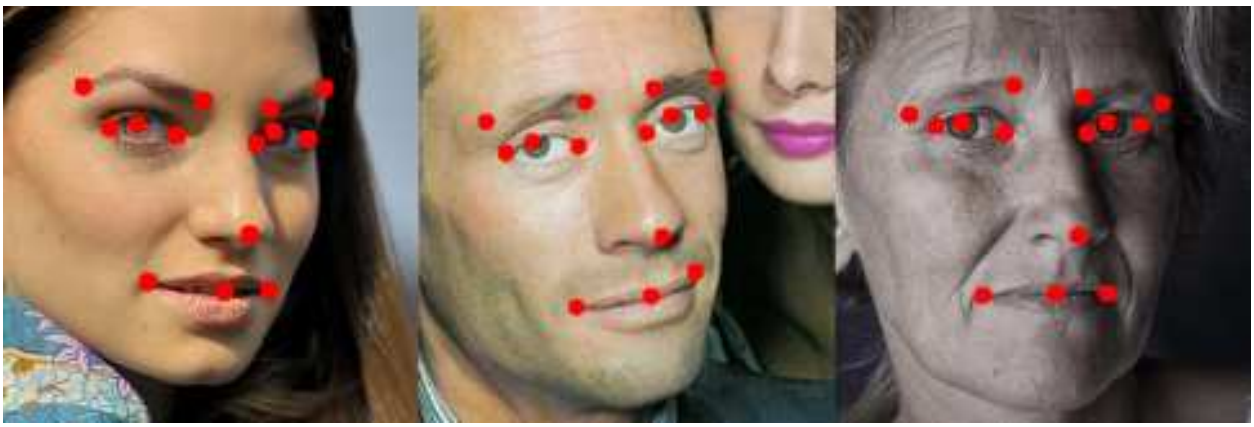
PREPOZNAVANJE KLJUČNIH TAČAKA NA LICU

Petar Simić mi15/247 mi15247@matf.bg.ac.rs

1) Uvod

U ovom radu se bavim prepoznavanjem ključnih tačaka na licu ljudi koristeći neuronske mreže, tačnije uz pomoć konvolutivnih neuronskih mreža. Konvolutivne mreže i sama njihova duboka struktura mi omogućava da sa velikom preciznošću lociram ključne tačke. U mom radu ključne tačke na licu su početak i kraj obrve, levi i desni kraj oka, zenica, vrh nosa, levi i desni kraj usne i sredina usne. Konvolutivne neuronske mreže su dobile ime po konvoluciji, operatoru koji se primenjuje u obradi slika i signala. Značajna primena je u algoritmima za detekciju ivica, pomoću kojih se otkrvaju objekti na slici. Prednost konvolutivnih mreža u odnosu na obične je velika kada se gleda sa aspekta mog rada. Na primer, kada se kao ulaz koristi slika dimenzije 200x200 piksela, to se pretvara u 40000 neurona na ulaznom sloju, što je jako veliki broj i praktično je nemoguće treniranje potpuno povezanih mreža. Još ako je slika u boji treba uzeti u obzir i 3 kanala kojima se predstavljaju osnovne boje(rgb) što nas dovodi do 120000 neurona u ulaznom sloju. Ideja konvolutivnih mreža je da postavi veći broj slojeva za otkrivanje bitnih osobina ulaznih podataka. Nedostatak konvolutivnih mreža je taj što zahteva značajne hardverske resurse, konkretno za treniranje mreže u mom radu je potrebno oko 10,5gb ram memorije.

https://www.etran.rs/common/pages/proceedings/ETRAN2017/VI/IcETRAN2017_paper_VI1_1.pdf

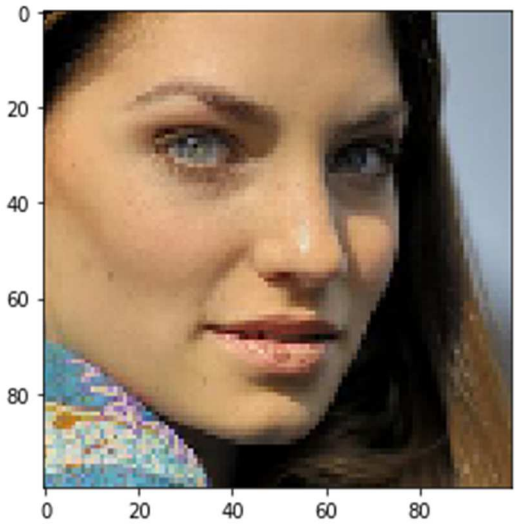


2) Opis rešenja

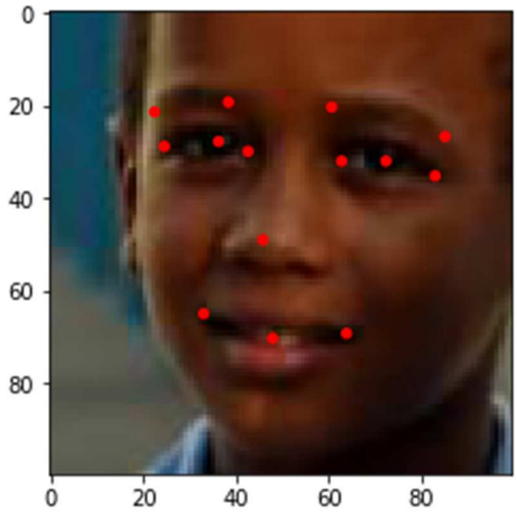
Pre svega za rešavanje ovog problema mi trebaju podaci. U folderu „data“ se nalazi folder „images“ sa 6000 slika lica, i csv datoteka koja sadrži ime slike i koordinate ključnih tačaka lica za tu sliku. Nakon učitavanja podataka sledi njihovo pretprocesiranje, u mom slučaju je to skaliranje rezolucije slike na 100x100 piksela i skaliranje koordinata ključnih tačaka na [-0,5 : 0,5]. Slike koje imam su razlicitih rezolucija i zato je potrebno da ih skaliram na istu rezoluciju jer se neuronske mreže mnogo bolje snalaze sa tako spremnim podacima, takođe skaliranje koordinata na [-0,5 : 0,5] omogucava stabilniji rad mreže.

Primer pretprocesirane slike:

```
array([-0.30534351, -0.30534351, -0.01526716, -0.27099237,  0.14503819,
       -0.25190839,  0.27480918, -0.30152673, -0.24045801, -0.20610687,
       -0.17557251, -0.21374047, -0.08015266, -0.18702289,  0.11068702,
       -0.16793892,  0.15267175, -0.19847327,  0.23664123, -0.17938933,
        0.09541982,  0.0496183 , -0.14885497,  0.16412216,  0.03435117,
        0.18702292,  0.14122134,  0.18320608])
```

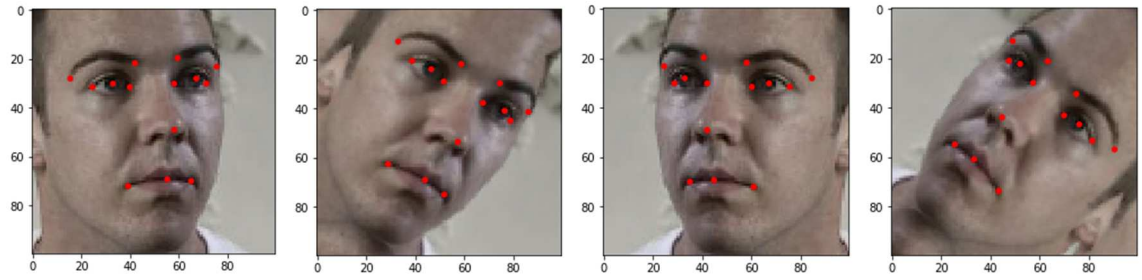


primer pretprocesiranog podatka:



Sledeći korak koji sam radio je povećanje broja podataka. S obzirom da sa što većim brojem podataka model daje preciznije i bolje lociranje tačaka, odradio sam preslikavanje slike kao odraz u ogledalu i rotaciju svake slike za neki slučajan ugao. Tako da od početnih 6000 slika dolazimo do 24000 ulaznih podataka.

primer:



Sledeći korak je deljenje podataka na trening skup i skup za validaciju, dizajn same arhitekture neuronske mreže, i treniranje modela. Arhitektura mreže je sleldeća:

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 100, 100, 3)	0
conv2d_1 (Conv2D)	(None, 100, 100, 8)	224
conv2d_2 (Conv2D)	(None, 100, 100, 8)	584
max_pooling2d_1 (MaxPooling2	(None, 50, 50, 8)	0
conv2d_3 (Conv2D)	(None, 50, 50, 16)	1168
conv2d_4 (Conv2D)	(None, 50, 50, 16)	2320
max_pooling2d_2 (MaxPooling2	(None, 25, 25, 16)	0
conv2d_5 (Conv2D)	(None, 25, 25, 32)	4640
conv2d_6 (Conv2D)	(None, 25, 25, 32)	9248
max_pooling2d_3 (MaxPooling2	(None, 12, 12, 32)	0
conv2d_7 (Conv2D)	(None, 12, 12, 64)	18496
conv2d_8 (Conv2D)	(None, 12, 12, 64)	36928
max_pooling2d_4 (MaxPooling2	(None, 6, 6, 64)	0
conv2d_9 (Conv2D)	(None, 6, 6, 128)	73856
conv2d_10 (Conv2D)	(None, 6, 6, 128)	147584

```
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 3, 3, 128)         0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 3, 3, 256)         295168
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 3, 3, 256)         590080
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 1, 1, 256)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               65792
_________________________________________________________________
dropout_1 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               32896
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_3 (Dense)              (None, 28)                3612
=================================================================
Total params: 1,282,596
Trainable params: 1,282,596
Non-trainable params: 0
_________________________________________________________________
```

Koristio sam 2D konvolutivne slojeve zbog rada sa slikama, MaxPooling2D slojeve zbog redukcije dimenzionalnosti. Nakon toga kada sam sveo ulaz na 256 bitnih instanci sam koristio potpuno povezane slojeve (Dense). Takodje sam koristio Dropout postavljen na 0.25 zbog robusnosti moje mreže.

Sada dolazi na red treniranje mreže. Da u procesu učenja ne bih koristio baš sve instance (stohastički), koristio sam mini_batch. Ideja mini_batch-a je ta da u proces učenja propustam određen broj instanci koji nije prevelik i nije premali, jer to bitno utiče na samo vreme treniranja mreže. Trening sam obavio kroz 50 epoha i pratio ponašanje funkcije greške na test skupu i skupu za validaciju. S obzirom da radim sa realnim brojevima kao funkciju greške sam koristio srednje-kvadratnu grešku(mse).

```
Train on 21600 samples, validate on 2400 samples
Epoch 1/50
21600/21600 [==============================] - 38s 2ms/step - loss: 0.0037 -
val_loss: 0.0014
Epoch 2/50
21600/21600 [==============================] - 31s 1ms/step - loss: 0.0017 -
val_loss: 0.0013
Epoch 3/50
21600/21600 [==============================] - 31s 1ms/step - loss: 0.0016 -
val_loss: 0.0012
Epoch 4/50
21600/21600 [==============================] - 31s 1ms/step - loss: 0.0013 -
val_loss: 0.0011
Epoch 5/50
21600/21600 [==============================] - 31s 1ms/step - loss: 0.0012 -
val_loss: 9.9773e-04
Epoch 6/50
```

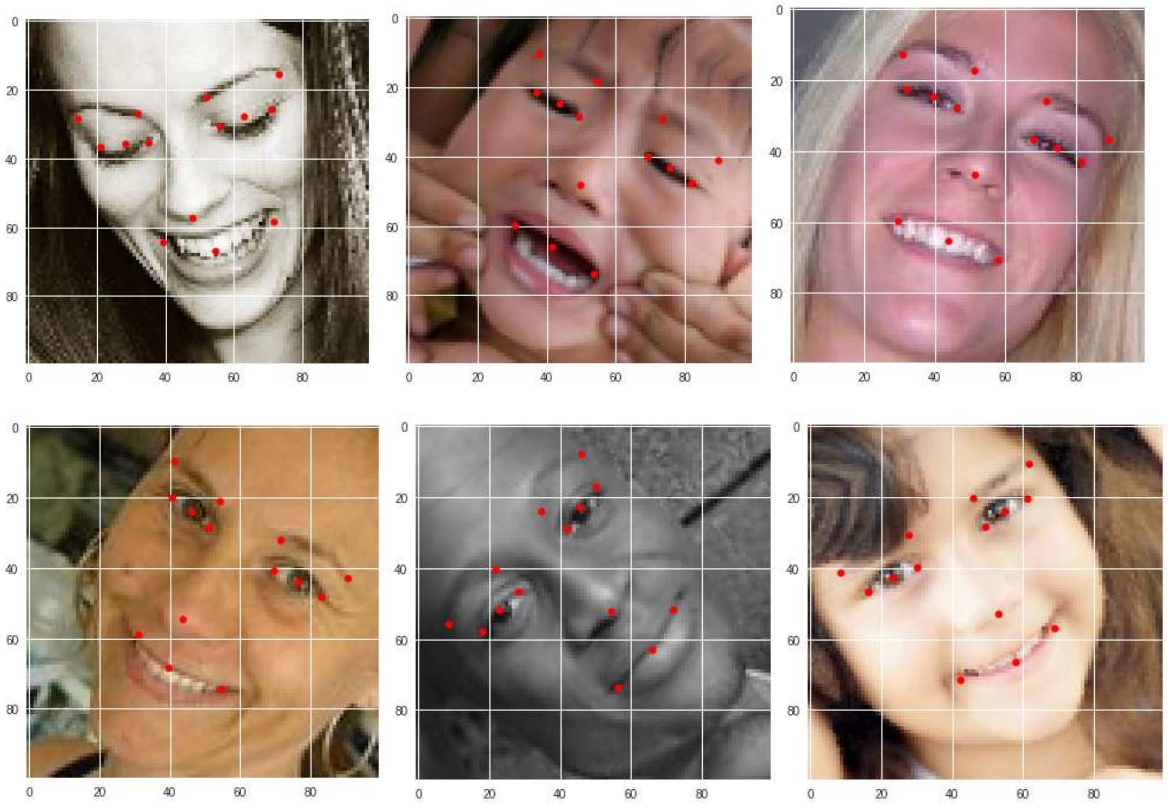
```
21600/21600 [==============================] - 31s 1ms/step - loss: 0.0011 -
val_loss: 0.0010
Epoch 7/50
21600/21600 [==============================] - 31s 1ms/step - loss: 9.6503e-
04 - val_loss: 9.4426e-04
Epoch 8/50
21600/21600 [==============================] - 32s 1ms/step - loss: 9.4637e-
04 - val_loss: 7.7187e-04
Epoch 9/50
21600/21600 [==============================] - 33s 2ms/step - loss: 0.0010 -
val_loss: 0.0011
Epoch 10/50
21600/21600 [==============================] - 33s 2ms/step - loss: 0.0012 -
val_loss: 0.0011
Epoch 11/50
21600/21600 [==============================] - 33s 2ms/step - loss: 9.1345e-
04 - val_loss: 7.9465e-04
Epoch 12/50
21600/21600 [==============================] - 33s 2ms/step - loss: 9.5620e-
04 - val_loss: 8.2554e-04

Epoch 00012: ReduceLROnPlateau reducing learning rate to
0.00010100000474974513.
Epoch 13/50
21600/21600 [==============================] - 33s 2ms/step - loss: 6.8394e-
04 - val_loss: 5.7431e-04
Epoch 14/50
21600/21600 [==============================] - 33s 2ms/step - loss: 6.0470e-
04 - val_loss: 5.4447e-04
Epoch 15/50
21600/21600 [==============================] - 33s 2ms/step - loss: 5.8031e-
04 - val_loss: 5.3681e-04
Epoch 16/50
21600/21600 [==============================] - 33s 2ms/step - loss: 5.6159e-
04 - val_loss: 5.2370e-04
Epoch 17/50
21600/21600 [==============================] - 33s 2ms/step - loss: 5.4805e-
04 - val_loss: 5.2596e-04
Epoch 18/50
21600/21600 [==============================] - 33s 2ms/step - loss: 5.3676e-
04 - val_loss: 5.3204e-04
Epoch 19/50
21600/21600 [==============================] - 33s 2ms/step - loss: 5.2708e-
04 - val_loss: 5.2143e-04
Epoch 20/50
21600/21600 [==============================] - 33s 2ms/step - loss: 5.1990e-
04 - val_loss: 5.0197e-04

Epoch 00020: ReduceLROnPlateau reducing learning rate to 1.0000000474974514e-
05.
Epoch 21/50
21600/21600 [==============================] - 33s 2ms/step - loss: 4.9611e-
04 - val_loss: 4.9229e-04
Epoch 22/50
21600/21600 [==============================] - 33s 2ms/step - loss: 4.9075e-
04 - val_loss: 4.9598e-04
Epoch 23/50
```

```
21600/21600 [==============================] - 33s 2ms/step - loss: 4.8578e-
04 - val_loss: 4.9536e-04
Epoch 24/50
21600/21600 [==============================] - 33s 2ms/step - loss: 4.8295e-
04 - val_loss: 4.9212e-04
Epoch 25/50
21600/21600 [==============================] - 33s 2ms/step - loss: 4.8838e-
04 - val_loss: 4.9328e-04

Epoch 00025: ReduceLROnPlateau reducing learning rate to 1.0000000656873453e-
06.
Epoch 26/50
21600/21600 [==============================] - 32s 2ms/step - loss: 4.8203e-
04 - val_loss: 4.9080e-04
Epoch 27/50
21600/21600 [==============================] - 33s 2ms/step - loss: 4.8058e-
04 - val_loss: 4.8987e-04
Epoch 28/50
21600/21600 [==============================] - 32s 1ms/step - loss: 4.7926e-
04 - val_loss: 4.8953e-04
Epoch 29/50
21600/21600 [==============================] - 32s 1ms/step - loss: 4.8039e-
04 - val_loss: 4.9041e-04
Epoch 30/50
21600/21600 [==============================] - 32s 1ms/step - loss: 4.8012e-
04 - val_loss: 4.9066e-04
Epoch 31/50
21600/21600 [==============================] - 32s 1ms/step - loss: 4.7795e-
04 - val_loss: 4.9033e-04
Epoch 32/50
21600/21600 [==============================] - 33s 2ms/step - loss: 4.7639e-
04 - val_loss: 4.9047e-04
Epoch 33/50
21600/21600 [==============================] - 32s 1ms/step - loss: 4.7987e-
04 - val_loss: 4.9144e-04

Epoch 00033: ReduceLROnPlateau reducing learning rate to 1.0000001111620805e-
07.
Epoch 34/50
21600/21600 [==============================] - 32s 1ms/step - loss: 4.8159e-
04 - val_loss: 4.9126e-04
Epoch 35/50
21600/21600 [==============================] - 32s 2ms/step - loss: 4.8077e-
04 - val_loss: 4.9118e-04
Epoch 36/50
21600/21600 [==============================] - 32s 2ms/step - loss: 4.7907e-
04 - val_loss: 4.9116e-04
Epoch 37/50
21600/21600 [==============================] - 32s 1ms/step - loss: 4.7714e-
04 - val_loss: 4.9106e-04
Epoch 38/50
21600/21600 [==============================] - 32s 1ms/step - loss: 4.8040e-
04 - val_loss: 4.9106e-04

Epoch 00038: ReduceLROnPlateau reducing learning rate to 1.000000082740371e-
08.
Epoch 39/50
```

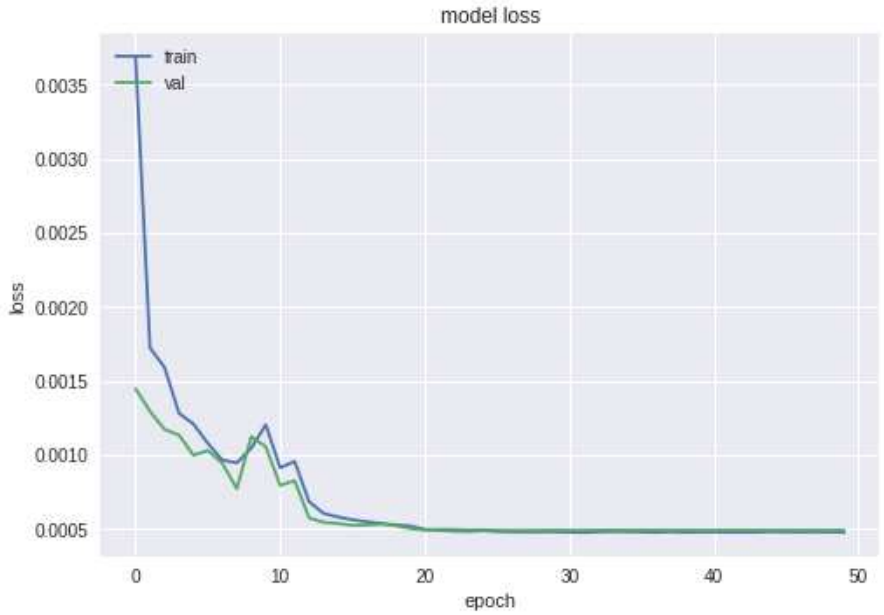
```
21600/21600 [==============================] - 32s 1ms/step - loss: 4.7726e-
04 - val_loss: 4.9105e-04
Epoch 40/50
21600/21600 [==============================] - 32s 1ms/step - loss: 4.7871e-
04 - val_loss: 4.9105e-04
Epoch 41/50
21600/21600 [==============================] - 32s 1ms/step - loss: 4.7919e-
04 - val_loss: 4.9105e-04
Epoch 42/50
21600/21600 [==============================] - 32s 1ms/step - loss: 4.7681e-
04 - val_loss: 4.9103e-04
Epoch 43/50
21600/21600 [==============================] - 32s 1ms/step - loss: 4.7769e-
04 - val_loss: 4.9103e-04

Epoch 00043: ReduceLROnPlateau reducing learning rate to 1.000000082740371e-
09.
Epoch 44/50
21600/21600 [==============================] - 32s 1ms/step - loss: 4.7807e-
04 - val_loss: 4.9103e-04
Epoch 45/50
21600/21600 [==============================] - 32s 1ms/step - loss: 4.8113e-
04 - val_loss: 4.9103e-04
Epoch 46/50
21600/21600 [==============================] - 32s 1ms/step - loss: 4.7945e-
04 - val_loss: 4.9103e-04
Epoch 47/50
21600/21600 [==============================] - 32s 1ms/step - loss: 4.7835e-
04 - val_loss: 4.9103e-04
Epoch 48/50
21600/21600 [==============================] - 32s 1ms/step - loss: 4.7962e-
04 - val_loss: 4.9103e-04

Epoch 00048: ReduceLROnPlateau reducing learning rate to 1.000000082740371e-
10.
Epoch 49/50
21600/21600 [==============================] - 32s 1ms/step - loss: 4.7903e-
04 - val_loss: 4.9103e-04
Epoch 50/50
21600/21600 [==============================] - 32s 2ms/step - loss: 4.7693e-
04 - val_loss: 4.9103e-04
```

3) Poglavlje sa eksperimentalnim rezultatima

Vizuelizacija dobijenih rezultata:

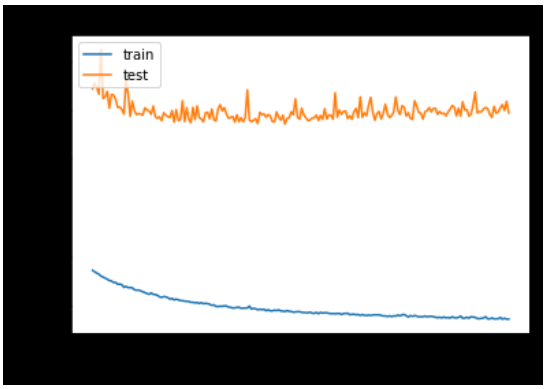


Grafik koji predstavlja kretanje funkcije greške kroz epohe.

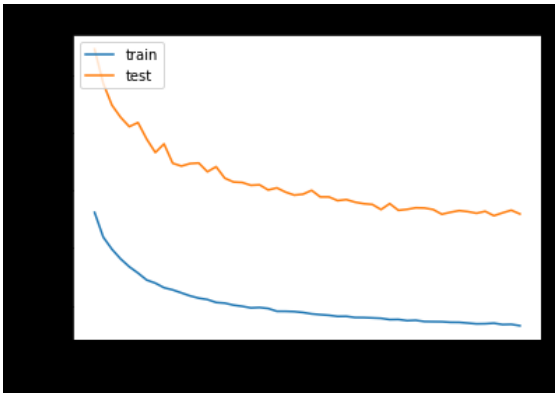


Pronašao sam rad identičan mom: http://flothesof.github.io/convnet-face-keypoint-detection.html

S obzirom da sam takođe koristio konvolutivne neuronske mreže za rešavanje ovog problem idealan je da uporedimo dobijene rezultate. Oni su prvo koristili jednostavnu arhitekturu mreže sa potpuno povezanim slojevima i dobili su loše rezulte.



Nakon toga su optimizovali svoju mrežu i dobili značajno bolje rezultate.



Prema grafikonima za funkciju greške ispada da sam ja dobio bolje rezultate, ali prilikom vizuelizacije njihovih i mojih test instanci vidimo da su oba rešenja prilično dobra i da naše mreže dobro pogađaju lokacije ključnih tačaka. Oni su nakon ovoga odradili i prilično zanimljivu stvar, naime na osnovu koordinata ključnih tačaka na licu su dodavali ljudima brkove.

Za kodiranje sam koristio programski jezik Python uz razne biblioteke(keras, numpy, matplotlib, ...).

Projekat sam razvijao uz pomoć google colab-a. Pokušao sam da pokrenem trening na sopstvenom računaru ali nije uspelo jer je previše zahtevan, zbog toga sam koristio colab. Vršio sam trening na njihovol virtualnoj mašini i na CPU-u i na GPU-u, kada se trenira na CPU-u trening jedne epohe traje oko 6 minuta, što dovodi do toga da treniranje celokupne neuronske mreže traje približno 5 sati. Kada se koristi GPU trening traje znatno kraće, oko 30 sekunti po epohi u proseku, pa sveukupno vreme za koje se mreža može istrenirati je oko 25 minuta. Nisam siguran koliko ram memorije su mi stavili na raspolaganje preko te virtualne mašine, ali znam da je za moj projekat potrebno oko 10,5gb.

4) Zaključak

Na kraju kada se podvuče crta zadovoljan sam naučenim i zadovoljan sam dobijenim rezultatima. Mislim da za rešavanje ovog problema postoji dosta efikasnije rešenje koje nisam uspeo da dosegnem jer je po meni previše vremena i memorije bilo potrebno da se istrenira ova mreža. Dalje unapređenje ovog projekta je moguce kroz razne stvari, na primer moze da se napravi aplikacija koja ce da dodaje neke efekte u realnom vremenu.

5) Literatura

1) https://www.etran.rs/common/pages/proceedings/ETRAN2017/VI/IcETRAN2017_paper_VI1_1.pdf
2) https://stackoverflow.com/questions/34902477/drawing-circles-on-image-with-matplotlib-and-numpy
3) https://keras.io/
4) https://arxiv.org/abs/1511.07289
5) http://flothesof.github.io/convnet-face-keypoint-detection.html