Univerzitet u Tuzli

Fakultet elektrotehnike

Automatika i robotika

Zadaća iz predmeta

„Robotika i mašinska vizija“

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

Asistent: Mirza Hodžić

Student: Irvana Hrustić

Tuzla, juni, 2022.godine

Sadržaj

Zadatak 1

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

Rješenje:

Funkcija za rotaciju slike koju ćemo koristiti je: def rotate (img , angle , rotPoint = None)
img - naša slika
angle - ugao za koji ćemo da zarotiramo sliku
rotPoint - tačka rotacije
U našem kodu tačka rotacije je centar naše slike.

```python
import cv2 as cv
import numpy as np

#ucitamo sliku
img = cv.imread('NAR.jpg')
cv.imshow('Original', img)
#rotiramo sliku
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 (img , 60)
cv. imshow ( ' Rotated ' , rotated )
```

Illustration 1: Kod za rotaciju slike u pythonu, za 60 stepeni

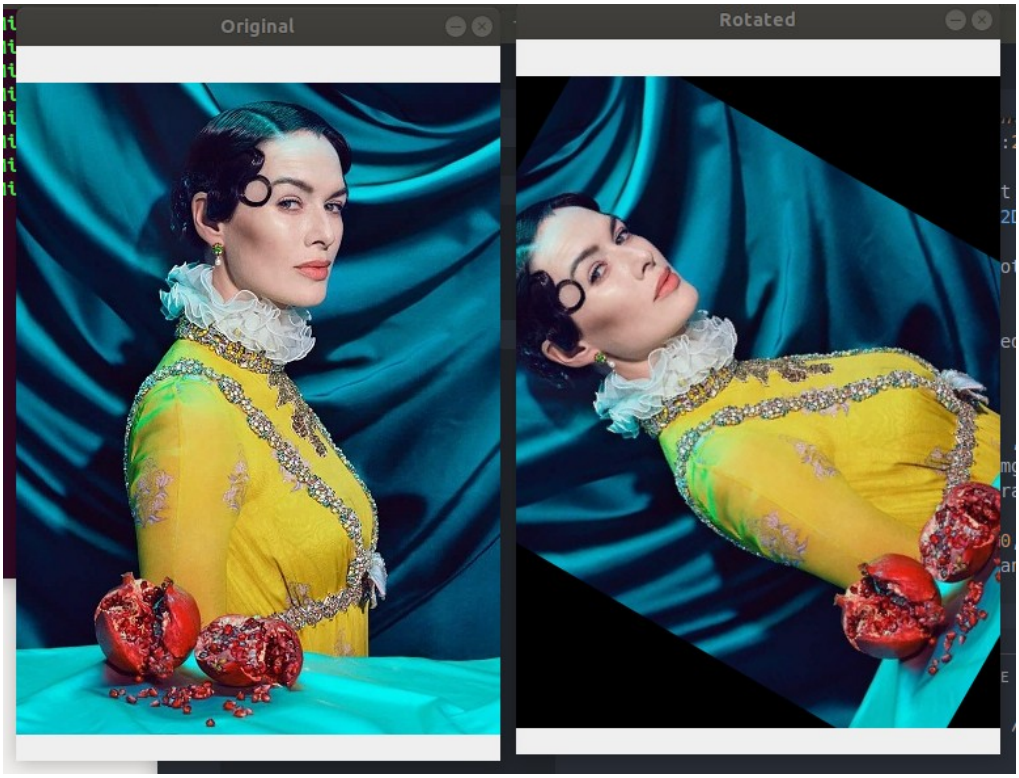


Illustration 2: Rezultat rotacije slike

Funkcija za translaciju slike koju ćemo koristiti je: def translate (img , x, y)
img - naša slika
x - translacija izvršena po x-osi
y - translacija izvršena po y-osi

```
18
19  #transliramo sliku
20  def translate (img , x, y):
21      transMat = np. float32 ([[1 ,0 ,x], [0,1,y]])
22      dimensions = (img.shape [1], img.shape [0])
23      return cv. warpAffine (img , transMat , dimensions )
24
25  translated = translate (img , 100, 150) #po x za 100, po y za 150
26  cv. imshow ( ' Translated ' , translated )
27
28  cv.waitKey(0)
```

Illustration 3: Kod za translaciju slike u python, za 100 po x-osi i za 150 po y-osi

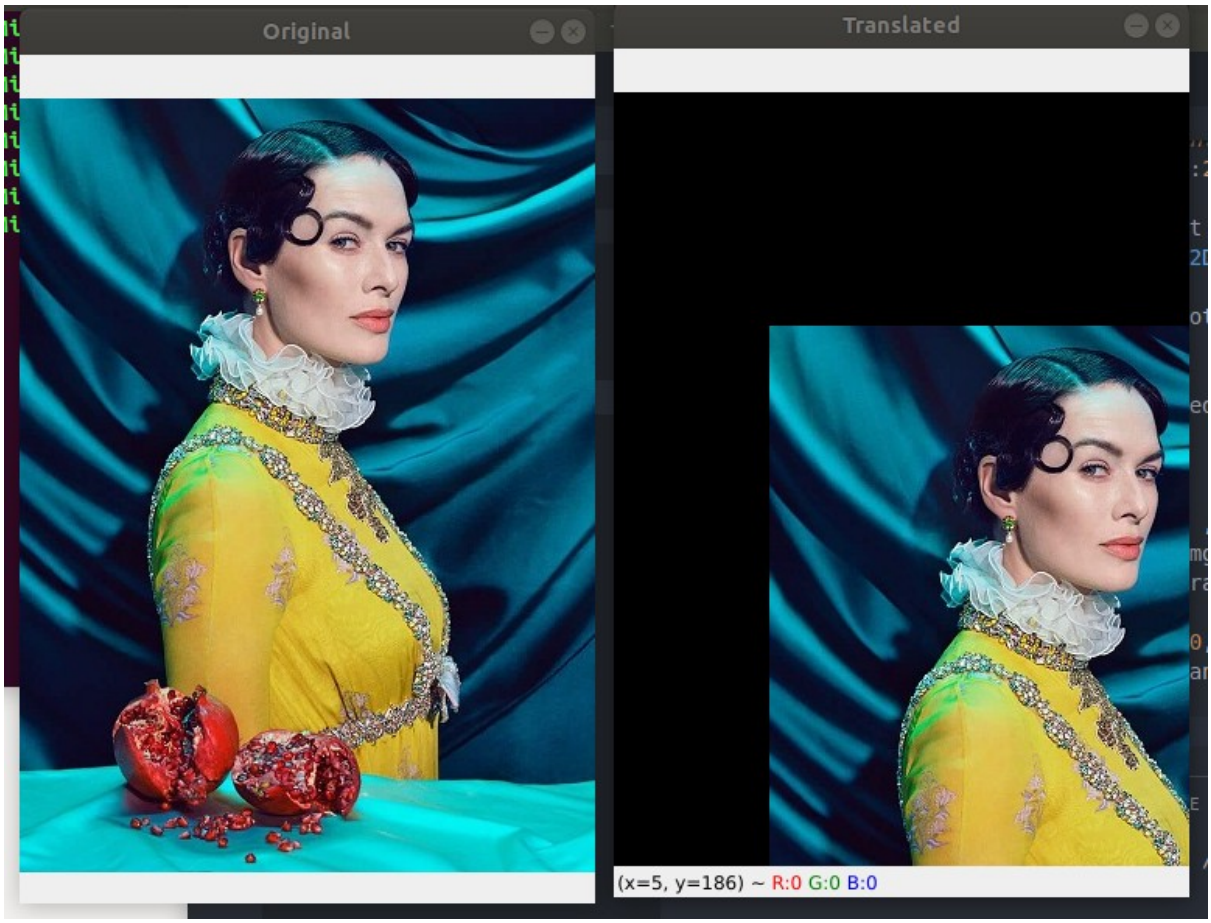


Illustration 4: Rezultat translacije slike

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.

Osnovna osobina operacija nad pikselima/tačkama je da svaka nova vrijednost piksela ovisi samo do originalnog piksela na istoj poziciji. Mogućnosti ovih operacija nad tačkama je, međutim, ograničena. One ne mogu, na primjer zagladiti ili izoštriti sliku, stoga osnovu za

poboljšanje kvaliteta slike čine metode za filtriranje slike. Filteri se koriste za uklanjanje šuma ili neželjenih elemenata na slici, dok filteri zasnovani na prvom i drugom izvodu se koriste za određivanje ivica na slici. I operacije nad tačkama i filteri izvode mapiranje 1:1 koordinata slike (tj. geometrija slike se ne mijenja). No, filteri koriste više od jednog piksela originalne slike za računanje nove vrijednosti piksela. Tradicionalna i provjerena klasifikacija filtera je na linearne i nelinearne filtere na osnovu matematičkih osobina funkcija filtera, tj. da li je rezultujuća vrijednost piksela izračunata na osnovu originalnih vrijednosti piksela primjenom linearnih ili nelinearnih izraza. Linearni filteri uključuju mean, Laplacian, Laplacian Gaussiana filtere, dok nelinearni filteri uključuju median, maksimalne, minimalne filtere, te Sobelove, Prewittove i Canny filtere. Poboljšanje kvaliteta slike može biti postignuto u dva domena: prostornom i frekventnom domenu. Prostorni domen čine svi pikseli slike. Udaljenosti na slici (u pikselima) odgovara stvarnim udaljenostima u mikrometrima, inčima i slično.

Jedna od primarnih upotreba i linearnih i nelinearnih filtera za poboljšanje slike jeste redukcija ili uklanjanje šuma, te zamagljenje (bluranje) slike. Bluranje slike se često koristi pri preprocesiranju slike kako bi se uklonili mali detalji sa slike prije ekstrakcije većih objekata.

Gaussov filter

Gaussov filter je vrlo važan iz teoretskih i praktičnih razloga. Kernel je izveden iz radijalnog simetričnog oblika kontinualne 2D Gaussove funkcije

$$f(x,y) = \frac{1}{2\pi\sigma^2}\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)$$

pri čemu je diskretna aproksimacija određena sa dva parametra:
• željene veličine kernela
• vrijednosti σ, standardne devijacije Gaussove funkcije, koji određuje širinu maske.

Gaussova maska ima zvonasti oblik, sa elementima velike vrijednosti u centru i elementima čija vrijednost simetrično opada idući prema krajevima maske. Ovo kao rezultat daje odziv koji je donekle sličan težiranom usrednjavanju, odnosno mean kernelu. Primjena Gaussovog filtera ima efekat zagladivanja slike, ali na nešto drugačiji načine nego mean filter. Prvo, stepen zagladivanja je kontrolisan izborom parametra standardne devijacije σ, a ne apsolutne veličine kernela, kao u slučaju mean filtera. Drugo, Gaussova funkcija ima osobinu da je Fourierova trasformacija takođe Gaussova funkcija, što je čini prikladnom i za filtere u frekventnom domenu. Gaussova funkcija sa velikom vrijednošću σ predstavlja niskopropusni filter koji potiskuje dio slike sa prostorno visokofrekventim sadržajem. Gaussov filter zagladuje sliku, ali uz gubitak visoko-frekventnih detalja slike.

```
zadatak2.py > ...
1   import cv2 as cv
2
3   img = cv.imread('NAR.jpg')
4   cv.imshow('Original', img)
5
6   #Gaussian blur
7   gaussian = cv.GaussianBlur(img, (5, 5), 0)
8   cv.imshow('Gaussian blur', gaussian) #less blur, godi vise ocima
9
```

Illustration 5: Kod za Gaussian filter

Averaging filter

Zaglađuje trenutnu sliku postavljajući svaki piksel jednak vrijednosti prosječnog piksela svoje specifične matrice u susjedstva.

```
10  #Avergaing filter
11  average = cv.blur(img, (5, 5))
12  cv. imshow ('Average blur', average)
13
```

Illustration 6: Kod za Averaging flter

Median filter

Kao alternativni kernel koji zaglađuje šum, ali ne zaglađuje ivice, koristi se median filter. Median filter je nelinearan filter koji prevazilazi osnovna ograničenja mean filtera, ali na račun nešto većih proračunskih troškova. Pri filtriranju, svaki piksel ulazne slike je zamijenjen sa statističkim medianom njegovog N × M susjedstva, umjesto sa srednjom vrijednošću. Ovaj filter bolje čuva detalje na ivicama, uklanja šum, posebno izolovane pojave šuma (kao što je salt and pepper šum). Median vrijednost m skupa brojeva predstavlja broj za koji je polovina brojeva manja od m i polovina brojeva veća od m, odnosno to je broj u sredini niza sortiranih brojeva. S obzirom da je median vrijednost dobijena na osnovu susjedstva posmatranog piksela, ovaj filter je mnogo robustniji na outlier-e i ne kreira nerealistične nove vrijednosti piksela. Na ovaj način se sprečava zamagljenje ivica na slici i time gubitak detalja slike. Međutim, median operator zahtijeva sortiranje vrijednosti piksela u susjedstvu za svaku lokaciju piksela na slici, a što povećava proračunske zahtjeve.

Očito je da je median filter vrlo dobar za uklanjanje impulsnog ili salt and pepper šuma uz minimalnu degradaciju kvaliteta slike. Razlog tome leži u činjenici da salt and pepper šum uključuje izolovane piksele koji su bijeli (otud so - eng. salt) ili crni (biber - eng. pepper), kao rezultat izostavljanja vrijednosti zbog greške pri prijenosu podataka. U slučaju median maske, većina piksela ispod maske imaju mali intenzitet, tako da rezultujući piksel nije pogoden pikselom impulsnog šuma.

```
14  #Median blur, good for salt and pepper noise
15  median = cv.medianBlur(img, 5)
16  cv.imshow('Median blur', median)
17
18  cv.waitKey(0)
```

Illustration 7: Kod za Median filter

Usporedba

Illustration 9: Original

Illustration 8: Rezultat Gaussian filtera

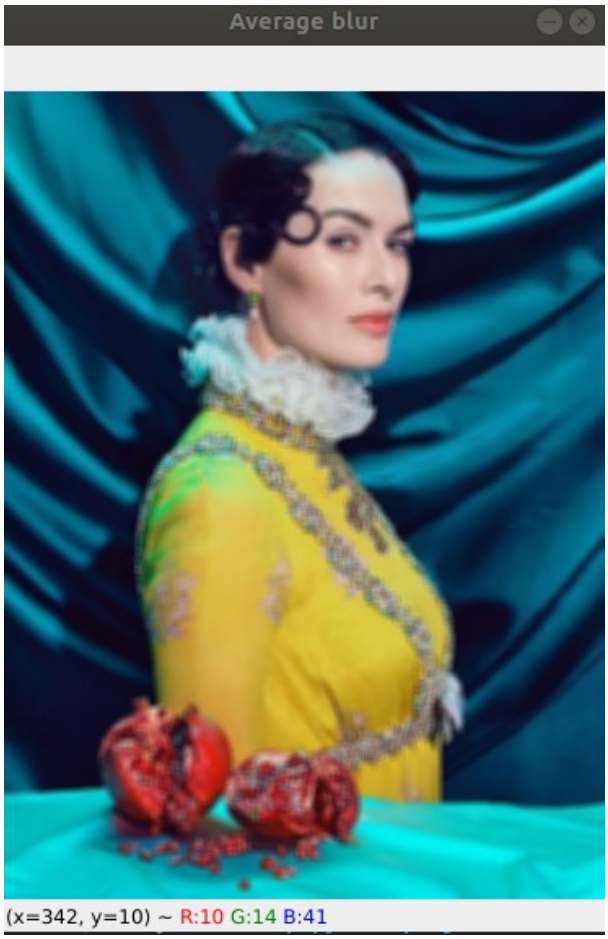


Illustration 10: Rezultat Averaging filtera

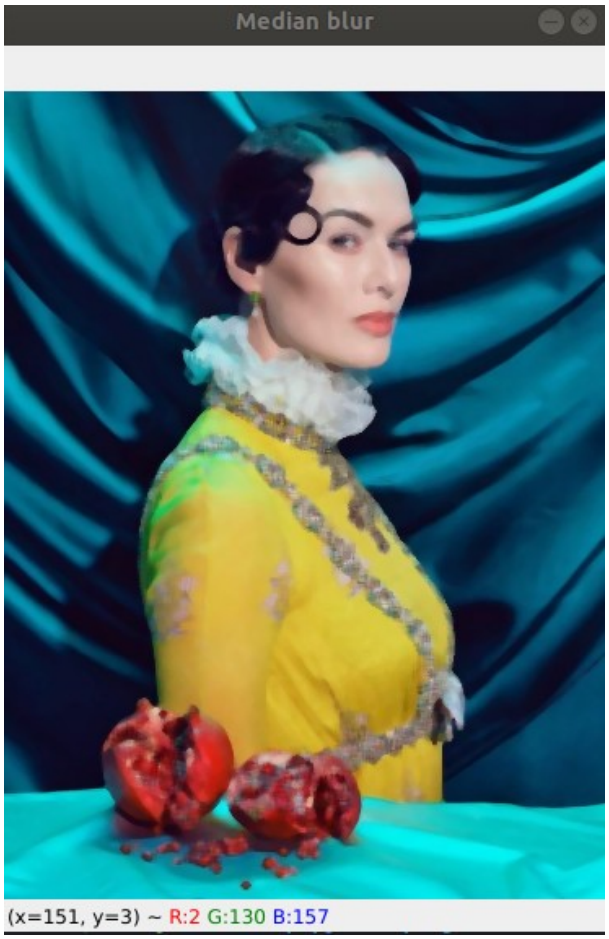


Illustration 11: Rezultat Median filtera

Zadatak 3

Navesti i pokazati u pythonu filtere za detekciju ivica.

Filteri za detekciju ivica koje ćemo da koristimo su Laplacian, Sobel i Canny edge detector.

Laplacian filter

```python
zadatak3.py > ...
import cv2 as cv
import numpy as np

#ucitavanje slike
img = cv.imread('NAR.jpg')
cv.imshow('Original', img)

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

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

```

Illustration 12: Kod za Laplacian filter

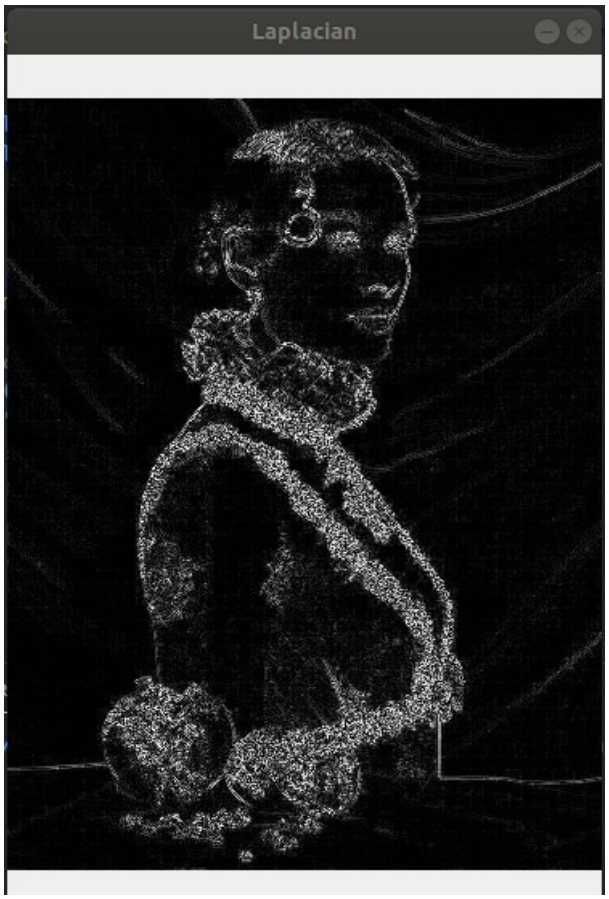


Illustration 13: Rezultat Laplacian filtera

Sobel filter

```
15
16  #Sobel
17  sobelx = cv.Sobel(gray, cv.CV_64F, 1, 0)
18  sobely = cv.Sobel(gray, cv.CV_64F, 0, 1)
19  combined_sobel = cv.bitwise_or(sobelx, sobely)
20
21  cv.imshow('Sobel X', sobelx)
22  cv.imshow('Sobel Y', sobely)
23  cv.imshow('Combined Sobel', combined_sobel)
24
```

Illustration 14: Kod za Sobel filter

Illustration 15: Rezultat Sobel filtera

Canny edge detector

```
24
25  #canny edge detector
26  canny = cv.Canny(gray, 150, 175)
27  cv.imshow('Canny', canny)
28
29  cv.waitKey(0)
```

Illustration 16: Kod za Canny edge detector

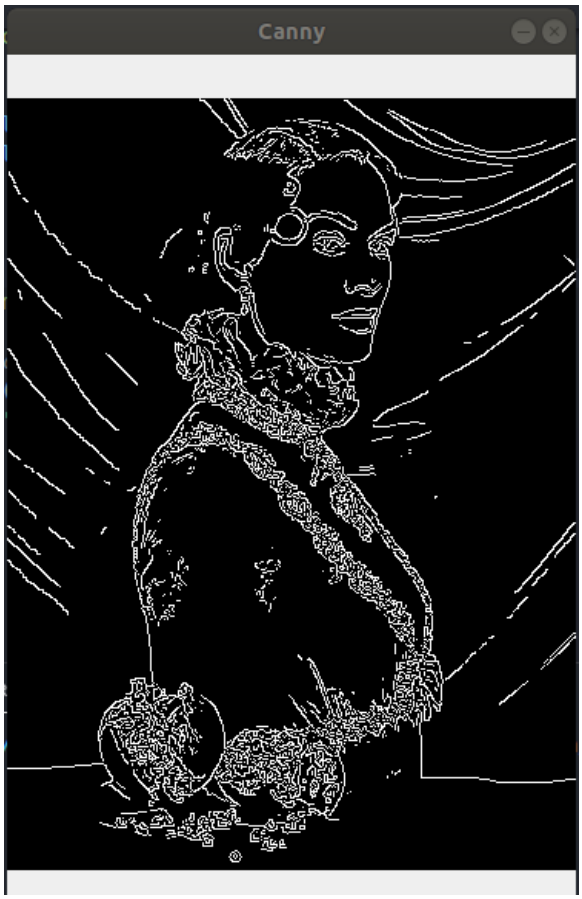


Illustration 17: Rezultat Canny filtera

Zadatak 4

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

Histogrami predstavljaju frekvencijsku raspodjelu i opisuju učestalost vrijednosti intenziteta koji se pojavljuju na slici. Ovaj koncept se može jednostavno opisati primjenom grayscale slike. Histogram *h* za grayscale sliku *I* sa vrijednostima intenziteta u području *I (m, n) € [0, K - 1]* će sadržavati ukupno *K* elemenata. Svaki element histograma je definisan kao

h (i) = broj piksela n_i *u I sa intenzitetom i za svako* $0 \leq i < K$.

Prema tome, *h (0)* je broj piksela sa vrijednošću *0, h (1)* je broj piksela sa vrijednošću *1*, i tako dalje. Rezultat računanja histograma je jednodimenzionalni vektor *h* dužine *K*. Budući da histogram ne sadrži informacije o tome gdje se na slici nalaze pikseli uračunati za svaki

element *h (i),* histogram ne sadrži informacije o prostornom rasporedu piksela slike. Stoga slika ne može biti rekonstruisana na osnovu histograma, jer nedostaju prostorne informacije.

Iz navedenog se može zaključiti da više različitih slika mogu imati potpuno isti histogram. Glavna funkcija histograma je ilustrovanje statističkih informacija o slici (tj. raspodjele vrijednosti intenziteta slike) u kompaktnom obliku. Računanje histograma 8-bitne grayscale slike koja sadrži vrijednosti intenziteta izmedu *0* i *255* se izvodi brojanjem koliko puta se svaka siva vrijednost (*0-255)* pojavljuje na slici.

Svaki “bin”histograma je inkrementiran svaki put kada je njegova vrijednost pronadena na slici. Slika koja ima *K* mogućih vrijednosti će imati tačno *K* elemenata u nizu histograma. S obzirom da bin predstavlja podjelu skale intenziteta, moguće je za računanje histograma slike koristiti manji broj binova nego što slika ima mogućih intenziteta. Ovo je naročito bitno za 16-bitne i 32-bitne slike, gdje korištenje po jednog elemenata za svaki mogući intenzitet na slici nije praktično. Umjesto toga, jedan bin/element histograma će predstavljati raspon vrijednosti intenziteta. Ova tehnika se često naziva “binning”, jer prikazuje raspon vrijednosti intenziteta piksela u obliku binova – kutija.

Primjenom histograma moguće je uočiti probleme sa ekspozicijom ili kontrastom. Na primjer, ako histogram slike ima veliko područje intenziteta na jednom kraju koji su slabo ili potpuno neiskorišteni, dok na drugom kraju ima mnogo elemenata sa visokim vrijednostima, tada se može reći da slika ima nepravilnu ekspoziciju.

Histogram grayscale slike

histogram.py > ...

```python
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt

img = cv.imread('NAR.jpg')
#cv.imshow('Original', img)

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

#cv.waitKey(0)

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

plt.show()
```

Illustration 18: Kod za histogram slike

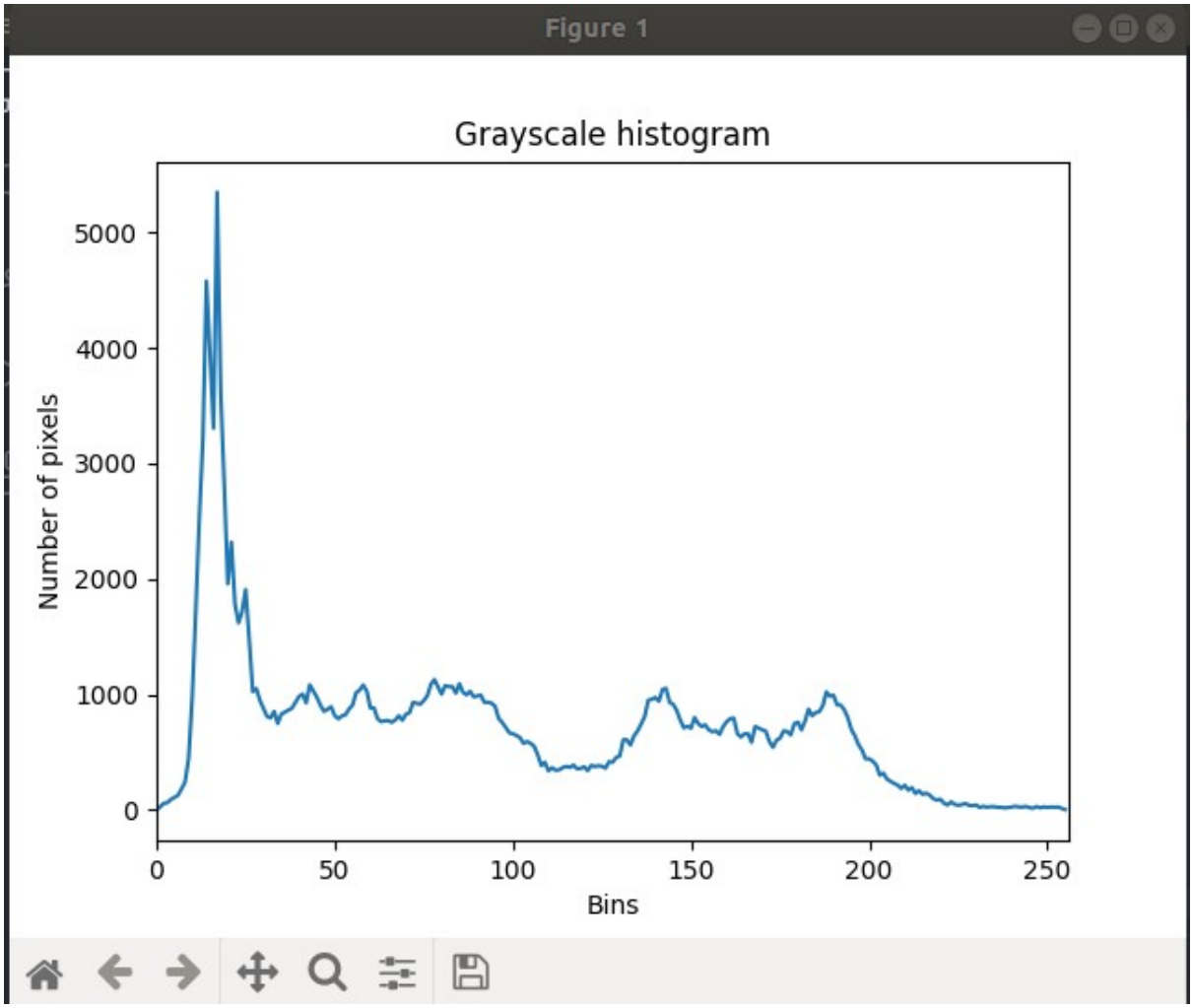


Illustration 19: Grayscale histogram

Crno-bijelu sliku mozemo dobiti koristeci thresholding.

```python
import cv2 as cv
import matplotlib.pyplot as plt

img = cv.imread('NAR.jpg')
cv.imshow('Original', img)

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

#simple thresholding
threshold, thresh = cv.threshold(gray, 100, 255, cv.THRESH_BINARY)
cv.imshow('Simple Thresholded', thresh)
#inverse of image
threshold, thresh_inv = cv.threshold(gray, 100, 255, cv.THRESH_BINARY_INV)
cv.imshow('Simple Thresholded Inverse', thresh_inv)

#adaptive thresholding
adaptive_thresh = cv.adaptiveThreshold(gray, 255, cv.ADAPTIVE_THRESH_GAUSSIAN_C,
cv.THRESH_BINARY, 13, 3) # GAUSSIAN or MEAN, ovisi od situacije
cv.imshow('Adaptive Thresholding', adaptive_thresh)

cv.waitKey(0)
```

Illustration 20: Kod za crno-bijelu sliku

Simple thresholding i simple inverse thresholding

Illustration 21: Simple thresholding

Illustration 22: Simple inverse thresholding

Adaptive thresholding i grayscale slika

Illustration 24: Adaptive thresholding

Illustration 23: Grayscale slika

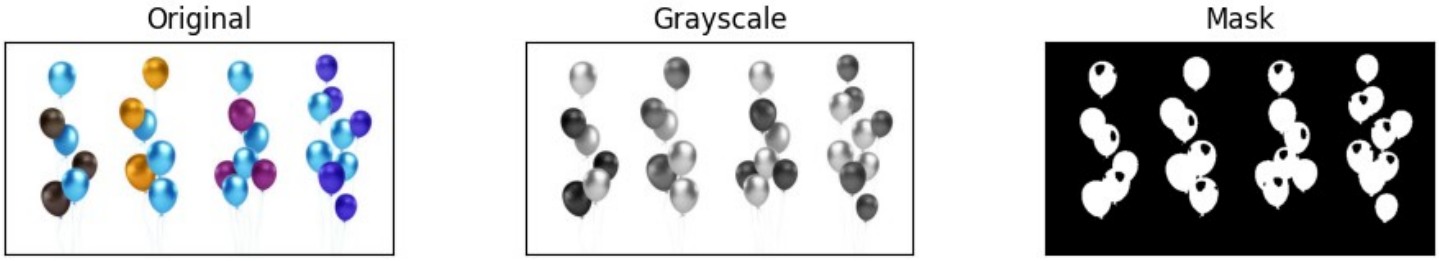
Zadatak 5

Navesti i pokazati u pythonu metode morfološke obrade slike.

Ovdje imamo prikazanu originalu sliku, grayscale te slike, te masku te slike. Maska je binarna slika koja sadrži kvadrate od piksela.

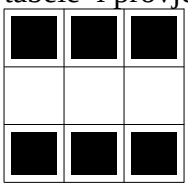
```python
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt

img = cv.imread('balonii.jpg')
gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
_, mask = cv.threshold(gray, 220, 255, cv.THRESH_BINARY_INV)

kernal = np.ones((3,3), np.uint8) # 2*2 square shape applied on our image on the place of spots
```

Illustration 25: Kod za grayscale i mask od image(img)



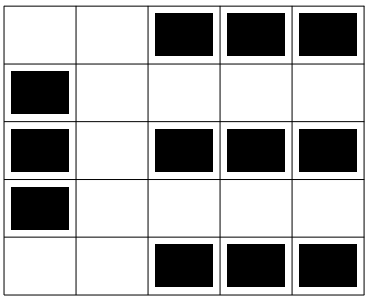
Primjer za eroziju

Ova tabela nam predstavlja kernal u našem kodu. S njom ćemo da prelazimo preko donje tabele i provjeriti preklapanje.

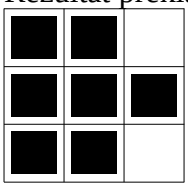


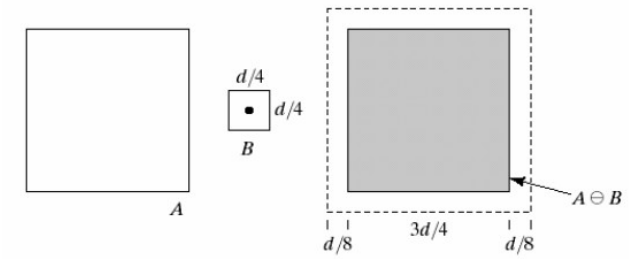
Uslov:

if kompletno preklapanje IZLAZ – 1
if djelimicno preklapanje IZLAZ – 0
if nema preklapanja IZLAZ - 0

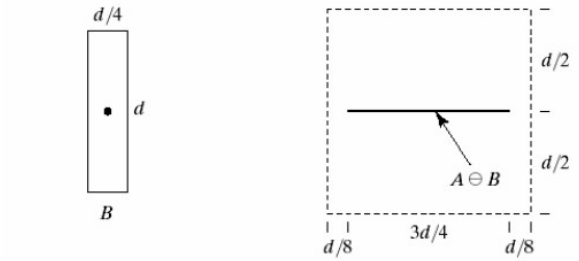


Rezultat preklapanja:





$$A \ominus B = \{z | (B)_z \subseteq A\}$$

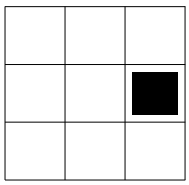


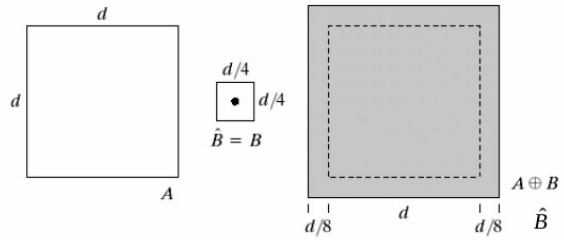
Erozija je skup pomeraja, z, takav da se struktuirajući element, B, transliran za z i skup A preklapaju u svim elementima

Primjer za dilataciju

Rezultat preklapanja:

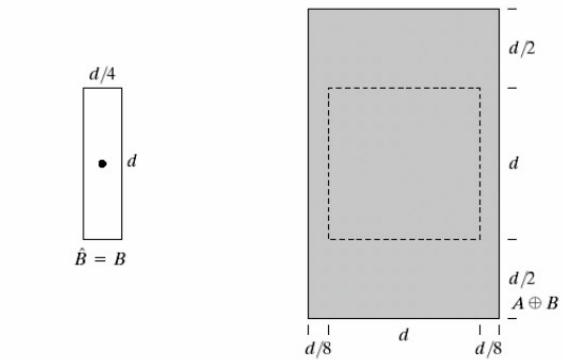
Uslov:
if kompletno preklapanje IZLAZ – 1
if djelimicno preklapanje IZLAZ – 1
if nema preklapanja IZLAZ - 0





$$A \oplus B = \{z | (\hat{B})_z \cap A \neq \emptyset\}$$

$$A \oplus B = \{z | [(\hat{B})_z \cap A] \subseteq A\}$$



B – struktuirajući element

Dilatacija je skup pomeraja, z, za koje se refleksija struktuirajućeg elementa, $\hat{B}$, i skup A preklapaju bar u jednom elementu

```
10
11  dilation = cv.dilate(mask, kernal, iterations = 4) #povecati broj iteracija, ili dimenzije kvadrata/pravougaonika iznad
12  erosion = cv.erode(mask, kernal, iterations = 2)
13
14  opening = cv.morphologyEx(mask, cv.MORPH_OPEN, kernal) #erozija, pa dilatacija
```

Illustration 26: Kod za eroziju, dilataciju i otvaranje

Otvaranje i zatvaranje

Otvaranje:
- glača konture objekta
- kida uske prevlake
- eliminiše tanke izbočine

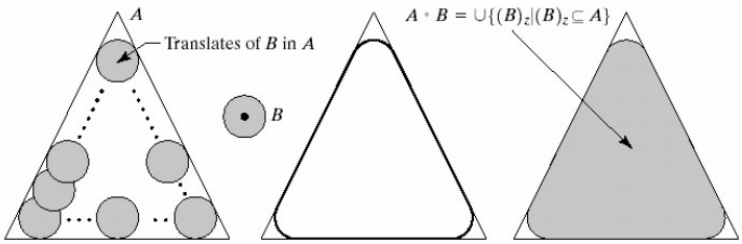
$$A \circ B = (A \ominus B) \oplus B$$

Zatvaranje:
- glača delove konture objekta
- spaja uske prekide
- eliminiše male šupljine
- popunjava prekide u konturi

$$A \bullet B = (A \oplus B) \ominus B$$

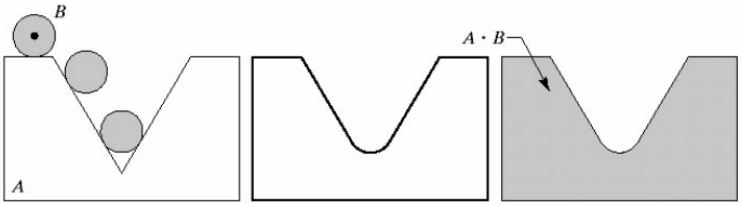
Geometrijska interpretacija otvaranja i zatvaranja

Otvaranje:



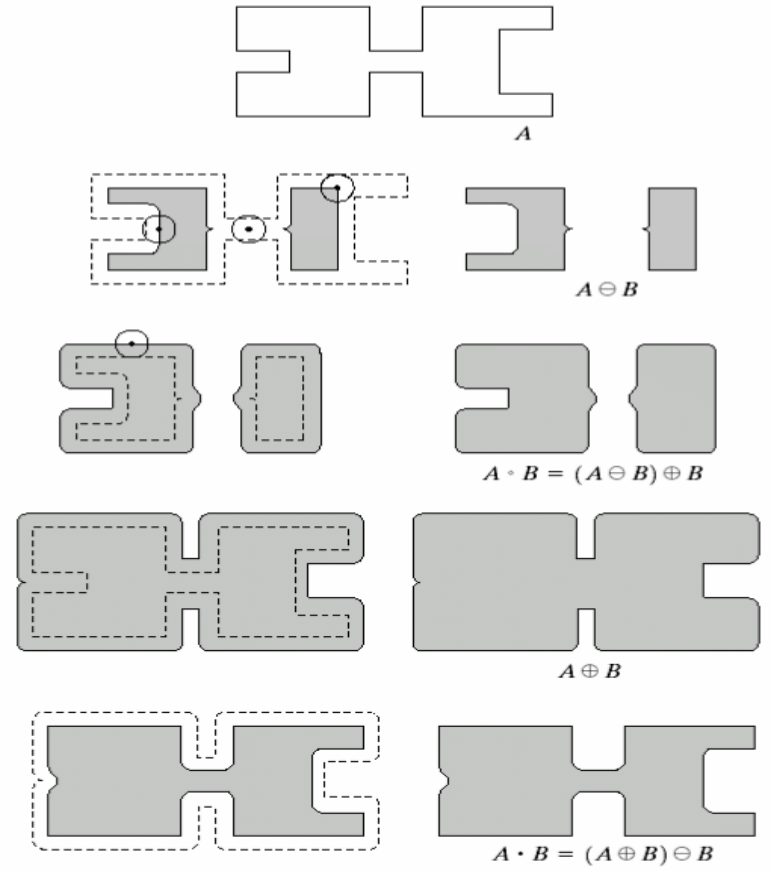
Struktuirajući element se ''kotrlja'' duž unutrašnje granice objekta A

Zatvaranje:



Struktuirajući element se ''kotrlja'' duž spoljnje granice objekta A

Ilustracija otvaranja i zatvaranja



Morfološki gradijent

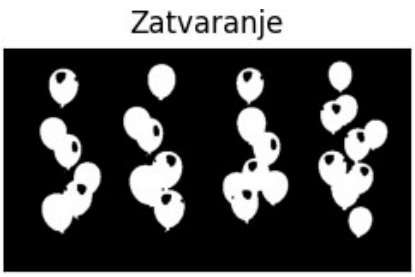
$$g = (f \oplus b) - (f \ominus b)$$

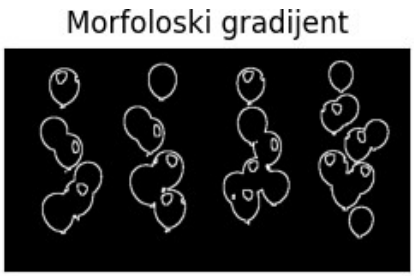
“Top-hat” transformacija

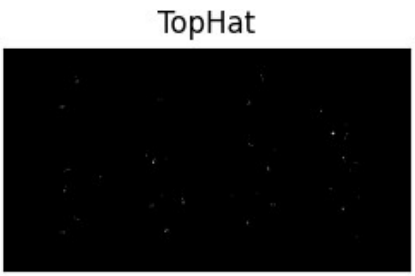
$$h = f - (f \circ b)$$

```
15   closing = cv.morphologyEx(mask, cv.MORPH_CLOSE, kernal) #dilatacija, pa erozija //bolji rezultati
16
17   mg = cv.morphologyEx(mask, cv.MORPH_GRADIENT, kernal) #razlika izmedju erozije i dilatacije slike
18   th = cv.morphologyEx(mask, cv.MORPH_TOPHAT, kernal) #razlika izmedju grayscale slike(u nasem slucaju) i otvaranja slike
19
```

Illustration 27: Kod za zatvaranje, morfoloski gradijent i top-hat transformaciju



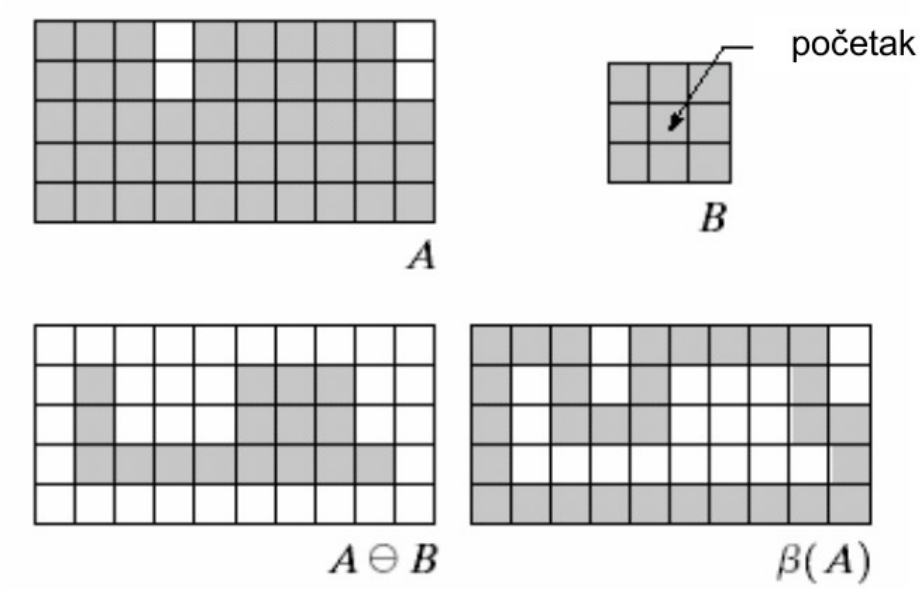




Izdvajanje granica

Granica skupa A je razlika skupa A i njegove erozije:

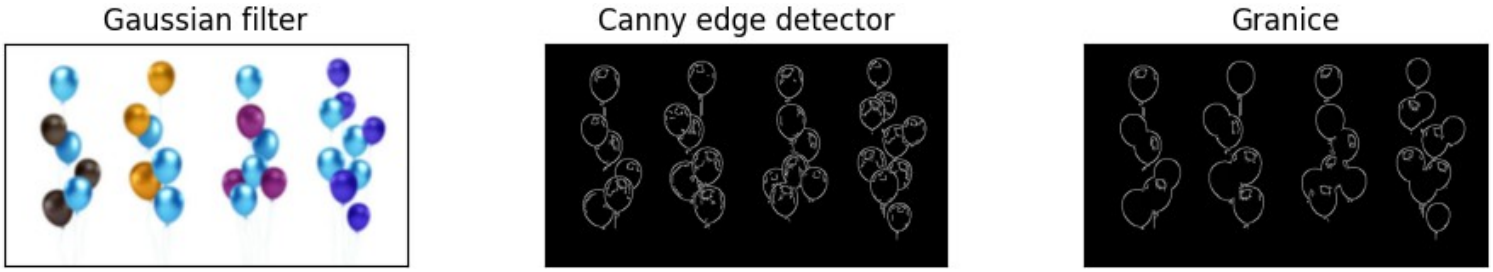
$$\beta(A) = A - A \ominus B$$



Gauss i Canny edge detector smo obradili u zadatku3.

```
gauss = cv.GaussianBlur(img, (5,5), cv.BORDER_DEFAULT) #Gaussian filter
canny = cv.Canny(img,  100, 150) #Canny edge detector
borders = canny - erosion #granice

titles = ['Original', 'Grayscale', 'Mask', 'Dilatacija', 'Erozija', 'Otvaranje', 'Zatvaranje', 'Morfoloski gradijent',
'TopHat', 'Gaussian filter', 'Canny edge detector', 'Granice']
images = [img, gray, mask, dilation, erosion, opening, closing, mg, th, gauss, canny, borders]

for i in range(12):
  plt.subplot(4, 3, i+1), plt.imshow(images[i], 'gray')
  plt.title(titles[i])
  plt.xticks([]), plt.yticks([])

plt.show()
```

Illustration 28: Kod za izdvajanje granica, GAussian filter i Canny edge detector



Ostali primjeri morfološke obrade slike su: “Hit-or-Miss”, izdvajanje povezanih komponenata, podebljavanje, morfološko glačanje, popunjavanje regiona, konveksna ljuska, istanjivanje, koncept skeletona, potkresivanje, granulometrija, segmentacija tekstura.

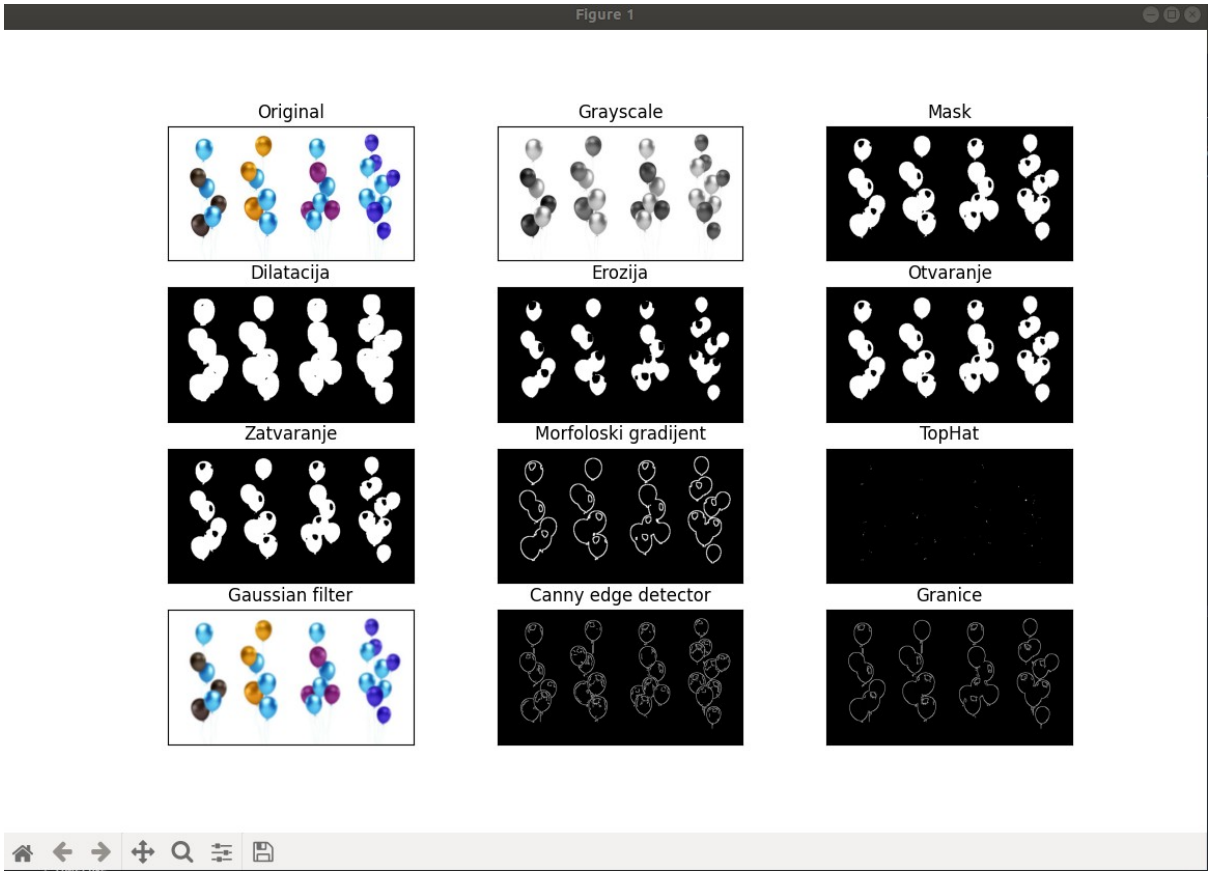


Illustration 29: Kompletan 5. zadatak

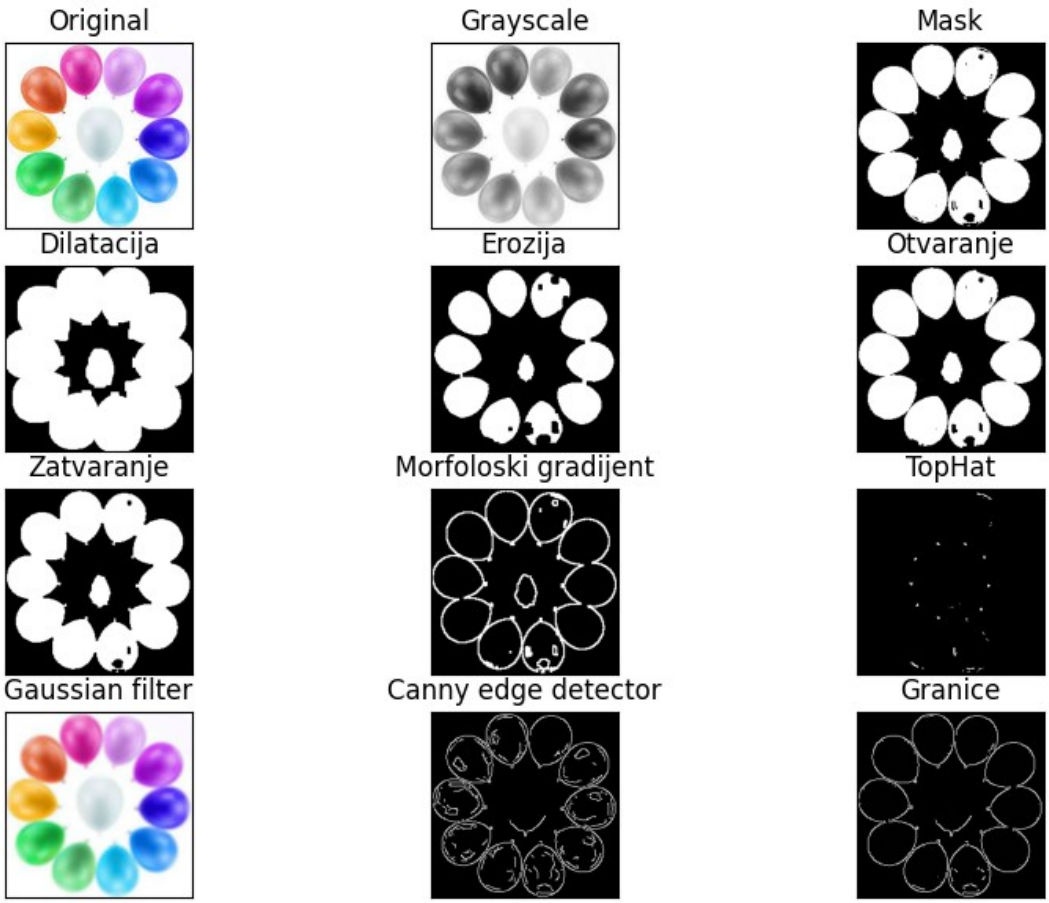


Illustration 30: Primjer2

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.

Neuronske mreže su dobre za određivanje modela kod sistema za koji nisu poznati parametri. Mana im je što ne znamo fizikalna značenja tih parametara (jedan parametar je čvor/neuron u neuronskoj mreži). Inspiracija je došla iz biologije, gdje je jedan neuron jedna nervna ćelija i ona šalje/ispusti, za neku električnu pobudu, električni impuls. Imat ćemo sistem koji se sastoji od mnogo neurona koji su poredani u slojeve.

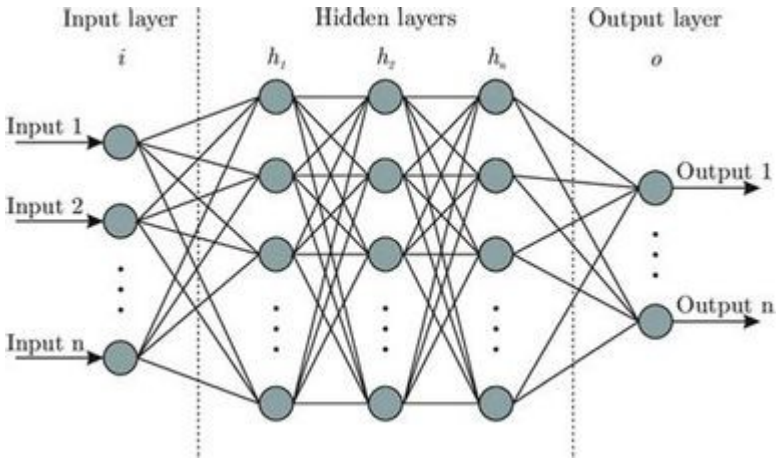


Illustration 31: Izgled neuronske mreže

Neuron računa sumu nekih impulsa koji dolaze u njega i izvrši neku funkciju. Signali koji ulaze se uvijek računaju nekim brojem. Ti brojevi su koeficijenti i parametri sistema. Nama je cilj optimizirati parametre sistema.

Neuronska mreža je crna kutija preko koje mi oponašamo ponašanje stvarnog sistema. Koristimo ih kada nam nije poznat model sistema. U mašinskoj viziji ih možemo koristiti ako pretpostavimo da nam je jedna slika multivarijabilni sistem, gdje je jedan parametar sistema neka osobina slike. Neuronske mreže možemo koristiti za klasifikaciju (npr. Support Vector Machine – SVM).

Najbolje aktivacijske funkcije za neuronske mreže su RELU funckija i SOFTMAX funckija za obradu digitalne slike. Kada sliku ubacimo u neuronsku mrežu, mreža će odrediti koja je vjerovatnoća da ta slika pripada određenoj klasi. (npr. da li se na slici nalazi ptica ili brod).

Za međuslojeve uvijek se koristi RELU funckija, a za izlazne slojeve SOFTMAX funkcija. Jedna dizajnirana neuronska mreža ne može pokriti sve naše potrebe! Potrebna je ogromna količina podataka za neuronsku mrežu. Najbolje je koristiti slike kvadratne veličine, da su što manje, a jasnije. Potrebno je istrenirati sve slike.

Pokušavamo smanjiti veličinu signala koji dolaze u određene slojeve neuronske mreže. Želimo smanjiti broj podataka, a u isto vrijeme želimo veći broj slika. Ukoliko nemamo podataka možemo generisati nove pomoću faker generatora. To će nam generisati novu vrstu podataka (npr. možemo flipovati postojeću sliku da bismo povećali broj slika). Ova metoda nije preporučljiva.

Objašnjenje koda

Biblioteke u pythonu koje služe za rad sa neuronskim mrežama su keras i tensorflow. Keras u sebi sadrži neke klase.

In [1]:
```
import keras
from keras.datasets import cifar10
from keras.models import Sequential
from keras import datasets, layers, models
from keras.utils import np_utils
from keras import regularizers
from keras.layers import Dense, Dropout, BatchNormalization
import matplotlib.pyplot as plt
import numpy as np
```

Illustration 32: Biblioete koje koristimo u pythonu za duboko učenje

In [2]:
```
(train_images, train_labels), (test_images, test_labels) = datasets.cifar10.load_data()
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170500096/170498071 [==============================] - 3s 0us/step
```

Illustration 33: Učitavanje slika

load_data() će da učita set slika iz cifar10.

In [3]:
```
# Checking the number of rows (records) and columns (features)
print(train_images.shape)
print(train_labels.shape)
print(test_images.shape)
print(test_labels.shape)
```

```
(50000, 32, 32, 3)
(50000, 1)
(10000, 32, 32, 3)
(10000, 1)
```

print(train_images. shape) će da nam ispiše veličinu slika.

Keras preko tensorflow vraća funkcije za tensorflow. Tensorflow koristi CPU, ukoliko imamo posvećenu grafičku karticu onda je najbolje nju koristiti jer će uveliko smanjiti vrijeme obrade.

Ako smo postigli tačnost od 60.0% to znači da smo dobro dizajnirali neuronsku mrežu, dok sve preko ovisi od kvaliteta podataka, njihove raznovrsnosti, te broja podataka za obradu.

In [4]:
```
# Checking the number of unique classes
print(np.unique(train_labels))
print(np.unique(test_labels))
```

```
[0 1 2 3 4 5 6 7 8 9]
[0 1 2 3 4 5 6 7 8 9]
```

Illustration 34: Ispis labela

In [5]:
```
# Creating a list of all the class labels
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
               'dog', 'frog', 'horse', 'ship', 'truck']
```

Illustration 35: Kreiraje niza

Kreiran je niz od 10 elemenata.

In [6]:
```
# Visualizing some of the images from the training dataset
plt.figure(figsize=[10,10])
for i in range (25):    # for first 25 images
  plt.subplot(5, 5, i+1)
  plt.xticks([])
  plt.yticks([])
  plt.grid(False)
  plt.imshow(train_images[i], cmap=plt.cm.binary)
  plt.xlabel(class_names[train_labels[i][0]])

plt.show()
```

Illustration 36: Kod za prikaz slika

plt.show() funckija nam služi da prikažemo slike iz datasheet-a.

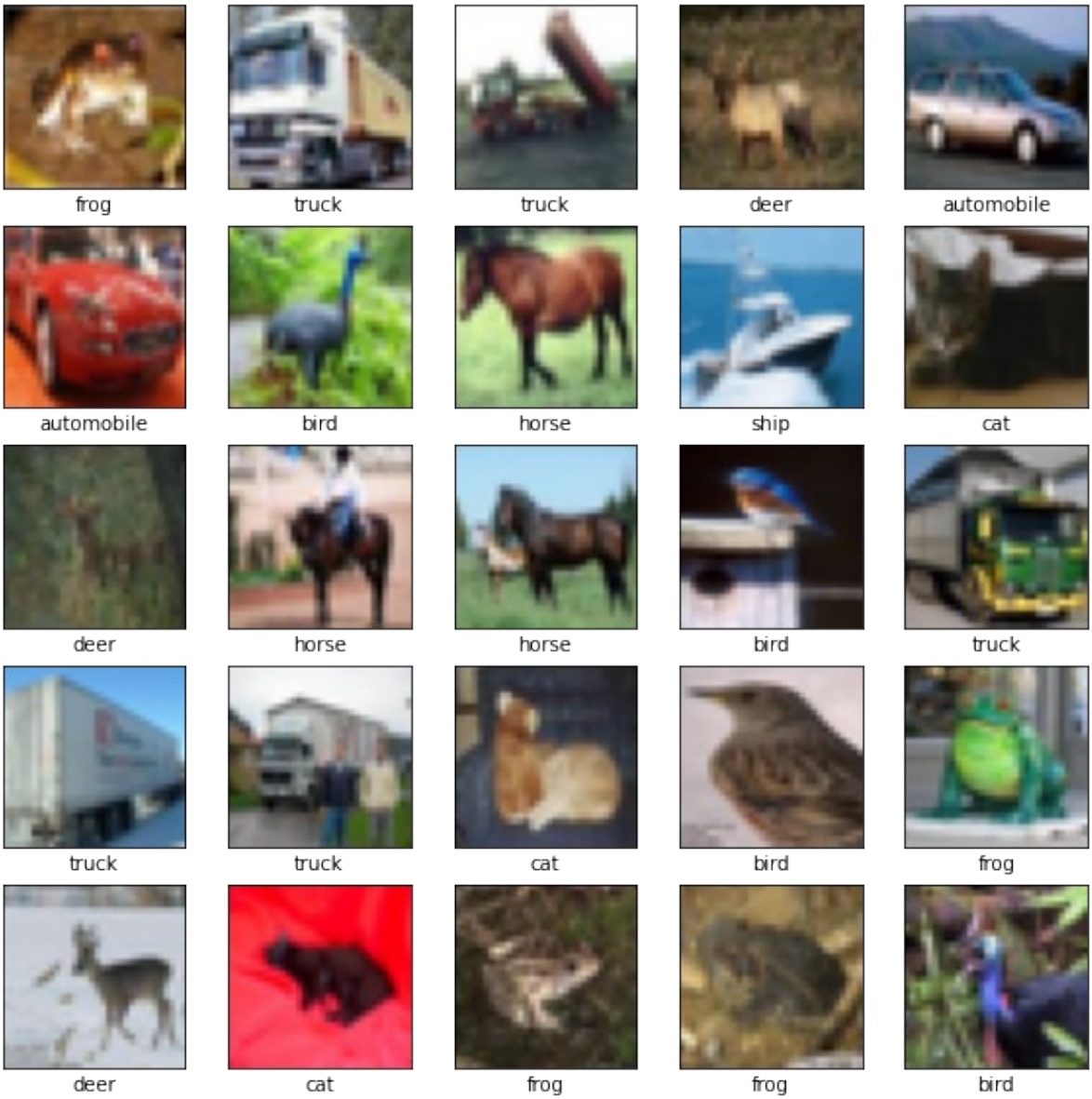


Illustration 37: Prikazane slike iz datasheet-a

```
In [7]:
# Converting the pixels data to float type
train_images = train_images.astype('float32')
test_images = test_images.astype('float32')
```

Slike se konvertuju u float32 da bi keras i numpy mogli raditi sa njima.

```
# Standardizing (255 is the total number of pixels an image can have)
train_images = train_images / 255
test_images = test_images / 255
```

Pretvara se svaka labela slike. Labele slike su nam potrebne da bi znali šta se nalazi na njima.

```
# One hot encoding the target class (labels)
num_classes = 10
train_labels = np_utils.to_categorical(train_labels, num_classes)
test_labels = np_utils.to_categorical(test_labels, num_classes)
```

Urađeno je one hot kodiranje. Iz niza class_name stvorena je jedna riječ gdje je 0 = airplane, 1 = automobile, 2 = bird, itd.

U kodu ispod je implementirana konvoluciona neuronska mreža.

In [8]:
```
# Creating a sequential model and adding layers to it

model = Sequential()

model.add(layers.Conv2D(32, (3,3), padding='same', activation='relu', input_shape=(32,32,3)))
model.add(layers.BatchNormalization())
model.add(layers.Conv2D(32, (3,3), padding='same', activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.MaxPooling2D(pool_size=(2,2)))
model.add(layers.Dropout(0.3))

model.add(layers.Conv2D(64, (3,3), padding='same', activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.Conv2D(64, (3,3), padding='same', activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.MaxPooling2D(pool_size=(2,2)))
model.add(layers.Dropout(0.5))

model.add(layers.Conv2D(128, (3,3), padding='same', activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.Conv2D(128, (3,3), padding='same', activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.MaxPooling2D(pool_size=(2,2)))
model.add(layers.Dropout(0.5))
```

Sa model = Sequential() pravimo neuronsku mrežu.

input_shape(32, 32, 3) nam određuje dimenzije slike koju postavljamo. Dimenzije su 32x32 piksela, sa 3 dimenzije dubine za RGB.

U prvom sloju biramo aktivacijske funkcije: activation = `relu`, padding = `same`.
Veličina prvog sloja je 32, a veličina kernela pomoću kojeg se prelazi preko slike je 3x3. Uradi se flip slike, sračuna se suma proizvoda svega i to se postavi u jedan piksel. Moramo uraditi padding slike, najpoznatiji je padding zero gdje sliku dopunimo nulama. Padding same sliku ne dopuni nulama već vrijednostima sa slike.

model.add(layers.BatchNormalisation())
Naša datasheet ima mnogo podataka, preko 10.000. Običaj je da se odvoji 70 - 80 % za treniranje mreže. Uzmemo jedan batch slika i proslijedimo h neuronskoj mreži za treniranje. Na ovaj način smanjujemo kompleksnost jer smo smanjili broj podataka.

Dobija se stohastički gradijent. Treba pronaći neku kvadratnu funkciju da bismo koeficijente mreže izrazili preko te funkcije i minimizirali je. Košijev gradijent descent zahtijeva prvi izvod, nije numerički zahtjevan, ali je u biti spor. S druge strane Njutnov metod optimizacije ima osobinu kvadratnog završavanja, jako brz, ali ima ogromnu kompleksnost. Potrebno je invertovati Hessijan. Ovaj metod sam za neuronsku mrežu sa mnogo parametara nije poželjan.

Zbog toga ćemo koristiti kombinaciju Košijeve i Njutnove metode. Počet ćemo sa Košijevom metodom jer ona u početku pravi veike korake, a završit ćemo sa Njutnovom metodom. Da bismo invertovali Hessijan najbolje bi ga bilo aproksimirati. To ćemo učiniti pomoću metode brzog učenja. Uzet ćemo neki proizvoljan korak u početku, između 0 i 1 kod Košijeve metode, pa ćemo taj korak polahko smanjivati jer je Njutnova metoda poznata po tome što ima male korake. Kako smanjujemo korake to sve više liči na Njutnovu metodu, i na taj način prelazimo iz Košijeve u Njutnovu metodu. Ovo se zove LVM metoda.

Metoda brzog učenja je ključni faktor kod neuronskih mreža. Potrebno je izabrati najbolju odgovarajuću brzinu učenja. Ne bi trebala biti previše mala zbog dugog protoka vremena, niti prevelika jer ćemo dobiti numeričke oscilacije.

Batch će da uzme samo jedan dio podataka da trenira na njima, te će sve to ponoviti u iteracijama. Ovo doprinosi smanjenju kompleksnosti i uvodi se stohastičnost u naš račun. Stohastičnost nije cilj, već posljedica batch-a. Stohastički gradijent jeste kada završimo na prvom minimumu, sa prvim batch-om. Sa drugim batch-om se naš vektor pomjeri, iskočimo iz prvog minimuma u neku tačku, pa nastavimo dalje ka stavrnom minimumu.

```
model.add(layers.Conv2D(32, (3,3), padding='same', activation='relu'))
model.add(layers.BatchNormalization())
```

U ovom dijelu je ubačen još jedan konvolucioni sloj, koji nije neophodan.

```
model.add(layers.MaxPooling2D(pool_size=(2,2)))
```

Dimenzije su 2x2. Od izlaznog sloja će uzeti sliku i prelaziti preko nje, slično kao što radi konvolucioni kernel. Pošto je dimenzija 2x2, od 4 piksela, uzet će najveći i staviti ga na lokaciju(0,0), i tako do kraja slike. Smanjuje se kompleksnost računa, veličina slike, feature(osobine) slike će biti lokalno invarijantne. Pooling pomjeri osobine slike.

Dimenzije slike su 32x32, a izlazni sloj treba biti 10 neurona(10 klasa u koje klasifikujemo). Pokušavamo svo vrijeme sliku da izravnamo, treba da je pretvorimo u vektor od 10 elemenata(MaxPooling).

```
model.add(layers.Dropout(0.3))
```

1/3(0.3) neurona neurološke mreže će izbaciti. Slučajno će odabrati 1/3 neurona/konekcija i izbaciti ih. Smanjuje kompleksnost i mreža ne postaje previše zavisna o jednom neuronu/koeficijentu.

Ostale konvolucione slojeve dodajemo kako želimo, metodom pokušaja i greške.

Zadnji sloj se sastoji od aktivacijskih funkcija, batch normalisation-a, dropout-a i posljednjeg sloja.

`model.add(layers.Flatten())` će da uzme sve slike što izlaze iz neurona i napravi od njih jedan vektor da bi mogao dobiti output(izlaz) koji je sastavljen od 10 neurona.

model.add(layers.Dense(num_classes, activation='softmax')) Na zadnji sloj smo dodali SOFTMAX funkciju da bismo dobili na kraju vjerovatnoću.

Pomoću model.summary() ćemo da ispišemo podatke o mreži.

```
model.add(layers.Flatten())
model.add(layers.Dense(128, activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.Dropout(0.5))
model.add(layers.Dense(num_classes, activation='softmax'))    # num_classes = 10

# Checking the model summary
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 32, 32, 32)        896
_________________________________________________________________
batch_normalization (BatchNo (None, 32, 32, 32)        128
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 32, 32, 32)        9248
_________________________________________________________________
batch_normalization_1 (Batch (None, 32, 32, 32)        128
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 16, 16, 32)        0
_________________________________________________________________
dropout (Dropout)            (None, 16, 16, 32)        0
_________________________________________________________________
```

Ovo će da nam iskompajlira mrežu.

In [9]:
```
model.compile(optimizer='adam', loss=keras.losses.categorical_crossentropy, metrics=['accuracy'])
```

Kod ispod radi treniranje mreža tako što ubacimo slike za trening, labele slika, veličinu batch-a, broj iteracija za treniranje, podaci za validaciju. `Cifar10` je vratio test images, ali ih u stvarnosti moramo sami napraviti.

Od ukupnih podataka najbolje je uzeti 70% za treniranje, 15% za testne podatke i 15% za validaciju.

In [10]:
```
history = model.fit(train_images, train_labels, batch_size=64, epochs=100,
                    validation_data=(test_images, test_labels))
```

U kodu ispod ćemo ispisati na ekran naše najbitnije krive, a to su krive tačnosti i kriva gubitka. Kriva gubitka se računa za epohe treniranja neuronske mreže. Nakon 60-ak iteracija ove dvije krive postaju slične. Ukoliko su one skoro pa isto, a opet različite, kao što je u slučaju ispod, dobijena je istrenirana mreža.

In [11]:
```
# Loss curve
plt.figure(figsize=[6,4])
plt.plot(history.history['loss'], 'black', linewidth=2.0)
plt.plot(history.history['val_loss'], 'green', linewidth=2.0)
plt.legend(['Training Loss', 'Validation Loss'], fontsize=14)
plt.xlabel('Epochs', fontsize=10)
plt.ylabel('Loss', fontsize=10)
plt.title('Loss Curves', fontsize=12)
```

Illustration 38: Kod za krivu gubitka

In [12]:
```
# Accuracy curve
plt.figure(figsize=[6,4])
plt.plot(history.history['accuracy'], 'black', linewidth=2.0)
plt.plot(history.history['val_accuracy'], 'blue', linewidth=2.0)
plt.legend(['Training Accuracy', 'Validation Accuracy'], fontsize=14)
plt.xlabel('Epochs', fontsize=10)
plt.ylabel('Accuracy', fontsize=10)
plt.title('Accuracy Curves', fontsize=12)
```

Illustration 39: Kod za krivu tačnosti

Ispod se nalaze dobijene krive:

Out[11]:
```
Text(0.5, 1.0, 'Loss Curves')
```

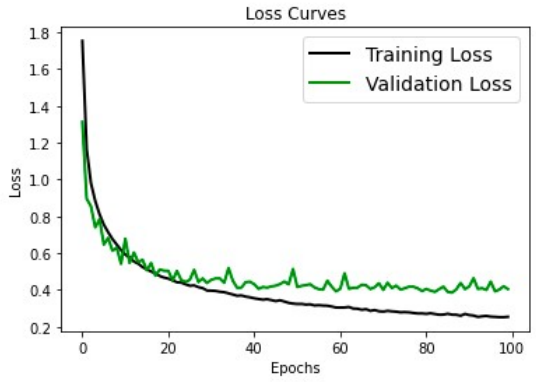


Illustration 40: Kriva gubitka

Out[12]:
```
Text(0.5, 1.0, 'Accuracy Curves')
```

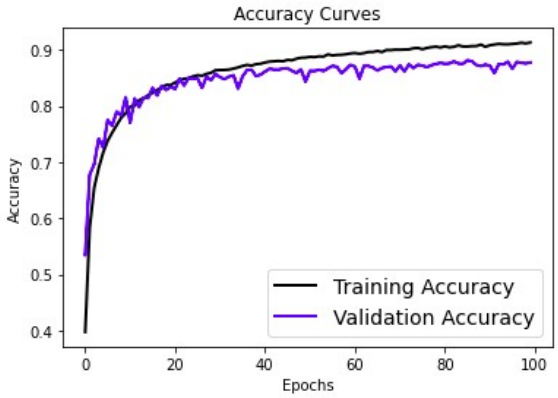


Illustration 41: Kriva tačnosti

Krive ne bi smjele biti dosta različite, u tom slučaju imamo neistreniranu (undertrained network) mrežu. Ovakva mreža ima malu vjerovatnoću da prepozna podatke(slike). Ovo možemo popraviti tako da dodamo više slojeva našoj mreži, više neurona i potrebno je manje raditi DROPOUT.

Ukoliko su krive skoro pa identične tada imamo pretreniranu(overtrained network) mrežu. Ovakva mreža nauči dobro na slikama za trening, ali se na testnim podacima(slikama) pokaže loše. Ovo možemo popraviti tako da više odradimo DROPOUT, potrebno je povećati raznovrsnost podataka, koristiti metodu generisanja podataka spomenutu na početku i sl.

In [13]:
```
# Making the Predictions
pred = model.predict(test_images)
print(pred)

# Converting the predictions into label index
pred_classes = np.argmax(pred, axis=1)
print(pred_classes)
```

U prvom dijelu ovog koda pravimo predikciju na testnim slikama. U drugom dijelu koda dobijenu predikciju, u kodu je vektor od 10xtestnih slika, za naše testne slike.

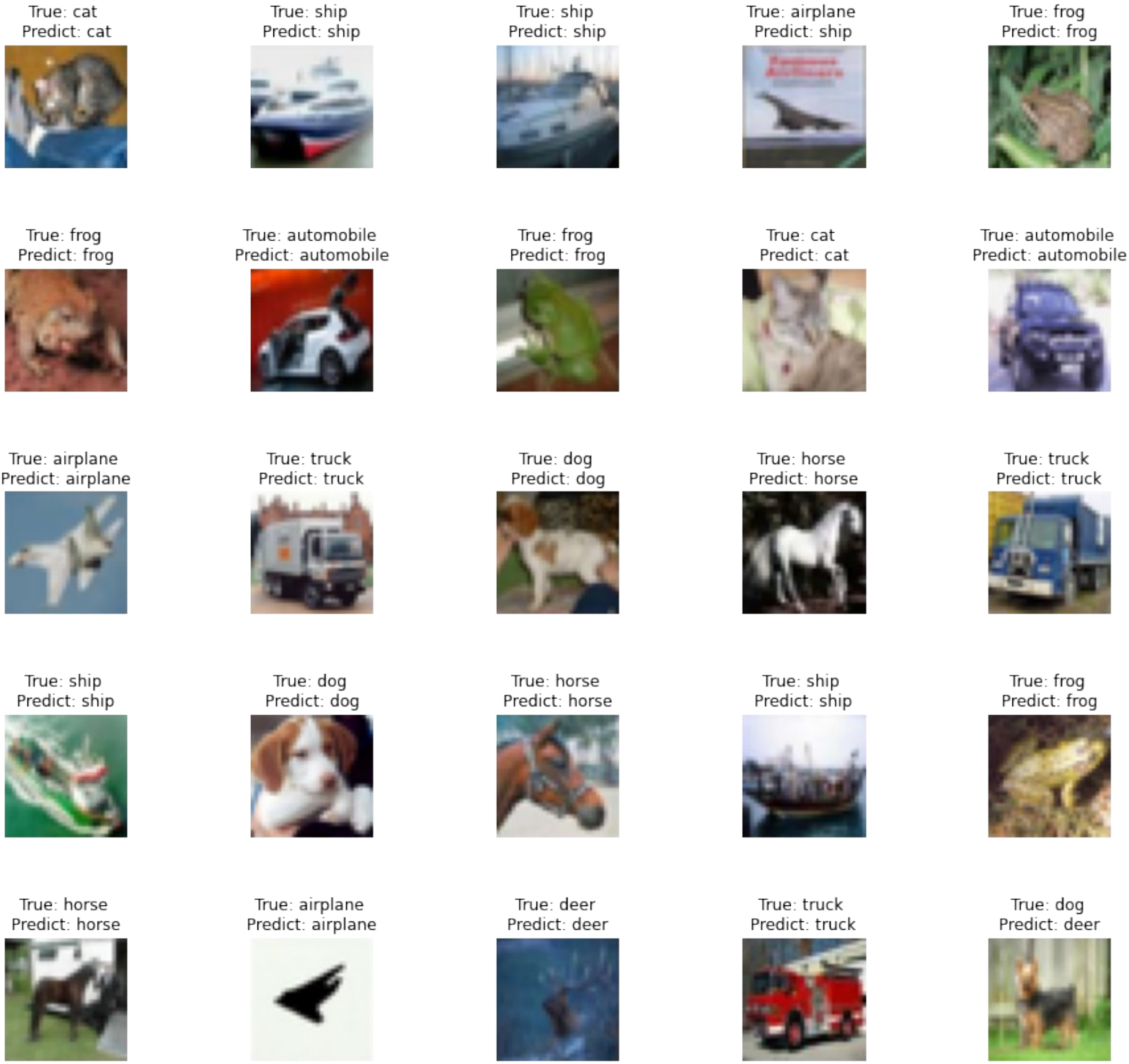
In [14]:
```
# Plotting the Actual vs. Predicted results

fig, axes = plt.subplots(5, 5, figsize=(15,15))
axes = axes.ravel()

for i in np.arange(0, 25):
    axes[i].imshow(test_images[i])
    axes[i].set_title("True: %s \nPredict: %s" % (class_names[np.argmax(test_labels[i])], class_names[pred_classes[i]]))
    axes[i].axis('off')
    plt.subplots_adjust(wspace=1)
```

U for petlji je iscrtano 25 slika iz testnih slika, na svakoj osi po jedna testna slika. Metodom subplot napravljena je raspodjela slika 5x5.

class_names[np.argmax(test_labels[i])] ispisuje se stvarna klasa, pa iscrtava

class_names[pred_classes[i]] ispisuje se predviđena klasa, pa iscrtava

Rezultat koji je dobijen ovim kodom nije u potpunosti tačan. Tačnost je 88%.



Napomena: U folderu Z6 se nalaze snimci koda sa stranice zadate u zadatku 6.