СЕМИНАРСКА РАБОТА ПО ДИГИТАЛНО ПРОЦЕСИРАЊЕ НА СЛИКА

Тема: Компресија на слики

Факултет за информатички науки и компјутерско инженерство

Ментор:

проф. д-р Ивица Димитровски

Изработил:

Никола Јорданоски

Скопје, Јуни 2024

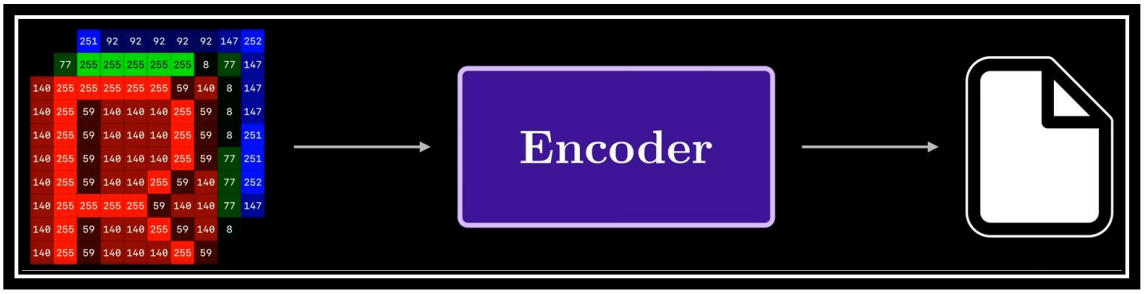
Содржина

Вовед

Со напредокот на технологијата, количината на податоци и информации продолжува да расте експоненцијално, што доведува до потреба од поголем простор за складирање и ефикасен начин на транспорт. Во овој момент техниките за компресија стапуваат на сцена. **Компресијата на слики**, поткатегорија на компресијата на податоци, специјално се насочува кон намалување на големината на дигиталните слики без значително нарушување на нивниот квалитет.

Концептот на компресија на слики датира од 1970-тите години, со значајни пресвртници како воведувањето на *JPEG* форматот на почетокот на 1990-тите. Со текот на годините, развиени се различни алгоритми и техники за постигнување поефикасна компресија.

Овој семинарски труд ќе ги разгледа 2-та основни типа на компресија, со и без загуба, како и различните алгоритми за компресија на слики. Ќе ги дискутираме предностите и недостатоците на овие техники, споредувајќи го нивното влијание врз квалитетот на сликите и нивната ефикасност. На крајот, ќе биде прикажана кратка демо апликација која користи некои од овие алгоритми за компресија.



Слика 1. Процес на компресија на слика

Компресија

Компресирањето на слики е процесот на намалување на големината на датотеката на сликата без значително влошување на нејзиниот квалитет. Ова се постигнува со отстранување на непотребните бајти или користење на алгоритми за поефикасно "реуредување" и презапишување на сликата.

Постигнување на најдобар квалитет на слика со добра ***стапка на компресија*** (просторот потребен да се зачува сликата) е клучно, но неколку други својства се исто така важни:

1. *Scalability (Скалирање):* Ова значи дека можете да го промените квалитетот на сликата без да треба целосно да ја декомпресирате и повторно компресирате. На пример корисно е за preview на слики за време на download.
2. *Region of Interest Coding* (Кодирање на регионот на интерес): Ова овозможува на одредени поважни делови од сликата да се зачува во квалитетот од помалку важните делови. На пример, на фотографија од човек, лицето може да се кодира во повисок квалитет отколку позадината.
3. *Meta Information (Мета информации):* Ова се дополнителни податоци зачувани заедно со сликата, како preview слики, детали за боја и текстура, или информации за авторот и авторските права на сликата. Помага при организирање, пребарување и прегледување на слики без целосно вчитување на истите.
4. *Processing Power (Процесорска моќ):* Различните техники на компресија бараат различни количини на процесирачка за да работат. Некои методи компресираат слики поефикасно, но бараат повеќе процесорска моќ, додека други се побрзи но помалку ефикасни.

Квалитетот на компресијата често се мери со **peak signal-to-noise ratio (PSNR)**. Ова е начин да се мери колку детали се изгубени во компресираната слика. Тоа е како да се споредува оригиналната слика со компресираната и да се види колку "шум" е додаден. Поголем PSNR значи дека сликата е поблиску до оригиналот и има подобар квалитет. Сепак, на крајот, како изгледа сликата, зависи од личната преференца.

Постојат 2 типа на компресија:

- **Компресија со загуба** (се користи кај web слкики, на социјални медиуми, e-mail attachments и др.)
- **Компресија без загуба** (се користи во медицината, архитектурата кај технички цртежи, графички дизајн и др.)

Компресија со загуба

Компресијата со загуба ја намалува големината на датотеката со трајно отстранување на помалку критичните, непотребни пиксели. Додека ова значително ја намалува големината на датотеката, прекумерното компресирање може да доведе до забележливо изобличување на сликата. Сепак, со внимателна примена, квалитетот може да се задржи.

Главна карактеристика на овој тип компресија е нејзината неповратност, оригиналната слика не може целосно да се врати, а повтореното компресирање го зголемува изобличувањето. И покрај тоа, компресијата со загуба е вредна за користење, особено на веб, каде што умереното изобличување на сликата е често дозволено.

Чест пример за компресија со загуба е **JPEG** форматот, како и други формати: **WebP**, **HEIF**, **JPEG 2000**.[1]

Слика 2. Оригинална верзија (9.5 MB)

Слика 3. Компресирана верзија (500 KB)

На прв поглед изгледа дека нема промена во квалитетот, но ако се погледне повнимателно може да се забележи пикселизирање кај засенчувањето.

[1] WebP, HEIF, JPEG 2000 – користат и компресија со загуба и компресија без загуба

Компресија без загуба

Компресијата без загуба ја намалува големината на датотеката без да отстрани никакви информации, обезбедувајќи сликата да изгледа скоро идентично со оригиналот, користејќи алгоритми за "реуредување" на сликата. Иако големината може да ја намали за голем процент, сепак е помалку ефикасна од компресијата со загуба. Идеална е за одржување на висок квалитет на сликата, што ја прави погодна за слики на производи и уметнички дела, каде што деталите се клучни.

Чест пример за компресија без загуба е **PNG** форматот, како и други формати: **GIF**, **BMP**, **RAW**, **WebP**, **HEIF**, **JPEG 2000**.[2]

Исто така, важно е дека овој вид на компресија овозможува сликата да се врати назад во својата првобитна состојба.

Слика 4. Оригинална верзија (8.29 MB)

Слика 5. Компресирана верзија (3.81 MB)

На прв поглед изгелда дека нема промена во квалиетот, но ако се погледне повнимателно може да се забележи дека започнало минимално пикселизирање низ цела слика.

[2] WebP, HEIF, JPEG 2000 – користат и компресија со загуба и компресија без загуба

Алгоритми за компресија

Една дигитална слика е составена од ситни точки наречени **пиксели**. Секој пиксел има вредност која се движи од 0 до 255. Kaj grayscale сликите, вредноста на секој пиксел одговара на сива нијанса, при што 0 е црно, а 255 е бело. Во сликите во боја, пикселите имаат вредности за црвена, зелена и сина (RGB), со што комбинацијата на овие вредности овозможува голем опсег на бои.

Алгоритам е збир на правила што треба да ги следи компјутерот. Двата методи на компресија со загуби и без загуби користат различни алгоритми за компресија за да постигнат помали големини на датотеки со управување на пикселите од сликата. Во суштина, целта на овие алгоритми е да ја намалат големината на сликата без да го жртвуваат нејзиниот квалитет. Пример алгоритми *transform coding, run-length encoding, arithmetic coding, LZW, flate/deflate, Huffman coding* и др.

PNG, JPEG и WebP и други, се формати на слики кои користат различни комбинации од овие алгоритми за ефикасно претставување и чување на сликата, според нивните соодветни стандарди.

Со разбирање на пикселите и разликата помеѓу компресија со загуби и без загуби, можеме да го избереме вистинскиот формат и алгоритми што ќе ги користиме врз основа на нашите потреби.

Алгоритми за компресија со загуба

Chroma Subsampling - ја намалува резолуцијата на информациите за хроминација (пример Cb, Cr) на сликата. Ја користи помалата чувствителност на човечкото око на боја во споредба со чувствителноста на осветленоста.

Процес:

1. Конвертирајте ја сликата од RGB во YCbCr, одвојувајќи ја осветленоста (Y) од хроминацијата (Cb и Cr).
2. Subsample на каналите Cb и Cr со аритметичка средина или испуштање на некои информации за бојата.

Вообичаени соодноси:

- 44:4:4: Нема subsampling, целосна резолуција за Y, Cb и Cr.

```
Y:  | Y0 | Y1 | Y2 | Y3 |
Cb: | Cb0 | Cb1 | Cb2 | Cb3 |
Cr: | Cr0 | Cr1 | Cr2 | Cr3 |
```

- 4:2:2: Cb и Cr се преполовени во хоризонтална насока.

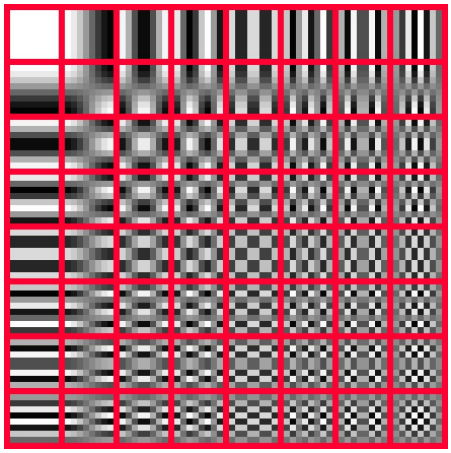
```
Y:  | Y0 | Y1 | Y2 | Y3 |
Cb: | Cb0 | Cb0 | Cb2 | Cb2 |
Cr: | Cr0 | Cr0 | Cr2 | Cr2 |
```

- 4:2:0: Cb и Cr се преполовени и во хоризонтална и во вертикална насока.

```
Y:  | Y0 | Y1 | Y2 | Y3 |
Cb: | Cb0 | Cb0 | Cb0 | Cb0 |
Cr: | Cr0 | Cr0 | Cr0 | Cr0 |
```

Transform Coding - најчесто користениот метод. Има 2 познати алгоритми **Discrete Cosine Transform (DCT)** и **Wavelet Transform** (на кој нема толку да се задржуваме).

Discrete Cosine Transform (DCT) - е широко користена трансформација во компресија на слики. Тој изразува низа од податочни точки како збир на косинусни функции кои осцилираат на различни фреквенции.



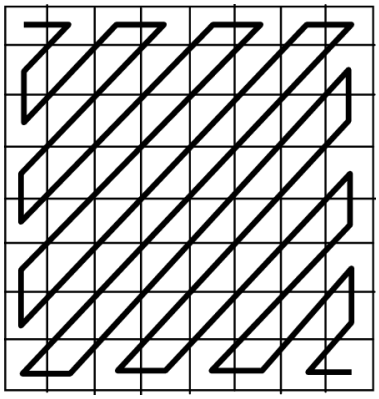
Слика 6. Матрица од 8x8 блокови од по 8x8 пикесли која се користи при DCT

Во JPEG, сликата се претвора од RGB во YCbCr. Ова ја дели сликата на една компонента на осветленост (Y) и две компоненти на хроминација (Cb и Cr). Потоа се користи Chroma subsample. Сликата потоа е поделена на **8x8 блокови** и секој блок се трансформира со помош на DCT. Ова ги конвертира податоците во доменот на фреквенција.

Секој блок 8x8 се обработува за да се произведе сет од 64 коефициенти на DCT.

Повеќето од значајните информации за сликата се концентрирани во ниските фреквенции (горниот лев агол на DCT матрицата), овозможувајќи повисоките фреквенции (долниот лев агол) да се отфрлат или поагресивно да се квантизираат. Коефициентите на повисоки фреквенции често се поделени со поголеми вредности, што ги прави поверојатно да се заокружат на нула. Овој чекор значително ја намалува прецизноста на многу коефициенти, особено оние што претставуваат помалку визуелно значајни детали. Коефициентите се наведени потоа во цик-цак шема. После тоа, ентропиското кодирање (обично со Huffman coding (Хуфманово кодирање) и RLE кодирање) стапува во игра кое е дел од алгоритмите без загуби.

DCT сам по себе не ги компресира сликите, но помага со другите алгоритми да се компресира поефикасно со фокусирање на компонентите на фреквенцијата, каде што човечкиот вид е помалку чувствителен на детали со висока фреквенција.



Слика 7. Цик-цак шема на листање

Wavelet Transform - разложува сигнал на мали бранови локализирани во времето, обезбедувајќи подобра компресија и квалитет на сликата. *JPEG 2000* ја користи оваа техника наместо DCT. Овој метод ги зачувува компонентите со пониска фреквенција со поголема прецизност, ефикасно компресирајќи ги повисоките фреквенции, што го прави супериорен за ракување со фини детали и остри рабови.

Квантизација во JPEG - по трансформацијата на DCT, 64-те коефициенти се делат со матрица за квантизација и се заокружуваат до најблискиот цел број.

Матрицата за квантизација одредува колку е компресирана секоја фреквентна компонента. Пониските фреквенции се зачувани попрецизно, додека повисоките фреквенции се компресирани поагресивно (обично целта е да се заокружи на 0).

Правилно избраната квантизација го минимизира забележливото губење на квалитетот, бидејќи човечкото око е помалку чувствително на детали со висока фреквенција.

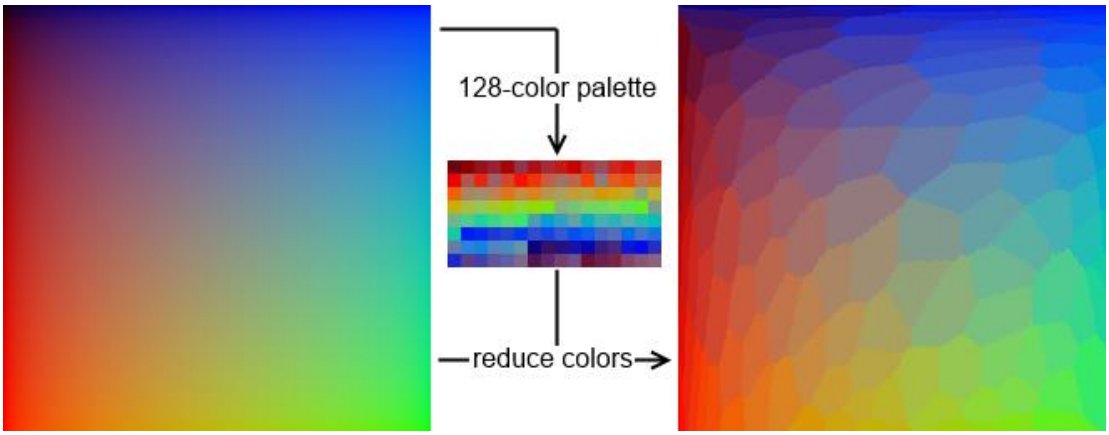
Квантизација на бои - го намалува бројот на различни бои на сликата, што го олеснува компресирањето.

- Палетата за цела слика користи фиксен сет на бои (на пр., 256 бои) за целата слика, при што секој пиксел се заменува со боја од оваа оваа палета.
- Блок палетата користи помал сет на бои за секој блок од сликата (на пр., 2 или 4 бои за секој блок од 4x4 пиксели).

Секој пиксел во оригиналната слика се споредува со боите во палетата за да се најде најблиското совпаѓање. Оваа споредба обично се прави со користење на метрика како Евклидово растојание.

Форматите како GIF и PNG го користат ова за да ја ограничат палетата на бои и да постигнат помали големини на датотеки.

Може да го користи алгоритмот *K-means* за изборот на бои.



Слика 8. Пример за Color Quantization

Векторска квантизација - ги компресира податоците за сликата преку претставување блокови од пиксели со помал сет на репрезентативни блокови.

Процес:

1. Сликата е поделена на блокови од пиксели кои не се преклопуваат.
2. Секој блок се споредува со шифрарник на репрезентативни блокови (вектори на кодови).
3. Секој блок е кодиран со индексот на најблискиот соодветен вектор на кодови во шифрата.

Книгата со шифри се генерира со користење на алгоритми како *K-Means* за групирање слични блокови заедно.

Предностите се тоа што постигнува високи коефициенти на компресија со намалување на вишокот слични блокови. Ограничувањата се дека квалитетот зависи од големината и точноста на шифрарникот. Лош шифрарник може да резултира со забележителни артефакти.

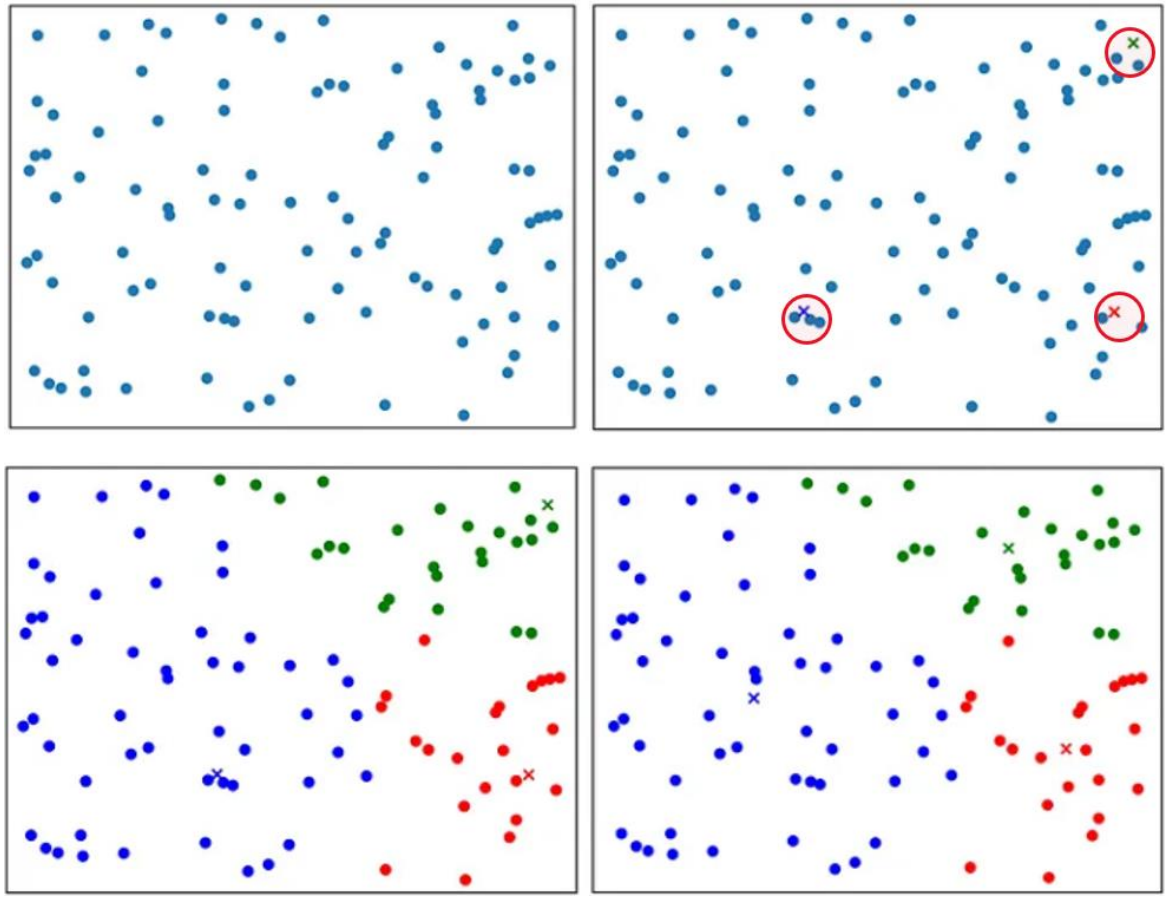
K-Means - е алгоритам за кластерирање кој се користи за партиционирање на точки во k кластери, каде што секоја точка припаѓа на кластерот со најблиската средина.

Процес:

1. По случаен избор изберете k иницијални центри на кластери (средства).
2. Доделете ја секоја податочна точка до најблискиот центар за кластер врз основа на Евклидов растојание.
3. Повторно пресметајте ги центрите за кластери како средна вредност од точките на податоци доделени на секој кластер.
4. Повторете ја задачата и ажурирајте ги чекорите додека не се спојат центрите за кластери.

Често се користи во квантизација на бои за да се намали бројот на различни бои на сликата.

Предностите се што ефикасно ја намалува палетата на бои на сликата, што го олеснува компресирањето, но изборот на k и почетните центри на кластерот може да влијае на конечниот резултат, слабата иницијализација може да доведе до неоптимални кластери.



Слика 9. Процесот на K-Means

Алгоритми за компресија без загуба

Run-length Encoding (RLE) - е едноставна форма на компресија на податоци што ги заменува секвенците од пиксели со парот (пиксел , број на повторувања).

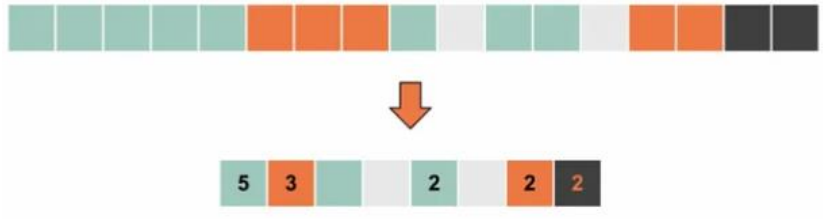
Процес:

1. Откриј секвенца каде се повторува истиот пиксел.
2. Заменете ја секоја секвенца со пар (пиксел , број на повторувања).

На пример, низата „AAAABBBCCDAA“ станува „4A3B2C1D2A“.

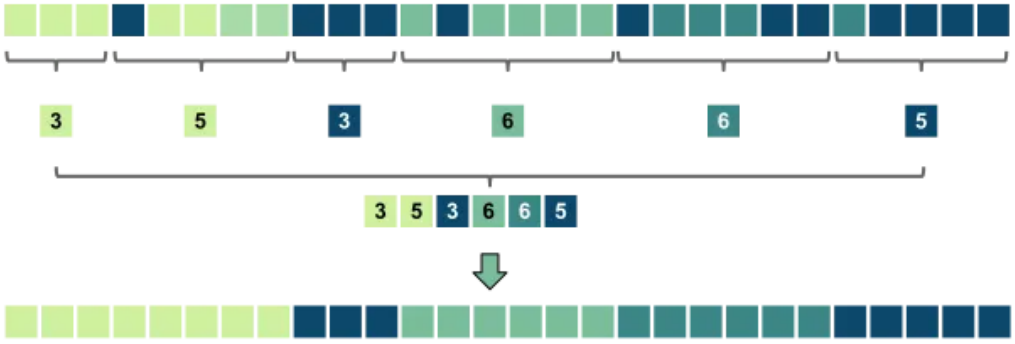
Лесно е за имплементација и разбирање и е ефикасно за податоци со многу повторувани знаци, како што се сликите. Но, тоа има свои ограничувања. Не е погоден за податоци со малку повторувања, бидејќи тоа ќе го направи спротивното, односно ќе зголеми големината на датотеката.

Постои RLE со **загуби** и **без загуби**.



Слика 10. RLE без загуба

Во **RLE со загуби**, можеме да ја подобриме компресијата со игнорирање на многу кратки секвенци. На пример, ако најкратката секвенца која ќе ја компресираме е 3, тогаш секвенците од 2 последователни пиксели ќе бидат прескокнати.



Слика 11. RLE со загуба

Исто така, постои решение за кога ќе се наиде на повеќекратни кратки секвенци. Тие се мешаат во една боја. Пристапот за спојување може да варира:

- Аритметичка средина на боја - избор на средна боја на кратките патеки.
- Личен избор - изборот може да зависи од специфичната слика и саканиот квалитет на компресија.

Huffman Coding (Хуфманово кодирање) - е популарен метод за ентропско кодирање кој доделува битови на податоците врз основа на нивните фреквенции, при што почестите симболи добиваат пократки кодови.

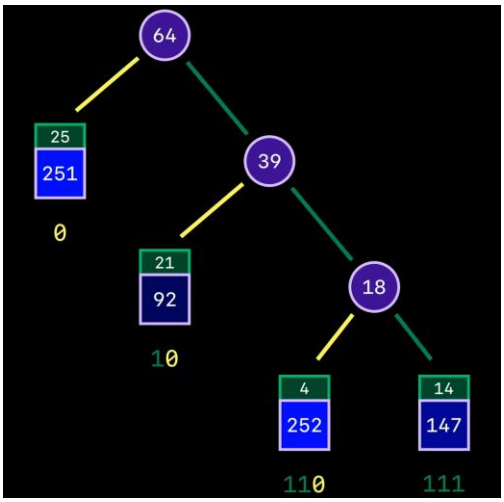
Процес:

1. Пресметајте ја фреквенцијата на секој пиксел на сликата.
2. Изградете бинарно дрво, каде што секој лист претставува пиксел, а патеката од коренот до листот го претставува кодот за пиксели. За левите гранки се доделува 0, а за десните 1.

Горе на дрвото се наоѓаат најфреквентите битови

Иако е ефикасен за компресирање на податоци со различни фреквенции на симболи, минимизирајќи ја просечната должина на кодот, недостатокот е што бара познавање на фреквенциите на симболите и градење на дрвото, што може да биде пресметковно интензивно.

Широко се користи во апликации за компресија на податоци како *DEFLATE, JPEG* и други формати.



Слика 12. Бинарно дрво на Хуфманово кодирање

Arithmetic Coding (Аритметичко кодирање) - е техника за ентропско кодирање која шифрира цела порака во еден број, обично дропка, со помош на користење на интервали. Нема да се фокусираме на ова многу бидејќи повеќе не се користи често, поради проблеми со авторските права.

Знае да е поефикасно од Хуфман кодирањето.

DEFLATE - е широко користен метод за компресија без загуби кој комбинира кодирање **LZ77/LZW** и **Huffman**. Прво, влезните податоци се компресирани со помош на LZ77/LZW, кој наоѓа повторувачки секвенци и ги заменува со референци за претходни појави. Потоа, излезот од чекорот LZ77/LZW дополнително се компресира со користење на Хуфман кодирањето за да се оптимизира должината на битот.

PNG го користи овој метод на компресија, но само откако ќе ја филтрира сликата (што е сосема друга тема за разговор).

LZ77, LZ78, LZW, LZSS сите припаѓаат на семејството Abraham Lempel и Jacob Ziv.

LZ77 - воведен од Abraham Lempel и Jacob Ziv во 1977 година, LZ77 користи лизгачки прозорец за проаѓање низ секвенците од податоци. Пронаоѓа совпаѓања во прозорецот и ги кодира како (растојание, должина) парови.

LZ78 - воведен од Lempel и Ziv во 1978 година како подобрување во однос на LZ77, LZ78 динамично гради мапа. Секоја нова фраза се додава во мапата и се кодира со помош на индекси во речник.

LZW (Lempel-Ziv-Welch) - развиен од Tery Welch во 1984 година, LZW е подобрување на LZ78. Иницијализира мапа со сите можни низи од еден знак, ги чита влезните низи и ги заменува со индекси на речник, динамично додавајќи нови секвенци. LZW се користи во формати како GIF.

LZSS (Lempel-Ziv-Storer-Szymanski) - варијација на LZ77 воведена од James Storer и Thomas Szymanski во 1982 година.

Процес:

1. Како LZ77, користи лизгачки прозорец за да најде повторени секвенци.
2. Користи бит за flag за да прави разлика помеѓу: дата (буквален податок) и референци за совпаѓање. Ако се најде совпаѓање, тоа се шифрира како (растојание, должина), во спротивно, буквален бајт е кодиран.

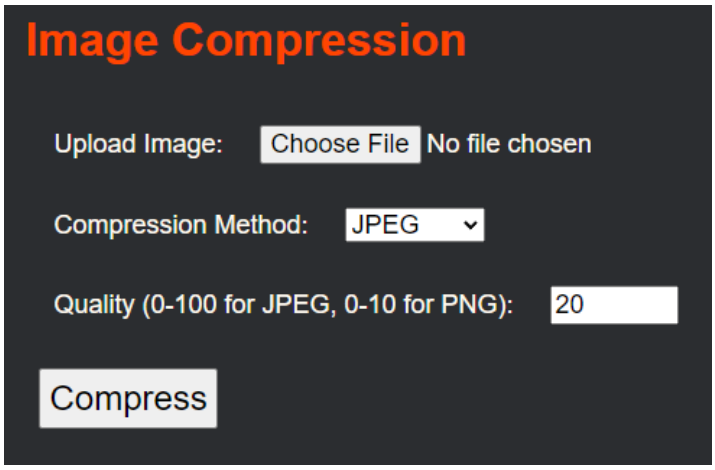
Голем недостаток е тоа што ако лизгачкиот прозорец во LZSS е премногу мал или не фаќа доволно податоци кои се повторуваат, ефикасноста на компресија страда. Ова е затоа што LZSS се потпира на наоѓање повторени секвенци во прозорецот за ефективна компресија.

Имплементација на едноставни алгоритми во демо апликација

Во продолжение ќе разгледаме една едноставна демо апликација која овозможува компресирање на сликите со JPEG, PNG методите за компресија, а ќе може и да се види и исходот од алгоритмите K-means и RLE.

Тек на извршување:

1. Избор и прикачување на слика за компресија. (задолжително)
2. Избор на формат или алгоритам за компресија. (доколку корисникот не избере, има default вредност)
3. Избор на јачината на компресија. (доколку корисникот не избере, има default вредност) Тука важно е да се напомене дека, за JPEG што е поголема вредноста на квалитетот тоа е помала компресијата, односно 100 значи дека квалитетот е целосно задржан, а 0 дека компресијата е најсилна. Кај PNG, вредностите се движат од 0 до 10. 10 значи најсилна компресија додека 0 дека оригиналната слика е задржана со ист квалитет. Односно JPEG и PNG се обратно пропорционални.
4. По притискање на копчето Compress, компресираната слика се превзема кај корисникот.



Слика 13. Изглед на демо апликација

Технологии кои се користени:

- **Python** (Back-end)
- **OpenCV** (библиотека за манипулација со слики)
- **Numpy** (библиотека за работа со низи и матрици)
- **Flask** (Front-end)

Со помош на OpenCV многу е лесно да се импелементира, JPEG компресијата е многу лесна за имплементирање. imencode функцијата на cv2, прима 4 аргументи: **екстензија** (во нашиот случај е .jpg), **image** е сликата која сакаме да ја компресираме и на крај се **параметрирте за која компресија да ја правиме и со кој интензитет**.

```
is_success, buffer = cv2.imencode( ext: ".jpg", image, params: [cv2.IMWRITE_JPEG_QUALITY, quality])
```

Слика 14. Имплементација на JPEG компресија во Python

Како и кај JPEG, и кај PNG имплементацијата е многу едноставна. imencode функцијата на cv2, прима 4 аргументи: **екстензија** (во нашиот случај е .png), **image** е сликата која сакаме да ја компресираме и на крај се **параметрирте за која компресија да ја правиме и со кој интензитет.**

```
is_success, buffer = cv2.imencode( ext: ".png", image, params: [cv2.IMWRITE_PNG_COMPRESSION, quality])
```

Слика 15. Имплементација на PNG компресија во Python

Функцијата **rle_encode** врши шифрирање со RLE алгоритмот на grayscale слика.

Започнува со израмнување на сликата во една димензионална низа од пиксели. Иницијализира променливи за да се следи претходната вредност на пиксели (**prev_pixel**) и бројач (**count**) за последователни појавувања на пиксели. Почнува од вториот пиксел циклусот. Ако тековниот пиксел се совпаѓа со претходниот (prev_pixel), тој го зголемува бројачот. Ако се разликува, го зачувува **tuple**-от (prev_pixel, count) во низата rle потоа го ажурира prev_pixel и го ресетира бројачот на 1.

По завршувањето на циклусот, го додава последниот tuple на rle.

```
def rle_encode(image):
    pixels = image.flatten()
    rle = []
    prev_pixel = pixels[0]
    count = 1
    for pixel in pixels[1:]:
        if pixel == prev_pixel:
            count += 1
        else:
            rle.append((prev_pixel, count))
            prev_pixel = pixel
            count = 1
    rle.append((prev_pixel, count))
    return rle
```

Слика 17. Имплементација на JPEG компресија во Python

1. **image.shape:** Ги враќа димензиите на влезната слика, кои се клучни за преобликување на сликата по компресија.
2. **image.reshape((-1, 3)):** Ја израмнува сликата во 2D низа каде што секој ред претставува пиксел со три канали (вредности RGB/BGR).
3. **np.float32(pixel_values):** Ги конвертира вредностите на пикселите во формат float32, неопходни за обработка со алгоритмот k-means.
4. **cv2.kmeans(pixel_values, k, None, criteria, 10, cv2.KMEANS_RANDOM_CENTERS):** Го применува алгоритмот за групирање k-means на вредностите на пиксели, групирајќи слични пиксели во k кластери. Алгоритмот се повторува до 10 пати (10 обиди) за да се најдат оптималните кластери врз основа на дадените критериуми за завршување (критериуми).
5. **centers = np.uint8(centers):** Ги конвертира центрите на кластерите (пресметани како float32) назад во формат uint8, што одговара на оригиналните вредности на пиксели на сликата.
6. **centers[labels.flatten()]:** Го мапира секој пиксел до најблискиот центар на кластерот врз основа на ознаките доделени со k-means, создавајќи ја компресираната слика.
7. **compressed_image.reshape(original_shape):** Ја преобликува компресираната слика назад во нејзините оригинални димензии, обезбедувајќи компатибилност за прикажување или понатамошна обработка.

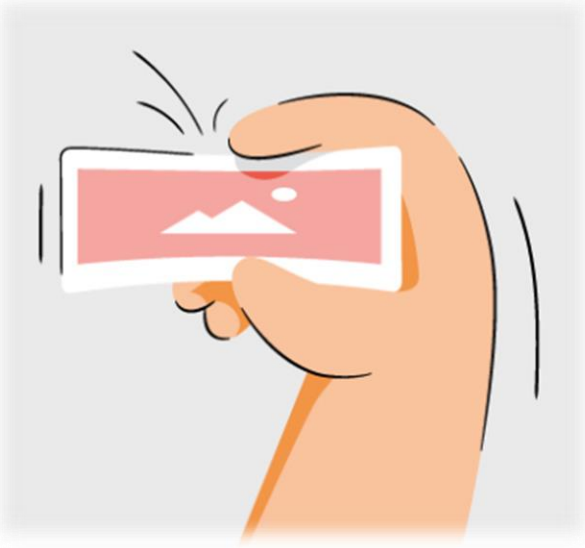
```python
def compress_image_with_kmeans(image, k=8):
    original_shape = image.shape
    pixel_values = image.reshape((-1, 3))
    pixel_values = np.float32(pixel_values)
    criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 100, 0.2)
    _, labels, centers = cv2.kmeans(pixel_values, k, bestLabels: None, criteria, attempts: 10, cv2.KMEANS_RANDOM_CENTERS)
    centers = np.uint8(centers)
    compressed_image = centers[labels.flatten()]
    compressed_image = compressed_image.reshape(original_shape)
    return compressed_image
```

Слика 16. Имплементација на JPEG компресија во Python

Заклучок

Компресијата на слики игра клучна улога во современиот дигитален свет, обезбедувајќи ефективно управување со големината на датотеките, во нашиот случај сликите. Алгоритмите за компресија се делат на два главни типа: **со загуби** и **без загуби**, иако може да има варијации и исклучоци. Алгоритмите без загуби, како што се Huffman coding и LZSS, овозможуваат компресија на податоците без да се губи оригиналната информација, што ги прави идеални за примените каде што е потребна точност, како што се медицинските и научните слики. Од друга страна, алгоритмите со загуби, како што се користат кај JPEG, значително ја намалуваат големината на сликите со одредено намалување на квалитетот, што ги прави погодни за секојдневна употреба, како што се веб-страниците и социјалните мрежи. Со напредокот на технологијата, овие алгоритми постојано се подобруваат, обезбедувајќи поголема ефикасност и подобар квалитет на компресираните слики, овозможувајќи полесно складирање и трансмисија на дигиталните слики. Се појавуваат и нови алгоритми и методи, како компресија заснована на длабоко учење која користи невронски мрежи за предвидување и кодирање на слики.

Со демо апликацијата на крајот можевме да го иведеме исходот од користењето на некои од овие алгоритми и како тие манипулираат сликата.



Слика 18. Карикатура “Компресија на слика“

Референци

- *https://en.wikipedia.org/wiki/Image_compression*
- *https://www.techtarget.com/whatis/definition/image-compression*
- *https://tiny-img.com/blog/image-compression/*
- *https://www.adobe.com/uk/creativecloud/photography/discover/lossy-vs-lossless.html#s1*
- *https://www.adobe.com/uk/creativecloud/photography/discover/lossy-compression.html#s4*
- *https://www.adobe.com/uk/creativecloud/photography/discover/lossless-compression.html*
- *https://www.cloudflare.com/learning/performance/glossary/what-is-image-compression/*
- *https://cloudinary.com/glossary/image-compression-algorithms*
- *https://towardsdatascience.com/clear-and-visual-explanation-of-the-k-means-algorithm-applied-to-image-compression-b7fdc547e410*
- *https://medium.com/@ayush.pegasus2801/image-compression-f3c5f5320630*