PREPOZNAVANJE KLJUČNIH TAČAKA NA LICU

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

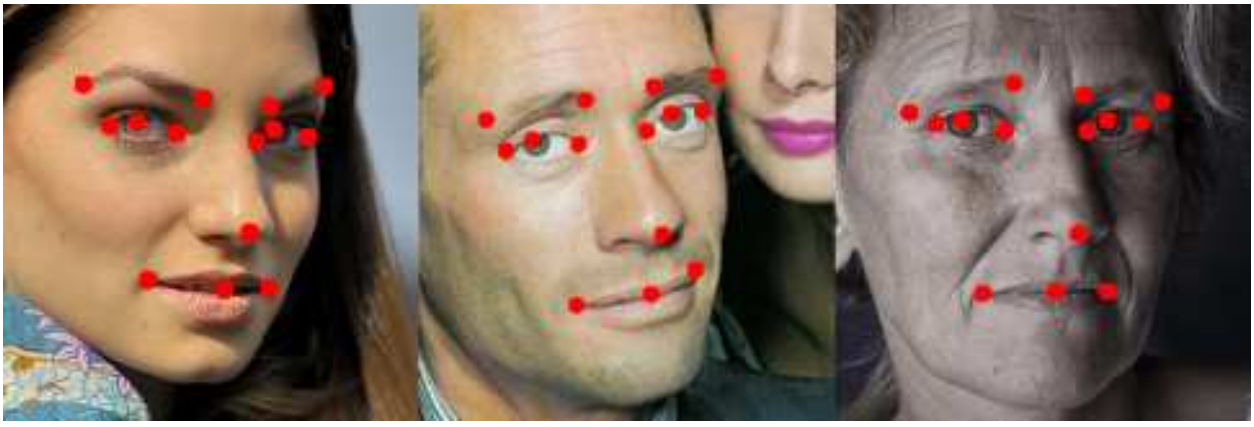
1) Uvod

Ovaj rad se bavi problemom prepoznavanja ključnih tačaka na licu ljudi koristeći neuronske mreže. Na samom početku treba da odaberemo tip neuronskih mreža koji ćemo koristiti. Nakon istraživanja dolazimo do zaključka da su konvolutivne neuronske mreže idealne za naš problem.

Konvolutivne neuronske mreže i sama njihova duboka struktura će nam omogućiti da sa velikom preciznošću lociramo ključne tačke na licima ljudi. U ovom radu ključne tačke su početak i kraj obrve, levi i desni kraj oka, zenica oka, vrh nosa, levi i desni kraj usne i sredina usta. Konvolutivne neuronske mreže su dobile ime po konvoluciji, operatoru koji se primenjuje u obradi slika i signala. Prednost konvolutivnih mreža u odnosu na ostale je velika kada se pogleda sa aspekta ovog rada. Na primer, kada se kao ulaz koristi slika dimenzije 200x200 piksela, ona se pretvara u 40.000 neurona na ulaznom sloju, što je jako veliki broj i praktično je nemoguće treniranje potpuno povezane mreže. Još ako je slika u boji treba uzeti u obzir i 3 kanala kojima se predstavljaju osnovne boje(RGB), što nas dovodi do 120.000 neurona na ulaznom sloju. Ideja konvolutivnih mreža je da postavi veći broj slojeva za otkrivanje bitnih osobina ulaznih podataka. Mana ovih mreža je ta što zahteva značajne hardverske resurse.

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

Primer podataka



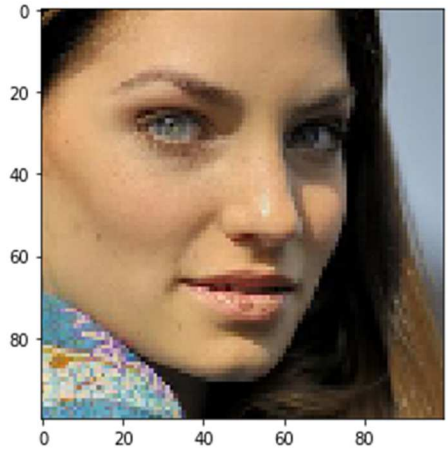
2) Opis rešenja

Pre svega za rešavanje ovog problema nam trebaju podaci. Oni se nalaze u folderu „data“ koji se sastoji iz foldera „images“ u kojem se nalazi 6000 slika lica, i csv fajla koji sadrži informacije o svakoj slici(ime slike i koordinate ključnih tačaka).

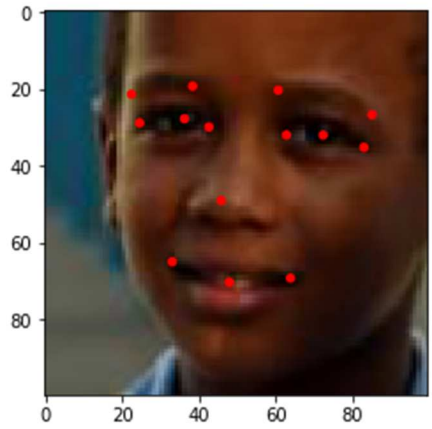
Nakon učitavanja podataka sledi njihovo pretprocesiranje. U našem slučaju to je skaliranje rezolucije slike na 100x100 piksela i skaliranje koordinata ključnih tačaka na interval [-0.5 : 0.5]. Ovo radimo zato što su slike koje imamo različitih rezolucija, te je potrebno da ih skaliramo na istu rezoluciju jer će naša mreža mnogo bolje da se snađe sa tako spremnim podacima, takođe skaliranje koordinata na pomenuti interval omogućava 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])
```

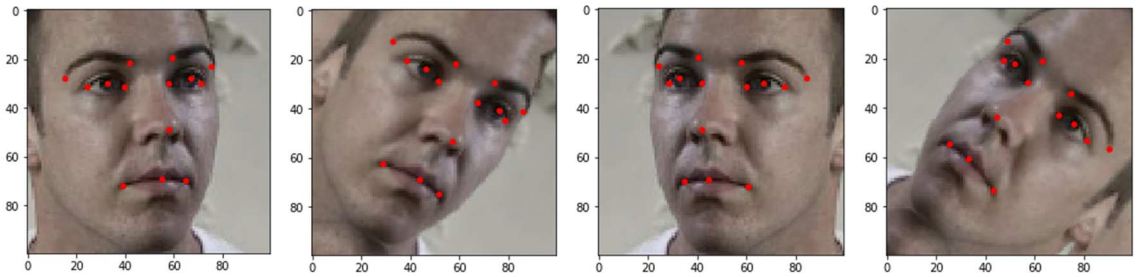


primer pretprocesiranog podatka:



Sledeći korak koji je potrebno da se uradi je povećanje broja podataka(augmentacija). To se radi sa ciljem veće preciznosti modela i boljeg lociranja tačaka. Što mreža ima više podataka za treniranje to će dati bolje rezultate.
U ovom radu je odrađeno preslikavanje slike kao odraza u ogledalu i rotacija slike za neki slučajan ugao. Prilikom svakog vida augmentacije treba voditi računa o validnosti novodobijenih podataka, pa je u našem slučaju veoma bitno da na augmentovanim podacima dobro pozicioniramo ključne tačke kako bi naša mreža mogla ispravno da se trenira.
U slučaju kada se vrši preslikavanje slike kao odraza u ogledalu, dobre pozicije ključnih tačaka su zadržane tako što se radi inverzovanje x koordinate slike(množenje sa -1). Kod rotacije za slučajan ugao ćemo originalne pozicije ključnih tačaka da pomnožimo matricom rotacije za dve dimenzije.
Na kraju smo od početnih 6000 slika dobili ispravnih 24.000 ulaznih podataka.

primer originalnog i augmentovanih podataka



Sada kada imamo spremne podatke, na red dolazi njihova podela na trening skup i skup za validaciju, dizajn same arhitekture neuronske mreže i treniranje modela.

arhitektura mreže

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
_________________________________________________________________
```

U ovom radu su korišćeni 2D konvolutivni slojevi zbog rada sa slikama, MaxPooling2D slojevi zbog redukcije dimenzionalnosti. Kada se ulaz sveo na 256 bitnih instanci korišćeni su potpuno povezani slojevi(Dense). Takođe je korišćen i Dropout postavljen na 0.25. Dropout je korišćen za regularizaciju. Ukratko poenta regularizacije je da dovede do bolje generalizacije mreže na slikama na kojima nije trenirana(dobijamo robusniju mrežu).

Sada na red dolazi samo treniranje mreže. Da u procesu učenja ne bih korisliti baš sve instance(full batch gradient descent), koristićemo mini_batch. Mini_batch je varijanta algoritma gradijentnog spusta. Ideja mini_batcha je ta da se slučajnom metodom izdvoje podskupovi podataka iz trening skupa, da se oni propuste kroz mrežu i da se koriste za računanje greške modela i ažuriranje koeficijenata.

Što je veći podskup, gradijent će biti precizniji, ali ako je podskup preveliki ili je u pitanju ceo trening skup, to može dovesti di problema sa memorijom jer je potrošnja memorije srazmerna veličini

podskupa. Može čak da predstavlja i ograničenje. U praksi se obično bira stepen dvojke za veličinu podskupa(najčešće od 32 do 256 instanci). Taj slučajni podskup obično dobro aproksimira ponašanje gradijenta na celom trening skupu.

https://machinelearningmastery.com/gentle-introduction-mini-batch-gradient-descent-configure-batch-size/

Treniranje mreže se obavlja kroz 50 epoha i prati se ponašanje funkcije greške na test skupu i skupu za validaciju. S obzirom da se radi sa realnim brojevima, kao funkciju greške koristimo srednje-kvadratnu grešku(mse).

trening

```
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
0.00010000000474974513.
Epoch 13/50
21600/21600 [==============================] - 33s 2ms/step - loss: 6.8394e-
04 - val_loss: 5.7431e-04
```

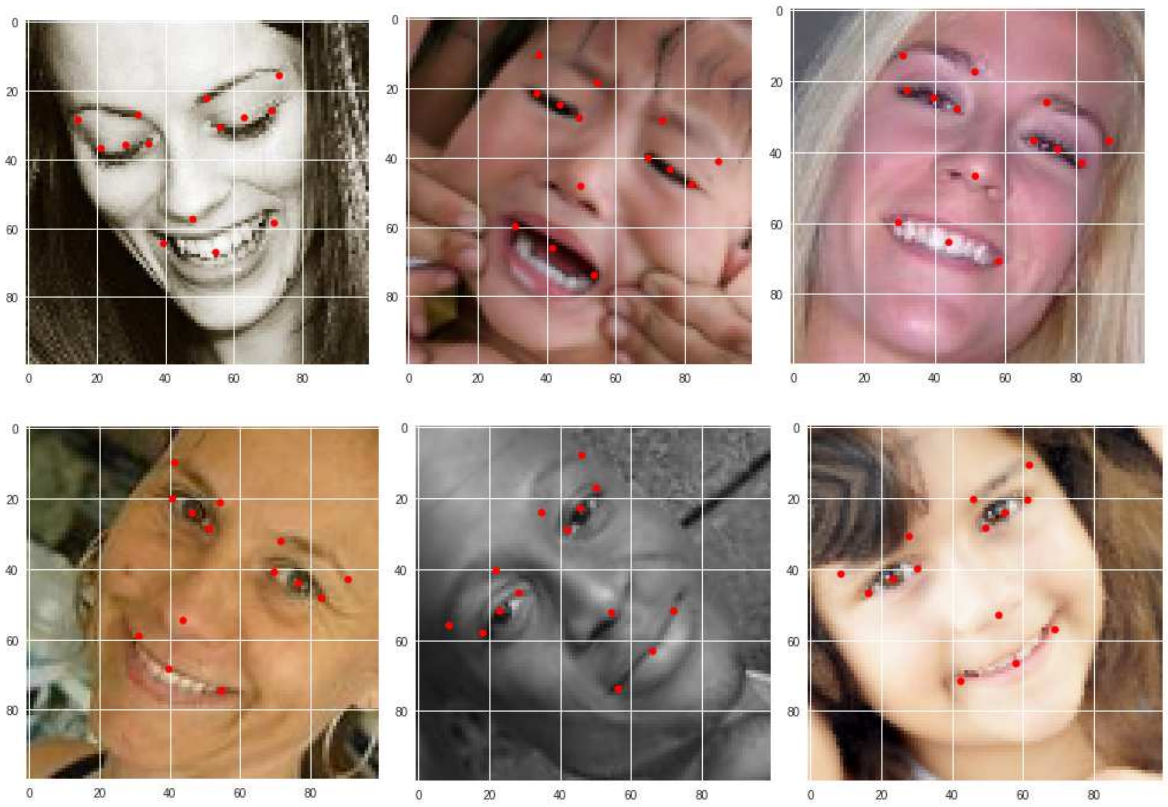
```
Epoch 14/50
21600/21600 [==============================] - 33s 2ms/step - loss: 6.0470e-
04 - val_loss: 5.4447e-04
Epoch 15/50
21600/21600 [==============================] - 33s 2ms/step - loss: 5.8031e-
04 - val_loss: 5.3681e-04
Epoch 16/50
21600/21600 [==============================] - 33s 2ms/step - 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
```

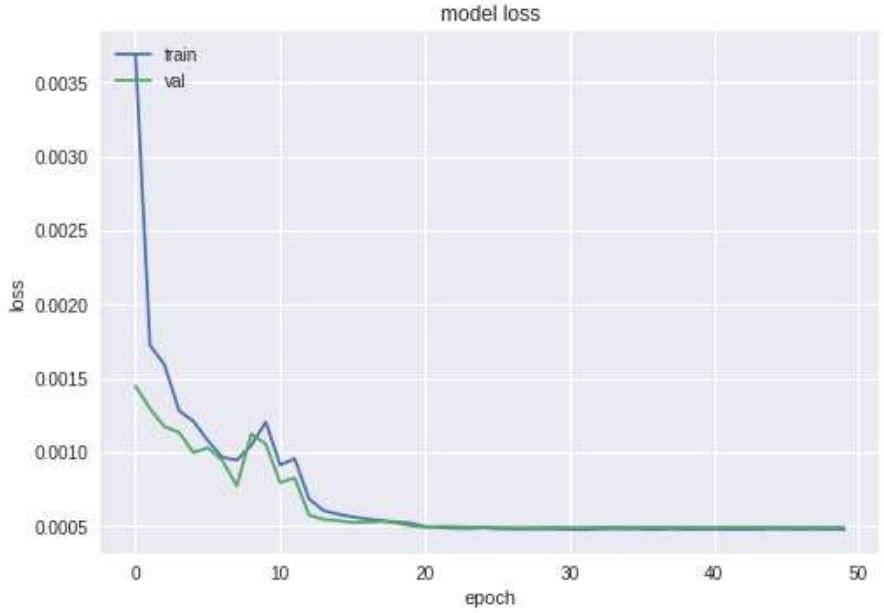
```
Epoch 47/50
21600/21600 [==============================] - 32s 1ms/step - loss: 4.7835e-
04 - val_loss: 4.9103e-04
Epoch 48/50
21600/21600 [==============================] - 32s 1ms/step - loss: 4.7962e-
04 - val_loss: 4.9103e-04

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:



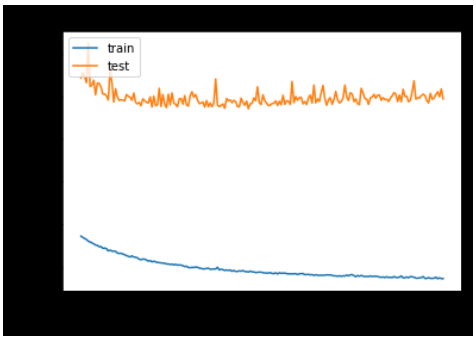
Grafik koji predstavlja kretanje funkcije greške kroz epohe.



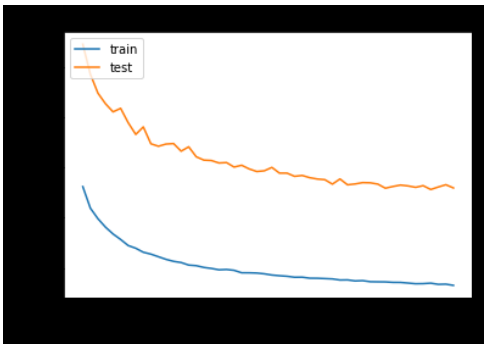
Na sledećem linku se nalazi rad identičan našem: http://flothesof.github.io/convnet-face-keypoint-detection.html

S obzirom da su u našem radu takođe korišćene konvolutivne neuronske mreže za rešavanje ovog problema, navedeni rad je idealan da se rezultati uporede.

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.



Kada uporedimo grafikone sa funkcijama greške uočavamo da su naši rezultati bolji, ali prilikom vizualizacije instanci iz skupa za validaciju vidimo da su oba rešenja prilično dobra i da obe mreže dobro pogađaju pozicije 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 u našem radu je korišćen programski jezik Python uz razne biblioteke(keras, numpy, matplotlib, ...).

Projekat je razvijan uz pomoć google colab-a zato što je previše zahtevan trening za običan računar. Treniranje se izvršavalo na Google-ovoj virtualnoj mašini specijalno namenjenoj za istraživanje. Uz pomoć nje smo dobili sve potrebne resurse za naš projekat. Postoji opcija biranja da li želimo da koristimo CPU ili GPU. Kada se koristi CPU 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 okvirno 25 minuta.

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 će 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
6) https://machinelearningmastery.com/gentle-introduction-mini-batch-gradient-descent-configure-batch-size/