_COMB_TRAIN, "bottleneck_{}.npy"
        X_CombTrain = X_CombTrain_load.reshape(X_CombTrain_load.shape[0], X_CombTrain_lo
        Y_CombTrain = np.load(os.path.join(SAVEDIR_COMB_TRAIN_LABELS, "bottleneck_label

        X_CVHuman_load = np.load(os.path.join(SAVEDIR_CV_HUMANS, "bottleneck_{}.npy".fo
        X_CVHuman = X_CVHuman_load.reshape(X_CVHuman_load.shape[0], X_CVHuman_load.shape
        Y_CVHuman = np.load(os.path.join(SAVEDIR_CV_HUMANS_LABELS, "bottleneck_labels_{
```

```
            X_CVAnime_load = np.load(os.path.join(SAVEDIR_CV_ANIME, "bottleneck_{}.npy".for
            X_CVAnime = X_CVAnime_load.reshape(X_CVAnime_load.shape[0], X_CVAnime_load.shape
            Y_CVAnime = np.load(os.path.join(SAVEDIR_CV_ANIME_LABELS, "bottleneck_labels_{}

            CombTrain_Loss, CombTrain_Accuracy = model.train_on_batch(X_CombTrain, Y_CombTra
            CVHuman_Loss, CVHuman_Accuracy = model.test_on_batch(X_CVHuman, Y_CVHuman)
            CVAnime_Loss, CVAnime_Accuracy = model.test_on_batch(X_CVAnime, Y_CVAnime)

            print("Epoch: {}, Step: {}, CombTr_Loss: {}, CombTr_Acc: {}, CVHum_Loss: {}, C

            avg_epoch_CombTr_loss += CombTrain_Loss / combTrain_bottleneck_files
            avg_epoch_CombTr_acc += CombTrain_Accuracy / combTrain_bottleneck_files
            avg_epoch_CVHum_loss += CVHuman_Loss / combTrain_bottleneck_files
            avg_epoch_CVHum_acc += CVHuman_Accuracy / combTrain_bottleneck_files
            avg_epoch_CVAnime_loss += CVAnime_Loss / combTrain_bottleneck_files
            avg_epoch_CVAnime_acc += CVAnime_Accuracy / combTrain_bottleneck_files

        print("Avg_CombTrain_Loss: {}, Avg_CombTrain_Acc: {}, Avg_CVHum_Loss: {}, Avg_CVHu

        CombTrain_loss.append(avg_epoch_CombTr_loss)
        CombTrain_acc.append(avg_epoch_CombTr_acc)
        CVHuman_loss.append(avg_epoch_CVHum_loss)
        CVHuman_acc.append(avg_epoch_CVHum_acc)
        CVAnime_loss.append(avg_epoch_CVAnime_loss)
        CVAnime_acc.append(avg_epoch_CVAnime_acc)

        model.save(os.path.join(SAVER, "modelDva.h5"))
        model.save_weights(os.path.join(SAVER, "model_weightsDva.h5"))
        print("Model and weights saved at epoch {}".format(epoch+1))


#Podaci o točnosti i greškama spremljeni u tablicu
log_frame = pd.DataFrame(columns = ["Epoch", "Comb_Train_Loss", "Comb_Train_Accuracy",
log_frame["Epoch"] = epoch_number
log_frame["Comb_Train_Loss"] = CombTrain_loss
log_frame["Comb_Train_Accuracy"] = CombTrain_acc
log_frame["CVHuman_Loss"] = CVHuman_loss
log_frame["CVHuman_Accuracy"] = CVHuman_acc
log_frame["CVAnime_Loss"] = CVAnime_loss
log_frame["CVAnime_Accuracy"] = CVAnime_acc
log_frame.to_csv("Logs/Log.csv", index = False)
```

TESTIRANJE EMOCIJA U STVARNOM VREMENU

```
#Ubacivanje potrebnih paketa
import  cv2
import numpy as np
from keras.models import load_model
from keras.applications import VGG16
import pyaudio

#Ucitavanje treniranog modela
EMOTION_DICT = {1:"LJUTNJA", 2:"STRAH", 3:"SRECA", 4:"NEUTRALNO", 5:"TUGA"}
model_VGG = VGG16(weights='imagenet', include_top=False)
model_top = load_model("Model_Save/model.h5")

#Ucitavanje klasifikatora za detekciju lica
face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')

#Postavke za snimanje zvuka
ref = 32768
FORMAT = pyaudio.paInt16
CHANNELS = 1
RATE = 16000
CHUNK = 2**10

p = pyaudio.PyAudio()
frames = []
ham = np.hamming(CHUNK)

stream = p.open(
    format=FORMAT,
    channels=CHANNELS,
    rate=RATE,
    input=True,
    output=True,
    frames_per_buffer=CHUNK
)

#Funkcija koja prebacuje snimljeni zvuk u decibele
def zvuk():
    data = stream.read(CHUNK)
    frames = np.fromstring(data, dtype=np.int16)
    data_f = np.fft.rfft(frames)
    s_mag = np.abs(data_f) * 2/np.sum(ham)
    s_dbfs = 20*np.log10(s_mag/ref)
```

```
    K = 120
    s_db = s_dbfs + K
    db = np.mean(s_db)
    db = round(db)

    return db

#Procjena emocija na temelju glasnoće
def zvuk_scale(db):
    if db>0 and db<=45:
        db_scale = np.array([0.05, 0.1, 0.1, 0.4, 0.1])
    elif db>=46 and db<=60:
        db_scale = np.array([0.1, 0.2, 0.3, 0.1, 0.2])
    elif db>=61 and db<=70:
        db_scale = np.array([0.3, 0.2, 0.2, 0.1, 0.1])
    elif db>=71 and db<=100:
        db_scale = np.array([0.4, 0.3, 0.05, 0, 0.1])
    else:
        print("Something is wrong")

    return db_scale

#Procjena emocija na temelju inteziteta pokreta
def flow_det(flow_a):
    flow_a = round(flow_a)
    if flow_a>0 and flow_a<=150000:
        flow_scale = np.array([0.1, 0.2, 0.2, 0.4, 0.1])
    elif flow_a>=150001 and flow_a<=350000:
        flow_scale = np.array([0.2, 0.2, 0.3, 0.1, 0.2])
    elif flow_a>=350001 and flow_a<=600000:
        flow_scale = np.array([0.3, 0.1, 0.2, 0.1, 0.1])
    elif flow_a>=600001 and flow_a<=15000000:
        flow_scale = np.array([0.4, 0.1, 0.1, 0, 0.1])
    else:
        print("Something is wrong")

    return flow_scale

#Procjena emocija na temelju mimike lica
def return_prediction(frame):
    faces = face_cascade.detectMultiScale(gray, 1.3, 5)
    for (x,y,w,h) in faces:
        face_clip = frame[y:y+h, x:x+w]
```

```
        cv2.imwrite('slika.jpg', cv2.resize(face_clip, (350, 350)))
    read_image = cv2.imread('slika.jpg')
    read_image = read_image.reshape(1, read_image.shape[0], read_image.shape[1], read_im
    read_image_final = read_image/255.0
    VGG_Pred = model_VGG.predict(read_image_final)
    VGG_Pred = VGG_Pred.reshape(1, VGG_Pred.shape[1]*VGG_Pred.shape[2]*VGG_Pred.shape[3]
    top_pred = model_top.predict(VGG_Pred)
    print("LJUTNJA: {}\nSTRAH: {}\nSREĆA: {}\nNEUTRALNO: {}\nTUGA: {}\n\n".format(top_

    return top_pred

#Funkcija izracuna konacnog rezultata
def calc_rez(sou, flo, mim):
    s = sou + flo + mim
    suma = np.sum(s)
    rez = []
    for i in s:
        pro = i/suma
        rez.append(pro)

    return rez

#Pokretanje kamere
cap = cv2.VideoCapture(cv2.CAP_DSHOW + 1)
_, f = cap.read()
old_gray = cv2.cvtColor(f, cv2.COLOR_BGR2GRAY)
text = "NONE"

#While petlja koja se vrti sve dok se kamera ne ugasi
while (cap.isOpened()):
    _, frame = cap.read()
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    font = cv2.FONT_HERSHEY_SIMPLEX
    cv2.putText(frame, "Last Emotion was "+str(text), (95,30), font, 1.0, (255, 0, 0),
    cv2.putText(frame, "Press SPACE: FOR EMOTION", (5,470), font, 0.7, (255, 0, 0), 2,
    faces = face_cascade.detectMultiScale(gray, 1.3, 5)
    for x,y,w,h in faces:
        cv2.rectangle(frame, (x,y), (x+w, y+h), (255, 0, 0), 2)
    flow = cv2.calcOpticalFlowFarneback(old_gray, gray, None, 0.5, 3, 15, 3, 5, 1.1, 0)
    old_gray = gray
    sound = zvuk()
    cv2.imshow("Image", frame)
```

```
    if cv2.waitKey(1) == ord(' '):
                flow_a = np.abs(flow)
        flow_a = np.sum(flow_a)
        print("OpticalFlow: {0:.2f}".format(flow_a))
        flow_a_scale = flow_det(flow_a)
        print("Sound: {}".format(sound))
        sound_scale = zvuk_scale(sound)
        mimicry = return_prediction(gray)
        rez = calc_rez(sound_scale, flow_a_scale, mimicry)
        emotion_label = rez[0].argmax() + 1
        emotion = EMOTION_DICT[emotion_label]
        text = emotion
        print("Rezultati:")
        print("LJUTNJA: {}\nSTRAH: {}\nSREĆA: {}\nNEUTRALNO: {}\nTUGA: {}\n".format(re
        print("Dominantna emocija: {}".format(emotion))

    if cv2.waitKey(1) == ord('q'):
        cap.release()
        cv2.destroyAllWindows()
        break

cap.release()
cv2.destroyAllWindows()
```