به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

# درس شبکه‌های عصبی و یادگیری عمیق

## تمرین سوم

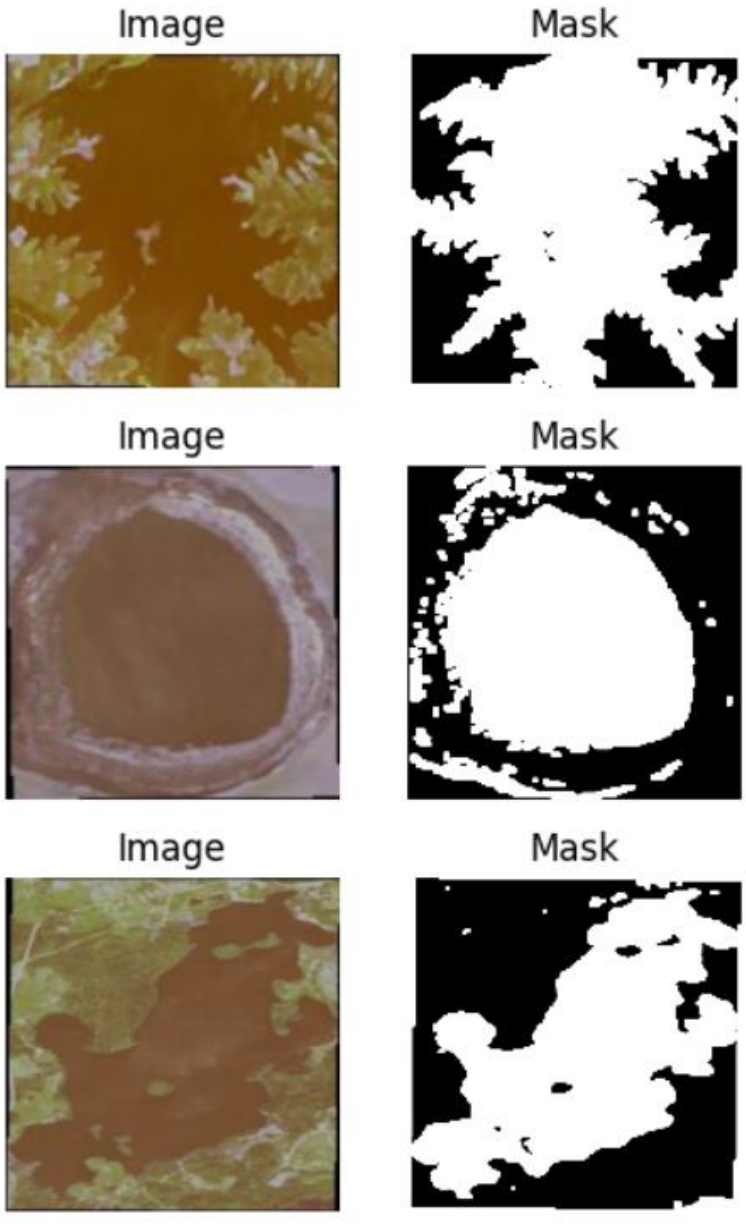
| | |
|---|---|
| نام و نام خانوادگی | فاطمه جلیلی – سالار صفردوست |
| شماره دانشجویی | 810199450 -810199398 |
| تاریخ ارسال گزارش | 1402/9/22 |

## فهرست

## پاسخ ۱- SAM

### 1-1. آماده سازی مجموعه داده

به صورت رندوم 3 عکس به همراه ماسک آن ها را نمایش می دهیم:




**شکل1-1. نمونه هایی از عکس های دیتا ست و ماسک مربوطه**

با بررسی عکس ها متوجه شدیم برخی از ماسک های دیتا ست کاملا سفید و یا کاملا سیاه هستند لذا در load کردن عکس ها threshold در نظر گرفتیم به این صورت که اگر بیش از مقدار مشخصی از ماسک سیاه یا سفید بود آن و عکس مربوطه در ماتریس ذخیره نکردیم

ابتدا با threshold 80 درصد کار را شروع کردیم و در قسمت کشیدن bounding_box متوجه شدیم این مقدار threshold باعث می شود در بسیاری از عکس ها bounding-box کل عکس در نظر گرفته شود لذا threshold را کم تر کردیم و برای پردازش سریع تر این مقدار را تا 60 درصد کاهش دادیم تا دیتا های کم تری داشته باشیم چرا که قسمت train بسیار زمان بر بود.

داده ها را به دو بخش train و test با نسبت 1 به 10 تقسیم می کنیم:

```
img_train shape: (495, 256, 256, 3)
mask_train shape: (495, 256, 256, 1)
img_test shape: (56, 256, 256, 3)
mask_test shape: (56, 256, 256, 1)
```

**شکل1-2. ابعاد ماتریس های** train وtest

## 2-1. بارگذاری مدل

Segment Anything Model (SAM) طراحی شده توسط Meta روشی برای تقسیم بندی اشیاء در تصاویر بر اساس اعلان های زبان طبیعی است . SAM از یک Vision Transformer (ViT-H) به عنوان رمزگذار تصویر استفاده می کند. مدل ViT-H گونه‌ای از معماری Vision Transformer است که یک مدل زبان در مقیاس بزرگ است که از قبل روی مجموعه داده‌های بزرگی از تصاویر آموزش داده شده است. مدل ViT-H تصویر ورودی را در یک نمایش برداری با ابعاد بالا رمزگذاری می کند.

SAM می تواند با 3 نوع prompt کار کند : به صورت text به آن گفته شود که چه object ای را جدا کند ، مختصاد یک باکس اطراف شی مورد نظر به آن داده شود و یا شی با نقاط کلیک مشخص شود.

prompt در یک نمایش برداری جداگانه کدگذاری می شود. سپس دو نمایش برداری با هم ترکیب شده و به رمزگشای ماسک ارسال می‌شوند، که ماسکی را برای شی مشخص شده توسط Prompt خروجی می‌دهد.

رمزگذار prompt یک رمزگذار متن ساده است که دستور ورودی را به یک نمایش برداری تبدیل می کند.

رمزگشای ماسک یک مدل ترانسفورماتور سبک وزن است که ماسک شی را از روی تصویر با توجه به prompt پیش‌بینی می‌کند .

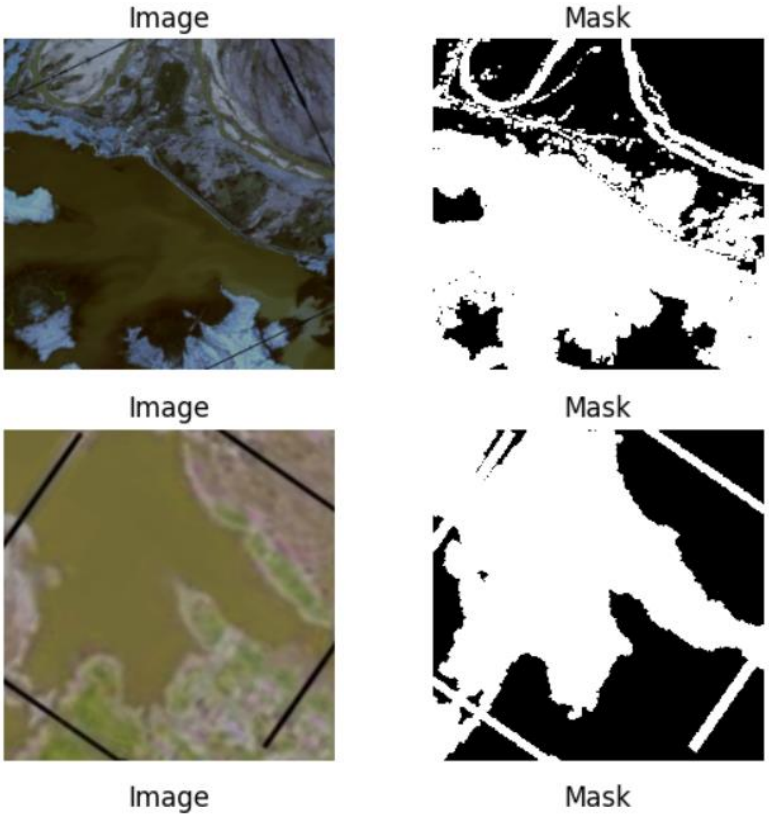
### 3-1. تقویت داده

همانطور که در ابتدا توضیح داده شد به جهت تشخیص درست bounding_box ها تصاویری که پهنه های آبی گسترده تا حدی که حجم بیش تر عکس را بپوشانند نشان می دادند را در ماتریس های مورد استفاده اضافه نکردیم .

طبق توصیه از کتابخانه Albumentations برای تقویت داده استفاده شده است به این ترتیب که تعدادی از عکس را هم به صورت افقی و هم به صورت عمودی Flip می کنیم و تا ماکسیمم 50 درجه می چرخانیم ، پهنه های آبی می توانند به شکل های مختلفی باشند لذا Flip کردن و چرخاندن آن ها گویی باعث می شود در دیتا ست خود پهنه های آبی بیش تری در جهت های مختلف را پوشش دهیم.

همچنین از gridDistotion استفاده می کنیم که یک اعوجاج الاستیک تصادفی روی عکس اعمال می کند و اثر کشیده شدن یا تاب برداشتن عکس را شبیه سازی می کند مشابه قبل پهنه های آبی کشیده شده یا تابیده شده می توانند نمایانگر پهنه های آبی جدیدی باشند که به دیتاست خود اضافه کردیم.

همچنین از HueSaturationValue استفاده می کنیم تغییرات رنگ و مقدار روشنایی عکس ها را تغییر داده و داده های جدیدی به دیتاست خود اضافه کنیم ، طبق داده ها بسیاری از عکس ها ماهواره ای داده شده رنگ آبی معمول برای پهنه های ابی را نداشتند لذا با این نوع تقویت داده باعث می شویم هنگام تست این گونه از تصاویر به مشکل نخوریم.



نمونه های Augment شده به روش های دیگر از روی عکس مشخص نیستند ولی دو نمونه از تصاویر augment شده با روش rotate را در بالا مشاهده می کنید.

## 4-1. بهینه ساز ، متریک و تابع هزینه

هنگام آموزش شبکه دو معیار خواسته شده را در هر epoch محاسبه می کنیم .

از بهینه ساز Adam که در اکثر انواع سوالات یادگیری استفاده می شود ، بهره می گیریم.

از آنجایی که در حال پیاده سازی segmentation هستیم از loss مربوط به همین امر که مربوط به معیاری که خواسته شده آن را محاسبه کنیم (Dice coefficient) است استفاده می کنیم ، این loss ترکیب همین معیار و cross entropy است و سعی می کند این دو را مینیمم کند.

برای پیاده سازی از کتابخانه monai استفاده می کنیم و به این صورت آن را تنظیم می کنیم:
sigmoid=True: این پارامتر نشان می‌دهد که ورودی تابع ضرر باید قبل از محاسبه تلفات از یک تابع فعال‌سازی سیگموئید عبور داده شود. زمانی استفاده می شود که خروجی مدل باید به توزیع احتمال بین 0 و 1 تبدیل شود.

Squared_pred=True: این پارامتر نشان می دهد که احتمالات پیش بینی شده باید قبل از محاسبه ضرر مجذور شوند. مربع کردن پیش‌بینی‌ها می‌تواند به تأکید بر تفاوت‌های بزرگ‌تر بین احتمالات پیش‌بینی‌شده و برچسب‌های هدف کمک کند، و به طور بالقوه حساسیت مدل را به تفاوت‌های ظریف بهبود می‌بخشد.

Reducation=mean: این پارامتر نحوه کاهش یا تجمیع مقادیر تلفات در تمام پیکسل ها یا وکسل های ورودی را مشخص می کند. در این مورد، کاهش روی «میانگین» تنظیم می‌شود، به این معنی که مقادیر تلفات در تمام عناصر ورودی به‌طور میانگین محاسبه می‌شوند و در نتیجه یک مقدار اسکالر نشان‌دهنده ضرر کلی برای دسته است.

### 5-1. Fine-Tune کردن مدل

مطابق آنچه خواسته شده دو جز Vision decoder و Prompt decoder را feeze می کنیم و مدل را روی بخش Mask decoder آموزش می دهیم

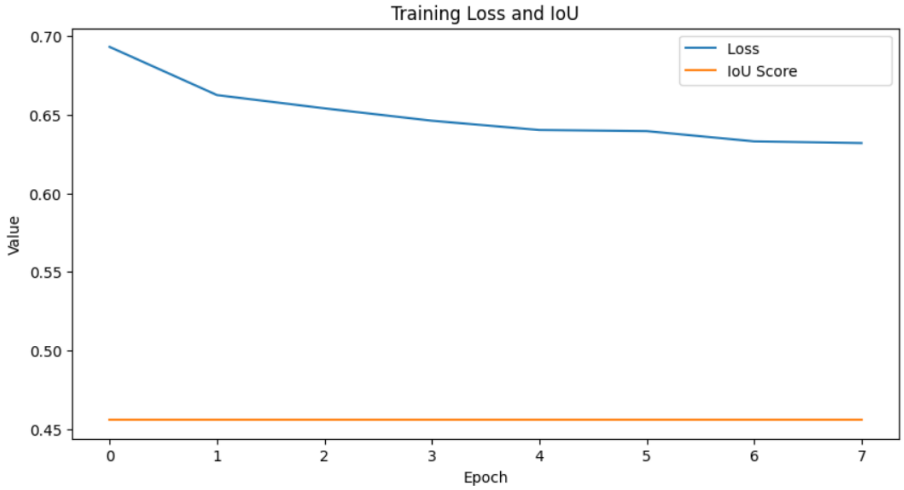
Bounding_box ها به صورت محاسبه مختصاد کوچک ترین و بزرگ ترین نقاط که روی ماسک سفید هستند محاسبه می شود.

هنگام پیاده سازی اولیه متریک dice coefficient هنگام ترین کردن پرینت نکردیم و به جهت وقت گیر بودن دوباره آموزش مدل این معیار را جداگانه در ادامه با آموزش روی بخشی از داده ها گزارش می کنیم:

مدل اصلی آموزش داده شده که از آن در ادامه استفاده می کنیم :

مطابق شکل می بینیم که متریک IOU در طول آموزش تغییر چندانی نکرده است و برابر 0.45 است و loss توضیح داده شده به حدود 0.63 می رسد که حد قابل قبولی است.

```
100%|██████████| 330/330 [08:58<00:00,  1.63s/it]
EPOCH: 0
Mean loss: 0.6932275555533919
Mean IoU: 0.4557304921776357
100%|██████████| 330/330 [08:42<00:00,  1.58s/it]
EPOCH: 1
Mean loss: 0.6625483774201759
Mean IoU: 0.4557304921776357
100%|██████████| 330/330 [08:43<00:00,  1.59s/it]
EPOCH: 2
Mean loss: 0.6541142632691903
Mean IoU: 0.4557304921776357
100%|██████████| 330/330 [08:28<00:00,  1.54s/it]
EPOCH: 3
Mean loss: 0.6462202097546856
Mean IoU: 0.4557304921776357
100%|██████████| 330/330 [08:22<00:00,  1.52s/it]
EPOCH: 4
Mean loss: 0.6403340332665289
Mean IoU: 0.45573049819016725
100%|██████████| 330/330 [08:33<00:00,  1.56s/it]
EPOCH: 5
Mean loss: 0.6396234728712006
Mean IoU: 0.4557304921776357
100%|██████████| 330/330 [08:38<00:00,  1.57s/it]
EPOCH: 6
Mean loss: 0.633090736384166
Mean IoU: 0.4557304921776357
100%|██████████| 330/330 [08:39<00:00,  1.57s/it]EPOCH: 7
Mean loss: 0.6320083490081712
Mean IoU: 0.4557304921776357
```




مدل ترین شده با دیتا های کم تر صرفا برای گزارش متریک Dice coefficient :

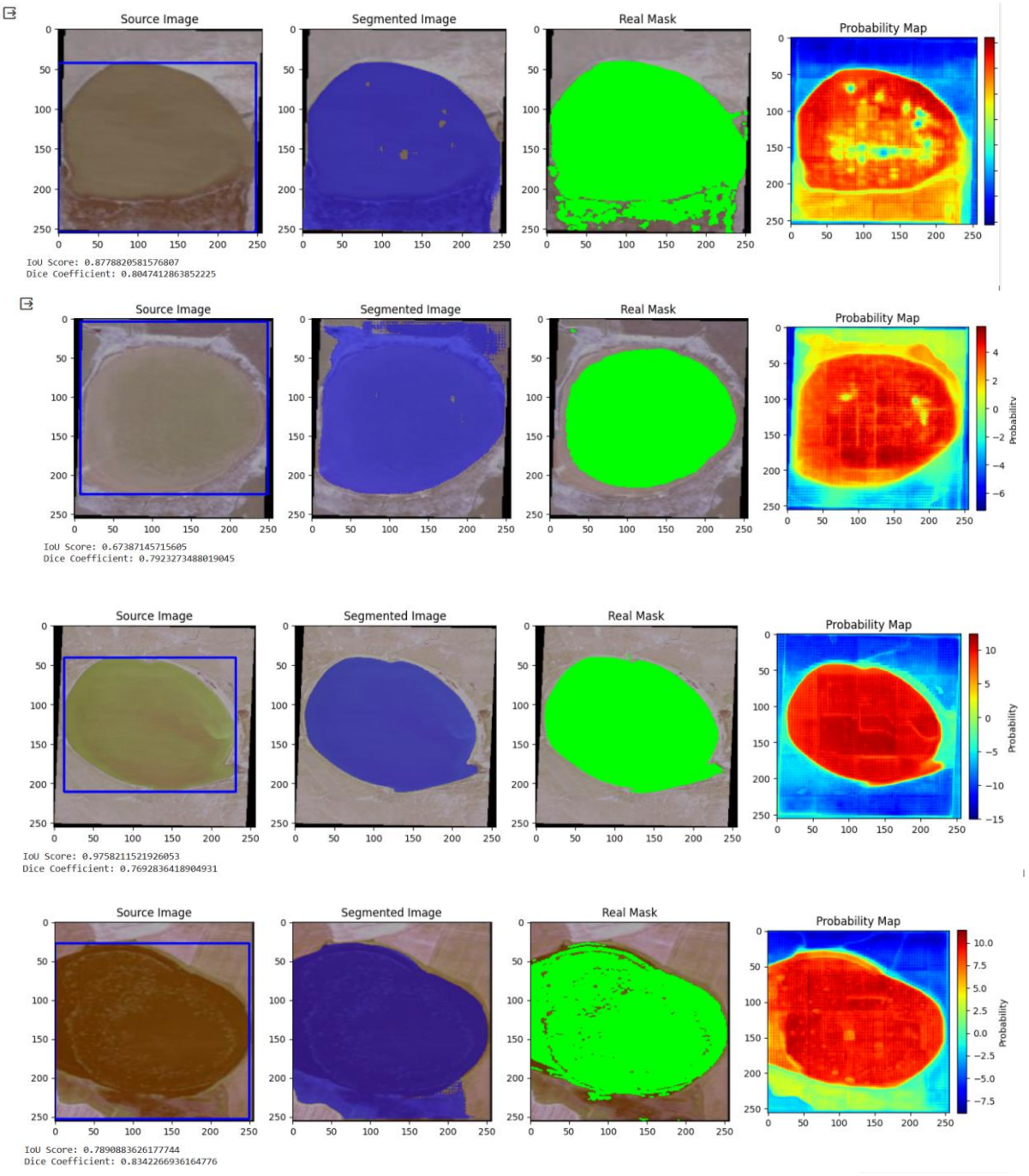
```
100%|██████████| 60/60 [01:35<00:00,  1.59s/it]
EPOCH: 0
Mean loss: 0.8618556165529311
Mean IoU: 0.5232272677951388
Mean Dice Coefficient: 0.6847043465195167
100%|██████████| 60/60 [01:32<00:00,  1.54s/it]
EPOCH: 1
Mean loss: 0.7946125135228622
Mean IoU: 0.5232272677951388
Mean Dice Coefficient: 0.6847043465195167
100%|██████████| 60/60 [01:30<00:00,  1.51s/it]
EPOCH: 2
Mean loss: 0.783695095192256
Mean IoU: 0.5232272677951388
Mean Dice Coefficient: 0.6847043465195167
100%|██████████| 60/60 [01:30<00:00,  1.51s/it]
EPOCH: 3
Mean loss: 0.784703345002412
Mean IoU: 0.5232272677951388
Mean Dice Coefficient: 0.6847043465195167
100%|██████████| 60/60 [01:30<00:00,  1.51s/it]
EPOCH: 4
Mean loss: 0.7776406917625609
Mean IoU: 0.5232272677951388
Mean Dice Coefficient: 0.6847043465195167
100%|██████████| 60/60 [01:30<00:00,  1.51s/it]
EPOCH: 5
Mean loss: 0.7751521820764207
Mean IoU: 0.5232272677951388
Mean Dice Coefficient: 0.6847043465195167
```

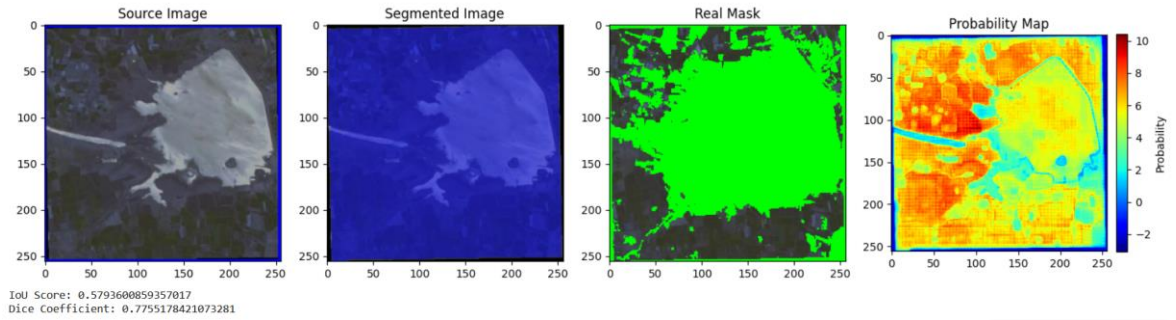
مدل را برای چند epoch ترین کردیم که حدود متریک dice coefficient را نشان دهیم مشاهده می شود این معیار هم در طول ترین تغییر چندانی نمی کند و حدود 0.68 است که قابل قبول است.

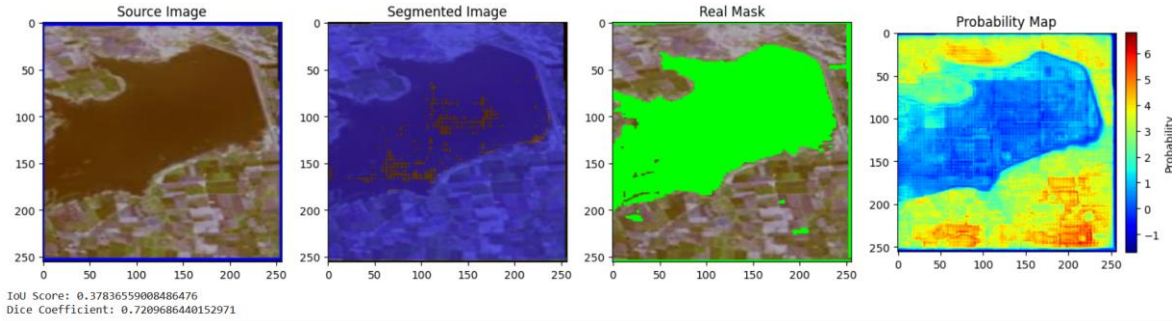
کد این قسمت به صورت کامنت در cell بعد از Cell مربوط به ترین قرار گرفته است.

## 6-1. ارزیابی نتایج

روی 5 نمونه از دیتا های تست نتایج خواسته شده شامل عکس اصلی به همراه box ، عکس segment شده یا همان عکس اصلی به همراه ماسک پیش بینی شده ، ماسک اصلی و probability map را نشان می دهیم:






ما ابتدا تصاویر بیش تری در هنگام Augmentکردن اضافه کرده بودیم در این حالت مدل overfit کرده بود و متریک های روی داده های تست و ترین تفاوت زیادی داشتند ، منتها بعدا که از این حالت خارج شدیم حدود متریک ها روی داده های ترین و تست متناسب است .

علارغم گزارش داده هایی تستی که خوب سگمنت شده اند برخی از داده ها هم به درستی سگمنت نمی شوند که بیش تر به علت مشخص نشدن درست box است چرا که کل عکس box در نظر گرفته شده است مثل دو عکس آخر چرا که داده های دیتا ست بسیاری دچار چرخش بودند و اطراف عکس در ماسک مربوطه سفید بود که باعث خطا می شد در کدی کامنت شده سعی کردیم اطراف عکس ها را ببریم ولی نتایج بهتر نشد. شاید بتوان با ارائه روشی بهتر برای پیدا کردن box عملکرد مدل را بهبود بخشید.

## پاسخ ۲ – آشنایی و پیاده سازی مدل Faster RCNN

### 1-2. توضیحات مدل ها

مدل RCNN: در این مدل ما نیاز داریم که با الگوریتمی خاص مانند Selective Search، نواحی که قصد کلسیفای کردن آن‌ها را داریم داشته باشیم. طرز کار این نوع خاص شبکه این است که با فرض داشتن این نواحی، تک به تک آن‌ها را به سایز مشخصی تبدیل می‌کند و سپس از یک شبکه‌ی عصبی کانوولوشنال عبور می‌دهیم و در نهایت خروجی را به یک کلسیفایر از نوع svm و یک رگرسور bounding box می‌دهیم.

مدل Fast RCNN: در این مدل همچنان به الگوریتمی برای داشتن نواحی محتمل داریم، تنها به جای آنکه تک به تک نواحی را به یک شبکه‌ی کانوولوشنال بدهیم، یکبار تمامی عکس را به این شبکه داده و فیچر مپی به دست می‌آوریم؛ سپس روی این فیچر مپ عملیات spatial pyramid pooling را برای تک به تک نواحی محتمل روی فیچر مپ پیاده‌سازی می‌کنیم تا به برداری با سایز مشخص از اعداد برسیم.

در نهایت این بردار را از دو لایه‌ی fully connected عبور می‌دهیم و خروجی را هم به یک کلسیفایر سافت مکس و هم به یک رگرسور خطی با شبکه‌ی عصبی می‌دهیم.

مدل Faster RCNN: در این مدل دیگر فرض می‌شود می‌خواهیم حتی نواحی محتمل و پیشنهادی را نیز با یک شبکه‌ی عصبی جدا به دست آوریم که به این شبکه RPN می‌گوییم.

ابتدا تصاویر را از یک شبکه‌ی کانوولوشنال مانند vgg16 و resnet عبور می‌دهیم تا فیچر مپی را به دست آوریم. سپس، خروجی به دست آمده را به این شبکه‌ی RPN می‌دهیم و انتظار دو خروجی نوع کلاس‌های نواحی و ضریب اطمینان و bounding box مربوط به ناحیه‌های پیشنهادی داخل تصویر را از آن داریم.

سپس با استفاده از این مقادیر برای هر ناحیه به دست آمده، آن را از فیچر مپ اولیه جدا می‌کنیم و از لایه‌ی ROI pooling layer عبور می‌دهیم. دقت شود در این قسمت برعکس Fast RCNN، pyramid pooling نداریم و هدف این لایه مجدد به این شکل است که از نواحی با سایزهای متفاوت در فیچر مپ ما را به آرایه‌ای با سایز یکسان از فیچر مپ برساند تا بتوانیم برای تمامی نواحی در مرحله‌ی از بعد از شبکه‌ی یکسانی استفاده کنیم.

در انتها خروجی ROI به دست آمده را به چند fully connected layer می‌دهیم و دو خروجی کلاس و رگرسیون bounding box را از کل شبکه می‌گیریم.

**مقایسه:** روش RCNN از لحاظ زمانی بسیار وقت‌گیر است، چرا که باید تک به تک عکس‌ها را از شبکه‌ی کانوولوشنالی عبور دهیم و سپس تصمیم بگیریم کدام ناحیه چه ویژگی‌هایی دارد.

روش Fast RCNN این مشکل را با تنها عبور دادن یک بار عکس ورودی از شبکه‌ی کانوولوشنال تا حدی بهتر کرده و باعث سریعتر شدن شبکه و همچنین کمتر شدن خطای آن می‌شود.

روش Faster RCNN حتی قسمت تشخیص نواحی را که قسمتی وقتگیر برای دو روش قبل بوده را نیز با یک شبکه‌ی RPN جایگزین می‌کند و هم دقت و هم سرعت را تا حد زیادی بالا می‌برد.

**کاربرد هر بخش:**

لایه‌ی کانوولوشن به ما فیچر مپ لازم برای تشخیص نواحی محتمل و همچنین به دست آوردن خروجی نهایی را می‌دهد.

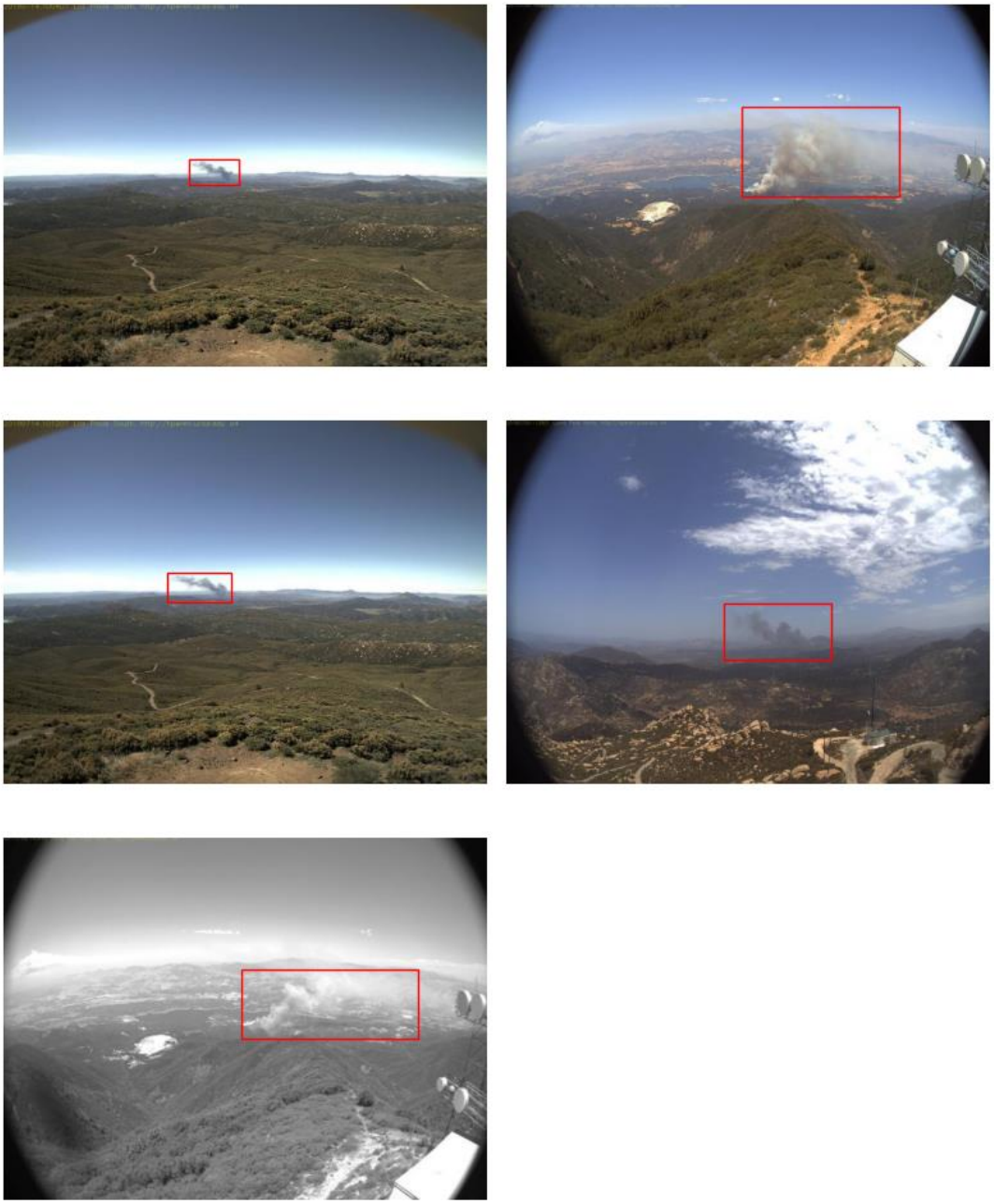
RPN به ما یک سری از نواحی محتمل را به شکل مختصات و کلاس و ضریب اطمینان آن‌ها می‌دهد.

ROI برای آن است که پس از جدا کردن نواحی منتخب RPN از فیچر مپ، آن‌ها را به بهترین نحو به سایزی مشخص از داده‌ها تبدیل کنیم.

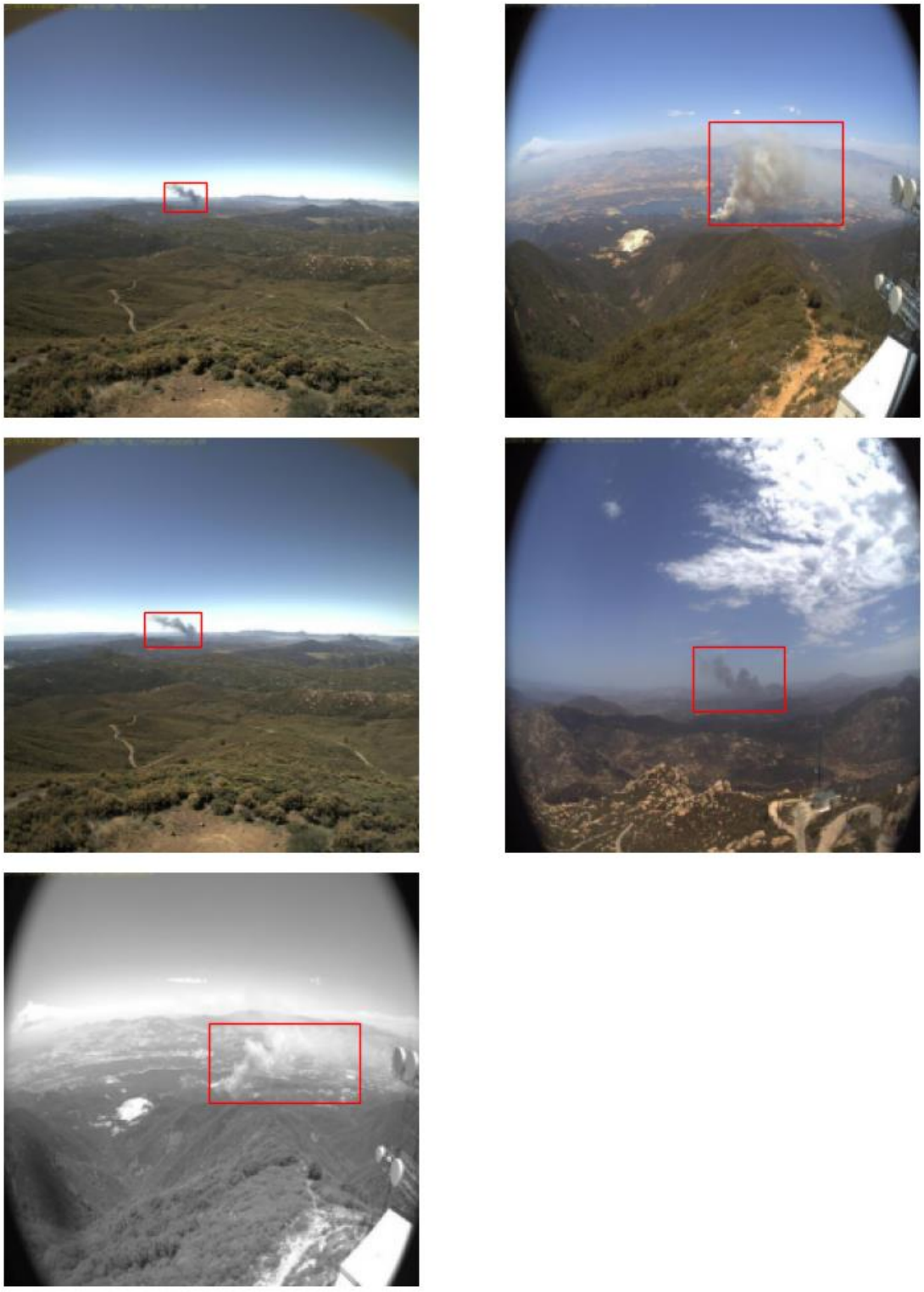
Classification برای آن است که در نهایت اعداد خروجی ROI را از شبکه‌ای بگذرانیم و بررسی کنیم که مربوط به کدام یک از کلاس‌های تعریف شده می‌باشد. (پس زمینه، کلاس 1، کلاس 2 و ...)

Regression برای آن است که در نهایت اعداد خروجی ROI را از شبکه‌ای بگذرانیم و مختصات دقیق‌تری از آنچه توسط RPN به دست آمده را برای مستطیل نهایی به دست آوریم.

## 2-2. پیش پردازش



**خواندن تصاویر و bounding box آن‌ها و نمایش 5 نمونه**



تغییر سایز تصاویر و bounding box آن‌ها به سایز لازم برای ورودی vgg16 و نمایش 5 نمونه

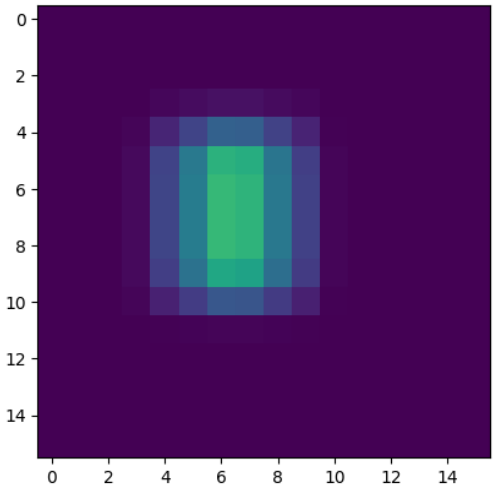
## 3-2. آموزش شبکه

در مرحله‌ی اول نیاز به آموزش شبکه‌ی RPN داریم. برای اینکار لازم است که فرمت لیبل‌های خروجی را به شکلی در بیاوریم که مقاله پیشنهاد داده است.

به این منظور لیبل‌های قسمت کلسیفای کردن با سایز K*H*W می‌باشند، که H و W برابر 16 می‌باشند که سایز فیچر مپ vgg16 هست و K نمایش‌دهنده‌ی تعداد anchorهای در نظر گرفته شده با اسکیل‌های متفاوت است و در این شبکه 9 anchor متفاوت را در نظر گرفته‌ایم. (خود مقاله تعداد سطرها را به صورت 2*K در نظر می‌گیرد که با فرض قرار دادن لایه‌ی سافت مکس است، اینجا به علت وجود دو کلاس پس زمینه و دود تنها K در نظر می‌گیریم که برای کلسیفای کردن 2 کلاسه کافی است.) مقادیر موجود در این لیبل با استفاده از IOU به دست آمده با bounding box واقعی از اسلاید دادن مستطیل‌های anchor روی تصویر اصلی به دست آمده است.

```
[[ 56.  56.]
 [ 56.  84.]
 [ 56. 102.]
 [ 84.  56.]
 [ 84.  84.]
 [ 84. 102.]
 [102.  56.]
 [102.  84.]
 [102. 102.]]
```

**طول و عرض هر anchor در نظر گرفته شده**




**نمونه احتمالات(iou) یک anchor برای یک عکس خاص به ازای مکان‌های متفاوت**

لیبل‌های قسمت رگرسیون خطی با سایز 4K*H*W می‌باشند، در واقع برای هر anchor ما 4 عدد به دست می‌آوریم که به ترتیب داری مقادیر مرکز bounding box واقعی، و 1.2 برابر اسکیل‌های bounding

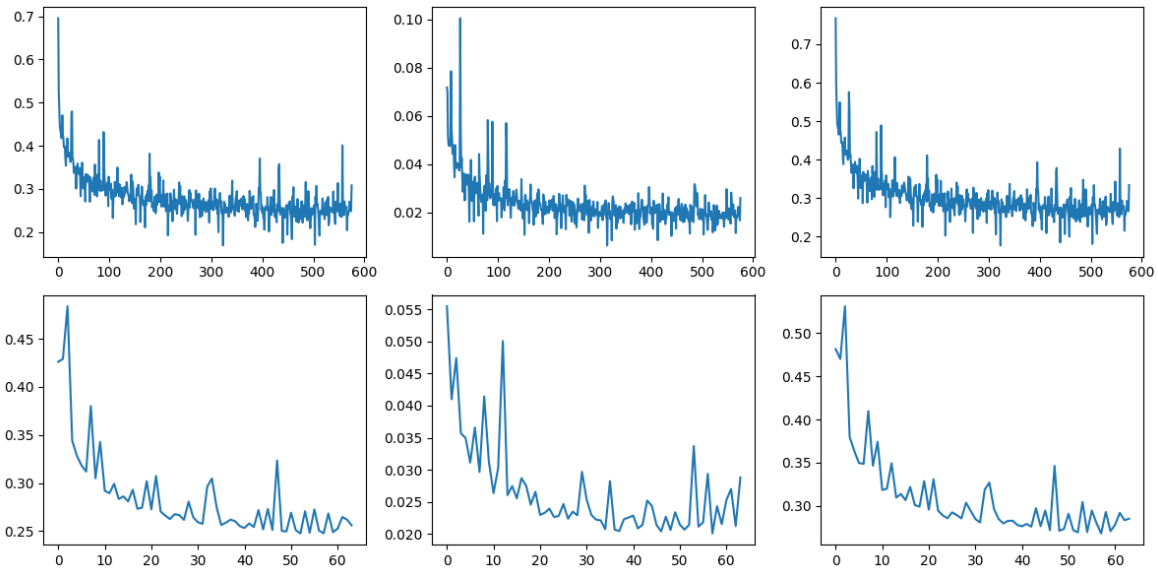
boxهای واقعی می‌باشند. (علت بزرگتر گرفتن سایز این است که ناحیه‌ی به دست آمده قرار است ناحیه‌ی پیشنهادی برای جستجو باشد و باید شکل را به طور خوبی در برگرفته باشد تا از خطا جلوگیری کنیم.)

برای آموزش شبکه، دو loss تعریف می‌کنیم. Loss اول مربوط به کلسیفایکیشن می‌باشد و از نوع BCE می‌باشد. Loss دوم از نوع SmoothL1 می‌باشد که خود مقاله‌ی اصلی گفته است. در نهایت Loss کل را به صورت زیر تعریف می‌کنیم:

$$L\_tot = L\_cls + gamma * L\_reg$$

که مقدار گاما پارامتری آزاد است و در اینجا آن را برابر 1 قرار می‌دهیم.

شبکه را با اپتیمایزر SGD و لرنینگ ریت 1.5، batch_size=32 و تعداد epoch=64 آموزش می‌دهیم. (این پارامترها به صورت تجربی به دست آمدند تا به بهترین نتیجه برسیم، به طور مثال به ازای لرنینگ ریت‌های بالاتر مقادیر loss به بینهایت می‌رسید.



**توابع هزینه: سطر اول برای ترین و سطر دوم برای تست**

**ستون اول برای Loss_tot، ستون دوم برای Loss_reg و ستون سوم برای Loss_cls**

پس از انجام آموزش RPN به سراغ انتخاب نواحی منتخب می‌رویم و با استفاده از تابع زیر این نواحی را از روی فیچر مپ جدا می‌کنیم.

این کار توسط فانکشن ROI_Pooling انجام می‌شود.

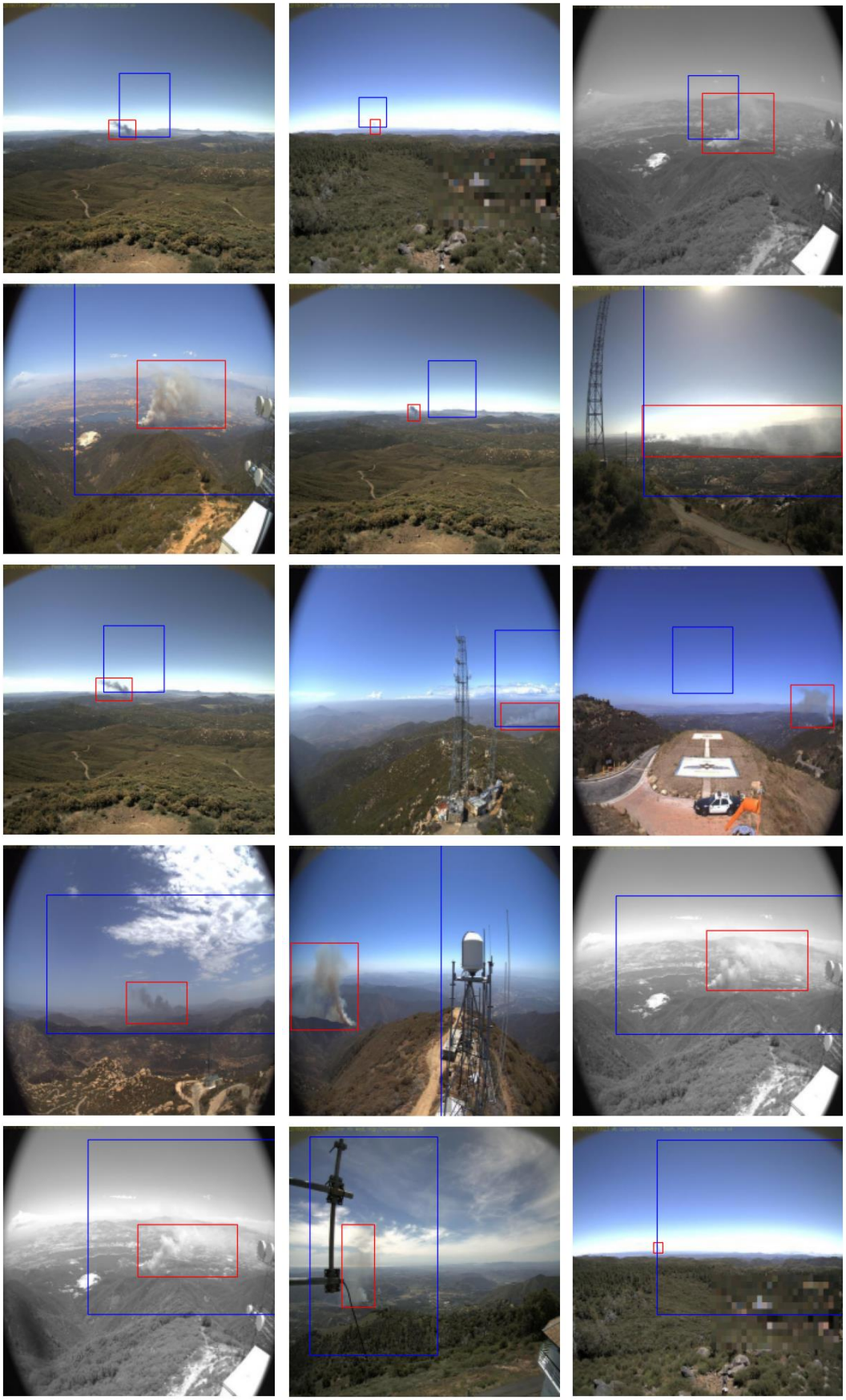
## 4-2. بررسی داده های تست

در این قسمت خروجی‌های به دست آمده توسط RPN را برای داده‌های train، test و evaluation نمایش می‌دهیم. شکل‌های زیر دارای مستطیل قرمز به معنای bounding boxهای واقعی و مستطیل آبی به معنای bounding boxهای به دست آمده می‌باشد.(در صفحه‌ی بعد)

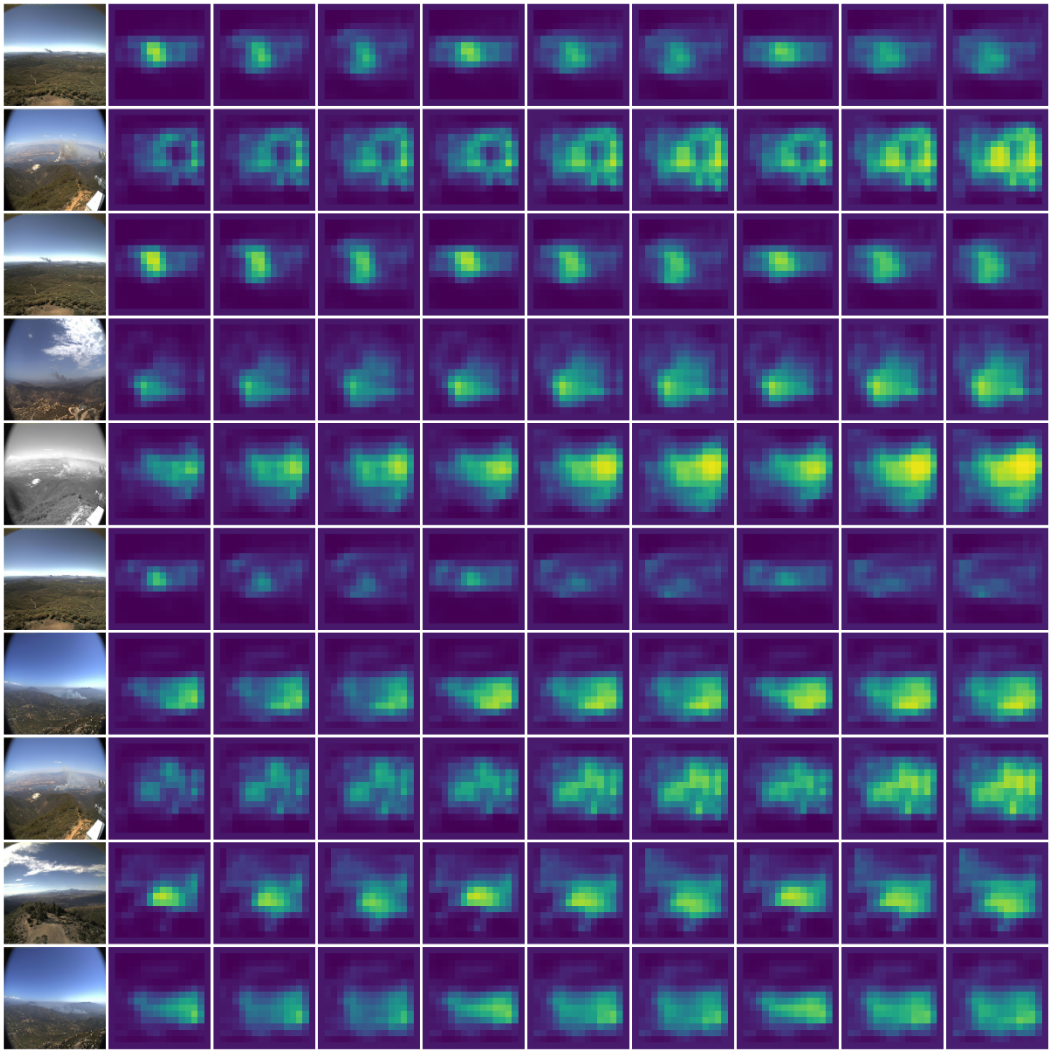
به نظر می‌رسد به داده‌های train بیشتری برای دقت بیشتر شبکه نیاز داریم، همچنین برای رسیدن برای دقت‌های بالاتر می‌توان از تعداد anchorهای بیشتر بهره برد، وضوح فیچر مپ را بالاتر برد (با تغییر backbone برای رسیدن به وضوح بالاتر در فیچر مپ) و یا حتی معماری شبکه را تغییر داد.

قسمت‌های زیادی در این شبکه به طور کلی وجود دارد که وجوب حضور آن‌ها مبهم است، مثلا اینکه چرا دو مرحله بودن برای رسیدن به جواب مناسب است.

همچنین پیچیدگی طراحی شبکه بیش از حد است و دیباگ کردن قسمت‌های مختلف بسیار مشکل می‌باشد و تعیین تأثیر تغییر هایپرپارامترها سخت می‌باشد.



خروجی rpn به ترتیب از چپ به راست برای داده‌های train، evaluation و test



هیت مپ y_cls برای چند عکس از داده‌های ترین