به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

درس شبکه‌های عصبی و یادگیری عمیق

تمرین سوم

پرسش ۱ و ۲	نام و نام خانوادگی	علی خرم فر
	رایانامه	khoramfar@ut.ac.ir
پرسش ۱ و ۲	نام و نام خانوادگی	علی رمضانی
	رایانامه	Ali.ramezani.۹۶@ut.ac.ir
	تاریخ ارسال پاسخ	۱۴۰۳/۰۳/۰۲

- هر دو نویسنده در هر دو پرسش همکاری داشته اند.

فهرست

فهرست تصاویر

پرسش۱ – پیاده سازی مدل U-net

بخش ۱-۱:آماده سازی مجموعه داده

ابتدا بعد از افزودن کتابخانه های لازم، سعی کردیم اطلاعات دیتا را از طریق فایل همراه آن بخوانیم:

Patient	RNASeqCluster	MethylationCluster	miRNACluster	CNCluster	RPPACluster	OncosignCluster	COCCluster	histological_type	neoplasm_histologic_grade	tumor_tissue_site	laterality	tumor_location
TCGA_CS_4941	2.0	4.0	2	2.0	NaN	3.0	2	1.0	2.0	1.0	3.0	2.0
TCGA_CS_4942	1.0	5.0	2	1.0	1.0	2.0	1	1.0	2.0	1.0	3.0	2.0
TCGA_CS_4943	1.0	5.0	2	1.0	2.0	2.0	1	1.0	2.0	1.0	1.0	2.0
TCGA_CS_4944	NaN	5.0	2	1.0	2.0	1.0	1	1.0	1.0	1.0	3.0	6.0
TCGA_CS_5393	4.0	5.0	2	1.0	2.0	3.0	1	1.0	2.0	1.0	1.0	6.0

شکل ۱- اطلاعات همراه دیتا

از آنجایی که در این مسئله، نیاز به آیدی شخص بیمار نداریم، بلکه تنها مدل باید روی هر عکس، محل تومور را تشخیص داده و پیکسل های آن را دسته بندی کند، سعی کردیم، در قالب یک دیتافریم، محل هر عکس و ماسک مرتبط به آن را بخوانیم.

	images	mask
0	/kaggle/input/lgg-mri-segmentation/kaggle_3m/T...	/kaggle/input/lgg-mri-segmentation/kaggle_3m/T...
1	/kaggle/input/lgg-mri-segmentation/kaggle_3m/T...	/kaggle/input/lgg-mri-segmentation/kaggle_3m/T...
2	/kaggle/input/lgg-mri-segmentation/kaggle_3m/T...	/kaggle/input/lgg-mri-segmentation/kaggle_3m/T...
3	/kaggle/input/lgg-mri-segmentation/kaggle_3m/T...	/kaggle/input/lgg-mri-segmentation/kaggle_3m/T...
4	/kaggle/input/lgg-mri-segmentation/kaggle_3m/T...	/kaggle/input/lgg-mri-segmentation/kaggle_3m/T...

شکل ۲ - محل ذخیره سازی تصاویر و ماسک آن

مهم است هر تصویر و ماسک آن دقیقا در یک ردیف قرار بگیرند، زیرا دیتای موجود در دایرکتوری ها مرتب نیست، در نتیجه این موضوع را چک کردیم.

```
##Test
idx = np.random.randint(len(df))
print(data['mask'][idx])
print(data['images'][idx])
```

```
/kaggle/input/lgg-mri-segmentation/kaggle_3m/TCGA_FG_7643_20021104/TCGA_FG_7643_20021104_18_mask.tif
/kaggle/input/lgg-mri-segmentation/kaggle_3m/TCGA_FG_7643_20021104/TCGA_FG_7643_20021104_18.tif
```

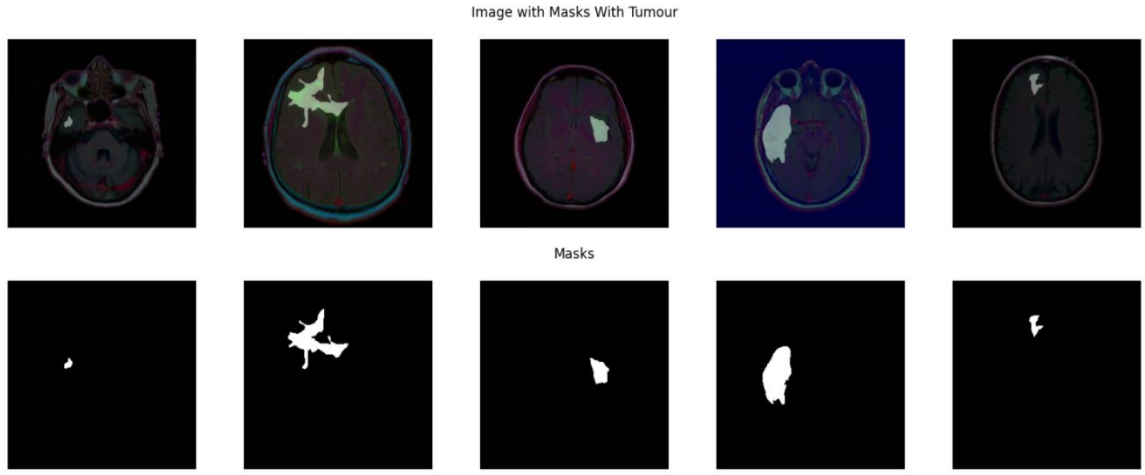
شکل ۳ - یکسان بودن ماسک و تصویر مربوط به آن

در نهایت با افزودن یک ستون دیگر به دیتا، وجود یا عدم وجود تومور در هر عکس نیز مشخص شد.

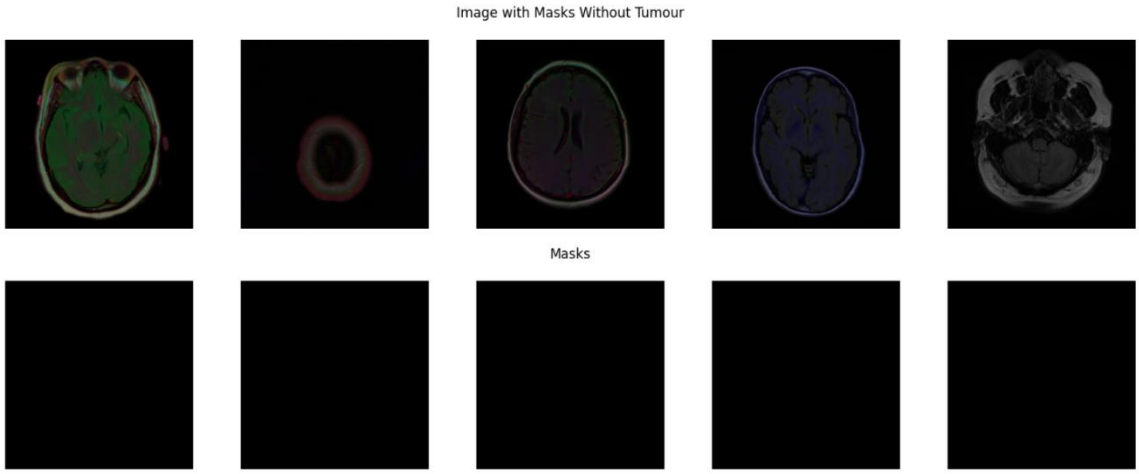
	images	mask	tumour
0	/kaggle/input/lgg-mri-segmentation/kaggle_3m/T...	/kaggle/input/lgg-mri-segmentation/kaggle_3m/T...	0
1	/kaggle/input/lgg-mri-segmentation/kaggle_3m/T...	/kaggle/input/lgg-mri-segmentation/kaggle_3m/T...	0
2	/kaggle/input/lgg-mri-segmentation/kaggle_3m/T...	/kaggle/input/lgg-mri-segmentation/kaggle_3m/T...	1
3	/kaggle/input/lgg-mri-segmentation/kaggle_3m/T...	/kaggle/input/lgg-mri-segmentation/kaggle_3m/T...	1
4	/kaggle/input/lgg-mri-segmentation/kaggle_3m/T...	/kaggle/input/lgg-mri-segmentation/kaggle_3m/T...	1
...	...	...	...

شکل ۴ - افزودن ستون وجود یا عدم وجود تومور

سپس به بررسی تصاویر پرداختیم:



شکل ۵ - تصاویر شامل تومور همراه با ماسک



شکل ۶ - تصاویر بدون تومور همراه با ماسک

در پایان این بخش دیتا را به نسبت خواسته شده تقسیم کردیم:

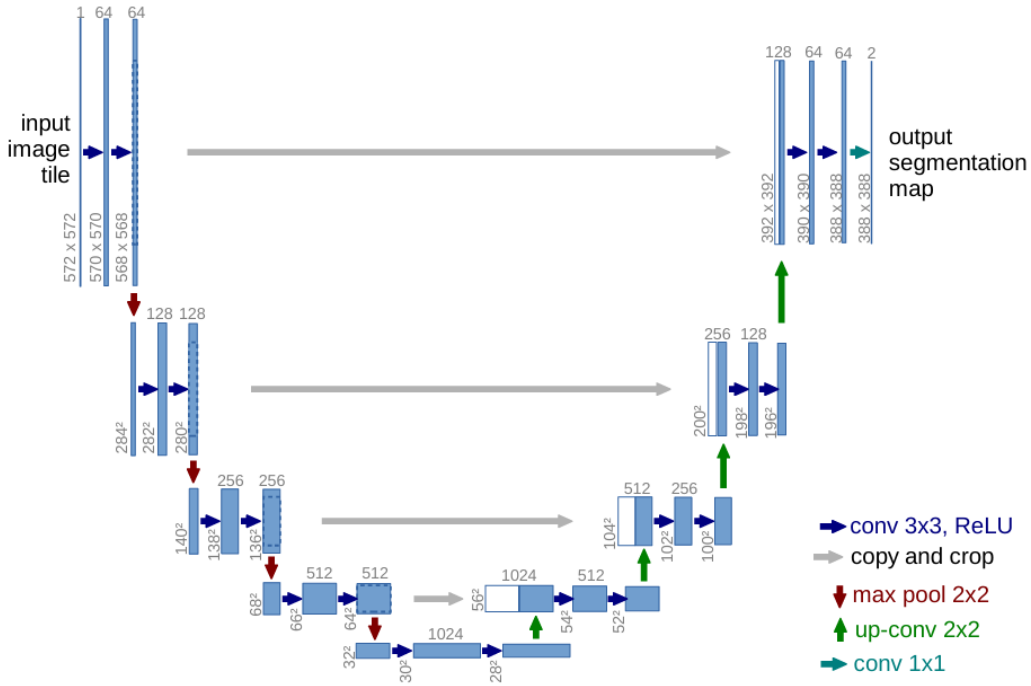
```
print(f'train data: {len(train_data)}')
print(f'valid_data: {len(valid_data)}')
print(f'test_data : {len(test_data)}')
```

```
train data: 3143
valid_data: 393
test_data : 393
```

شکل ۷ - تقسیم بندی دیتا به سه بخش

بخش ۲-۱: پیاده سازی مدل:

البته در نوت بوک ابتدا به تقویت داده پرداخته ایم، اما در این بخش طبق ساختار سوالات به بررسی مدل Unet میپدازیم:



شکل ۸ - مدل **Unet**

ساختار مدل U net دقیقا شبیه یک حرف U است، در این مدل ابتدا در طی چند مرحله feature map ها با استفاده اعمال مداوم شبکه های convolution و maxpooling بدست می آیند، در این بخش عملا مدل down sampling انجام میدهد، سپس در بخش دوم U، مدل با استفاده از transpose Convolution، شروع به up sampling میکند، با این تفاوت که همزمان اطلاعات موجود از بخش متناظر قبلی را نیز

concat میکند، اینکار هم به آموزش سریعتر مدل کمک میکند و هم اطلاعات از دسته رفته را برای فرایند up sample به مدل باز میگرداند.

ما این مدل را با دو تابع پیاده سازی کردیم:

```
def conv_down(inputs, num_filter):

    conv1 = Conv2D(num_filter, (3, 3), padding='same')(inputs)
    act1 = Activation('relu')(conv1)
    conv2 = Conv2D(num_filter, (3, 3), padding='same')(act1)
    bn = BatchNormalization(axis=3)(conv2)
    act2 = Activation('relu')(bn)
    drp = Dropout(0.1)(act2)
    pool = MaxPooling2D(pool_size=(2, 2))(drp)

    return conv2, pool, drp

def conv_up(input1, input2, num_filter1, num_filter2):

    up1 = Conv2DTranspose(num_filter1, (2, 2), strides=(2, 2), padding='same')(input1)
    up2 = concatenate([up1, input2], axis=3)
    conv1 = Conv2D(num_filter2, (3, 3), padding='same')(up2)
    act1 = Activation('relu')(conv1)
    conv2 = Conv2D(num_filter2, (3, 3), padding='same')(act1)
    bn1 = BatchNormalization(axis=3)(conv2)
    act2 = Activation('relu')(bn1)
    drp = Dropout(0.1)(act2)
    return drp
```

شکل ۹ - توابع کمکی در پیاده سازی مدل **u net**

تابع اول بخش ابتدایی یعنی down sample مدل را هندل میکند و تابع دوم بخش دوم یعنی up sample مدل را هندل میکند، در نهایت مدل یک تابع فعالساز sigmoid نیز خواهد داشت:

```
def unet(input_size=(128,128,3)):

    inputs = Input(input_size)

    conv1, pool1, _ = conv_down(inputs, 32)
    conv2, pool2, _ = conv_down(pool1, 64)
    conv3, pool3, _ = conv_down(pool2, 128)
    conv4, pool4, _ = conv_down(pool3, 256)
    _, _ , drp = conv_down(pool4, 512)

    drp1 = conv_up(drp, conv4, 512, 256)
    drp2 = conv_up(drp1, conv3, 256, 128)
    drp3 = conv_up(drp2, conv2, 128, 64)
    drp4 = conv_up(drp3, conv1, 64, 32)

    out = Conv2D(1, (1, 1), activation='sigmoid')(drp4)

    return Model(inputs=[inputs], outputs=[out])
```

شکل ۱۰ - مدل **Unet**

همچنین تعداد پارامتر های مدل نیز در ادامه دیده میشود:

```
==========================================================================================
Total params: 9,246,145
Trainable params: 9,243,201
Non-trainable params: 2,944
__________________________________________________________________________________________
```

شکل ۱۱ - پارامتر های مدل

بخش ۳- ۱: تقویت داده:

در این بخش به تقویت داده پرداختیم، از آنجایی که کتابخانه tensorflow + keras توابع مناسب را دارد، نیازی به استفاده از Albumentation ندیدیم، در این بخش تکنیک های زیر را روی تصاویر و ماسک های آن اعمال کردیم:

```
image_datagen = ImageDataGenerator(rotation_range=0.2, width_shift_range=0.05, height_shift_range=0.05, shear_range=0.05,
                                   zoom_range=0.05, horizontal_flip=True, fill_mode='nearest')

mask_datagen = ImageDataGenerator(rotation_range=0.2, width_shift_range=0.05, height_shift_range=0.05, shear_range=0.05,
                                  zoom_range=0.05, horizontal_flip=True, fill_mode='nearest')
```

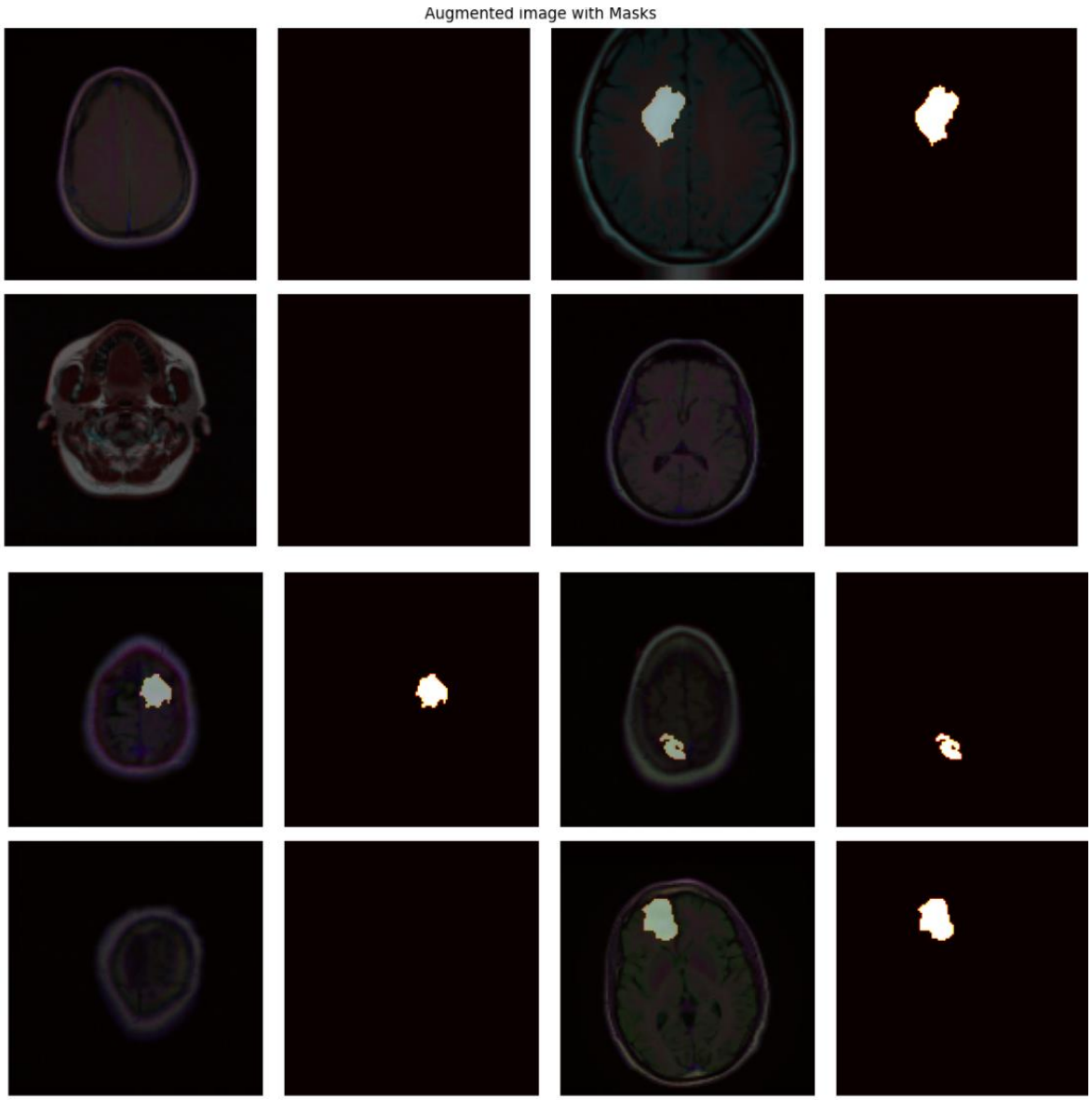
شکل ۱۲ - تکنیک های تقویت داده

میبنیم که از تغییرات ساده ای برای این موضوع استفاده کردیم، مقدار کمی داده ها را rotate کرده ایم، و همچنین در ارتفاع و در عرض کمی جابجا شده اند، و همچنین از Zoom و Horizental flip که مواردی مناسب برای تقویت داده ساده هستند استفاده کردیم، با توجه به حساسیت داده، که تصاویر MRI است، بهتر است تقویت داده در همین حد باشد تا توزیع داده ها به سمت داده های غیرواقعی نرود و تنها تغییرات اندکی روی آن ها ایجاد شود، همچنین تمام پیکسل های تصاویر و ماسک آن ها نیز نرمال شده اند:

```
for (img, mask) in train_gen:
    img = img/255.
    mask = mask/255.
    mask[mask >= 0.5] = 1
    mask[mask < 0.5] = 0

    yield (img, mask)
```

شکل ۱۳ - نرمال سازی تصاویر

در نهایت یک batch از داده ها را نمایش دادیم:



شکل ۱۴ - یک بچ از داده های آموزشی بعد از تقویت داده

بخش ۴-۱: بهینه ساز و توابع هزینه:

در این بخش به پیاده سازی متریک های IOU و Dice و همچنین تابع هزینه مبتنی بر Dice پرداختیم:

```
def dice_coef(y_true, y_pred, smooth=100):

    intersection = K.sum(y_true * y_pred)
    union = K.sum(y_true) +  K.sum(y_pred)
    return (2 * intersection)/(union + smooth)

def dice_loss(y_true, y_pred, smooth=100):
    return -dice_coef(y_true, y_pred, smooth)

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

شکل ۱۵ - بهینه ساز و تابع هزینه

متریک IOU یا به عبارتی Inetersection over Union در حقیقت حاصل تقسیم اشتراک بر اجتماع است، بدیهی است که هر چه مقدار اشتراک به اجتماع نزدیک باشد این مقدار نزدیک به ۱ و در حالت برعکس نزدیک به صفر خواهد شد، معیار Dice نیز بسیار شبیه است، با این تفاوت که حاصل تقسیم اشتراک بر مجموع هر یک از مجموعه هاست، تعریف دقیق این دو متریک در ادامه آمده است:

```
Dice = 2 |AnB| / (|A|+|B|) = 2 TP / (2 TP + FP + FN)
```

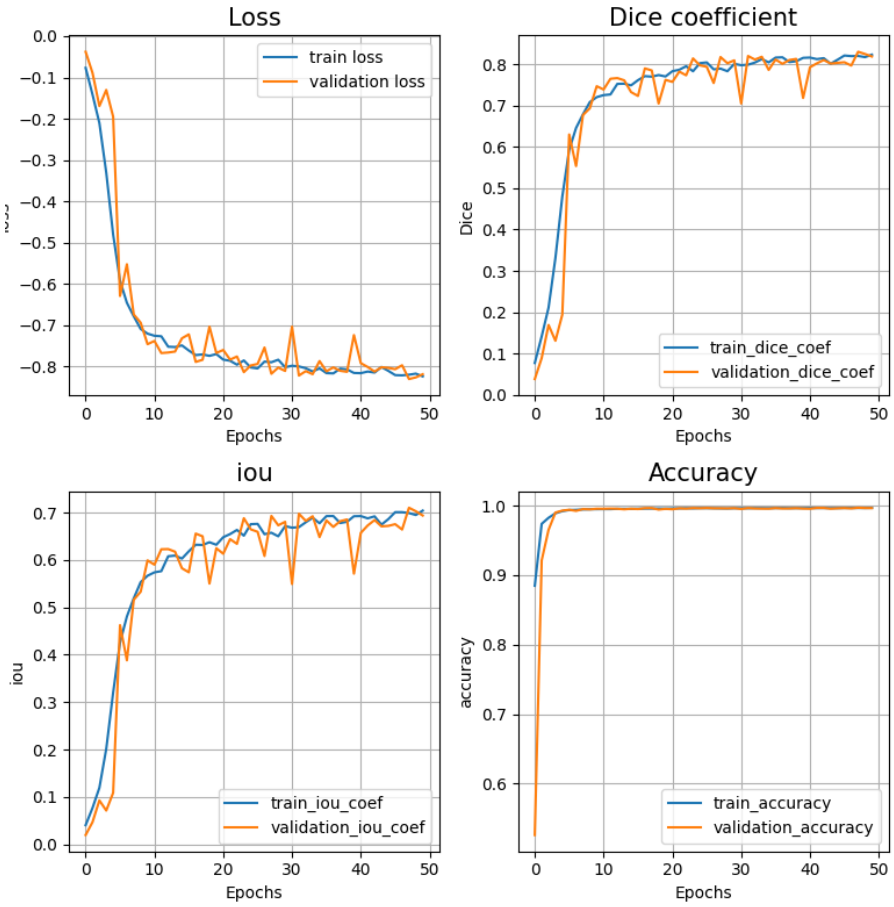
```
Jaccard = |AnB| / |AUB| = TP / (TP + FP + FN)
```

شکل ۱۶ - تعریف دقیق دو متریک **IOU** و **Dice**

من در این سوال از Dice loss همانطور که در بالا تعریف شده است(-Dice coeff) و همینطور optimizer Adamax استفاده کردم که سرعت همگرایی بهتری نسبت به Adam ایجاد کرد.

بخش ۵-۱: آموزش مدل:

در این بخش مدل را با اندازه epoch ۵۰ با نرخ یادگیری ۰.۰۰۱ آموزش دادم و نتایج مدل به شرح زیر می باشد:



شکل ۱۷ - نمودار آموزش مدل

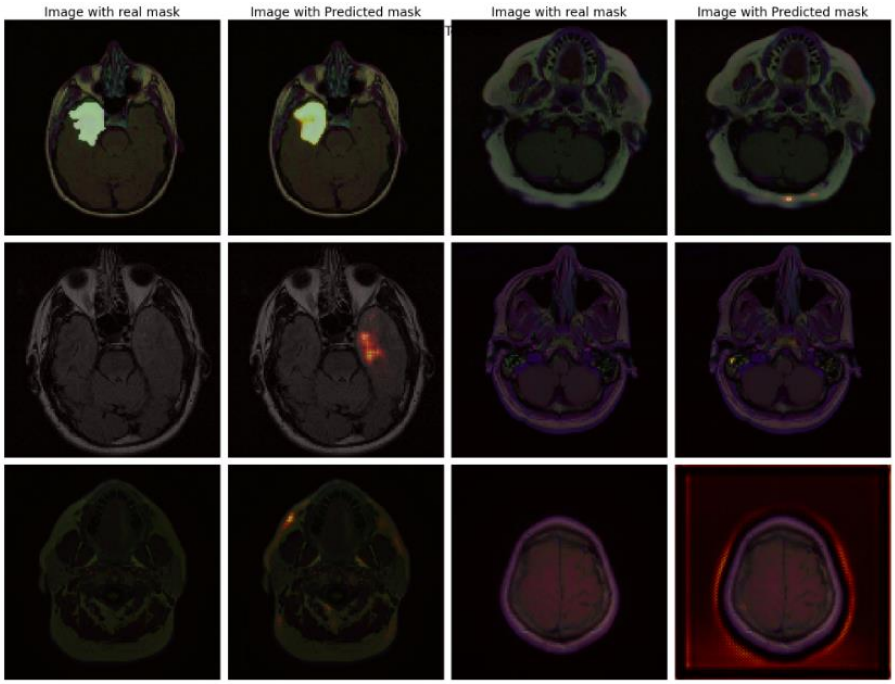
همچنین نتایج تست مدل نیز به صورت زیر است:

```
Found 393 validated image filenames.
Found 393 validated image filenames.
9/9 [==============================] - 4s 413ms/step - loss: -0.8277 - binary_accuracy: 0.9965 - iou_coef: 0.7057 - dice_coef: 0.8270
Test loss: -0.83
Test acc: 99.65
Test IOU Coefficent: 0.71
Test Dice Coefficent: 0.83
```

میبنیم که مدل به دقت بسیار خوبی روی متریک های iou , Dice رسیده است که حاصل از آموزش خوب مدل است.

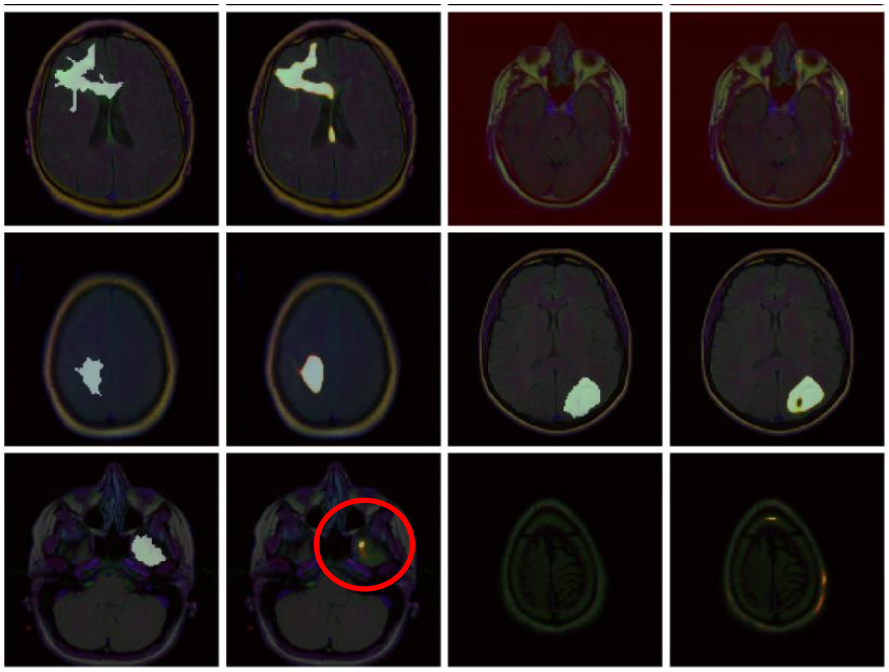
بخش ۶-۱: ارزیابی مدل:

در این بخش روی یک batch همه تصاویر آن بخش، به همراه ماسک اصلی و همچنین پیش بینی شده مدل را نمایش دادیم که نتایج قابل دیدن است:

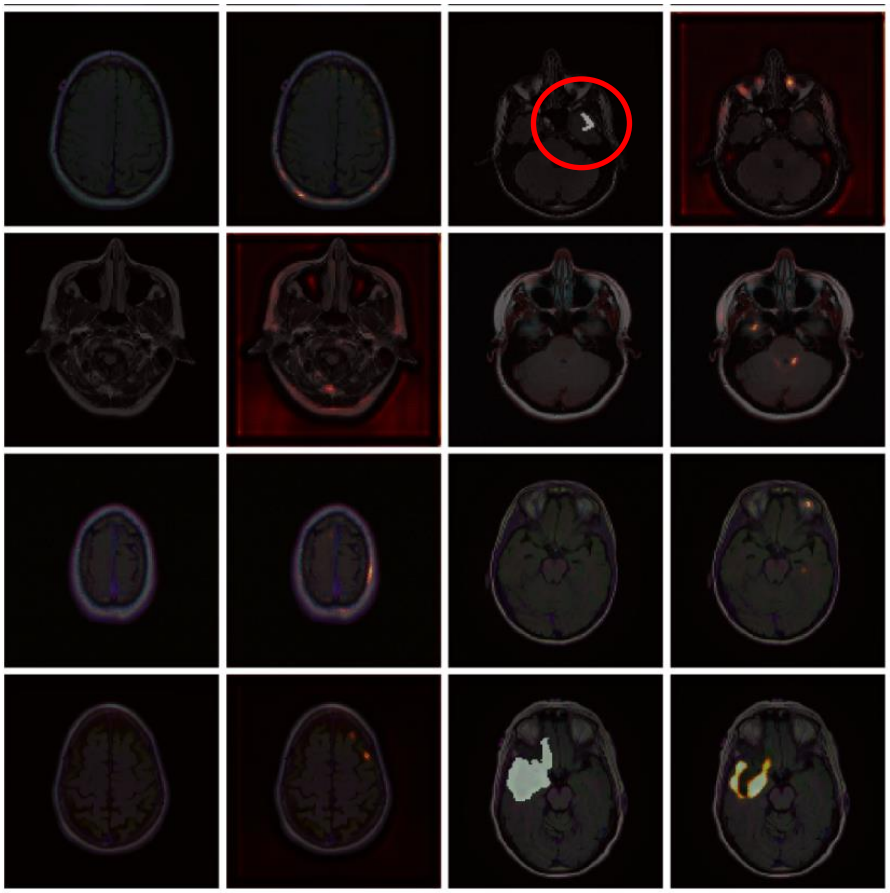


شکل ۱۸ - تشخیص تومور توسط مدل

در ۶ تصویر بالا میبینم که تنها تصویر دارای تومور به دقت شناسایی شده است و سایر موارد نیز دارای تومور تشخیص داده نشده اند



روی ۴ تصویر بالا میبینم که تنها یک مورد(پایین سمت جپ) فاقد تومور است، البته مدل گویا در خال تشخیص تومور بوده است و اگر آموزش ادامه می یافته احتمالا این مورد نیز به درستی تشخیص داده میشده است، این بخش را با دایره قرمز مشخص کرده ام.



و روی ۸ تصویر مقابل نیز میبینیم مدل در عمده موارد تشخیص کاملا درستی داشته است و تنها در مورد بالا سمت راست که تومور خیلی موچک است، نتوانسته تشخیص خوبی داشته باشد.

در مجموع مدل عمکلرد بسیار خوبی به نمایش گذاشته است.

جدول مقادیر هزینه و متریک ها روی هر سه مجموعه آموزش، ارزیابی و تست در ادامه آمده است:

	Train	Validation	Test
Loss	-۰.۸۲	-۰.۸۱۸۵	-۰.۸۳
Dice	۰.۸۲۲۹	۰.۸۱۸۴	۰.۸۳
IOU	۰.۷۰۴۴	۰.۶۹۳۹	۰.۷۱
Accuracy	۰.۹۹۶۸	۰.۹۹۶۵	۰.۹۹

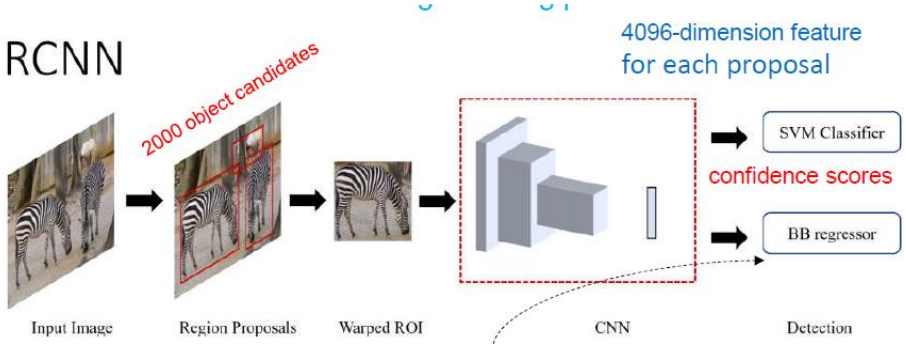
جدول ۱- مقادیر متریک و هزینه مدل روی هر سه مجموعه داده

پرسش ۲– تشخیص موجودات زیر آب

سوالات تشریحی

مقایسه مدل‌های Region-based CNNs

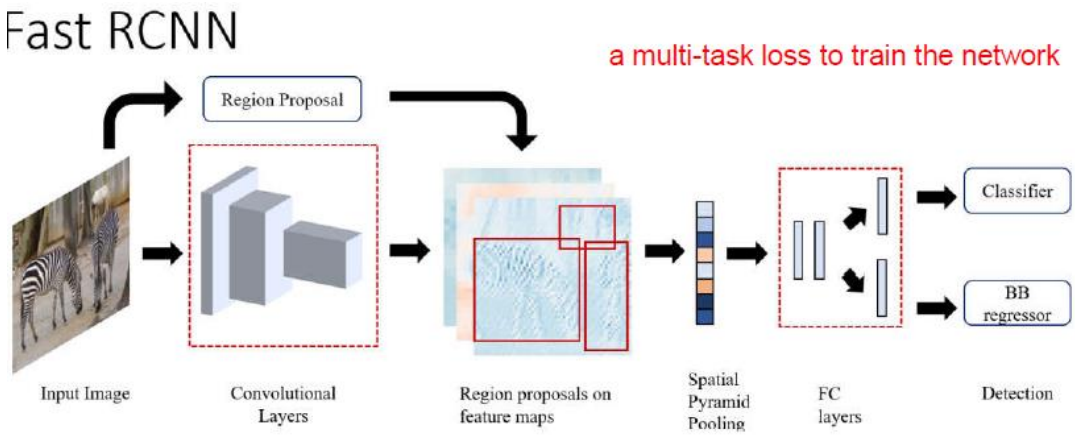
در مدل RCNN یا Regions with CNN features ابتدا با کمک الگوریتم‌هایی مانند Selective Search تعدادی Region Proposals از تصویر استخراج می‌شود. این نواحی به عنوان پیشنهاد برای تعیین نواحی نهایی استفاده می‌شوند. پس از استخراج Region Proposals ، همه آن‌ها Scale شده و به یک شبکه عصبی کانولوشنی یا CNN داده می‌شوند تا ویژگی‌های آن‌ها استخراج شود. در مرحله آخر نیز با یک دسته‌بندی SVM کلاس هر ناحیه انتخاب‌شده تعیین می‌شود. این مدل باتوجه به استفاده از CNN برای استخراج ویژگی‌ها نسبت به مدل‌های سنتی دقت تشخیص بالاتری داشته ولی از طرفی این فرایند طولانی و زمانبر است.



شکل ۱۹ معماری **RCNN**

در معماری FAST R-CNN که یک نسخه بهبود داده‌شده از R-CNN است، مانند همان روش قبلی ابتدا Region Proposals استخراج شده و سپس به یک شبکه CNN داده می‌شود تا Feature map تولید شوند. پس از اینکه Region Proposals روی Feature map قرار گرفتند، با استفاده از تکنیک ROI Pooling به اندازه‌های ثابت تبدیل می‌شوند. سپس این ویژگی‌های استخراج شده به یک لایه FC داده می‌شود که در نهایت به یک طبقه‌بند و یک Bounding Box Regressor داده می‌شوند.

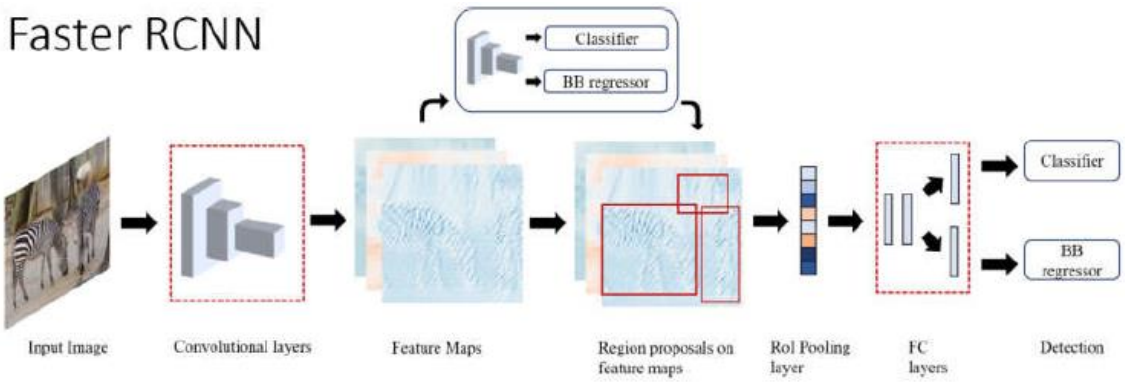
این معماری باتوجه به اینکه فقط یکبار کل تصویر به شبکه CNN داده می‌شود، نسبت به R-CNN سرعت بالاتری داشته و باتوجه به بهبود مرحله آموزش، دقت بالاتری ارائه می‌دهد. هرچند که استخراج Region Proposals هنوز بسیار زمان‌بر بوده و از معایب این مدل است. تصویر این معماری در شکل زیر مشاهده می‌شود:



شکل ۲۰ معماری **Fast RCNN**

معماری Faster R-CNN نسخه بهبودیافته Fast R-CNN است که از شبکه Region Proposal Network یا RPN استفاده می‌کند. در این معماری پس از اینکه کل تصویر به یک شبکه CNN داده شد، شبکه RPN روی Feature maps اعمال شده تا Region Proposals استخراج شود. این Region Proposals به همراه Feature maps در مرحله بعد به لایه Roi Pooling داده شده پس از تحویل به لایه FC در نهایت توسط یک طبقه‌بند Bounding Box Regressor نواحی تشخیص داده‌شده خروجی داده می‌شود.

این مدل باتوجه به اینکه از الگوریتم تشخیص Region Proposals جداگانه به طور مستقیم روی تصویر استفاده نکرده و به صورت یکپارچه از شبکه RPN استفاده می‌کند، کارایی بهتری دارد. هرچند که این مدل هم به منابع زیادی برای آموزش نیاز داشته و باتوجه به اینکه پیچیده‌تر است، دقت بالاتری برای تنظیم آن نیاز است.



شکل ۲۱ معماری **Faster RCNN**

مقایسه مدلهای one-stage و two-stage

معماری‌های تشخیص اشیاء به دو دسته اصلی تقسیم می‌شوند: معماری یک‌مرحله‌ای (One-Stage Detectors) و معماری دومرحله‌ای (Two-Stage Detectors). هر یک از این دو دسته مزایا و معایب خاص خود را دارند و در کاربردهای مختلف مورد استفاده قرار می‌گیرند.

معماری دو مرحله‌ای (Two-Stage Detectors) ابتدا Region Proposals را تولید کرده و سپس این نواحی را برای تشخیص دقیق‌تر و دسته‌بندی به کار می‌گیرد. برای مثال در مدل Faster R-CNN، مرحله اول شامل تولید Region Proposals توسط شبکه RPN و مرحله دوم شامل دسته‌بندی و تنظیم دقیق مکان این نواحی است. نمونه‌های معروف:R-CNN , Fast R-CNN , Faster R-CNN.

Two-Stage Detectors	
مزایا	دقت بالاتر در تشخیص اشیاء به دلیل فرآیند دو مرحله‌ای که شامل استخراج نواحی دقیق‌تر و استفاده از شبکه‌های عمیق‌تر
معایب	پیچیدگی بیشتر در پیاده‌سازی - سرعت پایین‌تر نسبت به مدل‌های یک‌مرحله‌ای به دلیل فرآیند دو مرحله‌ای و محاسبات بیشتر
کاربرد	کاربردهایی که نیاز به دقت بسیار بالا دارند مانند تجزیه و تحلیل تصاویر پزشکی، تشخیص اشیاء در تصاویر ماهواره‌ای و سیستم‌های امنیتی.

معماری یک مرحله‌ای (One-Stage Detectors) Region Proposalو تشخیص اشیاء را به صورت همزمان و در یک مرحله انجام می‌دهد. نمونه‌های معروف : YOLO که خود شامل نسخه‌های مختلف است.(SSD (Single Shot MultiBox Detector و RetinaNet .

One-Stage Detectors	
مزایا	سادگی بیشتر در پیاده‌سازی - سرعت بالاتر به دلیل انجام عملیات تشخیص در یک مرحله، مناسب برای کاربردهای بی‌درنگ یا (Real-Time)
معایب	دقت کمتر نسبت به مدل‌های دومرحله‌ای به ویژه در شناسایی اشیا کوچک - حساسیت بیشتر به تغییرات اندازه و شکل اشیاء در تصویر
کاربرد	کاربردهایی که نیاز به سرعت بالا و پردازش بی‌درنگ دارند مانند سیستم‌های رانندگی خودکار، نظارت ویدئویی و استفاده در گوشی‌های هوشمند.

OHEM

Online hard example mining یا OHEM برای رفع مشکل عدم توازن بین نمونه‌های مثبت و منفی در تشخیص اشیا استفاده می‌شود. در طی فرایند آموزش، OHEM به صورت دینامیک نمونه‌های سخت (نمونه‌هایی که تشخیص آنها برای مدل دشوار است) را انتخاب می‌کند و بر روی این نمونه‌ها تمرکز می‌کند. این روش کمک می‌کند تا با انتخاب نمونه‌های منفی دشوارتر، که اطلاعات بیشتری برای آموزش دارند، مشکل عدم تعادل برطرف شود.

OHEM	
مزایا	متوازن کردن نمونه‌های مثبت و منفی - با تمرکز بر نمونه‌های سخت، مدل بهبود یافته و قادر به مدیریت بهتر موارد دشوار می‌شود که این امر دقت کلی تشخیص را افزایش می‌دهد.
کاربرد	در تشخیص اشیایی که در آنها عدم توازن قابل توجهی بین نمونه‌های مثبت و منفی وجود دارد، استفاده می‌شود. این روش به خصوص در محیط‌های پیچیده مانند تشخیص اشیاء زیر آب که تشخیص اشیا از پس‌زمینه دشوار است، بسیار مفید است.

Soft-NMS

Soft-NMS یک بهبود نسبت به الگوریتم NMS است که در تشخیص اشیا استفاده می‌شود. در حالی که NMS باکس های همپوشان را بر اساس یک آستانه حذف می‌کند، Soft-NMS به جای حذف، امتیاز اطمینان باکس های همپوشان را کاهش می‌دهد و این امکان را فراهم می‌کند که با دقت بیشتری با تشخیص‌ها برخورد شود.

Soft-NMS	
مزایا	کاهش منفی‌های اشتباه یا False Negative: با کاهش امتیازات به جای حذف کامل باکس های همپوشان ، Soft-NMS احتمال از دست دادن مثبت‌های واقعی را کاهش می‌دهد. همچنین تغییرات جزئی در NMS استاندارد دارد و به راحتی می‌توان آن را در الگوریتم‌های مختلف تشخیص اشیا به کار برد.
کاربرد	کمک به تشخیص بهتر در محیط‌های پیچیده مثل شناسایی دقیق‌تر گونه‌های دریایی و یا تحلیل تصاویر ماهواره‌ای و فضایی برای شناسایی اشیاء مختلف روی زمین یا در

	فضا و یا در در سیستم‌های نظارتی و امنیتی، می‌تواند به شناسایی دقیق و کاهش نرخ هشدارهای کاذب کمک کند.

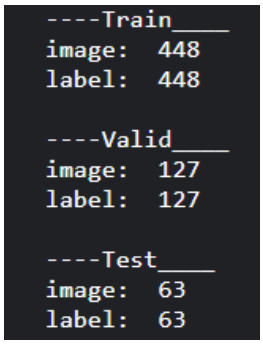
GIOU

Generalized Intersection over Union یا GIOU یک معیار پیشرفته تر از IOU است که در تشخیص اشیا استفاده می‌شود. GIOU نه تنها ناحیه همپوشانی بین باکس‌های پیش‌بینی‌شده و واقعی را در نظر می‌گیرد، بلکه نواحی غیر همپوشان را نیز در محاسبات خود لحاظ می‌کند.

GIOU	
مزایا	بهبود دقت: با در نظر گرفتن نواحی غیر همپوشان، GIOU می‌تواند در مواقعی که ناحیه‌های پیش‌بینی‌شده و واقعی هیچ همپوشانی ندارند نیز مفید باشد و عملکرد مدل را در فرآیند آموزش بهبود دهد.
کاربرد	GIOU در الگوریتم‌های تشخیص اشیا برای بهبود دقت و بهینه‌سازی ناحیه‌های پیش‌بینی شده استفاده می‌شود. این روش در محیط‌های پیچیده و با چالش‌های خاص مانند تشخیص اشیای زیر آب که نیاز به دقت بالاتری دارند، کاربردی است.

EDA و پیش‌پردازش دادگان

EDA کمک می‌کند تا داده‌ها بهتر فهمیده شده و تصمیمات بهتری در مورد مدل‌سازی و تحلیل‌های پیشرفته گرفته شود. این تحلیل‌ها برای فهم بهتر از داده‌ها و کشف الگوها و فرضیات اولیه در مورد روابط میان متغیرها استفاده می‌شود. هدف اصلی EDA فراهم کردن نگاهی دقیق‌تر از داده‌ها قبل از اعمال مدل‌های پیچیده یادگیری ماشین و شبکه عصبی است. در دیتاست ارائه شده تعداد داده‌های مربوط به هر مجموعه به صورت زیر است :



شکل ۲۲ تعداد داده‌های هر مجموعه

در این مسئله ۹ کلاس داریم که شامل موجودات مختلف دریایی است. در دیتاست ارائه شده کلاس‌ها به صورت عددی و به فرمت YOLO ارائه شده اند. اسامی این دسته‌ها به صورت زیر است:

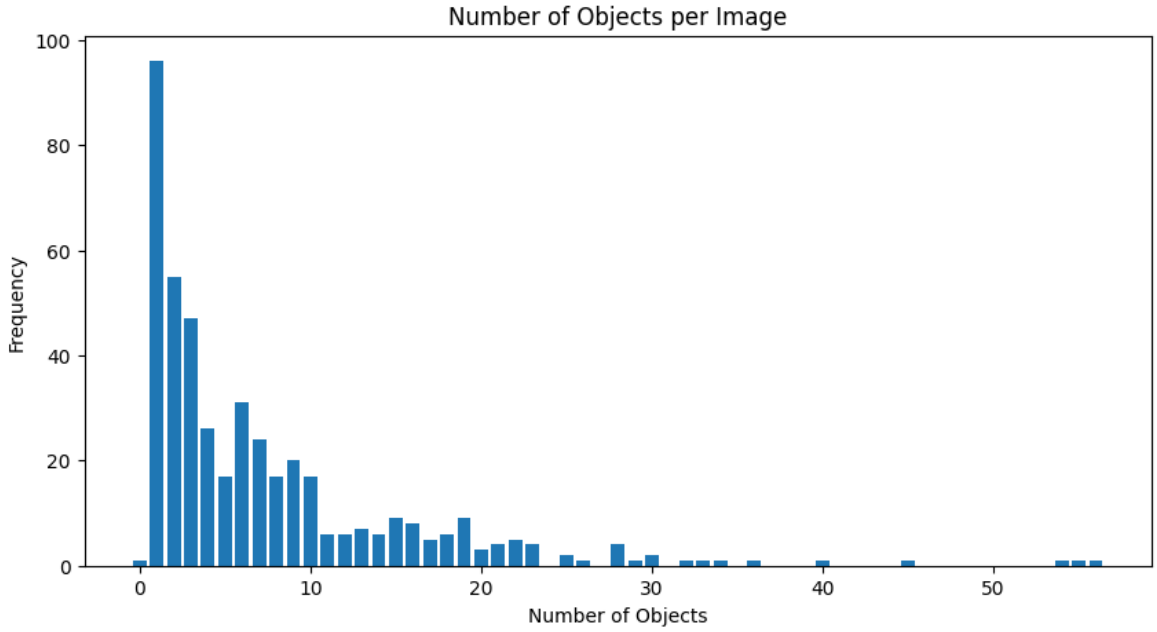
classes = ['fish', 'jellyfish', 'penguin', 'puffin', 'shark', 'starfish', 'stingray']

قبل از بررسی داده‌ها با کمک تابعی که برای نمایش تصاویر به همراه BBox آن‌ها پیاده‌سازی شده برخی تصاویر رندم از مجموعه train را نمایش می‌دهیم تا با ماهیت کلی مسئله و مقاله آشنا شویم:

شکل ۲۳ برخی تصاویر از مجموعه **Train** به همراه **Bbox**

تعداد اشیا در هر تصویر

تحلیل تعداد اشیاء در هر تصویر یک گام مهم در EDA است که به فهم بهتر توزیع و تعداد اشیاء موجود در تصاویر کمک می‌کند. این اطلاعات به ما کمک می‌کند تا پیچیدگی داده‌ها را بهتر درک کنیم و استراتژی‌های مناسبی برای مدل‌سازی و بهبود عملکرد الگوریتم‌های تشخیص اشیاء اتخاذ کنیم.

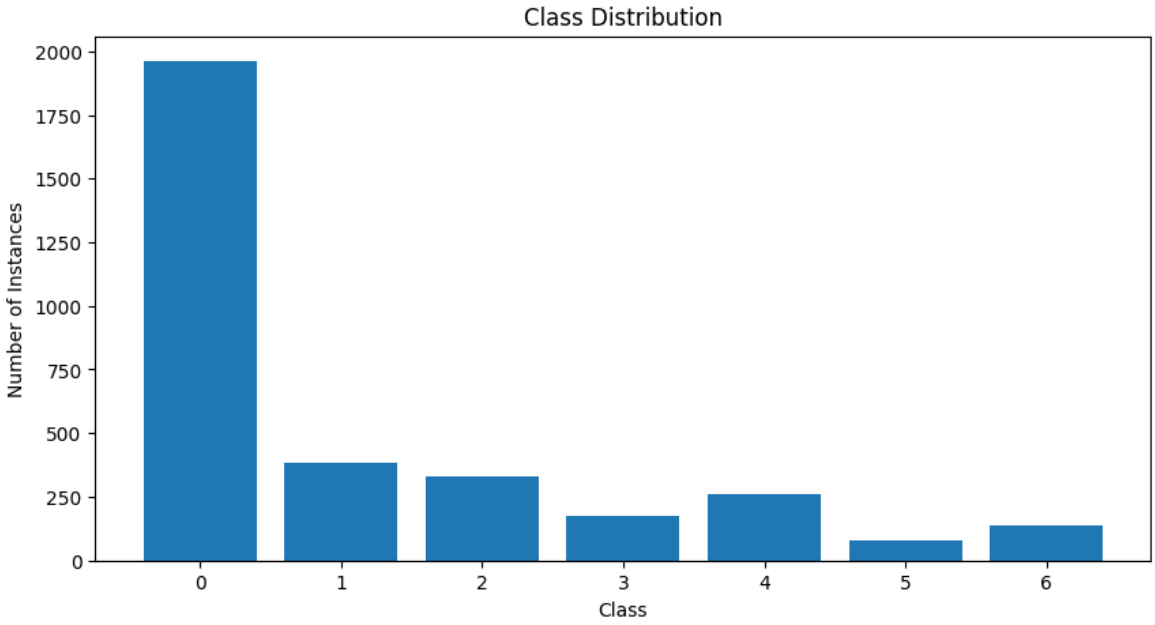


شکل ۲۴ تعداد اشیا در هر تصویر

نمودار هیستوگرام بالا تعداد اشیاء در هر تصویر را نشان می‌دهد. محور افقی تعداد اشیاء در هر تصویر و محور عمودی فراوانی تصاویر با آن تعداد از اشیاء را نمایش می‌دهد. همان‌طور که مشاهده می‌شود، بیشتر تصاویر دارای تعداد کمی از اشیاء (کمتر از ۱۰) هستند. این نشان می‌دهد که بسیاری از تصاویر نسبتاً ساده هستند و شامل تعداد محدودی از اشیاء می‌باشند. هرچند برخی از تصاویر دارای تعداد زیادی از اشیاء هستند که احتمالاً دارای چالش در مدل تشخیص اشیاء خواهیم بود.

تعداد تصاویر در هر کلاس

تحلیل تعداد تصاویر در هر کلاس یکی دیگر از مراحل مهم EDA است که به فهم بهتر از توزیع داده‌ها بین کلاس‌های مختلف کمک می‌کند. این تحلیل به ما نشان می‌دهد که آیا توزیع داده‌ها بین کلاس‌ها متوازن است یا خیر. این اطلاعات می‌تواند به شناسایی نیاز به تکنیک‌های مختلف برای متوازن‌سازی داده‌ها کمک کند و در نتیجه به بهبود عملکرد مدل منجر شود.

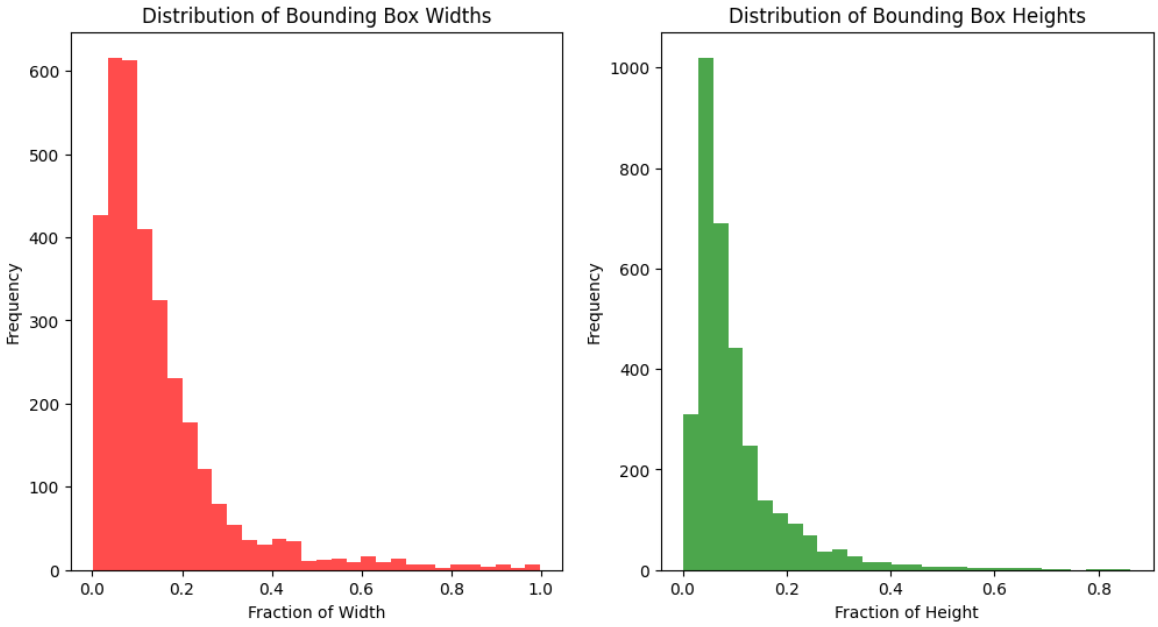


شکل ۲۵ تعداد تصاویر در هر کلاس

همان‌طور که مشاهده می‌شود، کلاس ۰ تعداد بسیار بیشتری تصاویر نسبت به سایر کلاس‌ها دارد. این نابرابری می‌تواند مشکلاتی را در فرآیند آموزش مدل ایجاد کند، زیرا مدل ممکن است به سمت کلاس‌هایی که تعداد بیشتری نمونه دارند، تمایل پیدا کند.

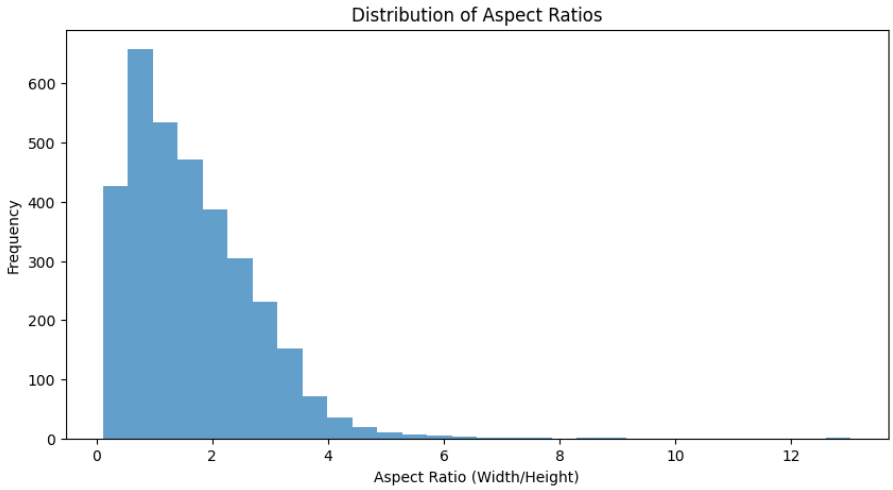
توزیع عرض و ارتفاع Bounding Box

این تحلیل به ما کمک می‌کند تا اندازه‌های اشیاء را در تصاویر بهتر درک کنیم.



شکل ۲۶ توزیع عرض و ارتفاع **Bounding Box**

تحلیل نسبت ابعاد (Aspect Ratios)

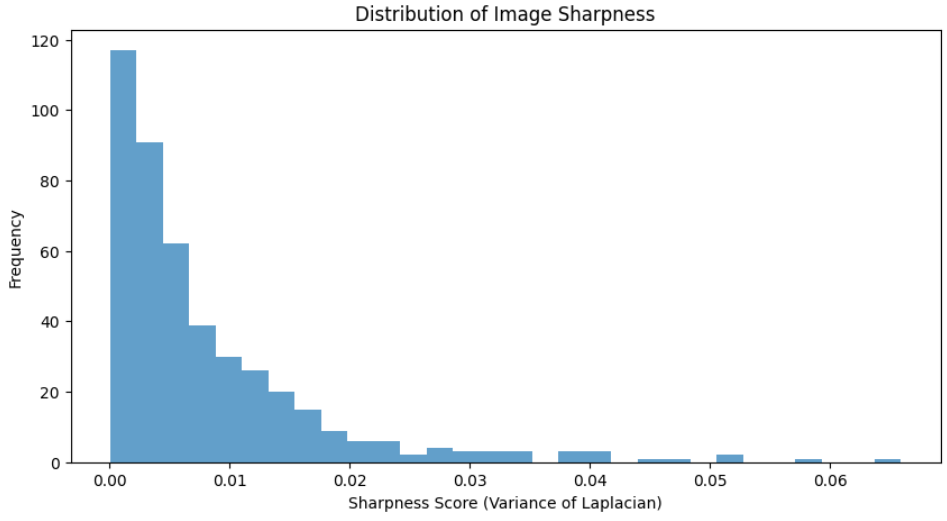


شکل ۲۷نسبت ابعاد تصاویر

همان‌طور که مشاهده می‌شود، اکثر باکس‌ها دارای **Aspect Ratios** کوچکتر از ۲ هستند. این نشان می‌دهد که بیشتر اشیاء در تصاویر نسبتاً مربعی یا کمی مستطیلی هستند. همچنین تعداد زیادی از از آن‌ها دارای نسبت کمتر از ۱ هستند که نشان می‌دهد این جعبه‌ها بیشتر بلند و باریک هستند. در مدل‌های تشخیص اشیاء مانند Faster R-CNN، استفاده ازanchors با نسبت‌های مختلف می‌تواند به بهبود دقت در تشخیص اشیاء با نسبت‌های مختلف کمک کند.

وضوح تصاویر (Sharpness)

تصاویر با وضوح پایین ممکن است باعث کاهش دقت مدل شوند، بنابراین بررسی وضوح تصاویر می‌تواند به ما کمک کند تا کیفیت داده‌های آموزشی را ارزیابی کنیم و اقدامات مناسبی برای بهبود آن انجام دهیم.

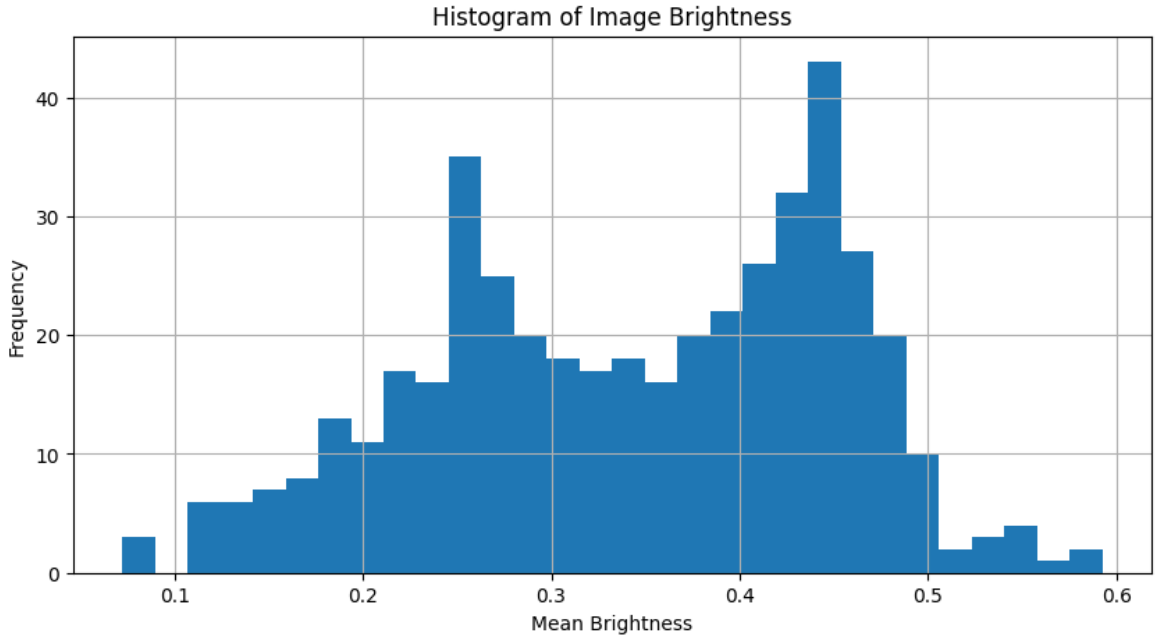


شکل ۲۸ توزیع وضوح تصاویر

بیشتر تصاویر دارای وضوح کمی هستند که می‌تواند چالش‌هایی را برای مدل ایجاد کند. استفاده از تکنیک‌های پیش‌پردازش تصویر برای افزایشsharpening یا کاهش نویز برای بهبود کیفیت تصاویر به بهبود کارایی مدل کمک می‌کنند.

روشنایی تصاویر (Brightness)

تصاویر با روشنایی بسیار کم یا بسیار زیاد ممکن است باعث کاهش دقت مدل شوند، بنابراین بررسی روشنایی تصاویر می‌تواند به ما کمک کند تا کیفیت داده‌های آموزشی را بررسی کنیم تا پیش‌پردازش‌های لازم را بر روی آن‌ها انجام دهیم.



شکل ۲۹ توزیع میانگین روشنایی تصاویر

تعداد زیادی از تصاویر دارای روشنایی متوسط بین ۰.۳ تا ۰.۴ هستند. این نشان می‌دهد که بیشتر تصاویر به طور کلی دارای روشنایی مناسبی هستند. باتوجه به اینکه که تصاویر در زیر آب هستند، تعداد کمی از تصاویر دارای روشنایی بسیار کم یا بسیار زیاد هستند که ممکن است بر عملکرد مدل تاثیر بگذارد.

پیش‌پردازش تصاویر و تقویت داده

در این قسمت از مسئله قصد داریم که داده‌ها را برای آموزش تقویت کنیم. برای این کار از تکنیک‌های مختلفی استفاده می‌شود. قصد ما در این تمرین پیاده‌سازی روش‌هایی است که در مقاله مرجع استفاده شده‌است.

Dataset Preprocessing

```
#New transforms
transforms = A.Compose([
    A.Resize(500, 400),
    A.HorizontalFlip(p=0.5),
    A.RandomCrop(height=500, width=400, p=0.2),
    A.RandomBrightnessContrast(brightness_limit=0.2, contrast_limit=0.2, p=0.5),
    A.GaussNoise(var_limit=(10.0, 50.0), p=0.5),
    A.RGBShift(r_shift_limit=20, g_shift_limit=0, b_shift_limit=-15, p=0.5),
    A.Normalize(mean=(0.5, 0.5, 0.5), std=(0.5, 0.5, 0.5)),

], bbox_params=A.BboxParams(format='yolo'))
```

شکل ۳۰پیش‌پردازش‌های انجام شده روی تصاویر

Resize:

این کار به یکسان کردن اندازه تصاویر کمک می‌کند و باعث می‌شود که تصاویر ورودی به مدل هم‌اندازه باشند. این یکنواختی می‌تواند به بهبود عملکرد مدل کمک کند.

HorizontalFlip:

چرخاندن تصاویر به صورت افقی می‌تواند به افزایش تنوع داده‌ها کمک کند و از overfitting جلوگیری کند. با چرخاندن تصاویر، مدل با نماهای مختلف از اشیاء آشنا می‌شود.

RandomCrop:

این تکنیک می‌تواند به بهبود توانایی مدل در تشخیص اشیاء کمک کند، زیرا مدل باید قادر به تشخیص اشیاء در قسمت‌های مختلف تصویر باشد.

RandomBrightnessContrast:

با این پیش‌پردازش، مدل با تصاویر با روشنایی و کنتراست مختلف آشنا شود و مقاومت بیشتری در برابر تغییرات نور خواهد داشت.

GaussNoise:

اضافه کردن نویز به تصاویر می‌تواند به مدل کمک کند تا مقاومت بیشتری در برابر نویزهای مختلف داشته باشد و عملکرد بهتری در شرایط واقعی داشته باشد.

RGBShift:

با تغییرات رنگی مختلف در تصاویر مدل مقاومت بیشتری در برابر تغییرات رنگی خواهد داشت. جانوران زیر آب ممکن است باتوجه به شرایط زیستی مختلف رنگ مختلف داشته باشند.

Normalize:

نرمال‌سازی تصاویر با میانگین و انحراف معیار ۰.۵ به بهبود دقت و کارایی مدل کمک می کند.

باتوجه به تغییرات انجام شده، لازم است تا (bbox_params=A.BboxParams(format='yolo') را در Transformها لحاظ کنیم تا فرمت لیبل تصاویر پس از پیش‌پردازش و تقویت داده‌ها حفظ شود. زیرا که برخی تکنیک‌های تقویت‌داده نسبت و جایگاه هر ناحیه از تصویر را تغییر می‌دهد.

Mosaic Augmentation

Mosaic Augmentation یک تکنیک پیشرفته برای افزایش داده‌ها در شبکه‌های عصبی است که به در تشخیص اشیاء بسیار مفید است. این تکنیک اولین بار توسط مدل YOLO نسخه ۴ معرفی شد و به طور گسترده در مدل‌های مختلف تشخیص اشیاء استفاده می‌شود. در این روش، چهار تصویر مختلف به طور تصادفی انتخاب می‌شوند و به یک تصویر جدید با ابعاد بزرگ‌تر ترکیب می‌شوند. هر یک از چهار تصویر به اندازه‌ای تغییر داده می‌شود که یک چهارم تصویر نهایی را تشکیل دهد.

مزایای Mosaic Augmentation

این روش می‌تواند تنوع زیادی به داده‌های آموزشی اضافه کند و باعث شود که مدل با نمونه‌های بیشتری از اشیاء در شرایط مختلف آشنا شود. با افزایش تنوع داده‌ها، مدل کمتر به داده‌های آموزشی وابسته می‌شود و عملکرد بهتری بر روی داده‌های جدید خواهد داشت و احتمال Overfitting کاهش می‌یابد. همچنین بهبود عمومیت مدل یا Generalization یکی دیگر از مزایای این روش است. به طور کلی با ترکیب تصاویر، تعداد نمونه‌های آموزشی به طور موثری افزایش می‌یابد بدون اینکه نیاز به جمع‌آوری داده‌های جدید باشد.

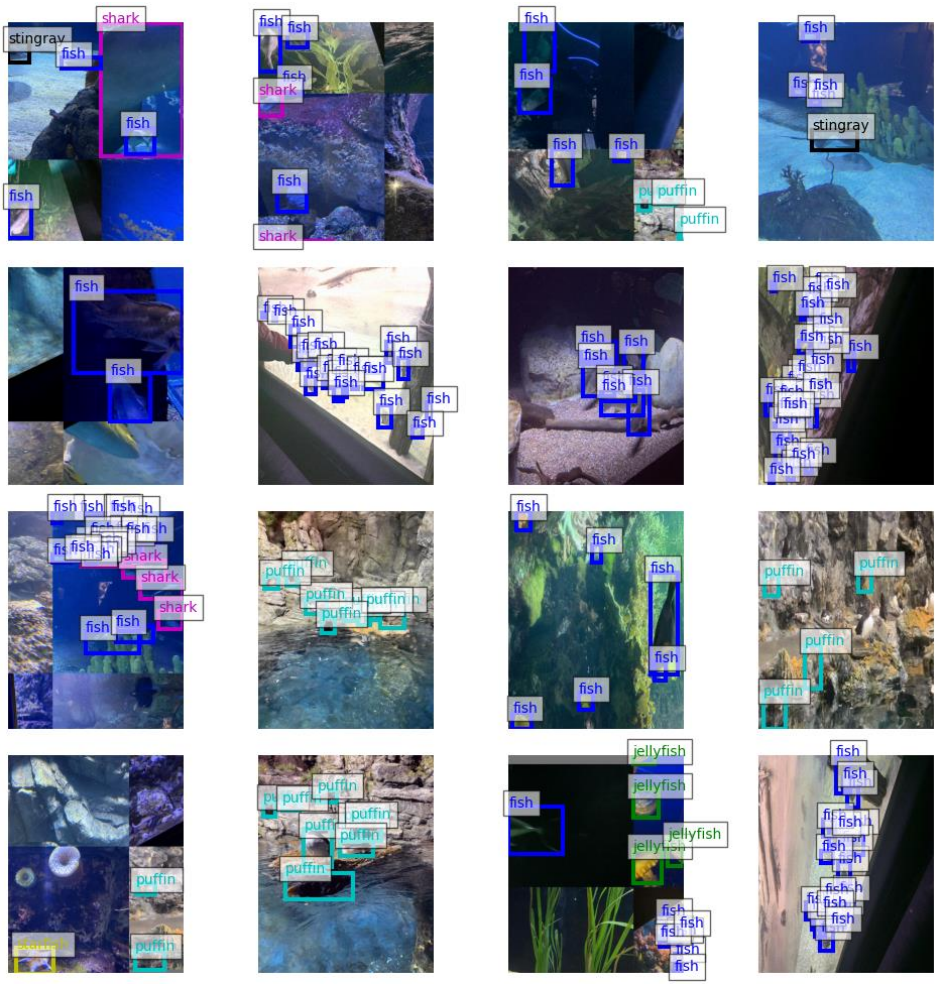
معایب Mosaic Augmentation

این تکنیک نسبت به تکنیک‌های ساده‌تر افزایش داده‌ها پیچیده‌تر است و نیاز به پیاده‌سازی دقیق دارد. این مورد در پیاده‌سازی این برای این مدل بسیار چالش برانگیز بود و در صورتی که Bounding Box ها به درستی تنظیم نشوند، ممکن است مشکلاتی در تشخیص اشیا به وجود بیاید.

کاربرد های Mosaic Augmentation

برای مجموعه داده‌های کوچک که نیاز به افزایش تعداد نمونه ها دارند، این تکنیک می‌تواند بسیار مفید باشد. همچنین اگر که تعداد داده‌ها مناسب بود، بسیاری از مدل‌های شبکه عصبی نیاز به تنوع داده‌ها دارند که این تکنیک می‌تواند عملکرد مدل را بهبود بخشد. در پیاده‌سازی این تابع احتمال ۰.۲ را به صورت پیشفرض در نظر گرفتیم که در ۲۰ درصد مواقع این تکنیک اعمال می‌شود.

پس از اعمال تکنیک **Mosaic** و دیگر روش‌های مذکور برای تقویت داده نتایج زیر حاصل شد که برخی تصاویر را مشاهده می‌کنیم:



شکل ۳۱ برخی تصاویر از مجموعه **Train** پس از پیش‌پردازش و تقویت داده‌ها

ساخت دیتالودر:

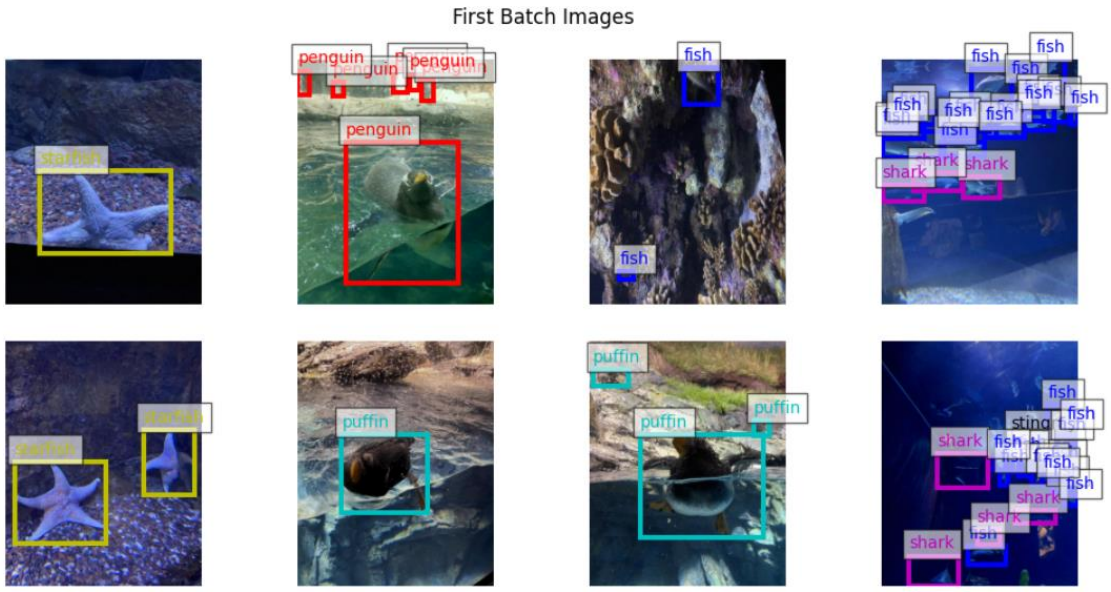
در این بخش ابتدا به ساخت دیتاست و سپس دیتالودر پرداختیم، لازم به ذکر است ابتدا ما دیتای مورد نیاز را در یک دیتافریم با فرمت زیر ذخیره کردیم:

	images	YOLO_format	Pascal_format
0	/kaggle/input/aquarium-data-cots/aquarium_pret...	[(0.2734375, 0.509765625, 0.33984375, 0.146484...	[(79, 447, 340, 597, 1), (577, 591, 767, 720, ...
1	/kaggle/input/aquarium-data-cots/aquarium_pret...	[(0.150390625, 0.75927734375, 0.30078125, 0.14...	[(0, 701, 231, 854, 1), (58, 457, 504, 607, 1)...
2	/kaggle/input/aquarium-data-cots/aquarium_pret...	[(0.20052083333333334, 0.888671875, 0.23046875...	[(65, 803, 242, 1016, 1), (278, 790, 457, 985,...
3	/kaggle/input/aquarium-data-cots/aquarium_pret...	[(0.7903645833333334, 0.0498046875, 0.19661458...	[(531, 0, 682, 102, 1), (303, 636, 474, 729, 1...
4	/kaggle/input/aquarium-data-cots/aquarium_pret...	[(0.5989583333333334, 0.482421875, 0.377604166...	[(315, 421, 605, 566, 4)]

شکل ۳۲- دیتای ذخیره شده در دیتافریم

در این بخش ما علاوه بر فرمت **YOLO**، فرمت **Pascal** را نیز ذخیره کردیم، همچنین **label** را نیز به عنوان المان آخر ذخیره شده تا بتوانیم از **Albumentation** در تغییر **annotation** ها نیز استفاده کنیم.

سپس به ساخت یک دیتاست از طریق همین دیتافریم اقدام کردیم، که transform و Mosaic آگمنتیشن را نیز توسط آن هندل کردیم، همچنین با استفاده از collate function در کتابخانه پایتورچ همه مقادیر annotation و class label ها را به اندازه بیشتری تعداد آبجکت موجود در یک batch با استفاده از مقدار ۱- پد کردیم، در نهایت نیز دیتالودر مورد نیاز در این بخش ساخته شد، بعد از ساخته شدن دیتالود، تعدادی از تصاویر موجود در آن را نیز نمایش دادیم



شکل ۳۳ - تصاویر حاصل از دیتالود در یک بچ

در نهایت خروجی دیتالودر، تصاویر، bbox های واقعی، و برچسب کلاس هر یک از bbox ها می باشد.

طراحی معماری Faster-RCNN:

برای طراحی مدل Faster-RCNN ابتدا از مدل ResNet۱۰۱ به عنوان استراج کننده ویژگی استفاده کردیم که کد آن را در ادامه میبیند، همچنین تعدادی از feature map ها را نیز در توسط این مدل تولید کردیم:

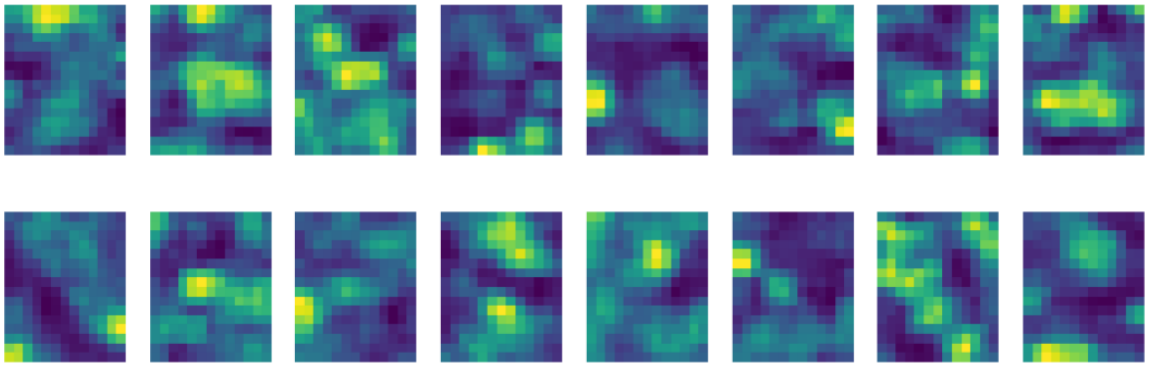
```
resnet = models.resnet101()
model_path = '/kaggle/input/resnet101/resnet101.pth'
resnet.load_state_dict(torch.load(model_path))
# resnet.eval()
```

```
<All keys matched successfully>
```

+ Code + Markdown

```
class FeatureExtractor(nn.Module):
    def __init__(self, original_model):
        super(FeatureExtractor, self).__init__()
        self.features = nn.Sequential(*list(original_model.children())[:-2])
        self.out_channels = 2048  # ResNet101 outputs 2048 channels from the last convolutional layer

    def forward(self, x):
        x = self.features(x)
        return x
```

شکل ۳۴ - استراج ویژگی و مدل **resnet۱۰۱**



شکل ۳۵- فیچرمپ های یک بچ آموزشی

همچنین علاوه بر شبکه RPN و بخش های مرتبط به آن که در بخش بعد توضیح میدهیم، از یک هد کلاسیفیکشن برای تشخیص کلاس هر یک از اشیا نیز استفاده کردیم که در ادامه میبنیم:

```
class ClassificationModule(nn.Module):
    def __init__(self, out_channels, n_classes, roi_size, hidden_dim=512, p_dropout=0.3):
        super().__init__()
        self.roi_size = roi_size
        self.avg_pool = nn.AvgPool2d(self.roi_size)
        self.fc = nn.Linear(out_channels, hidden_dim)
        self.dropout = nn.Dropout(p_dropout)
        self.cls_head = nn.Linear(hidden_dim, n_classes)
```

شکل ۳۶ - هد کلاسیفیکشن نهایی

میبنیم که در این قسمت یکی از مهمترین بخش ها استفاده از ROI pooling است، ROI pooling در حقیقت وظیفه هم اندازه کردن خروجی پروپوزال های مدل برای ورود به بخش Classification را بر عهده دارد.

همچنین برای محاسبه IOU و ایجاد ورودی مناسب به مدل و بدست آوردن Anchor های مثبت و منفی نیز TA محترم فایلی با عنوان hint ارسال نمودند که شامل توابع مورد نیاز با عنوان Utils بود که از آن ها استفاده کردیم.

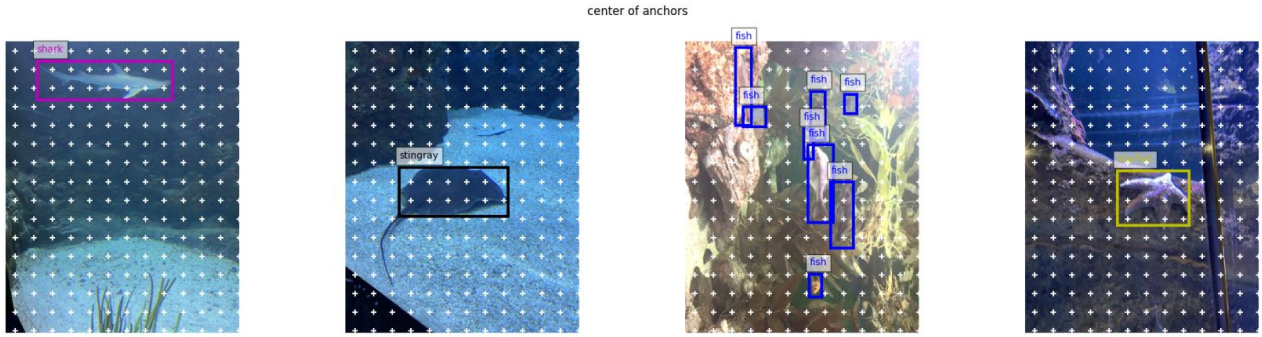
```python
def get_iou_mat(batch_size, anc_boxes_all, gt_bboxes_all):

    # flatten anchor boxes
    anc_boxes_flat = anc_boxes_all.reshape(batch_size, -1, 4)
    # get total anchor boxes for a single image
    tot_anc_boxes = anc_boxes_flat.size(dim=1)

    # create a placeholder to compute IoUs amongst the boxes
    ious_mat = torch.zeros((batch_size, tot_anc_boxes, gt_bboxes_all.size(dim=1)))

    # compute IoU of the anc boxes with the gt boxes for all the images
    for i in range(batch_size):
        gt_bboxes = gt_bboxes_all[i]
        anc_boxes = anc_boxes_flat[i]
        # ious_mat[i, :] = ops.box_iou(anc_boxes, gt_bboxes)
        ious_mat[i, :] = ops.box_iou(anc_boxes, gt_bboxes)


    return ious_mat
```

شکل ۳۷ - تابع **IOU** فراهم شده توسط **TA** در **Utils**

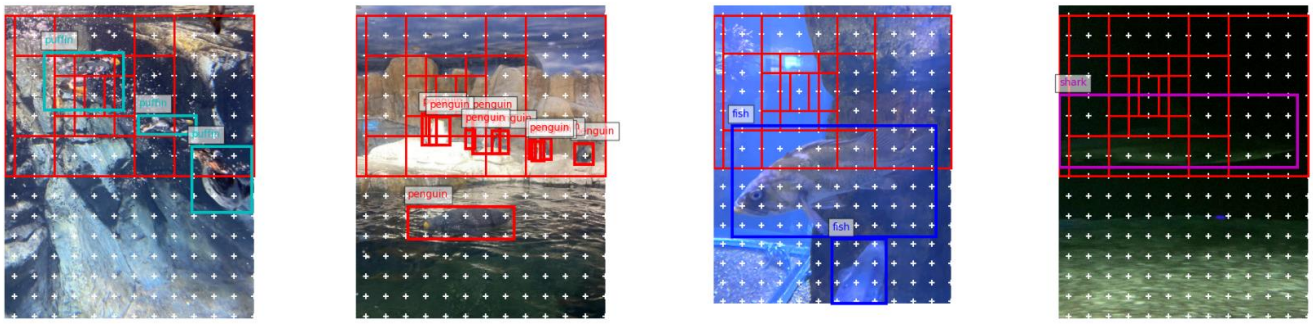
طراحی RPN:

در این بخش به طراحی مدل Region proposal network پرداختیم، این معماری بخش نوآور و بسیار جالب مقاله مورد نظر و مقاله FasterRCNN است که به بررسی قسمت های آن میپردازیم:

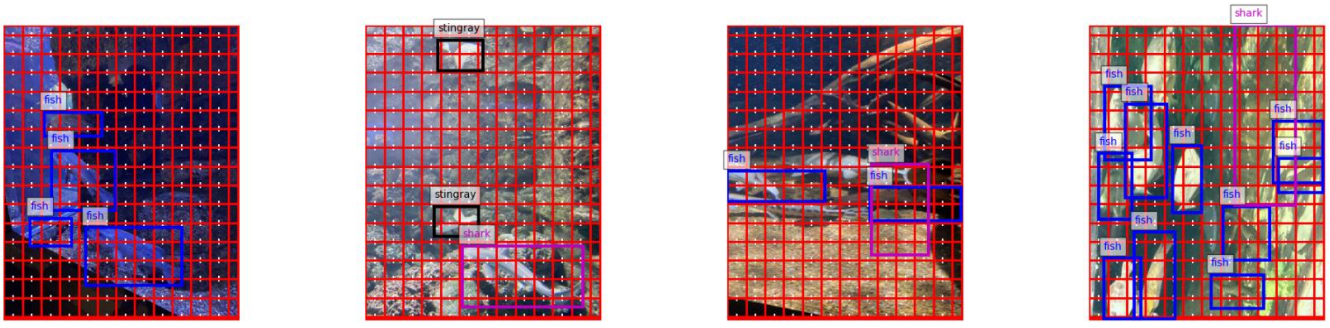
- در ابتدا لازم است Anchor هایی روی بخش های مختلف ورودی ایجاد کنیم، ما در این بخش به ازای هر پیکسل روی feature map که ابعاد ۱۳*۱۶ دارد، ۹ عدد انکرباکس ایجاد کردیم، این انکرباکس ها دارای سه اندازه مختلف و همچنین سه مقدار متفاوت ratio هستند که انواع مقادیر آبجکت ها را بتوانند تشخیص دهند، در ادامه تصاویر مرتبط به انکر ها نمایش داده میشود:



شکل ۴۰- مراکز **Anchor**



شکل ۳۹ - نمایش یک مورد از **Anchor box** های ۹ تایی



شکل ۳۸ - نمایش یک سایر از **Ancorbox** ها روی همه بخش های تصویر

ذکر شد که برای انتخاب مقادیر Anchor به طور تجربی مقادیر مختلفی را تست کردیم و مقادیر زیر را مناسب یافتیم:

```
# scales and ratios for anchor boxes
self.anc_scales = [4, 8, 16]
self.anc_ratios = [0.5, 1, 2]
self.n_anc_boxes = len(self.anc_scales) * len(self.anc_ratios)
```

شکل ۴۱ - تعداد، اسکیل و نسبت **Anchor**ها

دلیل انتخاب ما نیز بدیهی است، مقادیر ۱۶، با توجه به اندازه ۱۶*۱۳ روی فیچرمپ، تقریبا اندازه یک feature map بوده و مقادیر دیگر، نصف و یک چهارم این مقدار است تا از کوچکترین تا بزرگترین آبجکت های موجود قابل تشخیص باشد.

- تابع ایجاد proposals: در بخش دیگر برای طراحی RPN از تابع generate-proposals استفاده کردیم، این تابع با دریافت، همه انکرباکس ها و Offset های روی یک تصویر، مقادیر انکر را به مقادیر واقعی آن ها نزدیک کرده و این بار پروپوزال(انکر های اصلاح شده) را به عنوان ورودی میدهد.
- مدل Proposal: این بخش از مدل که بخشی از RPN است، یک شبکه Convulotion دارای دو هد، است، یک هد به ازای هر پیکسل از فیچر مپ ها، ۹ مقدار به ازای هر پروپوزال، تولید میکند، که در حقیقت یک مقدار classification است که آیا آن پروپوزال دارای آبجکت هست یا نه، و هد دوم، یک خروجی ۴*۹*۱۳*۱۶ تولید میکند، یعنی به ازای هر پیکسل فیچر مپ و ۹ تا پروپوزال آن، ۴ تا عدد خروجی میدهد، این اعداد همان مقادیر offset است که برای نزدیک کردن مقادیر انکر به مقادیر واقعی آن ها استفاده میشود.
- گردهم آوری همه بخش ها در یک مدل RPN: این بخش اصلی شبکه است که در حقیقت پروپوزال های نهایی را برای بخش بعدی یعنی ROI Pooling و در نهایت classification تولید میکند.
- درباره توابع هزینه در RPN: شبکه RPN یک شبکه دومنظوره است، به طوریکه هم، یک لاس classification نیاز دارد، و هم یک لاس regression در حقیقت، بخش اول برای تشخیص اینکه هر انکر شامل آبجکت هست یا نه و بخش دوم برای بدست آوردن مقادیر Offset استفاده میشود، ما توابع لاس را به شکل زیر استفاده کردیم:

```python
def calc_cls_loss(conf_scores_pos, conf_scores_neg, batch_size):
    target_pos = torch.ones_like(conf_scores_pos)
    target_neg = torch.zeros_like(conf_scores_neg)

    target = torch.cat((target_pos, target_neg))
    inputs = torch.cat((conf_scores_pos, conf_scores_neg))
    loss = F.binary_cross_entropy_with_logits(inputs, target, reduction='sum') * 1. / batch_size
    return loss

def calc_bbox_reg_loss(gt_offsets, reg_offsets_pos, batch_size):
    # Ensure gt_offsets and reg_offsets_pos are on the same device
    device = gt_offsets.device
    reg_offsets_pos = reg_offsets_pos.to(device)

    assert gt_offsets.size() == reg_offsets_pos.size()
    loss = F.smooth_l1_loss(reg_offsets_pos, gt_offsets, reduction='sum') * 1. / batch_size
    return loss
```

شکل ۴۲ - توابع **loss** مورد استفاده

در این بخش به بررسی این دو تابع هزینه میپردازیم:

- تابع calc-cls-loss: این تابع، input های مثبت (انکرهایی که مقادیر IOU آن ها در محاسبه با bbox دارای مقادیر بالاتر از threshold مثبت هستند) را به عدد یک، و input های منفی (انکرهایی که مقادیر IOU آن ها در محاسبه با bbox دارای مقادیر کمتر از threshold منفی هستند) به عدد صفر نزدیک میکند.
- تابع calc-bbox-reg-loss: این تابع مقادیر offset بدست آمده از مقایسه بین anchor ها و ground truth bbox ها را به مقادیر خروجی مدل از طریق L۱-loss نزدیک میکند.
- در این قسمت در مجموع از پارامتر های زیر استفاده شد:

Threshold +	Threshold -	#Anchor
۰.۷	۰.۲	۹*Feature_map size

- در نهایت همه بخش ها در یک مدل نهایی جمع آوری و آماده آموزش شد.

آموزش مدل:

در مورد loss بخش RPN در بخش قبل توضیح دادیم، لازم به ذکر است برای loss بخش classification نهایی که وظیفه تشخیص کلاس هر یک از اشیا و همینطور background را دارد، صرفا از یک cross entropy استفاده میشود.

در این بخش از پارامتر های زیر استفاده کردیم:

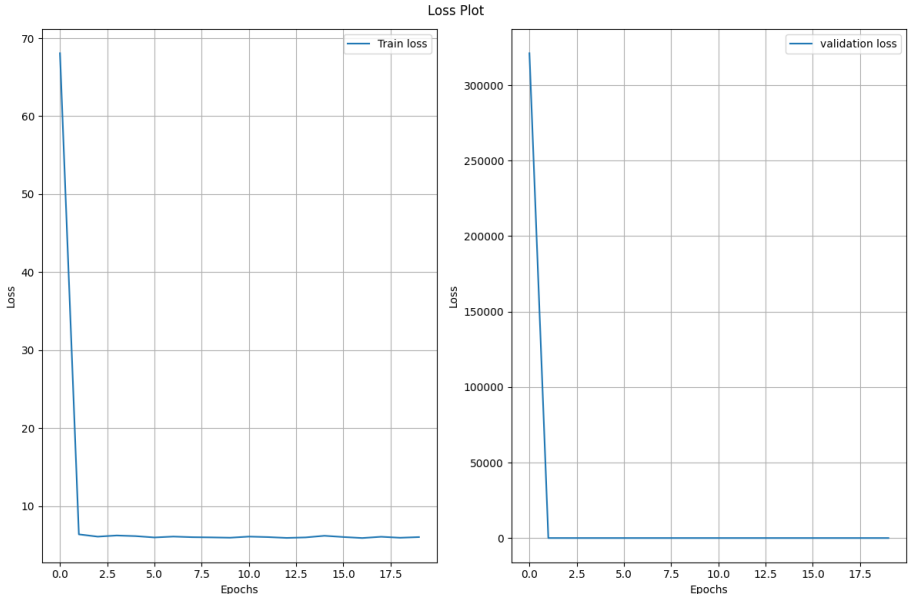
```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
img_size = (500, 400)
out_size = (16, 13)
n_classes = 8
roi_size = (2, 2)
out_channels = 2048

detector = TwoStageDetector(img_size, out_size, out_channels, n_classes, roi_size)
learning_rate = 0.01
detector = detector.to(device)
optimizer = optim.Adam(detector.parameters(), lr=learning_rate)
scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=5, gamma=0.1)
```

شکل ۴۳ - پارامتر های آموزش مدل

- سایز تصویر ورودی: img_size=(۵۰۰,٤۰۰)
- سایر فیچر مپ از resnet: out_size=(١٦, ١٣)
- تعداد چنل های خروجی از resnet: ۲۰۴۸
- تعداد کلاس های نهایی: ۸ شامل ۷ کلاس شی و یک بک گراند
- نرخ یادگیری: ۰.۰۱ شروع و کاهش هر ۵ مرحله ضرب در ۰.۱
- همچنین مقادیر batch-size = ۲۰ و number-worker = ۳ در نظر گرفتیم، که در رسیدن به لاس های کمتر بسیار تاثیر گذار بود، تعداد بیشتر worker ها آموزش را بسیار سریع میکند، زیرا لود شدن دیتا از طریق CPU سریع میشود.

```
train_loader = DataLoader(train_dataset, batch_size=20, shuffle=True, num_workers = 3, collate_fn= collatefn)
valid_loader = DataLoader(valid_dataset, batch_size=20, shuffle=False  num_workers = 3, collate_fn=collatefn)
```

در ادامه نمودارهای کاهش loss را میبنیم:



شکل ۴۴ - نمودار **loss** روی آموزش و ارزیابی

میبینیم که کاهش بسیار زیادی روی loss مشاهده میشود، و در نهایت مقادیر loss روی آموزش و ارزیابی به صورت زیر است:

```
Train Loss: 6.2061, Val Loss: 5.0612, Learning Rate: 1e-05
Saving the best model on validation data
Epoch [16/20]

Train Loss: 6.0487, Val Loss: 5.0695, Learning Rate: 1e-05
Epoch [17/20]

Train Loss: 5.9179, Val Loss: 5.0712, Learning Rate: 1e-05
Epoch [18/20]

Train Loss: 6.0803, Val Loss: 5.0719, Learning Rate: 1e-05
Epoch [19/20]

Train Loss: 5.9538, Val Loss: 5.0810, Learning Rate: 1e-05
Epoch [20/20]
```

که به محدود اعداد ۵ و ۶ رسیده است.

در ادامه کمی در مورد نحوه آموزش مدل صحبت میکنیم:

```
total_rpn_loss, feature_map, proposals, positive_anc_ind_sep, GT_class_pos = self.rpn(images, gt_bboxes, gt_classes)

pos_proposals_list = []
for i in range(len(images)):
    proposal_idxs = positive_anc_ind_sep[i]
    proposals_sep = proposals[proposal_idxs].detach().clone()
    # Ensure proposals_sep is 2D
    if proposals_sep.dim() == 1:
        proposals_sep = proposals_sep.unsqueeze(0)
    # Add batch index to proposals
    batch_indices = torch.full((proposals_sep.shape[0], 1), i, dtype=torch.float32, device=device)
    proposals_with_indices = torch.cat([batch_indices, proposals_sep], dim=1)
    pos_proposals_list.append(proposals_with_indices)

# Debugging statements

# Ensure GT_class_pos is on the same device as feature_map
GT_class_pos = GT_class_pos.to(device)

# Remove batch index before passing to classifier
proposals_list = [proposal[:, 1:] for proposal in pos_proposals_list]

cls_loss = self.classifier(feature_map, proposals_list, GT_class_pos)
total_loss = cls_loss + total_rpn_loss

return total_loss
```

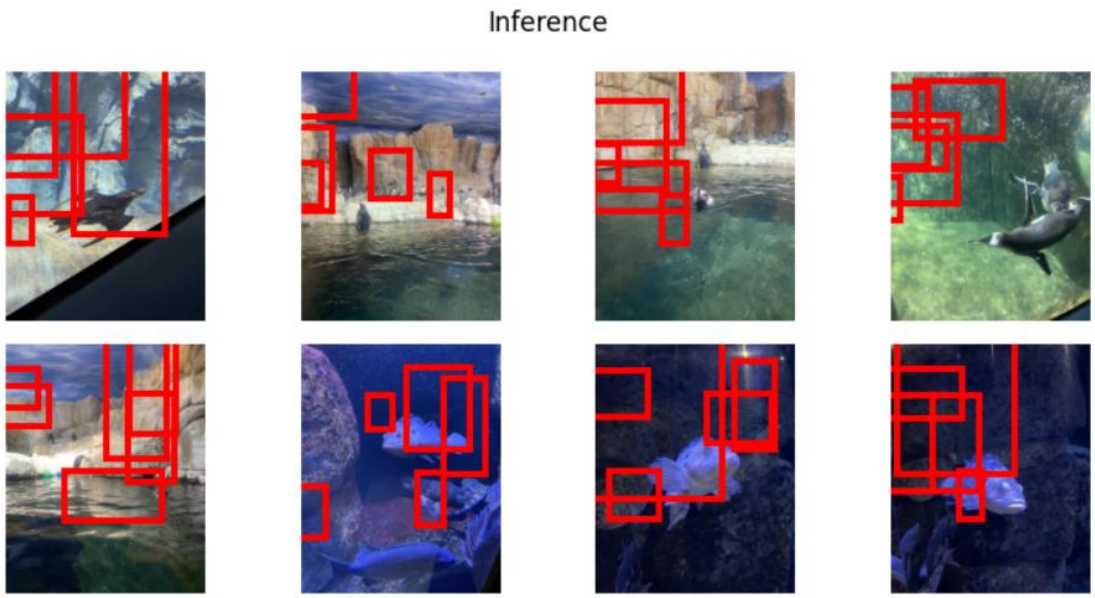
شکل ۴۵ - روند آموزش مدل **Faster RCNN**

- ابتدا، تصاویر و مقادیر صحیح bbox و کلاس لیبل ها وارد شبکه RPN شده، و مقادیر پیشنهادی پروپوزال ها، لیبل های کلاس و هزینه بخش RPN بدست می آید
- بعد از جداسازی و آماده سازی پروپوزال ها، این مقادیر به همراه برچسب کلاس ها وارد بخش classifier که شامل ROI pooling است شده و هزینه بخش classifier ساخته میشود، در نهایت مقادیر loss جمع شده و برای آموزش مدل استفاده میشود.

ارزیابی مدل:

برای ارزیابی مدل، از توابع inference استفاده میشود، نکته مهم در این توابع استفاده از تابع non max suppression است که به حذف تعداد زیادی از پروپوزال هایی که به یک آبجکت اشاره میکند میپردازد و از بین پروپوزال های دارای اشتراک که به یک آبجکت اشاره میکنند، بزرگترین آن ها را بر میگرداند.

در نهایت تصاویر خروجی مدل به شکل زیر هستند:



شکل ۴۶ - تصاویر نهایی خروجی از یک بچ داده تست