



دانشگاه تهران
دانشکده مهندسی برق و
کامپیوتر



درس شبکه‌های عصبی و یادگیری عمیق
تمرین سوم

متین بذرافshan - 810100093

شهریار عطار - 810100186

فهرست

1.....	پرسش 1. پیاده‌سازی مدل U-Net
1.....	1-1. آماده‌سازی مجموعه داده
1.....	شکل 1-1. تصاویر نمونه برش مغز به همراه ماسک نشان‌دهنده تومور
2.....	شکل 1-2. توزیع ماسک‌های داده
3.....	2-1. پیاده‌سازی مدل UNet
3.....	مقدمه
5.....	لایه‌های مورد استفاده
5.....	شرح بلوک‌ها
6.....	شکل 1-3. معماری شبکه پیاده‌سازی شده
7.....	پیاده‌سازی
9.....	1-3. داده‌افزایی
11.....	شکل 1-4. نمونه‌های از خروجی افزایش داده
12.....	4-1. بهینه‌سازی، متريک‌ها وتابع هزينه
12.....	سنجه‌ها
14.....	بهينه‌ساز
15.....	تابع هزينه
16.....	5-1. آموزش مدل
16.....	جدول 1-1. پارامترهای مورد استفاده برای آموزش
17.....	شکل 1-5. سنجه‌های داده‌های آموزش و صحت‌سننجی در طول یادگیری
18.....	6-1. ارزیابی مدل
18.....	جدول 1-2. سنجه‌های داده‌های آزمون
19.....	شکل 1-6. يك نمونه ماسک پيش‌بیني شده
19.....	شکل 1-7. يك نمونه ماسک نسبتا پيچيده
19.....	شکل 1-8. يك نمونه عدم تشخيص تومور به درستی
19.....	شکل 1-9. يك نمونه تشخيص كمتر از مقدار واقعی
20.....	شکل 1-10. يك نمونه تشخيص كاملا غلط
21.....	پرسش 2 - تشخيص موجودات زير آب
21.....	2-1. معرفی مدل Faster R-CNN
23.....	2-2. سوالات تشریحی
23.....	1-2-2. مقایسه مدل‌های Region-based CNNs
23.....	R-CNN (Region-based Convolutional Neural Network:) .1
23.....	Fast R-CNN: .2

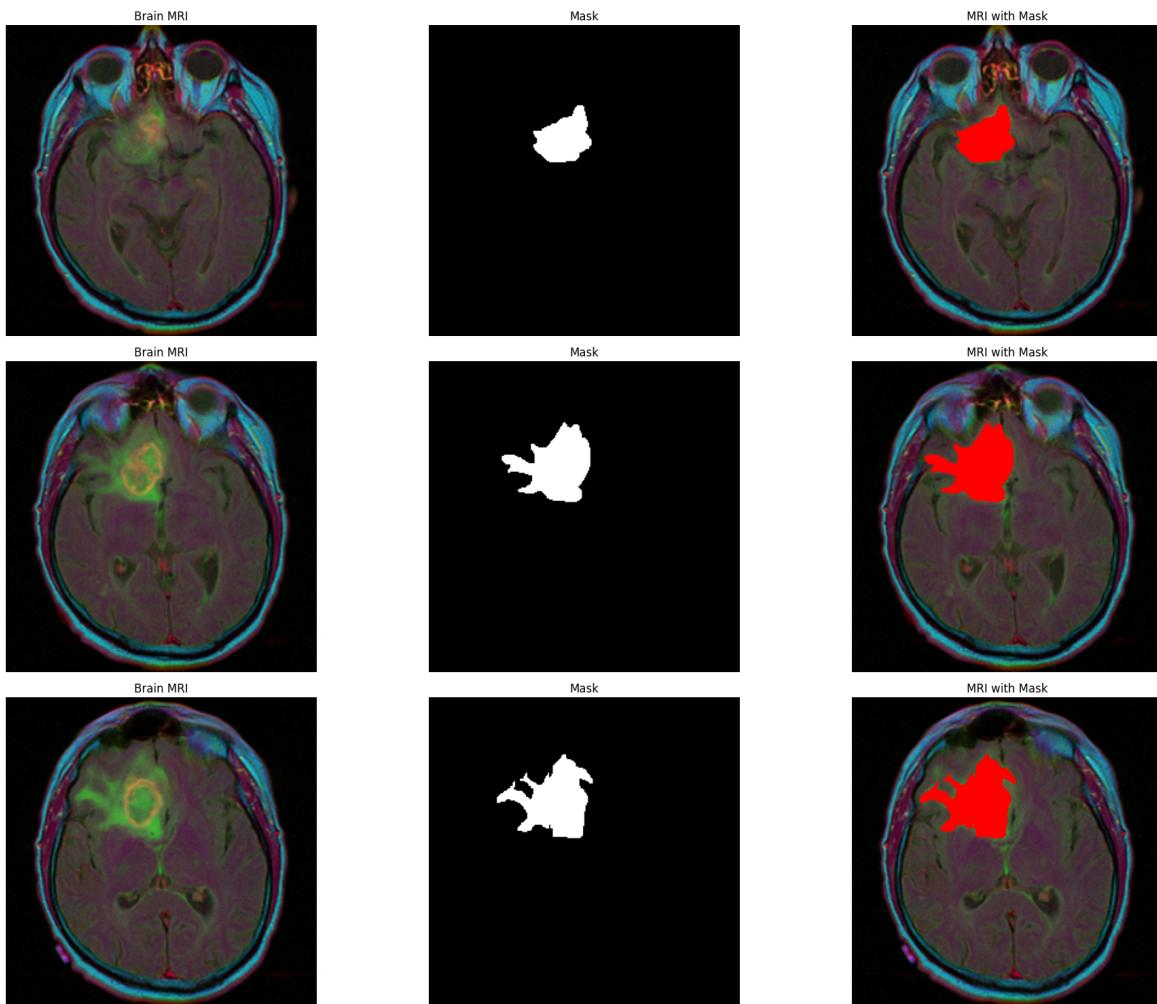
23.....	Faster R-CNN: .3
24.....	2-2-2 مقایسه مدل‌های two-stage و one-stage
24.....	One-Stage Detectors:
25.....	Two-Stage Detectors:
25.....	GIOU, Soft-NMS, OHEM .3-2-2
25.....	OHEM (Online Hard Example Mining:) .1
25.....	Soft-NMS: .2
26.....	GIOU (Generalized Intersection over Union:) .3
26.....	3-2 معرفی مجموعه داده
28.....	EDA و پیش‌پردازش 1-3-2
29.....	شكل 2-1. توزیع تعداد عکس‌ها در سه بخش دیتاست
29.....	شكل 2-2. توزیع حیوانات در عکس‌ها در بخش train دیتاست
30.....	شكل 2-3. توزیع حیوانات در عکس‌ها در بخش valid دیتاست
30.....	شكل 2-4. توزیع حیوانات در عکس‌ها در بخش test دیتاست
30.....	شكل 2-5. توزیع حیوانات در عکس‌ها در کل دیتاست
31.....	شكل 2-6. توزیع کل حیوانات در عکس‌ها در بخش train دیتاست
31.....	شكل 2-7. توزیع کل حیوانات در عکس‌ها در بخش validation دیتاست
32.....	شكل 2-8. توزیع کل حیوانات در عکس‌ها در بخش test دیتاست
32.....	شكل 2-9. توزیع اندازه bounding box-ها در عکس‌ها در کل دیتاست
33.....	شكل 2-10. توزیع اندازه طول و عرض ها در عکس‌ها در بخش train دیتاست
33.....	شكل 2-11. توزیع اندازه طول و عرض ها در عکس‌ها در بخش validation دیتاست
33.....	شكل 2-12. توزیع اندازه طول و عرض ها در عکس‌ها در بخش test دیتاست
34.....	شكل 2-13. توزیع اندازه طول و عرض ها در عکس‌ها در کل دیتاست
34 scatter	شكل 2-14. توزیع نسبت طول و عرض ها در عکس‌ها در کل دیتاست به صورت scatter
35.....	شكل 2-15. توزیع نسبت طول و عرض ها در عکس‌ها در کل دیتاست
36.....	شكل 2-16. نمونه‌هایی از عکس‌های موجود در دیتاست به صورت annotated
38.....	2-3-2 تقویت داده
39.....	شكل 2-17. مراحل تقویت داده به کمک Mosaic Augmentation
46.....	شكل 2-18. نمونه از داده تقویت شده به کمک Mosaic Augmentation
47.....	3-3-2 ساخت DataLoader
52.....	4-2 تعریف مسئله
52.....	2&1-4-2 طراحی معماری Faster R-CNN و طراحی RPN
52.....	شکل 2-19. معماری مدل Faster R-CNN
55.....	3-4-2 آموزش مدل

59.....	شکل 2-20. نمودارهای مربوط به آموزش مدل
60.....	4-4-2 ارزیابی مدل
63.....	شکل 2-21. نمونه خروجی شماره 1
63.....	شکل 2-22. نمونه خروجی شماره 2
64.....	شکل 2-23. نمونه خروجی شماره 3
64.....	شکل 2-24. نمونه خروجی شماره 4
65.....	شکل 2-25. نمونه خروجی شماره 5
65.....	شکل 2-26. نمونه خروجی شماره 6
66.....	شکل 2-27. نمونه خروجی شماره 7
66.....	شکل 2-28. نمونه خروجی شماره 8
67.....	شکل 2-29. نمونه خروجی شماره 9
67.....	شکل 2-30. نمونه خروجی شماره 10

پرسش 1. پیاده‌سازی مدل U-Net

1-1. آماده‌سازی مجموعه داده

مجموعه داده بخش‌بندی¹ MRI² مغز، شامل تصاویر MRI مغز بیماران به همراه ماسک تشخیص ناهنجاری می‌باشد. این مجموعه شامل عکس‌های 110 بیمار است. برای هر بیمار، تعدادی تصویر MRI برش‌های مغزی به همراه ماسک تومور هر برش وجود دارد. ماسک، یک عکس باینری می‌باشد که هر پیکسل آن می‌تواند صفر یا یک باشد. مقدار صفر نشان‌دهنده عدم وجود تومور (یا ناهنجاری) و مقدار یک به معنی وجود تومور می‌باشد.



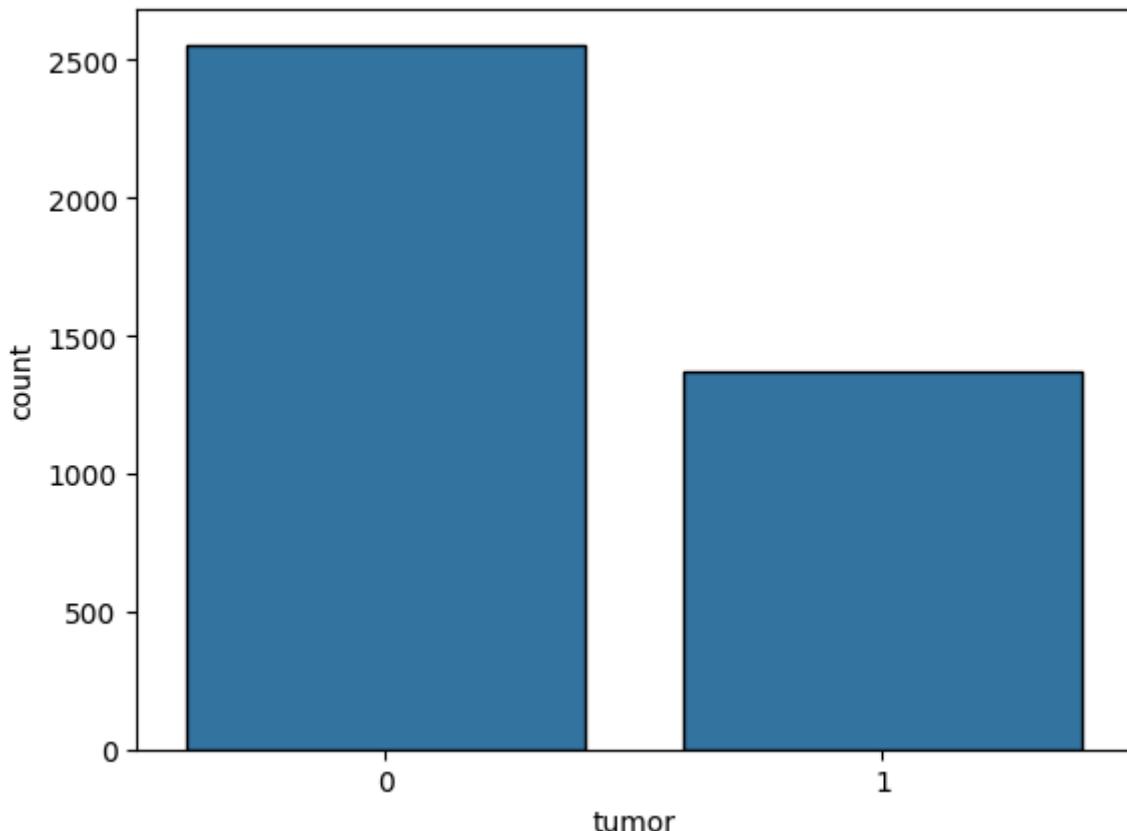
شکل 1-1. تصاویر نمونه برش مغز به همراه ماسک نشان‌دهنده تومور

¹ Segmentation

² Magnetic Resonance Imaging

هر تصویر MRI دارای اندازه $3 \times 256 \times 256$ میباشد و ماسکها ابعاد یکسانی دارند ولی دوبعدی هستند زیرا ماسکها بدون رنگ میباشند ولی تصاویر MRI دارای سه بعد هستند.

برای هر جفت ورودی (تصویر و ماسک) یک برچسب³ نسبت داده ایم که نشان دهنده وجود تومور یا عدم وجود آن در تصویر MRI میباشد. دقت کنید که وجود حتی یک پیکسل تومور در ماسک به وجود تومور در مغز منجر میشود.



شکل 1-2. توزیع ماسکهای داده

توجه داشته باشید، برای خواندن عکس‌ها ابتدا مسیر فایل‌ها را در یک دیتافریم ذخیره میکنیم و سپس از flow from directory استفاده میکنیم. همچنین مطمئن میشویم:

- عکس‌های MRI به سایز $3 \times 128 \times 128$ تغییر یابند.

³ Label

- مقادیر پیکسل‌ها را همانطور که در مقاله آمده است، با روش MinMaxScaling نرمالایز شوند. با توجه به فرمول این روش، چون حداقل مقادیر 255 و حداقل آن 0 می‌باشند، پس کافی است مقادیر بر 255 تقسیم شوند.

$$X_{\text{scaled}} = \frac{X - X_{\min}}{X_{\max} - X_{\min}}$$

- ماسک‌ها به سایز 128×128 تغییر یابند.
- آرایه نظیر هر ماسک با تنظیم آستانه^۴ به اندازه 0.5، فقط دارای مقادیر 0 و 1 باشند.

همانطور که گفته شده است، داده‌ها را به سه بخش آموزش^۵، صحبت سنجی^۶ و آزمون^۷ با نسبت 8 به 1 به 1 تقسیم کردیم.

توجه: دیتاست داده‌شده با دیتاست مورد بحث در مقاله متفاوت می‌باشد. به همین علت در ادامه تفاوت‌هایی در پیاده‌سازی و نتایج وجود دارد.

2-1. پیاده‌سازی مدل

UNet مقدمه

UNet یک معماری شبکه عصبی کانولوشنی (CNN) است که برای وظایف بخش‌بندی در حوزه بینایی ماشین طراحی شده است. این معماری در سال ۲۰۱۵ توسط اولاف رونهبرگر، فیلیپ فیشر و توماس بروکس ارائه شد و از آن زمان تاکنون در بسیاری از برنامه‌های تشخیص تصویر استفاده وسیعی داشته است.

نام "UNet" از ساختار U شکل شبکه گرفته است که شامل یک مسیر رمزگذار و یک مسیر رمزگشای است. مسیر رمزگذار، متضایر با تصویر ورودی، ابعاد فضایی را کاهش داده ویژگی‌ها را از تصویر استخراج می‌کند، در حالی که مسیر رمزگشای، ابعاد فضایی را بازیابی کرده و نقشه بخش‌بندی را تولید می‌کند.

⁴ Threshold

⁵ Train

⁶ Validation

⁷ Test

در ادامه، یک نمای کلی از معماری UNet آورده شده است:

۱. **مسیر رمزگذار**^۸: مسیر رمزگذار تصویر ورودی را دریافت می‌کند و با استفاده از لایه‌های کانولوشنی و لایه‌های پولینگ، ابعاد فضایی آن را به تدریج کاهش می‌دهد. این لایه‌ها ویژگی‌های تصویر را با حفظ اطلاعات فضایی استخراج می‌کنند.

۲. **پل**^۹: در پایین معماری شکل ل، یک اتصال پل وجود دارد که ویژگی‌های با کیفیت بالا یادگرفته شده توسط رمزگذار را نگه می‌دارد. این پل به رمزگشای اجازه می‌دهد در فرایند باز نمودن، به هر دو ویژگی نقشه‌های با کیفیت پایین و با کیفیت بالا دسترسی داشته باشد.

۳. **مسیر رمزگشا**^{۱۰}: مسیر رمزگشا ویژگی‌های کاهش یافته از رمزگذار را دریافت می‌کند و آن‌ها را به اندازه اولیه تصویر ورودی بزرگ می‌کند. هر مرحله از باز نمودن شامل ترکیبی از افزایش اندازه (به عنوان مثال با استفاده از تراکم خطی) و لایه‌های کانولوشنی است. هدف از رمزگشا بازیابی اطلاعات فضایی از دست رفته در فرایند رمزگذاری و تولید نقشه نهایی بخش‌بندی است.

۴. **اتصال‌های پرش**^{۱۱}: UNet از اتصال‌های پرش برای اتصال لایه‌های متناظر رمزگذار و رمزگشا استفاده می‌کند. این اتصالات ویژگی‌های رمزگذار را با ویژگی‌های رمزگشا در همان رزولوشن ارتباط می‌دهند. اتصالات پرش کمک می‌کنند تا جزئیات ریز را حفظ کنند و به شبکه اطلاعات محلی را ارائه دهند تا دقیق بخش‌بندی را بهبود بخشدند.

۵. **خروجی**: خروجی نهایی UNet یک نقشه بخش‌بندی است با همان ابعاد فضایی تصویر ورودی. هر پیکسل در نقشه بخش‌بندی یک برچسب کلاس را نشان می‌دهد که نشان‌دهنده دسته‌بندی پیش‌بینی شده برای پیکسل متناظر در تصویر ورودی است.

UNet به دلیل قابلیت مدیریت همزمان اطلاعات محلی و سراسری به محبوبیت رسیده است. اتصالات پرش به شبکه امکان استفاده از ویژگی‌های با کیفیت پایین و با کیفیت بالا را فراهم می‌کنند، که بهبود جزئیات ریز را حفظ کرده و آگاهی از متناظر های سراسری را حفظ می‌کنند. این معماری با موفقیت در برنامه‌های مختلف بخش‌بندی مانند تشخیص تصاویر پزشکی، تشخیص سلول‌ها و تشخیص اشیاء در صحنه‌های طبیعی مورد استفاده قرار گرفته است.

⁸ Encoder

⁹ Bridge

¹⁰ Decoder

¹¹ Skip Connections

لایه‌های مورد استفاده

1. **لایه ورودی (Input):** این لایه ورودی تصویر را به عنوان ورودی شبکه دریافت می‌کند. ابعاد ورودی به اندازه $(128, 128, 3)$ است که نشان‌دهنده ابعاد تصویر و عمق رنگ آن است.
2. **لایه‌های کانولوشن (Conv2D):** این لایه‌ها عملیات کانولوشن را روی ورودی انجام می‌دهند. هر لایه کانولوشن یک تعدادی فیلتر با اندازه ویژگی خاص خود دارد و با استفاده از تابع فعال‌سازی ReLU، نقاط قوی تصویر را برجسته می‌کند.
3. **لایه حذف تصادفی (Dropout):** این لایه‌ها با احتمال مشخص شده اتصالات موجود در شبکه را به صورت تصادفی غیرفعال می‌کنند. این کار باعث جلوگیری از بیش‌برازش¹² می‌شود.
4. **لایه‌های ادغام حداقل گیر (MaxPooling2D):** این لایه‌ها با استفاده از عملیات حذف نمونه‌ها حداقل گیری ابعاد تصویر را کاهش می‌دهند. این کار باعث می‌شود که ویژگی‌های مهم تصویر حفظ شده و تعداد پارامترها و محاسبات در شبکه کاهش یابد.
5. **لایه‌های افزایش اندازه (UpSampling2D):** این لایه‌ها با استفاده از عملیات ترکیب و افزایش اندازه (UpSampling) ابعاد تصویر را افزایش می‌دهند. این کار باعث می‌شود که اطلاعات دقیق‌تری در مقیاس بزرگ‌تر در دسترس باشد.
6. **لایه‌های الحاق (Concatenate):** این لایه‌ها از دو ورودی مختلف (لایه‌های قبلی) استفاده کرده و آن‌ها را در یک محور مشخص (محور سوم) ترکیب می‌کنند.
7. **لایه خروجی (Sigmoid و Conv2D):** در انتهای، با استفاده از لایه Conv2D با یک فیلتر به ابعاد $(1, 1)$ و تابع فعال‌سازی Sigmoid، خروجی نهایی شبکه تولید می‌شود. این خروجی یک تصویر به ابعاد ورودی است که مقادیر آن بین ۰ و ۱ قرار دارد و نشان می‌دهد که هر پیکسل از تصویر ورودی به چه احتمالی متعلق به دسته‌بندی مورد نظر است.

شرح بلوک‌ها

1. بلوک‌های Encoder:

این بلوک‌ها شامل چهار لایه (شش لایه، اگر تابع فعال‌ساز را جداگانه در نظر بگیریم) می‌باشند. یک لایه کانولوشنی که ابعاد فیلترهای آن 3×3 می‌باشد. با فعال‌ساز ReLU که در ادامه‌ی آن لایه Dropout می‌آید. این لایه‌ها دو بار تکرار می‌شوند. دقت کنید تعداد

¹² Overfitting

فیلترهای لایه‌های کانولوشنی در بلوک‌های مختلف، متفاوت است. همچنین ضریب dropout برابر 0.2 در نظر گرفته شده است.

علت تفاوت نرخ dropout با مقاله، جلوگیری از بیش‌برازش به علت ساده‌تر بودن وظیفه شبکه مورد استفاده می‌باشد.

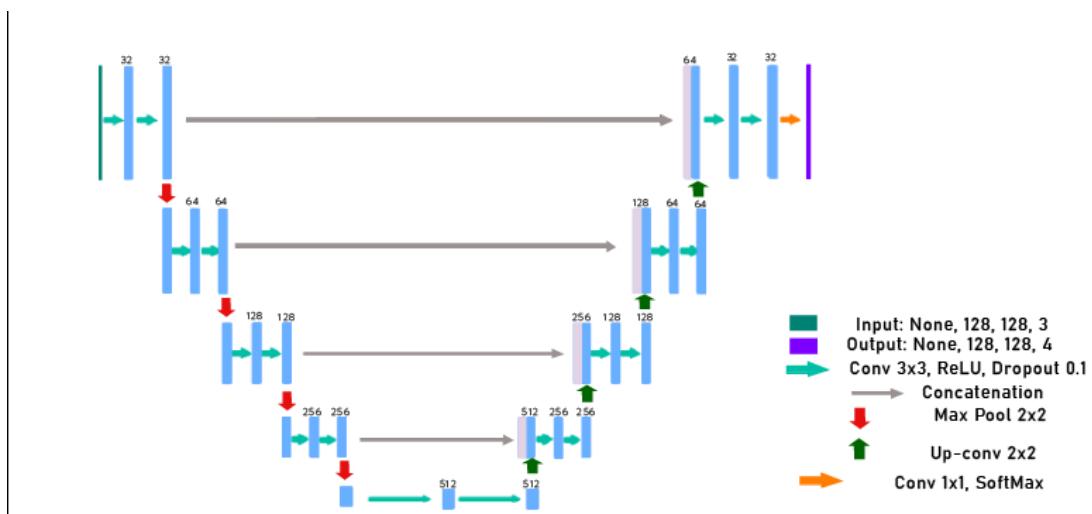
Conv2D -> ReLU -> Dropout -> Conv2D -> ReLU -> Dropout

.2. اتصال بلوک‌های Encoder با یک MaxPooling2D با اندازه 2×2 انجام می‌شود.

.3. بلوک‌های Decoder

این بلوک‌ها در ابتدا با فیچر مپ هم‌ابعادشان در بخش encoding الحق می‌شوند و سپس تعدادی لایه مانند بخش encoder دارند و در انتهای لایه UpSample2D مورد استفاده قرار می‌گیرد.

Concatenate Conv2D -> ReLU -> Dropout -> Conv2D -> ReLU ->
Dropout -> UpSample2D



شکل 1-3. معماری شبکه پیاده‌سازی شده

مدل پیاده‌سازی شده، همان مدل شرح داده شده در مقاله می‌باشد، اما به علت تفاوت در خروجی، تابع فعال‌ساز¹³ لایه آخر شبکه، متفاوت می‌باشد. در دیتاست مقاله، چهار نوع بخش‌بندی برای تومور مغزی وجود دارد اما در دیتاست مورد استفاده، تنها وجود و یا عدم وجود تومور مورد بحث است. بنابراین در شبکه خود در لایه آخر از تابع Sigmoid استفاده کرده‌ایم.

البته توجه داشته باشید، به علت ساده‌تر بودن وظیفه این شبکه در اینجا در مقایسه با وظیفه اصلی اش (چهار نوع بخش‌بندی) می‌توانستیم مدل ساده‌تری را پیاده‌سازی کنیم؛ ولی تصمیم گرفتیم معماری شبکه مقاله را حفظ کنیم و با تنظیم هایپرپارامترها به خواسته خود برسیم.

```
def build_unet(dropout):

    input = Layers.Input((128, 128, 3))
    conv1_1 = Layers.Conv2D(32, 3, activation = 'relu', padding
= 'same')(input)
    drop1_1 = Layers.Dropout(dropout)(conv1_1)
    conv1_2 = Layers.Conv2D(32, 3, activation = 'relu', padding
= 'same')(drop1_1)
    drop1_2 = Layers.Dropout(dropout)(conv1_2)

    pool1 = Layers.MaxPooling2D(pool_size=(2, 2))(drop1_2)
    conv2_1 = Layers.Conv2D(64, 3, activation = 'relu', padding
= 'same')(pool1)
    drop2_1 = Layers.Dropout(dropout)(conv2_1)
    conv2_2 = Layers.Conv2D(64, 3, activation = 'relu', padding
= 'same')(drop2_1)
    drop2_2 = Layers.Dropout(dropout)(conv2_2)

    pool2 = Layers.MaxPooling2D(pool_size=(2, 2))(drop2_2)
    conv3_1 = Layers.Conv2D(128, 3, activation = 'relu', padding
= 'same')(pool2)
    drop3_1 = Layers.Dropout(dropout)(conv3_1)
    conv3_2 = Layers.Conv2D(128, 3, activation = 'relu', padding
= 'same')(drop3_1)
    drop3_2 = Layers.Dropout(dropout)(conv3_2)

    pool3 = Layers.MaxPooling2D(pool_size=(2, 2))(drop3_2)
```

¹³ Activation Function

```

    conv4_1 = Layers.Conv2D(256, 3, activation = 'relu', padding
= 'same')(pool3)
    drop4_1 = Layers.Dropout(dropout)(conv4_1)
    conv4_2 = Layers.Conv2D(256, 3, activation = 'relu', padding
= 'same')(drop4_1)
    drop4_2 = Layers.Dropout(dropout)(conv4_2)

    pool4 = Layers.MaxPooling2D(pool_size=(2, 2))(drop4_2)
    conv5_1 = Layers.Conv2D(512, 3, activation = 'relu', padding
= 'same')(pool4)
    drop5_1 = Layers.Dropout(dropout)(conv5_1)
    conv5_2 = Layers.Conv2D(512, 3, activation = 'relu', padding
= 'same')(drop5_1)
    drop5_2 = Layers.Dropout(dropout)(conv5_2)

    up1 = Layers.Conv2D(256, 2, activation = 'relu', padding =
'same')(Layers.UpSampling2D(size = (2,2))(drop5_2))
    merge1 = Layers.concatenate([drop4_2, up1], axis = 3)
    conv6_1 = Layers.Conv2D(256, 3, activation = 'relu', padding
= 'same')(merge1)
    drop6_1 = Layers.Dropout(dropout)(conv6_1)
    conv6_2 = Layers.Conv2D(256, 3, activation = 'relu', padding
= 'same')(drop6_1)
    drop6_2 = Layers.Dropout(dropout)(conv6_2)

    up2 = Layers.Conv2D(128, 2, activation = 'relu', padding =
'same')(Layers.UpSampling2D(size = (2,2))(drop6_2))
    merge2 = Layers.concatenate([drop3_2, up2], axis = 3)
    conv7_1 = Layers.Conv2D(128, 3, activation = 'relu', padding
= 'same')(merge2)
    drop7_1 = Layers.Dropout(dropout)(conv7_1)
    conv7_2 = Layers.Conv2D(128, 3, activation = 'relu', padding
= 'same')(drop7_1)
    drop7_2 = Layers.Dropout(dropout)(conv7_2)

    up3 = Layers.Conv2D(64, 2, activation = 'relu', padding =
'same')(Layers.UpSampling2D(size = (2,2))(drop7_2))
    merge3 = Layers.concatenate([drop2_2, up3], axis = 3)
    conv8_1 = Layers.Conv2D(64, 3, activation = 'relu', padding
= 'same')(merge3)
    drop8_1 = Layers.Dropout(dropout)(conv8_1)
    conv8_2 = Layers.Conv2D(64, 3, activation = 'relu', padding

```

```

= 'same')(drop8_1)
drop8_2 = Layers.Dropout(dropout)(conv8_2)

up4 = Layers.Conv2D(32, 2, activation = 'relu', padding =
'same')(Layers.UpSampling2D(size = (2,2))(drop8_2))
merge4 = Layers.concatenate([drop1_2, up4], axis = 3)
conv9_1 = Layers.Conv2D(32, 3, activation = 'relu', padding
= 'same')(merge4)
drop9_1 = Layers.Dropout(dropout)(conv9_1)
conv9_2 = Layers.Conv2D(32, 3, activation = 'relu', padding
= 'same')(drop9_1)
drop9_2 = Layers.Dropout(dropout)(conv9_2)

output = Layers.Conv2D(1, (1,1), activation =
'sigmoid')(drop9_2)

return keras.models.Model(inputs = input, outputs = output)

model = build_unet(0.2)

```

۳-۱. داده افزایی^{۱۴}

افزایش داده برای تصاویر و ماسک در بخش‌بندی (Segmentation) فرایندی است که با تغییر تصاویر و ماسک‌های موجود، تنوع بیشتری در مجموعه داده ایجاد می‌کند. این روش به ما کمک می‌کند تا مدل‌های بخش‌بندی را بهبود دهیم، عملکرد آن‌ها را افزایش دهیم و در برخورد با چالش‌های متنوع در تصاویر و ماسک‌ها مقاومت بیشتری داشته باشیم.

افزایش داده با استفاده از روش‌های مختلفی انجام می‌شود. در زیر روش‌های مورد استفاده در افزایش داده برای تصاویر و ماسک‌ها در بخش‌بندی آورده شده است:

```

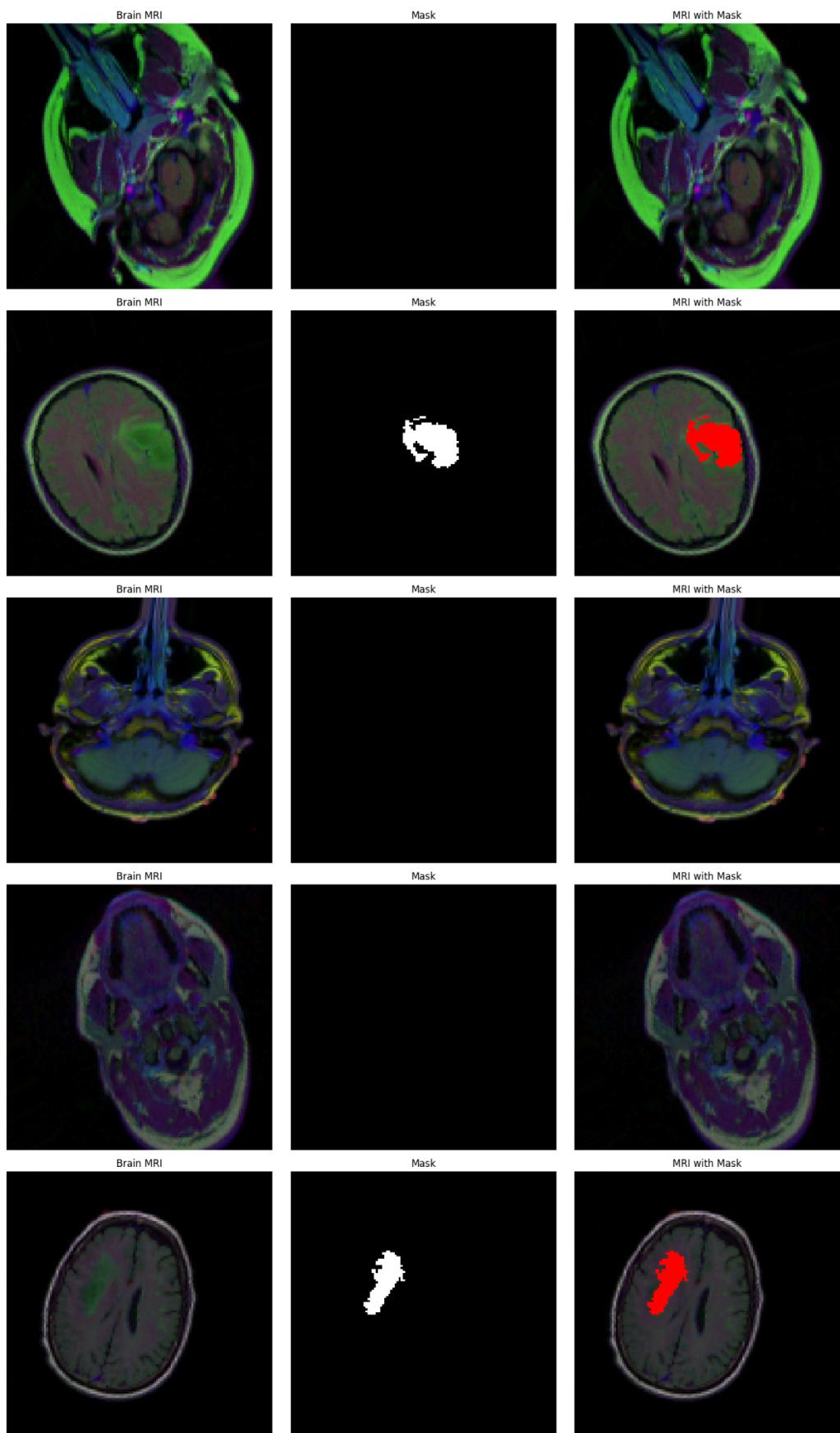
aug_methods = dict(
    rotation_range=45,
    width_shift_range=0.1,
    height_shift_range=0.1,
    shear_range=0.2,
    zoom_range=0.2,

```

^{۱۴} Data Augmentation

```
horizontal_flip=True,  
vertical_flip=True,  
brightness_range=(0.8, 1.2)  
)
```

- **rotation_range**: این پارامتر میزان زاویه چرخش تصاویر را تعیین می‌کند. در اینجا مقدار 45 درجه تنظیم شده است، بنابراین تصاویر به صورت تصادفی می‌توانند به چپ و راست تا 45 درجه چرخش کنند.
- **width_shift_range**: این پارامتر میزان جابجایی افقی تصاویر را تعیین می‌کند. این مقدار بین 0 و 1 است و نسبت به عرض تصویر مشخص می‌کند چقدر می‌تواند تصویر را به صورت تصادفی به سمت چپ یا راست جابجا کند. در اینجا مقدار 0.1 تنظیم شده است.
- **height_shift_range**: این پارامتر میزان جابجایی عمودی تصاویر را تعیین می‌کند. به طریق مشابه با پارامتر قبلی، این مقدار بین 0 و 1 است و نسبت به ارتفاع تصویر مشخص می‌کند چقدر می‌تواند تصویر را به صورت تصادفی به سمت بالا یا پایین جابجا کند. مقدار 0.1 در اینجا تنظیم شده است.
- **shear_range**: این پارامتر میزان انحراف تصاویر را تعیین می‌کند. انحراف به معنای تغییر شکل تصویر با حفظ مساحت است. مقدار 0.2 در اینجا تنظیم شده است، بنابراین تصاویر به صورت تصادفی می‌توانند به سمت راست یا چپ انحراف پیدا کنند.
- **zoom_range**: این پارامتر میزان بزرگنمایی و کوچکنمایی تصاویر را تعیین می‌کند. مقدار 0.2 در اینجا تنظیم شده است، بنابراین تصاویر به صورت تصادفی می‌توانند بزرگتر یا کوچکتر شوند.
- **horizontal_flip**: این پارامتر برای تصمیم گیری درباره تصاویر افقی بازتاب داده شده است. اگر مقدار True باشد، تصاویر به صورت تصادفی ممکن است افقی بازتاب شوند.
- **vertical_flip**: این پارامتر برای تصمیم گیری درباره تصاویر عمودی بازتاب داده شده است. اگر مقدار True باشد، تصاویر به صورت تصادفی ممکن است عمودی بازتاب شوند.
- **brightness_range**: این پارامتر برای تنظیم محدوده روشنایی تصاویر استفاده می‌شود. مقدار (0.8, 1.2) به عنوان یک محدوده برای تنظیم روشنایی انتخاب شده است. در این حالت، تصاویر به صورت تصادفی می‌توانند روشنایی کمی کمتر یا بیشتری داشته باشند.



شکل ۱-۴. نمونه‌های از خروجی افزایش داده

4-1. بهینه‌سازی، متريک‌ها و تابع هزینه

سنجه¹⁵ ها

قبل از شروع توضیح، لازم است با چهار مفهوم زیر آشنا باشیم:

- **تشخیص صحیح مثبت (True Positive - TP):** تعداد نمونه‌هایی که به درستی تشخیص داده شده‌اند و در واقع اعضای مثبت هستند.
- **تشخیص صحیح منفی (True Negative - TN):** تعداد نمونه‌هایی که به درستی تشخیص داده شده‌اند و در واقع اعضای منفی هستند.
- **تشخیص غلط مثبت (False Positive - FP):** تعداد نمونه‌هایی که اشتباهها به عنوان اعضای مثبت تشخیص داده شده‌اند، در حالی که در واقع اعضای منفی هستند.
- **تشخیص غلط منفی (False Negative - FN):** تعداد نمونه‌هایی که اشتباهها به عنوان اعضای منفی تشخیص داده شده‌اند، در حالی که در واقع اعضای مثبت هستند.

این مفاهیم معمولاً در مسائل مربوط به تشخیص الگو، دسته‌بندی و ارزیابی عملکرد مدل‌های یادگیری ماشینی استفاده می‌شوند.

معیارهای Dice Coefficient و Intersection over Union بخش‌بندی هستند که برای اندازه‌گیری میزان تطابق بین نتایج تشخیص و واقعیت مورد استفاده قرار می‌گیرند. در ادامه توضیحی درباره هریک از این معیارها ارائه می‌شود:

1. **IoU:** این معیار میزان همپوشانی بین ناحیه تشخیص شده و ناحیه واقعی را می‌سنجد. ابتدا باید ناحیه اشتراک بین دو بخش را پیدا کنیم و سپس نسبت این ناحیه اشتراک به مجموع ناحیه‌های تشخیص شده و واقعی را محاسبه کنیم. فرمول محاسبه IoU به صورت زیر است:

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

با توجه به مفاهیم گفته شده می‌توان فرمول زیر را نیز ارائه کرد:

$$IoU = \frac{TP}{TP + FN + FP}$$

¹⁵ Metric

که در آن مساحت اتحاد معادل مجموع مساحت ناحیه تشخیص شده و ناحیه واقعی است و مساحت اشتراک معادل مساحت بخش مشترک بین دو ناحیه می‌باشد. مقدار IOU بین ۰ و ۱ قرار می‌گیرد، که ۱ نشان دهنده تطابق کامل بین ناحیه تشخیص شده و ناحیه واقعی است.

```
smooth = 100
def iou(y_true, y_pred):
    intersection = K.sum(y_true * y_pred)
    sum_ = K.sum(y_true + y_pred)
    jac = (intersection + smooth) / (sum_ - intersection + smooth)
    return jac
```

: این معیار نیز میزان همپوشانی بین ناحیه تشخیص شده و ناحیه واقعی را اندازه‌گیری می‌کند. در این معیار، تنها مساحت اشتراک بین دو بخش مورد توجه قرار می‌گیرد و مساحت کلی هر دو بخش در نظر گرفته نمی‌شود. فرمول محاسبه ضریب دایس به صورت زیر است:

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

یا می‌توان نشان داد که:

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

ضریب دایس نیز بین ۰ و ۱ قرار می‌گیرد، که ۱ نشان دهنده تطابق کامل بین ناحیه تشخیص شده و ناحیه واقعی است.

```
smooth = 100
def dice_coef(y_true, y_pred):
    y_truef = K.flatten(y_true)
    y_predf = K.flatten(y_pred)
    And = K.sum(y_truef * y_predf)
    return((2 * And + smooth) / (K.sum(y_truef) + K.sum(y_predf) + smooth))
```

دقیق کنید که در هر دو فرمول برای جلوگیری از تقسیم بر صفر، مقداری به اسم smooth تعریف کرده‌ایم.

بهینه‌ساز^{۱۶}

یک الگوریتم بهینه‌سازی پرکاربرد در یادگیری عمیق است که برای بهروزرسانی وزن‌های شبکه عصبی در فرآیند آموزش استفاده می‌شود. این الگوریتم ترکیبی از روش‌های RMSprop و Momentum است که قابلیت تطبیق با نرخ یادگیری را دارد و در عمل بهبود قابل توجهی را در سرعت و کیفیت آموزش شبکه‌های عمیق ایجاد می‌کند.

در ادامه توضیحی درباره مکانیزم عملکرد Adam ارائه می‌شود:

- **محاسبه گرادیان:** برای هر پارامتر شبکه، گرادیان نسبت به آن محاسبه می‌شود. این گرادیان‌ها نشان‌دهنده جهت و میزان تغییرات وزن‌ها بر اساستابع هدف (تابع هزینه) می‌باشند.
- **تطبیق نرخ یادگیری:** Adam با استفاده از میانگین مومنتوم اول و میانگین مربعات گرادیان، نرخ یادگیری را برای هر پارامتر تطبیق می‌دهد. این تطبیق باعث می‌شود نرخ یادگیری برای هر پارامتر متفاوت باشد و به بهبود کیفیت و سرعت آموزش کمک می‌کند.

```
initial_learning_rate = 3e-4
optimizer = tf.keras.optimizers.Adam(
    learning_rate = initial_learning_rate,
    beta_1 = 0.9,
    beta_2 = 0.999,
    epsilon = None,
    decay = 1e-3/32,
    amsgrad = False
)
```

- **learning_rate:** این پارامتر نرخ یادگیری اولیه را تعیین می‌کند.
- **beta_2 و beta_1:** این‌ها نرخ کاهش نمایی برای اولین و دومین لحظه‌های گرادیان هستند. آن‌ها کنترل‌کننده تأثیر گرادیان‌های گذشته بر بهروزرسانی فعلی هستند.
- **epsilon:** این یک مقدار کوچک است که به نمونه‌ها اضافه می‌شود تا پایداری عددی حفظ شود. اگر به None تنظیم شود، مقدار پیش‌فرض $1e-7$ استفاده می‌شود.

¹⁶ Optimizer

¹⁷ Adaptive Moment Estimation

- این پارامتر نرخ کاهش نرخ یادگیری در طول زمان را تعیین می‌کند. این به عنوان یک ضریب به نرخ یادگیری پس از هر بروزرسانی اعمال می‌شود. در این حالت، مقدار کاهش به $1e-3/32$ تنظیم شده است.

¹⁸تابع هزینه¹⁸

تابع خطای Binary Cross Entropy یک تابع هزینه معروف برای مسائل دسته‌بندی دو کلاسه است که در شبکه‌های عصبی استفاده می‌شود. این تابع معمولاً برای مسائلی که هدف آن‌ها تشخیص دسته‌بندی بین دو کلاس است، مورد استفاده قرار می‌گیرد. در این تابع، خطای بر اساس احتمالاتی که توسط شبکه پیش‌بینی می‌شود و واقعیت دسته‌ها تعیین می‌شود، محاسبه می‌شود.

فرمول تابع خطای Binary Cross Entropy به صورت زیر است:

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

در این فرمول:

- y مقدار واقعی دسته‌ها است که به صورت باینری (0 یا 1) مشخص می‌شود.
- \hat{y} مقدار پیش‌بینی شده توسط شبکه است که نیز به صورت باینری (0 یا 1) است.

این تابع خطای با مقایسه بین واقعیت و پیش‌بینی شبکه، خطای برآورده شده را محاسبه می‌کند. هدف این تابع ایجاد پیش‌بینی‌هایی است که به شکل بهینه واقعیت را تشخیص دهند. مقادیر کوچکتر تابع خطای نشان‌دهنده تطابق بهتر بین پیش‌بینی شبکه و واقعیت است.

¹⁸ Loss Function

5-1. آموزش مدل

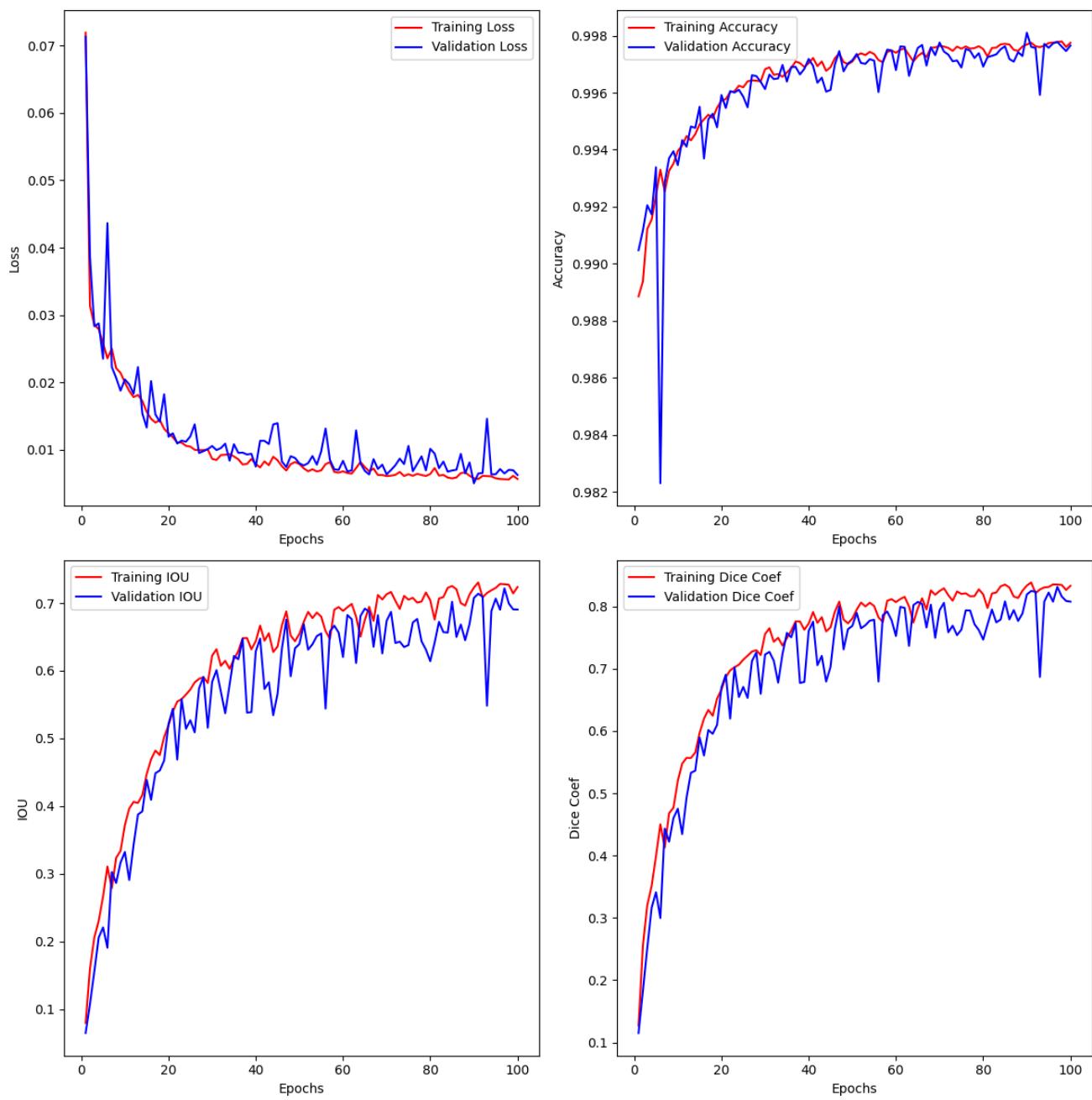
برای آموزش، از $batch\ size = 16$ و تعداد 100 دوره¹⁹ بهره جستیم.²⁰ در کل می‌توانید تمامی پارامترهای مورد استفاده شده را در جدول زیر ببینید.

Number of Epochs	100
Batch Size	16
Optimizer	Adam
Loss	binary Cross Entropy
Metrics	[Binary Accuracy, IOU, Dice Coefficient]

جدول 1-1. پارامترهای مورد استفاده برای آموزش

¹⁹ Epoch

²⁰ دقت کنید که در انتخاب $batch\ size$ با محدودیت حافظه GPU نیز همراه بودیم.



شکل ۱-۵. سنجه‌های داده‌های آموزش و صحتسنجی در طول یادگیری

6-1. ارزیابی مدل

برای ارزیابی مدل خود، از داده‌های آزمون استفاده کردیم. جدول زیر مقادیر سنجه‌ها را برای این داده‌ها نشان می‌دهد.

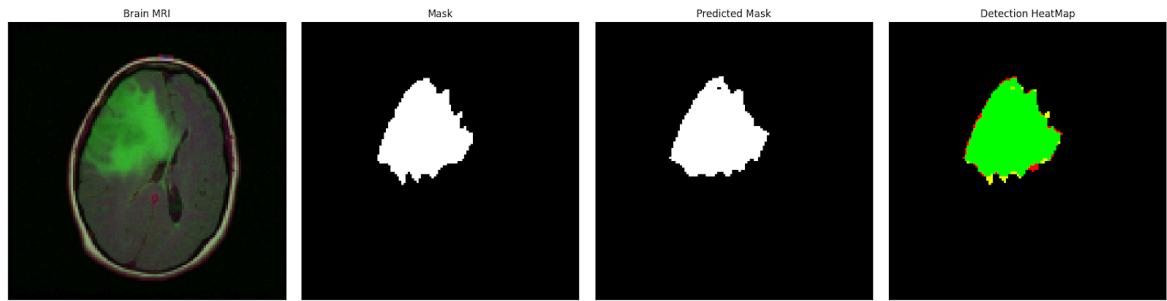
Dice Coefficient	0.78
IOU	0.64
Binary Accuracy	0.99

جدول 1-2. سنجه‌های داده‌های آزمون

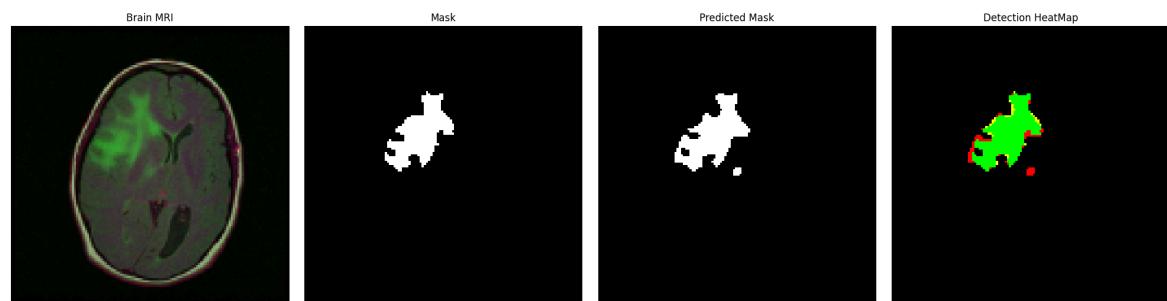
همچنین، پیش‌بینی²¹ را برای تمامی داده‌های تست انجام دادیم و علاوه بر نمایش ماسک پیش‌بینی شده، نقشه‌ای از تشخیص مدل برای پیکسل‌ها به نمایش گذاشتیم. این نقشه شامل:

- پیکسل‌های سبز: نشان‌دهنده تشخیص تومور به درستی توسط مدل می‌باشد. (True Positive)
- پیکسل‌های سیاه: نشان‌دهنده عدم تشخیص تومور به درستی توسط مدل می‌باشد. (True Negative)
- پیکسل‌های زرد: نشان‌دهنده عدم تشخیص تومور به غلط توسط مدل می‌باشد. (False Negative)
- پیکسل‌های قرمز: نشان‌دهنده تشخیص تومور به غلط توسط مدل می‌باشد. (False Positive)

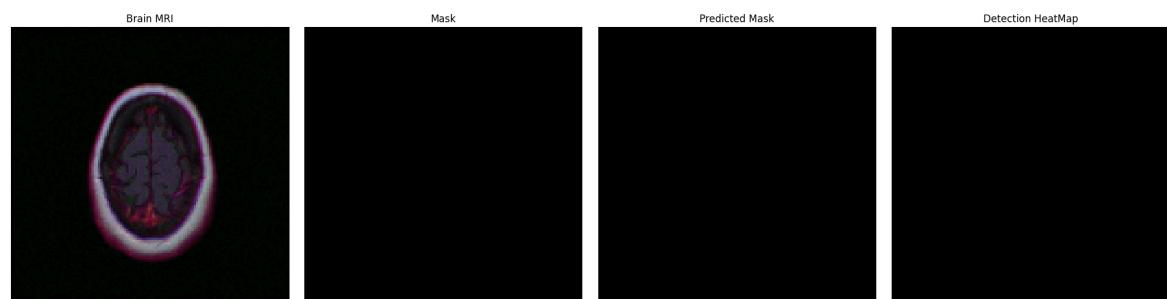
²¹ Predict



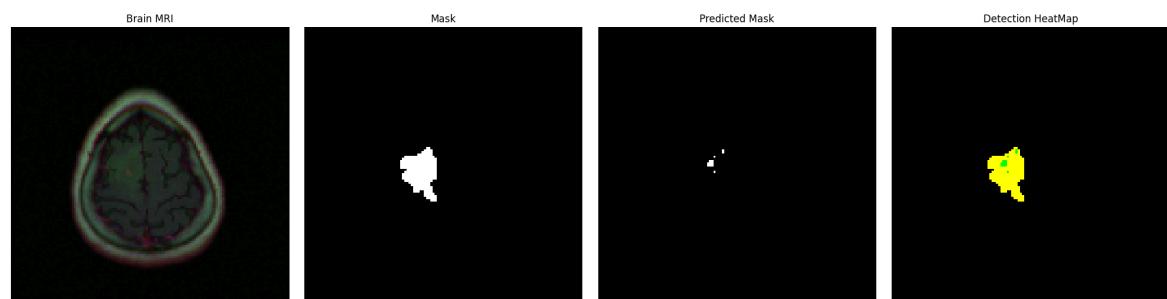
شکل 1-6. یک نمونه ماسک پیش‌بینی شده



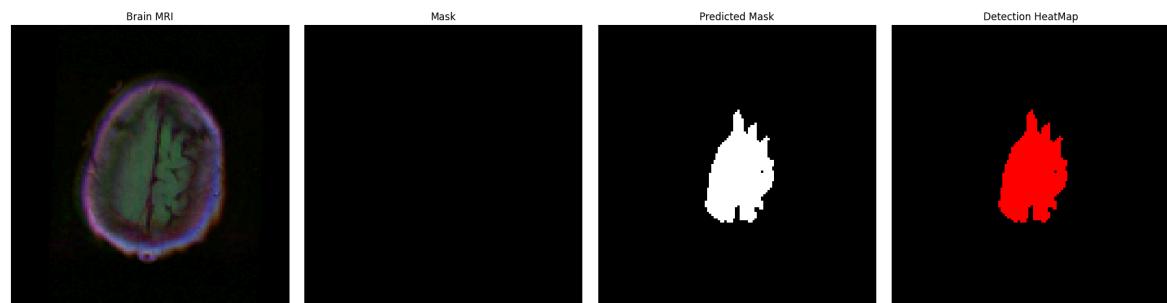
شکل 1-7. یک نمونه ماسک نسبتاً پیچیده



شکل 1-8. یک نمونه عدم تشخیص تومور به درستی



شکل 1-9. یک نمونه تشخیص کمتر از مقدار واقعی



شكل 1-10. یک نمونه تشخیص کاملاً غلط

پرسش 2 - تشخیص موجودات زیر آب

1-2. معرفی مدل Faster R-CNN

مدل Faster R-CNN (Region-based Convolutional Neural Network) یکی از مدل‌های محبوب در زمینه تشخیص اشیا در تصاویر است. این مدل توسط Shaoqing Ren و همکارانش در سال ۲۰۱۵ معرفی شد. Faster R-CNN از یک شبکه عصبی کانولوشنی برای استخراج ویژگی‌ها از تصاویر استفاده می‌کند و با استفاده از یک شبکه کانولوشنی ارائه‌دهنده ویژگی (مانند VGG، ResNet و ...) به نام CNN به دقت بالا و سرعت بالا در تشخیص اشیا در تصاویر می‌رسد.

اصلی‌ترین ویژگی Faster R-CNN این است که از یک نوع ویژگی‌یابی به نام "Region" استفاده می‌کند که برای تولید پیشنهادهای مناسب برای مکان‌های احتمالی اشیا در تصاویر مورد استفاده قرار می‌گیرد. سپس این پیشنهادها به عنوان ورودی به یک مازول دیگر با عنوان "Region of Interest (RoI) Pooling" داده می‌شوند تا ویژگی‌های مربوط به هر منطقه پیشنهادی استخراج شده و سپس با استفاده از لایه‌های کاملاً متصل و یک شبکه دسته‌بندی، اشیا در تصویر دسته‌بندی می‌شوند.

به دلیل استفاده از RPN برای تولید پیشنهادهای مناسب، Faster R-CNN نسبت به روش‌های قبلی مبتنی بر متداول‌ترین شبکه‌های کانولوشنی مانند R-CNN و SPP-Net بهبود چشمگیری را در سرعت و دقت داشته است.

تشخیص اشیا زیر آب یکی از موضوعات مهم در حوزه تشخیص اشیا می‌باشد و دلیل‌های اهمیت آن را می‌توان به شکل زیر طبقه بندی کرد:

1. کاربردهای متعدد و متنوع: تشخیص اشیا زیرآبی در کاربردهای مختلفی مانند اکتشاف و استخراج معادن زیرآبی، نظارت و پایش محیط‌زیست زیرآبی، تحقیقات علمی در زیست‌شناسی دریاهای و اقیانوس‌شناسی، مانیتورینگ سازه‌های زیرآبی مانند لوله‌ها و سکوهای نفتی، و همچنین در کارهای امنیتی مانند تشخیص اشیا ناشناخته در آب استفاده می‌شود.

2. چالش‌های خاص: محیط زیرآبی دارای چالش‌های خاصی مانند تاریکی، شن و ماسه در حرکت، شفافیت مختلف آب در اعمق مختلف، و وجود انواع مختلفی از اشیا با اندازه، شکل، و ویژگی‌های مختلف است. بنابراین، توسعه روش‌های دقیق و قابل اعتماد برای تشخیص اشیا در این محیط اهمیت دارد.

3. اهمیت محیط زیرآبی: بررسی و مطالعه محیط زیرآبی برای درک بهتر از اکوسیستم‌های آبی، مدیریت منابع آبی، حفظ تنوع زیستی، و پیش‌بینی تغییرات محیطی آینده ضروری است. در این راستا، تشخیص اشیا زیر آبی به عنوان یکی از ابزارهای مهم برای جمع‌آوری داده و اطلاعات در محیط زیرآبی مطرح است.

برای اطلاعات بیشتر درباره این مدل و معماری و کاربردهایش می‌توانید به [این لینک](#) مراجعه کنید.

2-2. سوالات تشریحی

1-2-2. مقایسه مدل‌های Region-based CNNs

هر سه مدل از Faster R-CNN و R-CNN، Fast R-CNN هستند و یکی از مزایای Fast R-CNN نسبت به R-CNN بعده سرعت CNN است.

:R-CNN (Region-based Convolutional Neural Network) .1

- معماری: در R-CNN، ابتدا مناطق احتمالی تصاویر با استفاده از یک الگوریتم پیشنهاد دهنده منطقه (مانند Selective Search) استخراج می‌شوند. سپس برای هر منطقه، یک شبکه عصبی کانولوشنی بر روی آن منطقه اعمال می‌شود تا ویژگی‌های مربوط به آن منطقه استخراج شود. در نهایت، از یک شبکه دسته‌بندی SVM برای تشخیص اشیا استفاده می‌شود.
- مشکلات: اجرای کند، زیرا برای هر منطقه احتمالی، یک CNN جداگانه باید اعمال شود که زمان بر است.

:Fast R-CNN .2

- معماری: از یک شبکه عصبی کانولوشنی برای استخراج ویژگی‌ها از تصویر استفاده می‌کند. سپس با استفاده از یک لایه مخصوص به اسم "RoI pooling" ویژگی‌های مربوط به هر منطقه احتمالی استخراج می‌شوند و به یک شبکه دسته‌بندی متصل می‌شوند.
- مزایا: سرعت بهبود یافته نسبت به R-CNN به دلیل استفاده از یک شبکه کانولوشنی برای استخراج ویژگی‌ها.
- مشکلات: باز هم، نیاز به استفاده از الگوریتم پیشنهاد دهنده منطقه مانند Selective Search وجود دارد که زمان بر است.

:Faster R-CNN .3

- معماری: یک مدل end-to-end است که از یک شبکه کانولوشنی به نام "Region Proposal Network (RPN)" برای تولید پیشنهادهای منطقه استفاده

می‌کند. این پیشنهادها سپس به لایه ROI pooling منتقل می‌شوند و با یک شبکه دسته‌بندی متصل می‌شوند.

- مزایا: این معماری بهبود قابل توجهی در سرعت نسبت به مدل‌های قبلی (R-CNN) و Fast R-CNN دارد زیرا فاز پیشنهاد منطقه و استخراج ویژگی‌ها به صورت همزمان انجام می‌شود.
- مشکلات: به طور کلی، Faster R-CNN دارای عملکرد بهتری نسبت به مدل‌های قبلی است و مشکلات کمتری دارد، اما باز هم ممکن است زمان بر باشد.

2-2-2 مقایسه مدل‌های two-stage و one-stage

مدل‌های تشخیص اشیا بر اساس دو دسته معماری Two-Stage و One-Stage دسته‌بندی می‌شوند، هر کدام از این دسته‌بندی‌ها ویژگی‌ها و مشکلات خاص خود را دارند. در ادامه، به مقایسه میان این دو دسته معماری پرداخته و نمونه‌های معروف هر کدام را ذکر می‌کنیم:

:One-Stage Detectors

- معماری: مدل‌های One-Stage به طور مستقیم و بدون نیاز به مرحله پیشنهاد منطقه اقدام به تشخیص اشیا می‌کنند. این مدل‌ها با استفاده از یک شبکه کانولوشنی ابتدایی و یک لایه دسته‌بندی به طور مستقیم اشیا را تشخیص می‌دهند.

- مزایا:
 - سرعت بالا: این مدل‌ها به دلیل عدم نیاز به مرحله پیشنهاد منطقه معمولاً سریع‌تر هستند.
 - садگی: از آنجایی که این مدل‌ها فاز پیشنهاد منطقه را ندارند، ساختار آن‌ها ساده‌تر است.

- معایب:
 - دقت کمتر: به طور کلی، مدل‌های One-Stage دارای دقیقت کمتری نسبت به Two-Stage هستند.

نمونه‌های معروف:

YOLO (You Only Look Once) .I

SSD (Single Shot MultiBox Detector) .II

RetinaNet .III

:Two-Stage Detectors

- معماری: مدل‌های Two-Stage ابتدا با استفاده از یک شبکه کانولوشنی منطقه‌های احتمالی تصویر را پیشنهاد می‌دهند. سپس با استفاده از لایه RoI pooling و یک شبکه دسته‌بندی، اشیا در منطقه‌های احتمالی تشخیص داده می‌شوند.
- مزایا:
 - . دقت بالا: این مدل‌ها معمولاً دقت بالاتری نسبت به One-Stage دارند، به خصوص در مواردی که اشیا کوچک یا متعدد در تصویر وجود دارند.
- معایب:
 - . سرعت کمتر: به دلیل دو مرحله در تشخیص، مدل‌های Two-Stage معمولاً کندتر هستند.

نمونه‌های معروف:

- . I. R-CNN (Region-based Convolutional Neural Network)
- . II. Fast R-CNN
- . III. Faster R-CNN

GIOU, Soft-NMS, OHEM .3-2-2

:OHEM (Online Hard Example Mining) .1

OHEM یک روش برای بالاگذرنمودن نمونه‌های پروپوزال مثبت و منفی در فرایند آموزش مدل‌های تشخیص اشیا است. این روش به جای استفاده از تمامی نمونه‌های منفی، فقط نمونه‌های دشوار (Hard Examples) برای آموزش مدل انتخاب می‌شوند.

:Soft-NMS .2

NMS (Non-Maximum Suppression) Soft-NMS یک نسخه بهبود یافته از الگوریتم است که به جای حذف دقیق نمونه‌های مکرر، امتیاز آن‌ها را کاهش می‌دهد، با کاهش امتیاز نمونه‌های مکرر به جای حذف آن‌ها، اطلاعات بیشتری از تصویر حفظ می‌شود.

:GIOU (Generalized Intersection over Union) .3

GIOU یک معیار اندازه‌گیری تطابق بین دو مستطیل (bounding box) است که از معیار IOU (Intersection over Union) بهبود یافته است. کاربرد این الگوریتم استفاده در الگوریتم‌های تشخیص اشیا و مواردی که نیاز به اندازه‌گیری دقیق تطابق بین مستطیل‌ها دارند می‌باشد.

$$[\text{GIoU} = \text{IoU} - \frac{\text{Area}(B \cup \hat{B}) - \text{Area}(A)}{\text{Area}(B \cup \hat{B})}]$$

در اینجا:

- IoU نشان دهنده Intersection over Union است.
- A و B به ترتیب نشان دهنده دو مستطیل واقعی و پیش‌بینی شده هستند.
- \hat{B} نشان دهنده مستطیلی است که حاصل از گسترش B است که بزرگترین مستطیلی است که هر دو مستطیل A و B درون آن قرار می‌گیرند.

3-2. معرفی مجموعه داده

مجموعه داده استفاده شده شامل عکس‌هایی از حیوانات زیر آب می‌باشد که با نام "Underwater Object Detection" در [این سایت](#) قابل دسترسی می‌باشد. این دیتابست شامل عکس به همراه لیبل‌هایی که موقعیت (bounding box) حیوانات هفت دسته (شامل ماهی، عروس‌دریاچی و ...) می‌باشد که به سه بخش train-validation-test تقسیم شده. مشخصات این مجموعه داده به صورت خلاصه به شکل زیر می‌باشد.

```
train: ./train/images
val: ./valid/images

nc: 7
names: ['fish', 'jellyfish', 'penguin', 'puffin', 'shark',
'starfish', 'stingray']
```

```
# Aquarium > raw-1024
https://public.roboflow.ai/object-detection/aquarium
```

Provided by [Roboflow](<https://roboflow.com>)
License: CC BY 4.0

![CreateML Output](<https://i.imgur.com/s4PgS4X.gif>)

Dataset Details

This dataset consists of 638 images collected by Roboflow from two aquariums in the United States: The Henry Doorly Zoo in Omaha (October 16, 2020) and the National Aquarium in Baltimore (November 14, 2020). The images were labeled for object detection by the Roboflow team (with some help from SageMaker Ground Truth). Images and annotations are released under a Creative Commons By-Attribution license. You are free to use them for any purposes personal, commercial, or academic provided you give acknowledgement of their source.

Class Breakdown

The following classes are labeled: fish, jellyfish, penguins, sharks, puffins, stingrays, and starfish. Most images contain multiple bounding boxes.

![Class Balance](<https://i.imgur.com/lFzeXsT.png>)

```
## Usage
```

The dataset is provided in many popular formats for easily training machine learning models. We have [trained a model with CreateML](<https://blog.roboflow.com/createme/>) (see gif above).

This dataset could be used for coral reef [conservation](<https://blog.roboflow.com/how-this-fulbright-scholar-is-using-computer-vision-to/>), [environmental health monitoring](<https://blog.roboflow.com/using-computer-vision-to-count-fish-populations/>), swimmer [safety](<https://roboflow.com/industries/safety-and-security>), pet analytics, automated feeding, and much more. We're excited to see what you build!

```
Aquarium Combined - v2 raw-1024
```

```
=====
```

This dataset was exported via [roboflow.ai](#) on November 18, 2020 at 7:55 PM GMT

It includes 638 images.

Creatures are annotated in YOLO v5 PyTorch format.

The following pre-processing was applied to each image:

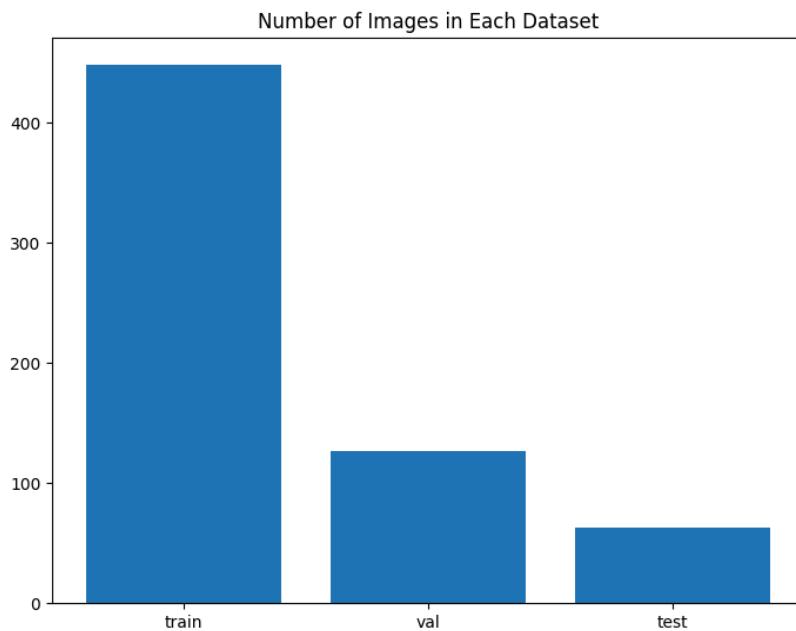
- * Auto-orientation of pixel data (with EXIF-orientation stripping)

- * Resize to 1024x1024 (Fit within)

No image augmentation techniques were applied.

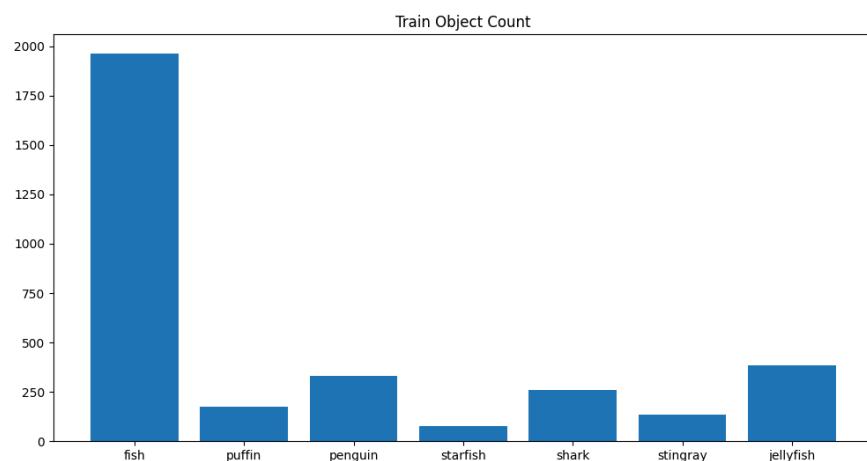
EDA . 1-3-2 و پیشپردازش

ابتدا به بررسی داده مورد نظر می‌پردازیم و کمی بیشتر با دیتاست آشنا می‌شویم. ابتدا توزیع کل عکس‌ها را در سه بخش دیتاست بررسی می‌کنیم:

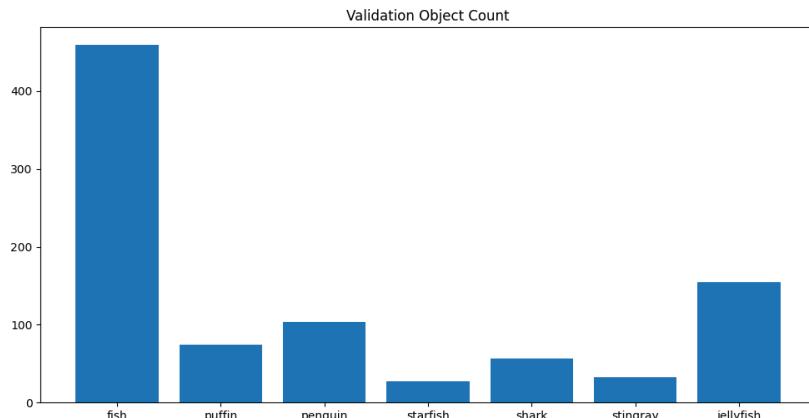


شکل ۱-۲. توزیع تعداد عکس‌ها در سه بخش دیتاست

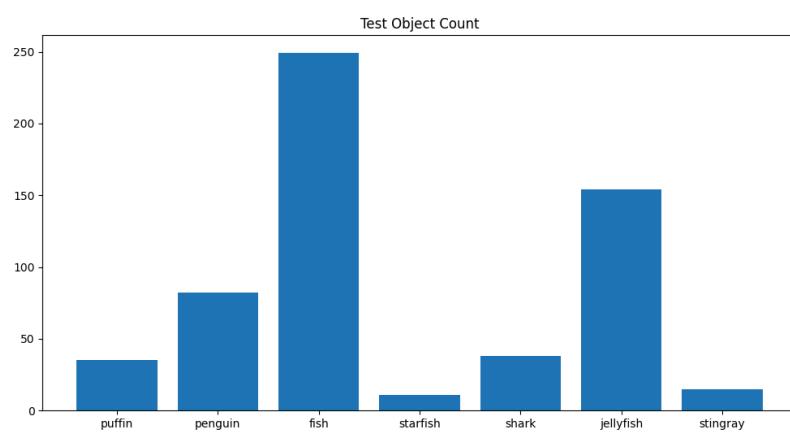
که همانگونه که مشاهده می‌شود و انتظار داشتیم بخش زیادی از دیتاست برای train کردن مدل استفاده می‌شود و بخشی برای validation و بخشی نیز برای test و ارزیابی عملکرد مدل. حال اگر به بررسی تعداد حیوانات موجود در عکس‌ها بپردازیم به نمودار زیر می‌رسیم.



شکل ۲-۲. توزیع حیوانات در عکس‌ها در بخش train دیتاست



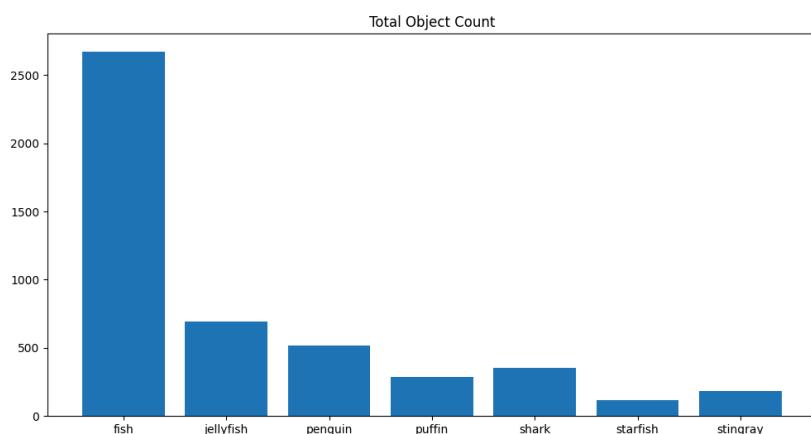
شکل 2-3. توزیع حیوانات در عکس‌ها در بخش valid دیتاست



شکل 2-4. توزیع حیوانات در عکس‌ها در بخش test دیتاست

که همانگونه می‌شود هر سه تقریباً توزیع یکسانی دارند، حال در مجموع اگر بررسی کنیم

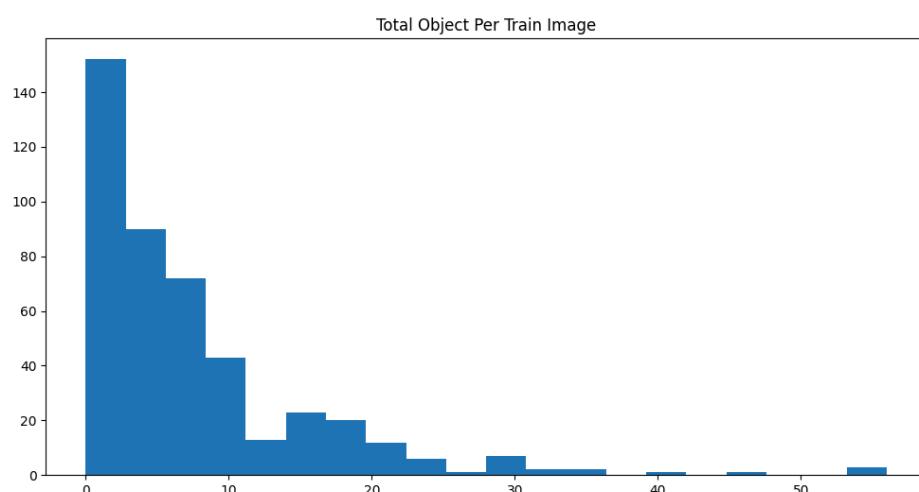
به نمودار زیر می‌رسیم:



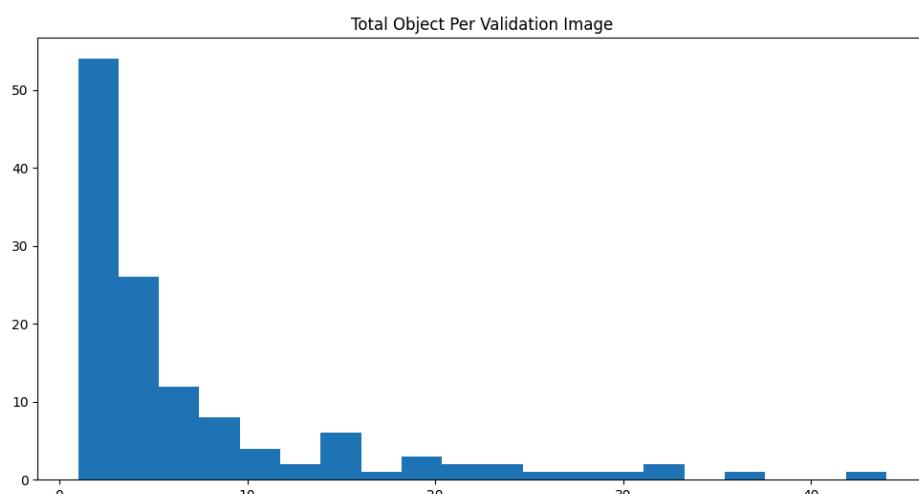
شکل 2-5. توزیع حیوانات در عکس‌ها در کل دیتاست

همانگونه که مشاهده می‌شود دیتاست مورد نظر دارای imbalance می‌باشد، این در عملکرد مدل تاثیر دارد و مدل برای داده‌هایی که دارای عکس ماهی هستند با accuracy و recall خوبی می‌تواند تشخیص دهد اما در بقیه بخش‌ها به دلیل عدم وجود داده کافی ممکن است مدل نتواند به درستی تصمیم بگیرد و به سمت کلاس‌هایی با داده زیاد متمايل شود، برای حل این مشکل می‌توان از راههایی مانند OHEM (که در مقاله به آن اشاره شده) نیز استفاده کرد اما در اینجا پیاده‌سازی نشده.

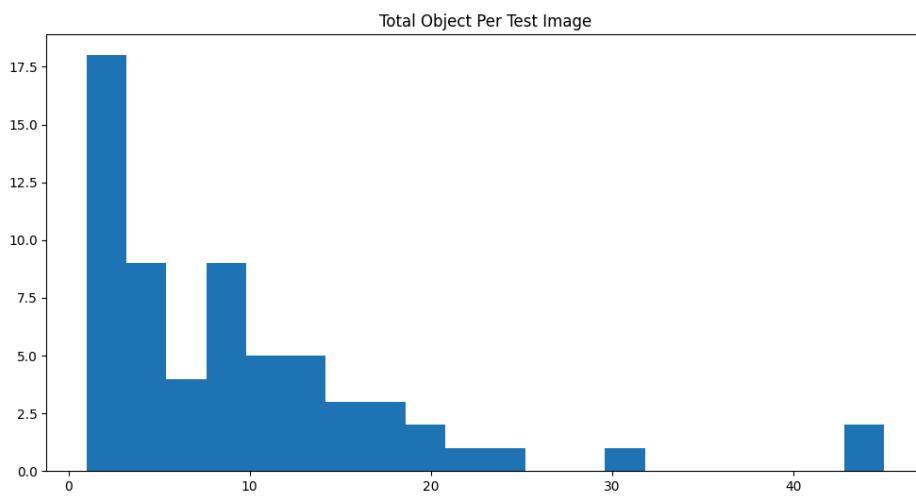
حال به بررسی تعداد کل حیوانات در عکس‌ها می‌پردازیم.



شکل 2-6. توزیع کل حیوانات در عکس‌ها در بخش train دیتاست



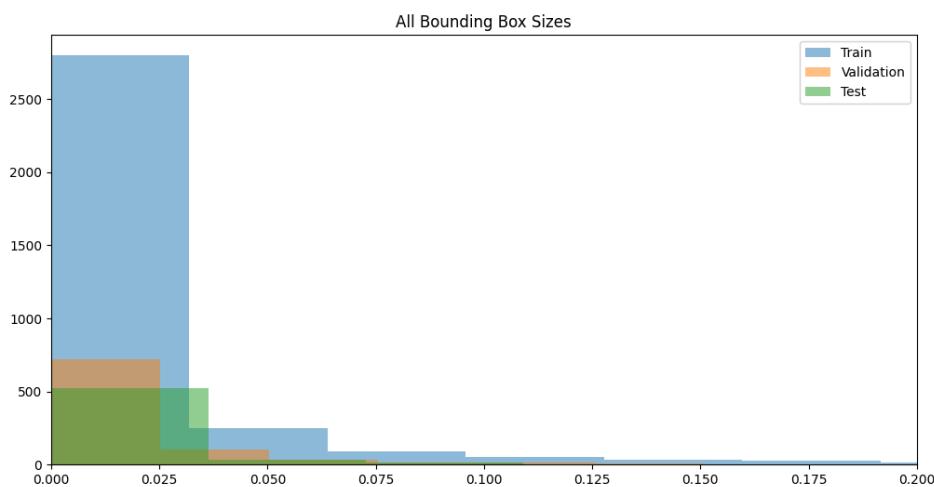
شکل 2-7. توزیع کل حیوانات در عکس‌ها در بخش validation دیتاست



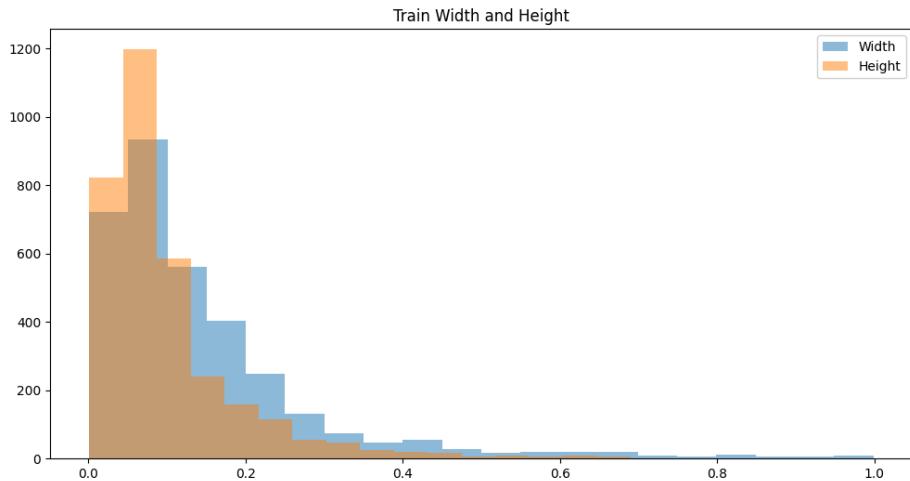
شکل 2-8. توزیع کل حیوانات در عکس‌ها در بخش test دیتاست

باز هم توزیع در این سه بخش تا حد خوبی شبیه به هم می‌باشد. اگر به نمودارها دقت شود مشاهده می‌کنیم که تعداد نسبتاً زیادی از عکس‌ها دارای تعداد بسیار کمی (0، 1 یا 2 تا) حیوان می‌باشند و تعداد عکس‌هایی که تعداد زیادی از حیوانات در آن‌ها باشد به نسبت کمتر است و این کار ما را در آموزش مدل راحت‌تر می‌کند.

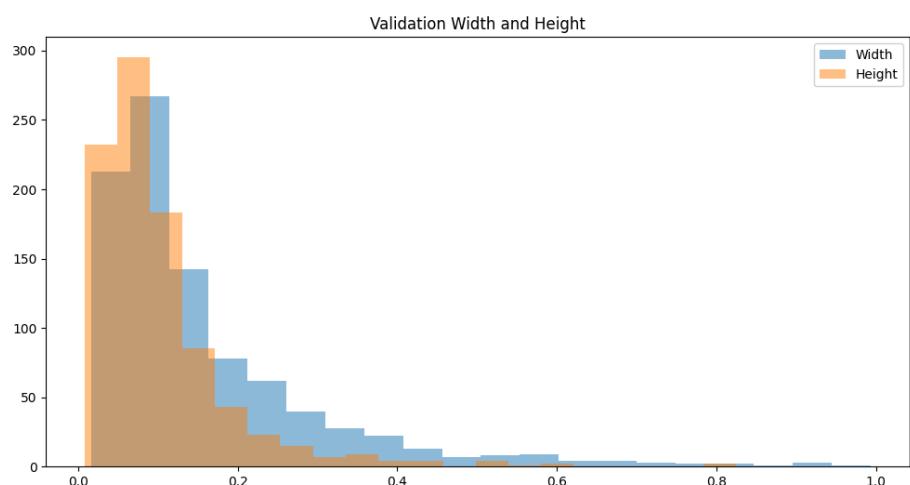
حال از آنجا که در شبکه Faster RCNN ما از تکنیکی به نام anchor generator استفاده می‌کنیم و نیاز داریم که اندازه‌های anchor-ها را مشخص کنیم پس توزیع اندازه bounding-box-ها و نسبت طول و عرض‌ها را نیز بررسی می‌کنیم.



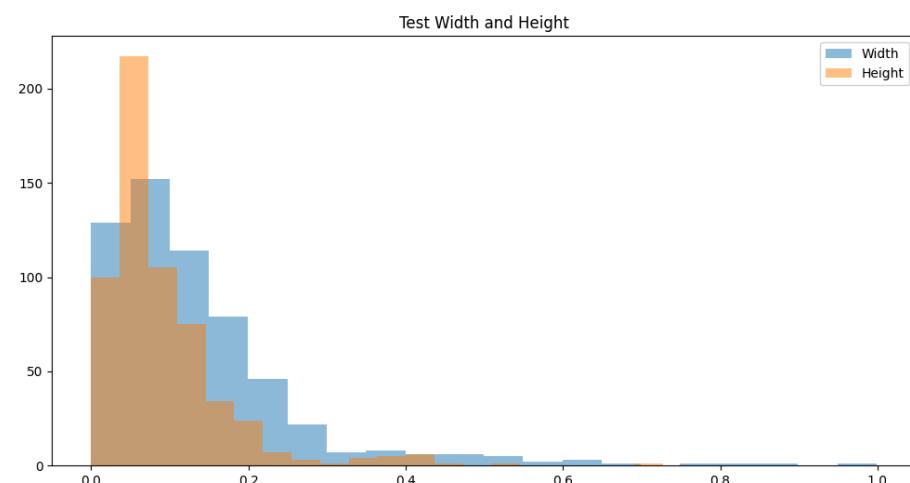
شکل 2-9. توزیع اندازه bounding box-ها در عکس‌ها در کل دیتاست



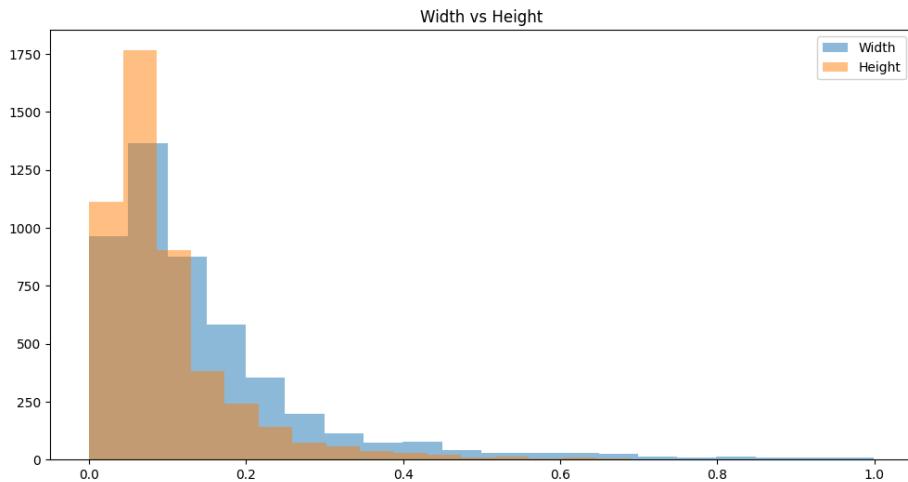
شکل 2-10. توزیع اندازه طول و عرض ها در عکسها در بخش train دیتاست



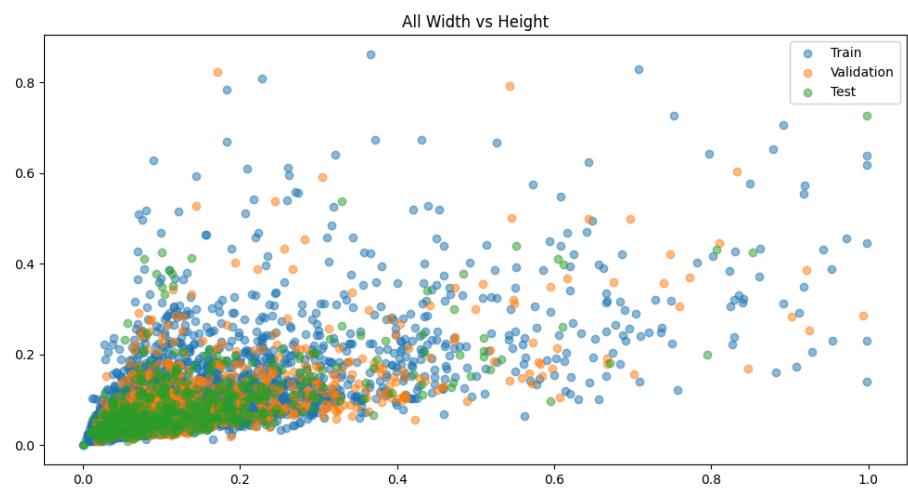
شکل 2-11. توزیع اندازه طول و عرض ها در عکسها در بخش validation دیتاست



شکل 2-12. توزیع اندازه طول و عرض ها در عکسها در بخش test دیتاست

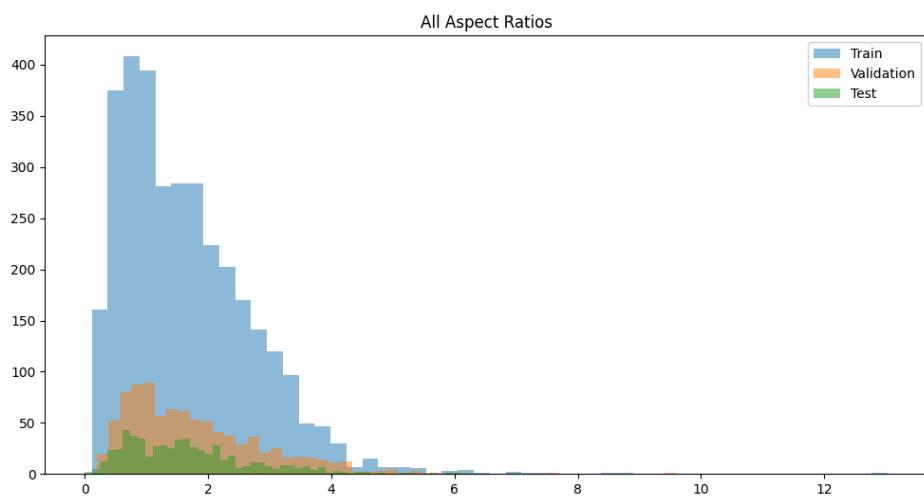


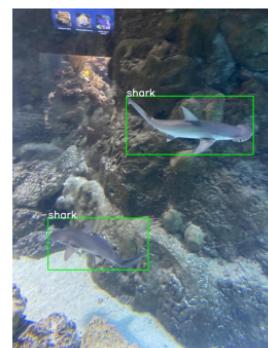
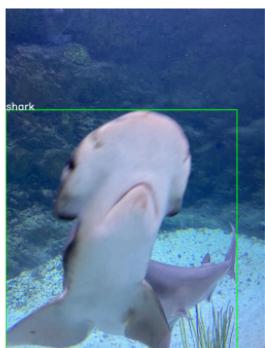
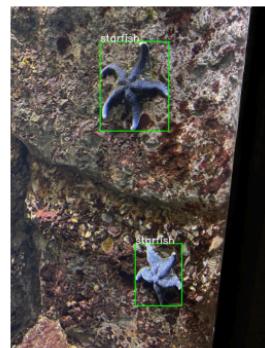
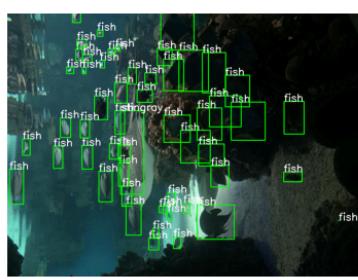
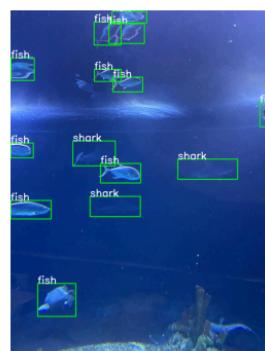
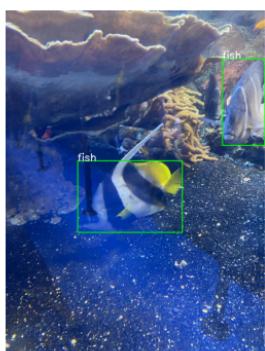
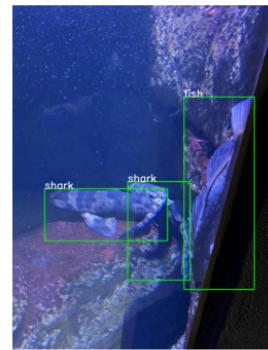
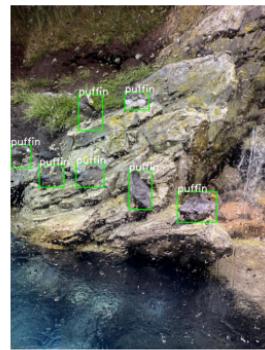
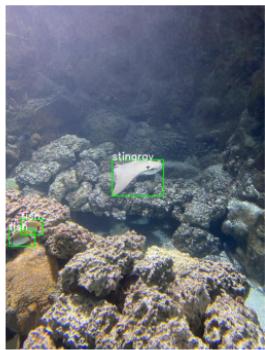
شکل 2-13. توزیع اندازه طول و عرض ها در عکسها در کل دیتاست



شکل 2-14. توزیع نسبت طول و عرض ها در عکسها در کل دیتاست به صورت scatter

همانگونه که از نمودارها می‌توان نتیجه گرفت بیشتر طول و عرض‌ها مقداری کمتر از 0.1 دارند و تقریباً می‌توان از [نمودار شکل 2-14](#) نتیجه گرفت که bounding box ها تقريباً به شكل مربع هستند و حالتی که که مستطیل خیلی کشیده باشد در داده‌ها وجود ندارد. [نمودار شکل 15-2](#) نيز اين فرضيه را تاييد مي‌کند.





شکل 2-16. نمونه‌هایی از عکس‌های موجود در دیتابست به صورت annotated

برای پیش‌پردازش داده‌ها ابتدا آن‌ها را تمیز می‌کنیم و بعد به کمک تکنیک تقویت داده، دیتاست را آماده آموزش مدل می‌کنیم. برای تمیز کردن داده صرفا نیاز است که اندازه آن‌ها را یکی کنیم که برای اینکار به شکل زیر عمل کردیم:

```
def resize_dataset_and_update_labels(image_dir, label_dir,
image_dest, label_dest, img_size=640, square=False):
    image_files = sorted(os.listdir(image_dir))
    for image_file in tqdm.tqdm(image_files):
        image_path = os.path.join(image_dir, image_file)
        image = resize(cv2.imread(image_path), img_size, square)
        cv2.imwrite(os.path.join(image_dest, image_file), image)

        label_path = os.path.join(label_dir, image_file[:-4] +
".txt")
        f = open(label_path, "r")
        lines = f.readlines()
        f.close()

        f = open(os.path.join(label_dest, image_file[:-4] +
".txt"), "w")
        for line in lines:
            class_id, x_center, y_center, width, height =
map(float, line.split())
            x_center *= img_size
            y_center *= img_size
            width *= img_size
            height *= img_size
            f.write("{} {} {} {} {}\n".format(int(class_id),
x_center, y_center, width, height))
        f.close()
```

2-3-2. تقویت داده

برای تقویت داده از شیوه‌های زیر استفاده کردیم (توجه کنید که با توجه به ساختار لیبل‌ها این مرحله باید به دقت انجام شود):

- **RandomHorizontalFlip (وارونگی افقی تصادفی)**: به صورت تصادفی تصویر را افقی می‌چرخانیم، در اینجا باید دقت کنیم که برچسب‌های اشیا در تصویر را نیز به روز کنیم. این تکنیک می‌تواند به شبکه کمک کند تا یاد بگیرد که اشیا را در جهات مختلف تشخیص دهد.
- **RandomVerticalFlip (وارونگی عمودی تصادفی)**: به صورت تصادفی تصویر را عمودی می‌چرخانیم، در اینجا نیز باید دقت کنیم که برچسب‌های اشیا در تصویر را نیز به روز کنیم. این تکنیک می‌تواند به شبکه کمک کند تا یاد بگیرد که اشیا را در جهات مختلف تشخیص دهد.
- **RandomRotation (چرخش تصادفی)**: به صورت تصادفی تصویر را با یک زاویه داده شده می‌چرخانیم، در اینجا نیز مانند مراحل قبلی باید دقت کنیم که برچسب‌های اشیا در تصویر را نیز به روز کنیم. این تکنیک می‌تواند به شبکه کمک کند تا یاد بگیرد که اشیا را در زوایای مختلف تشخیص دهد.
- **RandomResizedCrop (بریدن تصادفی و تغییر اندازه)**: به صورت تصادفی تصویر را به اندازه داده شده می‌بریم و آن را به اندازه اصلی تغییر اندازه می‌دهیم، در اینجا باید دقت کنیم که برچسب‌های اشیا در تصویر را نیز به روز کنیم. این تکنیک می‌تواند به شبکه کمک کند تا یاد بگیرد که اشیا را در اندازه‌های مختلف تشخیص دهد.
- **ColorJitter (تغییر رنگ)**: به صورت تصادفی روشنایی، کنترast، اشباع، و رنگ تصویر را تغییر می‌دهیم. این تکنیک می‌تواند به شبکه کمک کند تا یاد بگیرد که اشیا را در شرایط رنگی مختلف تشخیص دهد.
- **Mosaic Augmentation (تقویت موزاییک)**: تقویت موزاییک یک تکنیک است که چهار تصویر را با انتخاب تصادفی یک تصویر مرکزی و قرار دادن سه تصویر دیگر در اطراف آن به یکی تبدیل می‌کند. این می‌تواند به مدل کمک کند تا یاد بگیرد که اشیا را در زمینه‌های مختلف تشخیص دهد و عملکرد آن را بهبود بخشد. در اینجا نیز باید دقت کنیم که برچسب‌های اشیا در تصویر را به روز کنیم.

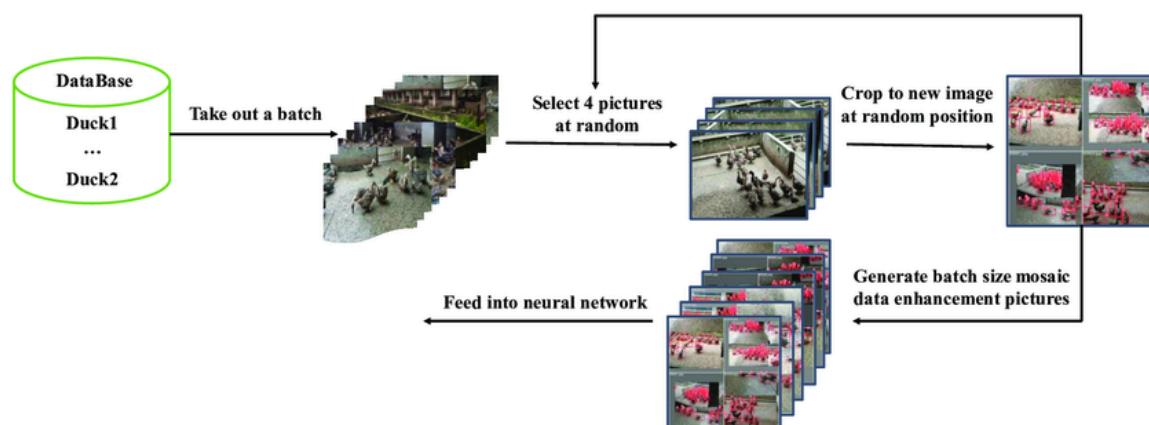
مزایای تقویت موزاییک عبارتند از:

- افزایش تنوع در داده‌های آموزشی: تقویت موزاییک با ترکیب تصاویر مختلف در یک ورودی منفرد، سطح بالاتری از تنوع در مجموعه داده‌های آموزشی را ایجاد

می‌کند. این کمک می‌کند تا مدل بهتر به سناریوها و تغییرات مختلف در داده‌های ورودی کلی شود⁴

معایب تقویت موزاییک عبارتند از:

- محدودیت در نمایش کامل اشیا: روش تقویت موزاییک، با ایجاد طیف وسیعی از تصاویر، ممکن است همیشه مختصات کامل اشیا را نشان ندهد. با این حال، مدل آموزش داده شده با استفاده از این تصاویر می‌تواند به طور سیستماتیک یاد بگیرد تا اشیا با مختصات ناشناخته یا ناقص را تشخیص دهد.



شکل 2-17. مراحل تقویت داده به کمک Mosaic Augmentation

برای پیاده‌سازی این بخش کدهای زیر را نوشتیم:

```

def get_train_aug():
    return A.Compose(
        [
            A.OneOf(
                [
                    A.Blur(blur_limit=3, p=0.5),
                    A.MotionBlur(blur_limit=3, p=0.5),
                    A.MedianBlur(blur_limit=3, p=0.5),
                ],
                p=0.5,
            ),
            A.ToGray(p=0.1),
        ]
    )
  
```

```
A.RandomBrightnessContrast(p=0.1),  
A.ColorJitter(p=0.1),  
A.RandomGamma(p=0.1),  
A.RandomRotate90(p=0.5),  
A.HorizontalFlip(p=0.5),  
A.VerticalFlip(p=0.5),  
A.ShiftScaleRotate(p=0.5),  
ToTensorV2(p=1.0),  
],  
bbox_params=A.BboxParams(  
    format="pascal_voc",  
    label_fields=["labels"],  
,  
)
```

```
def get_train_transform():  
    return A.Compose(  
    [  
        ToTensorV2(p=1.0),  
    ],  
    bbox_params=A.BboxParams(  
        format="pascal_voc",  
        label_fields=["labels"],  
,  
)  
  
def get_valid_transform():  
    return A.Compose(  
    [  
        ToTensorV2(p=1.0),  
    ],  
    bbox_params=A.BboxParams(  
        format="pascal_voc",  
        label_fields=["labels"],  
,  
)  
  
def infer_transforms(image):
```

```
transform = transforms.Compose(  
[  
    transforms.ToPILImage(),  
    transforms.ToTensor(),  
]  
)  
return transform(image)
```

```
def preprocess(image_name, image_dir, label_dir):  
    image_path = os.path.sep.join([image_dir, image_name])  
    name = image_name.split(".")[0]  
    label_name = name + ".txt"  
    label_path = os.path.sep.join([label_dir, label_name])  
  
    return image_path, label_path, label_name
```

```
def random_crop_savebboxes(  
    image_name, image_dir, label_dir, expected_h, expected_w,  
    min_area, min_visibility  
):  
    image_path, label_path, _ = preprocess(image_name,  
    image_dir, label_dir)  
  
    (bboxes, class_labels) = read_label(label_path)  
  
    transform = A.Compose(  
        [A.RandomResizedCrop(expected_h, expected_w)],  
        bbox_params=A.BboxParams(  
            format="yolo",  
            label_fields=["class_labels"],  
            min_area=min_area,  
            min_visibility=min_visibility,  
        ),  
    )
```

```

    transformed = transform(image=read_img(image_path),
bboxes=bboxes, class_labels=class_labels)
    transformed_image = transformed["image"]
    transformed_bboxes = transformed["bboxes"]
    transformed_class_labels = transformed["class_labels"]

    return transformed_image, transformed_bboxes,
transformed_class_labels

```

برای پیاده سازی تقویت داده موزاییک نیز از کد زیر استفاده کردیم:

```

def mosaic(
    image_file_list,
    image_dir,
    label_dir,
    output_image_dir,
    output_label_dir,
    mo_w,
    mo_h,
    scale_x,
    scale_y,
    min_area,
    min_visibility,
    show_image=False,
):
    new_img = np.zeros((mo_h, mo_w, 3), dtype="uint8")

    div_point_x = int(mo_w * scale_x)
    div_point_y = int(mo_h * scale_y)

    for i in range(len(image_file_list)):
        if i == 0:
            w0 = div_point_x
            h0 = div_point_y
            img_0, bboxes_0, class_labels_0 =
random_crop_savebboxes(
                image_file_list[0],
                image_dir,

```

```

        label_dir,
        h0,
        w0,
        min_area,
        min_visibility,
    )
new_img[:div_point_y, :div_point_x, :] = img_0

if len(bboxes_0) == 0:
    bboxes_0_new = []
else:
    bboxes_0_new = np.zeros((len(bboxes_0), 4))
    bboxes_0_new = bboxes_0_new.tolist()

for i, box in enumerate(bboxes_0):
    bboxes_0_new[i][0] = box[0] * scale_x
    bboxes_0_new[i][2] = box[2] * scale_x

    bboxes_0_new[i][1] = box[1] * scale_y
    bboxes_0_new[i][3] = box[3] * scale_y

elif i == 1:
    w1 = mo_w - div_point_x
    h1 = div_point_y
    img_1, bboxes_1, class_labels_1 =
random_crop_savebboxes(
        image_file_list[1],
        image_dir,
        label_dir,
        h1,
        w1,
        min_area,
        min_visibility,
    )
new_img[:div_point_y, div_point_x:, :] = img_1

if len(bboxes_1) == 0:
    bboxes_1_new = []
else:
    bboxes_1_new = np.zeros((len(bboxes_1), 4))
    bboxes_1_new = bboxes_1_new.tolist()

```

```

        for i, box in enumerate(bboxes_1):
            bboxes_1_new[i][0] = box[0] * (1 - scale_x) +
scale_x
            bboxes_1_new[i][2] = box[2] * (1 - scale_x)

            bboxes_1_new[i][1] = box[1] * scale_y
            bboxes_1_new[i][3] = box[3] * scale_y

    elif i == 2:
        w2 = div_point_x
        h2 = mo_h - div_point_y
        img_2, bboxes_2, class_labels_2 =
random_crop_savebboxes(
            image_file_list[2],
            image_dir,
            label_dir,
            h2,
            w2,
            min_area,
            min_visibility,
        )
        new_img[div_point_y:, :div_point_x, :] = img_2

        if len(bboxes_2) == 0:
            bboxes_2_new = []
        else:
            bboxes_2_new = np.zeros((len(bboxes_2), 4))
            bboxes_2_new = bboxes_2_new.tolist()

        for i, box in enumerate(bboxes_2):
            bboxes_2_new[i][0] = box[0] * scale_x
            bboxes_2_new[i][2] = box[2] * scale_x

            bboxes_2_new[i][1] = box[1] * (1 - scale_y) +
scale_y
            bboxes_2_new[i][3] = box[3] * (1 - scale_y)

    else:
        w3 = mo_w - div_point_x
        h3 = mo_h - div_point_y
        img_3, bboxes_3, class_labels_3 =
random_crop_savebboxes(

```

```

        image_file_list[3],
        image_dir,
        label_dir,
        h3,
        w3,
        min_area,
        min_visibility,
    )
new_img[div_point_y:, div_point_x:, :] = img_3

if len(bboxes_3) == 0:
    bboxes_3_new = []
else:
    bboxes_3_new = np.zeros((len(bboxes_3), 4))
    bboxes_3_new = bboxes_3_new.tolist()

for i, box in enumerate(bboxes_3):
    bboxes_3_new[i][0] = box[0] * (1 - scale_x) +
scale_x
    bboxes_3_new[i][2] = box[2] * (1 - scale_x)

    bboxes_3_new[i][1] = box[1] * (1 - scale_y) +
scale_y
    bboxes_3_new[i][3] = box[3] * (1 - scale_y)

new_class_labels = class_labels_0 + class_labels_1 +
class_labels_2 + class_labels_3
new_bboxes = bboxes_0_new + bboxes_1_new + bboxes_2_new +
bboxes_3_new

image_store_path = os.path.sep.join(
[
    output_image_dir,
    + image_file_list[0].split(".")[0]
    + "_"
    + image_file_list[1].split(".")[0]
    + "_"
    + image_file_list[2].split(".")[0]
    + "_"
    + image_file_list[3].split(".")[0]
    + ".jpg",
]

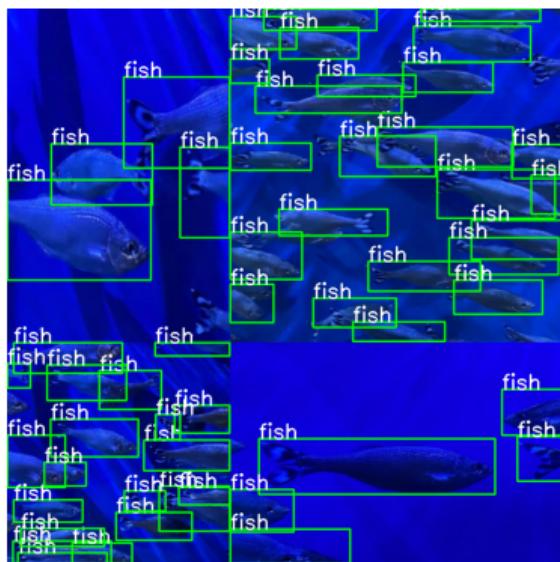
```

```

)
label_store_path = os.path.sep.join(
[
    output_label_dir,
    + image_file_list[0].split(".")[0]
    + "_"
    + image_file_list[1].split(".")[0]
    + "_"
    + image_file_list[2].split(".")[0]
    + "_"
    + image_file_list[3].split(".")[0]
    + ".txt",
]
)
save_img(new_img, image_store_path)
save_label(new_bboxes, new_class_labels, label_store_path)

if show_image:
cv2.imshow("Mosaic Image", new_img)
cv2.waitKey(0)

```



شکل 2-18. نمونه از داده تقویت شده به کمک Mosaic Augmentation

DataLoader .3-3-2 ساخت

برای ساخت دیتالودر برای train و valid به شکل زیر عمل کردیم

```
class CustomDataset(Dataset):
    def __init__(
        self,
        images_path,
        labels_path,
        img_size,
        classes,
        transforms=None,
        use_train_aug=False,
        train=False,
        mosaic=1.0,
        square_training=False,
    ):
        self.transforms = transforms
        self.use_train_aug = use_train_aug
        self.images_path = images_path
        self.labels_path = labels_path
        self.img_size = img_size
        self.classes = classes
        self.train = train
        self.square_training = square_training
        self.mosaic_border = [-img_size // 2, -img_size // 2]
        self.all_image_paths = []
        self.mosaic = mosaic

        for file_type in self.image_file_types:
            self.all_image_paths.extend(
                glob.glob(os.path.join(self.images_path,
file_type))
            )
        self.all_images = [
            image_path.split(os.path.sep)[-1] for image_path in
["*.jpg", "*.jpeg"]
        ]
        self.all_images = sorted(self.all_images)

        self.all_labels = [
            label_path.split(os.path.sep)[-1] for label_path in
```

```

glob.glob(os.path.join(self.labels_path, "*.txt"))
]

def __getitem__(self, idx):
    image = self.all_images[idx]
    label = self.all_labels[idx]
    return image, label

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

def collate_fn(batch):
    """
        To handle the data loading as different images may have
        different number
        of objects and to handle varying size tensors as well.
    """
    return tuple(zip(*batch))

```

عملکرد تابع `collate_fn` که برای رفع مشکل تعداد حیوانات مختلف در یک بچ است به این شکل است که این تابع یک دسته از نمونه‌ها را می‌گیرد، جایی که هر نمونه ممکن است از چندین عنصر تشکیل شده باشد (مانند یک تصویر و برچسب مربوطه). سپس تابع دسته‌ی نمونه‌ها را تغییر شکل می‌دهد تا عناصر هر نمونه را با هم گروه بندی کند، به جای گروه بندی نمونه‌ها با هم.

به عنوان مثال، فرض کنید که یک دسته از نمونه‌ها داریم که هر نمونه از یک tensor تصویر و یک tensor برچسب تشکیل شده است:

```

batch = [
    (image1, label1),
    (image2, label2),
    (image3, label3)
]

```

بعد از اعمال تابع ما می‌توانیم این نتیجه را ببینیم:

```
(  
    (image1, image2, image3),  
    (label1, label2, label3)  
)
```

این تغییر شکل دسته را طوری انجام می‌دهد که تمام تصاویر در یک tuple اولیه گروه‌بندی شده و تمام برچسب‌ها در یک tuple دومیه گروه‌بندی شده است.

```
def create_train_dataset(  
    train_dir_images,  
    train_dir_labels,  
    img_size,  
    classes,  
    use_train_aug=False,  
    mosaic=1.0,  
    square_training=False,  
):  
    train_dataset = CustomDataset(  
        train_dir_images,  
        train_dir_labels,  
        img_size,  
        classes,  
        get_train_transform(),  
        use_train_aug=use_train_aug,  
        train=True,  
        mosaic=mosaic,  
        square_training=square_training,  
    )  
    return train_dataset
```

```
def create_valid_dataset(valid_dir_images, valid_dir_labels,
img_size, classes, square_training=False):
    valid_dataset = CustomDataset(
        valid_dir_images,
        valid_dir_labels,
        img_size,
        classes,
        get_valid_transform(),
        train=False,
        square_training=square_training,
    )
    return valid_dataset

def create_train_loader(train_dataset, batch_size,
num_workers=0, batch_sampler=None):
    train_loader = DataLoader(
        train_dataset,
        batch_size=batch_size,
        num_workers=num_workers,
        sampler=batch_sampler,
        collate_fn=collate_fn,
    )
    return train_loader

def create_valid_loader(valid_dataset, batch_size,
num_workers=0, batch_sampler=None):
    valid_loader = DataLoader(
        valid_dataset,
        batch_size=batch_size,
        shuffle=False,
        num_workers=num_workers,
        sampler=batch_sampler,
        collate_fn=collate_fn,
    )
    return valid_loader
```

سپس با کمک توابع بالا به شکل زیر دیتالودرها را می‌سازیم:

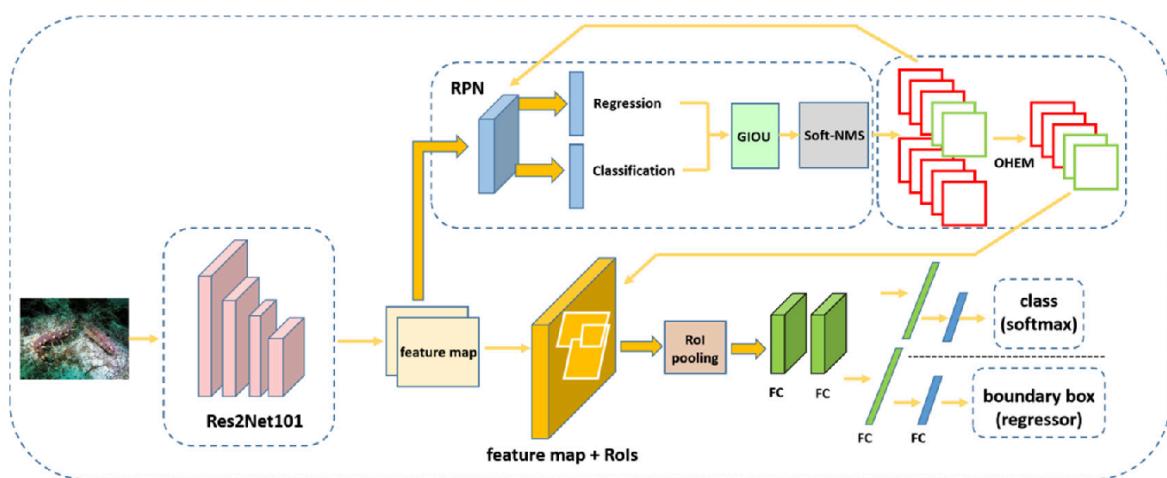
```
train_dataset = create_train_dataset(  
    TRAIN_IMAGES,  
    TRAIN_LABELS,  
    (640, 640),  
    classes,  
    use_train_aug=True,  
    mosaic=1.0,  
)  
  
valid_dataset = create_valid_dataset(  
    VAL_IMAGES,  
    VAL_LABELS,  
    (640, 640),  
    classes,  
)  
  
train_sampler = RandomSampler(train_dataset)  
valid_sampler = SequentialSampler(valid_dataset)  
  
train_loader = create_train_loader(train_dataset, batch_size=2,  
    num_workers=2, batch_sampler=train_sampler)  
valid_loader = create_valid_loader(valid_dataset, batch_size=2,  
    num_workers=2, batch_sampler=valid_sampler)
```

4-2. تعریف مسئله

همانگونه که پیشتر توضیح داده شد ما می‌خواهیم به کمک شبکه مورد نظر به شناسایی حیوانات زیر آب در یک دیتابست بپردازیم و برای این کار از بین مدل‌های Region-Based CNNs را استفاده می‌کنیم.

2.1-4-2. طراحی و معماری Faster R-CNN

برای اینکار از مدل پیشنهادی در شکل زیر استفاده کردیم:



شکل 2-19. معماری مدل Faster R-CNN

برخی از جزئیات پیاده‌سازی (مانند OHEM یا Soft NMS) در کد با شکل بالا تفاوت دارد. مهم‌ترین تفاوت این بخش این است که برای مدل backbone که کار آن استخراج ویژگی‌ها می‌باشد از مدل ResNet101 استفاده کردیم. تکنیک GIOU برای محاسبه هزینه نیز استفاده شده که در شکل بالا توضیح داده شده. در اینجا معماری بخش‌های RPN و بقیه مدل جدا نشده از توابع پیش‌فرض پایتورج استفاده شده که به شکل زیر در آمد: (در این بخش به خاطر اینکه از ساختار اماده مدل استفاده شده بخش جدایی برای RPN موجود نبود تا دو نمونه نمایش داده شود و همچنین در مدل custom نیز که به خاطر امتحان این بخش ساخته شد چون باید از مدل Devolution استفاده می‌شد تا feature map-ها به عکس تبدیل شوند به دلیل پیچیدگی زیاد و ساختار سخت انجام نشده اما ایده این کار به صورت کلی با مدل custom و استفاده از مدل Devolution و اعمال threshold 0.7 می‌باشد)

```

def create_model(num_classes=7):
    model_backbone =
        torchvision.models.resnet101(weights="DEFAULT")
        conv1 = model_backbone.conv1
        bn1 = model_backbone.bn1
        relu = model_backbone.relu
        max_pool = model_backbone.maxpool
        layer1 = model_backbone.layer1
        layer2 = model_backbone.layer2
        layer3 = model_backbone.layer3
        layer4 = model_backbone.layer4
        backbone = nn.Sequential(conv1, bn1, relu, max_pool,
layer1, layer2, layer3, layer4)
        backbone.out_channels = 2048
        # Here, we are using 5x3 anchors.
        # Meaning, anchors with 5 different sizes and 3 different
aspect ratios.
        anchor_generator = AnchorGenerator(
            sizes=((32, 64, 128, 256, 512),), aspect_ratios=((0.5, 1.0,
2.0),))
        )
        roi_pooler = ops.MultiScaleRoIAlign(
            featmap_names=["0"], output_size=7, sampling_ratio=2
        )
    model = FasterRCNN(
        backbone=backbone,
        num_classes=num_classes,
        rpn_anchor_generator=anchor_generator,
        box_roi_pool=roi_pooler,
        rpn_pre_nms_top_n_train=2000,
        rpn_pre_nms_top_n_test=1000,
        rpn_post_nms_top_n_train=2000,
        rpn_post_nms_top_n_test=1000,
        rpn_nms_thresh=0.7,
        )
    return model

```

```

model = create_model(num_classes=7)
optimizer = torch.optim.SGD(model.parameters(), lr=0.005,
momentum=0.9, weight_decay=0.0005)

```

دلیل استفاده از این مقدار k برای anchor-ها نیز در به خاطر نمودارهای بخش اول و تراکم ratio-ها در این سه عدد و همچنین توزیع اندازه bounding box-ها می‌باشد. همچنین می‌توانستیم مدل را از ابتدا پیاده‌سازی کنیم که مانند [این لینک](#) می‌شد که برای راحتی از این کلاس استفاده شده. معماری آن برای پیاده‌سازی به طور خلاصه به شکل زیر می‌باشد: (بالاتر معماری به طور کامل‌تر توضیح داده شده)

:Stage1

در این مرحله، مدل Faster R-CNN ابتدا تصاویر ورودی را به یک شبکه CNN، که به عنوان "backbone network" شناخته می‌شود، ارسال می‌کند و از آنجا که این شبکه ویژگی‌های مخصوصی از تصویر را استخراج می‌کند، یک Feature Map ایجاد می‌شود. سپس، با استفاده از شبکه RPN، از روی این نقشه ویژگی، مناطقی که احتمال وجود اشیا در آنها بالاست، استخراج می‌شود. در فاز آموزش این شبکه، با استفاده از اطلاعات ground truth که بر روی نقشه ویژگی پروژه می‌شوند، ابتدا تلاش می‌شود تا مناطقی که با جعبه‌های مرجع اشیا بیشترین همپوشانی را دارند (overlap)، شناسایی شوند. مناطق با همپوشانی بالا به عنوان مثبت و مناطقی با همپوشانی کم به عنوان منفی مشخص می‌شوند. سپس، RPN سعی می‌کند تا جعبه‌های مرجع را با جعبه‌های واقعی شی تطبیق دهد، که این کار با استفاده از یک تابع خطا که ترکیبی از دو نوع خطا است، انجام می‌شود.

:Stage2

در این مرحله، با استفاده از خروجی‌های مرحله قبل (پیش‌بینی‌های نواحی)، کلاس‌های اشیا در هر ناحیه پیش‌بینی می‌شود. به علت تفاوت در اندازه‌های نواحی، از تکنیک‌هایی مانند ROI Pooling برای رفع این مشکل استفاده می‌شود. همچنین، برای تطبیق نواحی پیشنهادی با جعبه‌های واقعی شی، یک تابع خطا و یک خروجی برای تعیین اختلاف بین آنها تعریف می‌شود، و سپس خطای نهایی از ترکیب این دو خطا به دست می‌آید.

در فاز پیش‌بینی، تصاویر از شبکه اصلی عبور می‌کنند و نقشه ویژگی تولید می‌شود. سپس، با استفاده از RPN، جعبه‌های مرجع را به دست می‌آوریم و فقط جعبه‌هایی که احتمال بیشتری برای حاوی اشیا دارند (مثلاً ۳۰۰ تا با بیشترین احتمال) را انتخاب می‌کنیم و آنها را به مرحله دوم می‌فرستیم تا کلاس‌ها و اختلافات را پیش‌بینی کنند. در انتهای، یک مرحله پردازش پس از پیش‌بینی وجود دارد به نام "non-maximum suppression" که جعبه‌هایی که همپوشانی زیادی دارند را ادغام می‌کند.

3-4-2. آموزش مدل

در این بخش به آموزش مدل می‌پردازیم. پارامترهایی که در بخش آموزش بررسی شدند شامل موارد زیر می‌باشد:

.1. **train_box_loss**: این پارامتر نشان دهنده میزان از دست دادن در آموزش مدل

در بخش محاسبه خطای مربعات مستطیل‌های محدود کننده (bounding box) است. این مقدار از تابع هدف مورد استفاده در فرآیند آموزش مدل به دست می‌آید و نشان دهنده نزدیکی خروجی مدل به اطلاعات واقعی است.

.2. **train_class_loss**: این پارامتر نشان دهنده میزان خطا در آموزش مدل در بخش تشخیص کلاس‌ها است. این مقدار نیز از تابع هدف مورد استفاده در فرآیند آموزش به دست می‌آید و نشان دهنده دقیقیت تشخیص کلاس‌های مختلف توسط مدل است.

.3. **valid_precision**: این پارامتر نشان دهنده دقیقیت مدل در پیش‌بینی‌های انجام شده بر روی داده‌های اعتبارسنجی است. دقیقیت (precision) به معنای تعداد اشیایی است که به درستی تشخیص داده شده‌اند به ازای تمامی اشیایی که مدل به عنوان قسمت مربوط به آن‌ها اعلام کرده است.

.4. **valid_recall**: این پارامتر نشان دهنده نسبت تعداد اشیایی است که به درستی تشخیص داده شده‌اند به تمامی اشیایی واقعی موجود در داده‌های اعتبارسنجی است. این مقدار نشان‌دهنده اندازه‌ی کلی از پوشش مدل است.

.5. **valid_mAP50**: این پارامتر نشان‌دهنده میانگین دقیقیت تشخیص (mean Average Precision) بر روی داده‌های اعتبارسنجی است، که بر اساس معیاری خاص (معمولًاً 50% IoU) است ولی در اینجا از 50% GIoU استفاده شده محاسبه می‌شود.

.6. **valid_mAP50_95**: این پارامتر نشان دهنده میانگین دقیقیت تشخیص بر روی داده‌های اعتبارسنجی است، که بر اساس معیاری خاص (معمولًاً از 50% تا 95% IoU) که در اینجا از GIoU استفاده شده) محاسبه می‌شود.

.7. **valid_box_loss**: مشابه **train_box_loss**، این پارامتر نشان دهنده میزان از دست دادن در بخش محاسبه خطای مربعات مستطیل‌های محدود کننده است، اما بر روی داده‌های اعتبارسنجی محاسبه می‌شود.

.8. **valid_class_loss**: مشابه **train_class_loss**، این پارامتر نشان دهنده میزان خطا در بخش تشخیص کلاس‌ها بر روی داده‌های اعتبارسنجی است.

```

train_box_loss = []
train_class_loss = []
valid_precision = []
valid_recall = []
valid_mAP50 = []
valid_mAP50_95 = []
valid_box_loss = []
valid_class_loss = []

for epoch in range(10):

    model.train()
    for images, targets in train_loader:
        images = list(image.to(device) for image in images)
        targets = [{k: v.to(device) for k, v in t.items()} for t in
targets]

        loss_dict = model(images, targets)
        losses = sum(loss for loss in loss_dict.values())

        predicted_boxes = loss_dict["boxes"]
        target_boxes = [t["boxes"] for t in targets]
        giou_loss = GIOU(predicted_boxes, target_boxes)
        losses += giou_loss

        train_box_loss.append(loss_dict["loss_box_reg"].item())

    train_class_loss.append(loss_dict["loss_classifier"].item())

    optimizer.zero_grad()
    losses.backward()
    optimizer.step()

    model.eval()
    for images, targets in valid_loader:
        images = list(image.to(device) for image in images)
        targets = [{k: v.to(device) for k, v in t.items()} for t in
targets]

        loss_dict = model(images, targets)
        losses = sum(loss for loss in loss_dict.values())

```

```

predicted_boxes = loss_dict["boxes"]
target_boxes = [t["boxes"] for t in targets]
giou_loss = GIOU(predicted_boxes, target_boxes)
losses += giou_loss

valid_box_loss.append(loss_dict["loss_box_reg"].item())

valid_class_loss.append(loss_dict["loss_classifier"].item())

images = list(image.to(device) for image in images)
targets = [{k: v.to(device) for k, v in t.items()} for t in
targets]

precision = []
recall = []
AP50 = []
AP50_95 = []

with torch.no_grad():
    outputs = model(images)

for i in range(len(targets)):
    target = targets[i]
    output = outputs[i]
    pred_boxes = output["boxes"]
    pred_scores = output["scores"]
    pred_labels = output["labels"]

    true_boxes = target["boxes"]
    true_labels = target["labels"]

    for label in range(7):
        true_boxes_label = true_boxes[true_labels ==
label]
        pred_boxes_label = pred_boxes[pred_labels ==
label]
        pred_scores_label = pred_scores[pred_labels ==
label]

        if len(true_boxes_label) == 0:
            precision.append(0)
            recall.append(0)

```

```

        AP50.append(0)
        AP50_95.append(0)
        continue

        if len(pred_boxes_label) == 0:
            precision.append(0)
            recall.append(0)
            AP50.append(0)
            AP50_95.append(0)
            continue

            iou = GIOU(true_boxes_label, pred_boxes_label)
            iou_thresholds = torch.linspace(0.5, 0.95,
10).to(device)
            true_positives =
            torch.zeros(len(iou_thresholds)).to(device)
            false_positives =
            torch.zeros(len(iou_thresholds)).to(device)
            false_negatives =
            torch.zeros(len(iou_thresholds)).to(device)

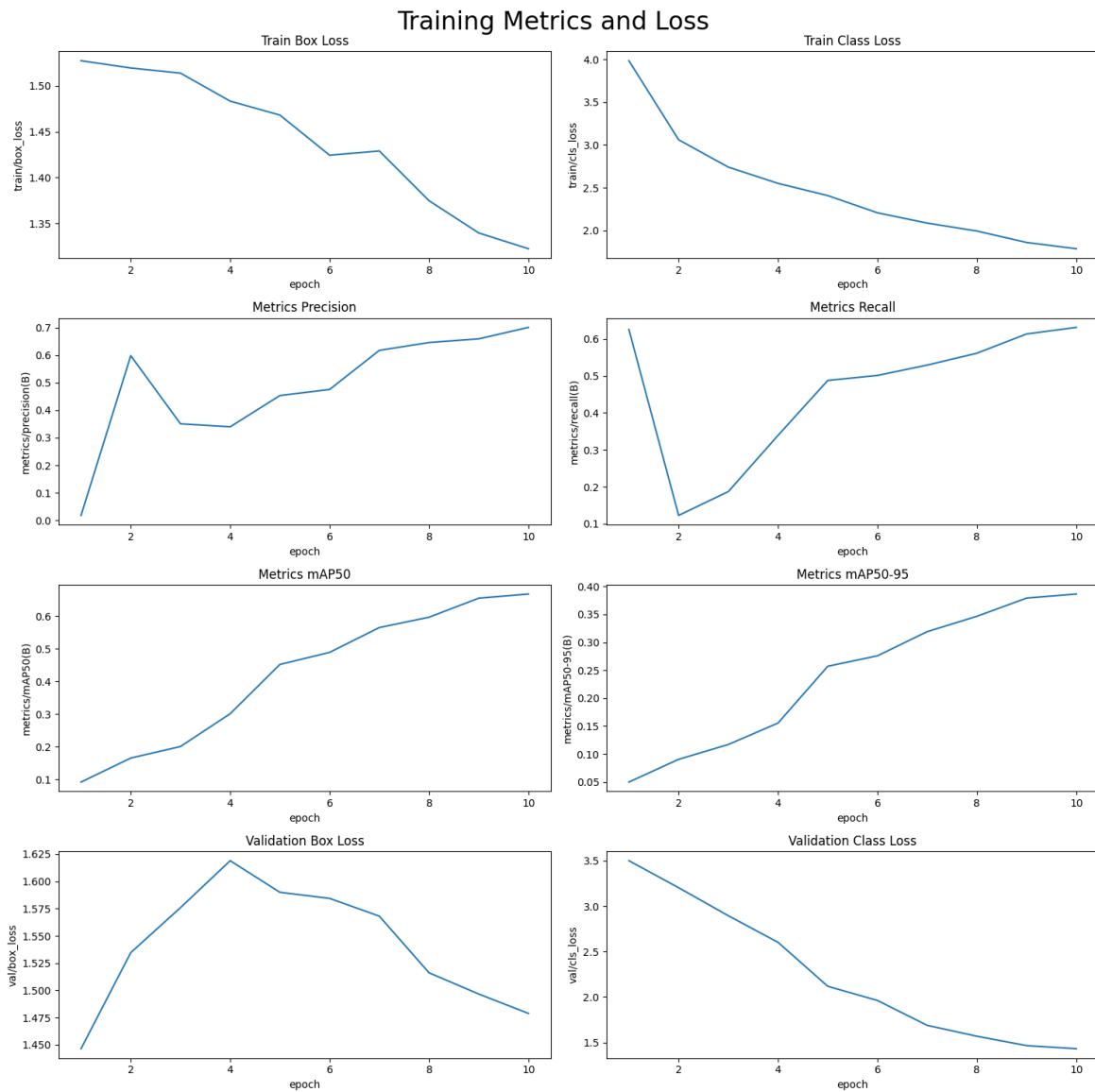
            for i, iou_threshold in
enumerate(iou_thresholds):
                true_positives[i] = torch.sum(iou >
iou_threshold)
                false_positives[i] = torch.sum(iou <=
iou_threshold)
                false_negatives[i] = len(true_boxes_label) -
true_positives[i]

                precision.append(true_positives /
(true_positives + false_positives))
                recall.append(true_positives / (true_positives +
false_negatives))

            AP50.append((precision[-1] * recall[-1]).mean())
            AP50_95.append((precision[-1] *
recall[-1]).mean())
            valid_precision.append(precision)
            valid_recall.append(recall)
            valid_mAP50.append(AP50)
            valid_mAP50_95.append(AP50_95)

```

که خروجی آن به شکل زیر می‌باشد:



شکل 2-20. نمودارهای مربوط به آموزش مدل

همانگونه که مشاهده می‌شود مدل به خوبی در حال آموزش می‌باشد و تمام پارامترها در حال converge کردن می‌باشند. دقت مدل در همه بخش‌ها در حال افزایش می‌باشد و میزان تابع هزینه هم برای بخش regression هم classification هم در حال کاهش می‌باشد. برای تابع هزینه از تابع پیش فرض مدل Faster R-CNN استفاده شده که به شکل زیر می‌باشد.

$$L(\{p_i\}, \{t_i\}) = \frac{1}{N_{cls}} \sum_i L_{cls}(p_i, p_i^*) + \lambda \frac{1}{N_{reg}} \sum_i p_i^* I_{reg}(t_i, t_i^*)$$

در این مدل، تابع هزینه یا خطا از دو بخش تشکیل شده است: خطای دسته‌بندی (Bounding Box Regression Loss) و خطای بازگشت به جعبه‌بندی (Classification Loss). این دو خطا با هم ترکیب می‌شوند تا تابع نهایی هزینه را تشکیل دهند.

.1 **Classification Loss**: این خطا، خطای دسته‌بندی بین دو کلاس (شی در مقابل عدم شی) است. برای محاسبه این خطا، از تابع خطای لگاریتمی (Log Loss) استفاده می‌شود.

.2 **Bounding Box Regression Loss**: این خطا، خطای محاسبه مکان دقیق شی در تصویر است. برای محاسبه این خطا، از تابع خطای L1 استفاده می‌شود.

4-4-2. ارزیابی مدل

حال خروجی مدل را بررسی می‌کنیم. برای این کار از کد زیر استفاده کردیم:

```
num = 0
for image, target in test_loader:
    image = list(image.to(device) for image in image)
    target = [{k: v.to(device) for k, v in t.items()} for t in target]

    with torch.no_grad():
        output = model(image)

    class_ids = output[0]["labels"]
    boxes = output[0]["boxes"]
    scores = output[0]["scores"]

    axs = plt.subplots(1, 2, figsize=(15, 15))

    original_image =
    np.copy(cv2.imread(os.path.join(TEST_IMAGES, image[0])))
    original_image = cv2.cvtColor(original_image,
cv2.COLOR_BGR2RGB)

    original_labels = open(os.path.join(TEST_LABELS,
target[0]["labels"][0])).readlines()
    original_boxes = [list(map(float, label.split()[1:])) for
label in original_labels]
```

```

for box in original_boxes:
    h, w, _ = original_image.shape
    x_min = int((box[0] - box[2] / 2) * w)
    y_min = int((box[1] - box[3] / 2) * h)
    x_max = int((box[0] + box[2] / 2) * w)
    y_max = int((box[1] + box[3] / 2) * h)

    cv2.rectangle(original_image, (x_min, y_min), (x_max,
y_max), (0, 255, 0), 2)
    cv2.putText(
        original_image,
        Idx2Label[int(box[0])],
        (x_min, y_min),
        cv2.FONT_HERSHEY_SIMPLEX,
        fontScale=1,
        color=(255, 255, 255),
        thickness=2,
    )

detected_image = np.copy(original_image)
detected_image = cv2.cvtColor(detected_image,
cv2.COLOR_RGB2BGR)

for i in range(len(boxes)):
    box = boxes[i]
    class_id = class_ids[i]
    score = scores[i]

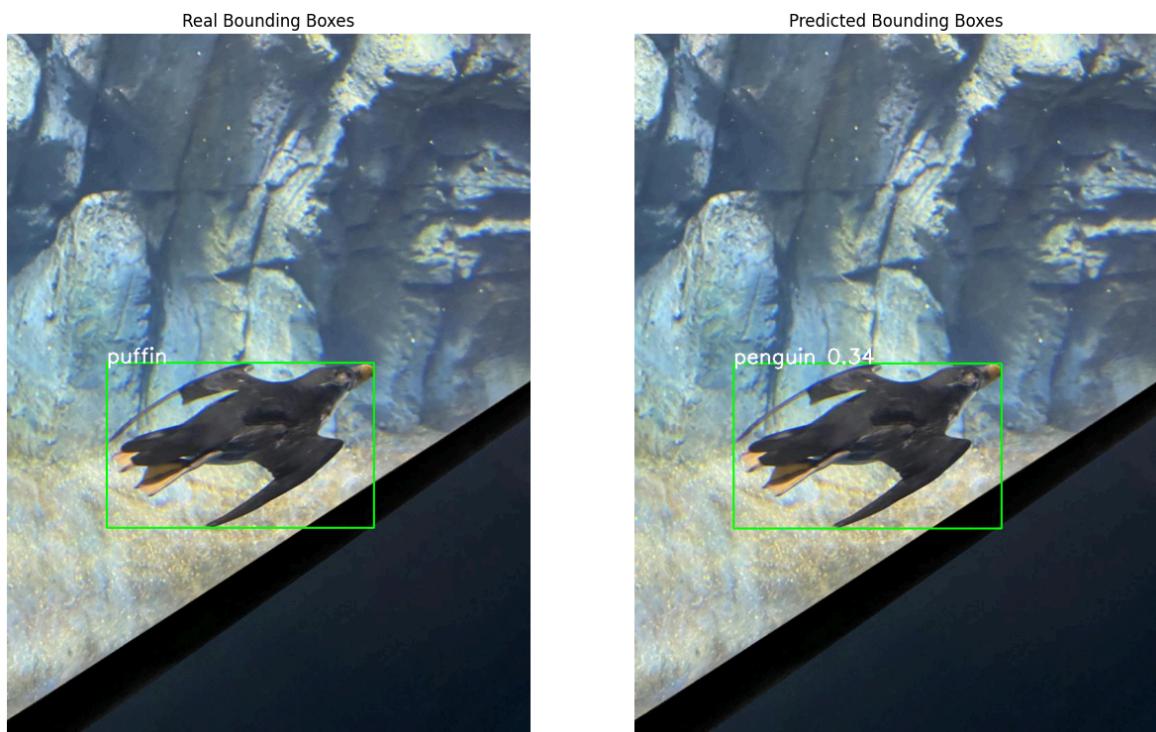
    h, w, _ = original_image.shape
    x_min = int((box[0] - box[2] / 2) * w)
    y_min = int((box[1] - box[3] / 2) * h)
    x_max = int((box[0] + box[2] / 2) * w)
    y_max = int((box[1] + box[3] / 2) * h)

    cv2.rectangle(detected_image, (x_min, y_min), (x_max,
y_max), (0, 255, 0), 2)
    cv2.putText(
        detected_image,
        Idx2Label[int(class_id)] + " " + str(score),
        (x_min, y_min),
        cv2.FONT_HERSHEY_SIMPLEX,
        fontScale=1,
    )

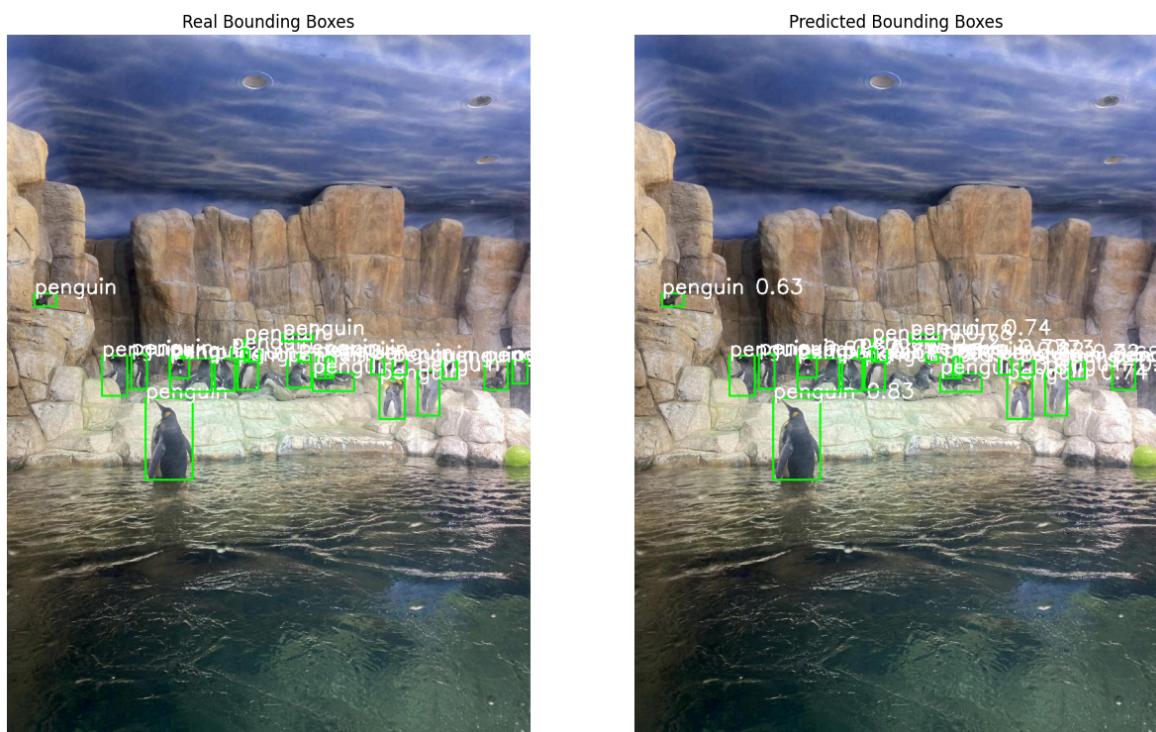
```

```
        color=(255, 255, 255),  
        thickness=2,  
    )  
  
    plt.subplot(1, 2, 1)  
    plt.title("Real Bounding Boxes")  
    plt.imshow(original_image)  
    plt.axis("off")  
  
    plt.subplot(1, 2, 2)  
    plt.title("Predicted Bounding Boxes")  
    plt.imshow(detected_image)  
    plt.axis("off")  
  
    num += 1  
  
    if num == 10:  
        break
```

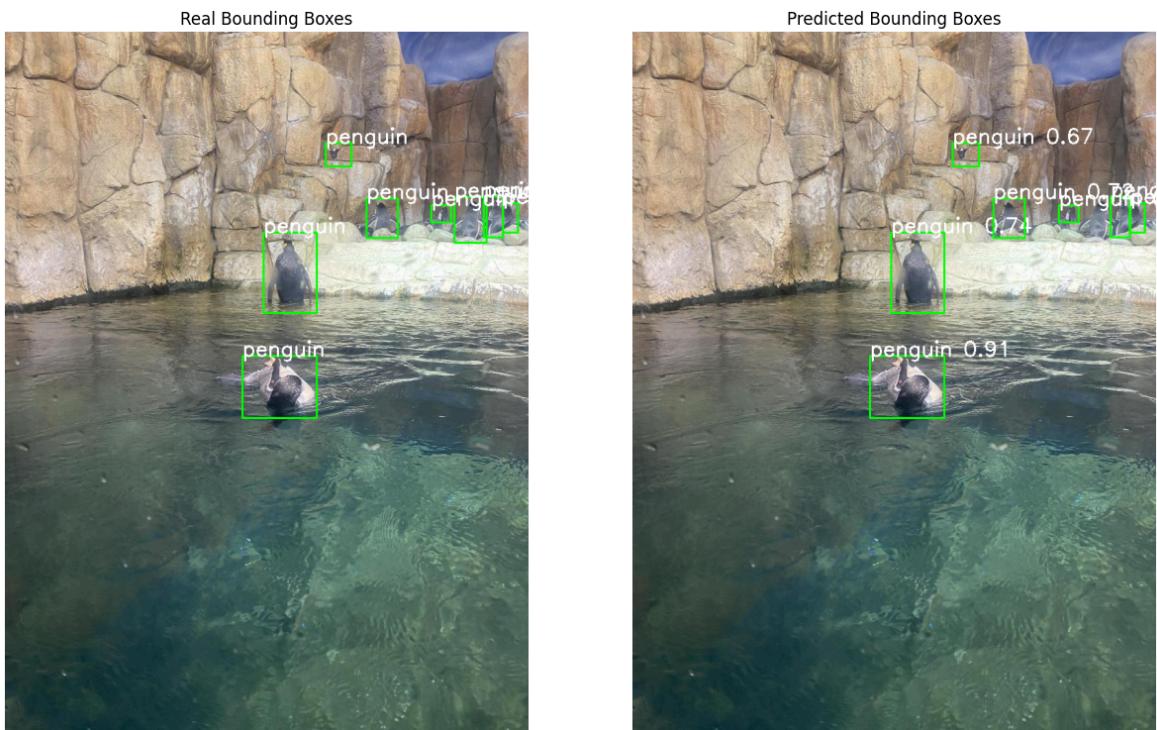
خروجی‌ها را برای ده نمونه از تست می‌توانید در عکس‌های صفحه بعد مشاهده کنید.



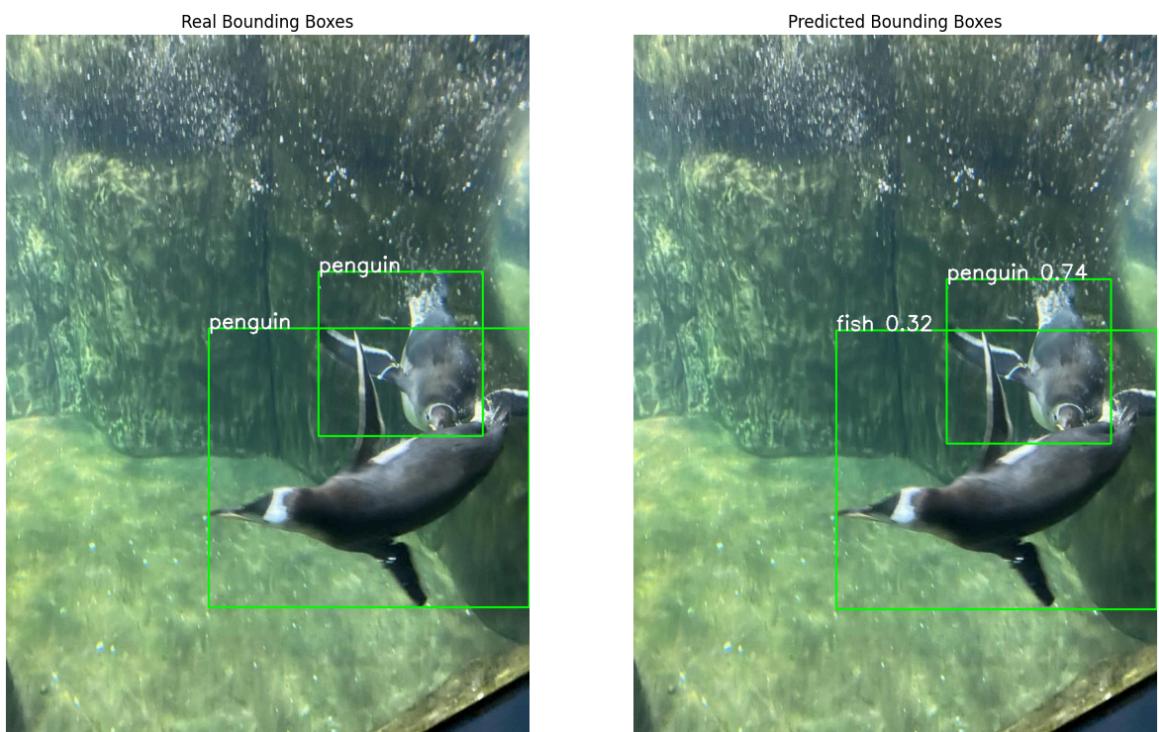
شکل 2-21. نمونه خروجی شماره 1



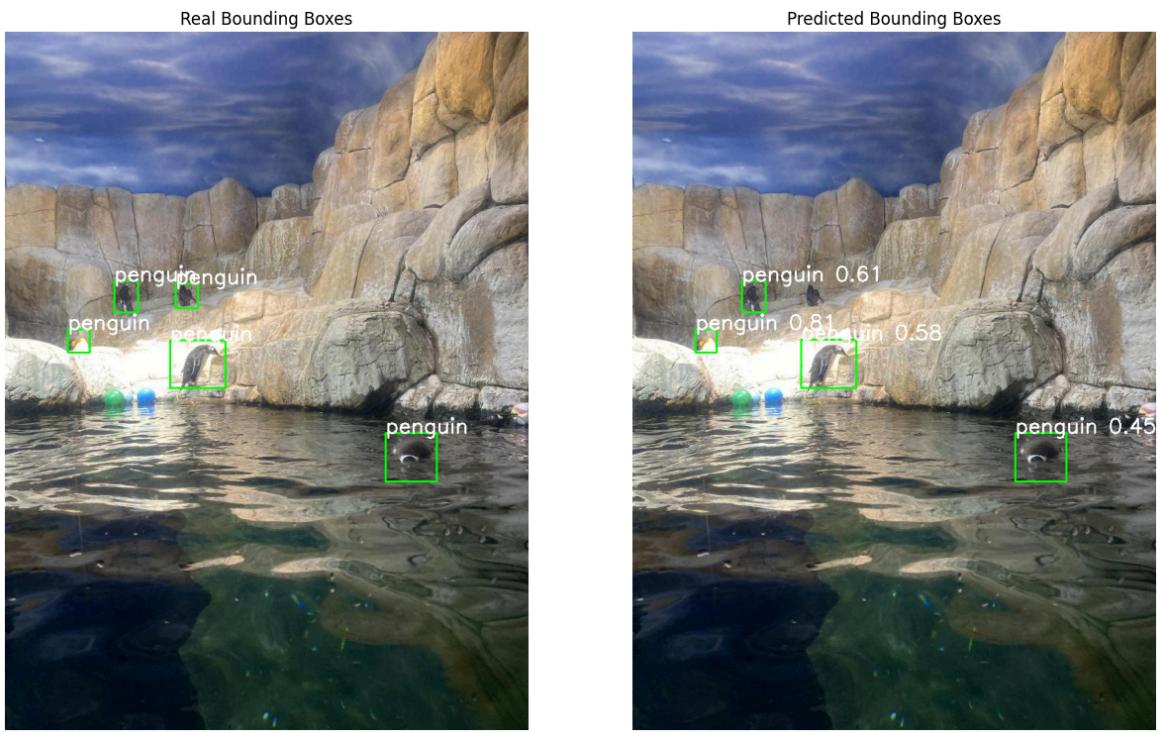
شکل 2-22. نمونه خروجی شماره 2



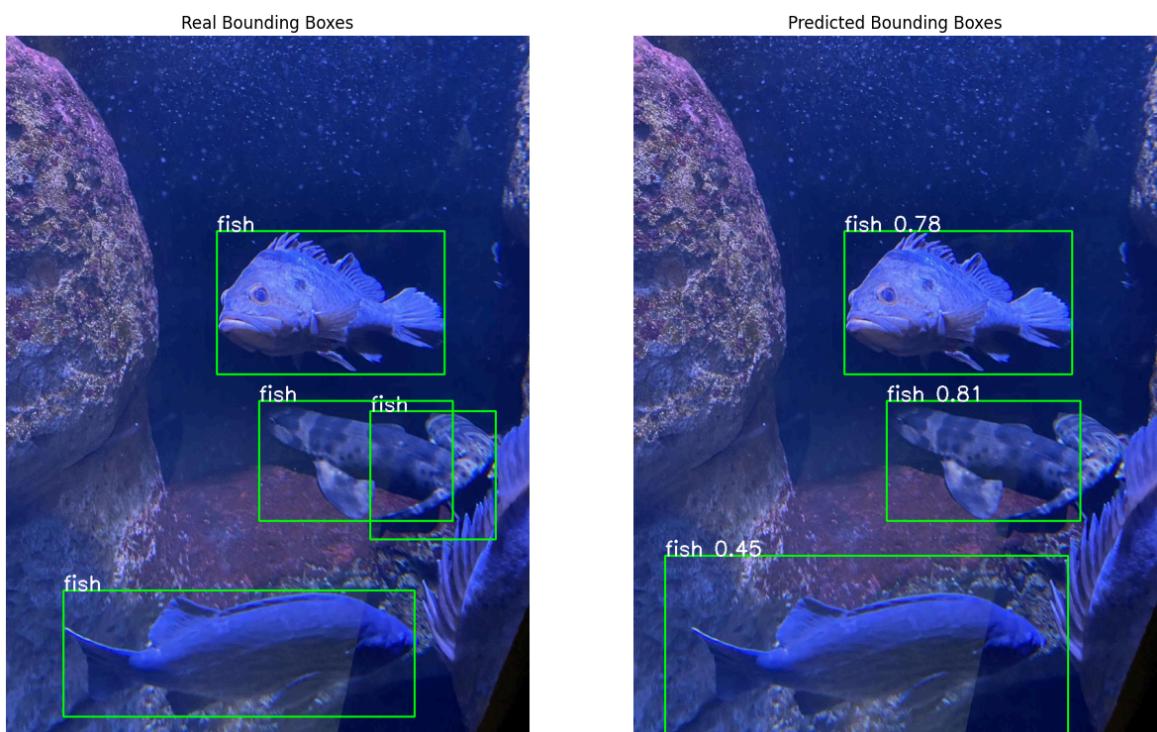
شکل 2-23. نمونه خروجی شماره 3



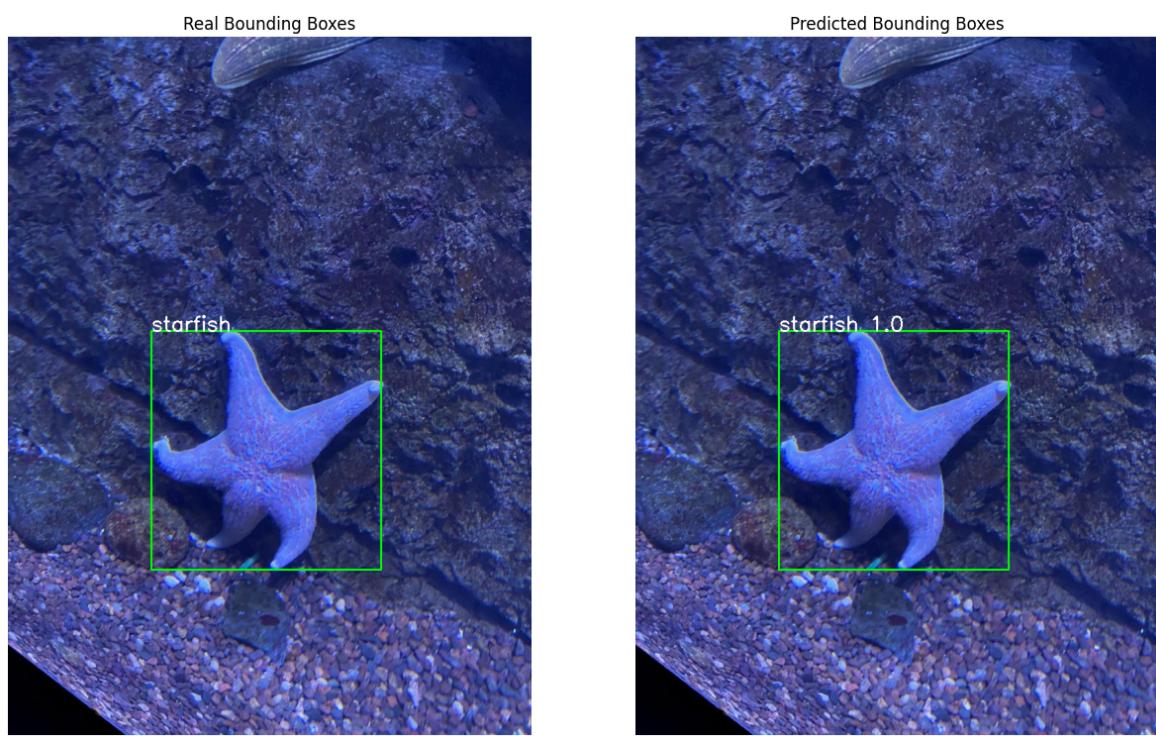
شکل 2-24. نمونه خروجی شماره 4



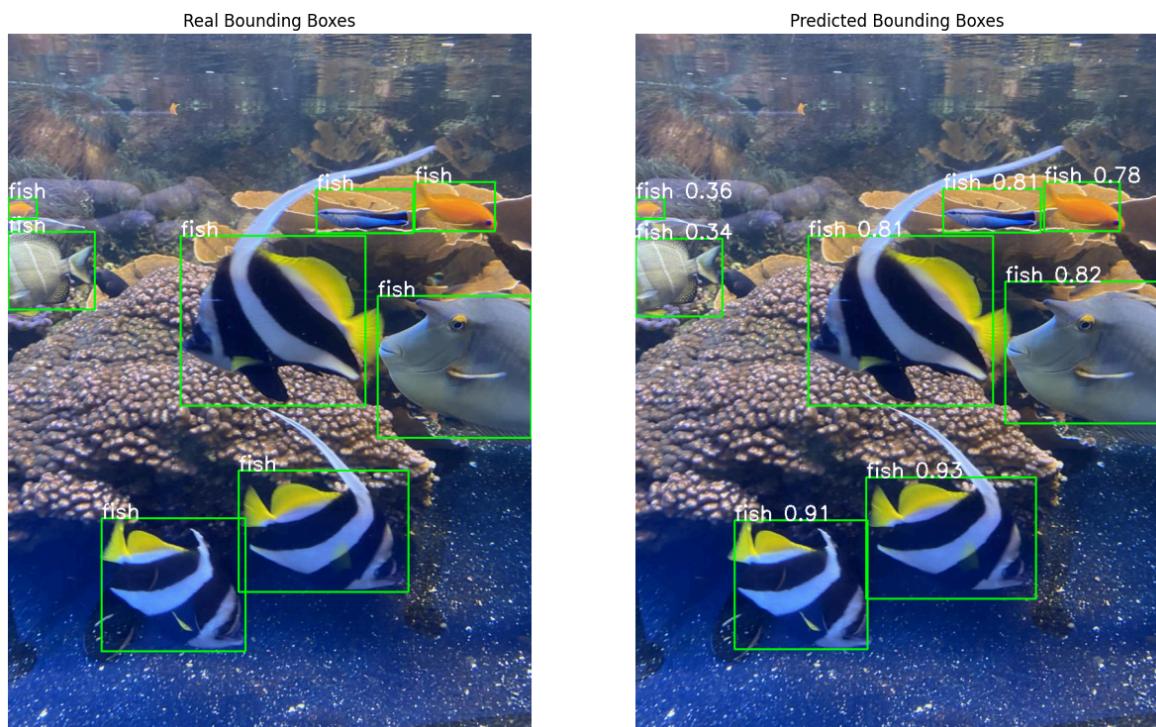
شکل 2-25. نمونه خروجی شماره 5



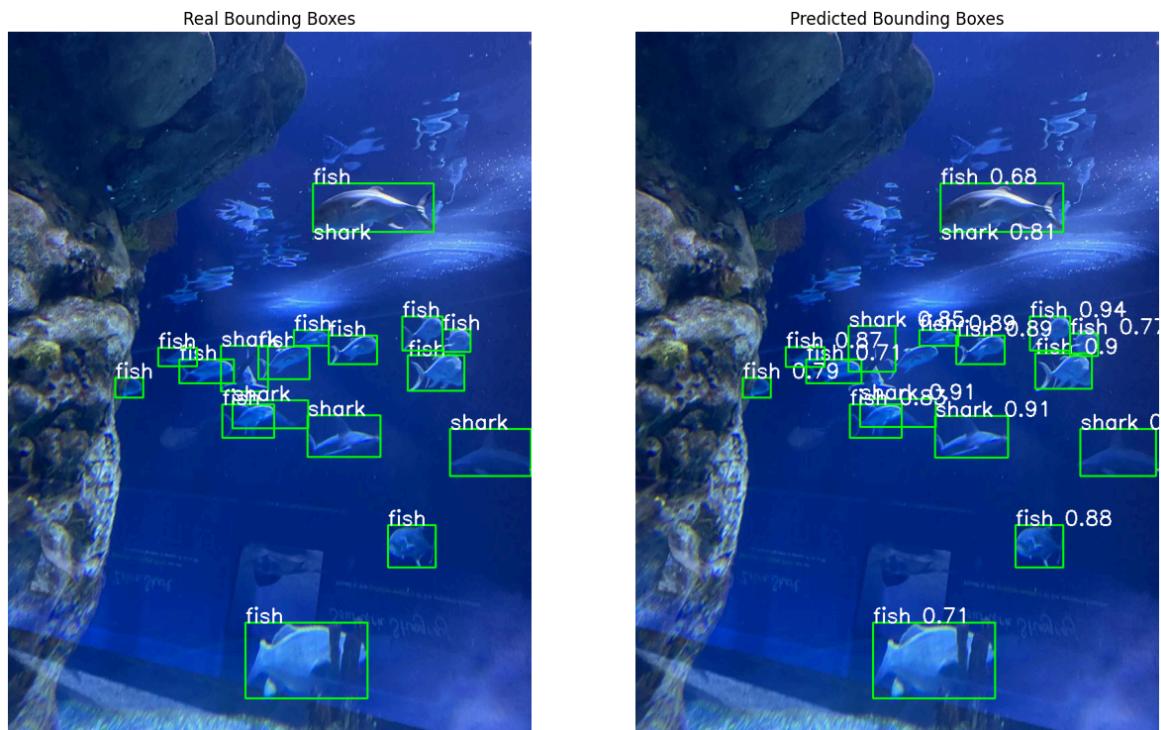
شکل 2-26. نمونه خروجی شماره 6



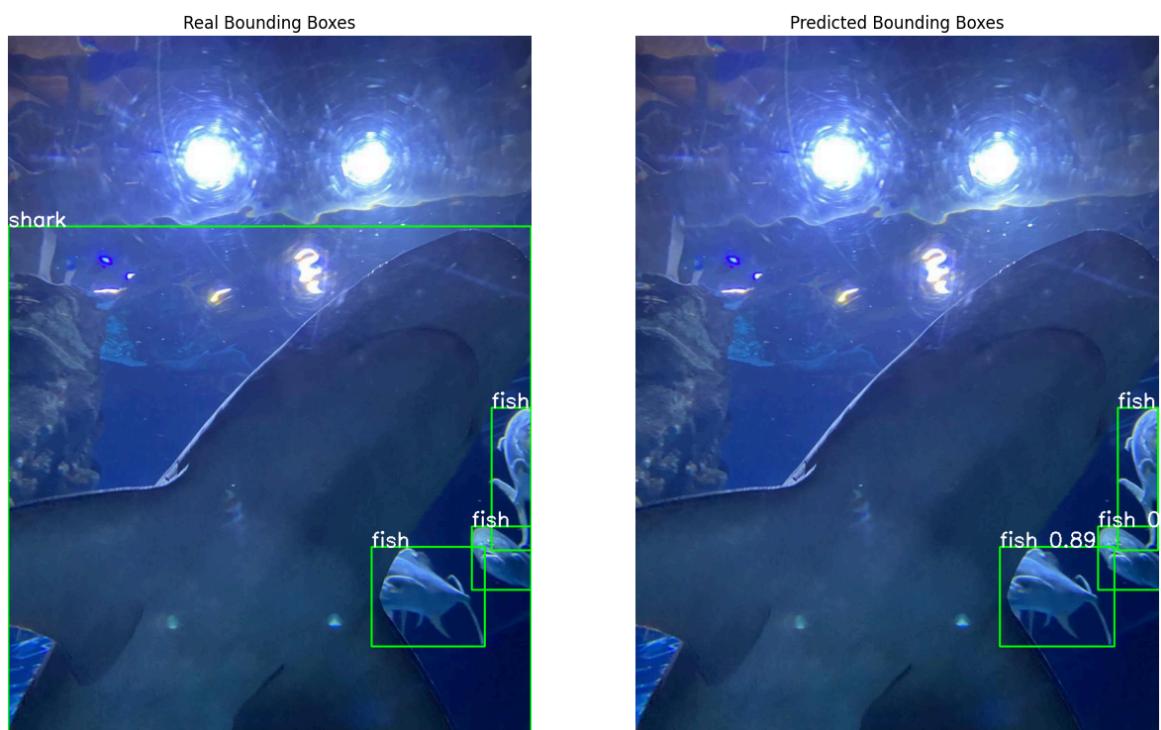
شکل 2-27. نمونه خروجی شماره 7



شکل 2-28. نمونه خروجی شماره 8



شکل-2-29. نمونه خروجی شماره 9



شکل-2-30. نمونه خروجی شماره 10

از تحلیل خروجی‌های مدل می‌توان نتیجه گرفت که مدل به دلیل imbalance-ای که در داده‌های train وجود داشت بخش زیادی از پیش‌بینی‌های مدل به سمت کلاس‌های این بخش می‌رود (مانند [شکل 1-2](#)). همچنین مدل برای تشخیص اجسام بسیار بزرگ (مانند [شکل 30-2](#) بخش کوسه) کمی مشکل دارد و همچنین برای اجسام بسیار ریز و با تراکم بالا (مانند [شکل 3-2](#) و [شکل 2-2](#)) کمی دچار اشکل است و نمی‌توانید به خوبی تشخیص دهد. راهکاری‌هایی برای حل این مشکلات از جمله استفاده از روش OHEM برای balance کردن یا تقویت بیشتر داده‌ها یا تغییر معماري مدل و همچنین آموزش بیشتر مدل (به دلیل سنگینی پردازش فقط ده ایپاک آموزش داده شده) نیز وجود دارد.