

Univerzitet u Tuzli
Fakultet elektrotehnike

Zadaća 2

Robotika i mašinska vizija

Mašinska vizija

Student:
Indir Karabegović

Profesor:
dr.sc. Naser Prljača, red. prof

Tuzla, jul, 2022.

Sažetak:

U ovom seminarskom radu ćemo se baviti rješavanjem zadataka vezanih za mašinsku viziju. Prvo ćemo proći kroz osnovne transformacije nad slikama, a nakon toga ćemo analizirati histogram slike. Kao posljednji zadatak ćemo analizirati primjenu vještačke inteligencije za potrebe obrade i analize slike.

Sadržaj

1. Zadaci.....	5
Rješenje: Zadatak 1.....	7
Rješenje: Zadatak 2.....	9
Rjesenje: Zadatak 3.....	13
Rješenje: Zadatak 4.....	16
Rješenje: Zadatak 5.....	20
Rješenje: Zadatak 6.....	23

1. Zadaci

Zadatak 1

Napisati python funkcije koje vrše rotaciju i translaciju slike. Napisati i python program koji testira ove dvije funkcije.

Zadatak 2

Objasniti Gaussov prostorni filter, averaging filter i median filter. Navesti razlike ova tri prostorna filtera i prikazati python kod sa primjerima na zašumljenim slikama.

Zadatak 3

Navesti i pokazati u pythonu filtere za detekciju ivica.

Zadatak 4

Objasniti histogram grayscale slike i pokaziti načine dobijanja crno-bijele slike.

Zadatak 5

Navesti i pokazati u pythonu metode morfološke obrade slike.

Zadatak 6

Na linku:

<https://www.kaggle.com/code/ektasharma/simple-cifar10-cnn-keras-code-with-88-accuracy/notebook>

se nalazi primjer dubokog učenja na CIFAR 10 datasetu. Potrebno je objasniti implementaciju i komentarisati rezultate.

Rješenje: Zadatak 1

U ovom zadatku je potrebno napisati python funkcije koje vrše rotaciju i translaciju slike. Također, potrebno je napisati i python program koji testira ove dvije funkcije.

Navedeni kod je napisan u narednom text box-u:

```
import numpy as np
import cv2 as cv

# Import image
img = cv.imread('images/lena.jpg')
cv.imshow('Original', img)
cv.waitKey(0)

# Convert to grayScale
gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
cv.imshow('Gray', gray)
cv.waitKey(0)

# Rotation
def rotate(img, angle, rotPoint=None):
    (height, width) = img.shape[:2]
    if rotPoint is None:
        rotPoint = (width // 2, height // 2)
    rotMat = cv.getRotationMatrix2D(rotPoint, angle, 1.0)
    dimensions = (width, height)
    return cv.warpAffine(img, rotMat, dimensions)

rotated = rotate(gray, -45)
cv.imshow('Rotated ', rotated)
cv.waitKey(0)

# Translation
def translate(img, x, y):
    transMat = np.float32([[1, 0, x], [0, 1, y]])
    dimensions = (img.shape[1], img.shape[0])
    return cv.warpAffine(img, transMat, dimensions)

translated = translate(gray, 150, 150)
cv.imshow(' Translated ', translated)
cv.waitKey(0)
```

U kodu smo prvo uključili dvije biblioteke koje će nam biti potrebne za realizaciju ovog zadatka. To su biblioteke numpy i cv2. **Numpy** biblioteka je namijenjena za rad sa nizovima, ali također ima i funkcije za rad sa lienarnom algebram, furijerovom transformacijom, kao i za rad sa matricama. **Cv2 je OpenCV**. Ova

biblioteka je namijenjena za obradu slike. U našem projektu vezanom za zadaću, konkretno u folderu *images*, nalaze se slike koje smo obrađivali. U većini slučajeva je to bila fotografija **lena.jpg**. Ova fotografija je izuzetno popularna u obradi slike. Slika se koristi čak od 1973. godine. Za učitvanje slike koristimo funkciju *imread*, a za prikaz slike koristimo funkciju *imshow*. Slika Lene (ili Lenne) je prikazna u nastavku:



Nakon toga smo od prvobitne slike, koja je u boji, napravili crno-bijelu sliku (grayscale) Lene, pomoću funkcije *cv.cvtColor*, a navedena funkcija je kao parametre prima sliku koju želimo konvertovati i parametar za način konvertovanja, u našem slučaju je to *cv.COLOR_BGR2GRAY* (u dokumentaciji za openCV imamo veliki broj *ColorConversionCodes*). Grayscale slika Lene je prikazana u nastavku:



Nakon što smo dobili grayscale sliku, sada ćemo da na nju izvršimo osnovne transformacije, a to su rotacija i translacija. Napravili smo odgovarajuću funkciju rotation koja kao parametre prima sliku i ugao za koji želimo rotirati sliku. Također, bitna nam je i tačka rotacije. Kod za navednu funkciju je prikazan u nastavku:

```
# Rotation
def rotate(img, angle, rotPoint=None):
    (height, width) = img.shape[:2]
    if rotPoint is None:
        rotPoint = (width // 2, height // 2)
    rotMat = cv.getRotationMatrix2D(rotPoint, angle, 1.0)
    dimensions = (width, height)
    return cv.warpAffine(img, rotMat, dimensions)
```

U našem slučaju je tačka oko koje se vrši rotacija centralna tačka slike, i to smo dobili tako što smo rotPoint postavili na centar slike. Matricu rotacije rotMat smo dobili pozivom funkcije cv.getRotationMatrix2D u koju smo kao parametre prosljedili navedenu tačku rotacije, ugao i scale (u dokumentaciji naveden kao Isotropic scale factor). Funkcija cv.WarpAffine je funkcija koja vrši remapiranje. Ova funkcija kao rezultat vraća zarotiranu sliku, koja je prikazana u nastavku:



U ovom slučaju je slika zarotirana za ugao od -45 stepeni. Naredna operacija koja nam je od značaja je translacija. Za implementaciju translacije slike smo napisali funkciju translate, koja kao parametre prima

sliku koju želimo translirati, te translaciju po svakoj od osa (u našem slučaju x i y). U nastavku je dat kod navedene funkcije:

```
# Translation
def translate(img, x, y):
    transMat = np.float32([[1, 0, x], [0, 1, y]])
    dimensions = (img.shape[1], img.shape[0])
    return cv.warpAffine(img, transMat, dimensions)
```

Nakon translacije smo dobili narednu sliku:

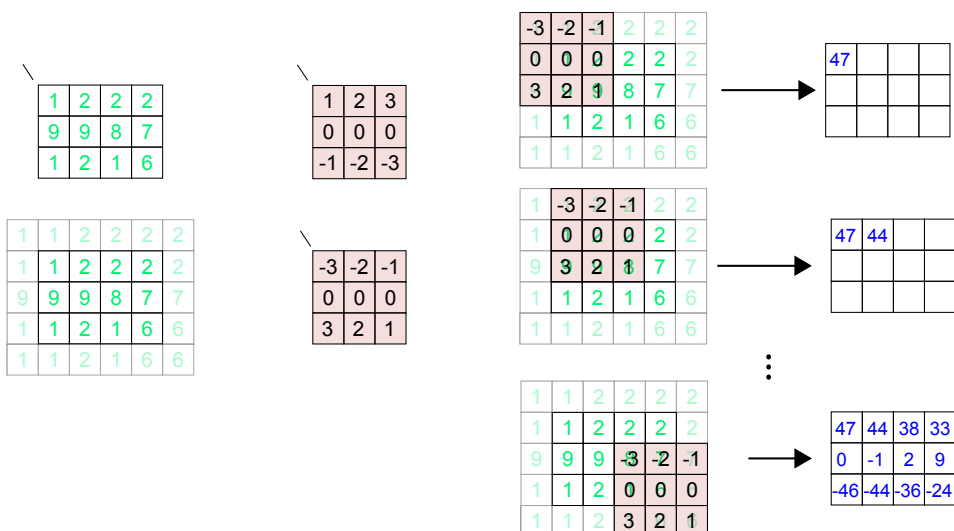


Rješenje: Zadatak 2

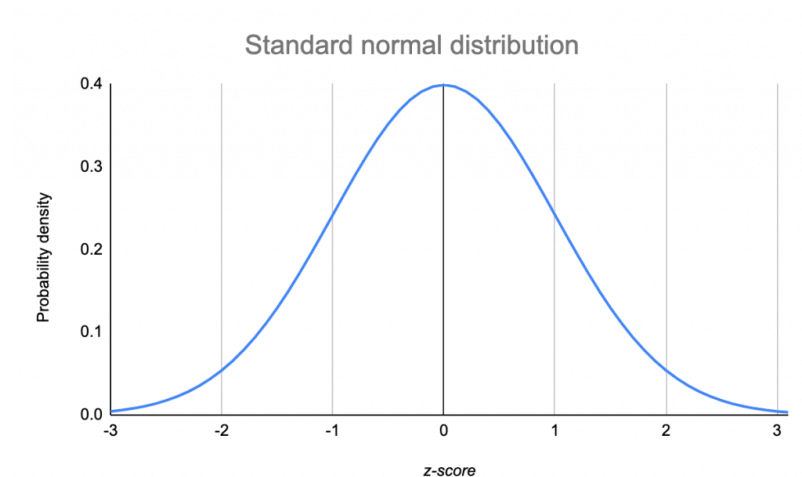
Na predavanju smo naučili da svaku sliku možemo predstaviti kao matricu određenih dimenzija. Ukoliko imamo garyscale sliku, svaki element matrice može poprimiti vrijednost od 0 do 255. Primjer jedne jednostavne grayscale slike i njene matricne interpretacije možemo vidjeti na narednoj ilustraciji:

254	143	203	176	108	229	177	220	192	9	229	142	139	64	0	63	28	8	98	82
27	68	231	75	141	107	149	210	13	239	141	35	68	242	110	208	244	0	33	86
94	42	17	215	230	234	47	41	98	180	55	253	235	47	122	200	78	110	152	100
9	186	192	71	104	193	85	171	37	233	18	147	174	1	143	211	176	188	192	68
179	20	238	192	190	132	41	249	22	134	83	133	110	254	176	238	188	234	51	204
232	25	0	183	174	129	61	30	110	189	0	173	197	183	153	43	22	87	68	118
235	35	151	185	129	81	239	170	195	94	38	21	67	101	58	37	196	149	52	154
155	242	64	0	104	109	189	47	138	254	225	156	31	181	121	15	126	35	252	205
223	114	79	129	147	6	201	68	89	107	58	44	233	84	38	1	62	5	231	218
55	188	237	188	90	101	131	241	68	133	124	151	111	28	190	4	240	78	117	145
152	155	229	78	90	217	219	105	116	77	39	49	2	9	214	181	205	118	135	33
182	94	176	199	20	149	57	223	232	113	32	45	177	15	31	179	100	119	208	81
224	118	124	172	75	29	69	180	187	195	41	44	8	170	158	101	131	31	28	112
238	83	38	7	83	69	173	163	98	237	67	227	18	218	248	237	75	192	201	148
88	195	224	207	140	22	31	118	234	34	182	116	23	47	68	242	189	152	110	248
140	37	101	230	246	145	122	64	27	58	239	1	225	145	91	100	98	90	40	195
251	4	178	139	121	95	97	174	249	182	77	112	223	186	182	82	65	252	83	196
179	180	223	230	37	182	148	78	170	19	17	4	184	176	183	102	83	81	132	206
173	137	185	242	181	181	214	49	74	238	197	37	98	102	15	217	148	8	102	188
85	9	17	222	18	210	70	21	78	241	184	216	93	93	208	102	153	212	119	47

Sa prethodne slike možemo vidjeti da pikseli koji imaju nižu vrijednost imaju tamniju boju, dok pikseli sa većom vrijednosti imaju svjetliju boju. Na ovaj način možemo predstaviti svaku grayscale sliku (npr. Lenu iz prethodnog primjera, koja će biti korištena i u ovom primjeru). Sada se postavlja pitanje kakvu vezu navedeno predstavljanje slike ima sa filtriranjem slike. Naime, prilikom filtriranja slike se preko navedene slike prevlači kernel određenih dimenzija. U obradi slike, Kernel (maska, matrica konvolucije) je mala matrica koja se koristi za zamućenje, izoštravanje, utiskivanje, detekciju ivica itd. Sve navedene operacije se postižu izvođenjem konvolucije između kernela i slike. Primjer prevlaćenja kernela preko slike je prikazan na narednoj slici:

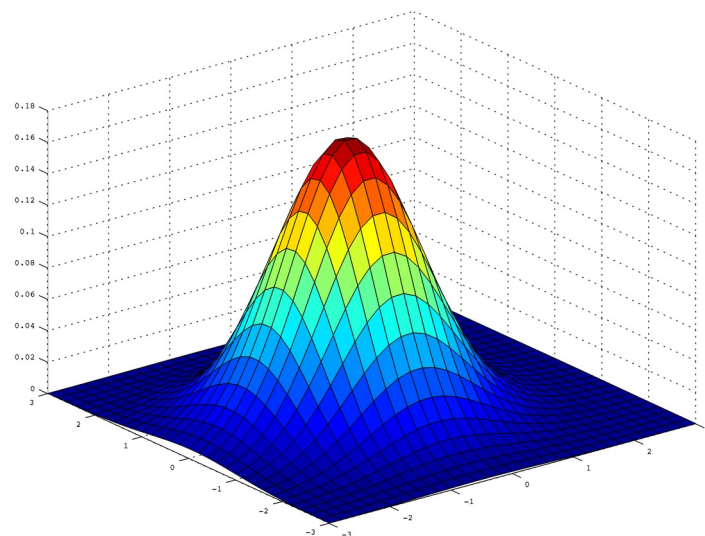


Prvi od filtera koji ćemo objasniti je Gaussov filter. Ono što je karakteristično za ovaj filter je Gausova raspodjela. Navedena raspodjela je prikazana na narednoj slici:



Da bismo kreirali Gaussov filter za obradu slike, potrebno je navedenu raspodjelu implementirati kao kernel matricu. U suštini, kernel se sastoji od pravougaonog niza brojeva koji prate Gausovu raspodjelu, odnosno normalnu distribuciju ili krivu u obliku zvona.

Kernel se sastoji od vrijednosti koje su veće u sredini i padaju prema vanjskim rubovima matrice, poput visine Gausove funkcije u 3D prostoru (prikazano na narednoj slici).



Kernel odgovara broju piksela koje uzimamo u obzir prilikom zamućenja svakog pojedinačnog piksela. Veći kerneli znace šire zamućenje širom šireg regiona, jer je svaki piksel modifikovan sa više okolnih piksela. Za svaki piksel koji će biti podvrgnut operaciji zamućenja, pravougaoni dio jednak veličini kernela uzima se oko samog piksela od interesa. Ove okolne vrijednosti piksela se koriste za izračunavanje prosjeka za novu vrijednost originalnog piksela na osnovu Gausove distribucije u samom kernelu. Primjer kernela Gausovog filtera je prikazan u nastavku:

1/16	1/8	1/16
1/8	1/4	1/8
1/16	1/8	1/16

Zahvaljujući raspodjeli, originalna vrijednost središnjeg piksela ima najveću težinu, tako da ne briše sliku u potpunosti. Isti proračun se izvodi za svaki piksel na originalnoj slici koja se zamagljuje, a konačna izlazna slika se sastoji od vrijednosti piksela izračunatih kroz proces.

Average (ili srednje) filtriranje je metoda “zaglađivanja” slika smanjenjem količine varijacije intenziteta između susjednih piksela. Average filter funkcionira tako što se kreće kroz sliku piksel po piksel, zamjenjujući svaku vrijednost prosječnom vrijednošću susjednih piksela, uključujući i sebe. Postoje neki potencijalni problemi:

Jedan piksel sa veoma nereprezentativnom vrijednošću može značajno uticati na prosječnu vrijednost svih piksela u svom okruženju. Kada se susjedstvo filtera nalazi na rubu, filter će interpolirati nove vrijednosti za piksele na rubu i tako će zamutiti tu ivicu. Ovo može biti problem ako su potrebne oštre ivice na izlazu.

Average = round($1+4+0+2+2+4+1+0+1$)/9 = 2

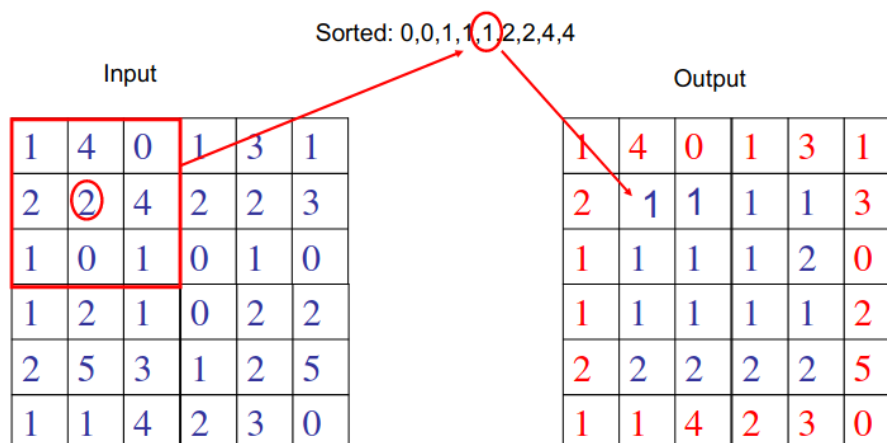
Input

1	4	0	1	3	1
2	2	4	2	2	3
1	0	1	0	1	0
1	2	1	0	2	2
2	5	3	1	2	5
1	1	4	2	3	0

Output

1	4	0	1	3	1
2	2	2	2	1	3
1	2	1	1	1	0
1	2	1	1	1	2
2	2	2	2	2	5
1	1	4	2	3	0

Median filtriranje je nelinearna metoda koja se koristi za uklanjanje šuma sa slika. Široko je rasprostranjena jer je vrlo efikasna u uklanjanju šuma uz očuvanje ivica. Posebno je efikasna metoda u uklanjanju šuma tipa „sol i biber“. Median filter radi tako što se kreće kroz sliku piksel po piksel, zamjenjujući svaku vrijednost srednjom vrijednošću susjednih piksela. Obrazac susjeda se naziva "kernel" (ranije objašnjeno). Median se izračunava tako što se prvo sortiraju sve vrijednosti piksela iz prozora u numerički red, a zatim se piksel koji se razmatra zamjenjuje srednjom (median) vrijednošću piksela.



Kada smo objasnili sve tražene filtere u ovom zadatku. Preostala je realizacija python koda. U zadatku je navedeno da je filtere potrebno primjeniti na zašumljenu sliku. Prema tome, navest ćemo neke tipove šuma s kojima se možemo susresti:

- Gaussian Noise, Rayleigh Noise, Erlang (Gamma) Noise, Uniform Noise, Salt and Paper Noise itd.

Pošto u zadatku nije navedeno, nećemo se fokusirati na analizu svakog od navedenih šumova. U nastavku je predstavljen python kod za navedeni zadatak:

```
import cv2 as cv
import numpy as np
import random

def sp_noise(image, prob):
    # Add salt and pepper noise to image
    # prob: Probability of the noise
    output = np.zeros(image.shape, np.uint8)
    thres = 1 - prob
    for i in range(image.shape[0]):
        for j in range(image.shape[1]):
            rdn = random.random()
            if rdn < prob:
                output[i][j] = 0
            elif rdn > thres:
                output[i][j] = 255
            else:
                output[i][j] = image[i][j]

    return output

# Import image
img = cv.imread('images/lena.jpg')
cv.imshow('Original', img)
cv.waitKey(0)
```

```

# Convert to grayScale
gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
cv.imshow('Gray', gray)
cv.waitKey(0)

# Noise
noise_img = sp_noise(gray, 0.05)
cv.imshow('Salt and Paper noise', noise_img)
cv.waitKey(0)

# Avergaing filter
average = cv.blur(noise_img, (3, 3))
cv.imshow('Average blur', average)
cv.waitKey(0)

# Gaussian
gaussian = cv.GaussianBlur(noise_img, (3, 3), 0)
cv.imshow('Gaussian blur', gaussian)
cv.waitKey(0)

# Median blur , good for salt and pepper noise
median = cv.medianBlur(noise_img, 3)
cv.imshow(' Median ', median)
cv.waitKey(0)

```

U ovom primjeru ćemo, kao i u prethodnom, kao sliku koristiti poznatu Lenu. Prvo smo sliku Lene konvertovali u grayscale sliku. Nakon toga smo na sliku dodali salt and paper šum. Pošto se u zadatku traži da se navedeni algoritmi primjene na zašumljenu sliku, u ovom slučaju smo napisali funkciju `sp_noise` koja impementira salt and paper šum. U nastavku je prikazana slika Lene sa navedenim šumom:



Nakon toga smo na sliku sa šumom primjenili navedene filtere te dobili sljedeća rješenja:

Average filter i gaussian blur:



Nakon toga smo primjenili Median filter (koji je pogodan za uklanjanje salt and paper šuma) i dobili sljedeći rezultat:



Iz ovog možemo primjetiti da je median filter izuzetno pogodan za otklanjanje salt and paper šuma, dok Gaussian blur i Average filter nisu. Razlog zašto je Median dobar u ovom slučaju jeste što se vrši sortiranje u niz, a pošto znamo da je salt and paper šum takav da na sliku dodaje bijele piksele (255) i crne piksele (0),

ovaj filter crne piksele postavlja na početak niza, a bijele na kraj, te na taj način neutrališe njihovo djelovanje. Kao osobinu Average filtera smo naveli da on traži prosječnu vrijednost svih piksela u susjedstvu, te ako imamo neko veliko odstupanje npr. Paper – crni piksel unutar bijelih piksela, navedeni piksel će napraviti relativno veliki uticaj na susjedne piksele. Zbog toga navedeni filter nije pogodan za ovaj problem. Također, vidimo da ni gaussov filter nije pogodan za navedeni problem, a razlog toga je, kao i u prethodnom slučaju, rad samog algortima.

U nastavku je dat kod u kojem smo pokazali sposobnost Gaussovog filtera za bluring efekat:

```
# Gaussian effect for bluring
gaussian = cv.GaussianBlur(gray, (5, 5), cv.BORDER_DEFAULT)
cv.imshow('Gaussian bluring', gaussian)
cv.waitKey(0)
```

Rezultat nam je dao sljedeću sliku:



Sa navedene slike možemo vidjeti da smo uz pomoć Gaussian filtera sa kernelom dimenzija (5*5) uspješno zamutili sliku. Povećanjem dimenzija kernela, slika postaje sve više i više zamućena,

Rješenje: Zadatak 3

U ovom zadatku je potrebno navesti i pokazati filtere za detekciju ivica. Na laboratorijskim vježbama smo za detekciju ivica koristili sljedeće filtere: Laplacian filter, Sobel filter i Canny filter.

Laplasov filter je detektor ivica koji se koristi za izračunavanje drugih izvoda slike, mjereći brzinu kojom se prve derivacije/izvodi mijenjaju. Ovo određuje da li je promjena u vrijednostima susjednih piksela od ruba ili kontinuirane progresije. Kerneli Laplasovog filtera obično sadrže negativne vrijednosti u unakrsnim elementima, centrirano unutar niza. Uglovi su ili nulte ili pozitivne vrijednosti. Centralna vrijednost može biti negativna ili pozitivna. Sljedeći kernel predstavlja primjer kernela kod Laplasovog filtera:

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Sobel operator izvodi 2-D mjerenje prostornog gradijenta na slici i tako naglašava regije visoke prostorne frekvencije koje odgovaraju ivicama. Obično se koristi za pronalaženje približne apsolutne magnitude gradijenta u svakoj tački na ulaznoj slici u sivim tonovima. Barem u teoriji, operator se sastoji od para 3×3 konvolucionih kernela kao što je prikazano na narednoj slici. Drugi kernel je u suštini isti kao prvi, samo zarotiran za 90 stupeni. Ovo je vrlo slično Roberts Cross operatoru.

-1	0	+1
-2	0	+2
-1	0	+1

Gx

+1	+2	+1
0	0	0
-1	-2	-1

Gy

Ovi kerneli su dizajnirani da maksimalno reaguju na ivice koje se kreću verikalno i horizontalno u odnosu na mrežu piksela, po jedan kernel za svaku od dvije okomite orijentacije. Kerneli se mogu posebno primijeniti na ulaznu sliku, kako bi se proizvela odvojena mjerenja komponente gradijenta u svakoj orijentaciji (nazvali smo ih Gx i Gy). Oni se zatim mogu kombinovati zajedno kako bi se pronašla apsolutna veličina gradijenta u svakoj tački i orijentacija tog gradijenta.

Canny operater je dizajniran da bude optimalan detektor ivica (prema određenim kriterijumima --- postoje i drugi detektori koji takođe tvrde da su optimalni u odnosu na malo drugačije kriterijume). Kao ulaz uzima sliku u sivoj skali, a kao izlaz proizvodi sliku koja prikazuje pozicije praćenih diskontinuiteta intenziteta.

Canny operater radi u višestepenom procesu. Prije svega, slika je izgladnena Gaussovom konvolucijom. Zatim se jednostavan 2-D operator prve derivacije (nešto kao Roberts Cross) primjenjuje na izgladnenu sliku kako bi se istakli dijelovi slike s visokim prvim prostornim derivatima. Rubovi stvaraju izbočine na slici magnitude gradijenta. Algoritam zatim prati vrh ovih grebena i postavlja na nulu sve piksele koji se zapravo ne nalaze na vrhu grebena kako bi se dobila tanka linija u izlazu, proces poznat kao ne-maksimalna supresija. Proces praćenja pokazuje histerezu koju kontrolišu dva praga: T1 i T2, sa T1 > T2. Praćenje može početi samo u

tački na grebenu višem od T1. Praćenje se zatim nastavlja u oba smjera od te točke sve dok visina grebena ne padne ispod T2. Ova histereza pomaže da se osigura da se bučne ivice ne razbiju na više fragmenata ivica.

Kada smo analizirali svaki od navedenih detektora ivica, napisan je sljedeći python kod:

```
import cv2 as cv
import numpy as np

img = cv.imread('images/lena.jpg')
cv.imshow('Original', img)
cv.waitKey(0)

gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
cv.imshow('Gray', gray)
cv.waitKey(0)

# Lablacian filter
lap = cv.Laplacian(gray, cv.CV_64F)
lap = np.uint8(np.absolute(lap))
cv.imshow('Laplacian', lap)
cv.waitKey(0)

# Sobel filter
sobelx = cv.Sobel(gray, cv.CV_64F, 1, 0)
sobely = cv.Sobel(gray, cv.CV_64F, 0, 1)
combined_sobel = cv.bitwise_or(sobelx, sobely)
cv.imshow('Sobelx', sobelx)
cv.waitKey(0)
cv.imshow('Sobely', sobely)
cv.waitKey(0)
cv.imshow('Combined Sobel', combined_sobel)
cv.waitKey(0)

canny = cv.Canny(gray, 150, 175)
cv.imshow('Canny', canny)
cv.waitKey(0)
```

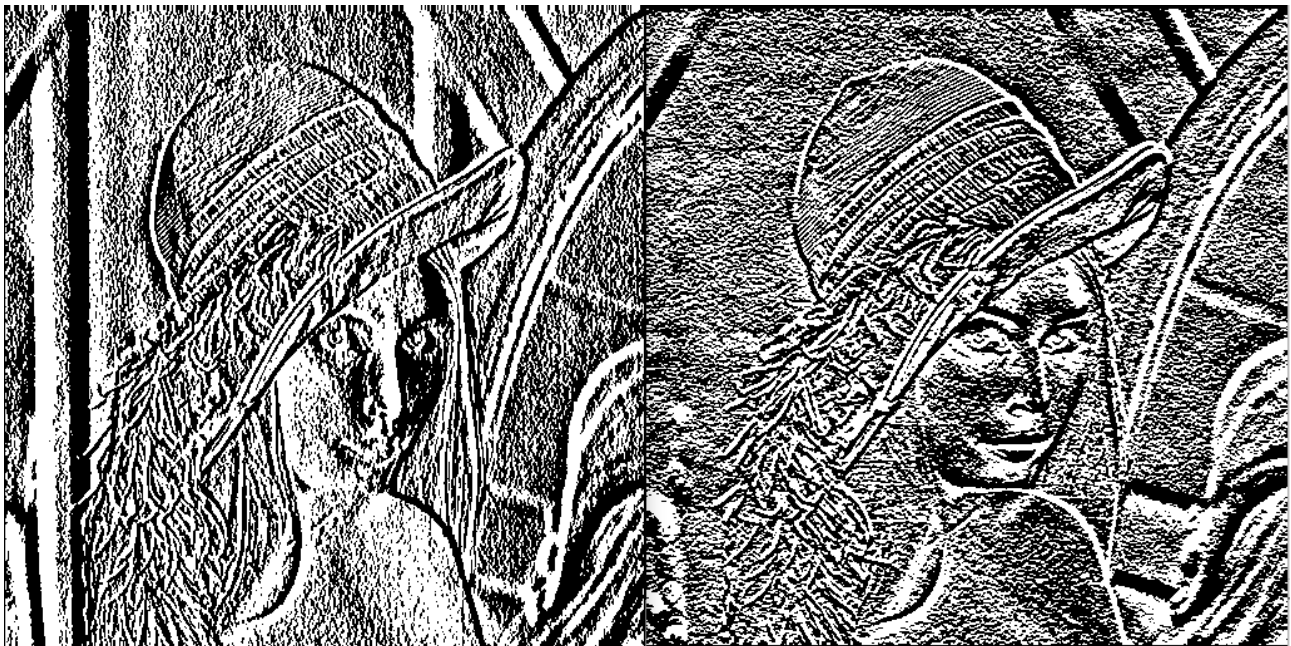
U ovom zadatku smo prvo učitali sliku (Lena, kao i u prethodnim zadacima), te nakon toga izvršili konverziju slike u grayscale. Potom smo primjenili svaki od ranije navedenih operatora (detektori ivica). Laplasov detektor ivica smo dobili pozivom funkcije Laplacian koja kao parametre prima sliku nad kojom želimo izvršiti operaciju, te odgovarajući specifični parametar CV_64F (dokumentacija: *If you want to detect both edges, better option is to keep the output datatype to some higher forms, like cv.CV_16S, cv.CV_64F etc, take its absolute value and then convert back to cv.CV_8U*).

Laplasov filter je dao sljedeći rezultat za detekciju ivica:



Nakon toga, primjenili smo ranije objašnjeni Sobelov filter. Prvo smo kernele posebno primijeniti na ulaznu sliku, kako bi se proizvela odvojena mjerenja komponente gradijenta u svakoj orijentaciji. Nakon toga smo ih kombinovali zajedno kako bi se pronašla apsolutna veličina gradijenta u svakoj tački i orijentacija tog gradijenta.

Sobelx i sobely su prikazani respektivno na narednoj slici:



Potom, kao što je ranije navedeno, prikazana je kombinacija primjene navedena dva kernela:



Kao posljednji detektor ivica, korišten je Canny operator. Primjenom ovog operatora smo dobili sljedeći rezultat:

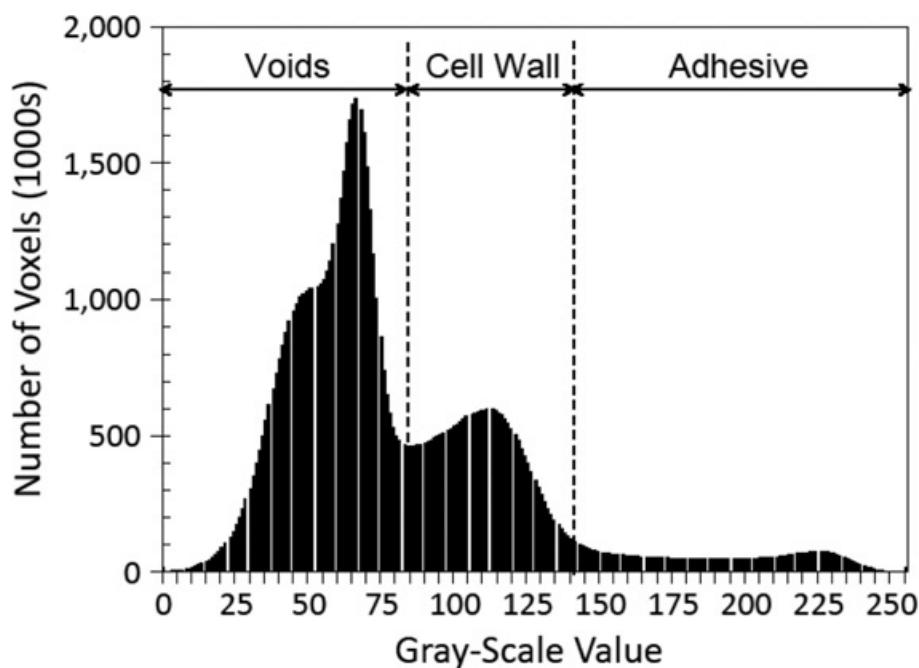


Rješenje: Zadatak 4

Histogram je vrsta grafa koji se široko koristi u matematici, posebno u statistici. Histogram predstavlja frekvenciju pojave specifičnih pojava koje se nalaze unutar određenog raspona vrijednosti, a raspoređene su u uzastopnim i fiksnim intervalima. Učestalost pojavljivanja podataka predstavljena je šipkom, pa stoga jako liči na grafikon.

Jedna od stvari koja nas zanima na digitalnoj fotografiji jest koliko su snimljeni pikseli svijetli ili tamni, odnosno osvjetljenje fotografije. Osvjetljenje pojedinog piksela je predstavljeno cjelobrojnomo, pozitivnom vrijednošću između 0 i 255. 0 predstavlja potpuno tamno, a 255 potpuno svijetlo. Ako ne govorimo o fotografiji u boji nego samo o sivoj skali (engl. grayscale) 0 predstavlja crnu boju, a 255 bijelu. Kada se radi o fotografijama u boji tada je logično najtamnija nijansa određene boje označena s 0, a najsvjetlija nijansa s 255.

Histogram zapravo promatra fotografiju u cjelini i određuje koliki broj piksela ima određeno (isto) osvjetljenje. To znači da će na osi apscisa biti vrijednosti osvjetljenja od 0 do 255, a na osi ordinata broj piksela koji imaju odgovarajuće osvjetljenje. Valja primijetiti da je i ovdje prisutna, za histograme karakteristična podjela na kategorije, s tom razlikom da sada kategorija nije raspon između dvaju brojeva nego kvantizirana vrijednost između 0 i 255. Na narednoj slici prikazan je primjer histograma digitalne fotografije. Dio histograma gdje je smješten najveći broj vrijednosti osvjetljenja naziva se raspon tonova (engl. tonal range). Raspon tonova jako varira za različite fotografije i ne postoji neki idealni histogram koji bi trebalo primijenjivati na sve fotografije.



Kada smo objasnili histogram grayscale slike, potrebno je napisati odgovarajući python kod koji će prikazati neke grayscale slike i njihove histograme, te na osnovu datih histograma ćemo napisati odgovarajuću analizu. Python kod za navedeni slučaj je dat u nastavku:

```

import cv2 as cv
import matplotlib.pyplot as plt

# Image read
img = cv.imread('images/lena.jpg')
cv.imshow('Original', img)
cv.waitKey(0)

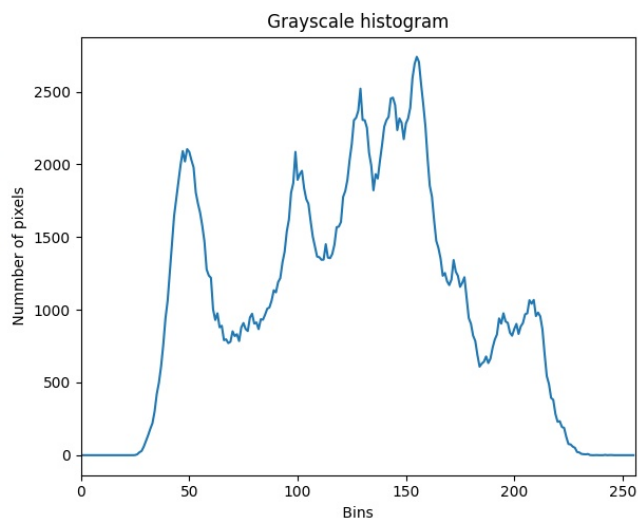
# Grayscale
gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
cv.imshow('Gray', gray)
cv.waitKey(0)

gray_hist = cv.calcHist([gray], [0], None, [256], [0, 256])
plt.figure()
plt.title(' Grayscale histogram ')
plt.xlabel(' Bins ')
plt.ylabel(' Nummber of pixels ')
plt.plot(gray_hist)
plt.xlim([0, 256])

plt.show()

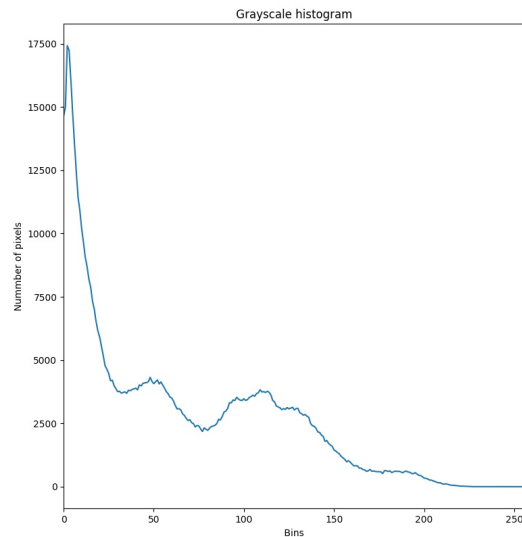
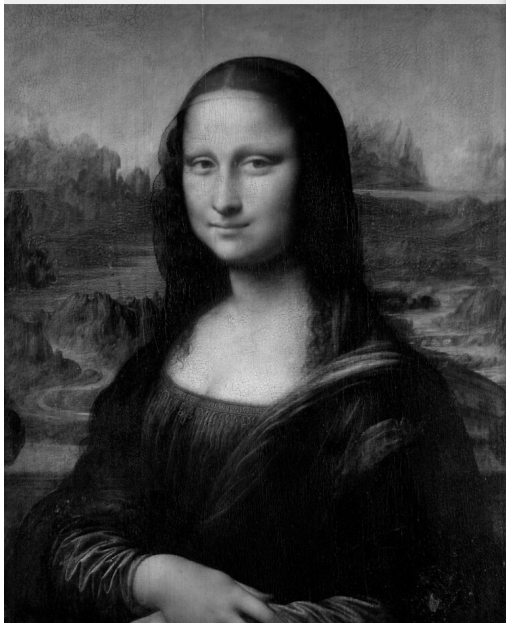
```

U ovom zadatku smo prvo učitali sliku i konvertovali je u grayscale sliku. Nakon toga smo za proračun histograma koristili funkciju `calcHist` (argumenti su navedeni u dokumentaciji, pretežno za C++). Za sliku Lene iz prethodnih primjera, dobili smo sljedeći histogram:



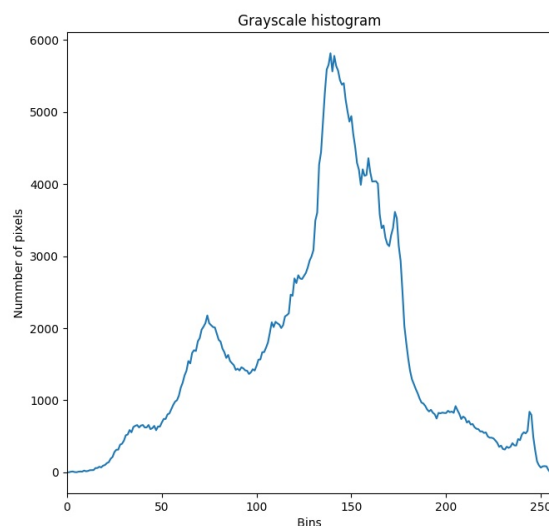
Ono što možemo primjetiti, jeste to da na slici Lene, prema ovoj analizi, nemamo potpuno crnih i potpuno bijelih piksela, te da najviše piksela ima vijednost između nekih 30 do 225. Sada ćemo prikazati histogram za još neke slike:

Histogram slike MolaLisa-grayscale:



Sa histograma ove slike možemo vidjeti da na slici dominira crna boja, te da imamo izuzetno malo svijetlih piksela.

Histogram slike Napoleona:



Nakon što smo analizirali i objasnili histogram grayscale slike, sada ćemo se posvetiti metodama dobijanja grayscale slike.

Grayscale je proces pretvaranja slike iz drugih prostora boja, npr. RGB, CMYK, HSV, itd. do nijansi sive. Ona varira između potpuno crne i potpuno bijele.

Važnost sive boje:

- Smanjenje dimenzija: Na primjer, u RGB slikama postoje tri kanala u boji i imaju tri dimenzije dok su slike u nijansama sive jednodimenzionalne.
- Smanjuje složenost modela: Razmislite o treniranju neuronskog članka o RGB slikama od 10x10x3 piksela. Ulazni sloj će imati 300 ulaznih čvorova. S druge strane, istoj neuronskoj mreži će biti potrebno samo 100 ulaznih čvorova za slike u sivim tonovima.
- Da bi drugi algoritmi radili: Mnogi algoritmi su prilagođeni da rade samo na slikama u sivim tonovima, npr. Funkcija Canny koja služi za detektovanje ivica unapred implementirana u OpenCV biblioteci radi samo na slikama u nijansama sive itd.

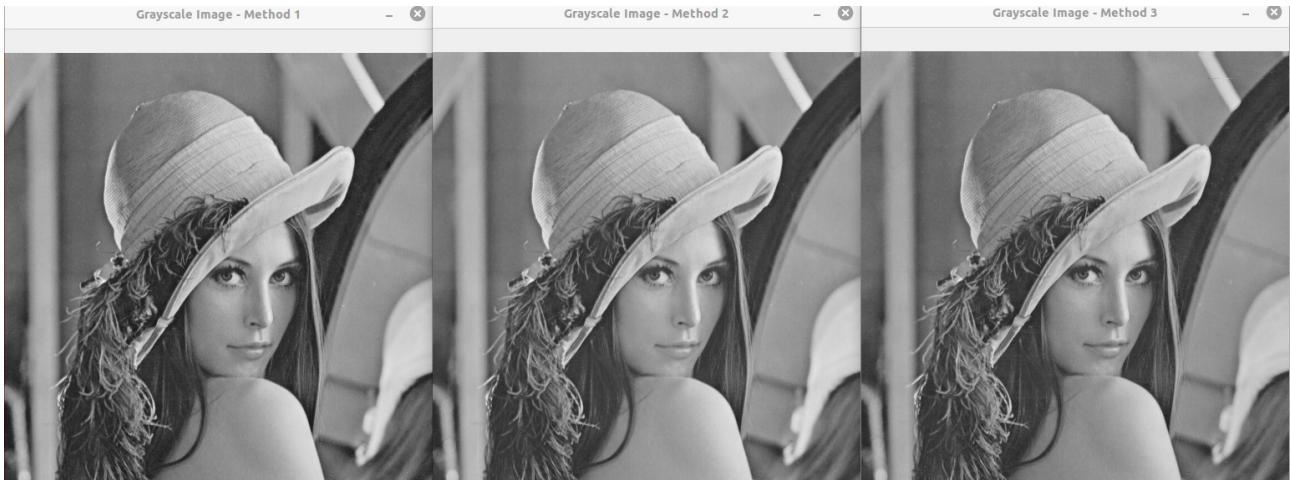
U nastavku ćemo prikazati tri metoda za dobijanje grayscale slike:

- Metoda koja koristi cv.cvtColor() funkciju
- Metoda koja koristi cv.imread() funkciju sa dodatnim flag-om 0
- Metoda koja koristi manipulaciju pixela (Average metod)

Python kod za navedene metode je prikazan u nastavku:

```
import cv2 as cv
# Method 1
img = cv.imread('images/lena.jpg')
cv.imshow('Original', img)
cv.waitKey(0)
gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
cv.imshow('Grayscale Image - Method 1', gray)
cv.waitKey(0)
# Method 2
img_gray = cv.imread('images/lena.jpg', 0)
cv.imshow('Grayscale Image - Method 2', img_gray)
cv.waitKey(0)
# Method 3
img = cv.imread('images/lena.jpg')
# Obtain the dimensions of the image array using the shape method
(row, col) = img.shape[0:2]
Take the average of pixel values of the BGR Channels to convert the colored image to grayscale image
for i in range(row):
    for j in range(col):
        # Find the average of the BGR pixel values
        img[i, j] = sum(img[i, j]) * 0.33
cv.imshow('Grayscale Image - Method 3', img)
cv.waitKey(0)
```


Pokretanjem ovog koda smo dobili sljedeće slike:



Možemo primjetiti da smo svakom od metoda kao rezultat dobili istu sliku. Koju od navedenih metoda ćemo koristiti zavisi od ličnog izbora. Najčešće se kao izbor koristi prva metoda.

Za dobijanje crno-bijele slike koristimo sljedeći kod:

```
# Threshold
threshold, thresh = cv.threshold(gray, 100, 255, cv.THRESH_BINARY)
cv.imshow(' Thresholded ', thresh)

threshold, thresh_inv = cv.threshold(
    gray, 100, 255, cv.THRESH_BINARY_INV)
cv.imshow(' Thresholded inverse ', thresh_inv)

adaptive_tresh = cv.adaptiveThreshold(
    gray, 255, cv.ADAPTIVE_THRESH_MEAN_C, cv.THRESH_BINARY, 11, 3)
cv.imshow(' Adaptive thresholding ', adaptive_tresh)
cv.waitKey(0)
```

Navedeni kod smo dodali na kod za histogram. Pomoću funkcije `threshold` dobijamo odgovarajuću crno-bijelu sliku. Pri čemu kao argumente funkcije unosimo sliku koju želimo obraditi, te unosimo prag koji želimo da nam bude granica između bijele i crne boje, također unosimo i parametar koji je specifičan za ovu funkciju. Adaptivni threshold dobijamo pozivom funkcije `adaptiveThreshold` koji također ima svoje specifične parametre.

Pokretanjem navedenog koda smo dobili sljedeći rezultat. Prvo ćemo prikazati rezultat funkcije `threshold` pri čemu prva slika ima parametar: `cv.THRESH_BINARY`, dok druga slika ima parametar: `cv.THRESH_BINARY_INV`. Slike su prikazane u nastavku:



Funkcija `adaptiveThreshold` nam je dala sljedeći rezultat:



Ono što je izuzetno bitno naglasiti jeste to da smo granicu u ovom slučaju dobili očitanjem sa histograma. Sada ćemo navedene metode primjeniti i na sliku Napoleona čiji smo histogram ranije analizirali.

Threshold:



Adaptive threshold:



Zadatak 5: Rješenje

Binarne slike mogu sadržavati brojne nedostatke. Konkretno, binarne regije proizvedene jednostavnim pragom su izobličene šumom i teksturom. Morfološka obrada slike ima za cilj uklanjanje ovih nesavršenosti uzimajući u obzir formu i strukturu slike. Ove tehnike se mogu proširiti na slike u sivim tonovima.

Morfološka obrada slike je skup nelinearnih operacija povezanih s oblikom ili morfologijom karakteristika na slici. Prema Wikipediji, morfološke operacije se oslanjaju samo na relativno sređivanje vrijednosti piksela, a ne na njihove numeričke vrijednosti, te su stoga posebno pogodne za obradu binarnih slika. Morfološke operacije se također mogu primijeniti na slike u sivim tonovima tako da su njihove funkcije prijenosa svjetlosti nepoznate i stoga njihove apsolutne vrijednosti piksela nisu od interesa ili su od manjeg značaja. Morfološke tehnike ispituju sliku s malim oblikom ili šablonom koji se zove strukturni element. Element strukturiranja se pozicionira na svim mogućim lokacijama na slici i uspoređuje se sa odgovarajućim susjedstvom piksela. Neke operacije testiraju da li se element "uklapa" u susjedstvo, dok druge testiraju da li "udara" ili siječe susjedstvo.


Dilatacija dodaje piksele granicama objekata na slici, dok erozija uklanja piksele na granicama objekata. Broj piksela koji se dodaju ili uklanjaju iz objekata na slici ovisi o veličini i obliku strukturnog elementa koji se koristi za obradu slike.

Dilatacija je proces u kojem se binarna slika proširuje iz svog originalnog oblika. Način na koji se binarna slika širi određen je elementom strukturiranja. Ovaj strukturni element je manje veličine u odnosu na samu sliku, a obično je veličina koja se koristi za strukturni element 3x3. Proces dilatacije je sličan procesu konvolucije, to jest, strukturni element se reflektuje i pomjera s lijeva na desno i odozgo prema dolje, pri svakom pomaku; proces će tražiti sve slične piksele koji se preklapaju između strukturnog elementa i binarne slike. Ako postoji preklapanje, onda će pikseli ispod središnje pozicije strukturnog elementa biti okrenuti u 1 ili crno.

Erozija je kontra-proces dilatacije. Ako dilatacija povećava sliku, onda erozija smanjuje sliku. Način na koji se slika skuplja određen je elementom strukturiranja. Strukturni element je obično manji od slike veličine 3x3. Ovo će osigurati brže vrijeme izračunavanja u poređenju sa većom veličinom strukturnog elementa. Gotovo slično procesu dilatacije, proces erozije će pomjeriti strukturni element s lijeva na desno i odozgo prema dolje. Na središnjoj poziciji, označenoj središtem strukturnog elementa, proces će tražiti da li postoji potpuno preklapanje sa strukturnim elementom ili ne. Ako nema potpunog preklapanja, tada će središnji piksel označen središtem strukturnog elementa biti postavljen na bijelo ili na 0.

Kada smo objasnili šta je morfološka obrada slike, te šta su erozija i dilatacija, preostaje nam da napišemo odgovarajući python kod, kojim ćemo implementirati navedene funkcionalnosti. Kod je dat u nastavku dokumenta:

```
import cv2 as cv

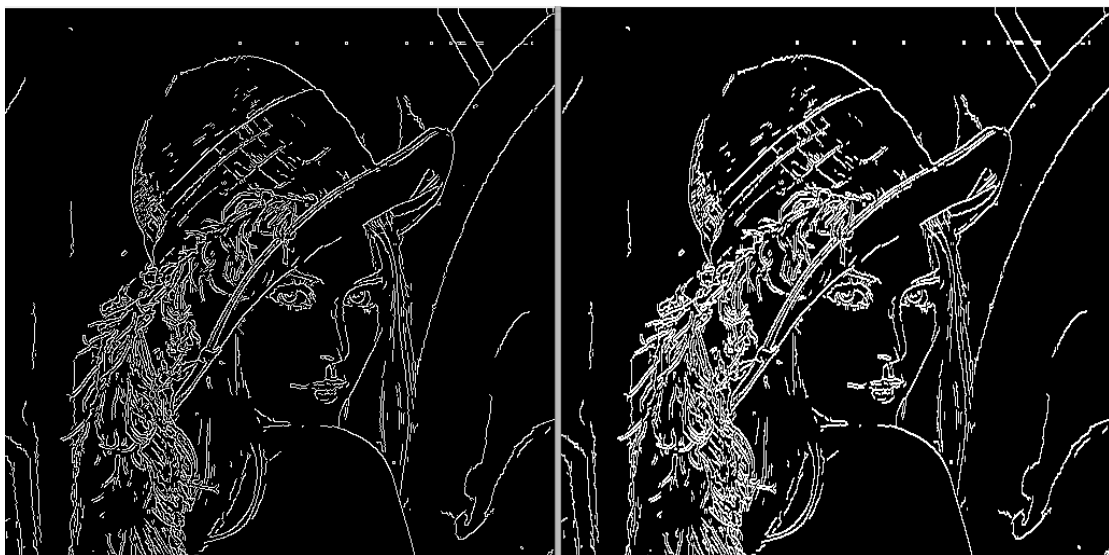
# Image read
img = cv.imread('images/lena.jpg')

cv.imshow('Original', img)
cv.waitKey(0)
```

```

# Grayscale
gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
cv.imshow('Gray', gray)
cv.waitKey(0)
# Gauss filter
blur = cv.GaussianBlur(img, (5, 5), cv.BORDER_DEFAULT)
cv.imshow(' Blured ', blur)
# canny edge detector
canny = cv.Canny(img, 125, 175)
cv.imshow(' Canny ', canny)
# dilatation
dilated = cv.dilate(canny, (7, 7), iterations=1)
cv.imshow(' Dilatation ', dilated)
# erosion
eroded = cv.erode(dilated, (7, 7), iterations=1)
cv.imshow(' Erosion ', eroded)
borders = canny - eroded
cv.imshow(' Borders ', borders)
cv.waitKey(0)
cv.waitKey(0)

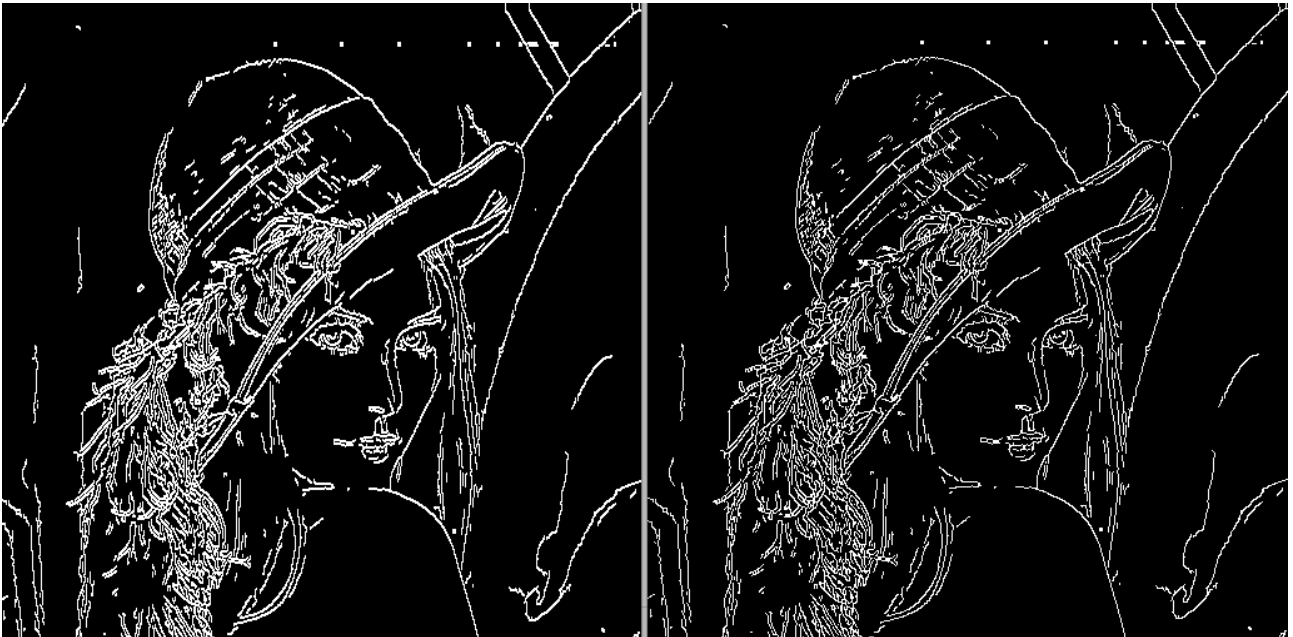
```

U ovom zadatku smo uradili sljedeće: Prvo smo učitali sliku, nakon toga smo sliku konvertovali u grayscale, te potom na nju primjenili Canny operator (objašnjen u prethodnom zadatku). Kada smo dobili canny sliku, na nju smo primjenili operator dilatacije. Za operaciju dilatacije potrebna nam je funkcija dilate koja kao parametre prima sliku nad kojom se vrši obrada, te veličina strukturnog elementa i broj iteracija (koliko je puta primjenjen algoritam – dilatacija). Na narednoj slici su prikazani rezultati canny operatora, te dilatacije koja je primjenjena na canny operator (respektivno):



Sa slike možemo primjetiti da nam je dilatacija u ovom slučaju proširila područje bijelih piksela, odnosno, djeluje nam kao da su linije podebljane.

Nakon toga smo primjenili eroziju, eroziju smo izvršili pozivom funkcije erode koja ima parametre identične kao i funkcija dilate, pri čemu se u ovom slučaju slika koje se učitava predstavlja sliku dobijenu dilatacijom i broj iteracija se odnosi na broj erozija u slici. Na narednoj slici su prikazane slike Lene dobijene dilatacijom i erozijom (respektivno):



Možemo primjetiti da smo erozijom smanjili područje bijelih piksela, odnosno, djeluje nam kao da su se linije stanjile. Još nam preostaje da dobijemo ivice. Ivice (borders) smo dobili oduzimanjem canny slike od slike dobijene erozijom (pogledati kod). Procesom oduzimanja ove dvije slike smo dobili sljedeću sliku:



Rješenje: Zadatak 6

U ovom zadatku ćemo analizirati primjer dubokog učenja (deep learning) na CIFAR-10 datasetu. Ono što je karakteristično za navedeni zadatak jeste korištenje dubokog učenja. Napredak u obradi slike je došao nakon razvijanja vještačke inteligencije, vještačkih neuronskih mreža itd.

Skup podataka CIFAR-10 (Kanadski institut za napredna istraživanja) je kolekcija slika koje se obično koriste za treniranje algoritama mašinskog učenja i kompjuterske vizije. To je jedan od najčešće korištenih skupova podataka za istraživanje mašinskog učenja. CIFAR-10 skup podataka sadrži 60.000 slika u boji 32x32 u 10 različitih klasa. 10 različitih klasa predstavljaju avione, automobile, ptice, mačke, jelene, pse, žabe, konje, brodove i kamione. Postoji 6.000 slika svake klase.

U kodu je korišten Keras za izgradnju modela klasifikacije slika treniranog na skupu podataka CIFAR-10. Model koristi sljedeće slojeve/funkcije:

- Za izradu modela: CNN, Maxpooling and Dense Layers.
- Aktivacijske funkcije: ReLU (u CNN slojevima za hendlanje piksela slike) i Softmax (za finalnu klasifikaciju).
- Za hendlanje overfitting-a: DropOut sloj
- Za normalizaciju/standardizaciju ulaza između slojeva: Batch Normalization sloj

Što se tiče samog koda, možemo ga analizirati na sljedeći način:

- Uključili smo biblioteke potrebne za navedenu aplikaciju
- Potrebno je učitati podatke iz dataseta, te podesiti podatke za treniranje i podatke za testiranje
- Potrebno je analizirati dataset koji imamo, te možemo uraditi vizualizaciju (prikaz) nekih slika iz dataseta
- Obrada podataka (postaviti vrijednosti svih piksela između 0 i 1 itd.)
- Kreiranje CNN modela pomoću Keras-a (podešavanje slojeva, biranje aktivacijskih funkcija itd)
- Kompajliranje modela
- Fitovanje modela
- Vizualizacija (Loss kriva i Accuracy kriva). Loss kriva - Poređenje gubitka u treningu sa gubitkom od testiranja u rastućim epohama. Accuracy kriva - Poređenje preciznosti treninga sa preciznošću testiranja u rastućim epohama.
- Predikcija rezultata

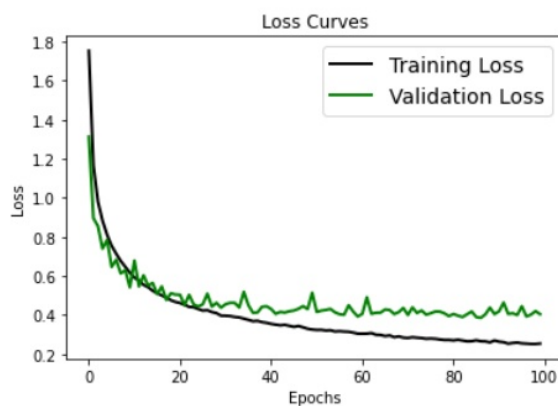
Zaključak: U ovom zadatku je kreirana neuronska mreža koja vrši klasifikaciju slika iz dataseta CIFAR-10 sa preciznošću od 88%. U analizi koda smo napisali korake koji se odnose na razvijanje date aplikacije, odnosno neuronske mreže. Generalno, vještačke neuronske mreže se mogu sastojati iz više slojeva koji su međusobno povezani, te u svaki od navedenih slojeva možemo postaviti određeni broj neurona, odrediti aktivacijsku funkciju svakog sloja itd. Navedne mreže su detaljno obrađene na predmetu Inteligentni sistemi. Ono što je karakteristično da kod navedenih mreža može doći do overfitting-a i underfitting-a.

Underfitting: Za statistički model ili algoritam mašinskog učenja se kaže da ima nedovoljno prilagođavanje (underfitting) kada ne može da uhvati osnovni trend podataka, tj. ima dobar učinak samo na podacima za treniranje, ali ima loš učinak na podacima testiranja.

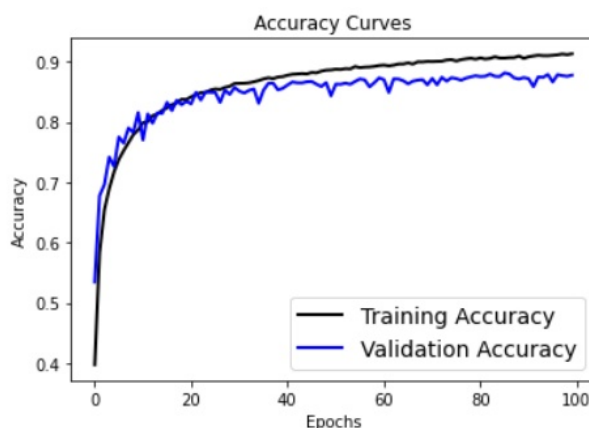
Overfitting: Za statistički model se kaže da je preopterećen (overfitovan) kada model ne daje tačna predviđanja na podacima za testiranje. Kada se model obuči s previše podataka, on počinje učiti i iz šuma i netočnih unosa podataka u naš skup podataka. Također to se dešava i kada testiranje na testnim podacima daje visoku varijansu. Tada model ne kategorizira podatke ispravno, zbog previše detalja i šuma.

Prilikom testiranja naše neuronske mreže, potrebno je izbjeći navedene dvije pojave. Pokretanjem koda iz navedenog primjera (kod dostavljen kao zadatak6.py – nije ubačen u dokument zbog velikog broja linija) dobili smo sljedeće odzive:

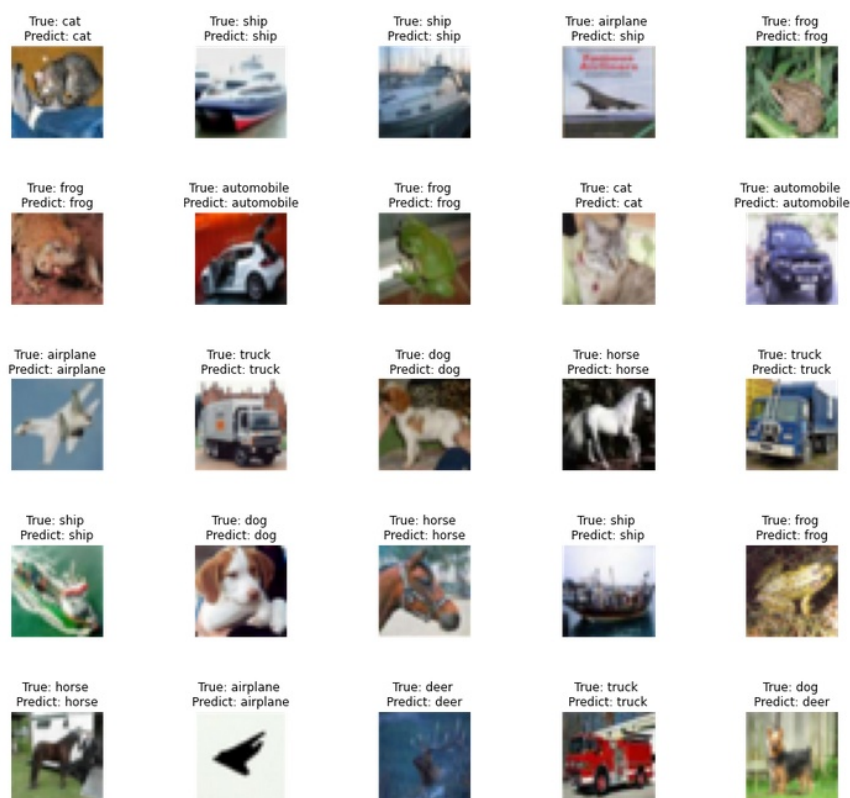
Loss kriva



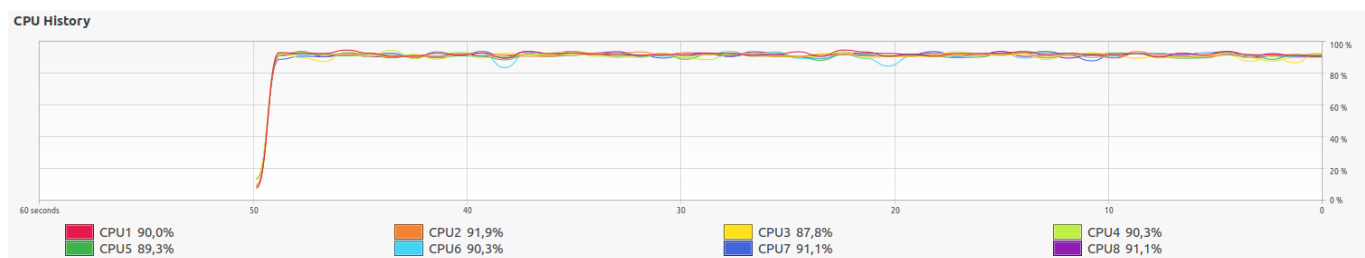
Kao što je ranije rečeno, Loss kriva predstavlja poređenje gubitka u treningu sa gubitkom od testiranja u rastućim epohama. Kada bi se zelena i crvena linija razlikovale u mnogo većoj mjeri, imali bismo underfitting, a kada bi se poklapale, odnosno, bile identične, imali bismo overfitting. Isto važi i za Accuracy krivu koja predstavlja poređenje preciznosti treninga sa preciznošću testiranja u rastućim epohama.



Na narednoj slici su predstavljene neke testne slike i njena klasifikacija u kategorije koje se nalaze u datasetu:



Možemo primjetiti da je u većini slučajeva predikcija dobra, odnosno, imamo preciznost od 88%. Navedena mreža nije istrenirana na računar koji je korišten za ovu analizu iz razloga što ima integrisanu grafičku karticu, a treniranje sa procesorom je prouzrokovala sljedeće:



Možemo primjetiti da je treniranje preopteretilo rad računara, uz sve to, za treniranje ove mreže na ovaj način (bez kvalitetne grafičke kartice) bi trajalo ekstremno dugo (možda čak i nekoliko sati).