SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 000

Prepoznavanje emocija iz izraza lica pomoću strojnog učenja

Matej Ciglenečki

Zagreb, lipanj 2020.

Umjesto ove stranice umetnite izvornik Vašeg rada.
Da bi ste uklonili ovu stranicu obrišite naredbu `\izvornik`.

Zahvaljujem se mentoru na izdvojenom vremenu kojim mi je pomogao sa korisnim savjetima i uvidima tijekom pisanja ovog završnog rada

SADRŽAJ

1. Uvod

Većina komunikacije među ljudima je neverbalna. Ton glasa, geste, govor tijela i izrazi lica često vjernije prenose informacije od riječi. Običaji, način govora i kontekst trenutačne situacije govor čine podložnim mnoštvu različitih interpretacija dok je neverbalna komunikacija intuitivnija i univerzalnija. Ljudski dojmovi o drugim ljudima stvoreni su iz naoko suptilnih ali itekako važnih oblika neverbalne komunikacije. Saznanja i objašnjenja kognitivnih procesa iza ovih krajnje uobičajenih pojava još uvijek se aktivno istražuju.

Strojno učenje se zadnjih godina počelo primjenjivati na probleme poput razumijevanje govora, raspoznavanje objekata na slikama i ostalim područjima koje su za čovjeka intuitivne i lake. Računalni vid kao novija grana znanosti u nekoliko godina naišla je na mnoštvo otkrića i inovacija. Susret sa računalnim vidom događa se gotovo svakodnevno, od kamera i sustava sposobnih za prepoznavanje lica koji su se počeli implementirati od strane mnogih država do personaliziranih filtara za lice popularne na društvenim mrežama. Jedan od takvih izazova računalnog vida je raspoznavanje ljudskih emocija na temelju izraza lica. Iako uporaba ove tehnologije još nije izašla u širu primjenu trenutno već postoje slučajevi gdje se ova tehnologija pokazala korisnom. Automobil s mogućnošću prepoznavanjem emocija može upozoriti vozača kad je umoran ili pod većim nesvjesnim stresom kako ne bi dovodio sebe i druge sudionike u prometu u opasnost. Raspoznavanje emocija nalazi primjeni i u razgovorima za posao. Kvalitetno treniran model objektivniji je od ljudskog ispitivača koji može imati subjektivne predrasude zbog čega u nekim slučajevima model može točnije procijeniti emocije osobe koju se ispituje, rezultirajući u bolji uvid u ponašanje i karakteristike osobe. Nadalje, ovakva se tehnologija može koristiti i u istraživanju tržišta. Mišljenjem korisnika o proizvodima dobivena putem razgovora i anketa često su netočna ili iskrivljena zbog nedostatka spomenute neverbalne komunikacije. Analiza emocija korisnika za vrijeme interakcije s proizvodom puno je reprezentativniji pokazatelj korisnikovog mišljenja.

2. Podatkovni skup

2.1. Uvod

Podatkovni skup sastavni je dio u izvedbi treniranja modela. Treniranje se svodi na ulazne i izlazne podatke gdje su ulazni podaci podskup podatkovnog skupa. Korišteni podatkovni skupovi su Cohn-Kanade (CK) i skup slika preuzetih sa Google-a na temelju ključne riječi. Podatkovni skup dijelimo na dva djela. Skup za treniranje i skup za testiranje. Svi podaci koji su u skupu za treniranje iskorištavaju se za treniranje i optimiziranje odabranog modela dok se skup za testiranje koristi samo za evaluaciju točnosti treniranog modela. 80% nasumičnih slika odabrane su za treniranje a ostalih 20% koristi se za evaluaciju.

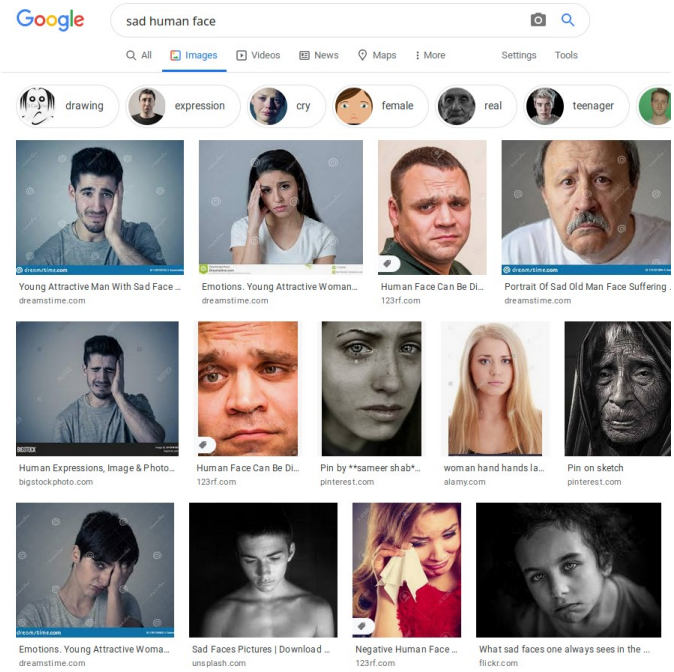
2.2. Extended Cohn-Kanade podatkovni skup

Cohn-Kanade podatkovni skup sastoji se od 593 sekvenci slika od 123 subjekta (osoba). Pojedina sekvenca sastoji se od 10 do 60 slika. Početna slika u sekvenci je neutralna emocija dok je zadnja slika vrhunac izraza emocije. Subjekti na slikama imaju od 18 do 50 godina, 69% su žene, 81% euro-Amerikanci i 6% su subjekti ostalih rase. Rezolucija pojedine slike iznosi 640x480 ili 640x490 piksela u 8-bitnom crno-bijelom ili 24 bitnom puno-bojnom formatu[7].

Slika 2.1: Primjer slike iz CK+ podatkovnog skupa

2.3. Ručno generirani podatkovni skup

Slike CK+ podatkovnog skupa slikane su u istom okruženju (ista prostorija u kojoj se slikaju subjekti, ista kamera, slična svjetlina slike...). Nedostatak raznolikosti među slikama stvara potrebu za uvođenjem apstraktnijih slika ljudskih lica ne bi li model klasificirao emocije ljudskih lica koja nisu slična samo CK+ podatkovnom skupu. Zbog toga se uvodi podatkovni skup Google slika. Google omogućuje pretraživanje slika po zadanom upitu te dodatnim parametrima koji olakšavaju pronalaženje ljudskog lica koji predstavlja određenu emociju. Npr. pronalazak ljudskih lica koji iskazuju tužnu emociju mogao bi biti "sad human face" sa tipom Google slike "lice". Rezultati tog upita prikazani su na slici 2.2.



Slika 2.2: Rezultati Google upita "sad human face"

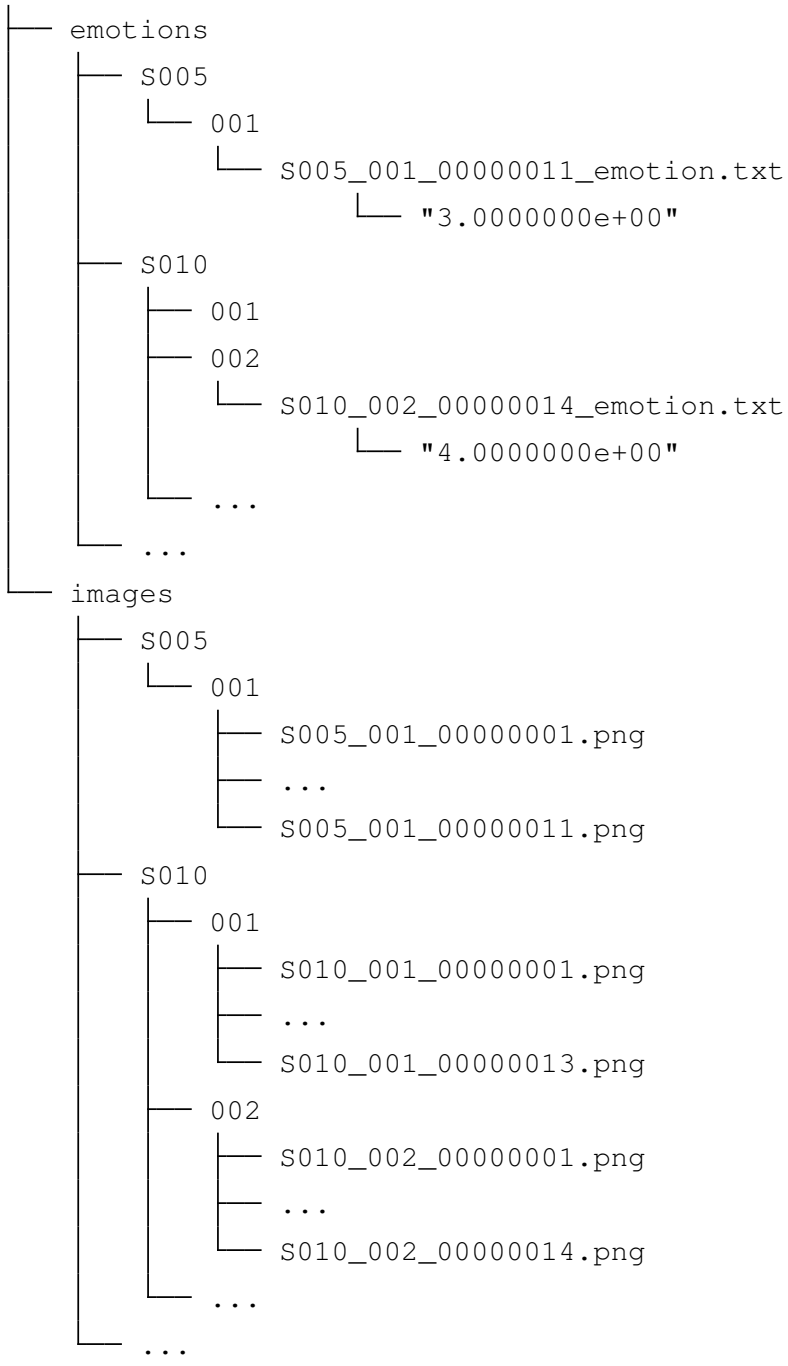
Rezultati ovog upita su slike veće raznolikosti (različit scenarij i drugačiji kut gledanja) što pridonosi apstrakciji emocije u ukupnom podatkovnom skupu. Dobivene slike ne odgovaraju uvijek nužno zadanom upitu zbog čega je potrebno dodatno provjeriti valjanost pojedine slike. Proces filtriranja objašnjen je u poglavlju 2.4.2

2.4. Priprema podatkovnih skupova

2.4.1. Priprema Cohn-Kanade podatkovnog skupa

Struktura podataka

Dijelovi podatkovnog skupa značajni za treniranje dijele se na direktorij sekvence i emocije. Svaki subjekt (npr. S005) ima svoj direktoriji u kojem se nalaze pod direktoriji za emociju koju je subjekt odglumio (npr. 001, 002...) a u njemu se nalaze sekvence. Za 327 sekvenca postoji odgovarajuća emocija koja je definirana samo za krajnju sliku sekvence i njezina putanja je definirana jednako kao i za sekvencu.



Slika 2.3: Struktura podataka CK+ podatkovnog skupa

Obrada podataka

Jedina slika u sekvenci za koju je definirana emocija je zadnja slika, što znači da je potrebno je potrebno odrediti vektor emocije ostalih slika u sekvenci na temelju krajnje vrijednosti emocije. Emocija za pojedinu sliku definirana je kao red emocija. Svaki in-

deks reda predstavlja emociju kojih ima kojih ima 8. Indeks pojedine emocije definiran je na slici 2.4 a vrijednost na indeksu predstavlja intenzitet emocije

```
emocije = {
    1: "neutral",
    2: "anger",
    3: "contempt",
    4: "disgust",
    5: "fear",
    6: "happy",
    7: "sadness",
    8: "surprise",
}
```

Slika 2.4: Deklaracija emocija

Znajući da je emocija za početnu sliku neutralna a za krajnju maksimalna sekvencijska emocija, emocije za ostale slike dodijeljene su linearno na način da se odredi intenzitet neutralne emocije 2.1 i sekvencijske emocije 2.2 gdje je n ukupan broj slika u sekvenci a i slika kojoj se određuje vektor emocije.

$$p_n = \frac{i-1}{n-1} \tag{2.1}$$

$$\begin{aligned} p_s &= 1 - p_n \\ p_s + p_n &= 1 \end{aligned} \tag{2.2}$$

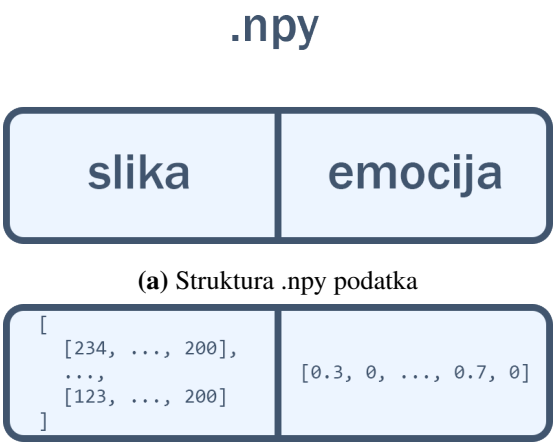
Primjer vektora emocije za sekvencu "disgust" koja sadrži 10 slika:

broj slike	vektor emocije
$i = 1$	[1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
...	...
$i = 6$	[0.4, 0.0, 0.0, 0.6, 0.0, 0.0, 0.0, 0.0]
...	...
$i = n = 10$	[0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0]
i	[p_n, 0.0, 0.0, p_s, 0.0, 0.0, 0.0, 0.0]

Slika 2.5: Vektor emocije za pojedinu pojedinu sliku u sekvneci u CK podatkovnom skupu

Spremanje slike i vektora emocije u .npy podatak

Za daljnje korištenje svaka slika i njezin vektor emocije će pretvorena u numpy red [2] i spremljen kao .npy podatak. Prvi element numpy reda je slika a drugi je odgovarajući vektor emocije. Ovime je osigurano da izračun emocija za pojedinu sliku je izračunat samo jedanput što će smanjiti vrijeme potrebno za treniranje. Nakon provođenja transformacije podataka u .npy podatak ukupan broj slika i pripadajućih vektora emocija iznosi 5703.



(a) Struktura .npy podatka

(b) Sadržaj .npy podatka

Slika 2.6: .npy podatak

2.4.2. Priprema Google podatkovnog skupa

Prikupljeni podaci

Umjesto preuzimanja jedne po jedne slike korištena je google-images-download skripta [9] koja preuzima sve moguće slike na temelju zadanog upita. Upiti korišteni za pronalaženje odgovarajućih emocija nalaze se na slici 2.7 a svi su popraćeni dodatnim "Google search" parametrom koji pretražuje slike samo ljudskih lica. Svaka slika spremljena je u direktoriji čije je ime upit koji je bio korišten prilikom preuzimanje te slike.

```
"sad human face"
"neutral human face"
"neutral expression"
"angry human face"
"angry expression"
"contempt"
"fear human face"
"fear expression"
"surprise human face"
"surprise expression"
"disgusted human face"
"disgust expression"
```

Slika 2.7: Upiti za preuzimanje ljudskih lica sa Google-a

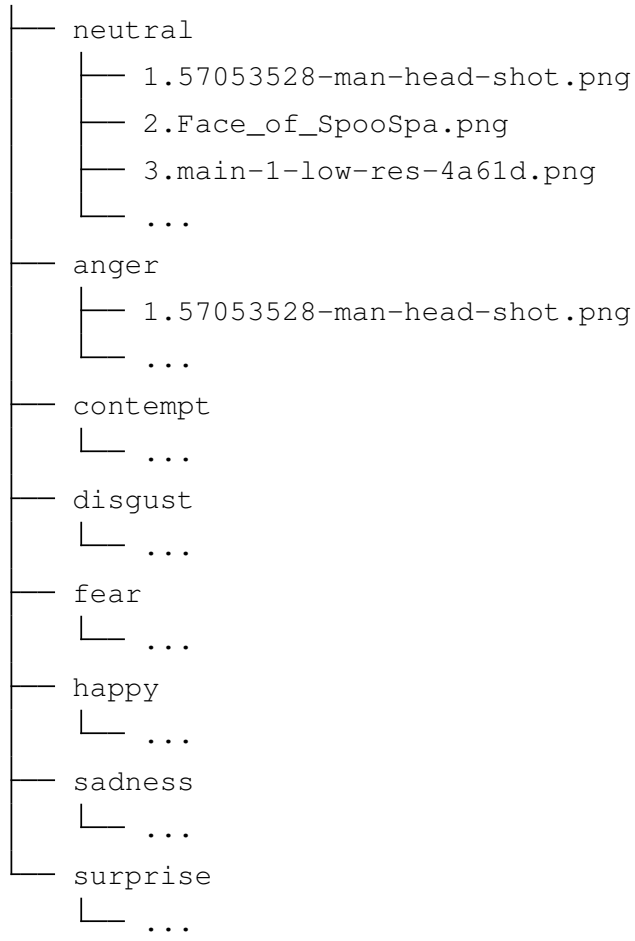
Nakon preuzimanja svih mogućih slika potrebno je ručno proći kroz svaki direktorij svakog upita i izbaciti slike koje ne zadovoljavaju sljedeće uvjete

- Na slici se nalazi samo jedno ljudsko lice
- Na slici se nalazi ljudsko lice čija emocija odgovara upitu pomoću kojeg je slika preuzeta
- Veličina slike je manja od 20MB
- Rezolucija slike je veća od 20px po duljini i visini
- Slika nije duplikat prethodno viđene slike
- Slika nije dio CK+ podatkovnog skupa

Uklanjanjem slika koje ne zadovoljavaju bilo koje od navedenih uvjeta dobiven ukupan broj slika povoljnih za treniranje iznosi 3160 a njihova ukupna veličina je 2,3GB.

Struktura podataka

Nakon filtriranja slika nepogodnih za treniranje potrebno je objediniti upite čiji su rezultati ljudska lica efektivno istih emocija. Primjer takva dva upita su "neutral human face" i "neutral expression". Nakon objedinjavanja rezultata upita i preimenovanja direktorija stvorena je struktura podataka dana na slici 2.8



Slika 2.8: Struktura podataka Google podatkovnog skupa

Obrada podataka

Obrada Google podatkovnog skupa bit će manje zahtjevna od CK+ podatkovnog skupa zbog toga što će se vektor emocije za pojedinu sliku odrediti samo na temelju upita korištenog za preuzimanje slike. Rezultat svakog vektora emocije bit će vektor dobiven metodom "One hot encoding". "One hot encoding" je metoda dodjele binarne vrijednosti za kategoriju u koju neki uzorak pripada ili ne pripada [8]. Ovom metodom uzorak (slika) može pripadati samo jednoj kategoriji (emocija). Slika koja pripada određenoj emociji za tu će emociju imati vrijednost 1 a za sve ostale 0. Na slici 2.9 prva kolumna označava ime direktorija u kojem se slike nalaze, druga kolumna predstavlja vektor emocije koje će slike u direktoriju poprimiti.

emocija (ime direktorija)	vektor emocije
neutral	[1,0,0,0,0,0,0,0]
anger	[0,1,0,0,0,0,0,0]
contempt	[0,0,1,0,0,0,0,0]
disgust	[0,0,0,1,0,0,0,0]
fear	[0,0,0,0,1,0,0,0]
happy	[0,0,0,0,0,1,0,0]
sadness	[0,0,0,0,0,0,1,0]
surprise	[0,0,0,0,0,0,0,1]

Slika 2.9: Vektor emocije za pojedinu emociju u Google podatkovnom skupu

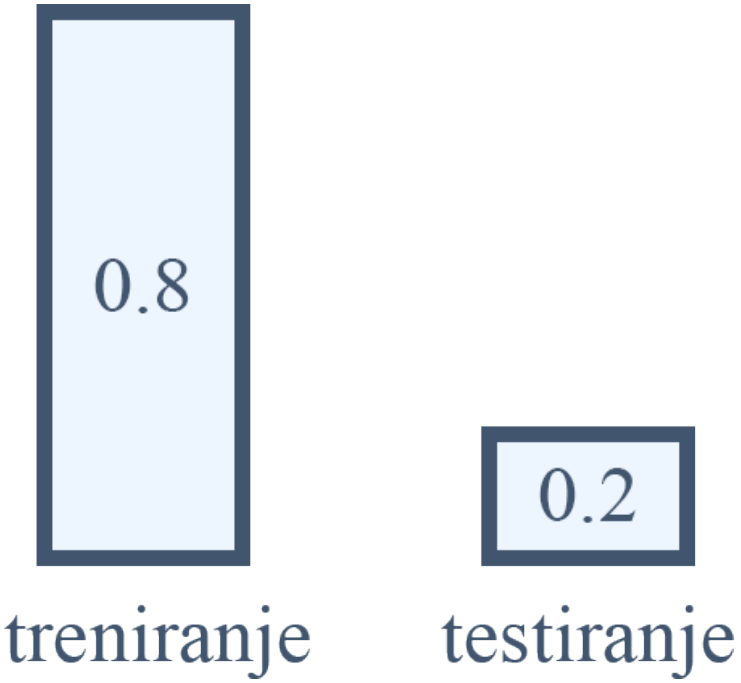
Spremanje slike i vektora emocije u .npy podatak

Nakon dodjele vektora emocije za pojedinu sliku potrebno je pretvoriti sliku i vektor emocije u .npy podatak. Kako se radi o slici i vektoru emocije, postupak pretvorbe slike i vektora emocije u pojedinačni .npy podatak jednak je kao i kod CK+ podatkovnog skupa definiranom u poglavlju 2.4.1.

2.4.3. Objedinjavanje podatkovnih skupova

Nakon obrade CK+ i Google podatkovnog skupa svi .npy podaci bit će spremljeni u direktoriji "ck" ili "Google" ovisno o tome iz kojeg je podatkovnog skupa slika dobivena. Struktura svih .npy podataka prikazana je na slici 2.11. Ovom strukturom moguće je definirati udio svakog podatkovnog skupa koji će biti korišten za treniranje modela. Poslije podjele u zasebne direktorije potrebno je odvojiti dio .npy podataka u test pod-direktoriji. U njemu će se nalaziti 20% svih .npy podataka koji će modelu biti neviđeni prilikom treniranja. Tih 20% .npy podataka koristit će se pri evaluaciji točnosti modela. Prikaz podjele podatkovnog skupa prikazan je na slici **??** Bitno je naglasiti da skup za evaluaciju ne smije ni u kojem trenutku biti izvor za treniranje mreže. Razlog toga je što model naučen na zadanom skupu slika će pri završetku treniranja biti u stanju sa vrlo visokom točnošću predvidjeti emociju slike koja je bila korištena prilikom treniranja. Evaluacija mora simulirati realnu situaciju u kojoj model nikad nije imao uvid u slike koje će biti korištene prilikom njegovog testiranja.

piechar slika



Slika 2.10: Vizualni prikaz podijeljenog podatkovnog skupa

```
numpy
├── ck
│   ├── test
│   │       ├── test_slika_i_emocija_0001.npy
│   │       ├── ...
│   │       └── test_slika_i_emocija_0570.npy
│   ├── slika_i_emocija_0001.npy
│   ├── ...
│   └── slika_i_emocija_5133.npy
└── google
    ├── test
    │       ├── test_slika_i_emocija_0001.npy
    │   │       ├── ...
    │   │       └── test_slika_i_emocija_0316.npy
    ├── slika_i_emocija_0001.npy
    ├── ...
    └── slika_i_emocija_2844.npy
```

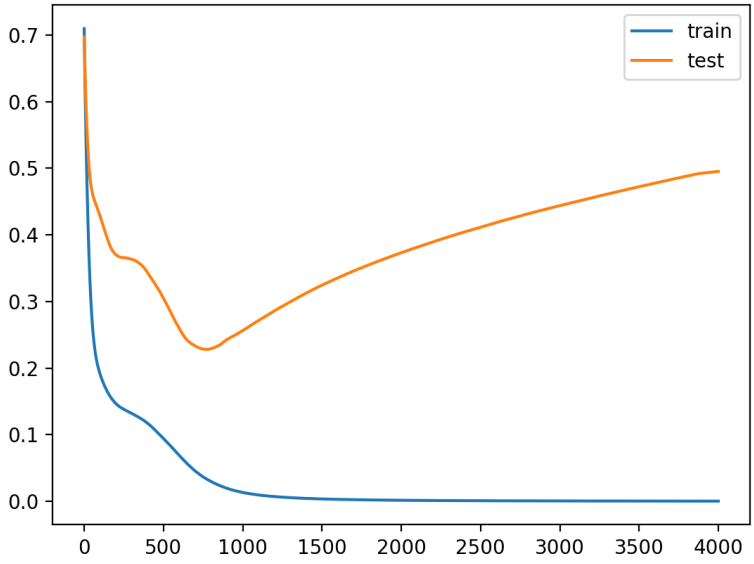
Slika 2.11: Struktura .npy podataka

3. Treniranje

Treniranje je proces u kojem model postupno mijenja svoje parametre ne bi li došao do idealnih parametara, čime bi model optimalno služio onome čemu je namijenjen. U slučaju klasifikacije za treniranje je potrebno imati skup podataka nad kojim će se provoditi treniranje i labele tih podataka koje govore klasu podataka. U slučaju klasifikacije emocija, skup podataka nad kojim se trenira su slike ljudskih lica dok je klasa emocija koja je prikazana na pojedinoj sliku. Kad treniranje započne, u model se pošalju podaci za koje model pokušava predvidjeti koje su klase ubačeni podaci. Nakon toga potrebno je usporediti predviđanja klasa koje je stvorio model sa pravim klasa. Pogrešku koju je model napravio prilikom predviđanja potrebno je javiti modelu ne bi li ispravio parametre. Mijenjanjem modela parametra model postaje sve bolji za predviđanje predviđanje rezultata na skupu za treniranje. Bitno je naglasiti da daljnjim treniranjem model ne postaje bolji za općeniti slučaj predviđanja. Treniranjem model se prilagođava podatkovnom skupu za treniranje. Podatkovni skup za treniranje je podskup svih podataka za koje je model namijenjen a to znači da skup za treniranje nije nužno najbolja mjera općenitog slučaja za podatke.

Treniranje modela nakon određenog trenutka može pogoršati sposobnost modela da predviđa klase na općenitom slučaju. Uzrok toga je što treniranjem nakon određenog trenutka model ima veću točnost na skupu za treniranje ali točnost na općenitim, neviđenim slučajevima postaje sve manja. Zbog toga je potrebno pronaći optimalan trenutak u kojem treba prekinuti treniranje modela. Prekidanje treniranja modela nije moguće odrediti pomoću točnosti modela na skupu za treniranje jer ta točnost neprestano raste daljnjim treniranjem. Zbog toga je potrebno uvesti validacijski skup koji neće biti korišten prilikom treniranja. Validacijski skup o tom slučaju igra ulogu neviđenih podataka nad kojima se mjeri točnost. Prilikom treniranja točnost nad validacijskim skupom će rasti do optimalnog trenutka gdje je greška predviđanaj najmanja. U tom trenutku točnost nad validacijskim skupom će biti maksimalna a daljnjim treniranjem točnost će se smanjivati jer model postaje previše prilagođen skupu za treniranje (engl. *overfitting*). Overfitting prikazan je na slici 3.1. Slika prikazuje overfitting zbog

toga što prilikom treniranja greška validacija poslije optimalne točke greška nad validacijskim skupom postaje sve veća i veća.



Slika 3.1: Primjer overfitting-a

3.1. Duboke neuronske mreže za analizu slika

3.1.1. Neuron

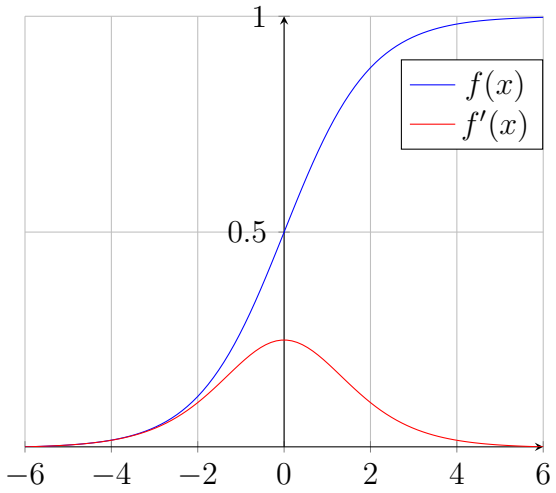
Neuron je osnovni dio neuronske mreže. Svaki neuron na svoj ulaz prima vektor x varijabli za koje neuron predviđa $\hat{y}$ izlaz. Skup x vrijednosti pomnožen je sa vektorom w koji predstavlja težine (engl. *weights*) i zbrojen sa vrijednošću b koja predstavlja sklonost (engl. *bias*). Težine neurona određuju povezanost između trenutnog i prethodnog neurona. Težištima neuroni utječu jedne na druge i određuju koliko su oni zavisni na prethodnom neuronu i za koje x_i varijable. Parametri w i b su promjenjivi i mijenjaju se prilikom treniranja a g je aktivacijska funkcija.

$$\boldsymbol{z} = w_1 x_1 + w_2 x_2 + ... + w_n x_n = \boldsymbol{w^T} \cdot \boldsymbol{x} \tag{3.1}$$

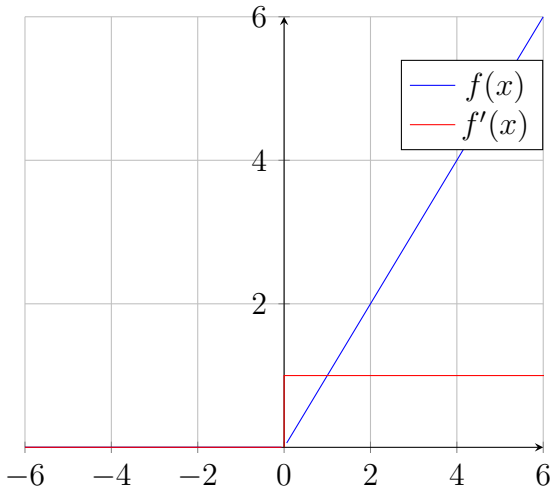
$$\hat{y} = g(\boldsymbol{z}) \tag{3.2}$$

3.1.2. Aktivacijska funckija

Nakon zbrajanja unutar neurona, izlaz neurona potrebno je provesti kroz aktivacijsku funkciju koja određuje koliko je signal tog neurona bio zastupljen u ukupnom izlazu, tj. koliko je neuron "aktiviran". Aktivnost predstavlja vezu između dva neurona a vrijednost aktivnosti govori koliko će rezultat jednog neurona utjecati na rezultat drugog. Cilj koji se postiže tom funkcijom je izlaz neurona bude postane nelinearan čime cijela neuronska mreža postaje nelinearna. Kad bi neuronska mreža uistinu bila linearan model njezina bi bila jednako dobra kao i regresijski model. Uvođenjem aktivacijske funkcije neuronska mreža može aproksimirati funkcije više reda što znači da su granice klasifikacija te neuronske mreže podijeljene granicom odluke (engl. *decision boundary*) koja nije linearna. Primjer takvih funckija je sigmoid prikazan na slici 3.2 i rektifikacijska funckija ReLU (engl. *Rectifier*) prikazana na slici 3.3. Prilikom aktivacije neurona koristiti će se rektifikacijska funkcija zbog toga što je opće primjenjiva, daje dobre rezultate i računanje je znatno zbog jednostavnosti funkcije.



Slika 3.2: Sigmoid



Slika 3.3: Linearna rektifikacijska funkcija (engl. *ReLU*)

3.1.3. Sloj neurona

Sloj neurona sastoji se od skupa neurona. Matematički, sloj neurona sastoji se od prethodno aktiviranog izlaza $\boldsymbol{a_{l_{i-1}}}$ sloja l_{i-1} i vektora $\boldsymbol{w_{l_i}}$ koji označava težišta za trenutni sloj l_i. U slučaju prvog sloja vrijedi $\boldsymbol{a} = \boldsymbol{x}$ jer prvi sloj ne ovisi ni o kojem prethodnom aktiviranom sloju. Račun izlaza sloja neurona računa se slično kao i kod pojedinog neurona. Za svaki sloj neurona l_i izlaz $\boldsymbol{z}$ jednak je umnošku prethodne aktivacije $\boldsymbol{a_{l_{i-1}}}$ sa trenutnim težinama sloja neurona $\boldsymbol{w_{l_i}}$ zbrojen sa sklonošću $\boldsymbol{b_{l_i}}$. Izlaz neurona $\hat{y}$ dobiva se tako da se vektor $\boldsymbol{z}$ aktivira aktivacijskom funkcijom g, u ovom slučaju g je funkcija ReLU prikazana na slici 3.3. Tako dobiven izlaz šalje se na sljedeći sloj neurona.

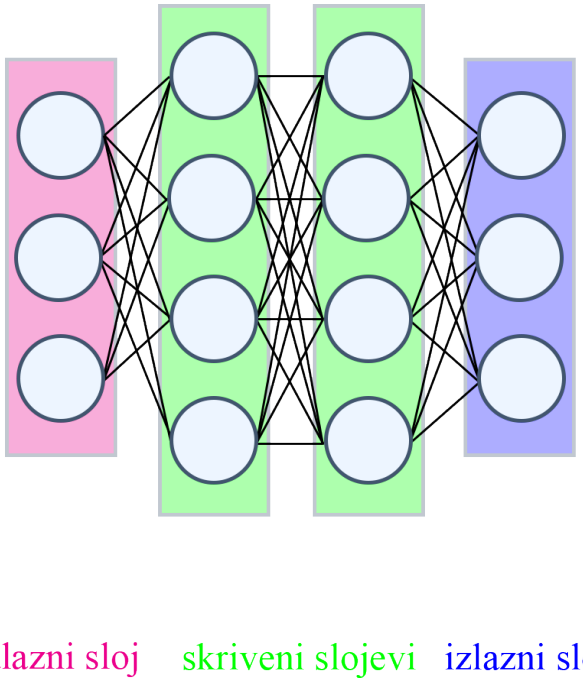
$$\boldsymbol{z_{l_i}} = \boldsymbol{w_{l_i}} \cdot \boldsymbol{a_{l_{i-1}}} + \boldsymbol{b_{l_i}} \tag{3.3}$$

$$\hat{y} = g(z) \tag{3.4}$$

3.1.4. Neuronska mreža

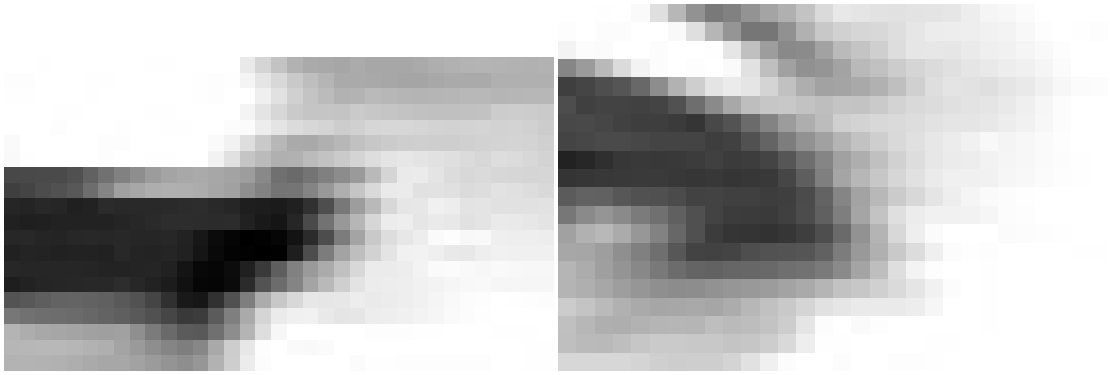
Neuronska mreža sastoji se od više slojeva neurona koji su međusobno povezani. Slojevi su povezani tako što je svaki neuronski sloj razine l_i povezan sa svakim neuronskim slojem razine l_{i+1}. Povezani su na način da se izlaz neuronskog sloja l_i dovodi na ulaz neuronskog sloja l_{i+1}. Prije dovođenja na ulaz l_{i+1} izlaz sloja l_i proveden je kroz aktivacijsku funkciju g čime je izlaz aktiviran. Ovaj algoritam prijenosa podataka

s jednog sloja na drugi zove se *feedforward* algoritam. Slojevi se dijele u tri različite grupe: ulazni, skriveni i izlazni sloj. Ulazni sloj je prvi sloj na koji se dovodi ulazni podataka, slika. Izlazni sloj je sloj koji na izlazu daje klasifikaciju ulazne slike, u ovom slučaju vektor predviđenih emocija. Skriveni slojevi su među-slojevi koji obrađuju podatke prethodnog izlaznog sloja neurona. U sredinom skrivenog područja obrađuje se najviše informacija



Slika 3.4: Različiti neuronski slojevi

Što je neuronski sloj dublje razine to je apstrakcija podataka koju on računa veća. Izlaz skrivenog l_i sloja poslat će se na ulaz sljedećeg skrivenog sloja l_{i+1} a tim prijelazom apstrakcija podataka će biti uvećana. Uloga svakog neuronskog sloja je da obrađuje podatke na razini apstrakcije definiranje njegovom dubinom. Većim brojem slojeva neurona stvara se dublja neuronska mreža koja može računati na većoj razini apstrakcije. Interpretacija razine apstrakcije u slučaju slika svodi se na promatranje različitih djelova slike različite veličine. Iako se promatraju različiti dijelovi slike, međuzavisnost neuronskih slojeva omogućava neuronskoj mreži da zna koje je svojstvo slike proizašlo iz prethodnog svojstva slike. Primjer bi bio pronalaženje ruba usnice i ruba oka na slici.



(a) Manji dio usnice, rub usnice **(b)** Manji dio oka, rub oka

Slika 3.5: Slični rubovi djelova lica

Iako ta dva rubovi izgledom mogu biti slični bitna spoznaja za neuronsku mrežu je da jedan rub proizlazi iz slike usnice a drugi iz slike oka. Tako na primjer rub usnice usmjeren prema gore ukazuje na emociju sreće, dok to za rub oka ne mora biti slučaj. Ovakvo širenje konteksta bitno je za ispravno treniranje neuronske mreže.

(a) Veći dio usnice **(b)** Veći dio oka

Slika 3.6: Dijelovi lica

3.1.5. Funkcija gubitka

Funkcija gubitka (engl. *loss function*) je funkcija koja govori koliko je neuronska mreža daleko ili blizu rezultata koji se želi postići. Rezultat funkcije gubitka bit će pogreška koju je mreža napravila prilikom predviđanja vektora emocije određene slike. Taj rezultat bit će korišten za ispravljanje parametara neuronske mreže. Funkcija gubitka

treba ukazati na kojim je predviđanjima neuronska mreža pogriješila i za koliko. Križni entropijski gubitak (engl. *Cross Entropy Loss*) je funkcija gubitka dana sa 3.5 koja će biti korištena prilikom određivanja gubitka. Funkcija prima dva parametra, ispravna vrijednost označenu kao vektor $\boldsymbol{x}$ je i predviđenu vrijednost označenu kao vektor $\boldsymbol{y}$. U slučaju predviđanja emocija $\boldsymbol{x}$ je stvaran vektor emocije za pojedinu sliku dok je $\boldsymbol{y}$ previđen vektor emocije stvoren na izlazu neuronske mreže. Vektor $\boldsymbol{w}$ predstavlja težišta za pojedini element vektora $\boldsymbol{x}$ i $\boldsymbol{y}$. Elementi vektora $\boldsymbol{w}$ su $w_1, w_2, ..., w_n$ gdje svaki w_i predstavlja koeficijent koji će biti pomnožen sa rezultatom gubitka. Razlog uvođenja težišta je neravnomjerna zastupljenost klasa u podatkovnom skupu za treniranje. Klase s manjim brojem uzoraka imat će veće težište prilikom računanja gubitka i samim time će biti "bitnije" prilikom treniranja. Kazna pogreške predviđanja za klasu emocije manjeg brojem uzoraka bit će veća nego u slučaju ne korištenja težišta.

$$CrossEntropyWeighted\left(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{w}\right) = -\sum_{i} w_i \left(x_i log\left(y_i\right)\right) \tag{3.5}$$

3.1.6. Treniranje neuronske mreže

Neuronska mreža predstavlja nelinearnu funkciju koja se sastoji od n varijabli a parametri su spomenuti parametri pojedinog neurona spomenuti na formuli 3.1, težišta neurona $\boldsymbol{w}2$ i sklonost neurona b. Mijenjanje tih parametra parametra na način da se rezultat funkcije gubitka smanjuje naziva se treniranje. Cilj treniranja je minimizirati gubitak tako da se pronađe lokalni minimum u n dimenzionalnom prostoru, gdje je broj n broj varijabli u neuronskoj mreži. Metoda pronalaska minimuma zove se spuštanje gradijentom (engl. *gradient descent*). Računanje spusta svodi se na izračun derivacije parametra neuronske mreže prema 3.6 gdje je greška neuronskog sloja E, a aktivacijski izlaz i w težina sloja.

$$\frac{\partial E}{\partial w} = \frac{\partial E}{\partial a} \cdot \frac{\partial a}{\partial w} \tag{3.6}$$

Ovom jednadžbom dobiven je smjer negativnog gradijenta koji vodi ka minimumu n dimenzionalne plohe. Brzina kojom neuronska mreža konvergira prema minimumu označena sa lr zove se stopa učenja (engl. *learning rate*). Gradijent se računa prilikom svake iteracije treniranja, tj. nakon svakog dovođenja slike na ulaz mreže. Izračunati gradijent na sloju l_i pomnožen sa stopom učenja lr povratnom vezom se šalje na prethodni sloj l_{i-1}. Sloj l_{i-1} koristiti će dobivenu vrijednost da ažurira svoja težišta sljedećim izračunom:

$$w_{l_{i-1}} - lr\frac{\partial E_{l_i}}{\partial w_{l_i}} \rightarrow w_{l_{i-1}} \tag{3.7}$$

Ovaj postupak naziva se backpropagration i primjenjuje se na sve slojeve neurona u neuronskoj mreži prilikom svake iteracije treniranja mreže.

3.1.7. Rezidualna neuronska mreža

Motivacija

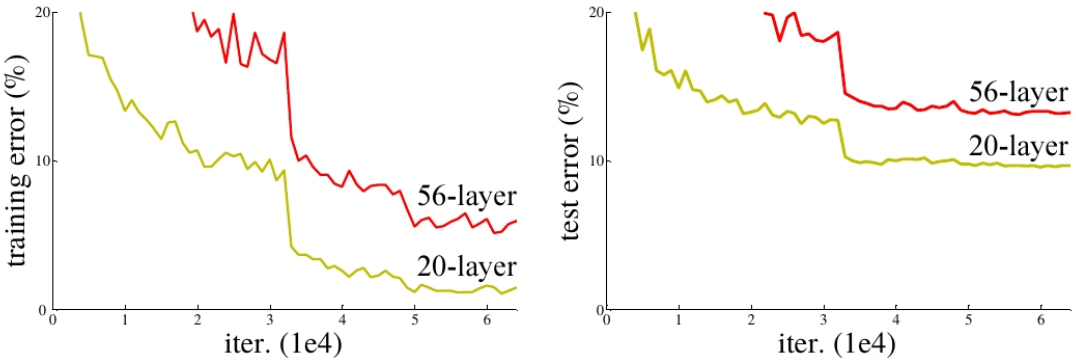
Problem prilikom treniranja duboke neuronske mreže je mogućnost nestajućeg gradijenta (engl. *vanishing gradient problem*) pri ranim neuronskim slojevima mreže. Prilikom backprogragation algoritma sloj l_i ažurirat će svoja težišta danom formulom 3.7. Bitno je naglasiti da vrijednost koja će ažurirati njegovo težište $lr\frac{\partial E_{l_n}}{\partial w_{l_n}}$ je gradijent izračunat od svih prethodnih slojeva $l_{i+1}, ..., l_n$. Ukoliko iznos gradijenta u više slučajeva bude manji od 1 težište neurona će s vremenom početi konvergirati prema 0. Što težište više konvergira prema 0 to je teže ažurirati njegovu vrijednost.

Prilikom treniranja neuronske mreže ulaz x koji prolazi kroz jedan neuronski sloj l bit će izračunat formulom 3.1. U nekim iteracijama treniranja poželjno je i izostaviti utjecaj nekog sloja l jer njegov utjecaj ne mora nužno biti potreban za iteraciju treniranja. Drugim riječima ulaz x prolazom kroz sloj l treba ostati nepromijenjen, čime bi izlaz y sloja l bio jednak x. Funkcija čija je vrijednost za argument x jednaka x zove se funkcija identiteta dana na 3.8.

$$f(x) = x \tag{3.8}$$

U praksi neuronski sloj ne pronalazi funkcija identiteta s lakoćom jer metoda pronalaska funkcije identiteta zahtjeva računalnu snagu koja je aproksimacija funkcije identiteta.

Opisan postupak je posebno problematičan kad je neuronska mreža velike dubine. Suprotno intuiciji, dodavanjem većeg broja slojeva na neuronsku mrežu gubitak neće uvijek nužno biti smanjen. Nakon određenog broja slojeva dodavanjem novih slojeva gubitak prilikom treniranja postaje sve veći a time točnost predviđanja manja. Primjer ovog problema dan je na slici 3.7.



Slika 3.7: Pogreška prilikom treniranja obične duboke neuronske mreže ovisno o dubini mreže [5]

Rezidualni blok

Rješenje problema spomenutog u motivaciji može se izbjeći korištenjem rezidualnog bloka. Taj prijedlog dan je prema [5]. Umjesto aproksimiranja identifikacijske funkcije $f(x) = x$ uvodi se izraz pomoću kojeg se funkcija identiteta lako izračuna.

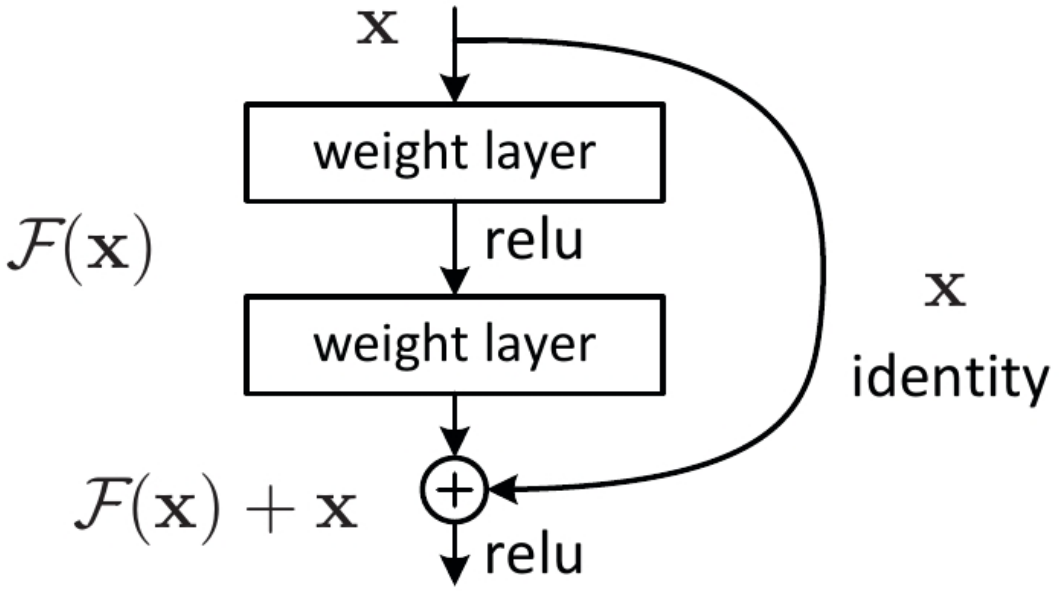
$$f(x) \coloneqq h(x) + x \tag{3.9}$$

Sad je za neuronsku mrežu lako pronaći funkciju identiteta na način da se postavi $f(x) = 0$ čime se dobiva:

$$h(x) = f(x) + x$$
$$h(x) = x$$

Ovime je osigurano da je originalan podatak x moguće proslijediti sljedećem neuronskom sloju l_{i+2} kroz neuronski sloj čak i ukoliko težišta w sloja l_i budu jednaka 0. Čak i ako neuronski sloj ne djeluje na podatak x on će ostati nepromijenjen čime je doprinos na buduće slojeve uvijek jednak ili bolji, što nije slučaj za obične duboke neuronske mreže. Ova činjenica omogućava dodavanje slojeva bez ikakvog gubitka točnosti.

Implementacija rezidualnog bloka prikazan na 3.8 ostvaruje se dodavanjem prečaca (engl. *skip connection*) između bloka l_i i bloka l_{i_2} koji služi da se podatak $\boldsymbol{x}$ koji se zbraja sa izlazom $f(x)$.



Slika 3.8: Pogreška prilikom treniranja obične duboke neuronske mreže ovisno o dubini mreže [5]

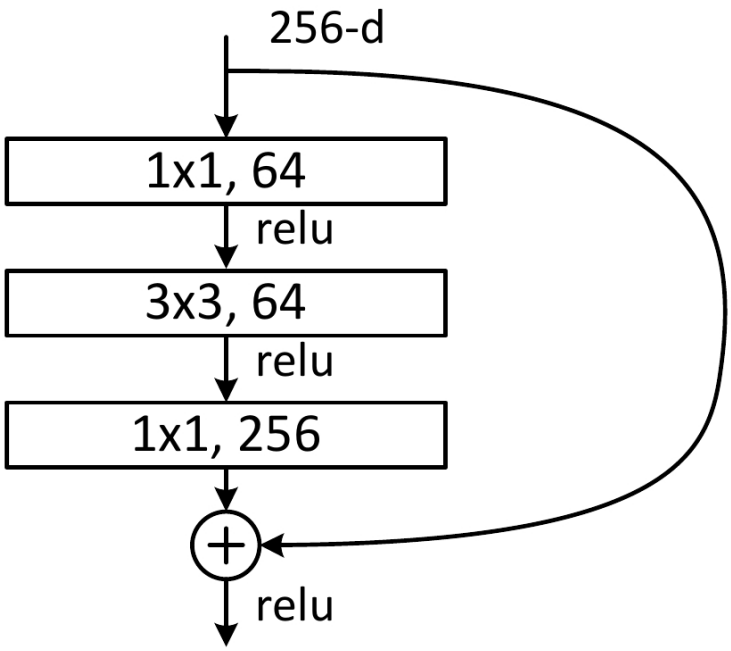
Primjer izračuna izlaza korištenjem rezidualnog bloka za aktiviran ulaz $\boldsymbol{a}_{\boldsymbol{l_i}}$, ReLU aktivacijsku funkciju g, težište $\boldsymbol{w}_{\boldsymbol{l_i}}$ te prinos $\boldsymbol{b}_{\boldsymbol{l_i}}$ gdje l_i redni broj neuronskog sloja.

$$\begin{aligned}
\boldsymbol{a}_{l_{i+2}} &= g(\boldsymbol{z}_{l_{i+2}} + \boldsymbol{a}_{l_i}) \\
\boldsymbol{a}_{l_{i+2}} &= g(\underbrace{\boldsymbol{w}_{l_{i+2}}\boldsymbol{a}_{l_{i+1}} + \boldsymbol{b}_{l_{i+2}}}_{0} + \boldsymbol{a}_{l_i}) \\
\boldsymbol{a}_{l_{i+2}} &= g(\boldsymbol{a}_{l_i}), \quad g \text{ je aktivacijska funckija ReLU} \\
\boldsymbol{a}_{l_{i+2}} &= \boldsymbol{a}_{l_i}
\end{aligned}$$

Ovime je pokazano da se po potrebi ulazni podatak lako može dobiti i na izlazu bez kompliciranih računskih operacija čime se izbjegava problema nestajućih gradijenta uz smanjenje korištenje računalne snage.

Arhitektura mreže ResNet50

Za stvaranje rezidualne neuronske mreže potrebno je ulančati više rezidualnih blokova. Konkretan rezidualan blok koji je korišten u ResNet50 arhitekturi prikazan je na slici 3.9. Cijela arhitektura ResNet50 mreže prikazana je na slici 3.10 označena sa stupcem "50-layer".



Slika 3.9: Rezidualni blok korišten u ResNet50 arhitekturi [5]

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Slika 3.10: ResNet50 arhitektura [5]

Podaci dovedeni na ulaz ResNet50 mreže prolaze kroz nju sljedećim redoslijedom:

– 1 sloj - Konvolucijski filter veličine 7×7 sa veličinom pomaka 2 koji slijedi konvolucijskim filterom veličine 3×3 uz maxpool metodu sa veličinom pomaka 2

– 9 slojeva

- Konvolucijski filter 1×1 ulaza veličine 64

- Konvolucijski filter 3×3 ulaza veličine 64
- Konvolucijski filter 1×1 ulaza veličine 256
- slojevi su ponovljeni 3 puta

– 12 slojeva
- Konvolucijski filter 1×1 ulaza veličine 128
- Konvolucijski filter 3×3 ulaza veličine 128
- Konvolucijski filter 1×1 ulaza veličine 512
- slojevi su ponovljeni 4 puta

– 18 slojeva
- Konvolucijski filter 1×1 ulaza veličine 256
- Konvolucijski filter 3×3 ulaza veličine 256
- Konvolucijski filter 1×1 ulaza veličine 1024
- slojevi su ponovljeni 6 puta

– 9 slojeva
- Konvolucijski filter 1×1 ulaza veličine 512
- Konvolucijski filter 3×3 ulaza veličine 512
- Konvolucijski filter 1×1 ulaza veličine 2048
- slojevi su ponovljeni 3 puta

– 1 sloj - zadnji sloj je sloj prosječnog sažimanja čiji je izlaz veličine 1000 na koji djeluje funkcija "softmax" koja izlazne veličine pojedine klase pretvara u vjerojatnost prisutnosti klase.

Ukupan broj navedenih slojeva je 50

3.1.8. Prijenosno učenje

Prijenosno učenje je tehnika u području strojnog gdje je naučenost modela za predviđanje jednog cilja prenesena na predviđanje drugog cilja. Potreba za ovom tehnikom javlja se iz nedovoljno velikog skupa podataka korištenog za treniranje. Zbog toga što podatkovni skup nije dovoljno velik, neuronska mreža neće biti dovoljno trenirana da predviđa emocija za neviđene slike. Unaprijed trenirana neuronska mreža ResNet50, koja je trenirana na ImageNet podatkovnom skupu [4], omogućava mreži da od

početka prepoznaje bitne značajke neviđene slike. Time je izbjegnuta potreba za treniranjem modela od početka jer su početni neuronski slojevi mreže već dobro naučeni za prepoznavanje bitnih značajki u slici, poput istaknutih objekta i uočljivijih oblika.

Zadnji slojevi unaprijed trenirane ResNet50 mreže mogu predviđati 1000 različitih značajki. Broj mogućih klasifikacijskih emocija je 8 zbog čega će biti potrebno dodati zadnji neuronski sloj na postojeću neuronsku mrežu tako da izlaz mreže bude dimenzije 8, a ne 1000 što odgovara broju različitih klasa korištenih u ImageNet podatkovnom skupu. Korištenjem većeg i apstraktnijeg podatkovnog skupa ova se tehnika može izostaviti.

3.2. Implementacija treniranja u PyTorch-u

3.2.1. Uvod

PyTorch okruženje je otvorenog koda (engl. *open source*) koja ubrzava proces implementacije strojnog učenja. Temeljne stvari strojnog učenja lakše je ostvariti u PyTorchu nego ručno pisati postupke nanovo. Iako PyTorch smanjuje kompleksnost koda i dalje je moguće po potrebi mijenjati detaljnije značajke i raditi promjene koje nisu nužno postojeće u trenutnom PyTorch okruženju pomoću nižeg Python okruženja. PyTorch će biti korišten prilikom učitavanja podataka, stvaranja podatkovnih skupova, inicijaliziranja hiperparametra, treniranja i testiranja.

3.2.2. Priprema podataka

U poglavlju 2.4 objašnjeno je stvaranje .npy podataka koji sadrže podatke slike i odgovarajućeg vektora emocije te slike. Sljedeći korak je iskoristiti te podatke za treniranje neuronske mreže koja će klasificirati emocije neviđenih slika ljudskih lica.

.npy podaci učitani su iz direktorija definiranog strukturom na slici 2.11. Prilikom učitavanja .npy podataka koristiti će se parametri `GOOGLE_TRAIN_SPLIT` i `CK_TRAIN_SPLIT` pomoću kojih je moguće kontrolirati koji će udio slika iz pojedinog podatkovnog skupa biti zastupljen u skupu za treniranje.

```
GOOGLE_TRAIN_SPLIT = 1
CK_TRAIN_SPLIT = 1
```

Slika 3.11: Udio .npy podataka pojedinog podatkovnog skupa

Sljedeći korak je stvaranje klase FERDataset (engl. *Facial Recognition Dataset*) koja implementira sučelja Dataset definiranog u PyTorchu. FERDataset sadrži dva argumenta: lokacija .npy podataka i funkcija za transformaciju slika. FERDataset sad ima pristup lokaciji .npy podataka i preko te lokacije će dohvaćati .npy podatke prilikom treniranja.

```
class FERDataset(Dataset):
    def __init__(self, filepaths_numpy, transform_image=None):
        self.filepaths_numpy = filepaths_numpy
        self.transform_image = transform_image
```

Slika 3.12: Pojednostavljena inicijalizacija FERDataset

Prilikom dohvata .npy podataka preko funkcije `__getitem__` .npy podatak je potrebno raspakirati u sliku lica i vektor emocije te slike. Ukoliko je zadana funkcija transformacije slike nju je potrebno primijeniti na sliku nakon raspakiravanja .npy podataka. Opis postupka transformacije slika nalazi se u poglavlju 3.2.2.

```
def __getitem__(self, idx):
        name_npy = filepaths_npy(idx)
        image, emotion = load_npy(name_npy)
        image = transform(image)
        return image, emotion
```

Slika 3.13: Pojednostavljeni kod dohvata i raspakiravanja .npy podataka

Podjela podatkovnog skupa

Inicijalizacijom klase koja implementira PyTorch sučelje Dataset, u ovom slučaju FERDataset, omogućeno je korištenje metoda koje lako mogu podijeliti podatkovni skup u skup za treniranje i evaluaciju. Podjela zasebnih .npy podataka već je odrađena u poglavlju 2.4 a struktura direktorija podijeljenih .npy podataka dana je na slici 2.11 i 2.10. Pomoću te dvije strukture, čija je putanja definirana sa varijablama `FILEPATHS_NUMPY` i `FILEPATHS_NUMPY`, potrebno je stvoriti zasebni podatkovni skup za treniranje i evaluaciju. Na svaku od tih putanja prethodno djeluju navedene varijable `GOOGLE_TRAIN_SPLIT` i `CK_TRAIN_SPLIT` koje određuju udio zastupljenosti pojedinog podatkovnog skupa.

```
train_dataset = FERDataset(filepaths_numpy=FILEPATHS_NUMPY,
                      transform_image=transform_image_train)
test_dataset = FERDataset(filepaths_numpy=FILEPATHS_NUMPY_TEST,
                          transform_image=transform_image_test)
```

Slika 3.14: Implementacija FERDataset klasa

Transformacija slike

Prije nego što se podaci dobiveni iz `train_dataseta` mogu koristiti za treniranje neuronske mreže potrebno ih je transformirati u pravilan oblik. Postupci transformiranja slike navedeni su ispravnim poretkom:

- Promjena veličine – iako je ovaj korak je već ostvaren u poglavlju 2.4 promjena veličine slike bit će ponovno primijenjena iz sigurnosnih razloga. Sve slike lica su postavljene na fiksnu veličinu iznosa 224x224 piksela, `IMG_SIZE = 224`, zbog uzastopne mogućnosti dijeljenja broja sa 2, pogodna za ResNet50 arhitekturu u kojoj se dijelovi slike često prepolavljaju. Promjena veličine slike izvršava se s obzirom na centar slike.
- Crno-bijelo – slike lica su u crno-bijele slike iz razloga što boje na slici ne trebaju igrati ulogu u predviđanju emocija slike lica. Oblik, kontrast i rubovi na slici lica su dovoljno značajni za prepoznavanje ekspresije ljudskog lica. Crno-bijela transformacija ne mijenja trodimenzionalnu dimenziju boja slike već samo postavlja svaku od tri RGB boja na jednaku vrijednost.
- Pretvorba u Tensor – sliku ljudskog lica potrebno je pretvoriti u tip podataka PyTorch Tensor [3] čime je PyTorch-u omogućeno da primjenjuje sve moguće ugrađene funkcije koje mogu djelovati na tip podataka PyTorch Tensor. Transformaciju koja slijedi inače nebi bilo moguće primijeniti.
- Normalizacija – korak prije slanja slike u neuronsku mrežu je normalizacija. Kako se koristi unaprijed trenirana neuronska mreža ResNet50, potrebno je provesti normalizaciju čiji parametri ovise o podatkovnom skupu koji su bili korišteni prilikom treniranja ResNet50 mreže. Ta normalizacija dana je od strane ImageNet-a [1] čiji je podatkovni skup korišten prilikom treniranja ResNet50 neuronske mreže.

Transformacije slika u PyTorch-u implementira se tako da se u `transform.Compose` zada lista željenih transformacija. Redoslijed transformacija izvršavat će se poretkom

elemenata u listi transformacija. Cjelokupnu transformaciju, kao argument pod imenom `transform_image_train`, u svom konstruktoru prima `FERDataset`, koji od sad ima pristup cjelokupnoj transformaciji. Kod i parametri transformacija dani su na slici 3.15

```
transform_image_train = transforms.Compose([
    transforms.Resize(IMG_SIZE),
    transforms.Grayscale(num_output_channels=3),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[
        0.229, 0.224, 0.225])
])
```

Slika 3.15: Postupak transformacije slike

Augmentacija slike

Augmentacija podataka (engl. *data augmentation*) je skup ne obaveznih transformacije koje povećavaju ukupnu apstraktnost podataka a samim time mogu stvoriti nepostojeće ali korisne podatke u skupu za treniranje. Ove transformacije simuliraju raznolik podatkovni skup na način da rade manje promjene na pojedinoj slici. Primjerice, nasumično povećavanje svjetline slike ili nasumična rotacija slike za kut `-AUG_DEGREE`. Na ovaj način neuronska mreža će naučiti klasificirati slike čak i ako su značajke poput svjetline slike i rotacije slične na cijelom podatkovnom skupu. To je upravo i slučaj u CK+ podatkovnom skupu. Sva su lica horizontalno poravnata čime a svjetlina slika je konzistentna. Treniranje takvog podatkovnog skupa bez augmentacije, neuronska mreža će imati lošija predviđanja emocije za slike lica gdje je lice blago zarotirano ili za znatno svjetlije ili tamnije neviđene slike. Uvođenjem augmentacije prilikom treniranja proširuje se primjena neuronske mreže koja kao rezultat može predviđati emocije za širu i apstraktniju domenu neviđenih slika. Augmentacija slike mora se obaviti prije normalizacije slike jer se augmentacija izvršava na slici kao takvoj a ne na PyTorch Tensor-u. Pojedina augmentacija primjenjivati će se nasumičnim intenzitetom za svaku sliku čime se osigurava da model ne bude pretreniran za specifičnu vrijednost augmentacije.

Augmentacije koje će biti korištene prilikom treniranja su:

- Nasumično horizontalno zrcaljenje – često korištena augmentacija za koju će model bolje predviđati emocije za općenitije slike u kojima zrcaljenje slike ili

kamere koja slika lice neće utjecati na rezultat predviđanja.

- Promjena svjetline -
- Rotacija slike – nasumična rotacija između -15 i 15 stupnjeva

Kod za potpunu transformaciju izgleda ovako:

```
transform_image_train = transforms.Compose([
    transforms.Resize(IMG_SIZE),
    transforms.Grayscale(num_output_channels=3),
    transforms.RandomHorizontalFlip(),
    transforms.ColorJitter(brightness=1)
    transforms.RandomAffine(degrees=(-AUG_DEGREE, AUG_DEGREE)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[
        0.229, 0.224, 0.225])
])
```

Slika 3.16: Implementacija FERDataset klasa

3.2.3. Hiperparametri

Funkcija gubitka

Funkcija gubitka koja se koristi prilikom treniranja opisana je u poglavlju 3.1.5

```
def CrossEntropyLossWeighted(pred, soft_targets, weights):

    # logsoftmax umjesto logaritma jer
    # izlaz modela nije proveden kroz softmax
    logsoftmax = nn.LogSoftmax(dim=1)

    plain_loss = - soft_targets * logsoftmax(pred)
    weighted_loss = weights * (plain_loss)
    result = torch.mean(torch.sum(weighted_loss, dim=1))
    return result
```

Stopa učenja

Definicija stope učenja napisana je u poglavlju 3.1.6 kao i područje na koje ona djeluje u formuli 3.7. Odabrana stopa učenja će biti postavljena na inicijalnu vrijednost 0.01

koja se dijeli sa 10 za svaku četvrtinu epohe, dakle vrijednosti su 0.01, 0.001, 0.0001, 0.00001. Razlog smanjivanja stope učenja prilikom treniranja je finija konvergencija prema minimumu. Ukoliko se ne koristi metoda smanjivanja stope učenja prilikom treniranja stopa učenja bit će postavljena na 0.0001.

```
LEARNING_RATE = 0.0001
```

Broj epoha

Broj epoha (engl. *epoch*) je broj koji predstavlja koji će se puta treniranje provesti nad cijelim skupom za treniranje. Odabrana vrijednost je 100.

```
EPOCHS = 100
```

Veličina serije

Veličina serije (engl. *batch size*) je veličina grupe slika koje se odjednom šalju neuronskoj mreži. Odabrana vrijednost iznosit će 16. To znači da će model u jednoj seriji predviđati emocije za 16 slika. Korištenjem serija funkcija gubitka kao rezultat vraća pogrešku za cijelu seriju.

```
BATCH_SIZE = 100
```

Slojevi ResNet50 mreže

Spomenutu unaprijed treniranu mrežu ResNet50 potrebno je uvesti u projekt. Srećom, PyTorch već ima ugrađenu unaprijed treniranu ResNet50 mrežu treniranu nad bogatim podatkovnim skupom ImageNet [4]. Prilikom treniranja mreže koristi se metoda prijenosnog učenja, više u poglavlju 3.1.8, gdje su svi slojevi mreže već unaprijed trenirani. Konvencija prilikom prijenosnog učenja je zamrznuti slojeve odgovorne za prepoznavanje početnih i temeljnih djelova slika, poput većih objekata, rubova i geometrijskih oblika. Težine preostalih slojeva bit će ažurirane prilikom treniranja za specifičan problem klasifikacije, u ovom slučaju klasifikacija emocije. Kako je ResNet50 trenirana na 1000 raznovrsnih klasa ima vrlo dobru mogućnost prepoznavanja raznolikih objekta na slici. U ovom slučaju slike lica su i dalje raznolika no postoje značajke na slikama lica koje su uvijek zajedničke (prisutnost različitih djelova lica). Zbog toga će zamrznuti slojevi biti svi osim zadnja tri sloja. Kad bi problem klasifikacije bio sličan podatkovnom skupu ImageNet zamrzavanje bi se provelo nad svim slojevima ResNet50 mreže.

```
resnet50 = models.resnet50(pretrained=True)
```

```
# zamrzavanje svih slojeva osim zadnja 3
for model_block in list(resnet50.children())[:-3]:
        for param in model_block.parameters():
            param.requires_grad = False
```

Optimizator

Optimizator koji će utjecati na mijenjanje težišta neurona prilikom backpropagation algoritma je optimizator Adam [6]. Za pronalaženje optimalnog rješenja, Adam koristi stohastički gradijentni spust uz moment djeluje na brzinu konvergencije prilikom iteracije. Moment djeluje na svaku varijablu koja se trenira.

`optimizer = optim.Adam(resnet50.parameters(), lr=LEARNING_RATE)`

3.2.4. Treniranje

Inicijalizacijom svih potrebnih hiperparametara mreža je spremna za treniranje. Jedna iteracija treniranja izvršava se nad cijelim podatkovnim skupom za treniranje. Broj takvih iteracija bit će izvršen `EPOCHS` puta. Prije početka treniranja potrebno je model prebaciti u stanje `train` čime je označeno da će model mijenjati svoja težišta prilikom iteracija. Prilikom iteracija jedne epohe događa sljedeće:

`train_loader` dohvaća jednu seriju podataka `batch`. Iz serije se izvlače podaci `face` i `emotions`. To su lista slika i emocija a obje liste sadržavaju `BATCH_SIZE` tj. 16 unikatnih slika lica i odgovarajućih emocija koje više neće biti dohvaćene prilikom trenutne epohe. 16 slika lica šalju se u model koji kao izlaz vraća 16 predviđenih emocija na temelju slika koje je upravo vidio. Funkcija gubitka (engl. *loss function*) uspoređuje predviđanja modela sa istinitim vektorom emocija `loss_func(...)` tako da računa gubitak koji je model proizveo svojim predviđanjem, točnije, koliko je model pogriješio prilikom predviđanja emocije za slike trenutne serije. Funkcija gubitka nakon toga računa derivacije gubitka za svaki parametar `loss.backwards()` a optimizator Adam koristi izračunate derivacije i stopu učenja da ažurira težišta modela `optimizer.step()`. Prije izračuna derivacija optimizator izbriše prethodno izračunate derivacije prošle serije pozivom `optimizer.zero_grad()`.

```
for epoch in range(EPOCHS):

        train_loss = 0.0
        batch_loss = 0
        val_loss = 0

        model.train()
        for batch in train_loader:
            face, emotions = batch
            optimizer.zero_grad()
            outputs = model(face) # not softmaxed
            emotions = emotions
            loss = loss_func(pred=outputs.float(),
                             soft_targets=emotions.float(),
                             weights=WEIGHTS)
            loss.backward()
            optimizer.step()
            # emotion size is same as batch size
            train_loss += float(loss) * int(emotions.size(0))

        train_loss = train_loss / train_size
        train_losses.append(train_loss)
```

Slika 3.17: Pojednostavljen kod treniranja modela

Treniranje modela prestaje nakon `EPOCHS` epoha a sprema se svaki model koji je bolji od najboljeg prethodnog. Ovime do tad najbolji model biti spremljen čak i ukoliko treniranje prestane prerano. Najbolji model mjeren je po kriteriju točnosti nad testnim podatkovnim skupom koji će biti objašnjen u sljedećem poglavlju.

```
if(epoch_acc > best_acc):
        best_acc = epoch_acc
        best_model = copy.deepcopy(resnet50.state_dict())
        torch.save(best_model, filename)
```

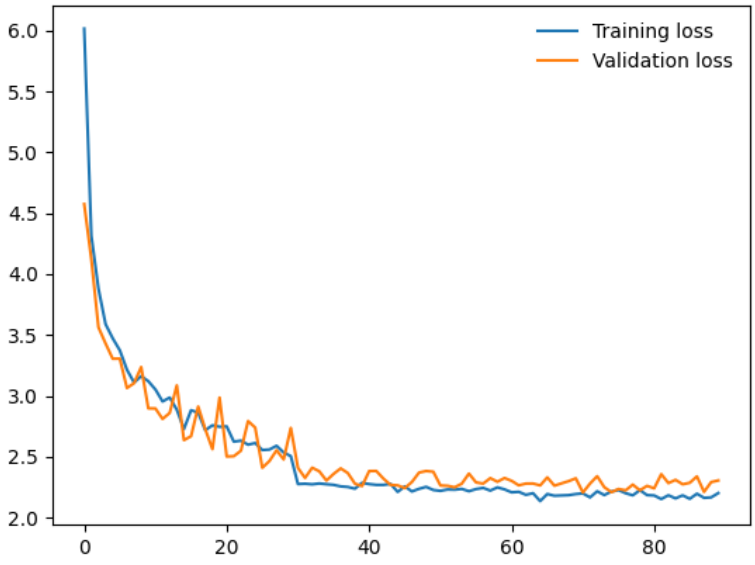
4. Rezultati

Opisati postupak evaluacija

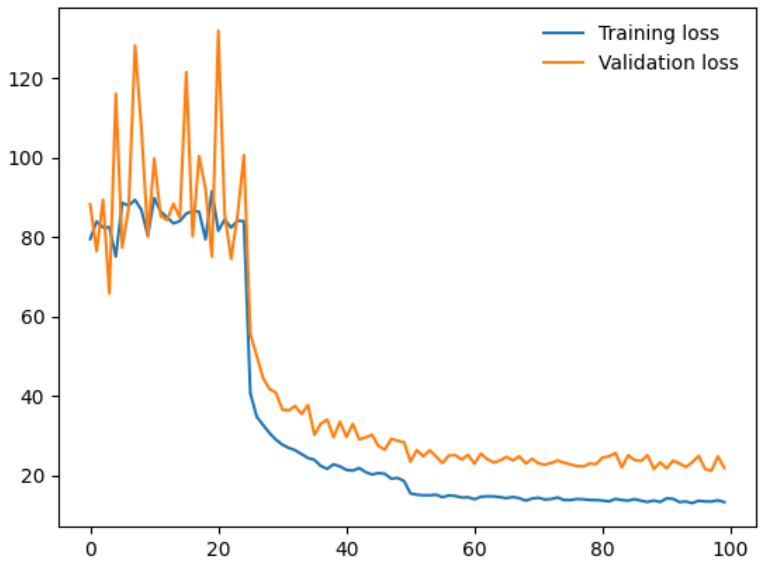
Opisati razlike Ck+ vs. CK+ Google

Generirati sliku konfuzijske matrice (podjeliti sa najvećim brojem prije toga

Opisati razlike Ck+ vs. CK+ Google dodatno opisati, 5 slika random



Slika 4.1: Greška prilikom treniranja nad CK+ podatkovnim skupom



Slika 4.2: Greška prilikom treniranja nad Google i CK+ podatkovnim skupom

4.0.1. Konfuzijska matrica

```
[17, 1 , 4 , 4 , 3 , 0 , 11, 3]
[4 , 19, 2 , 5 , 1 , 1 , 5 , 2]
[3 , 0 , 6 , 4 , 3 , 0 , 3 , 0]
[4 , 6 , 1 , 22, 4 , 2 , 4 , 1]
[0 , 0 , 1 , 5 , 12, 1 , 5 , 6]
[1 , 0 , 2 , 3 , 2 , 23, 1 , 3]
[11, 5 , 2 , 3 , 3 , 4 , 31, 3]
[1 , 2 , 1 , 2 , 2 , 1 , 0 ,26]
```

Slika 4.3: Konfuzijska matrica

5. Zaključak

Korištenjem dva podatkovna skupa i alata PyTorch istrenirani model može do određene točnosti predviđati koju emociju izražava slika ljudskog lica. Uvođenjem dodatnog ručno generiranog Google podatkovnog skupa poboljšana je apstrakcija prepoznavanja emocija koja dopušta modelu da bolje predviđa emocije sa neviđenih slika lica koje nisu slične slikama iz podatkovnog skupa CK+. Točnost predviđanja emocija može biti poboljšana uvođenjem većeg i apstraktnijeg podatkovnog skupa koji sadrži raznolike slike ljudskih lica.

LITERATURA

[1] Imagenet normalization. URL https://github.com/pytorch/examples/blob/97304e232807082c2e7b54c597615dc0ad8f6173/imagenet/main.py#L197-L198.

[2] Numpy array. URL https://numpy.org/doc/stable/reference/generated/numpy.array.html.

[3] Pytorch tensor. URL https://pytorch.org/docs/stable/tensors.html.

[4] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, i L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. U *CVPR09*, 2009.

[5] K. He, X. Zhang, S. Ren, i J. Sun. Deep residual learning for image recognition. U *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, stranice 770–778, 2016.

[6] Diederik P. Kingma i Jimmy Ba. Adam: A method for stochastic optimization, 2014. URL http://arxiv.org/abs/1412.6980. cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

[7] Patrick Lucey, Jeffrey Cohn, Takeo Kanade, Jason Saragih, Zara Ambadar, i Iain Matthews. The extended cohn-kanade dataset (ck+): A complete dataset for action unit and emotion-specified expression. stranice 94 – 101, 07 2010. doi: 10.1109/CVPRW.2010.5543262.

[8] rakshithvasudev. One hot encoding. URL https://hackernoon.com/what-is-one-hot-encoding-why-and-when-do-you-have-to-use-it-e3c618

[9] Hardik Vasa. google-images-download. URL https://github.com/hardikvasa/google-images-download.

Prepoznavanje emocija iz izraza lica pomoću strojnog učenja

Sažetak

Ljudske emocije temeljni dio svih mogućih ljudskih radnji. Treniranjem modela pomoću strojnog učenja koji može klasificirati emocije na temelju slika ljudskih lica otvaraju se mogućnosti za objektivno analiziranje ljudskog ponašanja i razmišljanja u široko primjenjivim situacijama. Uz dva podatkovna skupa ljudskih lica, ResNet50 mrežu i prijenosnim učenjem ponovno istrenirani model može klasificirati neviđene slike ljudskog lica sa 60% točnosti.

Ključne riječi: strojno učenje, duboko učenje, rezidualne neuronske mreže, emocija, lice

Recognizing Emotions From Facial Expressions Using Machine Learning

Abstract

Human emotions are the base of all human activities. Training a model by using a machine learning method which can later classify emotions given an image of a human facial expression opens up a possibility for an objective analysis of human behaviour and thought process in diverse situations. With two datasets containing images of human facial expressions, ResNet50 network and transferred learning method the trained model can classify emotions given an unseen image of human facial expression with an accuracy of 60%

Keywords: machine learning, deep learning, residual neural network, emotion, face