به نام خدا

دانشگاه تهران

پردیس دانشکده‌های فنی

دانشکده مهندسی برق و کامپیوتر

جبر خطی

پروژه پایانی

نورا زارعی

۸۱۰۱۹۹۴۳۳

بهمن ۱۴۰۲

فهرست گزارش

جداسازی پس زمینه با Robust PCA:

سوالات مفهومی:

۱. تنسور یک مفهوم ریاضی است که مفهوم اسکالرها، بردارها و ماتریس ها را تعمیم می‌دهد. به زبان ساده، تنسور یک آرایه چند بعدی از اعداد است. تنسورها را می توان برای نمایش و کار کردن با ساختارهای داده پیچیده استفاده کرد و به طور گسترده در زمینه‌های مختلف مانند فیزیک، مهندسی و علوم کامپیوتر استفاده می‌شود.

در پردازش تصویر ، از تنسور برای نشان دادن داده‌های تصویر به گونه‌ای استفاده می‌شود که روابط مکانی بین پیکسل‌ها را بررسی می‌کند. به عنوان مثال، یک تصویر رنگی را می‌توان به عنوان یک تنسور سه بعدی با ابعاد مربوط به عرض، ارتفاع و رنگ تصویر نشان داد. تنسورها در عملیات‌هایی مانند کانولوشن، ادغام و استخراج ویژگی در الگوریتم‌های پردازش تصویر استفاده می‌شوند که امکان بررسی و تجزیه و تحلیل کارآمد داده‌های تصویر را فراهم می‌کند.

۲. Robust PCA فرم گسترش یافته روش قدیمی PCA است که برای کاهش ابعاد و تجزیه و تحلیل داده‌ها استفاده می‌شود. در حالی که PCA فرض می‌کند که داده‌ها ترکیبی از یک ماتریس با رتبه پایین و نویز کم است، Robust PCA داده‌های بسیار دور افتاده را نیز در نظر می‌گیرد. هدف PCA یافتن مولفه‌های اصلی است که در جهت حداکثر واریانس در داده‌ها هستند.

PCA فرض می‌کند که داده‌ها را می‌توان به خوبی توسط یک ماتریس با رتبه پایین توصیف کرد و هر گونه انحراف از این ساختار رتبه پایین به دلیل نویز است. این روش به دنبال تجزیه داده‌ها به دو ماتریس است: یک ماتریس با رتبه پایین که ساختار اصلی را نشان می‌دهد و یک ماتریس پراکنده که نویز یا نقاط پرت را نشان می‌دهد.

از طرف دیگر، Robust PCA با در نظر گرفتن وجود داده‌های پرت یا انحراف فاحش در داده‌ها، فرضیات PCA را کاهش می‌دهد. این روش فرض می‌کند که داده‌ها را می‌توان به یک ماتریس با رتبه پایین، یک ماتریس پراکنده نشان دهنده نقاط پرت و یک ماتریس نویز تجزیه کرد. هدف این تکنیک بازیابی ماتریس با رتبه پایین و ماتریس پراکنده از داده‌های مشاهده شده، حتی در حضور داده‌های دور افتاده است.

الگوریتم‌های Robust PCA به طور معمول از تکنیک‌های بهینه سازی قوی برای تخمین اجزای ماتریس‌ها استفاده می‌کند. این الگوریتم‌ها به گونه‌ای طراحی شده‌اند که نسبت به موارد پرت مقاوم‌تر باشند و می‌توانند تجزیه دقیق‌تری را در حضور داده‌های خراب و یا نویز ارائه دهند.

Algorithm 2: The bias method for identifying outliers for robust centered PCA.

Input:

X: the data matrix.

r: the desired number of principal components.

k: the desired number of outliers.

ALG: an algorithm for identifying outliers for robust uncentered PCA.

Output: A set O of k outliers.

1. Select a sufficiently large value b. (See Section 5 for the explicit formula: $b = 10\|X\|_F$.)
2. Append b to each column of X, creating X_b of size $(m+1) \times n$.
3. Run ALG on X_b to compute $r+1$ principal components and identifying k outliers for robust uncentered PCA.
4. Return the k outliers computed in Step 3 as the output.

Figure۱ الگوریتم ۱ و شبه کد Robust PCA

۳. Randomized SVD، Truncated SVD و Higher-order SVD روش‌های دیگر و یا تقریبی از روش استاندارد SVD است. در ادامه توضیح مختصری از هر یک آمده:

SVD: یک تکنیک فاکتورسازی ماتریس است که یک ماتریس را به سه ماتریس تجزیه می‌کند، U, Σ, V^T. با توجه به ماتریس ورودی $A_{m\times n}$، با SVD داریم: $A = U.\Sigma.V^T$ که در آن U یک ماتریس متعامد $m \times m$، Σ یک ماتریس قطری $m \times n$ با مقادیر غیر منفی که به آنها مقادیر منفرد گفته می‌شود، و V^T یک ماتریس متعامد $n \times n$ است. SVD بسیار دقیق است و تجزیه بهینه ماتریس را فراهم می‌کند.

Randomized SVD: یک الگوریتم تقریب سریع برای محاسبه تجزیه تقریبی رتبه پایین یک ماتریس است. از تکنیک های تصادفی سازی برای تخمین موثر مقادیر منفرد و بردارهای منفرد متناظر یک ماتریس استفاده می کند. با استفاده از روش های نمونه گیری تصادفی و ترسیم ماتریسی، می تواند یک SVD تقریبی را با هزینه محاسباتی بسیار کمتر در مقایسه با SVD استاندارد محاسبه کند. این الگوریتم در ادامه توضیح داده شده است. ماتریس $A_{m\times n}$ را در نظر بگیرید. میخواهیم دو ماتریس $B_{m\times k}$ و $C_{k\times n}$ را طوری بدست بیاوریم که $k \ll n$ و $A \approx BC$. برای این کار دو مرحله نیاز است.

مرحله ۱: یک پایه تقریبی برای محدوده A محاسبه کنید. یک ماتریس Q میخواهیم به طوری که $A \approx QQ^TA$.

مرحله ۲: از Q استفاده کرده و ماتریس A راتجزیه می کنیم.

الگوریتم:

$$B = Q^T A$$
$$B = \widetilde{U}\Sigma V$$
$$U = Q\widetilde{U}$$
$$A \approx U\Sigma V^T$$

Truncated SVD: روشی است که SVD را تنها با حفظ مقادیر منفرد top-k و بردارهای منفرد مربوط به آنها تقریب می زند. به جای محاسبه SVD کامل، که می تواند از نظر محاسباتی برای ماتریس های بزرگ پیچیده باشد، این روش تنها k بزرگترین مقادیر منفرد و بردارهای منفرد مرتبط با آنها را حفظ می کند. تقریب بدست آمده یک نمایش رتبه پایین تر از ماتریس اصلی است. این روش معمولا برای کاهش ابعاد و کاهش نویز در تجزیه و تحلیل داده ها استفاده می شود.

Higher-order SVD: SVD استاندارد روی ماتریس ها اعمال شده و استفاده می شود، Higher-order SVD می تواند تنسورهای سه بعدی یا بیشتر را کنترل کند. یک تنسور را به یک تنسور هسته و مجموعه ای از ماتریس های عامل برای هر حالت (بعد) تجزیه می کند. تنسور هسته نشان دهنده تعاملات بین حالت ها است در حالی که ماتریس های فاکتور اطلاعات خاص حالت را دریافت می کنند. فرض می کنیم $A \in \mathbb{C}^{I_1 \times I_2 \times \ldots \times I_m \times \ldots \times I_M}$ که در آن M تعداد حالت ها و ترتیب تنسور است.

فرض کنیم $U_m \in \mathbb{C}^{I_m \times I_m}$ یک ماتریس واحد شامل بردارهای منفرد سمت چپ حالت استاندارد $A_{[m]}$ باشد به طوری که ستون jام U_m مربوط به jامین بزرگترین مقدار منفرد $A_{[m]}$ باشد. توجه باید داشته باشیم که ماتریس حالت U_m به تعریف خاص حالت m بستگی ندارد. با خواص ضرب چند خطی داریم:

$$A = A \times (I, I, \ldots, I) = A \times (U_1 U_1^H, U_2 U_2^H, \ldots, U_M U_M^H)$$
$$= (A \times (U_1^H, U_2^H, \ldots, U_M^H)) \times (U_1, U_2, \ldots., U_M)$$

که در آن ^{H}. ترانهاده مزدوج است. تساوی دوم به این دلیل است که U_m ها ماتریس های واحد هستند.

هسته تنسور به صورت زیر تعریف می شود:

$$S \coloneqq A \times (U_1^H, U_2^H, \ldots, U_M^H)$$

HOSVD تنسور A به صورت زیر است:

$$A = S \times (U_1, U_2, \ldots, U_M)$$

که ساختار بالا نشان می دهد هر تنسور یک HOSVD دارد.

تفاوت اصلی بین این روش ها کیفیت محاسبات، دقت تقریبی و توانایی آنها برای حل انواع خاصی از داده ها و یا مسئله ها است. Randomized SVD و Truncated SVD تقریب سریعتری از SVD کامل ارائه می دهند و از آنها برای مجموعه داده هاس مقیاس بزرگ استفاده می شود، درحالی که Higher-order SVD مفهوم SVD را به تنسورها گسترش می دهد. با این حال، این تقریب ها ممکن است برخی از خطاها را در مقایسه با راه حل دقیق ارائه شده توسط SVD استاندارد ایجاد کنند.

۴. Nuclear norm ماتریس A به صورت $\left|\left|A\right|\right|_*$ تعریف شده و به صورت مجموع مقادیر منفرد است:

$$\left|\left|A\right|\right|_* = \sum_i \sigma_i(A)$$

پیاده سازی:

در ابتدا ویدیو را در ساختار تنسور خوانده و یک فریم به صورت رندوم از آن نمایش می دهیم.

Figure ۲ فریم دلخواه از فیلم

همچنین سایر موار خواسته شده به صورت زیر خواهد بود:

$$Resolution: 360 \times 640$$

$$Dimensions: [360, 640, 3, 901]$$

که بعد اول نشان دهنده height ویدیو، بعد دوم نشان دهنده width ویدیو، بعد سوم نشان دهنده اینکه ویدیو rgb است و بعد آخر تعداد فریم های ویدیو را نشان می دهد.

```
indiantraffic = VideoReader("Indiantraffic.avi");
height = indiantraffic.Height;
width = indiantraffic.Width;
duration_in_sec = indiantraffic.Duration;
frame_rate = indiantraffic.FrameRate;
frame_num = round(frame_rate * duration_in_sec);
rgb_film = indiantraffic.read;
rgb_film_size = size(rgb_film);
frame_ind = randi(frame_num);
frame_data = rgb_film(:,:,:,frame_ind);
figure
imshow(frame_data);
title("'Indiantraffic.avi', Frame Number = " + num2str(frame_ind), Interpreter = "latex");
xlabel("Time = " + num2str(frame_ind/rgb_film_size(4)*duration_in_sec) + " sec", Interpreter = "latex");
```

Figure ۳ قطعه کد خواندن ویدیو

فیلم را سیاه و سفید کرده:

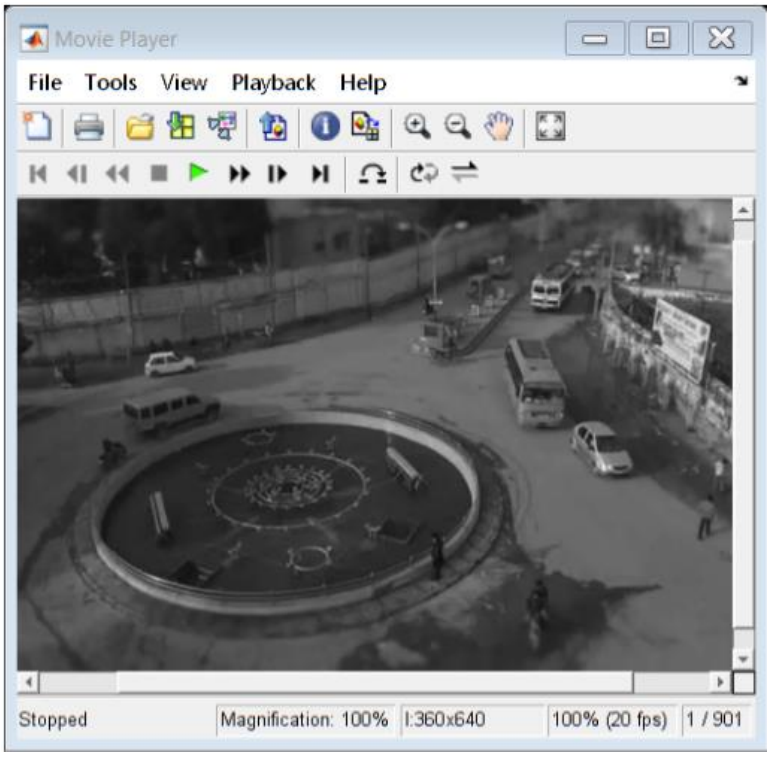

Figure ۴ ویدیو بعد از سیاه و سفید شدن

سپس به آن نویز salt and pepper اضافه می کنیم:

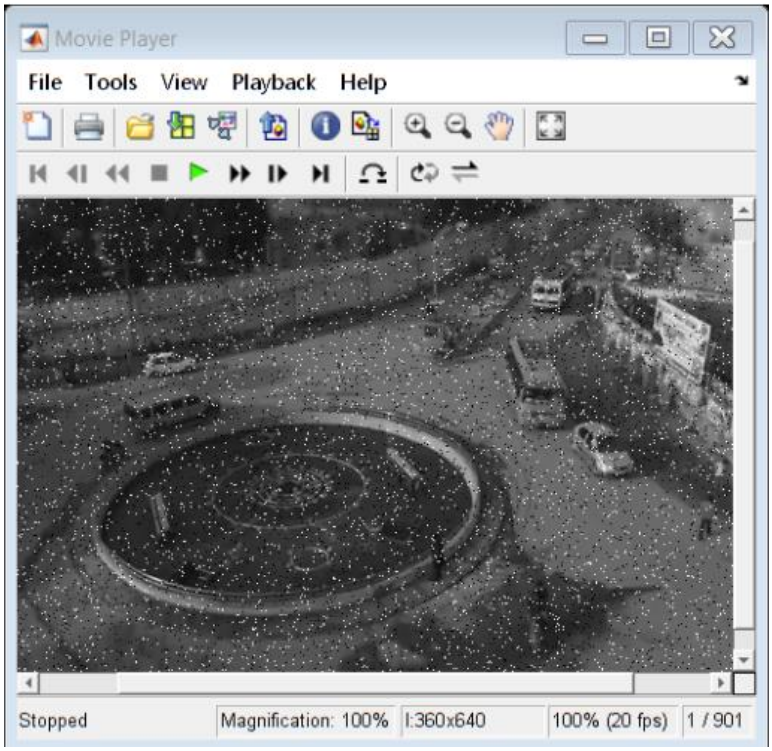

Figure ۵ ویدیو پس از اضافه کردن نویز salt and pepper

```
gray_film = zeros(height, width, frame_num);
noisy_gray_film = zeros(height, width, frame_num);
for i = 1:frame_num
    gray_film(:,:,i) = double(rgb2gray(rgb_film(:,:,:,i))) / 2^8;
    noisy_gray_film(:,:,i) = imnoise(gray_film(:,:,i), 'salt & pepper');
end
implay(gray_film);
```

Figure ۶ قطعه کد سیاه و سفید کردن و اضافه کردن نویز

که کد بالا برای سیاه و سفید کردن مقدار هر پیکسل را از [0,256] به [0, 1] می رساند و سپس نویز گفته شده را به آن اضافه میکند.

خطای Reconstruction: خطای بازسازی در PCA نشان دهنده تفاوت بین نقاط داده اصلی و پیش بینی آنها در فضای کم بعدی تعریف شده توسط مؤلفه های اصلی است. این کمیت نشان می دهد که چگونه نمایش با ابعاد پایین تر اطلاعات را از داده های اصلی حفظ می کند. خطای بازسازی کمتر نشان دهنده حفظ بهتر اطلاعات است.

حال الگوریتم Robust PCA را با روش بهینه سازی PCP که در زیر روش و الگوریتم آن آورده شده است پیاده سازی کردیم.

Algorithm 1 (Principal Component Pursuit by Alternating Directions [32, 51])

initialize: $S_0 = Y_0 = 0, \mu > 0$.
while not converged **do**
  compute $L_{k+1} = \mathcal{D}_\mu(M - S_k - \mu^{-1}Y_k)$;
  compute $S_{k+1} = \mathcal{S}_{\lambda\mu}(M - L_{k+1} + \mu^{-1}Y_k)$;
  compute $Y_{k+1} = Y_k + \mu(M - L_{k+1} - S_{k+1})$;
end while
output: L, S.

Figure ۷ الگوریتم PCP

```
function [L, S] = robust_pca(X, lambda, mu, tol, max_iter)

    [M, N] = size(X);
    unobserved = isnan(X);
    X(unobserved) = 0;
    normX = norm(X, 'fro');

    L = zeros(M, N);
    S = zeros(M, N);
    Y = zeros(M, N);

    for iter = (1:max_iter)
        L = Do(1/mu, X - S + (1/mu)*Y);
        S = So(lambda/mu, X - L + (1/mu)*Y);
        Z = X - L - S;
        Z(unobserved) = 0;
        Y = Y + mu*Z;
        err = norm(Z, 'fro') / normX;
        if (iter == 1) || (mod(iter, 10) == 0) || (err < tol)
            fprintf(1, 'iter: %04d\terr: %d\n',iter, err);
        end
        if (err < tol)
            break;
        end
    end
end

function r = So(tau, X)
    r = sign(X) .* max(abs(X) - tau, 0);
end

function r = Do(tau, X)
    [U, S, V] = svd(X, 'econ');
    r = U*So(tau, S)*V';
end
```

Figure ۸ قطعه کد الگوریتم Robust PCA با روش PCP

**به دلیل اینکه لپتاپم نمی توانست کل ویدیو را پردازش کند فقط موفق به پردازش ۲۰۰ فریم اول شدم.

ویدیو را در ابتدا flat کرده و تنسور موجود را به صورت برداری می کنیم:

```
X = zeros(200, height*width);
for i = (1:200)
    frame = read(indiantraffic, i);
    frame = rgb2gray(frame);
    X(i,:) = reshape(frame,[],1);
end
```

Figure ۹ فلت کردن ویدیو

سپس با انتخاب و پیدا کردن لاندا مناسب، الگوریتم Robust PCA را برای تجزیه به LowRank و Sparse پیاده سازی می کنیم و خطای reconstruction را نیز گزارش می دهیم.

```
lambda = 0.00005;
tol = 1e-5;
maxIter = 5;
[lowRankTensor, sparseTensor] = robust_pca(X, lambda/3, 10*lambda/3, tol, maxIter);
```

Figure ۱۰ پیاده کردن الگوریتم Robust PCA بر روی وکتور

در انتها LowRank به عنوان پس زمینه و Sparse به عنوان پیش زمینه خواهد بود و این دو در فایل های ویدیو به نام های background.avi و foreground.avi ذخیره می شوند.

```
foreground = VideoWriter('foreground.avi');
background = VideoWriter('background.avi');
foreground.FrameRate = frame_rate;
background.FrameRate = frame_rate;
open(foreground);
open(background);
range = 255;
map = repmat((0:range)'./range, 1, 3);
sparseTensor = medfilt2(sparseTensor, [5,1]);
for i = (1:size(X, 1))
    frame1 = reshape(X(i,:),height,[]);
    frame2 = reshape(lowRankTensor(i,:),height,[]);
    frame3 = reshape(abs(sparseTensor(i,:)),height,[]);
    frame2 = mat2gray(frame2);
    frame3 = mat2gray(frame3);
    frame2 = gray2ind(frame2,range);
    frame3 = gray2ind(frame3,range);
    frame2 = im2frame(frame2,map);
    frame3 = im2frame(frame3,map);
    writeVideo(foreground,frame3);
    writeVideo(background,frame2);
end
close(foreground);
close(background);
```

Figure ۱۱ کد ذخیره سازی فیلم های جداسازی شده

Figure ۱۲ عکسی از ویدیو background

Figure ۱۳ عکسی از ویدیو foreground

حال با استفاده از این روش می خواهیم پس زمینه یک فریم را جداسازی کنیم.

```
lambda = 1 / sqrt(max(size(gray_film)));
maxIter = 500;
tol = 1e-5;
[L, S] = robust_pca(X, lambda/3, 10*lambda/3, tol, maxIter);
```

```
iter: 0330      err: 4.006767e-05
iter: 0340      err: 3.821857e-05
iter: 0350      err: 3.650916e-05
iter: 0360      err: 3.493128e-05
iter: 0370      err: 3.351610e-05
iter: 0380      err: 3.222424e-05
iter: 0390      err: 3.093129e-05
iter: 0400      err: 2.961311e-05
iter: 0410      err: 2.845872e-05
iter: 0420      err: 2.730526e-05
iter: 0430      err: 2.620304e-05
iter: 0440      err: 2.533683e-05
iter: 0450      err: 2.437387e-05
iter: 0460      err: 2.354148e-05
iter: 0470      err: 2.262001e-05
iter: 0480      err: 2.175412e-05
iter: 0490      err: 2.102730e-05
iter: 0500      err: 2.037651e-05
```

```
subplot(1,2,1)
imshow(L)
title("Low Rank Matrix", Interpreter = "latex");
subplot(1,2,2)
imshow(S)
title("Sparse Matrix", Interpreter = "latex");
```

Figure ۱۴ قطعه کد پیاده سازی الگوریتم Robust PCA روی یک فریم

نتایج آن به صورت زیر خواهد بود:

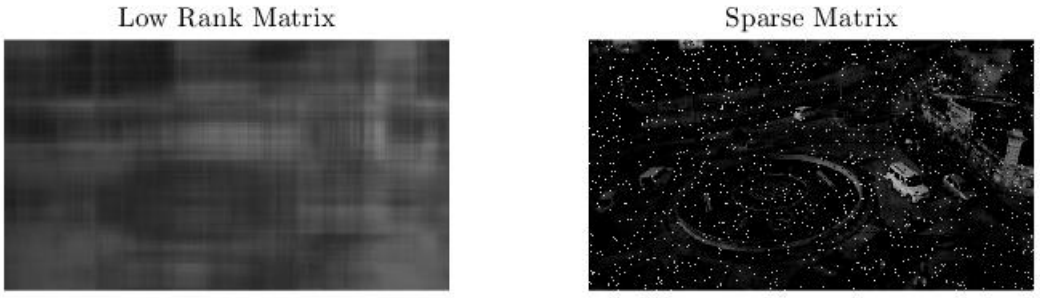

Figure ۱۵ نتایج پیاده سازی الگوریتم روی یک فریم

همان طور که قابل مشاهده است در این روش نمی توان تنها یک فریم را جداسازی کرد زیرا این روش جداسازی هر چه تعداد فریم های بیشتری داشته باشیم، جداسازی دقیق تر و بهتر صورت می گیرد.

نشانه گذاری دیجیتال:

سوالات مفهومی:

۵. Watermarking تکنیکی است که برای جاسازی اطلاعات، مانند لوگو یا اعلامیه حق چاپ، در محتوای دیجیتالی مانند تصاویر، صدا یا ویدئو استفاده می شود. دو نوع اصلی Watermark وجود دارد: قابل مشاهده و نامرئی.

۱. Watermark قابل مشاهده: در این روش، Watermark به وضوح روی محتوا قابل مشاهده است، که معمولاً در بالای تصویر یا ویدیوی اصلی قرار می‌گیرد. برای برندسازی یا حفاظت از حق چاپ استفاده می شود، اما به راحتی قابل جابجایی است.

۲. Watermark نامرئی: این روش اطلاعات را به گونه ای در محتوا جاسازی می کند که برای حواس انسان نامحسوس باشد اما با استفاده از الگوریتم های تخصصی قابل شناسایی یا استخراج باشد. این برای حفاظت از حق چاپ قوی تر است.

فرآیند کلی استخراج اطلاعات پنهان از Watermark های نامرئی شامل مراحل زیر است:

- تشخیص: اولین مرحله تشخیص وجود Watermark در محتوا است. این معمولاً شامل تجزیه و تحلیل محتوا و جستجوی الگوها یا تغییراتی است که نشان دهنده وجود Watermark است.
- محلی سازی: هنگامی که Watermark شناسایی شد، گام بعدی این است که مکان دقیق آن را در محتوا بومی سازی کنید. این تضمین می کند که فقط اطلاعات جاسازی شده بدون تغییر محتوای اصلی استخراج می شود.
- استخراج: پس از بومی سازی، اطلاعات تعبیه شده از محتوا استخراج می شود. این می تواند شامل تکنیک های مختلفی بسته به روش Watermarking مورد استفاده باشد. به عنوان مثال، در Watermark حوزه فرکانس، Watermark ممکن است در اجزای فرکانس خاصی از محتوا تعبیه شود که باید به دقت استخراج شوند.
- تایید: در نهایت اطلاعات استخراج شده برای اطمینان از صحت و صحت آن تایید می شود. این ممکن است شامل مقایسه آن با Watermark اصلی یا استفاده از تکنیک های رمزنگاری برای تأیید صحت آن باشد.

به طور کلی، Watermarking وسیله ای برای جاسازی و استخراج اطلاعات از محتوای دیجیتال، با اهداف مختلفی مانند حفاظت از حق نسخه برداری، احراز هویت و شناسایی محتوا فراهم می کند.

۶. Watermark مبتنی بر SVD روشی است که از تکنیک ریاضی SVD برای جاسازی و استخراج Watermark از محتوای دیجیتال مانند تصاویر استفاده می‌کند.

الگوریتم پنهان کردن:

۱. تصویر ورودی و Watermark: ماتریس A تصویر اصلی را نشان دهد و ماتریس W واترمارک را نشان دهد.

۲. انجام تجزیه SVD روی تصویر: ماتریس تصویر اصلی را با استفاده از SVD به سه ماتریس تجزیه کنید. در اینجا، U ماتریس متعامد بردارهای منفرد چپ، Σ ماتریس مورب مقادیر تکی است، و V^T ماتریس متعامد بردارهای منفرد سمت راست است.

۳. جاسازی واترمارک: مقادیر منفرد ماتریس تصویر را برای جاسازی واترمارک تغییر دهید. می توان با افزودن یا تفریق کسری از ماتریس واترمارک به/از مقادیر مفرد به دست آورد.

$$S + \alpha W = U_w S_w V_w^T$$

۴. بازسازی: ماتریس تصویر اصلاح شده را بازسازی می کنیم.

$$A_w = U B_w V^H$$

الگوریتم آشکارسازی:

۱. ورودی تصویر اصلاح شده: تصویر اصلاح شده A_w را به عنوان ورودی بگیرید.

۲. اجرای SVD: ماتریس تصویر اصلاح شده را با استفاده از SVD به سه ماتریس تجزیه کنید:

$$A_w^* = U^* B_w^* V_w^T$$

۳. جداسازی Watermark: این تفاوت نشان دهنده واترمارک تعبیه شده است:

$$D^* = U_w B_w^* V_w^T$$

$$W^* = \frac{1}{\alpha}(D^* - S)$$

۴. پس از پردازش اختیاری: از هر روش لازم پس از پردازش برای بهبود واترمارک استخراج شده و بهبود کیفیت یا دید آن استفاده کنید.

با دنبال کردن این مراحل، واترمارکینگ مبتنی بر SVD می‌تواند واترمارک‌ها را از تصاویر دیجیتال جاسازی و استخراج کند و در عین حال یکپارچگی و کیفیت محتوای اصلی را حفظ کند.

۷. محو کردن تصویر یک تکنیک رایج است که در پردازش تصویر برای کاهش نویز، صاف کردن جزئیات یا محو کردن مناطق خاص استفاده می شود. یکی از روش های محبوب برای محو کردن تصاویر، استفاده از Gaussian Blur است.

Gaussian Blur:

تعریف کرنل: هسته گاوسی ماتریسی از مقادیر است که اثر تاری را تعریف می کند. مقادیر موجود در هسته طبق یک توزیع گاوسی وزن می‌شوند، با وزن‌های بالاتر به سمت مرکز و وزن‌های کمتر به سمت لبه‌ها.

اثر اندازه هسته: اندازه هسته میزان تاری را تعیین می کند. اندازه هسته بزرگتر منجر به تاری قابل توجهی می شود، زیرا در هنگام محاسبه مقدار تاری برای هر پیکسل، همسایگی بزرگتری از پیکسل ها در نظر گرفته می شود. با این حال، هسته های بزرگتر نیز زمان محاسبه را افزایش می دهند.

ایجاد فیلتر:

- تولید هسته: برای ایجاد یک فیلتر گاوسی تاری، ابتدا یک تابع گاوسی دو بعدی بر اساس انحراف استاندارد مورد نظر (سیگما) و اندازه هسته تولید می‌کنید.
- Normalization: هسته را normalize کرده تا مطمئن شده که مجموع تمام مقادیر آن برابر با ۱ است. این تضمین می کند که روشنایی تصویر پس از محو شدن ثابت بماند.
- پیچیدگی: با استفاده از کانولوشن، هسته گاوسی را روی تصویر اعمال کنید. Convolution شامل لغزش هسته روی تصویر و محاسبه مجموع وزنی مقادیر پیکسل در همسایگی هسته برای هر پیکسل در تصویر خروجی است.

تاثیر اندازه هسته بر خروجی:

- اندازه هسته کوچکتر: جلوه تاری ظریفی ایجاد می کند و جزئیات بیشتری را در تصویر حفظ می کند.
- اندازه هسته بزرگتر: منجر به جلوه تاری بارزتر می شود، جزئیات ریزتر را صاف می کند اما به طور بالقوه وضوح و وضوح را کاهش می دهد.

به طور خلاصه، برای محو کردن یک تصویر با استفاده از فیلتر گاوسی، یک هسته گاوسی ایجاد می‌کنید، آن را عادی می‌کنید و آن را با تصویر اصلی در هم می‌پیچید. تنظیم اندازه هسته به شما این امکان را می‌دهد که میزان تاری را کنترل کنید، با اندازه‌های بزرگتر هسته که منجر به جلوه‌های تاری بارزتر می‌شود.

پیاده سازی:

**به دلیل اینکه لپتاپم نمی توانست کل ویدیو را پردازش کند فقط عملیات را روی یک فریم انجام دادم.

در ابتدا ویدیو و تصویر را خوانده و سایز تصویر را با رزولوشن تصویر ویدیو یکسان می کنیم:

```
video = VideoReader('background.avi');
hidden_image = imread('3.jpg');
height = video.Height;
width = video.Width;
hidden_image_resized = imresize(hidden_image, [height, width]);
```

Figure ۱۶ خواندن ویدیو و تصویر و یکسان نمودن سایز تصویر با ویدیو

حال یک فریم را انتخاب کرده و عملیات نهان سازی را روی آن انجام می دهیم:

```
frame_index = 100;
frame = read(video, frame_index);
[rows, cols, ~] = size(frame);

[U, S, V] = svd(double(rgb2gray(frame)));

alpha = 0.01;
S_new = S + alpha * double(rgb2gray(hidden_image_resized));
[U_w, S_w, V_w] = svd(S_new);
frame_hidden = uint8(U * S_new * V.');
```

Figure ۱۷ کد عملیات نهان سازی

در مرحله بعد عملیات آشکار سازی را داریم:

```
[U_1, S_w1, V_1] = svd(double(frame_hidden));
hidden_image_reconstructed = uint8(U_w * S_w1 * V_w');
hidden_image_extracted = 1/alpha*(double(hidden_image_reconstructed) - double(S));
```

Figure ۱۸ کد آشکار سازی

نمودار عناصر ماتریس مقادیر منفرد به صورت زیر خواهد بود:

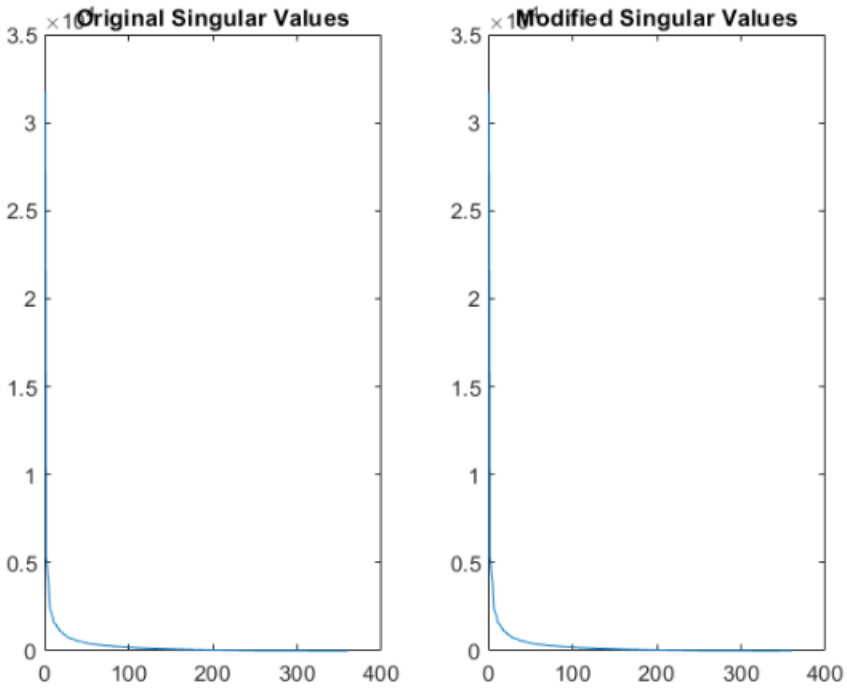

Figure ۱۹ نمودار عناصر ماتریس مقادیر منفرد قبل و پس از پنهان سازی به طور مجزا

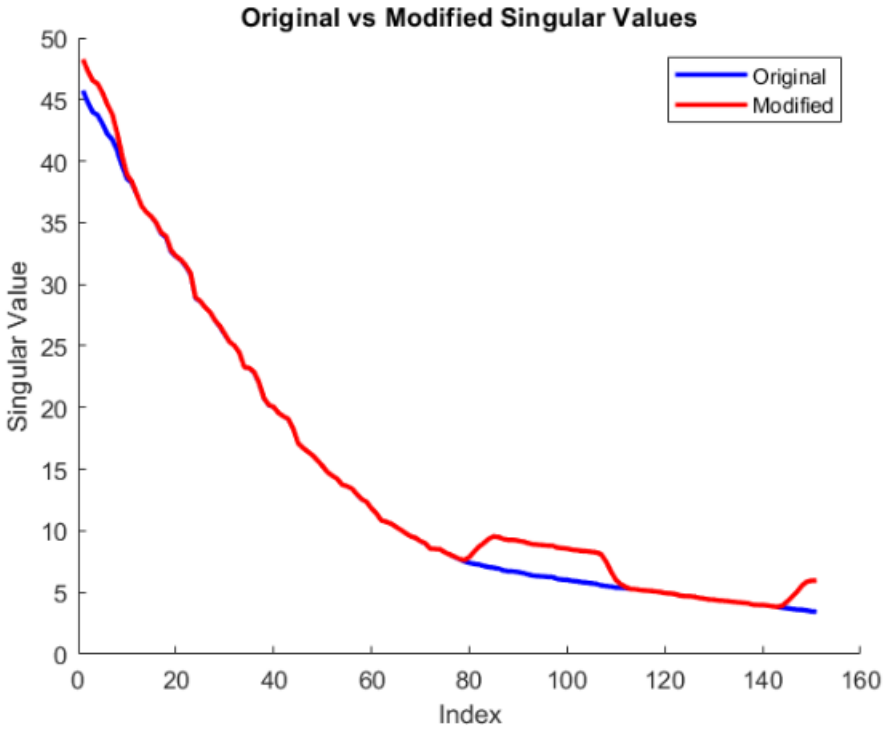

Figure ۲۰ نمودار عناصر ماتریس مقادیر منفرد قبل و پس از پنهان سازی

همان طور که قابل مشاهده است عناصر ماتریس مقادیر منفرد پیش و پس از پنهان سازی با یکدیگر تفاوت دارند و این اتفاق به این دلیل است که اطلاعات مخفی به تصویر پس از پنهان سازی اضافه شده است.

در نهایت تصویر پیش و پس از پنهان سازی و همچنین تصویر واترمارک پس از آشکار سازی به صورت زیر خواهد بود:

Figure ۲۱ تصاویر بدست آمده

در مرحله بعد یک kernel با سایز ۵ تشکیل داده. این هسته را گاوسی در نظر گرفته ایم به این دلیل که فیلتر گاوسی باعث ایجاد یک جریان نرم و ملایم در تصویر می شود که به معنای حذف جزئی اطلاعات و جزئیات تصویر است که منجر به کاهش نویز و ناهواری های تصویر می شود. همچنین دارای پارامترهایی مانند اندازه کرنل و انحراف معیار است تا تاثیر و دقت Blur را تنظیم کند.

```
filter_size = 5;
sigma = 1.5;
filter_kernel = fspecial('gaussian', [filter_size filter_size], sigma);

frame_filtered = imfilter(frame_hidden, filter_kernel, 'replicate');

[U_filtered, S_filtered, V_filtered] = svd(double(im2gray(frame_filtered)));
S_new_filtered = S_filtered + alpha * double(rgb2gray(hidden_image_resized));
frame_hidden_filtered = uint8(U_filtered * S_new_filtered * V_filtered');
```

Figure ۲۲ کد اعمال فیلتر و رمزنگاری

در نهایت این فیلتر را بر روی فریم رمزنگاری شده پیاده سازی کردهو نتایج را به صورت زیر خواهیم داشت:

Figure ۲۳ تصاویر پیش و پس از اعمال فیلتر

همان طور که در تصویر قابل مشاهده است، پس از اعمال فیلتر تصویر وضوح بیشتری دارد و جزئیات کم اهمیت و نویز از آن حذف شده است.