



دانشگاه تهران
دانشکده مهندسی برق و
کامپیوتر



درس شبکه‌های عصبی و یادگیری عمیق
تمرین سوم

محمد رضا محمد هاشمی - 810100206

مبینا مهرآذر - 810100216

فهرست

1.....	پرسش 1. سگمنتیشن تومور مغزی از روی تصاویر MRI.....
1.....	1-1. توصیف مدل ارائه شده.....
1.....	1-1-1. معماری VGG.....
1.....	1-1-2. معماری VGG16.....
4.....	1-2. آمادهسازی مجموعه داده.....
9.....	1-3. تقویت داده.....
10.....	1-4. بهینهساز، معیارها وتابع هزینه.....
11.....	1-4-1. معیار IoU Score.....
12.....	1-4-2. معیار Dice Coefficient.....
13.....	1-4-3. بهینهساز.....
14.....	1-4-4. توابع هزینه.....
15.....	1-5. پیادهسازی مدل.....
15.....	1-6. آموزش مدل.....
16.....	1-7. ارزیابی مدل.....
20.....	پرسش 2 - تشخیص تابلو های راهنمایی و رانندگی
20.....	2-1. آمادهسازی مجموعه داده.....
20.....	2-1-1. آشنایی با مجموعه داده.....
21.....	2-1-2. بررسی مجموعه داده در مقاله.....
23.....	2-1-3. تقسیم مجموعه داده.....
25.....	2-2. تنظیم دقیق و ارزیابی مدل تشخیص شی دو مرحله ای.....
25.....	2-2-1. توضیح مختصر در مورد مدل Faster R-CNN با شبکه پشتیبان ResNet50-FPN.....
26.....	2-2-2. اقدامات لازم برای آمادهسازی مدل و تنظیم آن.....
27.....	2-2-3. آمادهسازی و نرم افزاری مجموعه داده ورودی مدل.....
29.....	2-2-4. معیارهای ارزیابی مدل.....
30.....	2-2-5. بهینهساز وتابع هزینه.....
31.....	2-2-6. ارزیابی مدل.....
32.....	2-2-7. نمودار AP به ازای IoU های متفاوت.....
32.....	2-2-8. ارزیابی مدل روی اشیا با اندازه های متفاوت.....
33.....	2-2-9. نتایج پیش‌بینی یک نمونه تصویر از داده ارزیابی.....
35.....	2-3. تنظیم دقیق و ارزیابی مدل تشخیص شی تک مرحله ای.....

35.....	2-3-1. توضیح مختصر در مورد مدل SSD300 با شبکه پشتیبان VGG16
36.....	2-3-2. اقدامات لازم برای آماده‌سازی مدل و تنظیم آن
36.....	2-3-3. آماده‌سازی و نرم‌السازی مجموعه داده ورودی مدل
36.....	2-3-4. معیارهای ارزیابی مدل
36.....	2-3-5. بهینه‌ساز و تابع هزینه
38.....	2-3-6. ارزیابی مدل
38.....	2-3-7. نمودار AP به ازای 100 های متفاوت
39.....	2-3-8. ارزیابی مدل روی اشیا با اندازه‌های متفاوت
40.....	2-3-9. نتایج پیش‌بینی یک نمونه تصویر از داده‌ی ارزیابی
41.....	2-4. ارزیابی نتایج و مقایسه مدل‌ها

پرسش 1. سگمنتیشن تومور مغزی از روی تصاویر MRI

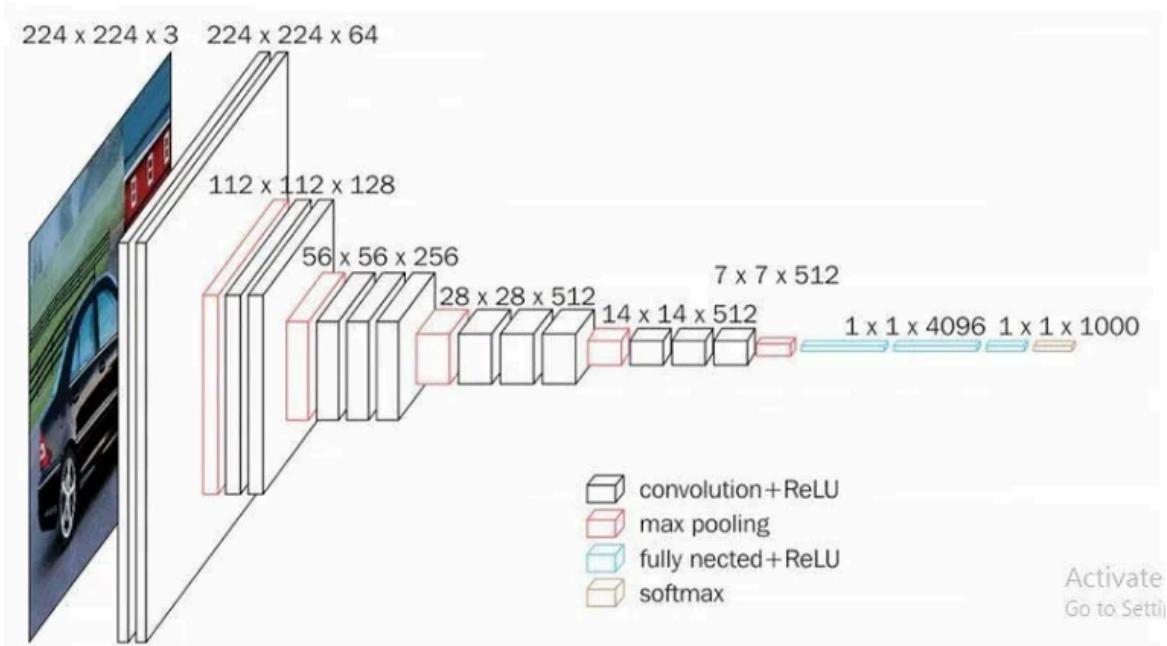
1-1. توصیف مدل ارائه شده

1-1-1. معماری VGG

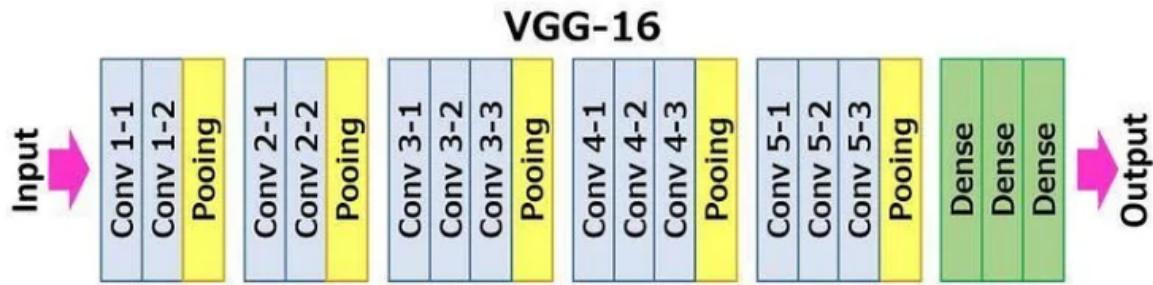
معماری VGG یک مدل کانولوشنال است که به نام دیگر ConvNet هم معروف می‌باشد. این مدل مانند بقیه مدل‌های کانولوشنال، یک لایه ورودی، یک لایه خروجی، و چندین لایه نهان یا میانی می‌باشد. این مدل یکی از بهترین مدل‌های Computer Vision محسوب می‌شود. طرز کار مدل به این شکل است که عمق تصاویر ورودی را به کمک فیلترهای کوچک به سایز (3×3) افزایش می‌دهد. این مدل معماری ساده‌ای دارد.

2-1-1. معماری VGG16

در این مسئله از مدل VGG16 با روش transfer learning برای سگمنتیشن تومور مغزی از روی تصاویر MRI استفاده می‌کنیم.



شکل 1. پارامترهای هر لایه در مدل VGG16



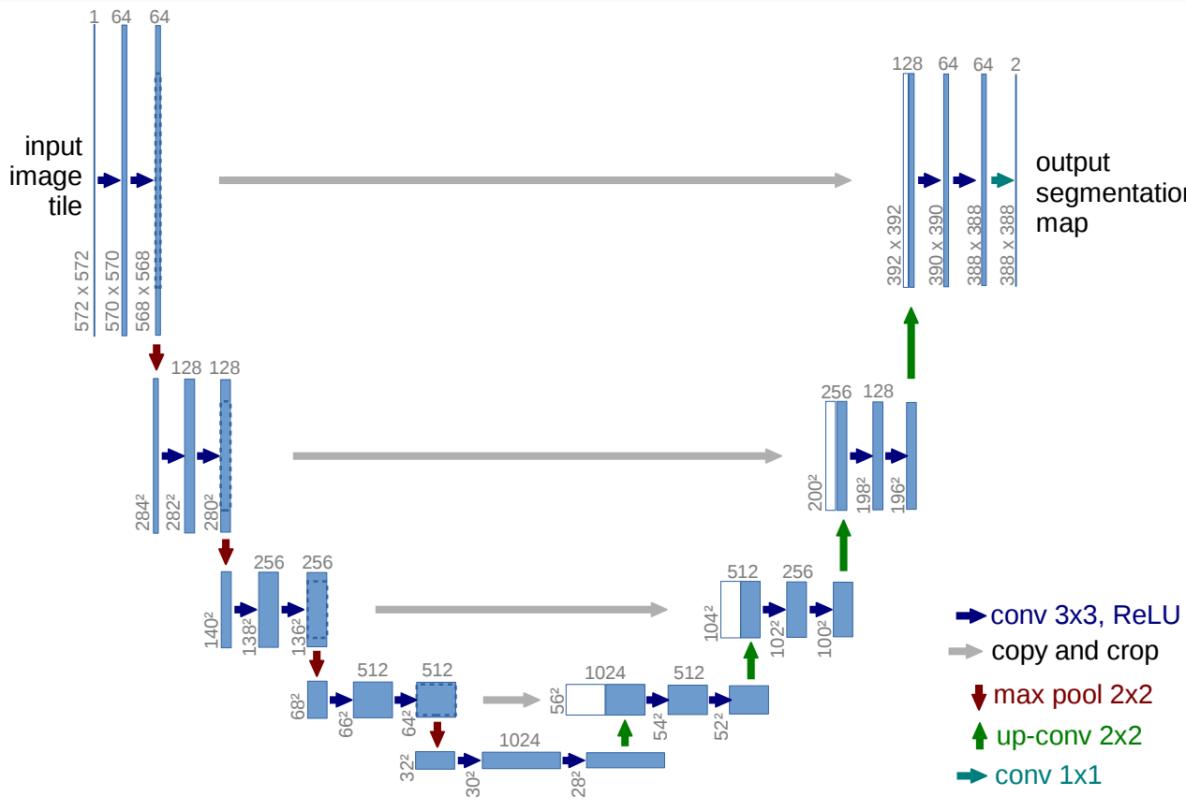
شکل ۱. لایه‌های مدل VGG16

حال به بررسی چند نکته در مورد این مدل می‌پردازیم:

- این مدل 16 لایه با وزن‌های قابل یادگیری دارد که در این مسئله از همین وزن‌ها استفاده شده است. به طور واضح‌تر تمامی لایه‌های دارای وزن‌های قابل یادگیری و بدون وزن قابل یادگیری مدل به شرح زیر است:
 - 13 لایه کانولوشنال،
 - 5 لایه مکس پول،
 - 3 لایه دنس
- مدل ورودی را در قالب تنسور هایی به اندازه 224 در 224 و با 3 کanal از نوع RGB می‌گیرد.
- مزیت این مدل، کم بودن تعداد هایپر پارامتر های آن است. این مدل مخصوصاً نوع فیلترهای کانولوشنال خود را به اندازه 3x3 با stride 1 تعیین کرده است. لایه مکس پول آن فیلتر 2x2 با stride 2 به اندازه 2 می‌باشد.
- از لایه‌های کانولوشن و مکس پولینگ مکررا در معماری استفاده شده است.
- تعداد فیلترهای لایه های کانولوشن مدل به شرح زیر است:
 - در اولین لایه 64 فیلتر،
 - در دومین لایه 128 فیلتر،
 - در سومین لایه 256 فیلتر،
 - در چهارمین لایه 512 فیلتر
- همانطور که در شکل مشخص است، سه لایه fully connected به همراه توابع فعال‌سازی به نام Relu پشت سر هم قرار دارد که دو لایه اول 4096 کanal دارد. سومین و آخرین لایه 1000 کanal دارد. در نهایت از یک لایه سافت مکس در این معماری استفاده شده است.

3-1-1. UNet معماری

معماری اصلی مدل به شکل زیر است:



شکل 1. ساختار مدل UNet

این مدل یک معماری U شکل از 4 بلوک encoder و به دنبال آن 4 بلوک decoder دارد که این دو دسته بلوک به کمک پلی به هم وصل هستند و داده ورودی از encoder به decoder منتهی می‌شود. همانطور که در شکل نمایش داده شده است. این مدل معماری تمام کانولوشنال دارد و در آن از لایه‌های dense یا flatten و یا لایه‌های مشابه استفاده نشده است.

این مدل به خودی خود دارای 31031685 پارامتر قابل یادگیری است که عدد بزرگی محسوب می‌شود. در این تمرین ما به جای بخش encode این مدل که در مقاله‌ی آن آورده شده، از VGG16 برای استخراج ویژگی‌های تصاویر ورودی استفاده شده است. با این کار و با freeze کردن بخش انکودر مدل، تعداد پارامترهای قابل یادگیری به عدد 2324353 کاهش می‌یابد.

مدل نهایی به صورت فایل model.gv پیوست شده قابل ارائه است.

این مدل واصلهایی از سمت encoder به سمت decoder پیاده‌سازی می‌کند که connection نام دارد. این واصلها اطلاعات مازاد بر خروجی decoder را در اختیار هر بلوک semantic feature concat شدن این اطلاعات، به تولید semantic feature های بهتر کمک می‌کند.^۱.

پل ارتباطی بین encoder و decoder تعریف می‌شود که داده‌ها پس از انکود شدن مراحل دیکدینگ را طی کنند و نتیجه نهایی حاصل شود.

بخش decoder مدل متناظر با encoder از جنس VGG16 طراحی شده که از روی بازنمایی انتزاعی^۲ حاصل، یک ماسک semantic segmentation از نوع باینری تولید می‌کند.

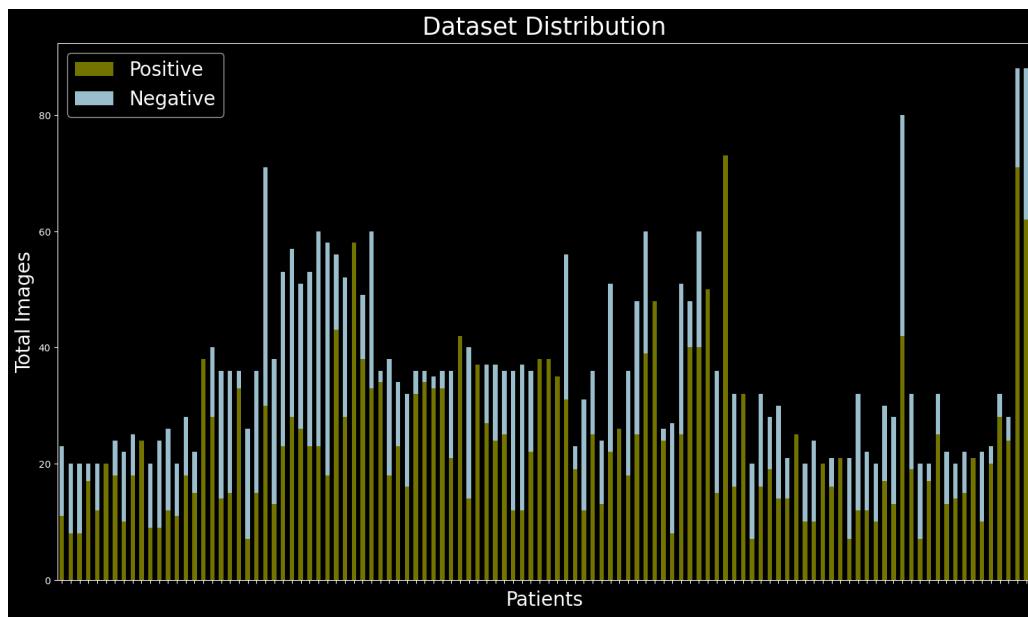
شکل ۱. ساختار لایه‌های مدل استفاده شده در این مسئله

۲-۱. آماده‌سازی مجموعه داده

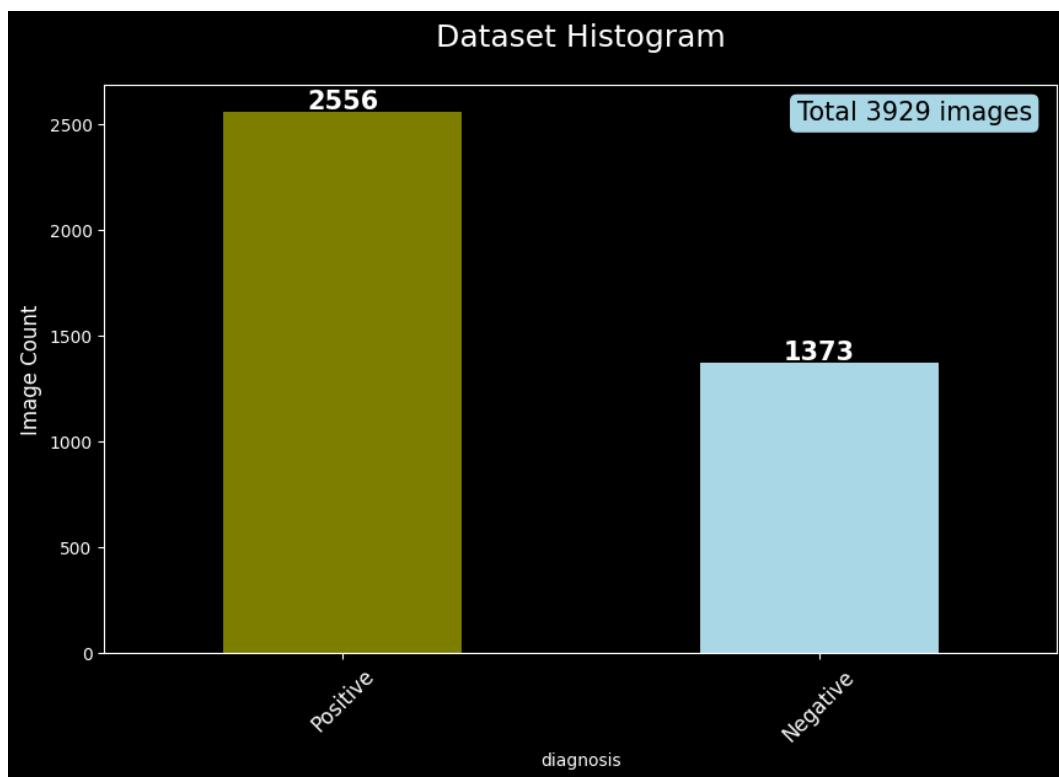
این مجموعه تصاویر را به طور مستقیم در dataframe ذخیره نمی‌کنیم، چرا که حجم رم محدود است. تنها آدرس‌های مربوطه را ذخیره کرده و در موارد نیاز، لود می‌کنیم. این مجموعه داده متعلق به ۱۱۰ بیمار بوده و شامل تصاویر از مغز هر بیمار به همراه ماسک تشخیص ناهنجاری در مغز او می‌باشد. توجه کنید اگر در مغز بیمار تومور نباشد، این ماسک به صورت تصویر تمام مشکی با مقادیر تمامی پیکسل ها به اندازه ۰ ذخیره شده است. همینطور ماسک برای تصاویر مغز دارای تومور، شامل یک عکس باینری بوده تمام ۰ بوده که در بخش‌هایی که تومور است، مقدار پیکسل ۱ بوده و به رنگ سفید در می‌آید.

^۱ skip connection helps in better flow of gradient while backpropagation, which in turn helps the network to learn better representation.

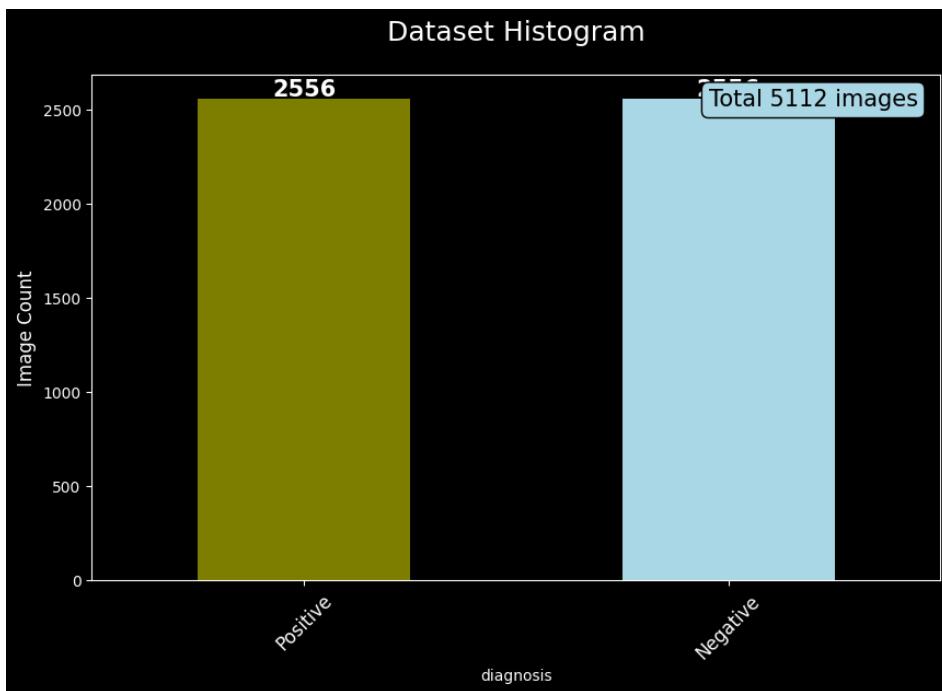
^۲ abstract representation



شكل 1. توزيع مجموعه داده اصلی

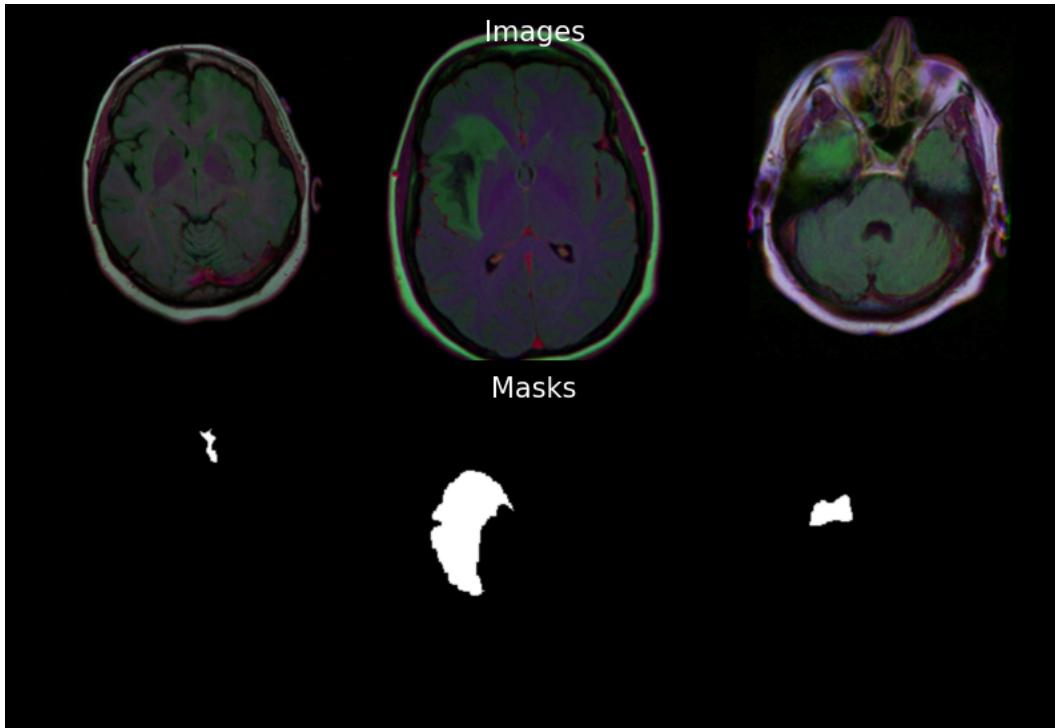


شكل 1. هیستوگرام مجموعه داده قبل از داده افزایی



شکل 2. هیستوگرام مجموعه داده بعد از داده افزایی

همینطور برای آشنایی بیشتر با مجموعه داده، چند نمونه از تصاویر به همراه ماسک متناظرشان را نمایش می‌دهیم:



شکل 2. تصاویر و ماسک متناظر

تصاویر MRI به ابعاد $3 \times 256 \times 256$ بوده که مقدار 256 ابعاد طول و عرض تصویر و مقدار 3 مربوط به تعداد کانال های رنگی به صورت Red Green Blue می باشد. تصاویر Mask هم به صورت $1 \times 256 \times 256$ بوده که مقدار یک مربوط به سیاه و سفید بودن تصویر می باشد.

این تصاویر را بررسی کرده و با توجه به تمام مشکی بودن یا نبودن مسک تصاویر MRI، پوزیتیو یا نگاتیو بودن آن را تعیین گردہ ایم. به عبارتی، ستون diagnosis دیتا فریم تصاویر دارای تومور مغزی 1 و مابقی تصاویر 0 در نظر گرفته شده است. برای این مجموعه داده کلاس 128 \times 128 \times 128 را تعریف کرده ایم که در متدهای `getitem` آن، سایز تصاویر مسک را به تغییر داده ایم.

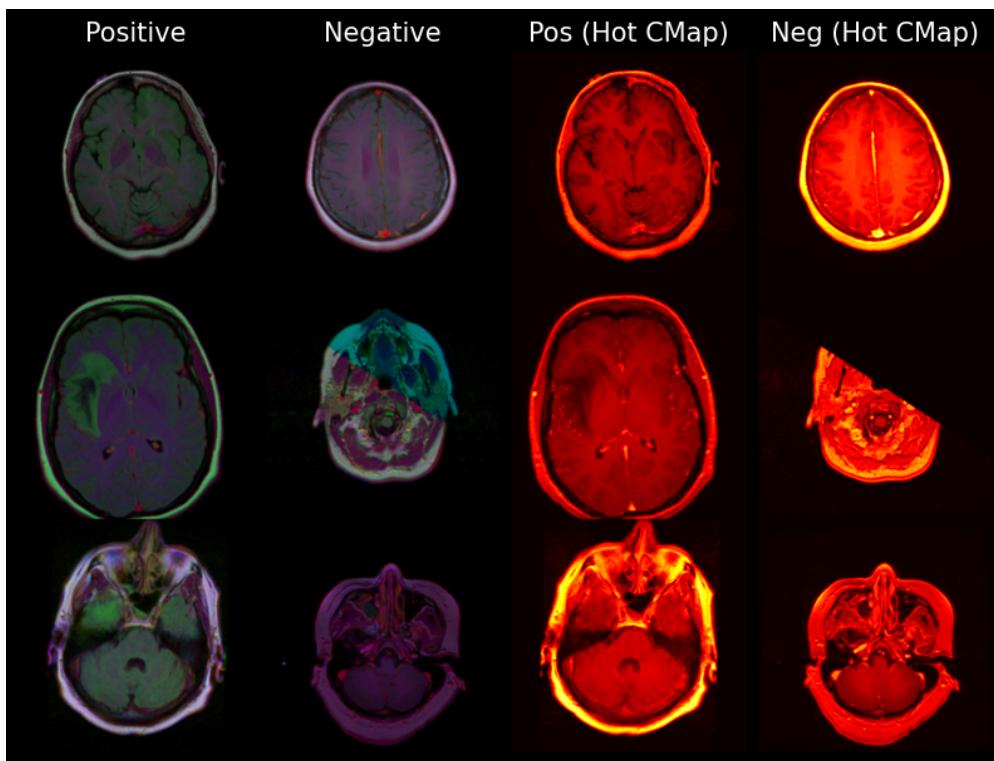
توجه کنید تصاویر را همانند مقاله مربوطه نرمالایز کرده ایم.

$$X_{new} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

از آنجایی که مجموعه داده بالانس نیست و این می تواند در عملکرد مدل اثر منفی بگذارد، این مجموعه داده را به کمک up sampling بالانس کرده و سپس آن را به سه دسته `train`, `test` و `validation` با نسبت های زیر تقسیم کرده ایم:

Dataset	Shape
Train	(4089, 4)
Validation	(511, 4)
Test	(511, 4)

همینطور برای آشنایی بیشتر با تصاویر، آنها را به صورت اولیه و همینطور با CMap از نوع Hot نمایش می‌دهیم. در این بررسی مشاهده می‌کنیم تصاویر دارای تومور مغزی در حالت Hot، محل تومور قابل شناسایی بوده و به رنگی متمایز و قابل تشخیص نمایش داده شده است. توجه کنید برای یادگیری مدل از CMap اولیه تصاویر استفاده خواهد شد.



شکل 2. بررسی تصاویر MRI دارای تومور و بدون تومور در حالت Hot CMap

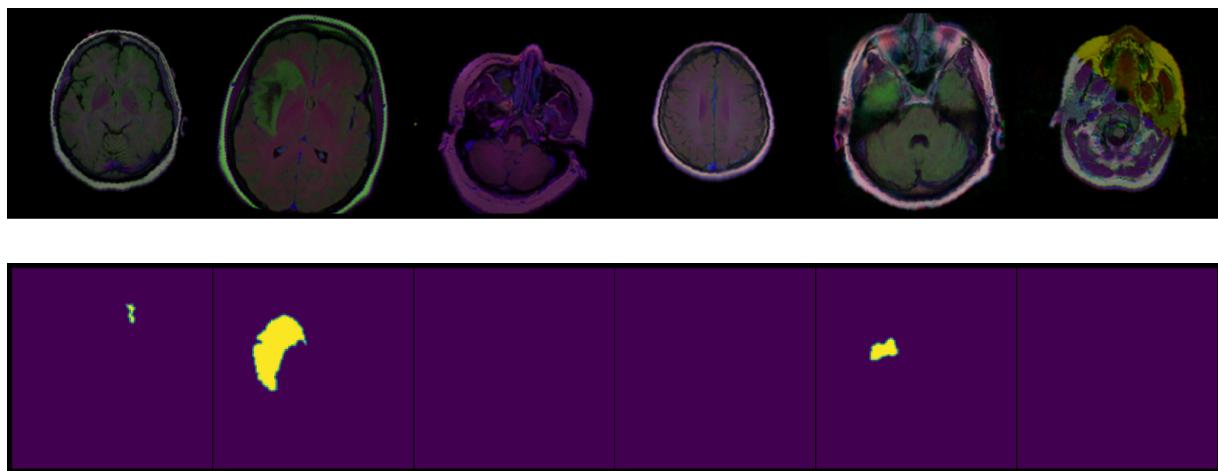
3-1. تقویت داده

در این مسئله از داده‌افزایی برای تغییر تصاویر به صورت جزئی، به منظور افزایش تنوع در مجموعه داده و کمک به یادگیری هر چه بهتر مدل استفاده می‌شود. بدین طریق مدل ما در برخورد با تصاویر و تشخیص ماسک متناظر، مقاومت بیشتر و عملکرد بهتری خواهد داشت. در این بخش میزان پارامترهای تقویت داده را به مقدار جزئی تغییر داده ایم.

با تعریف کلاس AugmentPair و پاس دادن آن به عنوان transformer به کلاس دیتاست، از تکنیک‌های augmentation مانند چرخش، تغییر مقیاس، تغییرات شدت روشنایی با پارامترهای تصادفی انتخاب شده از توزیع uniform در محدوده تعیین شده بهره می‌بریم.

Data Augmentation Method	Parameter(s)
Rotation	15
Scale	0.2
Brightness	(0.9, 1.1)

در شکل زیر چند تصویر تقویت شده به کمک ماسک‌های مربوطه نمایش داده شده است:



شکل ۱. سه نمونه از تصاویر تقویت شده به همراه ماسک متناظر

4-1. بهینه‌ساز، معیارها وتابع هزینه

در این مسئله، از دو معیار رایج ارزیابی بخش‌بندی استفاده کرده‌ایم. این معیارها به منظور ارزیابی میزان تطابق نتایج تشخیص و واقعیت استفاده می‌شوند. به علت ماهیت این متريک‌ها برای جلوگیری رخدان تقسیم بر صفر تعریف نشده، اى متغير smooth به مقدار خیلی کوچک 10^{-6} استفاده شده است.

در اين بخش به توضیح چندین متريک استفاده شده در تشخیص دسته‌بندی و ارزیابی عملکرد مدل‌های یادگیری ماشین می‌پردازیم:

- تشخیص صحیح مثبت، TP

یا همان True Positive، تعداد نمونه‌هایی که به درستی تشخیص داده شده‌اند که در واقعیت مثبت هم هستند، می‌باشد.

- **TN** تشخیص صحیح منفی،

یا همان True Negative، تعداد نمونه‌هایی که به درستی تشخیص داده شده‌اند که در واقعیت مثبت هستند، می‌باشد.

- **FP** تشخیص غلط مثبت،

یا همان False Positive، تعداد نمونه‌هایی که به غلط به عنوان اعضای مثبت تشخیص داده شده‌اند که در واقعیت منفی هستند، می‌باشد.

- **FN** تشخیص غلط منفی،

یا همان False Negative، تعداد نمونه‌هایی که به غلط به عنوان اعضای منفی تشخیص داده شده‌اند که در واقعیت مثبت هم هستند، می‌باشد.

1-4-1. معیار IoU Score

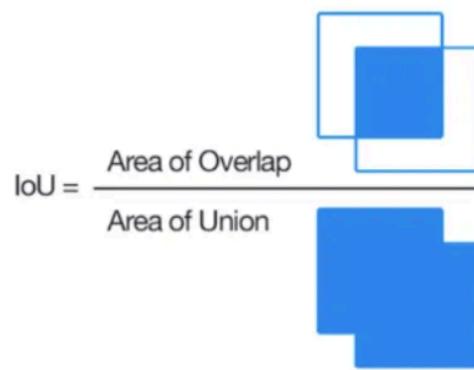
این معیار میزان همپوشانی بین ناحیه تشخیص شده و ناحیه واقعی را می‌سنجد. در ابتدا ناحیه اشتراک بین دو مورد را محاسبه کرده و سپس نسبت این ناحیه به دست آمده به مجموع دو ناحیه گزارش می‌شود.

$$IoU = \frac{\text{Intersection Area}}{\text{Union Area}} = \frac{|A \cap B|}{|A \cup B|}$$

به عبارتی این فرمول تعبیری به کمک معیارهای ارزیابی دسته بندی دارد که به شرح زیر است:

$$IoU = \frac{\text{Intersection Area}}{\text{Union Area}} = \frac{TP}{TP+FP+FN}$$

بدیهی است که این متريک مقداری بین 0 و 1 خواهد داشت و ما به دنبال ماکسیمایز کردن اين متريک هستيم.



2-4-1. معيار Dice Coefficient

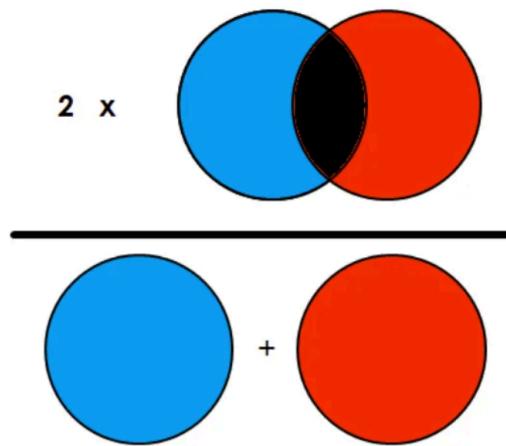
این معيار ميزان همپوشانی بین ناحيه تشخيص داده شده و ناحيه واقعی را اندازهگيري میکند. در اين معيار تنها مساحت مشترك بین اين دو ناحيه بررسی شده و مساحت کلي دو بخش اهميتي ندارد.

تفاوت اين متريک با متريک قبلی اين است که متريک Dice بر هم پوشانی بین بخش پيشбинی شده و بخش واقعی حساسیت بیشتری به خرج می‌دهد. اين یعنی حساسیت بیشتر نسبت به عدم تطابق های کوچک بین اين دو بخش.

$$Dice Coefficient = \frac{2 \times |A \cap B|}{|A| + |B|}$$

$$Dice Coefficient = \frac{2 \times TP}{2 \times TP + FP + FN}$$

اين متريک هم مقداری بين 0 و 1 خواهد داشت و ما به دنبال ماکسيمايز کردن اين متريک هستيم.



3-4-1. بهینه‌ساز

در این مسئله از اپتیمایزر Adam استفاده شده است. این اپتیمایزر یک الگوریتم بهینه‌سازی پر کاربردی دارد که برای به روز رسانی وزن‌های شبکه‌های عصبی در فرآیند یادگیری شبکه استفاده می‌شود. این الگوریتم از ترکیب روش قدیمی‌تر RMSProp و با استفاده از مفهوم Momentum قابلیت تطبیق با نرخ یادگیری مدل را دارد. این الگوریتم تاثیر خوبی روی کیفیت فرآید آموزش و همینطور سرعت فرآید دراد.

برخی از مراحل این الگوریتم به شرح زیر توضیح داده می‌شود:

- محاسبه مقدار گرادیان

گرادیان گیری فرآیندی است از جبر خطی که روی پارامترهای قابل یادگیری مدل صورت می‌گیرد. در نتیجه این فرآیند، جهت و میزان تغییرات وزن‌ها بر اساستابع هزینه مشخص می‌شود. این اطلاعات حاصل در تصمیم‌گیری در به روز رسانی پارامترها در مراحل بعدی نقش بهسزایی دارند.

- تطبیق با نرخ یادگیری

تابع اپتیمایزر Adam نرخ یادگیر پارامترها را منحصر به همان پارامتر تعیین می‌کند. این نرخ در هر مرحله بر اساس میانگین مونتوم اول و میانگین مربعات گرادیان توضیح داده شده در بخش قبل، مشخص می‌شود.

این روش به علت رفتار جداگانه با هر پارامتر، سرعت و عملکرد خوبی در بهینه‌سازی وزن‌ها دارد.

- مقادیر beta 1 و beta 2

این دو نرخ مقادیر کاهش نمایی برای اولین و دومین لحظه‌های گرادیان (گذشته از حال حاضر) هستند. در واقع این دو نرخ تعیین می‌کنند که تاثیر گرادیان‌های گذشته به به روزرسانی‌های کنونی چقدر باشد.

- نرخ یادگیری

نرخ یادگیری یا همان learning rate پارامتری است که میزان نرخ اولیه یادگیری مدل را مشخص می‌کند. این میزان معمولاً در ابتدای یادگیری مقادیر بیشتری داشته و به مرور با یادگیری بیشتر مدل، نرخ کاهش می‌یابد. توجه کنید میزان اولیه این نرخ به خوبی خود ضریب کوچکی است.

- میزان کاهش نرخ یادگیری

از آنجایی که خود مقدار نرخ یادگیری هم به مرور در فرآیند یادگیری کاهش می‌یابد، برای میزان کاهش این نرخ هم پارامتری تعریف شده است. میزان کاهش نرخ یادگیری باید به اندازه‌ای باشد که مدل به خوبی فرآیند یادگیری را سپری کند.

در این مدل، ما پارامترهای مدل و میزان اولیه نرخ یادگیری را به تابع Adam پاس داده‌ایم و بقیه پارامترها مقادیر دیفالت خود را می‌گیرند.

4-4-1. توابع هزینه

تابع هزینه یا Cost Function و یا Loss Function، در این مسئله از جمع دو تابع Binary

Dice loss و cross entropy محاسبه شده است.

Binary Cross Entropy •

این تابع هزینه یک تابع معروف برای مسائل دسته‌بندی دو کلاسه یا همان مسئله‌ی Classification می‌باشد و در شبکه‌های عصبی کاربرد زیادی دارد.

این تابع خطای اساس احتمالات که توسط شبکه پیش‌بینی می‌شوند و واقعیت دسته‌های مربوطه تعیین می‌کند.

این تابع هزینه لیبل واقعی و لیبل پیش‌بینی شده توسط شبکه را بررسی کرده و خطای حاصل را خروجی می‌دهد. با کمینه کردن میزان خطای خروجی این تابع، پیش‌بینی‌ها به واقعیت نزدیک‌تر شده و مدل عملکرد بهتری را نشان خواهد داد.

فرمول تابع خطای BDE به صورت زیر است:

$$BCE(y, \hat{y}) = - (y \cdot \log(\hat{y}) + (1 - y) \cdot \log(1 - \hat{y}))$$

که در این فرمول تابع هزینه، مقدار y همان مقدار واقعی دسته‌ها بوده که به صورت باینری 0 و 1 ای مشخص می‌شوند.

همینطور مقدار \hat{y} همان مقدار پیش‌بینی شده توسط مدل بوده که این مقدار هم باینری است.

Dice Loss •

$$Dice Loss = 1 - Dice Coefficient$$

$$Dice Loss = 1 - \frac{2 \times |A \cap B|}{|A| + |B|}$$

هدف این تابع هزینه مینیمایز کردن اختلاف بین محدوده واقعی با محدوده پیشبینی شده توسط مدل می‌باشد.

همانطور که بدیهی است، مقدار نهایی این خطا بین ۰ و ۱ بوده و ما سعی بر کمینه کردن این خطا و در نتیجه بیشینه کردن ضریب دایس داریم.

5-1. پیاده‌سازی مدل

مدل با مقدار learning rate برابر 3×10^{-4} و اپتیمایزر Adam و

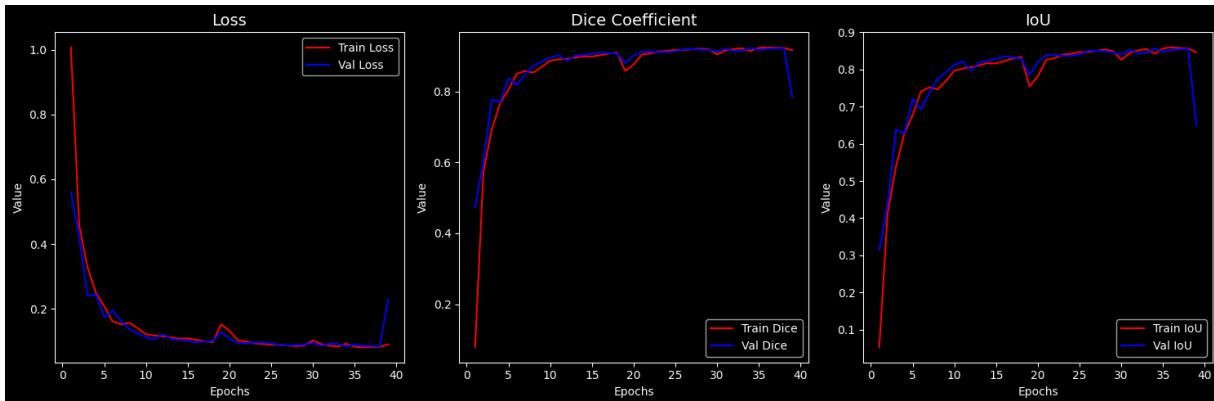
6-1. آموزش مدل

برای آموزش مدل به دست آمده از اپتیمایزر آدام، به روش early stopping به تعداد ایپاک 150 و میزان سایز بج به اندازه 32 بهره می‌بریم.

Epochs	150
Early stopping	True
Optimizer	Adam
Batch size	32
Metrics	IOU Score, Dice Coefficient
Loss Function	Binary Cross Entropy, Dice Loss

برای محاسبه هزینه این تسك باینری سگمنتیشن، از تابع هزینه binary cross entropy استفاده شده است. تابع هزینه Dice Loss به سادگی مقدار زیر را به هزینه کلی اضافه می‌کند:

$$\text{Dice Loss} = 1 - \text{Dice Coefficient}$$



شکل 1. نمودار دقت و خطای مدل

این مدل در حدود 45 اپیاک به نتیجه خوبی رسیده و فرآیند یادگیری را خاتمه می‌دهد. همانطور که مشاهده می‌شود، مدل به خوبی اورفیت نکرده و میزان ضریب دایس ترین و ولیدیشن همواره نزدیکی خوبی داشته‌اند که نشان‌دهنده‌ی عملکرد بهتر مدل می‌باشد. نمودار خطای مدل هم نزول بوده و ابتدا با شبیب بیشتر و به مرور این شبیب نزول کاهش داشته است.

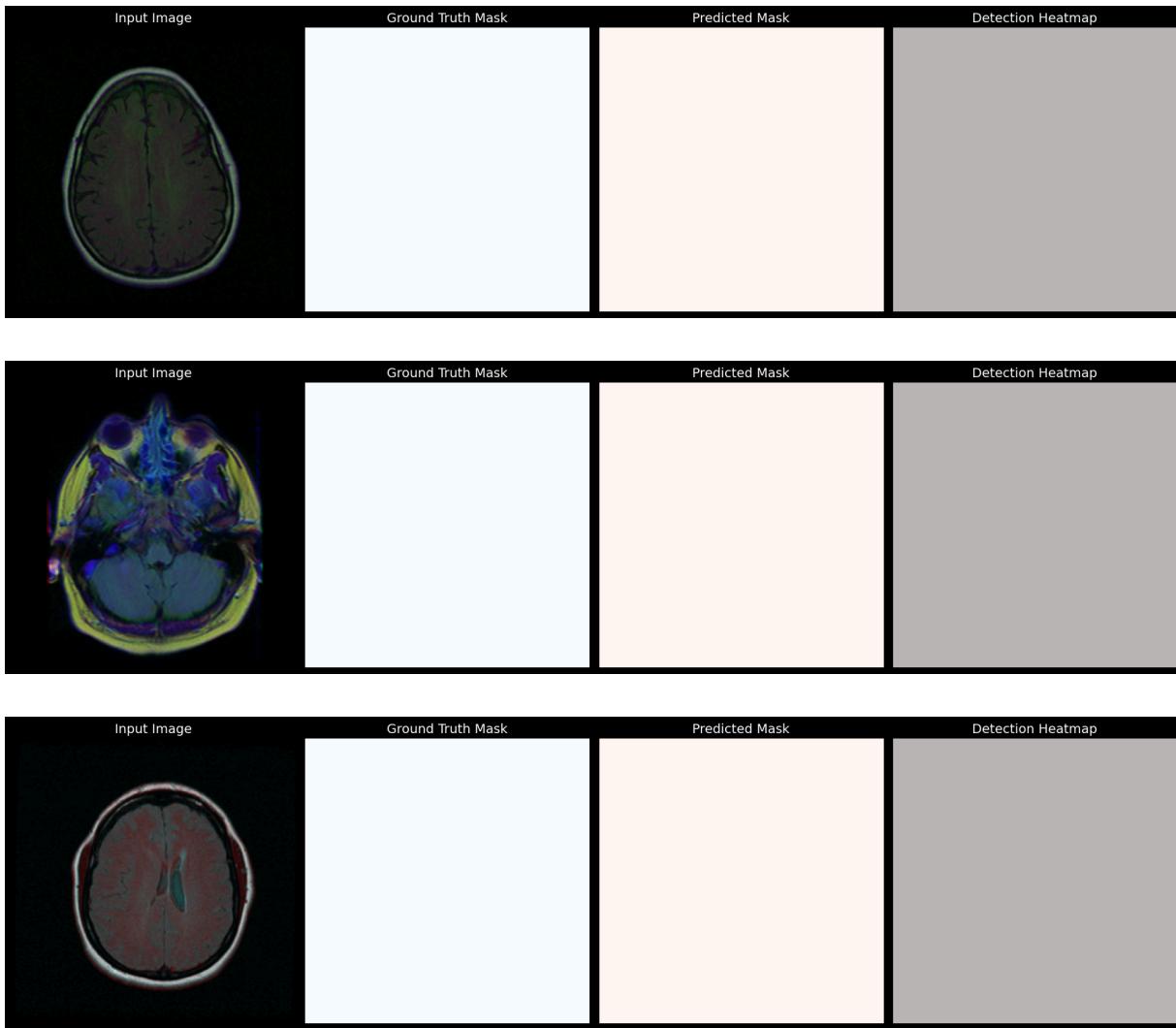
7-1. ارزیابی مدل

نتیجه ارزیابی مدل روی مجموعه داده‌ی تست به شرح زیر است:

|

Metric	Avg Dice	Avg IoU
Test Loader	0.9158	0.8449

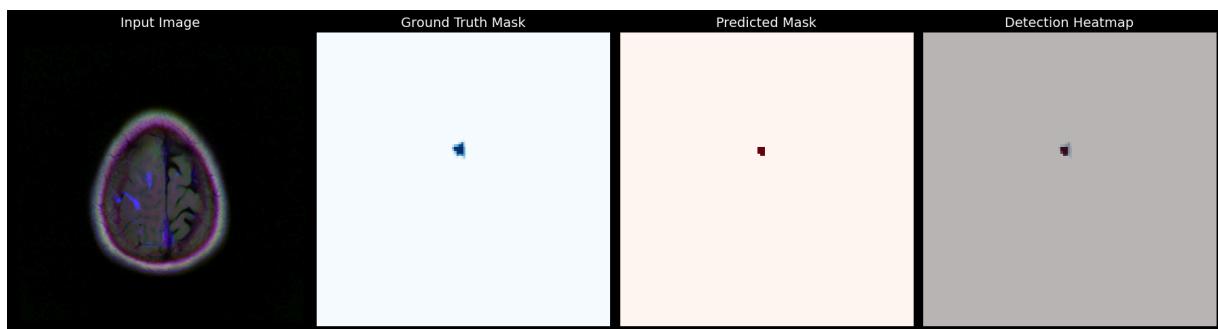
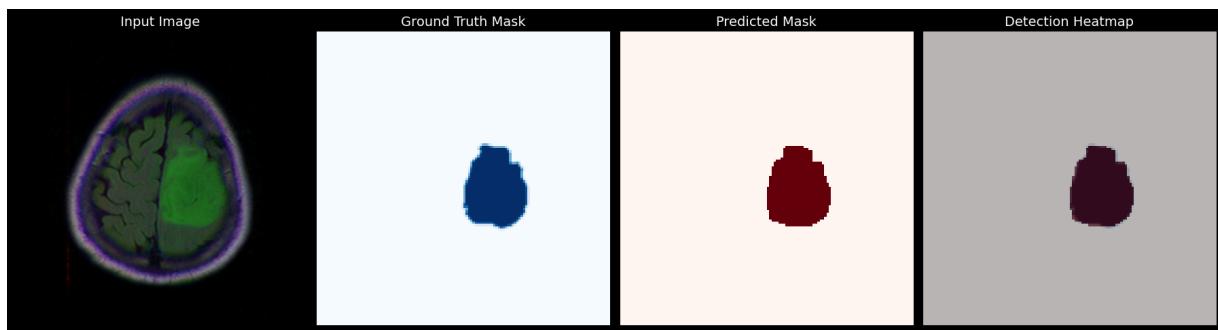
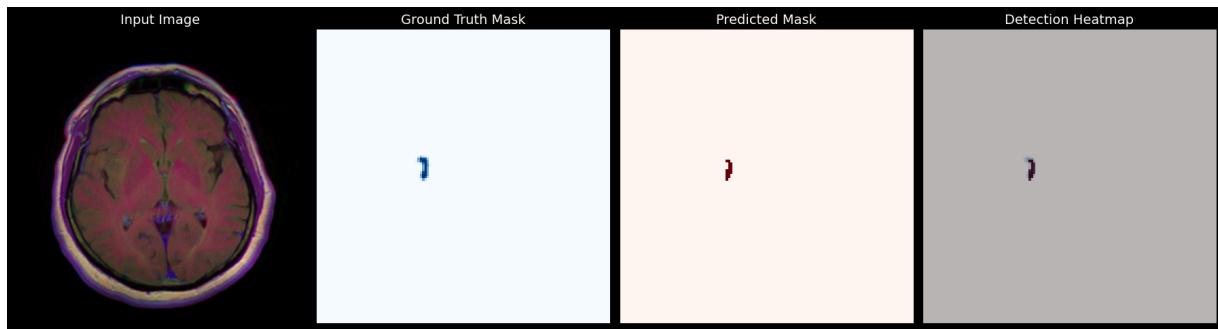
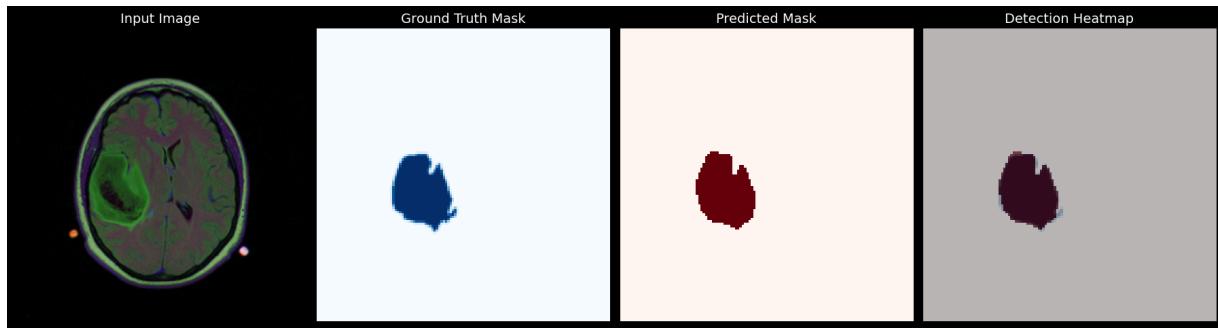
در این بخش چندین تصویر از مجموعه داده تست به همراه نتایج ماسک تشخیص داده شده توسط مدل و ماسک واقعی و همینطور میزان نمایشی از میزان همپوشانی این دو ماسک را بررسی می‌کنیم.

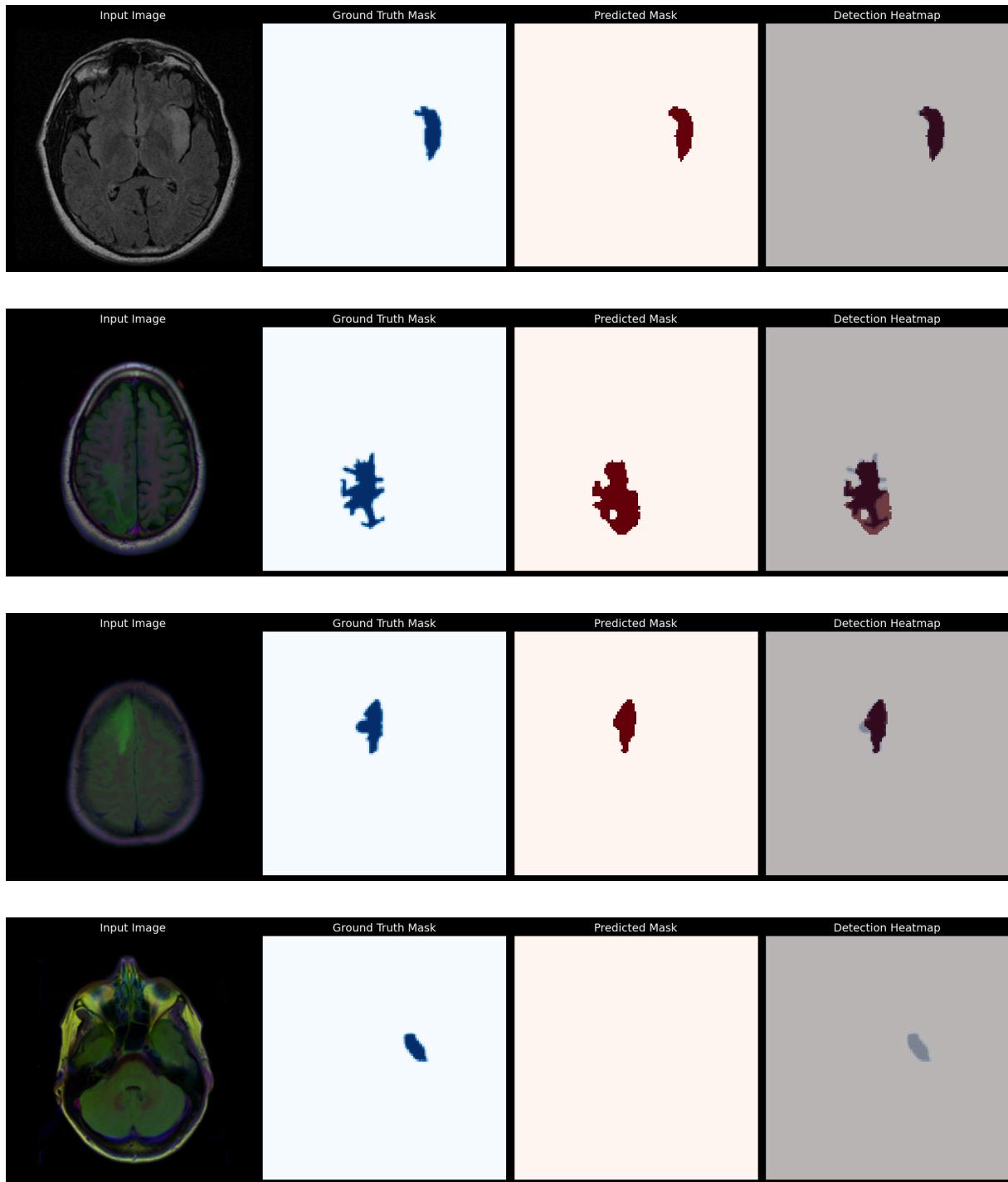


شکل ۱. نتایج مدل بر روی مجموعه داده تست بدون تومور

همانطور که مشخص است، مدل روی مجموعه داده بدون تومور با دقت خیلی خوب و صد درصدی عدم وجود تومور را تشخیص داده است.

در این تصاویر که ماسک واقعی به رنگ آبی و ماسک پیش‌بینی شده به رنگ قرمز است، ناحیه همپوشانی به رنگ خاکی درآمده که در واقع پیکسل‌هایی است که هم در ماسک واقعی و هم در ماسک پیش‌بینی شده مقدار غیر صفر داشته‌اند. توجه شود که در این مورد میزان threshold برای پیکسل‌های پیش‌بینی شده به اندازه‌ی ۰.۳ تعیین شده است. یعنی اگر مقدار پیکسلی بزرگ‌تر یا مساوی این مقدار بود، مقدار ۱ و در غیر این صورت مقدار ۰ گرفته است.





شکل ۱. نتایج مدل بر روی مجموعه داده تست دارای تومور

همینطور در مواردی که ماسک تصویر شامل تومور بوده، با دقت خوبی محل تومور را تشخیص داده و این میزان همپوشانی خوب و قابل قبول میباشد.

در تصویر آخر از این سری تصاویر مشخص است که مدل به اشتباه تومور را در مغز تشخیص نداده است و این به علت سخت بودن تشخیص از روی عکس مغز می‌باشد.

پرسش 2 - تشخیص تابلوهای راهنمایی و رانندگی

1-2. آماده‌سازی مجموعه داده

1-1-2. آشنایی با مجموعه داده

مجموعه داده مورد استفاده در این مسئله مجموعه داده German GTSDB یا همان Traffic Sign Detection Benchmark می‌باشد. این مجموعه شامل تصاویری از علائم راهنمایی و رانندگی به فرمت PPM می‌باشد. این فرمت یک فرمت رایج و غیر فشرده شده از تصاویر دارای کانال های رنگی می‌باشد.

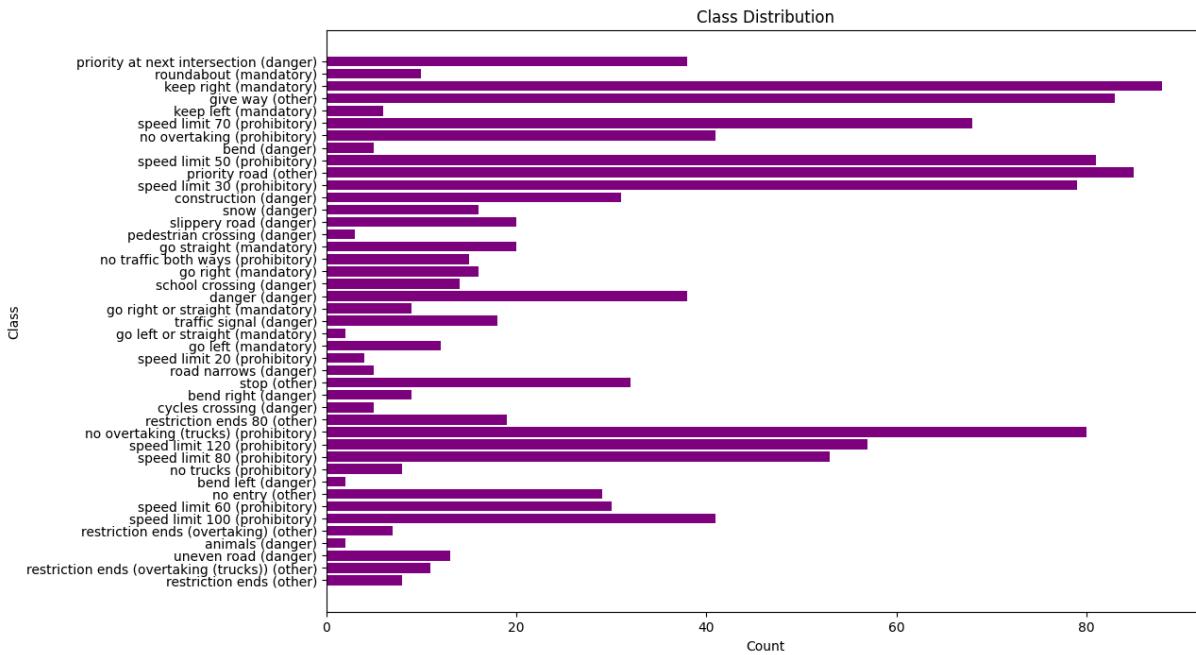
این مجموعه داده در مسائل تشخیص و شناسایی این علائم به کار می‌رود. تعداد تصاویر ذخیره شده 902 تصویر است که در همه‌ی این تصاویر در مجموعه 1206 علائم راهنمایی و رانندگی وجود دارد. بنابراین در برخی تصاویر بیش از یک مورد از علائم قرار دارد. این تصاویر در شرایط مختلف محیطی ثبت شده‌اند و این شرایط شامل پوشش جزئی بخشی از علائم، فاصله دوربین از علائم، شدت نور در صحنه می‌باشد.

در این بخش نمونه‌هایی از این تصاویر مجموعه داده به همراه علائم مشخص شده روی آنها را نشان می‌دهیم. این تشخیص شامل کادر بندی مورد مدنظر به همراه لیبل مربوطه می‌باشد که در شکل زیر نشان داده شده است:

نوع این علائم به انواع زیر محدود می‌شود:

Class	Category
1	Prohibitory
2	Danger
3	Mandatory
4	Other

بعد از تعریف این تقسیم بندی‌های موجود روی مجموعه داده به بررسی نحوه توزیع مجموعه داده در هر گروه از دسته بندی می‌پردازیم:



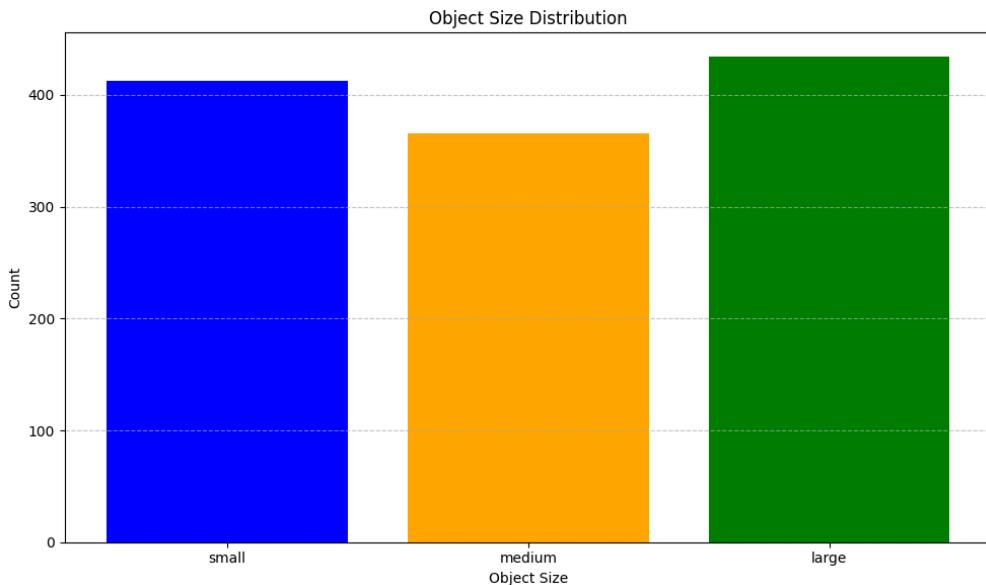
شکل 1. نحوه توزیع مجموعه داده در هر گروه از علائم

2-1-2. بررسی مجموعه داده در مقاله

همانطور که پیشتر ذکر شد، فاصله دوربین تا علائم مدنظر در مجموعه داده متغیر می‌باشد. به این منظور، در این بخش سه آستانه برای سایز اشیای موجود در تصویر تعیین می‌کنیم.

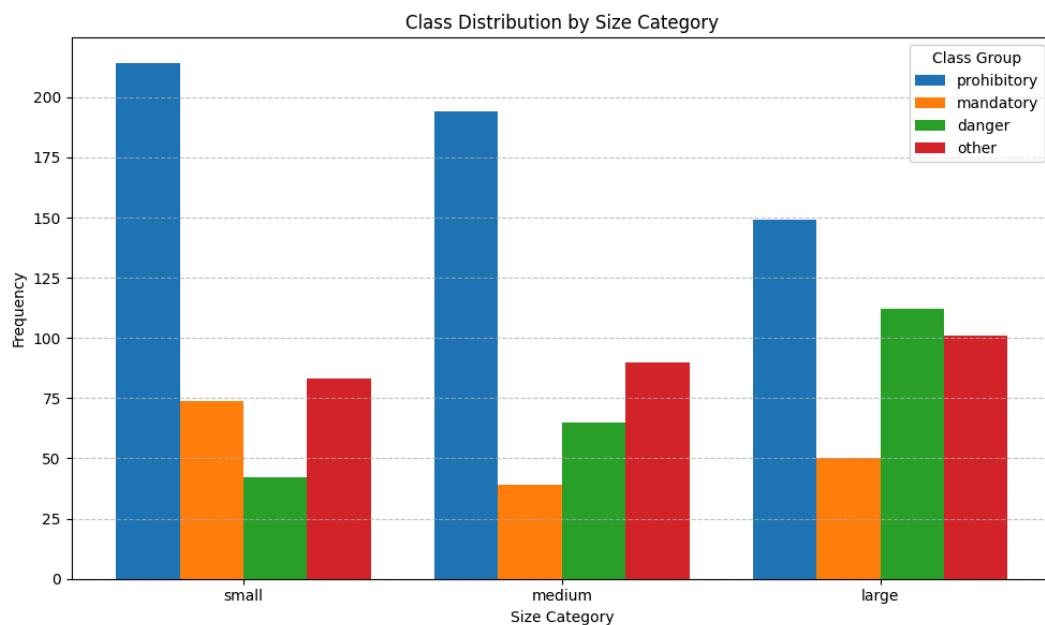
Group	Pixel sizes
Small	$width < 32$
Medium	$32 \leq width \leq 45$
Large	$width > 45$

بعد از این تقسیم بندی به بررسی توزیع مجموعه داده در هر گروه از دسته بندی می‌پردازیم:



شکل ۱. توزیع مجموعه داده در سه گروه مشخص شده

همانطور که مشخص است، این سه دسته تعداد تکرار تقریباً برابر دارند و این مورد باعث می‌شود مدل ما بهتر و قوی‌تر فرآیند یادگیری را انجام دهد.



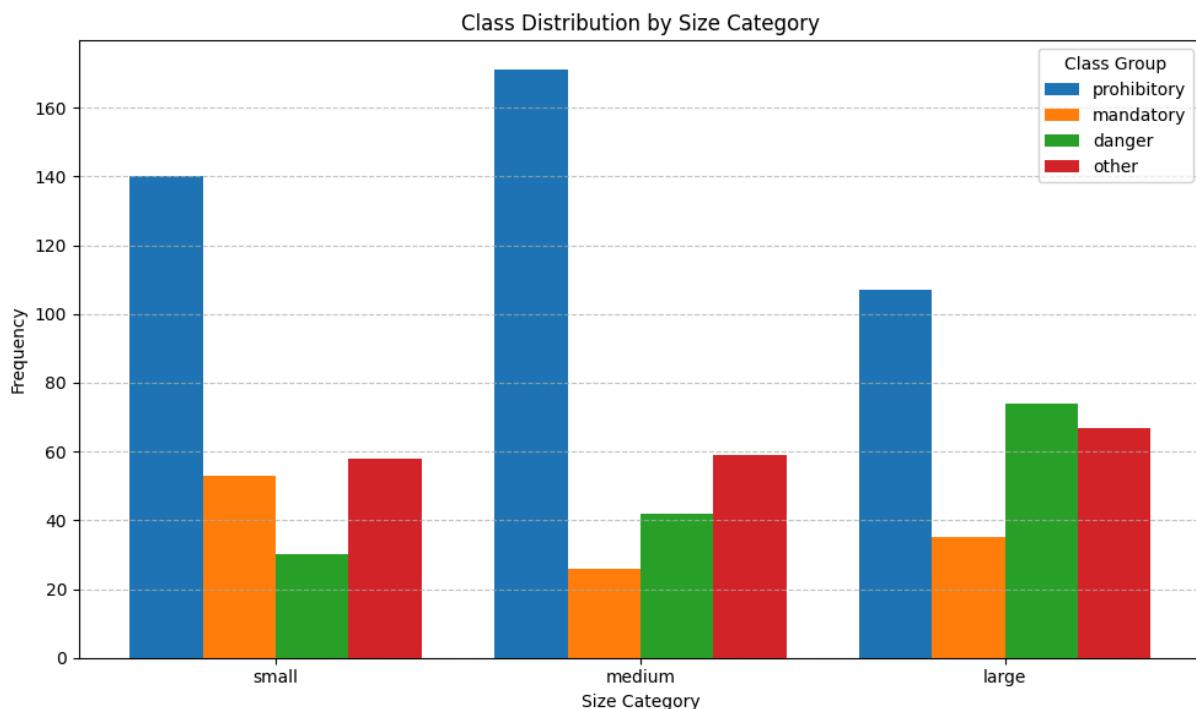
شکل ۱. توزیع مجموعه داده روی سه گروه تعریف شده با توجه به ۴ نوع از علائم

3-1-2. تقسیم مجموعه داده

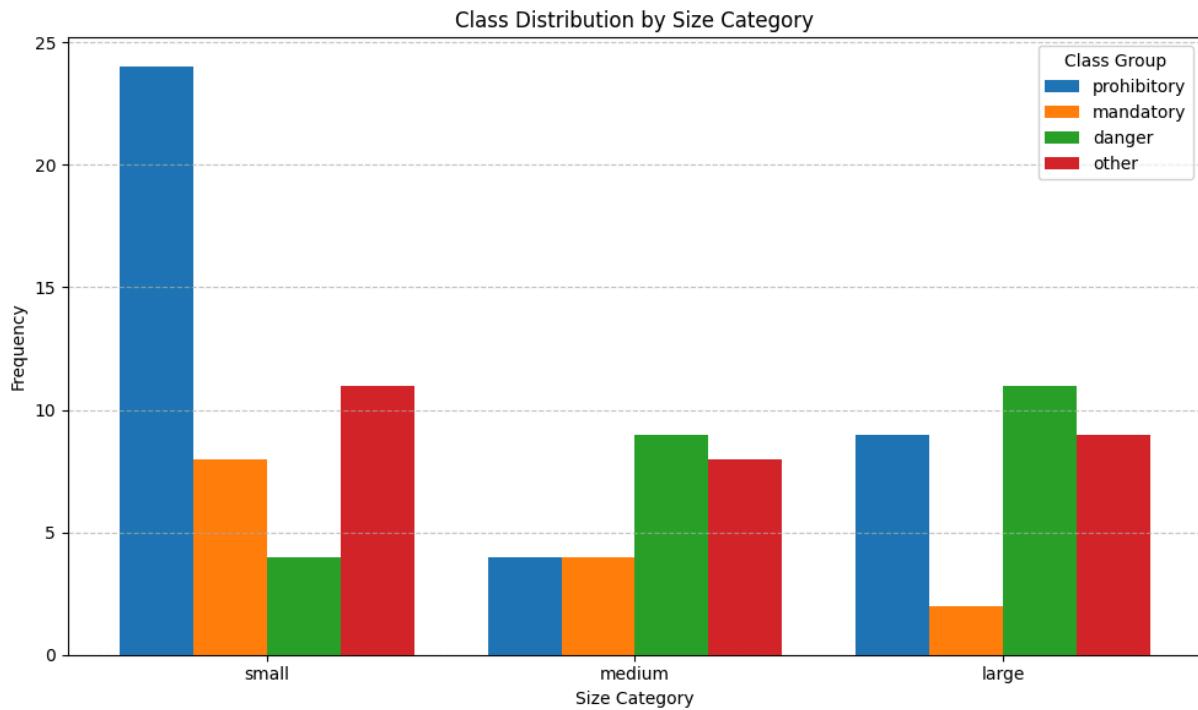
در این بخش مجموعه داده را به سه دسته train, validation و test تقسیم می‌کنیم که نسبت‌ها به صورت زیر هستند:

Group	Sizes
Train	72%
Validation	8%
Test	20%

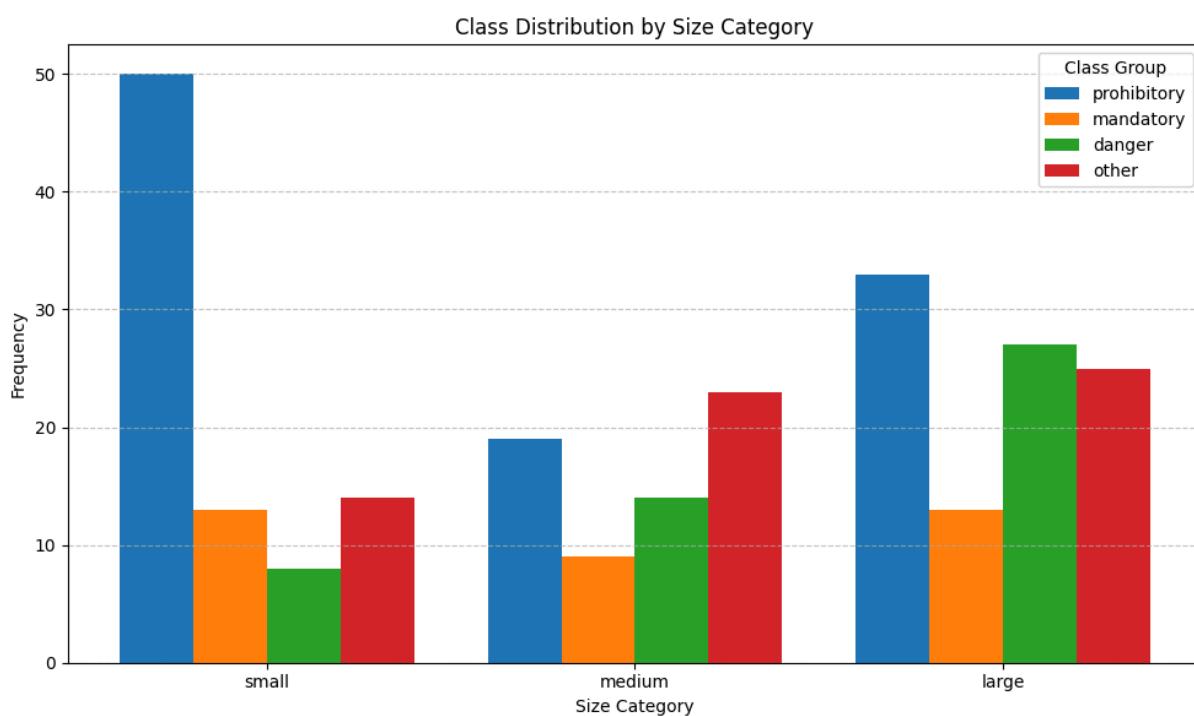
بعد از انجام این تقسیم بندی به بررسی توزیع هریک از این سه مجموعه در هر یک از سه گروه جدول مربوطه می‌پردازیم:



شکل 1. توزیع مجموعه داده Train



شكل 1. توزيع مجموعه داده Validation



شكل 1. توزيع مجموعه داده Test

2-2. تنظیم دقیق و ارزیابی مدل تشخیص شی دو مرحله‌ای

2-2-1. توضیح مختصر در مورد مدل **Faster R-CNN با شبکه پشتیبان ResNet50-FPN**

این مدل از یک معماری دو مرحله‌ای برای انجام تسک شناسایی و طبقه‌بندی اشیا در تصاویر استفاده می‌کند. هر یک از این دو مرحله یک وظیفه به خصوصی در کل شبکه ایفا می‌کنند تا در نهایت مدل به خوبی اشیا را تشخیص دهد و دسته بندی کند. این دو بخش به شرح زیر هستند:

Region Proposal Network •

در مرحله اول به کمک شبکه RPN از مدل، نواحی‌ای از عکس که حضور اشیا در آنها تشخیص داده شده شناسایی و ذخیره می‌شوند. در واقع مدل محدوده‌هایی را به کمک یک sliding window از تصویر ورودی استخراج می‌کند.

به طور کلی این شبکه کانولوشنال نواحی مستعد حضور اشیا در تصویر ورودی را می‌یابد و به شبکه بعدی ارائه می‌دهد.

Region of Interest •

در این بخش feature map های با سایز فیکس تولید می‌شوند که تطابق با لایه‌های fully connected را داشته باشند.

Head Network •

در مرحله بعدی، مسئله طبقه بندی و رگرسیون bounding box حل می‌شود. با اتمام این مرحله خروجی‌های RPN تا حد امکان تصحیح شده‌اند و برچسب box مربوطه هم مشخص می‌شود.

ResNet50-FPN • شبکه پشتیبان

این شبکه ویژگی‌های مربوط به اشیا را از تصاویر استخراج می‌کند. این استخراج به کمک یک شبکه عصبی با 50 لایه می‌باشد که ارتباط هایی از نوع skip connection در آن تعییه شده است. راجب این اتصالات در مسئله اول صحبت کرده‌ایم. در این لایه‌ها بلوک‌های ResNet از ناپدید شدن گرادیان‌ها در محاسبات ریاضی جلوگیری می‌کنند تا ویژگی‌ها استخراج شده مقادیر منطقی و مفید داشته باشند.

به طور کلی این شبکه نواحی ورودی را به طور جداگانه بررسی می‌کند.

همانطور که در اسم این شبکه پیداست، بخش دیگر آن شامل Feature Pyramid Network یا همان FPN می‌باشد که به شبکه قابلیت تشخیص اشیا در اندازه‌های مختلف را می‌دهد. در واقع این زیرشبکه برای فرآیند یادگیری از ترکیب ویژگی‌های چند مقیاسی از سطوح مختلف شبکه استفاده می‌کند.

2-2-2. اقدامات لازم برای آماده‌سازی مدل و تنظیم آن

برای آماده‌سازی مدل از تابع زیر استفاده شده است:

```
def get_model(num_classes):
    model = fasterrcnn_resnet50_fpn(weights="COCO_V1")
    in_features = model.roi_heads.box_predictor.cls_score.in_features
    model.roi_heads.box_predictor = torchvision.models.detection.faster_rcnn.FastRCNNPredictor(
        in_features, num_classes
    )
    return model
```

این تابع مدل از پیش آموزش‌داده شده روی مجموعه داده COCO را از کتابخانه را گرفته و وزن‌های مدل را دریافت می‌کند.

از آنجا که وزن‌های مدل همه جز Roi head به صورت پری‌تریند هستند ابتدا تلاش می‌کنیم با فریز کردن تمام قسمت‌های مدل بجز roi head آن را به گونه‌ای آموزش دهیم که با وزن‌های پری‌ترین شده مدل بالا تری iou را به دست بیاوریم و پس از آن حال از آنجا که roi head به حد معقولی ترین شده ابتدا rpn را از freeze درآورده و باقی مدل را فریز می‌کنیم تا rpn را جهت پیدا کردن بهترین پروپوزال‌ها ترین کنیم و از آنجا که roi head جهت بهترین iou ترین شده بوده است این مرحله برای rpn به کمک head به خوبی انجام می‌شود. پس از آن که از این مرحله تمام شد حال دوباره Roi head را جهت دستیابی به برترین map فاین تیون می‌کنیم و در نهایت از آنجا که مدل بخش به حد رضایت بخشی ترین شده است حال تمامی مدل را از freeze درآورده و کل مدل شامل بک بود آن را بر اساس داده موجود فاین تیون می‌کنیم.

بر بخش بعدی تعداد فیچرهای ورودی برای لایه طبقه‌بندی را از مدل Region of Interest یا همان RoI head دریافت می‌کنیم. در این مدل تعداد کلاس‌های مجموعه داده ورودی سنت می‌شود.

در بخش آخر هم پیش‌بینی کننده باکس‌های احاطه‌گر اشیا را از مدل می‌گیریم و به آن ورودی‌های مورد نیازش را می‌دهیم.

3-2-2. آماده‌سازی و نرمال‌سازی مجموعه داده ورودی مدل

برای این بخش کلاسی تعریف شده که در کانستراکتور کلاس، مسیر داده annotation و مسیر تصاویر ورودی دریافت می‌شوند. همینطور هنگام فراخوانی این کلاس، transform ورودی را برای تبدیل‌سازی مقادیر به تنسور ست می‌کنیم. همینطور در این کلاس لیل‌های تصاویر را به id مپ می‌کنیم که مقادیر عددی داشته باشند. از آنجایی که مجموعه داده‌های تصاویر حجم بزرگ دارند، از لود همه‌ی آنها در تک مرحله جلوگیری کرده و تنها هنگام نیاز هر تصویر، آن را لود کرده، را روی آن اعمال کرده و به همراه همه‌ی دیتای لازم از آن مسیر خروجی می‌دهیم.

مزیت این کلاس استفاده و ارث بری آن از کلاس Dataset از torch.utils.data می‌باشد که به ما قابلیت همگام‌سازی با دیتا تایپ Dataloader ها را می‌دهد.

```

class FullIJCNN2013Dataset(Dataset):
    def __init__(self, annotations_file, images_dir, transform=None):
        self.annotations = self._load_annotations(annotations_file)
        self.images_dir = images_dir
        self.transform = transform
        self.label_to_id = {
            "prohibitory": 1,
            "danger": 2,
            "mandatory": 3,
            "other": 4
        }

    def _load_annotations(self, annotations_file):
        category_mapping = {
            "prohibitory": list(range(0, 6)) + list(range(7, 11)) + [15, 16],
            "danger": list(range(18, 32)) + [11],
            "mandatory": list(range(33, 41)),
            "other": [6, 12, 13, 14, 17, 32, 41, 42]
        }

        category_lookup = {}
        for category, ids in category_mapping.items():
            for cid in ids:
                category_lookup[cid] = category

        annotations = {}
        with open(annotations_file, "r") as file:
            for line in file:
                parts = line.strip().split(";")
                img_name = parts[0]
                bbox = tuple(map(int, parts[1:5]))
                class_id = int(parts[5])
                category = category_lookup.get(class_id, "other")

                if img_name not in annotations:
                    annotations[img_name] = []
                annotations[img_name].append({"bbox": bbox, "label": category})
        return annotations

    def __len__(self):
        return len(self.annotations)

    def __getitem__(self, idx):
        img_name = list(self.annotations.keys())[idx]
        img_path = os.path.join(self.images_dir, img_name)
        image = Image.open(img_path).convert("RGB")

        objects = self.annotations[img_name]
        bboxes = torch.tensor([obj["bbox"] for obj in objects], dtype=torch.float32)
        labels = torch.tensor(
            [self.label_to_id[obj["label"]]] for obj in objects), dtype=torch.int64
        )

        target = {
            "boxes": bboxes,
            "labels": labels
        }

        if self.transform: image = self.transform(image)

        return image, target

```

توجه کنید روی مجموعه داده فرآیند نرم‌السازی خارج از تبدیل به تنسور انجام نشده است و دلیل آن، مشاهده عملکرد ضعیف تر مدل روی مجموعه داده نرم‌ال شده می‌باشد.

4-2-2. معیارهای ارزیابی مدل

در این مسئله از معیارهای ارزیابی به شرح زیر استفاده شده است:

• معیار Intersection over Union

این معیار نسبت مساحت مشترک ناحیه واقعی با ناحیه پیشبینی شده توسط مدل را به اجتماع این دو ناحیه را گزارش می‌دهد.

معیار IoU به صورت زیر محاسبه می‌شود:

$$IoU = \frac{\text{Intersection Area}}{\text{Union Area}} = \frac{|A \cap B|}{|A \cup B|}$$

به عبارتی این فرمول تعبیری به کمک معیارهای ارزیابی دسته بندی دارد که به شرح زیر است:

$$IoU = \frac{\text{Intersection Area}}{\text{Union Area}} = \frac{TP}{TP+FP+FN}$$

بدیهی است که این متريک مقداری بین 0 و 1 خواهد داشت و ما به دنبال ماقسیمازی کردن این متريک هستیم.

• معیار Mean Average Precision

ابتدا به توضیح معیار Average Precision یا AP می‌پردازیم. این معیار میانگین دقت در کلاس‌های مختلف از مجموعه داده می‌باشد. مقدار عددی این متريک هم از محاسبه مساحت زیر نمودار Precision-Recall به دست می‌آید.

در این بخش به توضیح چندین متريک استفاده شده در تشخیص دسته‌بندی و ارزیابی عملکرد مدل‌های یادگیری ماشین می‌پردازیم:

• تشخیص صحیح مثبت، TP

یا همان True Positive، تعداد نمونه‌هایی که به درستی تشخیص داده شده‌اند که در واقعیت مثبت هم هستند، می‌باشد.

• تشخیص صحیح منفی، TN

یا همان True Negative، تعداد نمونه‌هایی که به درستی تشخیص داده شده‌اند که در واقعیت مثبت هستند، می‌باشد.

• تشخیص غلط مثبت، FP

یا همان False Positive، تعداد نمونه‌هایی که به غلط به عنوان اعضای مثبت تشخیص داده شده‌اند که در واقعیت منفی هستند، می‌باشد.

• تشخیص غلط منفی، FN

یا همان False Negative، تعداد نمونه‌هایی که به غلط به عنوان اعضای منفی تشخیص داده شده‌اند که در واقعیت مثبت هم هستند، می‌باشد.

بنابراین ابتدا امتیازهای پیش‌بینی شده توسط مدل را دریافت کرده و آنها را به label‌های پیش‌بینی شده تبدیل می‌کنیم. از اطلاعات ماتریکس سردرگمی، مقادیر فوق را به دست آورده و معیارهای recall و precision را طبق آنها به دست می‌آوریم. مساحت زیر نمودار Precision-Recall را به دست آورده و AP حاصل شده است.

میانگین این نمرات AP ترکیب شده، mAP را تولید می‌کند و به ما اطلاع می‌دهد که مدل چقدر خوب عمل می‌کند.

بدیهی است که این متريک مقداری بین 0 و 1 خواهد داشت و ما به دنبال ماکسیمایز کردن اين متريک هستيم.

2-2-5. بهینه‌ساز و تابع هزينه

• بهینه‌ساز

در این مسئله به منظور بهبود عملکرد مدل در فرآيند Fine-Tuning از تابع بهینه ساز AdamW از torch استفاده شده است. اين نسخه حالت پيش‌رفته‌تر بهینه ساز Adam است که پيش‌تر توضيحات آن را ارائه کرده‌ایم. اين اپتيمایزر برای حل مشکل Weight Decay طراحی شده و دليل نامگذاري آن هم همين مورد است. بر برخی مسائلی که از Adam استفاده می‌شود، پس از هر روز رسانی وزن‌های قابل یادگیری و تغییر مقادیر گرادیان پارامترها، ممکن است مشکل ناخواسته Weight Decay رخ دهد که اپتيمایزر AdamW به کمک فرآيند نرم‌السازی خود، باعث می‌شود اگر گرادیان وزن به خصوصی خیلی بزرگ باشد یا به مقدار خیلی زیاد تغییر کند، فرآيند نرم‌السازی به نحوی عمل می‌کند که مخرج نرم‌السازی هم به اندازه کافی بزرگ بوده و در نتیجه وزن مربوطه کمتر از وزن‌هایی که گرادیان آنها به کندی و به مقدار کم تغییر می‌کنند، regularize شود. در واقع نرخ جريمه وزن تحميل شده به هر وزن مدل مستقييم از وزن ديگر مدل و همينطور مستقل از نحوه به روز رسانی گرادیان ها می‌باشد.

• تابع هزينه

در این مسئله از تابع هزینه دیفالت مدل Fast RCNN با پشتیبان ResNet50-FPN استفاده شده است. این تابع همان Negative Log Likelihood Loss برای بخش طبقه‌بندی مدل می‌باشد. در بخش رگرسیون مدل از تابع هزینه Smooth L1 Loss استفاده شده است.

در این بخش به توضیح این دو تابع می‌پردازیم:

Negative Log Likelihood Loss

این تابع با نام مخفف NNLLLoss در مسائل طبقه‌بندی و مخصوصاً در مسائلی که با مدل‌های احتمالاتی سروکله می‌زنند، کاربرد دارد. این مدل‌های احتمالاتی تفاوت بین توزیع احتمال پیش‌بینی شده و توزیع واقعی طبقات را اندازه‌گیری می‌کند.

$$NLLLoss = - \log(\hat{y}_y)$$

\hat{y}_y is the predicted probability of the true class y

Smooth L1 Loss

این با نام دیگر Huber Loss در مسائل رگرسیون و مخصوصاً در مسائلی که outlier ها در داده ما مشاهده می‌شود، کاربرد دارد. این تابع دو متريک L1 و L2 را با هم ترکيب می‌کند و يك بالانس و تعادل بين precision و robustness برقرار می‌سازد.

$$\text{Smooth L1 Loss} = \text{if } |x - y| < \delta : \frac{1}{2} (x - y)^2$$

$$\text{Smooth L1 Loss} = \text{otherwise} : \delta|x - y| - \frac{\delta^2}{2}$$

x is the predicted value,

y is the ground truth value,

$\delta > 0$ is a hyperparameter controlling the threshold

6-2-2. ارزیابی مدل

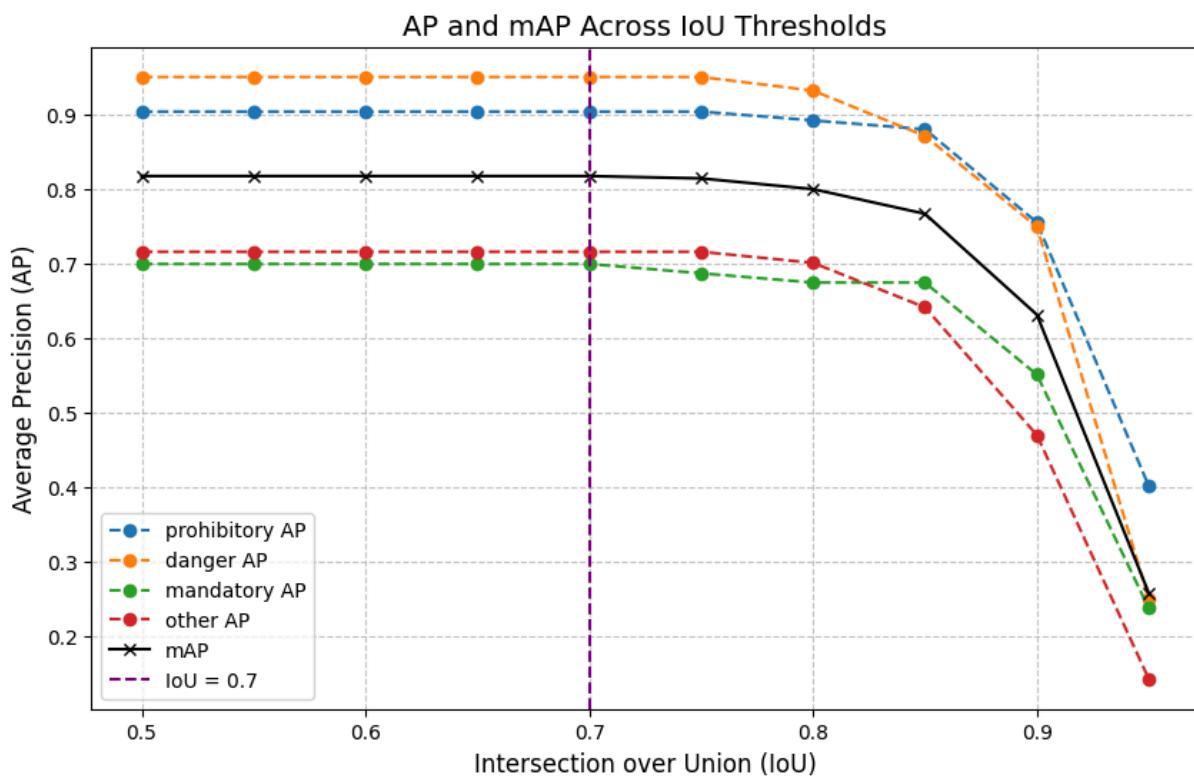
نتایج متريک‌های تعریف شده را پس از ارزیابی مدل روی مجموعه داده تست را گزارش می‌دهیم:

IoU Loss	Map (IoU Threshold: 0.5)
0.6116	0.8180

همانطور که مشاهده می شود، این مدل به نتیجه خوبی به روی میانگین ap ها روی threshold رسیده است. این نتیجه در نمودار پخش بعدی مشخص تر است.

7-2-2. نمودار AP به ازای IoU های متفاوت

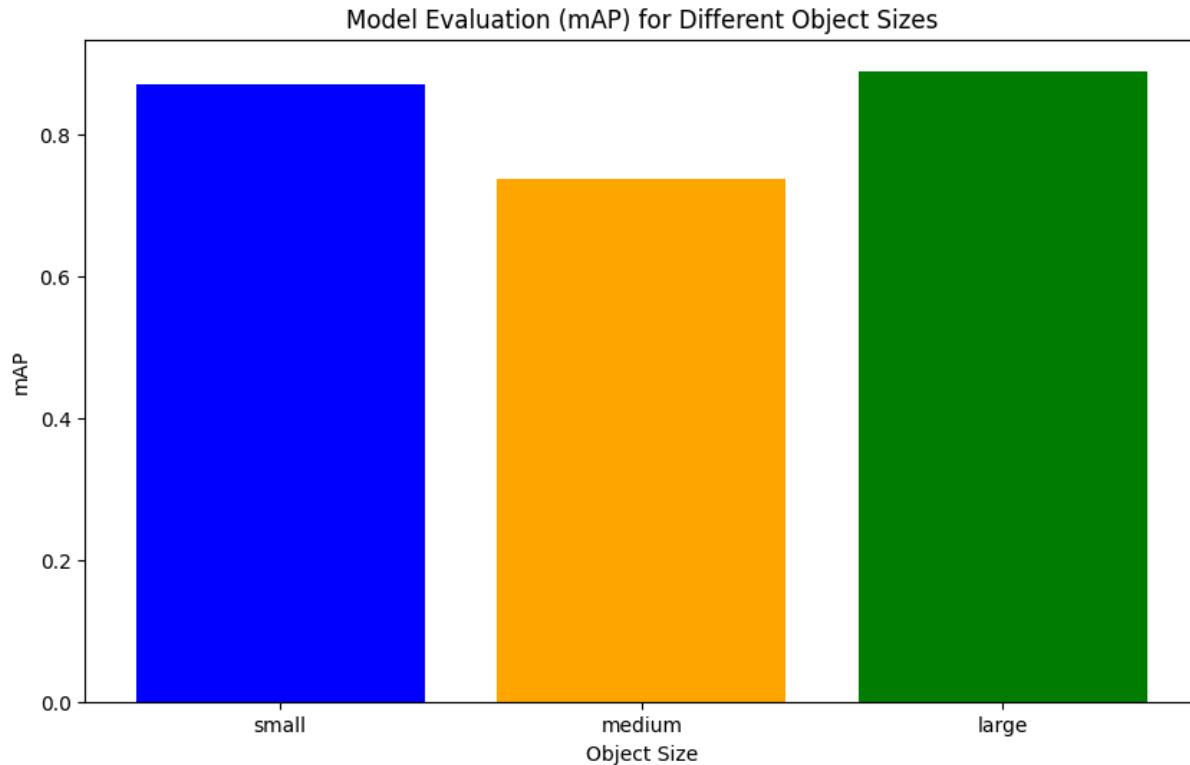
در این پخش نمودار AP به ازای مقادیر IoU های از 0.5 تا 0.95 را نمایش می دهیم:



همانطور که مشاهده می شود، با افزایش IoU مقادیر ap ها و map اکیدا کاهشی است زیرا با بالا رفتن IoU threshold تعداد tp ها به صورت کلی کاهش و fp ها افزایش می یابد و در نتیجه این فرایند طبیعی است.

8-2-2. ارزیابی مدل روی اشیا با اندازه های متفاوت

همانطور که توضیح داده شده بود، مجموعه داده را به سه دسته اشیای small، medium، large تقسیم کردیم. حال به تحلیل ارزیابی ها روی این چند دسته می پردازیم:



این ارزیابی بر اساس $\text{IOU} = 0.5$ انجام شده است و همانطور که مشاهده می شود مدل در هر سه سایز مختلف map مناسبی ارائه شده است. همچنین با دقت به توزیع دیتا پوینت های دیتابست به دلیل دقت پایین تر مدل در سایز مديوم ميرسيم زيرا ديتاى كمتري داشتيم و مدل به كيفيت باقى سايز ها ترين نشد.

2-2-9. نتایج پیش‌بینی یک نمونه تصویر از داده‌ی ارزیابی

در این بخش، خروجی مدل روی یک تصویر از مجموعه داده تست را نمایش می‌دهیم:



همانطور که مشاهده می شود، در نمونه های بالا مدل کادر های مناسبی انتخاب کرده است و همچنین تابلو ها به درستی classify کرده است.

3-2. تنظیم دقیق و ارزیابی مدل تشخیص شی تک مرحله‌ای

3-3-1. توضیح مختصر در مورد مدل SSD300 با شبکه پشتیبان VGG16

این مدل با نام Single Shot Multibox Detector، از یک الگوریتم single shot مراحله‌ای برای انجام تسک شناسایی و طبقه‌بندی اشیا در تصاویر استفاده می‌کند. که بالانسی را بین سرعت و دقت را رعایت می‌کند. همینطور به علت تفاوت تک مرحله‌ای بودن مدل که منجر به افزایش سرعت عملکرد آن شده است، در تسک های real time کاربرد زیادی دارد.

همانطور که در اسم این شبکه پیداست، بخش دیگر آن شامل Feature Pyramid Network یا همان FPN می‌باشد که به شبکه قابلیت تشخیص اشیا در اندازه‌های مختلف را می‌دهد. در واقع این زیرشبکه برای فرآیند یادگیری از ترکیب ویژگی‌های چند مقیاسی از سطوح مختلف شبکه استفاده می‌کند.

در زیر توضیحی راجب معماری این مدل آمده است:

• شبکه پشتیبان VGG16

این مدل از شبکه کانولوشنال با 16 لایه به نام VGG16 به عنوان شبکه پشتیبان استفاده می‌کند تا ویژگی‌های تصاویر ورودی را استخراج کند. این شبکه معماری ساده‌ای دارند و در استخراج ویژگی‌های با کیفیت عملکرد خوبی دارد.

• لایه‌های ویژگی مازاد

این لایه‌های کانولوشنال که پس از شبکه پشتیبان جای می‌گیرند، قابلیت پیش بینی روی اسکیل‌های متفاوت را به مدل می‌دهند.

• MultiBox Head

از آنجایی که قرار است مدل کلی class probability و bounding box را مستقیماً از روی feature map ها در اسکیل‌های گوناگون به دست بیاورد، در این بخش سلول‌هایی متناظر با هر feature map تعییه شده که هر کدام مسئول پیش‌بینی اشیا در محدوده سایز مشخصی هستند.

• Default Boxes - Anchors

جعبه های لنگر یا همان anchor box هایی که از پیش تعریف شده اند با نسبت های مختلف برای هر سلول feature map استفاده می شوند. پیش بینی ها با تنظیم این لنگرهای در طول تمرین اصلاح خواهند شد.

2-3-2. اقدامات لازم برای آماده سازی مدل و تنظیم آن

برای آماده سازی مدل از تابع زیر استفاده شده است:

```
def create_ssd_model(num_classes=5, size=300):
    model = torchvision.models.detection.ssd300_vgg16(weights=SSD300_VGG16_Weights.DEFAULT)
    in_channels = _utils.retrieve_out_channels(model.backbone, (size, size))
    num_anchors = model.anchor_generator.num_anchors_per_location()
    model.head.classification_head = SSDClassificationHead(
        in_channels=in_channels, num_anchors=num_anchors, num_classes=num_classes)
    model.transform.min_size = (size, )
    model.transform.max_size = size
    return model
```

این تابع مدل از پیش آموزش داده شده پیشفرض را گرفته و وزن های مدل را دریافت می کند. بر بخش بعدی تعداد چنل های ورودی و anchor های ورودی را به ترتیب از utils و از مدل دریافت می کنیم. این موارد در head جدید تعریف شده برای مسئله ما یعنی در مدل SSDClassificationHead سرت خواهند شد. در این مدل تعداد کلاس های مجموعه داده ورودی سرت می شود.

جهت ترین کردن این مدل از وزن های دیفالت آن استفاده کردیم و ابتدا به کمک ارلی استاپینگ تمامی قسمت های مدل به جز ClassificationHead را فریز کردیم و هد را ترین کردیم سپس تمامی مدل را از freeze در آوردیم و آن را فاین تیون کردیم.

3-3-2. آماده سازی و نرمال سازی مجموعه داده ورودی مدل

این بخش، مانند بخش قبلی صورت گرفته است.

4-3-2. معیارهای ارزیابی مدل

در این مسئله از همان معیارهای ارزیابی اشاره شده در بخش قبل استفاده می شود.

5-3-2. بهینه ساز و تابع هزینه

- بهینه ساز

در این مسئله هم از بهینه ساز AdamW استفاده شده است که پیشتر توضیحات مربوطه ارائه شد.

• تابع هزینه

در این مسئله از تابع هزینه دیفالت مدل SSD300 با پشتیبان VGG16 استفاده شده است.
این تابع دو بخش اصلی دارد که به شرح زیر است:

بخش Localization Loss

یا همان هزینه مکانیابی مدل، مختصات Bounding Box پیش‌بینی شده توسط مدل برای اشیا را با Bounding Box واقعی مقایسه کرد و پنالتی را به کمک تابع هزینه Smooth L1 Loss محاسبه می‌کند.

$$L_{Smooth\ L1} = \text{if } |y - \hat{y}| < \delta, \frac{1}{2} (y - \hat{y})^2$$

$$L_{Smooth\ L1} = \text{otherwise, } \delta|y - \hat{y}| - \frac{\delta^2}{2}$$

δ is a hyperparameter that determines the threshold between L1 and L2

بخش Confidence Loss

یا همان هزینه اطمینان مدل، میزان اطمینان مدل از کلاس پیش‌بینی‌اش را اندازه گیری می‌کند.

به کمک تابع هزینه Cross Entropy Loss محاسبه می‌کند.

$$L_{CE} = -\frac{1}{N} \sum_{i=1}^N \sum_{j=1}^C y_{ij} \log(\hat{y}_{ij})$$

C : Number of Classes

y_{ij} : Binary indicator (whether class j is the correct classification for sample i)

\hat{y}_{ij} : predicted probability of class j for sample i

$$L_{CE} = -\frac{1}{N} \sum_{i=1}^N [y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)]$$

6-3-2. ارزیابی مدل

نتایج متريکهای تعریف شده را پس از ارزیابی مدل روی مجموعه داده تست را گزارش می‌دهیم:

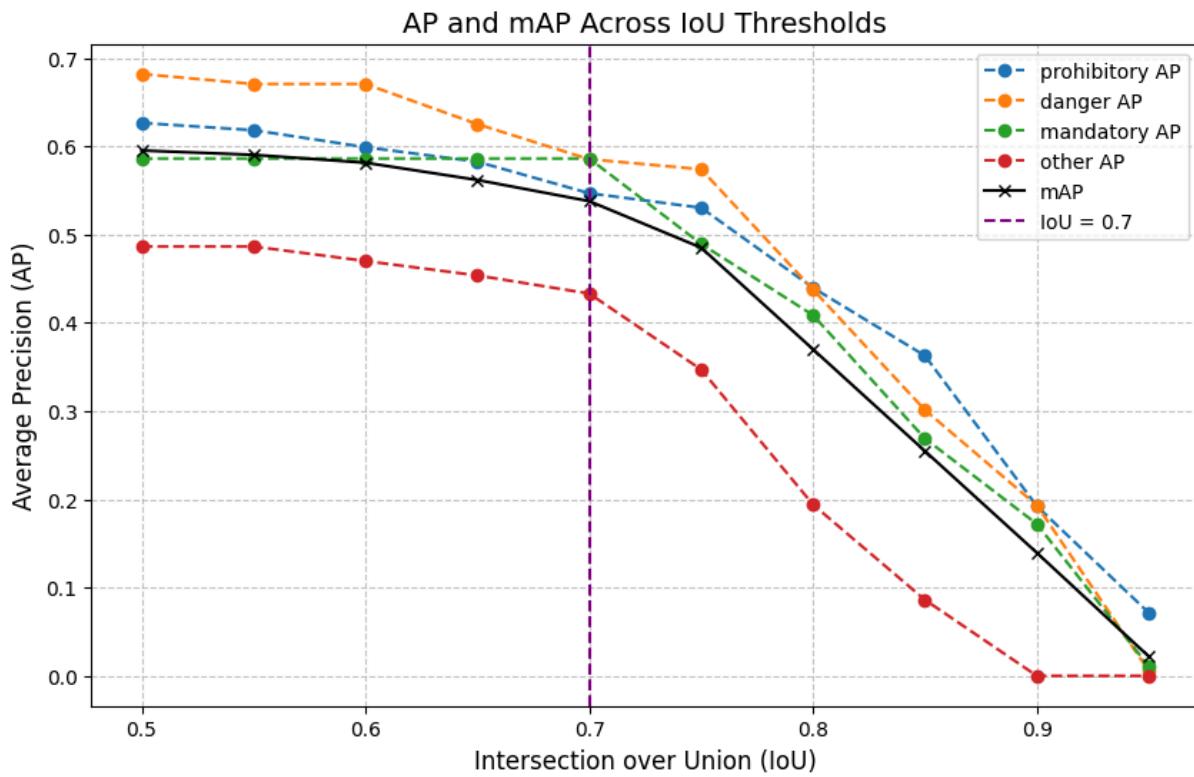
IoU Loss	Map (IoU Threshold: 0.5)
0.5055	0.5951

همانطور که مشاهده می‌شود، این مدل به نتیجه نسبتاً خوبی به روی میانگین ap ها روی threshold 0.5 رسیده است. این نتیجه در نمودار بخش بعدی مشخص‌تر است.

علت پایین بودن این مورد نسبت به مدل قبلی، single shot بودن این مدل است که در این مقایسه نتیجه کمی پایین تر نشان داده اما در عمل مدل سریعی می‌باشد و تفاوت سرعت مدل را در طول پروژه احساس کرده‌ایم.

7-3-2. نمودار AP به ازای IoU های متفاوت

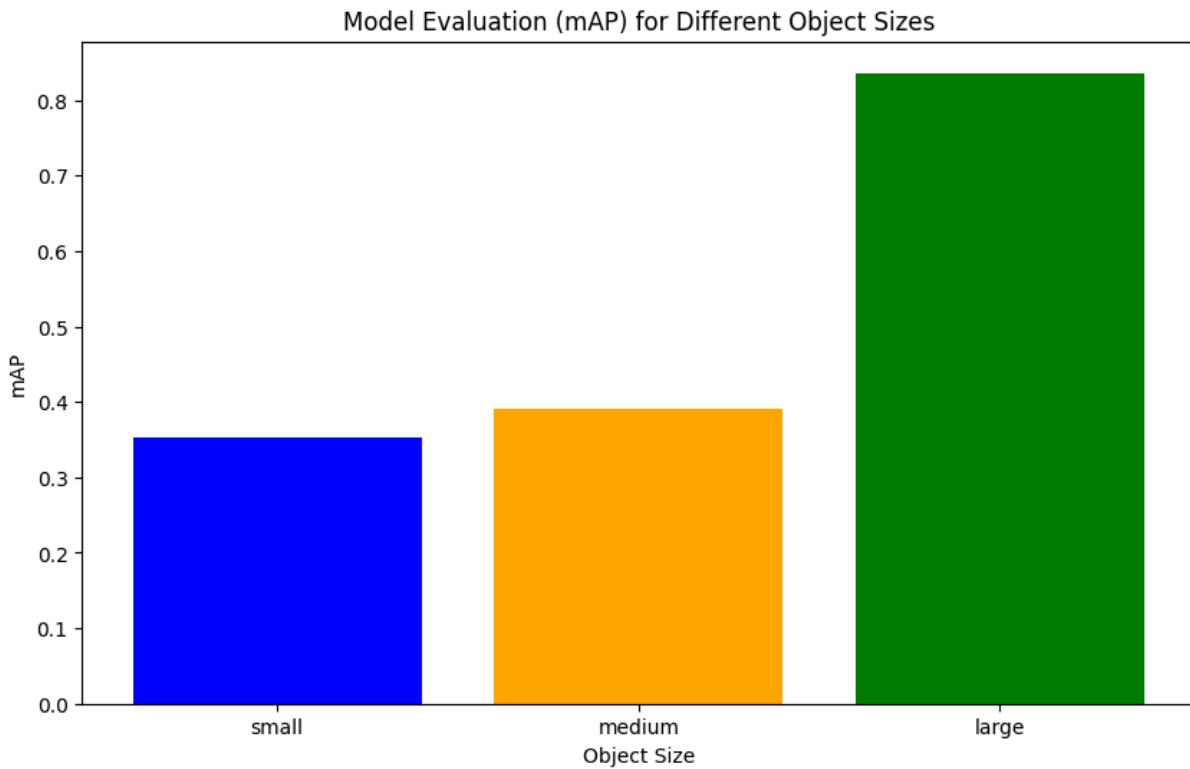
در این بخش نمودار AP به ازای مقادیر IoU های از 0.5 تا 0.95 را نمایش می‌دهیم:



همانطور که مشاهده می‌شود، مشابه موضوع مطرح شده در مورد مدل قبل در این مورد نیز صادق است که با افزایش IoU کاهش map را مشاهده می‌کنیم.

8-3-2. ارزیابی مدل روی اشیا با اندازه‌های متفاوت

همانطور که توضیح داده شده بود، مجموعه داده را به سه دسته اشیای small، medium و large تقسیم کرده بودیم. حال به تحلیل ارزیابی‌ها روی این چند دسته می‌پردازیم:



همانطور که مشاهده می شود ، این مدل به دلیل تک مرحله ای بودنش عملکرد ضعیفی در آبجکتهای small یا medium دارد زیرا به دلیل تک مرحله ای بودنش از قادر های پیش فرضی استفاده میکند که توانایی عملکرد ان بر روی ابجکت هایی که سایز کوچکی دارد را بسیار تحت تاثیر قرار میدهد.

3-2-9. نتایج پیشビینی یک نمونه تصویر از داده ای ارزیابی

در این بخش، خروجی مدل روی یک تصویر از مجموعه داده تست را نمایش می دهیم:

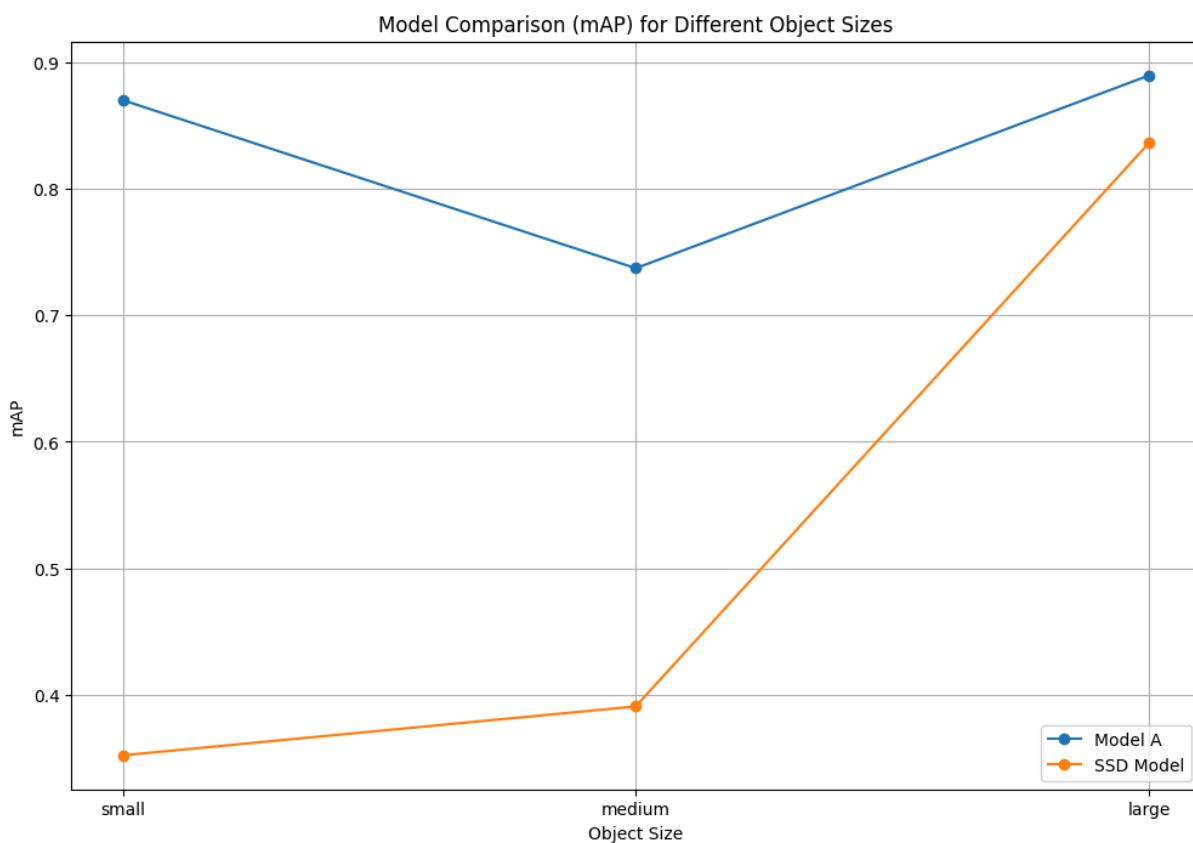


همانطور که مشاهده می‌شود، قادر های انتخاب شده دقیق‌تر در مقابل مدل faster rcnn داشته است.

4-2. ارزیابی نتایج و مقایسه مدل‌ها

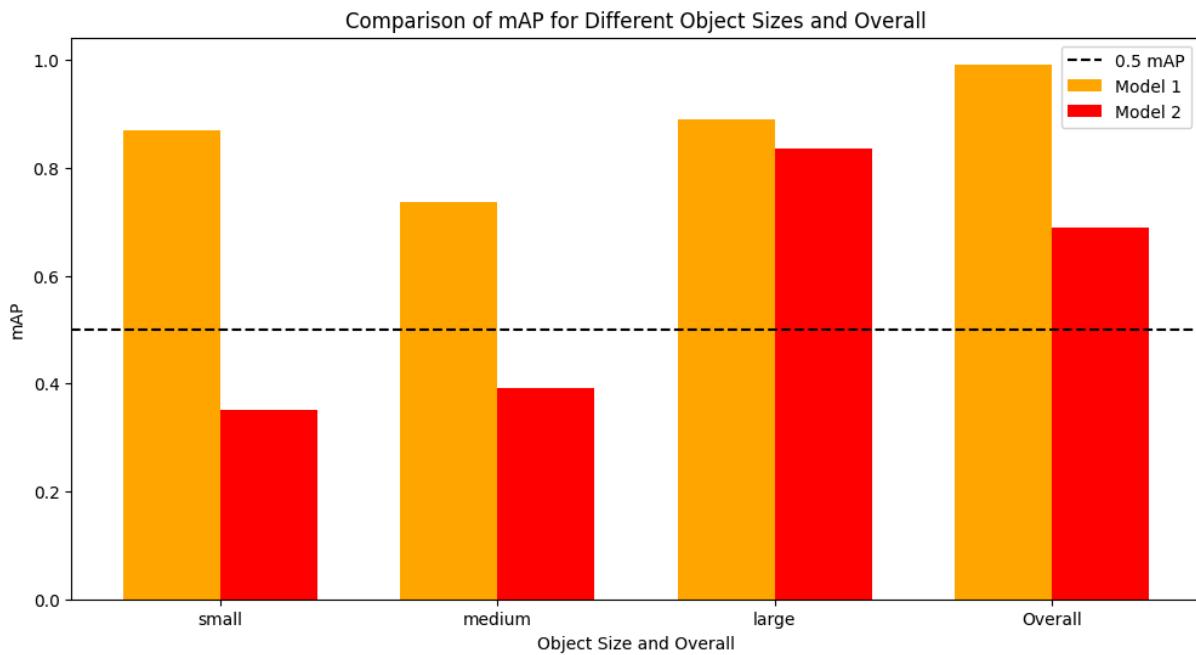
برای بررسی عملکرد این دو مدل، آنها را روی یک مجموعه داده ارزیابی می‌کنیم.

نتایج به شرح زیر است:



همانطور که مشاهده می‌شود، مدل ssd را داریم که با کاهش سایز آبجکت‌ها دچار کاهش معناداری در map خود می‌شود در حالی که faster rcnn این موضوع را نشان نمیدهد و این مدل در تمامی سایز‌های موجود عملکرد نسبتاً یکنواختی را ارائه داده است که علت این تفاوت وجود لایه rpn در مدل faster rcnn است. مدل ssd به کمک قادر های پیش فرض و با تبدیلاتی بر روی این قادر های پیشنهادی آبجکت‌های خود را دیتکت می‌کند و این موضوع موجب محدودیت این مدل در تشخیص سایز‌های مختلف قادر شده است و در نهایت منتهی به ضعف عملکرد این مدل در تشخیص آبجکت‌های کوچک و متوسط شده است زیرا فرایند سختی جهت تبدیل قادر های پیش

فرض به کادر های مورد نیاز دارد. در حالی که مدل Faster rcnn به دلیل وجود لایه `rpn` در طی فرایند ترین شدن خود می آموزد که کادر های مناسبی برای هر عکس پیشنهاد دهد و این کادر های پیشنهاد شده به دلیل هوشمندی و مدل آموزش داده شده جهت پیشنهاد این کادر ها عملکرد بسیار بهتری در پیش بینی کادر های مورد نیاز دارد که منجر به ارائه عملکردی یکنواخت در اشیا با سایز های مختلف شده است.



با بررسی این نمودار در میباییم دو مدل روی هر 3 دسته داده تست خودشان چه عملکردی داشته است. همانطور که از بار های اخر مشخص است، به طور Overall مدل اول روی داده بهتر عمل کرده است اما مدل دوم هم از مقدار mAP برابر 0.5 نتیجه بهتری داشته و میتوان گفت در کل بد عمل نکرده. علت این تفاوت در معماری دو مدل میباشد که یکی دو مرحله ای و دیگری single shot و سریع است.

این نمودار مشابه نمودار قبلی اما واضح تر و به سبک مقاله میباشد.