Klasifikacija slika tenzorskim metodama

Jandrijević, Luka

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Luka Jandrijević

KLASIFIKACIJA SLIKA TENZORSKIM METODAMA

Diplomski rad

Voditelj rada:
prof. dr. sc. Zlatko Drmač

Zagreb, rujan 2023.

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Sadržaj

Uvod

U svijetu koji je sve više orijentiran prema digitalnim medijima, količina vizualnih podataka koja se svakodnevno generira je ogromna. To donosi nove izazove, ali i prilike za napredak u polju računalne vizije, posebno u kontekstu klasifikacije slika. Klasifikacija slika je složen zadatak koji zahtijeva sofisticirane metode kako bi se postigla visoka točnost i efikasnost. Među različitim pristupima, metode temeljene na tenzorima pokazuju obećavajuće rezultate. Slike prirodno zamišljamo u obliku matrice, no ako želimo video, to jest niz slika, prikazati na sličan način, potrebna nam je nešto općenitije od matrica. Tenzori, kao općenitija verzija vektora i matrica, omogućuju bolje korištenje višedimenzionalnih svojstava slika. Pomoću njih lako možemo reprezentirati razne višedimenzionalne stvari te koristeći operacije nad njima, lako i manipuliramo s takvim podatcima.

U poglavlju 1 ćemo nešto više reći o tenzorima općenito te navesti neke od operacija koje možemo provoditi nad njima. Zatim, u poglavlju 2 opisujemo dekompoziciju matrice poznatu kao SVD te ju primjenjujemo za kompresiju crno-bijelih slika, ali i slika u boji. Kasnije, u poglavlju 3 će biti opisane dvije vrste tenzorskih dekompozicija, SVD višeg reda (HOSVD) te *tensor train* dekompozicija u kojem također dajemo primjer kako kompresirati slike te pomoću toga klasificirati slike. Na kraju, u poglavlju 4, bavimo se problemom klasifikacije rukom pisanih znamenaka te dajemo rezultate koje smo dobili klasificirajući set podataka. Problem riješavamo koristeći dva pristupa. U prvom pristupu primjenjujemo HOSVD, a u drugom pristupu primjenjujemo *tensor train* dekompoziciju kako bi raspoznali o kojoj znamenci se radi na slici. Baza podataka slika koji koristimo se naziva MNIST baza podataka koja sadrži sveukupno 70000 raznovrsnih slika rukom pisanih znamenaka. Takvu bazu podataka je prvo potrebno razdvojiti na bazu za treniranje i bazu za testiranje te zatim primijeniti spomenute algoritme kako bi dostigli što veću točnost.

U radu se koristi programski jezik Python koji ima raznovrsnih pomoćnih funkcija koje možemo iskoristiti kao pomoć u klasifikaciji te je njegovo znanje poželjno ali ne i nužno za shvaćanje ovog rada.

Poglavlje 1

Tenzori

U ovom poglavlju ćemo opisati neke osnovne stvari o tenzorima i uvesti notaciju kojom ćemo se služiti kroz čitavi rad. Kao i matrice, tenzori su još jedan matematički objekt pomoću kojih opisujemo fizičke pojave i oni nam olakšavaju opisivanje raznih svojstava nad tim pojavama. Tenzore možemo shvatiti kao višedimenzionalne nizove, odnosno, tenzori generaliziraju koncept matrica i vektora u više dimenzija. Zbog toga, poželjno je dobro znanje iz područja linearne algebre kao što su baze vektorskih prostora ali i općenito operacije s matricama kako bi se rad razumio što bolje. Svrhu tenzora možemo bolje razumjeti na sljedećem primjeru. Zamislimo slike kao matrice ili tablice, gdje svaka ćelija sadrži broj koji označava intenzitet boje za svaki piksel na ekranu. No video na drugu ruku ne možemo tako lako prikazati matricom - trebamo više dimenzija. Video možemo prikazati kroz više slika, gdje svaka slika predstavlja trenutak u videu. Kada te slike postavimo u niz, dobivamo niz matrica. Drugim riječima, dobivamo nešto slično kao na slici 1.2, gdje su prikazani frontalni odsječci tenzora. Ako želimo zamisliti tenzore s još više dimenzija, možemo razmisliti o nizu video zapisa, što bi rezultiralo tenzorom reda 4.

Da bi lakše razlikovali o kojem matematičkom objektu je riječ, skalare ćemo označavati malim tiskanim slovom ($a, b, \dots$), vektore ćemo označavati malim podebljanim slovom ($\mathbf{a}, \mathbf{b}, \dots$), realne matrice označavamo podebljanim velikim slovima ($\mathbf{A}, \mathbf{B}, \dots$), dok tenzore označavamo velikim pisanim slovima ($\mathcal{A}, \mathcal{B}, \dots$). Red tenzora označava dimenziju tenzora, tako je naprimjer tenzor $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{2\times3\times5}$, tenzor reda 3. Iz ovoga slijedi da je skalar tenzor reda 0, vektor je tenzor reda 1, dok je matrica tenzor reda 2. Tenzori koji su reda 3 ili više nazivamo tenzori višeg reda, ali ovdje ćemo se najviše fokusirati na tenzore koji su upravo reda 3 za bolje shvaćanje nekih pojmova i lakšu vizualizaciju. Slično kao i kod matrica, elemente tenzora $\mathcal{A} \in \mathbb{R}^{I_1\times I_2\cdots\times I_N}$ ćemo označavati s $a_{i_1,i_2,\dots,i_N}$ ili ponekad $\mathcal{A}(i_1, \dots, i_N)$ za lakšu asocijaciju u programiranju pri čemu je $1 \leq i_n \leq I_n, n = 1, 2, \dots, N$.

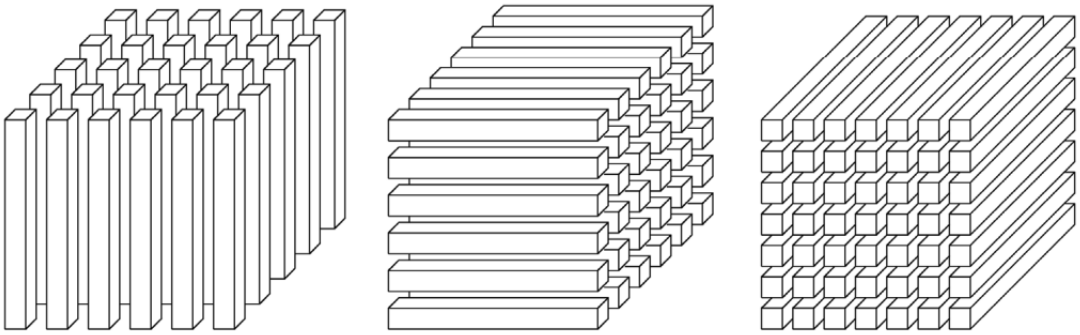
Neke od operacija koje ćemo primjenjivati u radu s tenzorima su već poznate od prije kao što su skalarni produkt ili norma i te operacije se analogno mogu definirati nad ten-

zorima. Ostale operacije, poput matricizacije tenzora, bit će definirane kasnije u ovom poglavlju.

1.1 Osnovni koncepti

U svrhu razumijevanja operacija nad matricama, često smo koristili pojmove redova i stupaca. Međutim, s obzirom na to da su tenzori višedimenzionalni objekti, takve definicije nisu primjenjive. Stoga, kako bismo mogli raditi s tenzorima, uvodimo koncept moda. Mod kod tenzora se odnosi na fiksiranje jednog od indeksa tenzora. Analogno možemo definirati vektor u modu n, odnosno nit u modu n, tako da fiksiramo sve indekse osim n-tog od nekog tenzora $\mathcal{A} \in \mathbb{R}^{I_1 \times I_2 \cdots \times I_N}$. Tako možemo reći da su stupci matrice ustvari isto što i nit u modu 1, dok su retci isto što i nit u modu 2. Ako imamo tenzor $\mathcal{A}$ reda 3 tada ćemo nit u modu 1, 2 i 3 označavati s $a_{:jk}$, $a_{i:k}$ i $a_{ij:}$ respektivno.

Slično, možemo definirati odsječke kao dvodimenzionalne presjeke tenzora tako da fiksiramo sve osim dva indeksa. Opet, ako imamo tenzor $\mathcal{A}$ reda 3, njegove odsječke[1] ćemo označavati s $\mathbf{A}(i, :, :)$, $\mathbf{A}(:, j, :)$ i $\mathbf{A}(:, :, k)$. Primjere takvih vrsta podtenzora možemo vidjeti na slikama 1.1 i 1.2.

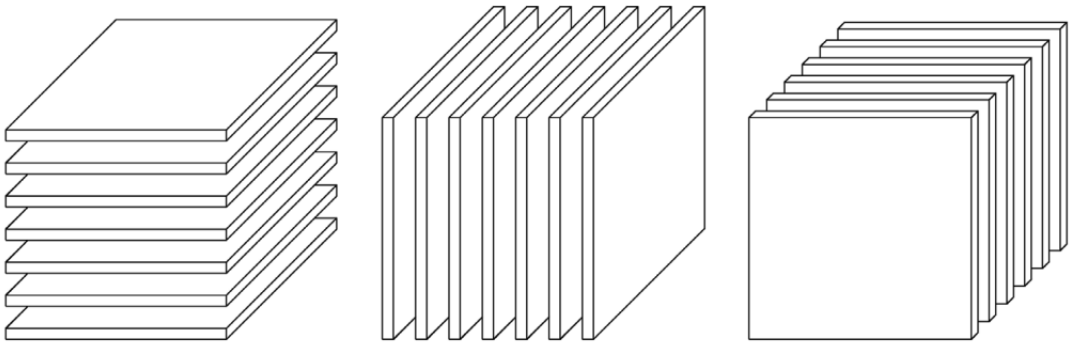


Slika 1.1: Nit u modu 1, 2 i 3 tenzora reda 3

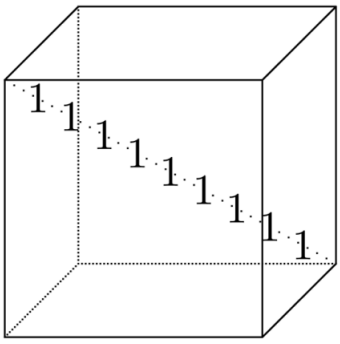
Definicija 1.1.1. *Za tenzor $\mathcal{A} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ kažemo da je dijagonalan ako $a_{i_1 i_2 \ldots i_N} \neq 0$ samo ako $i_1 = i_2 = \cdots = i_N$.*

Matricizacija (eng. unfolding) je koncept koji se često koristi kod tenzora i vrlo je koristan jer služi da se tenzor prikaže u obliku matrice i time se olakšaju neke operacije. Na primjer, ako imamo tenzor dimenzija $3 \times 4 \times 2$, tada se takav tenzor može prikazati kao

[1] Odsječke tenzora reda 3 razlikujemo na horizontalne, lateralne i frontalne



Slika 1.2: Horizontalni, lateralni i frontalni odsječci tenzora reda 3



Slika 1.3: Dijagonalni tenzor dimenzija $I \times I \times I$ s jedinicama po dijagonali

matrica dimenzija 6×4 ili možda pak kao matrica dimenzija 3×8. Matricizaciju tenzora $\mathcal{A} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ po modu n definiramo kao matricu koja se dobije tako da niti u modu n posložimo kao stupce te takvu matricu označavamo $\mathbf{A}_{(n)}$. Element tenzora na poziciji $(i_1, i_2, \ldots, i_N)$ se tada mapira u element matrice na poziciji (i_N, j), pri čemu je

$$j = 1 + \sum_{\substack{k=1 \\ k \neq n}}^{N} (i_k - 1) J_k, \qquad \text{pri čemu je} \qquad J_k = \prod_{m=1, m \neq n}^{k-1} I_m.$$

Na prvi pogled, formula za izračunavanje elementa matrice dobivena matricizacijom izgleda kompliciranо, ali je jednostavnije ovaj koncept shvatiti pomoću primjera.

Primjer 1.1.2. *Neka je* $\mathcal{A} \in \mathbb{R}^{3\times4\times2}$ *neki tenzor kojemu su frontalni odsječci:*

$$A(:,:,1) = \begin{bmatrix} a_{111} & a_{121} & a_{131} & a_{141} \\ a_{211} & a_{221} & a_{231} & a_{241} \\ a_{311} & a_{321} & a_{331} & a_{341} \end{bmatrix}, \quad A(:,:,2) = \begin{bmatrix} a_{112} & a_{122} & a_{132} & a_{142} \\ a_{212} & a_{222} & a_{232} & a_{242} \\ a_{312} & a_{322} & a_{332} & a_{342} \end{bmatrix}$$

Jer je tenzor $\mathcal{A}$ *reda* 3*, imat ćemo* 3 *moguće matrizacije moda n:*

$$A_{(1)} = \begin{bmatrix} a_{111} & a_{121} & a_{131} & a_{141} & a_{112} & a_{122} & a_{132} & a_{142} \\ a_{211} & a_{221} & a_{231} & a_{241} & a_{212} & a_{222} & a_{232} & a_{242} \\ a_{311} & a_{321} & a_{331} & a_{341} & a_{312} & a_{322} & a_{332} & a_{342} \end{bmatrix},$$

$$A_{(2)} = \begin{bmatrix} a_{111} & a_{211} & a_{311} & a_{112} & a_{212} & a_{312} \\ a_{121} & a_{221} & a_{321} & a_{122} & a_{222} & a_{322} \\ a_{131} & a_{231} & a_{331} & a_{132} & a_{232} & a_{332} \\ a_{141} & a_{241} & a_{341} & a_{142} & a_{242} & a_{342} \end{bmatrix},$$

$$A_{(3)} = \begin{bmatrix} a_{111} & a_{211} & a_{311} & a_{121} & a_{221} & \dots & a_{331} & a_{141} & a_{241} & a_{341} \\ a_{112} & a_{212} & a_{312} & a_{122} & a_{222} & \dots & a_{332} & a_{142} & a_{242} & a_{342} \end{bmatrix}.$$

Bitna stvar za napomenuti je da poredak kojim preslikavamo niti u modu n u stupce nije bitan dok god smo konzistentni u daljnim izračunima s takvim poredkom. Zanimljivo je primjetiti da matritizaciju možemo i obrnuti, odnosno, matrice bi mogli pretvarati u tenzore obrnutim postupkom. Takav proces pretvorbe matrice u tenzor se zove tenzorifikacija (eng. folding).

1.2 Operacije s tenzorima

Kao i kod matrica, norma i skalarni produkt su veoma česte operacije koje primjenjujemo nad tenzorima. Takve funkcije nam omogućuju da izračunamo udaljenost između dva tenzora i koristimo ih kako bismo procijenili koliko dobro jedan tenzor aproksimira drugi. Što je manja udaljenost, to znači da je aproksimacija bolja. Njihova definicija je slična kao i kod matrica, jedino što ovdje moramo generalizirati takve operacije.

Definicija 1.2.1. *Neka je* $\mathcal{A} \in \mathbb{R}^{I_1\times I_2\times\cdots\times I_N}$ *proizvoljni tenzor reda* N*. Norma od* $\mathcal{A}$ *je tada jednaka:*

$$\|\mathcal{A}\| = \sqrt{\sum_{i_1=1}^{I_1}\sum_{i_2=1}^{I_2}\cdots\sum_{i_N=1}^{I_N} a_{i_1 i_2 \dots i_N}^2}.$$

Definicija je analogna Frobeniusovoj normi za matrice koja se označava s $\|\mathbf{A}\|_F$.

Definicija 1.2.2. *Neka su* $\mathcal{A}, \mathcal{B} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ *dva tenzora jednakog reda. Njihov skalarni produkt je tada jednak*

$$\langle \mathcal{A}, \mathcal{B} \rangle = \sum_{i_1=1}^{I_1} \sum_{i_2=1}^{I_2} \cdots \sum_{i_N=1}^{I_N} a_{i_1 i_2 \ldots i_N} b_{i_1 i_2 \ldots i_N}.$$

Iz ovoga slijedi direktno $\langle \mathcal{A}, \mathcal{A} \rangle = \|\mathcal{A}\|^2$.

Prirodno se postavlja pitanje možemo li tenzore međusobno množiti kao što to možemo s matricama ili možemo li tenzore i matrice množiti međusobno, i ako da, kako? Odgovor na to pitanje je potvrdan, a ovdje ćemo definirati množenje u modu n gdje množimo tenzore s matricom.

Definicija 1.2.3. *Množenjem u modu n tenzora* $\mathcal{A} \in \mathbb{R}^{I_1 \times I_2 \cdots \times I_N}$ *s matricom* $\mathbf{U} \in \mathbb{R}^{J \times I_n}$ *u oznaci* $\mathcal{A} \times_n \mathbf{U}$ *dobivamo tenzor dimenzija* $I_1 \times \cdots \times I_{n-1} \times J \times I_{n+1} \times \cdots \times I_N$ *čiji su elementi:*

$$(\mathcal{A} \times_n \mathbf{U})_{i_1 \ldots i_{n-1}\, j\, i_{n+1} \ldots i_N} = \sum_{i_n=1}^{I_n} a_{i_1 i_2 \ldots i_N} u_{j i_n}.$$

Ovakva ideja se može prikazati i matrično kao

$$\mathcal{Y} = \mathcal{X} \times_n \mathbf{U} \iff \mathbf{Y}_{(n)} = \mathbf{U}\mathbf{X}_{(n)}.$$

Zanimljiva činjenica je da ako imamo ulančano množenje s različitim modovima, tada je redoslijed množenja nebitan. Odnosno vrijedi:

$$\mathcal{A} \times_n \mathbf{P} \times_m \mathbf{R} = \mathcal{A} \times_m \mathbf{R} \times_n \mathbf{P} \quad \text{za } n \neq m.$$

Ako su modovi isti u ulančanom množenju, odnosno ako imamo $n = m$, tada vrijedi

$$\mathcal{A} \times_n \mathbf{P} \times_n \mathbf{R} = \mathcal{A} \times_n (\mathbf{R}\mathbf{P}).$$

Moguće je tenzore množit i s drugim tenzorima, naravno, samo u određenim slučajevima. Imamo sljedeću definiciju.

Definicija 1.2.4. $\binom{m}{n}$*-produkt tenzora* $\mathcal{A} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ *s tenzorom* $\mathcal{B} \in \mathbb{R}^{J_1 \times J_2 \times \cdots \times J_M}$*, gdje vrijedi* $I_n = J_m$*, je definiran kao:*

$$\mathcal{C} = \mathcal{A} \times_n^m \mathcal{B},$$

gdje je $\mathcal{C} \in \mathbb{R}^{I_1 \times \cdots \times I_{n-1} \times I_{n+1} \times \cdots \times I_N \times J_1 \times \cdots \times J_{m-1} \times J_{m+1} \times \cdots \times J_M}$*. Elementi od* $\mathcal{C}$ *su jednaki:*

$$\sum_{i=1}^{I_n} \mathcal{A}(i_1, \ldots, i_{n-1}, i, i_{n+1}, \ldots, i_N)\, \mathcal{B}(j_1, \ldots, j_{m-1}, i, j_{m+1}, \ldots, j_M).$$

Podsjetit ćemo se također i definicije Kroneckerovog produkta za matrice jer će nam biti potrebna u dokazu u kasnijim poglavljima.

Definicija 1.2.5. *Ako su* $\mathbf{A} \in \mathbb{R}^{m\times n}$ *i* $\mathbf{B} \in \mathbb{R}^{p\times q}$ *dvije matrice, tada je Kroneckerov produkt matrica* $\mathbf{A}$ *i* $\mathbf{B}$*, označen s* $\mathbf{A}\otimes\mathbf{B}$*, matrica dimenzija* $(mp \times nq)$ *definirana kao:*

$$\mathbf{A}\otimes\mathbf{B} = \begin{bmatrix} a_{11}\mathbf{B} & \cdots & a_{1n}\mathbf{B} \\ \vdots & \ddots & \vdots \\ a_{m1}\mathbf{B} & \cdots & a_{mn}\mathbf{B} \end{bmatrix}.$$

Navest ćemo neka od svojstava Kroneckerovog produkta.

Teorem 1.2.6. *Za matrice* $\mathbf{A}$ *i* $\mathbf{B}$ *vrijedi*

$$(\mathbf{A}\otimes\mathbf{B})^T = \mathbf{A}^T\otimes\mathbf{B}^T.$$

Dokaz. Neka je $\mathbf{A}$ matrica dimenzija $m \times n$. Tada vrijedi

$$(\mathbf{A}\otimes\mathbf{B})^T = \begin{bmatrix} a_{11}\mathbf{B} & \cdots & a_{1n}\mathbf{B} \\ \vdots & \ddots & \vdots \\ a_{m1}\mathbf{B} & \cdots & a_{mn}\mathbf{B} \end{bmatrix}^T = \begin{bmatrix} a_{11}\mathbf{B}^T & \cdots & a_{m1}\mathbf{B}^T \\ \vdots & \ddots & \vdots \\ a_{1n}\mathbf{B}^T & \cdots & a_{mn}\mathbf{B}^T \end{bmatrix} = \begin{bmatrix} a_{11} & \cdots & a_{m1} \\ \vdots & \ddots & \vdots \\ a_{1n} & \cdots & a_{mn} \end{bmatrix}\otimes\mathbf{B}^T = \mathbf{A}^T\otimes\mathbf{B}^T.$$

□

Teorem 1.2.7. *Ako su matrice* $\mathbf{A}$ *i* $\mathbf{B}$ *ortogonalne tada je i matrica* $\mathbf{A}\otimes\mathbf{B}$ *ortogonalna.*

Teorem 1.2.8. *Za matrice* $\mathbf{A}$*,* $\mathbf{B}$*,* $\mathbf{C}$ *i* $\mathbf{D}$ *vrijedi*

$$(\mathbf{A}\otimes\mathbf{B})(\mathbf{C}\otimes\mathbf{D}) = \mathbf{AC}\otimes\mathbf{BD}.$$

Poglavlje 2

Dekompozicija matrice

Dekompozicija matrica, a tako i tenzora, ima puno svrha, među kojima je i klasifikacija slika. Za početak ćemo reći nešto o takozvanoj dekompoziciji na singularne vrijednosti (eng. Singular Value Decomposition) ili skraćeno SVD, a zatim ćemo tu ideju proširiti i na tenzore te ćemo takvu dekompoziciju zvati HOSVD (eng. Higher Order Singular Value Decomposition).

SVD će nam omogućiti aproksimaciju nižeg ranga te pomoću njega možemo kompresirati slike. Za početak ćemo dokazati teorem koji nam kaže da se svaka matrica može faktorizirati kao produkt više matrica.

2.1 Osnovni teorem SVD-a i aproksimacija

U ovom dijelu ćemo reći nešto više o egzistenciji SVD-a te o aproksimacijama koristeći SVD. Dokazi sljedećih tvrdnji se također mogu pronaći u [5]. Prije teorema, podsjetimo se definicije norme za matrice, poznatiju kao i 2-norma.

Definicija 2.1.1. *Matrična* 2*-norma matrice* $\mathbf{A}$ *jednaka je*

$$||\mathbf{A}||_2 = \max_{\mathbf{x}\neq 0} \frac{||\mathbf{A}\mathbf{x}||_2}{||\mathbf{x}||_2} = \sqrt{\lambda_{\max}(\mathbf{A}^*\mathbf{A})},$$

gdje je $\mathbf{A}^*$ *adjungirana matrica matrice* $\mathbf{A}$*, dok je* $\lambda_{\max}(\mathbf{A}^*\mathbf{A})$ *maksimalna svojstvena vrijednost matrice.*

Teorem 2.1.2. *Neka je* $\mathbf{A} \in \mathbb{R}^{m\times n}$ *proizvoljna realna matrica i neka vrijedi* $m \geq n$*. Tada postoje ortogonalne matrice* $\mathbf{U} \in \mathbb{R}^{m\times m}$ *i* $\mathbf{V} \in \mathbb{R}^{n\times n}$ *i postoji dijagonalna matrica* $\mathbf{\Sigma} = \boldsymbol{diag}(\sigma_1, \sigma_2, \ldots, \sigma_n) \in \mathbb{R}^{m\times n}$ *te* $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0$*, takve da vrijedi:*

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T.$$

Napomena 2.1.3. *Uvjet $m \geq n$ nije nikakva restrikcija, ako to ne vrijedi tada teorem možemo primijeniti na A^T.*

Dokaz. Ako je $\mathbf{A}$ nulmatrica, odnosno $\mathbf{A} = 0$, tada za $\mathbf{\Sigma}$ možemo također staviti da je $\Sigma = 0$, a $\mathbf{U}$ i $\mathbf{V}$ mogu biti proizvoljne ortogonalne matrice i tvrdnja će vrijediti. Pretpostavimo sada da je $\mathbf{A} \neq 0$. Ostatak dokaza provodimo indukcijom po n.

Baza indukcije je $n = 1$ te tada imamo matricu $\mathbf{A}$ dimenzije $m \times 1$. Definiramo jedinični vektor $\mathbf{u}$ kao

$$\mathbf{u} = \frac{\mathbf{A}}{||\mathbf{A}||_2}.$$

Sada ga proširimo matricom $\hat{\mathbf{U}}$ tako da je matrica dimenzija $m \times m$ ortogonalna. Odnosno dobijemo da je $\mathbf{U}$ jednak

$$\mathbf{U} = \left[\mathbf{u}, \hat{\mathbf{U}}\right].$$

Preostale matrice $\mathbf{V} \in \mathbb{R}^{1\times 1}$ i $\mathbf{\Sigma} \in \mathbb{R}^{m\times 1}$ definiramo kao

$$\mathbf{\Sigma} = \begin{bmatrix} ||A||_2 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \quad \mathbf{V} = 1$$

i tvrdnja će vrijediti za svaki $m \geq 1$ i $n = 1$.

Pretpostavimo sada da tvrdnja vrijedi za svaku matricu dimenzije $(m-1) \times (n-1)$. Neka je $\mathbf{v}$ jedinični vektor za kojeg vrijedi

$$||\mathbf{A}||_2 = \max_{||x||_2=1} ||\mathbf{A}\mathbf{x}||_2 = ||\mathbf{A}\mathbf{v}||_2.$$

Odnosno, vektor $\mathbf{v}$ je takav vektor na kojem se dostiže maksimum 2-norme. Sada definiramo vektor

$$\mathbf{u} = \frac{\mathbf{A}\mathbf{v}}{||\mathbf{A}\mathbf{v}||_2}. \tag{2.1}$$

Kao i prije, nadopunimo sada upravo definirane vektore $\mathbf{u}$ i $\mathbf{v}$, matricama $\tilde{\mathbf{U}}$ i $\tilde{\mathbf{V}}$ na način da matrice $\mathbf{U_0} = \left[u, \tilde{\mathbf{U}}\right]$ i $\mathbf{V_0} = \left[v, \tilde{\mathbf{V}}\right]$ budu ortogonalne. Dimenzija matrice $\mathbf{U_0}$ će tada biti $m \times m$, a dimenzije matrice $\mathbf{V_0}$ je $n \times n$. Raspisivanjem dobijemo

$$\mathbf{U_0}^T \mathbf{A} \mathbf{V_0} = \begin{bmatrix} \mathbf{u}^T \\ \tilde{\mathbf{U}}^T \end{bmatrix} \mathbf{A} \left[v, \tilde{\mathbf{V}}\right] = \begin{bmatrix} \mathbf{u}^T\mathbf{A}\mathbf{v} & \mathbf{u}^T\mathbf{A}\tilde{\mathbf{V}} \\ \tilde{\mathbf{U}}^T\mathbf{A}\mathbf{v} & \tilde{\mathbf{U}}^T\mathbf{A}\tilde{\mathbf{V}} \end{bmatrix}.$$

Izračunajmo sada neke elemente ove matrice. Zbog definicije vektora $\mathbf{u}$ i $\mathbf{v}$ dobijemo da je prvi element ove matrice jednak

$$\mathbf{u}^T\mathbf{A}\mathbf{v} = \frac{\mathbf{v}^T\mathbf{A}^T}{||\mathbf{A}\mathbf{v}||_2}\mathbf{A}\mathbf{v} = \frac{||\mathbf{A}\mathbf{v}||_2^2}{||\mathbf{A}\mathbf{v}||_2} = ||\mathbf{A}\mathbf{v}||_2 = ||\mathbf{A}||_2 = \sigma_1.$$

Zbog ortogonalnosti matrica $\mathbf{U}_0$ i $\mathbf{V}_0$ znamo da je $\tilde{\mathbf{U}}^T\mathbf{u} = 0$. Zbog ortogonalnosti i jednakosti 2.1 slijedi

$$\tilde{\mathbf{U}}^T\mathbf{A}\mathbf{v} = \tilde{\mathbf{U}}^T\mathbf{u}||\mathbf{A}\mathbf{v}||_2 = 0 \cdot ||\mathbf{A}\mathbf{v}||_2 = 0.$$

Neka je sada $\mathbf{A}_1$ definirano kao

$$\mathbf{A}_1 = \mathbf{U_0}^T\mathbf{A}\mathbf{V_0} = \begin{bmatrix} \sigma_1 & \mathbf{w}^T \\ 0 & \mathbf{B} \end{bmatrix},$$

gdje je:

$$\mathbf{w}^T = \mathbf{u}^T\mathbf{A}\tilde{\mathbf{V}}, \quad \mathbf{B} = \tilde{\mathbf{U}}^T\mathbf{A}\tilde{\mathbf{V}}.$$

Unitarna invarijantnost nam daje

$$\sigma_1 = ||\mathbf{A}||_2 = ||\mathbf{U}_0^T\mathbf{A}\mathbf{V}_0||_2 = ||\mathbf{A}_1||_2. \tag{2.2}$$

Za proizvoljni ne nul vektor $\mathbf{z}$ vrijedi

$$||\mathbf{A}_1||_2 = \max_{\mathbf{x}\neq 0} \frac{||\mathbf{A_1}\mathbf{x}||_2}{||\mathbf{x}||_2} \geq \frac{||\mathbf{A}_1 z||_2}{||\mathbf{z}||_2}.$$

Odnosno,

$$||\mathbf{A}_1||_2||\mathbf{z}||_2 \geq ||\mathbf{A}_1\mathbf{z}||_2.$$

Ako je $\mathbf{z}$ jednak

$$\mathbf{z} = \begin{bmatrix} \sigma_1 \\ \mathbf{w} \end{bmatrix},$$

slijedi

$$\begin{aligned} ||\mathbf{A}_1||_2^2||\mathbf{z}||_2^2 = ||\mathbf{A}_1||_2^2\left(\sigma_1^2 + ||\mathbf{w}||_2^2\right) \geq ||\mathbf{A}_1\mathbf{z}||_2^2 = \left\|\mathbf{A}_1\begin{bmatrix}\sigma_1 \\ \mathbf{w}\end{bmatrix}\right\|_2^2 \\ = \left(\sigma_1^2 + \mathbf{w}^T\mathbf{w}\right)^2 + ||\mathbf{B}\mathbf{w}||_2^2 \geq \left(\sigma_1^2 + ||\mathbf{w}||_2^2\right)^2. \end{aligned}$$

Znači da imamo nejednakost

$$||\mathbf{A}_1||_2\left(\sigma_1^2 + ||\mathbf{w}||_2^2\right) \geq \left(\sigma_1^2 + ||\mathbf{w}||_2^2\right)^2.$$

Dijeljenjem dobijemo

$$||\mathbf{A}_1||_2^2 \geq \sigma_1^2 + ||\mathbf{w}||_2^2.$$

Iz 2.2 slijedi da je

$$\sigma_1^2 \geq \sigma_1^2 + ||\mathbf{w}||_2^2.$$

Pokazali smo da je $\|\mathbf{w}\|_2^2 = 0$, to jest $\mathbf{w} = 0$. Vidimo da vrijedi

$$\mathbf{U_0}^T \mathbf{A} \mathbf{V_0} = \begin{bmatrix} \sigma_1 & 0 \\ 0 & \mathbf{B} \end{bmatrix}.$$

Po pretpostavci indukcije matrica $\mathbf{B}$ se može zapisati kao

$$\mathbf{B} = \mathbf{U}_1 \mathbf{\Sigma}_1 \mathbf{V}_1^T.$$

Uvrštavanjem dobijemo

$$\mathbf{U}_0^T \mathbf{A} \mathbf{V}_0 = \begin{bmatrix} \sigma_1 & 0 \\ 0 & \mathbf{U}_1 \mathbf{\Sigma}_1 \mathbf{V}_1^T \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & \mathbf{U}_1 \end{bmatrix} \begin{bmatrix} \sigma_1 & 0 \\ 0 & \mathbf{\Sigma}_1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & \mathbf{V}_1^T \end{bmatrix}.$$

Iz čega slijedi tvrdnja.

Da bi pokazali kako su singularne vrijednosti u padajućem poretku, sjetimo se permutacijskih matrica koje su također ortogonalne. Time možemo dobiti poredak singularnih vrijednost kakav želimo jer ortogonalne matrice čine multiplikativnu grupu. □

Definicija 2.1.4. *Stupce $u_1, u_2, \ldots, u_m$ matrice $\mathbf{U}$ zovemo lijevi singularni vektori, stupce $v_1, v_2, \ldots, v_n$ matrice $\mathbf{V}$ zovemo desni singularni vektori, a realne brojeve σ_i zovemo singularne vrijednosti.*

U praktičnoj primjeni SVD ćemo često imati da je $m \gg n$ pa promotrimo pobliže takav slučaj. Ako raspišemo matrice $\mathbf{U}$, $\mathbf{V}$ i $\mathbf{\Sigma}$ dobijemo

$$\begin{bmatrix} \vdots & & \vdots \\ u_1 & \cdots & u_m \\ \vdots & & \vdots \end{bmatrix} \begin{bmatrix} \sigma_1 & 0 & \cdots & 0 \\ 0 & \sigma_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_n \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix} \begin{bmatrix} \vdots & & \vdots \\ v_1 & \cdots & v_n \\ \vdots & & \vdots \end{bmatrix}^T.$$

Primijetimo da će nakon n koraka svaki član u množenju biti jednak 0 jer Σ samo na dijagonali ima pozitivne vrijednosti. Zato ima smisla uzeti samo prvih n članova, a ostale nije potrebno množiti. Time ćemo smanjiiti vrijeme izvršavanja, a takav proces se onda zove tanki SVD. Ako definiramo k kao $k = min(n, m)$ tada možemo zapisati matricu $\mathbf{A}$ kao

$$\mathbf{A} = \mathbf{U}_k \mathbf{\Sigma}_k \mathbf{V}_k^T.$$

gdje $\mathbf{U}_k, \mathbf{V}_k^T$ znači da sadržavaju samo prvih k stupaca matrice $\mathbf{U}_k$ i $\mathbf{V}^T$ dok $\mathbf{\Sigma}_k$ sadrži samo prvih k singularnih vrijednosti. Matricu $\mathbf{A}$ možemo zapisati i kao zbroj matrica ranga 1:

$$\mathbf{A} = \sum_{i=1}^{k} \sigma_i u_i v_i^T.$$

Budući da su singularne vrijednosti sortirane silazno, to implicira da su prvih nekoliko vrijednosti značajno veće od ostatka, pa stoga ima smisla aproksimirati matricu $\mathbf{A}$ uzimajući u obzir samo prvih nekoliko singularnih vrijednosti.

Teorem 2.1.5. *Neka je $\mathbf{A} \in \mathbb{R}^{m\times n}$ proizvoljna matrica i njen rang označimo s $r = rang(\mathbf{A}) \leq min(n, m)$. Za*

$$\mathbf{A}_k = \sum_{i=1}^{k} \sigma_i u_i v_i^T, \quad k < r,$$

pri čemu k označava da smo uzeli prvih k lijevih i desnih singularnih vektora i prvih k singularnih vrijednosti, vrijedi

$$\min_{rang(X)\leq k} \left\|\mathbf{A} - \mathbf{X}\right\|_2 = \left\|\mathbf{A} - \mathbf{A_k}\right\|_2 = \sigma_{k+1}.$$

Dokaz. Potrebno je dokazati nekoliko stvari. Prvo, da vrijedi

$$\left\|\mathbf{A} - \mathbf{A_k}\right\|_2 = \sigma_{k+1}. \tag{2.3}$$

Drugo, da za proizvoljnu matricu $\mathbf{B}$ ranga najviše k, vrijedi

$$\left\|\mathbf{A} - \mathbf{A_k}\right\|_2 \leq \left\|\mathbf{A} - \mathbf{B}\right\|_2. \tag{2.4}$$

Kako bi dokazali jednadžbu 2.3 jednostavnim raspisivanjem dobijemo

$$\mathbf{A} - \mathbf{A_k} = \sum_{i=1}^{n} \sigma_i u_i v_i^T - \sum_{i=1}^{k} \sigma_i u_i v_i^T = \sum_{i=k+1}^{n} \sigma_i u_i v_i^T,$$

Što znači da vrijedi

$$\left\|\mathbf{A} - \mathbf{A_k}\right\|_2 = \left\|\mathbf{U} diag\left(0, \ldots, 0, \sigma_{k+1}, \ldots, \sigma_n\right) \mathbf{V}^T\right\|_2 = \sigma_{k+1},$$

pri čemu zadnja jednakost vrijedi jer su singularne vrijednosti poredane silazno. Dokažimo sada nejednakost 2.4. Neka je matrica $\mathbf{B} \in \mathbb{R}^{m\times n}$ proizvoljna matrica takva da je $rang(B) \leq k$. Tada prema teoremu o rangu i defektu slijedi da je jezgra netrivijalna, odnosno nul-potprostor od $\mathbf{B}$ je veći ili jednak $n - k$. Označimo dimenziju jezgre s $d \leq n - k$.

Nul-potprostor od $\mathbf{B}$ je tada razapet vektorima $x_1, \ldots, x_d$, u oznaci span$\{x_1, \ldots, x_d\}$. Jer je potprostor razapet vektorima $v_1, v_2, \ldots, v_{k+1}$ dimenzije $k+1$, tada vrijedi

$$\text{span}\{x_1, \ldots, x_d\} \cap span\{v_1, \ldots, v_{k+1}\} \neq \{0\}.$$

Znači da postoji neki vektor z, pa tako i jedinični vektor, takav da pripada presjeku. Jer je $Bz = 0$ i

$$\mathbf{A}z = \sum_{i=1}^{k+1} \sigma_i \left(v_i^T z\right) u_i,$$

tada vrijedi

$$\left\|\mathbf{A} - \mathbf{B}\right\|_2^2 \geq \left\|(\mathbf{A} - \mathbf{B})z\right\|_2^2 = \left\|\mathbf{A}z\right\|_2^2 = \sum_{i=1}^{k+1} \sigma_i^2 \left(v_i^T z\right)^2 \geq \sigma_{k+1}^2.$$

□

Svrhu teorema 2.1.5 možemo dobro vidjeti na primjeru kompresije crno-bijelih slika jer njih možemo lagano prikazati pomoću matrica. Crno-bijele slike se mogu prikazati pomoću matrica tako da svaki element u matrici predstavlja broj između 0 i 255 te predstavlja intenzitet crne boje u svakom pikselu.

Uzmimo za primjer sliku 2.1 i programski jezik Python. Za početak, učitamo sliku koja ima originalne dimenzije 1917×1515, odnosno 2904255 piksela, ali smo je ovdje umanjili radi preglednosti.

```
from matplotlib.image import imread
import os
import numpy as np
import matplotlib.pyplot as plt

imgLoad = r'bear.png'
X = imread(imgLoad)
img_org = plt.imshow(X)
X = np.mean(X, -1); # Convert RGB to grayscale

img = plt.imshow(X)
img.set_cmap('gray')
plt.axis('off')
plt.show()
```

Zatim, u Pythonu vrlo laganu možemo dobiti SVD od matrice koja predstavlja našu sliku koristeći funkciju *svd* iz biblioteke *numpy*. Koristit ćemo tanki SVD te napisati funkciju koja prima k i pomoću njega aproksimira matricu $\mathbf{X}$. Uzimamo $k \times k$ podmatricu matrice $\mathbf{U}$, $\mathbf{\Sigma}$ i $\mathbf{V}^T$.

Napomena 2.1.6. *Varijablu S je potrebno pretvoriti u dijagonalnu matricu s np.diag.*

Slika 2.1: Originalna slika u boji i crno-bijela s 2904255 piksela

```
U, S, VT = np.linalg.svd(X, full_matrices=False) #za tanki SVD
    postavljamo full_matrices = False
S = np.diag(S) #pretvori S u dijagonalnu matricu

def approxMatrix(k):
    Xapprox = U[:,:k] @ S[0:k,:k] @ VT[:k,:]
    plt.figure()
    img = plt.imshow(Xapprox)
    img.set_cmap('gray')
    plt.show()
```

Na kraju, funkciju koju smo upravo definirali možemo pozvati sa parametrima $k = 5$, $k = 20$, $k = 50$ te $k = 100$ i kao rezultat dobijemo sliku 2.2.

```
for k in (5,20,50,100):
    approxMatrix(k)
```

Možemo primjetiti da kada postavimo k na mali broj, kvaliteta slike značajno padne, ali za dovoljno veliki k skoro pa ni ne razlikujemo aproksimaciju od originalne slike.

Na slici 2.3 možemo vidjeti graf singularnih vrijednosti na logaritamskoj skali i vidimo kako singularne vrijednosti na početku značajno padaju. Odnosno, prvih 20-ak vrijednosti

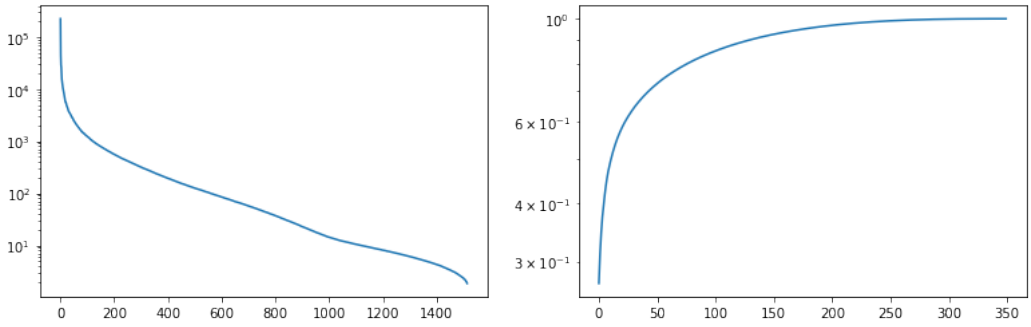
(a) veličina: $1917 \cdot 5 + 5 + 5 \cdot 1515 = 17165$ px, omjer kompresije: $2904255/17165 = 169.20$

(b) veličina: $1917 \cdot 20 + 20 + 20 \cdot 1515 = 68660$, omjer kompresije: $2904255/68660 = 42.30$

(c) veličina: $1917 \cdot 50 + 50 + 50 \cdot 1515 = 171650$ px, omjer kompresije: 16.92

(d) veličina: $1917 \cdot 100 + 100 + 100 \cdot 1515 = 343300$ px, omjer kompresije: 8.46

Slika 2.2: Aproksimacija slike s (a) $k = 5$, (b) $k = 20$, (c) $k = 50$, (d) $k = 100$



(a) Singularne vrijednosti

(b) Odnos sume prvih k singularnih vrijednosti i sume svih singularnih vrijednostima dobivene pomoću formule 2.5

Slika 2.3: Analiza singularnih vrijednosti crno-bijele slike 2.1

su značajno bitnije od ostalih vrijednosti te zato već s $k = 20$, slika je dobro vidljiva i već tada prepoznajemo o kojoj slici je riječ. Također, možemo vidjeti odnos sume prvih k singularnih vrijednosti i svih singularnih vrijednosti, odnosno, ako je $n < m$ za matricu $\mathbf{A} \in \mathbb{R}^{m \times n}$, onda iz grafa možemo iščitavati:

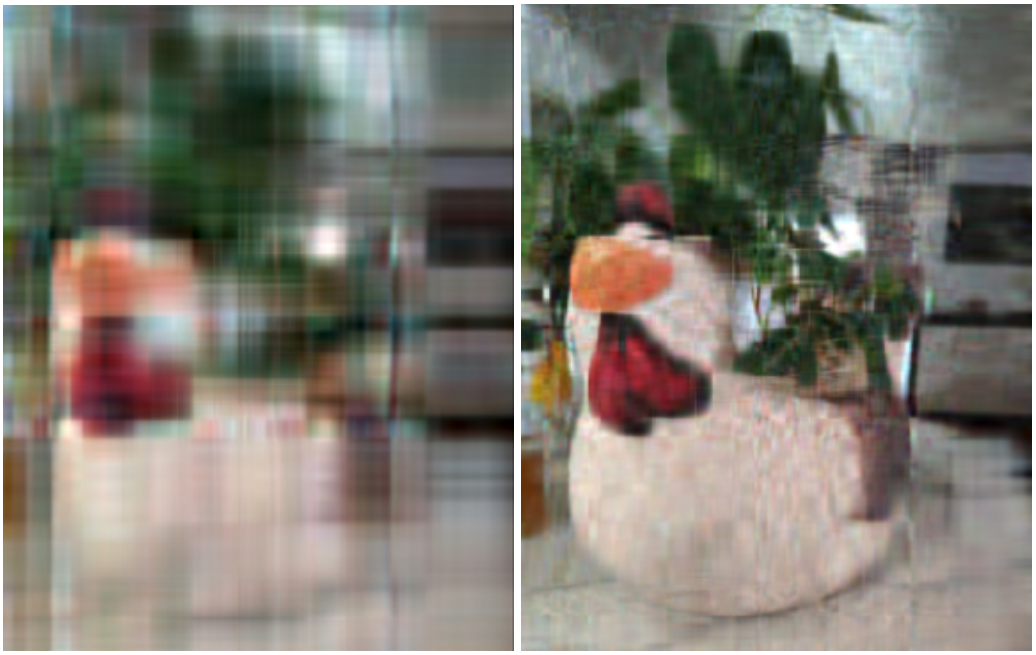
$$\frac{\sum_{i=1}^{k} \sigma_i}{\sum_{i=1}^{n} \sigma_i} \tag{2.5}$$

Opet, vidimo da za $k \approx 30$, aproksimacija je vrlo dobra, a kako k raste dobivamo bolju aproksimaciju ali greška se ne smanjuje toliko brzo kao na početku. U Pythonu, graf bi nacrtali sljedećim kodom:

```
plt.figure(1)
plt.semilogy(np.diag(S))
plt.title('Singularne vrijednosti')
plt.show()

plt.figure(2)
plt.semilogy(np.cumsum(np.diag(S))/np.sum(np.diag(S)))
plt.title('Odnos prvih $k$ singularnih vrijednosti sa svim singularnim
    vrijednostima')
plt.show()
```

Slike u boji bismo isto mogli kompresirati koristeći SVD, ali uz malo komplikacija. Postoje 3 kanala koja tvore sliku u boji: crveni, zeleni i plavi. Za svaki taj kanal moramo posebno napraviti SVD dekompoziciju gdje je svaki kanal reprezentiran kao matrica čiji su elementi između 0 i 255 i oni predstavljaju intenzitet određene boje na slici. Nakon što smo napravili kompresiju pomoću SVD-a, sva 3 kanala potrebno je spojit da stvore novu, kompresiranu sliku.



(a) $k = 5$ (b) $k = 20$

(c) $k = 50$ (d) $k = 100$

Slika 2.4: Kompresirane slike, veličina i omjer kompresije su kao i na slici 2.2

Poglavlje 3

Dekompozicije tenzora

U ovom poglavlju opisat ćemo neke dekompozicije tenzora. Za početak opisujemo HOSVD, proširenu verziju SVD-a, a zatim ćemo opisati *tensor train* dekompoziciju te dati par primjera gdje se takve dekompozicije koriste.

3.1 HOSVD

SVD višeg reda (eng. Higher order SVD), ili skraćeno HOSVD, poznatija je još kao i specijalni slučaj Tuckerove dekompozicije u kojem su podtenzori ortogonalni te proširuje koncept SVD-a kod tenzora. Svrha HOSVD-a je da dekompozira tenzor na način gdje se takozvani jezgreni tenzor množi matricama po svakom modu. Odnosno, ako na primjer imamo tenzor $\mathcal{A} \in \mathbb{R}^{I \times J \times K}$ reda 3 tada ćemo imati

$$\mathcal{A} = \mathcal{S} \times_1 \mathbf{U}^{(1)} \times_2 \mathbf{U}^{(2)} \times_3 \mathbf{U}^{(3)}.$$

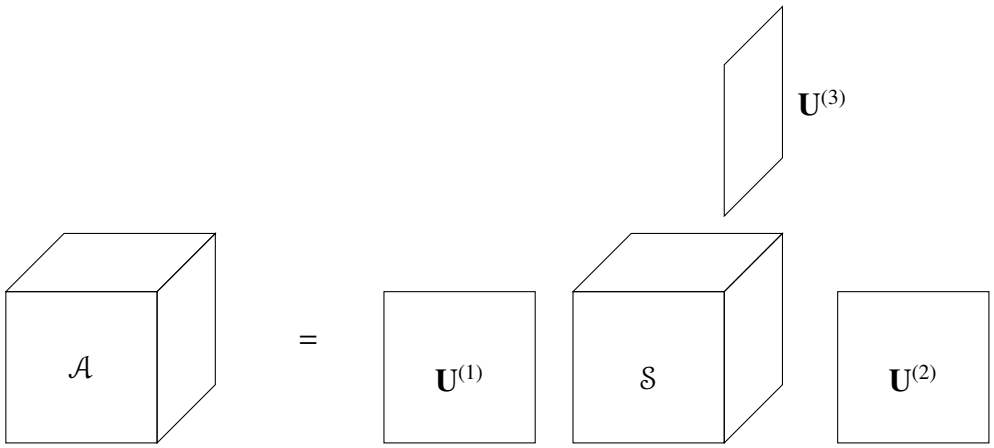
Ovakvu dekompoziciju možemo bolje shvatiti promatrajući sliku 3.1.

Napomena 3.1.1. *Matrica je tenzor reda* 2*, pa SVD matrice* $\mathbf{A}$ *se može zapisati i kao*

$$\mathbf{A} = \mathbf{\Sigma} \times_1 \mathbf{U} \times_2 \mathbf{V}.$$

Koristeći svojstva množenja moda n i Kroneckerovog produkta, za neki tenzor $\mathcal{A}$ reda 3 vrijedi da se može matrično zapisati po modovima:

$$\begin{aligned}
\mathbf{A}_{(1)} &= \mathbf{U}^{(1)} \mathcal{S}_{(1)} \left(\mathbf{U}^{(3)} \otimes \mathbf{U}^{(2)} \right)^T, \\
\mathbf{A}_{(2)} &= \mathbf{U}^{(2)} \mathcal{S}_{(2)} \left(\mathbf{U}^{(3)} \otimes \mathbf{U}^{(1)} \right)^T, \\
\mathbf{A}_{(3)} &= \mathbf{U}^{(3)} \mathcal{S}_{(3)} \left(\mathbf{U}^{(2)} \otimes \mathbf{U}^{(1)} \right)^T.
\end{aligned}$$



Slika 3.1: Primjer HOSVD-a za tenzor reda 3

Primijetimo da ako su matrice $\mathbf{U}^{(1)} \in \mathbb{R}^{I\times P}$, $\mathbf{U}^{(2)} \in \mathbb{R}^{J\times Q}$, $\mathbf{U}^{(3)} \in \mathbb{R}^{K\times R}$ te tenzor $\mathcal{S} \in \mathbb{R}^{P\times Q\times R}$, odnosno $\mathcal{S} \in \mathbb{R}^{P\times QR}$, tada je na primjer matrica $\mathbf{A}_{(1)}$ dobro definirana jer dobijemo da je dimenzija od $\mathbf{U}^{(1)}\mathcal{S}_{(1)}$ jednaka $I\times QR$, a dimenzija Kroneckerovog produkta $\mathbf{U}^{(3)}\otimes\mathbf{U}^{(2)}$ je tada $KJ \times RQ$ pa transponirana matrica ima dimenzije $QR \times KJ$. Znači da desna strana nakon svih množenja ima dimenziju $I \times JK$ što je dimenzija matrice $\mathbf{A}_{(1)}$. Vrijedi i općenitija verzija ovakvog množenja koja je pokazana u dokazu teorema 3.1.2 u jednadžbi 3.4.

Više o ovakvom prikazu matriciziranih tenzora možete pročitati u [7] gdje je također dan dokaz idućeg teorema.

Teorem 3.1.2. *Svaki tenzor $\mathcal{A}$ dimenzija $I_1 \times I_2 \times \cdots \times I_N$ se može faktorizirati kao:*

$$\mathcal{A} = \mathcal{S} \times_1 \mathbf{U}^{(1)} \times_2 \mathbf{U}^{(2)} \times_3 \cdots \times_N \mathbf{U}^{(N)}, \tag{3.1}$$

i vrijedi

- $\mathbf{U}^{(n)} = \left(\mathbf{u}_1^{(n)}, \mathbf{u}_2^{(n)}, \ldots, \mathbf{u}_{I_n}^{(n)}\right)$ *su ortogonalne matrice dimenzija $I_n \times I_n$ gdje je vektor $\mathbf{u}_i^{(n)}$ i-ti singularni vektor u n-tom modu*
- *Tenzor $\mathcal{S}$, dimenzija $I_1 \times I_2 \times \cdots \times I_N$ još zovemo i jezgreni tenzor, podtenzori $\mathcal{S}_{i_n=\alpha}$ su dobiveni fiksiranjem n-tog indeksa koji je jednak α i za njega vrijedi sljedeće:*
  1. *Ortogonalnost: dva podtenzora $\mathcal{S}_{i_n=\alpha}$ i $\mathcal{S}_{i_n=\beta}$ su ortogonalni u smislu skalarnog umnoška, za sve moguće vrijednosti n:*

$$\left\langle \mathcal{S}_{i_n=\alpha}, \mathcal{S}_{i_n=\beta} \right\rangle = 0 \quad \forall \alpha \neq \beta.$$

2. *Redoslijed:*

$$\|\mathcal{S}_{i_n=1}\| \geq \|\mathcal{S}_{i_n=2}\| \geq \cdots \geq \|\mathcal{S}_{i_n=I_n}\| \geq 0 \quad \forall n = 1, \ldots, N.$$

Odnosno, ako singularne vrijednosti definiramo kao:

$$\sigma_i^{(n)} = \|\mathcal{S}_{i_n=i}\|,$$

tada vrijedi:

$$\sigma_1^{(n)} \geq \sigma_2^{(n)} \geq \cdots \geq \sigma_{I_n}^{(n)} \quad \forall n = 1, \ldots, N. \tag{3.2}$$

Dokaz. Dokaz pokazuje vezu između HOSVD od $\mathcal{A}$ i SVD od njegovih matricizacija. Neka je $\mathcal{A} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ proizvoljni tenzor. Definiramo tenzor $\mathcal{S} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ kao

$$\mathcal{S} = \mathcal{A} \times_1 \mathbf{U}^{(1)^T} \times_2 \mathbf{U}^{(2)^T} \times_3 \cdots \times_N \mathbf{U}^{(N)^T}. \tag{3.3}$$

Pri čemu su $\mathbf{U}^{(n)}$ ortogonalne matrice. Jednakost 3.3 možemo zapisati u matričnom obliku:

$$\mathbf{A}_{(n)} = \mathbf{U}^{(n)} \cdot \mathbf{S}_{(n)} \cdot \left(\mathbf{U}^{(n+1)} \otimes \mathbf{U}^{(n+2)} \otimes \cdots \otimes \mathbf{U}^{(N)} \otimes \mathbf{U}^{(1)} \otimes \mathbf{U}^{(2)} \otimes \cdots \otimes \mathbf{U}^{(n-1)}\right). \tag{3.4}$$

Razmotrimo mod n matricizaciju od $\mathcal{A}$, $\mathbf{A}_{(n)}$ i njegov SVD:

$$\mathbf{A}_{(n)} = \mathbf{U}^{(n)} \mathbf{\Sigma}^{(n)} \left(\mathbf{V}^{(n)}\right)^T, \tag{3.5}$$

gdje je $\mathbf{\Sigma}^{(n)} = diag\left(\sigma_1^{(n)}, \sigma_2^{(n)}, \ldots, \sigma_{I_n}^{(n)}\right)$ i za tu matricu znamo da vrijedi $\sigma_1^{(n)} \geq \sigma_2^{(n)} \geq \cdots \geq \sigma_{I_n}^{(n)} \geq 0$ i matrice $\mathbf{U}^{(n)}$, $\mathbf{V}^{(n)}$ su ortogonalne. To možemo napraviti za svaki $n = 1, \ldots, N$.

S r_n označimo najmanji indeks takav da je $\sigma_{r_n}^{(n)} > 0$. Uspoređujući jednadžbe 3.4 i 3.5 i s obzirom da su matrice ortogonalne, dobijemo da je:

$$\mathbf{S}_{(n)} = \mathbf{\Sigma}^{(n)} \cdot \mathbf{V}^{(n)^T} \cdot \left(\mathbf{U}^{(n+1)} \otimes \mathbf{U}^{(n+2)} \otimes \cdots \otimes \mathbf{U}^{(N)} \otimes \mathbf{U}^{(1)} \otimes \mathbf{U}^{(2)} \otimes \cdots \otimes \mathbf{U}^{(n-1)}\right).$$

Iz čega slijedi da za proizvoljne ortogonalne matrice $\mathbf{U}^{(1)}$, $\mathbf{U}^{(2)}, \ldots, \mathbf{U}^{(n-1)}$, $\mathbf{U}^{(n+1)}, \ldots, \mathbf{U}^{(N)}$ vrijedi

$$\left\langle \mathcal{S}_{i_n=\alpha}, \mathcal{S}_{i_n=\beta} \right\rangle = 0, \quad \forall \alpha \neq \beta,$$

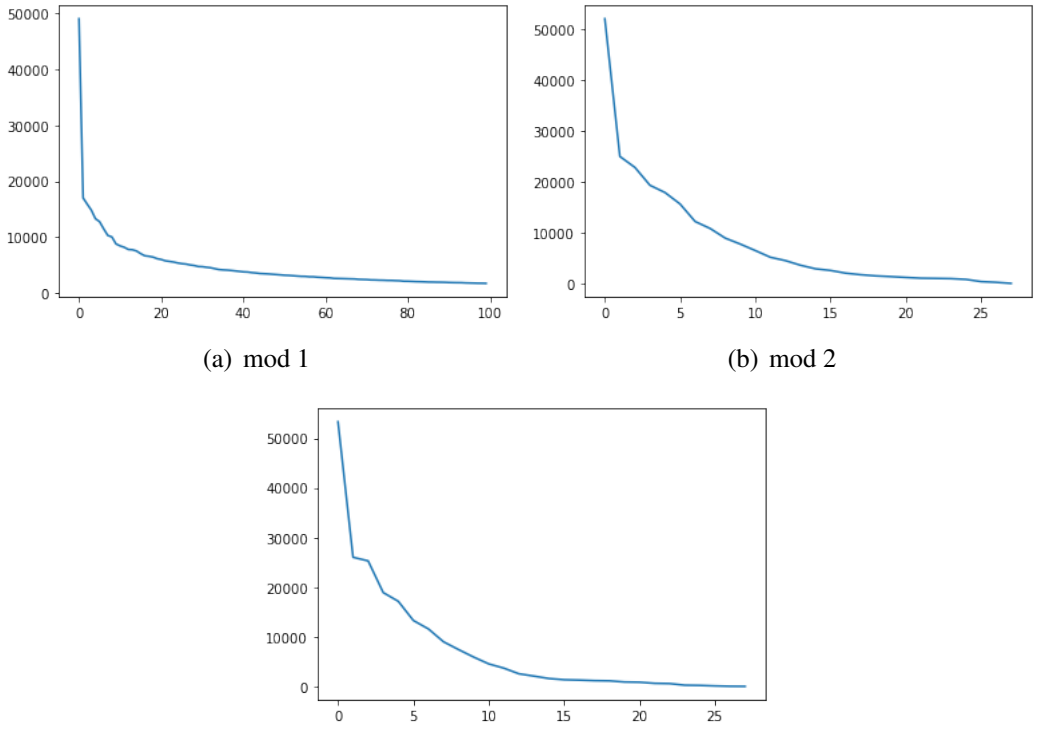
te

$$\|\mathcal{S}_{i_n=1}\| = \sigma_1^{(n)} \geq \|\mathcal{S}_{i_n=2} = \sigma_2^{(n)}\| \geq \cdots \geq \|\mathcal{S}_{i_n=I_n}\| = \sigma_{I_n}^{(n)} \geq 0,$$

i, ako je $r_n < I_n$, tada:

$$\|\mathcal{S}_{i_n=r_n+1}\| = \sigma_{r_n+1}^{(n)} = \cdots = \|\mathcal{S}_{i_n=I_n}\| = \sigma_{I_n}^{(n)} = 0.$$

Konstruirajući matrice $\mathbf{U}^{(1)}$, $\mathbf{U}^{(2)}, \ldots, \mathbf{U}^{(n-1)}$, $\mathbf{U}^{(n+1)}, \ldots, \mathbf{U}^{(N)}$ na sličan način kao što smo to napravili s $\mathbf{U}^{(n)}$, $\mathcal{S}$ se može konstruirati tako da budu zadovoljeni svi zahtjevi iskaza teorema. Također, kao što vidimo iz jednakosti 3.5, sve matrice $\mathbf{U}^{(1)}$, $\mathbf{U}^{(2)}, \ldots, \mathbf{U}^{(N)}$ i tenzor $\mathcal{S}$ koji zadovoljavaju uvjete se mogu pronaći pomoću SVD od $\mathbf{A}_{(1)}, \mathbf{A}_{(2)}, \ldots . \mathbf{A}_{(N)}$, gdje $\mathcal{S}$ slijedi iz 3.3. □



(a) mod 1

(b) mod 2

(c) mod 3

Slika 3.2: Singularne vrijednosti jezgrenog tenzora u svim modovima

Uzmimo tenzor iz MNIST baze podataka od nasumično odabranih 1000 slika znamenaka. Takav tenzor je dimenzija $1000 \times 28 \times 28$, pri čemu 28×28 označava dimenzije slika. Primijenimo li HOSVD nad takvim tenzorom uzimajući rang $\{100, 28, 28\}$ u funkciji *tucker* iz *tensorly* biblioteke, možemo prikazati singularne vrijednosti u svim modovima jezgrenog tenzora kao što možemo vidjeti na slici 3.2. Singularne vrijednosti su u padajućem poretku baš kao što je istaknuto u iskazu teorema 3.1.2.

Dokaz teorema 3.1.2 i jednadžba 3.5 nam ujedno daje i algoritam kojim možemo izračunati HOSVD nekog tenzora $\mathcal{A}$ reda N. Lako možemo dobiti singularne matrice moda n, $\mathbf{U}^{(n)}$, kao lijeva singularna matrica od matricizacije tenzora u modu n. Znači, moramo izračunati N različitih SVD matrica dimenzija $I_n \times I_1 I_2 \dots I_{n-1} I_{n+1} \dots I_N$ gdje je $1 \leq n \leq N$. Zatim, jezgreni tenzor $\mathcal{S}$ se dobije izračunavanjem:

$$\mathcal{S} = \mathcal{A} \times_1 \mathbf{U}^{(1)^T} \times_2 \mathbf{U}^{(2)^T} \times_3 \cdots \times_N \mathbf{U}^{(N)^T}. \tag{3.6}$$

Algoritam možemo prvo napisati u pseudokodu za lakše razumijevanje.

U Pythonu, HOSVD funkcija bi se mogla realizirati na sljedeći način:

Algorithm 1 HOSVD

Require: Tenzor $\mathcal{A} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$
  inicijaliziraj praznu listu U_lista
  for $n = 1$ to N **do**
    $\mathbf{A}_{(n)} \leftarrow$ Matriciziraj tenzor $\mathcal{A}$ u modu n
    izračunaj SVD od $\mathbf{A}_{(n)}$, $\mathbf{A}_{(n)} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$
    spremi $\mathbf{U}$ u listu U_lista
  end for
  Izračunaj $\mathcal{S} = \mathcal{A} \times_1 \mathbf{U}^{(1)^T} \times_2 \mathbf{U}^{(2)^T} \times_3 \cdots \times_N \mathbf{U}^{(N)^T}$ gdje $\mathbf{U}^{(n)}$ dobijemo iz varijable U_lista
  return U_lista, $\mathcal{S}$

```
from tensorly.tenalg import mode_dot

def hosvd(tensor):
    U_matrices = []

    for mode in range(tensor.ndim):
        # matricizacija tenzora u modu n
        unfold = np.moveaxis(tensor, mode, 0)
        unfold = unfold.reshape(unfold.shape[0], -1)

        # izracunaj svd matrice i U stavi u listu matrica
        U, S, Vt = np.linalg.svd(unfold, full_matrices=False)
        U_matrices.append(U)

    S_tensor = tensor
    for mode, U in enumerate(U_matrices):
        # izracunaj jezgreni tenzor formulom iz dokaza
        S_tensor = mode_dot(S_tensor, U.T, mode)
    return U_matrices, S_tensor
```

Funkcija *hosvd* prima tenzor kao argument te vraća listu matrica $\mathbf{U}^{(n)}$ i jezgreni tenzor $\mathcal{S}$. Na početku funkcije definiramo varijablu *U_matrices* koja predstavlja listu matrica koje ćemo koristiti u izračunavanju jezgrenog tenzora. Zatim prolazimo po svim modovima tenzora i matriciziramo tenzor u mod *n*. Nakon što izračunamo SVD matrice, spremamo varijablu *U* u listu matrica. Nakon što smo dobili sve potrebne matrice, tenzor računamo po formuli i spremamo ga u varijablu *S_tensor*, pri tome smo koristili funkciju *mode_dot* iz biblioteke *tensorly*. Na kraju funkcije vraćamo listu matrica i izračunati tenzor. U biblioteci tensorly se nalazi napravljena funkcija za izračunavanje Tuckerove dekompozicije, odnosno HOSVD dekompozicije koju pozivamo kao:

```
import tensorly as tl
from tensorly.decomposition import tucker

```

```
S_tensor, U_matrices = tucker(tensor, rank=tensor.shape)
```

Također vraća jezgreni tenzor i listu matrica. U funkciju moramo proslijediti tenzor koji želimo dekomponirati, u našem slučaju je to varijabla *tensor*, te dimenziju tog tenzora koju možemo dobiti pozivajući *tensor.shape*. Usporedimo li implementaciju iz *tensorly* biblioteke i naše implementacije *hosvd* funkcije pomoću *time* funckije, primjetit ćemo da na primjer za izvršavanje dekompozicije tenzora dimenzija $5 \times 5 \times 5 \times 5 \times 5 \times 5$, *tensorly* biblioteka radi otprilike 3 puta brže nego naša implementacija. Zbog toga, u daljnjem radu ćemo koristiti funkciju iz *tensorly* biblioteke.

Jedinstvenost singularnih vektora u HOSVD-u je do na predznak ako su elementi realne vrijednosti.

Da bi provjerili radi li nam funkcija kako očekujemo, možemo probati pomnožiti matrice i tenzor koji nam funkcija vraća i usporediti ga s originalnim tenzorom. Kako bi to napravili, pišemo funkciju *reconstruct_tensor* koji prima 2 argumenta. Koristeći funkciju *multi_mode_dot*, naša funkcija bi izgledala ovako:

```
from tensorly.tenalg import multi_mode_dot

def reconstruct_tensor(S_tensor, U_matrices):
    # rekonstruiraj tenzor
    reconstructed_tensor = multi_mode_dot(S_tensor, U_matrices,
    transpose=False)

    return reconstructed_tensor
```

Kao što smo to napravili kod matrica, tako i tenzore možemo aproksimirati koristeći HOSVD. Iz teorema 3.1.2 slijedi da se tenzor $\mathcal{A} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$ reda 3 onda može zapisati pomoću singularnih vektora i matrica kao:

$$\mathcal{A} = \sum_{i=1}^{I_3} \mathbf{A}_i \times_3 \mathbf{u}_i^{(3)}, \quad \mathbf{A}_i = \mathcal{S}(:,:,i) \times_1 \mathbf{U}^{(1)} \times_2 \mathbf{U}^{(2)},$$

gdje su $\mathbf{u}_i^{(3)}$ stupci matrice $\mathbf{U}^{(3)}$, a $\mathbf{A}_i$ su matrice dimenzija $I_2 \times I_3$, odnosno tenzori dimenzija $I_2 \times I_3 \times 1$. Također vrijedi:

$$\langle \mathbf{A}_i, \mathbf{A}_j \rangle = tr\left[\mathbf{U}^{(2)} \mathcal{S}(:,:,i)^T \left(\mathbf{U}^{(1)}\right)^T \mathbf{U}^{(1)} \mathcal{S}(:,:,j) \left(\mathbf{U}^{(2)}\right)^T\right] = tr\left[\mathcal{S}(:,:,i)^T \mathcal{S}(:,:,j)\right] = 0,$$

jer je $tr(\mathbf{AB}) = tr(\mathbf{BA})$, pri čemu $tr(\mathbf{A})$ označava trag matrice $\mathbf{A}$ koji je definiran kao

$$tr(\mathbf{A}) = \sum_{i=1}^{n} a_{ii}.$$

Odnosno, vidimo da svaki odsječak u modu 3 tenzora $\mathcal{A}$ bi mogli zapisati u ortogonalnoj bazi $(\mathbf{A}_i)_{i=1}^{r_3}$ gdje smo s r_3 označili broj pozitivnih singularnih vrijednosti u modu 3.

Dobivamo:

$$\mathcal{A}(:,:,j) = \sum_{i=1}^{r_3} z_i^{(j)} \mathbf{A}_i, \tag{3.7}$$

pri čemu je $z_i^{(j)}$ j-ta komponenta vektora $\mathbf{u}_i^{(3)}$ te

$$\mathbf{A}_i = \mathcal{S}(i,:,:) \times_1 \mathbf{U}^{(1)} \times_2 \mathbf{U}^{(2)}.$$

Primijetimo da jezgreni tenzor nema svojstvo da je općenito dijagonalni tenzor kao što smo imali slučaj kod SVD-a, ali, promatrajući 3.2 vidimo da su ipak sortirani nekako. Ako promatramo tenzor reda 3 tada im je najveća energija sadržana u gornjem lijevom kutu i kako se pomičemo prema donjem desnom kutu, vrijednost im opada.

Zato, kao i prije, možemo se ograničiti na prvih k baznih matrica $\mathbf{A}_i$ te prvih k singularnih vektora u modu 3. Aproksimaciju nižeg ranga tenzora $\mathcal{A}$ tada dobijemo kao:

$$\mathcal{A} = \sum_{i=1}^{k} \mathbf{A}_i \times_3 \mathbf{u}_i^{(3)}, \quad \mathbf{A}_i = \mathcal{S}(i,:,:) \times_1 \mathbf{U}^{(1)} \times_2 \mathbf{U}^{(2)}.$$

Greška koja nastaje je ograničena sljedećim izrazom:

$$\|\mathcal{A} - \mathcal{A}'\| \leq \sum_{i=k+1}^{r_3} (\sigma_i^{(3)})^2.$$

Koristeći HOSVD, odnosno Tucker dekompoziciju i funkciju iz *tensorly*-a, opet možemo aproksimirati sliku. Sljedećim programom možemo proizvoljnu sliku kompresirati.

```
import matplotlib.pyplot as plt
import tensorly as tl
import numpy as np
from tensorly.decomposition import tucker


# ucitaj sliku
image = imread(imgLoad)
image = tl.tensor(image, dtype='float64')

def to_image(tensor):
    im = tl.to_numpy(tensor)
    im -= im.min()
    im /= im.max()
    im *= 255
    return im.astype(np.uint8)

# rank of the Tucker decomposition
```

```
tucker_rank = [25, 25, 3]

# tucker dekompozicija
core, tucker_factors = tucker(image, rank=tucker_rank, init='random',
    tol=10e-5)
tucker_reconstruction = tl.tucker_to_tensor((core, tucker_factors))

# prikazi kompresiranu sliku
fig = plt.figure(figsize=(10, 10))
ax = fig.add_subplot(1, 3, 3)
ax.set_axis_off()
ax.imshow(to_image(tucker_reconstruction))
ax.set_title('Tucker')

plt.show()
```

Pri čemu je varijabla *imgLoad* ime slike te koristimo pomoćnu funkciju *to_image* kako bi iz tenzora dobili sliku. Varijablu *tucker_rank* koristimo da bi definirali koliko želimo aproksimirati jezgreni tenzor. Jer je slika u boji, broj 3 označava da aproksimiramo koristeći sve boje.

Dobivena aproksimacija se može vidjeti na slici 3.3.

3.2 Tensor Train

Tensor train dekompozicija (TT dekompozicija) je uvedena kako bi se pokušali izbjeći nedostatci u HOSVD dekompoziciji, a poznatija je i kao "*matrix product state*" dekompozicija u svijetu fizike. Kako smo i prije spomenuli, za HOSVD tenzora $\mathcal{A}$ je bilo potrebno čuvati jezgreni tenzor istih dimenzija kao $\mathcal{A}$ te smo morali računati i N matrica. Za veliki N, ovakvo spremanje podataka je sporo i nije efikasno. Zato, da bi se izbjegla ovisnost o dimenziji, *tensor train* dekompozicija dekompozira tenzor reda N u produkt tenzora reda 3. Isto tako, koristeći algoritme temeljenim na SVD-u, izračunamo faktore u dekompoziciji.

Definicija 3.2.1. *Neka je $\mathcal{A} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ tenzor reda N. Kažemo da je tenzor $\mathcal{A}$ u TT-formatu ako se njegovi elementi mogu zapisati u obliku:*

$$\mathcal{A}(i_1, \ldots, i_N) = \mathbf{G}(i_1, :)\mathcal{G}_2(:, i_2, :) \ldots \mathcal{G}_{N-1}(:, i_{N-1}, :)\mathbf{G}_N(:, i_N), \tag{3.8}$$

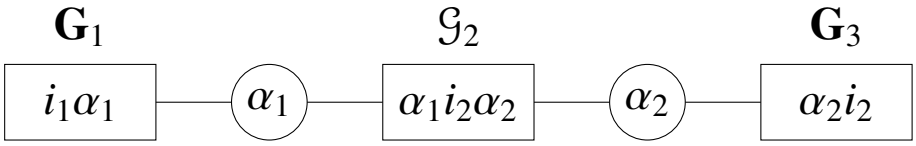
pri tome:

- $\mathbf{G}_1 \in \mathbb{R}^{r_1 \times I_1}$, $\mathbf{G}_N \in \mathbb{R}^{r_{N-1} \times I_N}$ *i* $\mathcal{G}_k \in \mathbb{R}^{r_{k-1} \times I_k \times r_k}$ *za* $k = 1, \ldots, N$ *se nazivaju TT-jezgre.*
- r_k *za* $k = 1, \ldots, N$ *se nazivaju TT rangovi i* $r_0 = r_N = 1$.
- $r = \max\limits_{1 \leq k \leq N} r_k$ *se naziva maksimalni TT-rang.*

(a) varijabla *tucker_rank* = {25, 25, 3}; veličina: 86437 piksela; omjer kompresije: 33.59

(b) varijabla *tucker_rank* = {50, 50, 3}; veličina: 179109 piksela; omjer kompresije: 16.21

Slika 3.3: Kompresirane slike dobivene pomoću HOSVD dekompozicije



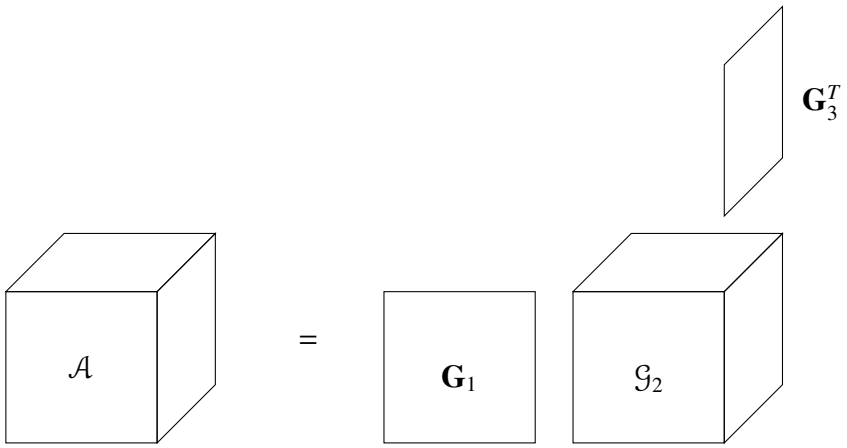
Slika 3.4: *Tenzor train* koristeći indekse za tenzor reda 3

Jednadžbu 3.8 možemo napisati pomoću indeksa na sljedeći način:

$$\mathcal{A}(i_1, \ldots, i_N) = \sum_{\alpha_1, \ldots, \alpha_{N-1}} \mathbf{G}_1(i_1, \alpha_1)\mathcal{G}_2(\alpha_1, i_2, \alpha_2) \ldots \mathbf{G}_N(\alpha_{N-1}, i_N), \tag{3.9}$$

te se takva *tensor train* dekompozicija zapisana koristeći indekse vizualizira pomoću slike 3.4.

Sa slike možemo zaključiti sljedeće. Pravokutni čvorovi sadrže indekse iz izvornog tenzora i barem jedan pomoćni indeks. Kružni čvorovi sadrže samo jedan pomoćni indeks. Dva pravokutna čvora su povezana ako i samo ako imaju zajednički pomoćni indeks α_k.



Slika 3.5: Primjer *tenzor train*-a za tenzor reda 3

Iz slike također vidimo zašto se ovakva dekompozicija zove *tensor train* (hrv. tenzorski vlak), ovakvo množenje upravo možemo interpretirati kao vagone, odnosno kao vlak.

Koristeći definiciju $\binom{m}{n}$-produkta tenzora, jednadžbu 3.9 možemo zapisati i kao

$$\mathcal{A} = \mathbf{G}_1 \times_2^1 \mathcal{G}_2 \times_3^1 \mathcal{G}_3 \times_3^1 \cdots \times_3^1 \mathbf{G}_N.$$

Ako imamo tenzor $\mathcal{A}$ reda 3 tada je njegova *tensor train* dekompozicija jednaka

$$\mathcal{A} = \mathcal{G}_2 \times_1 \mathbf{G}_1 \times_3 \mathbf{G}_3^T, \tag{3.10}$$

zbog toga što je

$$\begin{aligned}
\mathcal{A}(i_1, i_2, i_3) &= \mathbf{G}_1(i_1, :) \times_2^1 \mathcal{G}_2(:, i_2, :) \times_3^1 \mathbf{G}_3(:, i_3) \\
&= \sum_{\alpha_1, \alpha_2} \mathbf{G}_1(i_1, \alpha_1) \mathcal{G}_2(\alpha_1, i_2, \alpha_2) \mathbf{G}_3(\alpha_2, i_3) \\
&= \sum_{\alpha_2} \left(\mathcal{G}_2 \times_1 \mathbf{G}_1\right)(i_1, i_2, \alpha_2) \mathbf{G}_3(\alpha_2, i_3) \\
&= \left(\mathcal{G}_2 \times_1 \mathbf{G}_1 \times_3 \mathbf{G}_3^T\right)(i_1, i_2, i_3).
\end{aligned}$$

Jednadžbu 3.10 također možemo vizualizirati slikom 3.5.

Svaki faktor u TT dekompoziciji se dobije pomoću SVD-a od pojedine matricizacije u modu n tenzora $\mathcal{A}$. Tako na primjer, prvi član je izračunat pomoću SVD od $\mathbf{A}_{(1)} = \mathbf{G}_1 \mathbf{\Sigma}_1 \mathbf{V}_1^T$.

Kao i prije, pomoću TT dekompozicije je također moguće aproksimirati tenzore. Razmatramo samo prvih k stupaca od $\mathbf{G}_1$ i takvu novu matricu označimo s $\hat{\mathbf{G}}_1$. Posljedično,

postavimo $\hat{\mathcal{G}}_2 \in \mathbb{R}^{k \times n_e \times n_p}$ kao tenzor od prvih k mod 1 elemenata od $\mathcal{G}_2$. Aproksimirani tenzor je tada jednak

$$\mathcal{A} = \hat{\mathcal{G}}_2 \times_1 \hat{\mathbf{G}}_1 \times_3 \mathbf{G}_3^T.$$

Najčešće ćemo razmatrati aproksimacije tenzora u TT formatu sa preciznošću ϵ. Tako ćemo originalni tenzor $\mathcal{A}$ zamijeniti tenzorom $\mathcal{B}$ u TT formi za koju vrijedi

$$\|\mathcal{A} - \mathcal{B}\|_F \leq \epsilon \|\mathcal{B}\|_F.$$

Za početak, napomenimo da možemo dobiti granicu za rang r_k. Svaki α_k se pojavljuje dva puta u jednadžbi 3.9 pa je odozdo omeđen rangom matricizacije od $\mathcal{A}$:

$$\mathbf{A}_{(k)} = \mathbf{A}_{(k)}(i_1, \ldots, i_k; i_{k+1} \ldots i_N) = \mathcal{A}(i_1, \ldots, i_N), \tag{3.11}$$

gdje prvih k indeksa predstavljaju redove u matrici $\mathbf{A}_{(k)}$, a zadnjih $N - k$ indeksa predstavljaju stupce. Također, veličina ovakve matrice je

$$\left(\prod_{s=1}^{k} I_s\right) \times \left(\prod_{s=k+1}^{N} I_s\right).$$

Sljedeći teorem i dokaz nam daju konstruktivni način kako dobiti TT dekompoziciju nekog tenzora.

Teorem 3.2.2. *Ako za svaku matricizaciju $\mathbf{A}_{(k)}$ koja je zapisana u formi 3.11, N dimenzionalnog tenzora $\mathcal{A}$ vrijedi*

$$rang\ \mathbf{A}_{(k)} = r_k,$$

tada postoji TT dekompozicija s TT rangovima ne većim od r_k

Dokaz. Razmotrimo matritizaciju $\mathbf{A}_{(1)}$. Rang te matrice je jednak r_1. Također, može se zapisati kao:

$$\mathbf{A}_{(1)} = \mathbf{U}\mathbf{V}^T,$$

ili, koristeći indekse:

$$\mathbf{A}_{(1)}(i_1; i_2, \ldots, i_N) = \sum_{\alpha_1}^{r_1} \mathbf{U}(i_1, \alpha_1)\mathbf{V}(\alpha_1; i_2, \ldots, i_N).$$

Matrica $\mathbf{V}$ se može zapisati kao:

$$\mathbf{V} = \mathbf{A}_{(1)}^T \mathbf{U} \left(\mathbf{U}^T \mathbf{U}\right)^{-1} = \mathbf{A}_{(1)}^T \mathbf{W},$$

ili koristeći indekse:

$$\mathbf{V}(\alpha_1; i_2, \ldots, i_N) = \sum_{i_1=1}^{I_1} \mathcal{A}(i_1, \ldots, i_N)\mathbf{W}(i_1, \alpha_1).$$

Matrica $\mathbf{V}$ se može gledati i kao $(N-1)$ dimenzionalni tenzor $\mathcal{V}$ gdje je $(\alpha_1 i_2)$ jedan produženi indeks:

$$\mathcal{V}(\alpha_1 i_2, i_3, \ldots, i_N) = \mathbf{V}(\alpha_1; i_2, \ldots, i_N).$$

Promotrimo matricizacije $\mathbf{V}_{(2)}, \ldots, \mathbf{V}_{(N)}$. Pokazujemo da vrijedi rang $(\mathbf{V}_{(k)}) \leq r_k$. Za k-ti mod, TT rang je jednak r_k pa možemo zapisati

$$\mathbf{A}_{(k)}(i_1, \ldots, i_k; i_{k+1}, \ldots, N) = \sum_{\beta=1}^{r_k} \mathbf{F}(i_1, \ldots, i_k; \beta)\mathbf{G}(\beta; i_{k+1}, \ldots, i_N).$$

Iz ovoga dobijemo

$$\begin{aligned}
\mathbf{V}_{(k)}(i_1, \ldots, i_{k+1}; i_{k+2}, \ldots, i_N) &= \sum_{i_1=1}^{I_1} \mathbf{A}_{(k)}(i_1, \ldots, i_k; i_{k+1}, \ldots, i_N)\mathbf{W}(i_1, \alpha_1) \\
&= \sum_{i_1=1}^{I_1} \sum_{\beta=1}^{r_k} \mathbf{F}(i_1, \ldots, i_k; \beta)\mathbf{G}(\beta; i_{k+1}, \ldots, i_N)\mathbf{W}(i_1, \alpha_1) \\
&= \sum_{\beta=1}^{r_k} \mathbf{H}(\alpha_1 i_2, \ldots, i_k; \beta)\mathbf{G}(\beta; i_{k+1}, \ldots, i_N),
\end{aligned}$$

gdje je

$$\mathbf{H}(\alpha_1 i_2, \ldots, i_k; \beta) = \sum_{i_1=1}^{I_1} \mathbf{F}(i_1, \ldots, i_k; \beta)\mathbf{W}(i_1, \alpha_1).$$

Razdvojili smo indekse od $\mathbf{V}_{(k)}$ i rang $(\mathbf{V}_{(k)}) \leq r_k$. Ovaj proces nastavimo indukcijom. Razmotrimo matricu $\mathbf{V}_{(k)}$ i razdvojimo indeks (α_1, i_2):

$$\mathcal{V}(\alpha_1 i_2, i_3, \ldots, i_N) = \sum_{\alpha_2=2}^{r_2} \mathcal{G}_2(\alpha_1, i_2, \alpha_2)\mathcal{V}'(\alpha_2 i_3, i_4, \ldots, i_N),$$

te time dobijemo idući jezgreni tenzor $\mathcal{G}_2$. Ovaj proces ponavljamo do $\mathcal{G}_N$. □

U dokazu teorema 3.2.2 opisali smo algoritam kako dobiti TT dekompoziciju proizvoljnog tenzora i taj algoritam se naziva TT-SVD algoritam.

Kao što smo rekli, zbog efikasnosti, često ćemo htjeti raditi s aproksimacijama. Zato ćemo imati aproksimaciju nižeg ranga:

$$\mathbf{A}_{(k)} = \mathbf{R}_{(k)} + \mathbf{E}_{(k)}, \quad \text{rang}(\mathbf{R}_{(k)}) = r_k, \quad \|\mathbf{E}_{(k)}\| = \epsilon_k, \quad k = 1, \ldots, N-1. \tag{3.12}$$

Teorem 3.2.3. *Neka matricizacije $\mathbf{A}_{(k)}$ tenzora $\mathcal{A}$ zadovoljavaju 3.12. Tada TT-SVD izračunava tenzor $\mathcal{B}$ u TT formatu sa TT rangovima r_k i vrijedi:*

$$\|\mathcal{A} - \mathcal{B}\|_F \leq \sqrt{\sum_{k=1}^{N} \epsilon_k^2}.$$

Dokaz. Dokaz se provodi indukcijom. Za $N = 2$, tvrdnja vrijedi zbog svojstva SVD-a. Neka je $N > 2$. Tada je matricizacija $\mathbf{A}_{(1)}$ dekomponirana kao:

$$\mathbf{A}_{(1)} = \mathbf{U}_{(1)}\mathbf{\Sigma}\mathbf{V}_{(1)} + \mathbf{E}_{(1)} = \mathbf{U}_{(1)}\mathbf{B}_{(1)} + \mathbf{E}_{(1)},$$

gdje je $\mathbf{U}_1$ dimenzija $I_1 \times r_1$ te ima ortogonalne stupce i $\|\mathbf{E}_{(1)}\| = \epsilon_1$. Matrica $\mathbf{B}_{(1)}$ je prirodno povezana s tenzorom $\mathcal{B}_1$ reda $N-1$ s elementima $\mathcal{B}(\alpha_1 i_2, i_3, \ldots, i_N)$ i bit će kasnije dekompozirana s TT-SVD algoritmom. Znači da je $\mathbf{B}_{(1)}$ aproksimirana nekom matricom $\hat{\mathbf{B}}_{(1)}$. Pomoću SVD-a slijedi da je $\mathbf{U}_{(1)}^T\mathbf{E}_{(1)} = 0$ pa je:

$$\|\mathcal{A} - \mathcal{B}\|_F^2 = \|\mathbf{A}_{(1)} - \mathbf{U}_{(1)}\hat{\mathbf{B}}_{(1)}\|_F^2 = \|\mathbf{A}_{(1)} - \mathbf{U}_{(1)}(\hat{\mathbf{B}}_{(1)} + \mathbf{B}_{(1)} - \mathbf{B}_{(1)})\|_F^2.$$

Daljnim raspisom dobijemo da je to jednako:

$$\|\mathbf{A}_{(1)} - \mathbf{U}_{(1)}(\hat{\mathbf{B}}_{(1)} + \mathbf{B}_{(1)} - \mathbf{B}_{(1)})\|_F^2 = \|\mathbf{A}_{(1)} - \mathbf{U}_{(1)}\mathbf{B}_{(1)}\|_F^2 + \|\mathbf{U}_{(1)}(\mathbf{B}_{(1)} - \hat{\mathbf{B}}_{(1)})\|_F^2.$$

Zbog toga što $\mathbf{U}_{(1)}$ ima ortogonalne stupce vrijedi i iz pretpostavke teorema imamo:

$$\|\mathcal{A} - \mathcal{B}\|_F^2 \leq \epsilon_1^2 + \|\mathbf{B}_{(1)} - \hat{\mathbf{B}}_{(1)}\|_F^2. \tag{3.13}$$

Također, matricu $\mathbf{B}_{(1)}$ možemo izraziti kao

$$\mathbf{B}_{(1)} = \mathbf{U}_{(1)}^T\mathbf{A}_{(1)}.$$

Iz ortogonalnosti stupaca matrice $\mathbf{U}_{(1)}$ slijedi da je udaljenost od k-te matricizacije tenzora $\mathcal{B}_1$ reda $N-1$ do r_k-tog ranga matrice ne može biti veći od ϵ_k pa indukcijom slijedi da je:

$$\|\mathbf{B}_{(1)} - \hat{\mathbf{B}}_{(1)}\|_F^2 \leq \sum_{k=2}^{N-1} \epsilon_k^2.$$

Zajedno s 3.13 dokaz je završen.

□

Korolar 3.2.4. *Za tenzor $\mathcal{A}$ i rangova ograničenih s r_k najbolja aproksimacija za $\mathcal{A}$ u Frobenisiovoj normi sa TT rangovima ograničeni s r_k uvijek postoji (označimo je s $\mathcal{A}^{best}$) i TT aproksimacija dobivena s TT-SVD algoritmom je kvazi-optimalna:*

$$\|\mathcal{A} - \mathcal{B}\|_F \leq \sqrt{N-1}\|\mathcal{A} - \mathcal{A}^{best}\|_F.$$

Više o dokazima ovih tvrdnji se može pronaći u [8]. Napišimo sada pseudokod za TT-SVD uz pomoć prethodnih dokaza kako bi bolje razumjeli *tensor train* dekompoziciju.

Algorithm 2 TT-SVD

Require: Tenzor $\mathcal{A} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$, tražena preciznost ϵ
  inicijaliziraj potrebne varijable
  izračunaj $\delta = \frac{\epsilon}{\sqrt{N-1}}\|A\|_F$
  privremeni tenzor $\mathcal{C} = \mathcal{A}$, $r_0 = 1$
  for $k = 1$ to $N-1$ **do**
    $\mathbf{C} = \text{reshape}\left(\mathcal{C}, \left[r_{k-1}n_k, \frac{numel(C)}{r_{k-1}n_k}\right]\right)$
    provedi SVD s δ aproksimacijom matrice $\mathbf{C}$, $\mathbf{C} = \mathbf{U\Sigma V}^T + \mathbf{E}$, $\|\mathbf{E}\|_F \leq \delta, r_k = \text{rank}_\delta(\mathbf{C})$

    Nova jezgra: $\mathbf{G}_k = \text{reshape}(\mathbf{U}, [r_{k-1}, n_k, r_k])$
    $\mathbf{C} = \mathbf{\Sigma V}^T$
  end for
  $\mathbf{G}_N = \mathbf{C}$
  return $\mathbf{G}_1, \ldots, \mathbf{G}_N$

Implementaciju TT-SVD-a algoritma možemo napraviti koristeći sljedeći kod u Pythonu:

```
def tt_svd(A, epsilon):
    """
    Izracunaj TT dekompoziciju koristeci TT-SVD algoritam

    Arguments:
    - A: numpy array, tenzor koji zelimo dekompozirati.
    - epsilon: float, preciznost.

    Returns:
    - G: list of numpy array, jezgreni tenzori.
    """
    # inicijalizacija
    N = A.ndim
    # delta oznacava granicu do koje dozvoljavamo gresku i izracunava se
    po formuli iz korolara
    delta = epsilon / np.sqrt(N - 1) * np.sqrt(np.sum(A**2))
```

```
16
17      C = A
18      # lista rangova
19      r = [1]
20
21      # lista jezgrenih tenzora
22      G = []
23
24      for k in range(1, N):
25          # koristimo -1 da automatski odredi broj stupaca
26          # r_{k-1}*I_k je dimenzija retka
27          C = C.reshape(r[k-1] * A.shape[k-1], -1)
28
29          # provedi SVD za matricu C
30          U, S, Vt = la.svd(C, full_matrices=False)
31          # izracunavamo rang r_k,
32          # min osigurava da r_k nije veci od ranga S
33          r_k = min(np.sum(np.cumsum(S[::-1]**2)**0.5 <= delta), len(S))
34          # r_k mora bit barem 1
35          r_k = max(1, r_k)
36          U, S, Vt = U[:,:r_k], S[:r_k], Vt[:r_k,:]
37          # stavi r_k u listu rangova
38          r.append(r_k)
39
40          # odredi novi jezgreni tenzor G_k i stavi u listu jezgrenih
    tenzora
41          G_k = U.reshape(r[k-1], A.shape[k-1], r_k)
42          G.append(G_k)
43
44          # Izracunaj novi C
45          C = np.diag(S) @ Vt
46
47      # Zadnji jezgreni tenzor
48      G_N = C.reshape(r[-1], A.shape[-1])
49      G.append(G_N)
50
51      # vrati tenzor u TT formatu
52      return G
```

Bitno je napomenuti da za manje epsilone funkcija neće raditi dobro kao što očekujemo, ali ovdje smo napisali za bolje shvaćanje algoritma. Zato je bolje koristiti već bolje istestirane funkcije s većom preciznošću iz biblioteke *tensorly*. Biblioteka *tensorly* sadrži funkciju

```
1       matrix_product_state(input_tensor, rank)
```

koja odradi TT dekompoziciju nad tenzorom i vraća listu faktora u dekompoziciji. Funkciju pozivamo tako da proslijedimo vlastiti tenzor umjesto varijable *input_tensor* te za

rank možemo staviti *input_tensor.shape* kojom dohvaćamo dimenzije našeg tenzora. Kao što vidimo, možemo koristiti tanki SVD (*truncated_svd*). Također, nakon što dobijemo jezgrene tenzore od tenzora, njih možemo iskoristiti da dobijemo originalni tenzor. Na primjer ako imamo:

```
    cores = matrix_product_state(input_tensor, rank)
```

Tada pozivom:

```
    original_tensor = tt_to_tensor(cores)
```

dobijemo originalni tenzor iz jezgrenih tenzora. Funkcija iz *tensorly* biblioteke je također za otprilike trostruko brža nad tenzorima reda 3 pa ćemo iz tog razloga koristiti već gotovo napravljene funkcije.

Kao i prije, pomoću *tensor train* dekompozicije možemo dobiti aproksimaciju slika. Sljedećim kodom proizvoljnu sliku možemo kompresirati:

```
import numpy as np
import matplotlib.pyplot as plt
import tensorly as tl
from tensorly.decomposition import matrix_product_state

image_path = 'flower.jpg'
image = plt.imread(image_path)

normalized_image = image / 255.0

# TT rang za dekompoziciju
rank = [1, 30,30, 1]

compressed_image = matrix_product_state(normalized_image, rank)

reconstructed_image = tl.tt_to_tensor(compressed_image)


plt.imshow(reconstructed_image, cmap='gray')
plt.axis('off')
plt.show()
```

Pokretanjem ovog koda dobit ćemo sliku 3.6

(a) *rank* = $\{1, 25, 25, 1\}$; veličina: 161559 piksela; omjer kompresije: 17.98

(b) *rank* = $\{1, 50, 50, 1\}$; veličina: 323109 piksela; omjer kompresije: 8.99

Slika 3.6: Kompresirana slika dobivena pomoću *tensor train* dekompozicije

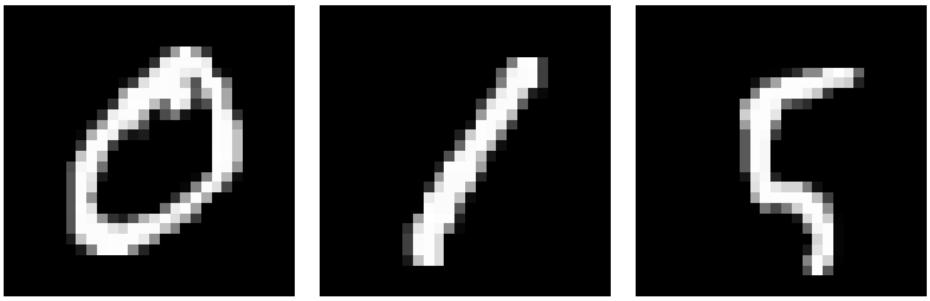
Poglavlje 4

Primjer klasifikacije slika pomoću Pythona

Sada ćemo klasificirati slike koristeći HOSVD i *tensor train* dekompozicije. Za klasifikaciju rukom pisanih znamenki koristit ćemo prvo HOSVD metodu, a zatim ćemo opisati i postupak klasifikacije znamenki pomoću *tensor train* algoritma. Oba algoritma su implementirana koristeći programski jezik Python.

4.1 Klasifikacija znamenki pomoću HOSVD algoritma

Kao primjer klasifikacije slika pomoću tenzorskih metoda, uzet ćemo MNIST bazu podataka u kojoj se nalazi više desetaka tisuća slika rukom pisanih znamenaka od 0 do 9. Primjer slika znamenaka nula, jedan i pet iz baze podataka možemo vidjeti na slici 4.1. Primijetimo, nisu sve znamenke "lijepo" prikazane pa neki postotak greške će uvijek biti prisutan. Dimenzija svake slike je 28×28 te ćemo njih predstavljati pomoću matrice istih



Slika 4.1: Primjer slika iz MNIST baze podataka

dimenzija. Također, za svaku znamenku ćemo uzeti samo prvih 100 slika te znamenke za testiranje pa ćemo za svaku znamenku imati tenzor dimenzija $28 \times 28 \times 1000$.

Za klasifikaciju znamenki pomoću HOSVD metode ćemo provoditi sljedeći postupak. Za svaku sliku znamenke provodimo HOSVD te novu sliku znamenke zapišemo u bazi koristeći 3.7 i izračunamo grešku. Promatrajući singularne vrijednosti znamenaka vidjet ćemo da je dovoljno uzeti 10 baznih matrica znamenaka umjesto 100 za dosta veliku točnost i manju količinu računanja. Ukratko rečeno, za nepoznatu znamenku $\mathbf{Z}$ računat ćemo minimizaciju greške prikaza u bazi:

$$\min_z \left\| \mathbf{Z} - \sum_j z_j A_j \right\|_2 .$$

Ako definiramo funkciju G kao:

$$G(z) = \frac{1}{2} \left\| \mathbf{Z} - \sum_j z_j A_j \right\|_2^2 = \frac{1}{2} \left\langle \mathbf{Z} - \sum_j z_j A_j, \mathbf{Z} - \sum_j z_j A_j \right\rangle .$$

Baza $(\mathbf{A}_i)_{i=1}^k$ je ortogonalna pa dobijemo da je funkcija $G(z)$ jednaka

$$G(z) = \frac{1}{2}\langle \mathbf{Z}, \mathbf{Z} \rangle - \sum_j z_j \langle \mathbf{Z}, \mathbf{A}_j \rangle + \frac{1}{2} \sum_j z_j^2 \langle \mathbf{A}_j, \mathbf{A}_j \rangle .$$

Tada točku minimuma dobijemo pomoću derivacije:

$$\frac{\partial G}{\partial z_j} = -\langle \mathbf{Z}, \mathbf{A}_j \rangle + z_j \langle \mathbf{A}_j, \mathbf{A}_j \rangle = 0,$$

pa koeficijente nepoznate znamenke računamo formulom:

$$z_j = \frac{\langle \mathbf{Z}, \mathbf{A}_j \rangle}{\langle \mathbf{A}_j, \mathbf{A}_j \rangle}.$$

Za dohvaćanje baznih matrica koristit će se funkcija *get_base_matrices*() koja vraća 10 baznih matrice pojedine znamenke. Napomenimo da je dohvaćanje podataka iz MNIST baze podataka te razdvajanje iste te baze na testne i trening podatke vrlo jednostavna i obavljamo ju prije definicije funkcije.

```
1 import numpy as np
2 from tensorly.decomposition import tucker
3 import tensorly as tl
4 from keras.datasets import mnist
5
```

```
(train_images, train_labels), (test_images, test_labels) = mnist.
    load_data()
k = 10

def get_base_matrices(digit):
    # odaberi 1000 slika znamenki "digit"
    digits = train_images[train_labels == digit][:1000]
    digits = np.transpose(digits, (1, 2, 0))

    A = digits.reshape(28, 28, -1)

    # izvedi HOSVD i rang odaberemo na onoliko koliko imamo baznih
    matrica i ovisno koliko su slike velikih dimenzija
    core, factors = tucker(A, rank=[28, 28, k])  #

    # lista baznih matrica
    A_list = []
    for i in range(k):
        S = np.zeros((28, 28, k))
        S[:, :, i] = core[:, :, i]
        A = tl.tucker_to_tensor((S, factors))
        # pretvori u matricu tako da odaberemo i-tu matricu u modu 3
        A = A[:, :, i]
        A_list.append(A)
    return A_list
```

Također, definiramo funkciju *calculate_error*() koja izračunava koeficijente znamenke i traži najmanju grešku s baznim matricama. Funkcija je vrlo jednostavna, a možemo ju definirati na sljedeći način:

```
def calculate_error(Z, A_list):
    # kopiraj originalni tenzor
    error = Z.copy().astype(np.float64)
    # prodi po cijeloj listi i izracunaj koeficijente i gresku
    for A in A_list:
        z = np.sum(Z * A) / np.sum(A * A)
        error -= z * A

    error = np.sqrt(np.sum(error * error))

    return error
```

Sada koristimo ove dvije funkcije kako bismo za svaku znamenku izračunali njegove bazne matrice i izračunali grešku za svaku od znamenaka kako bi odredili koju znamenku nepoznata slika predstavlja.

```
def accuracy():
    # bazne matrice za svaku znamenku
```

```
    base_matrices_list = [get_base_matrices(dig) for dig in range(10)]

    # Traverse the test set
    n = len(test_labels)
    estimate = np.zeros(n)
    error = [0] * 10

    for i in range(n):
        # slika Z iz testnog seta podataka
        Z = test_images[i]

        for j in range(10):
            error[j] = calculate_error(Z, base_matrices_list[j])

        # Classification of the image 'Z'
        estimate[i] = np.argmin(error)

    # Overall accuracy of the algorithm
    res = np.column_stack((test_labels, estimate))
    accuracy = np.sum(res[:, 0] == res[:, 1]) / n * 100
    print(accuracy)
```

Deklariramo niz *error* duljine 10 koji predstavlja znamenke od 0 do 9 te je na početku inicijaliziramo na 0. Zatim prolazimo po slikama iz testnog seta podataka i za svaku sliku izračunavamo grešku svih znamenki i spremamo tu grešku u niz *error*. Na kraju, proglašavamo da je znamenka ona kod koje je dosegnuta najmanja greška.

4.2 Klasifikacija znamenki pomoću tensor train algoritma

Sada slijedi opis algoritma pomoću kojeg ćemo klasificirati rukom pisane znamenke. Svaka slika je dimenzija 28×28. Kako bi mogli pomoću *tensor train* algoritma klasificirati isti set podataka, moramo prvo pametno odabrati broj dimenzija te preoblikovati originalni tenzor nad kojim ćemo trenirati podatke. Umjesto da imamo tenzor $\mathcal{A} \in \mathbb{R}^{28\times 28\times n}$, gdje je n broj slika, razvući ćemo broj piksela u jednu dimenziju te ćemo svaku znamenku pisati u drugoj dimenziji. Odnosno, imat ćemo tenzor $\mathcal{A} \in \mathbb{R}^{784\times n_e\times n_d}$ pri čemu n_e označava broj slika pojedine znamenke, a n_d označava broj znamenaka, odnosno on je jednak 10 u našem slučaju jer postoji 10 znamenaka. Za zadanu sliku $z \in \mathbb{R}^{n_i}$ nepoznate znamenke, želimo odrediti kojoj od znamenaka je taj vektor najbliži. Ovdje većinom koristimo oznake slične kao u radu [2] gdje je napravljena klasifikacija osoba.

Koristeći jednakost 3.10 i jer je tenzor $\mathcal{A}$ reda 3, matricu $\mathcal{A}(:, e, :)$ možemo zapisati kao:

$$\mathcal{A}(:, e, :) = \mathbf{G}_2^e \times_i \mathbf{G}_1 \times_d \mathbf{G}_3^T,$$

pri čemu je $\mathbf{G}_2^e = \mathcal{G}(:, e, :)$ matrica. Time dobijemo da je e-ta znamenka p jednaka

$$\mathcal{A}(:, e, p) = \mathbf{G}_1 \mathbf{G}_2^e g_3^p,$$

gdje je $g_3^p = \mathbf{G}(:, p)$. Odnosno, ako stavimo da je $\mathcal{C} = \mathcal{G}_2 \times_i \mathbf{G}_1$ onda dobijemo

$$\mathcal{A}(:, e, p) = \mathbf{C}^e g_3^p,$$

pri čemu je $\mathbf{C}^e = \mathcal{C}(:, e, :) = \mathbf{G}_1 \mathbf{G}_2^e$.

Ovdje, g_3^p predstavlja p-ti stupac od $\mathbf{G}_3$ te sadržava koordinate znamenke p u toj bazi.

Za nepoznatu sliku z nepoznate znamenke, želimo odrediti koordinate α_e od z u svakoj od n_e baza $\{\mathbf{C}\}_{e=1,\ldots,n_e}$ i usporediti svaki α_e za $e = 1, \ldots, n_e$ s koordinatama od n_p znamenaka u istoj bazi koje su predstavljane stupcima od $\mathbf{G}_3$. Odnosno, za svaki $e = 1, \ldots, n_e$ računamo:

$$\min_{\alpha_e} \|\mathbf{C}^e \alpha_e - z\|_2,$$

te onda i za svaki $p = 1, \ldots, n_p$ računamo

$$D_{TT}(e.p) := \|\hat{\alpha}_e - g_3^p\|_2.$$

Udaljenost između z i znamenke p je tada dana jednadžbom

$$dist(z, \mathcal{A}(:, :, p)) = \min_e D_{TT}(e, p).$$

Kako bi ovaj algoritam implementirali u Pythonu, potrebno je prvo dobiti tenzor $\mathcal{A}$, a to možemo napraviti na sljedeći način:

```
(train_images, train_labels), (test_images, test_labels) = mnist.
    load_data()

train_images = train_images.T
train_images = train_images.reshape(784, -1)
print(train_images[:, train_labels == 0][:,2].shape)
ne = 1000
num_digits = 10

tensor_digits = np.zeros((784, ne, 10))
for d in range(10):
    for i in range(ne):
        tensor_digits[:,i,d] = train_images[:, train_labels == d][:,i]

G1, G2, G3 = tensor_train(tensor_digits, rank=[1, 10, 100, 1])
G1 = np.squeeze(G1)
G3 = np.squeeze(G3)
```

Ovim kodom dobijemo tenzor $\mathcal{A}$ dimenzija $n_i \times n_e \times 10$. Na početku, *training_images* preoblikujemo tako da spojimo dimenzije slika u jednu te kasnije punimo *tensor_digits* varijablu kako bi dobili traženi tenzor.

Sada možemo implementirati algoritam koji će za zadani ulaz z koja predstavlja nepoznatu sliku i za jezgrene tenzore $\mathbf{G}_1$, $\mathcal{G}_2$ te $\mathbf{G}_3$, klasificirati sliku z kao znamenku p primjenjujući postupak koji smo prethodno opisali. Takav algoritam je još poznat i kao $TT3D$ algoritam jer se primjenjuje za tenzore reda 3.

```
def tt3d(z, G1, G2, G3):
    z_prime = np.dot(G1.T, z)
    D_TT = np.full((ne, num_digits), np.inf)  # initialize array with
    infinity values
    for e in range(ne):
        alpha_e = np.linalg.solve(G2[:, e, :], z_prime)
        for p in range(num_digits):
            g3p = G3[:, p]
            D_TT[e, p] = np.linalg.norm(alpha_e - g3p)

    # find the minimum
    e_prime, p_prime = np.unravel_index(np.argmin(D_TT, axis=None), D_TT
    .shape)

    # classify z as digit p'
    return p_prime
```

Na početku računamo $\hat{z} = \mathbf{G}_1^T z$ te za svaki *ne* riješavamo linearnu jednadžbu $\mathbf{G}_2^e \alpha_e = \hat{z}$ za α_e. Nakon toga, za svaku znamenku računamo udaljenost, odnosno računamo $\|\alpha_e - g_3^p\|_2$ pri čemu je $g_3^p = \mathbf{G}_3(:, p)$. Na kraju svega toga, moramo pronaći minimum u D_{TT}, odnosno tražimo $argmin_{e,p}(D_{TT})$. Kako bi to pronašli, na početku funkcije je potrebno deklarirati dvodimenzionalno polje koje na početku sadrži beskonačne vrijednosti. Znamenku p' vraćamo kao rezultat. Na sličan način izračunamo točnost ovakvog algoritma.

```
def acc2():
    n = len(test_labels)
    estimate = np.zeros(n)
    for i in range(n):
        # image 'Z' from the test set
        Z = test_images[i].swapaxes(0,1).reshape(-1)

        # classification of the image 'Z'
        estimate[i] = tt3d(Z, G1, G2, G3)

    # overall accuracy of the algorithm
    res = np.column_stack((test_labels, estimate))
    accuracy = np.sum(res[:, 0] == res[:, 1]) / n * 100
    print(accuracy)
```

Pozivom funkcije *acc*2 ispisujemo točnost algoritma.

4.3 Rezultati

U ovom dijelu poglavlja, odradit ćemo evaluaciju točnosti ova dva algoritma nad istim setom podataka. Za početak ćemo obraditi točnost HOSVD algoritma. Za određivanje točnosti smo koristili skup od 10000 testnih slika znamenaka. Ako promjenimo varijablu k u algoritmu, tada će se i promjeniti broj baznih matrica koje uzimamo u funkciji *get_base_matrices* za svaku pojedinu znamenku te samim time, točnost će biti veća ili manja.

k	2	5	10	15	20
Točnost	86.71%	91.7%	94.38%	94.67%	95.18%

Tablica 4.1: Točnost HOSVD algoritma

Povećanjem baznih matrica s $k = 15$ na $k = 20$ nismo dobili preveliko povećanje točnosti a vrijeme izvršavanja bi se produžilo. Također, već sa $k = 2$ dobivamo dosta veliku točnost od 86.71%.

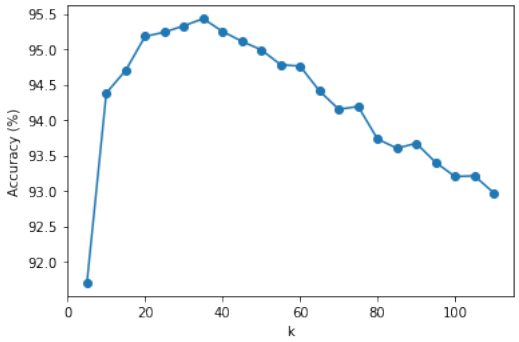
Slično tako, u *tensor train* algoritmu za klasifikaciju možemo mijenjati varijablu *ne* koja uvjetuje koliko prolazimo u for petlji u *tt3d* funkciji te se ujedno koristi i u stvaranju tenzora nad kojim vršimo *tensor train* dekompoziciju. Samim time će se i dimenzije tenzora $\mathbf{G}_1$, $\mathcal{G}_2$ i $\mathbf{G}_3$ razlikovati.

ne	100	500	1000	2000	4000
Točnost	65.77%	70.87%	80.88%	83.17%	85.98%

Tablica 4.2: Točnost *tensor train* algoritma

Odmah ćemo primjetiti da je točnost pomoću HOSVD-a puno bolja nego pomoću *tensor train*. Najveći skok u točnosti se dogodi s $ne = 500$ na $ne = 1000$ gdje je algoritam skoro čak i za 10% točniji.

Vidimo da točnost u HOSVD algoritmu raste kako raste i varijabla k, odnosno broj baznih matrica. No, ako napravimo graf koji prikazuje odnos točnosti i broj baznih matrica primijetit ćemo da situacija nije takva. Algoritam najbolje radi kada postavimo $k \approx 38$ te ako je k iznad 40 algoritam će raditi sa sve manjom točnošću. Razlog zbog čega dobivamo sve manju točnost je zato što povećanjem baznih matrica zahtijevamo sve veću podudaranost slika s baznim matricama. S druge strane, varijablu *ne* možemo povećati onoliko puta koliko imamo slika pojedinih znamenaka. U ovom slučaju, svaka znamenka ima otprilike 6000 slika, ali povećavanjem varijable *ne* ne bi dobili drastičan skok u točnosti.



Slika 4.2: Graf točnosti HOSVD algoritma s obzirom na k

Također, kao što smo spomenuli i na početku ovog poglavlja, nismo očekivali točnost od 100% jer su neke znamenke čudno zapisane. Tako na primjer, znamenku 6, algoritmi mogu prepoznati kao znamenku 0 ili obratno, ako su znamenke napisano loše. Na slici 4.3 vidimo neke od znamenaka koje su algoritmi krivo prepoznali. Očekivano je neke znamenke kao što je 6 ili 9 prepoznao kao 0 jer sve te znamenke izgledaju slično ako je rukopis nepregledan. Također, znamenke 7 i 1 izgledaju slično u dosta slučajeva pa je očekivano da će algoritmi ponekad i takve slučajeve krivo klasificirati.

Uzmimo sada $ne = 1000$ te $k = 10$. Kao što je prije spomenuto, neke znamenke će biti češće krivo klasificirane od drugih. Na primjer, znamenka jedinice je jednostavno za zapisati dok na primjer znamenku osmice je ipak malo kompliciranije zapisati. Samim time će algoritmi teže klasificirati takve znamenke pa će točnost cjelokupne baze podataka za testiranje biti manja.

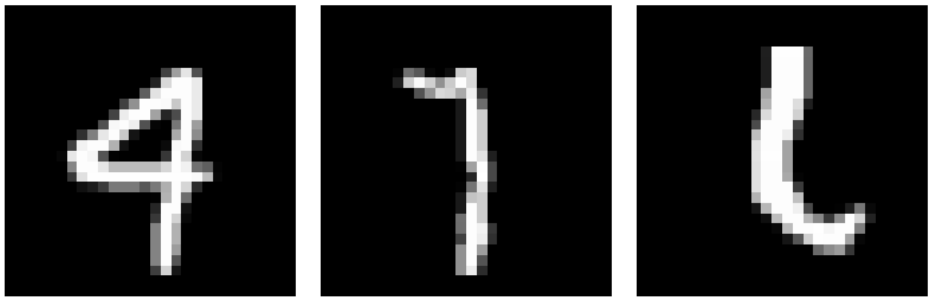
0	1	2	3	4	5	6	7	8	9
98.4%	99.5%	92.5%	93.6%	94.7%	92%	96.1%	91.2%	91.9%	92.6%

Tablica 4.3: Točnost pojedinih znamenaka HOSVD algoritma s $k = 10$

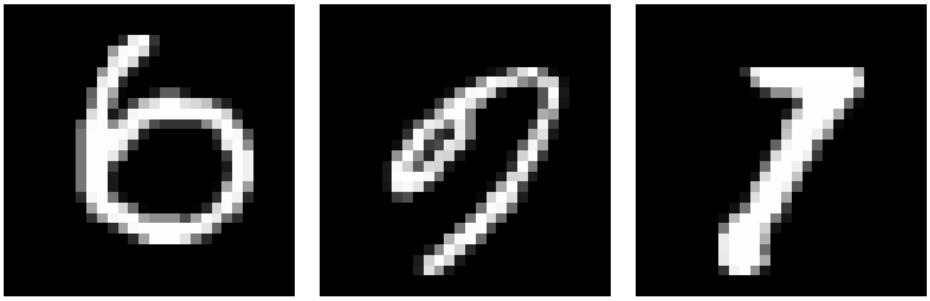
0	1	2	3	4	5	6	7	8	9
88.3%	99.6%	81.3%	75.3%	73.9%	77.3%	88.4%	83.4%	70%	68.9%

Tablica 4.4: Točnost pojedinih znamenaka *tensor train* algoritma s $ne = 1000$

Ako bolje promotrimo točnost algoritma na pojedinim znamenkama odmah uočavamo da je zaista znamenka osmice među najteže prepoznatim znamenkama dok je znamenka jedinice prepoznata u više od 99% slučajeva što je izuzetno visoka točnost. U *tensor train*



(a) 4 prepoznao kao 9 (b) 7 prepoznao kao 1 (c) 6 prepoznao kao 1



(d) 6 prepoznao kao 0 (e) 9 prepoznao kao 0 (f) 7 prepoznao kao 1

Slika 4.3: Neke od pogrešno prepoznatih znamenaka

algoritmu, znamenka devetke ima najmanji postotak točnosti. Razlog tomu je vjerojatno što algoritam pomiješa znamenku devetke s znamenkom nule jer u više slučajeva znaju izgledati identično.

Bibliografija

[1] *Tensorly biblioteka*, `http://tensorly.org/stable/index.html`, lipanj 2023.

[2] D. Brandoni, *Tensor decompositions for Face Recognition*, hal-02196526 (2017).

[3] D. Brandoni i V. Simoncini, *TENSOR-TRAIN DECOMPOSITION FOR IMAGE RECOGNITION*, hal-02196526f (2019).

[4] L. De Lathauwer, B. De Moor i J. Vandewalle, *A MULTILINEAR SINGULAR VALUE DECOMPOSITION*, SIAM Journal on Matrix Analysis and Applications (2000).

[5] Z. Drmač, *Numerička Analiza 1*, PMF Odsjek za matematiku (2009).

[6] P. Gelß, *The Tensor-Train Format and Its Applications*, Fachbereich Mathematik und Informatik der Freien Universität Berlin (2019).

[7] T. G. Kolda, *Multilinear operators for higher-order decompositions*, Society for Industrial and Applied Mathematics (2006).

[8] I. Oseledets, *Tensor train decomposition*, SIAM Journal on Scientific Computing (2011).

[9] D. Pavlović i A. Novak, *Dekompozicija matrice na singularne vrijednosti i primjene*, Hrvatski matematički elektronički časopis (lipanj 2013).

[10] S. Poštić, *Tenzorske metode u prepoznavanju lica*, PMF Odsjek za matematiku (2018).

[11] D. Tock, *Tensor Decomposition and its Applications*, University of Chester (2010).

Sažetak

U ovom radu su opisani pojmovi i operacije usko vezane uz multilinearnu algebru i objekte koje nazivamo tenzori. Opisane su osnovne operacije koje provodimo nad tenzorima te opisujemo matričnu dekompoziciju SVD te taj koncept proširujemo na tenzore. Zatim, promatramo još jednu tenzorsku dekompoziciju pod nazivom *tensor train* dekompozicija koja ne ovisi toliko o redu tenzora koliko HOSVD ovisi.

Dekompozicija tenzora pomoću HOSVD-a i *tensor train*-a su također implementirane u ovom radu kako bi se njihov algoritam što bolje razumio. Oba algoritma zasnivana su na dokazima tvrdnji koje daju konstruktivan način njihove egzistencije. Pomoću tih algoritama možemo tenzore aproksimirati vrlo dobro. Samim time i zbog toga što slike možemo također reprezentirati pomoću tenzora, koristimo ta dva algoritma kako bi aproksimirali slike.

Rješavamo problem klasifikacije rukom pisanih znamenaka koristeći ove dvije dekompozicije pri čemu su se tenzori pokazali izuzetno korisnim. Prvi način rješavanja problema klasifikacije znamenaka smo riješili pomoću HOSVD metode. Zatim, isti taj problem rješavamo pomoću *tensor train* dekompozicije. Oba riješenja pokazala su se efikasnima te su sve implementacije rađene u programskom jeziku Python. Ipak, pomoću HOSVD algoritma smo dobili malo veću preciznost nego s *tensor train* algoritmom. Također, biramo različite parametre kako bi dostigli različite točnosti. Bitno je vrlo pažljivo birati takve parametre kako bi dostigli što veću preciznost algoritma jer ponekad nije najbolji pristup birati što veće parametre kao što smo to vidjeli na primjeru klasifikacije uz pomoć HOSVD algoritma. Na kraju rada također možemo vidjeti koje znamenke su algoritmi krivo pogodili. Očekivano, neke jednostavnije znamenke imaju veću točnost pogađanja od drugih.

Summary

This paper describes concepts and operations closely related to multilinear algebra and objects known as tensors. It explains basic operations performed on tensors and extends the concept of matrix decomposition through SVD to tensors. Additionally, it explores another tensor decomposition method called *tensor train* decomposition, which is less dependent on tensor rank compared to HOSVD.

Tensor decompositions using HOSVD and *tensor train* are implemented in this study to better understand their algorithms. Both algorithms are based on proofs that provide constructive ways of obtaining them. These algorithms are effective for approximating tensors, especially given that images can also be represented as tensors. Therefore, both methods are used to approximate images.

The problem of classifying handwritten digits is addressed using these two tensor decompositions, and tensors have proven to be exceptionally useful in this context. The first approach to digit classification is solved using the HOSVD method. Subsequently, the same problem is tackled using *tensor train* decomposition. Both solutions are efficient, and all implementations are carried out in the Python programming language. However, the HOSVD algorithm yields slightly higher accuracy compared to the *tensor train* algorithm. Additionally, different parameters are chosen to achieve varying levels of accuracy. Careful parameter selection is essential for achieving high algorithm accuracy, as demonstrated in the digit classification example using the HOSVD algorithm. Finally, the paper presents the misclassified digits, where it is expected that simpler digits have higher accuracy in classification than others.

Životopis

Rođen sam 24. listopada 1998. godine u Zagrebu. Pohađao sam X. gimnaziju "Ivan Supek" na dvojezičnom programu nakon čega upisujem PMF Matematiku. Svoje obrazovanje nastavljam na PMF-u upisavši smjer računarstva i matematike na diplomskom studiju.