



Prolećni semestar, 2021/22

PREDMET:

CS374 - Veštačka inteligencija

Aplikacija za prepoznavanje emocija

Projektni zadatak

Asistent:	Lazar Mrkela
Student:	Aleksa Cekić
Indeks:	4173

Niš, 2022

Uvod	2
CNN	3
Ulazni sloj	3
Skriveni sloj	4
Izlazni sloj	4
Izrada aplikacije za prepoznavanje emocije	4
Prikupljanje podataka	4
Augmentacija slike za detektovanje emocija na licu	7
Kreiranje modela korišćenjem CNN-a	7
Kompajlovanje modela	9
Treniranje podataka	10
Testiranje podataka	10
Diskusija o poboljšanju tačnosti	15
Zaključak	16
Literatura	17

Uvod

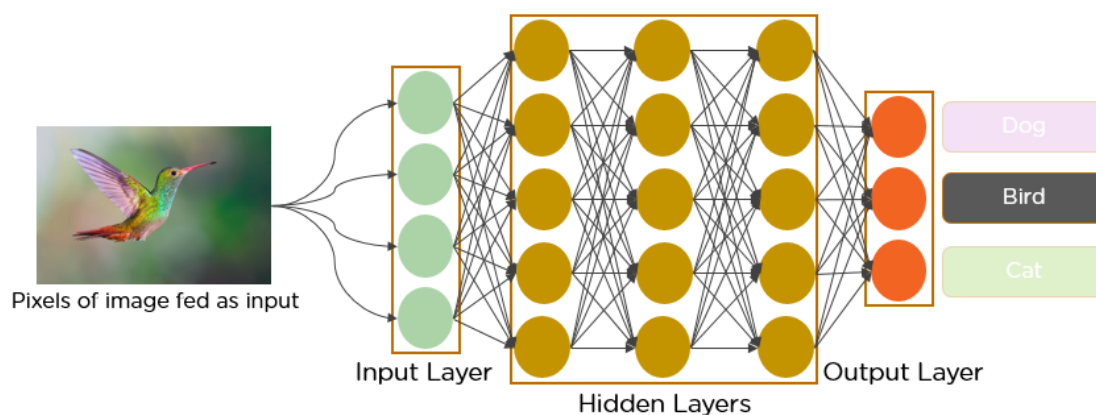
Ova aplikacija je urađena korišćenjem CNN (Convolutional neural network) algoritama koji je baziran na neuronskim mrežama. U machine learning-u, ili tačnije deep learning-u, CNN je algoritam koji predstavlja tip classifera koji proizvodi oznaku klase (npr. ptica, avion) za neke objekte koji postoje unutar neke slike, pri čemu se taj algoritam odlikuje u rešavanju raznih problema. Aplikacija funkcioniše tako što su podaci istrenirani da prepoznaju emociju koju osoba pokazuje preko web kamere. Više o algoritmu i aplikaciji možete naći u dokumentaciji.

CNN

CNN predstavlja algoritam koji je baziran na biološkim neuronskim mrežama. Ovaj algoritam se koristi u prepoznavanju pattern-a u radnji sa podacima. Neuronske mreže generalno su prepoznatljive po tome što su organizovane po slojevima, svaka sa svojim težinama i biasima.

Tipična neuralna mreža ima od nekolicine do stotinu, hiljadu, ili čak milion veštačkih neurona ili jedinica, umreženih u serije slojeva, gde je svaki neuron povezan sa oba sloja sa obe strane. Slojevi mogu da se podele na:

- Ulazni sloj
- Skriveni sloj
- Izlazni sloj



Slika 1 - Prikaz CNN mreže

Ulazni sloj

Početni sloj, tj. ulazni sloj, predstavlja zapravo podatke koji dolaze na ulaz mreže. U slučaju ovog projekta oni predstavljaju podatke slika sa kojima ćemo raditi. Zbog toga što koristimo RGB slike kao ulaz, ulazni sloj ima tri kanala koji odgovaraju red, green i blue kanalima.

Skriveni sloj

Skriveni sloj, tj. konvolucioni sloj predstavlja osnovu slojeva CNN-a, gde se sadrži veliki broj kernela (težina), koji izdvajaju karakteristike koje razlikuju sliku jedna od druge. Oni su povezani linkovima i svaki link predstavlja jedinstvenu težinu, koje koristi za operaciju konvolucije da bi se proizveo izlaz ili mapa aktivacije trenutnog konvolucionog neurona.

Izlazni sloj

Izlazni sloj signalizira način na koji mreža reaguje na naučene informacije. To može biti predstavljeno raznim načinima npr. da se prikazuje neka oznaka koju opisuje sliku.

Tok informacija kroz neuronsku mrežu odvija se na dva načina. Prilikom treniranja podataka ili radi u uobičajnom režimu tj. nakon što se istrenira. Više o ovom algoritmu možete pročitati na [linku](#).

Izrada aplikacije za prepoznavanje emocije

Sada kada smo opisali osnove kako CNN funkcioniše, možemo da se snađemo sa izradom ovog projektnog zadatka malo bolje. Tema ovog projektnog zadatka jeste sama izrada aplikacije za prepoznavanje emocija koristeći konvolucionih neuronski mreža.

Prikupljanje podataka

Pre nego da počnemo sa izradom moramo da iskoristimo set podataka na čemu možemo da istreniramo program da prepozna je emocije. Koristićemo dataset [fer-2013](#) koji možemo javno da preuzmemo sa Kaggle. Ovaj set podataka sadrži 7 emocija:

1. Angry
2. Disguist
3. Fear

4. Happy
5. Sad
6. Surprise
7. Neutral

Pre svega moramo da importujemo neke neophodne biblioteke koje ćemo koristiti. Biblioteke koje su nam neophodne za rad su:

- Numpy
- Tensorflow
- OpenCV
- Pandas

Sve ove biblioteke možete instalirati komandom:

pip install -r requirements.txt

Prvo podatke isčitavamo i čuvamo u df promenljivoj odakle možemo da reshapujemo podatke.

```
from keras_preprocessing.image import ImageDataGenerator
from keras.models import Model
from keras.layers import Input, Dense, Flatten, Dropout, BatchNormalization
from keras.layers.convolutional import Conv2D
from keras.layers.pooling import MaxPooling2D
from keras.optimizers import Adam

df = pd.read_csv('../input/fer2013/fer2013/fer2013.csv')
df.head()

# Preprocessing
x_train = []
y_train = []
x_test = []
y_test = []
```

Slika 2 - Prikaz biblioteka koje su potrebne za treniranje podataka

Promenljive `x_test`, `x_train` sadrže piksele dok `x_train`, `y_train` sadrže emocije. S obzirom na to da `x_test` i `x_train` sadrže broj pixdela u formi stringa, moramo da castujemo podatke u unsinged integere.

```
# Preprocessing
x_train = []
y_train = []
x_test = []
y_test = []

for index, row in df.iterrows():
    k = row['pixels'].split(" ")
    if row['Usage'] == 'Training':
        x_train.append(np.array(k))
        y_train.append(row['emotion'])
    elif row['Usage'] == 'PublicTest':
        x_test.append(np.array(k))
        y_test.append(row['emotion'])

x_train = np.array(x_train, dtype='uint8')
y_train = np.array(y_train, dtype='uint8')
x_test = np.array(x_test, dtype='uint8')
y_test = np.array(y_test, dtype='uint8')
```

Slika 3 - Prikaz filtriranja podataka na emocije i na pixele

Promenljive `y_test` i `y_train` sadrže jednodimenzionalne kodirane podatke, koje moramo da povežemo u kategorične podatke za efektivniji trening.

```
y_train = to_categorical(y_train, num_classes=7)
y_test = to_categorical(y_test, num_classes=7)

x_train = x_train.reshape((x_train.shape[0], 48, 48, 1))
x_test = x_test.reshape((x_test.shape[0], 48, 48, 1))
```

Slika 4 - Prikaz reshapovanja podataka

Augmentacija slike za detektovanje emocija na licu

Augmentacija slike može da se učini korišćenjem ImageDataGenerator koja se nalazi unutar Keras biblioteke. Augmentaciju koristimo radi poboljšanja performanse modela koji se generiše.

```
# Image Augmentation for FER
datagen = ImageDataGenerator(
    rescale=1. / 255,
    rotation_range=10,
    horizontal_flip=True,
    width_shift_range=0.1,
    height_shift_range=0.1,
    fill_mode='nearest'
)

testgen = ImageDataGenerator(rescale=1. / 255)

datagen.fit(x_train)

batch_size = 64
```

Slika 5 - Prikaz augmentacije slike

rescale: normalizuje vrednost pixela tako što ga deli sa 255.

horizontal_flip: okreće sliku horizontalno

fill_mode: popunjava sliku ako nije dostupna nakon kropiranja

rotation_range: rotira sliku 0-90 stepeni.

Podaci će biti generisani u veličini 64 (batch_size).

Korišćenjem data generatora je najbolji način da se trenira veliki broj podataka.

Kreiranje modela korišćenjem CNN-a

Podaci će biti generisani u veličini 64 (batch_size).

Korišćenjem data generatora je najbolji način da se trenira veliki broj podataka.

Dezajniramo CNN model za detekciju emocija korišćenjem API. Kreiramo blokove tako što koristimo Conv2D slojeve, Batch-Normalization, Max-Pooling2D, Dropout, Flatten, i kada ih spojimo zajedno na kraju koristimo Dense Layer za izlazni sloj.

Kreiramo `fer_model` koji kao parametar prima ulazni sloj tj. veličinu ulaznog sloja a vraća model koji je spreman za trening.

```
def fer_model(input_shape=(48, 48, 1)):  
    # First input model  
    visible = Input(shape=input_shape, name='input')  
    num_classes = 7  
  
    # Block 1  
    conv1_1 = Conv2D(64, kernel_size=3, activation='relu', padding='same', name='conv1_1')(visible)  
    conv1_1 = BatchNormalization()(conv1_1)  
    conv1_2 = Conv2D(64, kernel_size=3, activation='relu', padding='same', name='conv1_2')(conv1_1)  
    conv1_2 = BatchNormalization()(conv1_2)  
    pool1_1 = MaxPooling2D(pool_size=(2, 2), name='pool1_1')(conv1_2)  
    drop1_1 = Dropout(0.3, name='drop1_1')(pool1_1)  
    conv2_1 = Conv2D(128, kernel_size=3, activation='relu', padding='same', name='conv2_1')(drop1_1)  
    conv2_1 = BatchNormalization()(conv2_1)  
    conv2_2 = Conv2D(128, kernel_size=3, activation='relu', padding='same', name='conv2_2')(conv2_1)  
    conv2_2 = BatchNormalization()(conv2_2)  
    conv2_3 = Conv2D(128, kernel_size=3, activation='relu', padding='same', name='conv2_3')(conv2_2)  
    conv2_3 = BatchNormalization()(conv2_3)  
    pool2_1 = MaxPooling2D(pool_size=(2, 2), name='pool2_1')(conv2_3)  
    drop2_1 = Dropout(0.3, name='drop2_1')(pool2_1)  
    conv3_1 = Conv2D(256, kernel_size=3, activation='relu', padding='same', name='conv3_1')(drop2_1)  
    conv3_1 = BatchNormalization()(conv3_1)  
    conv3_2 = Conv2D(256, kernel_size=3, activation='relu', padding='same', name='conv3_2')(conv3_1)  
    conv3_2 = BatchNormalization()(conv3_2)  
    conv3_3 = Conv2D(256, kernel_size=3, activation='relu', padding='same', name='conv3_3')(conv3_2)  
    conv3_3 = BatchNormalization()(conv3_3)  
    conv3_4 = Conv2D(256, kernel_size=3, activation='relu', padding='same', name='conv3_4')(conv3_3)  
    conv3_4 = BatchNormalization()(conv3_4)  
    pool3_1 = MaxPooling2D(pool_size=(2, 2), name='pool3_1')(conv3_4)  
    drop3_1 = Dropout(0.3, name='drop3_1')(pool3_1)  
    conv4_1 = Conv2D(256, kernel_size=3, activation='relu', padding='same', name='conv4_1')(drop3_1)  
    conv4_1 = BatchNormalization()(conv4_1)  
    conv4_2 = Conv2D(256, kernel_size=3, activation='relu', padding='same', name='conv4_2')(conv4_1)  
    conv4_2 = BatchNormalization()(conv4_2)  
    conv4_3 = Conv2D(256, kernel_size=3, activation='relu', padding='same', name='conv4_3')(conv4_2)  
    conv4_3 = BatchNormalization()(conv4_3)  
    conv4_4 = Conv2D(256, kernel_size=3, activation='relu', padding='same', name='conv4_4')(conv4_3)  
    conv4_4 = BatchNormalization()(conv4_4)  
    pool4_1 = MaxPooling2D(pool_size=(2, 2), name='pool4_1')(conv4_4)
```

Slika 6.1 - Prikaz prvog dela CNN modela za FER


```

pool4_1 = MaxPooling2D(pool_size=(2, 2), name='pool4_1')(conv4_4)
drop4_1 = Dropout(0.3, name='drop4_1')(pool4_1)

# Block 5
conv5_1 = Conv2D(512, kernel_size=3, activation='relu', padding='same', name='conv5_1')(drop4_1)
conv5_1 = BatchNormalization()(conv5_1)
conv5_2 = Conv2D(512, kernel_size=3, activation='relu', padding='same', name='conv5_2')(conv5_1)
conv5_2 = BatchNormalization()(conv5_2)
conv5_3 = Conv2D(512, kernel_size=3, activation='relu', padding='same', name='conv5_3')(conv5_2)
conv5_3 = BatchNormalization()(conv5_3)
conv5_4 = Conv2D(512, kernel_size=3, activation='relu', padding='same', name='conv5_4')(conv5_3)
conv5_4 = BatchNormalization()(conv5_4)
pool5_1 = MaxPooling2D(pool_size=(2, 2), name='pool5_1')(conv5_4)
drop5_1 = Dropout(0.3, name='drop5_1')(pool5_1)
flatten = Flatten(name='flatten')(drop5_1)
output = Dense(num_classes, activation='softmax', name='output')(flatten)
model = Model(inputs=visible, outputs=output)

Cekic, 6/14/22, 11:05 PM • created a model after waiting more than 11 hours for it to train the data, s

# Summary
print(model.summary())

return model

```

Slika 6.2 - Prikaz drugog dela CNN modela za FER

Kompajlovanje modela

Kompajlujemo model korišćenjem Adam biblioteke. Podešavamo optimizator učenja da bude 0.001, a ako se tačnost modela ne poboljša za neki broj epoha koristimo decay da smanjuje broj učenja.

```

model = fer_model()
opt = Adam(learning_rate=0.0001, decay=1e-6)
model.compile(loss='categorical_crossentropy', optimizer=opt, metrics=['accuracy'])

```

Slika 7 - Prikaz kompajlovanja modela

Treniranje podataka

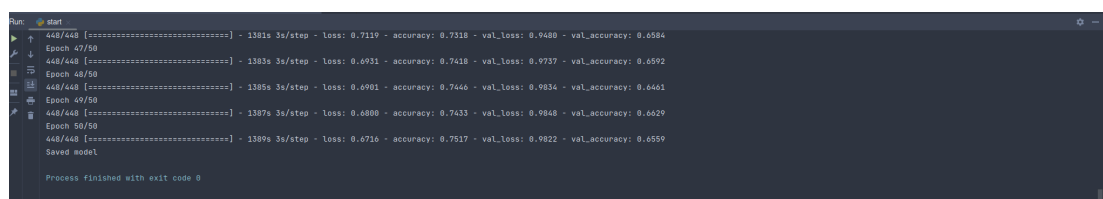
Kada smo spremili i reshapovali podatke tako da budu spremne za treniranje i testiranje može da se pokrene trening podataka. Treniranje se izvršava sa 50 epoha i svi podaci se čuvaju u model.json fajlu a težine u face_recognition_model.h5. Jednoj epohi je potrebno najmanje 20 minuta da istrenira podatak, dok za 50 epoha je potrebno minimalno 11 sati.

```
num_epochs = 50
history = model.fit(train_flow,
                    steps_per_epoch=len(x_train) / batch_size,
                    epochs=num_epochs,
                    verbose=1,
                    validation_data=test_flow)

model_json = model.to_json()
with open("model.json", "w") as json_file:
    json_file.write(model_json)
model.save_weights("face_recognition_model.h5")
print("Saved model")
```

Slika 8 - Prikaz kreiranja model.json fajla

Nakon više od 11 sati treniranja podataka, uspešnost treniranja je sa 75% preciznosti. Svi podaci koji su istrenirani se čuvaju u model.json, a težine se nalaze u face_recognition_model.h5 fajlu i mogu da se testiraju.



```
Run: start
448/448 [=====] - 1381s 3s/step - loss: 0.7119 - accuracy: 0.7318 - val_loss: 0.9480 - val_accuracy: 0.6584
Epoch 47/50
448/448 [=====] - 1383s 3s/step - loss: 0.6931 - accuracy: 0.7418 - val_loss: 0.9737 - val_accuracy: 0.6592
Epoch 48/50
448/448 [=====] - 1385s 3s/step - loss: 0.6901 - accuracy: 0.7446 - val_loss: 0.9854 - val_accuracy: 0.6401
Epoch 49/50
448/448 [=====] - 1307s 3s/step - loss: 0.6800 - accuracy: 0.7433 - val_loss: 0.9848 - val_accuracy: 0.6629
Epoch 50/50
448/448 [=====] - 1389s 3s/step - loss: 0.6716 - accuracy: 0.7517 - val_loss: 0.9822 - val_accuracy: 0.6559
Saved model
Process finished with exit code 0
```

Slika 9 - Prikaz konzole nakon završetka treniranja podataka

Testiranje podataka

Nakon što je treniranje podataka završeno možemo da nastavimo dalje sa testiranjem podataka. Podatke testiramo u real time-u preko web kamere. To možemo da izvedemo korišćenjem OpenCV biblioteke.

Pre svega potrebno je da pročitamo podatke koje smo istrenirali i težine koji su smešteni u fajlovima.

```
# loading model and weights
model = model_from_json(open("model.json", "r").read())
model.load_weights('face_recognition_model.h5')
```

Slika 10 - Prikaz čitanja modela

Nakon toga potrebno je iskoristiti `haarcascade_frontalface_default` biblioteku radi detekcije pozicije lica, i nakon čega cropujemo lice. Takođe neklarišemo niz naziva emocija. Pored toga pozivamo sa `cv2.VideoCapture` da bismo upalili web kameru.

```
face_haar_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
camera = cv2.VideoCapture(0)
emotion_labels = ['Angry', 'Disgust', 'Fear', 'Happy', 'Sad', 'Surprise', 'Neutral']
```

Slika 11 - Prikaz poziva `haarcascade_frontalface_default` biblioteke

Jedino što nam ostaje na kraju je da očitamo frejmove i da primenimo preprocesse korišćenjem `OpenCV` biblioteke.

```

while True:
    res, frame = camera.read()
    labels = []
    gray_image = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    faces = face_haar_cascade.detectMultiScale(gray_image)

    for (x, y, w, h) in faces:
        cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 255), 2)
        roi_gray = gray_image[y:y + h, x:x + w]
        roi_gray = cv2.resize(roi_gray, (48, 48), interpolation=cv2.INTER_AREA)

        if np.sum([roi_gray]) != 0:
            roi = roi_gray.astype('float') / 255.0
            roi = img_to_array(roi)
            roi = np.expand_dims(roi, axis=0)

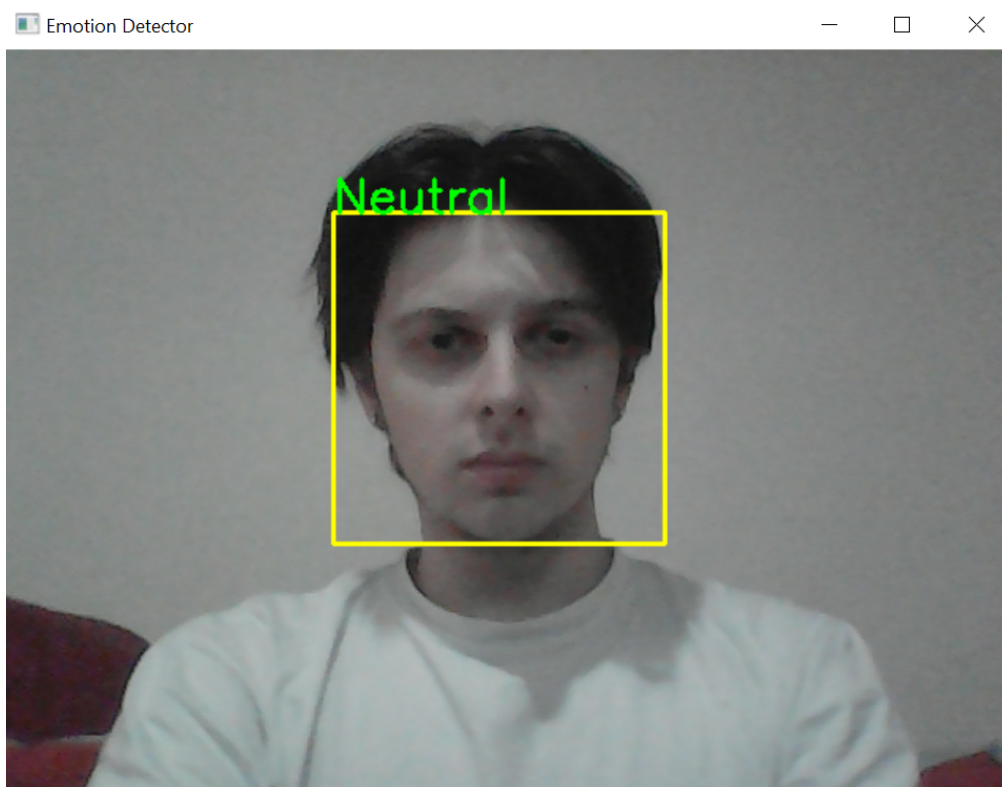
            prediction = model.predict(roi)[0]
            label = emotion_labels[prediction.argmax()]
            label_position = (x, y)
            cv2.putText(frame, label, label_position, cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
        else:
            cv2.putText(frame, 'No Faces', (30, 80), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
    cv2.imshow('Emotion Detector', frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

camera.release()
cv2.destroyAllWindows()

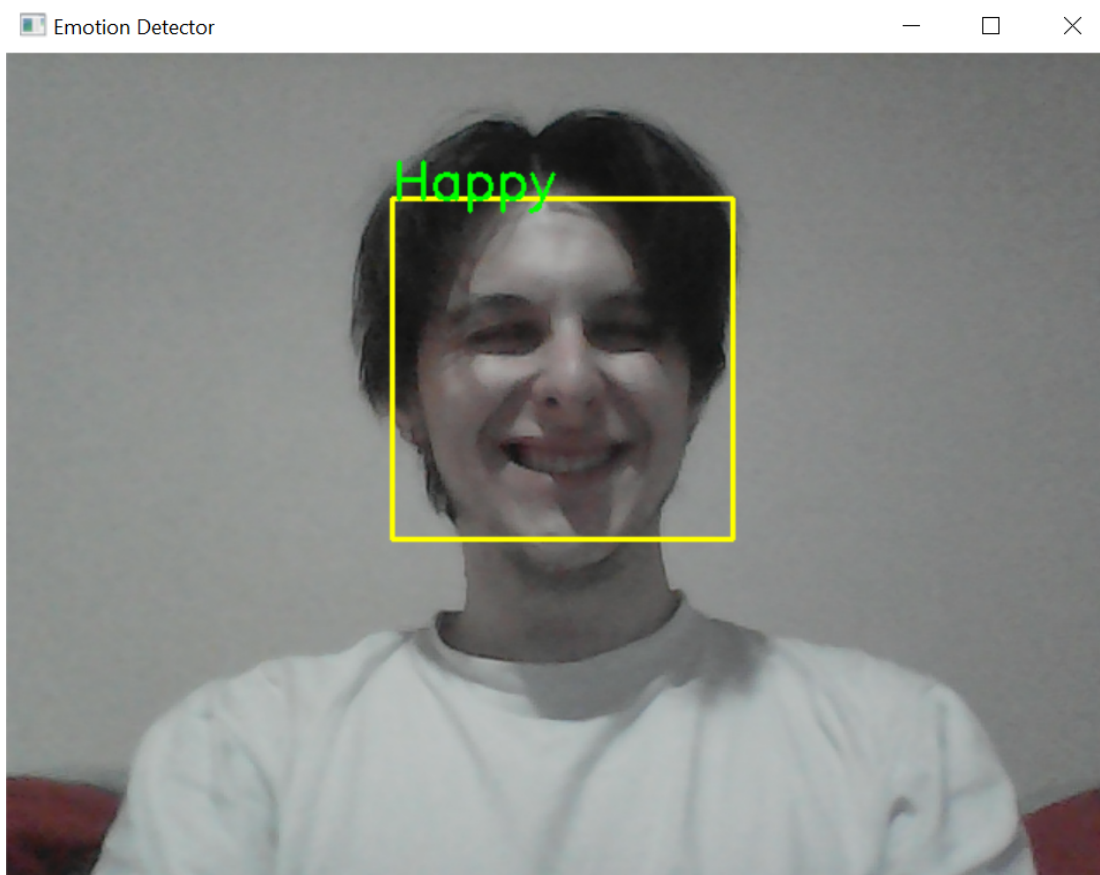
```

Slika 12 - Prikaz učitavanje frejmova

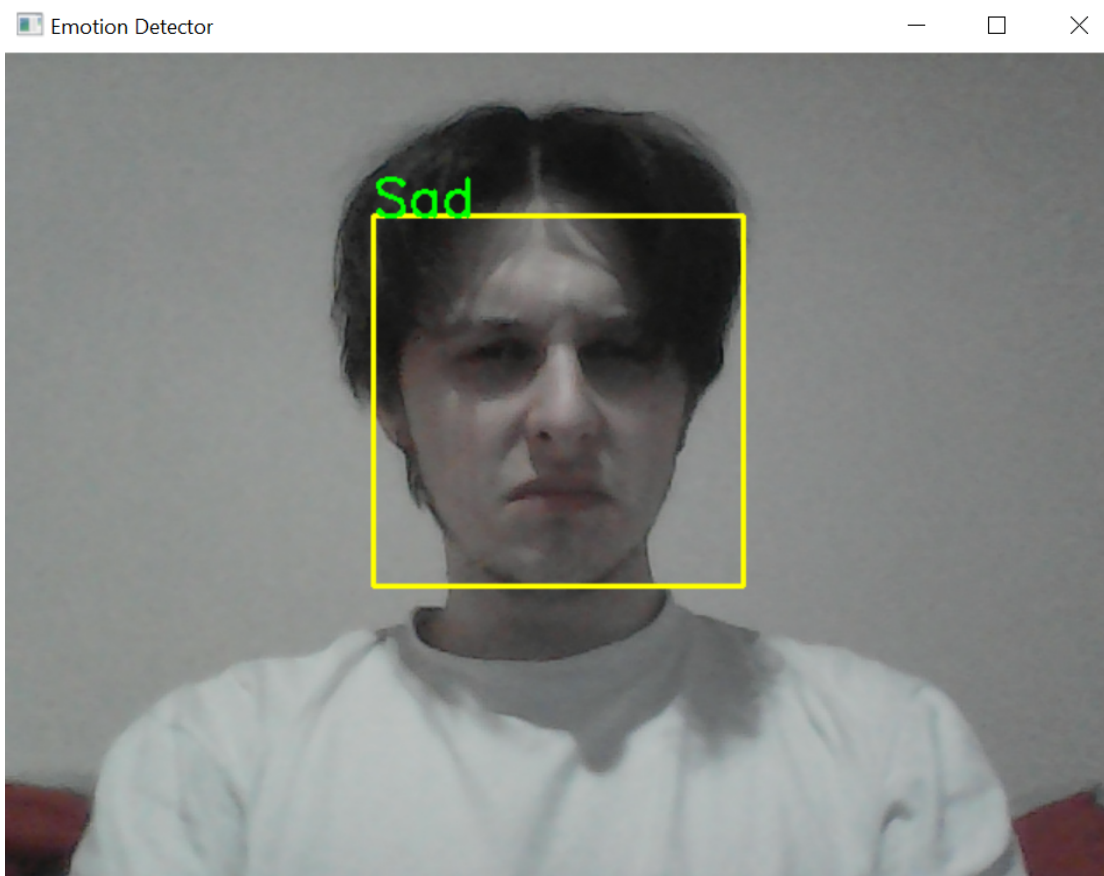
Na kraju možemo proveriti da li je treniranje podataka prošlo dobro ili ne. Na slici vidimo finalni prikaz aplikacije. Kao što vidimo na ovom primeru ovaj kvadrat pronalazi lice i označava emociju na ekranu.



Slika 13.1 - Prikaz emocije 1



Slika 13.2 - Prikaz emocije 2



Slika 13.3 - Prikaz emocije 3

Diskusija o poboljšanju tačnosti

Poboljšanje tačnosti može da se ostvari povećanjem broja epoha od 50 na 100 epoha. S obzirom na to da bi imao više vremena da istrenira podatke, tačnost klasifikacije bi se znatno povećala. Vremenski je potrebno 20 minuta za vreme treniranja po jednoj epohi, ukoliko bi se broj epoha povećao na 100 ukupno vreme treniranja bi prelazilo 33 sati što može dovesti do znatnog povećanja tačnosti. Takođe neke od tehnika koje mogu da se primene da bi se povećala tačnost pored epoha bi bilo menjanje kernela (da se poveća ili smanji), da se poveća broj slojeva u mreži što može doći do boljih rezultata kao i promena batch veličine (podataka za treniranje). Prilikom menjanja nekih od ovih podataka previše puta ili ukoliko se previše treniraju podaci, može doći do overfitting podataka.

Zaključak

Ovaj projektni zadatak, kao i ovaj predmet se pokazao kao uvod u veštaču inteligenciju i kako ona funkcioniše. Neke od tehnika kao što je machine learning i deep learning su pružili dobar uvid u to kako podaci mogu da se treniraju i kako korišćenjem raznih algoritama mogu da se reše razni problemi.

U ovom projektu je postignuta realizacija aplikacije za prikaz emocija korišćenjem CNN algoritma za prepoznavanje objekata na slikama i sa podacima koji su preuzeti sa [Keggle](#) platforme.

Podaci koji su istrenirani sa uspešnosima od 75% su čuvani kao modeli, nakon čega su korišćene uz web kameru. Aplikacija učitava frejm po frejm na čemu se istreniran model može testirati i rezultate koje smo dobili su uglavnom pozitivni.

Ceo kod možete pronaći na mom [GitHub](#) repozitorijumu.

Literatura

Analytics Vidhya - Learn Machine learning, artificial intelligence, business analytics, data science, big data, data visualizations tools and techniques.

(n.d.). Analytics Vidhya. Retrieved June 19, 2022, from <https://www.analyticsvidhya.com/>

Brownlee, J. (2019, May 12). *How to develop a CNN from scratch for CIFAR-10 photo classification.* Machine Learning Mastery.

<https://machinelearningmastery.com/how-to-develop-a-cnn-from-scratch-for-cifar-10-photo-classification/>

CNN Explainer. (n.d.). Github.io. Retrieved June 19, 2022, from

<https://poloclub.github.io/cnn-explainer/>

Login - LAMS :: Learning activity management system. (n.d.).

Metropolitan.Ac.Rs:8080. Retrieved June 19, 2022, from <http://lams.metropolitan.ac.rs:8080/lams/index.do>