SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I

INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

TRANSFORMACIJA POSTOJEĆE REALNE SLIKE U ANIMIRANU SLIKU

Projektni zadatak iz kolegija Obrada slike i računalni vid

Matija Kaučić

Osijek, 2021.

SADRŽAJ

1. UVOD

Jedno od primarnih upotreba transformacije slika je putem filtara. Oni se koriste putem društvenih mreža, ali i kao razne aplikacije na našim pametnim mobitelima. No, čak i puno prije današnjeg doba, ljudi su na jedan ili drugi način „transformirali“ slike. Vjerojatno najpoznatiji primjer bila bi karikatura. U ovom projektom zadatku, neću ići baš do tolikog ekstrema, ali prikazati ću Vam postupak pretvaranja realne slike u animiranu sliku.

U projektnom zadatku je upotrijebljeno nekoliko postupaka obrade slike te je u nastavku rada opisan princip rada programskog rješenja.

1.1. Opis projektnog zadatka

Cilj projektnog zadatka jest implementirati algoritam koji će od ulazne realne slike napraviti njenu animiranu. Preporuča se upotreba kombinacije bilateralnog filtera i adaptivnog thresholdinga kako bi se stvorio efekt animacije.

2. PREGLED PODRUČJA TEME

Različiti filteri i transformacije slika postaju sve više i više dijelom ljudske svakodnevice. Moderni pametni mobiteli već u sebi imaju razne opcije poput postupka uljepšavanja fotografije na prednjoj kameri, gdje pomicanjem klizača možemo odabrati stupanj uljepšavanja. Prema zadanim postavkama klizač je stavljen na nulu, no pomicanjem njega dobivamo stupanj uljepšavanja, odnosno zamućenja detalja što daje efekt uljepšavanja. Nadalje, društvene platforme poput Instagrama i Snapchata su, između ostalog, korisnicima privlačne zbog raznih maštovitih filtera koje imaju u ponudi. Na tržištu već postoji velik broj rješenja gore spomenutog problema tako da se za probiti na tržište mora razmišljati o inovativnim metodama i, ne samo o kvaliteti rješenja, već i o samoj brzini obrade fotografije. Jedan od najpoznatijih primjera transformacije u animiranu sliku je bio popularan tekuće godine i pomogao je dodatno popularizirati platformu TikTok, a zvao se Disney Cartoon Face Effect.

3. KORIŠTENI ALATI

Za razvoj ovog projekta sam u potpunosti koristio programski jezik Python, a radio sam u okolišu razvojnog okruženja Spyder koji se nalazi unutar platforme Anaconda. Također, unutar samog Pythona, koristio sam biblioteke OpenCV i Matplotlib.

3.1. Python

Python je programski jezik opće namjene, interpretiran i visoke razine kojeg je stvorio Guido van Rossum 1990. godine (prva javna inačica objavljena je u veljači 1991. godine), ime dobiva po televizijskoj seriji Monty Python's Flying Circus. Po automatskoj memorijskoj alokaciji, Python je sličan programskim jezicima kao što su Perl, Ruby, Smalltalk itd. Python dopušta programerima korištenje nekoliko stilova programiranja. Objektno orijentirano, strukturno i aspektno orijentirano programiranje stilovi su dopušteni korištenjem Pythona te ova fleksibilnost čini Python programski jezik sve popularnijim. Python se najviše koristi na Linuxu, no postoje i inačice za druge operacijske sustave.

3.2. Anaconda

Anaconda je distribucija programskih jezika Python i R za znanstveno računarstvo (informacijske znanosti, aplikacije na bazi strojnog učenja, obrada velike količine podataka, prediktivna analitika i sl.) kojoj je cilj pojednostaviti upravljanje i implementaciju paketa. Distribucija uključuje pakete pogodne za Windows, Linux i macOS. Osnovana je i još uvijek održavana od strane Anaconda Inc., koja su stvorili Peter Wang i Travis Oliphant 2012. godine. Također, postoji i manja, bootstrap verzija koja se zove Miniconda.

3.3. Spyder

Spyder je integrirano višeplatformsko razvojno okruženje otvorenog koda za znanstveno programiranje bazirano na jeziku Python. Spyder sadrži niz istaknutih integriranih paketa u znanstvenom paketu Python, uključujući NumPy, SciPy, Matplotlib,

pande, IPython, SymPy i Cython, kao i druge softvere otvorenog koda. Pušteno je na tržište pod licencom MIT-a. Prvotno je kreirano od strane Pierre Raybauta u 2009. godini, a od tad je svakodnevno nadzirano i poboljšavano uz pomoć Python programera i Python zajednice. Spyder je također proširiv i putem vanjskih dodataka, uključuje podršku interaktivnih alata za proučavanje podataka te ima ugrađene instrumente za osiguravanje kvalitete. Njegova višeplatformska podrška je još jedan od velikih prednosti, putem Anaconde na Windowsima, MacPorts-a na macOS-u te raznim inačicama na Linuxu poput Arch Linux, Debian, Fedora, Ubuntu i sl.

3.4. OpenCV

OpenCV (*Open Source Computer Vision Library*) je biblioteka kojoj je glavna uloga upravljanje računalnim vidom u stvarnom vremenu. Originalno je napisana u programskom jeziku C++ i primarni jezik korištenja je C++, ali koristi se i u Pythonu, Javi, JavaScriptu i MATLABU. Razvijena je od strane tvrtke Intel 2000.godine. OpenCV je višeplatformska biblioteka i besplatna je za korištenje pod Apache 2 licencom. Neke od aplikacija ove biblioteke su prepoznavanje pokreta, prepoznavanje objekata, mobilna robotika, AR (proširena stvarnost), prepoznavanje lica itd.

3.5. Matplotlib

Matplotlib je biblioteka namijenjena za crtanje u programskom jeziku Python te sadržava numerički dodatak NumPy. Ona osigurava objektno orijentiran API za ugrađivanje crteža u aplikacije korištenjem grafičkih setova alata opće namijene poput Tkinter, wxPython, Qt ili GTK. Također, postoji i pylab sučelje koje je napravljeno kako bi nalikovalo na MATLAB. Ova biblioteka je originalno napisana od strane John D. Huntera 2003. godine, a od tad je zadobila aktivnu zajednicu koja se brine o njenom održavanju.

4. PROGRAMSKO RJEŠENJE

Programsko rješenje ostvareno je u programskom jeziku Python, koristeći biblioteke OpenCV i Matplotlib (Slika 4.1.). Ideja za programsko rješenje sastojala se od dva dijela: istaknuti rubove i izgladiti/zamutiti boje na slici kako bi se postigao efekt animiranosti.

```
import cv2 #for image processing
import matplotlib.pyplot as plt
```

Slika 4.1. Uključivanje potrebnih biblioteka

Prvi korak razvoja algoritma bilo je učitavanje slike na kojoj želimo izvršiti transformaciju. To sam postigao korištenjem naredbe imread iz biblioteke OpenCV (Slika 4.2.) koja za argument prima putanju željene slike.

```
#loading an image
ImagePath = "Images/lenna.bmp"
originalImage = cv2.imread(ImagePath)
ReSized1 = cv2.resize(originalImage, (960, 540))
```

Slika 4.2. Učitavanje slike

ReSized1 označava sliku kojoj je promijenjena veličina kako bi se na kraju prikazali svi koraci algoritma na jednom grafu.

Zatim, iduće je potrebno transformirati sliku iz BGR (blue – green – red) načina rada u grayscale slike, odnosno sliku u sivim tonovima (Slika 4.3.). To radimo kako bi smanjili kompleksnost same slike jer kod BGR svakom pikselu se pridodaju tri broja (po jedan za svaki od kanala boje), a kod načina rada u sivim tonovima svaki piksel sadrži samo jednu brojku. Slika je prije svega ništa drugo nego skup brojeva. Također, neki od idućih koraka rade puno bolje u grayscale načinu rada, npr. detekcija rubova. Samu pretvorbu smo ostvarili korištenjem naredbe cvtColor kojoj smo za argumente predali izvornu sliku i COLOR_BGR2GRAY što označava pretvorbu iz BGR u grayscale.

```
#converting the image to grayscale
grayScaleImage = cv2.cvtColor(originalImage, cv2.COLOR_BGR2GRAY)
ReSized2 = cv2.resize(grayScaleImage, (960, 540))
```

Slika 4.3. Pretvaranje slike u grayscale

Treći korak algoritma je izglađivanje slike. Kako bi to postigli dodajemo efekt zamućenja, odnosno blur. To sam ostvario korištenjem medianBlur() funkcije (Slika 4.4.). Kod nje, središnjem pikselu se dodjeljuje srednja vrijednost svih piksela koji padaju pod njen kernel, odnosno jezgru i tako se kreira efekt zamućenja. Funkcija medianBlur() za argumente prima sliku nad kojom želimo napraviti promjenu (za koju sam odabrao sliku iz prethodnog koraka) te gore opisani iznos srednje vrijednosti.

```
#applying median blur to smoothen the image
smoothGrayScale = cv2.medianBlur(grayScaleImage, 5)
ReSized3 = cv2.resize(smoothGrayScale, (960, 540))
```

Slika 4.4. Izglađivanje slike

Dalje, četvrti korak je detekcija rubova koju sam ostvario korištenjem tehnike adaptivnog threshlodinga. Vrijednost threshloda je srednja vrijednost vrijednosti područja susjednih piksela umanjena za konstantu C, a C je konstanta oduzeta od srednje ili ponderirane sume susjednih piksela. Također, za tip thresholda sam odabrao binarni. U programskom kodu, to sam ostvario korištenjem funkcije adaptiveThreshold() (Slika 4.5.) koja za argumente prima željenu sliku, maksimalnu vrijednost, adaptivnu metodu, veličinu bloka i konstantu C.

```
#retrieving the edges for cartoon effect by using adaptive thresholding technique
getEdge = cv2.adaptiveThreshold(smoothGrayScale, 255, cv2.ADAPTIVE_THRESH_MEAN_C, cv2.THRESH_BINARY, blockSize=9, C=9)
ReSized4 = cv2.resize(getEdge, (960, 540))
```

Slika 4.5. Detekcija rubova

Peti korak je pripremanje slike maske primjenom bilateralnog filtera kako bi se uklonio šum i kako bi rubovi ostali dovoljno oštri. Pripremamo posvijetljenu sliku koju ćemo u sljedećem koraku *maskati*, odnosno maskirati, sa slikom detektiranih rubova i tako ćemo postići konačan rezultat. Ovo postižemo funkcijom bilateralFilter() koja za argumente prima željenu sliku koja je sada ona originalno odabrana slika, konstantu d, sigmaColor i sigmaSpace varijable koje su u pravilu jednake i ako im je vrijednost veća od 150 slika dobiva animiran doživljaj.

```
#applying bilateral filter to remove noise and keep edge sharp as required
colorImage = cv2.bilateralFilter(originalImage, d=9, sigmaColor=300, sigmaSpace=300)
ReSized5 = cv2.resize(colorImage, (960, 540))
```

Slika 4.6. Pripremanje slike maske

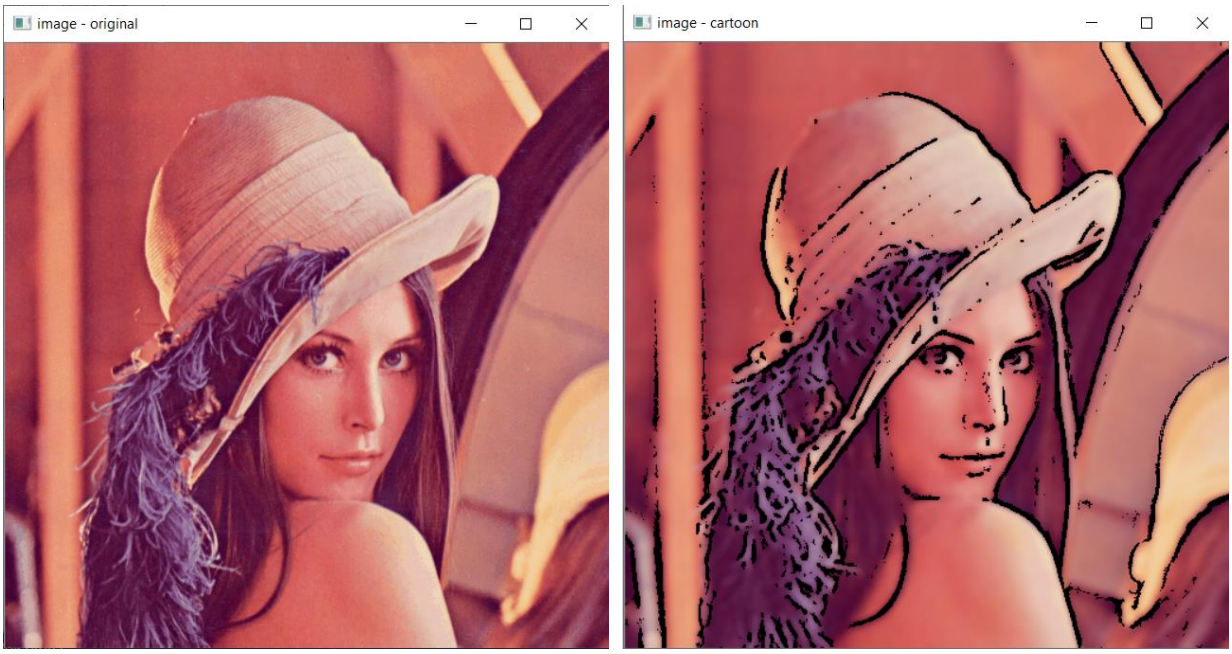
Zadnji korak algoritma je kombiniranje zadnja dva koraka tako što ih „maskiramo“. Koristimo bitwise operaciju (Slika 4.7.) na dvije slike s bilateralnim filterom kako bi to uspjeli, a za argument edge dodajemo sliku detekcije rubova. Maska je samo skup brojeva pa je zato ova operacija moguća. Ovo je konačni rezultat koraka obrade slike i vidimo kako se nazire animiranost slike.

```
#masking edged image with our "BEAUTIFY" image
cartoonImage = cv2.bitwise_and(colorImage, colorImage, mask=getEdge)
ReSized6 = cv2.resize(cartoonImage, (960, 540))
```

Slika 4.7. Završno kombiniranje slike pomoću bitwise operacije

5. OSTVARENI REZULTATI

Algoritam sam testirao na slici lenna.bmp (Slika 5.1.) koja je jako popularna u području obrade slike. Lenna ili Lena je standardna slika za isprobavanje transformacija i obrada korištena još od davne 1973. godine. To je fotografija švedskog modela Lene Forsén.



Slika 5.1. Lenna, originalna slika i slika nakon transformacije

Kako bih pokazao sve korake postupne transformacije slike iz realne u animiranu, koristeći Matplotlib biblioteku sam odlučio sam sve korake prikazati na jednom grafu. Prije samo crtanja svih slika, potrebno je bilo pretvoriti sve sliku u RGB način rada (Slika 5.2.) jer im je predefinirano BGR tip boja te je slikama onda prevladavao plavi ton boja.

```
Re1 = cv2.cvtColor(ReSized1, cv2.COLOR_BGR2RGB)
Re2 = cv2.cvtColor(ReSized2, cv2.COLOR_BGR2RGB)
Re3 = cv2.cvtColor(ReSized3, cv2.COLOR_BGR2RGB)
Re4 = cv2.cvtColor(ReSized4, cv2.COLOR_BGR2RGB)
Re5 = cv2.cvtColor(ReSized5, cv2.COLOR_BGR2RGB)
Re6 = cv2.cvtColor(ReSized6, cv2.COLOR_BGR2RGB)
```

Slika 5.2. Pretvorba slika iz BGR u RGB

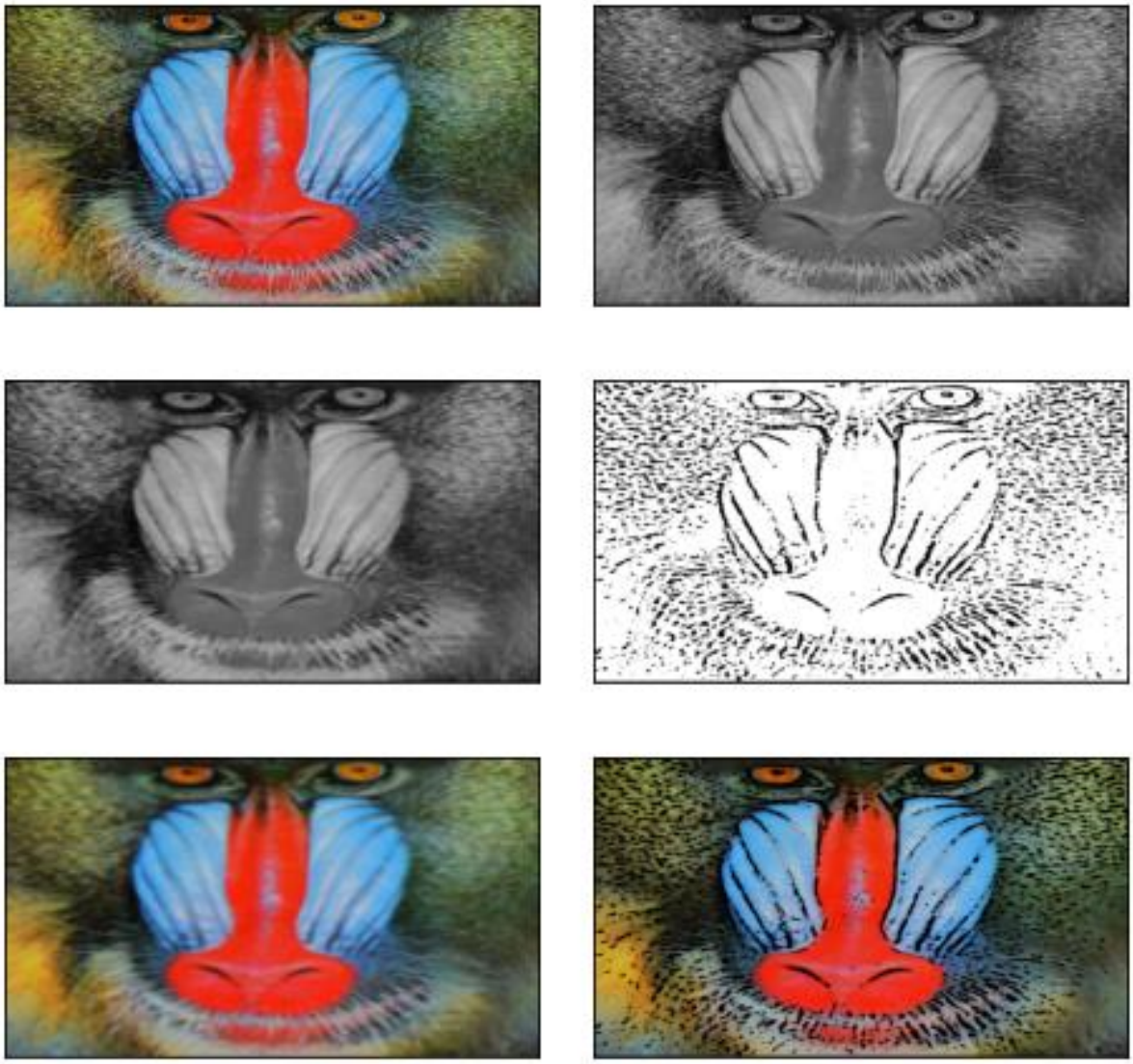
Crtanje svih slika odjednom sam postigao algoritmom prikazanim na slici 5.3.

```
images=[Re1, Re2, Re3, Re4, Re5, Re6]
fig, axes = plt.subplots(3,2, figsize=(8,8), subplot_kw={'xticks':[], 'yticks':[]}, gridspec_kw=dict(hspace=0.1, wspace=0.1))
for i, ax in enumerate(axes.flat):
    ax.imshow(images[i], cmap='gray')
plt.show()
```

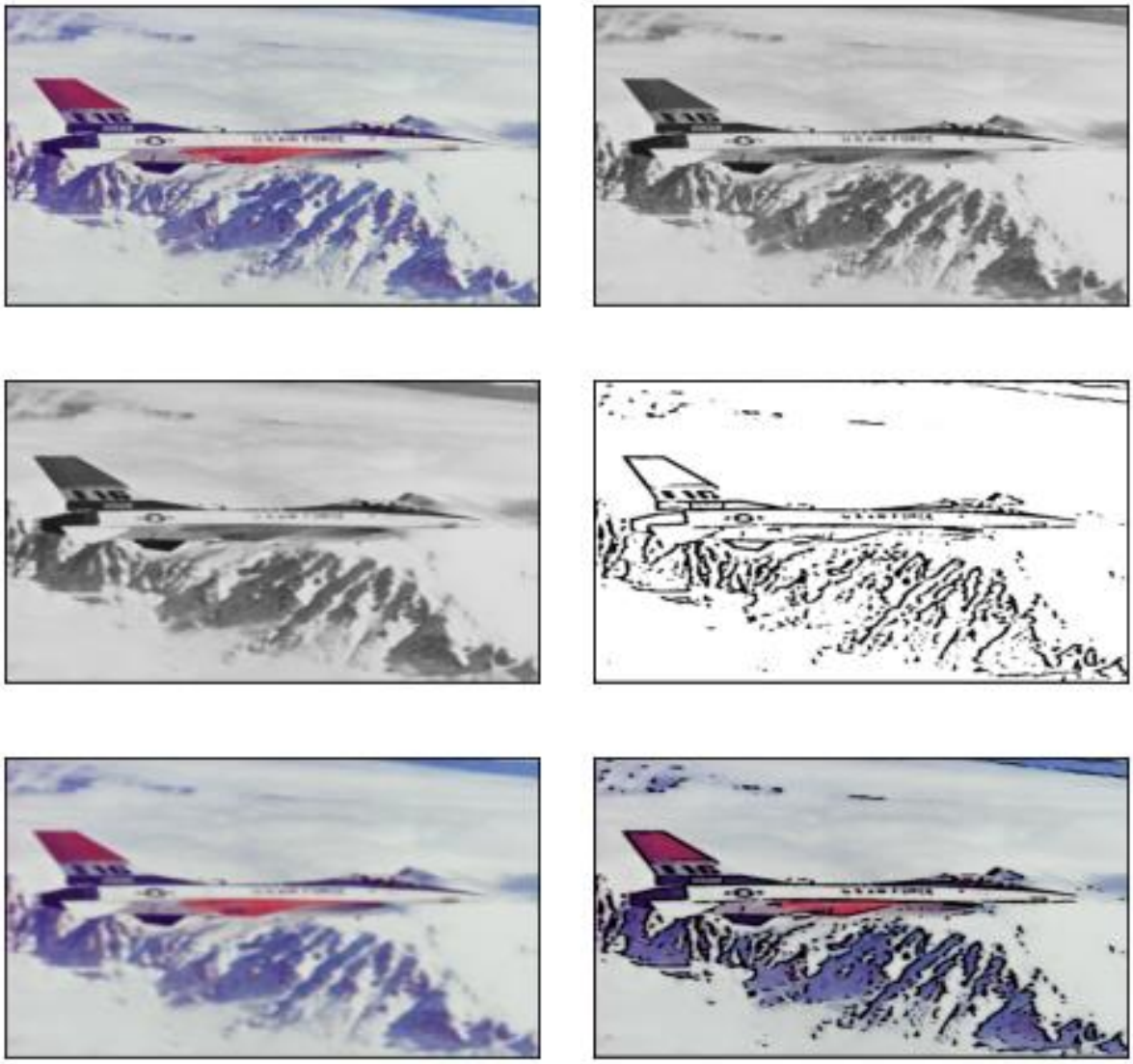
Slika 5.3. Crtanje svih koraka transformacije slike

U nastavku prilažem neke od rezultata uz sve korake prikazana u dosadašnjem dijelu rada. Prvih nekoliko slika su poznate testne slike poput lenna, baboon, airplane, barbara a slike poslije su nasumično odabrane testne slike.

Primjer 5.1. Slika lenna

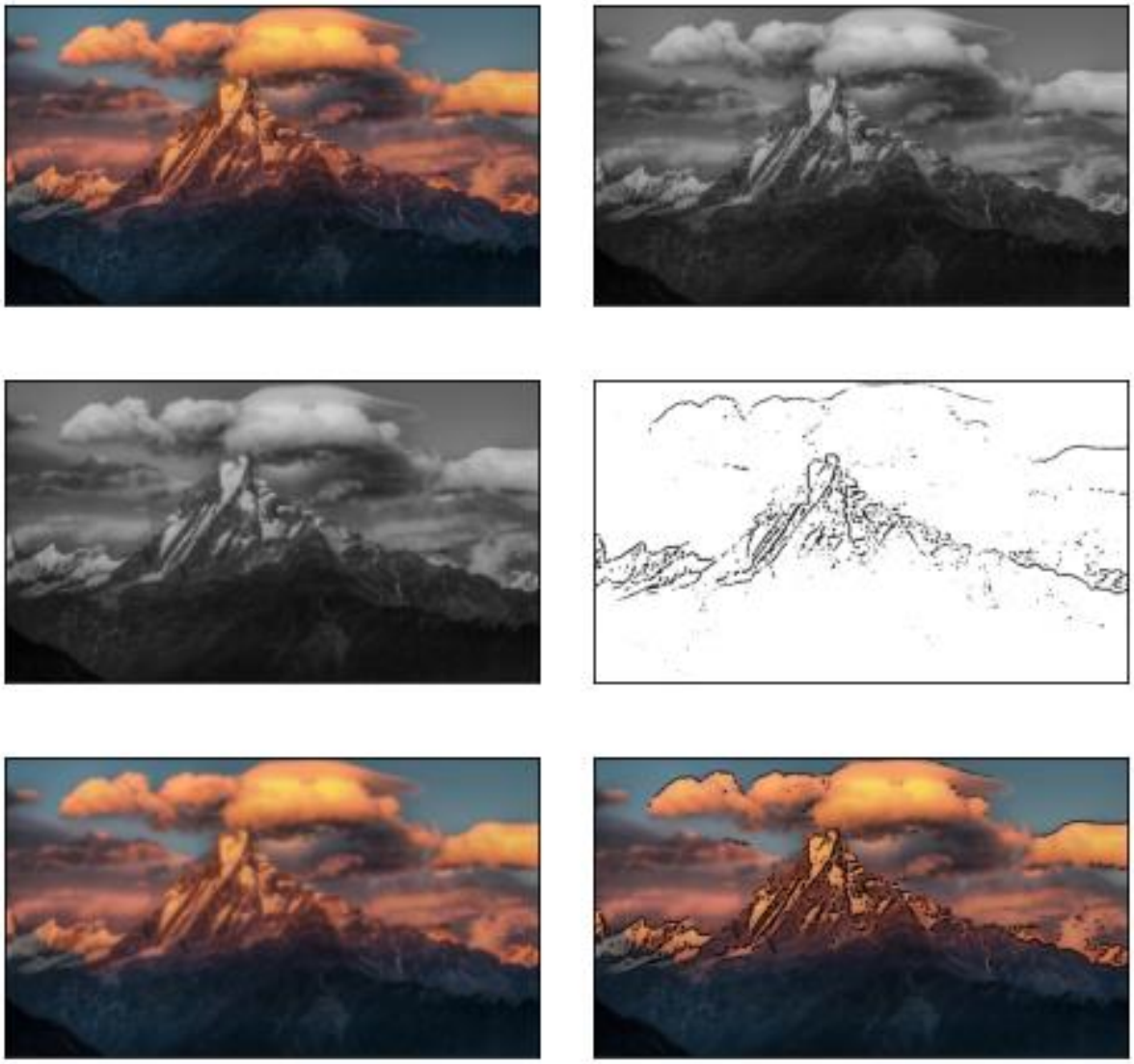


Primjer 5.2. Slika baboon

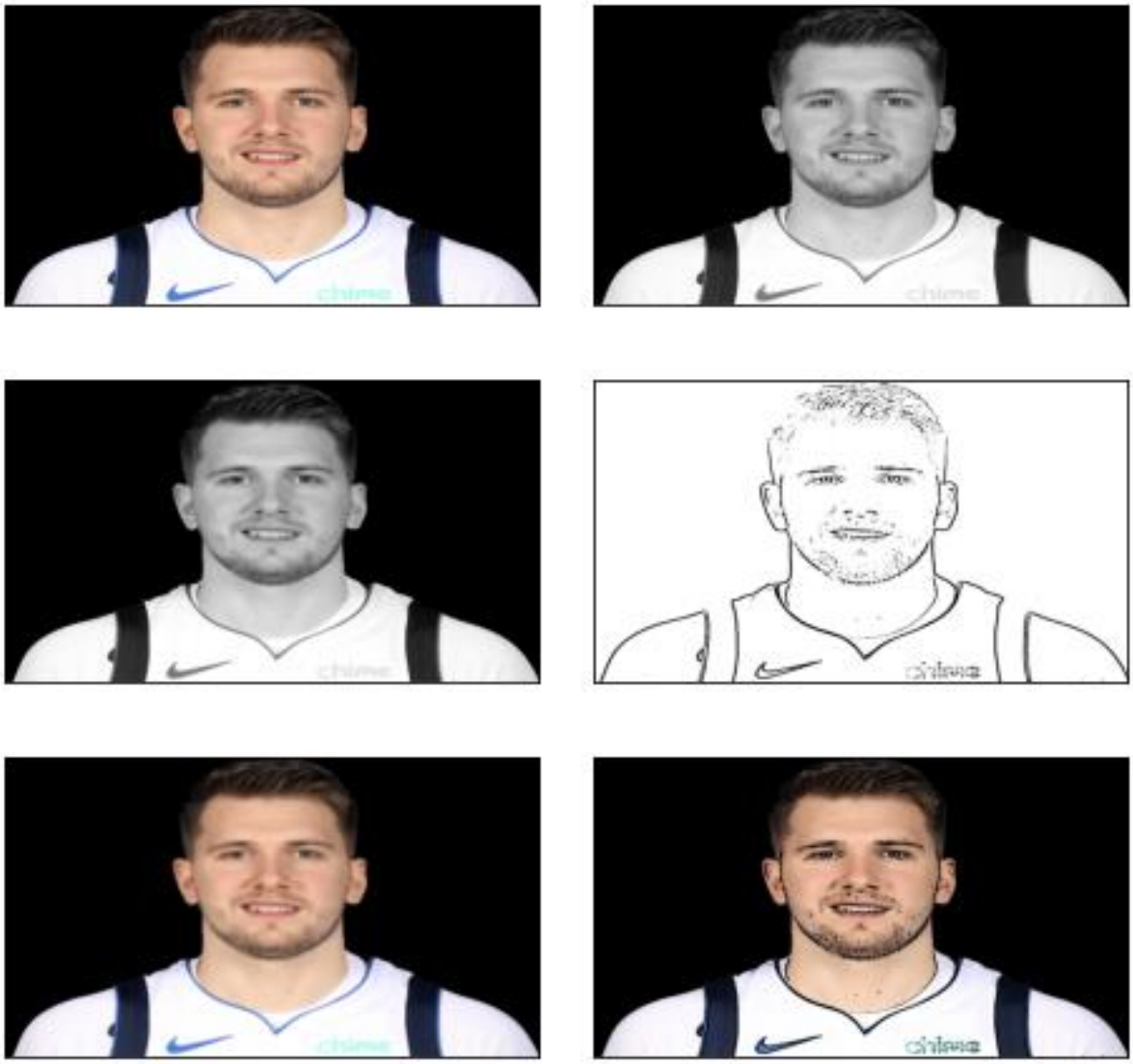


Primjer 5.3. Slika airplane

Slika 5.4. Slika barbara



Primjer 5.5. Slika planina



Primjer 5.6. Slika košarkaša

6. ZAKLJUČAK

U sklopu projektnog zadatka realiziran je algoritam za transformaciju realne slike u animiranu. Za ostvarivanje željenih rezultata korišten je programski jezik Python i biblioteke OpenCV i Matplotlib. Rješenje se testiralo na proizvoljnom skupu slika.

Ostvareni rezultati su prihvatljivi, ali ne i odlični. Rezultati su puno kvalitetniji na slikama sa svijetlom pozadinom dok su one slike s tamnom pozadinom davali puno lošije rezultate. Smatram da je razlog tomu što je veliki dio moje ideje izvedbe projektnog zadatka bio temeljen na isticanju rubova koji su teško vidljivi kod slika s tamnom pozadinom.

Daljnje poboljšanje projekta bi uključivalo razvijanje boljeg algoritma same transformacije te korištenje drugih specifičnijih filtera i biblioteka. Postizanje dobrih, ali ne i odličnih rezultata na ovom, ne pretjerano teškom, problemu ukazuje na to da možda postoje drugi, bolji načini obrade slike od onog korištenog u ovom radu.

LITERATURA

https://www.python.org/ - Python dokumentacija

https://opencv.org/ - OpenCV biblioteka

https://matplotlib.org/ - Maplotlib biblioteka

https://www.anaconda.com/ - Anaconda

https://www.spyder-ide.org/ - Spyder IDE