Računalni model strojnog vida za detekciju ljudi

Grabušić, Emil

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Emil Grabušić

Zagreb, 2023

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Mentor:
Dr. sc. Tomislav Stipančić, dipl.ing.

Student:
Emil Grabušić

Zagreb, 2023

Izjavljujem da sam ovaj rad izradio samostalno koristeći znanje stečeno tijekom studija i navedenu literaturu.

Zahvaljujem se mentoru dr. sc. Tomislavu Stipančiću na pruženoj pomoći, ugodnoj suradnji, povjerenju i strpljenju tijekom izrade ovog rada.

Posebno se zahvaljujem svojoj obitelji na podršci i razumijevanju tokom studiranja te se zahvaljujem svojim prijateljima koji su uvijek bili tu i vjerovali u moj uspjeh te mi učinili studentske dane malim nezaboravnim dijelom mog života.

Emil Grabušić

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE
Središnje povjerenstvo za završne i diplomske ispite
Povjerenstvo za završne i diplomske ispite studija strojarstva za smjerove:
proizvodno inženjerstvo, računalno inženjerstvo, industrijsko inženjerstvo i menadžment, inženjerstvo materijala i mehatronika i robotika

Sveučilište u Zagrebu Fakultet strojarstva i brodogradnje	
Datum	Prilog
Klasa: **602 – 04 / 23 – 6 / 1**	
Ur.broj: **15 - 1703 - 23 -**	

ZAVRŠNI ZADATAK

Student: **Emil Grabušić** JMBAG: **0035223552**

Naslov rada na hrvatskom jeziku: **Računalni model strojnog vida za detekciju ljudi**

Naslov rada na engleskom jeziku: **Machine vision computing model for human detection**

Opis zadatka:

Algoritmi za detekciju ljudi na slikama ili s video materijala nalaze se u domeni strojnog i računalnog vida te uključuju nekoliko koraka provedbe. Mogu biti temeljeni na jednostavnijim klasifikatorima koji su unaprijed trenirani koristeći različite metode. Često čine podlogu za razvoj kompleksnijih algoritama kao što su oni za predviđanje akcija ili namjera ljudi.

U radu je potrebno:

- proučiti metode strojnog vida za detekciju ljudi na slikama ili video materijalima te se upoznati s OpenCV programskom bibliotekom za strojni vid,
- upoznati se te objasniti princip rada HOG algoritma (engl. Histograms of Oriented Gradients),
- razviti računalni model za detekciju ljudi koji koristi HOG algoritam u programskom jeziku Python,
- evaluirati rad modela kroz eksperimente u sklopu Laboratorija za projektiranje izradbenih i montažnih sustava te dati kritički osvrt.

U radu je potrebno navesti korištenu literaturu i eventualno dobivenu pomoć.

Zadatak zadan:

30. 11. 2022.

Zadatak zadao:

Doc. dr. sc. Tomislav Stipančić

Datum predaje rada:

1. rok: 20. 2. 2023.
2. rok (izvanredni): 10. 7. 2023.
3. rok: 18. 9. 2023.

Predviđeni datumi obrane:

1. rok: 27. 2. – 3. 3. 2023.
2. rok (izvanredni): 14. 7. 2023.
3. rok: 25. 9. – 29. 9. 2023.

Predsjednik Povjerenstva:

Prof. dr. sc. Branko Bauer

SADRŽAJ

POPIS SLIKA

POPIS TABLICA

SAŽETAK

U teorijskom dijelu temeljito je objašnjen HOG algoritam koji se u ovom radu uz pomoću SVM klasifikatorom koristi u svrhu detekcije ljudi. U prvom dijelu je objašnjen sam postupak dobivanja histograma orijentiranih gradijenata te je prikazana povijest samog algoritma. Prikazane su i neke fotografije koje na slikovit način prikazuju samo razumijevanje algoritma. Zatim je objašnjen SVM klasifikator da bi se na kraju zaokružila priča i potkrijepila sva teorija sa praktičkim dijelom gdje se pomoću OpenCV programske biblioteke na video materijalu provodi detekcija ljudi. Na samom kraju izložen je kratak osvrt na rad samog algoritma.

Ključne riječi: HOG, Python, histogram, SVM

SUMMARY

In the theoretical part, the HOG algorithm is thoroughly explained, which is used in this paper with the help of the SVM classifier for the purpose of human detection. In the first part, the process of obtaining the histogram of oriented gradients is explained, and the history of the algorithm itself is presented. Some photos are also shown that graphically show the understanding of the algorithm itself. Then the SVM classifier was explained in order to round off the story and substantiate all the theory with a practical part where the detection of people is carried out on video material using the OpenCV program library. At the very end, a brief overview of the operation of the algorithm itself is presented.

Keywords: HOG, Python, histogram, SVM

1. Uvod

HOG (engl. Histograms of Oriented Gradients) spada u algoritme koji se koriste u domeni strojnog odnosno računalnog vida te pomažu u detekciji prepoznavanja dvodimenzionalnih ili trodimenzionalnih predmeta kao što su naprimjer ljudsko lice.

Računalni vid je područje koje koristi različite metode za prikupljanje informacija vezanih za sliku s ciljem dobivanja numeričkog ili simboličnog zapisa. U današnje vrijeme računalni vid ima sve veći spektar primjene. Kao područje primjene možemo ga naći u autoindustriji gdje se koristi u svrhu autonomne vožnje te u različitim kućanskim robotima koji se koriste računalnim vidom za bolje snalaženje u prostoru te prepoznavanje prepreka. Isto tako prisutan je u našim mobilnim uređajima koje mnogi put koristimo za zabavu i fotografiranje koje prilikom uporabe kamere koristi isto tako tehnologiju prepoznavanja lica. U današnje vrijeme ova tehnologija se još uvijek razvije te pronalazi svrhu primjene u različitim područjima, a cilj joj je digitalno razumijevanje slike i mogućnost elektroničke predodžbe ljudskog vida.

Rad je fokusiran na objašnjenje HOG algoritma u domeni računalnog vida koji se koristiti za detekciju ljudi. Algoritam se implementira pomoću OpenCV programske biblioteke u programskom jeziku Python. U početku rada se objašnjava opći princip rada algoritma te se naposlijetku primjenjuje sam algoritam u programskom jeziku kako bi se detektirali ljudi na slikama i videozapisima.

2. HOG (Histograms of Oriented Gradients)

2.1 Povijest HOG algoritma

Prva istraživanja na koja su temeljena na principu rada HOG algoritama danas opisao je u svom patentu Robert K. McConell 1986. godine. Kasnije se 1994. godine koncept upotrijebio od strane Mitsubishi Electric Research Laboratories. No u proširenoj je upotrebi od 2005. godine kada su Navneet Dalal i Bill Triggs objavili svoj znanstveni rad na temu HOG detekcije. U svom radu usredotočili su se na otkrivanje pješaka na slikama, a kasnije su kroz pokuse proširili primjenu na otkrivanje ljudi u videozapisima, kao i životinja te automobila.[1]

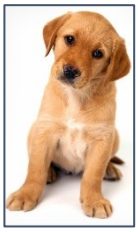
2.2 Princip rada HOG algoritma

HOG koristi se u računalnom vidu i obradi slike u svrhu detekcije objekta na temelju prebrojavanja učestalosti pojave orijentacija gradijenata u dijelovima slike koji su predstavljeni pomoću mreže ćelija.

Algoritam se fokusira na oblik ili strukturu objekta na način da identificiramo piksele na rubu objekta te smjer kretanja ruba pomoću magnitude i orijentacije gradijenta koji se računa unutar manjih dijelova slike. U nastavku ćemo vidjeti na koji način ćemo dobiti histograme orijentiranih gradijenata odnosno prvo na koji način se računa magnituda i usmjerenje gradijenta. Tako ćemo sada proći kroz dijelove u kojima ćemo objasniti princip rada samog algoritma.

2.2.1 Učitavanje fotografije

Postupak formiranja HOG algoritma objašnjen je na primjeru fotografije. Za ovaj primjer se odabrana fotografija dijeli na kvadratiće veličine 8x8 piksela pa se prema tome za početak mijenja originalna veličinu fotografije sukladno veličini kvadratića. U ovom primjeru veličina fotografije mijenja se u 64x128 piksela. Fotografija može biti bilo koje veličine samo da je veličina pogodna za analizu koja se provodi pomoću HOG algoritma. Jedini parametar koji može ovisiti o veličini fotografije je samo vrijeme koje je potrebno za analizu fotografije. U nastavku je vidljiva fotografija koja je odabrana za primjer u veličini od 64x128 piksela.

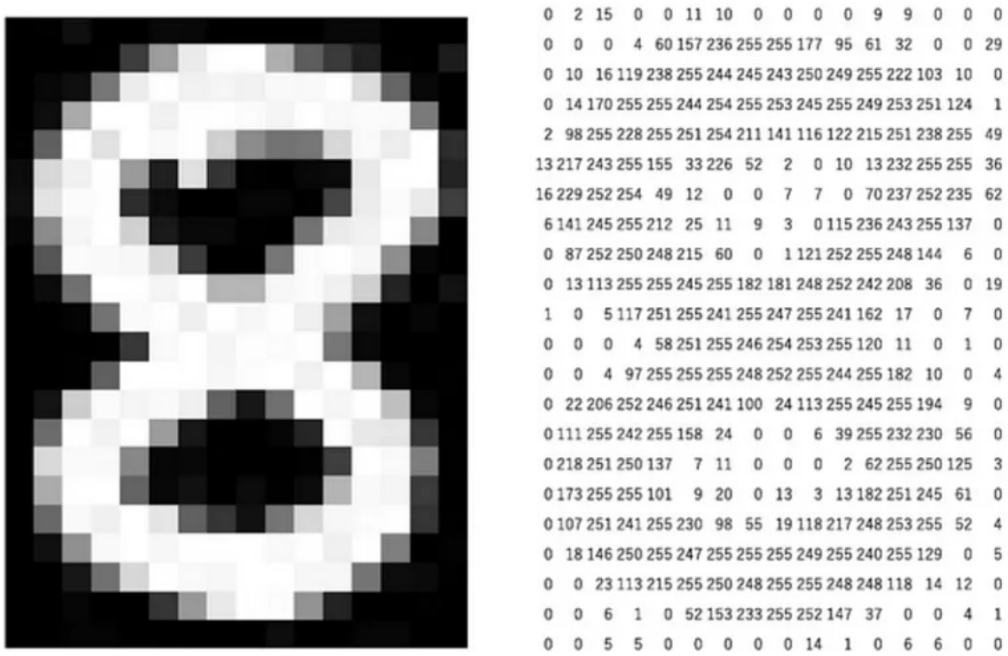


Slika 1. Primjer slike 64x128 [2]

Sljedeći korak je računanje vrijednosti magnitude gradijenata u x i y smjeru te orijentacije odnosno kuta između x i y gradijenta za svaki korak. U sljedećem poglavlju objašnjen je princip kojim su definirane navedene vrijednosti.

2.2.2 Izračun gradijenata

Za izračun gradijenata svakog pojedinog piksela potrebno je poznavati intenzitete piksela. Za crno-bijelu sliku oni imaju samo jednu vrijednost koja je iznosa od 0 do 255. Za vrijednost 0 piksel je crne boje dok se povećavanjem intenziteta pojačava i njegova svjetlina te tako on postaje sve svjetliji i svjetliji sve dok za vrijednost od 255 ne postane potpuno bijel. U nastavku je prikazana slika kada bi ona bila opisana samo vrijednostima pojedinih piksela.



Slika 2. Slika i način na koji računalo pohranjuje sliku [2]

Pošto je fotografija koja je odabrana za primjer na kojem će biti definiran princip rada HOG algoritma, u boji, ona sadrži po tri vrijednosti za svaki piksel. Te vrijednosti predstavljaju intenzitet prisutnosti pojedinih boja u pikselu i svaka od tih triju vrijednosti za pojedinu boju, R (red), G (green), B (blue), može sadržati vrijednosti u rasponu od 0-255. Za daljnji postupak potrebno je iz te tri matrice vrijednosti intenziteta dobiti jedinstvenu matricu vrijednosti za svaki piksel. Kako bi se dobila jedinstvena odnosno glavna vrijednost piksela koristi se jedna od više metoda.

Prva metoda koja se može koristiti je metoda prosjeka koja je sasvim trivijalna. Jednostavnim zbrojem svih triju vrijednosti i dijeljenjem sa brojem tri dobi se naša tražena vrijednost intenziteta piksela. Ova metoda sadrži malu grešku jer svakoj boji pridodaje istu

vrijednost što u realnom svijetu i nije slučaj. Zna se da je oko pod utjecajem okoline u kojoj se nalazi najosjetljivije na zelenu boju zatim crvenu i tek onda plavu. Zato bi se trebalo primijeniti težinsko značenje svake boje.

$$grayscale = \frac{R + B + G}{3} \quad 1)$$

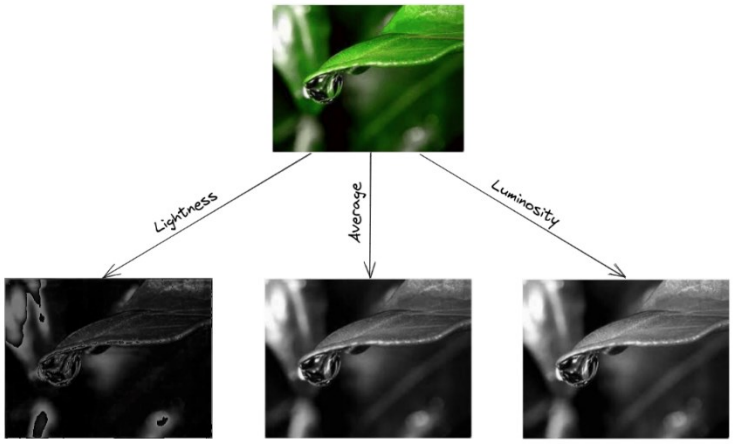
Druga metoda je metoda luminoziteta. Kod ove metode jedinstvena vrijednost piksela dobije se provođenjem već unaprijed zadane formule. Ova metoda je puno preciznija u odnosu na prošlu zbog već navedenih razloga gdje svakoj boji posebno pridodaje njezina težina.

$$grayscale = 0{,}3 * R + 0{,}59 * G + 0{,}11 * B \quad 2)$$

Još jedna od metoda je metoda lakoće. Ova metoda sadrži najveću grešku jer ne uzima u obzir jednu od triju komponenti. Metoda lakoće zasniva se na tome da se izračuna prosjek maksimalne i minimalne vrijednosti unutar RGB skupa od triju vrijednosti. Kao što je rečeno rekli ova metoda sadrži najveću grešku.

$$grayscale = \frac{\max(RGB) + \min(RGB)}{2} \quad 3)$$

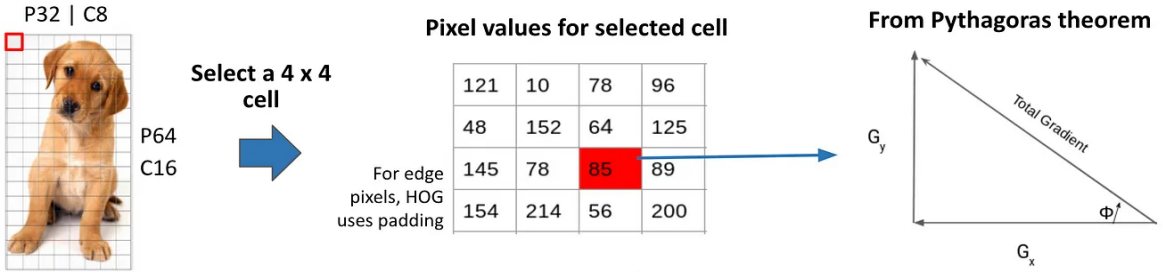
U nastavku je vidljiv izgled pretvorbe RGB formata slike u crno-bijeli format pomoću triju navedenih metoda.



Slika 3. Pretvorba RGB u B&W fotografiju [3]

Nakon dobivenih jedinstvenih vrijednosti za svaki piksel slijedi izračunavanje magnituda i smjerova pojedinih gradijenata. Gradijenti su zapravo promjene u x i y koje se mogu prikazati sa vektorom koji ima svoju magnitudu i smjer. Čim su promjene izražajnije magnituda je veća, a smjer vektora označuje smjer u kojem se promjena događa. Sada slijedi fokusiranje na izračun gradijenata što je i tema ovog poglavlja, a nakon toga slijedi prikaz magnituda gradijenata i usmjerenja u obliku histograma.

Ako se uzme mali dio slike u ovom slučaju ćelija od 4x4 piksela kao na slici (Slika 4.) može se izračunati gradijent za svaki piksel. Podjela će se vršiti sa ćelijama 8x8 piksela no princip izračuna gradijenata je isti. Prvo se izvuče vrijednosti piksela kao što je objašnjeno. Vrijednosti su prikazane na slici (Slika 4.).



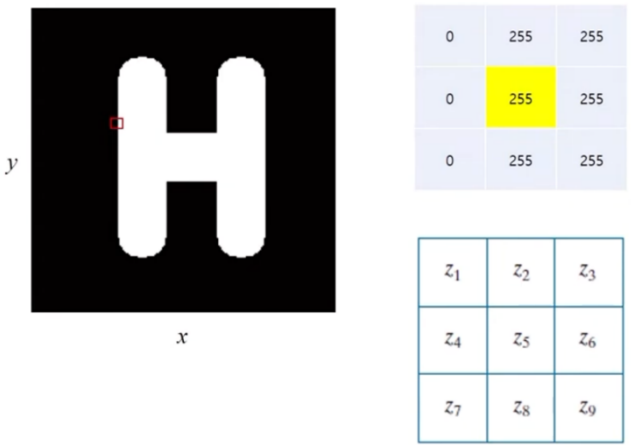
Slika 4. Izračun gradijenata i orijentacije [2]

Crvenom bojom je označen piksel za kojeg se računa magnituda i orijentacija gradijenta. Prvo će se izračunati gradijent odnosno promjenu u x smjeru. Kako bi se to provelo potrebno je promatrati vrijednosti lijevo i desno od označenog piksela. Isto tako kako bi se izračunao gradijent odnosno promjenu u y smjeru potrebno je promatrati vrijednosti iznad i ispod promatranog piksela pa će rezultantni gradijenti u x i y smjeru iznositi:

$$G_x = 89 - 78$$
$$G_y = 64 - 56 \qquad (4)$$

Ovim procesom dobiju se dvije nove matrice koje govore o promjenama u x i y smjeru. Što su promjene izraženije time će totalni gradijent imati veću magnitudu, a te promjene su uvijek izraženije oko rubova. Na sličnom principu radi i Sobel operator koji se isključivo koristi za detekciju rubova kako u vertikalnom tako i u horizontalnom smjeru. U nastavku će se ukratko objasniti rad Sobel operatora.

U nastavku je vidljiva jednostavna fotografija (Slika 5.) kojom će se ukratko objasniti Sobel operator.

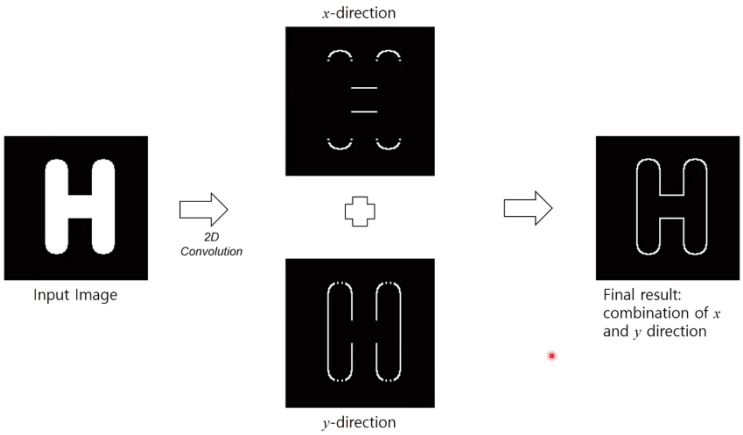


Slika 5. Primjer Sobel operatora [4]

Na fotografiji se vide vrijednosti piksela za odabrani promatrani dio slike. Nakon uvrštavanja vrijednosti u sljedeće formule dolazi se do vrijednosti koje govore u kojem smjeru se proteže rub, u ovom slučaju slova H.

$$g_x = \frac{\partial f}{\partial x} = (z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3) = 0$$
$$g_y = \frac{\partial f}{\partial y} = (z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7) = 1{,}020 \quad (5)$$

Nakon provedene derivacije u x i y smjeru vidljivo je da se promjene događaju u y smjeru što objašnjava da će se i rub na tom mjestu protezati u y smjeru. Daljnjim provođenjem postupka dobit će se rubovi u x i y smjeru te krajnjim spajanjem tih dviju slika rezultira konačnom prikazu ruba slova H do kojeg je došlo zbog različitih intenziteta piksela unutar slike.



Slika 6. Postupak otkrivanja rubova [4]

Nakon kratkog osvrta na Sobel operator vračamo se postupku izračuna magnitude i orijentacije gradijenata. Sada će se za već izračunate gradijente u x i y smjerovima u prethodnom dijelu odrediti njihova magnituda koja se računa pomoću Pitagorinog teorema kako je i naznačeno na slici (Slika 4.).

$$Magnituda = \sqrt{{G_x}^2 + {G_y}^2} = \sqrt{11^2 + 8^2} = 13{,}6 \quad (6)$$

Nakon izračuna magnitude potrebno je izračunati orijentaciju za isti taj piksel. Za to će poslužiti trokut sa slike (Slika 4.). Pomoću trigonometrijskih funkcija može se odrediti iznos kuta odnosno orijentacija.

$$\tan(\Phi) = \frac{G_y}{G_x}, \quad (7)$$

Sređivanjem izraza (7) dobit će se iznos kuta koji glasi:

$$\Phi = \operatorname{atan}\left(\frac{G_y}{G_x}\right) = 36^\circ \quad (8)$$

Sada imamo po dvije vrijednosti za svaki piksel. Jedna vrijednost se odnosi na magnitudu dok se druga vrijednost odnosi na orijentaciju. Iz tih vrijednosti će se formirati dvije matrice iz kojih će se kasnije definirati histogrami za svaki dio fotografije koju je odabrana za primjer.

Histogram općenito je grafikon koji se često koristi u području statistike. Histogram vizualizira numeričke podatke inducirajući broj točaka podataka koji se nalaze unutar određenog raspona vrijednosti. Ti se rasponi vrijednosti nazivaju klase odnosno spremnici. Čim je veća frekvencija odnosno učestalost podataka u klasi, koju još nazivamo binom time će biti i viša razina stupca koji je definiran za određenu klasu podataka. U nastavku će se opisati metode kojima se mogu realizirati histogrami ovisno o rasponu klase i načinu unosa vrijednosti u tu istu klasu.

2.2.3 Stvaranje histograma

Histogrami se realiziraju na više načina prema [5], kao što je već spomenuto, ovisno o rasponu klase i shvaćanju pripadnosti vrijednosti u područje klase. U nastavku će biti objašnjene pojedine metode stvaranje histogram u ovom slučaju to će biti histogram orijentiranih gradijenata koji će na x-osi sadržavati raspodjelu orijentacija dok će na y-osi sadržavati frekvenciju vrijednosti.

Kod prve metode koja je najjednostavnija svaki stupanj orijentacije posjeduje svoju klasu. U slučaju kada se uzme određeni piksel iz fotografije i očita mu se orijentacija gradijenta za taj piksel, na mjesto klase iste te vrijednosti dodjeljuje se vrijednost 1. Taj se postupak ponavlja za svaki piksel te se frekvencija kod istih orijentacija povećava. Ako odabrani piksel sadrži vrijednost 36, kraj vrijednosti 36 stavlja se broj koji označava broj ponavljanja za tu orijentaciju npr.

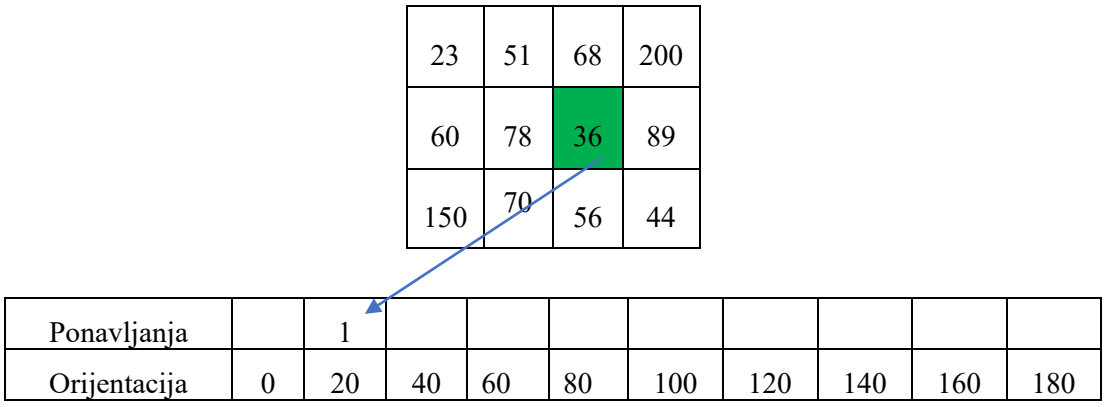
23	51	68	200
60	78	36	89
150	70	56	44

Ponavljanja									1				
Orijentacija	1	2	3	4	5…	33	34	35	36	37…	178	179	180

Tablica 1. Klasifikacija vrijednosti 1. način

Ova tablica se kasnije može upotrijebiti za izradu histograma.

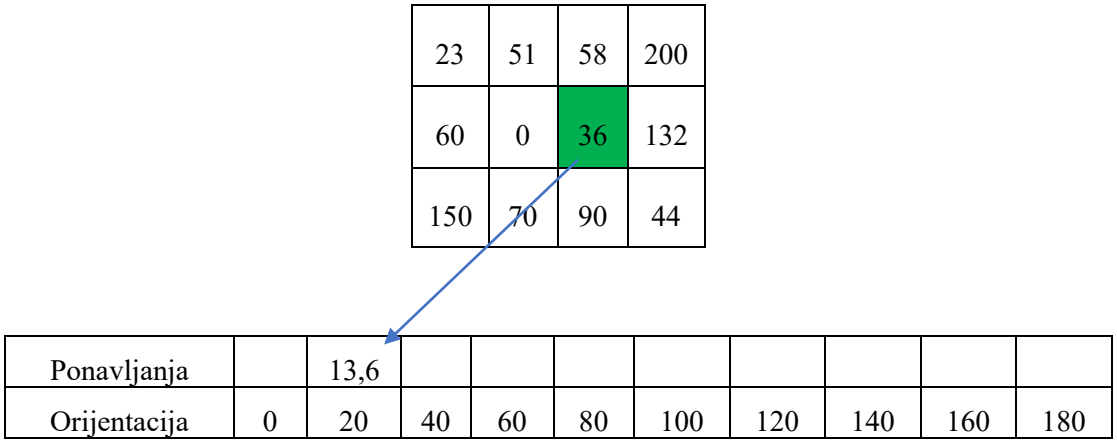
Sljedeća metoda koja se koristi relativno je slična prijašnjoj, samo što se u ovoj metodi tablica klasificira na 9 polja. Svako polje sadrži raspon vrijednosti od 20, zajedno sa 9 klasa čine ukupni kut od 180 stupnjeva. Tako će se za svaki piksel dobiti matrica veličine 9x1 koja se kasnije možemo upotrijebiti za stvaranje samog histograma. U nastavku je prikazan način stvaranja matrice.



23	51	68	200						
60	78	36	89						
150	70	56	44						
Ponavljanja		1							
Orijentacija	0	20	40	60	80	100	120	140	160	180

Tablica 2. Klasifikacija vrijednosti 2. način

Treća metoda kojom se mogu formirati same matrice koristi se isto podjelom od 9 klasa no u njoj umjesto da se piše frekvencija ponavljanja orijentacija, piše se vrijednost magnitude za određene piksele. U nastavku je prikazan primjer formiranja takve matrice. Vrijednost magnitude jednaka je 13,6 dok je vrijednost orijentacije jednaka prijašnjim slučajevima.



23	51	58	200						
60	0	36	132						
150	70	90	44						
Ponavljanja		13,6							
Orijentacija	0	20	40	60	80	100	120	140	160	180

Tablica 3. Klasifikacija vrijednosti 3. način

Ova metoda popunjava samo jednu klasu orijentacije iako se vrijednost iste te orijentacije nalazi između dvije klase što nam govori da metoda možda sadrži neke greške kao i u prošle dvije metode. Kako bi se izbjegla ova greška u nastavku će se koristiti metoda koja dijeli vrijednost magnitude ovisno o razini pripadnosti nekoj klasi. Dalje će se obraditi još jedna metoda koja će se koristiti dalje u radu za izradu histograma.

Četvrta metoda klasifikacije vrijednosti za izradu histograma dijeli gradijente između dviju klasa, na način da klasi čija je vrijednost bliže vrijednosti orijentacije pojedinog piksela dodijeli veća vrijednost magnitude. U nastavku je vidljiv primjer na kojem se koristi ova posljednja od četiriju metoda.

23	51	68	200
60	78	36	89
150	70	56	44

Ponavljanja		2,72	10,88							
Orijentacija	0	20	40	60	80	100	120	140	160	180

Tablica 4. Klasifikacija vrijednosti 4. način

Način na koji se dijelim vrijednost magnitude je jednostavan. Izračunom postotka pripadnosti vrijednosti magnitude ovisno o pripadnosti orijentacije pojedinoj klasi. Postupak se može vidjeti u nastavku.

$$K_1 = \frac{(\Phi - \Phi_1)}{(\Phi_2 - \Phi_1)} = 2{,}72 \qquad K_2 = \frac{(\Phi - \Phi_2)}{(\Phi_2 - \Phi_1)} = 10{,}88 \qquad 9)$$

K_1 – vrijednost magnitude prve klase

K_2 – vrijednost magnitude druge klase

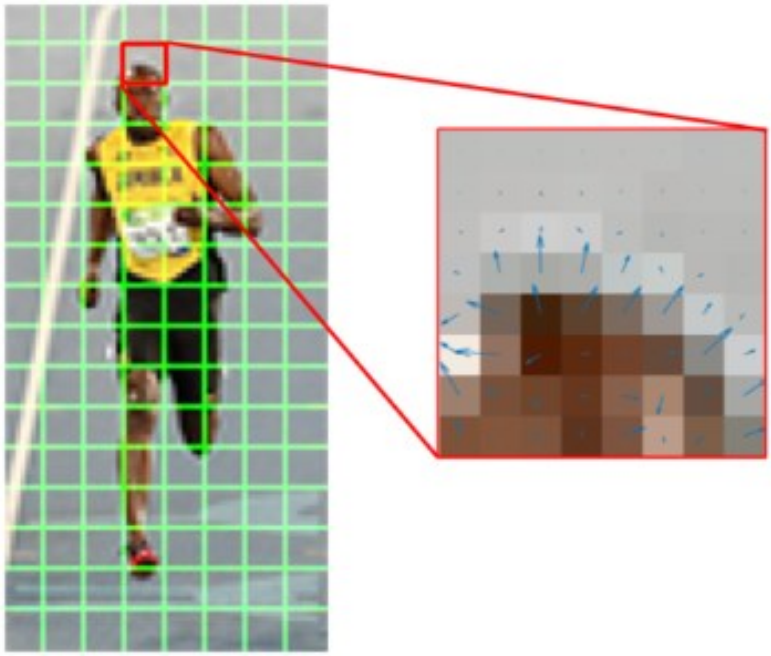
Φ – orijentacija piksela

Φ_1, Φ_2 – orijentacija piksela prve i druge klase

Kao što je navedeno ovo je metoda koja će se koristiti za stvaranje histograma dalje u procesu i jedina metoda kojom nastaju HOG algoritmi. U nastavku će se objasniti sljedeći korak koji će se objasniti stvaranje HOG algoritma.

2.2.4 Podjela slike

Nakon prethodnog poglavlja gdje je objašnjemo na koji način će se dobiti naše matrice kojima će se stvoriti histogrami slijedi podjela fotografije u ćelije sa po 8x8 piksela na sljedeći način kako je prikazano na slici (Slika 7.).



Slika 7. Dio slike 8x8 piksela [6]

Za svaku od tih podjela dobit će se 9x1 matrica koja sadrži vrijednosti magnituda za svaki odabrani piksel s obzirom na raspon u kojem se nalaze orijentacije gradijenata. Na taj način će se formirati histogrami za svaki dio fotografije.

2.2.5 Normalizacija

Sljedeći korak je postupak normalizacije. Naime gradijenti su osjetljivi na svjetla područja slike pa ako podijelimo ta područja dobit ćemo zatamnjena na pojedinim mjestima. Isto tako možemo zaključiti da ćemo dobiti uži raspon vrijednosti kojima se moramo koristiti u postupku normalizacije. Općenito način na koji se normalizacija provodi pokazan je u nastavku.

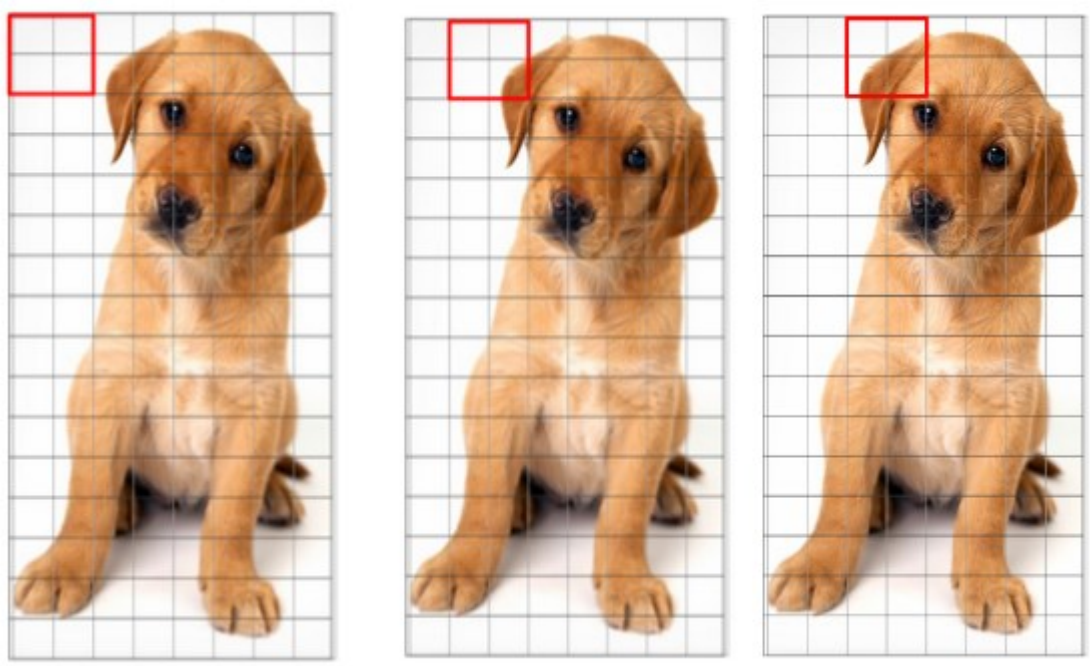
Ako uzmemo u obzir matricu veličine 1xn, npr. $V = [a_1, a_2, \dots., a_n]$. Metoda kojom će se dobiti normalizirane vrijednosti za vektor V je sljedeća:

$$k = \sqrt{(a_1)^2 + (a_2)^2 + \cdots + (a_3)^2}, \quad 10)$$

$$V_n = \left(\frac{a_1}{k}, \frac{a_2}{k}, \dots, \frac{a_3}{k}\right). \quad 11)$$

V_n – normalizirani vektor.

Na ovaj će se način dobiti i normalizirani vektor za našu matricu veličine 9x1. No slučaju ako se uzmu dijelovi slike koji su veličine 16x16 piksela kako bi normalizacija bila još efikasnija, stvorit će se dijelovi slika koji se sastoje od 4 matice veličine 9x1 ili samo jedne jedinstvenu matricu veličine 36x1. u nastavku na fotografiji (Slika 8.) prikazan je način pomicanja prozora odnosno kernela od 16x16 piksela kroz fotografiju.



Slika 8. Normalizacija pojedinih dijelova fotografije [2]

Nakon provedenog postupka normalizacije svih dijelova slike veličine 64x128 piksela dobit će se konačni vektor za cijelu fotografiju. Fotografija se sastoji od 7x15 blokova sa po 16x16 piksela. Kada se to pretvori u vektor, već znamo da naši 16x16 blokovi se sastoje od vektora veličine 36x1, množenjem sa ukupnom veličinom blokova daje konačni vektor veličine 3780x1.

7x15=105

Slika 9. Ukupni broj blokova [2]

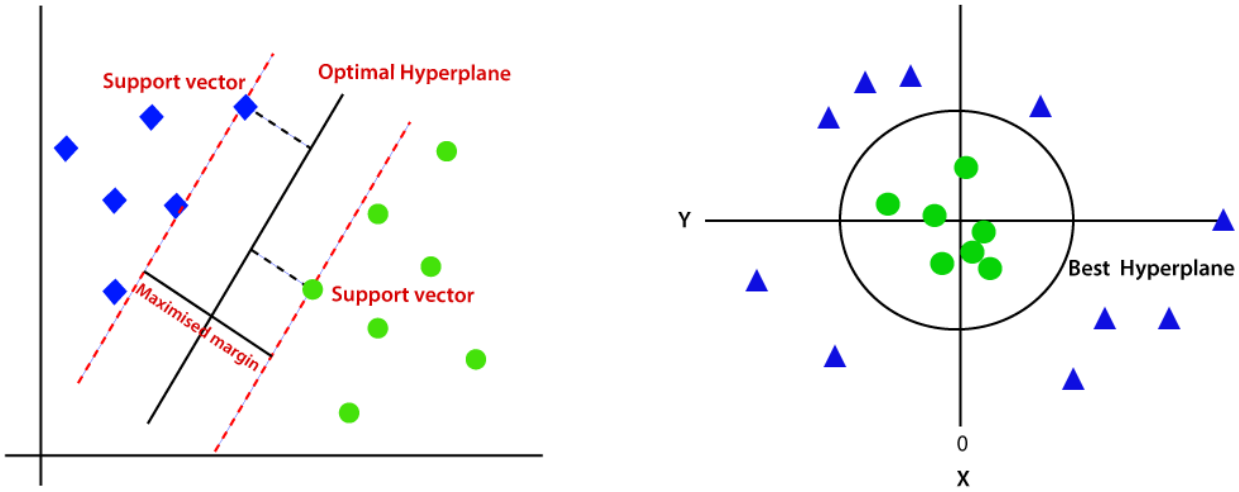
Krajnji izgled HOG fotografije bit će prikazan kasnije dok će u sljedećem poglavlju biti ukratko opisan SVM klasifikator koji služi za analizu svih dobivenih vrijednosti, te na kraju daje odgovore o tome što se na fotografiji nalazi.

3. SVM

SVM ili Support Vector Machine je jedan od nadzornih algoritama za strojno učenje, a koristi se za klasifikaciju i regresiju. Osnovna ideja SVM-a je pronaći hiperravninu (u slučaju dvodimenzionalnog problema to bi bila ravna linija) koja najbolje razdvaja dvije različite klase primjera u prostoru značajki.

Kada je riječ o klasifikaciji, SVM traži optimalnu hiperravninu koja ima najveću moguću udaljenost od najbližih primjera različitih klasa. Ti najbliži primjeri koji mogu biti u obliku podatkovnih točaka ili vektora nazivaju se potpornim vektorima što ujedno i definira ime SVM-a. Što je udaljenost između potpornih vektora veća, to će klasifikacijska granica biti bolje generalizirana i manje osjetljiva na šumove u podacima. Ta udaljenost između potpornih vektora se još naziva margina.

Postoje dva tipa SVM, a to su linearni i nelinearni SVM. Linearni SVM se koristi za podatke koji se daju linearno razdvojiti što znači da se skup podataka može klasificirati u dvije klase korištenjem jedne ravne linije, tada možemo definirati podatke kao linearno djeljive. Nelinearni SVM se koristi za nelinearno razdvojive podatke što znači da se skup podataka ne može klasificirati korištenjem ravne linije [7]. U nastavku je prikazan primjer linearne i nelinearne klasifikacije putem SVM-a. Na fotografiji s lijeva je prikazana linearna dok s desna nelinearna klasifikacija.



Slika 10. SVM klasifikacija

3.1 Izračun optimalne hiperravnine (pronalaženje maksimalne margine)

Prema podacima iz [6] hiperravnina se može zapisati kao skup točaka x koji zadovoljavaju izraz (12):

$$w^T x_i + b = 0, \tag{12}$$

gdje je w normala na hiperravninu razdvajanja odnosno vektor težine, x_i su primjeri (točke) dok je b skalarni pomak koji određuje položaj hiperravnine u odnosu na ishodište.

Ako pripremimo sve primjere za učenje tako da je njihova najmanja udaljenost od hiperravninve jednaka jedan tada ćemo dobiti sljedeća dva ograničenja koja se odnose na skup za učenje:

$$\begin{aligned} w^T x_i + b &\geq 1 \quad za\ y_i = 1 \\ w^T x_i + b &\leq -1 \quad za\ y_i = -1 \end{aligned} \tag{13}$$

Za potporne vektore kao što je objašnjeno da su to točke koje se nalaze na margini ove nejednakosti su jednakosti.

Udaljenost plohe od do točke x može se izraziti prema [6] sljedećim izrazom:

$$r = \frac{w^T x + b}{\|w\|} \tag{14}$$

Prema izrazu (14) i (13), udaljenost do vektora podrške jednaka je :

$$r = \frac{1}{\|w\|} \tag{15}$$

Vrijednost margine jednaka je udaljenosti između dvaju vektora podrške i nju možemo jednostavno definirati kao:

$$M = \frac{2}{\|w\|} \tag{16}$$

Naš cilj je dobiti maksimalan iznos margine koji možemo dobiti minimiziranjem funkcije $\emptyset(w)$. Način na koji se provodi naziva se kvadratni optimizacijski problem uz ograničenja. Potrebno je naći w i b kako bi sljedeći izraz bio minimalan:

$$\emptyset(w) = \frac{1}{2} w^t w, \tag{17}$$

uz uvjet da za sve $\{(x_i, y_i)\}$ vrijedi sljedeće:

$$y_i(w^T x_i + b) \geq 1 \tag{18}$$

U praksi, implementacije SVM-a kao što su "libsvm" ili "scikit-learn" pružaju funkcije koje automatski pronalaze hiperravninu za određeni skup podataka i probleme klasifikacije.

4. Implementacija HOG algoritma u programskom jeziku Python u svrhu detekcije ljudi

4.1 OpenCV programska biblioteka

Za izradu programa za detekciju ljudi pomoću HOG-a korištena je OpenCV biblioteka.

OpenCV (Open Source Computer Vision Library) je popularna biblioteka otvorenog koda koja pruža širok spektar funkcija i algoritama za računalni vid. OpenCV je napisan u C++-u, ali pruža i veze za različite jezike uključujući Python i Javu.

OpenCV pruža alate za obradu slika i videa, detekciju i praćenje objekata, izračunavanje optičkog toka, kalibraciju kamera, strojno učenje, računalni vid u stvarnom vremenu, analizu uzoraka i mnoge druge funkcionalnosti. Biblioteka je široko korištena u područjima poput računalnog vida, robotske vizije, sigurnosti, medicinskog slikanja, interaktivnih sustava i druge industrijske primjene. Ima bogatu dokumentaciju i aktivnu zajednicu korisnika koja pruža podršku i resurse za rad s bibliotekom. Uz to, postoje brojni primjeri koda i tu torijali dostupni na internetu koji olakšavaju upotrebu OpenCV-a.

4.2 HOG algoritam u programskom jeziku Python

U programskom jeziku Python već postoji gotova naredba HOG koja pretvara našu ulaznu sliku u novi oblik pomoću histograma orijentiranih gradijenata. Princip rada HOG algoritma objašnjen je u prethodnom poglavlju, a u nastavku će se to znanje prenijeti na programski jezik. U nastavku će biti objašnjen način rada hog algoritma.

```
hog(cell, orientations=9, pixels_per_cell=(2, 2), cells_per_block=(2, 2),
block_norm='L2-Hys' visualize=True, transform_sqrt=False, feature_vector=True, multichannel=True)
```

Slika 11. HOG funkcija u Pythonu

Sad će u nastavku biti objašnjen svaki parametar HOG funkcije u programskom jeziku Python.

Na prvo mjesto unutar naredbe dolazi fotografija odnosno izvor fotografije na kojoj se dalje provodi sam proces filtriranja fotografije.

Parametar *orientations* govori o tome u koliko dijelova je podijeljena orijentacija gradijenata jer svaki gradijent sadrži svoju magnitudu i orijentaciju. U našem slučaju to je 9 dijelova.

Parametar *pixel_per_cell* određuje veličinu ćelija pomoću kojih se dijeli slika na više manjih dijelova unutar kojih se računaju magnituda i orijentacija gradijenata za svaku taj dio slike. To znači da je u ovom slučaju fotografija podijeljena na ćelije veličine 8x8 piksela.

Sljedeći parametar je *cells_per_block* koji nakon što smo podijelili sliku u manje dijelove ponovno grupira te manje dijelove u svrhu uvođenja lokalne normalizacije i poboljšanja robusnosti prikaza značajki. Parametar *cells per block* određuje veličinu bloka u smislu broja ćelija.

Parametrom *block_norm* određujemo vrstu normalizacije. 'L2-Hys' primjenjuje normalizaciju L2-norme. Ova vrsta normalizacije dodatno pojačava kontrast i omogućuje bolje filtriranje šumova izazvanih varijacijama osvjetljenja.

Kako bi dobili prikaz slike nakon što smo proveli HOG algoritam, potreban nam je parametar *visualize*. On nam omogućuje vizualizaciju parametara svakog gradijenta tako da stvara linije ili strelice koje su proporcionalne magnitudama gradijenata i usmjerene u smjeru orijentacije.

Sljedeći Parametar je *transform_sqrt.* Kada je taj parametar jednak *True* on pomaže u ublažavanju utjecaja jakih gradijenata u jako osvijetljenim područjima i naglašava gradijente u područjima s niskim kontrastom.

Parametar *feature_vector* predstavlja informacije o lokalnom obliku i rubu slike. Može se koristiti kao ulaz za algoritme strojnog učenja za zadatke poput otkrivanja objekata, klasifikacije slika ili drugih zadataka računalnog vida.

Multichannel je parametar koji označava da se posljednja dimenzija slike smatra kanalom boja.

U nastavku rada će biti objašnjeno kako i na koji način izračunati gradijente za odabranu fotografiju pomoću sobel operatora te kako dobiti vrijednosti magnitude i orijentacije. Dok ćemo kasnije u radu prikazati rezultat korištenja HOG algoritma.

4.2.1 Izračun gradijenata te magnitude i orijentacije

Proces računanja iznosa orijentacije i magnitude u Python-u je već unaprijed definiran pozivom funkcije *"norm, angle = cv2.cartToPolar(gradx,grady,angleInDegrees=True)."* Funkcija *"cv2.cartToPolar"* izbacuje dvije vrijednosti od kojih se jedna odnosi na vrijednost magnitude, u ovom slučaju naziva *"norm",* a druga se odnosi na vrijednost same orijentacije naziva *"angle"*.

Parametri koje je potrebno navesti su gradijenti u *x* i *y* smjeru koji se mogu dobiti pozivom funkcije *cv2.Sobel(img, cv2.CV_64F,1,0,ksize=1)* koja se poziva iz OpenCv biblioteke naredbom *"import"*. Parametar *"angleInDegrees"* ako je postavljen kao *"True"* govori da je će vrijednosti orijentacija biti zapisane u stupnjevima. U suprotnom ako je postavljen kao *"False"* vrijednosti orijentacija će biti zapisane u radijanima. U nastavku ćemo se prvo fokusirati na izračun gradijenata i filtriranje fotografije pomoću sobel operatora u *x* i *y* smjeru te će na posljetku biti prikazan sam izgled fotografije nakon provedbe sobel operatora.

Sobel operator u programskom jeziku Python dolazi zajedno sa OpenCV bibliotekom kao što je navedeno te se mora pozvati naredbom *"import"*. Sobel operator na temelju promjena odnosno gradijenata u *x* i *y* smjeru pronalazi rubove kao što je navedeno u poglavlju 2.2.2.

Funkcija kojom se poziva Sobel operator je *" cv2.Sobel()"*. Ona se sastoji od četiri parametra. Prvi parametar nam služi za unos fotografije nad kojim će se izvršiti operacija. Drugi parametar *"cv2.CV_64F"* služi za određivanje vrste podataka slika izlaznog gradijenta. U ovom slučaju to će biti 64-bitni zapis koji će osigurati točan prikaz vrijednosti gradijenata. Treći parametar nam govori o smjeru računanja gradijenata. Ako je redoslijed nule i jedinice *"1, 0"* tada se provodi filtriranje fotografije odnosno računanje gradijenata u smjeru x-osi dok u situaciji kada je redoslijed suprotan, znači *"0, 1 "* tada se provodi filtriranje fotografije odnosno računanje gradijenata u smjeru y-osi. Posljednjim parametrom određujemo veličinu kernela. Sobelov operator obično koristi kvadratnu jezgru s unaprijed definiranom veličinom za izračunavanje gradijenata u smjerovima x i y. Veličina kernela određuje opseg susjedstva koji se uzima u obzir za izračun gradijenta. Obično se uzima u obzir veličina kernela od 1, 3, 5 i 7. Povećanjem kernela obuhvaćamo veći broj informacija u računanju naših gradijenata. Tako rubovi postaju glađi i sadrže manje detalja dok smanjenjem kerenela rubovi postaju oštriji, ali postoji veći broj smetnji na fotografiji. U nastavku će biti prikazan kod u Pythonu zajedno sa utjecajem veličine kernela na naš konačni rezultat filtriranja fotografije.

Sobel kod:

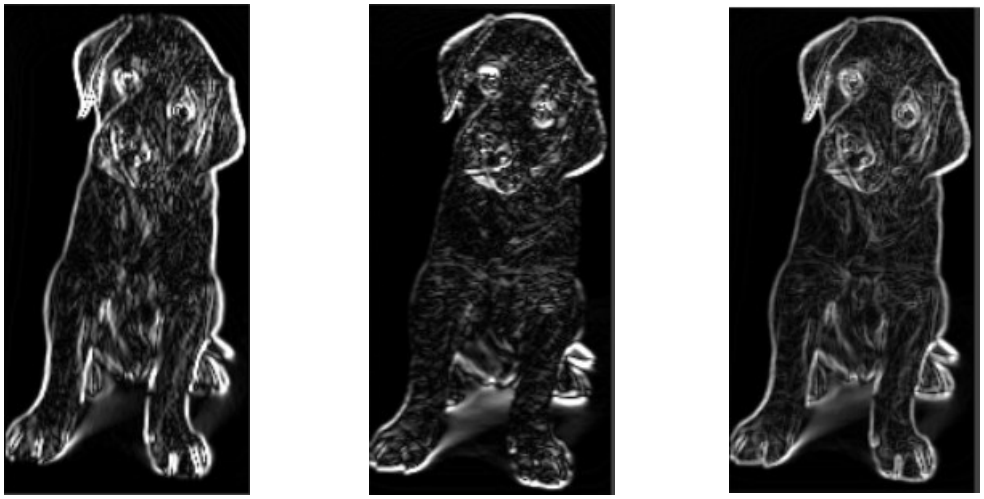
```
import cv2
import numpy as np

image=cv2.imread('C:/Emil/Faks/Zavrsni_rad/puppy_image.jpeg',
cv2.IMREAD_GRAYSCALE)

sobel_x = cv2.Sobel(image, cv2.CV_64F, 1, 0, ksize=3)  # Sobel operator u x-smjeru
sobel_y = cv2.Sobel(image, cv2.CV_64F, 0, 1, ksize=3)  # Sobel operator u y-smjeru
sobel_x = cv2.convertScaleAbs(sobel_x)
sobel_y = cv2.convertScaleAbs(sobel_y)
sobel_povezano = cv2.addWeighted(sobel_x, 0.5, sobel_y, 0.5, 0)

cv2.imshow('orginalna slika', image)
cv2.imshow('Sobel X', sobel_x)
cv2.imshow('Sobel Y', sobel_y)
cv2.imshow('Sobel povezano', sobel_povezano)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Originalna slika je u RGB formatu no za ovu primjenu je pomoću naredbe cv2.IMREAD_GRAYSCALE pretvorena u crno bijelu sliku sa vrijednostima piksela od 0 do 255. Jednostavnim pozivanjem Sobel operatora naredbom *"cv2.Sobel"* i uvrštavanjem odgovarajućih parametara, koji su prethodno objašnjeni, dobiju se gradijenti u x i y smjeru koji će se u nastavku prikazati u obliku fotografije pomoću naredbe *"cv2.imshow"*. Isto tako biti će prikazana i razlika u korištenju različitih veličina kernela.

Slika 12. prikazuje originalnu sliku prije nego što smo proveli filtriranje odnosno Sobel operator nad njom.

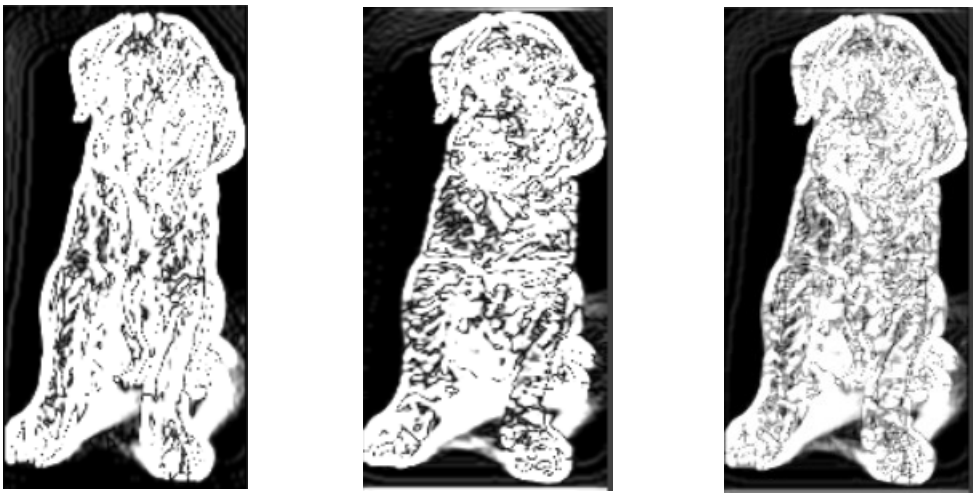
Slika 12. Originalna slika

Slijedi nam prikaz fotografija nakon što smo proveli filtriranje. Vidjet ćemo prikaz fotografija za filtriranje u smjeru x-osi i u smjeru y-osi te na posljetku njihov spoj.



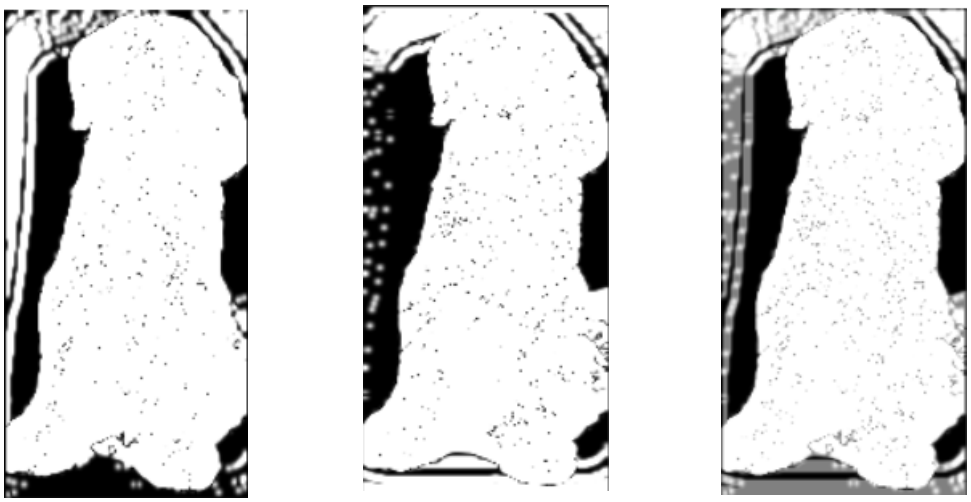
Slika 13. Sobel filtriranje, kernel 3

Slika 13. prikazuje izgled fotografije nakon filtriranja Sobelo-m. Na skroz lijevoj fotografiji filtriranje je provedeno u x-smjeru, na srednjoj u y-smjeru, dok je na skroz desnoj fotografiji povezano filtriranje u x i y smjeru. Na fotografiji Slika 14. prikazano je filtriranje Sobelo-m istim redoslijedom samo je ovaj puta povećana veličina kernela.



Slika 14. Sobel filtriranje, kernel 5

Kod povećanja kernela vidimo da rubovi postaju sve jasniji, ali pošto su sada uzeta veća područja nad kojima se računaju gradijenti slika postaje sve više zamućenija i nejasnija svojim sadržajem. U nastavku je prikazano filtriranje kernelom veličine 7 koje još više dovodi u pitanje sadržajem fotografije.

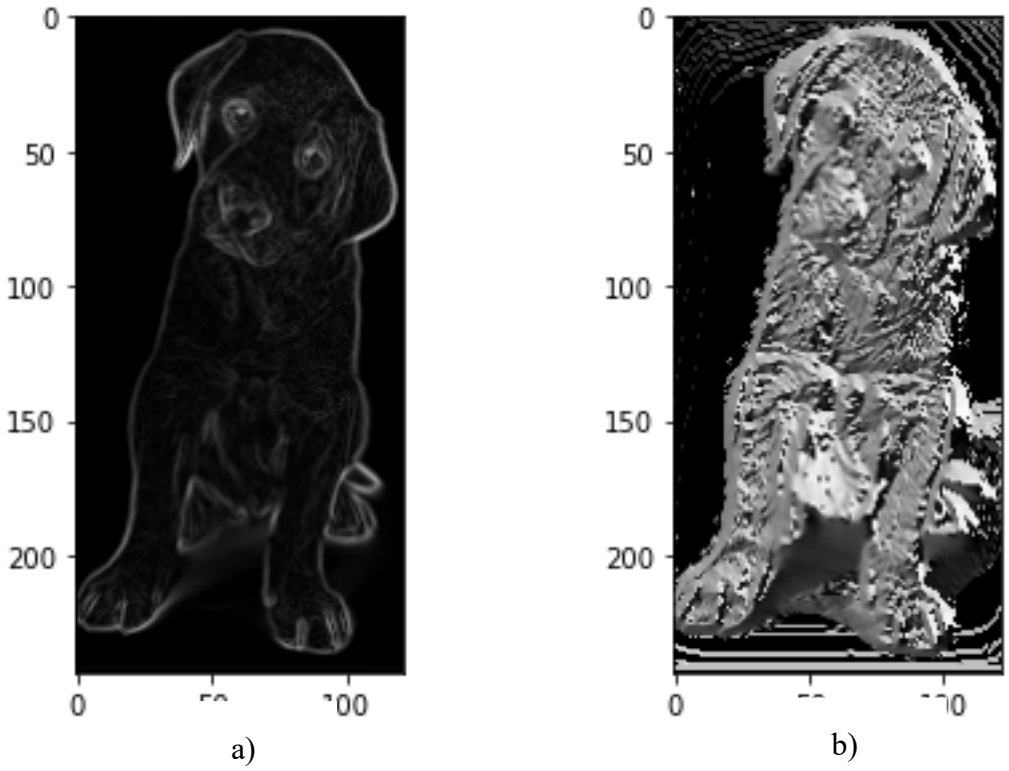


Slika 15. Sobel filtriranje kernel 7

Na slici Slika 15. vidljivo je kako su obilježeni rubovi na područjima gdje rubova ni nema zbog većih područja koja se uzimaju u obzir. Nakon izračuna gradijenata i prikaza istih na primjeru fotografije u nastavku će se ti gradijenti koristiti za izračun magnitude i orijentacije gradijenata.

Kako se računaju orijentacija i magnituda gradijenata to je već navedeno pod (6), i (7). Za računanje magnitude i gradijenata u Pythonu imam već unaprijed definiranu funkciju koja dolazi iz OpenCV programske biblioteke *"cv2.cartToPolar()"*. Pošto je objašnjeno kako izračunati gradijente koji su jedini parametri potrebni za izračun magnitude i orijentacije u nastavku će biti

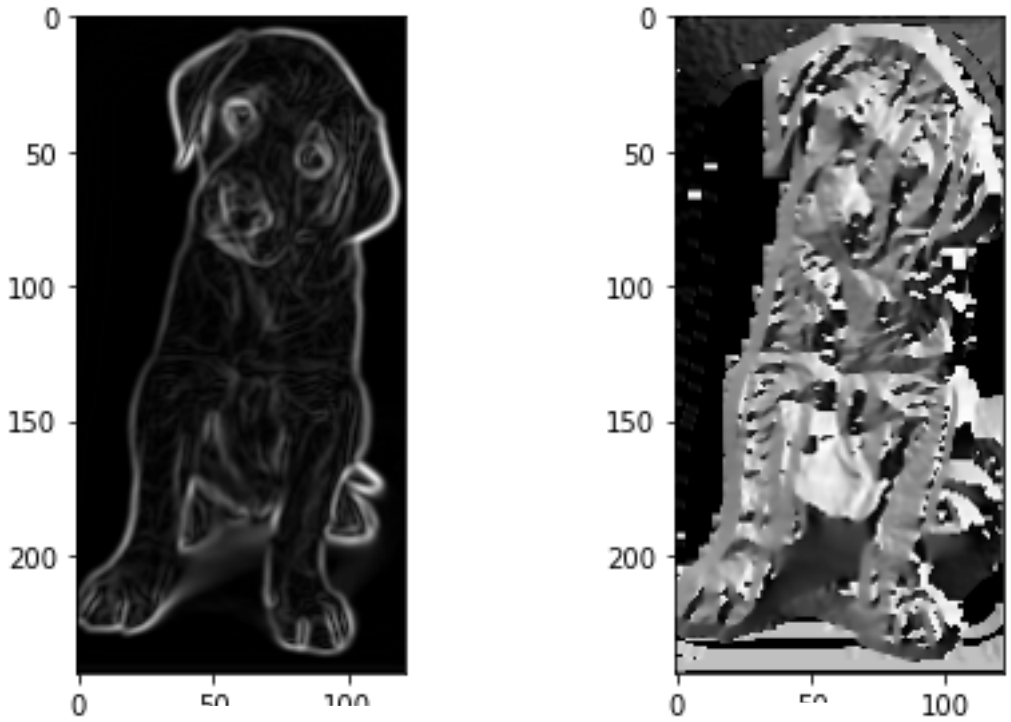
prikazan rezultat odnosno izgled orijentacija i magnituda na istoj fotografiji na kojoj je bio prikazan rezultat gradijenata.



a) b)

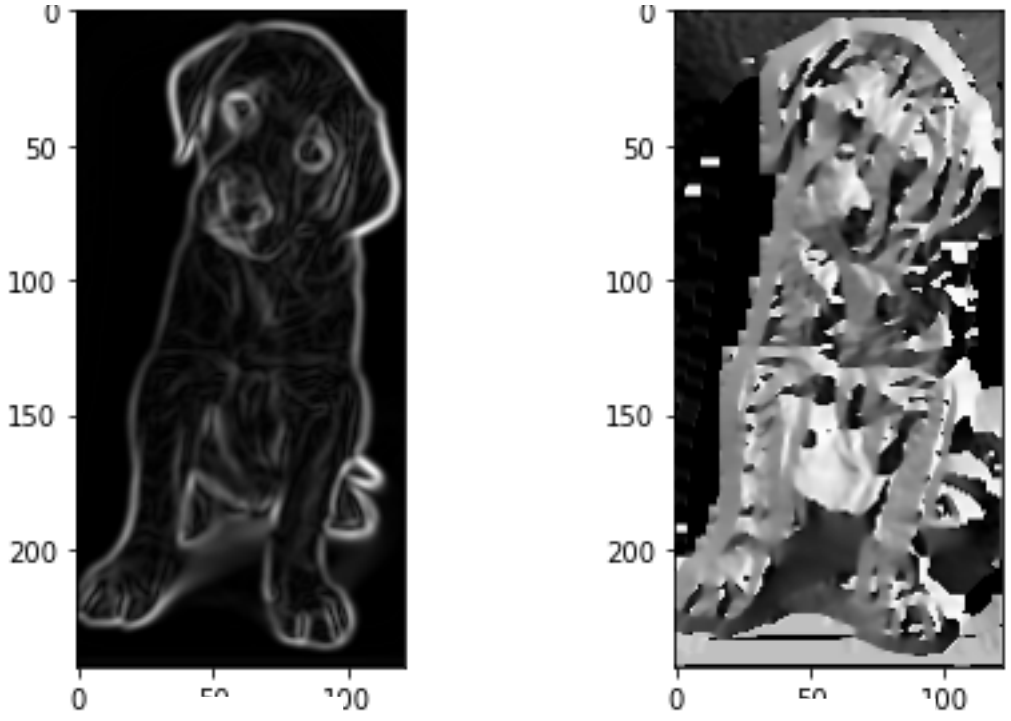
Slika 16. a) magnituda b) orijentacija gradijenata, kernel veličine 1

Slijedi izgled fotografije sa veličinom kernela 5 i 7. Nakon tih računanja magnitude i orijentacija primijetiti će se istaknutiji rubovi kod prikaza magnitudom dok će kod prikaza orijentacijom slika imati glađe promjene s manje detalja.



a) b)

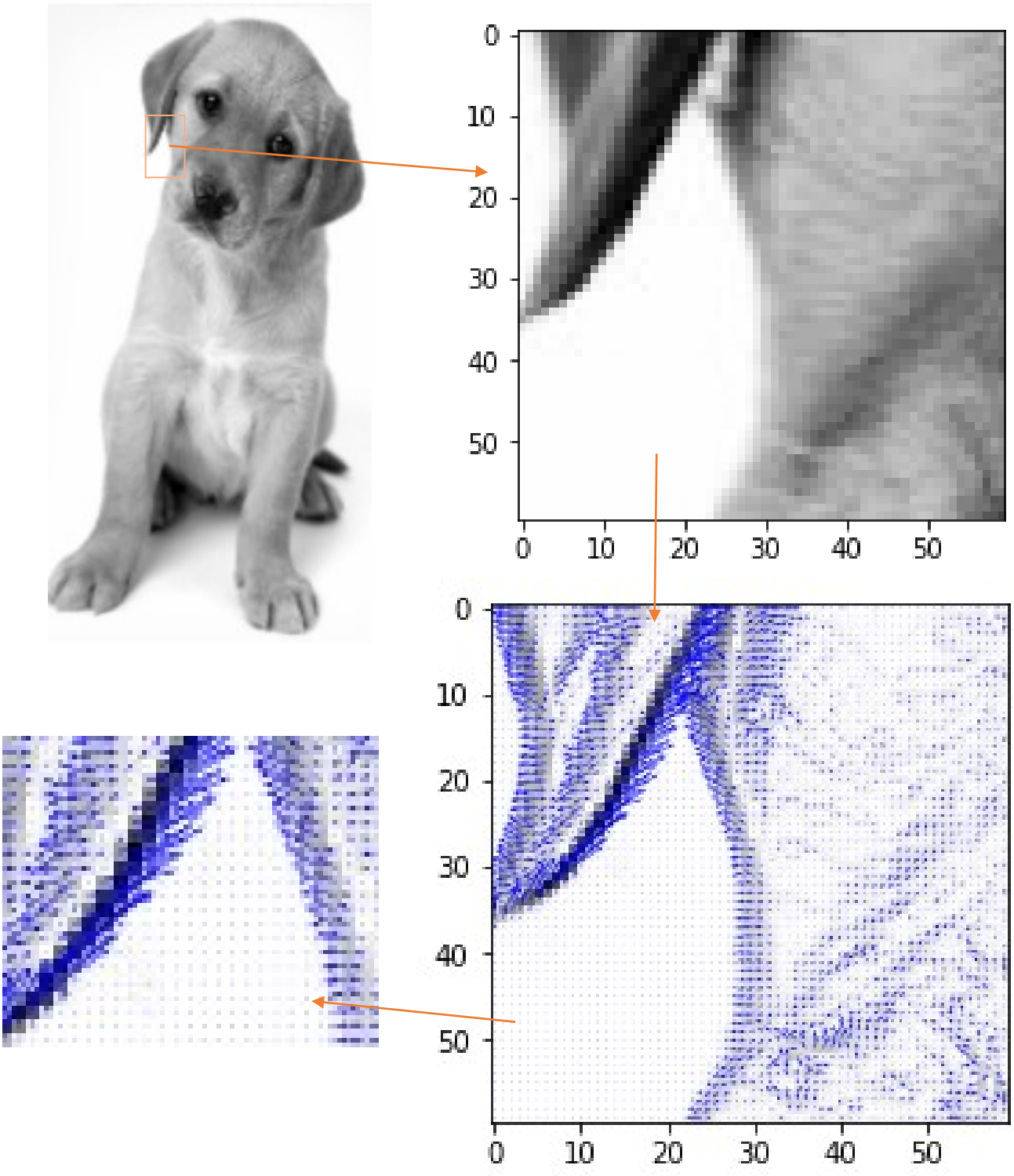
Slika 17. a) magnituda b) orijentacija, kernel veličine 5



a) b)

Slika 18. a) magnituda b) orijentacija, kernel veličine 7

U nastavku će taj iznos magnitude i orijentacije biti prikazan na slikoviti način pomoću vektora koje simboliziraju intenzitet i orijentaciju prema [8].



Slika 19. Interpretacija magnitude i orijentacije pomoću vektora

Funkcija kojom smo ostvarili prikaz magnitude i orijentacije je sljedeća:

plt.quiver(gradx, grady, color='blue') [6],

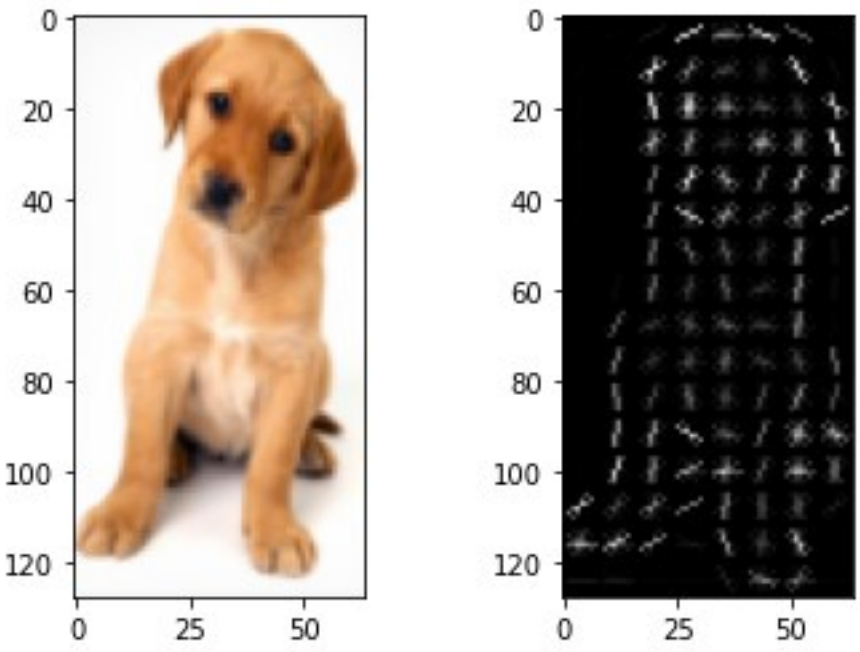
Na slici Slika 19. vidljivo je kako je magnituda gradijenata najveća oko rubova što je ujedno i način na koji se pomoću HOG algoritma, pomoću odabranog klasifikatora, detektiraju rubovi, a zatim im se i pridodaje značenje koje opisuje samu fotografiju. U nastavku slijedi izgled fotografije prilikom detekcije rubova pomoću samog HOG algoritma.

Kod koji će se koristiti u daljnjoj obradi slike koji koristi HOG algoritam u programskom jeziku Python dan je sljedećim izrazom:

```
from skimage.io import imread, imshow
from skimage.transform import resize
from skimage.feature import hog
import matplotlib.pyplot as plt


img = imread('C:/Emil/Faks/Zavrsni_rad/puppy_image.jpeg')
resized_img = resize(img, (128,64))


fd, hog_image = hog(resized_img, orientations=9, pixels_per_cell=(4, 4),
                    cells_per_block=(2, 2), visualize=True, multichannel=True)


plt.subplot(1,2,1)
plt.imshow(resized_img)
plt.subplot(1,2,2)
plt.imshow(hog_image, cmap='gray')


pilt.show()
```

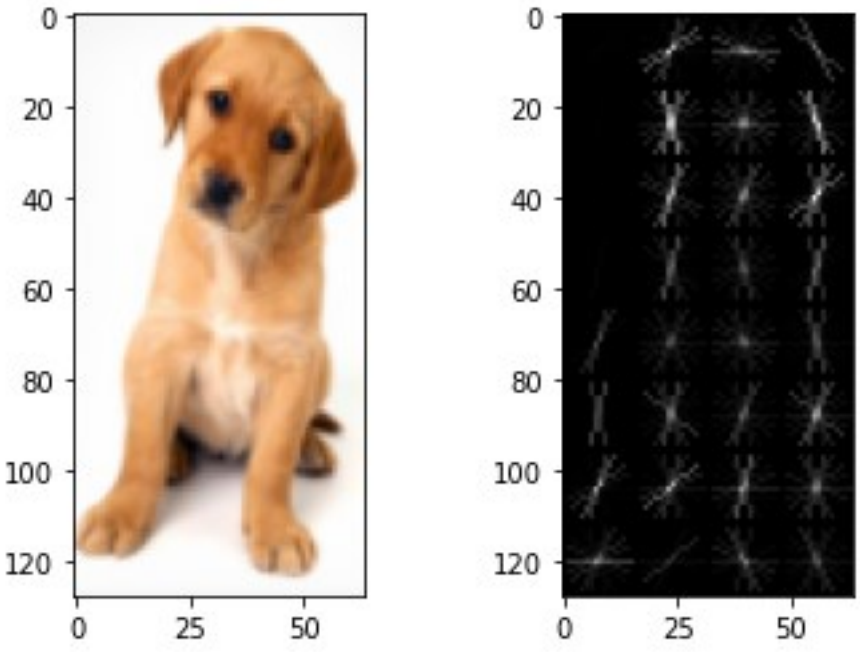
Kao i u poglavlju 2. gdje je objašnjeno na kojem principu radi HOG algoritam tako i ovdje u kodu prvo se promijeni veličina izvorne fotografije u ovom odgovarajući format. Zatim

koristimo HOG funkciju. Za ovaj primjer će se koristiti ćelije veličine 8x8 piksela, isto kao i u poglavlju 2. kako bi stekli dojam samog izgleda rezultata. Normalizacija će se isto vršiti s blokovima od 2x2 ćelije koje nakon spajanja tvore veličinu od 16x16 piksela. Krajnji rezultat nakon provođenja operacije prikazan je na slici Slika.20.



Slika 20. HOG filter s veličinom polja (8,8) piksela

Na slici Slika 21. prikazan je rezultat provođenja procesa pomoću ćelija veličine 16x16 piksela. Filtriranje je grublje i uzima se u obzir veće područje za izračun gradijenata. Normalizacija se provodi isto s ćelijama 2x2 što će u ovom slučaju povećati područje normalizacije na 32x32 piksela.



Slika 21. HOG filter s veličinom polja (16,16) piksela

Na slici Slika 21. vidljivi je manji broj detalja koji su se smanjili zbog podijele slike na veće dijelove. Ako povećamo ćelije doći će do smanjenja detalja na fotografiji, ali će se isto tako smanjiti i vrijeme potrebno za procesiranje. Sljedeće poglavlje prikazuje rezultat svega naučenog u detekciji ljudi pomoću HOG algoritma te linearnog SVM-a.

5. Detekcija ljudi pomoću HOG algoritma u programsko jeziku Python

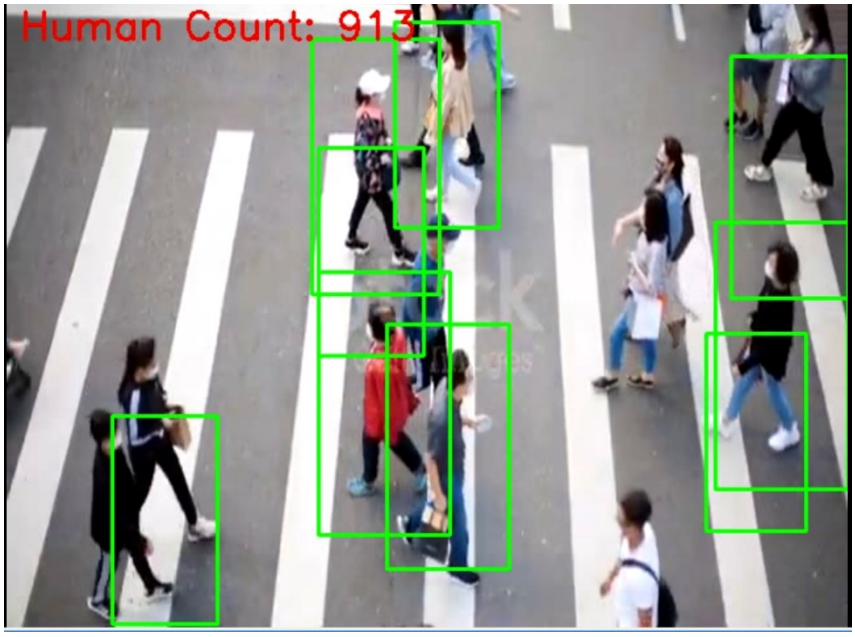
U ovom poglavlju koristit će se HOG algoritam kojim će se zajedno sa algoritmom za strojno učenje, odnosno SVM algoritmom za klasifikaciju, detektirati ljudi. Za sve to poslužit će OpenCV programska biblioteka unutar programskog jezika Python prema [8].

U nastavku je prikazan kod korišten za detekciju ljudi u programskom jeziku Python:

```
import cv2


hog = cv2.HOGDescriptor()
hog.setSVMDetector(cv2.HOGDescriptor_getDefaultPeopleDetector())
cap = cv2.VideoCapture('C:\Emil\Faks\Zavrsni_rad\moj_video1.mp4')
broj ljudi = 0


while(True):
   ret, frame = cap.read()
   if ret == True:
      frame = cv2.resize(frame, (640, 480))
      boxes, weights = hog.detectMultiScale(frame, winStride=(4, 4), padding=(8, 8),
scale=1.05)
      for (x, y, w, h) in boxes:
         cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)
         broj ljudi+= 1
      cv2.putText(frame, f'Human Count: {broj ljudi}', (10, 30),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)
      cv2.imshow("Detektiranje i zbroj ljudi", frame)
      if cv2.waitKey(25) == ord('q'):
         break


cap.release()
cv2.destroyAllWindows()
```

Prvo se poziva naredba *"cv2.HOGDescriptor()"* pomoću koje se izračunavaju HOG značajke. Pozivom naredbe *"hog.setSVMDetector(cv2.HOGDescriptor_getDefaultPeopleDetector())"* učitava se unaprijed obučeni SVM detektor za otkrivanje ljudi na temelju izračunatih HOG značajki. Nakon toga učitava se sam video pomoću naredbe *"cv2.VideoCapture()"* unutar koje se definira sam put te ime željenog videa za obradu. Ovaj algoritam se može koristiti i za spajanje na samu web kameru tako što se jednostavno uvrsti nula kao mjesto učitavanja sadržaja, a mogu se ubaciti i same fotografije. Nakon toga postavljen je broj ljudi na početku koji je jednak 0. Zatim se stvara beskonačna petlja *"while"* unutar koje se vrši obrada podataka. *"cap.read()"* služi za učitavanje podataka sa videa i daje dvije informacije. Prva informacija je dali je okvir učitan i daje nam podatke o slici koja je u tom trenutku očitana. Naredba *"hog.detectMultiScale()"* koristi se za otkrivanje objekata koji dolaze u različitim omjerima na slici. Ulazna slika je u sivim tonovima te kao rezultat poziva ove funkcije dobijemo granične okvire u kojima su otkriveni objekti u ovom slučaju ljudi zajedno s njihovim odgovarajućim težinama koje ocjenjuju vjerojatnost prisutnosti ljudi odnosno objekta koji je naučen od strane SVM-a za detekciju. Kasnije se u programu ubacuju kvadrati u materijal koji je pripremljen za samu obradu. Ti kvadrati služe isključivo za označavanje detektiranog objekta.

U nastavku je prikazana slika Slika 22. na kojoj je vidljiv način detekcije ljudi koji prolaze preko pješačkog prijelaza. Slika je nastala screenshot-om iz video materijala nad kojim je proveden algoritam za detekciju ljudi te je unutar samog rezultata još ubačen brojač ljudi. Sam brojač detektira mnogo više ljudi nego što ih je bilo unutar video uratka zbog nekih od nedostataka koji su navedeni u nastavku.



Slika 22. Rezultat Detekcije ljudi pomoću HOG algoritma

6. Osvrt na rezultate

Prilikom provođenja algoritma nad različitim materijalima od slika, videa pa do live snimki s web kamere uočeno je kako HOG algoritam u većini slučajeva detektira ljude, ali i u puno slučajeva griješi. Može se reći da ima određene probleme i ograničenja.

Algoritam je osjetljiv na promjene osvjetljenja na slici. To je navedeno i prije u samom teorijskom dijelu te se zbog toga koristi normalizacija no i dalje ako se osvjetljenje fotografije razlikuje od skupa podataka koji se koriste za treniranje i učenje algoritma doći će smanjenja efikasnosti detekcije. Sljedeći problemi koji se javljaju su problemi kod skaliranja.

HOG algoritam nije otporan na promjene veličine detektiranog objekta u ovom slučaju ljudi. Ako se veličina ljudi na promatranom materijalu znatno razlikuje od veličine ljudi unutar skupa za trening to isto tako može smanjiti performanse. Tako se npr. u slučaju kod korištenja web kamere ako se previše približimo uočava greška odnosno smanjuje se točnost same detekcije. Najbolji slučaj je kad je na kameri uhvaćeno cijelo tijelo.

Isto tako algoritam ima nedostataka sa samom orijentacijom ljudskog tijela. To znači da će neki put kod nakošenih video materijala algoritam imati problema sa detekcijom jer je orijentacija objekta odnosno ljudi različita u odnosu na skup podataka za treniranje.

Još jedan problem koji je uočen i može se vidjeti na samoj fotografiji Slika 22. je u tome da ako na slici imamo više ljudi koji se preklapaju ili se nalaze u neposrednoj blizini tada možemo očekivati poteškoće u pravilnoj detekciji i razgraničavanju objekata odnosno ljudi.

U prilogu se nalazi sam video na kojem se odvila detekcija ljudi pomoću HOG i SVM algoritma.

7. Zaključak

U ovom radu prikazana je računalni model strojnog vida za detekciju ljudi. Za to smo se koristili HOG algoritmom. Na početku rada stvorena je teorijska baza koja je potrebna za lakše razumijevanje stvaranja samog algoritma. Kasnije se samom implementacijom stečenog znanja stvara kod pomoću već unaprijed definirane OpenCV programske biblioteke u programskom jeziku Python. Korištenjem HOG algoritma kao temelja u procesu detekcije ljudi uočeni su neki nedostaci koji su navedeni u prethodnom poglavlju, ali samo proces detekcije ljudi dobro funkcionira u nekim idealnijim slučajevima.

Trenutno postoje različiti algoritmi kojima se mogu detektirati objekti i dodijeliti im se značenje, a koji su napredniji i bolji od samog HOG algoritma. Obzirom da se svakodnevno radi na razvoju samog računalnog vida koji dolazi sve više u primjenu unutar različitih industrija. Danas se računalni vid sve više koristi u autonomnoj vožnji, video nadzoru, praćenju lica, robotsku navigaciju, medicinsku detekciju itd. Samom pojavom umjetne inteligencije ova grana računalnih sustava sve više i više napreduje te postiže velike rezultate koji teže što preciznijoj i bržoj detekciji samih objekata.

Literatura

[1] Wikipedia, histogram orijentiranih gradijenata: https://en.wikipedia.org/wiki/Histogram_of_oriented_gradients#cite_note-1, Pristupljeno: 24. travnja 2023.

[2] Histogram of Oriented Gradients (HOG) — Simplest Intuition | by Skillcate AI | Medium: https://medium.com/@skillcate/histogram-of-oriented-gradients-hog-simplest-intuition-2392995f8010, Pristupljeno: 26. travnja 2023.

[3] How to Convert an RGB Image to a Grayscale: https://www.baeldung.com/cs/convert-rgb-to-grayscale, Pristupljeno: 5. svibnja 2023.

[4] YouTube: https://www.youtube.com/watch?v=9YpP1uGJcmM&ab_channel=MadePython, Pristupljeno: 1. lipnja 2023.

[5] Feature Engineering for Images: A Valuable Introduction to the HOG Feature Descriptor: https://www.analyticsvidhya.com/blog/2019/09/feature-engineering-images-introduction-hog-feature-descriptor/, Pristupljeno: 2. travnja 2023.

[6] Histogram of Oriented Gradients explained using OpenCV: https://learnopencv.com/histogram-of-oriented-gradients/, Pristupljeno: 3. ožujka 2023.

[7] SVM – Understanding the math – the optimal hyperplane: https://www.svm-tutorial.com/2015/06/svm-understanding-math-part-3/, Pristupljeno: 20. svibnja 2023.

[8] Real-time Human Detection with OpenCV: https://thedatafrog.com/en/articles/human-detection-video/, Pristupljeno: 2. ožujka 2023.

PRILOZI

Prilog 1

Poveznica s videom na YouTube-u:

https://www.youtube.com/watch?v=9MJ4DFXzrOM&ab_channel=EmilGrabu%C5%A1i%C4%87