Dinamička modalna dekompozicija u analizi video zapisa

Karlić, Luka

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Luka Karlić

DINAMIČKA MODALNA DEKOMPOZICIJA U ANALIZI VIDEOZAPISA

Diplomski rad

Voditelj rada:
prof. dr. sc. Zlatko Drmač

Zagreb, Srpanj, 2024.

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. _____

Mojoj obitelji i prijateljima. Bili ste mi vječna podrška i najveći obožavatelji, te ste mi učinili fakultetsko razdoblje najljepšim. Iako sam na PMF-u stekao mnoge prijatelje za život, htio bih posebnu zahvalu dati Karlu koji je vjerovao u mene kada sam i sam posumnjao. Dodatno, želim se zahvaliti doc. dr. sc. Andreju Novaku koji mi je pojasnio što uistinu znači studirati matematiku i pomogao donijeti najbolju odluku - upisati studij matematike. Na kraju, zahvale prof. dr. sc. Zlatku Drmaču na pomoći tijekom pisanja rada.

Sadržaj

Uvod

U kontekstu videonadzora javlja se sve veća potreba za metodama koje su sposobne u stvarnom vremenu obrađivati videozapise razdvajajući prvi plan od pozadine. To igra ključnu ulogu u raznim primjenama poput detekcije objekata, praćenja te prepoznavanja. Tradicionalni pristupi, primjerice robustni PCA (RPCA), već se dugo koriste u te svrhe, no suviše su računalno intenzivni. Samo jedna od mnogih primjena dinamičke modalne dekompzicije (DMD) nudi rješenje za ovaj izazov. Iako je izvorno razvijen u analizi nelinearnih dinamičkih sustava, DMD se može upotrijebiti za razlaganje složenih sustava na sastavne komponente. Ovaj rad istražuje primjenu DMD-a na učinkovitu separaciju videozapisa na pozadinu i prvi plan, pritom znatno smanjujući računalne resurse u odnosu na RPCA.

U prvom poglavlju pružamo kratak matematički pregled osnovnih pojmova i definicija s kojima ćemo se susretati kako bismo se podsjetili najvažnijih rezultata.

Drugo poglavlje donosi osnovnu definiciju dinamičke modalne dekompozicije, povezuje ju s Koopmanovim operatorom te prikazuje djelovanje na motivacijskom primjeru u kojem razdvajamo prostorno-vremenski signal na sastavne komponente.

U trećem poglavlju motivirani RPCA algoritmom najavljujemo kako bismo mogli upotrijebiti DMD algoritam u analizi videozapisa, točnije pri separaciji na pozadinu i prvi plan. Međutim, uočavamo i neke probleme u inicijalnoj ideji, stoga predstavljamo novi pristup koji ćemo opravdati nešto kasnije u radu.

Četvrtim poglavljem uvodimo poboljšanje DMD algoritma - sažeti DMD (eng. *compressed DMD*). Uspoređujemo ga s osnovnim DMD-om naglašavajući da ćemo se u ostatku rada baviti isključivo sažetim jer je znatno brži uz minimalnu razliku u točnosti. Nadalje, u tom poglavlju ukratko predstavljamo teoriju sažetog uzorkovanja matrice (eng. *compressed sensing*) i metodu skiciranja matrice (eng. *matrix sketching*).

Vrhunac rada predstavljen je u petom poglavlju gdje evaluiramo metodu sažetog DMD-a na mnoštvu javno dostupnih skupova podataka. Osim toga, pružamo uvid u parametre korištenog modela te navodimo korake postprocesuiranja slike. Uz to, navedeni su i brojčani rezultati četiriju korištenih metrika.

Poglavlje 1

Matematički pregled

Za početak ćemo navesti pregled osnovnih definicija i teorema kako bismo olakšali čitatelju razumijevanje. Većina materijala preuzeta je iz [2] i [9].

1.1 Osnovne definicije

Svojstvene vrijednosti i vektori mogu se definirati na razini linearnog operatora na konačno dimenzionalnom prostoru $\mathbb{R}^n$ ($\mathbb{C}^n$), no u praksi se najčešće koriste na nivou matrice. Naime, svakom linearnom operatoru u bilo kojem paru baza pripada matrica, stoga se problem svojstvenih vrijednosti operatora može reducirati na problem svojstvenih vrijednosti matrica.

Definicija 1.1.1 (**Svojstveni vektor**). *Neka je* $\mathbf{A} \in \mathbb{C}^{n\times n}$. *Vektor* $\mathbf{v} \in \mathbb{C}^n, \mathbf{v} \neq 0$, *je* ***svojstveni vektor*** *od* $\mathbf{A}$ *ako postoji skalar* λ *takav da je*

$$\mathbf{A}v = \lambda v. \tag{1.1}$$

U tom slučaju λ *je* ***svojstvena vrijednost****, a par* (x, λ) *je* ***svojstveni par*** *matrice* $\mathbf{A}$.

Vektorske i matrične norme osnovno su sredstvo koje se koristi kod ocjene grešaka numeričkih metoda, posebice u numeričkoj linearnoj algebri [9].

Definicija 1.1.2 (**Vektorska norma**). *Vektorska norma je svaka funkcija* $\|\cdot\| : \mathbb{C}^n \to \mathbb{R}$ *koja zadovoljava sljedeća svojstva:*

1. $\|x\| \geq 0, \forall x \in \mathbb{C}^n$, *a jednakost vrijedi ako i samo ako je* $x = 0$,
2. $\|\alpha x\| = |\alpha|\|x\|, \forall \alpha \in \mathbb{R}, \forall x \in \mathbb{C}^n$,

3. $||x+y|| \leq ||x||+||y||, \forall x, y \in \mathbb{C}^n$. *Ova je nejednakost poznatija pod imenom nejednakost trokuta (zbroj duljina bilo koje dvije stranice trokuta veći je od duljine treće stranice).*

Kada zamijenimo vektor $x \in \mathbb{C}^n$ matricom $A \in \mathbb{C}^{m\times n}$, dobijemo matričnu normu.

Definicija 1.1.3 (**Matrična norma**). *Matrična norma je svaka funkcija* $\|\cdot\| : \mathbb{C}^{m\times n} \to \mathbb{R}$ *koja zadovoljava sljedeća svojstva:*

1. $||A|| \geq 0, \forall A \in \mathbb{C}^{m\times n}$, *a jednakost vrijedi ako i samo ako je* $A = 0$,
2. $||\alpha A|| = |\alpha|||A||, \forall \alpha \in \mathbb{R}, \forall A \in \mathbb{C}^{m\times n}$,
3. $||A + B|| \leq ||A|| + ||B||, \forall A, B \in \mathbb{C}^{m\times n}$.

   Za matričnu normu kažemo da je konzistentna ako vrijedi
4. $||AB|| \leq ||A||\,||B||$

kad god je matrični produkt AB definiran.

Matrične norme mogu nastati na dva različita načina. Ako matricu **A** promatramo kao vektor s $m\times n$ elemenata, onda direktna primjena vektorskih normi daje sljedeće definicije:

1. ℓ_1 norma
$$|||A|||_1 := ||A||_S = \sum_{i=1}^{m}\sum_{j=1}^{n} \left|a_{ij}\right|,$$
2. ℓ_2 norma (euklidska, **Frobeniusova**, Hilbert–Schmidtova, Schurova)
$$|||A|||_2 := ||A||_F = (\mathrm{tr}\,(A^*A))^{1/2} = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} \left|a_{ij}\right|^2},$$
3. ℓ_∞ norma
$$|||A|||_\infty := ||A||_M = \max_{\substack{i=1,\ldots,m\\ j=1,\ldots,n}} \left|a_{ij}\right|.$$

S druge strane, matrične norme možemo dobiti kao **operatorske norme** iz odgovarajućih vektorskih normi korištenjem definicije

$$||A|| = \max_{x\neq 0} \frac{||Ax||}{||x||} \left(\text{ili } = \max_{||x||=1} ||Ax|| \right). \tag{1.2}$$

Uvrštavanjem odgovarajućih vektorskih normi u 1.2 slijedi:

1. matrična 1-norma, tzv. "maksimalna stupčana norma"
$$\|A\|_1 = \max_{j=1,\dots,n} \sum_{i=1}^{m} \left|a_{ij}\right|,$$
2. matrična 2-norma, spektralna norma
$$\|A\|_2 = \left(\rho\left(A^* A\right)\right)^{1/2} = \sigma_{\max}(A),$$
3. matrična ∞-norma, tzv. "maksimalna retčana norma"
$$\|A\|_\infty = \max_{i=1,\dots,m} \sum_{j=1}^{n} \left|a_{ij}\right|,$$

pri čemu je ρ oznaka za spektralni radijus kvadratne matrice (svojstvena vrijednost maksimalna po apsolutnoj vrijednosti)
$$\rho(B) = \max\{|\lambda| \mid \det(B - \lambda I) = 0\}, \quad (B \text{ kvadratna!}), \tag{1.3}$$
a σ je standardna oznaka za tzv. singularnu vrijednost matrice, koju ćemo uskoro upoznati.

Teorem 1.1.4 (Moore-Penroseov pseudoinverz)**.** *Neka je $A \in \mathbb{C}^{m\times n}$. Matricu G koja zadovoljava uvjete (P1)-(P4) zovemo generalizirani inverz od A.*
$$\begin{aligned} &(P1) \quad AGA = A \\ &(P2) \quad GAG = G \\ &(P3) \quad (AG)^T = AG \\ &(P4) \quad (GA)^T = GA \end{aligned}$$
Takav G uvijek postoji i jedinstven je, a označava se s $G = A^\dagger$.

Gornji uvjeti (P1)-(P4) zovu se *Penroseovi uvjeti*. Navedimo još neka korisna svojstva Moore-Penroseovog pseudoinverza:

1. $\left(A^\dagger\right)^\dagger = A$
2. $\left(A^T A\right)^\dagger = A^\dagger \left(A^T\right)^\dagger, \left(AA^T\right)^\dagger = \left(A^T\right)^\dagger A^\dagger$
3. $\mathcal{R}\left(A^\dagger\right) = \mathcal{R}\left(A^T\right) = \mathcal{R}\left(A^\dagger A\right) = \mathcal{R}\left(A^T A\right)$
4. $\mathcal{N}\left(A^\dagger\right) = \mathcal{N}\left(AA^\dagger\right) = \mathcal{N}\left(\left(AA^T\right)^\dagger\right) = \mathcal{N}\left(AA^T\right) = \mathcal{N}\left(A^T\right)$

Napomena 1.1.5. *U slučaju regularne matrice očito vrijedi $A^{-1} = A^\dagger$*

Problemi koje ćemo promatrati u ostatku ovog rada bavit će se realnim matricama, stoga ćemo od sada samo promatrati realne matrice, osim ako eksplicitno ne bude navedeno drukčije. Dakle, prethodni teorem nam kaže da za bilokoju matricu $A \in \mathbb{R}^{m\times n}$ pseudoinverz, označen s $A^\dagger \in \mathbb{R}^{n\times m}$, postoji i jedinstven je.

1.2 Matrična formulacija problema najmanjih kvadrata

Učestala situacija u matematici jest da imamo skup mjerenih podataka $(t_k, y_k)\,, k = 1, \ldots, n$, te želimo taj model aproksimirati funkcijom oblika $\varphi(t)$. Ako je $\varphi(t)$ opća linearna funkcija, tj. ako je

$$\varphi(t) = x_1\varphi_1(t) + \cdots + x_m\varphi_m(t),$$

gdje su $\phi_0, \cdots, \phi_m$ poznate (zadane) funkcije, onda bismo htjeli pronaći parametre x_j tako da mjereni podaci (t_k, y_k) zadovoljavaju

$$y_k = \sum_{j=1}^{m} x_j\varphi_j\,(t_k)\,, \quad k = 1, \ldots, n.$$

Označimo li

$$a_{kj} = \varphi_j\,(t_k)\,, \quad b_k = y_k,$$

prethodne jednadžbe možemo elegantnije zapisati u matričnom obliku:

$$Ax = b.$$

Budući da su matrica A i vektor b poznati, preostaje odrediti vektor x. Ako je A **kvadratna, invertibilna** matrica, onda postoji **jedinstveno** rješenje x za svaki b. Međutim, ako je A **singularna** ili **pravokutna** matrica, onda može postojati *jedno, niti jedno ili beskonačno mnogo* rješenja, što ovisi o b te prostorima koje razapinju stupci i retci matrice A.

Ako je mjerenih podataka više nego parametara, tj. ako je $n > m$, onda ovaj sustav ima više jednadžbi nego nepoznanica pa je *preodređen*. U ovom radu susretat ćemo se isključivo sa situacijom kada je $n \gg m$, odnosno kada je matrica A "visoka i mršava" (eng. *"tall-skinny matrix"*)

Iako postoji mnogo različitih načina da se odredi "najbolje" rješenje, iz statističkih razloga često odabir padne na metodu najmanjih kvadrata, tj. određujemo x tako da minimizira grešku $r = Ax - b$ (r često zovemo rezidual)

$$\min_x \|r\|_2 = \min_x \|Ax - b\|_2, \quad A \in \mathbb{R}^{n\times m}, \quad b \in \mathbb{R}^n, \quad x \in \mathbb{R}^m. \tag{1.4}$$

Ukoliko je $\text{rang}(A) < m$, onda rješenje x ovog problema ne može biti jedinstveno. Naime, A ima netrivijalan nul-potprostor $\mathcal{N}(A)$, stoga rješenju možemo dodati proizvoljan vektor iz $\mathcal{N}(A)$ bez da se rezidual promijeni.

Neka je $\hat{x}$ jedno od rješenja. Skup svih rješenja $\mathcal{S}$ možemo opisati na sljedeći način:

$$\mathcal{S} = \{x = \hat{x} + z \mid z \in \mathcal{N}(A)\}.$$

S druge strane, među svim rješenjima x problema najmanjih kvadrata uvijek postoji jedinstveno rješenje x najmanje norme, tj. koje još minimizira i $\|x\|_2$.
Točnije, ako je $\hat{x} \perp \mathcal{N}(A)$, onda je

$$\|x\|_2^2 = \|\hat{x}\|_2^2 + \|z\|_2^2$$

pa je $\hat{x}$ jedinstveno rješenje problema najmanjih kvadrata koje ima minimalnu euklidsku normu.

1.3 Matrične dekompozicije

Singularna dekompozicija matrice

Dekompozicija singularnih vrijednosti (eng. "singular value decomposition") ili, skraćeno, SVD, zasigurno spada u jednu od **najkorisnijih dekompozicija** i s teoretske strane (pri dokazima) i s praktične strane.
Sljedeći teorem o egzistenciji singularne dekompozicije može poslužiti i kao definicija singularnih vrijednosti i vektora matrice.

Teorem 1.3.1 (Singularna dekompozicija matrice)**.** *Neka je $A \in \mathbb{C}^{n\times m}$. Tada postoje unitarne matrice $U \in \mathbb{C}^{n\times n}$ i $V \in \mathbb{C}^{m\times m}$ takve da je*

$$U^*AV = \Sigma, \quad \Sigma = \operatorname{diag}\left(\sigma_1, \sigma_2, \ldots, \sigma_{\min\{m,n\}}\right), \tag{1.5}$$

pri čemu vrijedi

$$\sigma_1 \geq \sigma_2 \cdots \geq \sigma_{\min m,n} \geq 0.$$

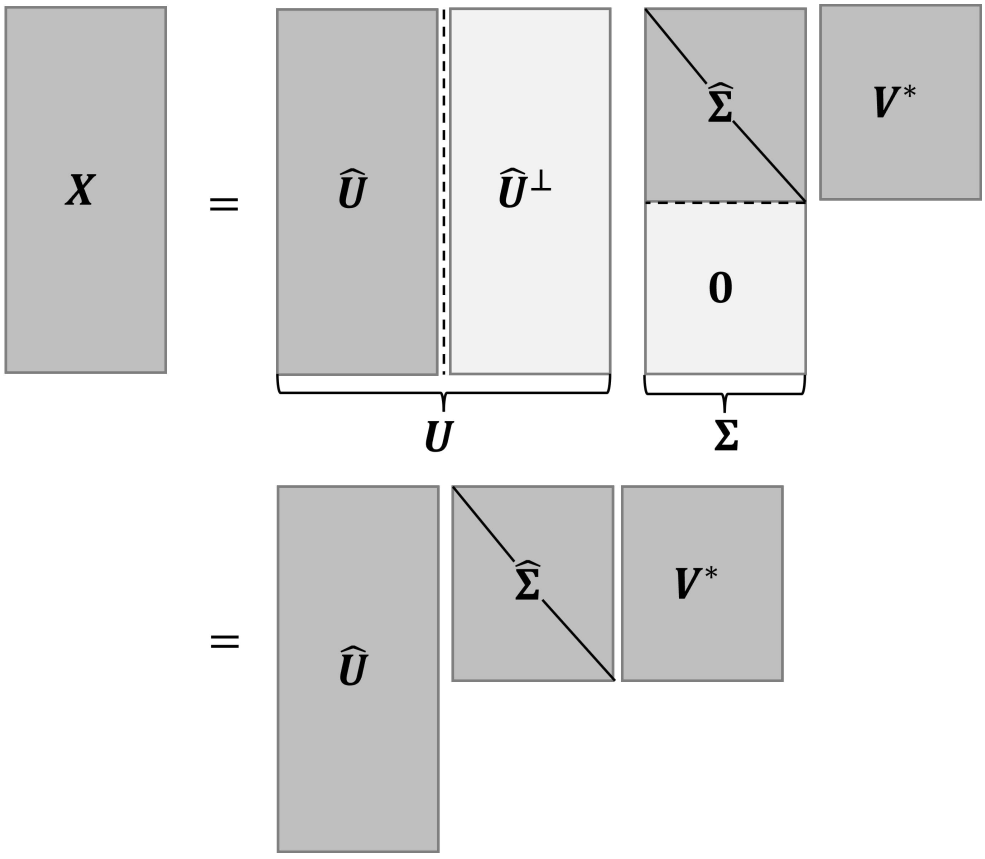
*Brojeve $\sigma_1, \sigma_2, \ldots, \sigma_{\min m,n}$ zovemo **singularne vrijednosti** matrice A. Stupce matrice U zovemo lijevi, a stupce matrice V desni singularni vektori matrice A.*

Dokaz. Vidi dokaz teorema 2.3.1 u [9]. □

U slučaju kada je $n \geq m$, a spomenuli smo da će nam to biti najzanimljiviji slučaj, matrica Σ ima najviše m netrivijalnih elemanata na dijagonali te se može zapisati kao $\Sigma = \begin{bmatrix} \hat{\Sigma} \\ \mathbf{0} \end{bmatrix}$. Broj netrivijalnih elemenata na dijagonali je ujedno i rang matrice A. Uobičajeno je tada A prikazati koristeći tzv. "štedljivi" SVD (eng. *economy SVD*), kojeg možemo vidjeti na slici 1.3:

$$A = U\mathbf{\Sigma}V^* = \begin{bmatrix} \hat{U} & \hat{U}^\perp \end{bmatrix} \begin{bmatrix} \hat{\mathbf{\Sigma}} \\ 0 \end{bmatrix} V^* = \hat{U}\hat{\mathbf{\Sigma}}V^*. \tag{1.6}$$

Jedno od najkorisnijih i najvažnijih svojstava SVD-a je to što matrici A daje optimalnu aproksimaciju nižim rangom r. To svojstvo pokazat ćemo idućim teoremom.



Slika 1.1: Shematski prikaz punog (1. red) i "štedljivog" SVD-a (2. red).

Teorem 1.3.2 (Eckart-Young)**.** *Neka je dana matrica X. Tada vrijedi:*

$$\underset{\tilde{X},\, t.d.\ \mathrm{rang}(\tilde{X})\leq r}{\operatorname{argmin}} \|X - \tilde{X}\|_F = \tilde{U}\tilde{\Sigma}\tilde{V}^*. \tag{1.7}$$

$\tilde{X}$ je optimalna aproksimacija nižim rangom r u smislu najmanjih kvadrata. U ovom slučaju $\tilde{U}$ i $\tilde{V}$ označavaju vodećih r stupaca od U, odnosno V, a $\tilde{\Sigma}$ sadrži vodeći $r \times r$ blok od Σ.

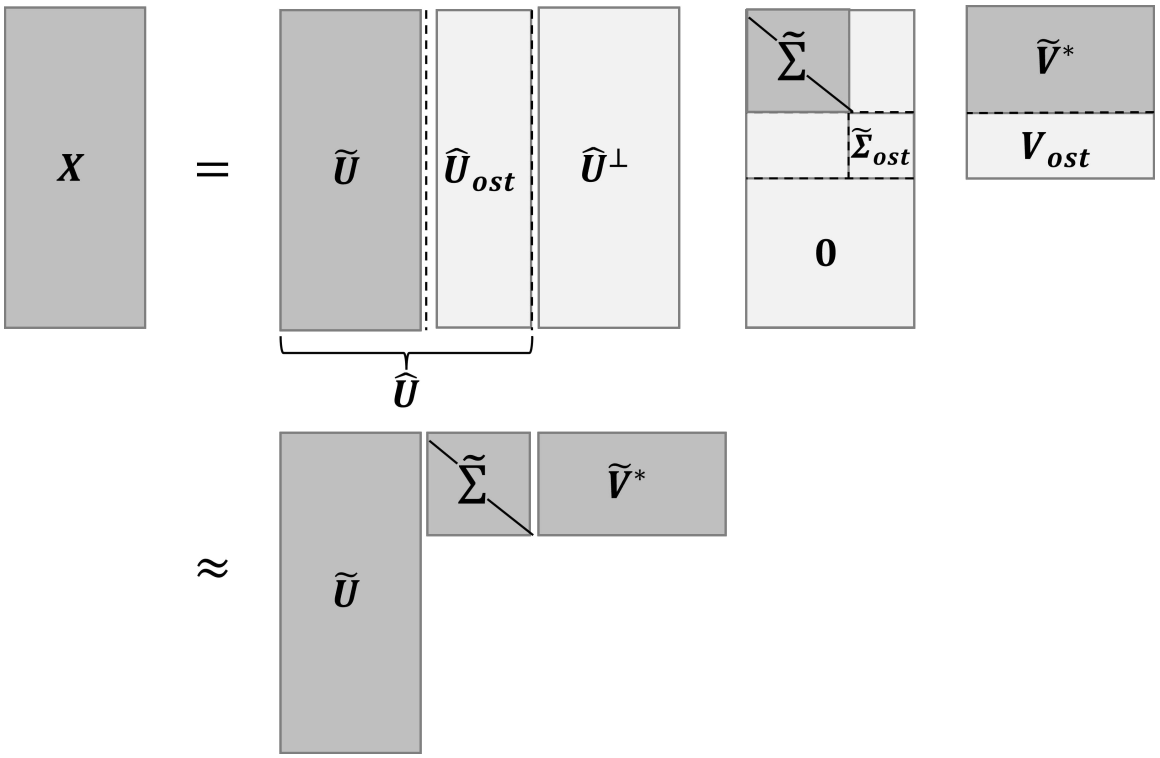
Napomena 1.3.3. *Teorem 1.3.2 vrijedi i za operatorsku 2-normu. Točnije, argmin je isti, no vrijednost minimuma je različita.*

Sada smo ujedno pokazali notaciju koju ćemo koristiti za "skraćeni" (eng. *truncated*) SVD. Rezultirajuću matricu zapisujemo kao $\tilde{X} = \tilde{U}\tilde{\Sigma}\tilde{V}^*$, a vizualno to možemo prikazati

slikom 1.2. Nadalje, s obzirom da je Σ dijagonalna, SVD-aproksimacija rangom r je dana sumom r matrica ranga 1:

$$\tilde{X} = \sum_{k=1}^{r} \sigma_k u_k v_k^* = \sigma_1 u_1 v_1^* + \sigma_2 u_2 v_2^* + \cdots + \sigma_r u_r v_r^*. \tag{1.8}$$

Uočimo, za dani rang r **ne postoji** bolja aproksimacija za X od $\tilde{X}$ u smislu ℓ_2 norme. Posljedično, visoko-dimenzionalni podaci mogu se dobro aprosimirati s "nekoliko" dominantnih uzoraka danih stupcima matrica $\tilde{U}$ i $\tilde{V}$.



Slika 1.2: Shematski prikaz "skraćenog" SVD-a. Indeks *ost* predstavlja "ostatak" nakon kraćenja.

Primjer: sažimanje slike

Ideju matrične aproksimacije najbolje možemo dočarati jednostavnim primjerom sažimanja slike [20]. Slika, u suštini zapravo matrica, prikladan je primjer s obzirom na tematiku os-

tatka ovog rada. Sliku u sivim tonovima (eng. *grayscale image*) možemo promatrati kao realnu matricu $X \in \mathbb{R}^{n\times m}$, gdje su n i m brojevi piksela u vertikalnom i horizontalnom smjeru. Neka nam je dana slika mopsa u gornjem lijevom kutu slike 1.3. Originalna slika sadrži 4032×3024 piksela. Uzet ćemo SVD dekompoziciju te slike, zatim na 1.4 prikazati singularne vrijednosti paralelno s kumulativnom energijom, a u 1.3 usporediti originalnu sliku s aproksimacijom $\tilde{X}$ za $r = 5, 20, 100$, gdje je r parametar redukcije SVD-a, odnosno rang. Uočimo da već za $r = 100$ rekonstruirana slika prilično točno opisuje polaznu sliku, pri čemu te vodeće singularne vrijednosti opisuju gotovo 70% varijance slike, a relativna greška ϵ_{rel} je manja od 5%. Dakle, uspjeli smo značajno sažeti sliku s obzirom da nam je dovoljno spremiti samo 100 vodećih stupaca matrica U i V zajedno sa 100 dijagonalnih elemenata od Σ. Za kraj, uočimo sa slike 1.4 da samo 628 (od ukupno 3024) singularnih vrijednosti opisuje 90% varijance slike.

Prava ortogonalna dekompozicija (POD)

Prava ortogonalna dekompozicija (eng. *Proper Orthogonal Decomposition*) usko je povezana s teoremom 1.3.2. Naime, to je postupak redukcije dimenzije u kojem se konstruira potprostor koji najbolje aproksimira dane vektore u smislu najmanjih kvadrata, što je upravo prethodni teorem 1.3.2.
Ako se vratimo na trenutak na zapis $\tilde{X} = \tilde{U}\tilde{\Sigma}\tilde{V}^*$, gdje je $\tilde{U} = U(:, 1 : r)$, $\tilde{\Sigma} = \Sigma(1 : r, 1 : r)$ te $\tilde{V} = V(:, 1 : r)$, onda uočavamo kako se stupci originalne matrice $\tilde{X}$ mogu zapisati u bazi koju čine stupci matrice $\tilde{U}$. Tu bazu nazivamo **POD** baza i taj pojam često ćemo spominjati u ostatku ovog rada.

Analiza nezavisnih komponenti (ICA)

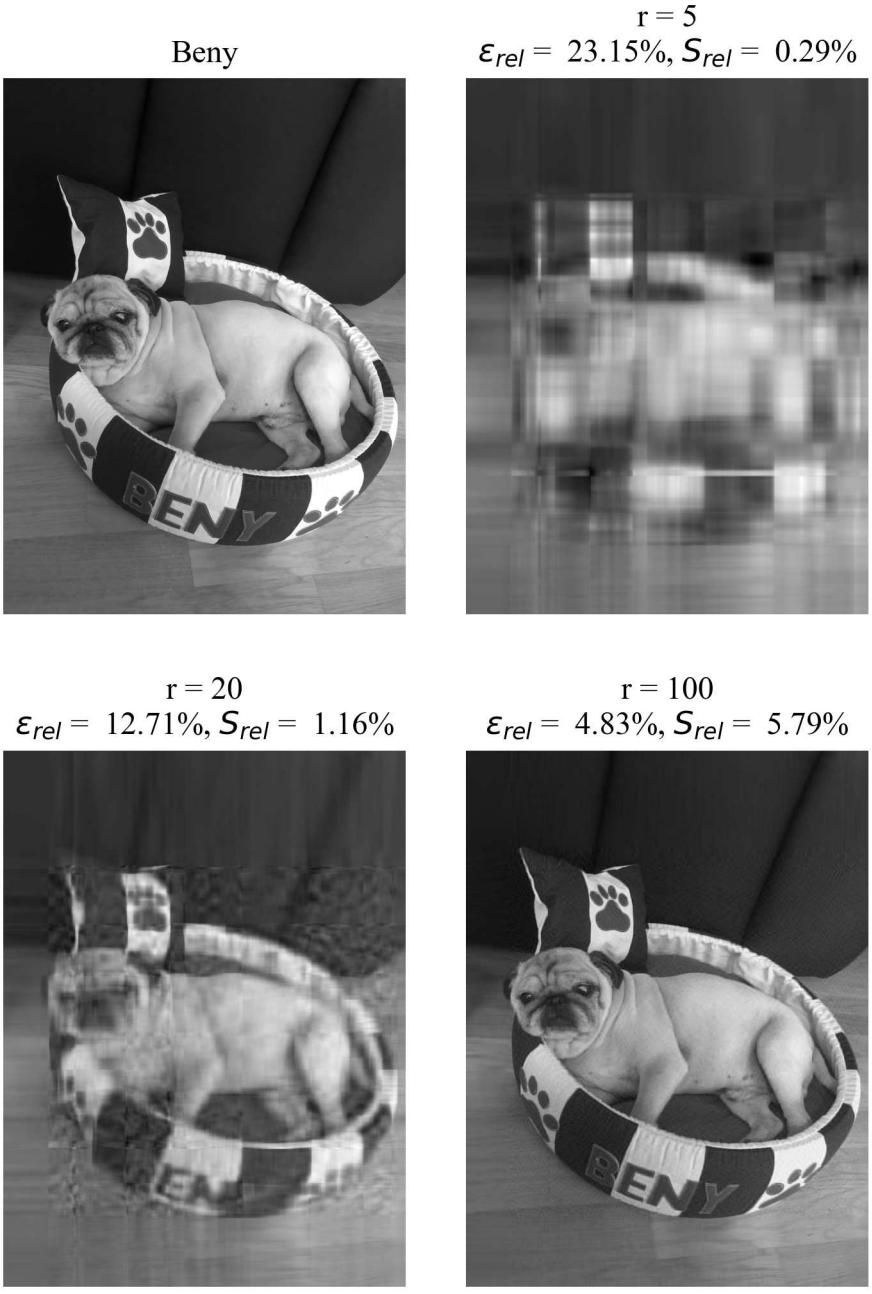
Analiza nezavisnih komponenti (eng. *Independent Component Analysis*) metoda je koja proširuje koncepte SVD-a te POD-a, a vrlo je vezana za problem separacije nepoznatog signala (eng. *blind signal separation (BSS)*) [20]. Taj problem matematički možemo postaviti na sljedeći način: "Ako nam je dano N linearnih kombinacija N različitih signala (ili skupova podataka), odredimo originalnih N signala." Odnosno, preglednije zapisano formulom

$$x_j(t) = a_{j1}s_1 + a_{j2}s_2 + \cdots + a_{jN}s_N \quad 1 \leq j \leq N. \tag{1.9}$$

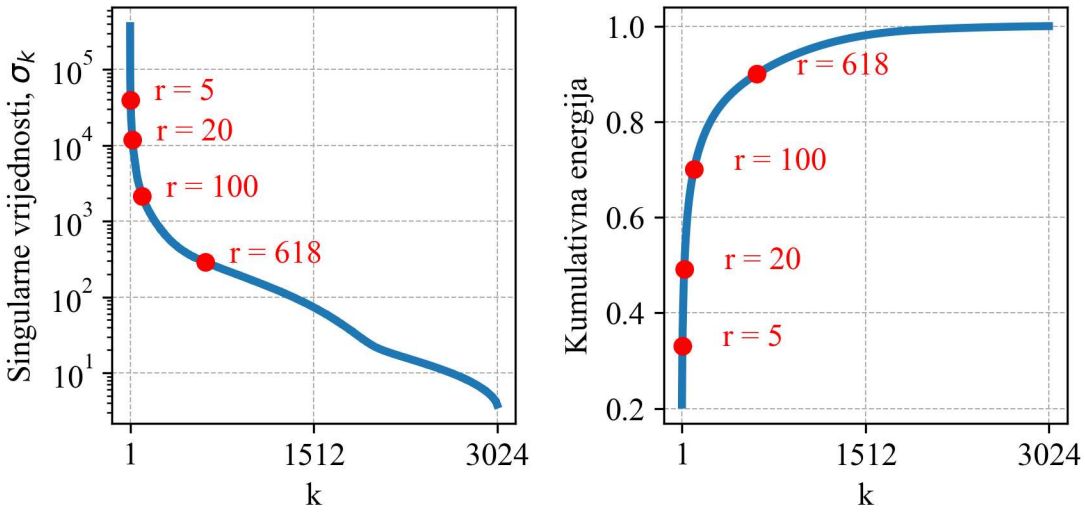
Kada to zapišemo matrično, vrijedi:

$$\mathbf{x} = \mathbf{A}\mathbf{s}, \tag{1.10}$$

gdje su $\mathbf{x}$ pomiješani signali, $\mathbf{s}$ su originalni signali koje pokušavamo odrediti, a $\mathbf{A}$ je matrica koeficijenata koja određuje kako su signali pomiješani. Na prvu se možda čini jednostavno rješenje uzimajući pseudoinverz matrice $\mathbf{A}$, no problem leži u činjenici da ne znamo



Slika 1.3: Primjer sažimanja slike koristeći tzv. "skraćeni SVD" za vrijednosti $r = 5, 20, 100$. ϵ_{rel} je relativna greška, dok S_{rel} označava relativnu veličinu slike. Primjerice, za $r = 20$ slika spremljena na računalo zauzimala bi samo 1.16% svoje početne veličine.



Slika 1.4: Lijevi graf prikazuje singularne vrijednosti σ_k u logaritamskoj skali za sliku mopsa iz 1.3, a desni graf prikazuje kumulativnu energiju u prvih k modova koja nam opisuje koliki postotak varijance slike opisujemo s tih prvih k modova.

niti matricu **A**, niti signal **s**. Dakle, cilj ICA metode jest aproksimirati koeficijente matrice **A** te vratiti signal **s** pod najopćenitijim mogućim pretpostavkama. U sklopu ovog rada nije nužno ulaziti dublje u problematiku, no detalji se mogu pronaći u [20].

Spektralna dekompozicija matrice

Iako se svojstvene vrijednosti iz 1.1.1 mogu definirati na razini linearnih operatora, najčešće se koriste na razini matrica. Iz navedene definicije vidimo da se zahtijeva netrivijalnost svojstvenog vektora, stoga se u suštini radi o smjerovima koji se ne mijenjaju djelovanjem matrice. Izdvojimo jedan bitan teorem za određeni tip matrica:

Teorem 1.3.4. *Neka je A kvadratna matrica reda n koja ima n linearno nezavisnih svojstvenih vektora $q_i, i = 1 \dots n$. Tada vrijedi:*

$$A = Q\Lambda Q^{-1}, \tag{1.11}$$

pri čemu je $Q = (q_1, \dots, q_n) \in \mathbb{C}^{n\times n}$, a $\Lambda = \mathrm{diag}(\lambda_1, \dots, \lambda_n)$, gdje su $\lambda_1, \dots, \lambda_n$ pripadajuće svojstvene vrijednosti.

Ovakvu dekompoziciju zovemo **spektralna dekompozicija** matrice, a moguća je samo za *dijagonalizabilne* matrice.

Poglavlje 2

Dinamička modalna dekompozicija (DMD)

Istraživanje struktura niže dimenzionalnosti u sklopu vrlo složenih sustava se pokazalo vrlo efektivnim - s računalne strane, ali i s teorijske strane reducirajući sustav na nešto s čime se može jednostavnije baratati. Što učiniti kada sustav kojeg želimo opisati ili ne poznajemo dovoljno da bismo ga opisali modelom ili je taj model suviše kompliciran? U tom slučaju možemo se okrenuti algoritmima *vođenim podacima* (eng. data-driven methods). DMD metoda jedna je od takvih, a u ovom poglavlju objasnit ćemo je detaljnije. Glavni dijelovi poglavlja inspirirani su s [2], [14] i [19].

2.1 Uvod u DMD

Metoda dinamičke modalne dekompozicije (DMD) nastala je u sklopu problematike dinamike fluida kao metoda za dekompoziciju složenih tokova u jednostavnu reprezentaciju temeljenu na prostorno-vremenskim koherentnim strukturama. Algoritam su inicijalno definirali Schmid i Sesterhenn [26] te prikazali sposobnost pružanja uvida iz visokodimenzionalnog prostora rješenja diskretiziranih diferencijalnih jednadžbi iz dinamike fluida. Vrlo brzo se proširio utjecaj ove metode, a ponajviše zahvaljujući činjenici što je to tzv. *metoda bez jednadžbe* (eng. "equation free method"). Točnije, metoda isključivo iz podataka može pružiti točnu dekompoziciju složenog sustava u prostorno-vremenske koherentne strukture koje se, međuostalom, mogu koristiti za kratkoročno predviđanje te kontrolu budućeg stanja. U širem smislu, DMD je brzo stekao popularnost nakon što su Mezić i dr. [24] te Rowley i dr. [25] pokazali povezanost s temeljnom nelinearnom dinamikom preko teorije Koopmanovog operatora.

U ovom poglavlju predstavit ćemo srž DMD algoritma, a osim toga spomenuti širi povijesni kontekst i razvoj metode, kao i pripadajuću matematičku pozadinu. Nadalje,

kako bismo razvili intuiciju, testirat ćemo metodu na jednostavnom primjeru koji će nam poslužiti i kao motivacija.

2.2 Šira slika

U svojoj srži, DMD možemo shvaćati kao idealnu kombinaciju tehnika redukcije prostorne dimenzionalnosti, primjerice POD iz 1.3, s Fourierovim transformacijama u vremenu. Na taj način korelirane prostorne modove istovremeno povezujemo s pripadajućim vremenskim frekvencijama. Metoda se oslanja isključivo na prikupljanje uzoraka podataka x_k (eng. "snapshots of data") o nekom dinamičkom sustavu u diskretnim trenucima t_k, gdje je $k = 1, 2, 3, \ldots, m$. Algoritamski gledano, kako ćemo pokazati kasnije, DMD je regresija podataka na lokalno linearne dinamike $x_{k+1} = Ax_k$, gdje je A izabran tako da minimizira $\|x_{k+1} - Ax_k\|_2$ za sve $k = 1, 2, 3, \ldots, m-1$. Prednost ovakvog pristupa je što je vrlo jednostavno za izvršiti te skoro da i nema pretpostavki na sustav s kojim radimo. Najskuplji dio je izračunati SVD matrice konstruirane uzorcima podataka.

DMD ima mnoštvo primjena i interpretacija, no možemo izdvojiti 3 glavne zadaće koje obavlja.

I. **Dijagnostika.** U svojim začecima DMD algoritam korišten je u svrhu karakterizacije složenog toka fluida. Točnije, iz mnogih visokodimenzionalnih sustava algoritam vraća ključne prostorno-vremenske podatke nižeg ranga. Ovo je vjerojatno i dan-danas primarna zadaća DMD metode.

II. **Procjena trenutnog stanja i predviđanje budućeg.** Sofisticiranija primjena DMD metode povezana je s korištenjem dominantnih prostorno-vremenskih struktura kako bi se izgradio dinamički model procesa kojeg opisujemo. Ovo je nešto zahtjevniji zadatak, pogotovo uzevši u obzir da je DMD ograničen vratiti najbolji (u smislu najmanjih kvadrata) *linearan* dinamički sustav koji odgovara pripadajućem *nelinearnom* sustavu kojeg opažamo. Dakle, pokušavamo predvidjeti buduće stanje sustava kojeg opisujemo isključivo podacima. U cijeloj priči je zbunjujuća činjenica da temeljna dinamika sustava može pokazivati višestruku dinamiku u vremenu i prostoru. Bez obzira na to, postoje strategije, primjerice pametno uzorkovanje podataka, koji omogućuju učinkovitost DMD-a za generiranje korisnih linearnih dinamičkih modela.

III. **Kontrola.** Krajnji i najizazovniji cilj DMD-a jest omogućivanje održivih i robusnih strategija upravljanja izravno iz uzorkovanja podataka. Uzevši u obzir činjenicu da pokušavamo linearizirati nelinearni sustav, razumno je očekivati da postoji samo kratkotrajan vremenski prozor (ako uopće postoji) na kojem bi se linearni i nelinearni model mogli donekle podudarati. Možemo se nadati da je to kratko razdoblje ipak dovoljno dugo za donošenje odluke koja utječe na buduće stanje sustava. U ovom slučaju DMD metoda, vođena isključivo podacima, omogućuje pristup teoriji upravljanja pružajući moćan mate-

matički alat za kontroliranje složenih dinamičkih sustava čije jednadžbe su nam nepoznate ili ih je teško računalno modelirati.

Važno je napomenuti da točnost i uspješnost DMD metode uvelike ovisi o tome na koju od prethodnih zadaća se oslanjamo.

2.3 Formulacija problema

U arhitekturi DMD metode tipično uzimamo u obzir podatke prikupljene iz nekog dinamičkog sustava

$$\frac{d\mathbf{x}}{dt} = \mathbf{f}(\mathbf{x}, t; \mu), \tag{2.1}$$

gdje je $\mathbf{x}(t) \in \mathbb{R}^n$, $n \gg 1$, vektor koji predstavlja stanje našeg dinamičkog sustava u trenutku t, μ sadrži parametre sustava, a $f(\cdot)$ predstavlja dinamiku. Općenito govoreći, dinamički sustav je reprezentiran sustavom običnih diferencijalnih jednadžbi koje su često nelinearne. Nadalje, dinamika iz 2.1 je u neprekidna, ali može inducirati pripadajuću diskretnu vremensku reprezentaciju u kojoj uzorkujemo sustav svakih Δt trenutaka i označimo vrijeme kao indeks $\mathbf{x}_k = \mathbf{x}(k\Delta t)$.
Evolucijom sustava 2.1 za Δt dobijemo preslikavanje F:

$$\mathbf{x}_{k+1} = F(\mathbf{x}_k). \tag{2.2}$$

Preslikavanje F možemo dobiti, primjerice, kao rezultat mjerenja, numeričke simulacije i sl.

Rješenje za 2.1 općenito nije moguće konstruirati, stoga se oslanjamo na numerička rješenja kako bismo evaluirali budući sustav. DMD metoda se okreće perspektivi bez jednadžbi, što svakako ima smisla s obzirom da dinamika $\mathbf{f}(\mathbf{x}, t; \mu)$ može biti nepoznata. Iz tog razloga samo mjerenja podataka sustava se koriste za aproksimaciju dinamike i predviđanje budućih stanja. DMD metoda konstruira približno lokalno linearan dinamički sustav

$$\frac{d\mathbf{x}}{dt} = \mathcal{A}\mathbf{x} \tag{2.3}$$

s početnim uvjetom $\mathbf{x}(0)$ te već poznatim rješenjem

$$\mathbf{x}(t) = \sum_{k=1}^{n} \phi_k \exp(\omega_k t)\, b_k = \mathbf{\Phi} \exp(\mathbf{\Omega} t)\mathbf{b}, \tag{2.4}$$

gdje su ϕ_k i ω_k svojstveni vektori i svojstvene vrijednosti matrice $\mathcal{A}$, a koeficijenti b_k koordinate od $\mathbf{x}(0)$ zapisane u bazi koju čine svojstveni vektori.
Neka nam je sada dana lokalno linearna aproksimacija neprekidnog sustava 2.1:

$$\frac{d\mathbf{x}}{dt} = \mathcal{A}\mathbf{x}, \quad \mathcal{A} \in \mathbb{C}^{n\times n}. \tag{2.5}$$

Uz takav neprekidni sustav uvijek je moguće opisati i analogni diskretizirani sustav uzorkovan svakih Δt vremenskih trenutaka:

$$\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k, \tag{2.6}$$

gdje je

$$\mathbf{A} = \exp(\mathcal{A}\Delta t). \tag{2.7}$$

$\mathcal{A}$ označava matricu iz neprekidnog sustava 2.3. Rješenje tog sustava može se jednostavno opisati u terminima svojstvenih vrijednosti λ_k i vektora ϕ_k:

$$\mathbf{x}_k = \sum_{j=1}^{r} \boldsymbol{\phi}_j \lambda_j^k b_j = \boldsymbol{\Phi}\Lambda^k \mathbf{b}. \tag{2.8}$$

Kao i prije, $\mathbf{b}$ su koeficijenti početnog uvjeta $\mathbf{x}_1$ u bazi razapetoj svojstvenim vektorima, tako da je $\mathbf{x}_1 = \boldsymbol{\Phi}\mathbf{b}$. DMD algoritam vraća spektralnu dekompoziciju 2.8 matrice $\mathbf{A}$ koja optimalno zadovoljava podatke x_k za $k = 1, 2, \ldots, m$ u smislu najmanjih kvadrata tako da je

$$\|\mathbf{x}_{k+1} - \mathbf{A}\mathbf{x}_k\|_2 \tag{2.9}$$

minimizirano za svaki $k = 1, 2, \ldots, m-1$.

Da bismo to postigli, možemo posložiti vektore stanja po stupcima u sljedeće matrice:

$$\mathbf{X} = \begin{bmatrix} | & | & & | \\ \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_{m-1} \\ | & | & & | \end{bmatrix}, \tag{2.10}$$

$$\mathbf{X}' = \begin{bmatrix} | & | & & | \\ \mathbf{x}_2 & \mathbf{x}_3 & \cdots & \mathbf{x}_m \\ | & | & & | \end{bmatrix}. \tag{2.11}$$

Na ovaj način dobili smo *visoke i mršave* ($n \gg m$) matrice. Prisjetimo se, podaci su vjerojatno uzorkovani iz nelinearnog dinamičkog sustava, no mi tražimo optimalnu lokalno linearnu aprosimaciju. U novim oznakama možemo 2.6 zapisati kao

$$\mathbf{X}' \approx \mathbf{A}\mathbf{X}, \tag{2.12}$$

čije rješenje u smislu najmanjih kvadrata je dano s

$$\mathbf{A} = \mathbf{X}'\mathbf{X}^{\dagger}, \tag{2.13}$$

gdje je † Moore-Penroseov pseudoinverz iz 1.1.4. Kao što je već i spomenuto, to rješenje minimizira grešku

$$\|\mathbf{X}' - \mathbf{A}\mathbf{X}\|_F \,, \tag{2.14}$$

gdje je $\|\cdot\|_F$ Frobeniusova norma iz 2.

Iz ovog zapisa jasno nam je zašto na rješenje 2.13 možemo gledati kao na linearnu regresiju podataka na dinamiku danu s $\mathbf{A}$. Međutim, bitno je za napomenuti kako postoji fundamentalna razlika između DMD metode i alternativnih tehnika zasnovanima na regresiji. Naime, ovdje pretpostavljamo da su podaci $\mathbf{x}_k$ u matrici $\mathbf{X}$ visokodimenzionalni, stoga je ta matrica *visoka i mršava*, odnosno $n \gg m-1$. Primjerice, ako je $n = 10^6$, tada A ima 10^{12} elemenata pa ju je izrazito teško reprezentirati ili faktorizirati. No, rang te matrice je najviše $m-1$, stoga umjesto da računamo s $\mathbf{A}$ direktno, prvo ćemo projicirati naše podatke na potprostor nižeg ranga koji je razapet s najviše $m-1$ POD modova te izračunati aproksimaciju nižeg ranga $\tilde{A}$. DMD algoritam zatim koristi taj operator nižeg ranga $\tilde{A}$ kako bi rekonstruirao preostale svojstvene vrijednosti i vektore operatora A bez da računa A eksplicitno.

Definicija 2.3.1 (Dinamička modalna dekompozicija)**.** *Neka je zadan dinamički sustav*

$$\frac{d\mathbf{x}}{dt} = \mathbf{f}(\mathbf{x}, t; \mu)$$

te 2 skupa podataka

$$\mathbf{X} = \begin{bmatrix} | & | & & | \\ \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_{m-1} \\ | & | & & | \end{bmatrix}, \tag{2.15}$$

$$\mathbf{X}' = \begin{bmatrix} | & | & & | \\ \mathbf{x}'_1 & \mathbf{x}'_2 & \cdots & \mathbf{x}'_{m-1} \\ | & | & & | \end{bmatrix} \tag{2.16}$$

takva da vrijedi $\mathbf{x}'_k = \mathbf{F}(\mathbf{x}_k)$*, gdje je* $\mathbf{F}$ *preslikavanje za koje vrijedi* $\mathbf{x}_{k+1} = \mathbf{F}(\mathbf{x}_k)$ *koje odgovara evoluciji sustava 2.3 za* Δt*, a* $\mathbf{x}_k \in \mathbb{R}^{n\times 1}$ *su vektori stanja.* ***Dinamička modalna dekompozicija*** *računa vodeću svojstvenu dekompoziciju linearnog operatora* $\mathbf{A}$ *koji najbolje opisuje podatke* $\mathbf{X}' \approx \mathbf{A}\mathbf{X}$*:*

$$\mathbf{A} = \mathbf{X}'\mathbf{X}^{\dagger}. \tag{2.17}$$

DMD modovi, također zvani i dinamički modovi, su svojstveni vektori matrice $\mathbf{A}$*, a svakom modu pripada odgovarajuća svojstvena vrijednost matrice* $\mathbf{A}$

2.4 Poveznica DMD-a i Koopmanovog operatora

DMD metoda aproksimira modove tzv. *Koopmanovog operatora*. Koopmanov operator je beskonačnodimenzionalan linearan operator koji predstavlja djelovanje nelinearnog dinamičkog sustava na Hilbertovom prostoru funkcija stanja (eng. *measurement functions)*. Valja napomenuti kako se Koopmanov operator ne oslanja na linearizaciju dinamike, već predstavlja, kao beskonačnodimenzionalan operator, tok dinamičkog sustava na funkcije stanja.

S druge strane, na DMD metodu možemo gledati kao na postupak računanja svojstvenih parova od konačnodimenzionalnog linearnog modela koji aproksimira beskonačnodimenzionalan Koopmanov operator. Budući da je operator linearan, dekompozicija nam daje stope rasta i frekvencije pridružene svakom modu. Nadalje, ukoliko je bazni dinamički sustav linearan, DMD metoda će vratiti vodeće svojstvene parove koje bismo dobili i drugim standardnim metodama za linearne diferencijalne jednadžbe, što znači da je naša definicija dobra.

Matematički gledano, Koopmanov operator $\mathcal{K}$ je beskonačnodimenzionalan linearan operator na Hilbertovom prostoru $\mathcal{H}$ svih funkcije stanja $g : \mathbb{C}^n \to \mathbb{C}$. Koopmanov operator još je poznat pod nazivom *operator kompozicije*, što je očito iz njegovog djelovanja na funkciju stanja g:

$$\mathcal{K}g = g \circ \mathbf{F} \tag{2.18}$$

$$\implies \quad \mathcal{K}g\left(\mathbf{x}_k\right) = g\left(\mathbf{F}\left(\mathbf{x}_k\right)\right) = g\left(\mathbf{x}_{k+1}\right), \tag{2.19}$$

pri čemu je $\mathbf{F}$ (dinamički) tok.

Aproksimacija Koopmanovog operatora uistinu je srž DMD algoritma. Kao što smo već rekli, diskretizacija sustava po vremenu Δt je linearna, iako je bazna dinamika vjerojatno nelinearna. Napomenimo još da je ovo značajno različito od linearizacije samog dinamičkog sustava. Spomenimo još kako je točnost aproksimacije Koopmanovog operatora uvjetovana izborom $\mathbf{y} = \mathbf{g}(\mathbf{x})$, no to izlazi iz okvira ovog rada, a može se pronaći u 3. poglavlju iz [19].

2.5 DMD algoritam

DMD metoda vraća, međuostalom, spektar matrice $\mathbf{A}$, lineariziranog diskretnog dinamičkog sustava 2.6. U praksi, dimenzije te matrice su velike, stoga je uvijek prvi korak smanjiti dimenziju koristeći reducirani SVD algoritam aproksimirajući matricu podataka $\mathbf{X}$ matricom nižeg ranga u terminima POD modova iz poglavlja 1.3. Najprije ćemo ukratko opisati svaki korak, nakon čega će slijediti i pseudokod.

1. Uzmimo reduciranu SVD dekompoziciju matrice podataka $\mathbf{X} \in \mathbb{C}^{n \times m}$

$$\mathbf{X} \approx \mathbf{U}\Sigma\mathbf{V}^*, \tag{2.20}$$

pri čemu je $\mathbf{U} \in \mathbb{C}^{n \times r}, \Sigma \in \mathbb{C}^{r \times r}$, a $\mathbf{V} \in \mathbb{C}^{m \times r}$. Naravno, stupci matrice $\mathbf{U}$ su već spomenuti POD modovi i tvore ortonormiranu bazu.

U ovoj fazi paralelno smo izabrali dimenziju r reduciranog SVD-a. Točnije, ukoliko u podacima postoji neka struktura nižeg ranga koja dobro opisuje podatke, tada će svojstvene vrijednosti brzo padati u 0 te će nam ostati samo konačno mnogo dominantnih modova.

2. Matricu $\mathbf{A}$ iz 2.13 možemo dobiti računajući pseudoinverz koristeći SVD metodu:

$$\mathbf{A} = \mathbf{X}'\mathbf{V}\Sigma^{-1}\mathbf{U}^*. \tag{2.21}$$

No, u praksi to nije učinkovito jer koristimo originalne dimenzije matrica podataka. Iskoristimo li novodobivenu ortonomiranu bazu, dobit ćemo $\tilde{\mathbf{A}}$, $r \times r$ projekciju pune matrice $\mathbf{A}$ na POD modove:

$$\tilde{\mathbf{A}} = \mathbf{U}^*\mathbf{A}\mathbf{U} = \mathbf{U}^*\mathbf{X}'\mathbf{V}\boldsymbol{\Sigma}^{-1}. \tag{2.22}$$

Ključno za uočiti jest da reducirana matrica $\tilde{\mathbf{A}}$ ima jednake ne-nul svojstvene vrijednosti kao i puna matrica $\mathbf{A}$, stoga možemo skroz zaobići direktno računanje s $\mathbf{A}$. Matrica $\tilde{\mathbf{A}}$ definira linearni model dinamičkog sustava u terminima POD koordinata:

$$\tilde{\mathbf{x}}_{k+1} = \tilde{\mathbf{A}}\tilde{\mathbf{x}}_k, \tag{2.23}$$

a s druge strane je također moguće rekonstruirati izvorne visokodimenzionalne vektore stanja: $\mathbf{x}_k = \mathbf{U}\tilde{\mathbf{x}}_k$.

3. Izračunajmo spektralnu dekompoziciju matrice $\tilde{\mathbf{A}}$:

$$\tilde{\mathbf{A}}\mathbf{W} = \mathbf{W}\boldsymbol{\Lambda}, \tag{2.24}$$

gdje su stupci od $\mathbf{W}$ svojstveni vektori, a $\boldsymbol{\Lambda}$ je dijagonalna matrica sa svojstvenim vrijednostima λ_k na dijagonali.

4. Sada možemo rekonstruirati spektralnu dekompoziciju od $\mathbf{A}$ koristeći spektralnu dekompoziciju od $\tilde{\mathbf{A}}$. Točnije, svojstvene vrijednosti od $\mathbf{A}$ su dane s $\boldsymbol{\Lambda}$, a svojstveni vektori od A (tj. DMD modovi) su dani stupcima matrice $\boldsymbol{\Phi}$:

$$\boldsymbol{\Phi} = \mathbf{X}'\mathbf{V}\Sigma^{-1}\mathbf{W}. \tag{2.25}$$

Modovi u 2.25 se još zovu i *egzaktni DMD modovi* jer je dokazano u [27] da su to egzaktni svojstveni vektori matrice $\mathbf{A}$. S druge strane, $\mathbf{\Phi} = \mathrm{UW}$ zovu se *projicirani DMD modovi*. Kada tražimo dinamički mod od $\mathbf{A}$ s pripadnom svojstvenom vrijednosti $\lambda_k = 0$, onda možemo koristiti egzaktne modove u slučaju da je $\phi = \mathbf{X}'\mathbf{V}\mathbf{\Sigma}^{-1}w \neq 0$. U suprotnom, koristimo projicirane modove.

Sada kada imamo niže-dimenzionalnu aproksimaciju ranga r svojstvenih vrijednosti i vektora, rješenje 2.8 možemo konstruirati za sva buduća vremena. Radi lakše notacije supstituirat ćemo $\omega_k = \ln(\lambda_k)/\Delta t$. Aproksimacija rješenja za sva buduća vremena t dana je s:

$$\mathbf{x}(t) \approx \sum_{k=1}^{r} \boldsymbol{\phi}_k \exp(\omega_k t)\, b_k = \mathbf{\Phi} \exp(\mathbf{\Omega} t)\mathbf{b}, \tag{2.26}$$

gdje je b_k početna amplituda svih modova, $\mathbf{\Phi}$ je matrica čiji su stupci DMD modovi $\boldsymbol{\phi}_k$, a $\Omega = \mathrm{diag}(\omega)$ je dijagonalna matrica sa svojstvenim vrijednostima ω_k na dijagonali. Dimenzija svojstvenih vektora ϕ_k podudara se s dimenzijom vektora stanja $\mathbf{x}_k$, a $\mathbf{b}$ je vektor koeficijenata b_k.

Preostaje nam jedino izračunati početne vrijednosti za b_k. No, to možemo dobiti vrlo jednostavno evaluirajući 2.26 za vrijeme $t_1 = 0$ dobivši $\mathbf{x}_1 = \mathbf{\Phi}\mathbf{b}$. Uzevši u obzir da $\mathbf{\Phi}$ općenito neće biti kvadratna matrica, početni uvjet $\mathbf{b}$ dobit ćemo računajući Moore-Penroseov pseudoinverz:

$$\mathbf{b} = \mathbf{\Phi}^{\dagger}\mathbf{x}_1. \tag{2.27}$$

U ovom trenutku sposobni smo zapisati cijeli algoritam na jednom mjestu u obliku MATLAB koda.

Algoritam 2.1: DMD funkcija

```
function [Phi,omega,lambda,b,Xdmd] = DMD(X1,X2,r,dt)
% Racuna DMD dekompoziciju od X1, X2
%
% ULAZ:
% X1 = X, matrica podataka (od 1. do predzadnjeg stupca)
% X2 = X', pomaknuta matrica podataka (od 2. do zadnjeg stupca)
% Stupci od X1 i X2 su stanja
% r = rang koji cemo koristiti u SVD-u
% dt = vremenski pomak od X1 do X2
%
% IZLAZ:
% Phi, DMD modovi
% omega, neprekidne DMD svojstvene vrijednosti
```

```
% lambda, diskretne DMD svojstvene vrijednosti
% b, vektor koji sadrzi magnitude modova iz Phi
% Xdmd, matrica rekonstruirana koristeci Phi, omega i b

%% DMD
[U, S, V] = svd(X1, 'econ');
r = min(r, size(U,2));

U_r = U(:, 1:r); % rezanje ranga
S_r = S(1:r, 1:r);
V_r = V(:, 1:r);
Atilde = U_r' * X2 * V_r / S_r; % nisko-dimenzionalne dinamike
[W_r, D] = eig(Atilde);
Phi = X2 * V_r / S_r * W_r; % DMD modovi

lambda = diag(D); % svojstvene vrijednosti u diskretnom vremenu
omega = log(lambda)/dt; % svojstvene vrijednosti u neprekidnom vremenu

%% izracunaj DMD modove i amplitude b
x1 = X1(:, 1);
b = Phi\x1;

%% DMD rekonstrukcija
mm1 = size(X1, 2); % mm1 = m - 1
time_dynamics = zeros(r, mm1);
t = (0:mm1-1)*dt; % vektor vremena
for iter = 1:mm1
    time_dynamics(:,iter) = (b.*exp(omega*t(iter)));
end
Xdmd = Phi * time_dynamics;
```

2.6 Jednostavan primjer

Da bismo bolje prikazali kako DMD algoritam djeluje, provest ćemo ga na jednostavnom primjeru dva pomiješana vremensko-prostorna signala. Točnije, cilj nam je demonstrirati sposobnost DMD-a da točno rastavi signal na svoje sastavne dijelove. Nadalje, usporedit ćemo rezultate s metodama PCA (eng: *Principal Component Analysis*) i ICA (eng: *Independent Component Analysis*). Primjer je preuzet iz 1.4 poglavlja u [19].

Neka su nam signali dani na sljedeći način:

$$f(x,t) = f_1(x,t) + f_2(x,t) \tag{2.28}$$
$$= \operatorname{sech}(x+3)\exp(i2.3t) + 2\operatorname{sech}(x)\tanh(x)\exp(i2.8t). \tag{2.29}$$

Pripadne sastavne signale $f_1(x,t)$ and $f_2(x,t)$ možemo vidjeti na slici 2.6(a)-(b). Prisutne frekvencije su redom $\omega_1 = 2.3$ i $\omega_2 = 2.8$ te nam daju različite prostorne strukture. Na slici 2.6(c) možemo uočiti i pomiješani (točnije, zbrojeni) signal $f(x,t) = f_1(x,t) + f_2(x,t)$. Sljedeći MATLAB kod kreira navedene signale:

Algoritam 2.2: Miješanje dvaju prostorno-vremenskih signala

```
%% definirajmo vremensku i prostornu diskretizaciju
xi = linspace(-10,10,400);
t = linspace(0,4*pi,200);
dt = t(2) - t(1);
[Xgrid,T] = meshgrid(xi,t);

%% kreirajmo 2 vremensko-prostorna uzorka/modela
f1 = sech(Xgrid+3) .* (1*exp(1j*2.3*T));
f2 = (sech(Xgrid).*tanh(Xgrid)).*(2*exp(1j*2.8*T));

%% pomijesajmo signale zajedno i kreirajmo matricu podataka
f = f1 + f2;
X = f.'; % matrica podataka - transponirana tako da su stanja po
    stupcima

%% vizualizacija f1, f2 i f
figure;
subplot(2,2,1);
surfl(real(f1));
shading interp; colormap(jet); view(-20,60);
set(gca, 'YTick', numel(t)/4 * (0:4)),
set(gca, 'Yticklabel',{'0','\pi','2\pi','3\pi','4\pi'});
set(gca, 'XTick', linspace(1,numel(xi),3)),
set(gca, 'Xticklabel',{'-10', '0', '10'});

subplot(2,2,2);
surfl(real(f2));
shading interp; colormap(jet); view(-20,60);
set(gca, 'YTick', numel(t)/4 * (0:4)),
```

```
set(gca, 'Yticklabel',{'0','\pi','2\pi','3\pi','4\pi'});
set(gca, 'XTick', linspace(1,numel(xi),3)),
set(gca, 'Xticklabel',{'-10', '0', '10'});

subplot(2,2,3);
surfl(real(f));
shading interp; colormap(jet); view(-20,60);
set(gca, 'YTick', numel(t)/4 * (0:4)),
set(gca, 'Yticklabel',{'0','\pi','2\pi','3\pi','4\pi'});
set(gca, 'XTick', linspace(1,numel(xi),3)),
set(gca, 'Xticklabel',{'-10', '0', '10'});
```

Navedeni kod ujedno kreira matricu $\mathbf{X}$ iz 2.15 koja je polazna točka za sve navedene algoritme dekompozicije. U kodu algoritma 2.3 provodimo DMD algoritam kojem prethodi redukcija ranga na $r = 2$. To činimo da bismo prikazali niskodimenzionalnu prirodu dekompozicije.

Algoritam 2.3: Primjena DMD-a na podacima

```
%% kreirajmo matrice podataka za DMD
X1 = X(:, 1:end-1);
X2 = X(:, 2:end);

%% SVD i rezanje ranga do ranga 2
r = 2; % rezanje ranga
[U, S, V] = svd(X1, 'econ');
Ur = U(:, 1:r);
Sr = S(1:r, 1:r);
Vr = V(:, 1:r);

%% definicija za Atildu i DMD modove
Atilde = Ur'*X2*Vr/Sr;
[W, D] = eig(Atilde);
Phi = X2*Vr/Sr*W;  % DMD modovi

%% DMD spektar
lambda = diag(D);
omega = log(lambda)/dt;

%% izracunajmo DMD rjesenje
x1 = X(:, 1);
b = Phi\x1;
```

```
time_dynamics = zeros(r,length(t));
% NAPOMENA: za potrebe citljivosti koristimo for petlju, inace se lako moze vektorizirati
for iter = 1:length(t),
    time_dynamics(:,iter) = (b.*exp(omega*t(iter)));
end;
X_dmd = Phi*time_dynamics;

subplot(2,2,4);
surfl(real(X_dmd'));
shading interp; colormap(jet); view(-20,60);
set(gca, 'YTick', numel(t)/4 * (0:4)),
set(gca, 'Yticklabel',{'0','\pi','2\pi','3\pi','4\pi'});
set(gca, 'XTick', linspace(1,numel(xi),3)),
set(gca, 'Xticklabel',{'-10', '0', '10'});

set(gcf, 'Color', 'w', 'Position', [500 500 400 300]);
set(gcf, 'PaperUnits', 'inches', 'PaperPosition', [0 0 4 3], 'PaperPositionMode', 'auto');
print('-dpng', '-loose', ['../figures/' sprintf('dmd_intro1.png')]);
```

Ovaj kod računa nekoliko bitnih značajki podataka kao što su singularne vrijednosti matrice $\mathbf{X}$, DMD modove $\mathbf{\Phi}$ te neprekidne DMD svojstvene vrijednosti ω. Svojstvene vrijednosti ω egzaktne su frekvencijama temeljnih signala koji čine miješani signal, pri čemu imaginarni dijelovi komponenata predstavljaju frekvenciju oscilacije modova.

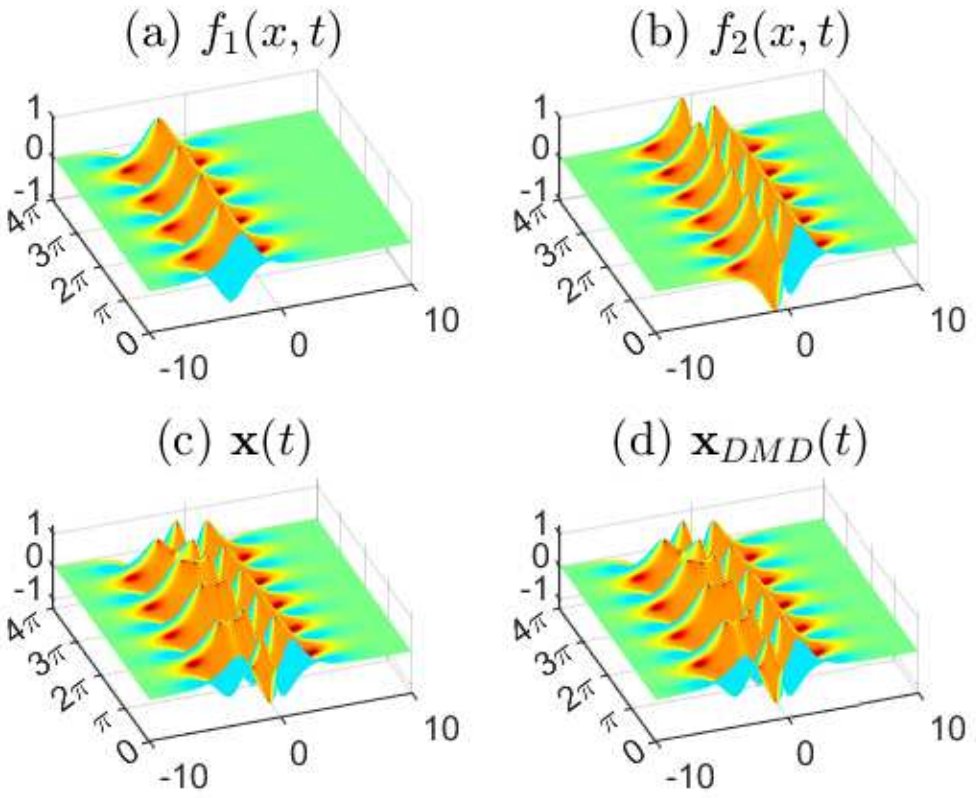
Iz formule 2.26 vidimo da je rekonstrukcija ranga $r = 2$ moguća. Time početne podatke podijelimo na sumu dva spojena prostorno-vremenska moda. Tu rekonstrukciju možemo vidjeti na slici 2.6. Na slici 2.6 (a) vidimo kako norme singularnih vrijednosti matrice $\mathbf{X}$ vrlo brzo opadaju počevši s trećom vrijednošću, što nam daje do znanja da rang $r = 2$ i te kako ima smisla. Slike 2.6(c)-(f) uspoređuju stvarne modove s onima koje nam je DMD dao. Vizualno možemo zaključiti da su gotovo jednaki stvarnim temeljnim modovima signala $f_1(x,t)$ i $f_2(x,t)$.

```
omega =

  -0.0000 + 2.8000i
  -0.0000 + 2.3000i
```

DMD neprekidne svojstvene vrijednosti

Kao što smo već spomenuli, želimo rezultate usporediti s rezultatima analize glavnih komponenti (PCA), odnosno analize nezavisnih komponenti (ICA).



Slika 2.1: *Primjer prostorno-vremenske dinamike dvaju signala. (a)* $f_1(x,t)$ *i (b)* $f_2(x,t)$ *iz 2.28 koji daju (c)* $f(x,t) = f_1(x,t) + f_2(x,t)$*. Funkciju* $f(x,t)$ *možemo prikazati s* $\mathbf{x}(t)$*. Izračunali smo DMD od* $\mathbf{x}$*, a rekonstrukcija ranga 2 prikazana je u (d). Rekonstrukcija je gotovo identična - vidimo da se DMD modovi i spektar podudaraju sa stvarnim vrijednostima temeljnih signala* $f_1(x,t)$ *i* $f_2(x,t)$*.*

Algoritam 2.4: PCA izračunat preko SVD-a

```
[U, S, V] = svd(X);
pc1 = U(:, 1); % prvi PCA mod
pc2 = U(:, 2); % drugi PCA mod
time_pc1 = V(:, 1); % vremenska evolucija od pc1
time_pc2 = V(:, 2); % vremenska evolucija od pc2
```

Algoritam 2.5: Brzi ICA

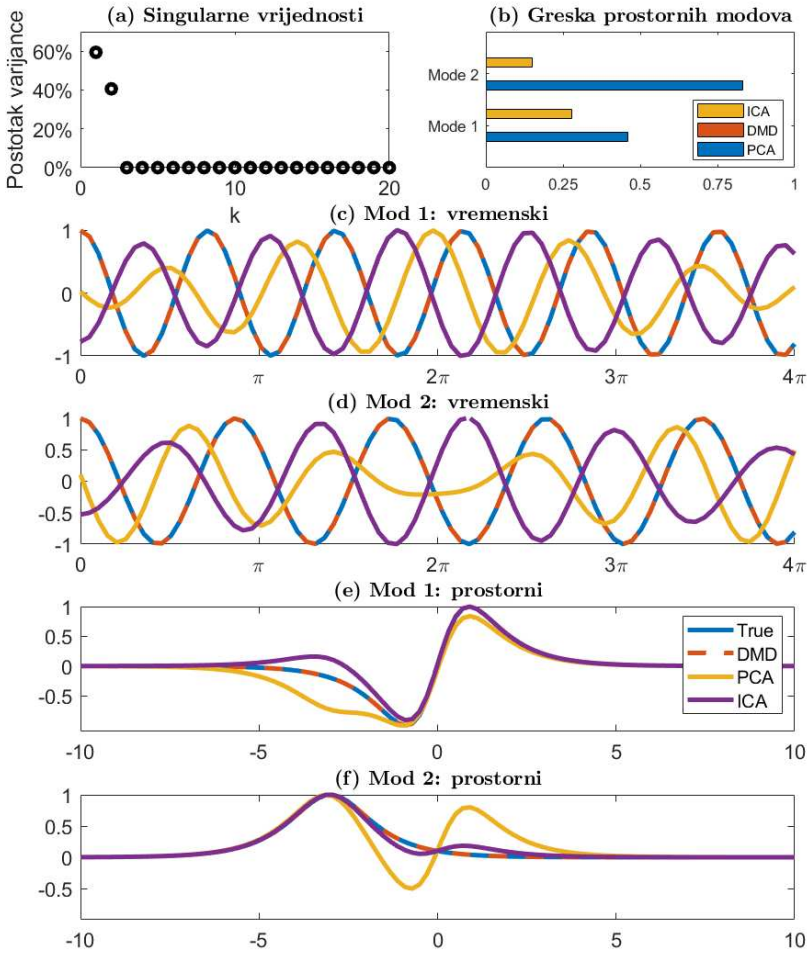
```
[IC, ICt , ~] = fastica(real(transpose(X)));
ic1 = IC(1, :); % prvi ICA mod
ic2 = IC(2, :); % drugi ICA mod
time_ic1 = ICt(:, 1); % vremenska evolucija od ic1
time_ic2 = ICt(:, 2); % vremenska evolucija od ic2
```

PCA i ICA algoritmi izdvajaju strukture iz danih podataka. Na slici 2.6 možemo usporediti njihove modove s onima od DMD metode te sa stvarnim rješenjem. PCA nema mogućnost odvojiti nezavisne signale $f_1(x,t)$ i $f_2(x,t)$, no barem točno vraća strukturu ranga $r = 2$. PCA modovi dekompozicije ranga 2 su izabrani tako da maksimiziraju varijancu podataka. Slika 2.6(e) i (f) pokazuje da su ti modovi međusobno izmiješani, a slika 2.6(c)-(d) prikazuje i vremensku dinamiku PCA modova.

S druge strane, rezultati ICA metode znatno su bolji jer ICA uzima u obzir nezavisnost signala, tj. forsira takvo rješenje. ICA modovi vidljivi su na slici 2.6(c)-(f).

Osim vizualnih usporedbi, priložit ćemo i kvantitativne rezultate. Slika 2.6(b) prikazuje ℓ_2 normu razlike između 2 stvarna prostorna moda i modova dobivenih navedenim trima algoritmima. PCA modovi su očito najdalji stvarnom rješenju, dok su DMD modovi točni gotovo na razini strojne preciznosti.

Ovom usporedbom uspjeli smo prikazati neke prednosti DMD metode i efikasnost u razdjeli prostorno-vremenskih podataka.



Slika 2.2: *Usporedba DMD-a s metodama PCA te ICA na primjeru prostorno-vremenskog signala iz 2.28. Singularne vrijednosti u (a) pokazuju da je rang-2 smislen. Na grafovima (c)-(f) uspoređuju se pripadni vremenski i prostorni modovi s pravim vrijednostima. DMD se slaže gotovo na razini strojne točnosti, dok ICA djeluje bolje od PCA, iako oba već vizualno odudaraju od stvarnog rješenja. U (b) vidimo ℓ_2 normu razlika prostornih modova i modova koje nam daju navedeni algoritmi. DMD modovi odgovaraju stvarnim modovima, stoga je greška jednaka nuli.*

Poglavlje 3

Primjena standardnog DMD-a na videozapise

U ovom poglavlju demonstrirat ćemo glavnu temu ovog rada, što je ujedno samo jedna od mogućih primjena DMD metode. Naime, pokazat ćemo kako možemo iskoristiti DMD algoritam da bismo tijekom obrade videozapisa odvojili pozadinu (eng. *background*) od prvog plana (eng. *foreground*). Kao bazni algoritam predstavit ćemo RPCA (eng. *Robust PCA*), metodu koja je vrlo raširena za potrebe separacije videa na pozadinu i prvi plan. Uvest ćemo motivacijski primjer kojim ćemo predstaviti nit vodilju za analizu videozapisa, a u kasnijim poglavljima podaci s kojima ćemo raditi dolaze iz familije snimki nadzornih kamera ili snimki iz prometa. Nadzorne kamere snimaju brzinom od 20 do 30 kadrova u sekundi (eng. *frames per second*), stoga predstavljaju prirodan kandidat dinamičkog sustava koji vremenski evoluira. DMD modovi koji su frekvencijom blizu ishodištu (nuli) predstavljaju ili neki statični dio videa ili nešto što se jako sporo mijenja, stoga se može poistovjetiti s pozadinom. S druge strane, modovi koji su udaljeni od ishodišta brže evoluiraju kroz vrijeme pa nam daju prvi plan. Većina sadržaja preuzeta je iz [14] i [19]

3.1 RPCA

Analiza glavnih komponenti (eng. *Principal Component Analysis*), skraćeno PCA, jedna je od najraširenijih metoda za redukciju dimenzije problema. To je vrlo jednostavna neparametarska metoda koju u praksi dobijemo uzimajući SVD dekompoziciju matrice. RPCA (eng. *Robust PCA*) svojevrsno je proširenje PCA metode koje vrlo dobro rješava problem redukcije dimenzije, pri čemu je robustno na potencijalno *oštećene* i/ili rijetke podatke (eng. *sparse data*). U ovom konkretnom slučaju, RPCA izvrsno radi jer uzima u obzir i ℓ_1 i ℓ_2 normu.

Primijenimo li RPCA na podatke koji potencijalno dolaze iz kompleksnog, nelinearnog

sustava, tražit će rijetke (eng. *sparse*) strukture u podacima, a istovremeno ostatak pokušati uklopiti u niži rang. Dokle god su podaci takve prirode, RPCA garantira da će podijeliti dane podatke $\mathbf{X}$ na $\mathbf{X} = \mathbf{L} + \mathbf{S}$, gdje je $\mathbf{L}$ nižeg ranga, a $\mathbf{S}$ je rijetka [3].

Dakle, za dani $\mathbf{X}$ imamo:

$$\mathbf{X} = \mathbf{L} + \mathbf{S}, \tag{3.1}$$

gdje je

$$\mathbf{L} \rightarrow \text{nižeg ranga},$$
$$\mathbf{S} \rightarrow \text{rijetka}.$$

Ključ RPCA algoritma leži u pravilnoj formulaciji problema. Naime, prethodni problem prevest ćemo u savladiv problem neglatke konveksne optimizacije (eng. *prinicpal component pursuit (PCP)*):

$$\arg\min \|\mathbf{L}\|_F + \lambda \|\mathbf{S}\|_1 \tag{3.2}$$
$$\text{tako da je } \mathbf{X} = \mathbf{L} + \mathbf{S}. \tag{3.3}$$

Ovdje PCP minimizira ponderiranu sumu Frobeniusove norme, $\|\mathbf{M}\|_F := \text{trag}\left(\sqrt{\mathbf{M}^*\mathbf{M}}\right)$, i ℓ_1-norme, $\|\mathbf{M}\|_1 := \sum_{ij} \left|m_{ij}\right|$ koje smo opisali u 1.1.3. Skalarni parametar regularizacije je nenegativan: $\lambda \geq 0$. Iz 3.2 vidimo da će za $\lambda \rightarrow 0$ matrica nižeg ranga "pokupiti" većinu podataka, $\mathbf{L} \rightarrow \mathbf{X}$, dok će povećanjem parametra λ rijetka matrica $\mathbf{S}$ sadržavati većinu početnih podataka, $\mathbf{S} \rightarrow \mathbf{X}$, jer se L razmjerno približava nul-matrici [3], [14].

Drugim riječima, parametar λ na neki način kontrolira dimenziju potprostora nižeg ranga. No, valja napomenuti kako nije potrebno znati rang matrice $\mathbf{L}$ apriori. Naime, Candès i dr. [3] su pokazali da izbor

$$\lambda = \frac{1}{\sqrt{\max(n, m)}}, \tag{3.4}$$

pri čemu je $\mathbf{X}$ dimenzije $n \times m$, daje točan rastav u slučaju da su matrice $\mathbf{L}$ i $\mathbf{S}$ međusobno inkoherentne, što je u praktičnim primjenama često zadovoljeno.

Postoji mnogo algoritama koje rješavaju problem iz 3.2, a jedan od njih je Lagrangeovim multiplikatorima. Programski kod koji to rješava može se pronaći na [23]

3.2 Standardni DMD za obradu videozapisa

DMD algoritam može se koristiti da bi se dobila slična separacija na niži rang i neku rijetku strukturu, no, za razliku od RPCA, nemamo nikakve garancije na točnost dobivenog rješenja. U slučaju DMD-a, vrlo bitna je interpretacija frekvencija ω_k. Značajke nižeg ranga za koje vrijedi $\left|\omega_j\right| \approx 0$ možemo okarakterizirati kao one koje se sporo mijenjaju

kroz vrijeme. Dakle, uzevši to u obzir, možemo uzeti neki prag tolerancije ϵ kojim bismo prikupili sve modove takve da je $\left|\omega_j\right| \leq \epsilon \ll 1$, čime ćemo dobiti traženu separaciju na matrice $\mathbf{L}$ i $\mathbf{S}$:

$$\mathbf{L} \approx \sum_{|\omega_k| \leq \epsilon} b_k \phi_k \exp(\omega_k t), \tag{3.5}$$

$$\mathbf{S} \approx \sum_{|\omega_k| > \epsilon} b_k \phi_k \exp(\omega_k t). \tag{3.6}$$

Primijetimo koliko je u ovom slučaju jednostavnije dobiti tražene matrice za razliku od rješavanja optimizacijskog problema iz 3.2. Obratimo sada pozornost na videozapis.

Kao što je već spomenuto, videozapis je prirodan primjer dinamičkog sustava koji vremenski evoluira. Neka nam je dan videozapis s m kadrova, pri čemu svaki kadar ima $n_x \times n_y = n$ piksela koje ćemo vektorizirati u $n \times 1$ vektore $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m$. DMD metodom moguće je pokušati rekonstruirati bilo koji kadar računajući $\mathbf{x}_{\text{DMD}}(t)$ u trenutku t. Valjanost rekonstrukcije uvelike ovisi o tome kako pojedini skup kadrova zadovoljava pretpostavke i kriterije DMD metode.

Da bismo rekonstruirali cijeli video, uzet ćemo $1 \times m$ vektor vremena $\mathbf{t} = [t_1\ t_2\ \cdots\ t_m]$, koji sadrži vremena koja odgovaraju kadrovima. Cijela sekvenca $\mathbf{X}$ se dobije na sljedeći način:

$$\mathbf{X}_{\text{DMD}} = \sum_{k=1}^{r} b_k \boldsymbol{\phi}_k e^{\omega_k \mathbf{t}} = \boldsymbol{\Phi}\, \text{diag}(\exp(\omega t))\mathbf{b}. \tag{3.7}$$

Uočimo, pomnožili smo $\boldsymbol{\phi}_k$, $n \times 1$ vektor, s $1 \times m$ vektorom $\mathbf{t}$, čime smo dobili $n \times m$ matricu, stoga dimenzije odgovaraju uvjetima početnog problema. Iz konstrukcije rješenja možemo uočiti da je, uzevši u obzir kontekst DMD algoritma, $\mathbf{x}_1 = \boldsymbol{\Phi}\mathbf{b}$, što zapravo znači da $\boldsymbol{\Phi}\mathbf{b}$ prikazuje prvi kadar videozapisa s redukcijom dimenzionalnosti određenom parametrom r.

Dakle, dijagonalna matrica frekvencija ω, tj. dijagonalna matrica svojstvenih vrijednosti $\Omega = \text{diag}(\omega)$, prilikom rekonstrukcije preostalih kadrova kontrolira kako se prvi kadar mijenja u vremenu. Očito slijedi da bilo koji dio videozapisa koji se ne mijenja kroz vrijeme, ili se vrlo malo mijenja, mora sadržavati pripadne Fourierove modove (ω_k) koji su u blizini ishodišta u kompleksnoj ravnini: $|\omega_k| \approx 0$. Ova opservacija ključna je u podjeli na pozadinu (aproksimacija za niži rang $\mathbf{L}$) i prvi plan (aproksimacija za rijetku matricu $\mathbf{S}$) koristeći DMD metodu.

Pretpostavimo da za neki ω_p, gdje je $p \in \{1, 2, \ldots, r\}$, vrijedi $\left|\omega_p\right| \approx 0$ (tipično je to samo za 1 mod) i neka vrijedi da su $|\omega_k|, \forall k \neq p$ udaljeni od ishodišta (od nule u kompleksnoj ravnini). Sada prethodnu jednadžbu 3.7 možemo zapisati na sljedeći način:

$$\mathbf{X}_{\text{DMD}} = \underbrace{b_p \boldsymbol{\phi}_p e^{\omega_p \mathbf{t}}}_{\text{pozadina videozapisa}} + \underbrace{\sum_{k \neq p} b_k \boldsymbol{\phi}_k e^{\omega_k \mathbf{t}}}_{\text{prvi plan videozapisa}}. \tag{3.8}$$

Pretpostavimo li da je $\mathbf{X} \in \mathbb{R}^{n\times m}$, tada rekonstrukcija DMD metodom također daje $\mathbf{X}_{\text{DMD}} \in \mathbb{R}^{n\times m}$. Međutim, iako je to u konačnici realna matrica, svaki pojedini član DMD rekonstrukcije potencijalno može biti kompleksan, tj. $b_k\boldsymbol{\phi}_k \exp(\omega_k \mathbf{t}) \in \mathbb{C}^{n\times m}, \forall k$. [1] Inače nam ovo ne bi predstavljalo problem, no u kontekstu separacije videozapisa htjeli bismo u konačnici imati realne vrijednosti jer ne znamo interpretirati kompleksne vrijednosti kao piksele. S druge strane, kompleksne vrijednosti vrlo su bitne za točnost i preciznost rezultata. Uzevši u obzir navedeno, prisiljeni smo pronaći pristup kako bismo se prilagoditi situaciji.

Neka nam je dana DMD aproksimacija nižeg ranga kao

$$\mathbf{X}_{\text{DMD}}^{\text{niži-rang}} = b_p\phi_p e^{\omega_p t}.$$

S obzirom da mora vrijediti

$$\mathbf{X} = \mathbf{X}_{\text{DMD}}^{\text{niži-rang}} + \mathbf{X}_{\text{DMD}}^{\text{rijetka}}, \tag{3.9}$$

onda DMD aproksimaciju rijetke matrice, tj.

$$\mathbf{X}_{\text{DMD}}^{\text{Sparse}} = \sum_{k\neq p} b_k\boldsymbol{\phi}_k e^{\omega_k \mathbf{t}},$$

koja ima realne vrijednosti (ne kompleksne) možemo dobiti jedino na sljedeći način:

$$\mathbf{X}_{\text{DMD}}^{\text{rijetka}} = \mathbf{X} - \left|\mathbf{X}_{\text{DMD}}^{\text{niži-rang}}\right|,$$

pri čemu $|\cdot|$ označava modul po svakom elementu matrice. No, isto tako, ovakav pristup nam očito može dati negativne vrijednosti u nekim elementima $\mathbf{X}_{\text{DMD}}^{\text{Sparse}}$ matrice. Naravno, u obradi slike, tj. videozapisa, nema smisla imati negativne vrijednosti piksela, stoga možemo napraviti sljedeće: negativne vrijednosti reziduala spremit ćemo u $n \times m$ matricu $\mathbf{R}$, a naposljetku ih pridodati nazad matrici $\mathbf{X}_{\text{DMD}}^{\text{niži-rang}}$:

$$\begin{aligned}\mathbf{X}_{\text{DMD}}^{\text{niži-rang}} &\leftarrow \mathbf{R} + \left|\mathbf{X}_{\text{DMD}}^{\text{niži-rang}}\right| \\ \mathbf{X}_{\text{DMD}}^{\text{rijetka}} &\leftarrow \mathbf{X}_{\text{DMD}}^{\text{rijetka}} - \mathbf{R}.\end{aligned}$$

Na ovaj način smo sačuvali magnitudu (modul) kompleksnih vrijednosti iz DMD rekonstrukcije, a istovremeno smo se držali prethodno spomenutih uvjeta za obradu slike (videozapisa) koji nalažu da pikseli budu realni pozitivni brojevi. Ovo je varijacija koja je korištena u [19].

Međutim, u praksi ovaj pristup nije se uspostavio najboljim jer smo prethodnom transformacijom uveli mnoge *artefakte* u sliku. Ono što smo u ovom radu koristili jest sljedeće: neka nam je dana DMD rekonstrukcija $\mathbf{X}_{\text{DMD}}$. S obzirom da to može imati kompleksne vrijednosti, uzet ćemo modul po svakom elementu matrice. Zatim, modelirat ćemo pozadinu videozapisa kao

$$\mathbf{X}_{\text{DMD}}^{\text{niži-rang}} = b_p\phi_p e^{\omega_p t}, \tag{3.10}$$

[1] Više o ovome može se pronaći u poglavlju 4.3 u [19].

a u tom slučaju rijetku reprezentaciju s realnim vrijednostima dobijemo kao:

$$\mathbf{X}_{\text{DMD}}^{\text{rijetka}} = \mathbf{X} - \mathbf{X}_{\text{DMD}}^{\text{niži-rang}}.$$

U ovom trenutku potencijalno možemo dobiti negativne vrijednosti. Međutim, za naš problem nije nužno bitno da uhvatimo točnu nijansu ili boju nekog elementa na slici, već nam je jedino važno uhvatiti razliku, stoga ne škodi *prebaciti* negativne vrijednosti u pozitivne uzimajući modul $|\cdot|$ po elementima matrice

$$\mathbf{X}_{\text{DMD}}^{\text{rijetka}} \to \left|\mathbf{X}_{\text{DMD}}^{\text{rijetka}}\right|. \tag{3.11}$$

U konačnici smo, zapravo, dobili sljedeće:

$$\mathbf{X}_{\text{DMD}}^{\text{rijetka}} = \left|\mathbf{X} - \mathbf{X}_{\text{DMD}}^{\text{niži-rang}}\right|. \tag{3.12}$$

Ova heuristika uspostavila se znatno efikasnijom od polazne ideje iz 4. poglavlja knjige [19], a konačno opravdanje ideje slijedit će u kasnijim poglavljima kad primijetimo da nam se problem svodi na klasifikacijski problem, odnosno na problem kojim želimo krajnji izlaz svesti isključivo na nule i jedinice, 0 i 1.

Napomena 3.2.1. *Razliku između obiju metoda možemo vidjeti na slici 3.1. Na gornjem nizu slika možemo vidjeti originalnu preporučenu metodu iz [19], dok se ispod nalazi netom spomenuta heuristika. Odmah je uočljivo da nova metoda puno bolje modelira pozadinu ne upijajući elemente prvog plana. No, vrijedi napomenuti da se prvi planovi ne razlikuju znatno. Usput rečeno, prvi planovi su umjetno posvijetljeni faktorom 5 kako bi se bolje uočili kontrasti.*

3.3 Motivacijski primjer

Osvrnimo se prvo na jednostavan primjer jednodimenzionalne dinamike u vremenu kako bismo demonstrirali problematiku. Naime, definirat ćemo funkciju koja se sastoji od dijela koji ne ovisi o vremenu - *pozadina* - pomiješanog s dinamikom koja ovisi o vremenu - *prvi plan*. Cilj nam je pomoću DMD algoritma uspješno izdvojiti oba dijela navedenog modela. Primjerice, možemo to definirati ovako:

$$f(x,t) = f_1(x) + f_2(x,t) \tag{3.13}$$
$$= 0.5\cos(x) + 2\,\text{sech}(x)\tanh(x)\exp(i2.8t), \tag{3.14}$$

gdje $f_1(x)$ ne ovisi o vremenu, dok $f_2(x,t)$ ovisi. Priložen je i MATLAB kod u 3.1.

Slika 3.1: Usporedba separacije na prvi plan i pozadinu iz skupa podataka *AVSS PV HARD* iz [22]. U gornjem retku nalazi se originalna heuristika za obradu videozapisa koja se spominje u [19], dok je u donjem retku heuristika koju smo upravo opisali na kraju ovog poglavlja. U originalnom primjeru uočavamo spomenute *artefakte* na slici koji su uneseni iz prvog plana u pozadinu, a isti se ne pojavljuju u novoj metodi obrade videozapisa. Prikazana je samo jedna sličica iz *AVSS PV HARD* skupa podataka.

Algoritam 3.1: Miješanje dvaju signala

```
%% Definirajmo vremensku i prostornu diskretizaciju
n = 200;
m = 80;
x = linspace (-15,15,n);
t = linspace (0,8*pi,m);
dt = t(2) - t(1);
[Xgrid ,T] = meshgrid(x,t);
%% dva vremensko-prostorna uzorka
f1 = 0.5* cos(Xgrid) .* (1+0*T); % NE ovisi o vremenu!
f2 = (sech(Xgrid).*tanh (Xgrid)) .* (2*exp(1j*2.8*T));
%% kreiranje zajednickog signala
X = transpose(f1 + f2); % matrica podataka
figure;
surfl(real(transpose(X));
shading interp; colormap(gray ); view (-20,60);
```

Sada samo slijedimo korake DMD algoritma koje smo već ranije definirali. U ovom slučaju (3.2) reducirali smo dimenziju problema uzevši $r = 50$ te izdvojili svojstvenu vrijednost 0 koju nam daje pozadinski mod.

Algoritam 3.2: DMD signala

```
%% matrice podataka
X1 = X(:,1:end -1);
X2 = X(:,2:end);

r = 50; % rezanje ranga
[U, S, V] = svd(X1, "econ");
Ur = U(:, 1:r);

Sr = S(1:r, 1:r);
Vr = V(:, 1:r);
%% Atilda i DMD modovi
Atilde = transpose(Ur)*X2*Vr/Sr;
[W, D] = eig(Atilde);
Phi = X2*Vr/Sr*W; % DMD modovi
%% DMD spektar
lambda = diag(D);
omega = log(lambda)/dt;
figure;
```

```
plot(omega , ".");
```

Svojstvene vrijednosti blizu ishodištu smatraju se pozadinskim modovima jer ne ovise o vremenu, stoga se i koriste za konstrukciju DMD pozadinskog moda.

Algoritam 3.3: Podjela na prvi plan i pozadinu

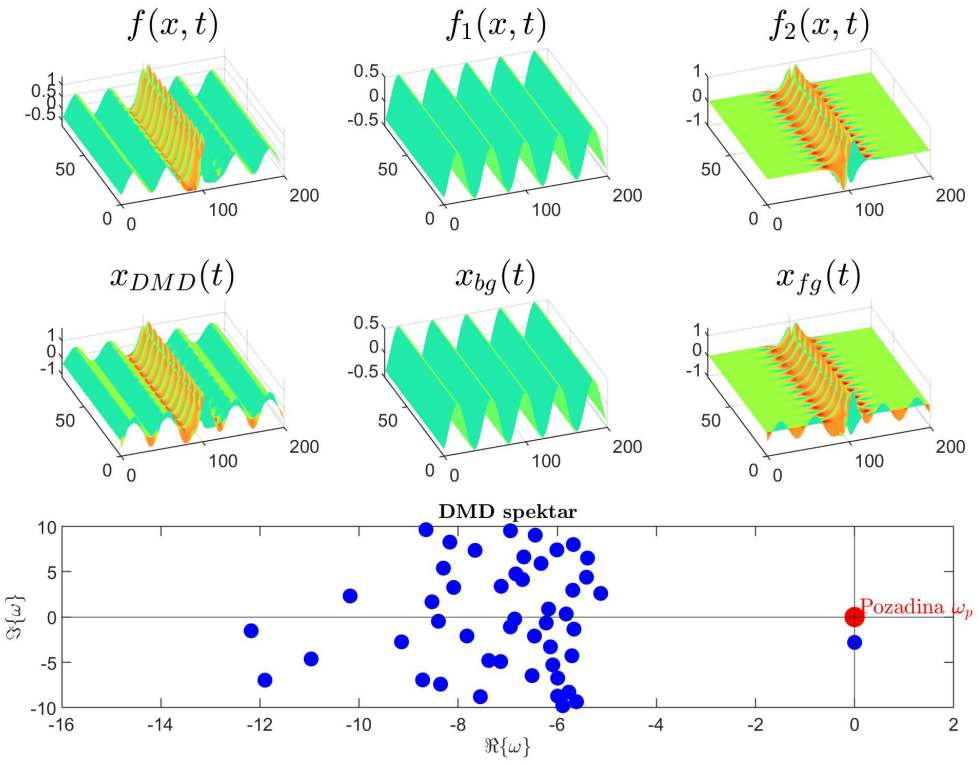
```
%% kreiranje matrica za DMD
bg = find(abs(omega)<1e-2);
fg = setdiff(1:r, bg);
omega_fg = omega(fg); % prvi plan
Phi_fg = Phi(:,fg); % DMD modovi za prvi plan
omega_bg = omega(bg); % pozadina
Phi_bg = Phi(:,bg); % DMD modovi za pozadinu
```

Nakon što imamo DMD prvi plan i pozadinu, možemo ih iskoristiti za cjelokupno DMD rješenje. U ovom slučaju dobili smo DMD rekonstrukciju pozadine i prvog plana ”videozapisa”. Sljedeći kod to opisuje:

Algoritam 3.4: Rekonstrukcija prvog plana i pozadine

```
%% racunanje DMD rjesenja za pozadinu
b = Phi_bg \ X(:, 1);
X_bg = zeros(numel(omega_bg), length(t));
for tt = 1:length(t),
    X_bg(:, tt) = b .* exp(omega_bg .* t(tt));
end;
X_bg = Phi_bg * X_bg;
X_bg = X_bg (1:n, :);
figure;
surfl(real(transpose(X_bg)));
shading interp; colormap(gray ); view (-20,60);
%% racunanje DMD rjesenja za prvi plan
b = Phi_fg \ X(:, 1);

X_fg = zeros(numel(omega_fg), length(t));
for tt = 1:length(t),
    X_fg(:, tt) = b .* exp(omega_fg .* t(tt));
end;
X_fg = Phi_fg * X_fg;
X_fg = X_fg (1:n, :);
figure;
surfl(real(transpose(X_fg)));
shading interp; colormap(gray ); view (-20,60);
```



Slika 3.2: Primjer separacije na pozadinu i prvi plan. Početni signal $f(x,t)$, prikazan u gornjem redu skroz lijevo, kreiran je kao linearna kombinacija vremenski neovisne komponente $f_1(x,t)$ (gornji red desno) i vremenski ovisne $f_2(x,t)$ (gornji red u sredini). U srednjem stupcu redom se nalaze DMD rekonstrukcija, pozadina te prvi plan. DMD spektralna distrubucija nalazi se u zadnjem redu, pri čemu je posebno označena svojstvena vrijednost najbliža ishodištu kojom je modelirana pozadina, $\omega_p \approx 10^{-15}$.

Prethodni kod bazira se na algoritmu koji je ranije opisan u 2.1. Jedina razlika je u koraku separacije DMD svojstvenih vrijednosti. Jednom kad smo ih izdvojili, rekonstrukcija pojedinog dijela se temelji na danim vrijednostima. Na slici 3.2 u gornjem retku možemo vidjeti originalni signal zajedno s pojedinačnim komponentama. Ispod toga nalazi se DMD rekonstrukcija, pozadina te prvi plan. Osim toga, možemo uočiti i DMD spektar, pri čemu je važna svojstvena vrijednost oko ishodišta kojom smo modelirali pozadinu.

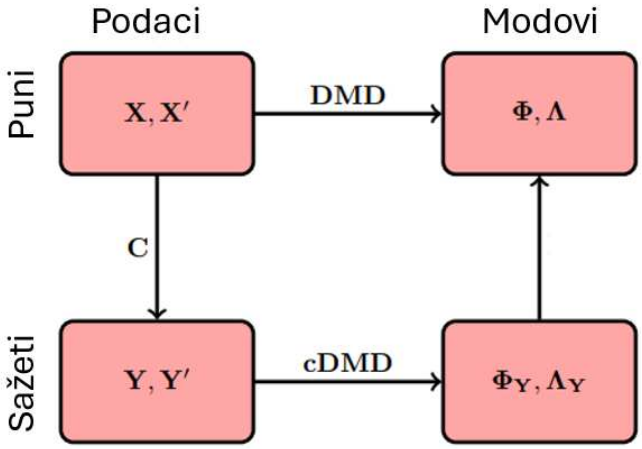
Poglavlje 4

Sažeti DMD za ekstrakciju pozadine videozapisa

U prethodnim poglavljima vidjeli smo da je *najskuplji* dio DMD metode zapravo računanje SVD-a. Unatoč svim nedavnim unaprjeđenjima poput *nasumičnog SVD-a* (eng. *randomized SVD*), i dalje je vrlo sporo izračunati SVD za videozapise visoke rezolucije. S druge strane, smanjivanjem rezolucije videozapisa izgubimo mnogo detalja, stoga bi bilo sjajno kada bismo mogli efikasno računati DMD u visokoj rezoluciji. U ovom poglavlju opisat ćemo metodu *sažetog DMD-a* (eng. *compressed DMD*), kraće cDMD. To je metoda koja integrira ideje sažetog uzorkovanja (eng. *compressed sensing*) i skiciranja matrica (eng. *matrix sketching*) u DMD. Uzevši to u obzir, umjesto da se računa DMD u punoj rezoluciji, pokazat ćemo da se može dobiti točna aproksimacija iz sažetog prikaza videozapisa. To nam otvara vrata mnogim drugim primjenama gdje smo odjednom u mogućnosti rješavati probleme znatno većih dimenzija. Poglavlje je motivirano s [2], [8], [10], [16] i [19].

4.1 Sažeti DMD

Sažeti DMD je računalno učinkovit alat za računanje DMD-a na nedovoljno uzorkovanim (eng. *under-sampled data*) ili komprimiranim podacima. Inicijalno je zamišljen za rekonstrukciju visoko-dimenzionalnih potpunih DMD modova iz prostorno rijetkih mjerenja koristeći sažeto uzorkovanje. Međutim, ubrzo se shvatilo sljedeće: ukoliko imamo pristup punim mjerenjima visoke dimenzije, onda bismo mogli mnoge računalno iscrpne korake DMD-a izbjeći provodeći ih na sažetim prikazima podataka, te time značajno smanjiti složenost algoritma. Posljedično, otvara se mogućnost korištenja DMD metode u realnom vremenu na videozapisima visoke rezolucije.
U ovom odjeljku fokusirat ćemo se isključivo na ovaj drugi pristup u kojem, zapravo, ubrzavamo DMD metodu koristeći kombinaciju sažetih i izvornih podataka. Shematski to



Slika 4.1: Shematski prikaz cDMD-a. Podaci se prvo sažmu lijevim množenjem matricom mjerenja $\mathbf{C}$. DMD se zatim izvodi na komprimiranom prikazu podataka. Konačno, puni DMD modovi rekonstruiraju se iz sažetih kao u jednadžbi 4.12

lako možemo ilustrirati slikom 4.1.

4.2 Sažeto uzorkovanje i skiciranje matrice

Algoritmi sažimanja srž su modernog softvera za obradu zvuka, slike te videozapisa, primjerice MPEG, JPEG i MP3 formata. Za potrebe matematičke pozadine sažetog DMD-a upotrijebit ćemo teoriju sažetog uzorkovanja (eng. *compressed sensing* ili *sparse sampling*) i skiciranja matrica (eng. *matrix sketching*).

Sažeto uzorkovanje temelji se na ideji da ne mjerimo neki signal ili sustav $\mathbf{x}$ u prostoru visoke dimenzije, već da umjesto toga uzmemo poduzorak znatno niže dimenzije $\mathbf{y}$ iz kojeg ćemo zatim aproksimirati ili rekonstruirati puni prostor stanja $\mathbf{x}$. Naravno, potrebno je zadovoljiti određene pretpostavke. Preciznije, u našem slučaju mora biti zadovoljeno da se podaci mogu na neki način sažeti, što je dakako istina za videozapise. Videozapis se, zbog svoje prirode da su uzastopne slike jako slične, može prikazati u bazi niže dimenzije. Primjerice, neka je $\mathbf{y} \in \mathbb{R}^p$, za $k < p \ll n$:

$$\mathbf{y} = \mathbf{C}\mathbf{x}. \tag{4.1}$$

Ako je $\mathbf{x}$ rijedak u $\mathbf{\Psi}$[1], onda možemo riješiti pododređen sustav jednadžbi

$$\mathbf{y} = \mathbf{C\Psi s} \tag{4.2}$$

po $\mathbf{s}$ i rekonstruirati $\mathbf{x}$. S obzirom da takav sustav ima beskonačno mnogo rješenja, tražit ćemo ono rješenje $\hat{\mathbf{s}}$ koje je *najrjeđe* (eng. *sparsest*), odnosno koje ima najviše nula. Međutim, iz literature o obradama rijetkih signala znamo da traženje najrjeđih rješenja uključuje ℓ_0 optimizaciju [2] koja je NP-teška. Snaga rijetkog uzorkovanja sada dolazi na vidjelo jer se pokaže da možemo umjesto toga, u određenim uvjetima na matricu $\mathbf{C}$, zamijeniti nedokučivu ℓ_0 optimizaciju s konveksnom ℓ_1 minimizacijom [3]:

$$\hat{\mathbf{s}} = \underset{\mathbf{s}'}{\operatorname{argmin}} \|\mathbf{s}'\|_1 , \qquad \text{t.d. vrijedi } \mathbf{y} = \mathbf{C\Psi s}'. \tag{4.3}$$

Dakle, ℓ_1-norma služi kao aproksimacija za traženje rijetkih rješenja za $\hat{\mathbf{s}}$. Da bi postojala garancija da će arhitektura sažetog uzorkovanja funkcionirati u probabilističkom aspektu, martica mjerenja $\mathbf{C}$ i rijetka baza $\mathbf{\Psi}$ moraju biti **inkoherentne**, što znači da retci od $\mathbf{C}$ moraju biti nekorelirani sa stupcima od $\mathbf{\Psi}$. Detalji slijede u [1].

S obzirom da ćemo se mi baviti videozapisima, odnosno nizovima slika, prirodan izbor mogu biti Fourierove baze ili valići (eng. *wavelets*), no u to nećemo ulaziti dublje.

Nadalje, već spomenuto matrično skiciranje nameće se kao drugi alat kojim se možemo služiti pri rješavanju problema sažimanja matrice podataka visoke dimenzionalnosti. Prednost je što su pretpostavke nešto blaže te postoji intuitivna generalizacija iz vektora u matrice. Tako jednakost 4.1 možemo izraziti matrično:

$$\mathbf{Y} = \mathbf{CX}, \tag{4.4}$$

gdje $\mathbf{C}$ ponovno označava prikladnu matricu mjerenja. Matrično skiciranje dolazi sa zanimljivim ogradama greške te se može koristiti kadgod matrica podataka $\mathbf{X}$ ima strukturu nižeg ranga. Primjerice, u [12] pokaže se da se singularne vrijednosti i desni singularni vektori mogu uspješno aproksimirati iz takvog sažetog prikaza.

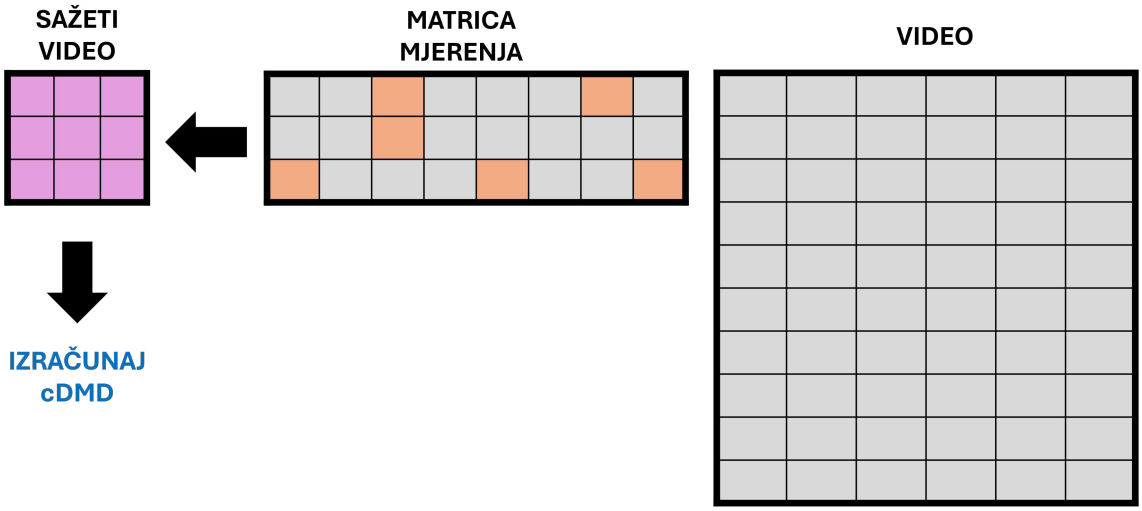
4.3 cDMD algoritam

cDMD algoritam je vrlo sličan standardnom DMD algoritmu u skoro svakom koraku sve do računanja DMD modova. Ključna razlika je što sada prvo sažmemo videozapis, što

[1] Biti *rijedak* u matrici $\mathbf{\Psi}$ znači da postoji vektor $\mathbf{s}$ koji ima *malo* ne-nul elemenata takav da je $\mathbf{x} = \mathbf{\Psi s}$. Drugim riječima, zapisali smo vektor $\mathbf{x}$ kao linearnu kombinaciju samo nekoliko stupaca od $\mathbf{\Psi}$.

[2] ℓ_0 je definirana kao broj ne-nul elemenata, tj. $\|x\|_0$ vraća broj elemenata vektora x koji su različiti od 0

[3] Više o tome može se saznati na [1] i [7]



Slika 4.2: Kompresija matrice koristeći rijetku matricu mjerenja.

možemo vidjeti na slici 4.2. Dakle, algoritam počinje generirajući matricu mjerenja $\mathbf{C} \in \mathbb{R}^{p\times n}$ u svrhu sažimanja matrica podataka kao u 2.11.

$$\mathbf{Y} = \mathbf{C}\mathbf{X}, \quad \mathbf{Y}' = \mathbf{C}\mathbf{X}', \tag{4.5}$$

pri čemu p označava broj uzoraka ili mjerenja.

Kao što smo već rekli, postoji fundamentalna pretpostavka da su ulazni podaci nižeg ranga. Videozapisi prirodno to zadovoljavaju zato što je svaki stupac od $\mathbf{X}$ i $\mathbf{X}' \in \mathbb{R}^{n\times m-1}$ rijedak u nekoj bazi transformacije $\mathbf{\Psi}$. Stoga, za dovoljno mnogo inkoherentnih mjerenja, sažete matrice $\mathbf{Y}$ i $\mathbf{Y}' \in \mathbb{R}^{p\times m-1}$ imaju slične korelacijske strukture kao njihove pripadne visokodimenzionalne inačice. cDMD zatim aproksimira svojstvene vrijednosti i vektore linearnog preslikavanja $\mathbf{A}_\mathbf{Y}$, gdje je procjenitelj dan s:

$$\hat{\mathbf{A}}_Y = \mathbf{Y}'\mathbf{Y}^\dagger = \mathbf{Y}'\mathbf{V}_\mathbf{Y}\mathbf{S}_\mathbf{Y}^{-1}\mathbf{U}_\mathbf{Y}^*, \tag{4.6}$$

pri čemu $*$ označava konjugirano transponiranje. Pseudoinverz $\mathbf{Y}^\dagger$, kao i u standardnom DMD-u, računamo koristeći SVD:

$$\mathbf{Y} = \mathbf{U}_\mathbf{Y}\mathbf{S}_\mathbf{Y}\mathbf{V}_\mathbf{Y}^*, \tag{4.7}$$

gdje su $\mathbf{U} \in \mathbb{R}^{p\times k}$ i $\mathbf{V} \in \mathbb{R}^{(m-1)\times k}$ skraćeni lijevi i desni singularni vektori. Dijagonalna matrica $\mathbf{S} \in \mathbb{R}^{k\times k}$ ima pripadajuće singularne vrijednosti na dijagonali, a k označava rang

skraćenog SVD-a.[4] Primijetimo da indeksom $\mathbf{Y}$ dajemo do znanja da radimo na sažetoj matrici $\mathbf{Y}$. Kao i u standardnom DMD algoritmu, nećemo eksplicitno računati iznimno veliku matricu $\hat{\mathbf{A}}_\mathbf{Y}$, već umjesto toga izračunamo model projiciran na lijeve singularne vektore

$$\tilde{\mathbf{A}}_\mathbf{Y} = \mathbf{U}_\mathbf{Y}^* \hat{\mathbf{A}_Y} \mathbf{U}_\mathbf{Y} \tag{4.8}$$

$$= \mathbf{U}_\mathbf{Y}^* \mathbf{Y}' \mathbf{V}_\mathbf{Y} \mathbf{S}_\mathbf{Y}^{-1}. \tag{4.9}$$

Kako se radi o transformaciji sličnosti, svojstveni parovi se mogu dobiti spektralnom dekompozicijom matrice $\tilde{\mathbf{A}}_\mathbf{Y}$:

$$\tilde{\mathbf{A}}_\mathbf{Y} \mathbf{W}_\mathbf{Y} = \mathbf{W}_\mathbf{Y} \Lambda_\mathbf{Y}, \tag{4.10}$$

gdje su stupci matrice $\mathbf{W}_\mathbf{Y}$ svojstveni vektori ϕ_j, a $\mathbf{\Lambda}_Y$ je dijagonalna matrica koja sadrži pripadne svojstvene vrijednosti λ_j. Zbog sličnosti matrica vrijedi $\mathbf{\Lambda} \approx \mathbf{\Lambda}_Y$, a posljedično DMD modovi sažete matrice $\mathbf{Y}$ dani su s:

$$\Phi_\mathbf{Y} = \mathbf{Y}' \mathbf{V}_\mathbf{Y} \mathbf{S}_\mathbf{Y}^{-1} \mathbf{W}_\mathbf{Y}. \tag{4.11}$$

Preostaje nam na kraju izvući pune DMD modove:

$$\mathbf{\Phi} = \mathbf{X}' \mathbf{V}_\mathbf{Y} \mathbf{S}_\mathbf{Y}^{-1} \mathbf{W}_\mathbf{Y}. \tag{4.12}$$

Uočimo da DMD modovi u 4.12 koriste originalne podatke $\mathbf{X}'$ zajedno s linearnim transformacijama dobivenim iz sažetih podataka $\mathbf{Y}$ i $\mathbf{Y}'$. Zaobišli smo računanje SVD-a na $\mathbf{X}$, što bi računalno bilo vrlo isrcpno.

Prethodne korake možemo sada pregledno napisati u cjelokupni algoritam 1 gdje će sličnosti sa standardnim DMD-om biti još uočljivije.

4.4 Matrice mjerenja

Osnovna matrica mjerenja $\mathbf{C}$ može se konstruirati uzimajući $p \times n$ nezavisnih slučajnih uzoraka iz Gaussove, uniformne ili, primjerice, Bernoullijeve distribucije. Može se pokazati da takve matrice mjerenja imaju optimalne karakteristike iz teorijske perspektive. Međutim, često u praksi nisu iskoristive za probleme velikih dimenzija zato što generiranje izrazito velikog broja (pseudo)slučajnih brojeva je skupo te memorijski zahtjevno. U okviru ovog rada nećemo ulaziti u detalje. Više o tome može se pročitati u [10].

[4] Umjesto predefiranog ranga k možemo koristiti optimalni *hard-threshold* kao u [11]. U ovom radu koristit ćemo tu opciju.

Algorithm 1 Sažeti DMD

Ulaz: Matrica $\mathbf{D} \in \mathbb{R}^{n\times m}$ koja sadrži sličice videozapisa izravnate po stupcima. Ova metoda računa aproksimaciju DMD dekompozicije, gdje su $\mathbf{\Phi} \in \mathbb{C}^{n\times k}$ DMD modovi, $\mathbf{b} \in \mathbb{C}^k$ su amplitude, a $\mathbf{V} \in \mathbb{C}^{k\times m}$ je Vandermondeova matrica koja opisuje vremensku evoluciju. Hiperparametri su k i p, rang i broj mjerenja. Mora vrijediti $n \geq m$, cijeli brojevi $k, p \geq 1$ i $k \ll m$ te $p \geq k$. Ovaj algoritam prepisan je iz [10].

$\mathbf{X}, \mathbf{X}' = \mathbf{D}$	matrica podataka
$\mathbf{C} = \text{rand}(p, m)$	Matrica mjerenja dimenzije $p \times m$
$\mathbf{Y}, \mathbf{Y}' = \mathbf{C} * \mathbf{D}$	Sažimanje ulaznih matrica
$\mathbf{U}, \mathbf{S}, \mathbf{V} = \text{svd}(\mathbf{Y}, k)$	Odrezani SVD
$\tilde{\mathbf{A}} = \mathbf{U}^* * \mathbf{Y}' * \mathbf{V} * \mathbf{S}^{-1}$	Metoda najmanjih kvadrata
$\mathbf{W}, \mathbf{\Lambda} = \text{eig}(\tilde{\mathbf{A}})$	Spektralna dekompozicija
$\mathbf{\Phi} \leftarrow \mathbf{X}' * \mathbf{V} * \mathbf{S}^{-1} * \mathbf{W}$	Puni modovi $\mathbf{\Phi}$.
$\mathbf{b} = \text{lstsq}(\mathbf{\Phi}, \mathbf{x}_1)$	Računanje amplitude koristeći $\mathbf{x}_1$ kao početni uvjet
$\mathbf{V} = \text{vander}(\text{diag}(\mathbf{\Lambda}))$	Vandermondeova matrica (proizvoljan korak).

4.5 Usporedba sa standardnim DMD-om

U ovom poglavlju skrenuli smo pozornost k ubrzavanju algoritma, no još uvijek se nismo osvrnuli na pouzdanost rješenja. Ima li smisla znatno ubrzati izvođenje ukoliko ćemo pritom platiti cijenu nepouzdanim i netočnim rezultatom? U ovom poglavlju na jednostavnom primjeru usporedit ćemo točnost i brzinu cDMD-a i standardnog DMD-a. Pokazat ćemo da je prelazak na cDMD opravdan. Primjer koji koristimo inspiriran je s [4].

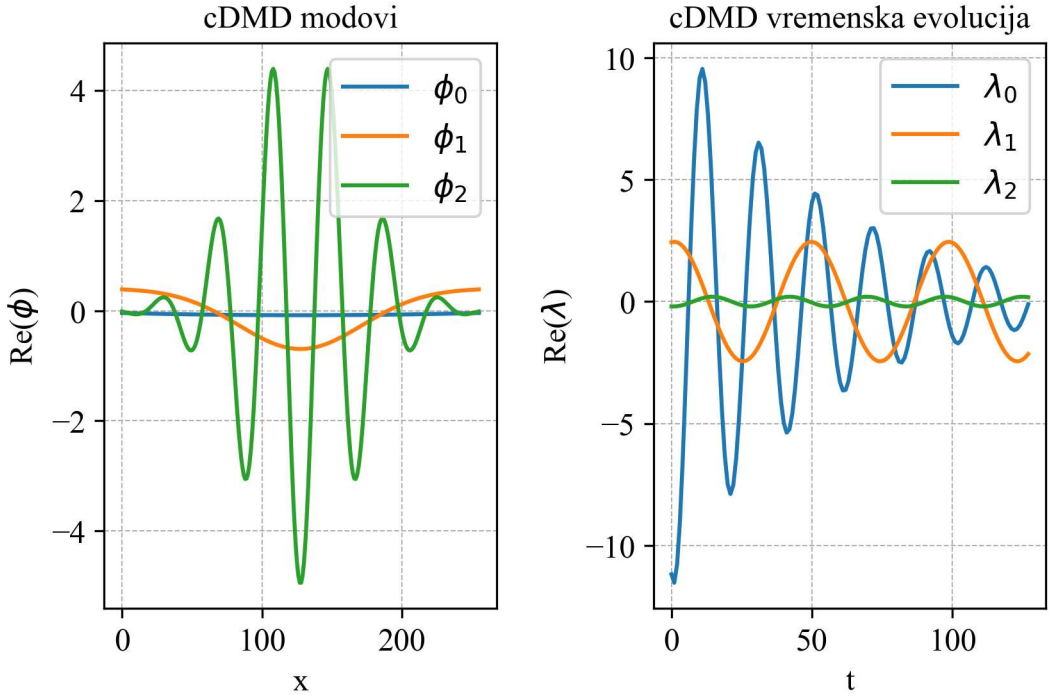
Kao i u ranijem primjeru 2.28, funkcija koju ćemo evaluirati dobivena je kao suma nekoliko baznih funkcija. Neka je dano

$$\begin{aligned} f(x,t) &= f_1(x,t) + f_2(x,t) + f_3(x,t), \\ f_1(x,t) &= e^{-\frac{x^2}{5}}\ \cos(4x)\, e^{(2.3i)t}, \\ f_2(x,t) &= (1 - e^{1-\frac{x^2}{6}})e^{(1.3i)t}, \\ f_3(x,t) &= \left(-\frac{x^2}{50} + 1\right) 1.1i^{-2t}, \end{aligned}$$

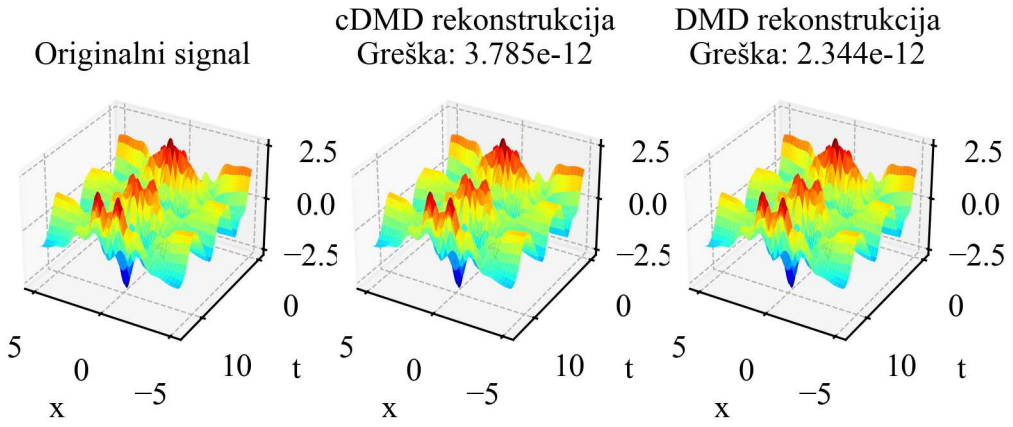
pri čemu je i standardna oznaka za imaginarnu jedinicu.

Na slici 4.3 možemo vidjeti kako izgledaju DMD modovi te vremenska evolucija dobivena cDMD metodom, dok skroz lijeva sličica 4.4 prikazuje isječak originalne funkcije u vremenu i prostoru. Osim toga, slika 4.4 prikazuje usporedbu rekonstrukcije standardnim DMD-om i cDMD-om. Vizualno je jasno da su vrlo slične, a greške reda veličine 10^{-12} to potvrđuju. Greška je zanemariva, a vremenska ušteda je značajna, što se lako uočava

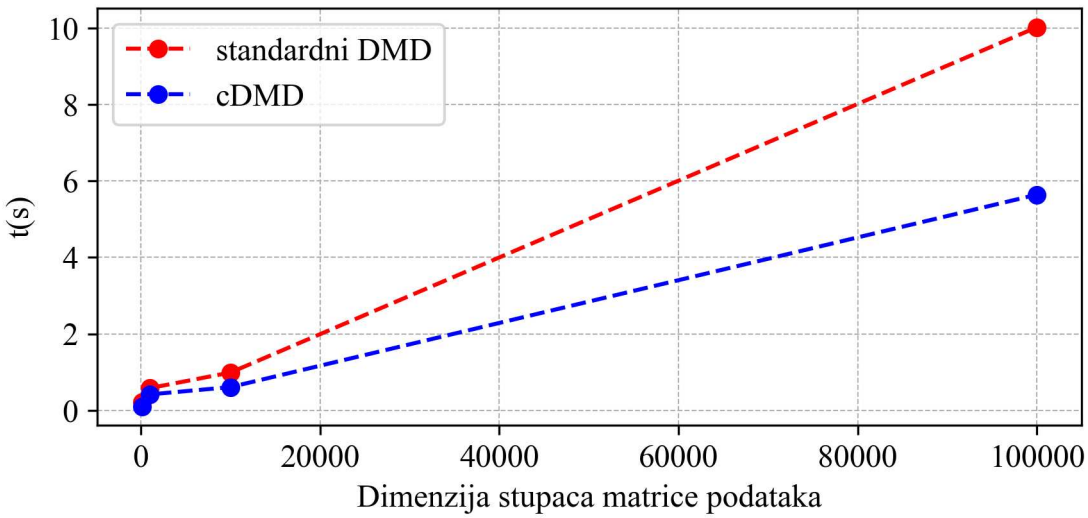
na slici 4.5. Porastom dimenzije sve više se osjeti značaj cDMD optimizacije, a očito je da slike i videozapisi zbog klasičnih rezolucija vrlo brzo postaju visoko-dimenzionalni problemi.



Slika 4.3: Modovi i vremenska evolucija dobiveni cDMD-om. Jasno se vidi kako se originalni signal ponaša u vremenu i prostoru.



Slika 4.4: Lijevo: isječak originalnog signala u vremenu i prostoru. Sredina: rekonstrukcija signala i greška dobiveni standardnom cDMD metodom. Desno: rekonstrukcija signala i greška dobiveni DMD metodom. Uočimo da je greška dvostruko veća kod cDMD-a, ali je zanemariva jer se radi o redu veličine 10^{-12}.



Slika 4.5: Usporedba brzine izvođenja standardnog DMD-a i cDMD-a u ovisnosti o dimenziji pojedinačnog stupca u matrici podataka. Napomenimo da točne brojke na y osi nisu nužno bitne jer ovise o performansama računala na kojem se izvodi. Bitno je za uočiti relativan omjer i usporedbu krivulja.

Poglavlje 5

Evaluacija cDMD-a

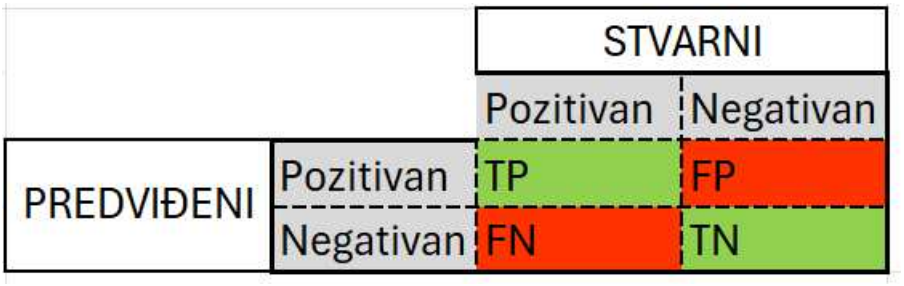
U dosadašnjem dijelu ovog rada imali smo priliku testirati DMD i varijacije DMD-a na jednostavnim primjerima, no htjeli bismo imati pouzdanu metodu i za probleme iz stvarnog svijeta. No, prije svega, moramo znati na pravilan način evaluirati metodu. Kako možemo vrednovati radi li *dobro* i što uopće znači *dobro* u ovom kontekstu?
Problem s kojim se hvatamo u koštac jest separacija pozadine od prvog plana u videozapisu. Jezikom računala to su, zapravo, *jedinice* (1), prvi plan, i *nule* (0), pozadina. Prilikom testiranja metode potrebno nam je unaprijed znati **stvarne vrijednosti** (eng. *ground truth*). Dakle, htjeli bismo izlaz neke varijacije DMD metode prevesti u nule i jedinice kako bismo mogli to usporediti sa stvarnim vrijednostima. Time smo problem u konačnici sveli na problem tzv. **binarne klasifikacije**. U ovom poglavlju prikazat ćemo rezultate cDMD metode na raznim skupovima podataka. Osim toga, definirat ćemo metrike kojima mjerimo uspješnost, vizualizirat ćemo neke međukorake cDMD-a kao što su DMD modovi i vremenske evolucije i pokazati konačni izlaz algoritma.

5.1 Matrica zabune i mjere uspješnosti

Kao što smo već dosta puta spomenuli, srž problema ovog rada jest odvajanje pozadine od prvog plana. Naime, u videozapisu bismo htjeli detektirati, primjerice, pješake, automobile, bicikle, bilo koje objekte u pokretu. Odemo li korak dalje do analize snimki videonadzora, onda je iz sigurnosnih razloga vrlo važno uočiti sve objekte koji se miču. Uzevši to u obzir, definirat ćemo metrike kojima ćemo mjeriti performanse DMD metode, a priroda problema navodi nas prvo do matrice zabune.

Matrica zabune

Matrica zabune (eng. *confusion matrix*) grafički je prikaz predviđenih i stvarnih vrijednosti klasa koji nam pomaže prilikom računanja mjera uspješnosti klasifikacijskih modela. Prikazujemo ju kao na slici 5.1.



Slika 5.1: Definicija matrice konfuzije. U našem slučaju *pozitivan* jest prvi plan, dok je pozadina *negativna*. U praksi se koriste engleske kratice za ove vrijednosti. Preciznije, TP (eng. *true positive*), FP (eng. *false positive*), FN (eng. *false negative*) te TN (eng. *true negative*).

Preciznost

Preciznost (eng. *precision*) mjera je uspješnosti definirana kao omjer ispravno predviđenih pozitivnih promatranja i ukupnog broja predviđenih pozitivnih. O toj mjeri možemo razmišljati kao o odgovoru na pitanje: *"Od svih instanci koje su predviđene kao pozitivne, koliko ih je* ***uistinu pozitivno"****?*

$$\text{preciznost} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{5.1}$$

Odziv

Odziv (eng. *recall*) mjera je uspješnosti definirana kao omjer ispravno predviđenih promatranja i ukupnog broja stvarno pozitivnih. Odgovara na pitanje: *"Od svih stvarnih pozitivnih instanci, koliko ih je* ***ispravno*** *predviđeno?* U literaturi se može još pronaći pod nazivom osjetljivost (eng. *sensitivity*). Izrazito je bitna u situacijama kada ne bismo htjeli propustiti pozitivan slučaj, primjerice u medicini ili u videonadzoru.

$$\text{odziv} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{5.2}$$

F1 ocjena

Niti preciznost niti odziv nisu suviše korisne ako ih gledamo zasebno budući da je jednostavno manipulirati rezultatima. Iz tog razloga definiramo $F1$ ocjenu (eng. $F1$ *score*), harmonijsku sredinu preciznosti i odziva. Vrlo je važna jer balansira preciznost i odziv, za razliku od, primjerice, aritmetičke sredine. Osim što jednom brojkom obuhvaća više različitih mjera, također daje relevantnije rezultate u slučaju kada je negativna klasa znatno brojnija od pozitivne, što je svakako slučaj u našem problemu pozadine i prvog plana. Naime, pozadina zauzima većinu pojedine slike u videozapisu. Sljedećom formulom definiramo $F1$ mjeru, a kada raspišemo do kraja, jasno je da više značaja pridaje pozitivnoj klasi:

$$\mathrm{F1} = \frac{2}{\frac{1}{\text{preciznost}} + \frac{1}{\text{odziv}}} = \frac{2 \times \text{preciznost} \times \text{odziv}}{\text{preciznost} + \text{odziv}} = \frac{\mathrm{TP}}{\mathrm{TP} + \frac{1}{2}(\mathrm{FP} + \mathrm{FN})}. \tag{5.3}$$

Matthewseov koeficijent korelacije

Uzevši u obzir da radimo s nebalansiranim klasifikacijskim problemom, moramo vješto odabrati pogodne metrike za mjerenje uspjeha. U tom kontekstu pojavljuje se i Matthewseov koeficijent korelacije, kraće MCC. Iako nije čest u strojnom učenju, MCC je zanimljiv zato što, slično kao i $F1$ mjera, koristi matricu konfuzije, a vraća samo jedan broj, odnosno jednu ocjenu. Međutim, glavna razlika leži u činjenici da MCC uzima u obzir sva 4 podatka u matrici konfuzija, dok $F1$ mjera izostavlja TN. MCC je definiran sljedećom formulom: [1]

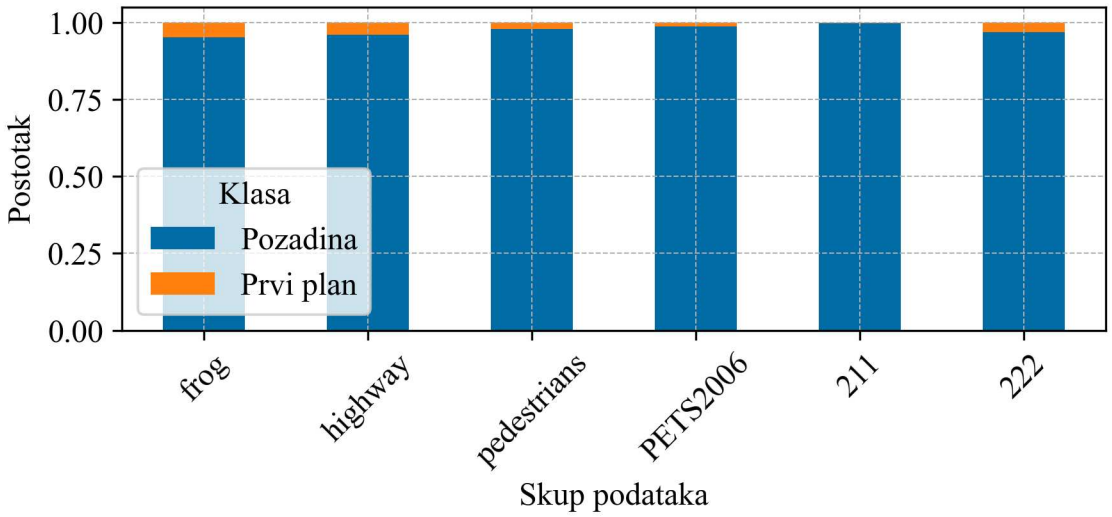
$$MCC = \frac{TN \times TP - FP \times FN}{\sqrt{(TN + FN)(FP + TP)(TN + FP)(FN + TP)}}. \tag{5.4}$$

Iz naziva je jasno da se radi o koeficijentu korelacije. Dakle, MCC poprima vrijednosti od -1 do 1. Davide Chicco [2], autor knjige *10 brzih savjeta za strojno učenje u računalnoj biologiji*, o MCC-u kaže: *MCC je visok samo ako je vaš klasifikator točan i na pozitivnim i na negativnim elementima*.

Napomena 5.1.1. *Lako je pokazati da radimo s nebalansiranim klasifikacijskim problemom promatrajući sliku 5.2. Odabrana je nasumična sličica stvarnih maski za neke od skupova podataka s kojima radimo.*

[1] Primijetimo da se u nazivniku množe 4 potencijalno jako velika broja, stoga treba biti oprezan da ne dođe do tzv. *overflow-a*

[2] eng. "Ten quick tips for machine learning in computational biology"



Slika 5.2: Omjer piksela koji pripadaju pozadini i prvom planu na nasumičnim slikama iz spomenutih skupova podataka. Iako je stvarni omjer drukčiji, graf nam daje barem uvid da se radi o prilično nebalansiranom klasifikacijskom problemu.

5.2 Skupovi podataka

DMD metoda testirana je i evaluirana na mnoštvu različitih javno dostupnih skupova podataka za problem separacije pozadine i prvog plana. Pojedinačni skupovi podataka pripadaju većim familijama skupova podataka, stoga ćemo ih tako i navesti. U tablici 5.1 mogu se pronaći nazivi svih korištenih skupova podataka, kao i neki metapodaci. Iako je naveden ukupan broj sličica koji je korišten, valja napomenuti da su svi videozapisi podijeljeni u segmente po 100 sličica radi lakšeg izvođenja.

Već iz naziva pojedinačnih skupova podataka jasno nam je koja je radnja u videozapisu, izuzev skupova iz *BMC synthetic* familije. S obzirom da su to umjetno kreirane scene, prilažemo nomenklaturu po kojoj slijede nazivi scena [28].
Ukupno postoje dva tipa scena (znamenka na mjestu *desetica* u imenu):

1. ulica;
2. kružni tok.

Za svaku scenu postoji pet tipova događaja (znamenka *stotica* u imenu):

1. oblačno, bez šuma u akviziciji;
2. oblačno, sa šumom;

3. sunčano, sa šumom;
4. maglovito, sa šumom;
5. vjetrovito, sa šumom.

Za svaku scenu razlikuju se dva scenarija korištenja (znamenka na mjestu *jedinica) u imenu*):

1. 10 sekundi bez objekata, zatim pokretni objekti tijekom 50 sekundi;
2. 20 sekundi bez događaja, zatim događaj (npr. izlazak sunca ili magla) tijekom 20 sekundi, na kraju 20 sekundi bez događaja.

5.3 Parametri modela

Izuzev početnih jednostavnih primjera, u ovom radu korišten je programski jezik Python, pri čemu je najvažnija biblioteka **PyDMD** [4], [15]. S obzirom da je javno dostupan cijeli izvorni kod i dokumentacija je opsežna, nećemo ulaziti u detalje. No, za potrebe reproduciranja rezultata važno je navesti parametre koji su korišteni, a nalaze se u tablici 5.2.

Napomena 5.3.1. *Prilikom pisanja ovog rada dao sam doprinos razvoju **PyDMD** paketa otkrivanjem i rješavanjem grešaka. Detalji se mogu pronaći na [18].*

5.4 Postprocesuiranje

Neka nam je dana cDMD rekonstrukcija $\mathbf{X}_{\text{cDMD}}$. Metodom opisanom na kraju poglavlja 3.2 dobijemo $\mathbf{X}_{\text{cDMD}}^{\text{niži-rang}}$ i $\mathbf{X}_{\text{cDMD}}^{\text{rijetka}}$. Tada je, u 3.10, spomenuto da će pozadinu najčešće činiti samo jedna svojstvena vrijednost najbliža ishodištu. U stvarnim situacijama ispostavilo se da to često nedovoljno dobro opisuje pozadinu (eng. *underfitting*). Tom problemu doskočili smo tako što modeliramo pozadinu uprosječivanjem varijabilnog broja modova, što se jasno vidi na slici 5.3. Lako se zaključi da jednim modom dobijemo dosta šuma koji je, izgleda, dobiven miješanjem slika iz sekvence, odnosno videozapisa. Na slici 5.4 mogu se vidjeti primjeri snimki s početka, sredine i s kraja jednog segmenta iz *highway* skupa podataka, što dočarava dinamiku tog segmenta.

Modelirali smo pozadinu, no nismo još uvijek gotovi. Htjeli bismo u konačnici svaku sliku u videozapisu svesti na samo 2 klase, 0 i 1, pozadinu i prvi plan, jer ćemo rješenje uspoređivati s binarnom maskom, odnosno stvarnim rješenjem (eng. *ground truth*) u kojem

Tablica 5.1: Korišteni skupovi podataka grupirani po familiji te s pripadajućim metapodacima.

Naziv familije	**Ime**	**Broj sličica**	**Rezolucija**
Segtrack [21]	frog	279	480x264
Change detection: baseline [13]	highway	1231	320x240
	pedestrians	800	360x240
	PETS2006	901	720x576
BMC synthetic 1 [28]	111	1499	640x480
	112	1502	640x480
	121	1499	640x480
	122	1503	640x480
	211	1499	640x480
	212	1499	640x480
	221	1499	640x480
	222	1499	640x480
	311	1499	640x480
	312	1499	640x480
BMC synthetic 2 [28]	321	1501	640x480
	322	1499	640x480
	412	1499	640x480
	421	1499	640x480
	422	1499	640x480
	512	1499	640x480
	521	1499	640x480
	522	1499	640x480

su pikseli s vrijednostima 0 pozadina, a 1 su prvi plan. U digitalnoj obradi slike najjednostavniji način za tzv. *binariziranje* slike jest korištenje globalnog praga (eng. *global threshold*). Naime, sve vrijednosti ispod nekog praga sive "odrezat" ćemo i dodijelit im vrijednost 0, dok će ostale vrijednosti postati 1. U suštini to je jednostavna stepenasta funkcija.

Postavlja se pitanje kako odrediti koja vrijednost praga je prikladna? Naravno, to ovisi o distribuciji intenziteta pojedinih piksela u slici. Koristili smo nešto napredniji način koji se često pojavljuje u računalnom vidu i obradi slike, tzv. Otsuovu binarizaciju, no nećemo ulaziti u detalje. [3] S obzirom da su u binarnoj masci u stvarnom rješenju elementi koji

[3] Ukratko, ideja je da se iz histograma intenziteta piksela pronađe prag koji optimalno dijeli sliku na točno

Tablica 5.2: Parametri korišteni za cDMD

Parametar	**Vrijednost**	**Kratki opis**
SVD rank	0	Rang 0 označava da će se izračunati optimalan rang za redukciju dimenzionalnosti r u SVD metodi. Optimalan rang računa se po uzoru na tzv. *optimal hard-threshold* iz [11]
compression matrix	”normal”	Matrica kompresije
opt	True	Računa optimalne amplitude kao u [16]
sorted eigs	”abs”	Sortira svojstvene vrijednosti (i modove te pripadajuće dinamike) padajuće po njihovom modulu.

predstavljaju prvi plan *kompaktni*, odnosno nemaju *rupe* u sebi, odlučili smo se za dva postupka postprocesuiranja. Prvo smo, nakon Otsuove binarizacije, napravili morfološko zatvaranje (eng. *morphological closing)* [4] jezgrom veličine 3×3 kako bismo pokušali zatvoriti rupe na prvom planu, a zatim smo počistili šum medijan-filtrom s istom jezgrom. Konačan rezultat vidljiv je na slikama 5.5, 5.6 i 5.7 gdje smo obuhvatili po jednu snimku gotovo svakog skupu podataka za testiranje.

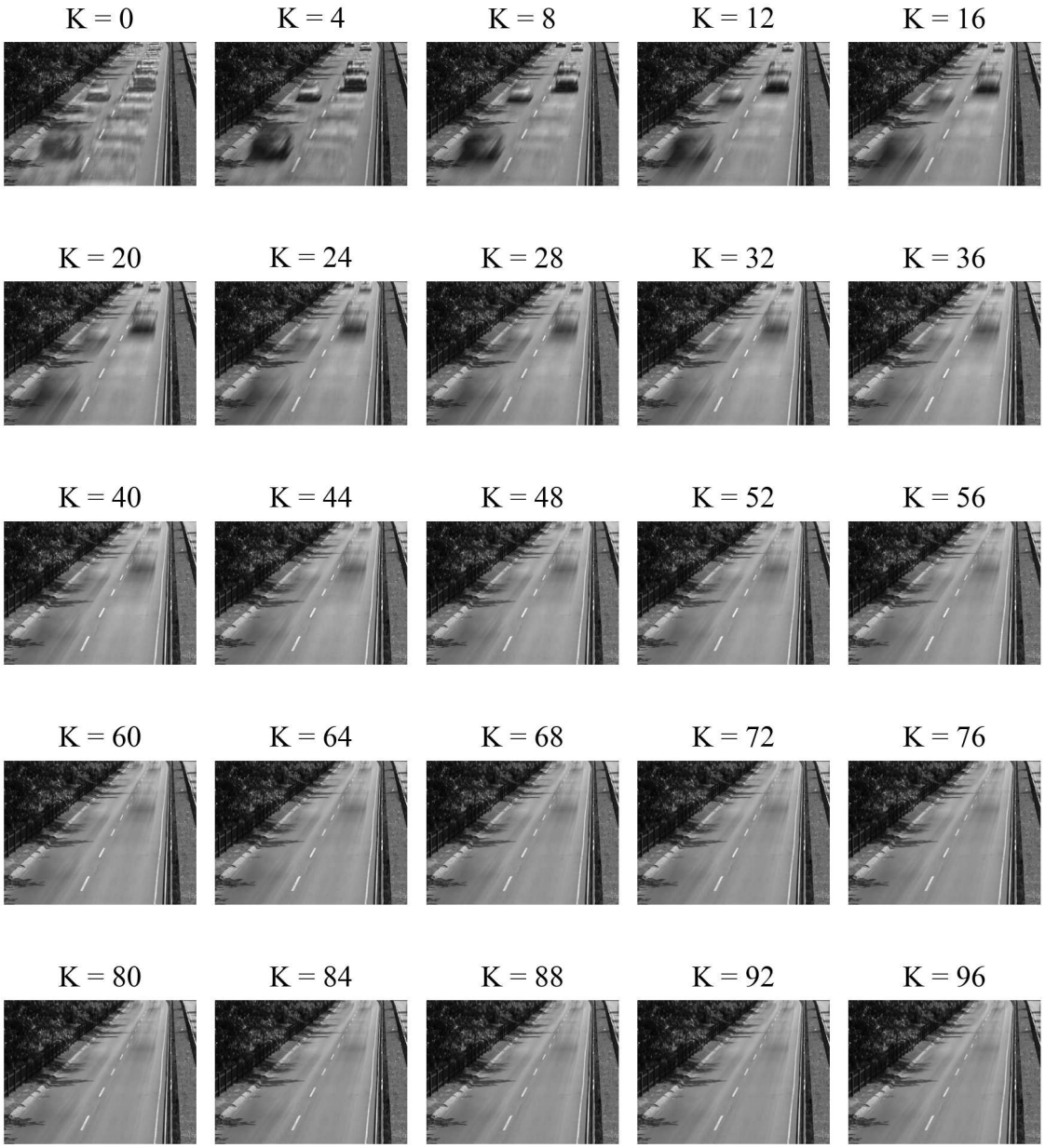
Na *highway* i *pedestrians* primjerima možemo detaljnije pogledati međukorake cijelog postupka. Primjerice, vrlo su informativne slike 5.8 i 5.9.

U gornjem lijevom kutu vidimo singularne vrijednosti modela, pri čemu su narančastom bojom obojane one koje su uvrštene u model, čime iščitavamo parametar redukcije dimenzionalnosti r u sklopu odrezanog SVD-a. Nadalje, isti red još čine diskretne, odnosno neprekidne svojstvene vrijednosti, definirane kao u poglavlju 2.5. Veličina točaka kojima su označene svojstvene vrijednosti proporcionalna je amplitudi, dok boja odgovara prostornoj dinamici modova u drugom redu te vremenskoj dinamici u trećem redu.

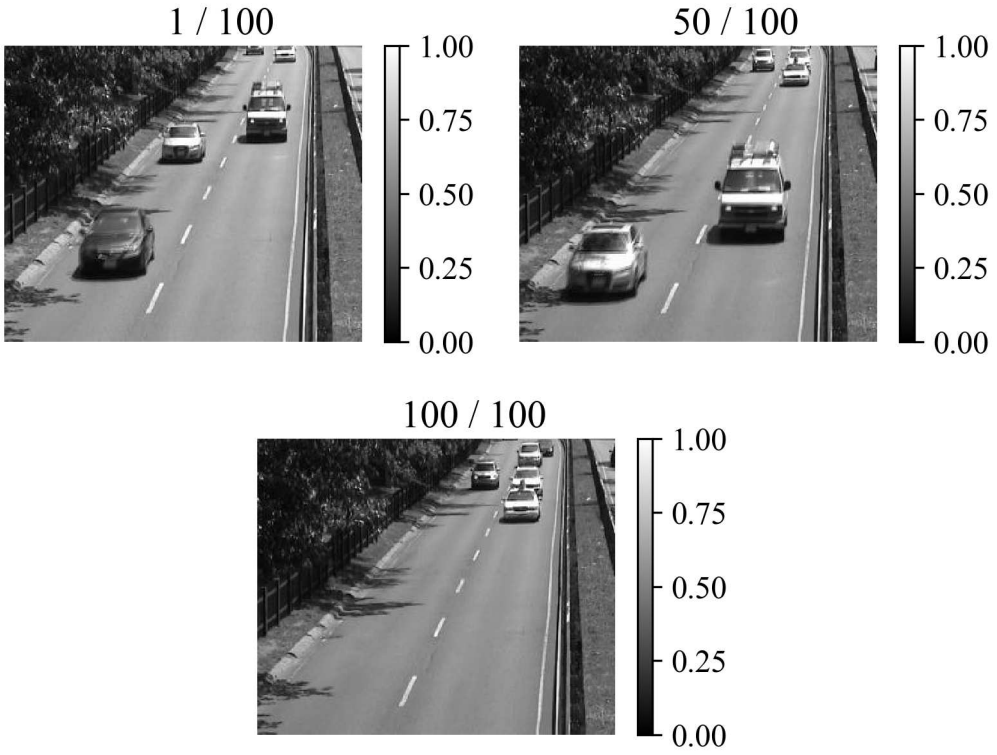
Ključno za uočiti jest da neprekidna svojstvena vrijednost najbliža ishodištu nosi većinu informacija o pozadini, što je jasno vidljivo u drugom redu na lijevoj slici. To se točno podudara s teorijom koju smo ranije razvili. No, isto tako vidimo da i drugi modovi nose

2 klase. Intuitivno je jasno da će to izvrsno raditi u slučaju bimodalnog histograma, odnosno histograma u kojem se jasno naziru 2 vrha odvojena od ostatka intenziteta. Više o tome možete saznati na [5]

[4]Mofološko zatvaranje (eng. *morphological closing)* u obradi slike naziv je za kombinaciju dilatacije i erozije jednu za drugom. Korisno je kada u binarnoj slici želimo zatvoriti sitne rupe na objektima prvog plana ili kada želimo počistiti točkice pozadine koje se nalaze na elementima prvog plana. Više o tome može se saznati na [6].



Slika 5.3: Prikaz problema gdje jedan mod ne može dovoljno dobro opisati pozadinu. Na ovim slikama vidi se model pozadine dobiven uprosječivanjem K modova, od 0 do 96. Konačan model, koji se ne vidi na ovoj slici, uzet je za $K = 100$.

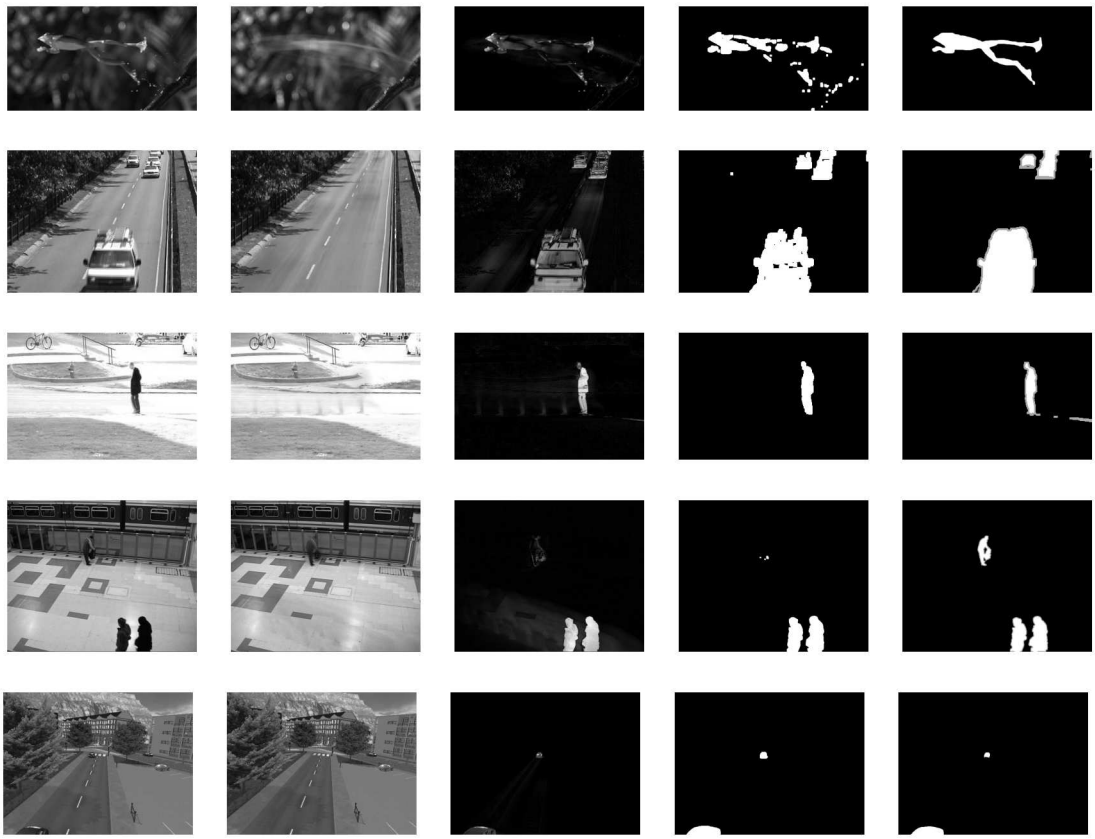


Slika 5.4: Primjer početne, srednje te završne snimke jednog segmenta iz skupa podataka *highway*.

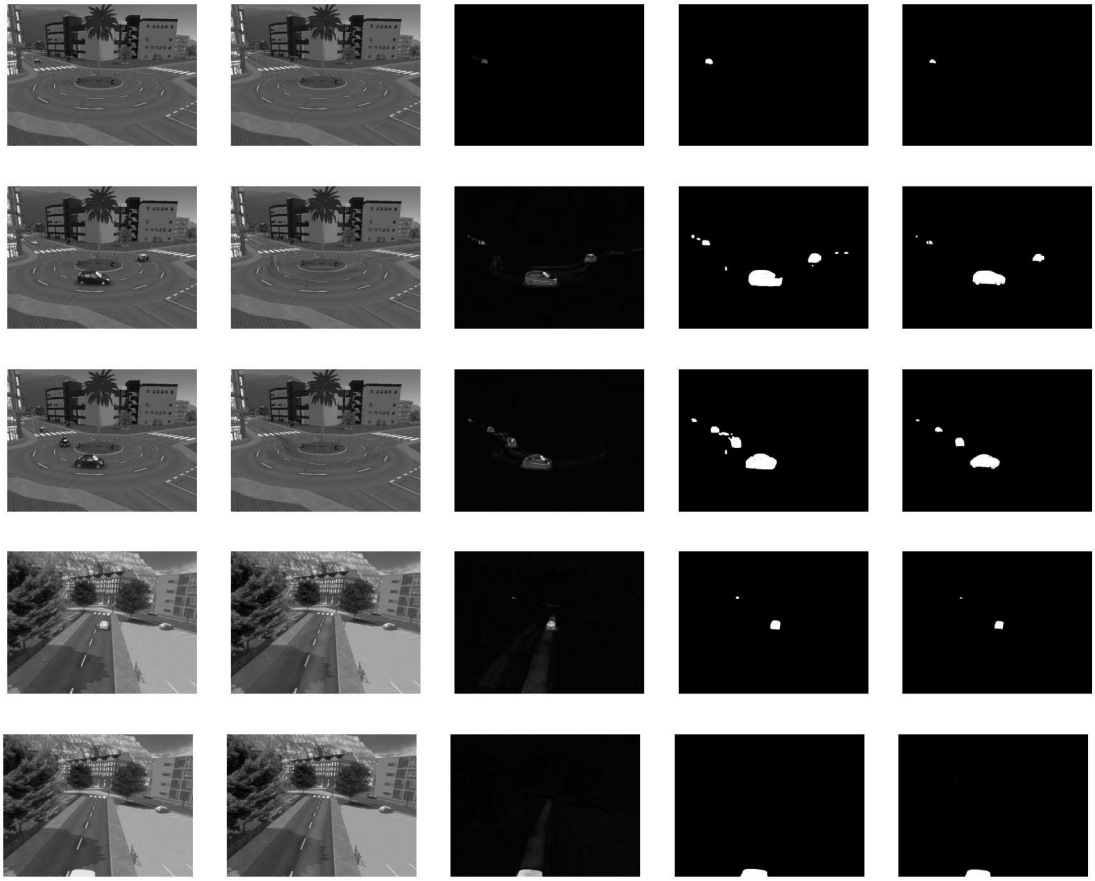
određeni dio informacija, čiju semantiku možemo prepoznati u samom modu. Graf vremenske dinamike prikazuje u kojim sličicama videozapisa je najveći utjecaj tog modda. Skala za pozadinu je vrlo stabilna, nema znatnih promjena, dok ostali modovi znatno više fluktuiraju. Naravno, jasno je da su prikazana samo tri moda, i to najvažnija po amplitudi. Uočimo, svojstvene vrijednosti dolaze u konjugirano-kompleksnim parovima, što je upravo simetričnost oko x osi.

5.5 Kvantitativni rezultati

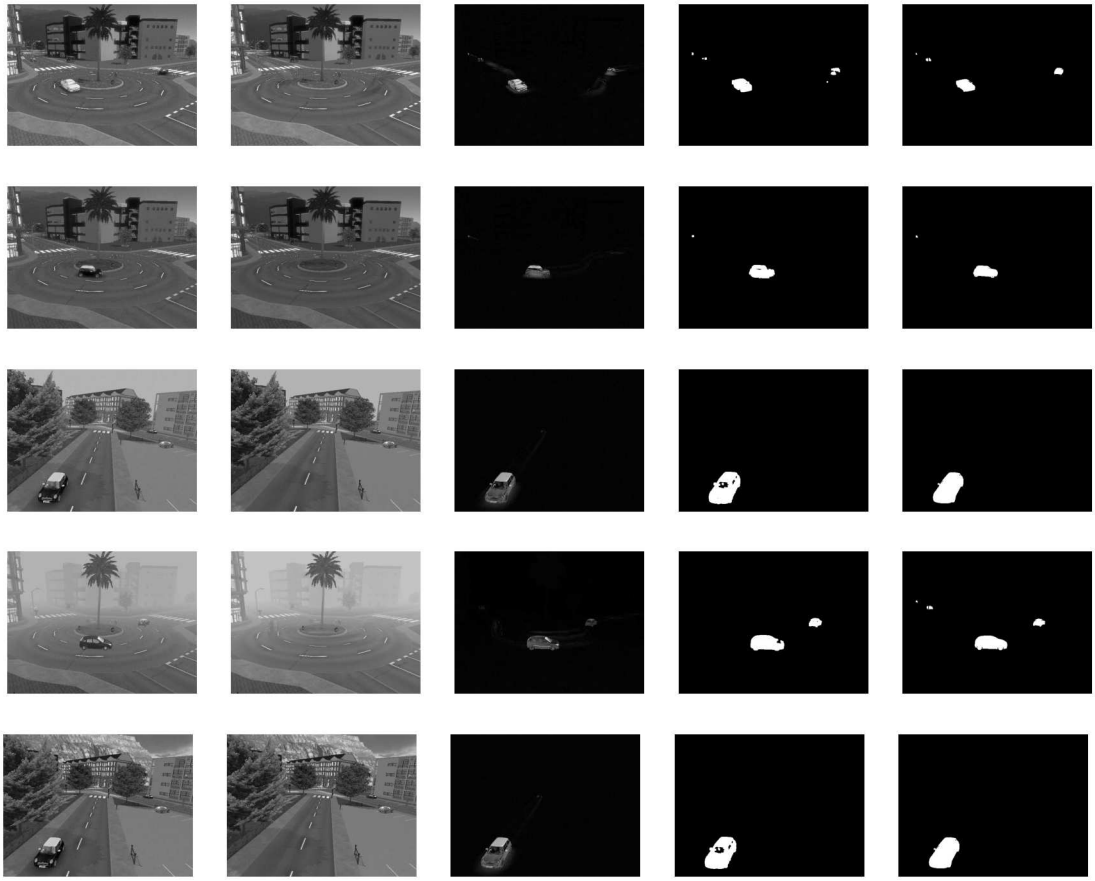
U ovom poglavlju uveli smo definicije četiriju metrika koje smo koristili za usporedbu rješenja. Sada ćemo tablično predstaviti rezultate na najuspješnijem segmentu svakog pojedinog skupa podataka. Osim toga, navest ćemo i rezultate za najbolja 3 i najboljih 5 segmenata, no radi jednostavnosti i preglednosti u ovim kategorijama odabrano je samo 10 najuspješnijih skupova podataka.



Slika 5.5: Izlazni model od originalne slike do konačnog rezultata nakon postprocesuiranja. Originalnu sliku vidimo u prvom stupcu. Pozadina i prvi plan (2. i 3. stupac) dobiveni su na način opisan u ovom poglavlju, dok u 4. stupcu vidimo rezultat nakon Otsuove binarizacije popraćene morfološkim zatvaranjem s jezgrom veličine 3×3 te medijan-filtrom veličine 3×3 kako bismo počistili šum. Peti stupac prikazuje stvarnu masku. Skupovi podataka počevši odozgo: *frog, highway, pedestrians, PETS*2006*,* 111.



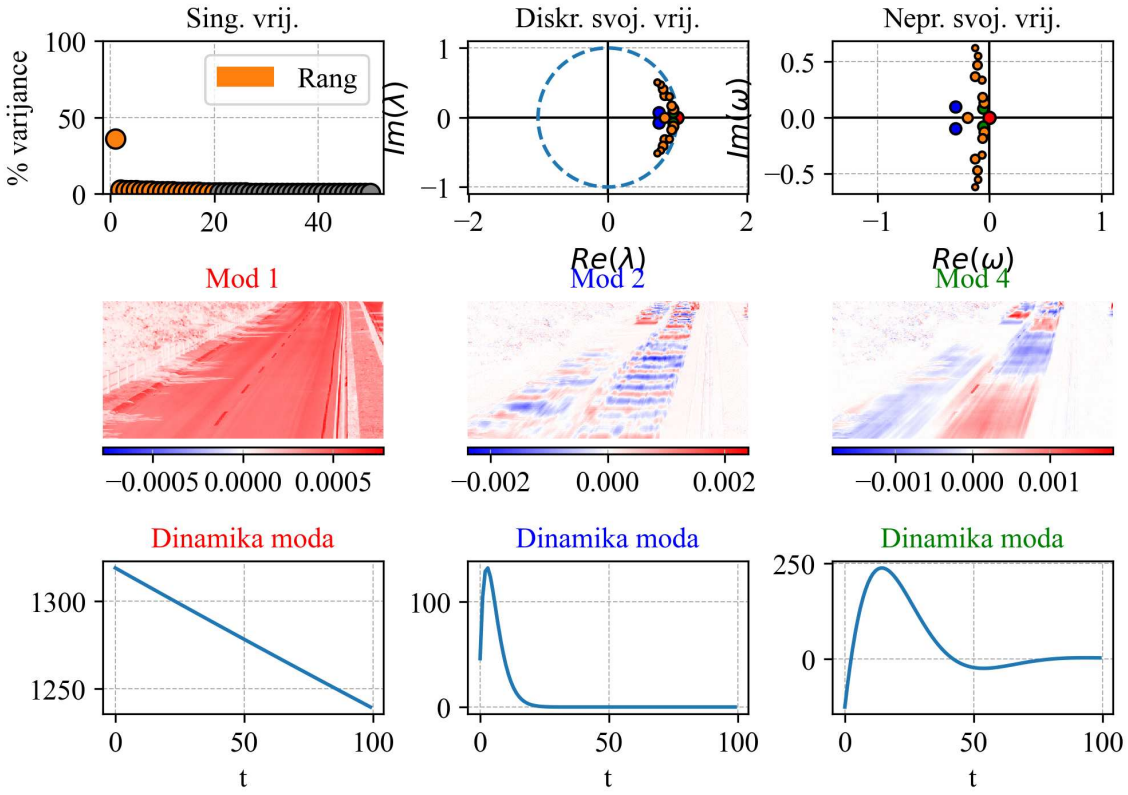
Slika 5.6: Izlazni model od originalne slike do konačnog rezultata nakon postprocesuiranja. Originalnu sliku vidimo u prvom stupcu. Pozadina i prvi plan (2. i 3. stupac) dobiveni su na način opisan u ovom poglavlju, dok u 4. stupcu vidimo rezultat nakon Otsuove binarizacije popraćene morfološkim zatvaranjem s jezgrom veličine 3×3 te medijan-filtrom veličine 3×3 kako bismo počistili šum. Peti stupac prikazuje stvarnu masku. Skupovi podataka počevši odozgo: $122, 221, 222, 311, 312$.



Slika 5.7: Izlazni model od originalne slike do konačnog rezultata nakon postprocesuiranja. Originalnu sliku vidimo u prvom stupcu. Pozadina i prvi plan (2. i 3. stupac) dobiveni su na način opisan u ovom poglavlju, dok u 4. stupcu vidimo rezultat nakon Otsuove binarizacije popraćene morfološkim zatvaranjem s jezgrom veličine 3×3 te medijan-filtrom veličine 3×3 kako bismo počistili šum. Peti stupac prikazuje stvarnu masku. Skupovi podataka počevši odozgo: $321, 322, 412, 421, 512$.

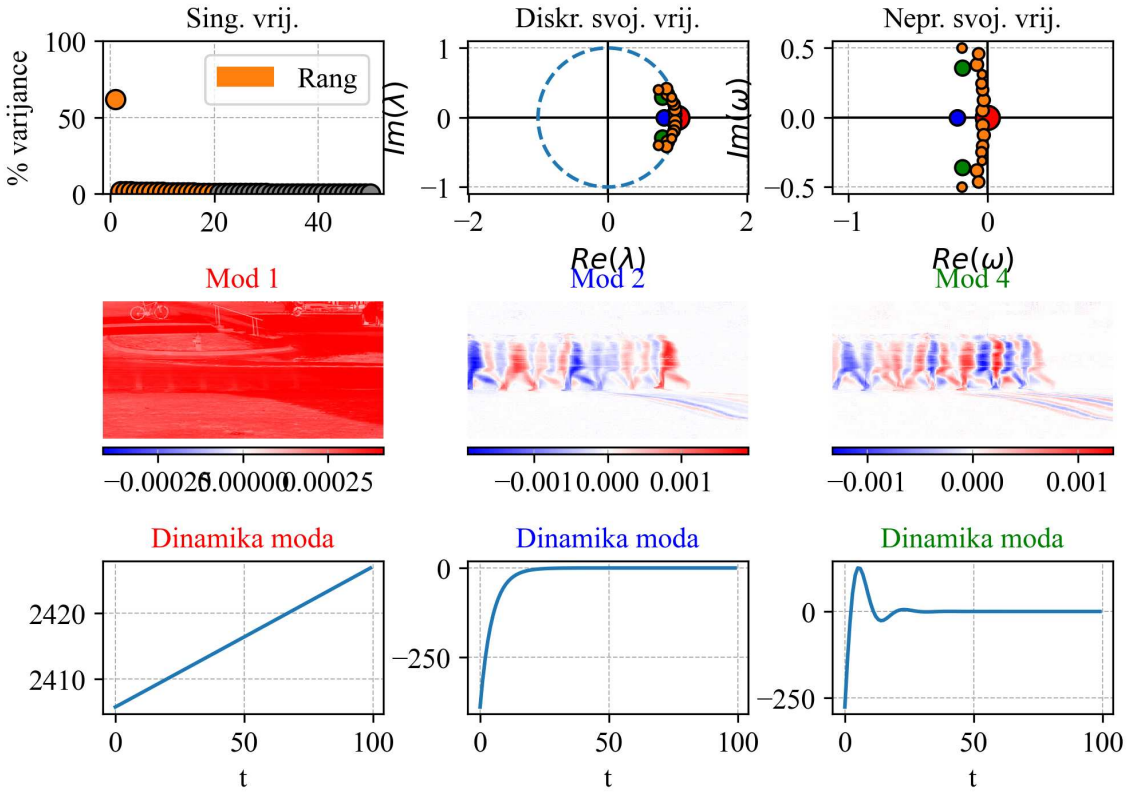
Tablica 5.3: Vrijednosti metrika na najboljem segmentu svih korištenih skupova podataka.

Skup podataka	Preciznost	Odziv	F1	MCC
pedestrians	0.822	0.933	0.874	0.874
PETS2006	0.857	0.782	0.800	0.801
421	0.813	0.759	0.783	0.782
111	0.666	0.938	0.768	0.783
112	0.701	0.841	0.759	0.764
212	0.679	0.925	0.752	0.761
522	0.665	0.861	0.750	0.754
222	0.662	0.862	0.748	0.752
211	0.703	0.875	0.742	0.756
highway	0.800	0.688	0.738	0.718
221	0.690	0.760	0.722	0.720
422	0.640	0.812	0.712	0.718
512	0.647	0.850	0.692	0.703
521	0.522	0.821	0.636	0.649
412	0.601	0.854	0.635	0.649
321	0.637	0.623	0.629	0.627
121	0.498	0.849	0.626	0.649
122	0.463	0.925	0.604	0.645
312	0.598	0.883	0.603	0.616
311	0.645	0.766	0.580	0.613
office	0.519	0.734	0.576	0.579
322	0.581	0.596	0.560	0.568
frog	0.725	0.441	0.532	0.534
411	0.500	0.936	0.515	0.541
511	0.297	0.913	0.363	0.420



Slika 5.8: Sažetak rezultata cDMD algoritma za 11. segment iz skupa podataka *highway*. Gornji red prikazuje singularne vrijednosti i svojstvene vrijednosti, srednji red prostornu dinamiku DMD modova, dok zadnji red vremensku dinamiku. Prvi mod nosi većinu informacija koje čine pozadinu.

Na početku ovog poglavlja objasnili smo da je $F1$ metrika najznačajnija metrika koju ćemo promatrati u sklopu analiziranja rezultata. U tablici 5.3 možemo vidjeti da naš model daje vrlo visoke vrijednosti po tom kriteriju, pri čemu je na samo jednom skupu podataka $F1$ ispod 0.5, i to radi niske preciznosti. Uočimo da je na čak 12 skupova podataka postignuta vrijednost veća od 0.7 za $F1$ metriku, što je sjajno. Preostale dvije tablice, 5.4 i 5.5, sadrže prosjeke najbolja 3, odnosno najboljih 5 segmenata, a jasno je da su rezultati i dalje vrlo visoki, posebice ako uzmemo u obzir da se svaki segment sastoji od 100 slika. Metrike na pojedinačnim slikama iz segmenta vidljive su na slikama 5.12 i 5.13. Ovime smo pokazali da se cDMD algoritam može dostojno nositi s problemom modeliranja pozadine i prvog plana. Što se tiče performansi usporediv je i s vodećim algoritmima po tom pitanju,



Slika 5.9: Sažetak rezultata cDMD algoritma za 4. segment iz skupa podataka *pedestrians*. Gornji red prikazuje singularne vrijednosti i svojstvene vrijednosti, srednji red prostornu dinamiku DMD modova, dok zadnji red vremensku dinamiku. Prvi mod nosi većinu informacija koje čine pozadinu.

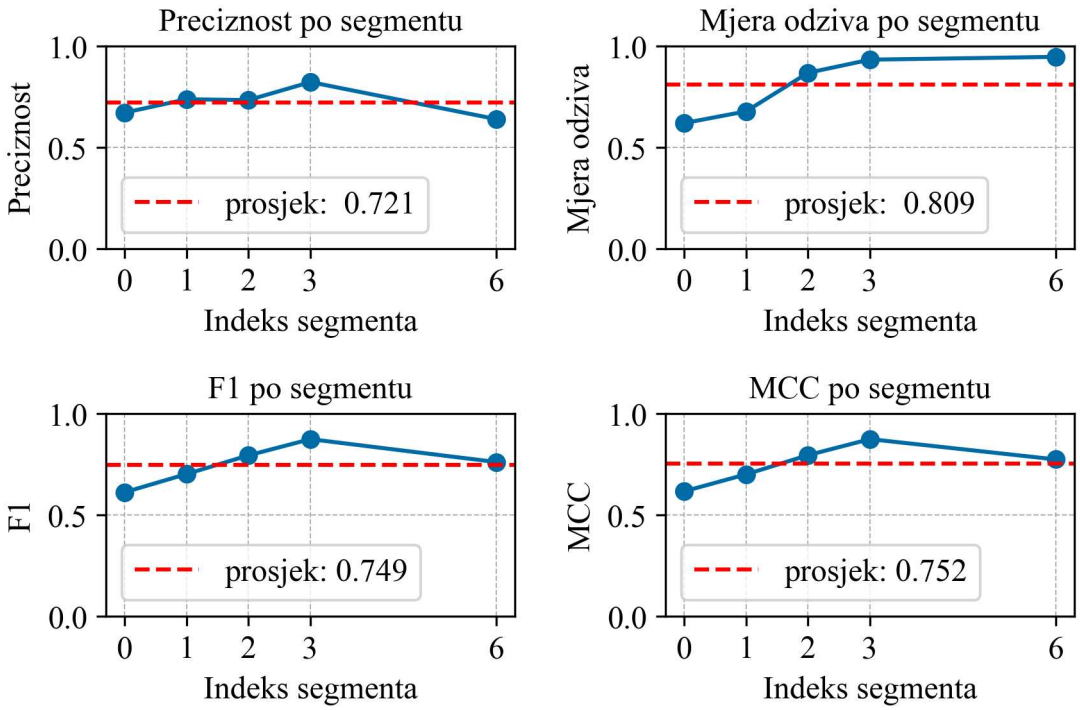
čiji se rezultati mogu iščitati u [10].

Tablica 5.4: Vrijednosti četiriju metrika za prosjek 3 najuspješnija segmenta. Prikazano je 10 skupova podataka sortirano po $F1$ metrici.

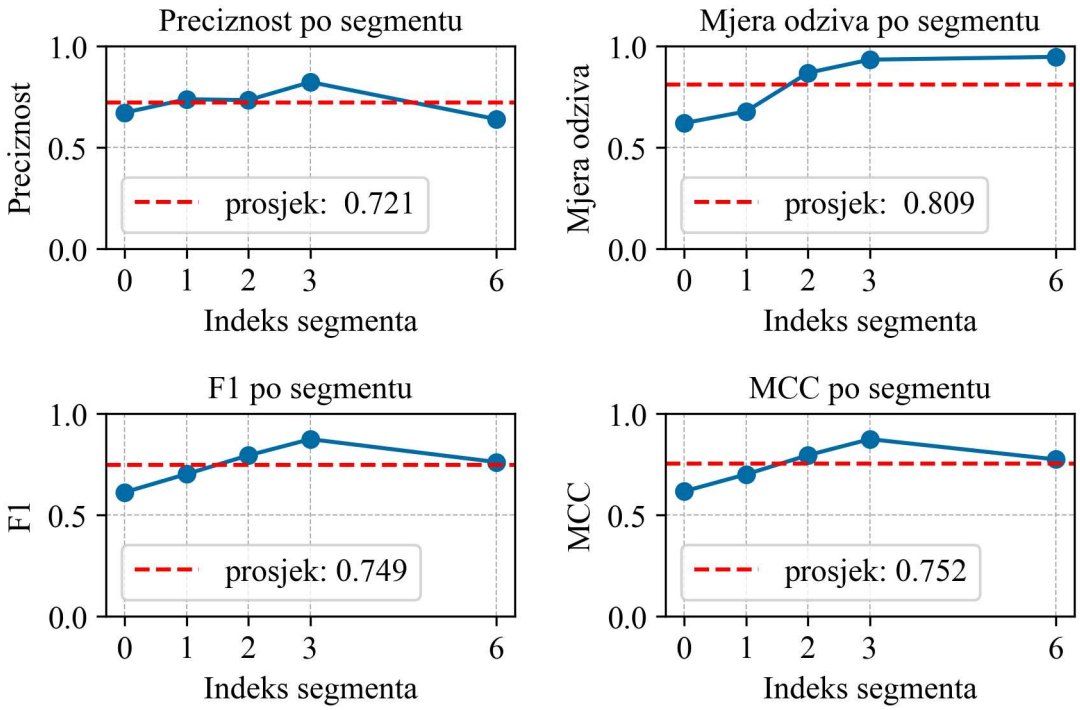
Skup podataka	Preciznost	Odziv	F1	MCC
pedestrians	0.732	0.916	0.810	0.814
421	0.764	0.769	0.764	0.763
111	0.685	0.884	0.752	0.764
112	0.648	0.895	0.736	0.748
highway	0.749	0.706	0.725	0.706
222	0.634	0.847	0.723	0.730
522	0.635	0.846	0.723	0.730
211	0.655	0.852	0.718	0.730
221	0.648	0.810	0.716	0.720
212	0.618	0.903	0.694	0.708

Tablica 5.5: Vrijednosti četiriju metrika za prosjek 5 najuspješnijih segmenata. Prikazano je 10 skupova podataka sortirano po $F1$ metrici.

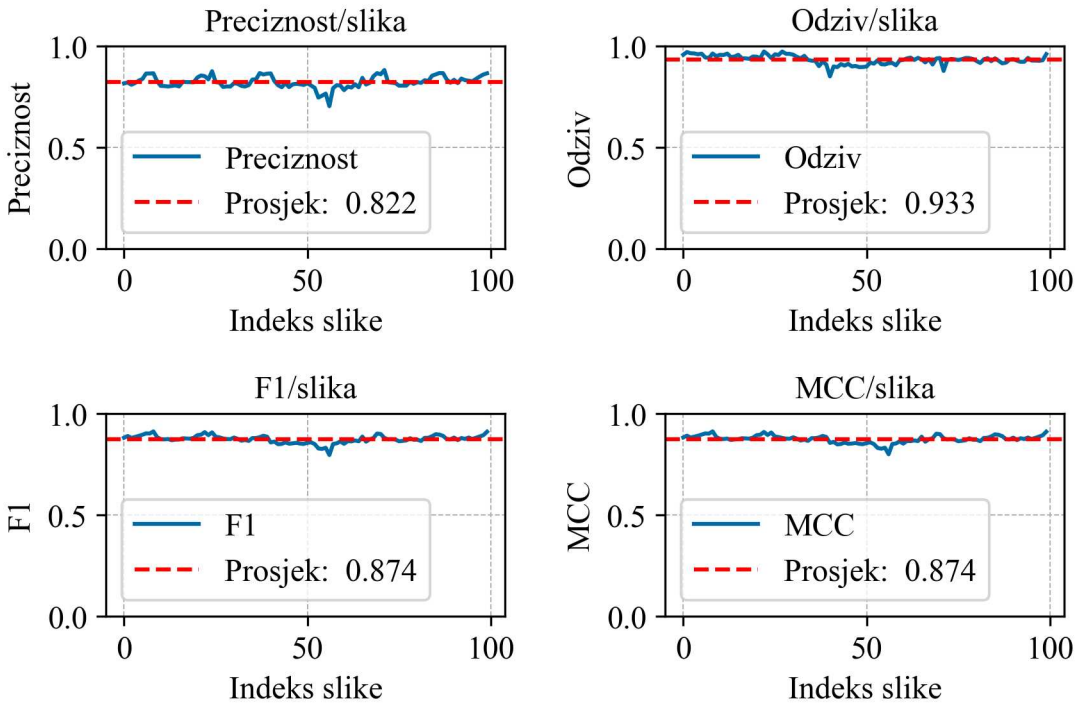
Skup podataka	Preciznost	Odziv	F1	MCC
pedestrians	0.721	0.809	0.749	0.752
111	0.632	0.871	0.717	0.731
222	0.644	0.818	0.716	0.722
421	0.720	0.760	0.714	0.719
highway	0.722	0.706	0.710	0.696
112	0.619	0.871	0.710	0.723
221	0.631	0.805	0.704	0.709
522	0.619	0.834	0.692	0.700
211	0.584	0.874	0.652	0.669
212	0.587	0.846	0.647	0.662



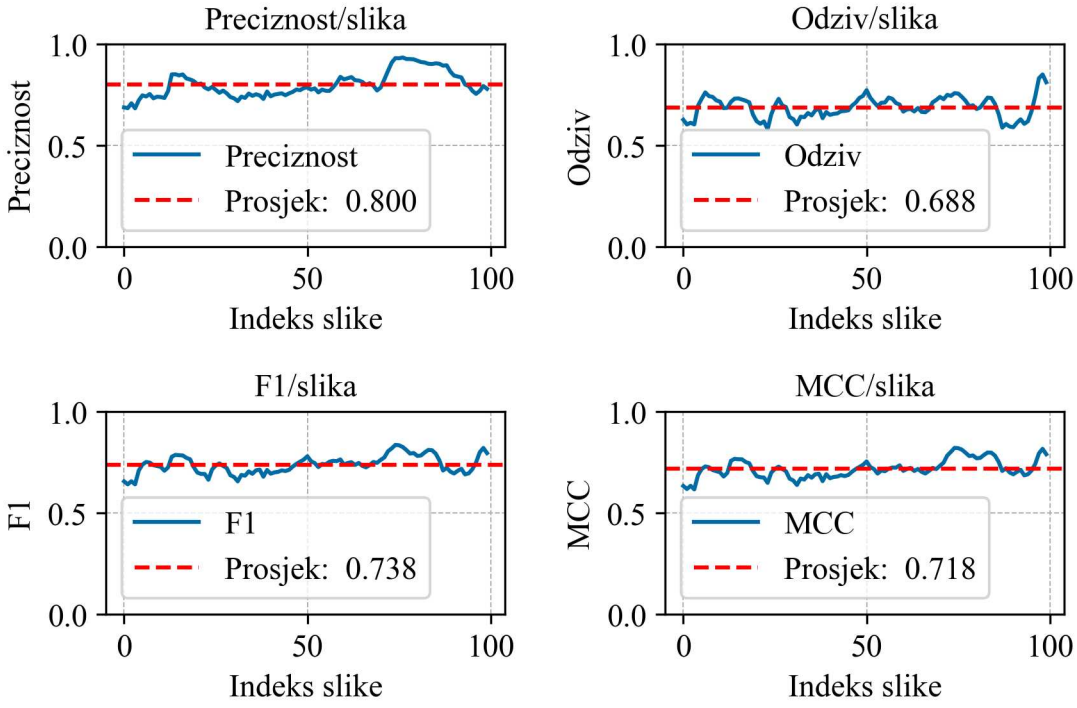
Slika 5.10: Grafički prikaz metrika za 5 najboljih segmenata *pedestrians* skupa podataka. Kriterij odabira jest $F1$ metrika.



Slika 5.11: Grafički prikaz metrika za 5 najboljih segmenata *111* skupa podataka. Kriterij odabira jest $F1$ metrika.



Slika 5.12: Grafički prikaz metrika sliku po sliku za jedan segment iz *pedestrians* skupa podataka.



Slika 5.13: Grafički prikaz metrika sliku po sliku za jedan segment iz *highway* skupa podataka.

Bibliografija

[1] Brunton, S. L., J. L. Proctor, J. H. Tu i J. N. Kutz: *Compressed sensing and dynamic mode decomposition*. J. Comput. Dyn., 2(2):165–191, 2015.

[2] Brunton, Steven L. i J. Nathan Kutz: *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*. Cambridge University Press, 2022, ISBN 9781009098489. https://doi.org/10.1017/9781009089517.

[3] Candès, E., X. Li, Y. Ma i J. Wright: *Robust principal component analysis*. Computing Research Repository, 2009.

[4] Demo, Nicola, Marco Tezzele i Gianluigi Rozza: *PyDMD: Python Dynamic Mode Decomposition*. Journal of Open Source Software, 3(22):530, 2018. https://doi.org/10.21105/joss.00530.

[5] Documentation, OpenCV: *Image Thresholding*. https://docs.opencv.org/3.4/d7/d4d/tutorial_py_thresholding.html. Accessed: 2024-05-17.

[6] Documentation, OpenCV: *Morphological Transformations*. https://docs.opencv.org/4.x/d9/d61/tutorial_py_morphological_ops.html. Accessed: 2024-05-17.

[7] Donoho, D.L.: *Compressed sensing*. IEEE Transactions on Information Theory, 52(4):1289–1306, 2006.

[8] Drmač, Zlatko, Igor Mezić i Ryan Mohr: *Data Driven Modal Decompositions: Analysis and Enhancements*. SIAM Journal on Scientific Computing, 40(4):A2253–A2285, 2018. https://doi.org/10.1137/17M1144155.

[9] Drmač, Zlatko, Vjeran Hari, Miljenko Marušić, Mladen Rogina, Sanja Singer i Saša Singer: *Numerička analiza: Predavanja i vježbe*. PMF - Matematički odjel, Sveučilište u Zagrebu, Zagreb, 2003.

[10] Erichson, N.B., S.L. Brunton i J.N. Kutz: *Compressed dynamic mode decomposition for background modeling*. J Real-Time Image Proc, 16:1479–1492, listopad 2019. https://doi.org/10.1007/s11554-016-0655-2.

[11] Gavish, Matan i David L. Donoho: *The Optimal Hard Threshold for Singular Values is* $4/\sqrt{3}$. IEEE Transactions on Information Theory, 60(8):5040–5053, 2014.

[12] Gilbert, A. C., J. Y. Park i M. B. Wakin: *Sketched SVD: Recovering spectral features from compressive measurements*. stranice 1–10, 2012. arXiv preprint arXiv:1211.0361.

[13] Goyette, Nil, Pierre Marc Jodoin, Fatih Porikli, Janusz Konrad i Prakash Ishwar: *Changedetection.net: A new change detection benchmark dataset*. U *2012 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, stranice 1–8, 2012. http://www.changedetection.net.

[14] Grosek, Jacob i J. Nathan Kutz: *Dynamic Mode Decomposition for Real-Time Background/Foreground Separation in Video*. ArXiv, abs/1404.7592, 2014. https://api.semanticscholar.org/CorpusID:18815881.

[15] Ichinaga, Sara M., Francesco Andreuzzi, Nicola Demo, Marco Tezzele, Karl Lapo, Gianluigi Rozza, Steven L. Brunton i J. Nathan Kutz: *PyDMD: A Python package for robust dynamic mode decomposition*, 2024.

[16] Jovanović, Mihailo R., Peter J. Schmid i Joseph W. Nichols: *Sparsity-promoting dynamic mode decomposition*. Physics of Fluids, 26(2), veljača 2014, ISSN 1089-7666. http://dx.doi.org/10.1063/1.4863670.

[17] Karlić, Luka: *Diplomski Rad*, 2024. https://github.com/karlic-luka/diplomski-rad, Accessed: 2024-06-08.

[18] Karlić, Luka: *Fix plot_summary() method which doesn't plot modes correctly when given snapshots_shape for 2D snapshots*. https://github.com/PyDMD/PyDMD/pull/519, 2024. Accessed: 2024-04-05.

[19] Kutz, J., Steven Brunton, Bingni Brunton i Joshua Proctor: *Dynamic Mode Decomposition: Data-Driven Modeling of Complex Systems*. studeni 2016, ISBN 978-1-611974-49-2.

[20] Kutz, Nathan: *Data-Driven Modeling & Scientific Computation: Methods for Complex Systems & Big Data*. 2013. https://api.semanticscholar.org/CorpusID:60351647.

[21] Li, Fuxin, Taeyoung Kim, Ahmad Humayun, David Tsai i James M. Rehg: *Video Segmentation by Tracking Many Figure-Ground Segments*. U *ICCV*, 2013. `https://web.engr.oregonstate.edu/~lif/SegTrack2/dataset.html`.

[22] Lids i: *i-Lids dataset for AVSS 2007*. Available online, 2007. `http://www.eecs.qmul.ac.uk/~andrea/avss2007_d.html`.

[23] Ma, Yi: *Low-Rank Matrix Recovery and Completion via Convex Optimization - Sample Code*, 2024. `https://people.eecs.berkeley.edu/~yima/matrix-rank/sample_code.html`, Accessed: 2024-01-25.

[24] Mezić, I.: *Analysis of fluid flows via spectral properties of the Koopman operator*. Annual Review of Fluid Mechanics, 45:357–378, 2013.

[25] Rowley, C. W., I. Mezić, S. Bagheri, P. Schlatter i D. S. Henningson: *Spectral analysis of nonlinear flows*. Journal of Fluid Mechanics, 645:115–127, 2009.

[26] Schmid, J. i J. Sesterhenn: *Dynamic mode decomposition of numerical and experimental data*. U *61st Annual Meeting of the APS Division of Fluid Dynamics*. American Physical Society, November 2008.

[27] Tu, J. H., C. W. Rowley, D. M. Luchtenburg, S. L. Brunton i J. N. Kutz: *On dynamic mode decomposition: Theory and applications*. Journal of Computational Dynamics, 1(2):391–421, 2014.

[28] Vacavant, Antoine, Thierry Chateau, Alexis Wilhelm i Laurent Lequièvre: *A Benchmark Dataset for Outdoor Foreground/Background Extraction*. U Park, Jong Il i Junmo Kim (urednici): *Computer Vision - ACCV 2012 Workshops*, stranice 291–300, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg, ISBN 978-3-642-37410-4.

Dodatak A

Python programski kod

Napomena A.0.1. *Programski kod priložen kao dodatak nije cjelokupan kod koji je korišten u radu. Ovdje je naveden samo podskup funkcija koje su napisane za potrebe rada te se ostatak može pronaći na Github repozitoriju [17].*

Napomena A.0.2. *Kod je naveden prepisujući određene ćelije iz tzv. Python bilježnica (eng. Python notebook). Uzevši to u obzir redoslijed kojim su funkcije napisane nije nužno pravilan, a za potrebe samostalnog testiranja moguće je pronaći detalje u [17].*

Napomena A.0.3. *Programski kod nije optimiziran. Taj korak je često preskočen radi jednostavnosti korištenja i lakše generalizacije.*

Napomena A.0.4. *S obzirom da je priloženi kod javno objavljen, komentari i dokumentacija napisani su na engleskom jeziku kako bi obuhvatili širu publiku.*

```
DATASETS = {
    "segtrack": {
        "base_path": "../../data/SegTrackv2",
        "image_path": "../../data/SegTrackv2/JPEGImages",
        "valid_datasets": [
            # "bird_of_paradise",
            # "birdfall",
            "frog",
            # "monkey",
            # "parachute",
            # "soldier",
            # "worm",
        ],
    },
    "change_detection_baseline": {
        "base_path": "../../data/baseline",
```

```
17          "valid_datasets": [
18              "highway",
19              "pedestrians",
20              "office",
21              "PETS2006",
22          ],
23      },
24      "change_detection_dynamic": {
25          "base_path": "../../data/dynamicBackground",
26          "valid_datasets": [
27              # "boats",
28              # "canoe",
29              # "fall",
30              # "fountain01",
31              # "fountain02",
32              # "overpass"
33          ],
34      },
35      "change_detection_intermittent_object_motion": {
36          "base_path": "../../data/intermittentObjectMotion",
37          "valid_datasets": [
38              # "parking",
39              # "sofa",
40              # "streetLight",
41              # "tramstop",
42              # "winterDriveway"
43          ],
44      },
45      "bmc_synthetic_1": {
46          "base_path": "../../data/bmc_synthetic1",
47          "valid_datasets": [
48              "111",
49              "112",
50              "121",
51              "122",
52              "211",
53              "212",
54              "221",
55              "222",
56              "311",
57              "312",
58          ],
59      },
60      "bmc_synthetic_2": {
61          "base_path": "../../data/bmc_synthetic2",
62          "valid_datasets": [
63              "321",
```

```
64              "322",
65              "411",
66              "412",
67              "421",
68              "422",
69              "511",
70              "512",
71              "521",
72              "522",
73          ],
74      },
75  }
76
77
78  def get_dataset_family(dataset: str) -> str:
79      """
80      Returns the family of a given dataset.
81
82      Parameters:
83      dataset (str): The name of the dataset.
84
85      Returns:
86      str: The family of the dataset.
87
88      Raises:
89      ValueError: If the dataset is invalid.
90      """
91      for family, data in DATASETS.items():
92          if dataset in data["valid_datasets"]:
93              return family
94      raise ValueError(f"Invalid dataset: {dataset}")
95
96
97  def get_valid_filenames(dataset: str, is_gt: bool = False) -> list:
98      """
99      Retrieves a list of valid filenames for a given dataset.
100
101     Args:
102         dataset (str): The name of the dataset.
103         is_gt (bool, optional): Specifies whether to retrieve ground truth filenames.
104             Defaults to False.
105
106     Returns:
107         list: A list of valid filenames.
108
109     Raises:
110         ValueError: If an error occurs while retrieving the filenames.
```

```
111
112     """
113     try:
114         family = get_dataset_family(dataset)
115         base_path = DATASETS[family]["base_path"]
116         if family == "segtrack":
117             with open(f"{base_path}/ImageSets/{dataset}.txt", "r") as f:
118                 return [line.strip() for line in f][1:]
119         start_frame, end_frame = np.loadtxt(
120             f"{base_path}/{dataset}/temporalROI.txt"
121         ).astype(int)
122         if is_gt:
123             return [f"gt{str(i).zfill(6)}" for i in range(start_frame, end_frame + 1)]
124         return [f"in{str(i).zfill(6)}" for i in range(start_frame, end_frame + 1)]
125     except ValueError as e:
126         print(f"Error: {e}")
127         return None
128
129
130 def get_groundtruth_path(dataset: str) -> str:
131     """
132     Returns the path to the ground truth directory for a given dataset.
133
134     Args:
135         dataset (str): The name of the dataset.
136
137     Returns:
138         str: The path to the ground truth directory.
139
140     Raises:
141         ValueError: If the dataset family is not found in the DATASETS dictionary.
142
143     """
144     try:
145         family = get_dataset_family(dataset)
146         base_path = DATASETS[family]["base_path"]
147         if family == "segtrack":
148             return f"{base_path}/GroundTruth/{dataset}/"
149         else:
150             return f"{base_path}/{dataset}/groundtruth/"
151     except ValueError as e:
152         print(e)
153         return None
154
155
156 def get_images_path(dataset: str) -> str:
157     """
```

```
158     Returns the path to the images directory for a given dataset.
159
160     Args:
161         dataset (str): The name of the dataset.
162
163     Returns:
164         str: The path to the images directory for the given dataset.
165              Returns None if the dataset is not found in the DATASETS dictionary.
166     """
167     try:
168         family = get_dataset_family(dataset)
169         base_path = DATASETS[family]["base_path"]
170         if family == "segtrack":
171             return f"{base_path}/JPEGImages/{dataset}/"
172         else:
173             return f"{base_path}/{dataset}/input/"
174     except ValueError as e:
175         print(e)
176         return None
177
178
179 def get_file_extension(dataset: str, is_gt: bool = False) -> str:
180     """
181     Get the file extension of the first file in the specified dataset.
182
183     Parameters:
184     - dataset (str): The name of the dataset.
185     - is_gt (bool): Flag indicating whether to get the file extension of the
186         ground truth file.
187         If False, it gets the file extension of the images file.
188
189     Returns:
190     - str: The file extension of the first file in the specified dataset.
191
192     """
193     if is_gt:
194         path = get_groundtruth_path(dataset)
195     else:
196         path = get_images_path(dataset)
197     files = os.listdir(path)
198     return files[0].split(".")[-1]
199
200
201 def get_initial_dataset_metadata(dataset: str) -> dict:
202     """
203     Retrieves the initial metadata for a given dataset.
204
```

```
205     Args:
206         dataset (str): The name of the dataset.
207
208     Returns:
209         dict: A dictionary containing the following metadata:
210             - image_filenames: A list of valid image filenames in the dataset.
211             - gt_filenames: A list of valid ground truth filenames in the dataset.
212             - image_extension: The file extension for the image files.
213             - gt_extension: The file extension for the ground truth files.
214             - image_path: The path to the images in the dataset.
215             - gt_path: The path to the ground truth files in the dataset.
216     """
217     metadata = {}
218     metadata["image_filenames"] = get_valid_filenames(dataset, is_gt=False)
219     metadata["gt_filenames"] = get_valid_filenames(dataset, is_gt=True)
220     metadata["image_extension"] = get_file_extension(dataset, is_gt=False)
221     metadata["gt_extension"] = get_file_extension(dataset, is_gt=True)
222     metadata["image_path"] = get_images_path(dataset)
223     metadata["gt_path"] = get_groundtruth_path(dataset)
224     return metadata
225
226
227 def get_dmd_video_and_metadata(
228     dataset: str, noise: bool = False, noise_std: float = 0.01
229 ) -> Tuple[np.ndarray, dict]:
230     """
231     Retrieves the frames and metadata for a given dataset.
232
233     Args:
234         dataset (str): The name of the dataset.
235         noise (bool, optional): Flag indicating whether to add noise to the frames.
236             Defaults to False.
237         noise_std (float, optional): The standard deviation of the noise.
238             Defaults to 0.01.
239
240     Returns:
241         Tuple[np.ndarray, dict]: A tuple containing the frames as a numpy array
242             and the metadata as a dictionary.
243     """
244     metadata = get_initial_dataset_metadata(dataset)
245     dataset_path = metadata["image_path"]
246     first_frame_path = os.path.join(
247         dataset_path, metadata["image_filenames"][0] + "." + metadata["image_extension"]
248     )
249     first_frame = cv2.imread(first_frame_path)
250     height, width = first_frame.shape[:2]
251     metadata["width"] = width
```

```
252     metadata["height"] = height
253
254     frames = []
255     for i in metadata["image_filenames"]:
256         frame_path = os.path.join(dataset_path, i + "." + metadata["image_extension"])
257         frame = cv2.imread(frame_path, cv2.IMREAD_GRAYSCALE)
258         frame = frame.reshape(-1).astype(np.float32) / 255
259         frames.append(frame)
260     frames = np.vstack(frames).transpose()
261     if noise:
262         frames += np.random.normal(0, noise_std, frames.shape)
263         frames = frames.clip(0, 1)
264     return frames, metadata
265
266
267 def get_groundtruth_video(dataset: str) -> np.ndarray:
268     """
269     Retrieves the ground truth video frames for a given dataset.
270
271     Args:
272         dataset (str): The name of the dataset.
273
274     Returns:
275         np.ndarray: An array of ground truth video frames.
276     """
277     metadata = get_initial_dataset_metadata(dataset)
278     gt_path = metadata["gt_path"]
279     frames = []
280     for i in metadata["gt_filenames"]:
281         frame_path = os.path.join(gt_path, i + "." + metadata["gt_extension"])
282         frame = cv2.imread(frame_path, cv2.IMREAD_GRAYSCALE)
283         frame = frame.astype(np.float32) / 255
284         frames.append(frame)
285     return np.asarray(frames)
286
287
288 def split_dmd_video_into_segments(
289     video: np.ndarray, segment_size: int = 100
290 ) -> np.ndarray:
291     """
292     Splits a DMD video into segments of a specified size.
293
294     Args:
295         video (np.ndarray): The input DMD video as a numpy array.
296         segment_size (int, optional): The size of each segment. Defaults to 100.
297
298     Returns:
```

```
299         np.ndarray: The segmented DMD video as a numpy array.
300
301     Raises:
302         None
303
304     """
305     n_frames = video.shape[1]
306     if n_frames <= segment_size:
307         print(f"Video has less than {segment_size} frames. Using all frames")
308         return video.reshape(video.shape[0], video.shape[1], 1), n_frames
309     n_segments = n_frames // segment_size
310     tmp_video = video[:, : n_segments * segment_size].copy()
311     video_segments = np.split(tmp_video, n_segments, axis=1)
312     video_segments = np.array(video_segments).transpose(1, 2, 0)
313     return video_segments, segment_size
314
315
316 def split_gt_video_into_segments(
317     gt_video: np.ndarray, segment_size: int = 100
318 ) -> np.ndarray:
319     """
320     Splits a ground truth video into segments of a specified size.
321
322     Args:
323         gt_video (np.ndarray): The ground truth video to be split.
324         segment_size (int, optional): The size of each segment. Defaults to 100.
325
326     Returns:
327         np.ndarray: An array of video segments, each of size segment_size.
328
329     Raises:
330         None
331
332     """
333     n_frames = gt_video.shape[0]
334     if n_frames <= segment_size:
335         print(
336             f"Ground truth video has less than {segment_size} frames. Using all frames"
337         )
338         return gt_video.reshape(1, *gt_video.shape), n_frames
339     n_segments = n_frames // segment_size
340     tmp_gt_video = gt_video[: n_segments * segment_size, :, :].copy()
341     gt_segments = np.split(tmp_gt_video, n_segments, axis=0)
342     gt_segments = np.asarray(gt_segments)
343     return gt_segments, segment_size
344
345
```

```python
def get_video_removed(video: np.ndarray, bg: np.ndarray) -> np.ndarray:
    """
    Subtracts the background from each frame in the video.

    Args:
        video (np.ndarray): The input video as a 3D numpy array.
        bg (np.ndarray): The background image as a 2D numpy array.

    Returns:
        np.ndarray: The video with the background subtracted from each frame.
    """
    for i in range(video.shape[0]):
        video[i, :, :] -= bg
    return video


def calc_iou(pred: np.ndarray, truth: np.ndarray) -> float:
    """
    Calculates the Intersection over Union (IoU) between two binary arrays.

    Parameters:
    pred (np.ndarray): The predicted binary array.
    truth (np.ndarray): The ground truth binary array.

    Returns:
    float: The IoU score between the predicted and ground truth arrays.
    """
    intersection = np.logical_and(pred, truth).sum()
    union = np.logical_or(pred, truth).sum()
    return intersection / union if union > 0 else 0


def calculate_classification_scores(
    y_true: np.ndarray, y_pred: np.ndarray, beta: int = 1
) -> Tuple[float, float, float, float]:
    """
    Calculate precision, recall, and F1 score for a binary classification problem.

    Parameters:
    - y_true (np.ndarray): Array of true labels (0 or 1).
    - y_pred (np.ndarray): Array of predicted labels (0 or 1).
    - beta (int): Beta value for F1 score calculation (default: 1).

    Returns:
    - Tuple[float, float, float, float]: Tuple containing precision, recall,
        F1 score, and beta value.
    """
```

```
393     tp = (y_true * y_pred).sum()
394     # tn = ((1 - y_true) * (1 - y_pred)).sum()
395     fp = ((1 - y_true) * y_pred).sum()
396     fn = (y_true * (1 - y_pred)).sum()
397 
398     EPSILON = 1e-7
399     precision = tp / (tp + fp + EPSILON)
400     recall = tp / (tp + fn + EPSILON)
401     f1 = (1 + beta**2) * (precision * recall)
402     f1 /= beta**2 * precision + recall + EPSILON
403     return precision, recall, f1
404 
405 
406 def calculate_classification_scores_for_video(
407     truth: np.ndarray, pred: np.ndarray, beta: int = 1
408 ) -> dict:
409     """
410     Calculates classification scores for each frame in a video.
411 
412     Args:
413         truth (np.ndarray): Ground truth labels for each frame in the video.
414         pred (np.ndarray): Predicted labels for each frame in the video.
415         beta (int, optional): Beta value for F-beta score calculation. Defaults to 1.
416 
417     Returns:
418         dict: A dictionary containing the following classification scores:
419             - precision_array: Array of precision scores for each frame.
420             - recall_array: Array of recall scores for each frame.
421             - f1_array: Array of F1 scores for each frame.
422             - mcc_array: Array of Matthews correlation coefficient scores for each frame.
423             - precision_mean: Mean precision score across all frames.
424             - recall_mean: Mean recall score across all frames.
425             - f1_mean: Mean F1 score across all frames.
426             - mcc_mean: Mean Matthews correlation coefficient score across all frames.
427     """
428     precisions = []
429     recalls = []
430     f1s = []
431     mccs = []
432     for i in range(truth.shape[0]):
433         precision, recall, f1 = calculate_classification_scores(
434             truth[i, :, :].flatten(), pred[i, :, :].flatten()
435         )
436         precisions.append(precision)
437         recalls.append(recall)
438         f1s.append(f1)
439         mcc = matthews_corrcoef(truth[i, :, :].flatten(), pred[i, :, :].flatten())
```

```
440             mccs.append(mcc)
441
442         stats_dict = {}
443         stats_dict["precision_array"] = precisions
444         stats_dict["recall_array"] = recalls
445         stats_dict["f1_array"] = f1s
446         stats_dict["mcc_array"] = mccs
447         stats_dict["precision_mean"] = np.mean(precisions)
448         stats_dict["recall_mean"] = np.mean(recalls)
449         stats_dict["f1_mean"] = np.mean(f1s)
450         stats_dict["mcc_mean"] = np.mean(mccs)
451         return stats_dict
452
453
454     def convert_dmd_shape_to_original(dmd_video, metadata):
455         """
456         Converts the shape of a DMD video to its original shape based on the provided metadata.
457
458         Args:
459             dmd_video (numpy.ndarray): The DMD video with the modified shape.
460             metadata (dict): A dictionary containing the metadata of the original video,
461                 including the height and width.
462
463         Returns:
464             numpy.ndarray: The DMD video with the original shape.
465
466         """
467         return dmd_video.transpose().reshape(-1, metadata["height"], metadata["width"])
468
469
470     def preprocess_frame(frame: np.ndarray, kernel_size: int = 3) -> np.ndarray:
471         """
472         Preprocesses a frame by applying thresholding, median blur, and morphological closing.
473
474         Args:
475             frame (np.ndarray): The input frame to be preprocessed.
476                 Should be in shape (height, width) and in the range [0, 1].
477             kernel_size (int, optional): The size of the kernel used for median blur
478                 and morphological closing. Defaults to 3.
479
480         Returns:
481             np.ndarray: The preprocessed frame, with the same shape as the input frame
482                 and values in the range [0, 255].
483
484         """
485         _, frame = cv2.threshold(
486             (frame * 255).astype(np.uint8), 0, 1, cv2.THRESH_BINARY + cv2.THRESH_OTSU
```

```
487         )
488         kernel = np.ones((kernel_size, kernel_size), np.uint8)
489         frame = cv2.medianBlur(frame, kernel_size)
490         frame = cv2.morphologyEx(
491             frame, cv2.MORPH_CLOSE, kernel, iterations=1
492         )  # NOTE: closing - PART 2
493         return frame.astype(np.uint8)
494
495
496     def preprocess_video(video: np.ndarray, kernel_size: int = 3) -> np.ndarray:
497         """
498         Preprocesses a video by applying a preprocessing function to each frame.
499
500         Args:
501             video (np.ndarray): The input video in shape (n_frames, height, width).
502             kernel_size (int, optional): The size of the kernel for preprocessing.
503                 Defaults to 3.
504
505         Returns:
506             np.ndarray: The preprocessed video in the same shape as the input video.
507         """
508         return np.array([preprocess_frame(frame, kernel_size) for frame in video])
509
510
511     def create_logger():
512         """
513         Create a logger object and configure it with the necessary settings.
514         The logger will save log messages to a file in the 'logs' directory,
515         with the name based on the current date and time.
516         """
517         logger = logging.getLogger(__name__)
518         logger.setLevel(logging.INFO)
519
520         # Create the 'logs' directory if it doesn't exist
521         logs_path = "E:/PMF/diplomski-rad/src/python-src/logs/"
522         if not os.path.exists(logs_path):
523             os.makedirs(logs_path)
524
525         # Create a file handler for logging
526         log_file = os.path.join(logs_path, f'{time.strftime("%Y%m%d-%H%M%S")}_cDMD.log')
527         handler = logging.FileHandler(log_file)
528         handler.setFormatter(logging.Formatter("%(asctime)s:%(levelname)s:%(message)s"))
529
530         # Add the file handler to the logger
531         logger.addHandler(handler)
532
533         return logger
```

```

logger = create_logger()
logger.info("Log file created successfully.")
```

```
def process_datasets(DATASETS):
    """
    Process datasets by performing various operations on each dataset.

    Args:
        DATASETS (dict): A dictionary containing information about the datasets.

    Returns:
        None
    """
    for data in DATASETS:
        for dataset_name in DATASETS[data]["valid_datasets"]:
            try:
                # Initialize tqdm outside the loop to avoid conflicts with logging
                progress_bar = tqdm(
                    desc=f"Processing {data} datasets",
                    unit="dataset",
                    total=len(DATASETS[data]["valid_datasets"]),
                )

                # Perform dataset processing operations
                video, metadata = get_dmd_video_and_metadata(dataset_name)
                height, width = metadata["height"], metadata["width"]
                logger.info(
                    f'Using dataset {dataset_name} from {metadata["image_path"]}
                    and ground truth from {metadata["gt_path"]}'
                )
                logger.info(f"Video shape: {video.shape}")

                segment_size = 100
                video_segments, segment_size = split_dmd_video_into_segments(
                    video, segment_size
                )
                n_segments = video_segments.shape[2]
                gt_video = get_groundtruth_video(dataset_name)
                gt_video_segments, segment_size = split_gt_video_into_segments(
                    gt_video, segment_size
                )

                current_dataset_stats = {}
                for segment_idx in range(n_segments):
```

```
                    logger.info(
                        f"Processing segment {segment_idx+1} out of {n_segments}"
                    )
                    segment = video_segments[:, :, segment_idx].copy()
                    gt_segment = gt_video_segments[segment_idx].copy()
                    np.random.seed(42)
                    dmd = CDMD(
                        svd_rank=svd_rank,
                        compression_matrix=cmat,
                        opt=optimized,
                        sorted_eigs=sorted_eigs,
                    ).fit(segment)
                    modes = dmd.reconstructed_data.transpose().reshape(
                        segment.shape[1], height, width
                    )
                    modes = np.abs(modes).astype(np.float32)
                    K = min(100, modes.shape[0])
                    bg = np.zeros_like(modes[0, :, :])
                    for i in range(K):
                        bg += modes[i, :, :]
                    bg /= K

                    reshaped_segment = (
                        segment.copy().transpose().reshape(-1, height, width)
                    )
                    video_removed = np.abs(
                        get_video_removed(reshaped_segment, bg)
                    ).clip(0, 1)
                    video_removed = preprocess_video(video_removed)
                    gt_segment = gt_segment.astype(np.uint8)

                    current_segment_dict = calculate_classification_scores_for_video(
                        gt_segment, video_removed
                    )
                    current_dataset_stats[segment_idx] = current_segment_dict
                    logger.info(
                        f"Segment {segment_idx+1} out of {n_segments} processed."
                    )

                    # Update tqdm
                    progress_bar.update(1)

                # Save dictionary for current dataset on the disk
                with open(f"{dataset_name}_stats.json", "w") as f:
                    json.dump(current_dataset_stats, f)
                logger.info(f"Dataset {dataset_name} processed.")

```

```
            except Exception as e:
                logger.error(f"Error processing dataset {dataset_name}: {str(e)}")
                # Update tqdm even if there's an error
                progress_bar.update(1)

            finally:
                # Close tqdm
                progress_bar.close()


process_datasets(DATASETS)
```

```
import ipywidgets as widgets
from IPython.display import display

original = convert_dmd_shape_to_original(segment, metadata)
fg = get_video_removed(original.copy(), bg)
fg = np.abs(fg).clip(0, 1)
gt = gt_segment.copy()


def plot_frames(frame_index):
    fig, axs = plt.subplots(1, 5, figsize=(15, 5))

    axs[0].imshow(original[frame_index], cmap="gray")
    axs[0].set_title(f"Originalna slicica: {frame_index}/{segment_size}")
    axs[0].axis("off")

    axs[1].imshow(bg, cmap="gray")  # NOTE: background is the same for all frames
    axs[1].set_title("Pozadina")
    axs[1].axis("off")

    axs[2].imshow(fg[frame_index], cmap="gray")
    axs[2].set_title("Prvi plan")
    axs[2].axis("off")

    _, otsu_img = cv2.threshold(
        (fg[frame_index] * 255).astype(np.uint8),
        0,
        1,
        cv2.THRESH_BINARY + cv2.THRESH_OTSU,
    )
    kernel = np.ones((3, 3), np.uint8)
    otsu_img = cv2.medianBlur(otsu_img, 3)
    otsu_img = cv2.dilate(otsu_img, kernel, iterations=2)
    axs[3].imshow(otsu_img, cmap="gray")
```

```
35      axs[3].set_title(f"Otsu + dilatacija")
36      axs[3].axis("off")
37
38      axs[4].imshow(gt[frame_index], cmap="gray")
39      axs[4].set_title("Stvarna maska")
40      axs[4].axis("off")
41
42      # put the colorbar under each image
43      cbar1 = fig.colorbar(
44          axs[0].imshow(original[frame_index], cmap="gray"),
45          ax=axs[0],
46          orientation="horizontal",
47          fraction=0.046,
48          pad=0.04,
49      )
50      cbar1.set_label("Intenzitet sive boje")
51      cbar2 = fig.colorbar(
52          axs[2].imshow(fg[frame_index], cmap="gray"),
53          ax=axs[2],
54          orientation="horizontal",
55          fraction=0.046,
56          pad=0.04,
57      )
58      cbar2.set_label("Intenzitet sive boje")
59      cbar3 = fig.colorbar(
60          axs[1].imshow(bg, cmap="gray"),
61          ax=axs[1],
62          orientation="horizontal",
63          fraction=0.046,
64          pad=0.04,
65      )
66      cbar3.set_label("Intenzitet sive boje")
67      cbar4 = fig.colorbar(
68          axs[3].imshow(otsu_img, cmap="gray"),
69          ax=axs[3],
70          orientation="horizontal",
71          fraction=0.046,
72          pad=0.04,
73      )
74      cbar4.set_label("Intenzitet sive boje")
75      cbar5 = fig.colorbar(
76          axs[4].imshow(gt[frame_index], cmap="gray"),
77          ax=axs[4],
78          orientation="horizontal",
79          fraction=0.046,
80          pad=0.04,
81      )
```

```
    cbar5.set_label("Intenzitet sive boje")
    plt.tight_layout()
    plt.show()


frame_slider = widgets.IntSlider(
    value=0, min=0, max=original.shape[0] - 1, description="Frame"
)
interactive_plot = widgets.interactive(plot_frames, frame_index=frame_slider)

display(interactive_plot)
```

```
def save_plot(fig, filename):
    """
    Save a matplotlib figure to a file.

    Parameters:
    - fig: The matplotlib figure object to save.
    - filename: The name of the file to save the figure to.

    Returns:
    None
    """
    fig.savefig(filename, bbox_inches="tight")


def get_dataset_results_per_segment(results, dataset_name):
    """
    Retrieves the results for a specific dataset and organizes them per segment.

    Args:
        results (dict): A dictionary containing the results for multiple datasets.
        dataset_name (str): The name of the dataset to retrieve results for.

    Returns:
        dict: A dictionary containing the results per segment,
            including precisions, recalls, f1-scores, MCCs,
            and the corresponding indices.
    """
    dataset_results = results[dataset_name]
    precisions = []
    recalls = []
    f1s = []
    mccs = []
    for segment_idx in dataset_results:
        precisions.append(dataset_results[segment_idx]["precision_mean"])
```

```
35          recalls.append(dataset_results[segment_idx]["recall_mean"])
36          f1s.append(dataset_results[segment_idx]["f1_mean"])
37          mccs.append(dataset_results[segment_idx]["mcc_mean"])
38      all_results = {
39          "precisions": precisions,
40          "recalls": recalls,
41          "f1s": f1s,
42          "mccs": mccs,
43          "indices": list(dataset_results.keys()),
44      }
45      return all_results
46
47
48  def visualize_dataset_results(df, name, save_path=None, title=None):
49      """
50      Visualizes different metrics per segment for a specific dataset, grouped by segment index.
51
52      Args:
53          df (DataFrame): The dataset results DataFrame grouped by dataset name
54              and by segment index.
55          name (str): The name of the dataset.
56          save_path (str, optional): The path to save the plot. Defaults to None.
57          title (str, optional): The title of the plot. Defaults to None.
58      """
59      indices = df["f1"][name].index.values.astype(int)
60      indices_idx = indices.argsort()
61      precisions = df["precision"][name].values
62
63      fig, ax = plt.subplots(2, 2, figsize=(10, 8))
64      ax = ax.flatten()
65      ax[0].plot(
66          indices[indices_idx],
67          precisions[indices_idx],
68          marker="o",
69      )
70      ax[0].axhline(
71          np.mean(precisions),
72          color="r",
73          linestyle="--",
74          label=f"prosjek: {np.mean(precisions) : .3f}",
75      )
76      ax[0].set_title("Preciznost po segmentu")
77      ax[0].set_xlabel("Indeks segmenta")
78      ax[0].set_xticks(indices)
79      ax[0].set_ylabel("Preciznost")
80      ax[0].set_ylim(0, 1)  # Set the y-axis limits
81      ax[0].legend()
```

```
82
83      recalls = df["recall"][name].values
84      ax[1].plot(
85          indices[indices_idx],
86          recalls[indices_idx],
87          marker="o",
88      )
89      ax[1].axhline(
90          np.mean(recalls),
91          color="r",
92          linestyle="--",
93          label=f"prosjek: {np.mean(recalls) : .3f}",
94      )
95      ax[1].set_title("Mjera odziva po segmentu")
96      ax[1].set_xlabel("Indeks segmenta")
97      ax[1].set_xticks(indices)
98      ax[1].set_ylabel("Mjera odziva")
99      ax[1].set_ylim(0, 1)  # Set the y-axis limits
100     ax[1].legend()
101
102     f1s = df["f1"][name].values
103     ax[2].plot(
104         indices[indices_idx],
105         f1s[indices_idx],
106         marker="o",
107     )
108     ax[2].axhline(
109         np.mean(f1s), color="r", linestyle="--", label=f"prosjek: {np.mean(f1s):.3f}"
110     )
111     ax[2].set_title("F1 po segmentu")
112     ax[2].set_xlabel("Indeks segmenta")
113     ax[2].set_xticks(indices)
114     ax[2].set_ylabel("F1")
115     ax[2].set_ylim(0, 1)  # Set the y-axis limits
116     ax[2].legend()
117
118     mccs = df["mcc"][name].values
119     ax[3].plot(
120         indices[indices_idx],
121         mccs[indices_idx],
122         marker="o",
123     )
124     ax[3].axhline(
125         np.mean(mccs), color="r", linestyle="--", label=f"prosjek: {np.mean(mccs):.3f}"
126     )
127     ax[3].set_title("MCC po segmentu")
128     ax[3].set_xlabel("Indeks segmenta")
```

```
129     ax[3].set_xticks(indices)
130     ax[3].set_ylabel("MCC")
131     ax[3].set_ylim(0, 1)  # Set the y-axis limits
132     ax[3].legend()
133 
134     fig.tight_layout()
135     fig.subplots_adjust(top=0.88)
136     if title is not None:
137         fig.suptitle(title, fontsize=16)
138 
139     if save_path is not None:
140         save_plot(fig, save_path)
141     else:
142         plt.show()
143 
144 
145 def get_k_best_segment_indices(results, datasets, metric="f1"):
146     """Returns indices of k best segments based on metric for each dataset.
147     Args:
148         results: dict of dicts, where keys are dataset names and values are dicts
149             with keys being segment indices and values being dicts with metrics.
150         datasets: list of dataset names.
151         metric: str, metric to be used for selecting best segments.
152     Returns:
153         top_k_indices: dict of lists, where keys are dataset names
154             and values are lists of indices of k best segments
155     """
156 
157     metric_column_name = {
158         "precision": "precision_array",
159         "recall": "recall_array",
160         "f1": "f1_array",
161         "mcc": "mcc_array",
162     }
163     # best segments based on metric
164     top_k_indices = {}
165     stats_column_name = metric_column_name[metric]
166     for dataset in datasets:
167         temp_results = results[dataset]
168         scores = [np.mean(stats[stats_column_name]) for stats in temp_results.values()]
169         top_k_indices[dataset] = sorted(np.argsort(scores)[-k:])
170     return top_k_indices
171 
172 
173 def get_pandas_df_from_results(results, top_k_indices, datasets):
174     columns = ["dataset", "segment", "precision", "recall", "f1", "mcc"]
175     df = pd.DataFrame(columns=columns)
```

```
176
177     for dataset in datasets:
178         for segment_idx in top_k_indices[dataset]:
179             segment = str(segment_idx)
180             df = df._append(
181                 {
182                     "dataset": dataset,
183                     "segment": segment,
184                     "precision": results[dataset][segment]["precision_mean"],
185                     "recall": results[dataset][segment]["recall_mean"],
186                     "f1": results[dataset][segment]["f1_mean"],
187                     "mcc": results[dataset][segment]["mcc_mean"],
188                 },
189                 ignore_index=True,
190             )
191     return df.copy()
192
193
194 def get_best_segments_in_df(df, metric="f1"):
195     """
196     Returns the best segments in the given DataFrame based on the specified metric.
197
198     Parameters:
199     - df (pandas.DataFrame): The DataFrame containing the segments.
200     - metric (str): The metric used to determine the best segments.
201         Default is 'f1'.
202
203     Returns:
204     - list: A list of the best segments.
205     """
206     # Function implementation goes here
207     best_segments = df.groupby("dataset").idxmax()
208     return df.loc[best_segments[metric]].copy()
209
210
211 def get_best_frame_per_dataset(best_segments, results):
212     """
213     Retrieves the best frame per dataset based on the given best_segments.
214
215     Parameters:
216     - best_segments (list): A list of best segments per dataset.
217
218     Returns:
219     - None
220     """
221     SEGMENT_SIZE = 100
222     best_frame_per_dataset = {}
```

```
223     for data_idx in best_segments.index:
224         dataset, seg_idx = data_idx
225         f1_results = results[dataset][str(seg_idx)]["f1_array"]
226         best_frame = np.argmax(f1_results)
227         actual_frame_idx = int(seg_idx) * SEGMENT_SIZE + int(best_frame)
228         best_frame_per_dataset[dataset] = actual_frame_idx
229     return best_frame_per_dataset.copy()
```

```
1  import os
2  import cv2
3  from tqdm import tqdm
4  import argparse
5
6
7  def create_temporal_roi_file(
8      target_dir: str, start_frame: int = 0, end_frame: int = 0
9  ) -> bool:
10     """
11     Create a temporal ROI (Region of Interest) file in the target directory.
12
13     Args:
14         target_dir (str): The directory where the temporal ROI file will be created.
15         start_frame (int, optional): The starting frame of the ROI. Defaults to 0.
16         end_frame (int, optional): The ending frame of the ROI. Defaults to 0.
17
18     Returns:
19         bool: True if the temporal ROI file is successfully created or already exists,
20             False otherwise.
21     """
22     temporal_roi_path = os.path.join(target_dir, "temporalROI.txt")
23     if not os.path.exists(temporal_roi_path):
24         try:
25             with open(temporal_roi_path, "w") as f:
26                 f.write(f"{start_frame} {end_frame}")
27             return True
28         except Exception as e:
29             print(f"Error: {e}")
30             return False
31     return True
32
33
34 def extract_images_from_video(video_path: str, target_dir: str) -> None:
35     """
36     Extracts images from a video and saves them to the target directory.
37
38     Args:
```

```
39          video_path (str): The path to the video file.
40          target_dir (str): The directory where the extracted images will be saved.
41
42      Returns:
43          None
44      """
45      cap = cv2.VideoCapture(video_path)
46      is_gt = "groundtruth" in target_dir
47      if is_gt:
48          n_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
49          temporal_roi_dir = os.path.dirname(target_dir)
50          success = create_temporal_roi_file(
51              temporal_roi_dir, start_frame=0, end_frame=n_frames - 1
52          )
53          if not success:
54              print(f"Temporal ROI file could not be created for {temporal_roi_dir}")
55              return
56
57      i = 0
58      while cap.isOpened():
59          ret, frame = cap.read()
60          if not ret:
61              break
62          image_name = f"gt{str(i).zfill(6)}.png" if is_gt else f"in{str(i).zfill(6)}.png"
63          cv2.imwrite(os.path.join(target_dir, image_name), frame)
64          i += 1
65      cap.release()
66      return
67
68
69  def main():
70      """
71      Extract images from videos.
72
73      This function takes two command-line arguments:
74      -i/--source_dir: Source directory containing the videos.
75      -o/--base_target_dir: Base target directory to store the extracted images.
76
77      It iterates over the videos in the source directory, creates target directories
78      for each video, and extracts images from the videos to the corresponding target
79      directories.
80
81      Example usage:
82      python extract_images_from_videos.py -i /path/to/source_dir -o /path/to/base_target_dir
83      """
84      parser = argparse.ArgumentParser(description="Extract images from videos.")
85      parser.add_argument(
```

```
86          "-i",
87          "--source_dir",
88          type=str,
89          required=True,
90          help="Source directory containing the videos.",
91      )
92      parser.add_argument(
93          "-o",
94          "--base_target_dir",
95          type=str,
96          required=True,
97          help="Base target directory to store the extracted images.",
98      )
99
100     args = parser.parse_args()
101
102     source_dir = args.source_dir
103     base_target_dir = args.base_target_dir
104     video_names = os.listdir(source_dir)
105     original_video_names = [
106         v for v in video_names if v.endswith(".mp4") and "gt" not in v
107     ]
108
109     target_dirs = [
110         os.path.join(base_target_dir, v.split(".")[0].split("_")[0])
111         for v in original_video_names
112     ]
113     for target_dir in target_dirs:
114         if not os.path.exists(target_dir):
115             print(f"Creating directory {target_dir}")
116             os.makedirs(target_dir)
117         if not os.path.exists(os.path.join(target_dir, "input")):
118             print(f'Creating input directory {os.path.join(target_dir, "input")}')
119             os.makedirs(os.path.join(target_dir, "input"))
120         if not os.path.exists(os.path.join(target_dir, "groundtruth")):
121             print(
122                 f'Creating groundtruth directory {os.path.join(target_dir, "groundtruth")}'
123             )
124             os.makedirs(os.path.join(target_dir, "groundtruth"))
125
126     mp4_files = [f for f in os.listdir(source_dir) if f.endswith(".mp4")]
127
128     for file in tqdm(mp4_files, desc="Extracting images", unit="file"):
129         video_name = file.split(".")[0]
130         target_dir = os.path.join(base_target_dir, video_name.split("_")[0])
131         target_dir = (
132             os.path.join(target_dir, "input")
```

```
            if "gt" not in video_name
            else os.path.join(target_dir, "groundtruth")
        )
        video_path = os.path.join(source_dir, file)
        print(f"Extracting images from {file} to {target_dir}")
        extract_images_from_video(video_path, target_dir)


if __name__ == "__main__":
    main()
```

Sažetak

U ovom radu koristimo dinamičku modalnu dekompoziciju (DMD) kako bismo na efikasan način u stvarnom vremenu odvojili prvi plan od pozadine u videozapisu. Svaku sličicu u videozapisu odvajamo na komponentu niskog ranga (eng. *low-rank component*) te rijetki dio (eng. *sparse*). Ova metoda inicijalno je razvijena za potrebe analiziranja nelinearnih dinamičkih sustava, no s obzirom da je pogonjena isključivo podacima (eng. *data driven*), pronašla je svoje primjene u raznim područjima, u ovom slučaju u računalnom vidu. DMD raščlanjuje originalni dinamički sustav na modove niskog ranga s poznatim Fourierovim komponentama u vremenu. Time dobivamo dekompoziciju koja uzima u obzir i vremensku i prostornu ovisnost. Strogo gledano, pozadinu čine DMD modovi s Fourierovim koeficijentima blizu ishodištu, dok je prvi plan sastavljen od nešto udaljenijih modova. Za razliku od RPCA, koji je vodeća metoda za separaciju, DMD-ova najzahtjevnija operacija jest samo jedna SVD dekompozicija, čime omogućuje procesuiranje u stvarnom vremenu.

Nadalje, opisujemo i poboljšanje standardne DMD metode. Sažeti DMD iskorištava prednosti teorije sažetog uzorkovanja (eng. *compressed sensing*) i skiciranja matrice (eng. *matrix sketching*) da bi znatno efikasnije aproksimirao DMD modove iz sažete reprezentacije početnog dinamičkog sustava, dok je preciznost skoro nepromijenjena. Videozapisi prirodno uvode problem velike dimenzionalnosti zbog jako velikog broja piksela, no sažetom reprezentacijom izbjegavamo računati SVD na vrlo velikoj matrici, a k tomu smo i u mogućnosti koristiti videozapise znatno veće rezolucije, čime očuvamo detalje.

Metodu u konačnici evaluiramo na mnoštvu javno dostupnih skupova podataka te navodimo brojčane rezultate. Kako bismo mjerili kakvoću, koristimo matricu zabune i četiri različite metrike - preciznost, odziv, F1 mjeru i Matthewseov koeficijent korelacije. Pokazali smo da vrlo uspješno može odvojiti pozadinu i prvi plan, čime otvara mnoge mogućnosti u računalnom vidu s primjenama na videonadzor.

Summary

In this thesis, we use dynamic mode decomposition (DMD) to efficiently separate the foreground from the background in a video in real time. Each frame in the video is separated into a low-rank component and a sparse component. This method was initially developed for the analysis of nonlinear dynamic systems, but since it is purely data-driven, it has found applications in various fields, in this case, in computer vision. DMD decomposes the original dynamic system into low-rank modes with known Fourier components over time. This provides a decomposition that considers both temporal and spatial dependencies. Strictly speaking, the background consists of DMD modes with Fourier coefficients close to the origin, while the foreground is composed of slightly more distant modes. Unlike RPCA, which is the leading method for separation, DMD's most demanding operation is just a single SVD decomposition, enabling real-time processing.

Furthermore, we describe an improvement of the standard DMD method. Compressed DMD leverages the advantages of compressed sensing theory and matrix sketching to significantly more efficiently approximate DMD modes from a compressed representation of the initial dynamic system, while maintaining nearly unchanged accuracy. Videos naturally introduce the problem of high dimensionality due to the very large number of pixels, but with compressed representation, we avoid computing SVD on a very large matrix, and we can also use videos of significantly higher resolution, thereby preserving details.

Ultimately, we evaluate the method on a multitude of publicly available datasets and provide numerical results. To measure quality, we use the confusion matrix and four different metrics - precision, recall, F1 score, and Matthews correlation coefficient. We have shown that it can successfully separate the background and foreground, opening many possibilities in computer vision applications for video surveillance.

Životopis

Rođen sam 3.lipnja 1998. godine u Zagrebu. Nakon završene osnovne škole Augusta Šenoe, upisujem prirodoslovno-matematičku V. gimnaziju u Zagrebu, gdje sam maturirao 2017. godine. Ljubav prema matematici i programiranju nastavlja se na Matematičkom odsjeku Prirodoslovno-matematičkog fakulteta u Zagrebu gdje upisujem preddiplomski studij Matematika. 2021. godine kao prvostupnik matematike upisujem diplomski studij Matematička statistika. Na apsolventskoj godini sudjelovao sam u Erasmus+ studentskoj razmjeni gdje me ugostilo Sveučilište *Université Paris-Est Créteil*, odsjek *Sciences et technologie*. Tijekom studiranja dodijeljene su mi tri nagrade za izvanredan uspjeh na izvannastavnim aktivnostima, ponajviše radi uspjeha u raznim natjecanjima u području podatkovne znanosti i programiranja.