Kvantizacija boja za uporabu Scikit-learn biblioteke za strojno učenje

Abramović, Luka

Undergraduate thesis / Završni rad

2019

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Pula / Sveučilište Jurja Dobrile u Puli**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:137:097961

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-04-17**

Repository / Repozitorij:

Digital Repository Juraj Dobrila University of Pula

Sveučilište Jurja Dobrile u Puli
Fakultet Informatike u Puli

LUKA ABRAMOVIĆ

KVANTIZACIJA BOJA ZA UPORABU SCIKIT-LEARN BIBLIOTEKE ZA STROJNO UČENJE

COLOR QUANTIZATION USING SCIKIT-LEARN MACHINE LEARNING LIBRARY

Završni rad

Pula, rujan, 2019. godine

Sveučilište Jurja Dobrile u Puli
Fakultet Informatike u Puli

LUKA ABRAMOVIĆ

KVANTIZACIJA BOJA ZA UPORABU SCIKIT-LEARN BIBLIOTEKE ZA STROJNO UČENJE

COLOR QUANTIZATION USING SCIKIT-LEARN MACHINE LEARNING LIBRARY

Završni rad

JMBAG: 0303069232, redoviti student

Studijski smjer: Informatika

Kolegij: Osnove IKT

Mentor: doc. dr. sc. Darko Etinger

Pula, rujan, 2019. godine

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisani **Luka Abramović**, ovime izjavljujem da je ovaj završni rad rezultat isključivo mojega vlastitog rada, da se temelji na mojim istraživanjima te da se oslanja na objavljenu literaturu kao što to pokazuju korištene bilješke i bibliografija. Izjavljujem da niti jedan dio završnog rada nije napisan na nedozvoljen način, odnosno da je prepisan iz kojega necitiranog rada, te da ikoji dio rada krši bilo čija autorska prava. Izjavljujem, također, da nijedan dio rada nije iskorišten za koji drugi rad pri bilo kojoj drugoj visokoškolskoj, znanstvenoj ili radnoj ustanovi.

Student

_____

U Puli, rujan, 2019. Godine

SADRŽAJ

UVOD 1
Kratka povijest fotografije i boja do danas 1
Kvatizacija boja općenito 3
RAZRADA TEME 5
Kvantizacija boja korištenjem K-Means 5
Gotovi primjer kvatizacije boja 5
Jednostavni primjer kvantizacije boja korištenjem k-means 7
Općenito o NumPy 7
Općenito o OpenCV 8
Primjer kvantizacije boja upotrebom scikit-learn biblioteke 13
Općenito o SciPy 13
Općenito o joblib 13
Glavna obilježja joblib-a 14
Instalacija scikit-learn biblioteke 15
Korištenje Anaconde i Jupyter Notebook-a 15
Pisanje koda u Jupyter Notebooku 17
ZAKLJUČAK 23
LITERATURA 25

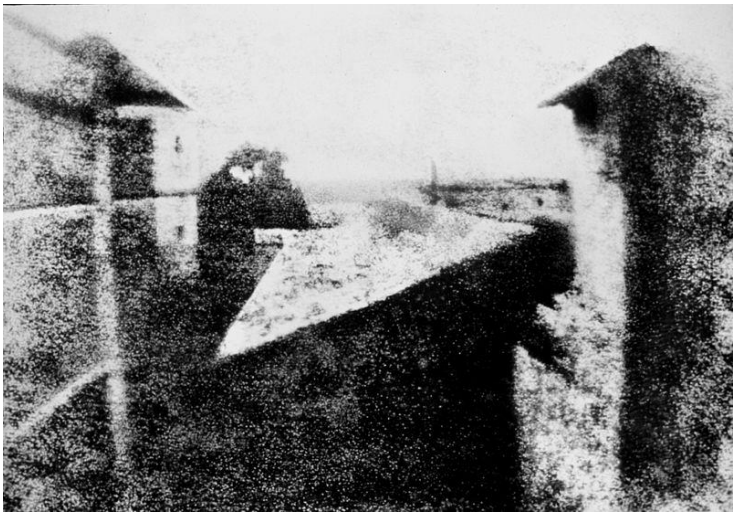
UVOD

Kratka povijest fotografije i boja do danas

Pojam fotografije i pojam boja danas je već poznat svim generacijama korisnika računala, od djece u predškolskoj do odraslih osoba treće dobi. Izum prve kamere, a samim time i pojam fotografije predstavljao je revoluciju u grafičkom dizajnu. Pojavom *camere obscure,* preteče današnjih fotoaparata, počela su razna dugogodišnja istraživanja kako bi se mogao uhvatiti određeni trenutak u stvarnosti i sačuvati na određenom opipljivom materijalu.

Profesor Joseph Petzval, je na Sveučilištu u Beču 1859. ovako definirao *cameru obscuru* i pojam fotografije: "*Camera obscura* može se definirati kao instrument za dobivanje, na ograničenoj udaljenosti, slike bilo kojeg broja objekata; te se u skladu s ovom definicijom brojna svojstva odjednom smatraju kao poželjnim. Bilo bi dobro u prvom redu navesti svojstva koja se mogu razumno zahtijevati, kako bi se naknadno ispitalo koliko takvi zahtjevi mogu biti zadovoljeni. Možemo razumno zahtijevati da slika bude dobro definirana ili oštra, da također bude dobro osvijetljena tako da pokazuje odgovarajuću svjetlost i hladovinu; nadalje, da će biti istinita prirodi, kao i da će ležati u ravnini. Ako je moguće, kamera bi također trebala istovremeno pružati slike kako bliskih tako i udaljenih objekata, trebala bi posjedovati veliko vidno polje i pružati uz zadovoljstvo velike ili male slike. I na kraju, instrument mora imati prikladan oblik i koštati što je manje moguće."[1]

Međutim problem sa prvim fotografija je bio kako zapravo sačuvati fotografiju. Postojala je mogućnost da se fotografija dobije korištenjem mračne komore u koju se pušta svjetlo kako bi se dobila boja, no i dalje je ostao glavni problem – trajnost fotografije. Taj problem je riješio Nicéphore Niepce tijekom 1820-ih godina u partnerstvu sa Jacques Mandé Daguerreom. Niepce se također smatra i autorom prve sačuvane fotografije u povijesti.



Slika 1. Nicéphore Niepce, prva fotografija

[1] Philosophical magazine and Journal of Science, fourth series, siječanj 1859.

Slika 2. Primjer crno-bijele blur fotografije

Kroz desetljeća su se pojavila mnoga istraživanja koja su dovela do postepenog usavršavanja tehnika fotografije. Fotografija se postepeno sve više usavršavala, ali unatoč usavršavanju, bila su potrebna desetljeća da se određeni trenutak na fotografiji uhvati onako kakav je, odnosno u boji. Iako su još u davnoj povijesti fotografije bile u mogućnosti stvoriti se pomoću medija koji reproducira boje, fotografija u boji je prevagnula u svakodnevnom korištenju tek tijekom 70ih godina prošlog stoljeća. Danas je fotografija kao takva naprednija nego ikad, više nema osobe koja ne može stvoriti fotografiju, svaki mobilni telefon ima kameru, a mobilni telefoni su danas kao medij za stvaranje fotografija čak i još više zastupljeni od današnjih digitalnih fotoaparata, zbog praktičnosti. Digitalne i napredne fotoaparate danas uglavnom koriste profesionalni fotografi i ljudi koji 'žive' od stvaranja fotografija. A danas se i gotovo svakim danom kvaliteta fotografije sve više i više povećava, od veličine (rezolucije) do kontrasta i svih ostalih aspekata.

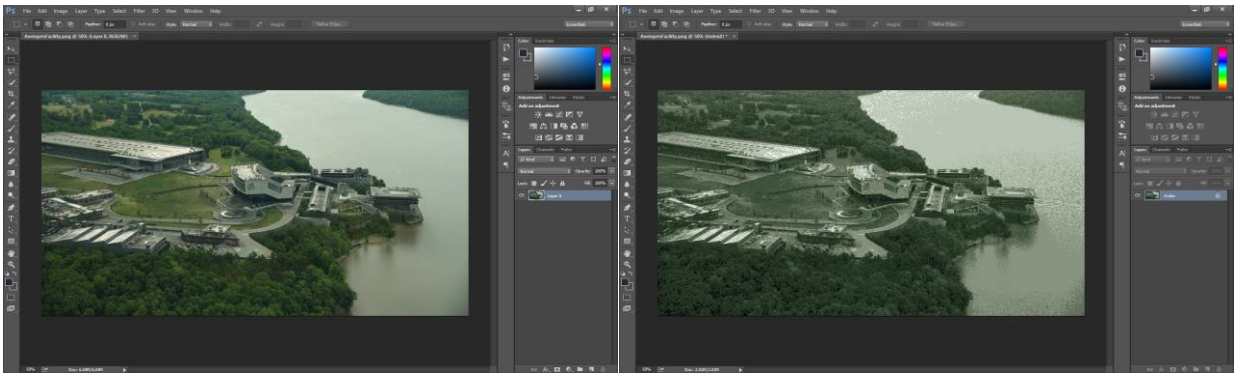
I ptice na grani znaju da danas živimo u digitalnom dobu, a proces digitalizacije se naravno, možda čak i u najvećoj mjeri, odrazio i na fotografije. Iako je nemoguće doći do konkretne brojke, postoji neka pretpostavka da se još u 2014. na društvene mreže postavljalo čak 1.8 milijardi fotografija dnevno,[2] a ako uzmemo u obzir da ne mora baš svaka uslikana slika završiti online, broj uslikanih fotografija općenito je zasigurno i još strahovito veći, možda čak i beskonačan.

[2] https://www.businessinsider.com/were-now-posting-a-staggering-18-billion-photos-to-social-media-every-day-2014-5

Kvatizacija boja općenito

Iako možda toga nismo svijesni, pojam kvantizacije boja susrećemo svakog dana u današnje digitalno doba. Sama kvatizacija boja danas se može provoditi automatski ili ručno. Primjerice, automatska kvatizacija se koristi prilikom jednostavne promjene filtera, efekata i ostalih postavki fotografija i videozapisa prilikom, recimo, objavljivanja fotografije na društvenim mrežama. Danas postoje automatski alati za obrađivanje fotografije koji se nalaze u svim mobilnim (pametnim) telefonima, a isto tako, aplikacije koje primarno služe za objavljivanje fotografija (npr. Instagram i Flickr), sadrže svoje vlastite mogućnosti za promjenu efekta boja, ali i kontrast, razinu svjetlosti fotografije i sl. Danas je korištenje crno-bijele tehnike vrlo popularno jer, iako su slike u boji sada dominantne u odnosu na crno-bijele, crno-bijela tehnika se još uveliko koristi, pogotovo kod umjetničkih fotografija, a isto tako se i često koristi kod svakodnevnih korisnika kao efekt za obradu fotografije prije objavljivanja.

Također, proces kvatizacije boja može se provoditi i manulano (ili ručno) u programima za obradu fotografija (npr. Adobe Photoshop) čije mogućnosti omogućavaju izmjene kontrasta boja i radi estetskog uljepšavanja fotografije prema željama korisnika (ili klijenta ukoliko korisnik uređuje fotografije za nekog drugog). Postoji i ugrađena podrška za kvantizaciju boja u mnogim rasterskim programima za obradu fotografije. Jedan od njih, i onaj najočitiji primjer je i taj već spomenuti Photoshop. Kvatizacija boja korištenjem tog softvera se radi izvodi korištenjem indeksiranih boja *(Image -> Mode -> Indexed Color).*



Slike 3. i 4. Primjer kvantizacije boja korištenjem programa Adobe Photoshop

Osim toga, iako se možda takav pristup i najrijeđe koristi, kvatizaciju boja je moguće provesti i pisanjem koda u programskom jeziku, kao na primjer u Pythonu. Upravo s tim ćemo se najviše pozabaviti u ovom završnom radu. Prije nego se konkretno s tim uhvatimo ukoštac, spomenut ćemo i definirati još neke određene pojmove koji su od velikog značaja u prezentaciji ovoga rada.

Strojno učenje je, najjednostavnije rečeno, dubinska analiza podataka stvari koje se ne daju ručno podesiti. Detaljnije rečeno, "strojno učenje grana je umjetne inteligencije koja se bave oblikovanjem algoritama koji svoju učinkovitost poboljšavaju na temelju empirijskih podataka. Strojno učenje jedno je od danas najaktivnijih i najuzbudljivijih područja računarske znanosti,

ponajviše zbog brojnih mogućnosti primjene koje se protežu od raspoznavanja uzoraka i dubinske analize podataka do robotike, računalnog vida, bioinformatike i računalne lingvistike."[3]

Scikit-Learn je *open-source* softver, odnosno biblioteka za strojno učenje u programskom jeziku Python. Radi se o jednostavnom alatu za rudarenje i analizu podataka. Napravljen je u NumPy, SciPy i matplotlib bibliotekama. Njegovim korištenjem može se provesti rudarenje podataka sa ili bez nadzora. Kako je objašnjeno i na naslovnoj stranici njegovog web sjedišta,[4] korištenjem scikit-learn-a moguća je upotreba:

- Klasifikacija – identificira kojoj kategoriji određeni objekt pripada.
- Regresija - predviđanje atributa kontinuirane vrijednosti koji je povezan sa objektom.
- Klasteriranje – automatsko grupiranje sličnih objekata u setove
- Dimenzionalna redukcija – smanjuje broj slučajnih varijabli koje se razmatraju
- Selekcija modela – usporedba, provjera i odabir parametara i modela
- Predprocesiranje – ekstrakcija i normalizacija značajki

[3] https://www.fer.unizg.hr/predmet/su
[4] https://scikit-learn.org/

RAZRADA TEME

"Video monitori prikazuju slike u boji modulirajući intenzitet triju glavnih boja (crvene, zelene i plave) na svakom pikselu na slici. U digitaliziranoj slici boja svaka se primarna boja kvantizira s razlučivosti od 8b kako bi se uklonili prepoznatljivi koraci kvantizacije u osvjetljenju, nijansi boje i zasićenosti boje. Dakle, digitalni zaslonski sustavi u cijeloj boji koriste 24b za specificiranje boje svakog piksela na zaslonu."[5]

Troškovi memorije velike brzine potrebne za podršku takvom zaslonu u boji u monitoru visoke razlučivosti čine mnoge aplikacije nepraktičnim. Alternativni pristup, koji se koristi na mnogim trenutno dostupnim zaslonima, je osigurati ograničen broj bitova, poput 8, za određivanje boje na svakom pikselu. Svaka od ovih 28 = 256 vrijednosti koristi se kao indeks u tablici boja koju definira korisnik. Svaki unos u tablici sadrži vrijednost 24 b koja specificira crvenu, zelenu i plavu komponentu boje. Na taj način korisniku je omogućeno da odabere mali podskup boja, nazvan paleta, iz čitavog raspona boja od 224 = 16 777 216. Nedostatak ove sheme je taj što ograničava broj boja koji se mogu istovremeno prikazati."

Kvantizacija boja korištenjem K-Means

S obzirom da se u ovom završnom radu radi o kvatizaciji boja za upotrebu scikit-learn biblioteke, koristit ćemo jedan od primjera koji se nalazi na samoj scikit-learn stranici, a radi se primjeru Kvatizacija boja korištenjem K-Means *(Color Quantization using K-means)*. U informatici se jako često susrećemo sa pojmom Rudarenja Podataka *(Data Mining)* koji je podijeljen u dvije vrste: rudarenje podataka sa nadzorom i rudarenje podataka bez nadzora. Ključna razlika između te dvije vrste je ta što su kod rudarenja podataka s nadzorom unaprijed poznati ulaz i izlaz, odnosno, smatra se da je došlo do pogreške samo ako su izlaze vrijednosti različite u odnosu na unaprijed zadane izlazne vrijednosti. U slučaju rudarenja podataka bez nadzora, poznat je samo izlaz tj. istina, te je cilj dobiti izlazne vrijednosti preko prirodne strukture skupa podataka. Primjeri metoda koje spadaju u rudarenje podataka BEZ nadzora su metoda asocijacije i metoda K-Means. K-Means metodu možemo definirati kao: grupiranje sličnih podataka, odnosno, definiranje broja grupa koje se žele dobiti uz određivanje centeroida.

"K-znači metoda je grupiranja opažanja u određeni broj međusobno povezanih skupina. "K" se odnosi na navedeni broj klastera. Postoje različite mjere udaljenosti kojima se može odrediti koja će opažanja biti dodata kojoj grupi. Algoritam ima za cilj minimizirati mjeru između centroida klastera i zadanog promatranja iterativnim dodavanjem opažanja bilo kojem klasteru i završiti kad se postigne mjera najmanje udaljenosti."[6]

Gotovi primjer kvatizacije boja

Prije nego se bacimo na konkretni zadatak koji ćemo prikazati u ovom završnom radu, kako bi smo što bolje istražili pojam kvantizacije boja korištenjem scikit-learn biblioteke, pokazat ćemo

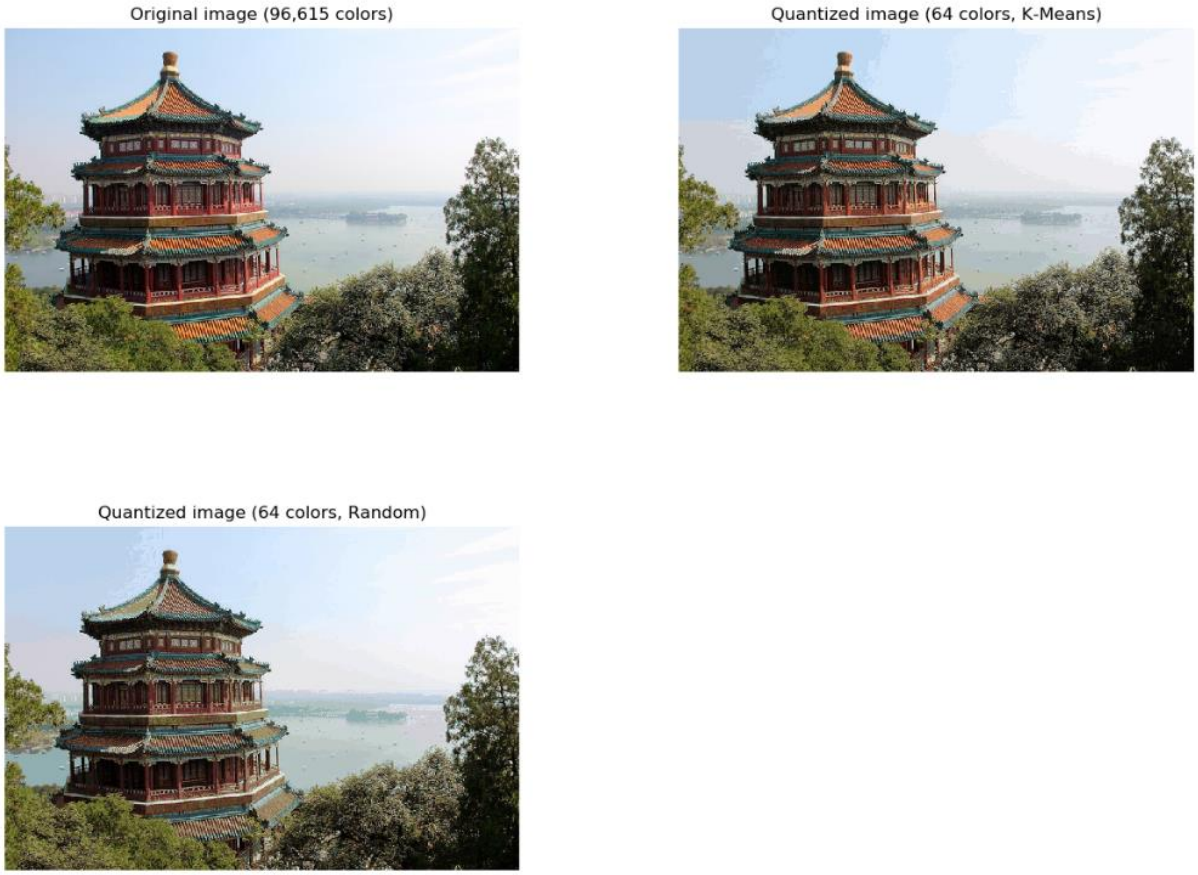
[5] Michael T. Orchard, Charles A. Bouman, Color Quantization of Images, Transactions on signal processing. vol. 39. No. 12. Prosinac 1991.
[6] http://kom.aau.dk/group/04gr742/pdf/kmeans_worksheet.pdf

jedan primjer kvatizacije boja korištenjem K-means metode koji se već nalazi na scikit-learn stranici.

Ovaj primjer sa scikit learn stranice "izvodi pikselnu Vektorsku kvantizaciju (VQ) slike ljetne palače (Kina), smanjujući broj boja potrebnih za prikazivanje slike sa 96.615 jedinstvenih boja na 64, a zadržavajući ukupnu kvalitetu izgleda. "[7]

"U ovom su primjeru pikseli predstavljeni u 3D-prostoru, a K-means koristi se za pronalaženje 64 skupina boja. U literaturi za obradu slike kodna knjiga dobivena od K-means (središta klastera) naziva se paleta boja. Korištenjem jednog bajta može se adresirati do 256 boja, dok RGB kodiranje zahtijeva 3 bajta po pikselu. Na primjer, format GIF datoteke koristi takvu paletu."[8]

"Za usporedbu, prikazana je i kvantizirana slika pomoću slučajnog koda (boje pokupljene nasumično)."[9]



Slika 5. Usporedba fotografija prije i nakon kvatizacije

[7] https://scikit-learn.org/stable/auto_examples/cluster/plot_color_quantization.html
[8] https://scikit-learn.org/stable/auto_examples/cluster/plot_color_quantization.html
[9] https://scikit-learn.org/stable/auto_examples/cluster/plot_color_quantization.html

Gore prikazani primjer je već ranije odrađen, ali u kasnijem dijelu ovog završnog rada ćemo se detaljnije pozabaviti tim primjerom, iznova ga isprobati i detaljno objasniti kod za njegovo izvršavanje.

Jednostavni primjer kvantizacije boja korištenjem k-means

Prije nego prikažemo primjer sa samim scikit-learnom, pokazat ćemo nešto jednostavniji primjer kojim ćemo izvesti donekle istu stvar, samo sa nešto jednostavnijim kodom, bez scikit-learn-a.[10]

Na početku pisanja Python skripte, potrebno je definirati potrebne biblioteke koje su potrebne za pravilnu egzekuciju koda. U ovom slučaju, potrebna nam je sveprisutna biblioteka NumPy, i možda još važnija biblioteka OpenCV, koja se brine za vizualni element, odnosno fotografiju.

Općenito o NumPy

"NumPy je temeljni paket znanstvenog računanja u Pythonu. To je knjižnica Python koja pruža objekt višedimenzionalnog niza, razne izvedene objekte (poput maskiranih nizova i matrica) i asortiman rutina za brze operacije na nizovima, uključujući matematičke, logičke manipulacije oblika, razvrstavanje, odabir, odabir, I/O , diskretne Fourierove transformacije, osnovna linearna algebra, osnovne statističke operacije, slučajna simulacija i još mnogo toga." [11]

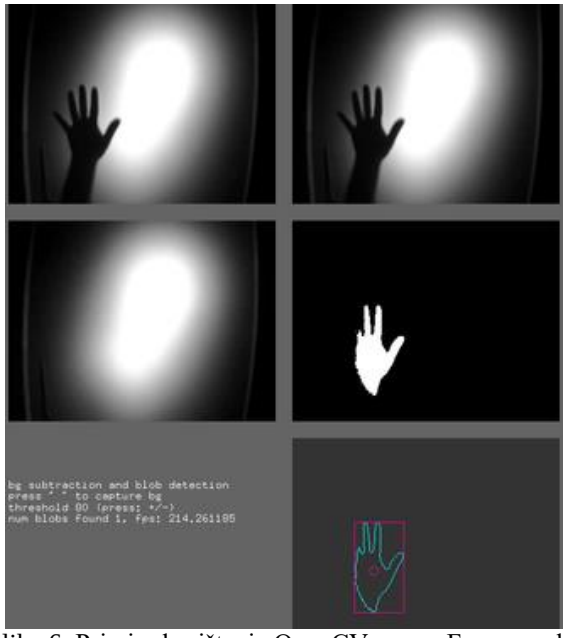
"U jezgri NumPy paketa je ndarray objekt. Ovo enkapsulira n-dimenzionalne nizove homogenih tipova podataka, pri čemu se mnoge operacije izvode u sastavljenom kodu za izvedbu. Nekoliko je važnih razlika između NumPy nizova i standardnih Python nizova:

- NumPy matrice imaju fiksnu veličinu pri kreiranju, za razliku od Python popisa (koji mogu dinamično rasti). Promjena veličine ndarray stvorit će novi niz i izbrisati izvornik.
- Svi elementi u NumPy matrici moraju biti istog tipa podataka i time će biti iste veličine u memoriji. Izuzetak: može se nalaziti niz (Python, uključujući NumPy) objekata, omogućujući tako nizove različitih veličina elemenata.
- NumPy nizi olakšavaju napredne matematičke i druge vrste operacija na velikom broju podataka. Takve se operacije obično izvode učinkovitije i s manje koda nego što je moguće koristeći Pythonove ugrađene posljedice.
- Sve veća mnoštvo znanstvenih i matematičkih Python-ovih paketa koristi NumPy matrice; da bismo učinkovito iskoristili većinu današnjeg znanstveno-matematičkog softvera koji se temelji na Pythonu, samo znati kako koristiti ugrađene vrste sekvenci Pythona nije dovoljno - treba znati i kako koristiti NumPy matrice."[12]

[10] https://www.youtube.com/watch?v=0AZXoY1ryJA
[11] NumPy User Guide, Release 1.11.0, Written by NumPy community, 2016.
[12] NumPy User Guide, Release 1.11.0, Written by NumPy community, 2016.

Općenito o OpenCV

OpenCV *(Open source computer vision)* je *open-source* biblioteka programskih funkcija za procesiranje fotografije te je bazirana na tzv. računalni vid. Stvorila ju je tvrtka Intel, a održava ju Willow Garage. Dostupna je za korištenje u programskim jezicima Python, C i C++.[13]



Slika 6. Primjer korištenja OpenCV u openFrameworks

Kako bismo ove dvije ranije biblioteke mogli normalno koristiti, bilo ih je potrebno zasebno instalirati u terminalu, odnosno cmd-u:

```
$ pip install numpy
$ pip install opencv-python
```

Jednom kada su te dvije biblioteke uspješno instalirane, možemo započeti sa pisanjem tog jednostavnog primjera kvantizacije boja korištenjem k-means. Naravno, jednom kada započnemo sa pisanjem koda, potrebno je importirati te dvije biblioteke kako bismo ih mogli neometano koristiti:

```
import numpy as np
import cv2
```

Jednom kada smo upisali taj uvodni dio skripte, možemo napisati sve potrebne naredbe, a osnovna je naravno cv2 naredba kojoj ćemo učitati fotografiju koju ćemo koristiti kao primjer:

```
#input image
inputImg = cv2.imread('C:/Users/Lukas/Desktop/cap vs cap.jpg',-1)
```

Korištenjem imread naredbne iz cv2 biblioteke omogućuje se učitavanje fotografije sa lokalnog diska, uz jasno prilaganje adrese te fotografije na lokalnom disku.

[13] Marvin Smith, Introduction to OpenCV

Slika 7. Fotografija za primjer[14]

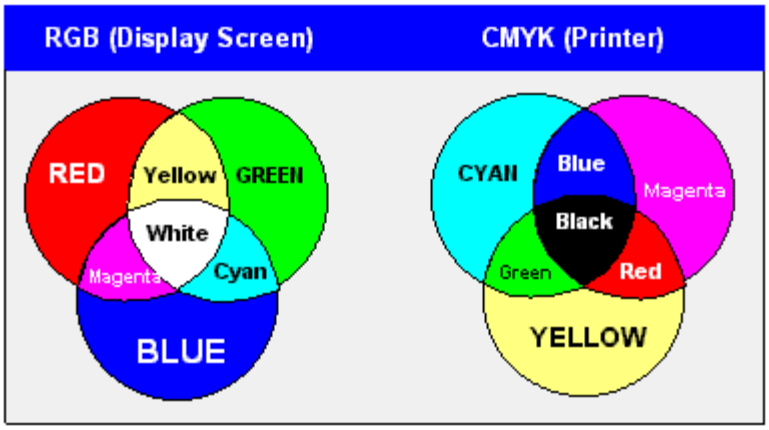
U sljedećem dijelu koda smo stvorili niz od tri stupca, koji su zaduženi za RGB model boja, odnosno vrijednosti nijansi tih boja za apsolutno svaki piksel fotografije, također je potrebno upisati dodatnu naredbu zbog činjenice da cv2.kmeans podržava samo float32 vrijednosti:

```
#Creating an array of 3 columns corresponding to R,G,B values of all
pixels
samples = inputImg.reshape((-1,3))

#Changing because cv2.kmeans tako only float32 values
samples = np.float32(samples)
```

RGB – "(*Red, Green, Blue*) odnosi se na sustav za predstavljanje boja koje se koriste na zaslonu računala. Crvena, zelena i plava mogu se kombinirati u različitim omjerima da bi se dobila bilo koja boja u vidljivom spektru. Razine R, G i B mogu se kretati u rasponu od 0 do 100 posto punog intenziteta. Svaka razina predstavljena je rasponom decimalnih brojeva od 0 do 255 (256 razina za svaku boju), što je ekvivalent rasponu binarnih brojeva od 00000000 do 11111111, ili šesterokutni od 00 do FF. Ukupni broj boja je 256 x 256 x 256 ili 16 777 216 mogućih boja."[15]

[14] Marvel's Avengers:Endgame, Marvel Studios, 2019.
[15] https://whatis.techtarget.com/definition/RGB-red-green-and-blue



Slika 8. Usporedba RBG i CMYK sustava za predstavljanje boja

Float32 – Koristi se za deklariranje 32-bitne numeričke varijable tip *float.* Koristi dvostruko manje memorije od *float64*, zbog čega izvođenje operacija sa tim varijablama ide puno brže. Ipak, s obzirom da se float uglavnom koristi za spremanje decimalnih brojeva, *float64* može spremiti brojeve sa puno većom preciznošću.

Sljedeći dio koda je dosta važan s obzirom da je riječ o kvatizaciji boja korištenjem k-means metode. Potrebno je zadati taj k, odnosno broj klastera za završni rezultat. Dodatak tome i postavljanje iteracijskog kriterija te dodatna linija koda zadužena za k-means klasteriranje.

```
#Setting k(Number of cluster in final result)
k_parameter = 4

#Setting k-means iterating termination criteria
criteria = (cv2.TERM_CRITERIA_EPS+cv2.TERM_CRITERIA_MAX_ITER,10,1.0)

#kmeans clustering
ret,labels,center =
cv2.kmeans(samples,k_parameter,None,criteria,10,cv2.KMEANS_RANDOM_CENT
ERS)
```

I na kraju, potrebno je vratiti fotografiju u njezinu izvornu veličinu nakon obrade, i naravno, potom prikazati tu obrađenu fotografiju. Završna linija će se pobrinuti da se prozor u kojem je ta nova fotografija prikazana zatvori pritiskom na bilo koju tipku.

```
#Changing center back to uint8, and reshaping to original image
center = np.uint8(center)
outputImg = center[labels.flatten()].reshape((inputImg.shape))

#Showing final output image
cv2.imshow('Result', outputImg)

#Closing all windows at any key press
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Jednom kada smo dovršili kod, možemo ga pokrenuti, u ovom slučaju smo koristili program IDLE (Python 3.7 32-bit), u kojem se skripta pokreće jednostavnim klikom *Run Module* u podizborniku *Run,* ili jednostavnim pritiskom na tipku F5. To pokreće Python Shell koji potom izbacuje rezultat obrađivanja fotografije spomenutim kodom.

Slika 9. Primjer fotografija nakon egzekucije koda

Već sam spomenuo da smo u ovom primjeru koristili četiri klastera (k_parameter), pa ćemo sada isprobati isti kod sa izmjenjenim brojem klastera.

Slika 10. Primjer fotografija nakon egzekucije koda sa 8 klastera

Kao što vidimo u gornjoj slici, spektar boja je nešto veći, zapravo dvostruko veći nego u prošlom primjeru. U sljedećem primjeru ćemo isprobati kod sa dvostruko manje klastera.

Slika 11. Primjer fotografija nakon egzekucije koda sa 2 klastera

Nakon egzekucije koda sa svega dva klastera, potpuno je jasno kakva je situacija. Fotografija je sačinjena od svega dva klastera, odnosno dvije boje. Unatoč minimalnom broju boja, još uvijek se može vidjeti izvorni koncept, odnosno prizor na fotografiji, iako fotografija sada izgleda samo kao sjena izvorne fotografije. Doduše, ove "sjene" se mogu iskoristiti prilikom kreiranja umjetničkih fotografija.

Na početku razrade teme ovog završnog rada smo prikazali primjer sa scikit-learn stranice, a u tom primjeru je korišten kod koji je koristio 64 klastera. Iako ćemo uskoro i kroz taj cijeli kod proći, još uvijek možemo vidjeti kako bi izgledalo pokretanje ovog jednostavnijeg koda sa istim brojem klastera.

Slika 12. Primjer fotografija nakon egzekucije koda sa 64 klastera

Kako se i dalo pretpostaviti, vidi se da je korištenjem 64 klastera spektar boja znatno veći, pa iako nije ni približan broju boja izvorne fotografije, sada je dovoljno velik da se može dobro razaznati prikaz na fotografiji te je od oka gotovo jednak onom izvornom.

Primjer kvantizacije boja upotrebom scikit-learn biblioteke

Za rad sa scikit-learn bibliotekom potrebno ju je, naravno, prvo instalirati, no treba imati na umu da korištenje scikit-learna zahtjeva i već spomenutu biblioteku NumPy, ali i biblioteke SciPy te joblib, koje možemo jednostavno instalirati korištenjem pip naredbe u command promptu. Budući da u trenutku pisanja ovog rada na računalu nije bila instalirana biblioteka matplotlib, koja će nam kasnije također biti potrebna, bilo je potrebno i nju instalirati na isti način.

```
$ pip install scipy
$ pip install joblib
$ pip install matplotlib
```

Informativno, samom instalacijom biblioteke matplotlib, automatski su instalirani paketi: matplotlib, six, python-dateutil, cycler, kiwisolver i pyparsing.

Općenito o SciPy

"SciPy je open-source ekosustav temeljen na Pythonu, softver za matematiku, znanost i inženjerstvo. Konkretno, ovo su neki od osnovnih paketa: NumPy, SciPy library, matplotlib, IPython, Sympy i pandas."[16] "SciPy se odnosi na nekoliko povezanih, ali različitih entiteta:

- SciPy ekosustav, zbirka softvera otvorenog koda za znanstveno računarstvo u Pythonu.
- Zajednica ljudi koji ovu hrpu koriste i razvijaju.
- Nekoliko konferencija posvećenih znanstvenom računanju u Pythonu - SciPy, EuroSciPy i SciPy.in.
- SciPy biblioteka, jedna od komponenti SciPy skupa, pruža brojne numeričke rutine.

Na temelju toga, SciPy ekosustav uključuje opće i specijalizirane alate za upravljanje podacima i računanje, produktivno eksperimentiranje i računanje visokih performansi."[17]

Općenito o joblib

"Joblib je skup alata za opskrbu laganim cjevovodima na Pythonu. Posebno:

- Transparentno predmemoriranje funkcija diskova i lijena ponovna procjena (memoriranje uzoraka)
- Jednostavno paralelno računanje

[16] https://www.scipy.org/
[17] https://www.scipy.org/about.html

Joblib je optimiziran da bude brz i robustan, posebno na velikim podacima i ima specifične optimizacije za numeričke nizove. Vizija je pružiti alate za lakše postizanje boljih performansi i obnovljivosti pri radu s dugotrajnim poslovima:

- Izbjegavajte dvaput izračunati istu stvar: kod se često iznova i iznova ponavlja, na primjer, prilikom izrade protokola za teške računske poslove (kao što je to slučaj u znanstvenom razvoju), ali ručno izrađena rješenja za ublažavanje ovog problema podliježu pogreškama i često dovode do neupadljivih rezultata.
- Ustrajati na disku transparentno: teško je učinkovito ustrajati proizvoljne objekte koji sadrže velike podatke. Korištenjem joblib mehanizma za spremanje sprječava se upornost pisana rukom i implicitno povezuje datoteku na disku s kontekstom izvršenja izvornog Python objekta. Kao rezultat toga, upornost joblib-a dobra je za obnavljanje statusa aplikacije ili računarskog posla.

Joblib addresses these problems while leaving your code and your flow control as unmodified as possible (no framework, no new paradigms). "[18]

Glavna obilježja joblib-a

1. "Prozirno i brzo predmemoriranje izlazne vrijednosti: memoriranje ili slična funkcionalnost za Python funkcije koje dobro rade za proizvoljne Python objekte, uključujući vrlo velike numeričke nizove. Odvojite upornost i logiku izvođenja protoka od logike domene ili algoritamskog koda pisanjem operacija kao skupa koraka s dobro definiranim ulazima i izlazima: Python funkcije. Joblib može svoje spremanje spremiti na disk i ponovo ga pokrenuti samo ako je potrebno:

```
>>> from joblib import Memory
>>> cachedir = 'your_cache_dir_goes_here'
>>> mem = Memory(cachedir)
>>> import numpy as np
>>> a = np.vander(np.arange(3)).astype(np.float)
>>> square = mem.cache(np.square)
>>> b = square(a)
________________________________________________________________________________
[Memory] Calling square...
square(array([[0., 0., 1.],
       [1., 1., 1.],
       [4., 2., 1.]]))
_____________________________________________________square - 0...s, 0.0min

>>> c = square(a)
>>> # The above call did not trigger an evaluation
```

Slika 13. Prvi primjer korištenja biblioteke joblib

[18] https://joblib.readthedocs.io/en/latest/

2. Sramotno paralelni pomagač: olakšati pisanje čitljivog paralelnog koda i brzo uklanjanje pogrešaka:

```
>>> from joblib import Parallel, delayed
>>> from math import sqrt
>>> Parallel(n_jobs=1)(delayed(sqrt)(i**2) for i in range(10))
[0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0]
```

Slika 14. Drugi primjer korištenja biblioteke joblib

3. Brza komprimirana postojanost: zamjena za kiseli krafnik za učinkovit rad na Python objektima koji sadrže velike podatke (joblib.dump i joblib.load)."[19]

Instalacija scikit-learn biblioteke

Scikit-learn je vrlo jednostavno instalirati. Jednom kada imamo instalirane sve potrebne biblioteke potrebne za njegov neometani rad, ponovnim korištenjem pip naredbe u cmd-u instaliramo scikit-learn biblioteku za strojno učenje (moguća je instalacija i korištenjem conde).

```
$ pip install scikit-learn
```

Korištenje Anaconde i Jupyter Notebook-a

U ranije spomenutom, jednostavnom primjeru, za skriptu u Pythonu smo pokretali preko Python Shell-a, koji je zapravo prvi i najbanalniji primjer za program koji interpretira kod napisan u programskom jeziku Python. S obzirom da je ovdje riječ o kompliciranijem i sofisticiranijem primjeru, te s obzirom da se sada koristimo scikit-learnom, za ovaj drugi primjer ćemo koristiti Anacondu i Jupyter Notebook.

Slika 15. Anaconda, logo

Slika 16. Jupyter Notebook, logo

Anaconda je *open-source* je najpoznatija platforma za korištenje kada se radi o programskim jezicima Python i R. "Anaconda distribucija otvorenog koda najlakši je način za nauku podataka Python / R i strojno učenje na Linuxu, Windows-u i Mac OS X. S više od 15 milijuna korisnika širom svijeta, to je industrijski standard za razvoj, testiranje i obuku o jednom stroju, koji omogućava pojedinim znanstvenicima podataka da:

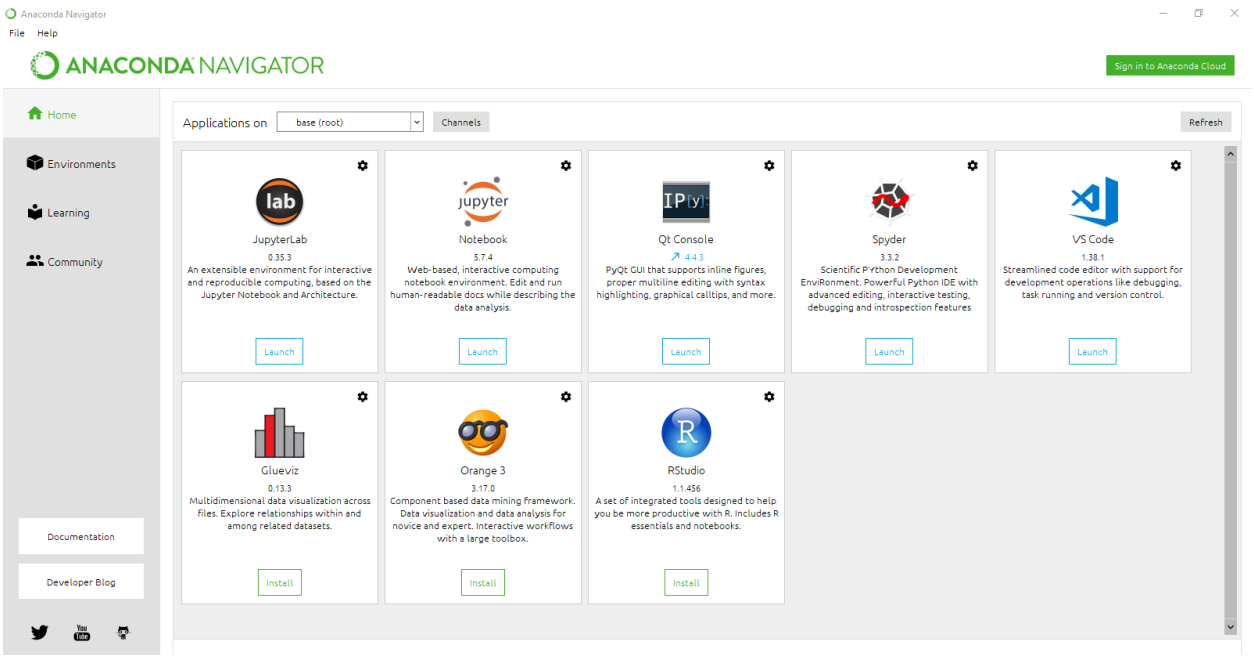
- Brzo preuzmimanje preko 1.500 Python/R paketa za znanost o podacima

[19] https://joblib.readthedocs.io/en/latest/

- Upravljajte bibliotekama, ovisnostima i okruženjima pomoću Conde
- Razvijanje i osposobljavanje modela strojnog učenja i dubokog učenja uz scikit-learn, TensorFlow i Theano
- Analiziranje podataka sa skalabilnošću i radnim učinkom pomoću Dask, NumPy, panda i Numba
- Vizualiziranje rezultata pomoću Matplotlib, Bokeh, Datashader i Holoviews"[20]

Anaconda se može pohvaliti da je svojim radom stekla povjerenje brojih svjetski-poznatih tvrtki širom svijeta. Neki od njih su i: BMW, Columbia University, HSBC Bank, National Grid, Samsung, US Army Corps and Engineers i dr.

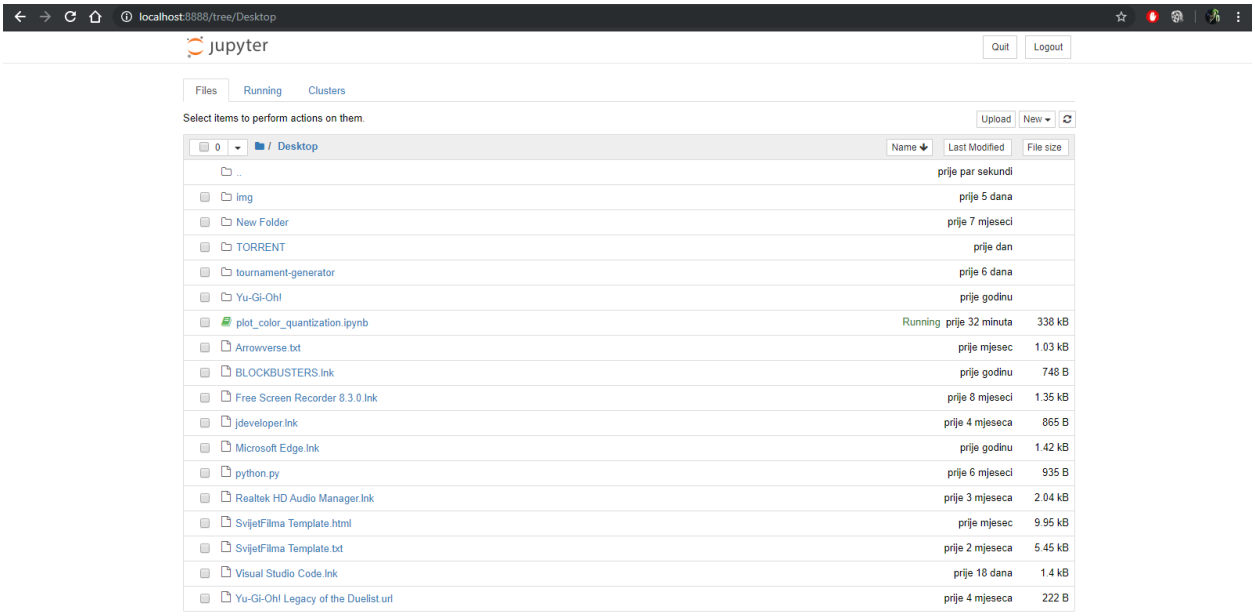
"**Jupyter Notebook** je *open-source* web-aplikacija koja omogućuje izradu i dijeljenje dokumenata koji sadrže kod uživo, jednadžbe, vizualizacije i narativni tekst. Upotrebe uključuju: čišćenje i transformaciju podataka, numeričku simulaciju, statističko modeliranje, vizualizaciju podataka, strojno učenje i još mnogo toga."[21]



Slika 17. Anaconda Navigator

Jupyter Notebook se može jednostavno pokrenuti iz Anaconda Navigatora, njegovim pokretanjem se aktivira lokalni server koji otvara mogućnosti Jupytera u web-pregledniku pod domenom *localhost:8888*. Početna stranica nakon otvaranja Jupytera pokazuje tzv. stablo svih podataka koje se nalaze na našem računalu, među tim podacima je jednostavno pronaći, eventualne, već postojeće Jupyter Notebook datoteke (ekstenzija *.ipynb) koje se nalaze na lokalnom disku.

[20] https://www.anaconda.com/distribution/
[21] https://jupyter.org/



Slika 18. Pogled na stablo podataka (mapa Desktop) u web-pregledniku nakon pokretanja Jupyter Notebooka

U gornjem desnom dijelu prozora vidimo gumb *new* (u podizborniku gumb *Python 3*)kojim započinje proces pisanja nove *.ipynb skripte u programskom jeziku Python. Novostorena skripta se automatski sprema u mapi u kojoj smo se nalazili prilikom pokretanja (Desktop) pod imenom *Untitled.ipynb.*

Pisanje koda u Jupyter Notebooku

Sada kada smo počeli sa pisanjem nove skripte, možemo početi detaljno objašnjavati kod sa scikit-learn web-stranice,[22] ali dio po dio kako bismo bili sigurni da je svaki dio koda, odnosno skripte, točan. Započinjemo sa klasičnim pisanjem početnog dijela koda:

```
print(__doc__)
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.metrics import pairwise_distances_argmin
from sklearn.datasets import load_sample_image
from sklearn.utils import shuffle
from time import time
```

Jednom kada smo upisali početni dio koda, možemo kliknuti na *Run* kako bismo bili sigurni da je taj dio koda dobro napisan. Jednom kada ga pokrenemo, kao odgovor ćemo dobiti sljedeću poruku: *Automatically created module for IPython interactive environment* (Automatski stvoren modul za IPython interkativno okruženje) što pokazuje da imamo sve potrebne predispozicije za daljnje pisanje koda. Ovdje vidimo da ćemo u daljenjem razvoju koda koristiti već odprije spomenute biblioteke numpy i matplotlib. Također koristimo pakete koje smo dobili samom instalacijom scikit-learna. Važno je uočiti paket slearn.cluster koji omogućuje korištenje K-

[22] https://scikit-learn.org/stable/auto_examples/cluster/plot_color_quantization.html

means metode, kao i slearn.datasets kojeg ćemo kasnije dodatno objasniti budući da je ključan za dio sa učitavanjem fotografija koja su korištena kao primjer za testiranje konačnog koda.

```
n_colors = 64

china = load_sample_image("flower.jpg")

china = np.array(china, dtype=np.float64) / 255

w, h, d = original_shape = tuple(china.shape)
assert d == 3
image_array = np.reshape(china, (w * h, d))
```

U ovom drugom dijelu koda zadajemo broj klastera, odnosno boja koje će imati fotografija nakon egzekucije skripte. Nakon toga zadajemo varijablu za učitavanje fotografije na kojoj će kod izvršiti svoje zadatke. Treći dio ovog dijela koda koristi numpy niz kako bi se koristila varijabla tipa float (64-bitni) umjesto zadanog integera od 8 bita. Podjela na 255 je važna kako bi plt.imshow funkcija radila bez problema sa podacima tipa float.[23] Nakon toga se uzimaju dimenzije učitane fotografije, odnosno originalne dimenzije te se transformiraju u dvodimenzionalni numpy niz. Za to je korišten *tuple,* odnosno vrsta niza, ali treba naglasiti da tuple niz je poredan prema zadanom poretku te se taj redoslijed ne može izmjeniti.[24] *Assert,* odnosno tvrdnja je zapravo, generalno gledano, činjenica koja se navodi u programu. Primjerice, u slučaju djeljenja, činjenica je da djeljitelj ne smije biti jednak nuli, a *assert* se brine da on to i nije.[25] Jednom kada je i to zadano, koristi se image_array koji će poprimiti vrijednosti nakon tog numpy preoblikovanja fotografije.

```
print("Fitting model on a small sub-sample of the data")
t0 = time()
image_array_sample = shuffle(image_array, random_state=0)[:1000]
kmeans = KMeans(n_clusters=n_colors,
random_state=0).fit(image_array_sample)
print("done in %0.3fs." % (time() - t0))

print("Predicting color indices on the full image (k-means)")
t0 = time()
labels = kmeans.predict(image_array)
print("done in %0.3fs." % (time() - t0))

codebook_random = shuffle(image_array, random_state=0)[:n_colors]
print("Predicting color indices on the full image (random)")
t0 = time()
labels_random = pairwise_distances_argmin(codebook_random,
                                          image_array,
                                          axis=0)
print("done in %0.3fs." % (time() - t0))
```

[23] https://scikit-learn.org/stable/auto_examples/cluster/plot_color_quantization.html
[24] https://www.w3schools.com/python/python_tuples.asp
[25] https://www.programiz.com/python-programming/assert-statement

U ovom dijelu koda zadajemo naredbu print koja će nam dati do znanja da je došlo do "uklapanja modela u njegov manji pod-uzorak podatka," kao i da je došlo do općeg vršenja zadataka zadanih u kodu na već odprije učitanoj fotografiji. Zadane su i varijable sa vremenom kako bi program mogao ispisati koliko je vremena bilo potrebno za izvršavanje svakog od tih zasebnih zadataka, što se poziva u naredbi print u zadnjeg redu svakog dijela koda. Također vidimo i da se prvi dio koda brine da broj boja na fotografiji upije vrijednost boja klastera kojega smo zadali ranije. Drugi i treći dio ovog dijela koda koriste oznake *(labels)*. Drugi dio predviđa broj indeksa boja na cijelog fotografiji korištenjem k-means metode (grupiranjem i određivanjem centeroida, dok treći dio radi to isto samo na *random* način, bez k-meansa, kako bi se na kraju mogle uočiti razlike kojih će biti, iako će oba ta rezultata, odnosno fotografije se sastojati od istog broja boja.

```
def recreate_image(codebook, labels, w, h):
    """Recreate the (compressed) image from the code book & labels"""
    d = codebook.shape[1]
    image = np.zeros((w, h, d))
    label_idx = 0
    for i in range(w):
        for j in range(h):
            image[i][j] = codebook[labels[label_idx]]
            label_idx += 1
    return image
```

Prije ispisivanja i prikaza konačnih rezultata, definirana je funkcija za rekreiranje slike koja koristi dimenzije i oznake, dodatne varijable, funkciju zeros iz numpy bibilioteke koja postavlja vrijednosti na nulu, ali ih u konačnici preko ugnježđene for petlje nanovo sastavljaju fotografiju prema svojoj visini i širini. Ta funkcija na kraju vraća fotografije koje će se prikazati kao rezultat.

```
plt.figure(1)
plt.clf()
plt.axis('off')
plt.title('Original image (96,615 colors)')
plt.imshow(china)

plt.figure(2)
plt.clf()
plt.axis('off')
plt.title('Quantized image (64 colors, K-Means)')
plt.imshow(recreate_image(kmeans.cluster_centers_, labels, w, h))

plt.figure(3)
plt.clf()
plt.axis('off')
plt.title('Quantized image (64 colors, Random)')
plt.imshow(recreate_image(codebook_random, labels_random, w, h))
plt.show()
```

Završni dio koda omogućuje konačno prikazivanje rezultata i fotografija nakon što je kod obavio kvantizaciju boja i sa i bez k-means metode. Prvi dio tog završnog dijela pokazuje originalnu fotografiju koju smo učitali i na kojoj je program radio, a potom i dvije izmjenjene fotografije,

jednu koja je prošla proces kvantizacije korištenjem k-means metode i jednu koja je prošla proces kvatizacije bez korištenja k-means metode.

```
Automatically created module for IPython interactive environment
Fitting model on a small sub-sample of the data
done in 2.732s.
Predicting color indices on the full image (k-means)
done in 1.846s.
Predicting color indices on the full image (random)
done in 1.565s.
```

Slika 19. Original image (96,615 colors)

Slika 20. Quantized image (64 colors, K-Means)

Slika 21. Quantized image (64 colors, Random)

Originalna fotografija koju smo učitali se sastoji od sveukupno 95,615 boja, dok se ostale dvije sastoje od svega 64 boje, iako je konceptualni prikz fotografije zadržan što je bolje moguće. Još jedna razlika postoji u tome što je korištenjem k-means metode, unatoč tom vrlo velikom smanjenju broja boja, prikaz fabule zadržan puno uvjerljivije, dok je kvatizacija bez k-means metode dala fotografiju koja nekako puno više djeluje kao crtež, odnosno umjetička slika koja djeluje obojana temperom.

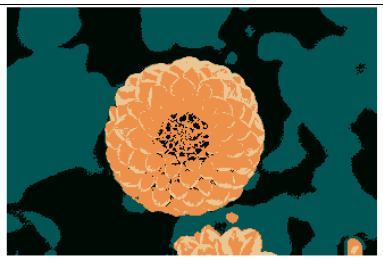
U prvom, jednostavnijem primjeru u ovom radu smo isprobali nekoliko procesa kvatizacije boja, sa različitim brojem klastera. To ćemo sada napraviti i ovdje, jednostavnom promjenom vrijednosti varijable n_colors. Korištenjem samo 8 klastera (boja):

Slika 22. Original image (96,615 colors)

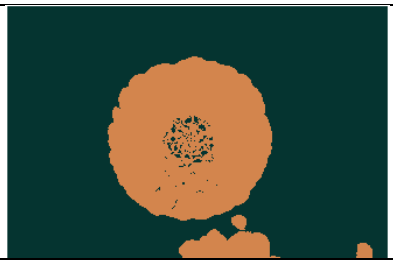
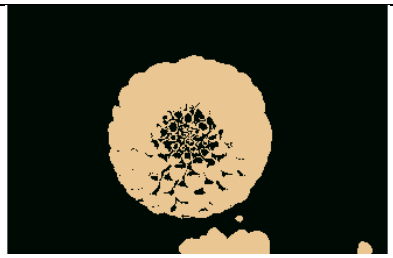
Slika 23. Quantized image (8 colors, K-Means)

Slika 24. Quantized image (8 colors, Random)

Iako smo to mogli vidjeti i u prethodnom primjeru, vidimo da je korištenjem kvatizacije boja sa k-means metodom, iako ima isti broj boja kao i fotografija koja je kvantizirana bez k-means metode, vidljivo je da je intenzitet boja nešto veći što daje puno bolju vjerodostojnost fotografije prema njezinom originalu, dok fotografija kvatizirana bez k-means metode djeluje poprilično blijedo. U sljedećem primjeru ćemo isprobati kvatizaciju boja sa 4 klastera.

		
Slika 25. Original image (96,615 colors)	Slika 26. Quantized image (4 colors, K-Means)	Slika 27. Quantized image (4 colors, Random)

Vidimo da kvatizirane slike sa samo 4 boje djeluju kao slike koje su samo narisane kistom, s tim da se naravno zbog korištenja k-means metode vidi da je fotografija koja je tu metodu koristila ispala nekako puno kvalitetnije i vizualno ugodnije.

		
Slika 28. Original image (96,615 colors)	Slika 29. Quantized image (2 colors, K-Means)	Slika 30. Quantized image (2 colors, Random)

Naravno, evidentno je slika ne može poprimiti ni približno originalni prikaz sa samo dvije boje na raspolaganju te da, kao što je rečeno i onom prvom primjeru na početku rada, ove kvantizirane fotografije djeluju samo kao sjena originalne, s tim da je fotografija kvatizirana k-means metodom zadržala barem to malo boja iz originalnog prikaza, dok fotografija kvatizirana bez k-means metode djeluje kao svojevrsni negativ fotografije koja je kvatizirana sa k-means metodom.

Ipak, važno je spomenuti jednu dodatnu stvar. U ovom primjeru sa scikit-learna koristili smo fotografiju sa cvijetom (flower.jpg), a u gotovom primjeru na početku razrade ove teme je korištena i fotografija sa kineskim pejzažom (china.jpg), za te slike nije bilo potrebno upisivati njihovu punu adresu na lokalnom disku jer je u scikit-learn dataset-u zadano da se te slike povlače iz mape *images*, koja je dio instaliranog scikit-learn paketa.

```
C:\Users\Lukas\Anaconda3\pkgs\scikit-learn-0.20.1-py37h343c172_0\Lib\site-packages\sklearn\datasets\images
```

Ipak, čak i ako ubacimo neku sliku unutar te mape, nazovimo je npr. 'abcde.jpg', te ju upišimo kao sliku za koju bi se trebala provesti kvatizacija boja. No, program će nam izbaciti grešku, a evo i zašto.

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-25-3744205e72bd> in <module>
     17
     18 # Load the Summer Palace photo
---> 19 china = load_sample_image("abcde.jpg")
     20
     21 # Convert to floats instead of the default 8 bits integer coding. Dividing by

~\Anaconda3\lib\site-packages\sklearn\datasets\base.py in load_sample_image(image_name)
    844                 break
    845     if index is None:
--> 846         raise AttributeError("Cannot find sample image: %s" % image_name)
    847     return images.images[index]
    848

AttributeError: Cannot find sample image: abcde.jpg
```

Slika 31. Atributska greška

Naime, iako se slika povlači iz te spomenute mape *images,* kvaka je u tome što je zadano da se slika može povući iz te mape samo ako je to definirano u datoteci *base.py* koja je također dio scikit-learn dataseta. Iako je taj base poprilično velik program, ovdje vidimo da je od ove greške sa atributima došlo u 846. liniji.

```
827     >>> from sklearn.datasets import load_sample_image
828     >>> china = load_sample_image('china.jpg')   # doctest: +SKIP
829     >>> china.dtype                              # doctest: +SKIP
830     dtype('uint8')
831     >>> china.shape                              # doctest: +SKIP
832     (427, 640, 3)
833     >>> flower = load_sample_image('flower.jpg') # doctest: +SKIP
834     >>> flower.dtype                             # doctest: +SKIP
835     dtype('uint8')
836     >>> flower.shape                             # doctest: +SKIP
837     (427, 640, 3)
838     """
839     images = load_sample_images()
840     index = None
841     for i, filename in enumerate(images.filenames):
842         if filename.endswith(image_name):
843             index = i
844             break
845     if index is None:
846         raise AttributeError("Cannot find sample image: %s" % image_name)
847     return images.images[index]
```

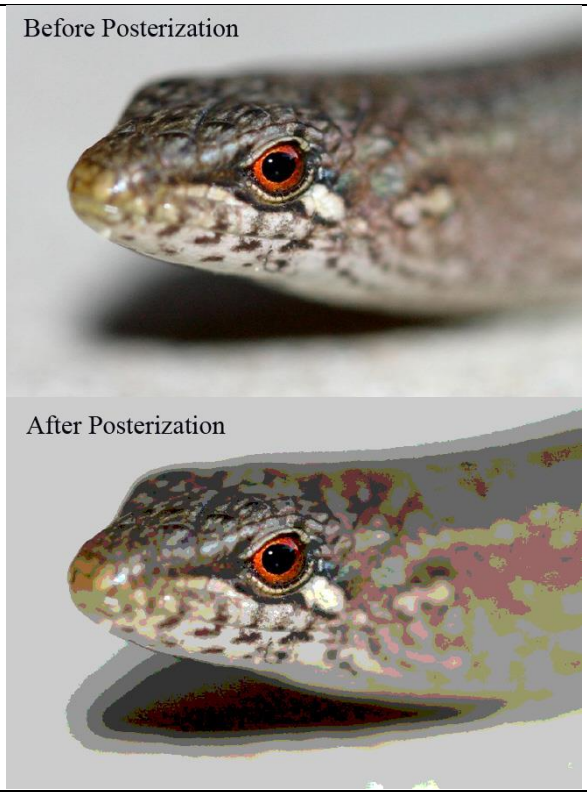
Slika 32. Dio koda iz datoteke base.py zadužen za umetanje fotografija u glavnu skriptu

U gornjoj slici vidimo da su zadane varijable i funkcije (narančasta slova) koje se pozivaju u glavnoj skripti koju smo već objasnili. Prema tome, bez obzira ako se slika nalazi u mapi, to nema smisla ako u datoteci base nema varijable koja poziva tu fotografiju, a također moraju biti zadane i precizne dimenzije fotografije.

ZAKLJUČAK

Kroz ovaj završni rad smo se detaljno upoznali kako sa fotografijom i njeznim bojama općenito, kvatizacijom boja, tako i sa k-means metodom općenito te kako spojiti te pojmove zajedno korištenjem programskog jezika Python.

Kvatizacija boja se u svakodnevnoj upotrebi koristi i prilikom posterizacije fotografije, a također postoji i ugrađena podrška za kvantizaciju boja u mnogim rasterskim programima za obradu fotografije.



Slika 33. Primjer posterizacije fotografije

U uvodnom dijelu smo se dotaknuli povijesti fotografije. Od daleke povijesti kada se fotografije nisu ni mogle trajno sačuvati, pa sve do danas, u digitalnom dobu kada dnevno nastane gotovo dvije milijarde fotografija dnevno. Danas nema što se sa fotografijom ne može izvesti. Oblikovanje fotografije dolazi u raznim oblicima i načinima. Više nije ni potrebno neko eksperimentalno znanje za bavljenje osnovnim aktivnostima koje se tiču oblikovanja fotografije. Iako su razvijeni mnogi softveri kako bi oblikovanje fotografije bilo što jednostavnije i pristupačnije, postoje i načini za oblikovanje fotografije koje koriste programeri korištenjem programskih jezika, a mi smo se u ovom završnom radu koristili popularnim Pythonom. Općenito smo objasnili i pojam kvatizacije boja, ali isto tako je bilo važno spomenuti i strojno učenje budući da je scikit-learn primarno biblioteka za strojno učenje. Bilo je potrebno objasniti i samu scikit-learn biblioteku, a kvatizacija boja korištenjem k-means metode je samo jedan od mnogih primjera koji se mogu naći na njezinom službenom web-sjedištu.

Prije nego što smo objasnili primjer sa scikit-learn stranice, napisali smo jedan jednostavni kod koji iako ne koristi scikit-learn biblioteku, na dobar, iako nešto jednostavniji način je objasnila kako radi kvatizacija boja upotrebom k-means metode. Objasnili smo osnovnu NumPy biblioteku te jednu vrlo zanimljivu OpenCV biblioteku koja je omogućila baratanje fotografijama. Taj jednostavniji kod smo objasnili na jednoj fotografiji te smo ga isprobali na više primjera, od kojih je svatko imao različit broj klastera u k-means metodi.

Detaljno smo objasnili primjer sa scikit-learn stranice, ponovno smo objasnili sve biblioteke koje je bilo potrebno instalirati kako bi program funcionirao. Iako su sve te biblioteke važne, sigurno da je bilo najvažnije instalirati tu samu scikit-learn biblioteku koja je zapravo i smisao cijelog ovog završnog rada. Testirali smo primjer korištenjem Anaconde i Jupyter Notebooka, ponovno smo izložili nekoliko primjera u kojima je korišten različit broj klastera, također smo objasnili i jednu kvaku u scikit-learnu koje zapravo nismo bili ni svejesni prije nego li smo slučajno naletili na nju. A riječ je činjenici da fotografija nad kojom želimo provesti kvatizaciju upotrebom scikit-learn biblioteke mora biti zadana u funkciji unutar skripte koja je automatski dodana u sustav instalacijom scikit-learna.

Iako pojam kvantizacije boja, kao takav, nije baš naširoko poznat osim u računalnoj branši, te je, ako nekoga zaustavite na ulici i pitate ga da vam definira što je to kvatizacija boja, vrlo je izgledno da vam ta osoba neće znati dati korektni odgovor. Ipak, i takvi ljudi su se u današnjem dobu sigurno barem jednom, iako vjerojatno indirektno, susreli sa pojmom ili procesom kvantizacije boja, a s obzirom na neprestani i svakodnevni napredak današnje tehnologije, u koju dakako spadaju i pojmovi fotografije i njezinih boja, sigurno je da će se to sve više utkati i među one korisnike računala kojima bavljenje kvatizacijom boja nije dio struke.

LITERATURA

1. Philosophical magazine and Journal of Science, fourth series, siječanj 1859.
2. We're now posting a staggering 1.8 billion photos to social media every day, Business Insider, 2014.
    a. https://www.businessinsider.com/were-now-posting-a-staggering-18-billion-photos-to-social-media-every-day-2014-5
3. Kolegij Strojno Učenje, Fakultet Elektronike i Računalstva, Zagreb
    a. https://www.fer.unizg.hr/predmet/su
4. Scikit-Learn Website
    a. https://scikit-learn.org/
5. Scikit-Learn, Plot color quantization example
    a. https://scikit-learn.org/stable/auto_examples/cluster/plot_color_quantization.html
6. Michael T. Orchard, Charles A. Bouman, Color Quantization of Images, Transactions on signal processing. vol. 39. No. 12. Prosinac 1991.
7. The K-Means Clustering Algorithm
    a. http://kom.aau.dk/group/04gr742/pdf/kmeans_worksheet.pdf
8. The Birth of Photography, A Chronology of the Experiments and Collaboration of Niépce and Daguerre
    a. https://www.hrc.utexas.edu/pressphotos/2010/gernsheim/pdf/The_Birth_of_Photography.pdf
9. Computer Vision with Open CV(Python) - Color Quantisation with K means clustering
    a. https://www.youtube.com/watch?v=0AZXoY1ryJA
10. NumPy User Guide, Release 1.11.0, Written by NumPy community, 2016.
    a. https://docs.scipy.org/doc/numpy-1.11.0/numpy-user-1.11.0.pdf
11. Marvin Smith, Introduction to OpenCV
    a. https://www.cse.unr.edu/~bebis/CS485/Lectures/Intro_OpenCV.pdf
12. SciPy Website
    a. https://www.scipy.org/
13. Scipy Website, about SciPy
    a. https://www.scipy.org/about.html
14. Joblib Documentation
    a. https://joblib.readthedocs.io/en/latest/
15. Anaconda Distribution Website
    a. https://www.anaconda.com/distribution/
16. Jupyter Website
    a. https://jupyter.org/
17. W3Schools, Python Tuples
    a. https://www.w3schools.com/python/python_tuples.asp
18. Programiz, Python programming, assert
    a. https://www.programiz.com/python-programming/assert-statement