به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

درس شبکه‌های عصبی و یادگیری عمیق

تمرین سوم

پرسش ۲	نام و نام خانوادگی	علی صفری
	شماره دانشجویی	۸۱۰۲۰۲۱۵۳
	مهلت ارسال پاسخ	۱۴۰۳.۰۹.۱۳

فهرست

شکل‌ها

جدول‌ها

مقدمه

برای پیاده‌سازی پروژه از بستر Google Colab به منظور کد نویسی و اجرا استفاده شده است. تمامی مراحل کد و اجرای آن در این گزارش به تفصیل شرح داده شده است.

کد های نوشته شده همگی در پوشه‌ی Code و با پسوند ipynd. ذخیره شده است.

پرسش۲–تشخیص تابلوهای راهنمایی و رانندگی

ابتدا دیتاست خواسته شده از لینک مربوطه دانلود شد و بعد از extract کردن در گوگل درایو ذخیره شد. در ادامه حل این سوال برای دسترسی به دیتاست به آدرسی از گوگل درایو خود اشاره خواهد شد. برای حل این سوال از کتابخانه Pytorch استفاده شده است اما علاوه بر آن در بخش‌های متفاوت بسته به نیاز کتابخانه‌های دیگری مانند torchmetrics برای محاسبه mAP استفاده شده است. از دیگر کتابخانه‌های مورد استفاده torchvision و matplotlib می‌باشد.

کد مربوط به بخش اول سوال در فایل HW3_2_1 ذخیره شده است. کد مربوط به بخش دوم و مدل Faster R-CNN در فایل HW3_2_1 و کد مربوط به بخش سوم و مدل SSD300 در فایل HW3_2_3 و همگی این فایل ها در فولدر Code و داخل Q2 ذخیره شده اند.

نکات حائز اهمیت در اجرای کدهای این سوال:

۱) تمامی عکس‌های دیتاست دانلود و اکسترکت شده در یک فولدر در گوگل درایو. مسیری که تمامی تصاویر و فایل‌های ذخیره شده در آن قرار گرفته اند : "/content/drive/MyDrive/GTSDB/FullIJCNN2013/".

پس برای اجرای صحیح کد باید تمامی تصاویر و محتویات فایل دانلود شده در مسیر فوق در گوگل درایو در دسترس باشند.

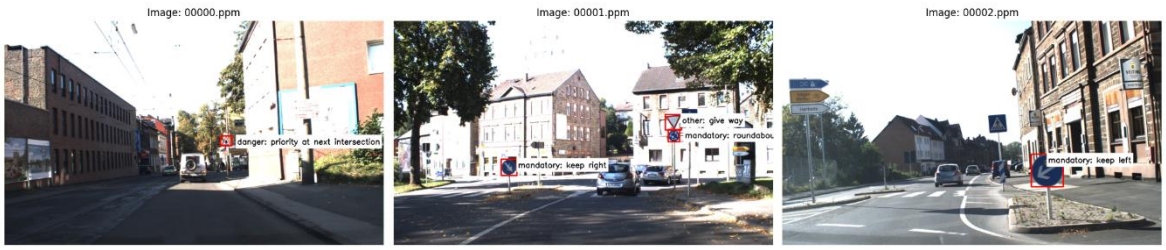
۲) برای اجرای صحیح کدها ابتدا باید کد موجود در فایل HW3_2_1 اجرا شود و سپس به سراغ کدهای دو بخش بعد بروید چرا که با اجرای فایل مذکور دو فایل با عنوان npy در همان مسیر قبل ذخیره خواهد شد که نتیجه تقسیم کل دیتاست به دو مجموعه آموزش و ارزیابی است. لذا بدون اجرای کد اول این دو فایل در گوگل درایو شما ایجاد نخواهد شد و نتیجتا اجرای کد بخش‌های بعد که مستلزم وجود دو فایل آموزش و تست با پسوند npy هستند با خطا مواجه خواهد شد.

۲-۱.آماده سازی مجموعه داده

این مجموعه داده حاوی ۹۰۰ عکس طبیعی از حالت ترافیکی در آلمان است که با فرمت ppm ذخیره شده است. هر عکس حاوی مشخصه (annotation) هایی از جمله شماره کلاس و همچنین مختصات region of interest در عکس است. این مشخصه‌ها برای هر عکس در فایلی تحت عنوان ground truth که در gt.txt ذخیره شده است قرار دارد. هر سطر از این فایل تکست حاوی شماره عکس، چهار عدد که بیان

کننده بازه region of interest یا اینجا همان bounding box ما هستند و در نهایت یک عدد به عنوان کلاس آن عکس گزارش شده است. البته لازم به ذکر است که یک عکس می‌تواند حاوی چند تابلو و به عبارتی چندین کلاس و ROI باشد. در این صورت یک شماره از عکس در فایل gt.txt در چند سطر تکرار شده است و در هر سطر bounding box مربوط به آن تابلو خاص و همچنین کلاس آن تابلو به عنوان کلاس گزارش شده است. مجموعه داده حاوی ۴۲ عدد به عنوان کلاس‌های متفاوت است اما به صورت کلی ما چهار دسته بندی برای تابلو ها داریم که شامل prohibitory و danger و mendatory و other می‌باشد. تقسیم بندی اینکه هر شماره کلاس متعلق به کدام کلاس است در فایل gt.txt آمده است. همچنین عکس‌ها دارای ابعاد متفاوت small و medium و large هستند و نحوه تقسیم بندی آنها بر اساس مقاله به این صورت است که عکس‌های کوچک‌تر از ۳۲ پیکسل در رده small بزرگتر از ۴۵ پیکسل large و بین این دو در رده medium قرار می‌گیرد.

در ادامه به نمایش سه عکس اول می‌پردازیم. هر تابلو با کادر قرمز نمایش داده شده و کلاس مربوطه به رنگ سفید نمایش داده شده است.



شکل ۱- نمایش سه عکس از مجموعه داده با **annotation**

در گام بعد مدل را بر اساس مقاله و بزرگی به سه دسته small که تصاویر کوچک تر از ۳۲ پیکسل بودند و medium بین ۳۲ تا ۴۵ پیکسل و large، بزرگتر از ۴۵ پیکسل تقسیم کردیم. همچنین نوع دیگری از تقسیم بندی تصاویر بر اساس کلاس آنها بوده است که به چهار کلاس prohibitory و danger و mandatory و other تقسیم شده اند. برای تشخیص اینکه هر شماره کلاس متعلق به کدام یک از این چهار کلاس است از فایل ground truth استفاده شده است.

```python
def get_size_category(self, width):
    """
    Categorize object based on width
    """
    if width < 32:
        return 'small'
    elif width < 45:
        return 'medium'
    else:
        return 'large'
```

شکل ۲- تابع تشخیص سایز عکس

```
# Dictionary mapping class IDs to their names
self.class_names = {
    0: "speed limit 20",
    1: "speed limit 30",
    2: "speed limit 50",
    3: "speed limit 60",
    4: "speed limit 70",
    5: "speed limit 80",
    6: "restriction ends 80",
    7: "speed limit 100",
    8: "speed limit 120",
    9: "no overtaking",
    10: "no overtaking (trucks)",
    11: "priority at next intersection",
    12: "priority road",
    13: "give way",
    14: "stop",
    15: "no traffic both ways",
    16: "no trucks",
    17: "no entry",
    18: "danger",
    19: "bend left",
    20: "bend right",
    21: "bend",
    22: "uneven road",
    23: "slippery road",
    24: "road narrows",
    25: "construction",
    26: "traffic signal",
    27: "pedestrian crossing",
    28: "school crossing",
    29: "cycles crossing",
    30: "snow",
    31: "animals",
    32: "restriction ends",
    33: "go right",
    34: "go left",
    35: "go straight",
    36: "go right or straight",
    37: "go left or straight",
    38: "keep right",
    39: "keep left",
    40: "roundabout",
    41: "restriction ends (overtaking)",
    42: "restriction ends (overtaking trucks)"
}
```

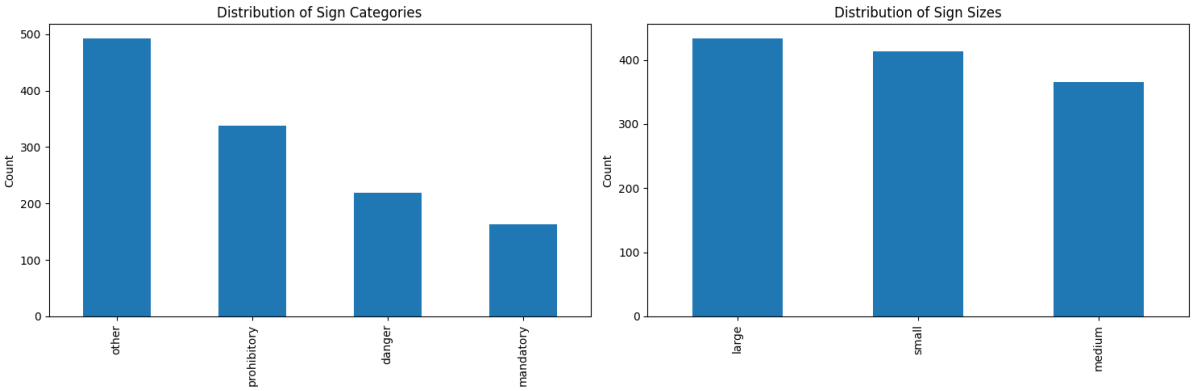
شکل ۳ - نام و شماره کلاس‌های موجود

```python
def get_sign_category(self, class_id):
    """
    Get category of traffic sign based on class ID
    """
    if class_id <= 5 or 7 <= class_id <= 5 or 15 <= class_id <= 16:   # Prohibitory signs
        return 'prohibitory'
    elif 18 <= class_id <= 31 or class_id == 11:  # Danger signs
        return 'danger'
    elif 33 <= class_id <= 40:  # Mandatory signs
        return 'mandatory'
    else:  # Other signs
        return 'other'
```

شکل ۴ - تابع تشخیص کلاس عکس با توجه به **class_id**

نکته مهم در اختصاص هر کلاس ایدی به کلاس مشخصی است که تمامی اعداد از فایل ground truth خوانده شده و در این قسمت بر همین اساس بازه‌ها تعریف شده اند.

در ادامه تابعی را نوشته و با توجه به اینکه هر عکس در چه کلاسی از نظر سایز و دسته بندی تابلو قرار دارد نمودار فراوانی را ترسیم کرده ایم.



شکل ۵ - توزیع فراوانی تصاویر از نظر اندازه و کتگوری

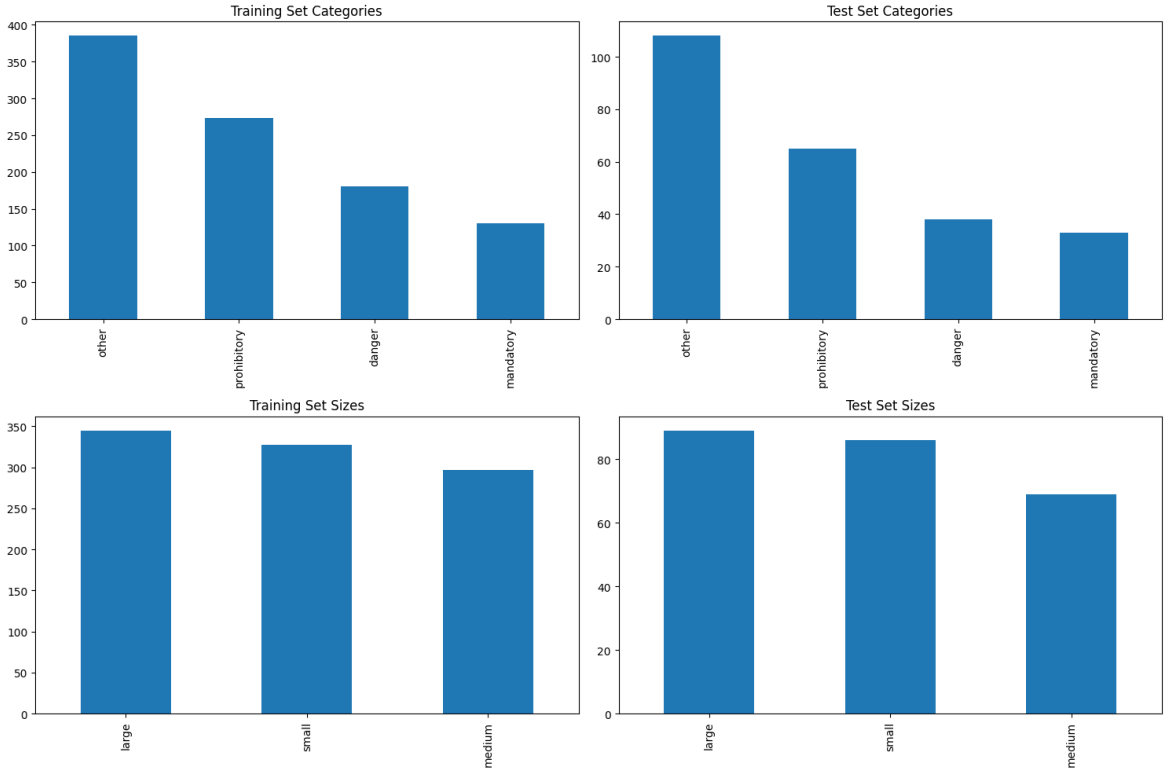
در مرحله بعد ۲۰ درصد داده‌ها را برای test و باقی را برای train اختصاص دادیم. برای این کار تابعی نوشته و بعد از تصادفی کردن ساختار کل تصاویر با استفاده از shuffle به تقسیم دیتا به این دو دسته کردیم. در ادامه کل دیتاست مربوطه را در فایلی تحت عنوان train_files و test_files با پسوند npy در گوگل درایو خود ذخیره کردم تا به این دیتا دسترسی داشته و در ادامه نیز از همین تصاویر برای آموزش و ارزیابی استفاده کنیم. پسوند npy نوعی فایل با فرمت باینری و ساخته شده با کتابخانه numpy است که حجم اندکی داشته و اطلاعات لازم که شامل اسم تصاویر ذخیره شده هستند را در خود نگه می‌دارد. در ادامه هر کجا که نیاز به تصاویر داشته باشیم تصاویری که در اینجا ذخیره شده اند را به عنوان تصاویر آموزش استفاده خواهیم کرد و هر کجا نیاز به annotation های مربوط به آن باشد از فایل ground truth

استفاده خواهیم کرد. با اینکار به جای صرف فضای اضافی برای ذخیره خود تصاویر در یک فولدر جدا صرفا نام تصاویر را ذخیره کرده ایم.

```python
def create_train_test_split(self, test_ratio=0.2):
    # Create split
    all_files = list(self.annotations.keys())
    np.random.seed(42)  # For reproducibility
    np.random.shuffle(all_files)

    split_idx = int(len(all_files) * (1-test_ratio))
    train_files = all_files[:split_idx]
    test_files = all_files[split_idx:]

    # Function to get distributions
    def get_distributions(files):
        categories = []
        sizes = []
        for f in files:
            for ann in self.annotations[f]:
                categories.append(self.get_sign_category(ann['class_id']))
                width = ann['x2'] - ann['x1']
                sizes.append(self.get_size_category(width))
        return categories, sizes

    # Get distributions for both sets
    train_categories, train_sizes = get_distributions(train_files)
    test_categories, test_sizes = get_distributions(test_files)

    # Plot distributions
    fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(15, 10))

    # Plot category distributions
    pd.Series(train_categories).value_counts().plot(kind='bar', ax=ax1, title='Training Set Categories')
    pd.Series(test_categories).value_counts().plot(kind='bar', ax=ax2, title='Test Set Categories')

    # Plot size distributions
    pd.Series(train_sizes).value_counts().plot(kind='bar', ax=ax3, title='Training Set Sizes')
    pd.Series(test_sizes).value_counts().plot(kind='bar', ax=ax4, title='Test Set Sizes')

    plt.tight_layout()
    plt.show()

    # Save split information
    save_path = "/content/drive/MyDrive/GTSDB/FullIJCNN2013/"
    print(train_files)
    np.save(os.path.join(save_path, 'train_files.npy'), train_files)
    np.save(os.path.join(save_path, 'test_files.npy'), test_files)
```

شکل ۶ - تابع **split data** و ذخیره اطلاعات و همچنین نمایش **distribution**

در داخل این تابع، تابعی برای نمایش فراوانی دو دسته هم نوشته شده است. فراوانی کلاس آموزش و ارزیابی پس اس تقسیم بندی به فرم زیر است.



شکل ۷ - فراوانی مجموعه **train** و **test**

۲-۲.تنظیم دقیق و ارزیابی مدل تشخیص شی دو مرحله ای

۲-۲-۱. تعریف شبکه **Faster R-CNN** و **Resnet-50** به عنوان **Backbone**

Faster R-CNN (Region-based Convolutional Neural Network)یک چارچوب دو مرحله‌ای برای تشخیص اشیا است که به دلیل دقت بالا به طور گسترده مورد استفاده قرار می‌گیرد. این روش ابتدا از طریق یک **Region Proposal Network (RPN)**به تولید **Region Proposal**ها پرداخته و سپس این نواحی پیشنهادی را طبقه‌بندی کرده و جعبه‌های مرزی آن‌ها را اصلاح می‌کند. این رویکرد-"coarse to-fine"امکان موقعیت‌یابی دقیق و شناسایی اشیا را در تصویر فراهم می‌کند Faster R-CNN .به‌ویژه در سناریوهایی که نیاز به دقت بالا دارند، موثر است، اگرچه از نظر سرعت نسبت به روش‌های یک مرحله‌ای مانند SSD یا YOLO کندتر است.

در این پروژه، Faster R-CNNبا **ResNet-50**، که یک **Feature Extractor**قوی و کارآمد است، ترکیب شده است) **ResNet-50** .مخفف Residual Network با ۵۰ لایه (از یک چارچوب **Residual Learning**استفاده می‌کند که مشکل **vanishing gradient**در شبکه‌های عمیق را با معرفی **skip connections**کاهش می‌دهد. این اتصالات امکان یادگیری نقشه‌های باقیمانده (residual mappings) را

به‌جای نقشه‌های مستقیم فراهم می‌کنند، و این موضوع باعث می‌شود که شبکه حتی در عمق زیاد، ویژگی‌های بهتر و دقیق‌تری یاد بگیرد.

Backbone ResNet-50نقش حیاتی در استخراج ویژگی‌های سلسله‌مراتبی و غنی از متن از تصاویر دارد. این روش اطمینان می‌دهد که جزئیات سطح پایین (مانند لبه‌ها) و انتزاع‌های سطح بالا (مانند بخش‌های اشیا) به‌طور موثری ثبت می‌شوند. تعادل بین کارایی محاسباتی و دقت آن، ResNet-50را به یک انتخاب محبوب برای وظایف تشخیص اشیا، به‌ویژه زمانی که منابع محاسباتی محدود است، تبدیل کرده است. نکته مهم دیگر این است که ResNet-50 روی ImageNet، Pre-train شده است که یک دیتاست بسیار بزرگ است.

در این پروژه، ResNet-50باعث می‌شود Faster R-CNN بتواند تشخیص دقیق علائم راهنمایی و رانندگی را انجام دهد، با بهره‌گیری از قابلیت آن در یادگیری ویژگی‌های دقیق در مقیاس‌های مختلف اشیا.

۲-۲-۲. آماده‌سازی اولیه برای تنظیم دقیق

۱. بارگیری مدل از پیش‌آموزش دیده

در ابتدا، مدل **Faster R-CNN**با Backbone از نوع **ResNet-50**که از پیش بر روی مجموعه داده **ImageNet**آموزش دیده است، بارگیری می‌شود. این مدل به عنوان پایه برای استخراج ویژگی استفاده می‌شود، و به دلیل پیش‌آموزش آن روی داده‌های عمومی، ویژگی‌های کلی تصویر را به خوبی شناسایی می‌کند.

۲. جایگزینی لایه‌های پایانی مدل

برای تطبیق مدل با مسئله تشخیص علائم راهنمایی و رانندگی، لایه‌های پایانی **Classifier**جایگزین می‌شوند. این لایه‌ها تعداد خروجی‌های مربوط به تعداد کلاس‌ها را تولید می‌کنند که شامل **۵ کلاس** است. چهار کلاس اول همان کتگوری‌هایی هستند که در قسمت اول سوال به آن اشاره کردیم: Prohibitory و Danger و Mandatory و Other و یک کلاس هم به عنوان Background. نکته بسیار مهم و حائز اهمیت در این بخش این است که کلاس background در Faster R-CNN به عنوان کلاس صفر از قبل رزرو شده است. و این بدان معنا است که هر کلاس دیگری که بخواهیم جایگزین کنیم باید از یک شروع شود.

```python
class GTSDBDataset(Dataset):
    def __init__(self, image_dir, annotations_file, image_names, transforms=None):
        self.image_dir = image_dir
        self.image_names = image_names
        self.transforms = transforms
        self.annotations = self._load_annotations(annotations_file)
        self.category_map = {
            'prohibitory': 1,
            'danger': 2,
            'mandatory': 3,
            'other': 4
        }
```

شکل ۸ - تعریف کلاس دیتاست و کتگوری‌ها

```python
def _create_model(self, num_classes):
    model = fasterrcnn_resnet50_fpn(pretrained=True)
    in_features = model.roi_heads.box_predictor.cls_score.in_features
    model.roi_heads.box_predictor = FastRCNNPredictor(in_features, num_classes)
    return model
```

شکل ۹ - ساخت اولیه مدل با بارگیری **Faster R-CNN** و تعریف دوباره تعداد کلاس‌ها با توجه به مسئله

خط اول کار مورد اول را انجام می‌دهد و مدل را بارگیری می‌کند.

اما خط دوم تعداد **ویژگی‌های ورودی** (Input Features)را که برای لایه cls_scoreدر **box predictor**مدل مورد نیاز است، استخراج می‌کند.

شرح دقیق

- model.roi_heads:
  - این بخش مربوط به **Region of Interest (RoI) heads**در مدل Faster R-CNN است. وظیفهRoI heads ، دو بخش زیر است:

۱. **طبقه‌بندی** (Classification)اشیاء در مناطق پیشنهادی (Regions of Interest)که توسط RPN (Region Proposal Network) تولید شده‌اند.

۲. **پیش‌بینی جعبه‌های مرزی** (Bounding Box Regression)برای آن اشیاء.

- box_predictor.
  - این بخش در RoI heads شامل لایه‌های نهایی است که:

۱. نمره هر کلاس را پیش‌بینی می‌کند.(cls_score)

۲. مختصات جعبه‌های مرزی را اصلاح می‌کند.(bbox_pred)

- cls_score:
  - این لایه **fully connected (FC)** است که نمرات مربوط به احتمال هر کلاس از جمله background را پیش‌بینی می‌کند.
- in_features:
  - این ویژگی تعداد ورودی‌هایی که لایه cls_score نیاز دارد را مشخص می‌کند. این ورودی‌ها از ویژگی‌هایی که در مراحل قبلی مانند Backbone و RoI Pooling استخراج شده‌اند، تشکیل می‌شوند.

در نتیجه متغیر in_features اکنون تعداد ویژگی‌های ورودی مورد نیاز برای لایه طبقه‌بندی را نگه می‌دارد. این مقدار در خط بعدی برای ساخت یک **box predictor** جدید استفاده می‌شود.

اما خط سوم **box predictor** اصلی مدل Faster R-CNN را با یک **box predictor** جدید جایگزین می‌کند که تعداد کلاس‌های آن برابر با کلاس‌های موجود در داده‌های پروژه است.

شرح دقیق

- FastRCNNPredictor:
  - این کلاس در کتابخانه torchvision تعریف شده و برای جایگزینی آسان **box predictor** در Faster R-CNN استفاده می‌شود.
  - این کلاس شامل دو بخش است:

۱. **classification head**: پیش‌بینی احتمال هر کلاس (به تعداد num_classes، شامل background).

۲. **bounding box regression head**: پیش‌بینی اصلاحات مختصات جعبه‌های مرزی.

- in_features:
  - تعداد ویژگی‌های ورودی که از خط اول به دست آمده و به طبقه‌بندی‌کننده جدید منتقل می‌شود.
- num_classes:

- تعداد کل کلاس‌ها در مجموعه داده. این مقدار شامل یک کلاس اضافه برای **background**است (برای مناطقی که هیچ شیئی ندارند).

چگونگی عملکرد

۱. **Box Predictor قدیمی:**

- مدل اصلی Faster R-CNN که از پیش‌آموزش دیده است) برای مثال روی COCO یا ImageNet)، یک **box predictor** با تعداد کلاس‌های عمومی دارد. مثلاً در COCO ، تعداد کلاس‌ها ۹۱ است. 80 کلاس اشیاء + ۱ کلاس background + 10 کلاس اضافی

۲. **Box Predictor جدید:**

- این خط، **box predictor** قدیمی را با یک مورد جدید جایگزین می‌کند که تعداد کلاس‌ها را برای داده‌های پروژه تنظیم می‌کند. برای مثال، اگر داده‌های شما شامل سه کلاس"prohibitory" ، "danger"و "other" و "mandatory"و background می‌باشد، مقدار num_classes برابر ۵ خواهد بود.

نتیجه

- box predictorجدید اکنون می‌تواند جعبه‌های مرزی و کلاس‌ها را برای تعداد دقیق کلاس‌های مورد نیاز در پروژه پیش‌بینی کند.

باقی عملیات‌های آماده سازی و پیاده‌سازی مدل که شامل آموزش آن هم می‌شود در ادامه و بسته به صورت سوال شرح داده شده است.

۲-۲-۳. عملیات آماده‌سازی و نرمال کردن

عملیات آماده‌سازی تصاویر برای مدل Faster R-CNN شامل مشخص کردن annotation هر عکس با توجه به فایل gt است. این مراحل شامل مشخص کردن bounding box و label هر عکس است. در گام بعد به سراغ normal کردن عکس می‌رویم. برای این کار مطابق عکس زیر عمل می‌کنیم.

```python
def get_transforms(train):
    transforms = []
    transforms.append(T.ToTensor())
    transforms.append(T.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]))
    if train:
        transforms.append(T.RandomHorizontalFlip(0.5))
    return T.Compose(transforms)
```

شکل ۱۰- تعریف تابع **transform** برای نرمال کردن و **augmentation**

اعداد موجود در دستور Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]) میانگین (Mean) و انحراف معیار Standard Deviation) یا (Std رنگ‌های قرمز(R) ، سبز (G) و آبی (B) در تصاویر دیتاست ImageNet هستند.

چرا این اعداد استفاده می‌شود؟

۱. **میانگین (Mean)** : این اعداد نشان‌دهنده میزان روشنایی میانگین هر کانال رنگی هستند. برای مثال، عدد ۰.۴۸۵ مربوط به کانال قرمز است و نشان می‌دهد که روشنایی این کانال به طور میانگین برابر با ۰.۴۸۵ است.

۲. **انحراف معیار (Std)** : این اعداد نشان‌دهنده پراکندگی یا پخش‌شدگی شدت رنگ‌ها در تصاویر هستند. به عنوان مثال، مقدار ۰.۲۲۹ برای کانال قرمز به این معنا است که شدت روشنایی در این کانال به اندازه این مقدار بالا و پایین می‌شود.

کاربرد نرمال‌سازی:

۱. **استاندارد کردن داده‌ها :**نرمال‌سازی باعث می‌شود داده‌های ورودی به شبکه عصبی به یک بازه استاندارد تبدیل شوند. این کار برای بهبود عملکرد مدل بسیار ضروری است. فرمول نرمال‌سازی برای هر کانال به شکل زیر است:

$$Pixel\_normalized = (Pixel\_original - Mean) / Std$$

در این فرمول:

- Pixel_originalمقدار اصلی هر پیکسل (بین ۰ و ۱) است.
- Meanمیانگین مربوط به کانال رنگی است (برای مثال، ۰.۴۸۵ برای قرمز).
- Stdانحراف معیار کانال رنگی است (برای مثال، ۰.۲۲۹ برای قرمز).

۲. **هماهنگی با مدل‌های پیش‌آموزش‌دیده :**مدل‌هایی که روی دیتاست ImageNet آموزش دیده‌اند، انتظار دارند داده‌های ورودی دارای ویژگی‌های آماری مشابه (میانگین و انحراف معیار) باشند. اگر داده‌ها به این صورت نرمال‌سازی نشوند، ممکن است دقت مدل کاهش یابد.

همانطور که در تصویر فوق قابل مشاهده است علاوه برا عملیات نرمال کردن عکس‌ها برای پیش پردازش عملیات داده‌افزایی کوچکی هم به صورت اضافه انجام شده است. این کار الزامی نبوده و صرفا به جهت افزایش دقت مدل به صورت dynamic و random در میان epoch ها چندین عکس به صورت افقی flip شده است. به صورت کلی این کار باعث می‌شود مدل الگوهای موجود را بهتر یاد گرفته و از overfit شدن دور شود. البته لازم به ذکر است که اگر دیتا در مجموعه train ما بوده باشد این کار صورت می‌گیرد و برای داده‌ی تست اینکار انجام نمی‌شود.

در نهایت با استفاده از T.Compose تنسور تولید شده ما قابل فراخوانی می‌شود. حال دیتا آماده آموزش است.

برای مهیا کردن دیتاست برای آموزش و ارزیابی کلاسی تحت عنوان GTSDBDataset تعریف شده است. در این کلاس که تا پیش از این چندین تابع به کار رفته در آن را هم دیده‌اید مراحل آماده سازی دیتاست برای آموزش انجام می‌شود. این مراحل به شرح زیر است:

۱) ابتدا annotation های هر عکس مشخص شده است. به این معنا که از فایل gt، Bounding box مشخص شده و شماره کلاس اون مشخص شده است.
۲) در گام بعد با توجه به شماره کلاس هر عکس مشخص شده که در کدام یک از چهار کلاس prohibitory و danger و mandatory و other تعلق دارد.
۳) در گام بعد یک دیکشنری برای هر عکس درست شده است که در اون box مربوطه و label اون عکس که به هر کلاس یک عدد از یک تا چهار داده شده نسبت داده شده و در نهایت image_id که اسم اون عکس هست.
۴) در نهایت عکس از تابع get_transform عبور می‌کند تا عملیات نرمال کردن و در صورتی که داده آموزش باشد، عملیات داده افزایی هم به صورت رندوم روی آن اعمال شود.

```python
class GTSDBDataset(Dataset):
    def __init__(self, image_dir, annotations_file, image_names, transforms=None):
        self.image_dir = image_dir
        self.image_names = image_names
        self.transforms = transforms
        self.annotations = self._load_annotations(annotations_file)
        self.category_map = {
            'prohibitory': 1,
            'danger': 2,
            'mandatory': 3,
            'other': 4
        }

    def _load_annotations(self, annotations_file):
        annotations = {}
        with open(annotations_file, 'r') as f:
            for line in f:
                filename, left, top, right, bottom, class_id = line.strip().split(';')
                if filename not in annotations:
                    annotations[filename] = []
                annotations[filename].append({
                    'x1': int(left),
                    'y1': int(top),
                    'x2': int(right),
                    'y2': int(bottom),
                    'class_id': int(class_id)
                })
```

شکل ۱۱- کلاس آماده سازی دیتاست بخش اول

```python
def get_category(self, class_id):
    if class_id <= 5 or 7 <= class_id <= 10 or class_id in [15, 16]:
        return 'prohibitory'
    elif 18 <= class_id <= 31 or class_id == 11:
        return 'danger'
    elif 33 <= class_id <= 40:
        return 'mandatory'
    return 'other'

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

def __getitem__(self, idx):
    img_name = self.image_names[idx]
    img_path = os.path.join(self.image_dir, img_name)
    img = Image.open(img_path).convert("RGB")

    boxes = []
    labels = []

    if img_name in self.annotations:
        for ann in self.annotations[img_name]:
            boxes.append([ann['x1'], ann['y1'], ann['x2'], ann['y2']])
            category = self.get_category(ann['class_id'])
            labels.append(self.category_map[category])

    if not boxes:
        boxes = torch.zeros((0, 4), dtype=torch.float32)
        labels = torch.zeros(0, dtype=torch.int64)
    else:
        boxes = torch.as_tensor(boxes, dtype=torch.float32)
        labels = torch.as_tensor(labels, dtype=torch.int64)

    target = {
        'boxes': boxes,
        'labels': labels,
        'image_id': torch.tensor([idx])
    }

    if self.transforms:
        img = self.transforms(img)

    return img, target
```

شکل ۱۲- کلاس آماده سازی دیتاست بخش دوم

```python
# Create datasets
train_dataset = GTSDBDataset(image_dir, annotations_file, train_names, get_transforms(train=True))
test_dataset = GTSDBDataset(image_dir, annotations_file, test_names, get_transforms(train=False))
```

شکل ۱۳ - ایجاد دیتاست با استفاده از کلاس **GTSDBDataset**

بعد از آماده شدن دیتاست برای آموزش و ارزیابی از DataLoader استفاده کردیم تا سرعت و استفاده از GPU را بهینه تر کنیم.

```
# Create data loaders
train_loader = DataLoader(
    train_dataset, batch_size=8, shuffle=True,
    collate_fn=lambda x: tuple(zip(*x)), num_workers=2
)
test_loader = DataLoader(
    test_dataset, batch_size=8, shuffle=False,
    collate_fn=lambda x: tuple(zip(*x)), num_workers=2
)
```

شکل ۱۴- تعریف **DataLoader**

در تعریف دیتالودر ابتدا دیتاست مربوط به هر بخش فرستاده شده است سپس batch size مساوی ۸ در نظر گرفته شده، سپس shuffle فقط برای آموزش روشن شده و در بخش بعد چگونگی ارسال عکس ها در دیتالودر مشخص شده که از زیپ استفاده شده، به این معنا که هر عکس و annotation مربوط بهش ارسال شده است. در انتها تعداد worker ها برابر دو قرار داده شده که به معنای **worker processes** است که به صورت همزمان به کار گرفته شده اند. این کار به افزایش سرعت کمک می‌کند.

همه چیز آماده و مهیا برا آموزش است. برای آموزش کلاس TrafficSignDetector نوشته شده است. و در این قسمت بخش مهم کار تابع train است. اما توابع evaluation هم داریم که در ادامه به آنها اشاره می‌کنیم.

در مرحله آموزش فرض‌هایی صورت گرفته است که به آنها اشاره می‌شود:

۱) **تعداد epoch**: این مقدار در مقاله یا صورت سوال اشاره نشده است و ما با توجه به زمانبر بودن اجرا و دست یافتن به دقت قابل قبول، تعداد ۱۰ ایپاک را در نظر گرفتیم.

۲) **Batch size**: این مقدار هم مشخصا تعریف نشده است. برای این سوال عدد ۸ را برای batch size در نظر گرفته‌ایم.

اکنون مدل آماده آموزش است و با استفاده از تابع زیر که در کلاس TrafficSignDetector تعریف شده این کار را انجام می‌دهیم. برای ارزیابی و چگونگی انجام آن در ادامه توضیح داده شده است.

۲-۲-۴. معیارهای IoU و mAP

IoU (Intersection over Union)

IoUیکی از معیارهای اصلی در ارزیابی دقت پیش‌بینی‌های مدل‌های تشخیص شی است IoU .میزان همپوشانی میان جعبه پیش‌بینی شده (Predicted Bounding Box) و جعبه واقعیت (Ground Truth Bounding Box)را محاسبه می‌کند. فرمول IoU به صورت زیر است:

- **مقدار بالا (مثلاً ۰.۷۵) :**پیش‌بینی باید دقیقاً با جعبه واقعیت مطابقت داشته باشد.
- **مقدار پایین (مثلاً ۰.۵) :**پیش‌بینی با همپوشانی جزئی نیز پذیرفته می‌شود.

در ارزیابی مدل:

- اگر IoU > **آستانه (مثلاً ۰.۵)** باشد، پیش‌بینی به عنوان درست (True Positive) پذیرفته می‌شود.
- در غیر این صورت، پیش‌بینی به عنوان خطا (False Positive) محسوب می‌شود.

mAP (Mean Average Precision)

mAPیک معیار ارزیابی کلی برای مدل‌های تشخیص شی است که هم **دقت (Precision)** و هم **بازخوانی (Recall)** را در بر می‌گیرد. ابتدا، برای هر کلاس **AP (Average Precision)**محاسبه شده و سپس میانگین آن‌ها به عنوان mAP گزارش می‌شود.

فرمول:

$$mAP = (1/N) \Sigma AP\_i$$

$$AP = \int p(r)\, dr$$

- **P(r):**دقت در یک مقدار خاص بازخوانی.
- **mAP:**میانگین AP برای تمامی کلاس‌ها.

mAPبا IoU در ارتباط است زیرا محاسبه AP به تعداد پیش‌بینی‌های درست (TPs) که از یک آستانه IoUعبور می‌کنند، بستگی دارد.

پیاده‌سازیIoU

در کد، IoUبه طور غیرمستقیم برای ارزیابی پیش‌بینی‌ها استفاده شده است IoU .در متریک‌های از پیش تعریف‌شده مثل **MeanAveragePrecision**از کتابخانه PyTorch Metrics پیاده‌سازی شده است:

```
metric = MeanAveragePrecision(iou_type="bbox", iou_thresholds=[0.5], class_metrics=True)
```

شکل ۱۵ - پیاده‌سازی **IoU**

iou_type="bbox" مشخص می‌کند که IoU برای جعبه‌ها محاسبه می‌شود.

این تابع خروجی‌های مدل را با داده‌های واقعیت مقایسه کرده و IoU و سایر معیارها مثل mAP را محاسبه می‌کند.

پیاده‌سازی mAP

mAPدر کد شما برای ارزیابی دقت مدل در طول فرآیند اعتبارسنجی استفاده می‌شود:

```
@torch.no_grad()
def evaluate(self, data_loader):
    self.model.eval()
    metric = MeanAveragePrecision(iou_type="bbox",iou_thresholds=[0.5], class_metrics=True)

    for images, targets in data_loader:
        images = [img.to(self.device) for img in images]
        targets = [{k: v.to(self.device) for k, v in t.items()} for t in targets]
        outputs = self.model(images)

        # Adjust labels to start from 0 before passing to the metric
        for target in targets:
            target['labels'] = target['labels'] - 1
        for output in outputs:
            output['labels'] = output['labels'] - 1

        metric.update(outputs, targets)

    computed_metrics = metric.compute()
    return computed_metrics['map'].item()
```

شکل ۱۶ - پیاده سازی تابع ارزیابی و **mAP**

در این تابع ابتدا مدل ارزیابی شده است سپس با استفاده از MeanAveragePrecision از مدل خواسته شده است که بر اساس bounding box مقدار IoU را محاسبه کند و در ادامه در قالب یک حلقه label های واقعی و پیشبینی شده آپدیت شده اند و در نهایت با استفاده از compute محاسبات انجام شده و مقدار mAP گزارش شده است. نکته مهم در این بخش این است که در تابع MeanAvaragePrecision فرض بر شروع کلاس‌ها از صفر است اما ما می‌دانیم که در مدل خود از عدد ۱ شروع کردیم چرا که کلاس background با عدد صفر از قبل رزرو شده بود. به همین دلیل شماره label منهای یک شده است تا بجای یک تا چهار از صفر تا سه باشد و عملیات را به درستی انجام دهد. در صورت عدم انجام این کار عدد mAP منفی یک گزارش می‌شود که صحیح نمی‌باشد. با اینکار مقدار mAP برای هر کلاس و با توجه به IoU threshold تعریف شده محاسبه می‌شود. نکته حائز اهمیت در این بخش این است که به صورت پیش فرض

و با توجه به صورت سوال این threshold برابر با **0.5** در نظر گرفته خواهد شد. اما برای ارزیابی بهتر مدل بهتر است با IoU های متفاوت بررسی کنیم که در بخش‌های بعدی به این مورد خواهیم پرداخت.

۲-۲-۵. بهینه‌ساز و تابع هزینه

تابع هزینه

برای تابع هزینه یا loss از آنچه در درون خود Faster R-CNN بوده استفاده کرده ایم که به شکل زیر پیاده سازی شده است:

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

شکل ۱۷- پیاده‌سازی تابع هزینه در مدل **Faster R-CNN**

خطاها (loss) در یک دیکشنری ذخیره می‌شوند و شامل چهار جزء هستند:

- **loss_classifier**: خطای Cross-Entropy برای کلاسه‌بندی.
- **loss_box_reg**: خطای Smooth L1 برای تنظیم موقعیت جعبه.
- **loss_objectness**: خطای Binary Cross-Entropy برای احتمال حضور شیء در RPN.
- **loss_rpn_box_reg**: خطای Smooth L1 برای تنظیم موقعیت جعبه در RPN.

فرمول کلی محاسبه خطا به شکل زیر است:

Total Loss = loss_classifier + loss_box_reg + loss_objectness + loss_rpn_box_reg

این خطای چند وظیفه‌ای برای فرآیند تشخیص دو مرحله‌ای طراحی شده است:

۱. **مرحله اول (RPN):**
   - خطای احتمال حضور شیء و خطای موقعیت جعبه در RPN شبکه تولید ناحیه را آموزش می‌دهند.

۲. **مرحله دوم:**
   - خطای کلاسه‌بندی و تنظیم موقعیت جعبه نتایج نهایی را بهبود می‌بخشند.

- **چرا Smooth L1 Loss انتخاب شده؟**
  - این تابع ترکیبی از مزایای L1 و L2 است:

- **حساسیت کم به نقاط پرت** :برخلاف L2 که در برابر نقاط پرت (outliers) حساسیت بالایی دارد، Smooth L1تاثیر این نقاط را کاهش می‌دهد.
- **آموزش پایدارتر** :زمانی که خطا کوچک است، مانند L2 رفتار می‌کند و زمانی که خطا بزرگ‌تر می‌شود، شبیه L1 رفتار می‌کند.

  - این ویژگی‌ها آن را برای تنظیم موقعیت جعبه‌ها (Bounding Boxes) بسیار مناسب می‌کند، زیرا جعبه‌های اولیه ممکن است فاصله زیادی از مقدار واقعی داشته باشند.

- **چرا Cross-Entropy انتخاب شده؟**

  - **معیار استاندارد کلاسه‌بندی** Cross-Entropy :به خوبی احتمال توزیع خروجی مدل را با برچسب‌های واقعی مقایسه می‌کند.
  - **قابلیت انطباق برای چندکلاسه** :این تابع به راحتی برای مسائلی با تعداد زیاد کلاس‌ها (مثل شناسایی اشیاء مختلف) قابل استفاده است.
  - **سادگی و اثربخشی** :در مسائلی مانند تشخیص اشیاء، این تابع به سرعت همگرایی مدل را تضمین می‌کند.

بهینه‌ساز

برای بهینه‌ساز از AdamW استفاده شده است. همچنین از lr_scheduler هم استفاده شده است تا امکان کاهش و تغییر lr به شکل بهتری مهیا باشد و عملکرد مدل را بهبود بخشد.

```
optimizer = torch.optim.AdamW(params, lr=0.0001, weight_decay=0.0005)
lr_scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=3, gamma=0.1)
```

شکل ۱۸- تعریف بهینه‌ساز برای مدل **Faster R-CNN**

ویژگی‌های:AdamW

- ترکیب نرخ یادگیری تطبیقی Adam با وزن‌زدایی مناسب.
- نرخ یادگیری پایین (۰.۰۰۰۱) برای تنظیم دقیق مدل پیش‌آموزش‌دیده. (عدد نرخ یادگیری اولیه این عدد فرض شده است و الزاما جایی اشاره نشده حتما از این عدد شروع شود.)
- وزن‌زدایی (Weight Decay) برابر ۰.۰۰۰۵ برای جلوگیری از بیش‌برازش.(Overfitting)

دلایل انتخاب AdamW به عنوان بهینه‌ساز

Adaptive Learning Rate.

- **چرا انتخاب شده؟**
  - AdamWاز نرخ یادگیری تطبیقی استفاده می‌کند، که باعث می‌شود مدل برای هر پارامتر نرخ یادگیری بهینه پیدا کند.
  - این ویژگی باعث می‌شود مدل سریع‌تر به جواب بهینه برسد، خصوصاً در مسائل پیچیده با تعداد پارامترهای زیاد.

Weight Decay بهینه

- **چرا مهم است؟**
  - در مدل‌هایی مانند Faster R-CNN و SSD که تعداد زیادی پارامتر دارند، جلوگیری از **بیش‌برازش** (Overfitting)ضروری است.
  - AdamWاز وزن‌زدایی (Weight Decay) مناسب استفاده می‌کند، که مستقیماً بر روی پارامترهای بهینه‌سازی اعمال می‌شود و باعث تنظیم بهتر شبکه می‌شود.

ترکیب مومنتوم(Momentum)

- **چرا مفید است؟**
  - مومنتوم به فرار از کمینه‌های محلی (Local Minima) کمک می‌کند و مدل را به سمت کمینه سراسری (Global Minimum) هدایت می‌کند.
  - این ویژگی خصوصاً در مسائل پیچیده مانند شناسایی اشیاء بسیار مفید است.

پایداری در آموزش

- AdamWبه دلیل رفتار پایدار در مسائل با داده‌های پیچیده و حجیم مثل COCO وGTSDB ، گزینه‌ای محبوب است.
- این پایداری به دلیل ترکیب نرخ یادگیری تطبیقی و Weight Decay موثر است.

اکنون که تمامی موارد مورد نیاز برای پیاده سازی و آموزش مدل توضیح داده و انجام شده است تابع train را پیاده‌سازی کرده‌ایم.

```python
def train(self, train_loader, valid_loader, num_epochs=10):
    params = [p for p in self.model.parameters() if p.requires_grad]
    optimizer = torch.optim.AdamW(params, lr=0.0001, weight_decay=0.0005)
    lr_scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=3, gamma=0.1)
    best_map = 0
    history = {'train_loss': [], 'val_map': []}
    for epoch in range(num_epochs):
        self.model.train()
        epoch_loss = 0
        print(f"\nEpoch {epoch + 1}/{num_epochs}")
        print('-' * 50)
        for batch_idx, (images, targets) in enumerate(train_loader):
            images = [img.to(self.device) for img in images]
            targets = [{k: v.to(self.device) for k, v in t.items()} for t in targets]
            optimizer.zero_grad()
            loss_dict = self.model(images, targets)
            losses = sum(loss for loss in loss_dict.values())
            # Log individual loss components
            print(f"Batch {batch_idx + 1}/{len(train_loader)} - "
                  f"Loss: {losses.item():.4f} "
                  f"(Cls: {loss_dict['loss_classifier'].item():.4f}, "
                  f"Box: {loss_dict['loss_box_reg'].item():.4f}, "
                  f"RPN Obj: {loss_dict['loss_objectness'].item():.4f}, "
                  f"RPN Reg: {loss_dict['loss_rpn_box_reg'].item():.4f})")
            losses.backward()
            optimizer.step()
            epoch_loss += losses.item()
        # Validation
        val_map = self.evaluate(valid_loader)
        lr_scheduler.step()
        avg_train_loss = epoch_loss / len(train_loader)
        history['train_loss'].append(avg_train_loss)
        history['val_map'].append(val_map)

        print(f"\nEpoch {epoch+1}: Avg Train Loss={avg_train_loss:.4f}, Validation mAP={val_map:.4f}")

        if val_map > best_map:
            best_map = val_map
            # Save the model to Google Drive
            save_path = '/content/drive/MyDrive/traffic_sign_models/best_model.pth'
            torch.save(self.model.state_dict(), save_path)
            print(f"Model saved to {save_path} with mAP: {best_map}")

    # Plotting training loss and validation mAP
    self._plot_training_history(history)
    return history
```

شکل ۱۹- تابع **train** در مدل **Faster R-CNN**

از آنجایی که الزامی به توضیح تک تک بخش‌های کد نبوده است نیازی نیست تنها به توضیح دو خط از سه خط اول می‌پردازیم که اهمیت ویژه ای در ساختار کد ما دارد.

۱. **params = [p for p in self.model.parameters() if p.requires_grad]:**

- این خط پارامترهای مدلی که باید گرادیان آن‌ها محاسبه و به‌روز شود را فیلتر می‌کند.
- **هدف:**

- وقتی بخواهید برخی از لایه‌ها (مانند لایه‌های اولیه در یادگیری انتقالی) ثابت بمانند و به‌روزرسانی نشوند، می‌توانید با تنظیم requires_grad=False این کار را انجام دهید. این کد فقط پارامترهایی که باید به‌روزرسانی شوند را انتخاب می‌کند.

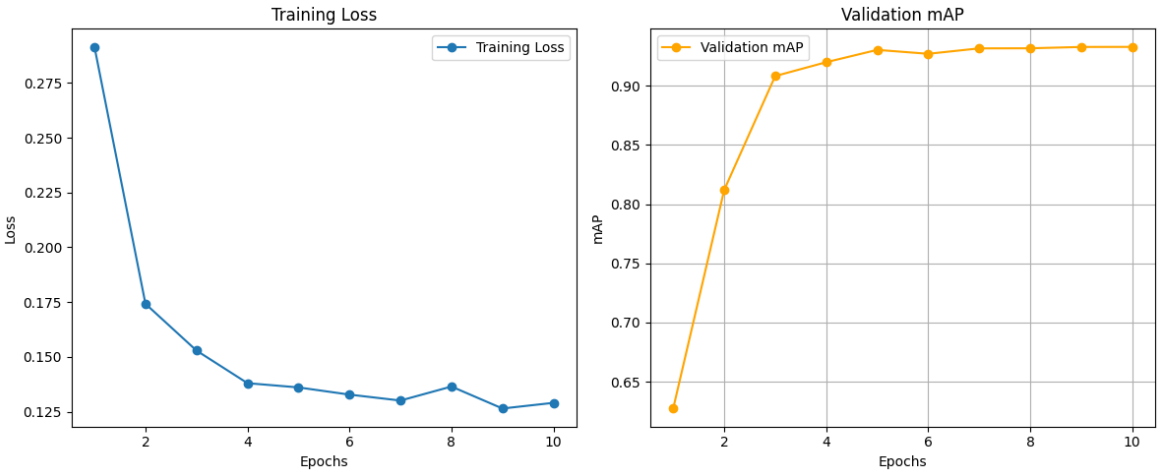
۲. **lr_scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=3, gamma=0.1):**

  - این خط یک برنامه‌ریز نرخ یادگیری از نوع StepLR را تعریف می‌کند.
  - **هدف:**

- نرخ یادگیری (Learning Rate) تأثیر زیادی در آموزش مدل دارد. این تابع هر چند دوره (مثلاً هر ۳ دوره) نرخ یادگیری را به میزان gamma(مثلاً ۰.۱) کاهش می‌دهد. این کاهش تدریجی باعث می‌شود که مدل در ابتدا سریع‌تر یاد بگیرد و در ادامه به آرامی به سمت همگرایی برود.

۲-۲-۶. ارزیابی مدل

اکنون تمام پارامترهای مورد نیاز مدل به صورت کامل تنظیم شده و مدل آموزش داده شده است. نتیجه آموزش و بررسی دقت آن بستگی به IoU تنظیم شده دارد. مدل را با استفاده از IoU = 0.5 در طول آموزش مورد ارزیابی قرار داده‌ایم. با این کار روند افزایش mAP با توجه به این threshold به خوبی قابل مشاهده است. برای این کار تابع evaluate نوشته شده است که بعد از هر ایپاک فراخوانی شده و کار ارزیابی با توجه به معیارهای تعیین شده را برای ما انجام می‌دهد. البته این مورد در صورت سوال خواسته نشده است و در این بخش به صورت اضافه انجام می‌گیرد. تصویر تابع evaluate در بخشی که مربوط به پیاده سازی mAP بود با عنوان " پیاده سازی تابع ارزیابی و mAP " آورده شده است. در این تابع ابتدا معیار mAP تعریف شده و سپس با توجه به label واقعی و پیشبینی شده توسط مدل و همینطور IoU تعریف شده که در اینجا برابر 0.5 است پیاده سازی شده است. اما نتیجه ارزیابی مدل در حین آموزش و در نهایت دقت هر کلاس و دقت نهایی در تصاویر زیر آورده شده است.



شکل ۲۰ - ارزیابی عملکرد مدل **Faster R-CNN** در حین آموزش با **IoU=0.5**

روند نزولی loss و صعودی mAP نشان از عملکرد صحیح و مناسب مدل است چرا که می‌دانیم مقدار loss در هر مرحله و با عملیات ریاضیتی داخل optimizer باید کاهش یابد و از طرفی مدل در هر گام باید بهتر از گام قبل عمل کند.

```
Evaluating for IoU threshold: 0.5
For IoU 0.5, mAP: 0.9329258799552917, mAP per class: [0.9723795056343079, 0.9512543678283691, 0.8841301202774048, 0.9239394664764404]
```

شکل ۲۱ - **mAP** هر کلاس

جدول ۱- جدول **mAP** با معیار **IoU = 0.5** برای هر کلاس و به صورت میانگین

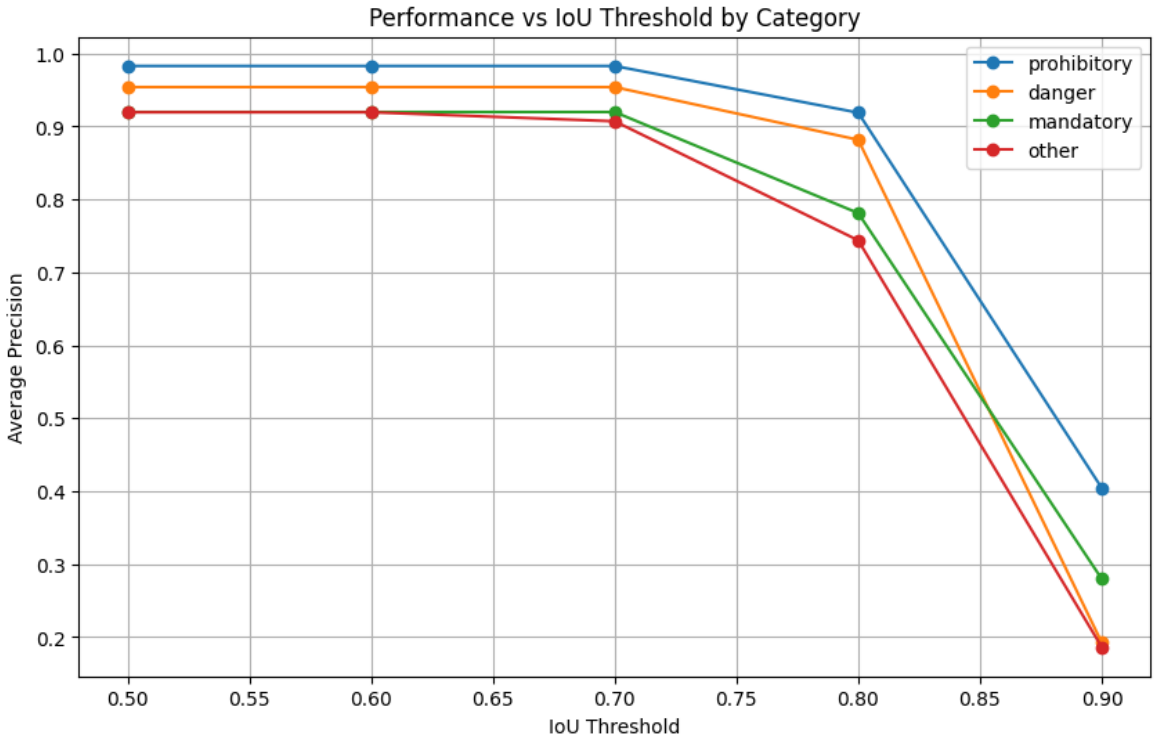
	mAP
Prohibitory	۹۷.۲۴
Danger	۹۵.۱۳
Mandatory	۸۸.۴۱
Other	۹۲.۳۹
میانگین	**۹۳.۲۹**

۲-۲-۷. ترسیم نمودار AP به ازای مقادیر متفاوت IoU

برای ارزیابی مدل با انواع IoU تابعی مجدد نوشته ایم تا با اختصاص مقادیر مختلف threshold بتوانیم ارزیابی لازم را انجام دهیم.

```python
def analyze_performance(self, test_loader, iou_thresholds=[0.5, 0.6, 0.7, 0.8, 0.9]):
    self.model.eval()
    results = []

    with torch.no_grad():
        for threshold in iou_thresholds:
            print(f"Evaluating for IoU threshold: {threshold}")
            # Remove num_classes argument
            metric = MeanAveragePrecision(iou_type="bbox", iou_thresholds=[threshold], class_metrics=True)

            for images, targets in test_loader:
                images = [img.to(self.device) for img in images]
                targets = [{k: v.to(self.device) for k, v in t.items()} for t in targets]
                outputs = self.model(images)

                # Adjust labels to start from 0 before passing to the metric
                for target in targets:
                    target['labels'] = target['labels'] - 1
                for output in outputs:
                    output['labels'] = output['labels'] - 1

                metric.update(outputs, targets)
            computed_metrics = metric.compute()

            # Convert map_per_class to a list
            map_per_class = computed_metrics['map_per_class'].tolist()
            mAP = computed_metrics['map'].item()

            print(f"For IoU {threshold}, mAP: {mAP}, mAP per class: {map_per_class}")

            results.append({
                'iou_threshold': threshold,
                'map': mAP,
                'map_per_class': map_per_class
            })

    return results
```

شکل ۲۲- تابع ارزیابی مدل با **threshold** های مختلف



شکل ۲۳ - نمودار مقادیر مختلف **mAP** برای هر کلاس با توجه به **threshold** انتخابی

این نمودار نشان‌دهنده رابطه بین میانگین دقت (mAP) و آستانه IoU برای دسته‌بندی‌های مختلف (prohibitory، danger، mandatory، otherاست. تحلیل نمودار را می‌توان به‌صورت زیر بسط داد:

همان‌طور که در نمودار مشاهده می‌کنیم، مقدار mAP با کاهش آستانه IoU افزایش می‌یابد. این موضوع منطقی است زیرا کاهش آستانه IoU به معنای کاهش سخت‌گیری در ارزیابی مدل است. در واقع، با کاهش مقدار IoU موردنیاز برای درستی پیش‌بینی، مدل می‌تواند تعداد بیشتری از باکس‌های پیش‌بینی‌شده را به‌عنوان True Positive در نظر بگیرد، که باعث افزایش میانگین دقت می‌شود.

در این نمودار، دقت برای تمامی کلاس‌ها در آستانه‌های پایین‌تر IoU بسیار بالا (بالای ۹۰ درصد) است، که نشان‌دهنده عملکرد کلی خوب مدل در تمامی دسته‌ها است. این نکته مثبت به ما می‌گوید که مدل به‌طور کلی توانایی بالایی در تشخیص صحیح دارد.

بااین‌حال، اختلافاتی میان کلاس‌ها دیده می‌شود:

- کلاس‌های **Mandatory**و **Other**دقت کمتری نسبت به **Prohibitory**و **Danger**دارند، به‌ویژه در آستانه‌های IoU بالا. این اختلاف می‌تواند به‌دلایل مختلفی باشد، از جمله تنوع کمتر داده‌های آموزشی برای این کلاس‌ها، شباهت بیشتر این کلاس‌ها به یکدیگر، یا چالش‌های بیشتری در تشخیص این نوع علائم.

- از سوی دیگر، کلاس **Prohibitory** عملکرد بسیار بهتری در تمامی مقادیر IoU دارد، که احتمالاً به‌دلیل تمایز واضح‌تر این کلاس‌ها از دیگر دسته‌ها یا تعداد بیشتر داده‌های مرتبط با این کلاس در مجموعه داده باشد.

این نمودار همچنین نشان می‌دهد که با افزایش آستانه IoU (مثلاً به ۰.۹)، دقت به شدت کاهش می‌یابد. این موضوع طبیعی است زیرا سخت‌گیری مدل در تشخیص درست افزایش می‌یابد و تنها پیش‌بینی‌هایی که به‌طور دقیق با باکس واقعی همپوشانی دارند به‌عنوان درست در نظر گرفته می‌شوند.

جمع‌بندی :نمودار نشان می‌دهد که مدل عملکرد بسیار خوبی دارد، اما توجه به اختلاف عملکرد میان کلاس‌ها می‌تواند راهنمایی برای بهبود مدل باشد. با افزایش تعداد داده‌ها یا تنوع آن‌ها برای کلاس‌های Mandatoryو Other ، می‌توان انتظار داشت که دقت این دسته‌ها نیز بهبود یابد.

جدول ۲ - ارزیابی مدل **Faster R-CNN** با **threshold** های متفاوت

	mAP				
	IoU = 0.9	IoU = 0.8	IoU = 0.7	IoU = 0.6	IoU = 0.5
Prohibitory	۳۰.۵۳	۸۹.۲۹	۹۷.۲۴	۹۷.۲۴	۹۷.۲۴
Danger	۲۶.۶۷	۷۷.۸۹	۹۳.۰۹	۹۵.۱۲	۹۵.۱۲
Mandatory	۲۴.۷۲	۸۲.۶۶	۹۱.۹۹	۸۸.۴۱	۸۸.۴۱
Other	۱۴.۲۰	۸۶.۰۷	۹۲.۳۹	۹۲.۳۹	۹۲.۳۹
میانگین	**۲۴.۰۳**	**۸۳.۹۸**	**۹۲.۷۸**	**۹۳.۲۹**	**۹۳.۲۳**

۲-۲-۸. ارزیابی مدل بر اساس اندازه

برای ارزیابی مدل بر اساس اندازه کافی است همان معیار mAP را در نظر داشته باشیم و با توجه به سایز تصاویر، mAP محاسبه شده برای تصاویر در سایز های مختلف را محاسبه کرده و میانگین گیری کنیم. به عبارت دیگر تصاویری که در هر کدام از سایزهای small, medium و large هستند را به صورت جداگانه و یک کلاس در نظر می‌گیریم و عملکرد و دقت مدل را در ارزیابی تصاویر با آن سایز محاسبه می‌کنیم.

برای اینکار تابعی با عنوان evaluate_by_size نوشته شده است.

```python
@torch.no_grad()
def evaluate_by_size(self, data_loader, size_thresholds=(32, 45)):
    """
    Evaluates the model's performance (mAP) for different object sizes.

    Parameters:
    - data_loader: DataLoader for the dataset to evaluate.
    - size_thresholds: Tuple of thresholds to categorize object sizes (small, medium, large).

    Returns:
    - size_map: Dictionary containing mAP for 'small', 'medium', and 'large' objects.
    """
    self.model.eval()
    size_metrics = {
        'small': MeanAveragePrecision(iou_type="bbox",iou_thresholds=[0.5], class_metrics=True),
        'medium': MeanAveragePrecision(iou_type="bbox",iou_thresholds=[0.5], class_metrics=True),
        'large': MeanAveragePrecision(iou_type="bbox",iou_thresholds=[0.5], class_metrics=True)
    }

    for images, targets in data_loader:
        images = [img.to(self.device) for img in images]
        targets = [{k: v.to(self.device) for k, v in t.items()} for t in targets]
        outputs = self.model(images)

        for target, output in zip(targets, outputs):
            # Adjust labels to start from 0
            target['labels'] = target['labels'] - 1
            output['labels'] = output['labels'] - 1

            # Get the size categories for each ground truth box
            size_categories = []
            for box in target['boxes']:
                width = box[2] - box[0]
                category = categorize_object_size(width, size_thresholds)
                size_categories.append(category)
```

شکل ۲۴- پیاده‌سازی تابع **evaluate by size** بخش اول

پارامترها :

۱. **data_loader**:

- این ورودی شامل داده‌هایی است که برای ارزیابی مدل استفاده می‌شود.
- به طور کلی، data_loaderیک مجموعه از تصاویر و باکس‌های محدودکننده مربوط به اشیاء (Ground Truth) است.

۲. **size_thresholds**:

- یک تاپل که آستانه‌هایی را برای طبقه‌بندی اندازه اشیاء به کوچک، متوسط و بزرگ مشخص می‌کند.
- مثلاً size_thresholds=(32, 45)به این معنی است که:
  - اشیاء با عرض کمتر از ۳۲ پیکسل کوچک هستند.

- اشیاء با عرض بین ۳۲ و ۴۵ پیکسل متوسط هستند.
- اشیاء با عرض بزرگ‌تر از ۴۵ پیکسل بزرگ محسوب می‌شوند.

ساختار کد:

۱.فعال‌سازی حالت ارزیابی مدل:

- مدل به حالت evalمی‌رود. در این حالت، لایه‌هایی مثل Dropout یا BatchNorm به‌صورت ثابت عمل می‌کنند و داده‌های جدیدی یاد نمی‌گیرند.

۲.تعریف دیکشنری برای نگه‌داری متریک‌ها:

- سه متریک MeanAveragePrecisionبرای اندازه‌های کوچک، متوسط و بزرگ تعریف شده‌اند.
- "iou_type="bboxنشان می‌دهد که ارزیابی برای bounding box انجام می‌شود.
- class_metrics=Trueبه این معنی است که عملکرد برای هر کلاس جداگانه نیز ثبت می‌شود.
- IoU threshold = 0.5 با توجه به صورت سوال.

۳.پردازش داده‌ها در حلقه:

در این حلقه، هر بار تعدادی تصویر و Ground Truth (targets) از data_loaderگرفته می‌شود.

انتقال داده‌ها به:GPU/CPU

- تصاویر وGround Truth ها به دستگاهی که مدل روی آن قرار دارد CPU یا GPU منتقل می‌شوند.

۴. پیش‌بینی مدل:

- مدلbounding box ها و کلاس‌های پیش‌بینی‌شده را برای تصاویر ورودی برمی‌گرداند.

۵. تنظیم برچسب‌ها:

- برچسب‌ها یک عدد کم می‌شوند تا از ۰ شروع شوند. این معمولاً به دلیل تفاوت فرمت داده‌های Ground Truthو مدل است.

۶. دسته‌بندی اندازه اشیاء:

- عرض هر باکس محاسبه می‌شود width = box[2] - box[0] :اختلاف بین مختصات x گوشه‌های چپ و راست.
- اندازه باکس بر اساس size_thresholdsبه یکی از دسته‌های small, medium, largeاختصاص داده می‌شود.

۷. جداسازی داده‌ها بر اساس اندازه:

داده‌های Ground Truth (باکس‌ها و برچسب‌ها) بر اساس اندازه جدا می‌شوند. این کار با استفاده از ایندکس‌هایی که مربوط به اندازه خاص هستند، انجام می‌شود.

۸. فیلتر کردن پیش‌بینی‌ها:

ابتدا IoU (Intersection over Union) بین باکس‌های پیش‌بینی‌شده و Ground Truth محاسبه می‌شود.

- پیش‌بینی‌هایی که IoU آن‌ها بیشتر از ۰ است، به عنوان پیش‌بینی معتبر انتخاب می‌شوند.

۹. به‌روزرسانی متریک:

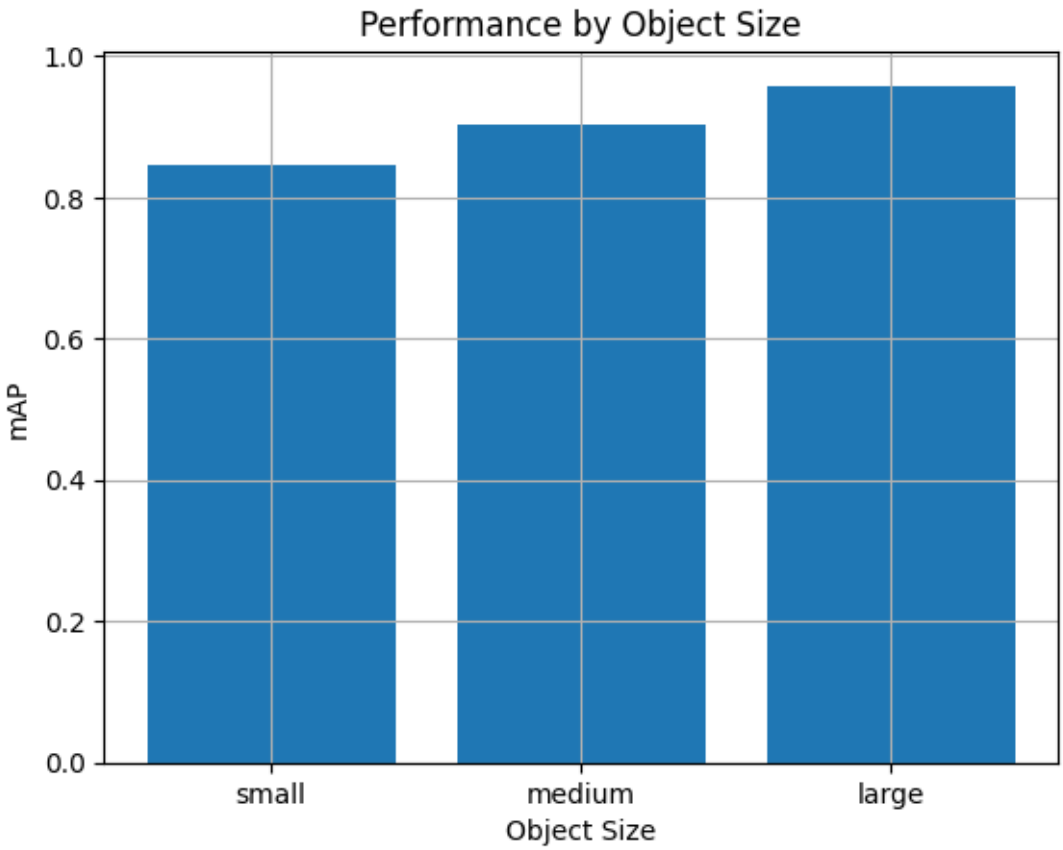
داده‌های مربوط به پیش‌بینی‌ها و Ground Truth برای اندازه خاص به متریک مربوطه اضافه می‌شوند.

۱۰. محاسبه mAP نهایی:

در انتها، mAPبرای هر اندازه محاسبه شده و به دیکشنری size_mapاضافه می‌شود.

نتیجه خروجی:

- این تابع یک دیکشنری به نام size_mapبرمی‌گرداند که شامل mAP برای سه اندازه small, medium, largeاست.



شکل ۲۵ - ارزیابی مدل **Faster R-CNN** بر اساس سایز

جدول ۳- ارزیابی مدل **Faster R-CNN** بر اساس سایز

mAP for IoU = 0.5			
Small	Medium	Large	
۸۴.۶۶	**۹۰.۲۵**	**۹۵.۸۱**	mAP for IoU = 0.5

این نمودار نشان‌دهنده عملکرد مدل در تشخیص اشیاء با اندازه‌های مختلف است. محور افقی سه دسته اندازه اشیاء small ، medium، large را نشان می‌دهد و محور عمودی مقدار mAP (میانگین دقت) را برای هر دسته اندازه مشخص می‌کند. تحلیل دقیق نمودار به شرح زیر است:

مشاهده کلی:

۱. **اشیاء کوچک:(small)**

- مقدار mAP برای این دسته کمترین مقدار را دارد و در حدود **0.8**است.
- این موضوع نشان می‌دهد که مدل در تشخیص اشیاء کوچک نسبت به دسته‌های دیگر عملکرد ضعیف‌تری دارد.

۲. **اشیاء متوسط:(medium)**

- مقدار mAP در این دسته به حدود **0.9** افزایش یافته است.
- این نشان می‌دهد که مدل در تشخیص اشیاء متوسط عملکرد بهتری نسبت به اشیاء کوچک دارد.

۳. **اشیاء بزرگ:(large)**

- مقدار mAP در این دسته تقریباً به **1.0** می‌رسد.
- این مقدار بالاترین دقت را نشان می‌دهد و بیانگر این است که مدل در شناسایی اشیاء بزرگ عملکرد بسیار خوبی دارد.

دلایل عملکرد متفاوت در اندازه‌ها:

۱. **عملکرد پایین در اشیاء کوچک:**

- **مشکل :**اشیاء کوچک معمولاً در تصاویر وضوح کمتری دارند و ممکن است به دلیل اندازه کوچک‌تر، باکس‌های پیش‌بینی‌شده توسط مدل نتوانند به‌خوبی با Ground Truth همپوشانی داشته باشند.
- **راه‌حل :**افزایش داده‌های آموزشی شامل اشیاء کوچک یا استفاده از مدل‌هایی با رزولوشن ورودی بالاتر می‌تواند عملکرد را بهبود دهد.

۲. **عملکرد متوسط در اشیاء متوسط:**

- مدل توانسته است اشیاء متوسط را با دقت خوبی شناسایی کند. این دسته معمولاً چالش خاصی ندارد، زیرا اندازه اشیاء نه آن‌قدر کوچک است که وضوح کم باشد و نه آن‌قدر بزرگ که با محدودیت‌های دیگر مواجه شویم.

۳. **عملکرد بالا در اشیاء بزرگ:**

- اشیاء بزرگ معمولاً در تصاویر وضوح بالایی دارند و پیش‌بینی باکس‌های محدودکننده برای این اشیاء آسان‌تر است.
- همچنین، همپوشانی (IoU) بین باکس‌های پیش‌بینی‌شده و Ground Truth برای اشیاء بزرگ بالاتر است، زیرا اختلاف مختصاتی کمتر تأثیرگذار است.

تحلیل کلی و نتیجه‌گیری:

- مدل در دسته‌بندی اشیاء با اندازه‌های بزرگ عملکرد فوق‌العاده‌ای دارد.
- با این حال، عملکرد مدل در تشخیص اشیاء کوچک قابل بهبود است. این مسئله ممکن است به دلیل چالش‌های مربوط به وضوح کم، داده‌های ناکافی یا طراحی معماری مدل باشد.
- عملکرد در اشیاء متوسط متعادل است و مدل در این دسته نیاز به تغییرات زیادی ندارد.

۱. **استفاده از معماری مناسب:**

- مدل‌هایی که به‌طور خاص برای شناسایی اشیاء کوچک طراحی شده‌اند (مانند مدل‌هایی با سطوح رزولوشن چندگانه) می‌توانند عملکرد را بهبود دهند.

۲. **تنظیم دقیق آستانه IoU:**

- کاهش آستانه IoU می‌تواند باعث شود مدل پیش‌بینی‌های بیشتری را به عنوان True Positive در نظر بگیرد، که به‌ویژه برای اشیاء کوچک مؤثر است.

۲-۲-۹. نمایش نمونه تصویر

با اتمام آموزش و با توجه به ارزیابی مدل با IoU = 0.5 حال می‌توانیم هر عکسی از دیتاست را به مدل داده و پیشبینی مدل و واقعیت را مقایسه کنیم. برای این کار تابعی نوشته شده است که یک عکس از

دیتاست به همراه مدل annotation ها و device اجرا که اینجا همان GPU هست را دریافت کرده و روی عکس مشخص می‌کند آنچه در gt هست و آنچه مدل پیشبینی کرده است.

```
# Visualize a sample prediction
print("\nVisualizing sample predictions...")
image_path = os.path.join(image_dir, test_names[20])
visualize_results_with_ground_truth(
    model=detector.model,
    image_path=image_path,
    annotations=test_dataset.annotations,
    device=detector.device
)

print("\nAll tasks completed!")
```

شکل ۲۶ - اجرای تابع نمایش تصاویر و مقایسه پیشبینی مدل و واقعیت برای یک نمونه تصویر

```python
def visualize_results_with_ground_truth(model, image_path, annotations, device, conf_threshold=0.5):
    """
    Visualize both model predictions and ground truth annotations
    """
    # Define category mapping
    category_map = {
        1: 'prohibitory',
        2: 'danger',
        3: 'mandatory',
        4: 'other'
    }
    # Define colors for different purposes (BGR format for OpenCV)
    colors = {
        'prediction': (255, 0, 0),    # Red for predictions
        'ground_truth': (0, 255, 0),  # Green for ground truth
        'match': (0, 0, 255)          # Blue for matching predictions
    }
    # Load and prepare image
    image = Image.open(image_path).convert("RGB")

    # Create a dummy target
    dummy_target = {
        'boxes': torch.zeros((0, 4), dtype=torch.float32),
        'labels': torch.zeros(0, dtype=torch.int64),
        'image_id': torch.tensor([0])
    }

    # Apply transforms using the custom Compose
    transform = get_transforms(train=False)
    image_tensor, _ = transform(image, dummy_target)

    # Get model predictions
    model.eval()
    with torch.no_grad():
        prediction = model([image_tensor.to(device)])[0]

    # Convert image to numpy for drawing
    image_np = np.array(image)

    # Get ground truth annotations for this image
    image_name = os.path.basename(image_path)
    gt_boxes = annotations.get(image_name, [])

    # Draw ground truth boxes first
    for gt in gt_boxes:
        x1, y1, x2, y2 = gt['x1'], gt['y1'], gt['x2'], gt['y2']
        category = get_category(gt['class_id'])  # You'll need to implement this based on your mapping
```

شکل ۲۷- تابع نمایش تصویر و مقایسه بخش اول

```
    # Draw ground truth box in green
    cv2.rectangle(image_np, (x1, y1), (x2, y2), colors['ground_truth'], 2)

    # Add ground truth label
    gt_label = f"GT: {category}"
    cv2.putText(image_np, gt_label, (x1, y1-10), cv2.FONT_HERSHEY_SIMPLEX,
                0.5, colors['ground_truth'], 2)

# Draw prediction boxes
for box, score, label in zip(prediction['boxes'], prediction['scores'], prediction['labels']):
    if score > conf_threshold:
        box = box.cpu().numpy()
        x1, y1, x2, y2 = map(int, box)
        predicted_category = category_map.get(label.item(), 'unknown')

        # Check if this prediction matches any ground truth box
        is_match = False
        for gt in gt_boxes:
            iou = calculate_iou(box, [gt['x1'], gt['y1'], gt['x2'], gt['y2']])
            if iou > 0.5:  # IOU threshold
                is_match = True
                break
        # Choose color based on whether it's a match
        color = colors['match'] if is_match else colors['prediction']
        # Draw prediction box
        cv2.rectangle(image_np, (x1, y1), (x2, y2), color, 2)
        # Add prediction label with confidence score
        pred_label = f"Pred: {predicted_category} ({score:.2f})"
        cv2.putText(image_np, pred_label, (x1, y2+20), cv2.FONT_HERSHEY_SIMPLEX,
                    0.5, color, 2)
# Add legend
legend_y = 30
for key, color in colors.items():
    cv2.putText(image_np, key, (10, legend_y), cv2.FONT_HERSHEY_SIMPLEX,
                0.5, color, 2)
    legend_y += 20
# Display the image
plt.figure(figsize=(12, 8))
plt.imshow(cv2.cvtColor(image_np, cv2.COLOR_BGR2RGB))
plt.axis('off')
plt.title(f"Traffic Sign Detection Results: {image_name}")
plt.show()
```

شکل ۲۸ - تابع نمایش تصویر و مقایسه بخش دوم

شکل ۲۹ - نمونه تصویر مقایسه مدل **Faster R-CNN** با **ground truth**

در تصویر فوق همپوشانی ۹۳ درصدی پیشبینی مدل با bounding box موجود در ground truth و همینطور پیشبینی صحیح کلاس را مشاهده می‌کنیم.

۲-۳.تنظیم دقیق و ارزیابی مدل تشخیص شی تک مرحله ای

روش پیاده‌سازی چندین پارامتر و بخش در این مدل مشابه مدل قبل است، بنابراین به جهت پرهیز از تکرار، در بخش‌های مشابه به بخش قبلی ارجاع داده شده است.

۲-۳-۱. تعریف شبکه SSD300 و VGG16 به عنوان Backbone

SSD (Single Shot MultiBox Detector)یک مدل شناسایی اشیا است که به دلیل سرعت و دقت مناسب برای کاربردهای بلادرنگ بسیار محبوب است SSD 300 .به نسخه‌ای از این مدل اشاره دارد که ورودی تصاویر آن با اندازه‌ی ۳۰۰ در ۳۰۰ پیکسل ثابت شده است. این انتخاب باعث کاهش هزینه محاسباتی می‌شود در حالی که دقت کافی برای شناسایی اشیا حفظ می‌شود.

Backbone: VGG16

VGG16به‌عنوان یک شبکه عصبی از پیش‌آموزش‌دیده عمل می‌کند که نقش استخراج ویژگی‌ها را در SSD برعهده دارد. این شبکه به دلیل معماری عمیق و توانایی در استخراج ویژگی‌های قوی، در بسیاری

از مسائل بینایی ماشین استفاده می‌شود. درSSD ، لایه‌های fully connected این شبکه حذف می‌شود و لایه‌های اضافی برای شناسایی چندمقیاسی اضافه می‌شوند.

معماری و روند عملکرد

۱. **پردازش تصویر ورودی**:

- ورودی مدل یک تصویر ۳۰۰ RGB 300xاست.
- این اندازه ثابت باعث ساده‌سازی فرآیند آموزش و کاهش نیازهای محاسباتی می‌شود.

۲. **استخراج ویژگی‌ها**:

- شبکه VGG16 تصویر ورودی را پردازش کرده و نقشه‌های ویژگی مختلفی را استخراج می‌کند.
- از برخی از این نقشه‌های ویژگی در لایه‌های میانی برای شناسایی چندمقیاسی استفاده می‌شود.

۳. **لایه‌های اضافی**:

- پس از لایه‌هایVGG16 ، چندین لایه کانولوشن افزوده می‌شود که ابعاد نقشه‌های ویژگی را به تدریج کاهش می‌دهند.
- این لایه‌ها یک هرم ویژگی ایجاد می‌کنند که مدل را قادر به شناسایی اشیا با اندازه‌های مختلف می‌سازد.

۴. **جعبه‌های پیش‌فرض(Default Boxes)**:

- SSDاز جعبه‌های پیش‌فرض با نسبت‌ها و اندازه‌های مختلف در هر سلول نقشه‌های ویژگی استفاده می‌کند.
- این جعبه‌ها در زمان آموزش با جعبه‌های مرجع (Ground Truth) تطبیق داده می‌شوند.

۵. **کلاسه‌بندی و مکان‌یابی**:

- مدل برای هر جعبه پیش‌فرض دو پیش‌بینی انجام می‌دهد:
  - احتمال تعلق به کلاس‌های مختلف.

- مختصات جعبه برای تنظیم دقیق موقعیت آن.

- این پیش‌بینی‌ها با استفاده از هسته‌های کوچک کانولوشن) مثل ۳ (3xانجام می‌شوند.

۶. **تابع هزینه:(Loss Function)**

- تابع هزینه شامل دو بخش است:
  - **هزینه کلاسه‌بندی** :بر اساس.Cross-Entropy
  - **هزینه مکان‌یابی** :بر اساس.Smooth L1 Loss

۷. **حذف حداکثرهای غیرضروری:(NMS)**

- پس از پیش‌بینی، برای حذف تشخیص‌های اضافی که همپوشانی زیادی دارند، از NMS استفاده می‌شود و جعبه‌ای با بالاترین امتیاز نگه داشته می‌شود.

مزایایSSD

- **سرعت بالا** :استفاده از روش تک‌مرحله‌ای باعث حذف شبکه‌های پیشنهاد منطقه و در نتیجه کاهش زمان پردازش می‌شود.
- **تشخیص چندمقیاسی** :استفاده از نقشه‌های ویژگی در چندین مقیاس توانایی مدل در شناسایی اشیا با اندازه‌های مختلف را افزایش می‌دهد.
- **معماری فشرده** :اندازه ورودی کوچک (300x300) نیازهای محاسباتی را کاهش داده و SSD را برای دستگاه‌های کم‌منابع مناسب می‌کند.

نقش VGG16 درSSD

- **ویژگی‌های از پیش‌آموزش‌دیده** VGG16 :که روی دیتاست‌هایی مثل ImageNet آموزش داده شده، ویژگی‌های اولیه‌ی قوی را برای مدل فراهم می‌کند.
- **ویژگی‌های سلسله‌مراتبی** :لایه‌های عمیق VGG16 الگوهای پیچیده را استخراج می‌کنند که برای تشخیص دقیق اشیا حیاتی است.
- **حذف و تطبیق** :با حذف لایه‌های fully connected و افزودن لایه‌های جدید، VGG16برای شناسایی تراکم بالا (dense prediction) بهینه می‌شود.

خلاصه روند کار

۱. دریافت یک تصویر ورودی ۳۰۰ × ۳۰۰

۲. پردازش تصویر توسط VGG16 برای استخراج ویژگی‌ها.

۳. اعمال لایه‌های کانولوشن اضافی برای تشخیص چندمقیاسی.

۴. تولید پیش‌بینی‌ها برای احتمال کلاسه‌ها و مختصات جعبه.

این معماری در کاربردهایی مثل رانندگی خودکار، نظارت، و استفاده در دستگاه‌های همراه که نیازمند شناسایی اشیا به‌صورت بلادرنگ هستند، استفاده می‌شود.

۲-۳-۲. آماده‌سازی اولیه برای تنظیم دقیق

```
def _create_model(self, num_classes):
    model = ssd300_vgg16(weights=SSD300_VGG16_Weights.COCO_V1)
    in_channels = [512, 1024, 512, 256, 256, 256]
    num_anchors = [4, 6, 6, 6, 4, 4]
    model.head = SSDHead(in_channels, num_anchors, num_classes)
    return model
```

شکل ۳۰- آماده‌سازی مدل برای مدل **SSD**

مدل SSD300 با معماری VGG16 و وزن‌های از پیش آموزش‌دیده شده روی مجموعه داده COCO بارگذاری شده است. روند کلی مشابه بخش قبلی است است.

هد اصلی مدل با SSDHeadجدیدی جایگزین شده است که تعداد کانال‌های ورودی(in_channels)، تعداد انکرها (num_anchors) و تعداد کلاس‌های دلخواه را می‌پذیرد.

دلیل :این تنظیمات به ما امکان می‌دهد مدل را برای داده‌های جدید (دیتاست سفارشی) بازآموزی کنیم و تنها پارامترهای مربوط به دسته‌بندی‌ها و تشخیص روی داده‌های ما تغییر کنند.

in_channels.۱ :

- این لیست نشان‌دهنده تعداد **کانال‌های ویژگی** است که از لایه‌های مختلف شبکه استخراج می‌شود.

توضیح:

- درSSD ، **لایه‌های خاصی از شبکه عصبی) مانند (VGG16** برای استخراج ویژگی‌های مکانی و مفهومی اشیاء استفاده می‌شوند.
- این لایه‌ها معمولاً لایه‌های آخر شبکه هستند که ویژگی‌های **انتزاعی‌تر و بزرگ‌تر** از تصویر ارائه می‌دهند.
- هر لایه خروجی (feature map) تعدادی کانال دارد که این کانال‌ها نشان‌دهنده عمق اطلاعات استخراج‌شده از تصویر هستند.
- به عنوان مثال:
  - **512کانال :**به این معنی است که در لایه‌ای خاص، ۵۱۲ ویژگی مختلف (فیلتر یا کرنل) از تصویر استخراج شده است.
  - این اعداد (۵۱۲، ۱۰۲۴، ...) مستقیماً از معماری مدل اصلی (VGG16) به دست آمده‌اند.

چرا این اعداد انتخاب شده‌اند؟

- این اعداد از ساختار معماری **VGG16**می‌آیند:
  - 512و 1024کانال‌ها از لایه‌های آخر VGG16 هستند.
  - لایه‌های اضافی با ابعاد کوچک‌تر (۲۵۶ کانال) به مدل اضافه می‌شوند تا از مقیاس‌های مختلف تصویر استفاده شود. این لایه‌ها در اصل بخشی از معماری SSD هستند.

num_anchors.۲ :

- این لیست تعداد **انکر باکس‌ها** را در هر لایه ویژگی نشان می‌دهد.

توضیح:

- **انکر باکس‌ها:(Anchor Boxes)**
  - درSSD ، برای هر سلول درfeature map ، چندین باکس پیش‌فرض(default boxesیا anchor boxes) با نسبت ابعاد (aspect ratios) مختلف تعریف می‌شود.
  - این باکس‌ها کمک می‌کنند که مدل اشیاء با اندازه‌ها و نسبت‌های مختلف را تشخیص دهد.

- **چرا تعداد انکرها متفاوت است؟**
  - هر لایه ویژگی (feature map) اندازه متفاوتی دارد و برای پوشش ابعاد مختلف اشیاء، تعداد متفاوتی انکر تعریف شده است.
  - به عنوان مثال:
    - لایه‌ای که ویژگی‌های کوچک‌تر را استخراج می‌کند، معمولاً نیاز به تعداد بیشتری انکر دارد (مثل ۶ انکر برای پوشش ابعاد مختلف).
    - در لایه‌هایی که مقیاس بزرگ‌تری دارند، تعداد انکرها کمتر است، زیرا اشیاء بزرگ‌تر و ساده‌تری را پوشش می‌دهند.

مقادیر:

- **4انکر :**برای لایه‌هایی که اشیاء کوچک‌تر را پوشش می‌دهند.
- **6انکر :**برای لایه‌هایی که اشیاء بزرگ‌تر یا پیچیده‌تر را پوشش می‌دهند.

چگونه این اعداد تعیین می‌شوند؟

- این اعداد از طراحی معماری SSD و استاندارد COCO یا Pascal VOC برای تشخیص اشیاء می‌آیند. درSSD300 ، مقادیر استاندارد 4و 6برای انکرها استفاده می‌شوند.
- هر انکر یک جعبه با نسبت ابعاد (aspect ratio) متفاوت دارد (مثل ۱:۱، ۱:۲، ۲:۱ و غیره).

جمع‌بندی:

۱. **in_channels:**
   - این اعداد از معماری **VGG16**می‌آیند و تعداد کانال‌های لایه‌های مختلف شبکه را نشان می‌دهند که برای استخراج ویژگی استفاده می‌شوند.

۲. **num_anchors:**
   - تعداد انکر باکس‌ها در هر لایه feature map است. این تعداد برای پوشش نسبت ابعاد مختلف اشیاء در مقیاس‌های متفاوت انتخاب شده و از طراحی استاندارد SSD300 تبعیت می‌کند.

چرا این دو بخش مهم‌اند؟

- این مقادیر به طور مستقیم به **اندازه و مقیاس اشیاء** در تصاویر ورودی مربوط می‌شوند.
- اگر این مقادیر درست تنظیم شوند، مدل می‌تواند **اشیاء با اندازه‌های مختلف** را بهتر شناسایی کند.

۲-۳-۳. عملیات آماده‌سازی و نرمال کردن

نکته حائز اهمیت و متفاوت در این مدل سایز تصاویر است. تصاویر در این مدل نهایتا همگی ۳۰۰ در ۳۰۰ خواهند شد. می‌توان پیش از شروع train و به صورت خارجی عکس‌ها را Resize کرد و یا تصاویر را با ابعاد اولیه و واقعی به مدل بدهیم و عکس‌ها در داخل مدل و به صورت داخلی و اتوماتیک ریسایز شوند. که ما گزینه دوم را انتخاب کردیم چرا که ریسایز کردن عکس‌ها قبل از آموزش باعث می‌شود در زمان ارزیابی با سایز نیاز باشد مجدد آنها را به سایزهای واقعی برگردانیم تا امکان ارزیابی به صورت صحیح مهیا باشد. همچنین با انجام چند بار تست راه دوم دقت بالاتری را نتیجه داد.

ابتدا مدل بدون داده افزایی خاصی مدل شد که نتیجه چندان قابل قبول نبود به همین دلیل اقدام به استفاده از augmentation های متفاوت کردیم. به عنوان مثال تغییر در نور و saturation انجام شد اما نتیجه به اندازه کافی خوب نبود. در نهایت با تکرار چندید حالت مختلف از augmentation (به صورت داینامیک) به این نتیجه رسیدیم که با horizontal flip به بهترین نتیجه می‌رسیم. در بخش ارزیابی نتیجه این اعمال داده افزایی نمایش داده خواهد شد. در بخش horizontal flip به جا به جایی bounding box ها هم توجه شده است به این معنا که عکسی که به صورت افقی flip می‌شود طبیعتا محل bounding box هم برای آن باید تغییر کند و متناظر با عکس در حالت جدید تنظیم شود.

```python
class RandomHorizontalFlip(object):
    def __init__(self, p=0.5):
        self.p = p

    def __call__(self, image, target):
        if random.random() < self.p:
            # Flip image
            image = F.hflip(image)

            # Flip bounding boxes
            if 'boxes' in target:
                w, _ = image.size
                boxes = target['boxes']
                boxes[:, [0, 2]] = w - boxes[:, [2, 0]]
                target['boxes'] = boxes

        return image, target

class ToTensor(object):
    def __call__(self, image, target):
        image = F.to_tensor(image)
        return image, target

class Normalize(object):
    def __init__(self, mean, std):
        self.mean = mean
        self.std = std

    def __call__(self, image, target):
        image = F.normalize(image, mean=self.mean, std=self.std)
        return image, target

def get_transforms(train):
    transforms = []
    if train:
        transforms.append(RandomHorizontalFlip(0.5))
    transforms.append(ToTensor())
    transforms.append(Normalize(mean=[0.485, 0.456, 0.406],
                                std=[0.229, 0.224, 0.225]))
    return transforms
```

شکل ۳۱ - نرمال‌سازی و آماده سازی تصاویر برای آموزش

در ادامه کلاسی به نام GTSDBDataset داریم که بسیار مشابه بخش قبلی است و در ساختار هیچ تفاوتی وجود ندارد و نهایتا دیتاست مورد نیاز را آماده می‌کند. استفاده از DataLoader و باقی موارد این قسمت هم دقیقا مشابه بخش قبلی است.

۲-۳-۴. معیارهای IoU و mAP

تعریف و دلیل استفاده از این معیار در بخش قبلی آورده شده است. نحوه پیاده‌سازی همچنین دقیقا مشابه بخش قبل است.

۲-۳-۵. بهینه‌ساز و تابع هزینه

تابع هزینه

برای تابع هزینه یا loss از آنچه در درون خود SSD300 بوده استفاده کرده ایم که به شکل زیر پیاده سازی شده است:

خطا شامل دو بخش است:

- **خطای کلاسه‌بندی** :خطای Cross-Entropy برای پیش‌بینی چندکلاسه.
- **خطای مکان‌یابی** :خطای Smooth L1 برای تنظیم موقعیت جعبه.

دلیل استفاده از این توابع برای تابع هزینه دقیقا مشابه بخش قبلی است. به همین جهت برای مطالعه علت استفاده از این توابع به بخش قبلی با همین عنوان مراجعه بفرمایید.

بهینه‌ساز

برای بهینه‌ساز از **Adam**استفاده شده است. همچنین از **lr_scheduler**نیز بهره گرفته شده تا امکان کاهش و تنظیم نرخ یادگیری بهینه‌تر فراهم شود و عملکرد مدل در طول آموزش بهبود یابد.

ویژگی‌های Adam :

۱. **ترکیب نرخ یادگیری تطبیقی(Adaptive Learning Rate) :**

- Adamاز نرخ یادگیری تطبیقی استفاده می‌کند، به این معنا که برای هر پارامتر نرخ یادگیری جداگانه و بهینه در نظر گرفته می‌شود.
- این ویژگی باعث می‌شود مدل سریع‌تر و با دقت بالاتر به حداقل سراسری (Global Minimum)برسد.

۲. **نرخ یادگیری پایین(0.0001) :**

- نرخ یادگیری اولیه برابر با ۰.۰۰۰۱ تنظیم شده است تا مدل پیش‌آموزش‌دیده به طور دقیق تنظیم (Fine-Tuned) شود.

- این مقدار به مدل کمک می‌کند که پارامترهای خود را به تدریج و بدون تغییرات ناگهانی بهبود دهد.

۳. **وزن‌زدایی(Weight Decay)** :

- مقدار Weight Decay برابر با ۰.۰۰۰۵ در نظر گرفته شده است.
- این ویژگی باعث جلوگیری از بیش‌برازش (Overfitting) می‌شود، به‌ویژه در مدل‌هایی مانند SSD که تعداد زیادی پارامتر دارند.

دلایل انتخاب Adam به عنوان بهینه‌ساز:

۱.Adaptive Learning Rate

- **چرا انتخاب شده؟**
  - Adam نرخ یادگیری تطبیقی دارد، که به مدل کمک می‌کند برای هر پارامتر نرخ یادگیری بهینه تعیین کند.
  - این ویژگی به‌خصوص در مدل‌های پیچیده مثل SSD که دارای تعداد زیادی پارامتر هستند، بسیار مفید است.
- **مزیت :**بهبود سرعت و دقت همگرایی.(Convergence)

۲. وزن‌زدایی(Weight Decay)

- **چرا مهم است؟**
  - Weight Decayبه‌عنوان یک روش مؤثر برای جلوگیری از بیش‌برازش عمل می‌کند.
  - درAdam ، این ویژگی به‌طور مستقیم بر روی به‌روزرسانی پارامترها تأثیر می‌گذارد و شبکه را بهینه‌تر تنظیم می‌کند.
- **مزیت :**جلوگیری از بیش‌برازش و بهبود تعمیم‌دهی (Generalization) مدل.

۳.ترکیب مومنتوم(Momentum)

- **چرا مفید است؟**

- مومنتوم به مدل کمک می‌کند که از کمینه‌های محلی (Local Minima) عبور کرده و به کمینه سراسری برسد.
- این ویژگی در مسائل پیچیده مانند شناسایی اشیاء، که ممکن است تابع هدف دارای چندین کمینه محلی باشد، بسیار ارزشمند است.

- **مزیت :**افزایش پایداری در آموزش و همگرایی بهتر.

۴. پایداری در آموزش

- **چرا انتخاب شده؟**
  - Adamدر مسائل با داده‌های پیچیده و حجیم) مثل مجموعه داده‌های COCO و GTSDBرفتار پایداری دارد.
  - این پایداری ناشی از ترکیب نرخ یادگیری تطبیقی و مومنتوم مناسب است.
- **مزیت :**عملکرد بهتر روی داده‌های متنوع و پیچیده.

چگونگی ترکیب باlr_scheduler

- از **lr_scheduler.MultiStepLR**برای تنظیم نرخ یادگیری استفاده شده است:

```
lr_scheduler = torch.optim.lr_scheduler.MultiStepLR(optimizer, milestones=[4, 8], gamma=0.1)
```

- **هدف:**
  - کاهش نرخ یادگیری در دوره‌های مشخص (Milestones) برای کمک به همگرایی بهتر.
  - این روش به مدل اجازه می‌دهد در مراحل اولیه سریع‌تر یاد بگیرد و در مراحل پایانی با دقت بیشتری تنظیم شود.

در مرحله آموزش فرض‌هایی صورت گرفته است که به آنها اشاره می‌شود:

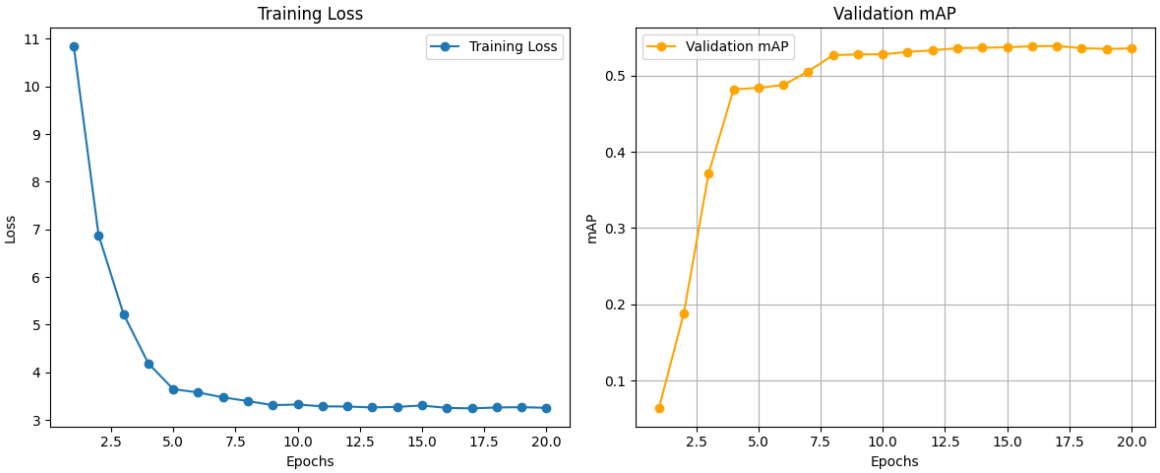
۳) **تعداد epoch**: این مقدار در مقاله یا صورت سوال اشاره نشده است و ما با توجه به زمانبر بودن اجرا و دست یافتن به دقت قابل قبول، تعداد ۲۰ ایپاک را در نظر گرفتیم.

۴) **Batch size**: این مقدار هم مشخصا تعریف نشده است. برای این سوال عدد ۸ را برای batch size در نظر گرفته‌ایم.

اکنون مدل آماده آموزش است و با استفاده از تابع زیر که در کلاس TrafficSignDetectorSSD تعریف شده این کار را انجام می‌دهیم. برای ارزیابی و چگونگی انجام آن در ادامه توضیح داده شده است.

۲-۳-۶. ارزیابی مدل

تابع ارزیابی در این بخش هم دقیقا مشابه بخش قبلی نوشته شده است. از نظر عملکرد و ساختار هیچ تفاوتی با بخش قبلی وجود ندارد اما نتایج به صورت زیر بدست آمده است.



شکل ۳۲- ارزیابی عملکرد مدل **SSD300** در حین آموزش با **IoU=0.5**

روند نزولی loss و صعودی mAP نشان از عملکرد صحیح و مناسب مدل است چرا که می‌دانیم مقدار loss در هر مرحله و با عملیات ریاضیتی داخل optimizer باید کاهش یابد و از طرفی مدل در هر گام باید بهتر از گام قبل عمل کند.

```
Evaluating for IoU threshold: 0.5
For IoU 0.5, mAP: 0.5355014204978943, mAP per class: [0.6791912913322449, 0.5977585315704346, 0.42457473278045654, 0.4404812157154083]
```

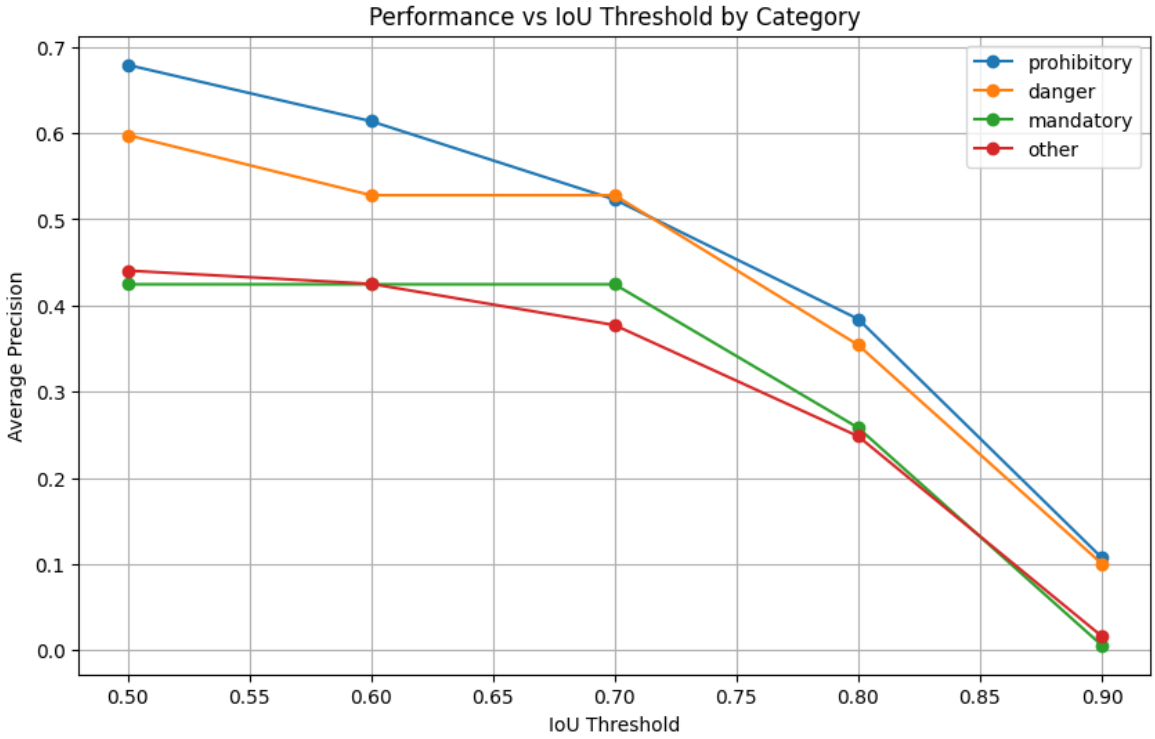
شکل ۳۳- **mAP** هر کلاس

جدول ۴- جدول **mAP** با معیار **IoU = 0.5** برای هر کلاس و به صورت میانگین

	mAP
Prohibitory	۶۷.۹۲
Danger	۵۹.۷۸
Mandatory	۴۲.۴۶
Other	۴۴.۰۵
میانگین	**۵۳.۵۵**

۲-۳-۷. ترسیم نمودار AP به ازای مقادیر متفاوت IoU

تابع و نحوه پیاده‌سازی این بخش هم دقیقا مشابه بخش قبلی است. اما نمودار و تحلیل آن به شرح زیر است.



شکل ۳۴- نمودار مقادیر مختلف **mAP** برای هر کلاس با توجه به **threshold** انتخابی

۱. مشاهده کلی:

۱. عملکرد دسته‌ها:

- دسته **prohibitory**عملکرد بالاتری نسبت به بقیه دسته‌ها دارد .AP برای این دسته در تمام آستانه‌های IoU بالاتر از دیگر دسته‌هاست.
- دسته **danger**نیز عملکرد مناسبی دارد اما نسبت به دسته prohibitory کمتر است.
- دسته‌های **mandatory**و **other**عملکرد ضعیف‌تری نسبت به دو دسته بالا دارند و در آستانه‌های IoU بالا کاهش دقت قابل توجهی دارند.

۲. **روند کاهش دقت:**

- با افزایش مقدار IoU از ۰.۵ به ۰.۹، میانگین دقت در تمام دسته‌ها کاهش می‌یابد. این کاهش طبیعی است زیرا آستانه IoU بالاتر به معنای سختگیری بیشتر در مطابقت پیش‌بینی مدل با جعبه واقعی (Ground Truth) است.

۲.تحلیل دسته‌ها:

الف) Prohibitory:

- **مشاهدات:**
  - این دسته در تمام آستانه‌های IoU بالاترین میانگین دقت را دارد و از حدود ۰.۷ IoU) (0.5 =شروع شده و تا حدود ۰.۳ (IoU = 0.9) کاهش می‌یابد.
- **دلیل عملکرد بهتر:**
  - علائم prohibitory معمولاً دارای طراحی ساده‌تر، شکل‌های واضح‌تر (دایره‌ای با رنگ قرمز) و تمایز بیشتری نسبت به پس‌زمینه هستند، که تشخیص آن‌ها را برای مدل آسان‌تر می‌کند.

ب) Danger:

- **مشاهدات:**
  - این دسته عملکردی نسبتاً خوب دارد، اما کمی پایین‌تر از دسته prohibitory است AP . از حدود ۰.۶ (IoU = 0.5) شروع شده و به ۰.۲ (IoU = 0.9) کاهش می‌یابد.
- **دلیل عملکرد:**

- علائم danger معمولاً مثلثی شکل هستند، اما ممکن است به دلیل پیچیدگی بیشتر طراحی (متن یا تصاویر داخلی)، تشخیص آن‌ها سخت‌تر باشد.

ج) Mandatory :

- **مشاهدات:**
  - عملکرد این دسته ضعیف‌تر از دو دسته بالا است .AP از حدود ۰.۴۵ (IoU = 0.5) شروع شده و به کمتر از ۰.۱۵ (IoU = 0.9) کاهش می‌یابد.
- **دلیل عملکرد ضعیف‌تر:**
  - علائم mandatory ممکن است طراحی پیچیده‌تری داشته باشند یا تنوع بیشتری در اندازه و شکل داشته باشند که تشخیص آن‌ها را دشوارتر می‌کند.

د) Other:

- **مشاهدات:**
  - این دسته پایین‌ترین عملکرد را دارد .AP از حدود ۰.۴ (IoU = 0.5) شروع شده و به نزدیک صفر (IoU = 0.9) کاهش می‌یابد.
- **دلیل عملکرد ضعیف‌تر:**
  - دسته other احتمالاً شامل علائم متفرقه‌ای است که ویژگی‌های مشخصی ندارند یا شباهت زیادی به پس‌زمینه دارند. این می‌تواند منجر به تشخیص نادرست یا از دست رفتن برخی از علائم شود.

۳.تحلیل کاهش دقت با:IoU

- **کاهش طبیعی:**
  - همان‌طور که IoU افزایش می‌یابد، مدل باید پیش‌بینی‌های دقیق‌تری انجام دهد. بنابراین، میانگین دقت در تمام دسته‌ها کاهش می‌یابد.
- **اختلاف عملکرد:**

- در IoU بالا (مثل ۰.۸ و ۰.۹)، اختلاف میان دسته‌ها بیشتر به چشم می‌آید. دسته‌های prohibitoryو danger همچنان بهتر عمل می‌کنند، اما mandatory و other دقت خود را به شدت از دست می‌دهند.

جمع‌بندی:

این نمودار نشان می‌دهد که مدل در شناسایی علائم prohibitory و danger عملکرد خوبی دارد، اما برای دسته‌های mandatory و other نیاز به بهبود وجود دارد. افزایش داده‌های آموزشی، تغییر انکرها و تنظیم دقیق‌تر مدل می‌تواند به بهبود این دسته‌ها کمک کند.

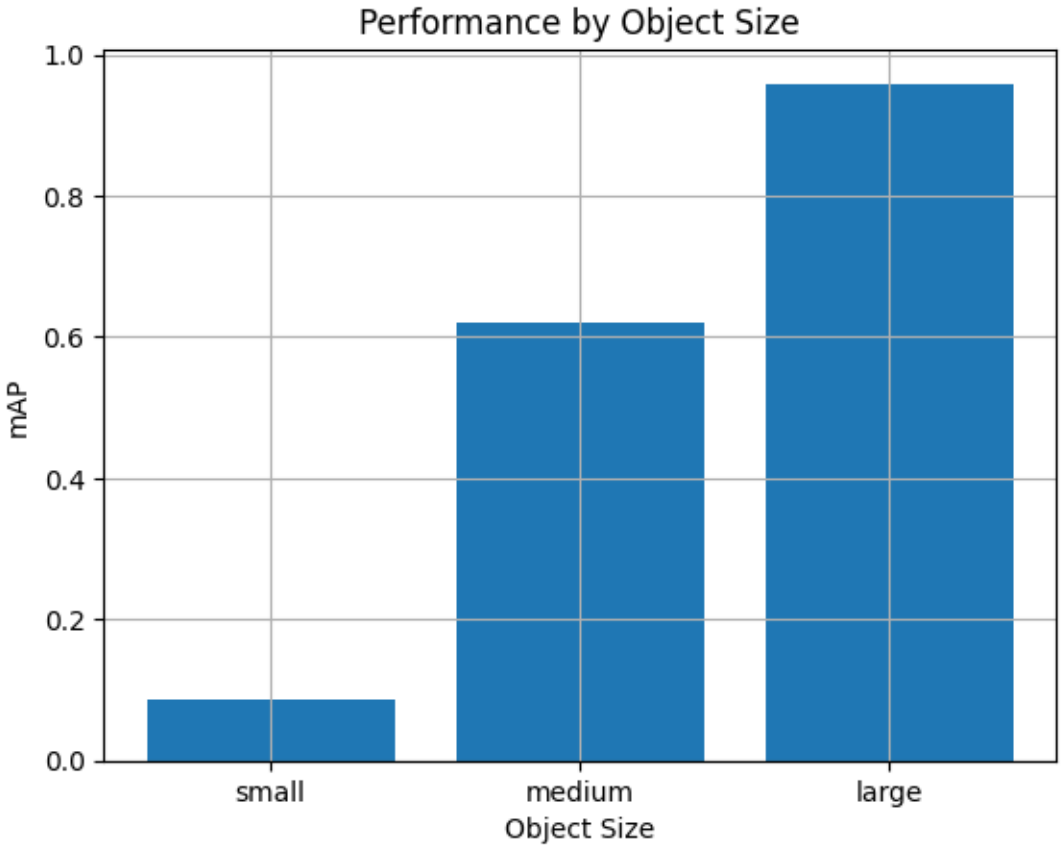
جدول ۵ - ارزیابی مدل **SSD300** با **threshold** های متفاوت

	mAP				
	IoU = 0.9	IoU = 0.8	IoU = 0.7	IoU = 0.6	IoU = 0.5
Prohibitory	۱۰.۷۳	۳۸.۴۲	۵۲.۲۹	۶۱.۳۸	۶۷.۹۲
Danger	۱۰.۰۳	۳۵.۴۱	۵۲.۸۰	۵۲.۸۰	۵۹.۷۸
Mandatory	۵.۶۶	۲۵.۸۱	۴۲.۴۶	۴۲.۴۶	۴۲.۴۶
Other	۱.۶۲	۲۴.۸۲	۴۲.۵۱	۴۲.۵۱	۴۴.۰۵
میانگین	**۵.۹۸**	**۳۱.۱۲**	**۴۶.۳۲**	**۴۹.۷۷**	**۵۳.۵۵**

دقت مدل کمتر از میزان قابل پیشبینی ما بوده است که در بخش ارزیابی و مقایسه نتایج دو مدل به شکل مفصلی در رابطه با علت کم بودن دقت نسبت به مدل دیگر و بالاتر نیامدن آن صحبت شده است.

۲-۳-۸. ارزیابی مدل بر اساس اندازه

تابع نوشته شده و نحوه اجرا دقیقا مشابه بخش قبلی است اما نمودار این بخش و تحیل آن به فرم زیر است.



شکل ۳۵- ارزیابی مدل **SSD300** بر اساس سایز

جدول ۶- ارزیابی مدل **SSD300** بر اساس سایز

mAP for IoU = 0.5			
Small	Medium	Large	
۸.۶۴	**۶۲.۰۲**	**۹۵.۸۷**	mAP for IoU = 0.5

این نمودار نشان‌دهنده عملکرد مدل در تشخیص اشیاء با اندازه‌های مختلف است. محور افقی سه دسته اندازه اشیاء small ، medium، large را نشان می‌دهد و محور عمودی مقدار mAP (میانگین دقت) را برای هر دسته اندازه مشخص می‌کند. تحلیل دقیق نمودار به شرح زیر است:

۱.مشاهده کلی:

۱. **اشیاء کوچک:(small)**

- این دسته پایین‌ترین عملکرد را دارد، با مقدار mAP کمتر از **0.2**.

- این نشان می‌دهد که مدل در شناسایی اشیاء کوچک به طور قابل توجهی ضعیف‌تر عمل می‌کند.

۲. **اشیاء متوسط (medium):**

- این دسته عملکرد بهتری نسبت به اشیاء کوچک دارد و mAP آن در حدود **0.6** است.
- مدل در این دسته به شکل قابل قبولی عمل می‌کند، اما همچنان بهبودهایی ممکن است.

۳. **اشیاء بزرگ (large):**

- این دسته بهترین عملکرد را با mAP نزدیک به **1.0** دارد.
- مدل در شناسایی اشیاء بزرگ عملکرد بسیار خوبی داشته و تقریباً به دقت کامل رسیده است.

۲.تحلیل دسته‌ها:

الف. اشیاء کوچک (small):

- **مشاهدات:**
  - مدل در شناسایی اشیاء کوچک ضعیف عمل می‌کند.
- **دلیل عملکرد ضعیف:**
  - اشیاء کوچک معمولاً وضوح کمتری دارند و ممکن است در تصاویر نویزی یا دارای پس‌زمینه شلوغ به خوبی قابل تشخیص نباشند.
  - ابعاد کوچک این اشیاء ممکن است باعث شود انکر باکس‌های پیش‌فرض مدل با این اشیاء هم‌خوانی نداشته باشند.

ب. اشیاء متوسط (medium):

- **مشاهدات:**
  - عملکرد مدل در شناسایی اشیاء متوسط قابل قبول است، اما هنوز جای بهبود دارد.
- **دلیل عملکرد بهتر:**

- اشیاء متوسط به اندازه کافی بزرگ هستند که جزئیات آن‌ها در تصاویر به وضوح قابل مشاهده باشد.
- انکر باکس‌های پیش‌فرض مدل به طور مؤثری می‌توانند این اشیاء را پوشش دهند.

ج. اشیاء بزرگ:(large)

- **مشاهدات:**
  - مدل در شناسایی اشیاء بزرگ عملکرد تقریباً بی‌نقص دارد.
- **دلیل عملکرد عالی:**
  - اشیاء بزرگ در تصاویر به وضوح قابل مشاهده هستند و انکر باکس‌ها به راحتی می‌توانند با این اشیاء تطابق داشته باشند.
  - این اشیاء معمولاً نویز کمتری دارند و شباهت کمتری به پس‌زمینه دارند.

۳.جمع‌بندی:

- **اشیاء بزرگ :**عملکرد عالی با mAP نزدیک به ۱.۰.
- **اشیاء متوسط :**عملکرد خوب اما با امکان بهبود.
- **اشیاء کوچک :**عملکرد ضعیف، نیازمند تمرکز بیشتر مدل بر این دسته است.

۲-۳-۹. نمایش نمونه تصویر

باز هم تابع نوشته شده و نحوه استفاده دقیقا مشابه بخش قبل است. اما تصویر نمونه برای این قسمت به فرم زیر است.

شکل ۳۶- نمونه تصویر مقایسه مدل **SSD300** با **ground truth**

در تصویر فوق همپوشانی ۷۵ درصدی پیشبینی مدل با bounding box موجود در ground truth و همینطور پیشبینی صحیح کلاس را مشاهده می‌کنیم.

۲-۴. ارزیابی نتایج و مقایسه مدل‌ها

۲-۴-۱. مقایسه مدل‌های Faster R-CNN و SSD بر اساس آستانه‌هایIoU و برای کلاس‌های متخلف

مقایسه کلی:

۱. **Faster R-CNN:**

- عملکرد بهتری در تمام آستانه‌هایIoU ، به خصوص در مقادیر بالاتر (۰.۸ و ۰.۹) دارد.
- مقدار mAP بالایی را برای اکثر دسته‌ها در آستانه‌های پایین‌تر IoU حفظ می‌کند.

۲. **SSD:**

- به طور کلی عملکرد ضعیف‌تری دارد و با افزایش آستانه IoU کاهش چشم‌گیری در عملکرد آن دیده می‌شود.

- به ویژه در دسته‌های "mandatory" و "other" ضعیف‌تر عمل می‌کند.

عملکرد به تفکیک دسته‌ها:

Prohibitory:

- **Faster R-CNN**:
  - mAPبالایی در تمام آستانه‌ها نشان می‌دهد؛ از مقدار ۰.۹۷۲ در IoU=0.5 شروع شده و به ۰.۳۰۵ در IoU=0.9 کاهش می‌یابد.
  - پایداری بهتری نسبت به SSD دارد.
- **SSD**:
  - mAPدر IoU=0.5 برابر با ۰.۶۷۹ است، اما در IoU=0.9 به ۰.۱۰۷ کاهش می‌یابد.
- **برنده:** Faster R-CNN
  - این مدل دقت و پایداری بیشتری برای این دسته دارد.

Danger:

- **Faster R-CNN**:
  - عملکرد خوبی دارد؛ از ۰.۹۵۱ در IoU=0.5 شروع شده و به ۰.۲۶۶ در IoU=0.9 کاهش می‌یابد.
  - در تمام آستانه‌ها mAP بالاتری نسبت به SSD دارد.
- **SSD**:
  - mAPاز ۰.۵۹۷ در IoU=0.5 شروع شده و در IoU=0.9 به ۰.۱۰۰ می‌رسد.
- **برنده**Faster R-CNN **:**
  - به دلیل بازدهی بهتر در تشخیص دقیق‌تر این دسته.

Mandatory:

- **Faster R-CNN**:
  - از ۰.۸۸۴ در IoU=0.5 شروع شده و به ۰.۲۴۷ در IoU=0.9 کاهش می‌یابد.
  - عملکرد نسبتاً پایدار در این دسته دارد.
- **SSD**
  - عملکرد ضعیفی دارد؛ از ۰.۴۲۴ در IoU=0.5 شروع شده و به ۰.۰۰۵ در IoU=0.9 کاهش می‌یابد.
- **برنده**Faster R-CNN **:**
  - SSDبه وضوح در تشخیص اشیاء کوچک‌تر یا پیچیده‌تر در این دسته ضعف دارد.

Other:

- **Faster R-CNN**:
  - mAPاز ۰.۹۲۳ در IoU=0.5 شروع شده و به ۰.۱۴۱ در IoU=0.9 کاهش می‌یابد.
  - در تمام آستانه‌ها بهتر از SSD عمل می‌کند.
- **SSD**:
  - mAPاز ۰.۴۴۰ در IoU=0.5 شروع شده و در IoU=0.9 به ۰.۰۱۶ کاهش می‌یابد.
- **برنده:** Faster R-CNN
  - عملکرد ضعیف SSD نشان‌دهنده دشواری آن در شناسایی اشیاء متنوع و نامعمول است.

مقایسه mAP کلی:

Faster R-CNN:

- مقدار mAP در تمام آستانه‌ها بالاتر است:
  - **IoU=0.5:** 0.932
  - **IoU=0.9:** 0.240

SSD:

- mAPکلی پایین‌تر است و کاهش بیشتری با افزایش IoU دارد:
  - **IoU=0.5:** 0.535
  - **IoU=0.9:** 0.057
- **برنده:** Faster R-CNN
  - این مدل در مجموع دقت بالاتری دارد و پایداری بیشتری در آستانه‌های IoU مختلف نشان می‌دهد.

دلایل تفاوت عملکرد:

چرا Faster R-CNN بهتر عمل می‌کند؟

۱. **شبکه پیشنهادی منطقه‌ای(RPN) :**

- Faster R-CNNاز یک شبکه پیشنهادی برای تولید جعبه‌های منطقه‌ای (Region Proposals)استفاده می‌کند که باعث شناسایی دقیق‌تر اشیاء، به ویژه در آستانه‌های بالای IoU می‌شود.
- این ویژگی به بهبود تشخیص دسته‌هایی مانند "mandatory" و "other" کمک می‌کند.

۲. **استخراج ویژگی‌های قوی‌تر:**

- Faster R-CNNاز یک شبکه عمیق‌تر و پیشرفته‌تر) مانند (ResNet برای استخراج ویژگی استفاده می‌کند، که توانایی آن را در شناسایی اشیاء پیچیده و جزئیات ظریف افزایش می‌دهد.

۳. **تنظیم دقیق جعبه‌ها:**

- این مدل دقت بالایی در پیش‌بینی مختصات جعبه‌های محدودکننده (Bounding Box) دارد که به حفظ دقت در آستانه‌های IoU بالاتر کمک می‌کند.

۴. **مدیریت بهتر مقیاس‌ها:**

- معماری Faster R-CNN به طور طبیعی برای مدیریت اشیاء با اندازه‌ها و مقیاس‌های مختلف مناسب‌تر است.

۵. **سازگاری بیشتر با داده‌های کم:**

- Faster R-CNNبه دلیل معماری پیشرفته‌تر و استفاده از ویژگی‌های عمیق‌تر، عملکرد پایدارتری در مجموعه داده‌های کوچک (مانند ۹۰۰ تصویر شما) دارد.
- شبکه پیشنهادی (RPN) این مدل نیاز به داده‌های کمتر برای یادگیری توزیع جعبه‌های پیشنهادی دارد، که آن را برای داده‌های کم مناسب‌تر می‌کند.

چرا SSD عملکرد ضعیف‌تری دارد؟

۱. **باکس‌های انکر ثابت:**
   - SSDبه شدت به باکس‌های انکر از پیش تعریف‌شده متکی است، که ممکن است با اندازه‌ها و نسبت‌های متفاوت اشیاء به خوبی تطبیق نداشته باشد، به ویژه برای اشیاء کوچک‌تر.

۲. **ویژگی‌های کم‌عمق‌تر:**
   - SSDاز نقشه‌های ویژگی کم‌عمق‌تر برای پیش‌بینی استفاده می‌کند، که توانایی آن را در استخراج ویژگی‌های پیچیده محدود می‌کند.

۳. **عدم وجود:RPN**
   - نبود شبکه پیشنهادی منطقه‌ای باعث می‌شود SSD پیش‌بینی‌های مستقیم انجام دهد، که دقت را کاهش داده و منجر به افزایش مثبت‌های کاذب می‌شود.

۴. **حساسیت به اشیاء کوچک:**
   - SSDدر شناسایی اشیاء کوچک یا دارای ویژگی‌های ظریف، مانند دسته"mandatory" ، به وضوح مشکل دارد.

۵. **نیاز به داده‌های بیشتر:**
   - SSDبرای یادگیری توزیع جعبه‌ها و تنظیم انکرها نیاز به تعداد زیادی داده آموزشی دارد. با توجه به اینکه تعداد داده‌های شما تنها **900تصویر** است، این مدل به دلیل معماری سبک‌تر و وابستگی به داده‌های بزرگ، عملکرد ضعیف‌تری نشان می‌دهد.

پس به صورت کلی می‌توان گفت:

- **Faster R-CNN:** به دلیل استفاده از RPN و معماری عمیق، توانایی بهتری در تعمیم‌دهی با داده‌های کم دارد. این موضوع باعث می‌شود حتی با تعداد محدود تصاویر (۹۰۰ تصویر)، مدل عملکرد مناسبی داشته باشد.
- **SSD:** به داده‌های بیشتر برای یادگیری بهتر نیاز دارد. تعداد محدود تصاویر باعث می‌شود این مدل نتواند انکرها را به خوبی تطبیق داده و اشیاء را به دقت تشخیص دهد.

نتیجه‌گیری نهایی:

- **Faster R-CNN** به دلیل سازگاری بیشتر با مجموعه داده‌های کوچک (۹۰۰ تصویر) و معماری پیشرفته‌تر، انتخاب بهتری برای شناسایی دقیق علائم راهنمایی و رانندگی است.
- **SSD** با وجود سبک‌تر بودن، برای مجموعه داده‌های کوچک مناسب نیست و دقت آن به شدت کاهش می‌یابد.

۲-۴-۲. مقایسه مدل‌های Faster R-CNN و SSD300 بر اساس اندازه اشیاء

.1مقایسه کلی:

۱. **Faster R-CNN**:
   - **اشیاء کوچک**: 0.846
   - **اشیاء متوسط**: 0.902
   - **اشیاء بزرگ:** 0.958
   - عملکردی یکنواخت و قوی در تمام اندازه‌ها دارد، با کاهش جزئی در اشیاء کوچک.

۲. **SSD300**:
   - **اشیاء کوچک**: 0.086
   - **اشیاء متوسط**: 0.620
   - **اشیاء بزرگ**: 0.958

- عملکرد بسیار ضعیف در اشیاء کوچک، متوسط برای اشیاء متوسط، و برابر با Faster R-CNN برای اشیاء بزرگ.

۲.عملکرد بر اساس اندازه اشیاء:

اشیاء کوچک:

- **Faster R-CNN:**
  - mAP: 0.846
  - عملکرد بسیار عالی در شناسایی اشیاء کوچک.
- **SSD300:**
  - mAP: 0.086
  - عملکرد بسیار ضعیف که نشان‌دهنده مشکل مدل در شناسایی اشیاء کوچک است.
- **برنده:** Faster R-CNN

اشیاء متوسط:

- **Faster R-CNN:**
  - mAP: 0.902
  - عملکرد بسیار خوب با کاهش جزئی نسبت به اشیاء بزرگ.
- **SSD300:**
  - mAP: 0.620
  - عملکرد متوسط اما به طور قابل توجهی پایین‌تر از Faster R-CNN.
- **برنده:** Faster R-CNN

اشیاء بزرگ:

- **Faster R-CNN:**
  - mAP: 0.958
  - عملکرد عالی و مشابه SSD300.
- **SSD300:**

- mAP: 0.958
- عملکرد برابر با.Faster R-CNN

- **برنده :**مساوی

۳. جمع‌بندی کلی:

- **Faster R-CNN**: عملکرد کلی بهتری در تمام اندازه‌ها دارد و در شناسایی اشیاء کوچک و متوسط بسیار بهتر عمل می‌کند.
- **SSD300**: تنها در شناسایی اشیاء بزرگ عملکرد قابل قبولی دارد، اما در اشیاء کوچک و متوسط بسیار ضعیف‌تر است.

دلایل تفاوت عملکرد:

چرا Faster R-CNN بهتر عمل می‌کند؟

۱. **شبکه پیشنهادی منطقه‌ای(RPN) :**

- استفاده از RPN باعث شناسایی دقیق‌تر اشیاء به خصوص برای اندازه‌های کوچک و متوسط می‌شود.

۲. **استخراج ویژگی‌های عمیق‌تر:**

- Faster R-CNNاز شبکه‌های عمیق‌تر برای استخراج ویژگی استفاده می‌کند که برای شناسایی جزئیات اشیاء کوچک مناسب‌تر است.

۳. **تنظیم دقیق جعبه‌ها:**

- روش دو مرحله‌ای این مدل باعث تنظیم بهتر جعبه‌ها می‌شود.

۴. **سازگاری بیشتر با داده‌های کم:**

- با توجه به تعداد کم تصاویر (۹۰۰ عکس)، این مدل بهتر از SSD می‌تواند تعمیم دهد و یاد بگیرد.

چرا SSD300 عملکرد ضعیف‌تری دارد؟

۱. **باکس‌های انکر ثابت:**

- SSDنمی‌تواند اشیاء کوچک یا متوسط را به خوبی با انکرهای پیش‌فرض شناسایی کند.

۲. **ویژگی‌های کم‌عمق‌تر:**

- ویژگی‌های کم‌عمق باعث می‌شود مدل در استخراج جزئیات اشیاء کوچک ضعیف عمل کند.

۳. **عدم وجود:RPN**

- نبود شبکه پیشنهادی باعث کاهش دقت و افزایش مثبت‌های کاذب می‌شود.

۴. **نیاز به داده‌های بیشتر:**

- SSDبرای یادگیری بهتر به تعداد بیشتری داده نیاز دارد و تعداد محدود (۹۰۰ تصویر) برای آن کافی نیست.

نتیجه‌گیری نهایی:

- **Faster R-CNN**به دلیل معماری پیشرفته‌تر، برای این مجموعه داده کوچک (۹۰۰ عکس) و شناسایی اشیاء کوچک و متوسط، انتخاب مناسب‌تری است.
- **SSD300**با وجود سبک‌تر بودن، برای مجموعه داده‌های کوچک مناسب نیست و در شناسایی دقیق اشیاء کوچک به شدت ضعف دارد.

۲-۴-۳. عوامل مؤثر بر عملکرد مدل‌ها

عملکرد مدل‌ها می‌تواند تحت تأثیر عوامل مختلفی قرار گیرد. در اینجا مهم‌ترین عوامل را بررسی می‌کنیم:

۱.کیفیت و تعداد داده‌ها

- **تأثیر:**

- تعداد کم داده‌ها (مانند ۹۰۰ تصویر در این مورد) می‌تواند منجر به یادگیری ناکافی مدل شود، به خصوص برای مدل‌هایی مانند SSD که به داده‌های زیاد وابسته هستند.
- کیفیت پایین تصاویر (مانند نویز، وضوح پایین یا پس‌زمینه‌های شلوغ) می‌تواند تشخیص اشیاء را دشوار کند.
- تنوع ناکافی در داده‌ها (مانند زاویه‌های مختلف، شرایط نوری متفاوت یا اندازه‌های مختلف اشیاء) می‌تواند دقت مدل را کاهش دهد.

- **راهکارها:**

۱. **افزایش داده‌ها:**

  - استفاده از تکنیک‌های افزایش داده (Data Augmentation) مانند چرخش، برش، تغییر مقیاس، و افزودن نویز برای ایجاد تصاویر جدید.
  - جمع‌آوری داده‌های بیشتر از شرایط واقعی.

۲. **تمیز کردن داده‌ها:**

  - حذف داده‌های نویزی یا کم‌کیفیت.

۳. **تنوع در داده‌ها:**

  - اطمینان از وجود تنوع در زاویه، اندازه، و شرایط نوری در تصاویر.

۲. اندازه اشیاء

- **تأثیر:**
  - مدل‌ها معمولاً در تشخیص اشیاء کوچک ضعیف عمل می‌کنند. همان‌طور که در نتایج مشخص شد، SSDدر تشخیص اشیاء کوچک عملکرد بسیار ضعیفی داشت.
  - اشیاء کوچک ممکن است در نقشه‌های ویژگی (Feature Maps) به وضوح دیده نشوند.
- **راهکارها:**

۱. **استفاده از انکرهای کوچک‌تر:**

  - تنظیم انکر باکس‌ها برای پوشش بهتر اشیاء کوچک.

۲. **مدل‌های چندرزولوشنه:**

  - استفاده از معماری‌هایی مانند Feature Pyramid Networks (FPN) یا EfficientDetکه اطلاعات چندرزولوشنه را برای شناسایی اشیاء کوچک ترکیب می‌کنند.

۳. معماری مدل

- **تأثیر:**

- مدل‌های پیچیده‌تر مانند Faster R-CNN دقت بالاتری دارند، اما زمان محاسباتی بیشتری نیاز دارند.
- مدل‌های سبک‌تر مانند SSD سرعت بیشتری دارند اما در دقت، به خصوص برای داده‌های کم، ضعیف‌تر عمل می‌کنند.

- **راهکارها:**

۱. **استفاده از معماری‌های ترکیبی:**

- معماری‌هایی مانند RetinaNet یا EfficientDet می‌توانند تعادلی میان دقت و سرعت ایجاد کنند.

۲. **تنظیم معماری موجود:**

- تغییر لایه‌های خروجی یا استفاده از انکرهای بیشتر در SSD برای بهبود دقت.

۴. بهینه‌سازی(Optimization)

- **تأثیر:**

- انتخاب نادرست بهینه‌ساز یا نرخ یادگیری می‌تواند منجر به یادگیری ناقص یا بیش‌برازش شود.
- در مدل‌هایی مانندSSD ، تنظیمات ضعیف می‌تواند بر تطبیق مدل با داده‌های کم تأثیر منفی بگذارد.

- **راهکارها:**

۱. **بهینه‌ساز مناسب:**

- استفاده از بهینه‌سازهایی مانند Adam یا AdamW که نرخ یادگیری تطبیقی دارند.

۲. **تنظیم نرخ یادگیری:**

- استفاده از Scheduler برای کاهش تدریجی نرخ یادگیری در طول آموزش.

۳. **کاهش گرادیان‌های ناپایدار:**

- استفاده از تکنیک‌های Clipping برای جلوگیری از انفجار گرادیان‌ها.

۵. تنظیمات هایپرپارامترها

- **تأثیر:**
  - مقادیر پیش‌فرض هایپرپارامترها (مانند تعداد انکرها، نسبت ابعاد، یا نرخ یادگیری) ممکن است برای مسئله خاص شما مناسب نباشند.
- **راهکارها:**

۱. **تنظیم انکرها:**

- تغییر تعداد و نسبت ابعاد انکرها برای انطباق با اندازه و شکل اشیاء در داده‌ها.

۲. **آزمایش نرخ یادگیری:**

- آزمایش مقادیر مختلف نرخ یادگیری و وزن‌دهی (Weight Decay) برای یافتن مقدار بهینه.

۶. محدودیت‌های سخت‌افزاری

- **تأثیر:**
  - مدل‌های پیچیده‌تر به سخت‌افزار قدرتمندتری نیاز دارند و ممکن است رویGPU های ضعیف‌تر عملکرد مناسبی نداشته باشند.
- **راهکارها:**

۱. **استفاده از مدل‌های سبک‌تر:**

- برای محدودیت سخت‌افزاری می‌توان از مدل‌های سبک‌تر مانند MobileNet SSDاستفاده کرد.

۲. **استفاده از تسریع‌کننده‌ها:**

- استفاده از TensorRT یا ONNX برای بهبود سرعت مدل.

جمع‌بندی راهکارها برای بهبود دقت و سرعت:

۱. **برای بهبود دقت:**

- افزایش داده‌های آموزشی با استفاده از Data Augmentation یا جمع‌آوری داده‌های بیشتر.
- استفاده از معماری‌هایی با قابلیت تشخیص بهتر اشیاء کوچک مانند.FPN
- تنظیم دقیق انکرها و نرخ یادگیری.

۲. **برای بهبود سرعت:**

- استفاده از مدل‌های سبک‌تر مانند.MobileNet SSD
- کاهش اندازه تصاویر ورودی برای کاهش زمان پردازش.
- استفاده از تسریع‌کننده‌های سخت‌افزاری مانند.TensorRT