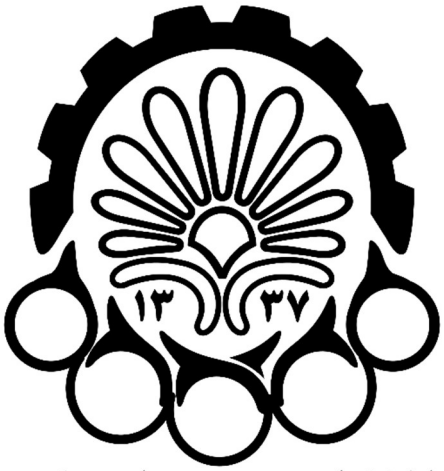


دانشگاه صنعتی امیرکبیر

(پلی تکنیک تهران)

دانشکده مهندسی کامپیوتر

تمر ین اول درس تصویرپردازی رقمی

دکتر رحمتی

غلامرضا دار ۴۰۰۱۳۱۰۱۸

بهار ۱۴۰۱

فهرست مطالب

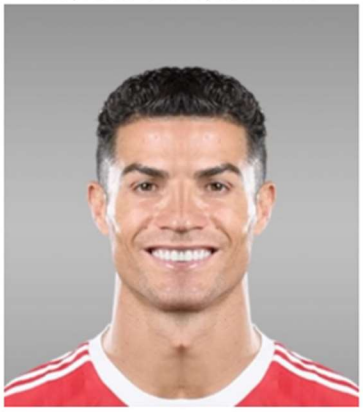
سوال ۱)

برای حل این سوال ابتدا عرض تصاویر را زوج میکنیم. سپس تصاویر را در جهت افقی به دو بخش مساوی تقسیم میکنیم. در نهایت با استفاده از تابع image_mirror پیاده سازی شده، برخی تصاویر را آینه میکنیم و در کنار تصاویر اصلی قرار میدهیم تا ترکیب های مورد نظر ساخته شود. در نهایت با استفاده از معیارهای SSIM , PSNR میزان شباهت تصاویر تولید شده و تصاویر اصلی را می‌سنجیم.

cristiano_ronaldo

LL | PSNR= 34.534 | SSIM= 0.850

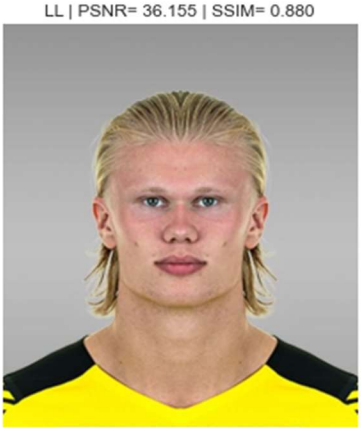
RR | PSNR= 34.534 | SSIM= 0.850



RL | PSNR= 31.523 | SSIM= 0.710

erling_haaland

LL | PSNR= 36.155 | SSIM= 0.880

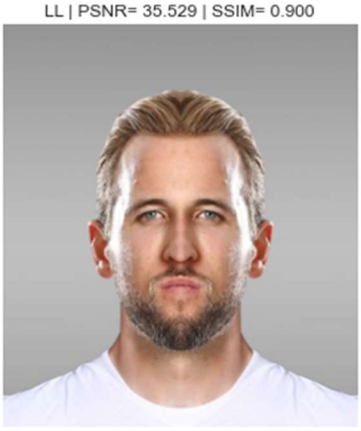


RR | PSNR= 36.155 | SSIM= 0.880

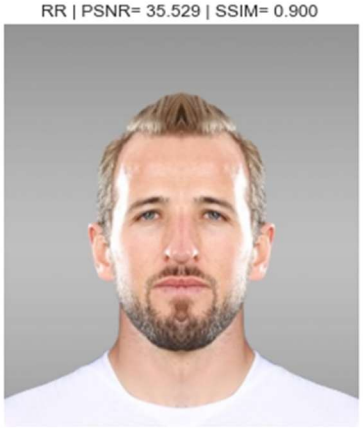
RL | PSNR= 33.145 | SSIM= 0.750

harry_kane

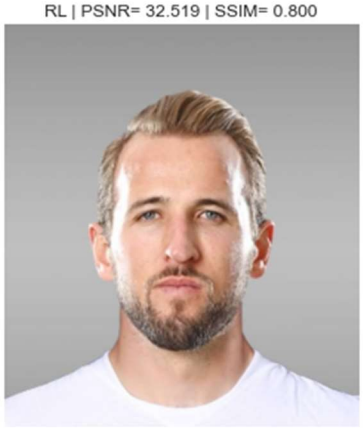
LL | PSNR= 35.529 | SSIM= 0.900



RR | PSNR= 35.529 | SSIM= 0.900

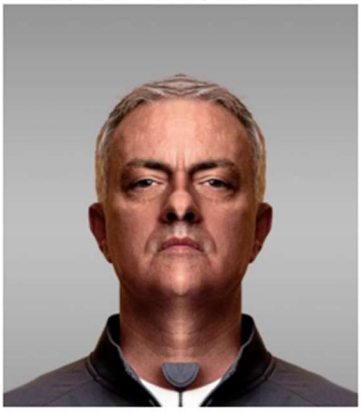


RL | PSNR= 32.519 | SSIM= 0.800

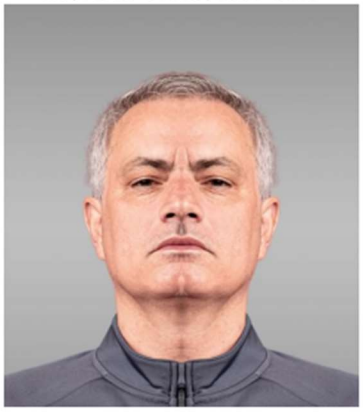


jose_mourinho

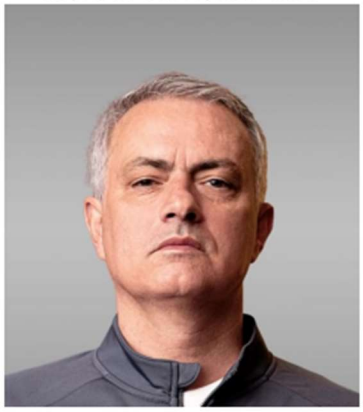
LL | PSNR= 34.552 | SSIM= 0.830



RR | PSNR= 34.552 | SSIM= 0.830



RL | PSNR= 31.541 | SSIM= 0.670



kevin_de_bruyne

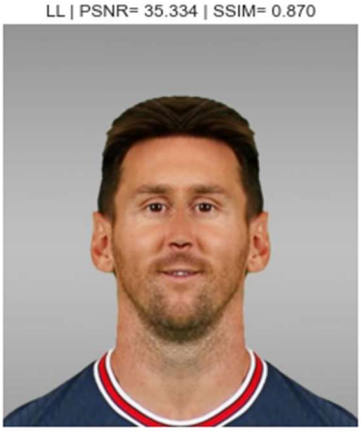
LL | PSNR= 34.959 | SSIM= 0.880

RR | PSNR= 34.959 | SSIM= 0.880

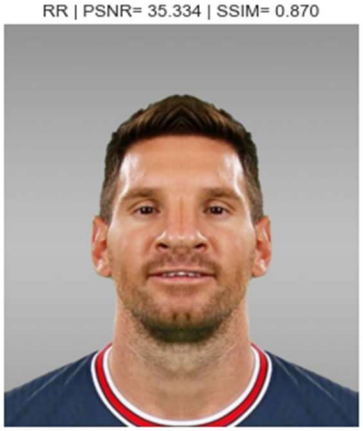
RL | PSNR= 31.949 | SSIM= 0.750

lionel_messi

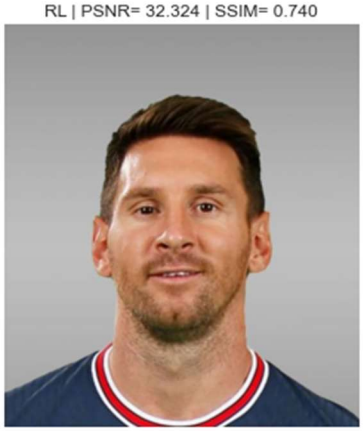
LL | PSNR= 35.334 | SSIM= 0.870



RR | PSNR= 35.334 | SSIM= 0.870

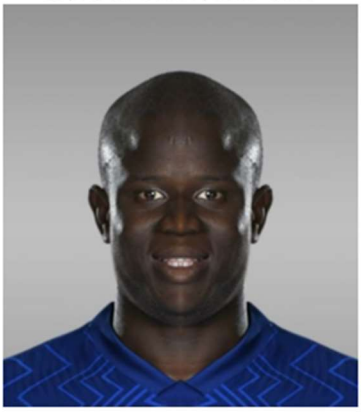


RL | PSNR= 32.324 | SSIM= 0.740



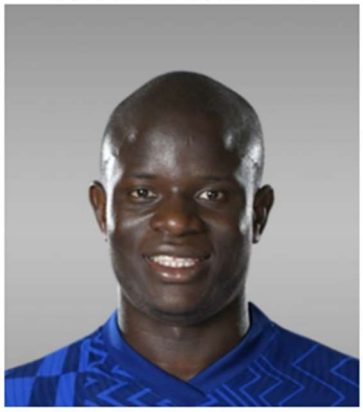
ngolo_kante

LL | PSNR= 34.554 | SSIM= 0.880



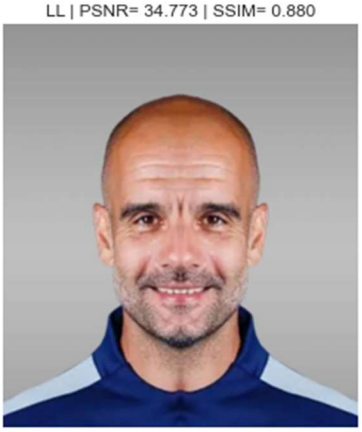
RR | PSNR= 34.554 | SSIM= 0.880

RL | PSNR= 31.543 | SSIM= 0.750

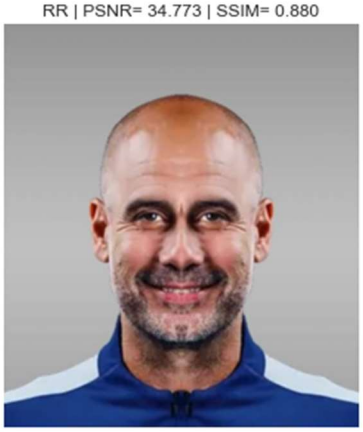


pep_guardiola

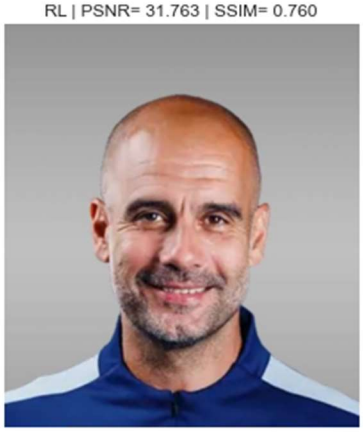
LL | PSNR= 34.773 | SSIM= 0.880



RR | PSNR= 34.773 | SSIM= 0.880



RL | PSNR= 31.763 | SSIM= 0.760



jurgen_klopp

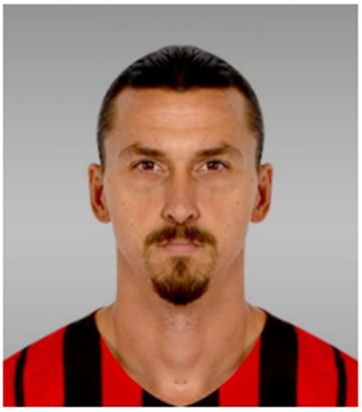
LL | PSNR= 35.009 | SSIM= 0.850

RR | PSNR= 35.009 | SSIM= 0.850

RL | PSNR= 31.998 | SSIM= 0.700

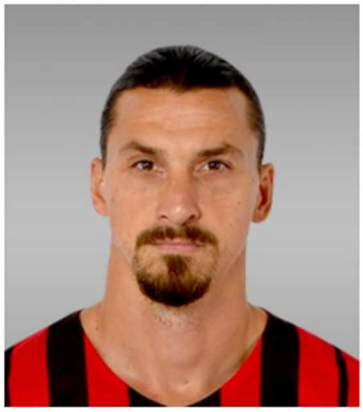
zlatan_ibrahimovic

LL | PSNR= 34.723 | SSIM= 0.880



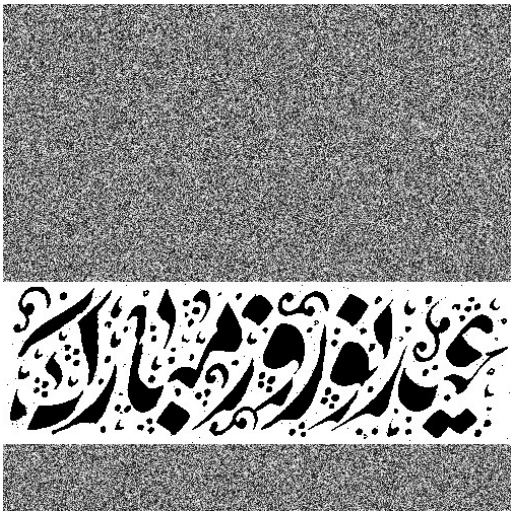
RR | PSNR= 34.723 | SSIM= 0.880

RL | PSNR= 31.713 | SSIM= 0.770

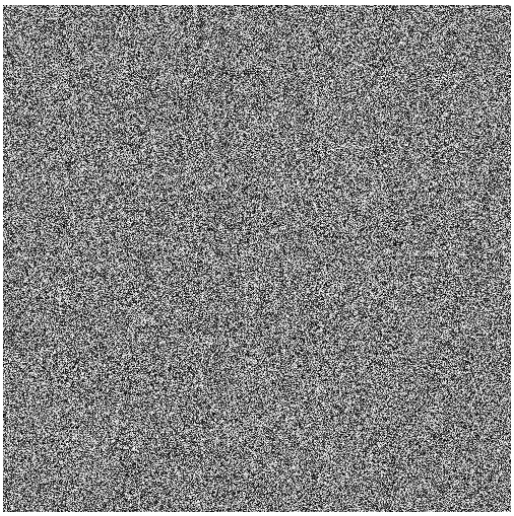


سوال ۲)

الف) طبق توضحات صورت سوال تصویر دوم را بر روی تصویر اول در جهت عمودی میلغزانیم. در هر مرحله دو تصویر را از هم کم میکنیم و نتیجه را ذخیره میکنیم. به دلیل اینکه تصاویر ۵۰۰ پیکسل ارتفاع داشتند، در کل ۱۰۰۰ حالت پیش می آید(500- تا 500+). در نهایت در تصویر ۵۵۲ ام با پیغام مخفی "عید نوروز مبارک" روبرو میشویم.



سایرتصاویرصرفا نویز هستند.



ب) برای این سوال ابتدا تصویر پیغام را به اندازه تصویر اصلی در می‌آوریم، سپس آن را دودویی میکنیم تا بتوانیم با دو مقدار ۰و۲۵۵ آن را در تصویر اصلی کدگذاری کنیم.

سپس به بررسی الگوریتم Decode میپردازیم و متوجه میشویم که نحوه کار این الگوریتم به این صورت است که ابتدا کانال قرمز و آبی تصویر حاوی پیغام مخفی را با هم جمع میکند قدرمطلق میگیرد. آنگاه با توجه به زوج یا فرد بودن هر مقدار در این تصویر جدید، تصمیم میگیرد که پیغام مخفی در پیکسل متناظر با هر مقدار ۲۵۵ است یا ۰.

بنابراین برای طراحی الگوریتم Encode باید طوری تصویر اصلی را تغییر دهیم که پس از انجام شدن مراحل بالا پیکسل متناظر با هر مقدار در تصویرخروجی، همان مقداری داشته باشد که تصویر پیغام دارد. با این تفاسیر کاری که در نهایت انجام شد این بود که ابتدا کانال آبی و قرمز در تصویر اصلی با هم جمع شدند و قدرمطلق آن حساب شد(image_mid). سپس به ازای هر پیکسل در این تصویر، پیکسل متناظر در تصویر پیغام بررسی شد، اگر آن پیکسل سیاه بود باید پیکسل متناظر در تصویر image_mid فرد میبود و اگر سفید بود زوج. برای تعدادی از پیکسل ها به طور اتفاقی این رابطه برقرار بود اما برای پیکسل هایی که این رابطه برایشان برقرار نبود یک واحد به مقدار کانال آبی تصویر اضافه یا کم کردیم(اعداد زوج و فرد با تغییر یک واحد به یکدیگر تبدیل میشوند). در نهایت تصویر پیغام در تصویر اصلی جاگذاری شد.

در تصویر زیر میتوانید تصویر اصلی پس از مرحله Encoding را مشاهده کنید که طبق خواسته سوال تغییر ظاهری ناچیزی دارد.

در نهایت پس از Decode کردن تصویر بالا با الگوریتم ذکر شده، پیغام پنهان شده نمایان شد.

Encode Algorithm

```python
img_mid = abs(img[:,:,0] - img[:,:,2])

for i in range(0, msg.shape[0]):
    for j in range(0, msg.shape[1]):

        # Encode white pixels
        if msg_thresh[i,j] == 255:
            if img_mid[i,j] % 2 == 1: # odd
                if img[i,j, 0] == 255:
                    img[i,j, 0] -= 1
                else:
                    img[i,j, 0] += 1

        # Encode black pixels
        if msg_thresh[i,j] == 0:
            if img_mid[i,j] % 2 == 0: # even
                if img[i,j, 0] == 0:
                    img[i,j, 0] += 1
                else:
                    img[i,j, 0] -= 1
```

[10] ✓ 19.1s

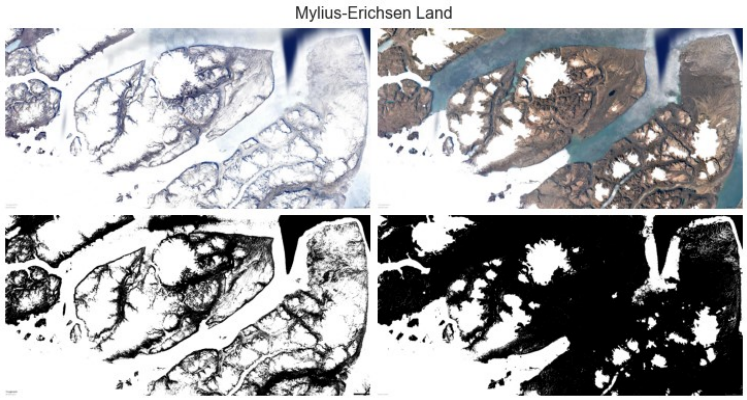
Decode Algorithm

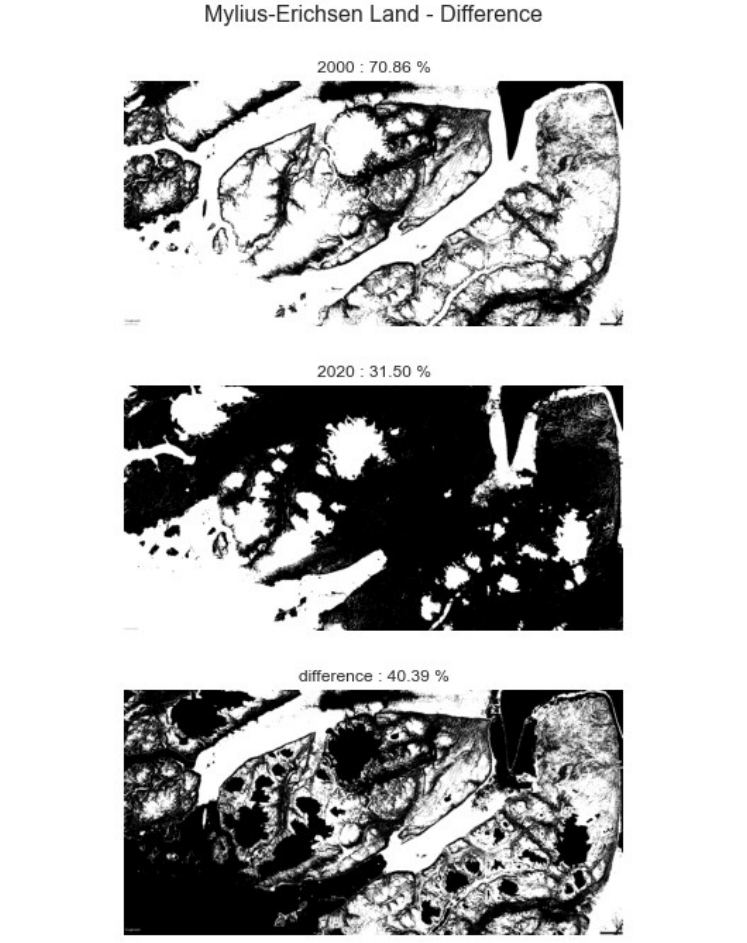
```python
img_mid = abs(img[:,:,0] - img[:,:,2])

# Decoding Result
result = np.zeros_like(img)

for i in range(0, result.shape[0]):
    for j in range(0, result.shape[1]):
        if img_mid[i,j] % 2 == 0:
            # if even --> white
            result[i,j] = 255.0
        else:
            # if odd --> black
            result[i,j] = 0.0
```

[13] ✓ 27.4s

سوال ۳)

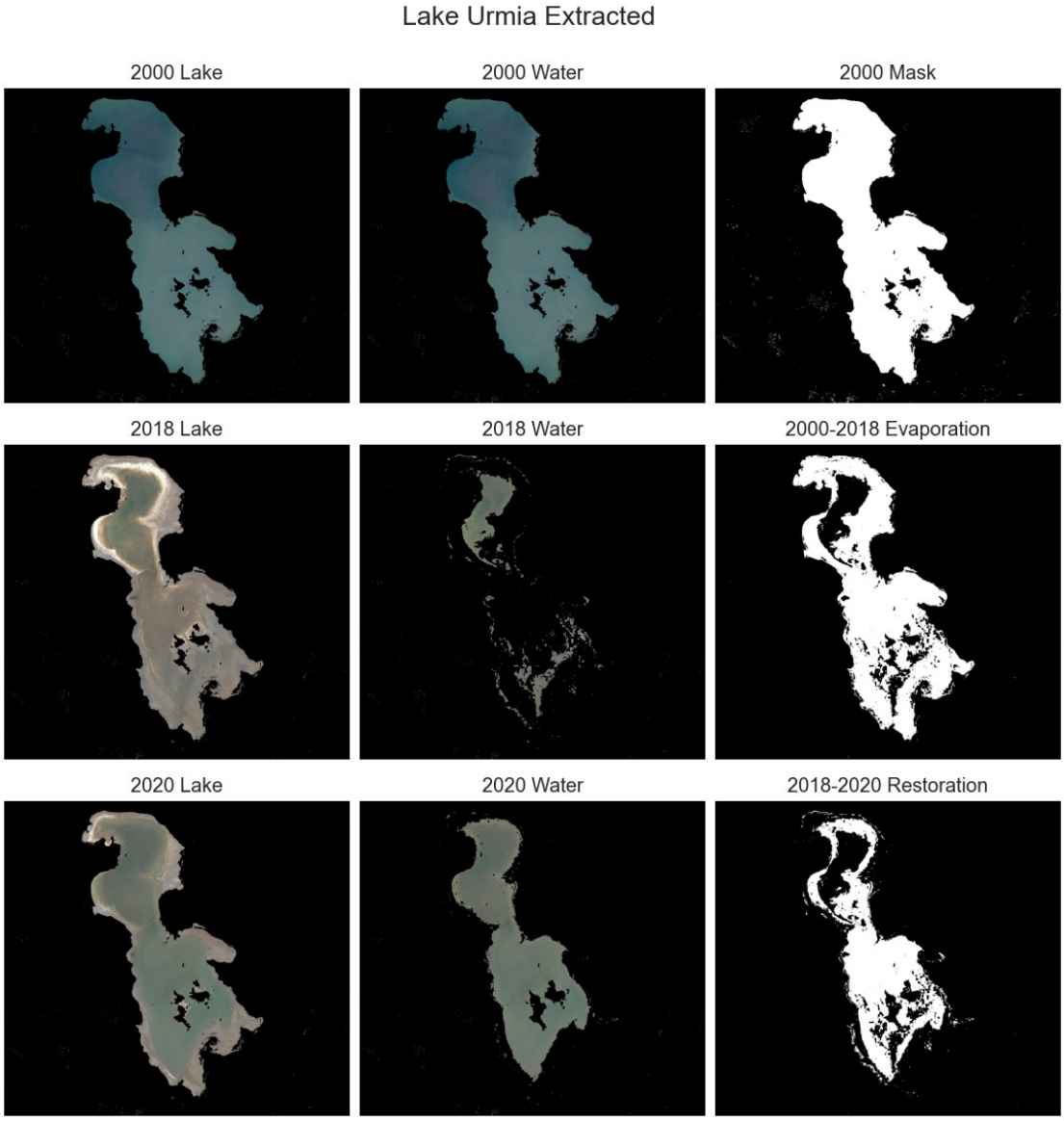
الف) در این بخش از سوال برای بدست آوردن درصد یخ آب شده بین سالهای ۲۰۰۰ تا ۲۰۲۰ ابتدا پیکسل های یخ تصویر اول و دوم را تشخیص میدهیم سپس از هم کم میکنیم (این تفاصل، پیکسل های مرتبط با یخِ آب شده است). برای بدست آوردن پیکسل های یخ/برف در این تصاویر با استفاده از Threshold هایی برای Brightness , Saturation پیکسل ها، پیکسل های متناظر با یخ را جدا کردیم. برای محاسبه درصد یخ آب شده نیز تعداد پیکسل های تفاضل(آب شده) را بر تعداد پیکسل های یخ تصویر مربوط به سال ۲۰۰۰ تقسیم میکنیم.





همانطور که در تصاویر میبینید حدود ۷۱ درصد تصویر در سال ۲۰۰۰ با یخ پوشیده بود. اما این عدد در سال ۲۰۲۰ به ۳۱ درصد رسید. در نهایت لازم به ذکر است که میزان **۵۷ درصد** یخ در طی این ۲۰ سال ذوب شده است.

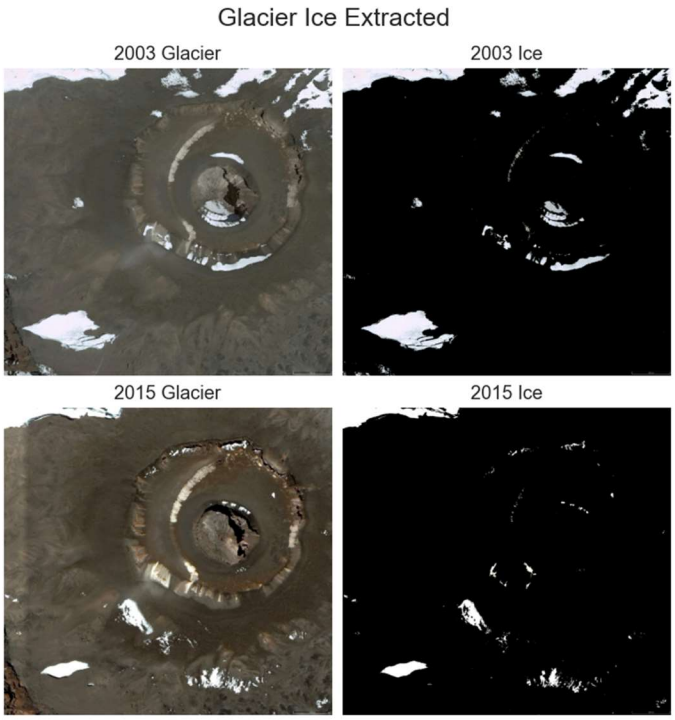
ب) برای تعیین میزان آب موجود در دریاچه ارومیه از روش دیگری استفاده کردیم. ابتدا تصاویر مربوط به این دریاچه را به فضای HSV بردیم و سپس پیکسل هایی که در یک بازه مخصوص(مختص رنگ آب دریاچه) قرار میگرفتند را جدا کردیم. همچنین تصویر سال ۲۰۱۸ به دلیل خشکسالی زیاد کمی چالشی بود. یکی از چالش های این تصویر شبیه بودن رنگ آب دریاچه و درخت های اطراف بود. به همین دلیل از ناحیه مربوط به آب دریاچه در سال ۲۰۰۰ به عنوان **ماسک** استفاده کردیم و ناحیه مربوط به دریاچه ارومیه را از پس زمینه جدا کردیم. این کار جداسازی آب از قسمت خشک دریاچه را بسیار راحت تر کرد. در نهایت با انجام عمل تفریق میزان آب خشک شده بین سالهای ۲۰۰۰ تا ۲۰۱۸ و هم چنین میزان آب بازیابی شده بین سال های ۲۰۱۸ و ۲۰۲۰ را محاسبه کردیم.



نتایج موردنیاز برای بخش ب و ج سوال را در تصویر زیر مشاهده میکنید.

```
Urmia 2000 Lake Volume: 13.565 km^3
Urmia 2018 Lake Volume: 0.525 km^3
Urmia 2020 Lake Volume: 2.866 km^3
B) Volume of water evaporated between 2000 & 2018: -13.040 km^3
C) Percentage of water restored between 2018 & 2020: 17.257 %
```

د) این بخش شباهت بسیار زیادی به تصویر اول داشت و از همان تکنیک ها برای جداسازی یخ/برف استفاده کردیم.

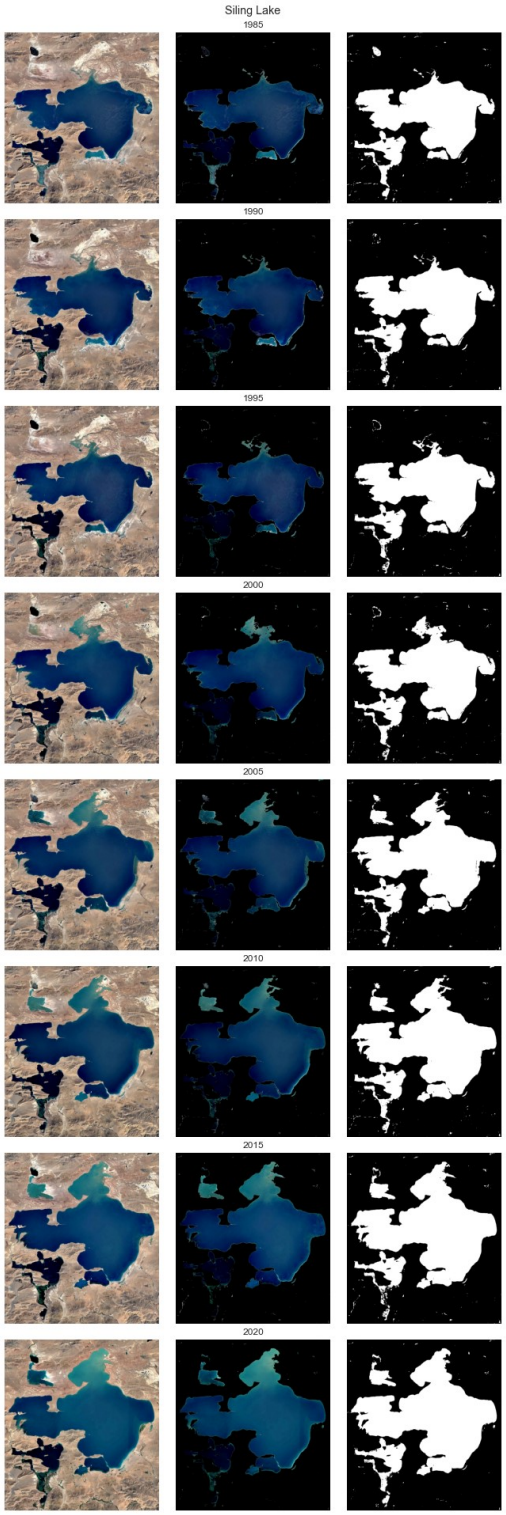


همچنین میزان یخ موجود در سال ۲۰۰۳ و ۲۰۱۷ و میزان یخ ذوب شده بین این دو سال را در تصویر زیر مشاهده میکنید.

```
Glacier 2003 Ice Volume: 1125715.026 m^3
Glacier 2015 Ice Volume: 316716.924 m^3
D) Volume of Ice melted between 2003 & 2015: 808998.103 m^3
```

نکته: در این بخش از سوال به اشتباه به جای سال ۲۰۱۷ عدد ۲۰۱۵ در تصاویر وارد شده است که صرفا اشتباه تایپی می باشد.

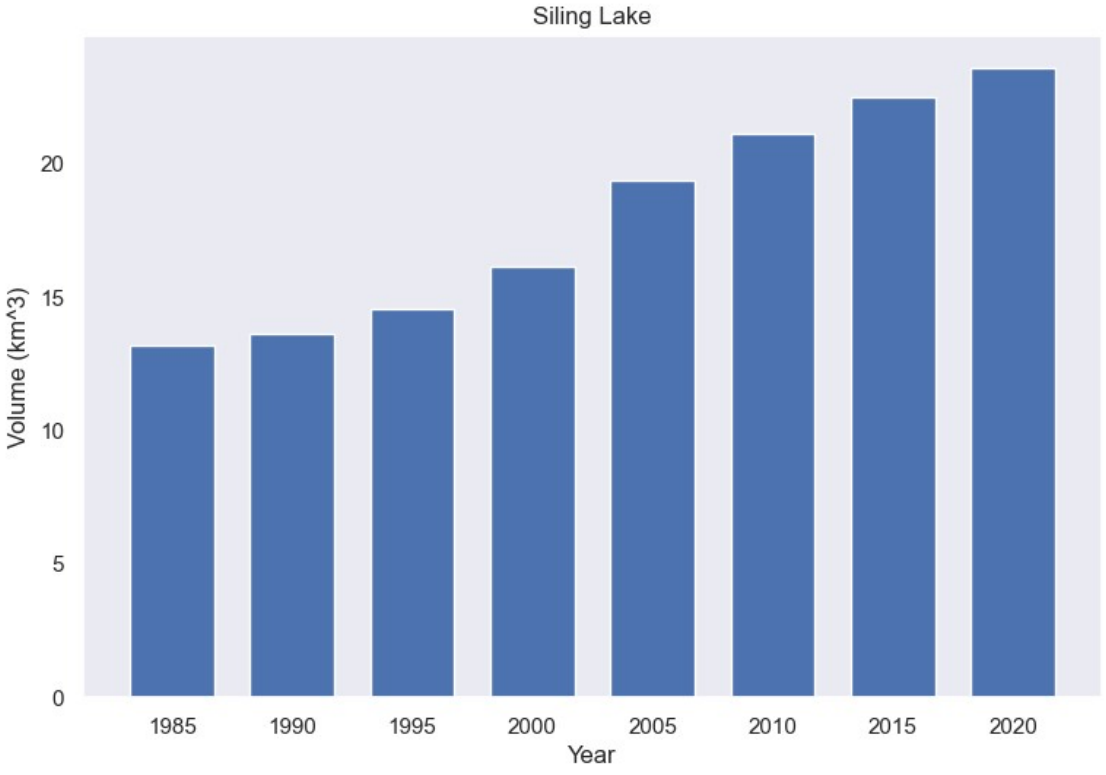
هـ) در این بخش مانند بخش دریاچه ارومیه ابتدا تصویر را به فضای HSV بردیم و با مشخص کردن بازه ای، به جداسازی پیکسل های متناظر با آب پرداختیم. نتیجه جداسازی آب در هر تصویر را میتوانید در شکل زیر مشاهده کنید.



سپس به محاسبه میزان آب موجود در دریاچه در سال های مختلف پرداختیم.

```
Siling Lake 1985 Volume: 13.199 km^3
Siling Lake 1990 Volume: 13.629 km^3
Siling Lake 1995 Volume: 14.533 km^3
Siling Lake 2000 Volume: 16.121 km^3
Siling Lake 2005 Volume: 19.348 km^3
Siling Lake 2010 Volume: 21.135 km^3
Siling Lake 2015 Volume: 22.514 km^3
Siling Lake 2020 Volume: 23.590 km^3
```

نمودار ستونی مربوط به میزان آب موجود در دریاچه، در سالهای مختلف را میتوانید در شکل زیر مشاهده کنید. میزان آب این دریاچه در ۳۵ سال گذشته افزایش حدود ۷۸ درصدی داشته است.



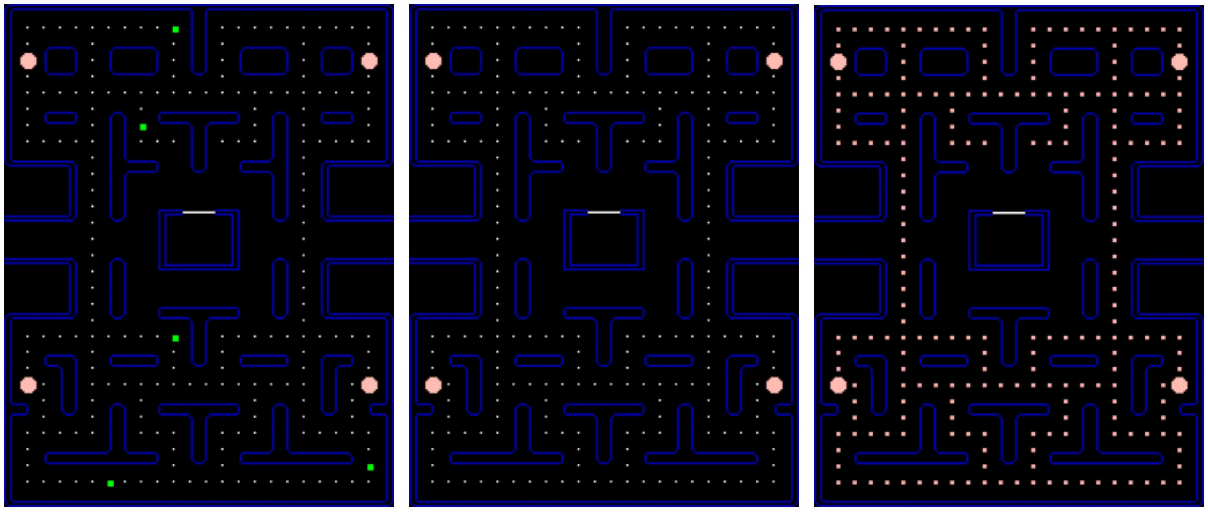
سوال ۴)

این سوال زمان‌برترین سوال این تمرین بود. ابتدا نیاز بود بررسی هایی انجام بدهیم تا با مسئله و جزییات آن بیشتر آشنا شویم. طی بررسی های انجام شده متوجه شدیم که فاصله بین هر دو دایره صورتی رنگ ۸ پیکسل بود نه ۱۲ پیکسل(طبق ادعای سوال). به همین دلیل تصمیم بر آن شد قدم های کاراکترها به جای ۳ پیکسل به اندازه ۲ پیکسل باشند که در هر ۴ قدم، پکمن به یک دایره جدید برسد.

در ابتدای حل این سوال تصاویر را لود کردیم و یک تابع کمکی برای فراخوانی تصاویر مختلف با استفاده از اسم ایجاد کردیم.

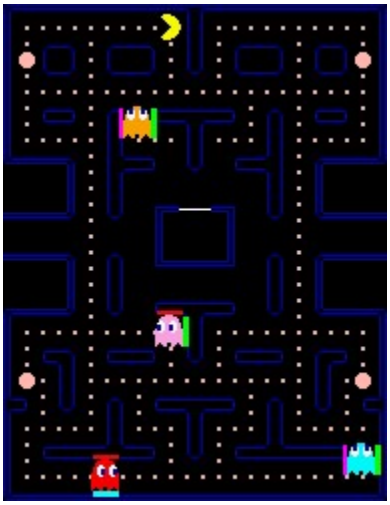
```python
def get_image(name):
    for i, n in enumerate(names):
        if n == name:
            return images[i]
    return None
```

سپس تصویر maze را فراخوانی کردیم و با حرکت بر روی پیکسل های تصویر و مقایسه رنگ پیکسل ها با رنگ صورتی، موقعیت دایره های صورتی را استخراج کردیم (پیکسل های سفید). در نهایت به صورت رندوم ۵ عدد از این نقاط را به عنوان موقعیت اولیه کاراکترهای بازی انتخاب کردیم (نقاط سبز رنگ).

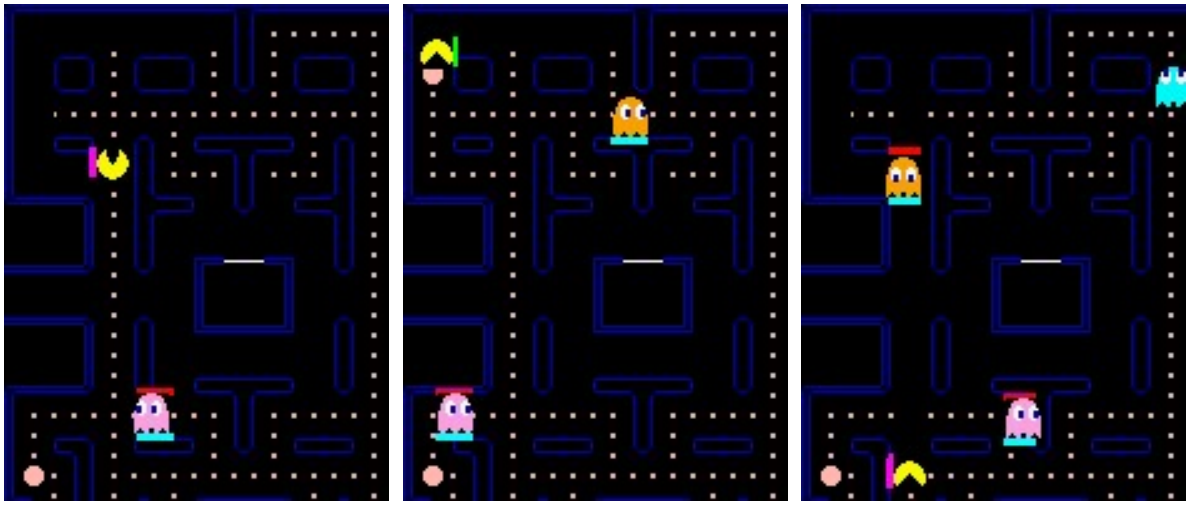


در مرحله بعد، دو کلاس برای کاراکترهای شبح و پکمن ایجاد کردیم. این دو کلاس وظیفه نگهداری و بروزرسانی ویژگی هایی مانند موقعیت، حالت، جهت و ... را بر عهده داشتند. همچنین این کلاس ها با گرفتن نقشه بازی به عنوان ورودی و با کمک تابع Render خود، کاراکترها را بر روی نقشه رسم میکردند.

برای حذف کردن دایره های صورتی به محض خورده شدن توسط پکمن، قبل از حرکت دادن پکمن یک ناحیه مشکی رنگ به اندازه آن را بر روش نقشه رسم میکردیم. این کار باعث میشد دایره های صورتی رنگی که توسط پکمن خورده میشدند در فریم های بعدی حذف شده باشند. لازم به ذکر است که نقشه بازی در هر فریم به شکلی که ذکر شد بروزرسانی میشد و به فریم بعد پاس داده میشد تا حالت فریم های قبل به فریم های بعد انتقال پیدا کند.



از دیگر کارهایی که نیاز بود انجام شود، تشخیص تصادم یا Collision Detection بود. همانطور که در تصویر بالا و همچنین تصاویر دیگر میبینید، ناحیه هایی به پهنای ۳ پیکسل، دورتادور کاراکترها در نظر گرفته شد و اگر در این نواحی پیکسلی با رنگ آبی(دیوار) موجود بود یک برخورد در آن جهت خاص تشخیص داده میشد و یک سری اتفاق‌ها می‌افتاد. از جمله ای‌ن اتفاق ها رسم کردن یک ناحیه مستطیلی رنگی در محل برخورد و حذف کردن جهت برخورد از جهت های مجاز برای حرکت بعدی کاراکتر بود.



```
Main

maze_img = maze_img_org.copy()
new_maze_img, pacman, ghosts = init_game(maze_img)
[374]

for i in range(1,500):
    new_maze_img, pacman, ghosts,isalive = gen_demo(new_maze_img, pacman, ghosts, i)
    if not isalive:
        print("Game Over", i)
        break
[377]

</>  Game Over 202
```

در ابتدای هر فریم، هر کاراکتر با توجه به جهت های مجازی که برایش باقی مانده است، جهتی را به صورت تصادفی برمیگزیند و به اندازه ۲ پیکسل در آن جهت حرکت میکند. همچنین با توجه به اطلاعات موجود در طرح سوال، تصویر مربوط به کاراکتر ها تغییر میکند (برای القای حس راه رفتن کاراکترها). اگر این کاراکتر خود پکمن باشد، یک سری اتفاقات دیگر نیز رخ میدهد. به عنوان مثال توپ صورتی زیر کاراکتر حذف میشود و همچنین برخورد با تک تک شبح ها بررسی میشود و در صورت برخورد با شبح ها بازی به اتمام میرسد.

در فایل های ارسالی تمرین یک **ویدیو** از اجرای بازی موجود است. در صفحه های بعد نیز تعدادی از فریم های بازی ارائه خواهد شد.

C) ویدیوی اجرای بازی در کنار فایل ارسالی قرار دارد.

سوال۵)

الف) یکی از کارهایی که چشم انسان انجام میدهد Brightness Adaptation است که به هنگامی که از یک محیط تاریک به یک محیط روشن میرویم رفته رفته تصویری که میبینیم روشن میشود تا به چشم آسیب نرسد.

ب) سنسور موبایل S22 از نوع tetracell است به این معنی که هر ۴ پیکسل مجاور را با هم ترکیب میکند و یک پیکسل تولید میکند. در نهایت تصاویر ذخیر شده با دوربین S22 دارای رزولوشن 12MP خواهد بود. بنابراین رزولوشن تصاویر گرفته شده با دوربین های این دو دستگاه یکسان خواهد بود.

رزولوشن تصاویر گرفته شده با آیفون ۱۳ برابر با 4032×4032 است و رزولوشن تصاویر گرفته شده با S22 برابر با 300×4000 است.

کیفیت تصاویر یک معیار کمی است و به همین دلیل قابل مقایسه نمی‌باشد. اما با این حال تصاویر گرفته شده با دوربین S22 مقداری پررنگ تر و مصنوعی تر به نظر میرسند. این تفاوت میتواند به دلیل انجام پس پردازش هایی روی تصاویر گرفته شده توسط این دستگاه باشد.

به دلیل وجود الگوریتم های فشرده سازی و عملکرد Adaptive آنها، حجم تصاویر گرفته شده با این دو دوربین ثابت نیست و با توجه به پیچیدگی صحنه و میزان نویز و جزییات موجود، متفاوت است. اما به طور کلی حجم تصاویر گرفته شده با هر دو دستگاه حدود ۱۰ مگابایت است.

ج) برای تبدیل تصاویر RGB به Grayscale میتوان از رابطه زیر استفاده کرد.

$$Grayscale = \alpha R + \beta G + \gamma B$$

اما همانطور که میدانیم بدون دانستن مقادیر B,G نمیتوان صرفا با داشتن مقدار Grayscale مقدار R را بازیابی کرد.

به شکل مشابه بدون دانستن مقادیرR,B نمیتوان صرفا با داشتن مقدار Grayscale مقدار G را بازیابی کرد.

و همچنین بدون دانستن مقادیر R,G نمیتوان صرفا با داشتن مقدار Grayscale مقدار B را بازیابی کرد.

د) Resampling: وقتی یک تصویر را Scale یا Rotate یا ... میکنیم، نیاز است با اضافه و کم کردن پیکسل هایی، تصویر حاصل را تولید کنیم.

Sub-sampling: یک روش برای کاهش اندازه و حجم تصویر است که با انتخاب هر n اُمین پیکسل باعث می‌شود اندازه نهایی تصویر کاهش بیابد.

هـ) بله فضاهای رنگی مختلف در محاسبه میزان شباهت تصاویر تاثیر دارند. فضای رنگی L*a*b و YUV به دلیل اینکه بر اساس Perception انسان ساخته شده اند برای این منظور مناسب‌تر اند. در این فضاها، تغییر اندکی در مقدار عددی رنگ، تاثیر اندکی در رنگ Perceive شده دارد. بنابراین محاسبه فاصله اقلیدسی بین دو رنگ در این فضاها تقریبا برابر با تفاوت Perceive شده این دو رنگ است.

فضای رنگی RGB, CMYK این ویژگی را ندارند و دو رنگ با فاصله اقلیدسی بسیار کم میتوانند از لحاظ بصری بسیار متفاوت باشند. تصاویر سطح خاکستری نسبت به تصاویر L*a*b و YUV میزان داده کمتری دارند و به همین دلیل نتیجه ضعیف تری خواهند داد. فضاهای HSV و HSL نیز در وسط قرار دارند و میتوان با استفاده از تکنیک هایی از این فضا برای تعیین میزان شباهت تصاویر استفاده های خوبی کرد.