|  |  |
| --- | --- |
| نام و نام خانوادگی | **علی فرتوت – محمد عرب زاده** |
| شماره دانشجویی | **810102221 - 810102206** |
| تاریخ ارسال گزارش | **1402.09.17** |

|  |  |  |
| --- | --- | --- |
|  | **به نام خدا**  **دانشگاه تهران**  **دانشکده‌ مهندسی برق و کامپیوتر** |  |
| **درس شبکه‌های عصبی و یادگیری عمیق**  **تمرین سوم** | | |

**فهرست**

[**پاسخ 1**. **SAM** 1](#_Toc115453335)

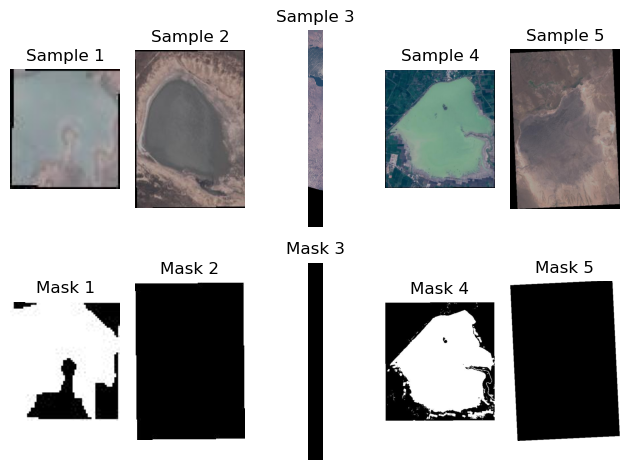
[**پاسخ ۲** **– Faster RCNN**](#_Toc115453337) ۸

سوال اول

بخش اول و سوم

ابتدا یک دیتاسیت با پایتورچ میسازیم که دیتا را به ما نشان دهد. این دیتاست PlotDataset نام دارد.

5 عکس رندوم را پلات کردیم. و حاصل به شکل زیر درآمد.



.همانطور از شکل معلوم است بعضی از تصاویر دارای rotation هستند. و حاصل این عمل ایجاد padding و مقدار گذاری 1برای کلاس آب میباشد!

برای رفع این مشکل از کتابخانه albumentations استفاده میکنیم. که جلوتر به آن اشاره خواهیم کرد.

بعد از پلات تصاویر به همراه ماسک. حال نوبت ساخت دیتاست مخصوص مدل sam میباشد.

در ابتدا تابعی برای پیدا کردن bounding box ها مینویسیم. این الگوریتم از ماسک استاده میکند. و اولین جایی که در آن مقدار pixel 0 نباشد رو ذخیره میکند. و این روند تا پیدا کردن آخرین pixel

0< ادامه پیدا میکند. و در آخر الگوریتم به ما 2 نقطه x و y مین و ماکس را میدهد.

حال باید خود دیتاست را بنویسم. داده را از فایل میخوانیم. و به ازای هر عکس image و mask سه transform زیر را انجام میدهیم.

1. همانگونه که گفته شد دیتا یک سری مشکلات دارد. مانند روتیشن که برای این کار از centercrop استفاده میکنیم. علت این است که ما نمیدانیم هر عکس چه مقدار چرخیده است برا همین نمیتوان از خود rotation استفاده کرد. از این رو centercrop با مقدار 10 درصد میزنیم تا padding موجود حذف شود. این کار در ازای از دست دادن اطلاعات به bounding box بسیار خوب کمک میکند.

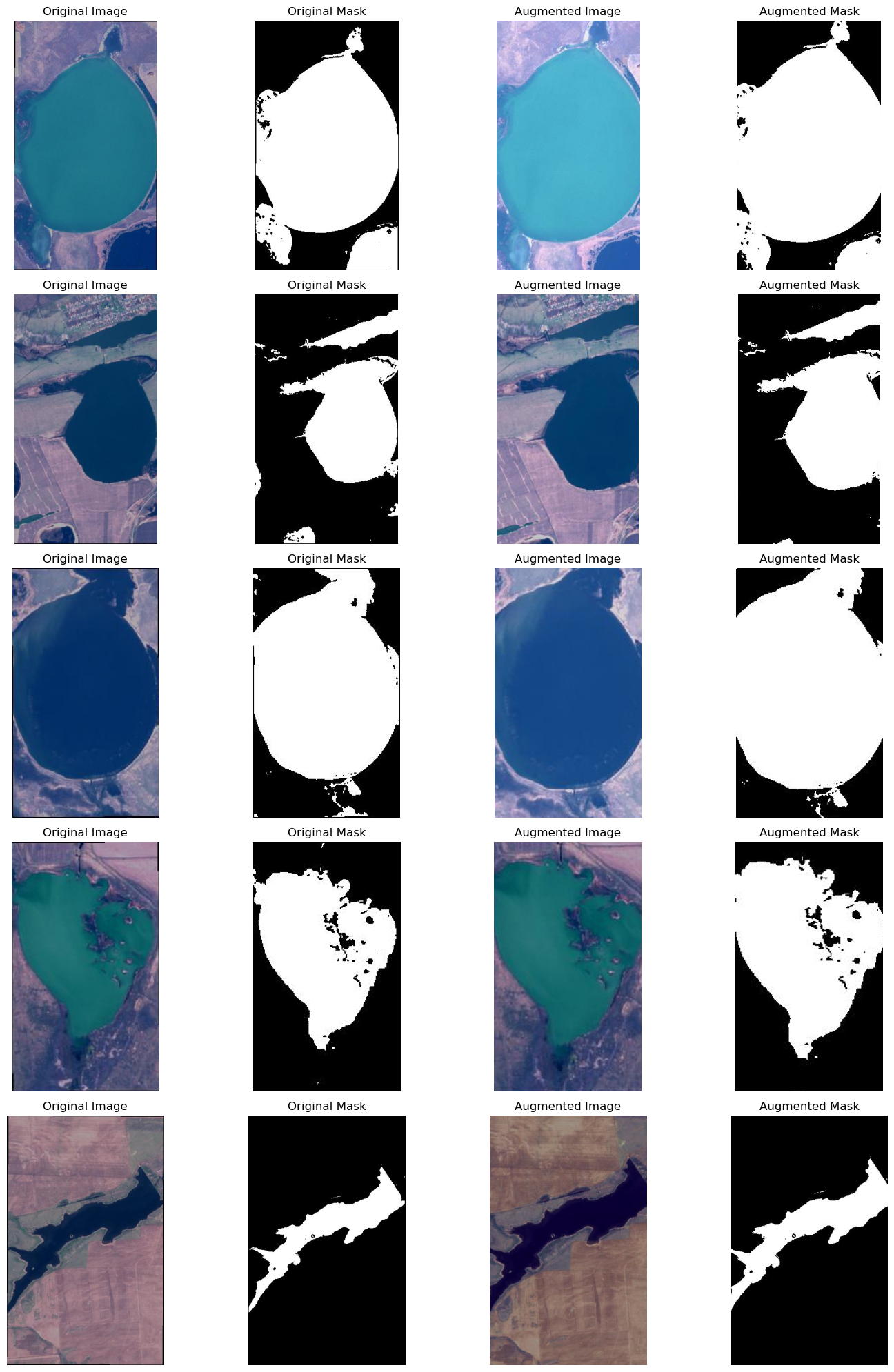
از طرفی بعضی از عکس ها به دلیل روز و شب و بودن کیفیت رنگ خود را از دست میدهند.

از این رو روشنایی و کنتراست و ستوریشن عکس ها را به صورت رندوم عوض میکنیم.

1. مورد بعدی تبدیل mask به فرمت 256 \* 256 ونرمال سازی آن بین 0 و 1 است. که این امر بسیار روشن است.
2. و مورد آخر عکس و ماسک را به تنسور پایتورچ تبدیل کرده ایم.

بعد از اعمال تغییرات processorمدل را صدا میزنیم تا اغییرات حالت پیش فرض را برای مدل اعمال کند.

عکس بعد از تقویت سازی به شکل زیر در آمده است.



**بخش دوم**

حال مدل را بارگذاری میکنیم. و لایه های prompt و vision encoder را فریز میکنیم.

مدل sam به صورت کلی دارای 3 بخش است که به اختصار آن را توضیح میدهیم.

1. Vision encoder که با استفاده از ViT یا همان vision transformers ابتدا ویژگی های تصاویر را با استفاده از لایه های attention پیدا میکند.
2. Prompt encoder که بر اساس فرمت پرامت داده شده ( متن، نقطه و bounding box ) ورودی را تحلیل میکند.
3. قسمت نهایی (decoder)که بعد از گرفتن فیچر های تصاویر و prompt داده شده سعی بر پیش بینی هر پیکس در تصاویر میکند.

ما در این تمرین بخش decoder را آموزش میدهیم و بقیه قسمت را فیریز میکنیم. تا فقط قسمت تعمیم دهی با دین پرامت و فیچر را بهبود دهیم.

**البته در این سوال مدل شبکه عصبی یک مقداری عوض شده است. بعد ازگرفتن خروجی از مدل sam بهترین پیشبینی را بر اساس iou\_scores خود مدل میگیریم. بعد از آن یک batchnormalization و یک تابع sigmoid پشت هم قرار میدهیم تا خروجی بین 0 و1 شود.**

**بخش چهارم**

حال نوبت نوشتن متریک ها و اپتیمایز و loss برای مدل هست. متریک ها iou و dice خواسته شده بود که پیاده سازی شدند. برای قسمت تابع خطا بعد از کلی جست و جو از بین تمامی توابع خطای موجود (Diceloss, Focalloss, Dice bceloss, MSE, MAE) BCEloss انتخاب شد. به چند دلیل:

1. تمامی موارد پیاده سازی شده اند و این تابع بهترین عملکرد را داشته است.
2. مسئله ما binary classification هست و BCEloss بسیار معقول هست.

برای بخش optimizer از adamw استفاده کردیم مانند خود مقاله اصلی متا با لرنینگ ریت 0.001. علت بزرگ بودن لرنینگ ریت این است که داده ما بسیار کم هست. و برای تغییرات نیاز به گام های بزرگ تر هستیم.

معمولا از سری adam برای مدل های سنگین و پیچیده استفاده میکنیم. و بعد از پیاده سازی adamw و adam؛ adamw عملکرد بهتری داشته است.

**بخش پنجم**

حال تمامی بخش های گفته شده را استفاده کرده ایم و مدل را fine-tune کرده ایم.

Batch-size = 2, lr = 0.001, epochs = 15, ratio\_split = 0.9

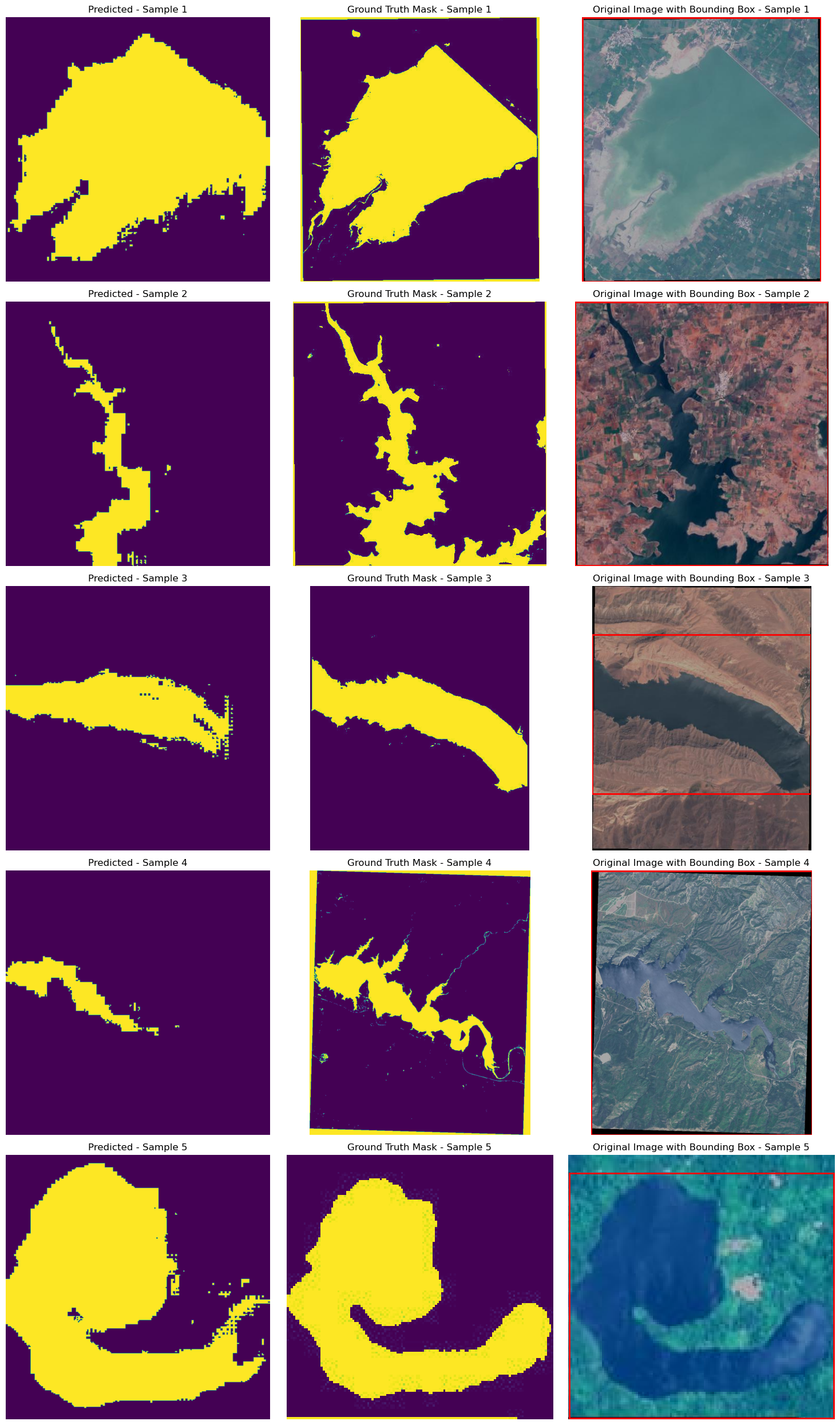
**بخش پنجم**

نتایج در آخر بدین صورت میباشد.

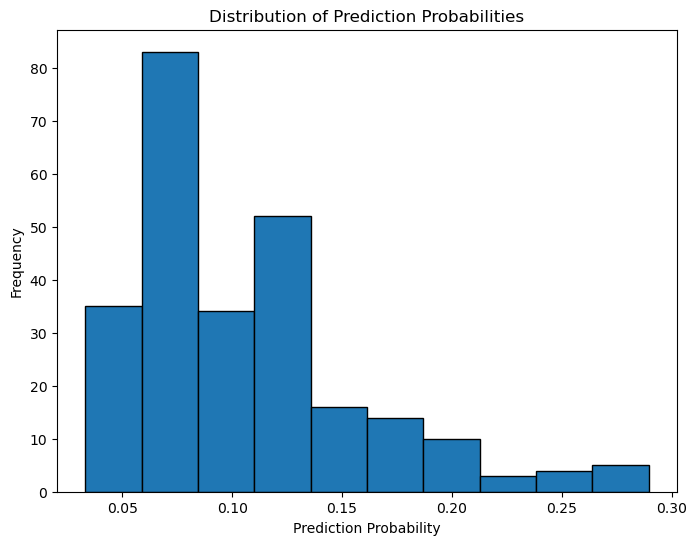
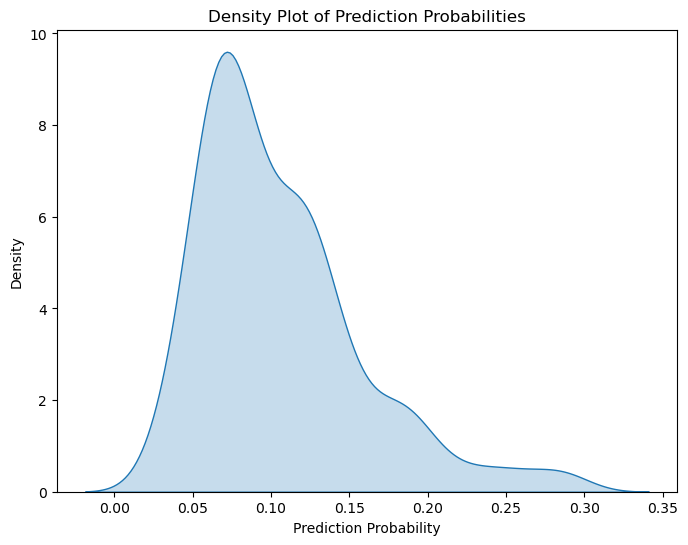


خطا به صورت محسوسی کاهش پیدا کرده است. و دقت نیز با دو متریک گفته شده افزایش داشته است.

علت پایین بودن iou میتوانند دلایل مختلفی داشته باشد. مثلا وجود نقص درخود ماسک truth که مشکلات زیادی داشت. مثلا در جایی که واقعا دریا بود 0 و درجایی که padding بود 1 در نظر گرفته شده بود. با این حال فیچر ها به صورت معقولی در اومده اند که در شکل زیر مشاهده میکنید.



از طرفی توزیع پیشبینی به شکل زیر میباشد.



که خب منطقی میباشد. چون معمولا اکثر داده 0 میباشند برا همین میانگین باید کمتر از 0.5 باشد.

سوال دوم

**بخش اول**

معماری Faster RCNN به چند دلیل به معماری های Fast RCNN و RCNN برتری دارد:

اول اینکه RPN را درون معماری شبکه جای می‌دهد، در حالی که دو معماری دیگر این زیر-شبکه را ندارند و این قسمت باعث می‌شود که هم آموزش روی تولید Proposal صورت بگیرد هم روی شناسایی اشیاء و این هم باعث می‌شود تا مدل با یک گذر Proposal ناحیه ای تولید و طبقه‌بندی انجام دهد به جای اینکه برای هر Proposal یک گذر از قسمت کانولوشنال کند و این سرعت را افزایش می‌دهد.

ثانیا، Faster RCNN دقیق‌تر است، چون تمام ویژگی‌های کانولوشنال در تمام Proposal ها مشترک اند و از یک لایه Pooling برای استخراج ویژگی برای هر Proposal استفاده می‌کند.

و ثالتا، Faster RCNN می‌تواند به صورت End-to-End آموزش داده شود، به این معنا که مدل می‌تواند برای تولید Proposal و شناسایی اشیاء بهینه‌سازی شود. در این حالت هم Performance بهتر و هم دقت بالاتری نسبت به مدل هایی که جداگانه برای هر Task آموزش داده می‌شوند، خواهد داشت.

*توضیحات مقاله:*

لایه‌های کانولوشنال ۱۳ لایه کانولوشنال دارند، ۱۳ لایه ReLU و ۴ لایه Pooling.

تمام لایه‌های کانولوشنال دارای kernel\_size = 3، padding = ۱ و stride = ۱ می‌باشند و تمام pooling ها هم دارای kernel\_size=2 و padding=0 و stride =۲ می‌باشند.

از آنجایی که تمام کانولوشن ها در Faster RCNN با padding = 1 گسترده شده‌اند، عکس اصلی به سایز(N+2) \*(M+2) تبدیل شده و سپس با کرنل ۳ به سایز قبلی باز‌می‌گردد. این روش اطمینان حاصل می‌کند که لایه‌های کانولوشنال سایز عکس‌ها را تغییر نمی‌دهند. سایز عکس فقط زمانی تغییر می‌کند که از لایه Pooling رد شود که هر بعد نصف می‌شود.

راه‌های پیشین برای فریم های تشخیص دهی زمان‌بر هستند، به عنوان مثال، RCNN از selective search استفاده می‌کند. Faster RCNN اما از RPN استفاده می‌کند تا این فریم ها را تولید کند به جای sliding window و selective search. این روش تولید فریم تشخیص را تسریع می‌بخشد. RPN دو بخش دارد: یک بخش anchor های طبقه‌بندی مثبت و منفی را با استفاده از Softmax به دست می‌آورد و بخش دیگر یک جواب دقیق با استفاده از حساب regression ناحیه مستطیلی به دست می‌آورد.

استفاده از RoI Pooling برای تغییر هر Proposal تولید شده به سایز pooledw\*pooledh با گام های زیر است:

1- از آنجایی که خروجی مختصات Proposal ها بعد از لایه Proposal سمت چپ عکس اصلی هستند، مقیاس فضایی در ابتدا استفاده می‌شود تا Proposal را به feature map تبدیل کند؛

2- Proposal ها را در هر feature map به خانه‌هایی به سایز pooledw\*pooledh تقسیم می‌کنیم؛

3-روی هر قسمت سلول ها، Max Pooling انجام می‌شود.

قسمت Classification دسته هایی را که هر Proposal به آن تعلق دارد توسطfeature map های پیشنهادی، لایه Fully Connected و Softmax محاسبه می‌کند و بردار احتمال را به عنوان خروجی می‌دهد.

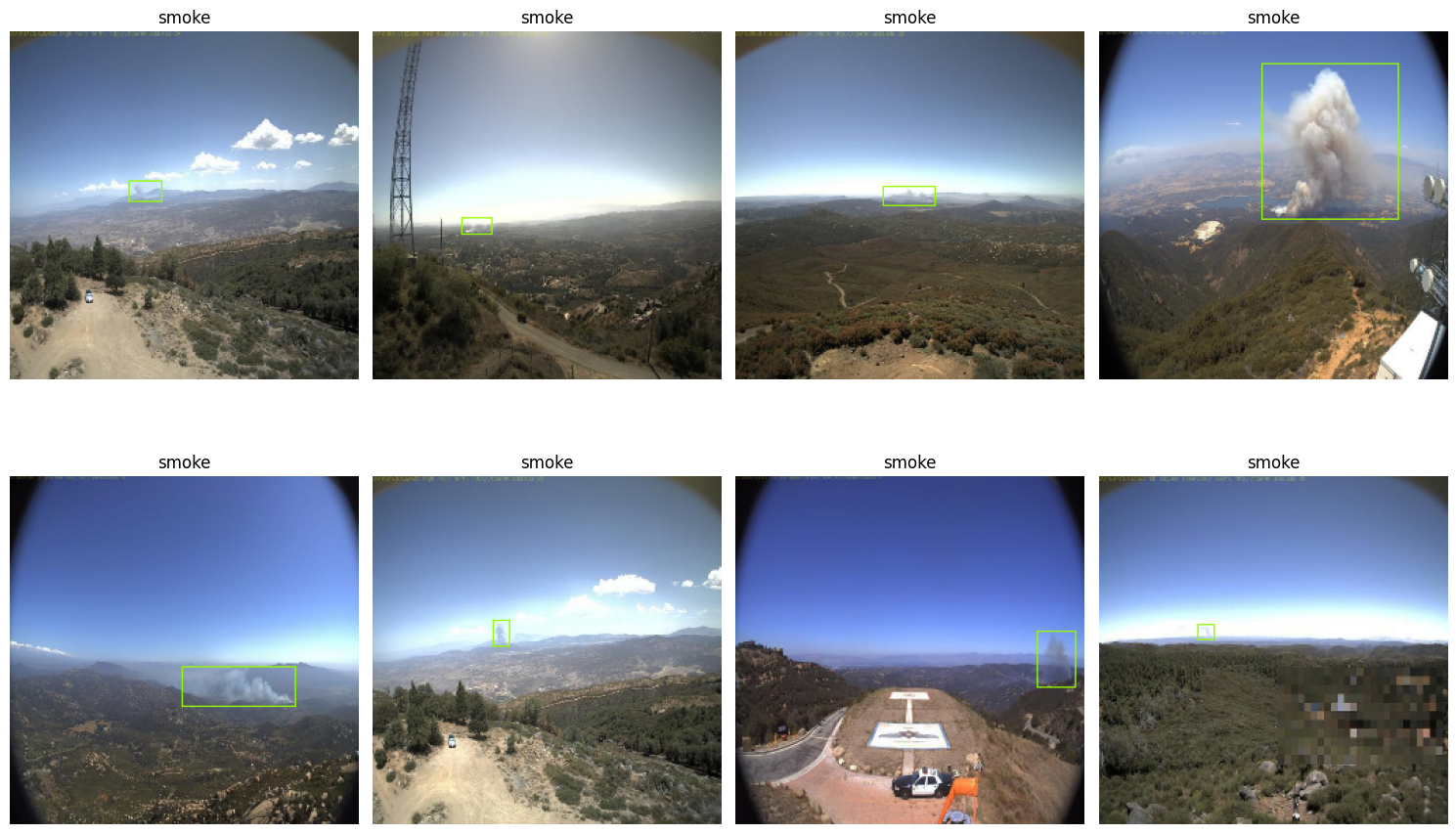
**بخش دوم**

برای نمایش پوشه train ابتدا آن را باز می‌کنیم. تعداد عکس‌های داخل آن را می‌شماریم تا تعداد عناصر Tensor نظیر آن را مشخص کنیم. مقدار Resize شده هر عکس را ۲۵۶ در ۲۵۶ در نظر‌ می‌گیریم و شروع می‌کنیم به خواندن.

هر عکس، یک annotation نظیر خودش را نیز دارد که آن را نیز نگه‌می‌داریم تا در نمایش از آن استفاده کنیم.

تابع get\_image\_annotation، برای هر عکس resize شده، annotation آن را نیز resize می‌کند.

پس از خواندن، 8 عکس random خوانده شده را همراه با annotation آن نمایش می‌دهیم که به صورت زیر است:

که مشاهده می‌شود برچسب همه عکس‌ها smoke است.

**بخش سوم**

معماری Backbone و RPN شبکه‌ی Faster RCNN در کلاس FasterRCNN تعریف شده است و متأسفانه وقت نشد که بقیه اجزاء تعریف یا پیاده‌سازی شوند.

تمام لایه‌های Convolutional دارای kernel\_size = 3، padding = ۱ و stride = ۱ می‌باشند و تمام pooling ها هم دارای kernel\_size=2 و padding=0 و stride =۲ می‌باشند.

۱۳ لایه کانولوشنال و ۱۳ لایه تابع تحریک ReLU پشت هم به ترتیب قرار دارند و ۴ دفعه AvgPool گرفته‌می‌شود. خروجی این شبکه کانولوشنال یک Feature Map است که تحویل قسمت Region Proposal Network می‌شود که دو خروجی از آن بیرون می‌آید: ۱- امتیاز objectness و ۲- bounding box های پیشنهادی