PROJEKAT IZ PREDMETA OSNOVE RAČUNARSKE INTELIGENCIJE NA TEMU **PREPOZNAVANJE EMOCIJA PUTEM KONVOLUCIONE NEURONSKE MREŽE**

Profesor:

Vladimir Ivančević

Student:

Miroslav Kosanović IN39/2017

Novi Sad, 2020.

Sadržaj

Odabir arhitekture

Postoji mnogo načina za odabir CNN arhitekture. U ovom projektu pokretano je više eksperimenata kako bih uspeo da dobijem što tačniju i efikasniju arhitekturu. Počeli sam testiranje sa određenim vrednostima, a svako sledeće testiranje je bio i jedan novi eksperiment. U svakom eksperimentu sam menjali druge parametre, i analizirali kako se to odražava na preciznost klasifikacije emocija.

Učitavanje skupa

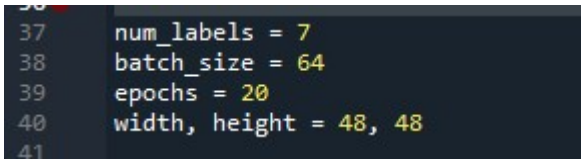
Na **slici 1.** Prikazano je učitavanje podataka iz CSV fajla.

Dataset koji sam koristio je FER2013, koji se sastoji od 34034 slika, od čega je 80% namenjeno za obuku, a 20% namenjeno testiranju.

```
df=pd.read_csv('fer2013.csv')

# print(df.info())
# print(df["Usage"].value_counts())

# print(df.head())
X_train,train_y,X_test,test_y=[],[],[],[]

for index, row in df.iterrows():
    val=row['pixels'].split(" ")
    try:
        if 'Training' in row['Usage']:
           X_train.append(np.array(val,'float32'))
           train_y.append(row['emotion'])
        elif 'PublicTest' in row['Usage']:
           X_test.append(np.array(val,'float32'))
           test_y.append(row['emotion'])
    except:
        print(f"error occured at index :{index} and row:{row}")

```

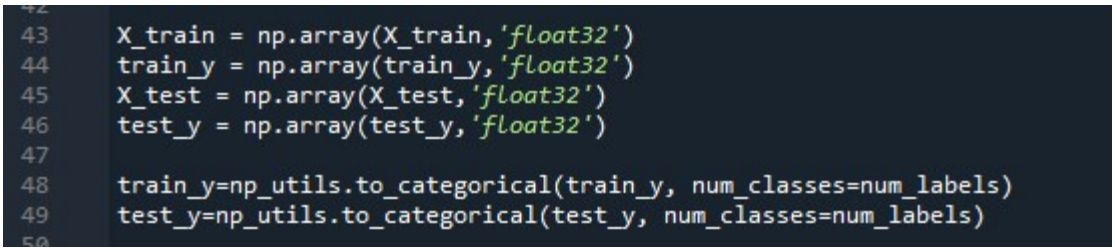
Slika 1.

Na **slici 2.** Prikazani su neki od parametara koje sam definisao, pri čemu ***num_labels*** predstavlja ukupan broj emocija (neutral, angry, happy, fear, surprise, disgust, sad), ***batch_size*** veličinu beča, ***epochs*** broj epoha, a ***width*** i ***height*** širinu i visinu slike.

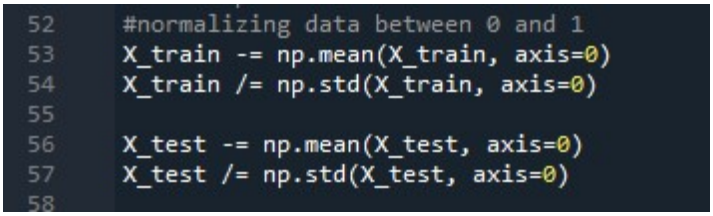


Slika 2.

Potom sam od listi ***X_train, train_y, X_test, test_y*** napravio numpy nizove (**Slika 3**), pošto biblioteka *Keras* koju sam koristio prihvata samo numpy niz kao ulaz. Tip podatka sam postavio na float32, kako bih kasnije lakše izvršio normalizaciju podataka od 0 do 1. Za normalizaciju sam koristio funkcije ***np.mean()*** za računanje aritmetičke sredine niza, i ***np.std()*** za računanje standardne devijacije (**Slika 4**).

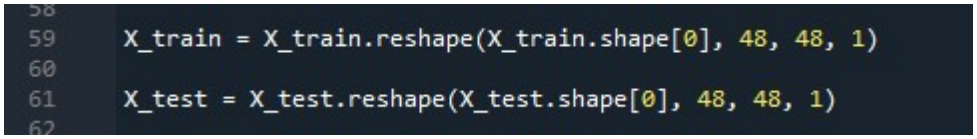


Slika 3.



Slika 4.

Nakon toga sam ulazni niz podelio u slike veličine 48x48, sa jednim kanalom (**Slika 5**)



Slika 5.

Prvo testiranje

Na **slici 6.** možemo videti parametre koje sam koristio prilikom prvog testiranja moje konvolucione neuronske mreže.

```
#1st convolution layer
model = Sequential()

model.add(Conv2D(64, kernel_size=(3, 3), activation='relu', input_shape=(X_train.shape[1:])))
model.add(Conv2D(64,kernel_size= (3, 3), activation='relu'))
# model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2), strides=(2, 2)))
model.add(Dropout(0.5))

#2nd convolution layer
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(Conv2D(64, (3, 3), activation='relu'))
# model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2), strides=(2, 2)))
model.add(Dropout(0.5))

#3rd convolution layer
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(Conv2D(128, (3, 3), activation='relu'))
# model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2), strides=(2, 2)))

model.add(Flatten())

#fully connected neural networks
model.add(Dense(1024, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(1024, activation='relu'))
model.add(Dropout(0.2))

model.add(Dense(num_labels, activation='softmax'))

# model.summary()
```

Slika 6.

Mrežu sam testirao kroz 20 epoha (**Slika 7**).

```
Epoch 20/20
449/449 - 343s - loss: 0.9775 - accuracy: 0.6260 - val_loss: 1.1480 - val_accuracy: 0.5734
```

Slika 7.

Kao što se može primetiti na **Slici 7.** preciznost prepoznavanja emocija validacionog skupa iznosi 57%, zbog čega sam odlučio da izmenim neke parametre, kako bih postigao veću preciznost.

Drugo testiranje

Prilikom drugog testiranja (**Slika 8**), želeo sam da vidimo da li je najbolje da koristim 1,2,3 ili 4 konvoluciona sloja. Nisam pokušavao sa 5 slojeva jer bi tada slika bila premala. Pošto su slike veličine 48x48, nakon prve konvolucije bile bi 24x24, nakon druge 12x12, nakon treće 6x6 i nakon četvrte 3x3, tako da nema smisla da dodajem i peti sloj.

```
nets = 4
model = [0]*nets

for i in range(4):
    model[i] = Sequential()

    model[i].add(Conv2D(24, kernel_size=(3, 3), activation='relu', input_shape=(X_train.shape[1:])))
    # model.add(BatchNormalization())
    model[i].add(MaxPooling2D(pool_size=(2,2),padding='same', strides=(2, 2)))
    model[i].add(Dropout(0.5))

    if i>0:
#2nd convolution layer
        model[i].add(Conv2D(48, (3, 3), activation='relu'))
        model[i].add(MaxPooling2D(pool_size=(2,2),padding='same', strides=(2, 2)))
        model[i].add(Dropout(0.5))

    if i>1:
#3rd convolution layer
        model[i].add(Conv2D(64, (3, 3), activation='relu'))
        model[i].add(MaxPooling2D(pool_size=(2,2),padding='same', strides=(2, 2)))

#4th convolution layer
    if i>2:
        model[i].add(Conv2D(128, (3, 3), activation='relu'))
        model[i].add(MaxPooling2D(pool_size=(2,2),padding='same', strides=(2, 2)))

    model[i].add(Flatten())

#fully connected neural networks
    model[i].add(Dense(1024, activation='relu'))
    model[i].add(Dropout(0.2))
    model[i].add(Dense(1024, activation='relu'))
    model[i].add(Dropout(0.2))

    model[i].add(Dense(num_labels, activation='softmax'))
```

Slika 8.

```
Epoch 10/10
449/449 - 140s 312ms/step - loss: 0.8888 - accuracy: 0.6680 - val_loss: 1.3780 - val_accuracy: 0.5007
CNN (C-P)x1: Epochs=10, Train accuracy=0.66805, Validation accuracy=0.50488

Epoch 10/10
449/449 - 103s 230ms/step - loss: 1.0108 - accuracy: 0.6212 - val_loss: 1.1547 - val_accuracy: 0.5589
CNN (C-P)x2: Epochs=10, Train accuracy=0.62116, Validation accuracy=0.55893

Epoch 10/10
449/449 - 99s 221ms/step - loss: 1.0220 - accuracy: 0.6101 - val_loss: 1.1677 - val_accuracy: 0.5603
CNN (C-P)x3: Epochs=10, Train accuracy=0.61009, Validation accuracy=0.56673

Epoch 10/10
449/449 - 91s 203ms/step - loss: 1.1681 - accuracy: 0.5552 - val_loss: 1.1894 - val_accuracy: 0.5444
CNN (C-P)x4: Epochs=10, Train accuracy=0.55516, Validation accuracy=0.54444
```

Slika 9.

Drugo testiranje sam izvršio kroz 10 epoha i, kao što se može primetiti na **Slici 9**, najveću preciznost imam kada koristim 3 konvoluciona sloja, pa ću u daljim testiranjima i ostaviti samo ta 3 sloja.

Treće testiranje

Nakon određivanja broja konvolucionih slojeva koji su mi potrebni, odlučio sam da sledeći parametar koji ću izmeniti bude broj kernela. Kroz svaku iteraciju sam povećavo taj broj (**Slika 10**)

```
nets = 6
model = [0]*nets

for i in range(6):
    model[i] = Sequential()

    model[i].add(Conv2D(i*8+8, kernel_size=(3, 3), activation='relu', input_shape=(X_train.shape[1:])))
    # model.add(BatchNormalization())
    model[i].add(MaxPooling2D(pool_size=(2,2),padding='same', strides=(2, 2)))
    model[i].add(Dropout(0.5))

#2nd convolution layer
    model[i].add(Conv2D(i*16+16, (3, 3), activation='relu'))
    model[i].add(MaxPooling2D(pool_size=(2,2),padding='same', strides=(2, 2)))
    model[i].add(Dropout(0.5))

#3rd convolution layer
    model[i].add(Conv2D(i*24+24, (3, 3), activation='relu'))
    model[i].add(MaxPooling2D(pool_size=(2,2),padding='same', strides=(2, 2)))

    model[i].add(Flatten())

#fully connected neural networks
    model[i].add(Dense(1024, activation='relu'))
    model[i].add(Dropout(0.2))
    model[i].add(Dense(1024, activation='relu'))
    model[i].add(Dropout(0.2))

    model[i].add(Dense(num_labels, activation='softmax'))

```

Slika 10.

Na **Slici 11** primećujem da najveću preciznost dobijam kada se u prvom konvolucionom sloju nalazi 64 kernela.

```
Epoch 10/10
449/449 - 65s 145ms/step - loss: 1.2614 - accuracy: 0.5173 - val_loss: 1.2752 - val_accuracy: 0.5082
CNN 8 maps: Epochs=10, Train accuracy=0.51729, Validation accuracy=0.51379

Epoch 10/10
449/449 - 68s 152ms/step - loss: 1.1302 - accuracy: 0.5695 - val_loss: 1.1976 - val_accuracy: 0.5481
CNN 16 maps: Epochs=10, Train accuracy=0.56947, Validation accuracy=0.54806

Epoch 10/10
449/449 - 163s 364ms/step - loss: 1.0343 - accuracy: 0.6059 - val_loss: 1.1733 - val_accuracy: 0.5545
CNN 24 maps: Epochs=10, Train accuracy=0.60587, Validation accuracy=0.55698

Epoch 10/10
449/449 - 144s 322ms/step - loss: 0.9720 - accuracy: 0.6327 - val_loss: 1.1836 - val_accuracy: 0.5653
CNN 32 maps: Epochs=10, Train accuracy=0.63269, Validation accuracy=0.57370

Epoch 10/10
449/449 - 171s 382ms/step - loss: 0.9432 - accuracy: 0.6439 - val_loss: 1.1311 - val_accuracy: 0.5773
CNN 48 maps: Epochs=10, Train accuracy=0.64387, Validation accuracy=0.57732

Epoch 10/10
449/449 - 209s 464ms/step - loss: 0.9151 - accuracy: 0.6563 - val_loss: 1.1744 - val_accuracy: 0.5665
CNN 64 maps: Epochs=10, Train accuracy=0.65628, Validation accuracy=0.57871
```

Slika 11.

Četvrto testiranje

Ovde ću da videti koliko je potrebno gusto povezanih jedinica u Dense() sloju.

```
nets = 8
model = [0]*nets

for i in range(8):
    model[i] = Sequential()

    model[i].add(Conv2D(64, kernel_size=(3, 3), activation='relu', input_shape=(X_train.shape[1:])))
    # model.add(BatchNormalization())
    model[i].add(MaxPooling2D(pool_size=(2,2),padding='same', strides=(2, 2)))
    model[i].add(Dropout(0.5))

#2nd convolution layer
    model[i].add(Conv2D(96, (3, 3), activation='relu'))
    model[i].add(MaxPooling2D(pool_size=(2,2),padding='same', strides=(2, 2)))
    model[i].add(Dropout(0.5))

#3rd convolution layer
    model[i].add(Conv2D(144, (3, 3), activation='relu'))
    model[i].add(MaxPooling2D(pool_size=(2,2),padding='same', strides=(2, 2)))

    model[i].add(Flatten())

#fully connected neural networks
    if i > 0:
        model[i].add(Dense(2**(i+4), activation='relu'))
        model[i].add(Dropout(0.2))
        model[i].add(Dense(2**(i+4), activation='relu'))
        model[i].add(Dropout(0.2))

    model[i].add(Dense(num_labels, activation='softmax'))

```

Slika 12.

Na **Slici 13.** vidim da je najbolja opcija sa 256 jedinica. Slojevi sa 1024 i 2048 jedinica imaju samo nešto bolji učinak, ali to nije vredno dodatnih računskih troškova.

```
Epoch 10/10
449/449 - 207s 461ms/step - loss: 1.0644 - accuracy: 0.6024 - val_loss: 1.1453 - val_accuracy: 0.5715
CNN 0N: Epochs=10, Train accuracy=0.60242, Validation accuracy=0.57147

Epoch 10/10
449/449 - 204s 453ms/step - loss: 1.2931 - accuracy: 0.4971 - val_loss: 1.2255 - val_accuracy: 0.5288
CNN 32N: Epochs=10, Train accuracy=0.49706, Validation accuracy=0.52884

Epoch 10/10
449/449 - 207s 460ms/step - loss: 1.1840 - accuracy: 0.5502 - val_loss: 1.1644 - val_accuracy: 0.5458
CNN 64N: Epochs=10, Train accuracy=0.55021, Validation accuracy=0.55447

Epoch 10/10
449/449 - 208s 464ms/step - loss: 1.1296 - accuracy: 0.5679 - val_loss: 1.1431 - val_accuracy: 0.5628
CNN 128N: Epochs=10, Train accuracy=0.56787, Validation accuracy=0.56283

Epoch 10/10
449/449 - 232s 516ms/step - loss: 1.0240 - accuracy: 0.6152 - val_loss: 1.1114 - val_accuracy: 0.5848
CNN 256N: Epochs=10, Train accuracy=0.61524, Validation accuracy=0.58484

Epoch 10/10
449/449 - 237s 529ms/step - loss: 0.9337 - accuracy: 0.6472 - val_loss: 1.1343 - val_accuracy: 0.5695
CNN 512N: Epochs=10, Train accuracy=0.64718, Validation accuracy=0.57091

Epoch 10/10
449/449 - 266s 592ms/step - loss: 0.9143 - accuracy: 0.6552 - val_loss: 1.1434 - val_accuracy: 0.5857
CNN 1024N: Epochs=10, Train accuracy=0.65520, Validation accuracy=0.58568

Epoch 10/10
449/449 - 321s 715ms/step - loss: 0.8382 - accuracy: 0.6851 - val_loss: 1.2039 - val_accuracy: 0.5860
CNN 2048N: Epochs=10, Train accuracy=0.68512, Validation accuracy=0.58596
```

Slika 13.

Peto testiranje

U ovom eksperimentu menjao sam broj dropout-a (**Slika 14**)

```
nets = 8
model = [0]*nets

for i in range(8):
    model[i] = Sequential()

    model[i].add(Conv2D(64, kernel_size=(3, 3), activation='relu', input_shape=(X_train.shape[1:])))
    # model.add(BatchNormalization())
    model[i].add(MaxPooling2D(pool_size=(2,2),padding='same', strides=(2, 2)))
    model[i].add(Dropout(i*0.1))

#2nd convolution layer
    model[i].add(Conv2D(96, (3, 3), activation='relu'))
    model[i].add(MaxPooling2D(pool_size=(2,2),padding='same', strides=(2, 2)))
    model[i].add(Dropout(i*0.1))

#3rd convolution layer
    model[i].add(Conv2D(144, (3, 3), activation='relu'))
    model[i].add(MaxPooling2D(pool_size=(2,2),padding='same', strides=(2, 2)))

    model[i].add(Flatten())

#fully connected neural networks
    model[i].add(Dense(256), activation='relu')
    model[i].add(Dropout(i*0.1))
    model[i].add(Dense(256), activation='relu')
    model[i].add(Dropout(i*0.1))

    model[i].add(Dense(num_labels, activation='softmax'))
```

Slika 14.

Za dalje testiranje ostavio sam vrednost 0.2, jer sa njom imao najbolju preciznost (**Slika 15**).

```
Epoch 10/10
449/449 - 203s 451ms/step - loss: 0.2393 - accuracy: 0.9161 - val_loss: 2.2416 - val_accuracy: 0.5503
CNN D=0: Epochs=10, Train accuracy=0.91609, Validation accuracy=0.56980

Epoch 10/10
449/449 - 203s 451ms/step - loss: 0.4901 - accuracy: 0.8198 - val_loss: 1.4782 - val_accuracy: 0.5851
CNN D=0.1: Epochs=10, Train accuracy=0.81978, Validation accuracy=0.58624

Epoch 10/10
449/449 - 194s 432ms/step - loss: 0.6828 - accuracy: 0.7470 - val_loss: 1.2295 - val_accuracy: 0.5913
CNN D=0.2: Epochs=10, Train accuracy=0.74701, Validation accuracy=0.59320

Epoch 10/10
449/449 - 201s 447ms/step - loss: 0.8921 - accuracy: 0.6656 - val_loss: 1.1392 - val_accuracy: 0.5815
CNN D=0.3: Epochs=10, Train accuracy=0.66558, Validation accuracy=0.58484

Epoch 10/10
449/449 - 282s 628ms/step - loss: 1.0629 - accuracy: 0.5989 - val_loss: 1.1404 - val_accuracy: 0.5684
CNN D=0.4: Epochs=10, Train accuracy=0.59891, Validation accuracy=0.56980

Epoch 10/10
449/449 - 239s 531ms/step - loss: 1.2087 - accuracy: 0.5429 - val_loss: 1.1614 - val_accuracy: 0.5550
CNN D=0.5: Epochs=10, Train accuracy=0.54290, Validation accuracy=0.55670

Epoch 10/10
449/449 - 266s 592ms/step - loss: 1.3517 - accuracy: 0.4849 - val_loss: 1.2602 - val_accuracy: 0.5169
CNN D=0.6: Epochs=10, Train accuracy=0.48494, Validation accuracy=0.51686

Epoch 10/10
449/449 - 310s 690ms/step - loss: 1.4967 - accuracy: 0.4178 - val_loss: 1.4496 - val_accuracy: 0.4595
CNN D=0.7: Epochs=10, Train accuracy=0.41785, Validation accuracy=0.45946
```

Slika 15.

Šesto testiranje

Na **Slici 16.** možemo videti kako izgleda mreža nakon svih izmena. Testirao sam je kroz 20 epoha. Dobijeni rezultat je prikazan na **Slici 17**, i ako ga uporedimo sa rezultatom na **Slici 7**, primećujem da sada imam veću preciznost.

Nakon ovoga uradio sam i poslednje testiranje, gde sam ubacio beč normalizaciju (BatchNormalization()), **Slika 18**. Ovde sam istrenirao mrežu kroz 40 epoha (**Slika 19**).

```
#1st convolution layer

model = Sequential()

model.add(Conv2D(64, kernel_size=(3, 3), activation='relu', input_shape=(X_train.shape[1:])))
#model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2),padding='same', strides=(2, 2)))
model.add(Dropout(0.2))

#2nd convolution layer
model.add(Conv2D(96, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2),padding='same', strides=(2, 2)))
model.add(Dropout(0.2))

#3rd convolution layer
model.add(Conv2D(144, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2),padding='same', strides=(2, 2)))

model.add(Flatten())

#fully connected neural networks
model.add(Dense(256), activation='relu')
model.add(Dropout(0.2))
model.add(Dense(256), activation='relu')
model.add(Dropout(0.2))

model.add(Dense(num_labels, activation='softmax'))
```

Slika 16.

```
Epoch 20/20
449/449 - 209s 466ms/step - loss: 0.3686 - accuracy: 0.8665 - val_loss: 1.6966 - val_accuracy: 0.5926
```

Slika 17.

```
61  #1st convolution layer
62
63  model = Sequential()
64
65  model.add(Conv2D(64, kernel_size=(3, 3), activation='relu', input_shape=(X_train.shape[1:])))
66  model.add(BatchNormalization())
67  model.add(MaxPooling2D(pool_size=(2,2),padding='same', strides=(2, 2)))
68  model.add(Dropout(0.2))
69
70  #2nd convolution layer
71  model.add(Conv2D(96, (3, 3), activation='relu'))
72  model.add(BatchNormalization())
73  model.add(MaxPooling2D(pool_size=(2,2),padding='same', strides=(2, 2)))
74  model.add(Dropout(0.2))
75
76  #3rd convolution layer
77  model.add(Conv2D(144, (3, 3), activation='relu'))
78  model.add(BatchNormalization())
79  model.add(MaxPooling2D(pool_size=(2,2),padding='same', strides=(2, 2)))
80
81  model.add(Flatten())
82
83  #fully connected neural networks
84  model.add(Dense(256, activation='relu'))
85  model.add(BatchNormalization())
86  model.add(Dropout(0.2))
87  model.add(Dense(256, activation='relu'))
88  model.add(BatchNormalization())
89  model.add(Dropout(0.2))
90
91  model.add(Dense(num_labels, activation='softmax'))
92
```

Slika 18.

```
Epoch 40/40
449/449 - 427s 951ms/step - loss: 0.1315 - accuracy: 0.9550 - val_loss: 2.1388 - val_accuracy: 0.5926
```

Slika 19.

Kao što primećujem na **Slici 19** val_accuracy posle 40 epoha, je isti kao i bez beč normalizacije posle 20 epoha (**Slika 17**).

Nakon izvršenih svih testiranja, zaključio sam da je najbolji rezultat posle prvog testiranja, jer tada nisam imao velike naznake overfitting-a, i s obzirom na to accuracy je bio najveći.

Videotester.py

Videotester.py predstavlja fajl čijom se upotrebom testiraju funkcije u faceRecognition.py fajlu. Testriranje se oslanja na Computer vision i njene OpenCV Haar kaskadne klasifikatore. Ovi klasifikatori su razvijeni na mašinskom učenju u kojem se kaskadna funkcija trenira iz skupa slika koje odgovovaraju i ne odgovaraju traženom pojmu. Na osnovu tog treninga se koristi u prepoznavanju objekata na drugim slikama.

Klasifikatori su izdeljeni u xml. fajlove i svaki fajl je posvećen prepoznavanju određenog objekta (prepoznavanje oka, dlanova, u našem slučaju lica). U ovom slučaju učitavam kaskadnu funkciju za prepoznavanje lica. U funkciji se slike skaliraju na manje pretvaraju u crno-bele radi lakšeg procesiranja jednog kanala vrednosti boje. Kada je ovo urađeno vrši se lociranje određenih karakteristika lica (ugrađenom funkcijom **detectMultiScale**) korišćenjem poznatih osobina iz **face_classifier** objekta. Parametri koji će biti prosleđeni ovoj funkciji jesu: vrednost sive boje, parametar koji određuje koliko je slika skalirana (standardna vrednost je 1.05 tj. 5%, postavio sam 1.32 da bih ubrzalo vreme izvršavanja), parametar koji govori koliko susednih pravougaonika treba svaki pravougaonik da ima (utiče na kvalitet analizirane slike, veća vrednost utiče na manje uočenih lica ali radi sa većom sigurnošću, dobre vrednosti za ovaj parametar su između 3 i 6).

```
face_haar_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
```

```
faces_detected = face_haar_cascade.detectMultiScale(gray_img, 1.32, 5)
```

Nakon upotrebe navedene funckije dobijamo: x koordinatu, y koordinatu, širinu i visinu pronađene karakteristike lica. Na osnovu ovih vrednosti formiramo pravougaonik poznat kao bounding box koji sadrži sve karakteristike koje sam pronašao. X i y koordinata predstavljaju početak bounding box-a pa ćemo dijagonalnu tačku pravougaonika (u odnosu na (x,y) tačku) pronaći sabiranjem sa vraćenim vrednostima.

```
for (x,y,w,h) in faces_detected:
    cv2.rectangle(test_img,(x,y),(x+w,y+h),(255,0,0),thickness=7)
    roi_gray=gray_img[y:y+w,x:x+h]#odsecanje regiona koji je bitan po nas(lice)
    roi_gray=cv2.resize(roi_gray,(48,48))
    img_pixels = image.img_to_array(roi_gray)
    img_pixels = np.expand_dims(img_pixels, axis = 0)
    img_pixels /= 255
```

Kao što se vidi po imenu promenljive u kojoj čuvamo rezultat haar funkcije, **detectMultiScale** nam vraća veći broj lica i svako lice nameravam obraditi.

Nakon ovoga sledi finalni korak u primeni modela inicijalizovanog na osnovu fer2013 json fajla.

Koristim **predict** funkciju nad modelom i prosleđujem obrađene fotografije kao parametar. Ova funkcija će pokušati da za svaku sliku vrati verovatnoću svake emocije (emocije su ugrađene u model inače bih moralo paziti o obliku tj. dužini svih nizova). Pronalazim za svaku obrađenu fotografiju najveću vrednost verovatnoće emocije i nju vraćamo kao predviđenu emociju.

```
        predictions = model.predict(img_pixels)

        #find max indexed array
        max_index = np.argmax(predictions[0])

        emotions = ('angry', 'disgust', 'fear', 'happy', 'sad', 'surprise', 'neutral')
        predicted_emotion = emotions[max_index]

        cv2.putText(test_img, predicted_emotion, (int(x), int(y)), cv2.FONT_HERSHEY_SIMPLEX, 1, (0,0,255)

    resized_img = cv2.resize(test_img, (1000, 700))
    cv2.imshow('Facial emotion analysis ',resized_img)
```

Ostatak koda je namenjen prikazu slike, bounding box-a i pronađene emocije u tekstualnom zapisu prikazanom iznad pronađenog lica.

Uputstvo za pokretanje programskog koda

Kod sam pokretao u Spyder (Anaconda).

U fajlu ***emotion_recognition.py*** se nalazi programski kod koji je namenjen za treniranje mreže, dok je za testiranje potrebno pokrenuti fajl ***videoTester.py***, nakon čega će se uključiti kamera.