LABORATORIJSKA VAJA 4 – FORMATI ZAPISA SLIK

IDEJA VAJE

Namen vaje je seznaniti študente s postopki izgubnega in brezizgubnega kodiranja digitalnih slik. V ta namen se bomo spoznali s formati BMP, PNG in JPEG ter njihovimi lastnosti v smislu postopka kodiranja in dekodiranja, gostote zapisa informacije ter napakami pri kodiranju oz. dekodiranju slik. Algoritme za kodiranje slik namreč glede na način zapisa slike delimo na algoritme, ki slike:

- kompresirajo (zmanjšajo velikost datoteke) in
- ne kompresirajo (sliko zapišejo v izvirni velikosti),

glede na to, če omogočajo kodiranje in dekodiranje brez napak pa na:

- izgubne (informacija o vsebini sliki se med zapisom delno zavrže oz. izgubi) in
- brez-izgubne (vsa informacija o vsebini sliki se pri zapisu ohrani).

V okviru vaje bomo poskusili udejanjiti preprosti kodirnik in dekodirnik slik v *formatu BMP*. Format BMP je format, ki se zelo pogosto uporablja v realnih razmerah, je preprost, a neučinkovit za kodiranje slik, saj gre pri BMP formatu za brez-izgubni in nekompresiran zapis vrednosti posameznih pikslov v sliki.

Nadalje se bomo v okviru vaje ogledali *format JPEG*. JPEG format omogoča izguben in kompresiran zapis slik, in je zaradi tega med formati, ki jih bomo obravnavali v okviru vaje najbolj uporaben s stališča zagotavljanja majhnih velikosti datotek. Pri kodiranju slik s tem formatom sta tako velikost in kvaliteta kodirane slike odvisna od nastavitve finosti kompresije, ki jo izbere uporabnik pri kodiranju slike. V okviru vaje se študenti seznanijo s postopkom dostopa do te nastavitve, ki jo udejanjijo s programsko python knjižnico OpenCV. Študenti sistematično ovrednotijo razmerje med gostoto zapisa informacije v sliki ter nastalo napako pri kodiranju.

V zadnjem delu vaje si bomo ogledali še *format PNG*. Format PNG kodira slike s kompresijo in brez izgube informacije. Različne izvedbe tega algoritma omogočajo različne načine kompresije, ki pa ne vplivajo na kvaliteto dekodirane slike, temveč na hitrost kodiranja in dekodiranja. Hitrejši postopki kompresije namreč manj „stisnejo“ vhodne slike in obratno. Tu študenti v okviru predstavljenih nalog sistematično ovrednotijo variante PNG kodiranja v smislu razmerja med hitrostjo kodiranja ter doseženo stopnjo kompresije.

Formata BMP in JPEG ste spoznali na predavanjih (Uvod v slikovne tehnologije). Dodatne informacije o obravnavanih foramtih, pa lahko najdete na prosto dostopnih spletnih virih kot je Wikipedia:

BMP: https://en.wikipedia.org/wiki/BMP_file_format
JPEG: https://en.wikipedia.org/wiki/JPEG
PNG: https://en.wikipedia.org/wiki/Portable_Network_Graphics

Prednosti in slabosti postopkov za kodiranje slik lahko povzamemo v sledeči tabeli:

Postopek	Prednosti	Slabosti
Nekompresiran zapis slike (kot npr.: BMP)	• Najpreprostejši postopki za kodiranje in dekodiranje • Zagotovljen zapis slike brez napak	• Največja velikost datotek
Brez-izgubna kompresija (kot npr.: PNG)	• Manjše velikosti datotek od zapisa brez kompresije • Zagotovljen zapis slike brez napak	• Počasnejše kodiranje in dekodiranje
Izgubna kompresija (kot npr.: JPEG)	• Nastavljiva velikost datotek glede na željeno kvaliteto	• Pri kodiranju pride do napak v sliki

TEORETIČNO OZADJE

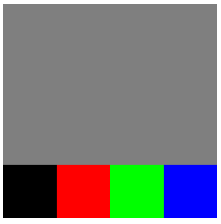
Format BMP: Za zapis barvnih slik najpogosteje uporabljamo 24-bitno barvno globino, kar pomeni, da je vsak piksel zapisan s trojico 8-bitnih vrednosti. Na nivoju celotne slike lahko torej barvno sliko interpretiramo kot trojico matrik z 8-bitnimi nepredznačenimi celoštevilskimi vrednostmi (t.j., z zalogo vrednosti celih števil z intervala $[0,255]$). V *formatu BMP* sliko shranimo tako, da za glavo datoteke neposredno zapišemo te celoštevilske vrednosti za vsak piksel posebej, eno za drugo. Glava datoteke sestoji iz:

- identifikacijskega niza „BM“, ki mu sledi
- dolžina datoteke,
- indeks začetka vrednosti pikslov,
- resolucija slike po vrsticah in stolpcih, ter
- bitna globina pikslov.

Sam zapis vrednosti pikslov ima par razlik od standardnega vrstnega reda, in sicer:

- koordinatno izhodišče je v spodnjem levem kotu slike (torej piksli si sledijo po stolpcih od leve proti desni ter od spodaj navzgod),
- vrednosti posameznega piksla pa so zapisane v vrstnem redu B,G,R.

Če vzamemo za primer sledečo sliko ločljivosti 4×4 pikslov:

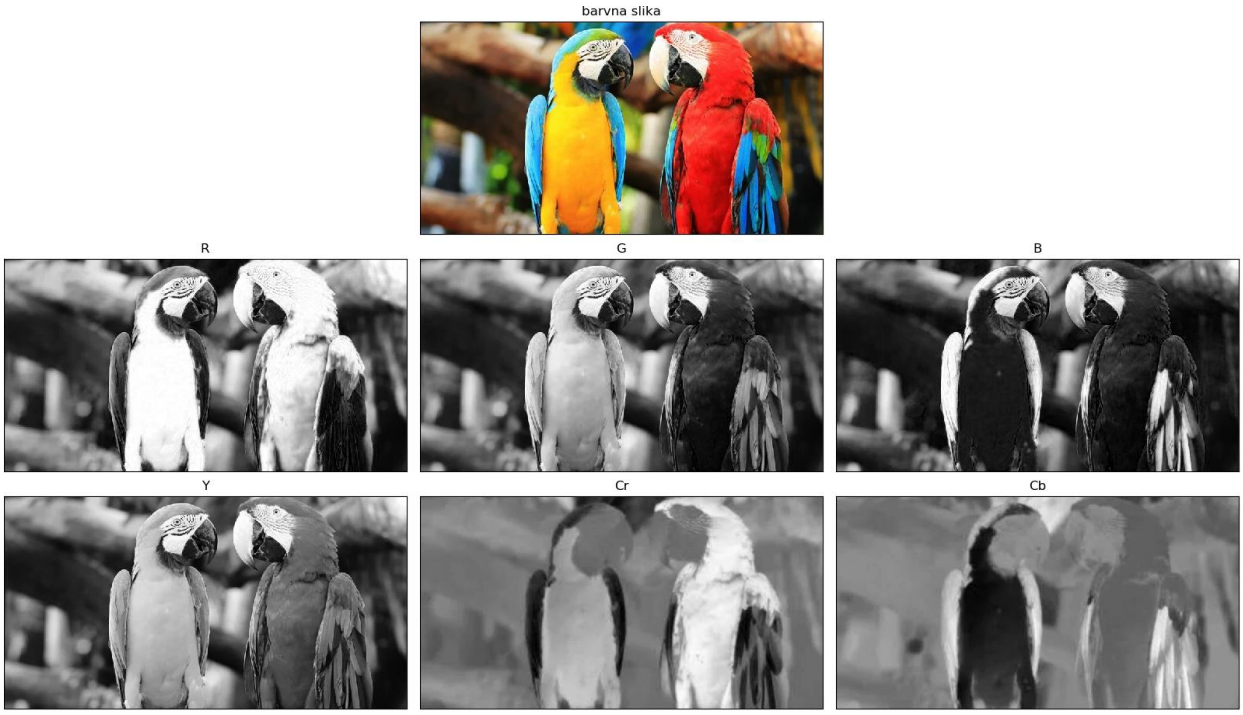


Slika je bila generirana s sledečo python kodo:

```
import struct
# For information regarding the BMP format, see:
# http://en.wikipedia.org/wiki/BMP_file_format

if __name__ == '__main__':
    # resolucija slike
    x, y = (4, 4)
    fname = 'test.bmp'
    with open(fname, 'wb') as fp:
        # format glave - little endian ter št. bajtov na argument
        fmt = '<QIIHHHH'
        # 'BM' == Windows bitmap identifier
        fp.write(b'BM' + struct.pack(fmt,
                                     # dolžina datoteke (piksli * 3 + glava)

                                     x * y * 3 + 26,
                                     # indeks prve vrednosti piksla
                                     26,
                                     # velikost DIB dela glave
                                     12,
                                     # resolucija po stolpcih in vrsticah
                                     x, y,
                                     # število slik
                                     1,
                                     # bitna globina (biti/piksel)
                                     24))

        fp.write(bytes([0, 0, 0]))
        fp.write(bytes([0, 0, 255]))
        fp.write(bytes([0, 255, 0]))
        fp.write(bytes([255, 0, 0]))
        fp.write(bytes([127, 127, 127]))
        fp.write(bytes([127, 127, 127]))
        fp.write(bytes([127, 127, 127]))
        fp.write(bytes([127, 127, 127]))
        fp.write(bytes([127, 127, 127]))
        fp.write(bytes([127, 127, 127]))
        fp.write(bytes([127, 127, 127]))
        fp.write(bytes([127, 127, 127]))
        fp.write(bytes([127, 127, 127]))
        fp.write(bytes([127, 127, 127]))
        fp.write(bytes([127, 127, 127]))
        fp.write(bytes([127, 127, 127]))
```

Iz kode vidimo, da prvi štirje zapisani piksli predstavljajo spodnjo vrstico slike, kjer imamo v BGR zapisu (0,0,0) = črna, (0,0,255) = rdeča, (0, 255, 0) = zelena in (255, 0, 0) = modra. Vsi ostali piksli v sliki so sive barve z BGR zapisom (127,127,127).

Format JPEG: Za razliko od BMP zapisa *format JPEG* temelji na principu optimizacije zapisa za informacijo, ki ima največji pomen v človeškemu sistemu vida. Prvi korak v ta namen je pretvorba barvnega prostora. Večino informacije iz slike ljudje namreč zajamemo iz svetlosti slike in ne iz njene barve. Barvni prostor RGB meša svetlost in barvo, saj vsi trije kanali nosijo informacije o obeh, tj. o svetlosti in barvi. Slika se v primeru JPEG formata zato najprej pretvori v tako-imenovani kromatski barvni prostor YCrCb, kjer kanal Y nosi informacijo o svetlosti slike, Cr in Cb pa predstavljata samo informacijo o barvi, v obliki razlike med rdečim oz. modrim kanalom in svetlostjo. Postopek pretvorbe barvnega prostora je prikazan spodaj:



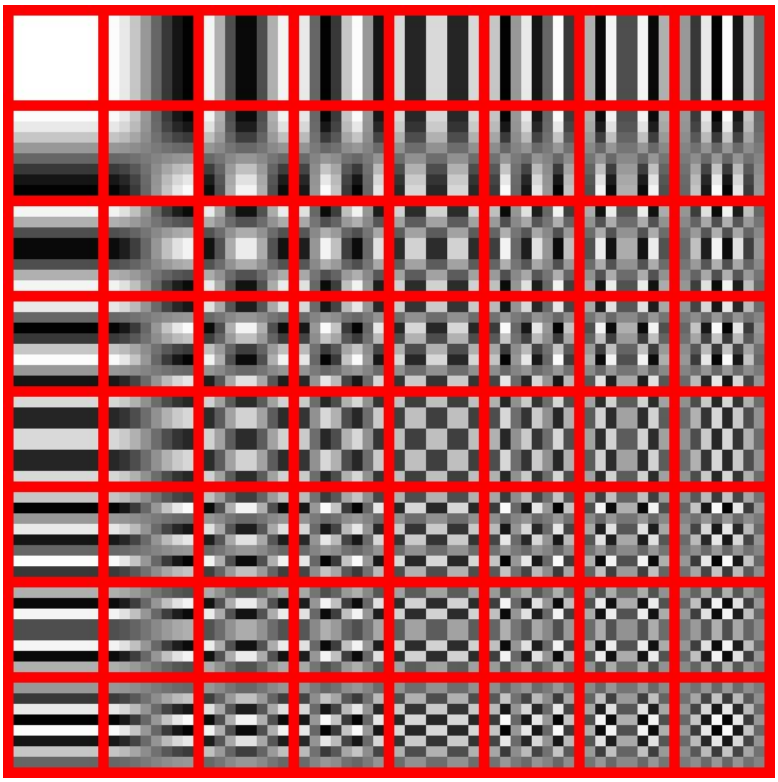
Kot je razvidno iz prikaza, so v zapisu RGB vsi trije kanali ostri in vsebujejo informacijo o svetlosti celotne slike, medtem, ko je v zapisu YCrCb oster samo kanal Y, ker je vsa informacija o svetlosti shranjena v njem, Cr in Cb pa vsebujeta samo informacijo o barvi. Ker je človeško oko bolj občutljivo na spremembo svetlosti kot na spremembo barve, lahko tu kot prvi korak kompresije kanala Cr in Cb podvzorčimo. Zapis JPEG ponuja tri načine shranjevanja kanalov Cr in Cb:

- Način 4:4:4: Brez podvzorčenja
- Način 4:2:2: Podvzorčenje za faktor 2 po stolpcih
- Način 4:2:0: Podvzorčenje za faktor 2 po stolpcih in vrsticah

Po podvzorčenju se vsak kanal posebej razbije na bloke 8×8 pikslov, t.j. na matrike g velikosti 8×8 elementov. Nato se izračuna dvorazsežna diskretna kosinusna transformacija (angl. Discrete Cosine Transform, DCT) vsakega bloka, po formuli

$$G_{u,v} = \frac{1}{4}\sum_{x=0}^{7}\sum_{y=0}^{7} g_{x,y}\, cos\left[\frac{(2x+1)u\pi}{16}\right] cos\left[\frac{(2y+1)v\pi}{16}\right],$$

kjer (x, y) predstavljata koordinate bloka pikslov g, in (u, v) predstavljata koordinate bloka DCT koeficientov G. Diskretno kosinusno transformacijo lahko interpretiramo kot projekcijo bloka pikslov na sledečo bazo vektorjev (izražavo po temeljnih funkcijah):



Po transformaciji dobimo bloke pikslov, predstavljene z DCT koeficienti. Korak DCT transformacije je brezizguben in popolnoma invertibilen, sledi pa mu izguben korak kvantizacije DCT koeficientov. Vsak DCT koeficient se deli s soležnim elementom v 8×8 kvantizacijski tabeli, nato pa se shrani samo njegov celoštevilski del. S tem se omeji zaloga vrednosti DCT koeficientov. Kot zadnji korak JPEG kompresije se nad kvantiziranimi DCT koeficienti izvede korak kompresije s Huffmannovim kodiranjem, ki bolj pogostim vrednostim pripiše krajša zaporedja bitov in obratno. Huffmanovo kodiranje je tudi pogosteje izbran algoritem za širše uporabljene ZIP arhive, zato vam tudi kompresija JPEG slik z ZIP arhivom bistveno ne zmanjša njihovih velikosti.

JPEG je izjemno priročen format za zapis digitalnih fotografij, kjer artifakti tega postopka kompresije niso lahko opazni. Pri kodiranju slik v JPEG formatu imamo tipično (npr. v knjižnjici

openCV, v programu GIMP ali drugih urejevalnikih slik) na voljo nastavitev kvalitete od 0 do 100. Vsak od nivojev kvalitete določa pripadajočo kvantizacijsko tabelo, ki nam praktično pove, katere DCT koeficiente obdržimo in katere zadušimo oziroma izbrišemo. Ko je kvaliteta nastavljena na 0, se vse komponente razen prve delijo z 2^{12}, torej je njihov celoštevilski del skoraj vedno 0. Od vsakega bloka nam tako ostane samo enosmerna komponenta, vsak 8×8 blok vhodne slike je tako predstavljen z eno barvo. To vidimo tudi, če primerjamo originalno sliko in sliko, shranjeno s tako kvaliteto:

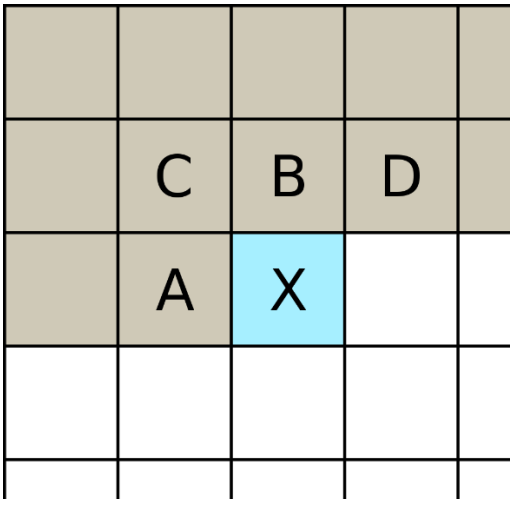
Vidimo, da so razen nekaterih blokov, ki vsebujejo visokofrekvenčne podrobnosti, skoraj vsi 8×8 bloki na spodnji sliki predstavljeni kot enobarvna področja brez dodatnih podrobnosti.

Format PNG: Za razliko od formata JPEG je *format PNG* primer brezizgubnega kodiranja. Slike še vedno kompresira in ima tipično mnogo manjše velikosti datotek od pripadajočih BMP slik z isto vsebino, hkrati pa to doseže brez popačenja vsebine slike. Na račun te karakteristike je velikost datotek PNG tipično nekje med BMP in JPG, proces kodiranja in dekodiranja pa je počasnejši.

Sam proces PNG kompresije sestoji iz dveh korakov:

- filtriranja in
- kompresije.

Pri filtriranju se surova RGB vrednost pikslov zamenja z vrednostjo, ki temelji na vrednosti sosednih pikslov. Soseščina piksla X je definirana po sledeči shemi:



Korak filtriranja glede na izbran nivo kompresije vrednost piksla X lahko zamenja z:

- X (filtriranje izklopljeno)
- $X - A$
- $X - B$
- $X - \left\lfloor \frac{A+B}{2} \right\rfloor$
- $X - P$, kjer P predstavlja tisto izmed vrednosti A, B ali C, ki je najbližja $A - B + C$

Korak filtriranja doseže, da se na popolnoma invertibilen način zmanjša zaloga vrednosti pikslov na področjih slik, ki so brez visokofrekvenčnih podrobnosti. Temu koraku sledi kompresija z algoritmom LZ77, ki poleg Huffmanovega algoritma predstavlja pogosteje uporabljen algoritem v ZIP arhivih. Ideja tu je, da algoritem sproti dopolnjuje slovar ponavljajočih se zaporedij bajtov v datoteki. Ponovitve nato nadomesti z ustrezno označenim številom ponovitev. Kot zgled, niz

```
"AAAAAAAAAAAAAAAAAAAABBBBBBBBBBAAAAAAAAAAAAAAAAAAAAAAAA"
```

lahko krajše zapišemo kot

```
20*"A"+10*"B"+25*"A"
```

V tem koraku velikost slovarja unikatnih nizov tako določa globino in hitrost kompresije.

IZVEDBA

Naloga 1 (1 točka)

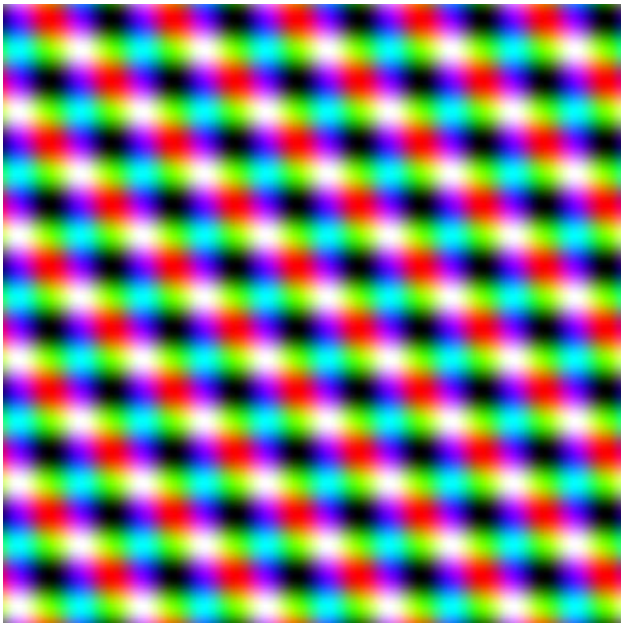
Dopolnite zgoraj podano kodo za generiranje BMP slike, tako, da zgenerirate sliko resolucije $N \times M = 512 \times 512$ pikslov. RGB barvo piksla na koordinati (x, y) določite po pravilu:

$$R = \left\lfloor 127.5 \sin\left(20\pi \frac{\frac{x}{M} + \frac{y}{N}}{2}\right) + 127.5 \right\rfloor$$

$$G = \left\lfloor 127.5 \sin\left(20\pi \frac{y}{N}\right) + 127.5 \right\rfloor$$

$$B = \left\lfloor 127.5 \sin\left(20\pi \frac{x}{M}\right) + 127.5 \right\rfloor$$

Ob upoštevanju koordinatnega sistema BMP zapisa. Sledeča slika prikazuje izgled pravilno zgenerirane slike:



Naloga 2 (2 točki)

Glede na zgoraj podano kodo za zapis BMP slik spišite skripto za branje BMP slik brez uporabe knjižnjice openCV oz. drugih podobnih knjižic, in preberite priloženo sliko parrots.bmp. Pri tem upoštevajte, da ima glava BMP datoteke lahko večjo dolžino od predstavljenega primera, je pa v vsakem primeru v glavi na podanem mestu zapisan točen indeks, kjer se začnejo vrednosti pikslov.

Prebrane vrednosti ustrezno obdelajte tako, da sliko lahko pravilno prikažete s funkcijo plt.imshow(). Vrednosti pikslov je treba torej spraviti v numpy array, ki mu je treba ustrezno spremeniti vrstni red vrstic in kanalov. Skripta deluje pravilno, če s plt.imshow(slika) brez dodatnih argumentov dobite enak izris, kot, če bmp sliko odprete v drugem programu za ogled slik, kot npr. GIMP, Chrome ali Firefox.

Pregledni opis postopka, ki ga morate udejanjiti je podan v nadaljevanju:

1. Odpri vhodno bmp datoteko v načinu za binarno branje vsebine (“rb”)
2. Preberi vsebino datoteke
3. Razčleni glavo datoteke, ki ima isti format zapisa, kot zgoraj podana koda za zapis BMP slike. Pri tem upoštevajte, da prva dva bajta na začetku datoteke predstavljata identifikator tipa datoteke, “BM”.
4. Iz glave datoteke preberi resolucijo slike ($N \times M$)
5. Inicializiraj numpy array ustrezne oblike in podatkovnega tipa za shranjevanje vrednosti pikslov
6. Iz glave datoteke preberi vrednost indeks prvega piksla
7. Preberi r, g, b vrednosti vsakega piksla posebej in jih zapiši na ustrezno mesto v numpy array

Naloga 3 (1 točka)

Kvaliteto izgubne kompresije slike tipično ovrednotimo kot razmerje med dvema veličinama:

- normirano velikostjo datoteke, ter
- normiranim nivojem popačenja slike.

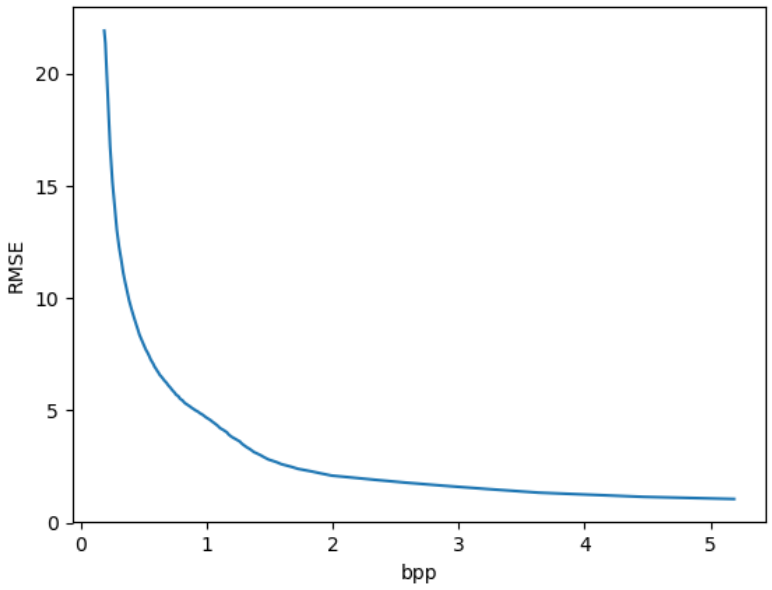
Normirano velikost datoteke določimo v enotah biti/piksel (angl. Bits per pixel, Bpp), ki je za datoteko, ki predstavlja sliko resolucije $N \times M$ določena kot:

$$bpp = \frac{8 \times (velikost\ datoteke\ v\ bajtih)}{NM}$$

Medtem, ko normirano popačenost določimo kot mero korena srednje kvadratne napake (RMSE) med originalno sliko I_{orig} in iz kompresirane datoteke dekodirano sliko, I_{dec},

$$\sqrt{\frac{1}{NM}\sum_{y=1}^{N}\sum_{x=1}^{M}\left(I_{orig}(x,y) - I_{dec}(x,y)\right)^2}$$

Z openCV naložite sliko parrots.png. Preučite dokumentacijo funkcije cv2.imwrite in ugotovite, kako shraniti sliko v zapisu JPEG z določenim nivojem kvalitete. Originalno (PNG) sliko posebej shranite v 101 različnih JPEG datotek s kvaliteto od 0 do 100 in celoštevilskim korakom med nivoji kvalitete. Nato preberite vsako od shranjenih JPEG datotek ter izračunajte njen bpp ter RMSE glede na originalno (PNG) sliko. Velikost datoteke v bajtih lahko dobite preko funkcije os.path.getsize. Razmerje med veličinama prikažite na grafu z bpp na X osi in RMSE na Y osi. Pričakovan izris je podoben sledečemu:



Naloga 4 (1 točka)

Pri vrednotenju algoritmov za brezizgubno kompresijo ne moremo govoriti o popačenju slik, ker le-tega ni, je pa pomenljiva veličina čas, potreben za kodiranje oz. dekodiranje datoteke. Preko te veličine ovrednotite nivoje kompresije PNG. To storite tako, da z openCV naložite večjo sliko IMG_2303.png. Večjo sliko tu uporabljamo zato, ker ima pythonova sistemska ura premalo časovno ločljivost za natančno merjenje časa dekodiranja pri manjših slikah. Preučite dokumentacijo funkcije cv2.imwrite in ugotovite, kako pri shranjevanju slike v zapisu PNG določiti nivo kompresije. Originalno sliko posebej shranite z nivoji kompresije od 0 do 9. Nato preberite vsako od shranjenih slik. Pri tem izračunajte njen bpp ter izmerite čas, potreben za dekodiranje.

Čas dekodiranja lahko izmerite s sledečo kodo:

```
t0 = time.time()
x = cv2.imread(fn)
t1 = time.time()
cas = t1 - t0
```

Razmerje med mero bpp in časom dekodiranja predstavite v obliki grafa, ki bi ob pravilnem izračunu moral izgledati podoben sledečemu: