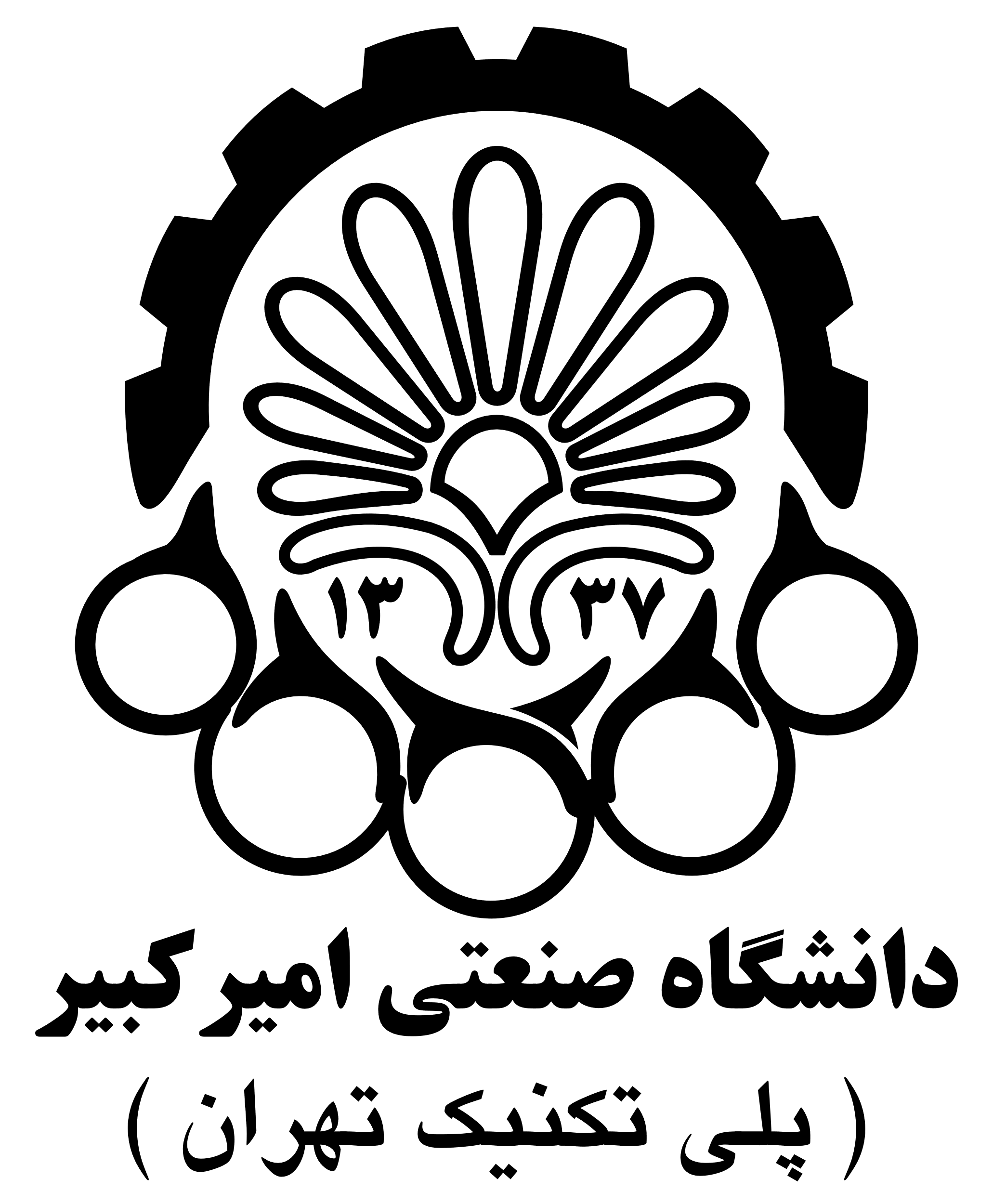
****

**دانشکده مهندسی کامپیوتر**

**تمر ین سوم درس بینایی ماشین**

**دکتر صفابخش**

**غلامرضا دار 400131018**

**بهار 1401**

فهرست مطالب

[سوال 1) 3](#_Toc119056295)

[سوال 2) 5](#_Toc119056296)

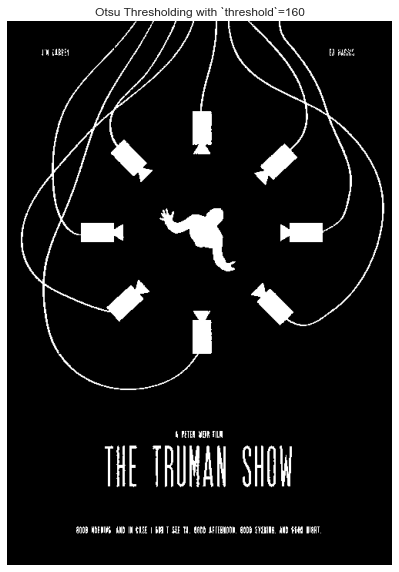
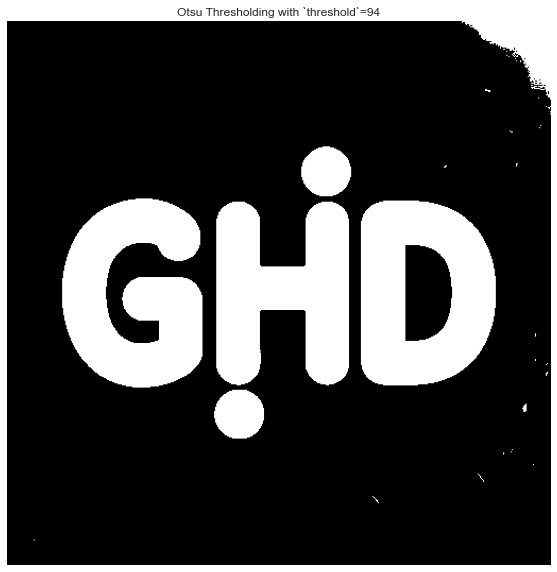
[سوال 3) 7](#_Toc119056297)

[سوال 4) 11](#_Toc119056298)

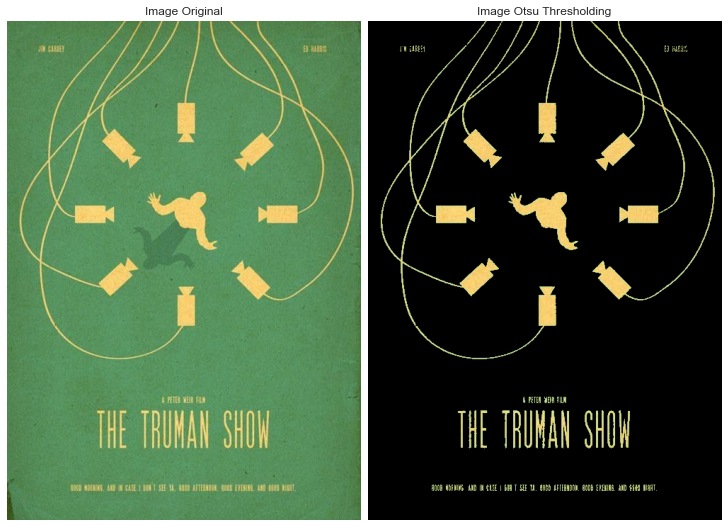
# سوال 1)

برای پیاده سازی الگوریتم Otsu از توضیحات و نمونه کد­های داکیومنتیشن OpenCV استفاده کردیم. *<https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html>* .

برای اطمینان از صحت پیاده سازی، پیاده سازی خود را با نتیجه تابع آماده موجود در کتابخانه OpenCV مقایسه کردیم که نتیجه بسیار نزدیک بود. هم تابع پیاده سازی شده و هم تابع آماده موجود، مقدار 160 را برای آستانه تصویر اول خروجی دادند.

*نتیجه آستانه گیری به کمک الگوریتم Otsu*

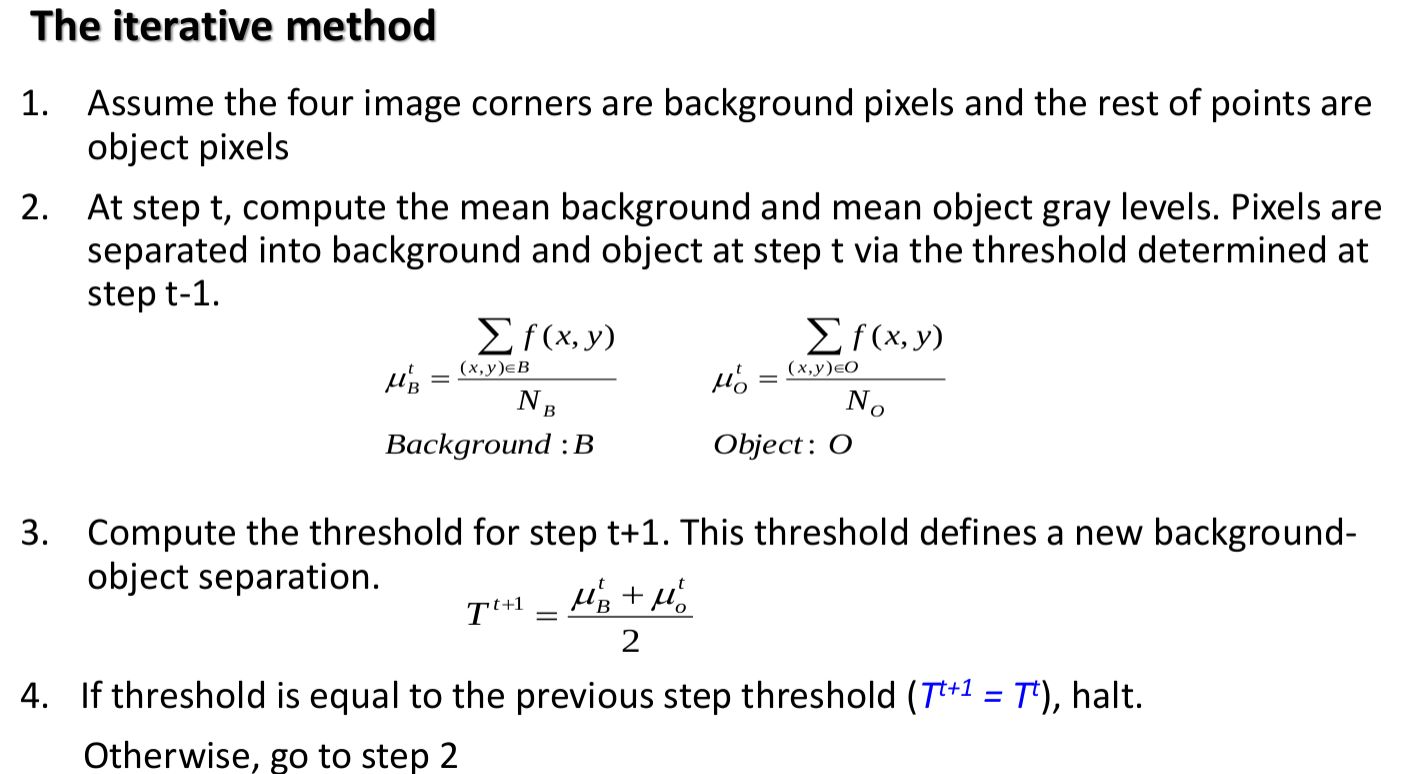




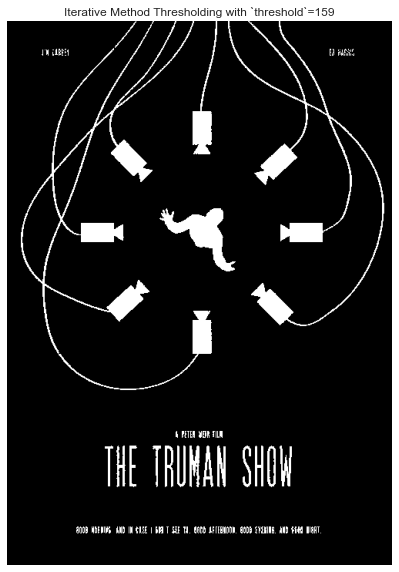
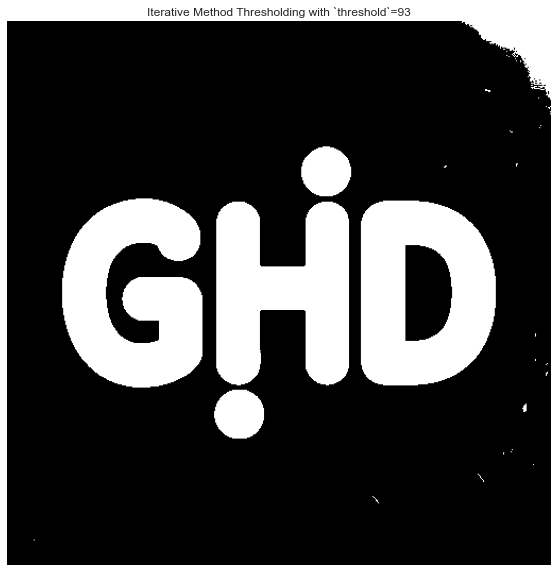
*اعمال نتایج صفحه قبل به عنوان ماسک به تصویر اصلی مطابق با خواسته سوال*

# سوال 2)

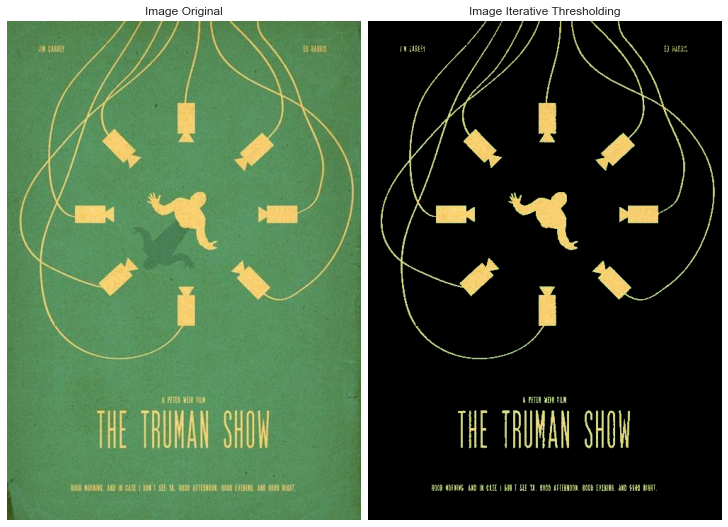
برای پیاده سازی الگوریتم Iterative از توضیحات اسلاید درس استفاده کردیم.

.

*الگوریتم Iterative Thresholding*

*نتیجه آستانه گیری به کمک الگوریتم Iterative*





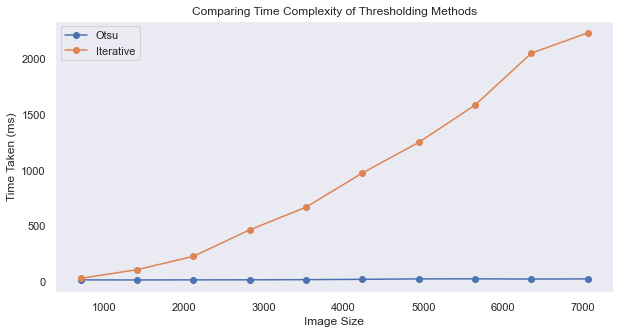
*اعمال نتایج صفحه قبل به عنوان ماسک به تصویر اصلی مطابق با خواسته سوال*

# سوال 3)

همان­طور که در دو بخش قبل دیدیم، این دو الگوریتم بر روی تصاویر انتخاب شده، نتایج یکسانی داشتند. با این وجود، این دو الگوریتم تفاوت­های اساسی در روش عملکرد و سرعت اجرا دارند.

الگوریتم Otsu به طور مستقیم بر روی هیستوگرام تصویر کار می­کند و با آزمایش کردن مقادیر مختلف برای آستانه از 0 تا 255، آستانه ای که باعث کمترین within-class variance شود را انتخاب می­کند. این الگوریتم تنها در بخش محاسبه هیستوگرام به پیکسل­های تصویر نیاز دارد و پس از آن تمام محاسبات بر روی هیستوگرام تصویر اعمال می­شوند. در نتیجه پیش­بینی می­کنیم این الگوریتم بر روی تصاویر با اندازه بسیار بزرگ نیز مشکلی نداشته باشد.

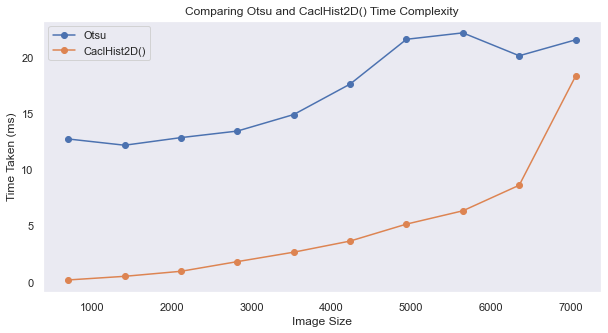
الگوریتم Iterative، در هر مرحله نیاز دارد تمام پیکسل­های تصویر را به دو دسته Background و Object تقسیم کند و بر روی تمام این پیکسل­ها(تمام پیکسل­های تصویر) میانگین گیری انجام دهد. این امر باعث می­شود با افزایش اندازه تصویر، این الگوریتم بسیار ناکارامد شود. با این وجود، برای تصاویر با اندازه معقول این دو الگوریتم تقریبا سرعت یکسانی دارند.



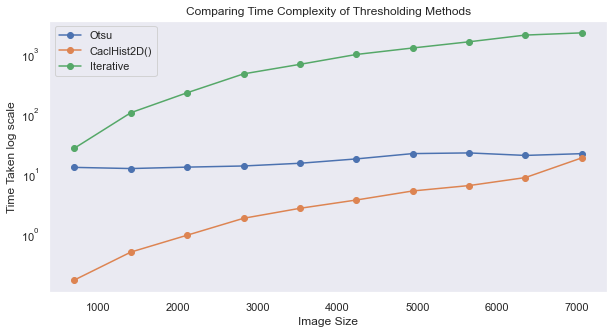
*نمودار زمان اجرا(میلی­ثانیه) بر حسب عرض تصویر(پیکسل)*

با انجام آزمایش­هایی توانستیم نظریه بالا را اثبات کنیم. به طور واضح مشاهده می­کنید که با افزایش ابعاد تصویر، الگوریتم Iterative بسیار کند می­شود و به چندین ثانیه زمان برای یافتن مقدار آستانه مناسب نیاز دارد در صورتی که الگوریتم Otsu حتی برای تصویری با عرض 7000 پیکسل نیز در حدود 22 میلی ثانیه به جواب بهینه رسیده است.

همچنین، همان­طور که پیش­تر ذکر شد، الگوریتم Otsu به طور کلی بر روی هیستوگرام تصویر کار می­کند که مستقل از ابعاد تصویر است، تنها زمانی که این الگوریتم به پیکسل­های تصویر دسترسی پیدا­می­کند، هنگام تولید هیستوگرام است (1 بار). با انجام آزمایشی دیگر این نظریه را نیز اثبات کردیم. دو نمودار زیر، سرعت رشد یکسانی دارند.

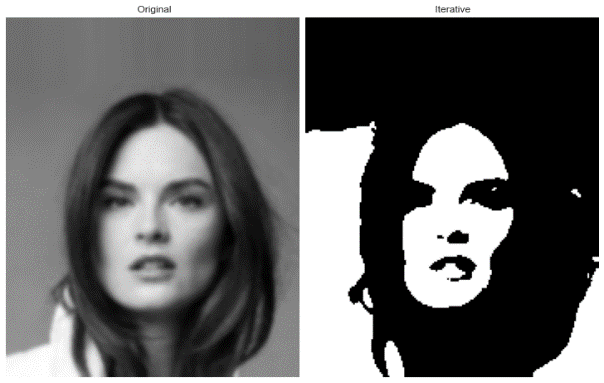


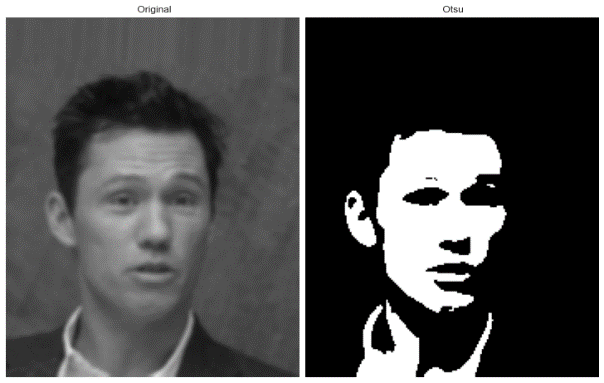
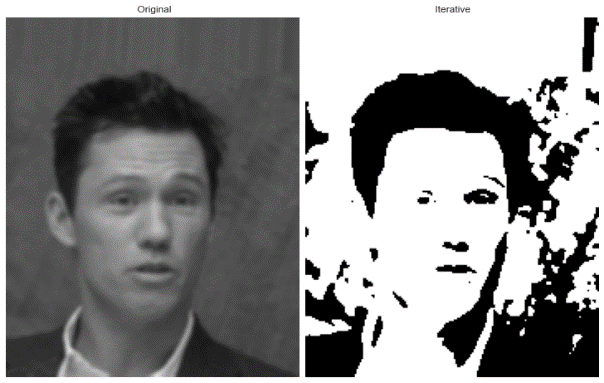
*مقایسه زمان اجرای کل الگوریتم Otsu و بخش محاسبه هیستوگرام*

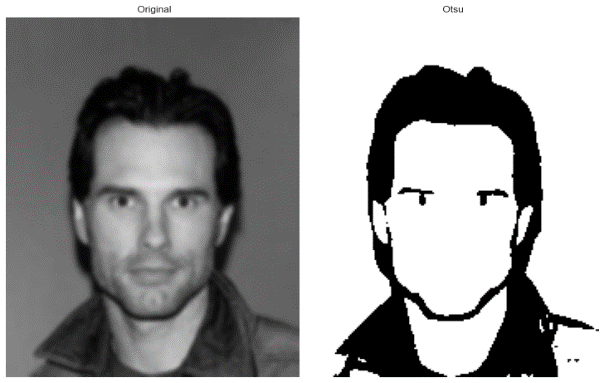
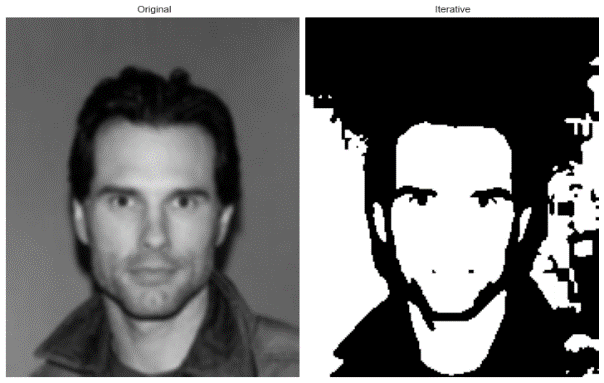


*مقایسه الگوریتم­ها با استفاده از مقیاس لگاریتمی*

در مورد دقت این دو الگوریتم می­توان حدس زد که چون الگوریتم Otsu تمام حالت­های ممکن برای آستانه گیری را آزمایش می­کند(0 تا 255) قطعا بهترین نتیجه را با توجه به معیار خود بدست خواهد آورد. اما الگوریتم Iterative به دلیل خاصیت پرشی که دارد ممکن است تعدادی از حالت­ها را بررسی نکند. با این وجود بدون انجام آزمایش نمی­توان نتیجه درستی گرفت. برای آزمایش این دو روش از تعدادی تصویر (تصاویر مربوط به مجموعه داده CelebA) کمک گرفتیم. این تصاویر را به کمک هر دو روش آستانه زدیم و آن تصاویری که نتیجه آستانه گیری آنها به ازای دو الگوریتم متفاوت بود را گزارش دادیم. برای اکثر تصاویر این دو الگوریتم مقدار آستانه بسیار نزدیک داشتند اما برای آن دسته از تصاویری که این دو الگوریتم متفاوت عمل کردند، الگوریتم Otsu (از نظر بنده) عملکرد بهتری داشت.

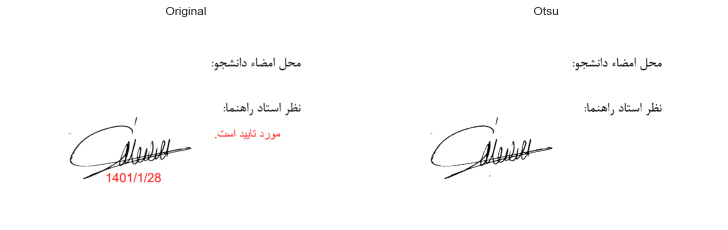




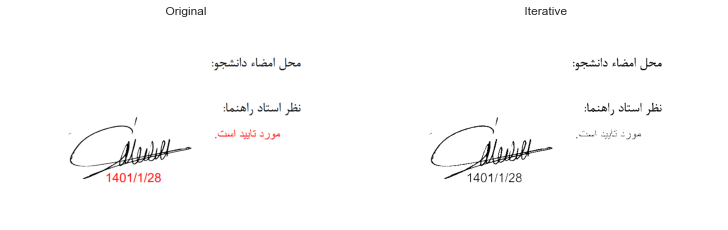


*آستانه­گیری­های سمت راست مربوط به روش Iterative و سمت چپ مربوط به روش Otsu*

با این وجود ممکن است تصاویری باشند که الگوریتم Iterative در آنها نتیجه بهتری دهد. به عنوان مثال یکی از تصاویر رندوم موجود در سیستم بنده:



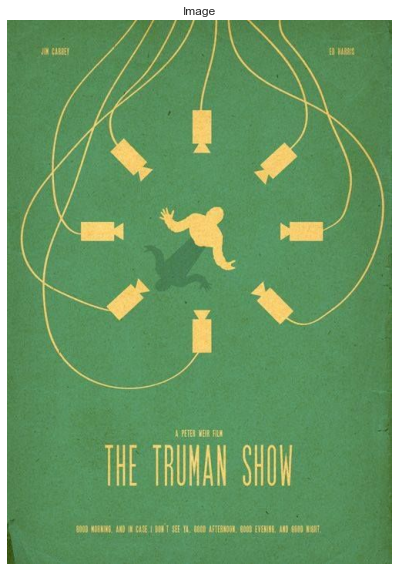
*نتیجه الگوریتم Otsu: متون قرمز از بین رفته اند*



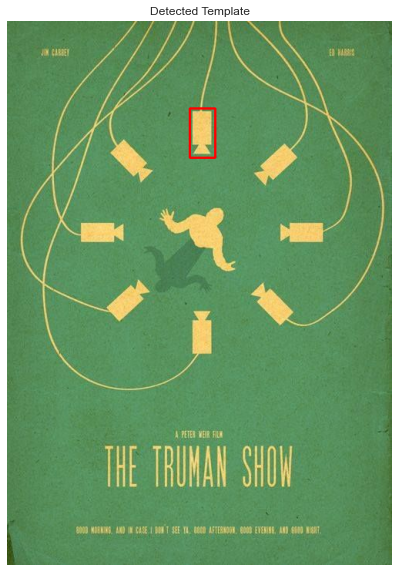
*نتیجه الگوریتم Iterative: متون قرمز باقی مانده اند*

# سوال 4)

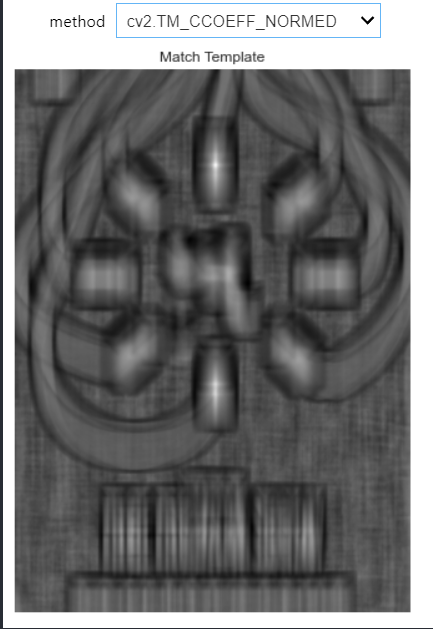
در این سوال قصد داریم به کمک روش تطبیق کلیشه یا Template Matching، دوربین­های موجود بر روی پوستر زیر را تشخیص دهیم. همان­طور که دیده می­شود تعدادی دوربین مشابه اما با میزان چرخش متفاوت در پوستر موجود هستند.



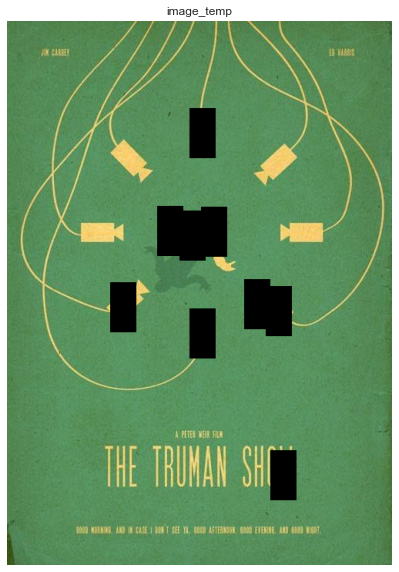
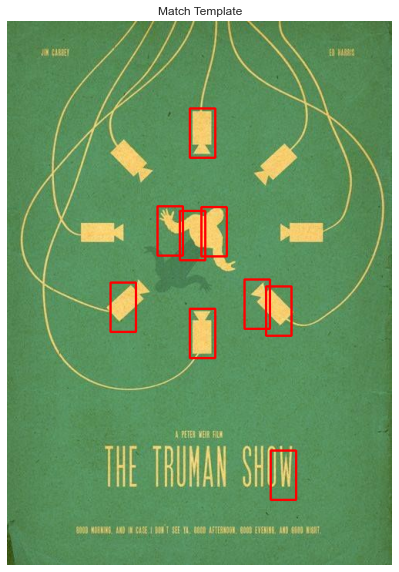
با استفاده از تابع matchTemplate موجود در کتابخانه OpenCV، به نتیجه زیر می­رسیم.



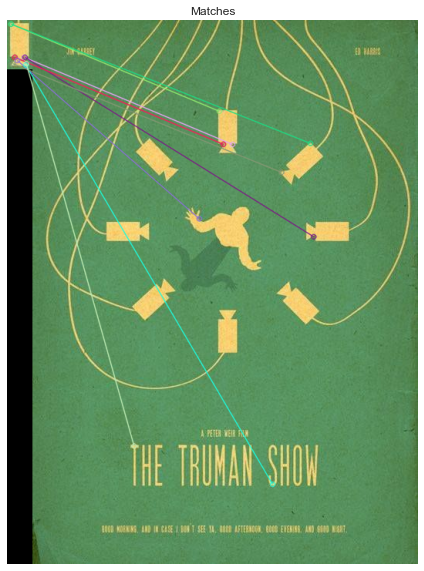
خروجی تابع matchTemplate مشابه تصویر زیر است که با استفاده از ماکسیمم گیری می­توان پراحتمال ترین موقعیت شیء را تشخیص داد.



برای یافتن بیش از یک شیء در تصویر می­توان هر بار که شیء پیدا می­شود، در آن ناحیه از تصویر ورودی، یک مستطیل سیاه رسم کرد. با این کار هر بار، نقطه تشخیص داده شده با بیشترین احتمال(که تا این لحظه یافته نشده است) یافت می­شود.



پس از شکست در مرحله قبل به سراغ روش­های FeatureMatching رفتیم. این روش­ها نیز به دلیل کمبود ویژگی منحصر­به­فرد و واضح در تصویر کلیشه(دوربین) به نتیجه خوبی نرسیدند.

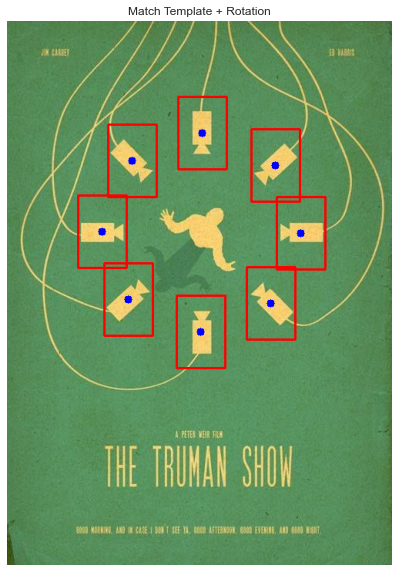


*نتیجه روش Feature Matching*

در نهایت تصمیم گرفتیم با چرخاندن تصویر کلیشه به دفعات، سعی کنیم دوربین­هایی با جهت­گیری های مختلف را شناسایی کنیم.



*نمونه کلیشه چرخانده شده*

**

*نتیجه نهایی تشخیص دوربین ها در تصویر*