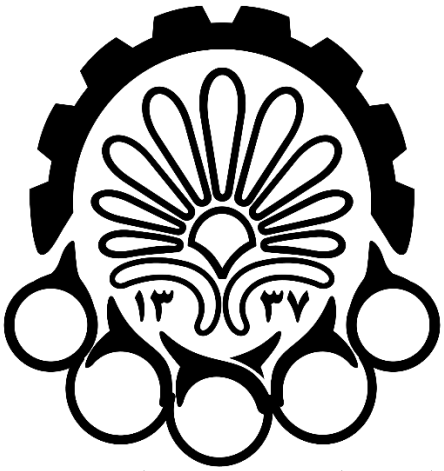


دانشگاه صنعتی امیرکبیر

(پلی تکنیک تهران)

دانشکده مهندسی کامپیوتر

تمرین چهارم درس پردازش تصویر

دکتر رحمتی

غلامرضا دار ۴۰۰۱۳۱۰۱۸

بهار ۱۴۰۱

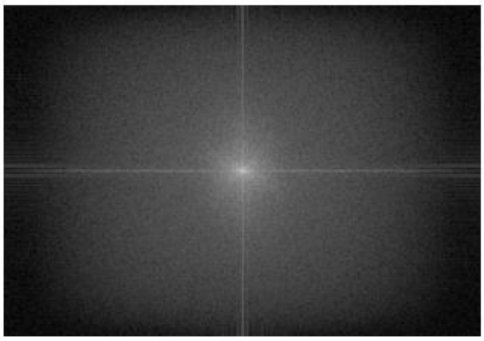
فهرست مطالب

سوال ۱)

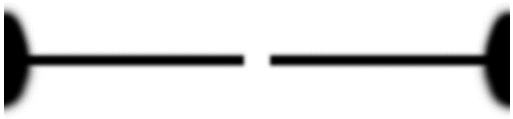
در این سوال قصد داریم با استفاده از تکنیک ها و ابزارهای مختلف، تصاویر داده شده که لزوما از کیفیت بالایی برخوردار نیستند را بهبود ببخشیم.

تصویر a)

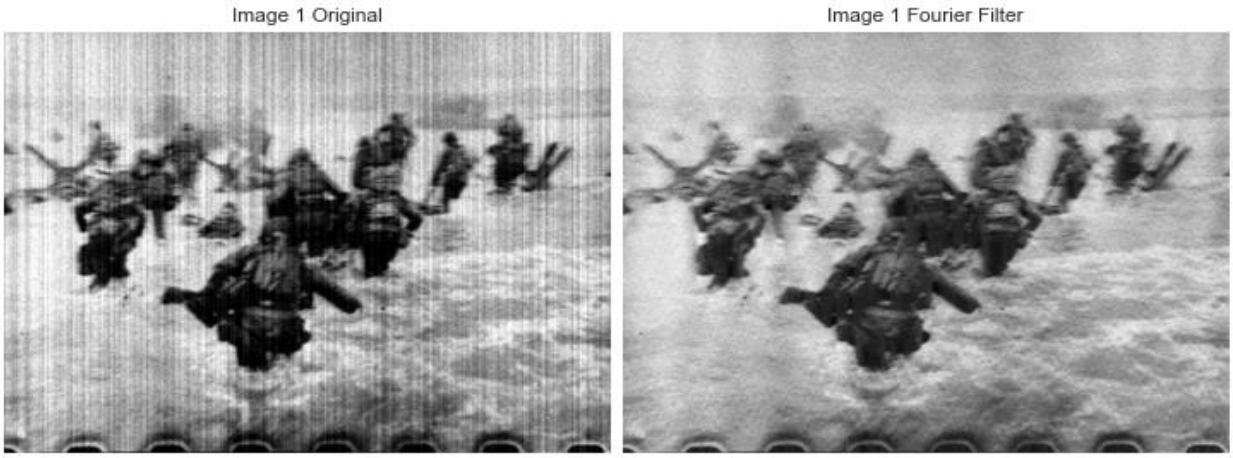
با یک نگاه کلی به این تصویر متوجه می‌شویم که این تصویر دارای خطوط پرودیک در راستای افقی است. بهترین راه برای حذف این نوع نویز، استفاده از تبدیل فوریه است. در شکل زیر طیف تبدیل فوریه تصویر را مشاهده می‌کنید.



با تولید ماسک زیر و اعمال آن به طیف فوریه، سعی می‌کنیم این تصویر را بهبود بخشیم.

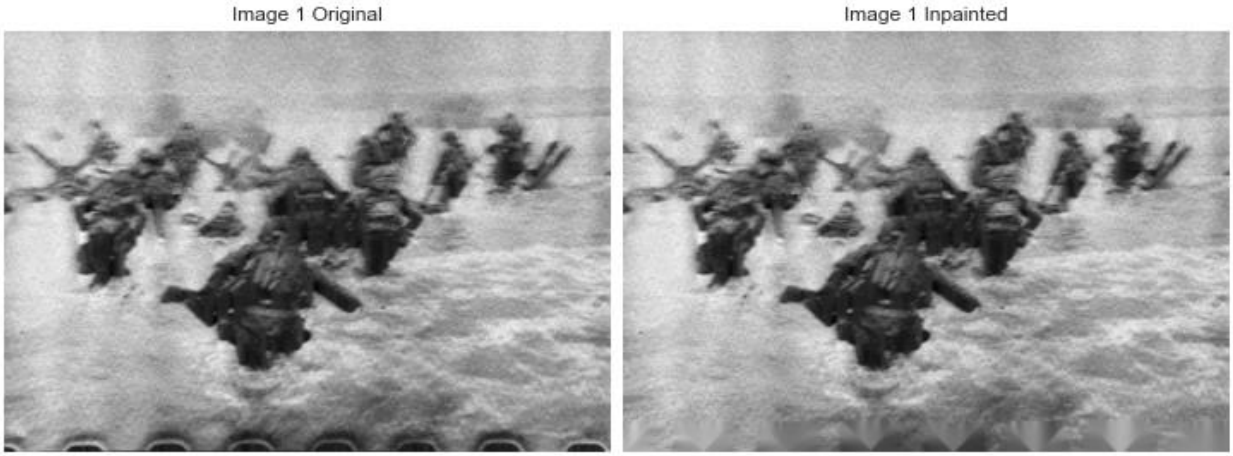


نتیجه اعمال فیلترینگ به کمک فوریه را مشاهده میکنید.

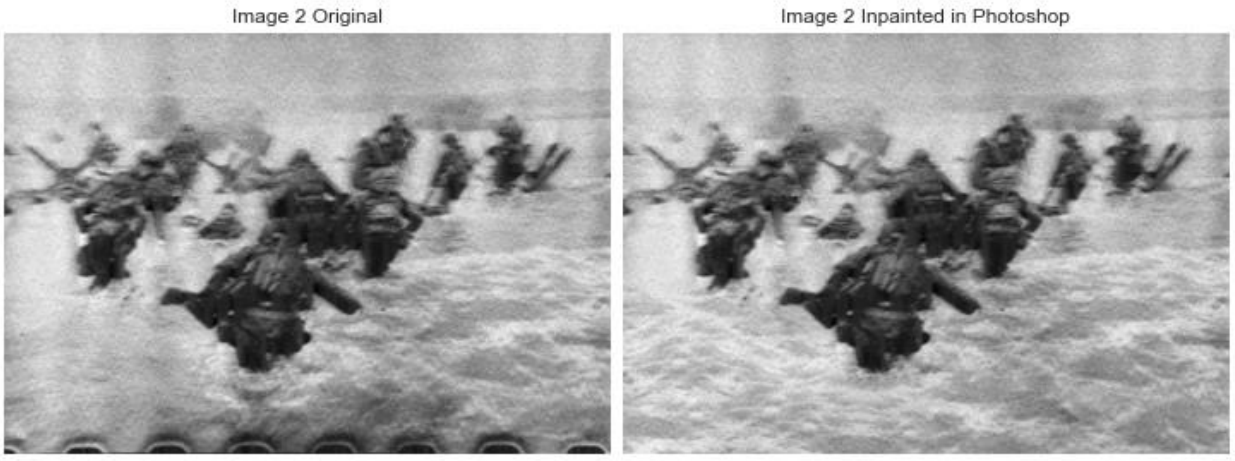


اگر به قسمت پایین تصویر دقت کنید مشاهده می‌شود که سوراخ های موجود بر روی فیلم، به تصویر دیجیتالی نیز انتقال داده شده اند. برای حذف این نواحی از تکنیک Image Inpainting استفاده میکنیم. برای این کار لازم است یک ماسک از نواحی دارای مشکل ایجاد کنیم.

نتیجه اعمال Inpainting را در تصویر زیر مشاهده میکنید.

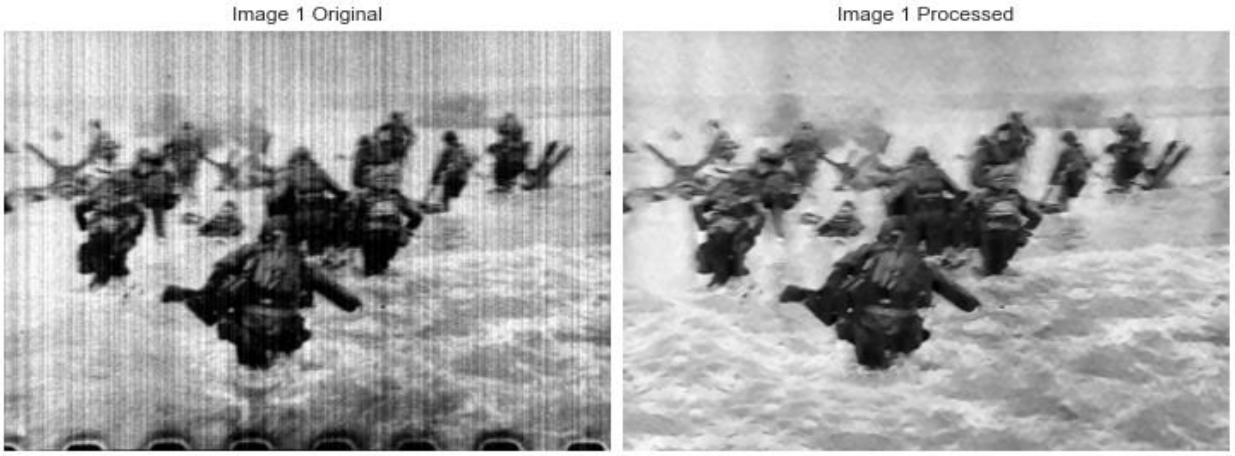


متاسفانه الگوریتم های موجود در کتابخانه opencv برای inpainting خیلی خوب عمل نمی‌کنند. به همین دلیل برای این قسمت از ابزار Content Aware Fill فوتوشاپ که عملکرد بهتری دارد استفاده میکنیم.



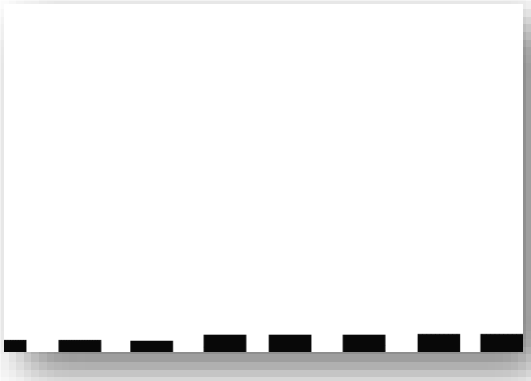
و پس از اعمال یک فیلتر بالاگذر، نتیجه نهایی بدست می‌آید.

و در مقایسه با تصویر اولیه:



تصویر b)

مانند تصویر قبل این تصویر نیز شامل بخش هایی در پایین تصویر است که نیاز به ترمیم به کمک Inpainting دارد. ماسک تولید شده برای این تصویر را مشاهده میکنید.



نتیجه اعمال Inpainting را در تصویر زیر مشاهده میکنید. همانطور که انتظار می‌رفت، نواحی مشکی پایین تصویر از بین رفتند.

در ادامه تصمیم گرفتیم تصویر را کمی Sharp تر کنیم. به این منظور ابتدا از Unsharp Mask استفاده کردیم.

نتایج کمی ضعیف تر از حد انتظار بود بنابراین تصمیم گرفتیم از Laplacian برای استخراج ویژگی های High frequency و اعمال آنها به تصویر استفاده کنیم.

تصویر c)

از مشکلات اصلی تصویر C وجود نویز فراوان در تصویر است. بنابراین قبل از هر کاری سعی میکنیم نویز موجود در تصویر را کاهش دهیم. برای این منظور ابتدا از Bilateral Filter استفاده میکنیم که یک نوع فیلتر Adaptive است. این فیلتر سطوح همرنگ را صاف میکند اما لبه ها را دست نخورده باقی می‌گذارد. یکی دیگر از الگوریتم‌های موجود برای کاهش نویز که در OpenCV پیاده سازی شده است، NLMeansDenoise نام دارد که به صورت محلی و با جستجوی نواحی یک شکل در یک پنجره مشخص، به رفع نویز تصویر می‌پردازد.

بعد از آزمایش های متعدد به این نتیجه رسیدیم که ابتدا NLMeansDenoise را اعمال کنیم و سپس BilateralFiltering را بر روی نتیجه آن اعمال کنیم. نتیجه این دو عمل را در تصویر زیر مشاهده میکنید.

در نهایت می‌توانید نتیجه بهبود ها را مشاهده کنید.

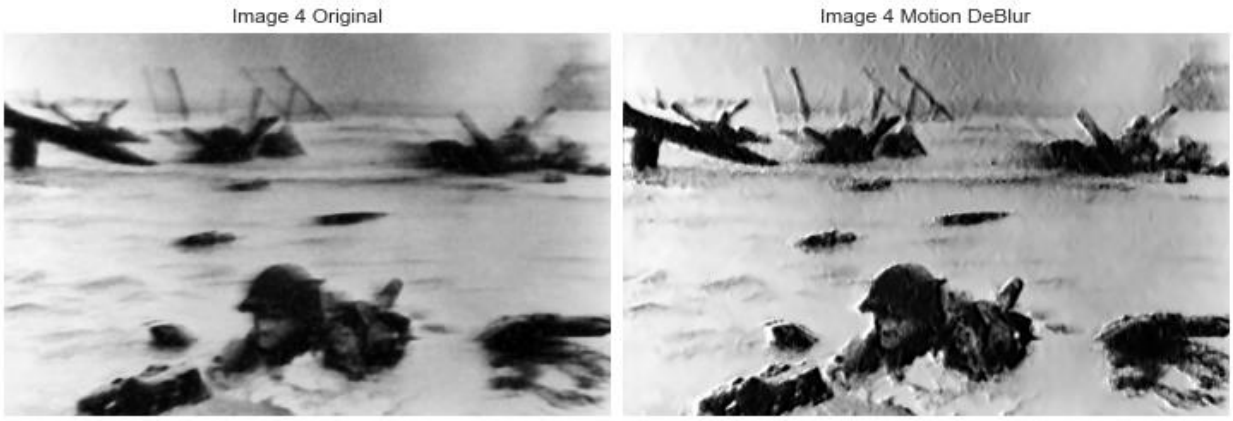
به عنوان تحقیق و بررسی بیشتر سری به تکنولوژی های کاهش نویز بر پایه Deep Learning زدیم. تکنولوژیOpen Image Denoise از کمپانی Intel را برای این بررسی ها انتخاب کردیم. نتیجه اعمال OIDN به تصویر C را می‌توانید در زیر مشاهده کنید.

پس از اندکی تنظیم رنگ به نتیجه زیر میرسیم. همانطور که مشاهده میشود این روش نویز را بسیار عالی حذف میکند اما مقداری Artifact به جای میگذارد.

تصویر d)

همانطور که مشاهده می‌کنید این تصویر دارای مقداری Motion Blur تقریبا افقی است. می‌خواهیم با استفاده از یک فیلتر به شکل زیر اثر این Blur را خنثی کنیم. البته قبل و بعد از اعمال این فیلتر کمی تصویر را Denoise میکنیم.

-۱	۰	۱
-۲	۱	۲
-۱	۰	۱



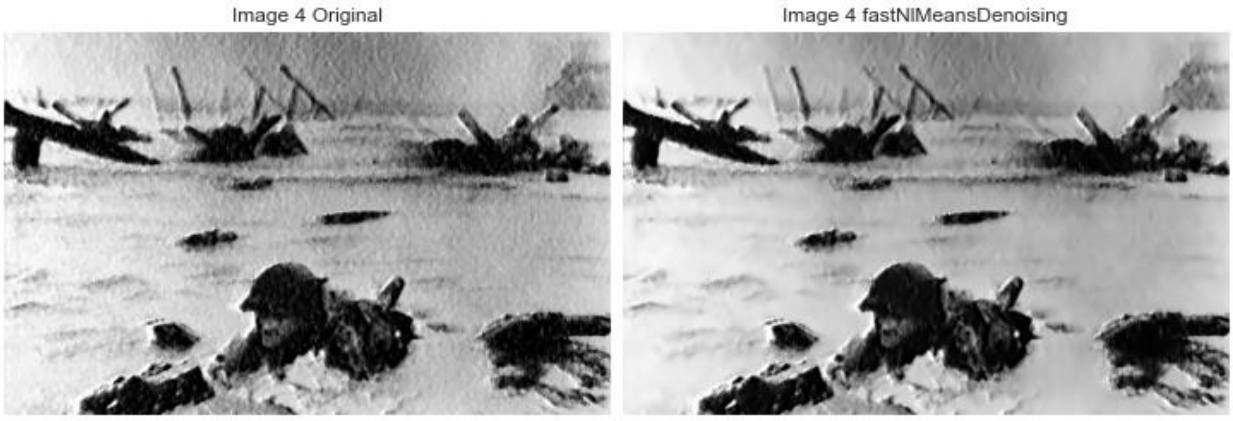
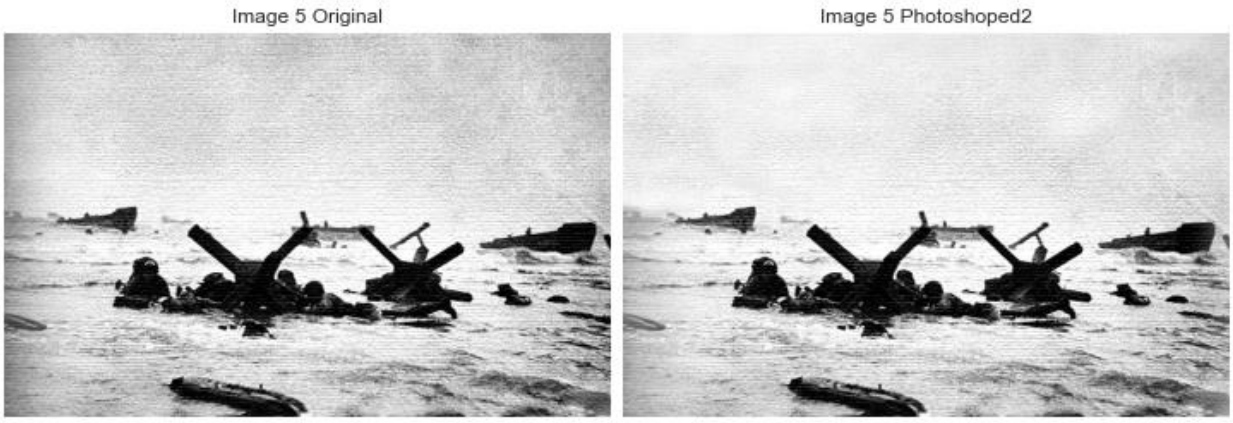


Image 4 Original
Image 4 fastNlMeansDenoising


Image 4 Original
Image 4 Processed

تصویر e)

با نگاه کردن به این تصویر متوجه می‌شویم که یکی از مشکلات اصلی این تصویر وجود Vignette در اطراف تصویر است. و این Vignette ساختار هندسی منظمی نیز ندارد(به عنوان مثال به صورت یک بیضی نیست) بنابراین حذف کردن اتوماتیک آن از تصویر کار دشواری است و احتمالا بهترین نتیجه را نیز نمی‌دهد. بنابراین برای حذف Vignette از این تصویر از فوتوشاپ و روش های دستی کمک گرفتیم. سایر مراحل بهبود این تصویر اما به صورت اتوماتیک انجام می‌شوند.



نتیجه اعمال مراحل قبل:

و کمی تنظیم نور :

تصویر f)

این تصویر شباهت زیادی به تصویر C دارد و روش بهبود آن نیز به همین جهت شبیه به تصویر C است. همچنین این تصویر دارای یک لکه مشکی رنگ در سمت چپ تصویر است که با استفاده از Inpainting بر طرف خواهد شد.

ابتدا با کمک Bilateral Filtering سطح نویز را کاهش می‌دهیم.

سپس با کمک NlMeansDenoise روند کاهش نویز را ادامه می‌دهیم.

دیدیم که Inpainting برای این تصویر کمی با مشکل روبرو شد به همین دلیل نتیجه Inpainting را نیز کمی نرم میکنیم.

حالا که نویز از تصویر حذف شده، تصویر را کمی Sharp میکنیم تا لبه ها و جزییات از دست رفته به دلیل Blur های متعدد تا حدی بازگردند.

مشابه تصویر C، این تصویر را نیز با کمک OIDN رفع نویز کردیم. حاصل را در تصویر زیر مشاهده میکنید. همچنین در این تصویر لکه موجود در تصویر به صورت دستی حذف شده. در این مثال، روش OIDN بسیار بهتر عمل کرد(به صاف بودن آسمان توجه کنید).

Image 1 Original

Image 1 Processed

Image 2 Original

Image 2 Processed

Image 5 Original

Image 5 Processed

Image 3 Original

Image 3 Processed

Image 6 Original

Image 6 Processed

سوال ۲)

در این سوال قصد داریم با روش های مختلف Interpolation جهت افزایش سایز تصاویر آشنا شویم.

A) روش Nearest Neighbor

پیاده سازی این روش بسیار ساده بود و تنها کاری که انجام دادیم این بود که هر چهار پیکسل از تصویر بزرگ، به یک پیکسل از تصویر اصلی نگاه میکردند و مقدار آن پیکسل را به عنوان مقدار خود برمی‌گزیدند.

B) روش Bilinear Interpolation

در این روش، ابتدا هر پیکسلِ تصویر بزرگ را به فضای تصویر کوچک میبریم(با تقسیم بر ۲) و ۴ پیکسل اطراف آن را مشخص میکنیم.

```
A = img[int(np.floor(i/s)), int(np.floor(j/s))]
B = img[int(np.floor(i/s)), int(np.ceil(j/s))]
C = img[int(np.ceil(i/s)), int(np.floor(j/s))]
D = img[int(np.ceil(i/s)), int(np.ceil(j/s))]
```

مقدار نهایی پیکسل مدنظر، برابر با ترکیب وزن دار این ۴ پیکسل می‌شود. برای محاسبه وزن ها از رابطه زیر استفاده میکنیم.

```
delta_x = j/s - np.floor(j/s)
delta_y = i/s - np.floor(i/s)

interpolated_value = \
    A*(1-delta_x)*(1-delta_y) +\
    B*(delta_x)*(1-delta_y)   +\
    C*(1-delta_x)*(delta_y)   +\
    D*(delta_x)*(delta_y)
```

نتیجه این Interpolation را در تصویر زیر مشاهده میکنید.

C) روش Nearest Value

این روش همانطور که در توضیحات سوال آمده ترکیب دو روش قبلی است و نزدیک‌ترین مقدار به مقدار تخمین زده شده توسط Bilinear Interpolation را انتخاب میکند.

نتیجه این Interpolation را در تصویر زیر مشاهده میکنید.

D) روش Non-Uniform

در این روش، ۴ تصویر ورودی داریم که اختلاف Sub-pixel ـی دارند. مقدار نهایی هر پیکسل در تصویر بزرگ، ترکیب وزن‌دار ۴ پیکسل نظیر(به کمک تقسیم بر ۲ پیکسل های نظیر را پیدا میکنیم) در تصاویر ورودی است. وزن هر تصویر، متمم Shift ـی است که آن تصویر از مبداء دارد.

برای اطلاع از جزییات پیاده سازی به نوت بوک مربوطه مراجعه شود.

نتیجه این Interpolation را در تصویر زیر مشاهده میکنید.

نکته: از آنجایی که برای پیاده سازی این توابع از Loop های بزرگ استفاده کردیم. بهتر است از Numba برای تسریع این توابع استفاده کنیم. در آزمایش زیر روش Bilinear Interpolation با استفاده از پایتون خام و Numba مورد مقایسه قرار گرفتند.

```
[10]
%%timeit
resized_img_bilinear = bilinear_interpolation(img_lr)

5.22 s ± 321 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)

[11]
%%timeit
resized_img_bilinear = bilinear_interpolation_jit(img_lr)

1.73 ms ± 148 µs per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

در این آزمایش، پایتون خام در ۵ ثانیه تصویر را تغییر اندازه میدهد و Numba در ۲ میلی ثانیه!

e) مقایسه PSNR

Method	PSNR
Cubic Interpolation (cv2)	25.60483129443059
Bilinear Interpolation (cv2)	25.118872531164126
Nearest Neighbor Interpolation (cv2)	25.027313181900475
Nearest Neighbor Interpolation	25.027313181900475
Bilinear Interpolation	23.354621957344435
Nearest Value Interpolation	23.486551575715033
Non Uniform Interpolation	22.786197269802905

سوال ۳)

در این سوال ابتدا مختصات گوشه های پنجره را بدست می‌آوریم و با کمک آنها طول و عرض تصویر حاصل را محاسبه میکنیم. لازم به ذکر است که در تصویر سمت راست، نقطه آبی رنگ از قصد بر روی کنج پنجره قرار نگرفته است. آزمایش ها نشان داد که این حالت نتیجه بهتری تولید میکند.

پس از بدست آوردن اندازه تصویر جدید و نقاط مدنظر، با استفاده از تابع get_perspective_transform() موجود در کتابخانه OpenCV، یک ماتریس تبدیل برای تبدیل کردن تصویر داده شده، به یک تصویر صاف بدست می‌آوریم. در این تصویر صاف، نقاط آبی، قرمز،زرد و سبز به گوشه های تصویر حاصل برده می‌شوند. در صفحه بعد میتوانید نتیجه این تبدیل ها را مشاهده کنید.

Image 1 Original

Image 1 Just Warped

Image 2 Original

Image 2 Just Warped

همانطور که مشاهده می‌شود این تصاویر کیفیت لازم را ندارند بنابراین در مرحله بعد با کمک Unsharp Mask سعی میکنیم کمی کیفیت این تصاویر را بهبود بدهیم.

Image 1 Original

Image 1 Enhanced

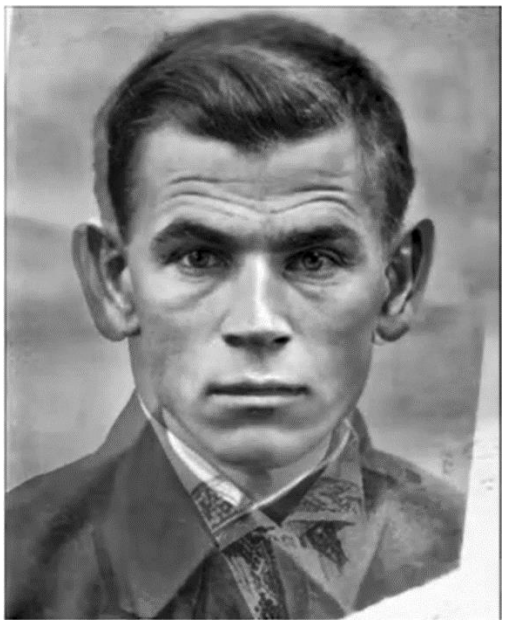
Image 2 Original

Image 2 Enhanced

سوال ۴)

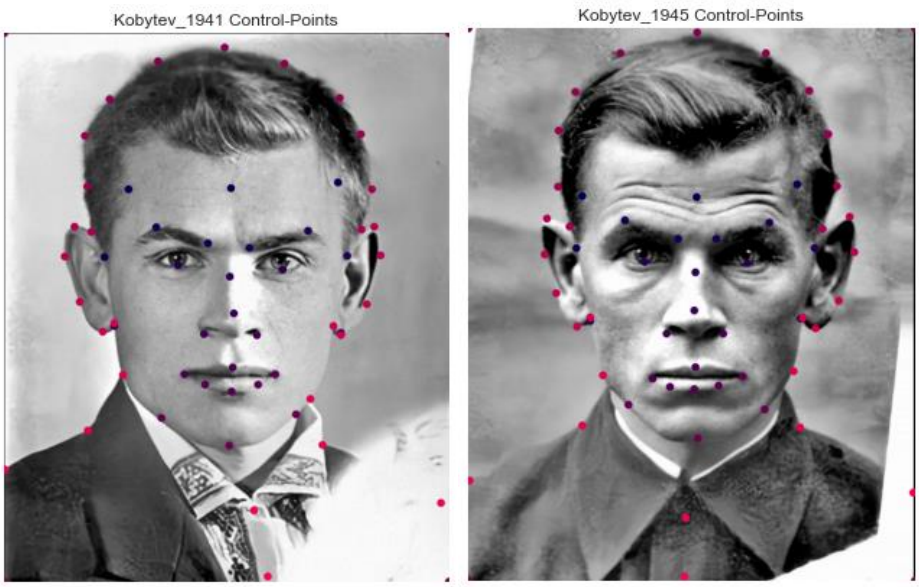
در این سوال قصد داریم دو روش برای Face Morphing را مقایسه کنیم. روش اول به موقعیت پیکسل ها کاری ندارد و صرفا مقادیر آنها را میکس میکند. روش دوم علاوه بر مقدار پیکسل ها، موقعیت پیکسل های نظیر را نیز میکس میکند که نتیجه بسیار بهتری بدست می‌آورد. برای بدست آوردن این پیکسل های متناظر، ابتدا یک سری نقطه کنترلی یکسان بر روی دو تصویر تولید میکنیم و از روی این نقاط یک مثلث بندی بر روی دو تصویر ایجاد میکنیم. پیکسل های مثلث های متناظر، هم از لحاظ مقدار و هم از لحاظ موقعیت هندسی، به یکدیگر تبدیل می‌شوند.

a) ابتدا با استفاده از روش Naiive سعی میکنیم دو چهره را به یکدیگر تبدیل کنیم. در این روش با استفاده از Linear Interpolation، مقدار هر پیکسل با مقدار پیکسل نظیر میکس می‌شود. این میکس شدن به کمک یک پارامتر α کنترل می‌شود. با تغییر دادن این پارامتر از ۰ تا ۱، نتیجه بین دو تصویر ورودی حرکت خواهد کرد.
پیشنهاد می‌شود ویدیوی مربوط به این بخش را در پوشه مربوطه مشاهده کنید. در زیر تصویر مربوط به حالتی که $\alpha = 0.5$ است را مشاهده میکنید.



میبینیم که افکت Ghosting یا Double Exposure که در این کاربرد نامطلوب است مشاهده می‌شود. دلیل این اتفاق مچ نبودن صددرصدی تصاویر است. البته لازم به ذکر است که این تصاویر تا حد بسیار زیادی با هم مچ هستند. اگر در این جا از تصاویر دیگری استفاده میکردیم نتیجه این متود از این هم ناخوشایند تر میشد.

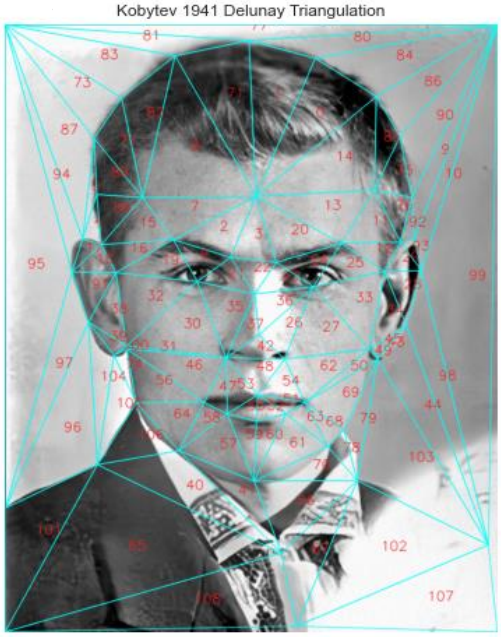
b) در این مرحله با استفاده از ابزاری که برای یکی از تمرین های قبل ساخته بودیم(point_picker) به انتخاب keypoint هایی میپردازیم. این نقاط در مرحله بعد برای مثلث بندی تصاویر مورد استفاده قرار میگیرند.
نکته: طبق آزمایش های انجام شده متوجه شدیم که بهتر است تعداد نقاط بیشتری نسبت به نقاط پیشنهاد شده در صورت سوال انتخاب کنیم.



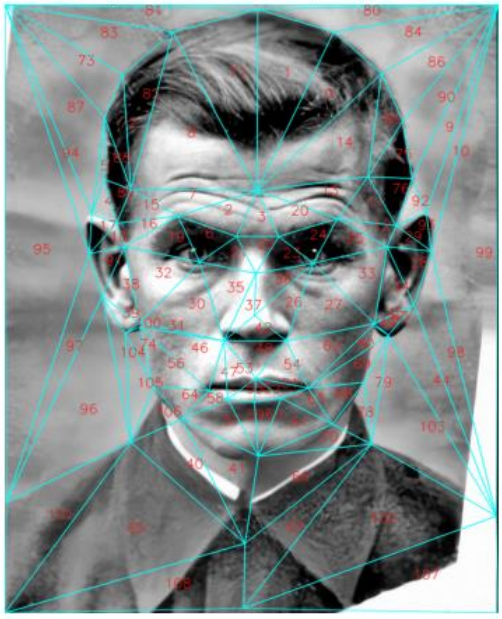
رنگ نقاط بر اساس ID آنها تنظیم شده است.

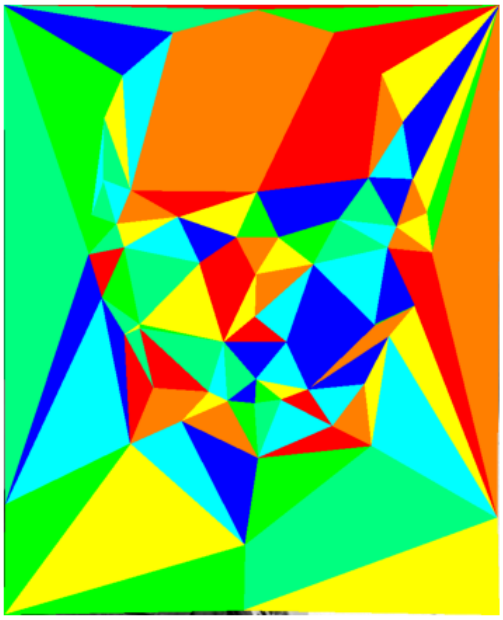
c) از نقاط بدست آمده در مرحله قبل استفاده میکنیم و با کمک Delunay Triangulation **فقط تصویر اول** را مثلث بندی میکنیم. نکته بسیار مهمی که وجود دارد این است که نباید مثلث بندی را با کمک الگوریتم Delunay Triangulation بیش از یک بار انجام دهیم. زیرا تضمینی وجود ندارد که مثلث های تولید شده توسط نقاط تصویر اول و تصویر دوم دقیقا به یک ترتیب باشند و مفهوم یکسانی نیز داشته باشند. این یکسان بودن مفهوم مثلث ها در این روش بسیار مهم است. بنابراین برای مثلث بندی تصویر دوم، از همان مثلث های تصویر اول استفاده میکنیم. به این شکل که پس از انجام مثلث بندی توسط Delunay Triangulation بر روی تصویر اول، به هر نقطه ID مثلثی که در آن قرار گرفته را میدهیم. از آنجایی که این نقاط دقیقا در تصویر دوم نیز وجود دارند، میتوانیم این اطلاعات را به تصویر دوم بدهیم و به صورت خودکار خواهیم فهمید که هر نقطه در تصویر دوم در کدام مثلث قرار دارد. با این کار تصویر دوم نیز مثلث بندی خواهد شد و مثلث ها با مثلث های تصویر اول دقیقا یکسان هستند(زیرا نقاط سازنده آنها یکسان هستند)

Kobytev 1941 Delunay Triangulation

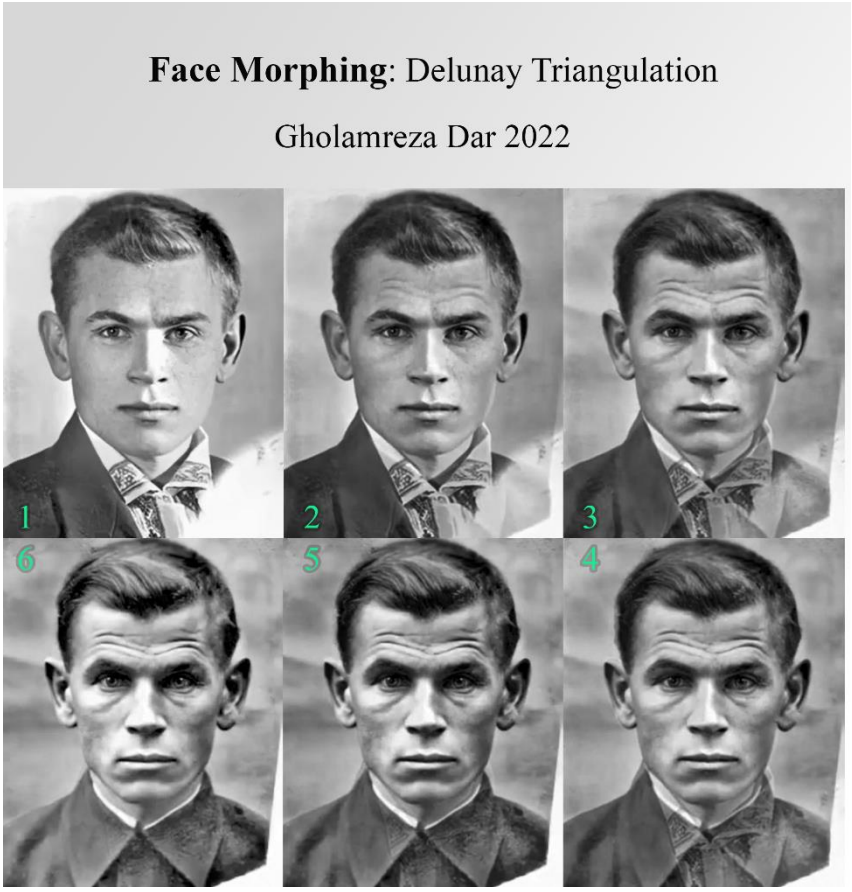


Kobytev 1945 Matched Delunay Triangulation





d) با استفاده از روابط و توضیحات داده شده در صورت سوال، این قسمت پیاده سازی شد و انیمیشن خواسته شده در قسمت بعد نیز تولید شد. پیشنهاد می‌شود برای این قسمت و قسمت بعد، به **پوشه** مربوطه مراجعه کنید و **ویدیو** تولید شده را مشاهده کنید.



e) انجام شد

سوال ۵)

a) برای این کار سه ماتریس تبدیل برای سه عمل تولید میکنیم. این ماتریس ها را در مختصات homogeneous قرار می‌دهیم تا بتوانیم آنها را با ضرب ماتریسی به یک ماتریس ترکیبی تبدیل کنیم.

$$A = \begin{bmatrix} 3 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$B = \begin{bmatrix} \cos(30) & \sin(30) & 0 \\ -\sin(30) & \cos(30) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$C = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 20 \\ 0 & 0 & 1 \end{bmatrix}$$

ماتریس های A,B و C به ترتیب ماتریس های تجانس، دوران و انتقال هستند. برای ترکیب کردن آنها کافیست آنها را در هم ضرب کنیم و ماتریس حاصل را بدست بیاوریم.

$$T = ABC$$
$$x_{new} = Tx$$

b) می‌دانیم که تبدیل خطی توانایی انجام انتقال را ندارد و همواره $T(0) = 0$ برقرار است. برای پشتیبانی از انتقال و انجام دادن Affine Transformation، از Homogeneous Coordinates استفاده میکنیم. با افزودن یک ستون به ماتریس ۲ در ۲ ای که در ترکیب خطی استفاده میکردیم به ماتریس زیر میرسیم که انتقال را نیز پشتیبانی میکند.

$$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x + t_x \\ y + t_y \\ 1 \end{bmatrix}$$

c) به الگوریتم Interpolation بستگی دارد. به عنوان مثال اگر از Nearest Neighbor استفاده کنیم تضمین می‌شود که همگرا شویم. اما اگر از روش‌های دیگر استفاده کنیم ممکن است این اتفاق نیفتد. به طور کلی، با بزرگ کردن یک تصویر، تعدادی اطلاعات جدید به دست می‌آید، اگر تضمین شود که با کوچک کردن دوباره این تصویر، اطلاعاتی که قرار است حذف شوند، همین اطلاعات جدید به دست آمده باشند، همگرایی بوجود می‌آید. به عنوان مثال در NN این اتفاق رخ می‌دهد.

d) بله، چون تمام تبدیلات هندسی دارای ماتریس های Invertible هستند ماتریس معکوس آن تبدیل وجود دارد و در نتیجه عملیات برگشت پذیر است. دلیل Invertible بودن ماتریس های تبدیلات هندسی این است که این تبدیلات Degenerate نیستند به این معنی که یک بردار را به یک نقطه تبدیل نمی‌کنند. به عنوان مثال تجانس با مقایس ۰ همه نقاط را به یک نقطه(0,0) تبدیل میکند در نتیجه Degenerate است و معکوس پذیر نیست.

e) نویز پریودیک، یک نویز quantitative است زیرا در سیگنال ورودی Convolve می‌شود.

نویز Salt & pepper یک نوع نویز Additive است به این معنی که به یک سیگنال ورودی اضافه شده است.

$$g = f + \nu$$
$$f = g - \nu$$

که در آن ν یک نویز Additive است و f تصویر اصلی بدون نویز است و g تصویر نویزی است.