Семинарска работа по

Дигитално процесирање на слика

Тема:

Cartoonifying an image

Изработил:

Борјан Ѓоргиев 213184

Скопје 2023

Содржина

1. Вовед

Како што можеби знаете, скицирањето или креирањето цртан филм не треба секогаш да се прави рачно. Во денешно време, многу апликации можат да ги претворат вашите фотографии да изгледаат како да се цртани. Темата на проектот кој што го избравме гласи “Cartoonifying an image”. Уште од најмала возраст се среќаваме со најразлични цртани филмови кои што се допадливи за човечкото око. Токму ова ни даде поттик да ја изработиме нашата семинарска работа.

За изработка на овој проект ги користевме добро познатите библиотеки Numpy, Open CV и Matplotlib од програмскиот јазик Python со кои што се запознавме на курсот по Дигитално процесирање на слика.

Во продолжение ќе ви покажеме како да ја конвертирате која било фотографија во слика како од цртан филм користејќи Python. Ние користевме оптимизирани алгоритми за машинско учење кои произведуваат јасни и остри слики.

Библиотеките OpenCV и NumPy ни се корисни за оваа операција. Резултатите се доста импресивни. Постојат некои суптилни разлики помеѓу реалните слики и сликите од цртан филм. Една од големите разлики е тоа што цртаната слика има дебели и јасно дефинирани рабови на секој објект. Друга забележлива разлика е тоа што цртаната слика има помалку различни варијанти на бои од вистинската слика. Наша цел е да ги дадеме овие карактеристики на вистинска слика.

2. Чекори за постигнување на ефектот Cartoonifying an Image:

2.1.Импортирање на соодветните библиотеки и вчитување на сликата

```
import cv2
import numpy as np

import matplotlib.pyplot as plt
```

Слика бр1. Импортирање на библиотеки

Импортираме OpenCV како cv2 за да ги користиме веќе имплементираните алгоритми за процесирање на слики, NumPy како np за брзи операции со слики и Matplotlib.pyplot како plt за приказ на резултати и меѓурезултати.

```
# Reading image
img = cv2.imread('image_name.jpg')
height, width = img.shape[:2] # Subtract height and width from image
max_dim = max(height, width) # Maximum from height and width
scaling_factor = 1
if max_dim > 1000:
 scaling_factor = 1000 / max_dim
 img = downsample_image(img, scaling_factor)
```

Слика бр 2. Читање на сликата за манипулација

img = cv2.imread(‘image_name.jpg’) - вчитување на посакуваната слика којашто сакаме да ја претвориме во слика од цртан филм

Бидејќи сликите може да бидат со огромни димензии и да е потребно поголемо време и процесирачка моќ, на истите им ја намалуваме големината според соодветно пресметан скалирачки фактор. Сликите со ширина или висина поголема од 1000 пиксели се намалуваат со скалирачки фактор 1000 / максимална димензија, со повик на функцијата downsample_image(), а потоа истите се зголемуваат со истиот скалирачки фактор со повик на функцијата upsample_image().

Функцијата како аргументи ги прима соодветната слика и скалирачкиот фактор. Целта на оваа функција е да пресмета нова намалена ширина и висина на сликата. Тоа се постигнува со следната линија код:

resized_image=cv2.resize(img,(new_width,new_height),interpolation=cv2.INTER_LINEAR)

Функцијата resize() од cv2 библиотеката како аргументи ги прима сликата чијашто големина ја менување, новите димензии на истата, а како последен аргумент за интерполација поставуваме вредност cv2.INTER_LINEAR - константа која го претставува методот на билинеарна интерполација. Билинеарната интерполација ги проценува вредностите на пикселите на новата слика врз основа на вредностите на нејзините соседни пиксели од оригиналната слика. Создава непречена транзиција помеѓу соседните пиксели и обезбедува добра рамнотежа помеѓу брзината и квалитетот за промена на големината.

Функцијата upsample_image() како аргументи ги прима соодветната слика и скалирачкиот фактор. Целта на оваа функција е да постигне спротивен ефект од функцијата downsample_image(), т.е. сликата врз која била применета таа функција да ја врати во првобитна сосотојба.

Со намалување на димензиите на сликите се губат детали од истите коишто со зголемување на сликата можно е да не се обноват, односно се добива на брзина, но се губи на квалитет.

2.2.Наоѓање на рабовите на оригиналната слика

```
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
# Convert the input image to gray scale
gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
gray = cv2.medianBlur(gray, 3)
# Perform adaptive threshold
edges = cv2.adaptiveThreshold(gray, 255,
cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 9, 8)
```

Слика бр 2. Читање на сликата за манипулација

Во вториот чекор ги одредуваме рабовите на сликата. За приказ на резултантните слики во matplotlib најпрвин потребно е да ја конвертираме сликата во RGB формат. Потоа вршиме конверзија во grayscale и правиме измазнување на сликата заради голема можност од шум во истата. Ја користиме функцијата cv2.medianBlur() со големина на кернел 3.

Со помош на Adaptive threshold техниката ја делиме сликата на помали региони базирани на интензивноста на регионите на сликата.

На овој начин може да се доловат подобро карактеристиките како рабовите и контурите на самата слика.

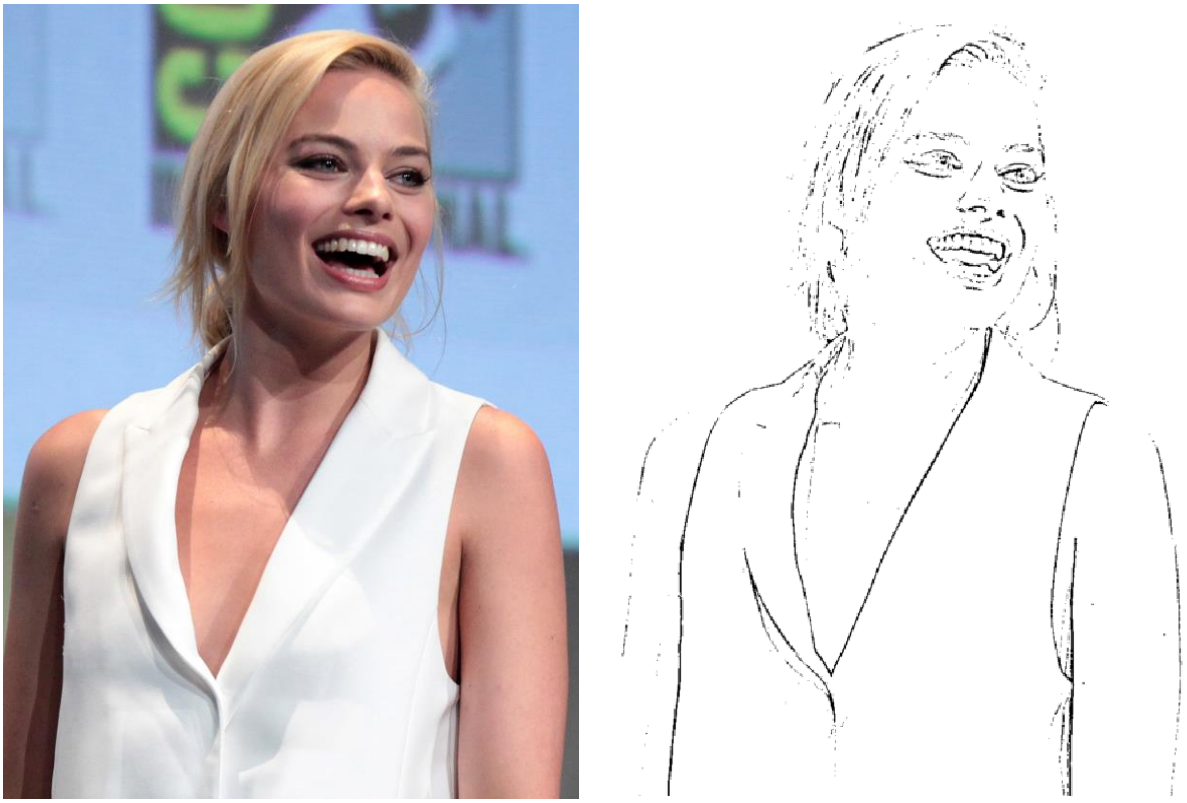
За оваа цел декларираме нова променлива наречена edges. Оваа променлива како вредност ќе го прими резултатот од функцијата adaptiveThreshold() од cv2 библиотеката којшто е бинарна слика. Како прв аргумент во оваа функција е сликата која што претходно беше претворена во grayscale. Вториот аргумент се однесува на максималната вредност на пикселите од резултатот. Во овој случај вредноста на истите изнесува 255(бела). Третиот аргумент е начинот на којшто ќе се пресметува праговата вредност. Вредноста е cv2.ADAPTIVE_THRESH_GAUSSIAN_C - константа според која праговата вредност е тежинска сума на соседните вредности во Гаусов прозорец.

Овој метод го зема предвид локалното соседство на секој пиксел и доделува тежини на соседните пиксели врз основа на Гаусова дистрибуција, каде што пикселите поблиску до центарот на соседството имаат поголема тежина, а пикселите подалеку имаат помала тежина. Четвртиот аргумент е типот на праг со вредност cv2.THRESH_BINARY - е

параметар кој што ни кажува колкава прагова вредност ќе биде употребена.Ова всушност означува дека пикселите од сликата кои што имаа вредност над прагот ќе бидат поставени на максимална вредност во нашиот случај тоа е 255(бела), а додека пак тие пиксели кои што имаат вредност под прагот ќе бидат поставени на 0(црна). Петтиот аргумент е големината на блокот од соседството којшто ќе се земе во предвид при пресметката на праговата вредност за секој пиксел.

Шестиот аргумент е константна вредност која што се одзема од тежинската сума на соседството.Ни помага во прилагодување на ефектот на прагот.

Резултатот после овој чекор е следен:



Слика бр 4. Оригинална слика (лево) и слика со рабови (десна)

2.3.Изведување на техника на квантизација на бои (Color quantization)

За постигнува на техника на квантизација на бои или намалување на бројот на бои во слика дефинираме функција cartoonize().

```
def cartoonize(img, k):
  # Defining input data for clustering
  data = np.float32(img).reshape((-1, 3))
  # Defining criteria
  criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 20,
1.0)
  # Applying cv2.kmeans function
  _, label, center = cv2.kmeans(data, k, None, criteria, 10, cv2.KMEANS_RANDOM_CENTERS)
  center = np.uint8(center)
  print(center)
  # Reshape the output data to the size of input image
  result = center[label.flatten()]
  result = result.reshape(img.shape)
  return result
```

Слика бр 5. Функција за изведување на квантизација на бои

Функцијата cartoonize прима два аргументи. Првиот е сликата врз којашто сакаме да ги извршиме соодветните манипулации, а вториот параметар k - претставува посакуваниот број на кластери, односно посакуваниот број на бои во резултантната слика.

`data = np.float32(img).reshape((-1, 3))`

Со оваа линија код влезната слика ја претвораме во 32-битна float репрезентација (np.float32(img)), за подобра понатамошна обработка.

Потоа добиениот резултат го преобликуваме од дводимензионална слика, во дводимензионална низа со 3 колони (.reshape((-1, 3))). Функцијата reshape го менува обликот на низата без да ги промени податоците. Првиот аргумент со вредност -1 означува дека бројот на редови ќе биде автоматски одреден според бројот на пиксели во сликата, а вториот аргумент со вредност 3 го одредува бројот на колони. Вредностите во колоните ќе бидат соодветните RGB вредности на пикселите.

criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 20, 1.0)

Новата променлива наречена criteria содржи два гранични случаи а тоа се: cv2.TERM_CRITERIA_EPS и cv2.TERM_CRITERIA_MAX_ITER.

Преку нив дефинираме терминирачки критериум за тоа кога да прекине да се извршува k-means алгоритмот. Во нашиот случај тоа ќе се случи после 20 итерации или пак додека не стигнеме до посакуваната точност од 1.0

_, label, center = cv2.kmeans(data, k, None, criteria, 10, cv2.KMEANS_RANDOM_CENTERS)

Со оваа линија код го применува k-means алгоритмот за кластерирање на влезниот параметар data. Вториот параметар со вредност k го претставува бројот на кластери којшто kmeans алгоритмот се обидува да го пронајде во податоците, т.е. Овој алгоритам ги групира сличните пиксели во k кластери според нивните RGB вредности. Во нашите примери овој параметар има вредност 12.

Третиот параметар се користи за да се специфицираат иницијалните центри на кластерите. Во нашиот случај, None означува дека иницијалните центри на кластерите треба да бидат избрани случајно.

Аргументот criteria е терминирачкиот критериум којшто го објаснивме погоре.Тој користи гранични случаи како cv2.TERM_CRITERIA_EPS и други параметри.

Бројот “10” во оваа функција означува бројот на пати колку што ќе биде употребен k-means алгоритам. Најдобриот резултат, односно тој со најдобар кластерирачки квалитет, ќе биде избран.

cv2.KMEANS_RANDOM_CENTERS:

Ова знаменце ни кажува дека центрите на кластерите треба да бидат избрани случајно. Со користењето на ова знаменце може да го намалиме ризикот од заглавување во локални минимуми за време на процесот на оптимизација на k-means алгоритмот. Добра пракса е да се искористи ова знаменце во повеќе итерации на алгоритмот со цел да се зголеми квалитетот и стабилноста на кластерирачките резултати.

Функцијата назад враќа 3 излеза од нив повеќе не интересираат излезите label и center.

label: Ова е еднодимензионална низа што ги содржи ознаките на кластерите доделени на секој пиксел од data параметарот. Секој елемент во низата label означува на кој кластер му припаѓа соодветниот пиксел.

center: Ова е дводимензионална низа што ги претставува конечните центри на кластерите добиени со алгоритмот k-means. Секој ред во center низата претставува центар на кластерот, а трите колони ги претставуваат вредностите на бојата RGB на тој центар.

center = np.uint8(center)- Центарот на кластерите се претворени во 8-битни цели броеви. Ова често се прави за да се прикаже бојата во сликите.

print(center)- Со оваа линија код во функцијата cartoonize го печатиме на стандарден излез центарот на кластерите.

result = center[label.flatten()] - Во оваа линија код, целта е да се заменат вредностите на пикселите од влезната слика со соодветните вредности на центарот на кластерот што се добиени од алгоритмот за кластерирање k-means.

label - претставува еднодимензионална низа која што ја објаснивме погоре.

label.flatten(): Низата label е 2D низа со иста форма како и влезната слика. Меѓутоа, за да извршиме индексирање според елементите, треба да ја израмниме оваа 2D низа во 1D низа. Израмнувањето значи распоредување на сите елементи во еден ред.

center- Дводимензионална низа која што ги претставува конечните центри на кластерите добиени со k-means алгоритмот.

center[label.flatten()]: Со користење на израмнетата label низа како индекс, оваа операција ја пресликува ознаката на кластерот на секој пиксел на соодветниот центар на кластерот во централната низа. Ова ефикасно ги заменува RGB вредностите на секој пиксел со RGB вредностите на центарот на кластерот на кој му припаѓа.

result = result.reshape(img.shape)

img.shape: Ова е низа што ги претставува димензиите на оригиналната влезна слика img. Содржи информации за бројот на редови, бројот на колони и бројот на канали во боја на сликата.

result.reshape(img.shape): Оваа операција ја преобликува 1D низата со резултати за да одговара на обликот наведен со img.shape. Резултирачката низа ќе има ист број на редови, колони и канали во боја како и оригиналната влезна слика.

На пример, ако оригиналната влезна слика img е слика во боја со димензии (висина, ширина, 3) каде што 3 ги претставува RGB каналите во боја, тогаш img.shape ќе биде нешто слично (висина, ширина, 3). Низата со резултати, која беше израмнета порано, треба да се преобликува за да одговара на оваа форма за правилно да ја претстави групираната слика.

Значи, линијата result = result.reshape(img.shape) суштински го трансформира резултатот од 1D низата назад во повеќедимензионална низа што одговара на обликот на оригиналната слика. Овој чекор е клучен за да се осигура дека групираната слика е во правилен формат за понатамошна обработка или визуелизација.

На крај функцијата cartoonize() ја враќа резултантната слика result.

Слика бр 6. Оригинална слика (лево) и квантизација на бои (десно)

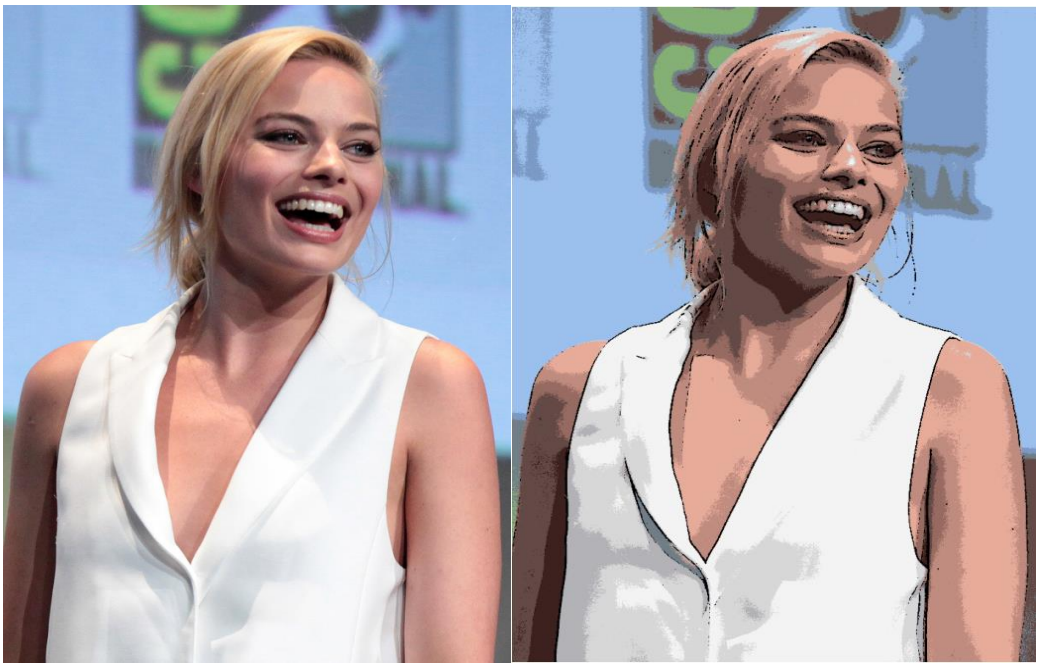
2.4.Комбинирање на маската со рабови и квантизираната слика

Последниот чекор е комбинирање на сликата со намален број на бои, односно квантизираната слика добиена во чекор 3, со маската со рабови добиена во чекор 2.

```
# Combine the result and edges to get final cartoon effect

cartoon = cv2.bitwise_and(result, result, mask=edges)

# Upsample the cartoonized image to the original size

cartoon_upsampled = upsample_image(cartoon, scaling_factor)
```

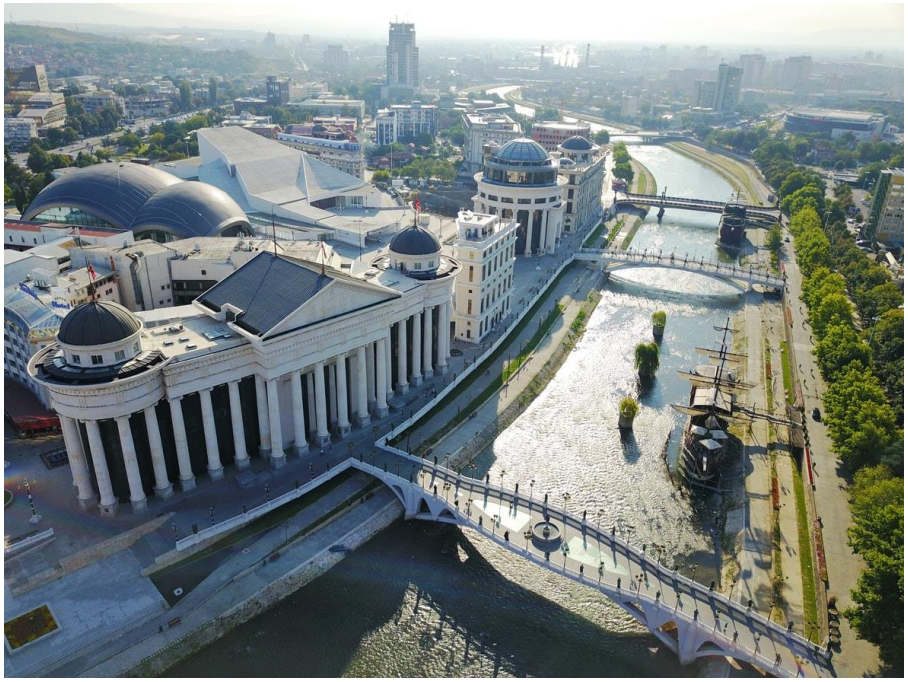
За постигнување на овој чекор ја користиме bitwise_and() функцијата од cv2 библиотеката.Оваа функција извршува операција И помеѓу две слики (првите два аргумента), пиксел по пиксел. Во нашиот случај операцијата И се извршува помеѓу истата слика result, којашто е квантизираната слика. Третиот аргумент mask = edges кажува дека операцијата И за пикселите во маската коишто имаат вредност 0 (црна боја) во излезната слика ќе имаат вредност 0, во спротивно ќе ја имаат вредноста на пикселот од сликата result. Променливата cartoon за вредност го има резултатот од оваа функција.

После ова правиме повик на претходно објаснетата функција upsample_image() со цел доколку димензиите на сликата биле намалени, скалирачкиот фактор е различен од 1, истата да се врати во првобитните димензии.



Слика бр 7. Комбинирање на маската со рабови и квантизираната слика

Во продолжение, уште неколку примери од апликацијата:



Слика бр 8. Оригинална слика

Слика бр 9. Финална слика

Слика бр 10. Оригинална и финална слика

Слика бр 11. Оригинална и финална слика

FINAL

Слика бр 12.. Оригинална и финална слика