PRIMER: TRENIRANJE JEDNOSTAVNOG ALGORITMA MAŠINSKOG UČENJA ZA KLASIFIKACIJU U PYTHON-U

Profesor:

Jelena Ršumović

Autori:

Mateja Đokić
Nemanja Trifković

Predmet:

Programiranje

Datum izdavanja:

Beograd, septembar, 2022.

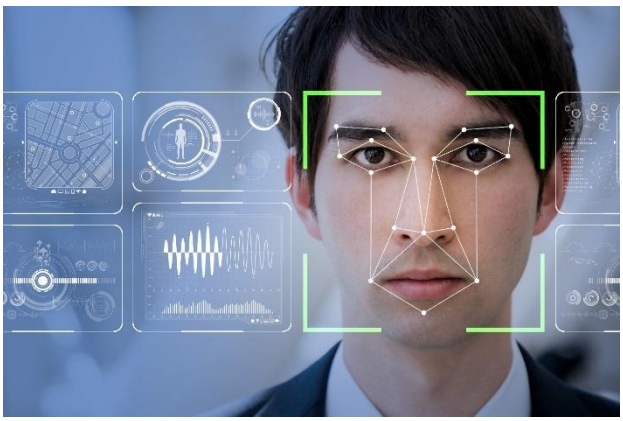
Sadržaj

alFaceRec

1. UVOD 3
2. TRENIRANJE PROGRAMA 4
   2.1. IMPLEMENTACIJA BIBLIOTEKA 4
   2.2. INICIJALIZACIJA 5
   2.3. PHOTO_IMAGE() FUNKCIJA 6
   2.4. UPDATE() FUNKCIJA 6
   2.5. MAIN() FUNKCIJA 7
   2.6. SCREENSHOT I KRAJ PROGRAMA 7
3. DATA 8
4. ZAKLUČAK 9
5. REFERENCE 10

1. UVOD

U ovom radu predstavili smo program za klasifikaciju lica geometrijskim oblicima koristeći python mašinsko učenje. Prepoznavanje lica postalo je popularna tema istraživanja nedavno zbog povećanja potražnje za bezbednošću kao i brz razvoj mobilnih uređaja. Postoji mnogo aplikacija na koje se može primeniti prepoznavanje lica kao kontrola pristupa, verifikacija identiteta, sigurnosni sistemi, sistemi za nadzor i mreže društvenih medija. Kontrola pristupa uključuje kancelarije, računare, telefone, bankomate, itd. Većina ovih obrazaca trenutno ne koristi lice kao standardni oblik odobravanja upisa, ali sa napredujući tehnologije u računarima, zajedno sa prefinjenijim algoritmima, prepoznavanje lica postaje sve popularnije u zameni lozinki i skenera otiska prsta. Od 11. septembra 2008. godine, zabrinutiji je naglasak na razvoju bezbednosnih sistema kako bi se osigurala bezbednost građana. Naime na mestima kao što su aerodromi i graničnim prelazima na kojima je neophodna verifikacija identifikacije, sistemi za prepoznavanje lica potencijalno imaju mogućnost ublažiti rizik.



Ovaj program se bazira na algoritmu pod nazivom “Haar Cascade”, nazvan po Alfredu Haar-u koji ga je pronašao 1909 godine. Njegov algoritam se bazira na elementima - features - koji su crno beli blokovi koji se koriste zajedno radi kreiranja nekih ponavljanja tj. paterna. Svaki od elemenata pojedinačno prolazi kroz sliku (ili kod videa u jednom frejmu) i traži mesto gde se slika poklapa sa elementom. Da bi ovaj program radio neophodno je da slika(frejm) bude crno-beo, pošto su i haar elementi takođe takvi. Zbog takvog preduslova tipa slike algoritam je puno jednostavniji, brži i zahteva manje resursa. Crno bela slika sadrži samo dve vrednosti (binarno vrednosna slika) crnu i belu, dok klasične RGB slike sadrže tri vrednosti koje stvaraju puno veću kompleksnost, i sporiji rad algoritma.

2. TRENIRANJE PROGRAMA

2.1. IMPLEMENTACIJA BIBLIOTEKA

Pre početka kodiranja programa, moramo uvesti neophodne Python biblioteke koje će nam biti potrebne, za rad našeg python programa.

U našem slučaju to su:

1. **OpenCV lib** => version 2.4 (open source external lib)

OpenCV (Open Source Computer Vision Library) je biblioteka softvera za računarsku viziju i mašinsko učenje otvorenog koda. OpenCV je napravljen da obezbedi zajedničku infrastrukturu za aplikacije računarskog vida i da ubrza upotrebu mašinske percepcije u komercijalnim proizvodima. U ovom programu koji smo kreirali ova biblioteka se koristi radi unošenja podataka iz .xml fajlova koje koristimo za prepoznavanje lica i drugih elemenata.

2. **Tkinter lib** => version 8.6 (native python lib)

Tkinter je standardna GUI biblioteka za Python. Python u kombinaciji sa ovom bibliotekom pruža brz i lak način za kreiranje GUI aplikacija. Tkinter pruža moćan objektno orijentisan interfejs za Tk GUI alat. Kreiranje GUI aplikacije koristeći Tkinter je lak zadatak. I ako nije toliko lep na prvi pogled poprilicno je efikasan pri bilo kakvom radu, tako da ne zauzima puno resursa niti je komplikovan za učenje. Ovde smo korisili Tk Gui da bi na ekranu pokazali direktan feed sa korisnikove veb kamere. Takođe Tk dolazi uz instalaciju Python-a tako da je smatramo python native biblilotekom.

3. **PyAutoGui** => version 0.9 (open source external lib)

PyAutoGUI je python biblioteka za automatizaciju koja se koristi za kliktanje, prevlačenje, skrolovanje, pomeranje itd. U ovom programu je korišćena ova biblioteka radi automatizacije slikanja ekrana radi kasnije mogućeg korišcenja tih slika da bi se program unapredio i poboljšao. Takođe bi mogla da se napravi implementacija automatskog konvertovanja tih slika u podatke koje bi mogli ponovo da koristimo u programu i tako ga učimo da bude sto precizniji.

```
from tkinter import NW, Tk, Canvas, PhotoImage
from tkinter import ttk
from tkinter import *
import cv2
import pyautogui
```

2.2. INICIJALIZACIJA

Na početku programa posle svih unesenih biblioteka moramo inicijalizovati razne promenljive i podatke koje ćemo koristiti unutar tog programa.

Prvo inicijalizujemo prostor u kome ćemo prikazati sliku koja će biti filtriran kroz nas “Haar Cascade” algoritam. Uz Tkinter inicijalizujemo prozor nazivamo ga “Face Recognition”, stavljamo da njegova minimalna širina i visina bude širina i visina naše slike (tj. direktnog feed-a iz veb kamera).

Sada kreiramo jednostavne promenljive koje ćemo koristiti da bi razlikovali različite objekte po bojama. Jednostavnim imenovanjem promenljivih i zadavanja njihovih tro-bajtnih vrednosti (tip bajt 0 - 255) tj. RGB vrednosti RGB(byte, byte, byte).

Kasnije u ovom dokumentu ćemo više reći o ovom delu programa, ali je najbitnije da u ovom delu unosimo srž algoritma, podatke koje koristimo za prepoznavanje.

Ovaj deo inicijalizacije je jako bitan pri obelezavanju traženih objekata, pošto u python-u ne možemo da specifikujemo tip praznih promenljivih ovde moramo promenljivima da zadamo vrednosti (koje su [[0, 0, 0, 0]]) da bi mogli da odredimo tip podataka.

Peti deo inicijalizacije je glavni deo uzimanja slike (frejmova) sa veb kamere da bi to posle ta slika mogla da se pročita i provuče kroz sve podatke radi traženja željenih objekata.

```
# Tkinter window init
root = Tk()
root.title("Face Recognition")
root.resizable(True,True)
root.minsize(640, 480)
root.config(background = "white")
```

```
# COLORS
color_red = (0, 0, 255)
color_green = (0, 255, 0)
color_blue = (255, 0, 0)

# CORD INIT
front_face_cord = [[0,0,0,0]]
front_face1_cord = [[0,0,0,0]]
front_face2_cord = [[0,0,0,0]]
eye_cord = [[0,0,0,0]]
smile_cord = [[0,0,0,0]]
```

2.3. PHOTO_IMAGE() FUNKCIJA

-je funcija dizajnirana radi unosa slike tj. prenosa slike od veb kamere u tip slike koju Tkinter biblioteka može da prikaže i manipuliše. Funkcija uzima sliku koja je dobijena kroz čitanje jendog frejma od snimka veb kamere, gde od te slike uzima visinu i širinu, kao i druge vizuelne podatke, koje posle koristi da kreira novu sliku kompatibilnu sa Tkinter bibliotekom.

```
# photo input
def photo_image(img):
    h, w = img.shape[:2]
    data = f'P6 {w} {h} 255 '.encode() + img[..., ::-1].tobytes()
    return PhotoImage(width=w, height=h, data=data,format='PPM')
```

2.4. UPDATE() FUNKCIJA

1. U update() funckiji koristimo photo_image() funckciju da dobijemo sliku koju funkcijom cv2.flip() okrećemo za 180 stepeni horizontalno. Kreiramo novu sliku sa cv2.cvtColor() funkcijom gde dodajemo parametar “BGR2GRAY” da sliku pretvorimo u monohromatski prikaz.
2. U ovom delu update() funkcije u naše već inicijalizovane promenljive za kordinate objekata koje tražimo, unosimo pomoću funkcije detectMultiScale() vrednosti tj. kordinate naših objekata da bi ih posle obeleželi na ekranu. Kroz ovu funkciju provlačimo našu monohromatsku sliku, koja mora biti takva da bi radio program, da bi dobili te vrednosti.
3. Obeležavanje objekata koje tražimo je u stvari proces gde mi na kordinama koje smo dobili iz prethodne funckcije (detectMultiScale()) iscrtavamo kvadrat koji ima boju u odnosu na objekat koji obeležava. Zbog nemogućnosti preglašavanja detectMultiScale() funkcije jednu na drugu postoje tri različite detekcije lica sve obeležene zelenim kvadratim. Takođe crvenim kvadratima su obeležene oči pronađenog lica, a i profil lica tj. u slučaju da je desna ili leva strana lica čoveka okrenuta prema kameri program će nacrtati plavi kvadrat oko profila te osobe.

```
# update func for canvas and image creation and edit
def update():
    ret, img = cap.read()
    img = cv2.flip(img, 1)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # SCEEN OBJECT DETECTION
    front_face_cord = front_face_data.detectMultiScale(gray)
    front_face1_cord = front_face1_data.detectMultiScale(gray)
    front_face2_cord = front_face2_data.detectMultiScale(gray)
    eye_cord = eye_data.detectMultiScale(gray)
    profile_cord = profile_data.detectMultiScale(gray)

# DRAWING RECTANGLES
    # triple face object display
    for (x, y, w, h) in front_face_cord:
        cv2.rectangle(img, (x, y),(x+w,y+h), color_green, 2)
    for (x, y, w, h) in front_face1_cord:
        cv2.rectangle(img, (x, y),(x+w,y+h), color_green, 2)
    for (x, y, w, h) in front_face2_cord:
        cv2.rectangle(img, (x, y),(x+w,y+h), color_green, 2)

    # eye and profile object display
    for (x, y, w, h) in eye_cord:
        cv2.rectangle(img, (x,y),(x+w,y+h), color_red, 2)
    for(x, y, w, h) in profile_cord:
        cv2.rectangle(img, (x, y), (x+w, y+h), color_blue , 2)
    if ret:
        photo = photo_image(img)
        canvas.create_image(0, 0, image=photo, anchor=NW)
        canvas.image = photo
    root.after(15, update)
```

2.5. MAIN() FUNKCIJA

Main() funkcija tehnički ne postoji u ovom programu, barem ne napisana tako, ali se ovaj deo smatra kao Main() pošto sve delove koda dovodi zajedno do krajnjeg postupka. U ovoj funkciji uz Tk kreiramo CANVAS u kome se iscrtava slika koju dobijamo od kamere. Ovde se takođe aktivira funkcija update(), odmah posle funkcije koja omogućava da se canvas sklopi i kreira => canvas.pack(). Posle update() funkcije se izvršava root.mainloop() koja loop-je root => ekran , window, gde je naša slika i prikazana. Na kraju mi oslobađamo naš frejm funkcijom cap.releas(), posle čega ponovo funkcija cap.VideoCapture(0) može da uzme novi frejm.

```
# main part of the program
canvas = Canvas(root, width=640, height=480, background =
"black")
canvas.pack()
update()
root.mainloop()
cap.release()
```

2.6. SCREENSHOT I KRAJ PROGRAMA

Posle tog neverovatnog dugog dela koda imamo jednostanu implementaciju slike ekrana sa PyAutoGui bibliotekom. Mi u varijablu stavljmo sliku ekrana funkcijom screenshot() i onda tu sliku (s) sa određenim imenom, stavljamo u određenu lokaciju specifikovanu u finkciji s.save(r'path/image.jpeg'). Tip slike nije toliko bitan, mada je korisna slika sto većeg kvaliteta. Ovaj deo programa je totalno neuticajan na ostatak programa, bitan je samo zbog mogućih budućih implementacija gde bi se ove slike mogle koristiti radi unapređivanja polaznih podataka sa kojima radimo. Ostao nam je još samo jednostavan print() koji se izvršava kada se ceo ovaj suvoparni kod izvrši.

```
# taking and saving a screenshot with pyAutoGui
s = pyautogui.screenshot()
s.save(r"screenshots/s.jpeg")

# just to see if the code went
#all the way to the end with no mistakes
print("Code Complete")
```

3. DATA

⇒ Ovo je malo bolji uvid u podatke zbog kojih ovaj program i radi.
⇒ U ovom programu su korišćeni .xml fajlovi.
⇒ XML (Extensible Markup Language) fajlovi su fajlovi koji se tipično koriste za opisivanje i skupljanje nekih podataka na jednom mestu.
⇒ Mi smo uneli ove podatke kroz .xml fajlove u promenljive koje se posle kroz funkcije koje svaki piksel slike ili svaku grupu od po par piksela provlače tj. upoređuje sa podacima u .xml fajlovi i ta podudaranja vraćaju i time daju poziciju traženih objekata.
⇒ Kao što je ranije naglašeno zbog korišćenja “Haar Cascade” algoritma i njegovih monohromatskih feature-a slika mora biti takođe monohromatska zbog malo pre rečenog piskel po piksel upoređivanja sa tim feature-ima.
⇒ Na kraju programa mi uzimamo sliku ekrana koju npr. ovim sajtom: https://products.aspose.app/svg/encoding/jpg-to-xml , možemo pretvoriti u podatke koje u budućnosti možemo koristi da unapredimo program i celukupni data set.
⇒ Mi smo koristili svega pet različitih .xml fajlova, data setova.
⇒ Program bi funkcionisao skoro isto bez dva data seta ali smo zbog boljeg objašnjavanja želeli da stavimo i ta dva.

⇒ Ti data setovi su:

1 front_face_default.xml => prepoznaje lice

2. front_face_alt.xm => prepoznaje lice (alternativni, malo drugačiji podaci)

3. front_face_alt2.xm => prepoznaje lice (alternativni, malo drugačiji podaci)

4. profile_face.xml => prepoznaje levu ili desnu stranu ljudskog lica - profil

5. eye.xml => prepoznaje ljudsko oko

```
# DATA
front_face_data =
cv2.CascadeClassifier(r'data/front_default.xml')
front_face1_data = cv2.CascadeClassifier (r'data/front_alt.xml')
front_face2_data = cv2.CascadeClassifier (r'data/front_alt2.xml')
eye_data = cv2.CascadeClassifier(r'data/eye.xml')
profile_data = cv2.CascadeClassifier(r'data/profile_face.xml')
```

4. ZAKLUČAK

Kreirali smo visoko performansno, skalabilno, okretno i jeftino prepoznavanje lica sistema. Celokupan kod zadatka prikazan na dokumentu može se pogledati na ovom linku.

U pregledu poslednjeg istraživanja, otkrili smo da se mnogi pristupi mogu koristiti za prepoznavanje lica da svaki metod ima različite prednosti i nedostatke, kao što su lokalni, globalni i hibridni metod. Postoje 2 tipa slike za tehniku prepoznavanja lica: nepokretna slika i video slika (sekvenca nepokretnih slika). Međutim, pronašli smo neke probleme u sistemu prepoznavanja lica kao što su:

(1) problem sa pozom

(2) problem sa osvetljenjem: zbog izvorne slike ima svetlosnog stanja ili različitih varijacija osvetljenja i gledanja.

(3) problem okoline: u stvari, kretanje i izraz ne mogu da kontrolišu prirodnu sliku.

(4) 3D problem: zbog toga što se u 3D slici mora koristiti više prostora za skladištenje, mnogo promenljivih, mala brzina i nedostatak testiranja baze podataka lica.

5. REFERENCE

[1] https://bit.ly/3DESDZi

[2] https://bit.ly/3Sj5vs1

[3] https://bit.ly/3R3ZheC

[4] https://bit.ly/2ntnwrv

[5] https://bit.ly/3dt97cf

[6] https://bit.ly/3C6zXAJ

[7] https://bit.ly/3BZaQQd

[8] https://bit.ly/3xEx735