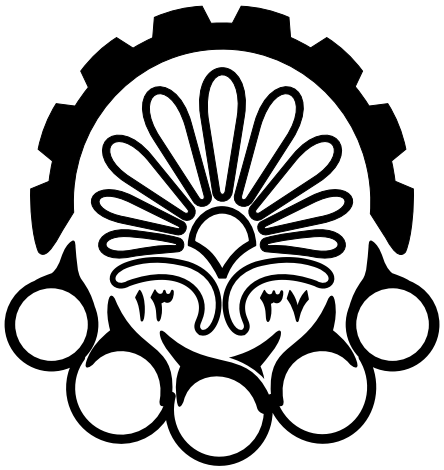


دانشگاه صنعتی امیرکبیر

(پلی تکنیک تهران)

دانشکده مهندسی کامپیوتر

تمر ین سوم درس بینایی ماشین

دکتر صفابخش

غلامرضا دار ۴۰۰۱۳۱۰۱۸

بهار ۱۴۰۱

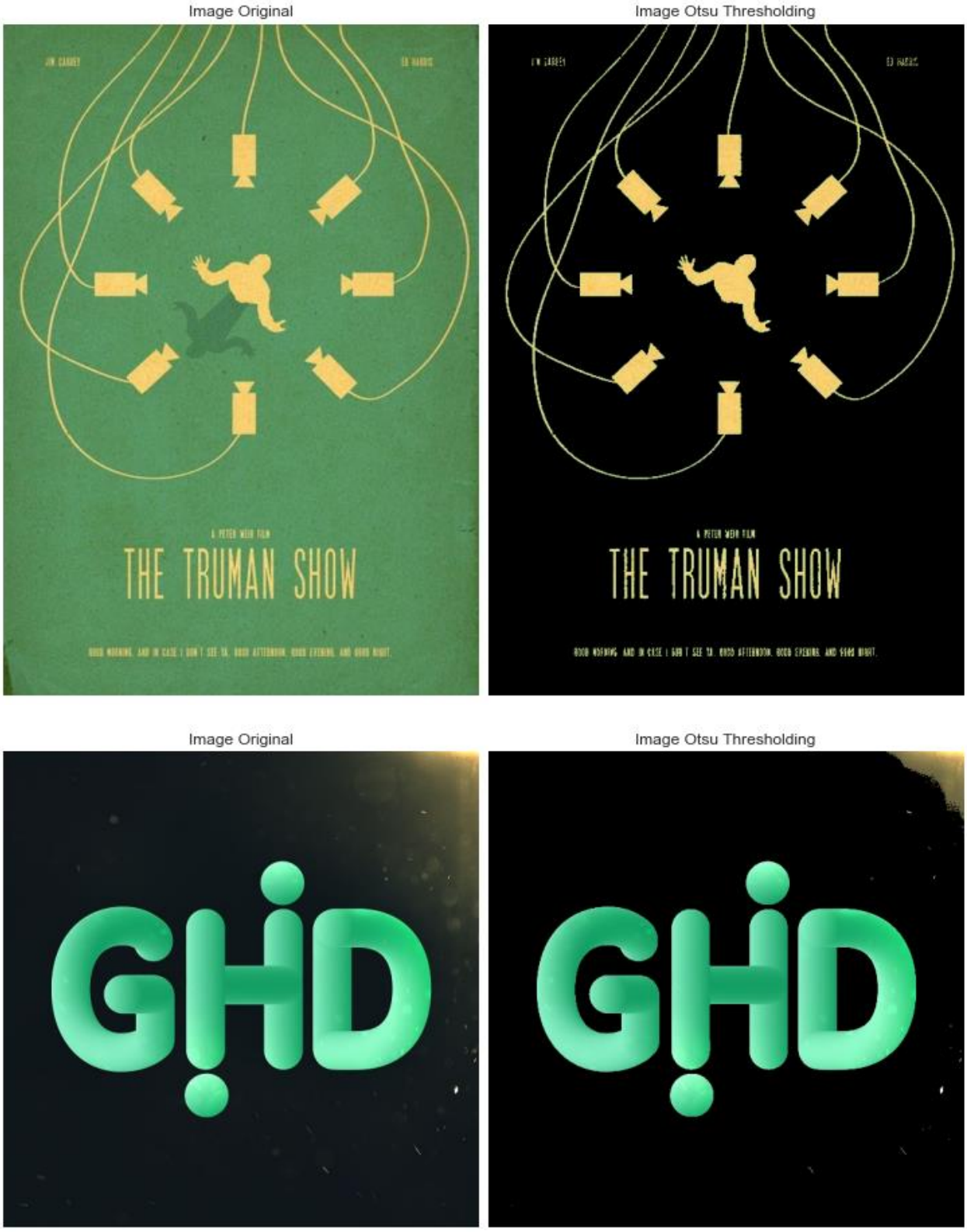
فهرست مطالب

سوال ۱)

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 ۱۶۰ 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 ۲)

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

The iterative method

1. Assume the four image corners are background pixels and the rest of points are object pixels
2. At step t, compute the mean background and mean object gray levels. Pixels are separated into background and object at step t via the threshold determined at step t-1.

$$\mu_B^t = \frac{\sum_{(x,y)\in B} f(x,y)}{N_B} \qquad \mu_O^t = \frac{\sum_{(x,y)\in O} f(x,y)}{N_O}$$

$Background: B$ $\qquad$ $Object: O$

3. Compute the threshold for step t+1. This threshold defines a new background-object separation.

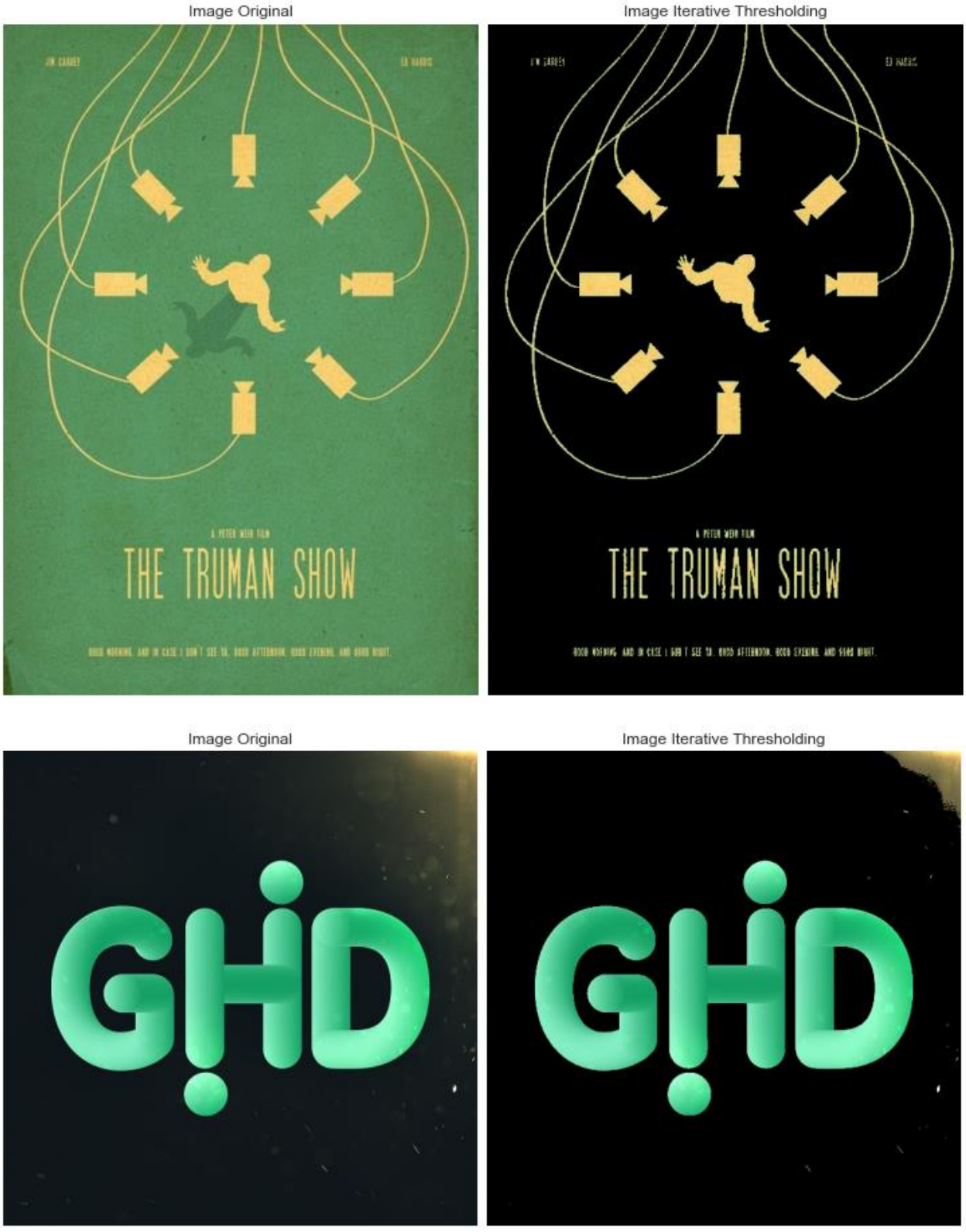
$$T^{t+1} = \frac{\mu_B^t + \mu_o^t}{2}$$

4. If threshold is equal to the previous step threshold ($T^{t+1} = T^t$), halt.

. Otherwise, go to step 2

الگوریتم *Iterative Thresholding*

نتیجه آستانه گیری به کمک الگوریتم *Iterative*



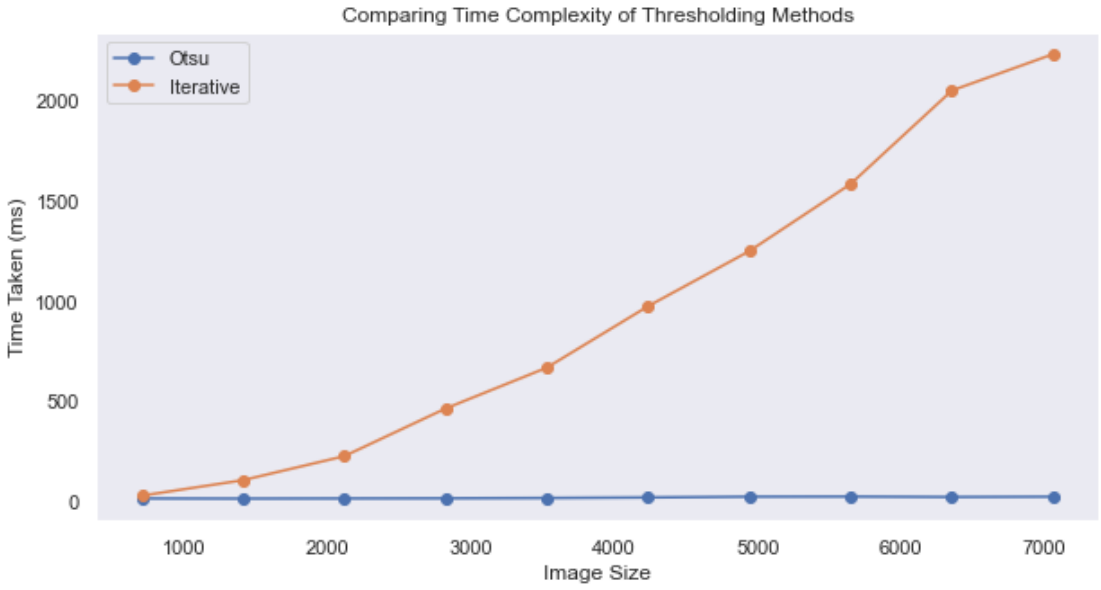
اعمال نتایج صفحه قبل به عنوان ماسک به تصویر اصلی مطابق با خواسته سوال

سوال ۳)

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 ۰ تا ۲۵۵، 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.

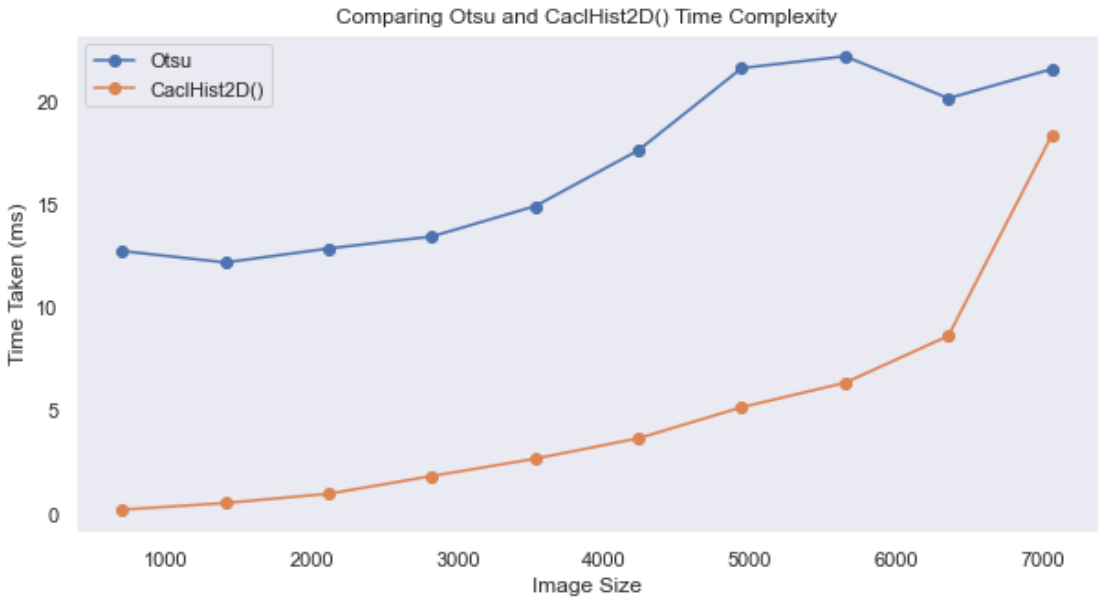
الگوریتم Iterative، در هر مرحله نیاز دارد تمام پیکسل‌های تصویر را به دو دسته Background و Object تقسیم کند و بر روی تمام این پیکسل‌ها(تمام پیکسل‌های تصویر) میانگین گیری انجام دهد. این امر باعث می‌شود با افزایش اندازه تصویر، این الگوریتم بسیار ناکارامد شود. با این وجود، برای تصاویر با اندازه معقول این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



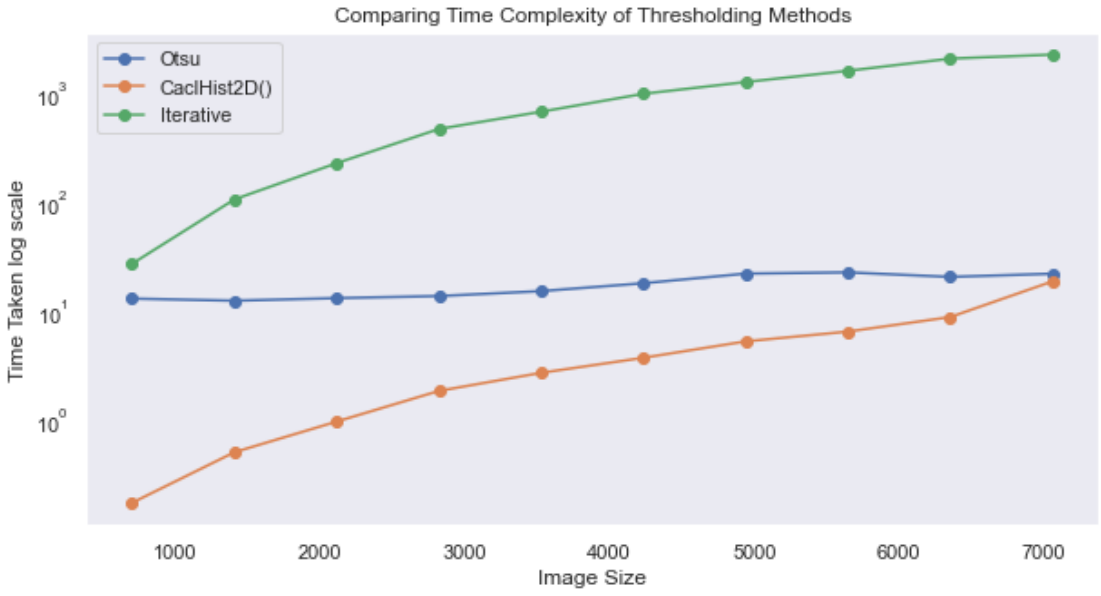
نمودار زمان اجرا(میلی‌ثانیه) بر حسب عرض تصویر(پیکسل)

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 ۷۰۰۰ پیکسل نیز در حدود ۲۲ 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 (۱ بار). با انجام آزمایشی دیگر این نظریه را نیز اثبات کردیم. دو نمودار زیر، سرعت رشد یکسانی دارند.

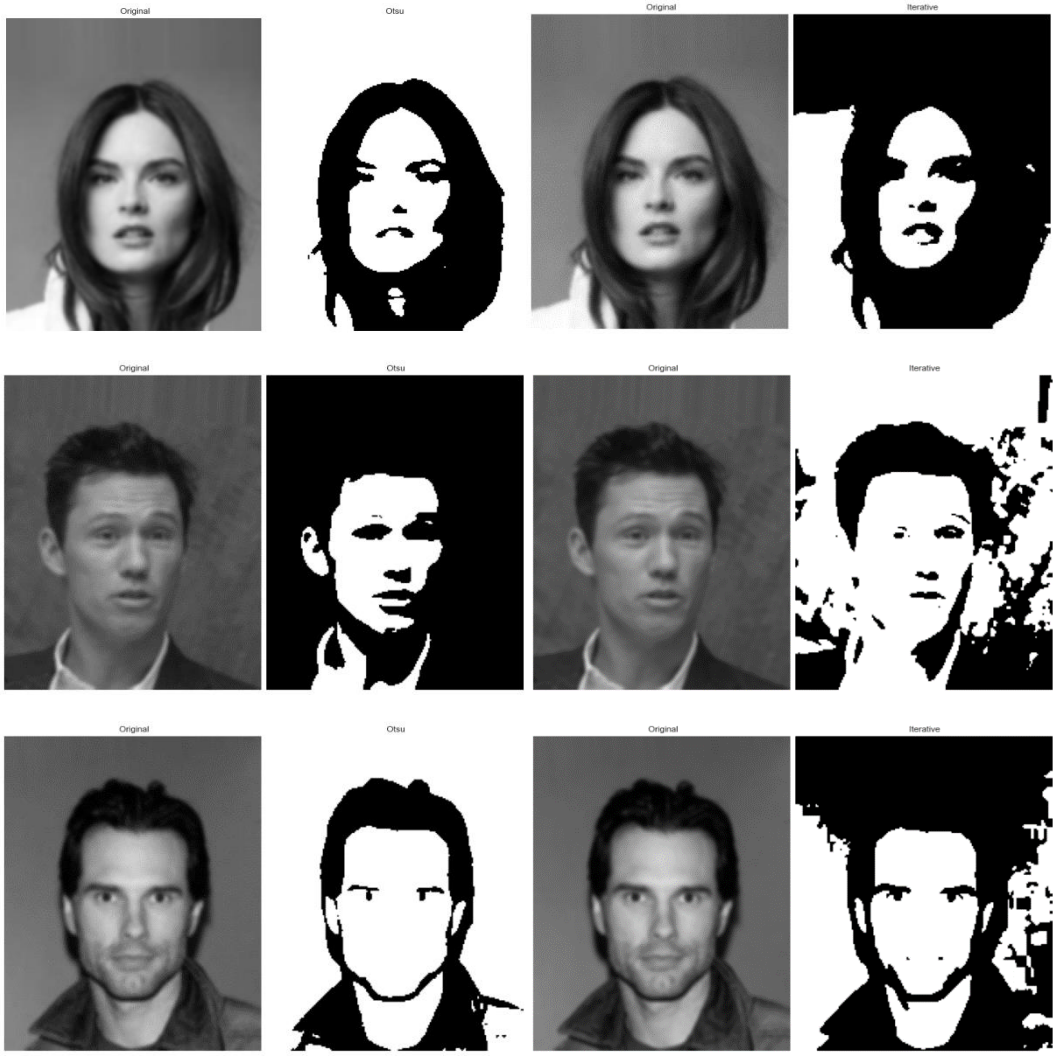


مقایسه زمان اجرای کل الگوریتم Otsu و بخش محاسبه هیستوگرام



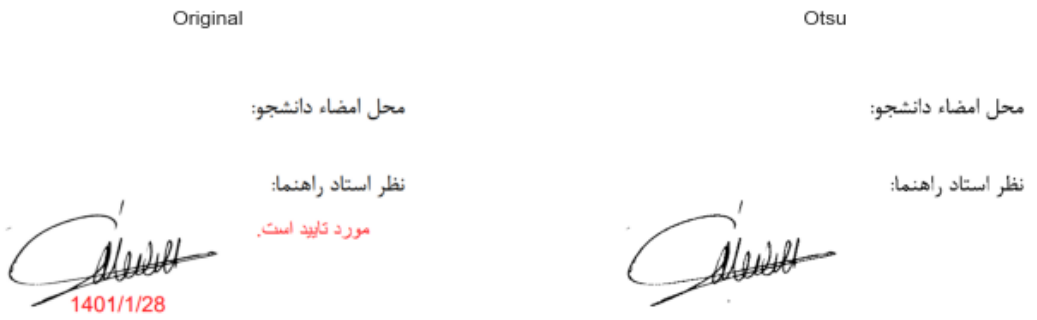
مقایسه الگوریتم‌ها با استفاده از مقیاس لگاریتمی

در مورد دقت این دو الگوریتم می‌توان حدس زد که چون الگوریتم Otsu تمام حالت‌های ممکن برای آستانه گیری را آزمایش می‌کند(۰ تا ۲۵۵) 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.

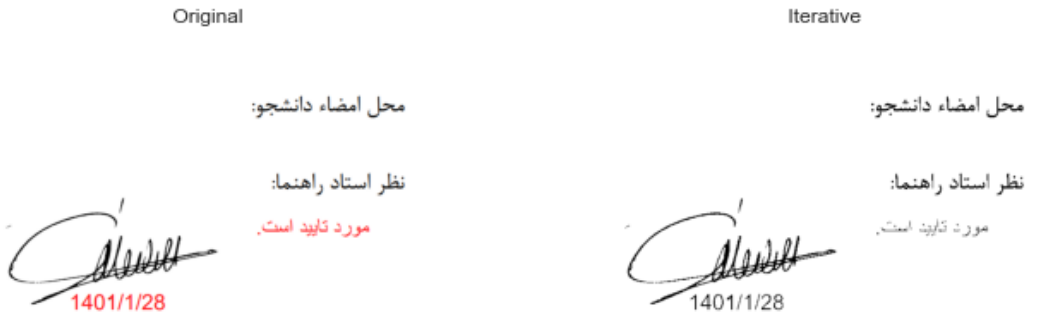


آستانه‌گیری‌های سمت راست مربوط به روش Iterative و سمت چپ مربوط به روش Otsu

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:



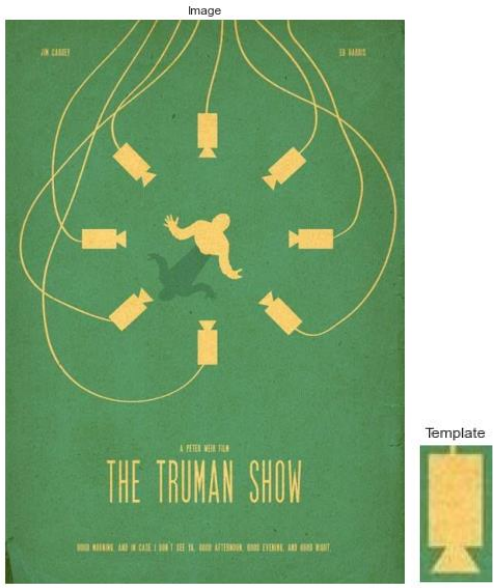
نتیجه الگوریتم Otsu: متون قرمز از بین رفته اند



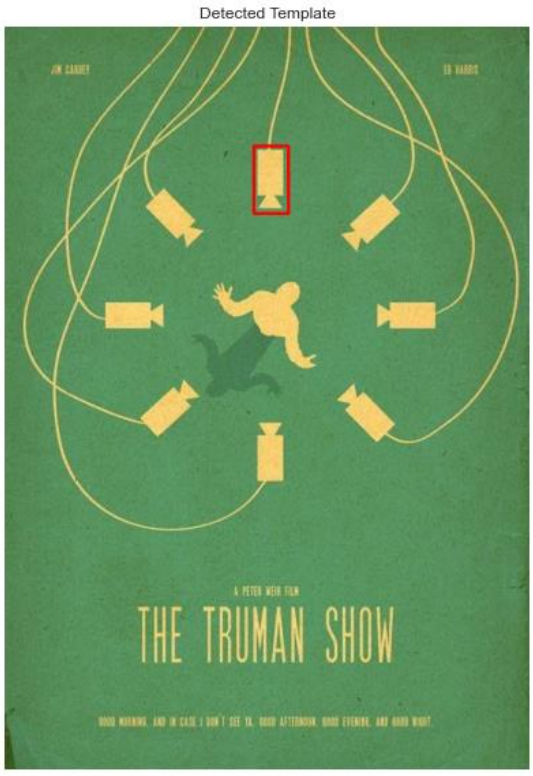
نتیجه الگوریتم Iterative: متون قرمز باقی مانده اند

سوال ۴)

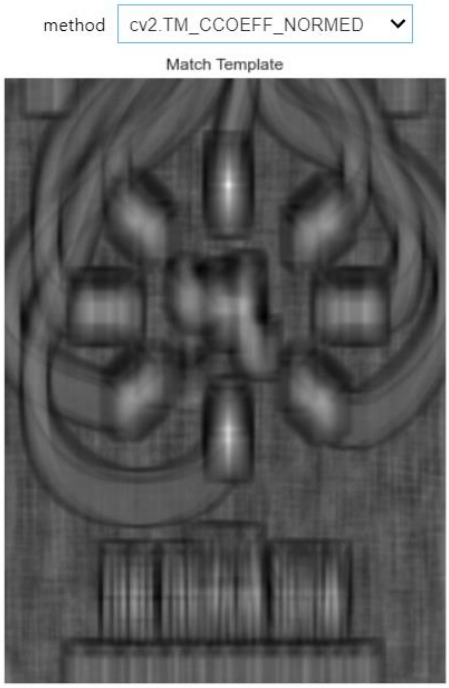
در این سوال قصد داریم به کمک روش تطبیق کلیشه یا Template Matching، دوربین‌های موجود بر روی پوستر زیر را تشخیص دهیم. همان‌طورکه دیده می‌شود تعدادی دوربین مشابه اما با میزان چرخش متفاوت در پوستر موجود هستند.



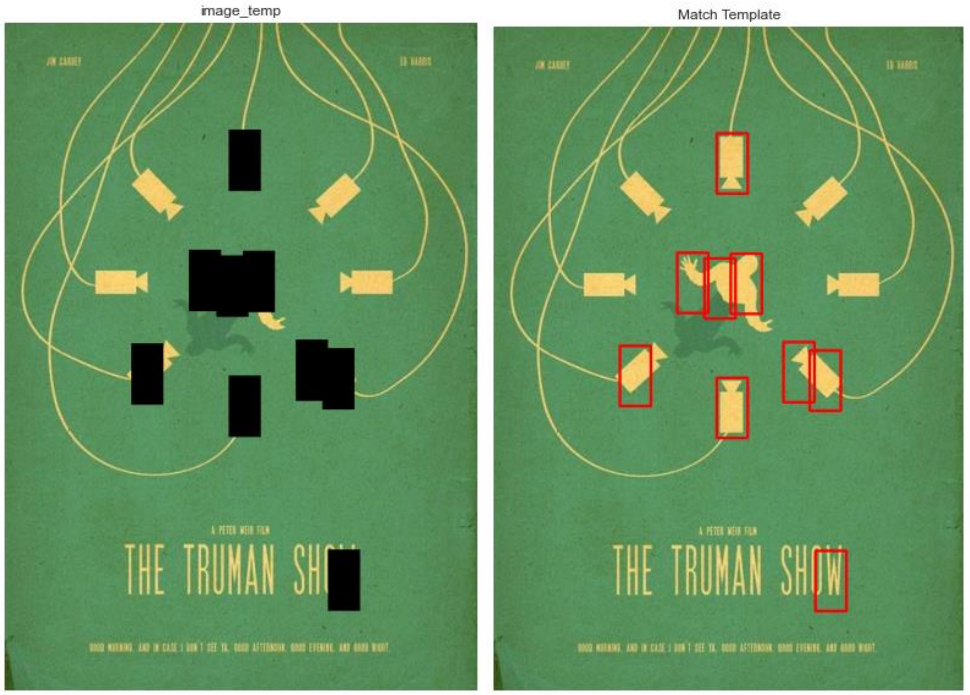
با استفاده از تابع matchTemplate موجود در کتابخانه OpenCV، به نتیجه زیر می‌رسیم.



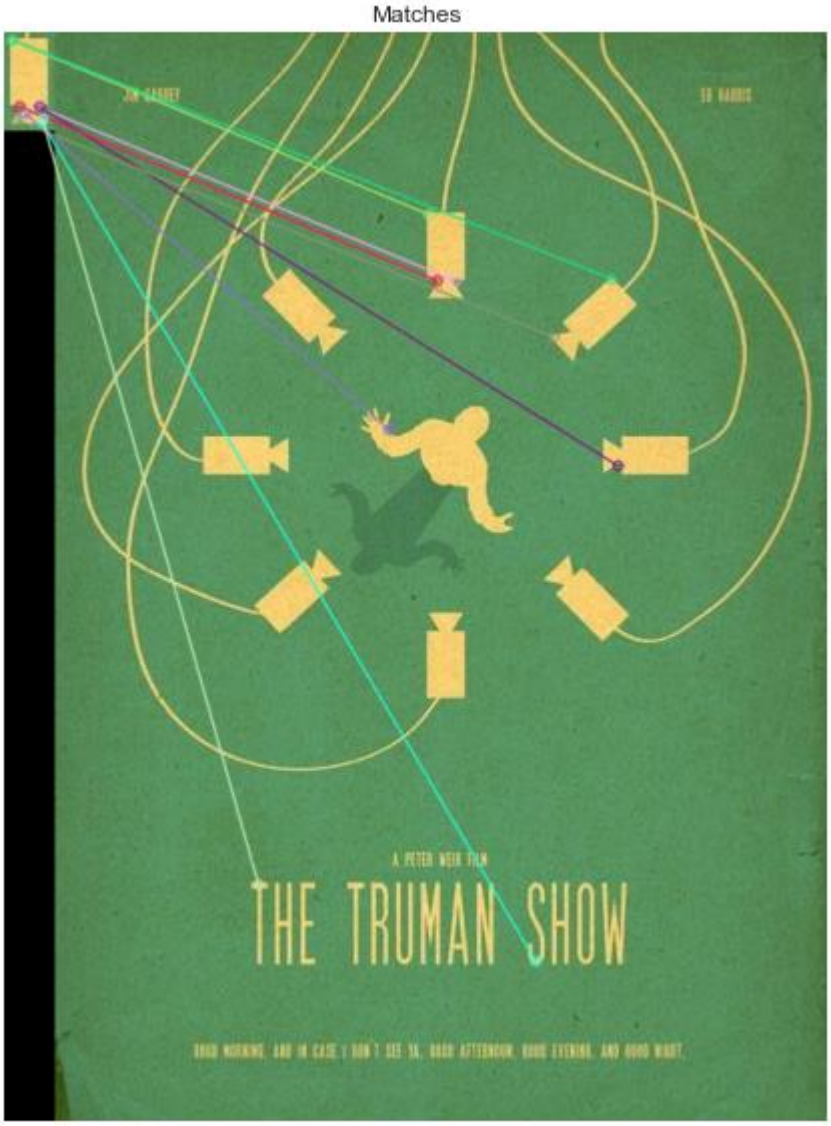
خروجی تابع matchTemplate مشابه تصویر زیر است که با استفاده از ماکسیمم گیری می‌توان پراحتمال ترین موقعیت شیء را تشخیص داد.



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.



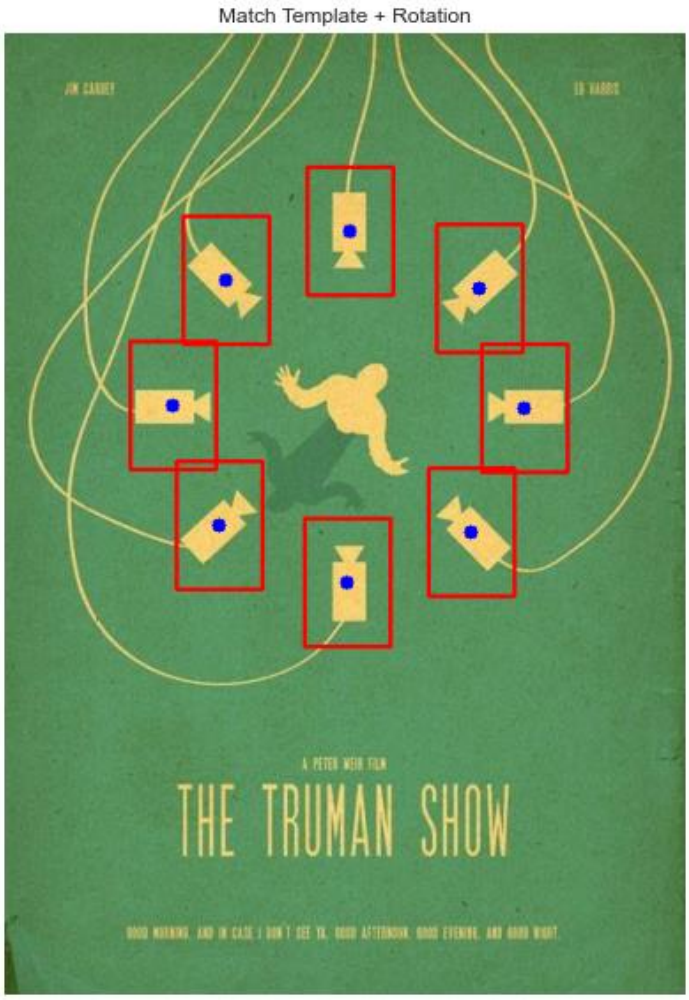
پس از شکست در مرحله قبل به سراغ روش‌های FeatureMatching رفتیم. این روش‌ها نیز به دلیل کمبود ویژگی منحصربه‌فرد و واضح در تصویر کلیشه(دوربین) به نتیجه خوبی نرسیدند.



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