Predefinisani projekat 3

Soft Kompjuting 2019/2020

Student:

Branko Gluvajić, SW65-2016

Zadatak:

Video zapis sadrži plato braon boje i ljude koji se kreću. Izvršiti prebrojavanje ljudi koji se u barem jednom trenutku nađu na platou.

Kreirati rešenje koje će ostvariti najmanji **mean absolute error** (MAE).

Pristup i način rešavanja:

Dato nam je 10 različitih video snimaka na kojima je potrebno prebrojati pešake koji se nalaze na platou braon boje.

Zaključujemo da će nam biti potrebna Python biblioteka koja omogućuje transformacije / obradu slika. Zbog toga odmah importujemo OpenCV biblioteku i krećemo sa radom.

Rešenje kroz kod:

Ovaj kod sadrži samo dve ručno definisane funkcije. Jedna će kroz matematičku jednačinu prave da detektuje ukoliko se pešak našao na platou, dok druga obrađuje snimak i vraća ukupan broj prebrojanih pešaka.

Za sada ćemo se skoncentrisati na drugu funkciju.

Prvo učitavamo video snimak uz pomoć VideoCapture funkcije OpenCV biblioteke:

```
video = cv2.VideoCapture(video_path)
```

Inicijalno definišemo **brojač pešaka** na vrednost **0**.

Implementiramo while petlju, koja traje sve dok se video ne završi.

Prvi korak u petlji predstavlja linija:

```
ret, frame = video.read()
```

Ret je promenljiva koja proverava da li je učitavanje video snimka proteklo bez grešaka. Ukoliko postoji greška, obustavlja se rad na video snimku, a ukoliko ne postoji – nastavljamo sa obrađivanjem.

Frame promenljiva je tipa **numpy.ndarray** i predstavlja frejmove video snimka.

Pikseli kolorizovanih frejmova su RGB tipa, što predstavlja 3 vrednosti.
Funkcijom OpenCV biblioteke cvtColor konvertujemo kolorizovane frejmove u "sivu" nekolorizovanu sliku:

```
# Converting RGB to GRAY_IMAGE
gray_image = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
```

Na taj način svaki piksel više nije RGB tipa, pa sada pikseli imaju samo jednu vrednost (smanjeno sa 3 na 1). Ta vrednost grubo rečeno predstavlja „nijansu sive boje".

Gaussian Blur

Gaussian Blur predstavlja primenu matematičkih funkcija na sliku, gde je rezultat "blurovanje" tj. izobličenje slike.
Ovo se najpre koristi u radu sa slikama, gde slike sadrže šumove, slikane su u mraku, sadrže takozvani "noise".

Na prikazanoj slici ispod se može videti rezultat korišćenja Gaussian Blur-a u manjoj meri i većoj meri.

The difference between a small and large Gaussian blur

U našem projektu ćemo iskoristiti baš Gaussian Blur, da bismo otklonili smetajuće šumove na frejmovima video snimka.

```
# Blurring sharp lines in image
gray_image = cv2.GaussianBlur(gray_image, (19, 19), 1)
```

Na početku smo definisali brojač pešaka na platou, ali smo isto tako definisali i početni frejm video snimka.

Ovaj frejm je u startu „None“ vrednosti, ali u prvoj iteraciji kroz frejmove dobija inicijalnu vrednost frejma koji je konvertovan u Gray Image, i nad kojim je izvršen Gaussian Blur. Ovaj inicijalni frejm će zadržati vrednost do kraja video snimka.

Ovaj korak je potreban da bi mogli da upoređujemo inicijalni frejm sa narednim frejmovima u video snimku. Svaki piksel upoređujemo. Ukoliko se pikseli ne razlikuju, ili se razlikuju u vrlo maloj meri – znamo da se figura u tom delu video snimka ne pomera, tj. statična je.

Ukoliko je razlika velika, ovo nam govori da je u našem slučaju verovatno pešak prešao na mestu gde se ovi pikseli nalaze.

Razliku u pikselima radimo na sledeći način, pozivanjem **absdiff** metode, koja oduzima vrednosti piksela prosleđenih frejmova:

```
# Calculates difference in pixelization between initial frame and current frame.
difference = cv2.absdiff(initFrame, gray_image)
```

NAPOMENA:

Ovo je jedan način rada gde upoređujemo svaki trenutni frejm sa inicijalnim frejmom. Drugi način bi bio da upoređujemo svaki trenutni frejm sa prethodnim frejmom.

Kada izvršimo ovu funkciju, dobijamo za svaki piksel vrednost koja predstavlja razliku piksela kroz frejmove.

Potrebno je da definišemo „**threshold**“ tj. prag gde ćemo sve vrednosti ispod datog praga postaviti na 0 (sve vrednosti ispod praga grupišemo), a sve veće vrednosti, koje prelaze dati prag postaviti na 255. Ovim razgraničavamo podatke malih i velikih vrednosti tj. težina.

Ako je potrebno vizualizovati ovaj način rada, možemo uzeti za primer da su sve male vrednosti crni pikseli, a sve velike vrednosti beli pikseli. Na ovaj način možemo dobiti u potpunosti crno platno, gde će naši pešaci biti bele boje (u direktnom prevodu).

Funkcijom cv2.dilate „uvećavamo“ tačke koje smo prethodno postavili na veću vrednost, jer su imale veću težinu. Prosleđujemo dati prag (threshold), kernel i broj iteracija u kojima će dilate funkcija da „uveća“ tačke.

```
# Calculates Threshold for Difference
threshold = cv2.threshold(difference, 20, 255, cv2.THRESH_BINARY)[1]
threshold = cv2.dilate(threshold, None, iterations=2)
```

Sada je sve spremno da pronađemo date konture na video snimku, tj. na svakom frejmu. Ovo ćemo uraditi uz pomoć funkcije cv2.findContours kojoj prosleđujemo trenutni threshold koji smo definisali, a i metode za pronalazak kontura.
Uzeli smo cv2.RETR_LIST i cv2.CHAIN_APPROX_SIMPLE parametre.

```
contours = cv2.findContours(threshold, cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)[0]
```

Dolazimo do dela gde se fokusiramo na konture:
For petljom prolazimo kroz sve konture na datom frejmu.
Ukoliko je površina date konture manja od **90***, prelazimo na sledeću konturu, jer ovo znači da je kontura suviše male površine da bi predstavljala bitan podatak – pešaka.

*Konkretno sam postavljao vrednosti na 200, 150, 100 i 90 da bi uvideo najoptimalniji prag za površinu.

Površina	MAE
200	**7.7**
150	**6.4**
100	**5.1**
90	**5.0**

Ukoliko je kontura površine veće od 90, za nju iscrtavamo figuru oblika pravougaonika.

BoundingRect funkcija prima kao ulazni parametar datu konturu, a vraća nam podatke X, Y, Širinu pravougaonika, Visinu pravougaonika.

Tačka (x, y) predstavlja donje-levu tačku pravougaonika, pa samim tim možemo izračunati i dijagonalnu, gornje-desnu na sledeći način:

```
(x, y, width, height) = cv2.boundingRect(contour)
# Bottom Left Point of a rectangle.
bottomLeftPoint = (x, y)
# Upper Right Point of a rectangle.
upperRightPoint = (x + width, y + height)
```

Sada imamo dovoljno podataka da nađemo centar pravougaonika, na osnovu koga ćemo moći da ispratimo da li je pešak prešao braon plato, jer je dati centar upravo sam pešak.

```
# Calculating Center Point on X scale and Y scale.
rectangleCenterX = (x + x + width) / 2
rectangleCenterY = (y + y + height) / 2
```

Ovde se prvi put upotrebljava jedina preostala „ručno-definisana“ funkcija koje detektuje prelazak pešaka preko platoa.

Definisali smo granicu koja ima vrednost 250 na Y osi. Upotrebom matematičke funkcije Y – Y1, gde nam je Y1 = 250, a Y vrednost „pešaka“ na Y osi, možemo da vidimo da li će razlika ove dve vrednosti preći ispod 1. Ukoliko pređe ispod 1, to znači da je pešak prešao datu granicu, a samim tim je i zakoračio na plato.

Zaključak:

Ovim načinom rada sam dobio MAE oko 5.0. Podešavanjem parametara za definisanje praga (threshold), pomeranjem granice za minimalnu površinu, i pomeranjem granice na Y osi sam dobijao rezultate koji se kreću između 9.1 i 5.0.

Nisam mogao da pronađem rezultat optimalniji od ovog, iako se on svakako može postići detaljnijim implementiranjem funkcija OpenCV biblioteke.

Ono što je najviše uticalo na prebrojavanje je situacija gde se ljudi kreću u grupama i ljudi koji uđu na plato, ali ne pređu našu granicu za prebrojavanje.