NASLOV PROJEKTNOG ZADATKA:	*Prepoznavanje gesti/znakova izvedenih rukom u video signalu*
KOLEGIJ:	*Primijenjeno strojno učenje*
IME I PREZIME STUDENTA:	*Lea Gregurić, David Mikulić*
AKADEMSKA GODINA:	*2022./2023.*
NAPOMENA:	

SADRŽAJ

1. UVOD

Ovaj projekt bavi se razvojem potpuno povezane neuronske mreže koristeći duboko učenje kako bi u stvarnom vremenu moglo prepoznavati geste izvedene rukom kroz video signal. Cilj ovog projekta je izrada Python aplikacije koja pali web kameru računala, te koristi istrenirani model za klasificiranje prikazane geste rukom.

U svrhu ovog projekta potrebno je prikupiti skup podataka slika gdje korisnik prikazuje gestu rukom. Podatke smo prikupljali sami kroz web kameru računala i za svaku od 5 različitih gesti skupili 2000 slika.

Zatim se jedna po jedna slika obrađuje koristeći Google-ova Mediapipe biblioteka koja iscrtava „kostur“ ruke ako ju pronađe na slici. „Kostur“ ruke sastoji se od 21 točke koje sadrže svoje X i Y koordinate. Podaci svake slike spremaju se u jedan red .csv datoteke. Prvi stupac služi za oznaku klase (0-4), a drugi sadrži podatke koordinata točaka kostura ruke dobivenih sa slike.

Pomoću generirane .csv datoteke trenira se model koristeći Python te biblioteke Keras i Numpy.

Nakon treniranja modela, evaluira se njegova performansa na testnom skupu i vrši se podešavanje hiperparametara ako je potrebno kako bi se postigla optimalna točnost.

Na kraju se izrađuje aplikacija koja pokreće web kameru pomoću bilioteke OpenCV i koristeći Mediapipe za praćenje i prepoznavanje ruku, te izrađeni model koji prepoznaje geste ruke.

2. IMPLEMENTACIJA RASPOZNAVANJA ODREĐENE GESTE RUKOM

Osnovni problem koji se rješava je identifikacija specifičnih gestikulacija ruke koje korisnik izvodi pred kamerom. Prepoznavanje gesti predstavlja izazovan zadatak zbog varijabilnosti u izvedbi, različitih pozadina i osvjetljenja, te različitih perspektiva kamere. Sustav treba uspješno identificirati geste unatoč ovim izazovima kako bi pružio korisnicima intuitivan način interakcije.

Postoje različiti pristupi za rješavanje problema prepoznavanja gestikulacija ruke kao što je ručno projektiranje značajki i klasifikacija.

Najpoznatiji i najrelevantiji framework, ujedno i korišten za ostvarenje ovog projekta, za praćenje kretnje i prepoznavanje ruke unutar slike ili video signala je platforma Mediapipe. Mediapipe je otvorena platforma za detekciju i praćenje različitih karakteristika u stvarnom vremenu. Za detekciju gesti ruke, mediapipe pruža gotov model i API koji omogućuju praćenje ključnih točaka ruke (landmarka) u stvarnom vremenu. Potrebno je prikupiti podatke (fotografije specifičnih gesti ruke), označiti ih, te na podacima dobivenim o rukama s prikupljenih fotografija istrenirati model da prepoznaje geste (svaka gesta je jedna označena klasa). Mediapipe služi samo za prepoznavanje ruke, ali ne i specifičnih gesti.

2.1. Priprema podatkovnog skupa

Učitavanje biblioteka: Za stvaranje podatkovnog skupa korištene su 3 biblioteke: os, csv, Mediapipe i OpenCV.
Inicijalizacija modula za detekciju i crtanje ruku: Pomoću bibilioteke Mediapipe inicijaliziramo modul za prepoznavanje ruke i iscrtavanje 21 točke ruke. Također stvara se objekt hands koji je instanca detektora ruke s određenim parametrima.

Stvaranje datoteke i učitavanje direktorija sa slikama: Pomoću biblioteke os i biblioteke csv dohvaća se direktorij sa slikama, te stvara datoteka u koju se upisuju podaci i pomoću metode DictWriter priprema se za pisanje.

Otvaranje slika i čitanja značajki ruke: Prvo se otvara jedan po jedan folder pomoću petlje (for directory in os.listdir(DATA_DIR)), pa onda se svaka slika otvara pojedinačno pomoću biblioteke OpenCV i njene metode imread. Zatim pročitanu sliku se mora konvertirati iz BGR modela boja u RGB model boja jer OpenCV sprema boju u BGR formatu[2] a Mediapipe koristi RGB format za detekciju i praćenje – za te se koristi metoda cv2.cvtColor(…). Stvara se novi objekt 'results' u koji se pohranjuju rezultati o detekciji i pozicijama značajki ruke prethodno konvertirane slike u RGB.

Obrada informacija o značajkama ruke: Prvo se provjerava postoji li uopće ruka na slici, te ako postoji za svaku ruku se pokreću dvije unutarnje petlje. Unutar prve unutarnje petlje za svaku značajku se izračunavaju x i y koordinate (x = hand_landmarks.landmark[i].x i y = hand_landmarks.landmark[i].y) koje su normalizirane u odnosu na veličinu slike. Te x i y koordinate pohranjuju se u listu x_ i y_ koje privremeno pohranjuju sve koordinate svih značajki za trenutnu ruku. Unutar druge unutarnje petlje za svaku značajku ruke ponovo se izračunavaju x i y koordinate, a zatim se od tih koordinata oduzima minimalna x i y koordinata koja su pohranjene u liste x_ i y_ - ovime se dobivaju relativne koordinate značajki u odnosu na najmanje x i y koordinate. Koriste se relativne koordinate zbog veće pouzdanosti u različitim uvjetima, ako su koordinate apsolutne model može postati istreniran da „misli“ da se neka gesta može nalaziti samo u nekom dijelu slike. Na kraju se te relativne koordinate pohranjuju u listu hand_data.

Pohrana u .csv datoteku: Pomoću već stvorenog objekta writer biblioteke csv, metodom writerow upisuju se sve relativne koordinate (spremljene u listu hand_data) s odgovarajućom oznakom za te podatke, oznaka se uzima uzeli iz imena foldera gdje se nalaze sve slike iste geste. Za svaku gestu je prikljupljeno 2000 podataka. Geste su „bok“, „okej“ , „peace“ , „like“ i „dislike“.

Sl. 2.1.1 Primjer slike iz podatkovnog skupa.

Sl. 2.1.2 Što detektira Mediapipe

```
0, 0.11011868715286255, 0.36872196197509766, 0.06665652990341187, 0.31428927183151245,
0.042151570320129395, 0.24031835794448853, 0.023547351360321045, 0.187771737575531, 0.0,
0.1558450162410736, 0.11297112703323364, 0.16916286945343018, 0.11374795436859131,
0.09205690026283264, 0.11573082208633423, 0.04371201992034912, 0.11777091026306152,
0.004326462745666504, 0.1489095687866211, 0.1778523325920105, 0.16535234451293945,
0.0965537428855896, 0.1737576723098755, 0.0437411367893219, 0.17925196886062622, 0.0,
0.17858868837356567, 0.20339196920394897, 0.2050086259841919, 0.13390761613845825,
0.217942476272583, 0.08728474378585815, 0.22708964347839355, 0.04543390870094299,
0.20131683349609375, 0.24363559484481812, 0.2395046353340149, 0.21128028631210327,
0.26305055618286133, 0.18850988149642944, 0.28270018100738525, 0.1668943166732788
```

Sl. 2.1.3Podaci pohranjeni u .csv datoteku za sliku 2.1.1

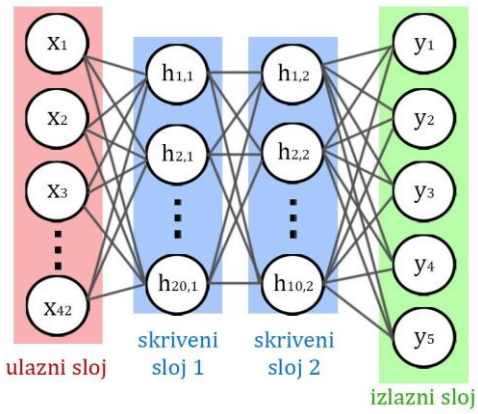
Broj 0 označen žutom bojom predstavlja oznaku (label) geste. Ostali brojevi predstavljaju koordinate značajnih točaka ruke.

2.2. Strukturiranje, treniranje i evaluacija potpuno povezane neuronske mreže

Učitavanje biblioteka: Za izradu modela korištene su 3 biblioteke: Numpy, Tensorflow Keras i metoda train_test_split iz biblioteke Scikit-learn. **Učitavanje podataka:** Podaci našeg skupa podataka učitavaju se iz .csv datoteke pomoću funkcije loadtxt iz Numpy biblioteke. Posebno se spremaju podaci o koordinatama točaka (x_dataset) i oznake pojedine geste (y_dataset).

Priprema podataka: Podaci se dijele na trening i testne podatke pomoću funkcije train_test_split te se pohranjuju u varijable x_train i x_test a oznake u y_train i y_test. Zbog ove podjele model uči pozicije točaka ruke i predviđa geste ruke.

Stvaranje modela: Stvara se model koristeći klasu Sequential biblioteke Keras te se zadaju sljedeći parametri: ulazni sloj ima ulaznu dimenziju (input shape) 42 što predstavlja broj značajki korištenih za prepoznavanje ruku, odnosno 21 X koordinata i 21 Y koordinata. Nakon njega dodaje se dropout sloj koji služi sprječavanju prenaučenosti modela. Dodaje se gusti sloj s 20 neurona i funkcijom aktivacije ReLU. Nakon njega slijedi još jedan dropout sloj radi regulacije, te još jedan gusti sloj od 10 neurona. Posljednji sloj, odnosno output sloj ima broj neurona jednak broju klasa (5) i funkcijom aktivacije softmax koja izračunava vjerojatnosti za svaku klasu.



Sl. 2.2.1 Prikaz strukture korištene neuronske mreže.

Kompilacija modela: Koristi se optimizator 'adam' prilikom kompajliranja modela, koji je napredna varijanta stohastičkog gradijentnog spusta (SGD). Za mjerenje gubitaka (loss) koristi se 'sparse_categorical_crossentropy' koji je pogodan za zadatke višeklasne klasifikacije. Definira se i metrika točnosti (accuracy) koja služi za ocjenjivanje izvedbe modela tijekom procesa treniranja.

Treniranje modela: Model se trenira na podatkovnom skupu za treniranje pomoću metode fit. Određuju se parametri treniranja: broj epoha, odnosno broj iteracija preko cijelog skupa podataka postavlja se na 500 ,te veličina grupa (batch size) za obradu gradijenata postavlja se na 128.

Evaluacija modela: Kada je model istreniran, evaluiraju se performanse modela na skupu za testiranje pomoću metode evaluate. Izračunava se i ispisuje gubitak, te točnost na testnom skupu.

Pohrana modela: Istrenirani model pohranjuje se u folder 'model' pomoću metode save. S time se omogućuje buduće korištenje modela, odnosno u aplikaciji.

2.3. Izvedba sučelja koje prepoznaje geste ruke u stvarnom vremenu

Učitavanje biblioteka: Za stvaranje podatkovnog skupa korištene su 4 biblioteke: OpenCV, Numpy, Mediapipe, te metoda load_model iz biblioteke keras.models.

Inicijalizacija modula za detekciju i crtanje ruku: Pomoću frameworka Mediapipe inicijalizira se modul za prepoznavanje ruke i iscrtavanje 21 točke ruke. Također stvara se objekt hands koji je instanca detektora ruke s određenim parametrima.

Učitavanje istreniranog modela: Kako bi se imalo na temelju čega pogađati geste ruke, učitava se istrenirani model pomoću metode load_model.

Inicijalizacija liste s imenima klasa: Stvara se lista class_names s pripadajućim imenima za svaku gestu ruke iz podatkovnog skupa kako bi se mogla ispisivati predikcija u obliku teksta.

Otvaranje video ulaza s kamerom: Pomoću biblioteke OpenCV i metode VideoCapture uključuje se kamera.

Čitanje sljedećeg framea kamere: Koristi se funkcija cv2.read() biblioteke OpenCV koja vraća dvije vrijenosti: 1. 'ret' (boolean) označava je li čitanje bilo uspješno i 'frame' (slika) predstavlja cijelu sliku pročitanu i sadrži informacije o pikselima i bojama koji čine sliku. Nakon toga se zrcalno okreće sliku kako bi bilo intuitivnije korištenje, kao ogledalo. Opet se pretvara iz BGR formata u RGB, te stvara novi objekt 'results' u koji se pohranjuju rezultati o detekciji i pozicijama značajki ruke prethodno konvertirane slike u RGB kao kod kreiranja skupa podataka.

Obrada informacija o značajkama ruke: Prvo se provjerava postoji li uopće ruka na slici, te ako postoji za svaku ruku se pokreću dvije unutarnje petlje. Unutar prve unutarnje petlje za svaku značajku se izračunavaju x i y koordinate (x = hand_landmarks.landmark[i].x i y = hand_landmarks.landmark[i].y) koje su normalizirane u odnosu na veličinu slike. Te x i y koordinate pohranjuju se u listu x_ i y_ koje privremeno pohranjuju sve koordinate svih značajki za trenutnu ruku. Unutar druge unutarnje petlje za svaku značajku ruke ponovo se izračunavaju x i y koordinate, a zatim se od tih koordinata oduzima minimalna x i y koordinata koja su pohranjene u liste x_ i y_ - ovime dobivamo relativne koordinate značajki u odnosu na najmanje x i y koordinate. Koriste se relativne koordinate zbog veće pouzdanosti u različitim uvjetima, ako su koordinate apsolutne model može postati istreniran da „misli“ da se neka gesta može nalaziti samo u nekom dijelu slike. Na kraju se te relativne koordinate pohranjuju u listu hand_data.

Prikaz „kostura“ ruke i predviđanje: Ovoga puta lista hand_data se ne pohranjuje u .csv datoteku već se koristi za prikaz „kostura“ ruke (koristeći mp_draw.draw_landmarks) i predviđanje geste ruke koju korisnik pokazuje. Kako bi se mogle prosljediti spremljene koordinate istreniranom modelu, konvertira se lista hand_data u Numpy niz koristeći np.array(). Nakon konverzije u Numpy niz, hand_data sadrži relativne koordinate točaka ruke kao linearni niz, da bi se koordinate mogle prosljediti modelu koristi se funkcija reshape(-1,42). Prvi argument -1 služi kako bi se automatski izračunala veličina prve dimenzije niza, a drugi je postavljen

na 42 kako bi se osiguralo da svaki podatak ruke ima 42 koordinate (21 x koordinata i 21 y koordinata). Nakon oblikovanja relativnih koordinata, koriste se kao ulazni podaci za istrenirani model. Pozivanjem metode predict(), model izvršava predviđanje temeljeno na ulaznim podacima hand_data i vraća izlazne vrijednosti za svaku klasu geste. Također se koristi parametar verbose=0 kako bi sprječili konstantan ispis u terminal, s obzirom da se ovo sve izvršava u beskonačnoj petlji.

Klasifikacija geste: Izlaz iz modela vraća vjerojatnost za svaku klasu zbog primjene aktivacijske funkcije softmax. Kako bi se uspješno klasificirala gesta koju je model predvidio da jest, pohranjuje se indeks najveće vrijednosti iz matrice prediction (prediction je matrica a ne lista jer metoda predict vraća matricu, zapravo imamo matricu od jednog reda) koristeći funkciju np.argmax(prediction). Taj indeks odgovara onome u listi class_names pa se ime klase dobiva sa class_name = class_names[classID]. Prvi red matrice (i jedini u ovom slučaju) prediction pretvara se u listu i pohranjuje u novu listu confidenceList. Samo vrijednost samopouzdanja u postotcima pohranjuje se u varijablu confidence funkcijama: str(round(confidenceList[classID]*100, 2))+'%' i tako se dobije samopuzdanje modela izraženo u postotcima.

Prikaz slike i teksta: Na kraju se prikazuje predviđena gestu i postotak samopouzdanja u obliku teksta pomoću metode cv2.putText(…). Metodom cv2.imshow('Output', frame) postavlja se ime prozora i prikazuje se na ekran korisniku.

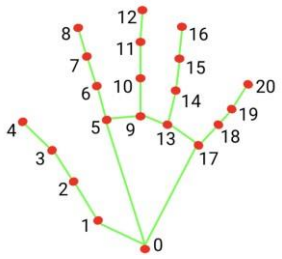
Izlaz iz programa: Kako se ovo sve vrti u jednoj beskonačnoj petlji mora se omogućiti način za izlaz iz programa, OpenCV sadrži metodu cv2.waitKey(*ms*) pomoću koje kada korisnik stisne zadanu tipku može prekinuti petlju i završiti program. Prilikom završavanja programa koriste se još metode cap.release() za oslobađanje resursa kamere i cv2.destroyAllWindows() koji zatvara sve otvorene prozore.

3. IMPLEMENTACIJA RJEŠENJA U JEZIKU PYTHON

Svi koraci detaljnije su objašnjeni u drugom poglavlju.

3.1. Stvaranje podatkovnog skupa iz prikupljenih slika u .csv datoteku

Za obradu prikupljenih podataka koristi se skripta createdataset.py koja koristeći Mediapipe u svakoj fotografiji pronalazi ruku, te označava 21 točku na ruci kako bi se odredio točan položaj. Svaka od tih točaka ima X i Y koordinatu koja se s odgovarajućom oznakom geste sprema u .csv datoteku – svaka fotografija je zauzela jedan red datoteke.



0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP

[1]

Sl. 3.1.1 Prikaz 21 značajke ruku

U ovom dijelu koda stvara se objekt hands koji ima parametre: static_image_mode = False (jer iako trenutno koristi statične slike radit će s video signalom), detektira maksimalno jednu ruku, te zadnja dva parametra su minimalna samopouzdanost da postoji ruka koju detektira.

```
7   mp_hands = mp.solutions.hands
8   mp_draw = mp.solutions.drawing_utils
9
10  hands = mp_hands.Hands(static_image_mode = False,
11                         max_num_hands=1,
12                         min_detection_confidence=0.5,
13                         min_tracking_confidence=0.5)
```

Sl. 3.1.2 Stvaranje modula detekcije ruke i objekta hands

```
16  DATA_DIR = './data'
17  file = open('data.csv', 'w')
18  fields = ('label','data')
19  writer = csv.DictWriter(file, fieldnames=fields, lineterminator='\n')
20
21  for directory in os.listdir(DATA_DIR):
22      for img_path in os.listdir(os.path.join(DATA_DIR, directory)):
23          hand_data = []
24          x_ = []
25          y_ = []
26
27          frame = cv2.imread(os.path.join(DATA_DIR, directory, img_path))
28          framergb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
29
30          results = hands.process(framergb)
31
32          if results.multi_hand_landmarks:
33              for hand_landmarks in results.multi_hand_landmarks:
34                  for i in range(len(hand_landmarks.landmark)):
35                      x = hand_landmarks.landmark[i].x
36                      y = hand_landmarks.landmark[i].y
37
38                      x_.append(x)
39                      y_.append(y)
40
41                  for i in range(len(hand_landmarks.landmark)):
42                      x = hand_landmarks.landmark[i].x
43                      y = hand_landmarks.landmark[i].y
44                      hand_data.append(x - min(x_))
45                      hand_data.append(y - min(y_))
46
47              writer.writerow({'label' : directory, 'data' : hand_data})
48
49
50  file.close()
```

Sl. 3.1.3 Spremanje koordinata značajka ruke u listu i upisivanje u .csv s odgovarajućim oznakama

3.2. Modeliranje neuronske mreže

U ovom koraku pripremljeni podaci se čitaju iz .csv datoteke, svaki redak sadrži sve koordinate ruke i oznaku. Oznake se spremaju u y_dataset, a koordinate u x_dataset. Time su se pripremili podaci za treniranje i testiranje.

```python
import numpy as np
import tensorflow as tf
from sklearn.model_selection import train_test_split

RANDOM_SEED = 42
NUM_CLASSES = 5

dataset = 'data.csv'
model_save_path = 'model'


x_dataset = np.loadtxt(dataset,
                       delimiter=',',
                       dtype='float32',
                       usecols=list(range(1, (42) + 1)))
y_dataset = np.loadtxt(dataset,
                       delimiter=',',
                       dtype='int32',
                       usecols=(0))

x_train, x_test, y_train, y_test = train_test_split(x_dataset,
                                                    y_dataset,
                                                    train_size=0.75,
                                                    random_state=RANDOM_SEED,
                                                    stratify=y_dataset)
```

Sl. 3.2.1 Prilagodba i čitanje podataka prije treniranja neuronske mreže

Ovaj dio koda definira strukturu neuronske mreže koja se koristi za prepoznavanje gesti ruke. Neuronska mreža je definirana kao sekvenca slojeva pomoću TensorFlow i Keras biblioteka.

```python

model = tf.keras.models.Sequential([
    tf.keras.layers.Input(shape=(42,)),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(20, activation='relu'),
    tf.keras.layers.Dropout(0.4),
    tf.keras.layers.Dense(10, activation='relu'),
    tf.keras.layers.Dense(NUM_CLASSES, activation='softmax')
])

```

Sl. 3.2.2 Izgled odabranog modela

3.3. Rezultati treniranja i pohrana modela

Ovaj dio koda predstavlja postupak kompilacije i treniranja neuronske mreže. Nakon kompilacije model se trenira pomoću funkcije model.fit(…), te nakon treniranja sažeto se prikazuje model pomoću funkcije model.summary(), pohranjuje povijest treninga što uključuje točnost i gubitke, te validaciju. Konačno pohranjuje se cijeli model pomoću model.save() u zadanu putanju.

```
model.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)

history = model.fit(
    x_train,
    y_train,
    epochs=500,
    batch_size=128,
    validation_data=(x_test, y_test)
)

model.summary()
train_accuracy = history.history['accuracy']
validation_accuracy = history.history['val_accuracy']
train_loss = history.history['loss']
validation_loss = history.history['val_loss']
model.save(model_save_path)
```

Sl. 3.3.1 Treniranje mreže, spremanje modela i vrijednosti gubitaka i preciznosti

3.4. Stvaranje sučelja

U ovom dijelu koda priprema se sučelje učitavanjem potrebnih bilioteka, te stvaranjem objekta hands kao i u prethodnim koracima gdje se tražila ruka u slici. Učitava se istrenirani model, te otvara kamera. Koristeći model.predict(…) na očitanim podacima iz videosignala predviđa se koju je gestu korisnik pokazao uz pomoć istreniranog modela.

```
import cv2
import numpy as np
import mediapipe as mp
from keras.models import load_model

mp_hands = mp.solutions.hands
mp_draw = mp.solutions.drawing_utils

hands = mp_hands.Hands(static_image_mode = False,
                       max_num_hands=1,
                       min_detection_confidence=0.5,
                       min_tracking_confidence=0.5)

model = load_model('model')

class_names = ['Bok','Peace','Like','Dislike','Okej']

cap = cv2.VideoCapture(0)
```

Sl. 3.4.1 Priprema sučelja za iscrtavanje ruku kreiranjem objekta hands

```
while True:
    ret, frame = cap.read()
    frame = cv2.flip(frame, 1)
    framergb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    result = hands.process(framergb)

    confidence = ''
    class_name = ''
    hand_data = []
    x_ = []
    y_ = []

    if result.multi_hand_landmarks:
        for hand_landmarks in result.multi_hand_landmarks:
            for i in range(len(hand_landmarks.landmark)):
                x = hand_landmarks.landmark[i].x
                y = hand_landmarks.landmark[i].y

                x_.append(x)
                y_.append(y)

            for i in range(len(hand_landmarks.landmark)):
                x = hand_landmarks.landmark[i].x
                y = hand_landmarks.landmark[i].y

                hand_data.append(x - min(x_))
                hand_data.append(y - min(y_))


            mp_draw.draw_landmarks(frame,
                                   hand_landmarks,
                                   mp_hands.HAND_CONNECTIONS)
```

Sl. 3.4.2 Traženje značajki ruku u svakom frameu video signala

```
            hand_data = np.array(hand_data)
            hand_data = hand_data.reshape(-1, 42)

            prediction = model.predict([hand_data], verbose=0)

            classID = np.argmax(prediction)

            confidenceList = list(prediction[0])
            confidence = str(round(confidenceList[classID]*100, 2))+'%'

            class_name = class_names[classID]

    cv2.putText(frame,
                class_name,
                (10, 50),
                cv2.FONT_HERSHEY_DUPLEX,
                1, (0,0,0), 8, cv2.LINE_AA)
    cv2.putText(frame,
                class_name,
                (10, 50),
                cv2.FONT_HERSHEY_DUPLEX,
                1, (255,255,255), 2, cv2.LINE_AA)

    cv2.putText(frame,
                confidence,
                (10, 80),
                cv2.FONT_HERSHEY_DUPLEX,
                0.7, (0,0,0), 3, cv2.LINE_AA)
    cv2.putText(frame,
                confidence,
                (10, 80),
                cv2.FONT_HERSHEY_DUPLEX,
                0.7, (255,255,255), 1, cv2.LINE_AA)
```

Sl. 3.4.3 Predviđanje pokazane geste i prikaz teksta na ekranu

Ovaj dio koda pomoću cv2.imshow('Output', frame) prikazuje video signal i tekst predviđanja na ekranu. Također omogućuje da se izađe iz programa pritiskom tipke ESC.

```
99      cv2.imshow('Output', frame)
100
101     key = cv2.waitKey(10) #ms
102     if key == 27:  # ESC
103         break
104
105
106
107
108 cap.release()
109 cv2.destroyAllWindows()
```

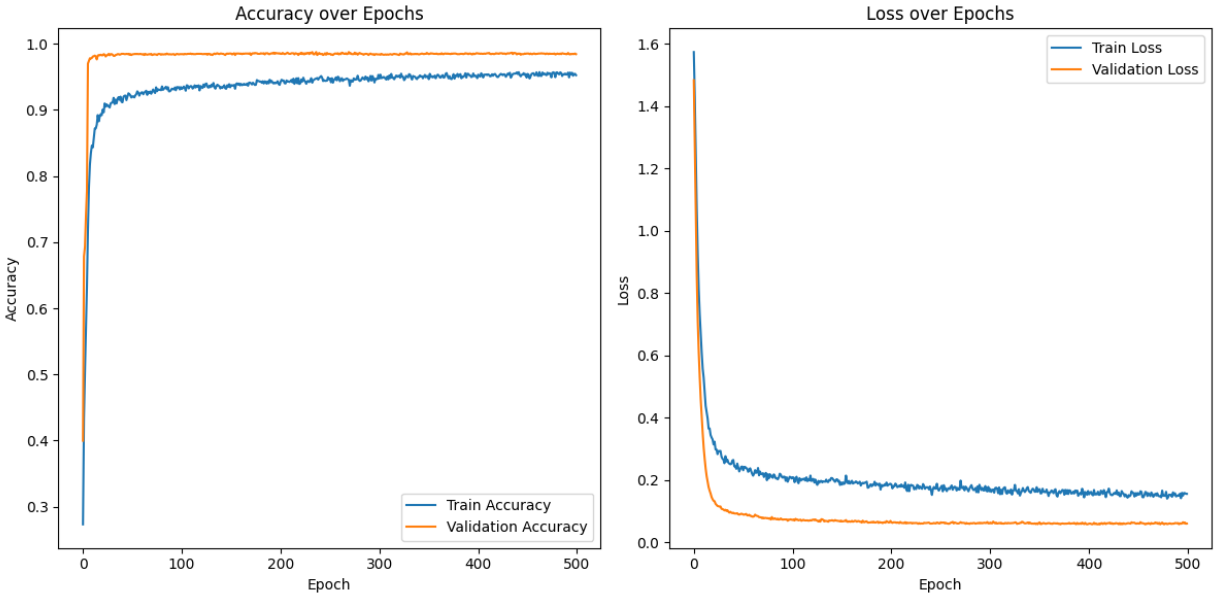
Sl. 3.4.4 Prikaz cijelog framea s tekstovima i izlaz iz programa

4. EVALUACIJA PREDLOŽENOG RJEŠENJA

4.1. Primjena modela na trening i validacijskim podacima

```
Epoch 499/500
53/53 [==============================] - 0s 2ms/step - loss: 0.1421 - accuracy: 0.9531 - val_loss: 0.0506 - val_accuracy: 0.9866
Epoch 500/500
53/53 [==============================] - 0s 2ms/step - loss: 0.1462 - accuracy: 0.9520 - val_loss: 0.0548 - val_accuracy: 0.9862
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dropout (Dropout)           (None, 42)                0

 dense (Dense)               (None, 20)                860

 dropout_1 (Dropout)         (None, 20)                0

 dense_1 (Dense)             (None, 10)                210

 dense_2 (Dense)             (None, 5)                 55

=================================================================
Total params: 1125 (4.39 KB)
Trainable params: 1125 (4.39 KB)
Non-trainable params: 0 (0.00 Byte)
```

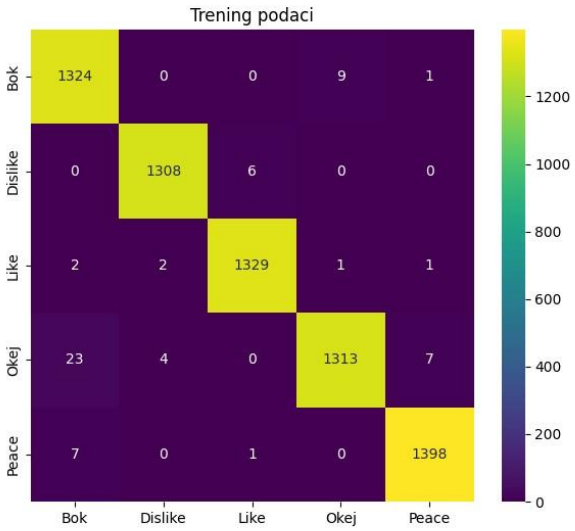
Sl. 4.1.1 Treniranje i sažetak modela



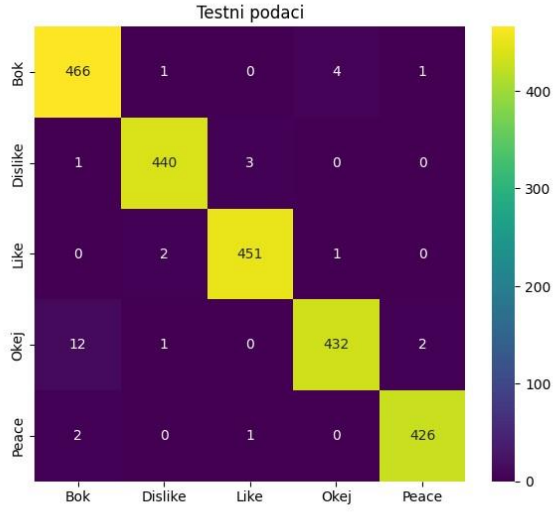
Sl. 4.1.2 Točnost i gubitak klasifikacije na trening i validacijskim podacima

Na prikazanim grafovima primjećuje se da se točnost validacije ne mijenja značajno nakon već 100 epoha, ali train accuracy lagano raste. Također nema značajne promjene u gubitku validacije nakon stotinjak epoha, ali train loss također lagano pada. Ipak, uobičajeno je da je validacijski accuracy veći nego accuracy na train skupu, međutim oba mjerenja točnosti su visoka i krivulje ne odstupaju puno jedna od druge, što ukazuje na dobru generalizaciju modela.

4.2. Matrica zabune modela



Sl. 4.2.1 Matrica zabune na trening podacima



Sl. 4.2.3 Matrica zabune na testnim podacima

Classification Report Trening podaci

	precision	recall	f1-score	support
Bok	0.97	0.99	0.98	472
Dislike	0.99	0.99	0.99	444
Like	0.99	0.99	0.99	454
Okej	0.99	0.97	0.98	447
Peace	0.99	0.99	0.99	429
accuracy			0.99	2246
macro avg	0.99	0.99	0.99	2246
weighted avg	0.99	0.99	0.99	2246

Sl. 4.2.2 Izvještaj o klasifikaciji – trening podaci

Classification Report Testni podaci

	precision	recall	f1-score	support
Bok	0.97	0.99	0.98	472
Dislike	0.99	0.99	0.99	444
Like	0.99	0.99	0.99	454
Okej	0.99	0.97	0.98	447
Peace	0.99	0.99	0.99	429
accuracy			0.99	2246
macro avg	0.99	0.99	0.99	2246
weighted avg	0.99	0.99	0.99	2246

Sl. 4.2.4 Izvještaj o klasifikaciji – testni podaci

Iz ovog izvještaja o klasifikaciji izvlačimo sljedeće zaključke o performansama modela:

Precision: Model ima visoku preciznosti za svaku klasu, što ukazuje na to da je većina predviđanja točna. Na primjer, za klasu "Bok", preciznost je 0.97, što znači da je 97% uzoraka koje je model označio kao "Bok" stvarno "Bok".

Recall: Recall za svaku klasu je također visok, što ukazuje na to da je model uspio pronaći većinu stvarnih instanci svake klase. Na primjer, za klasu "Okej", recall je 0.97, što znači da je model identificirao 97% stvarnih instanci "Okej".

F1-score: Visoke F1-score vrijednosti za svaku klasu sugeriraju da model dobro balansira između preciznosti i osjetljivosti (recall).

Support: Support predstavlja ukupan broj instanci u svakoj klasi.

Accuracy: Ukupna točnost modela je visoka (0.99), što ukazuje na to da je model točno klasificirao veliku većinu uzoraka. Macro avg: Srednja vrijednost preciznosti, osjetljivosti i F1-scorea po svim klasama. Visoke vrijednosti ukazuju na dobar balans između svih klasa.

Weighted avg: Srednja vrijednost preciznosti, osjetljivosti i F1-scorea po svim klasama, uzimajući u obzir broj instanci u svakoj klasi. Ova metrika sugerira da je model dobro raditi na klasama koje su manje zastupljene.

U cjelini, ovaj izvještaj ukazuje na vrlo dobre performanse modela s visokim točnostima, preciznostima i recallom za svaku klasu. Model dobro radi na razlikovanju među klasama i ima visoku sposobnost generalizacije.

4.3. Uvjeti trenirane mreže

Sl. 4.3.1 Detalji računala na kojem je trenirana mreža

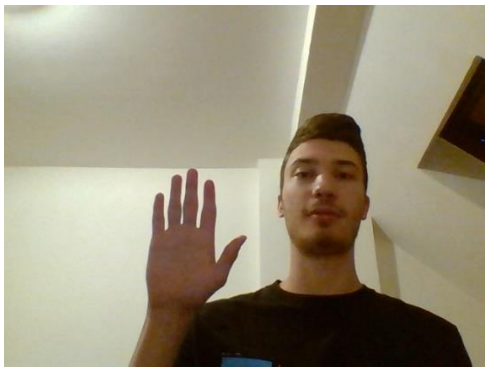
Hardware: Mreža je trenirana na računalo s procesorom Intel core i5-8300H, koji radi na frekvenciji 2.30GHz te 16GB radne memorije.

Software: Za realizaciju projekta korišten je Visual Studio Code i Python 3.10.3 na Windows 10, 64-bitnim operacijskim sustavom.

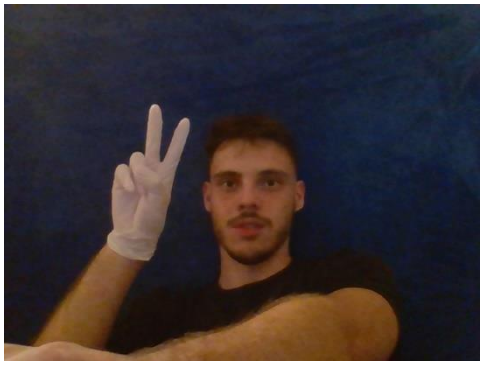
4.4. Testiranje u različitim uvjetima

U ovom poglavlju prikazujemo par testova u različitim uvjetima: različite osobe, osvjetljenje, udaljenost i kamera. Na slikama je prikazana matrica zabune i izvještaj o klasifikaciji. Za svaku gestu su napravljena četiri testa u različitim uvjetima. Detektirane su značajne točke ruku pomoću Mediapipe-a te pohranjene u novu .csv datoteku, koja je kasnije korištena za kreiranje matrice zabune.

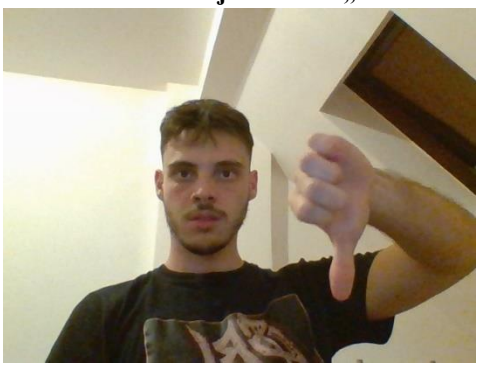
Sl. 4.4.1 Primjer iz testa „Bok“



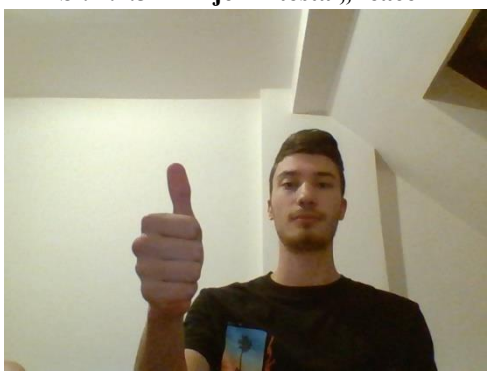
Sl. 4.4.2 Primjer iz testa „Bok“



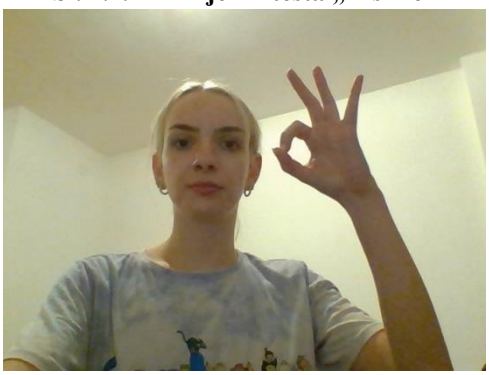
Sl. 4.4.3 Primjer iz testa „Peace“



Sl. 4.4.4 Primjer iz testa „Dislike“



Sl. 4.4.5 Primjer iz testa „Like“

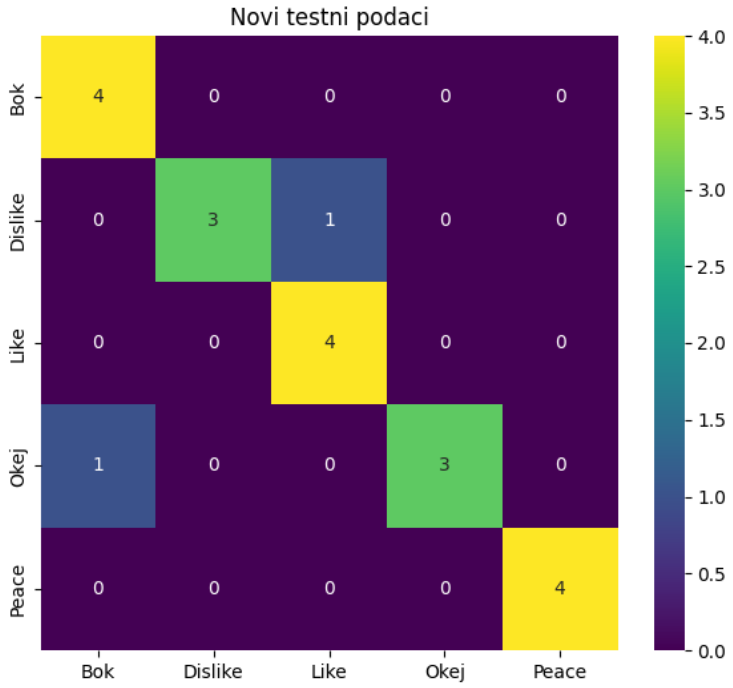


Sl. 4.4.6 Primjer iz testa „Okej“

Classification Report Novi testni podaci

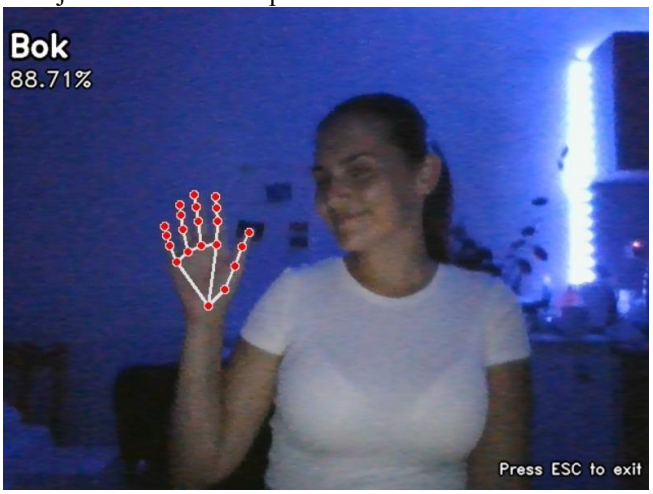
	precision	recall	f1-score	support
Bok	0.95	1.00	0.97	452
Dislike	0.99	0.99	0.99	439
Like	0.99	0.99	0.99	447
Okej	1.00	0.95	0.97	449
Peace	0.99	0.99	0.99	459
accuracy			0.98	2246
macro avg	0.98	0.98	0.98	2246
weighted avg	0.98	0.98	0.98	2246

Sl. 4.4.7 Izvještaj o klasifikaciji

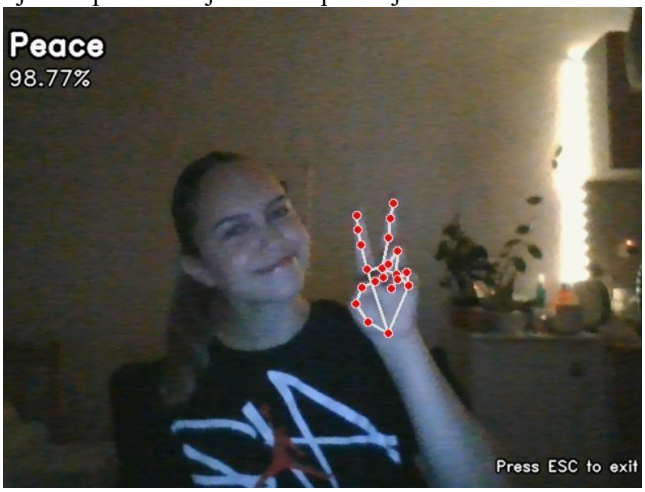


Sl. 4.4.8 Matrica zabune

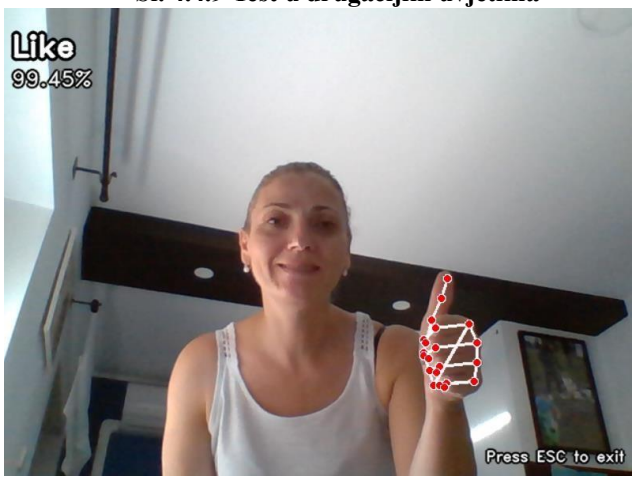
Na sljedećim slikama su prikazane različite osobe u različitim uvjetima pri testiranju unutar aplikacije.



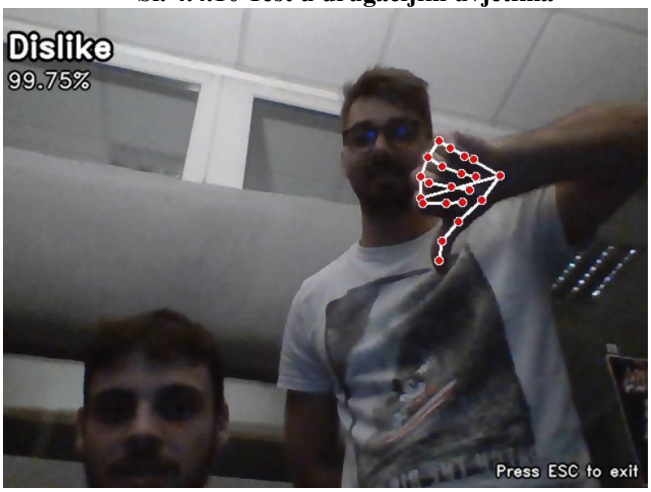
Sl. 4.4.9 Test u drugačijim uvjetima



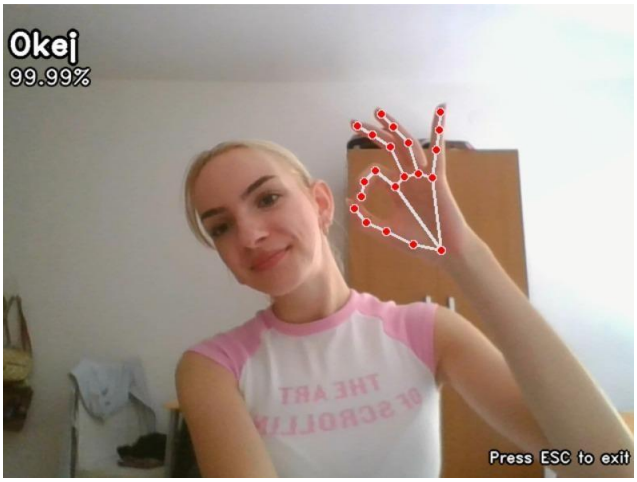
Sl. 4.4.10 Test u drugačijim uvjetima



Sl. 4.4.11 Test u drugačijim uvjetima



Sl. 4.4.12 Test u drugačijim uvjetima



Sl. 4.4.13 Test u drugačijim uvjetima

5. ZAKLJUČAK

Kroz ovaj projekt uspješno je implementiran sustav za prepoznavanje gesti ruke u stvarnom vremenu koristeći kombinaciju mediapipe biblioteke za praćenje ruku i neuronske mreže za klasifikaciju gesti. Ova tehnologija ima potencijal primjene u različitim kontekstima, kao što su interaktivne aplikacije, igre, nadzor uređaja i drugi interakcijski scenariji.

Pristup korištenjem relativnih koordinata točaka ruke omogućio je modelu da nauči prepoznavati geste neovisno o apsolutnoj poziciji ruke u kadru. To je omogućilo veću fleksibilnost u prepoznavanju gesti u različitim uvjetima.

Kvaliteta modela odražava se kroz postignutu točnost na testnim podacima, što je pokazatelj sposobnosti modela da generalizira prepoznavanje gesti na nepoznate primjere. Također, praćenje povijesti treninga omogućilo je bolje razumijevanje dinamike učenja modela.

Unatoč postignutim rezultatima, ovaj projekt također otvara vrata za daljnje poboljšanje. To uključuje optimizaciju hiperparametara, veće i raznovrsnije skupove podataka za trening, te dodatno fino podešavanje modela kako bi se postigla veća točnost i stabilnost u prepoznavanju gesti ruke.

S obzirom da smo podatke prikupljali sami, zadovoljni smo s performansama koje model postiže, i koliko dobro prepoznaje različite geste ruku.

6. LITERATURA

[1] Mediapipe gesture recognition task guide https://developers.google.com/mediapipe/solutions/vision/gesture_recognizer
[2] Color Spaces in OpenCV | Python https://www.geeksforgeeks.org/color-spaces-in-opencv-python/
[3] Feed Forward Neural Network https://deepai.org/machine-learning-glossary-and-terms/feed-forward-neural-network [4] OpenCV dokumentacija https://docs.opencv.org/
[5] Keras dokumentacija https://keras.io/api/