به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

درس شبکه‌های عصبی و یادگیری عمیق

تمرین سوم

نام و نام خانوادگی	محمدامین یوسفی
شماره دانشجویی	810100236
نام و نام خانوادگی	محمدرضا نعمتی
شماره دانشجویی	810100226

فهرست

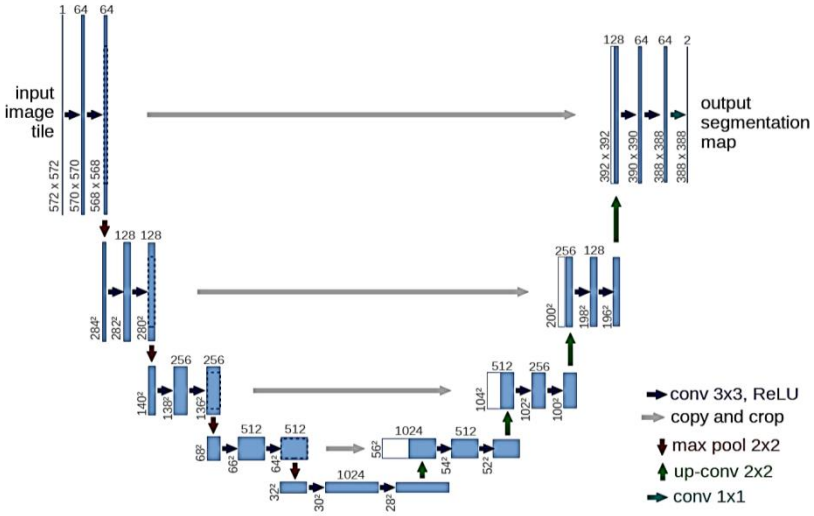
شکل‌ها و جدول‌ها

پرسش 1. سگمنتیشن تومور مغزی از روی تصاویر MRI

۱-۱. توصیف مدل ارائه شده

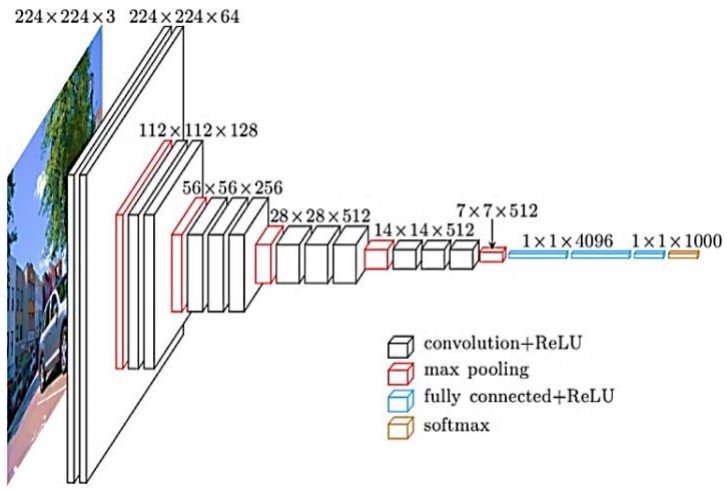
مدل پیشنهادی در مقاله به نام **UNet-VGG16** با یادگیری انتقالی، ترکیبی از معماری‌های U-Net و VGG16 است. این مدل برای سگمنتیشن تصاویر MRI مغز به منظور تشخیص تومور استفاده شده است.

- U-Net شامل دو مسیر اصلی است که معماری آن در شکل 1-1 آمده است:
  - مسیر **انکودر (encoder)**: برای استخراج ویژگی‌ها از تصویر.
  - مسیر **دیکودر (decoder)**: برای بازسازی تصویر به اندازه اصلی و انجام پیش‌بینی پیکسل به پیکسل.



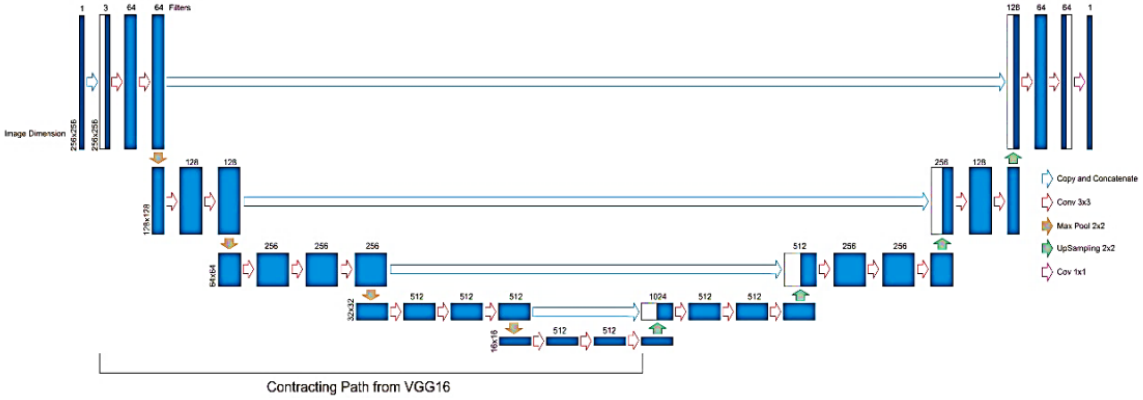
شکل 1-1. معماری مدل **U-Net**

- VGG16 به عنوان لایه انکودر در U-Net استفاده شده تا معماری U-Net ساده‌تر و کارآمدتر شود. معماری مدل VGG16، شکل 1-2، شامل 16 لایه trainable است که از فیلترهای کوچک 3*3 برای کانولوشن استفاده می‌کند. لایه‌های کانولوشن به صورت پیاپی با لایه‌های MaxPooling ترکیب می‌شوند تا ابعاد ویژگی‌ها کاهش یابد و در عین حال اطلاعات مهم حفظ شود. در انتهای شبکه، سه لایه Fully Connected برای پردازش و دسته‌بندی ویژگی‌های استخراج‌شده به کار می‌روند.



شکل 1-2. معماری مدل **VGG16**

- مدل U-Net در حالت عادی، 31,031,685 پارامتر دارد که همه‌ی آن trainable است. اما مدل UNet-VGG16 ارائه شده 17,040,001 پارامتر دارد که چون بخش انکودر را freeze کرده‌ایم، فقط 2,324,353 پارامتر trainable دارد که تاثیر بسیار زیادی بر سرعت یادگیری و منابع مورد نیاز دارد. معماری مدل ارائه شده در شکل 1-3 نمایش داده شده است.



شکل 1-3. معماری مدل **UNet-VGG16**

نقش هر قسمت:

- **لایه انکودر (VGG16):** ویژگی‌های تصویر را استخراج می‌کند. این بخش بر اساس یادگیری انتقالی (Transfer Learning)، در فرایند آموزش مدل freeze شده و از وزن‌های از پیش آموزش‌دیده از VGG16 استفاده میکند که باعث صرفه‌جویی در زمان محاسبات و افزایش دقت مدل می‌شود.

- **لایه‌دیکودر:**

شامل لایه‌های UpSampling و کانولوشن است که اندازه ماتریس ویژگی‌ها را به ابعاد اولیه باز می‌گرداند و پیکسل‌های مرتبط با تومور را مشخص می‌کند.

پس از ورود تصویر MRI به مدل، انکودر ویژگی‌های تصویر را استخراج کرده و دیکودر این ویژگی‌ها را بازسازی می‌کند تا محدوده‌های دقیق تومور پیش‌بینی شوند. در نهایت، نتایج سگمنتیشن با داده‌های واقعی (ground truth) مقایسه شده و دقت مدل سنجیده می‌شود.

انتخاب معماری VGG16

- **شباهت ساختاری به U-Net:**

VGG16به دلیل طراحی سلسله‌مراتبی و استفاده از هسته‌های کانولوشنی کوچک، شباهت زیادی به لایه انکودر U-Net دارد. این تطابق ساختاری باعث می‌شود ادغام VGG16 در معماری U-Net ساده‌تر و موثرتر باشد.

- **تعداد پارامترهای کمتر:**

با وجود عمق زیاد (16 لایه)، تعداد پارامترهای VGG16 نسبت به معماری‌های پیچیده‌تر کمتر است. این ویژگی باعث می‌شود که محاسبات سریع‌تر و کارآمدتر باشد، به خصوص در سیستم‌هایی با منابع سخت‌افزاری محدود.

- **وزن‌های از پیش آموزش‌دیده:**

VGG16 بر روی مجموعه داده ImageNet آموزش دیده است، که شامل میلیون‌ها تصویر متنوع است. این وزن‌های از پیش آموزش‌دیده به عنوان پایه‌ای قوی برای استخراج ویژگی‌های عمومی عمل می‌کنند، حتی در مسائل پزشکی مانند سگمنتیشن تومور.

- **موفقیت و پایداری اثبات‌شده:**

VGG16 یکی از شبکه‌های عصبی کانولوشنی استاندارد است که در بسیاری از مسائل بینایی کامپیوتری به کار گرفته شده و عملکرد بالای آن اثبات شده است. این اعتبار علمی باعث می‌شود انتخاب مطمئنی برای یادگیری انتقالی باشد.

نقش یادگیری انتقالی

- **کاهش زمان و هزینه محاسبات:**

با استفاده از وزن‌های از پیش آموزش‌دیده، نیازی به آموزش کامل مدل UNet-Vgg16 که بیش از 17 میلیون پارامتر دارد نیست، که سرعت آموزش را افزایش می‌دهد.

- **بهبود عملکرد روی داده‌های محدود:**

تصاویر MRI تومور مغزی معمولا محدود هستند و جمع‌آوری داده‌های پزشکی باکیفیت بالا دشوار است. یادگیری انتقالی باعث بهبود عملکرد مدل در شرایط کمبود داده می‌شود. از آنجایی که وزن‌ها قبلا بر روی مجموعه داده‌های بزرگ آموزش دیده‌اند، مدل بهتر می‌تواند ویژگی‌های عمومی تصاویر را استخراج کند.

- **کاهش خطر Overfitting**:

با freeze کردن بخش انکودر، مدل تنها برای یادگیری ویژگی‌های مرتبط با مسئله جدید (سگمنتیشن تومور) بهینه‌سازی می‌شود. این کار باعث کاهش خطر Overfitting به دلیل داده‌های محدود می‌شود.

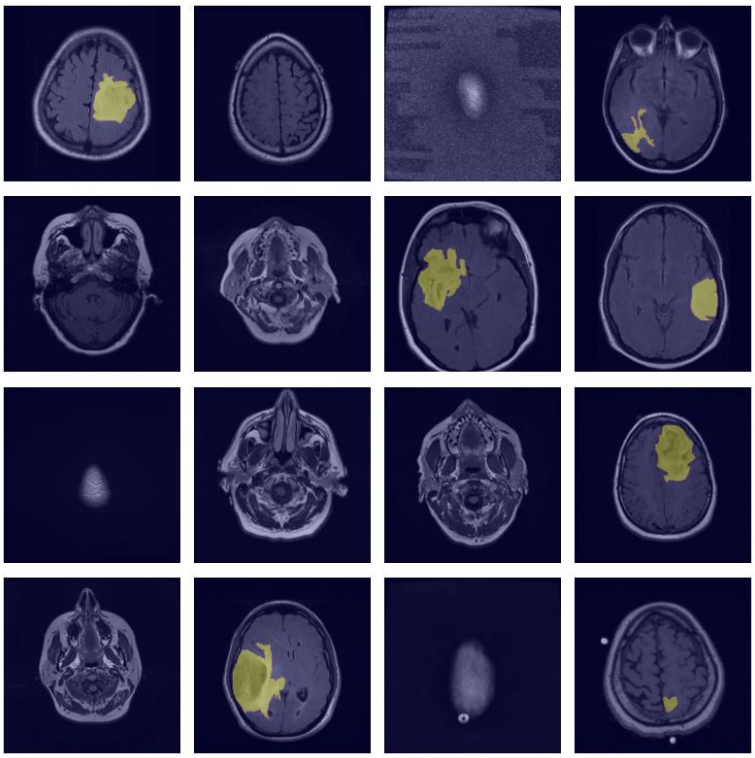
- **تعمیم بهتر مدل:**

یادگیری انتقالی از ویژگی‌هایی استفاده می‌کند که در مسائل متنوع آزمایش شده‌اند و این به مدل کمک می‌کند تا در داده‌های جدید و دیده‌نشده عملکرد بهتری داشته باشد.

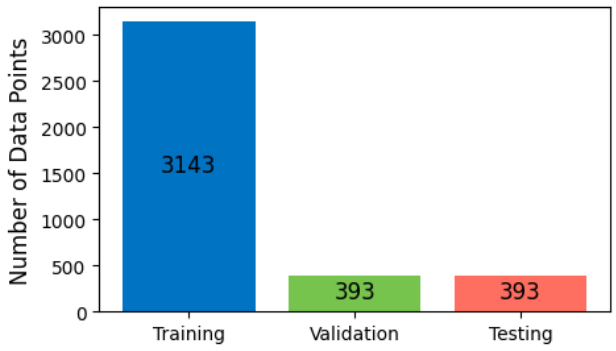
۱-۲. آماده‌سازی مجموعه داده

دیتاست **LGG Segmentation Dataset** شامل تصاویر MRI از بیماران مبتلا به تومور مغزی نوع Low-Grade Gliomas (LGG) است. این دیتاست شامل تصاویر MRI همراه با برچسب‌های سگمنتیشن است که نواحی تومور را مشخص می‌کند. نمونه‌هایی از آن به همراه ماسک (mask) سگمنتیشن در شکل 1-4 آمده است.

داده‌ها را با نسبت 80-10-10 به سه دسته آموزش (train)، اعتبارسنجی (validation) و ارزیابی (test) تقسیم می‌کنیم که تعداد هر دسته در شکل 1-5 قابل مشاهده است. البته توجه کنید که ساختار دیتاست به این گونه است که تعدادی بیمار دارد و هر بیمار، عکس‌های مربوط به خودش را دارد. در تقسیم بندی دیتاست، همه عکس‌ها به صورت تصادفی تقسیم شده‌اند و لزومی ندارد که همه عکس‌های یک بیمار در یک دسته باشند.



شکل 1-4. نمونه‌هایی از دیتاست **LGG Segmentation Dataset**



شکل 1-5. تعداد نمونه‌های دسته‌های آموزش، اعتبارسنجی و ارزیابی

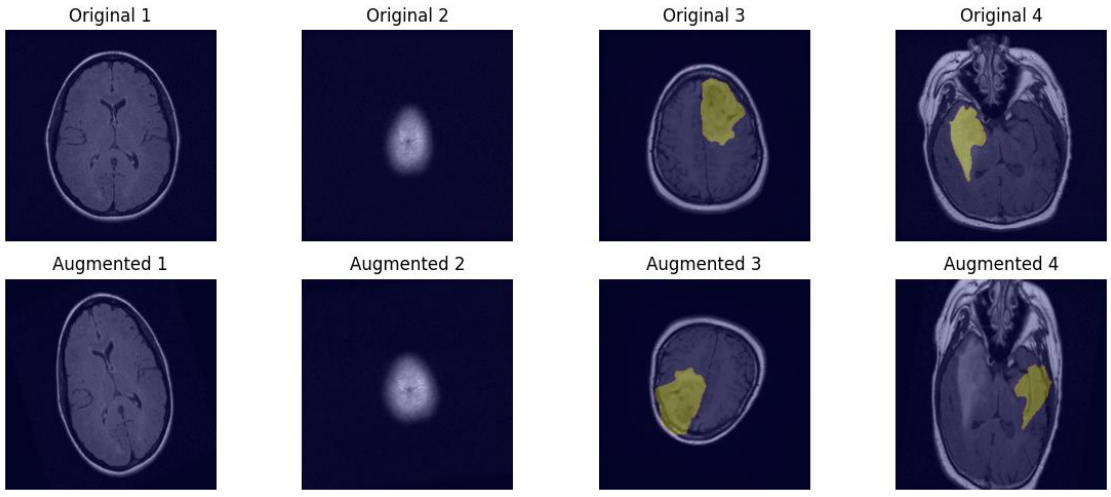
۱-۳. تقویت داده

Data augmentation به مدل‌های یادگیری عمیق کمک می‌کند تا عملکرد بهتری در مواجهه با داده‌های جدید و دیده‌نشده داشته باشند. با اعمال تغییراتی مانند چرخش، تغییر مقیاس، وارونه‌سازی، برش، و تنظیم روشنایی، تنوع داده‌ها افزایش می‌یابد، که این امر خطر Overfitting را کاهش می‌دهد. همچنین، این روش موجب می‌شود مدل ویژگی‌های عمومی‌تری از داده‌ها یاد بگیرد و دقت و پایداری آن در شرایط مختلف بهبود یابد. این فرآیند بخشی کلیدی از پیش‌پردازش است و زمینه‌ای قوی برای

آموزش مؤثر مدل فراهم می‌کند. جدول 1-1 انواع روش‌های تقویت داده استفاده شده و شکل 1-6 نمونه‌هایی از آن را نشان می‌دهد.

جدول 1-1. روش‌های انتخاب شده برای تقویت داده و توضیحات

Augmentation	Value	Description
zoom_range	0.2	این تکنیک بخش‌هایی از تصویر را بزرگ یا کوچک‌نمایی می‌کند که باعث می‌شود مدل بتواند ویژگی‌ها را در مقیاس‌های مختلف یاد بگیرد. این کار کمک می‌کند تا مدل در برابر تغییرات مقیاس مقاوم‌تر شود و ویژگی‌های مهم را در سطوح مختلف تصویر شناسایی کند.
rotation_range	25	تصاویر را در محدوده مثبت منفی می‌چرخاند. این عملیات تغییرات جزئی زاویه‌ای را شبیه‌سازی کرده و تعمیم‌پذیری مدل را بهبود می‌بخشد.
horizontal_flip	True	این روش تصاویر را به صورت تصادفی افقی برمی‌گرداند. این کار برای داده‌هایی که ویژگی‌هایشان در دو طرف تصویر یکسان است (مانند مغز در این دیتاست) مفید است و باعث می‌شود مدل از یادگیری الگوهای جهت‌دار غیرعمومی جلوگیری کند.
vertical_flip	True	این روش تصاویر را به صورت تصادفی عمودی برمی‌گرداند. این تکنیک در مسائلی که جهت عمودی اهمیت زیادی ندارد (مانند مغز) مفید است. البته باید در استفاده از آن دقت کرد، زیرا برای داده‌هایی مانند چهره انسانی ممکن است غیرمنطقی باشد.
brightness_range	[0.8, 1.2]	تنظیم روشنایی تصاویر به صورت تصادفی کمک می‌کند تا مدل در شرایط نوری مختلف عملکرد خوبی داشته باشد. این تکنیک باعث افزایش مقاومت مدل در برابر تغییرات محیطی می‌شود، خصوصا همانطور که در نمونه‌ها دیده شد، برخی از تصاویر MRI روشنایی کم یا زیادی دارند.



شکل 1-6. نمونه‌هایی از تصاویر تقویت‌شده به همراه نسخه اصلی آنها

۱-۴. بهینه‌ساز، معیارها و تابع هزینه

```python
def dice_coef(y_true, y_pred, smooth=100):
    y_true = K.cast(y_true, 'float32')
    y_pred = K.cast(y_pred, 'float32')
    y_true_flatten = K.flatten(y_true)
    y_pred_flatten = K.flatten(y_pred)

    intersection = K.sum(y_true_flatten * y_pred_flatten)
    union = K.sum(y_true_flatten) + K.sum(y_pred_flatten)
    return (2 * intersection + smooth) / (union + smooth)

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

def iou_score(y_true, y_pred, smooth=100):
    y_true = K.cast(y_true, 'float32')
    y_pred = K.cast(y_pred, 'float32')
    intersection = K.sum(y_true * y_pred)
    sum = K.sum(y_true + y_pred)
    iou = (intersection + smooth) / (sum - intersection + smooth)
    return iou
```

مقدار smooth برای این است که تقسیم بر 0 در محاسبه معیارها رخ ندهد.

IoU Score (Intersection over Union)

$$IoU = \frac{|A \cap B|}{|A \cup B|}$$

- A: ناحیه پیش‌بینی‌شده (Predicted Region)
- B: ناحیه واقعی یا Ground Truth

- IoU نسبت اشتراک ناحیه پیش‌بینی‌شده با ناحیه واقعی به اجتماع این دو ناحیه را اندازه‌گیری می‌کند. در واقع نشان می‌دهد که چه میزان از ناحیه پیش‌بینی‌شده با ناحیه واقعی همپوشانی دارد.
- به عنوان مثال، اگر مدل بخشی از یک تومور را به اشتباه در نظر نگیرد یا نواحی غیرمرتبط را به عنوان تومور پیش‌بینی کند، IoU کاهش می‌یابد.
- مقادیر این معیار بین 0 و 1 است؛ هرچه مقدار به 1 نزدیک‌تر باشد، دقت پیش‌بینی بالاتر است.

Dice Coefficient (F1 Score for Overlap)

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

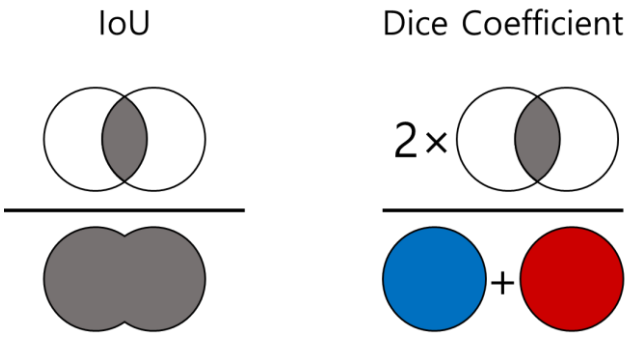
- A: ناحیه پیش‌بینی‌شده (Predicted Region)
- B: ناحیه واقعی یا Ground Truth

- Dice Coefficient دو برابر اندازه اشتراک A و B را به مجموع اندازه‌های آن‌ها نسبت می‌دهد. Dice بیشتر بر همپوشانی تاکید دارد و به عدم تطابق‌های کوچک حساس‌تر از IoU است. در عمل، این معیار معمولا برای ارزیابی دقیق‌تر نواحی که از نظر بالینی اهمیت دارند (مانند مرزهای تومور) استفاده می‌شود.
- مقادیر این معیار نیز بین 0 و 1 است و مقدار 1 نشان‌دهنده همپوشانی کامل است.

تفاوت‌ها و استفاده‌ها

- **IoU Score**: نگاهی کلی‌تر به میزان دقت همپوشانی دارد و برای ارزیابی عملکرد کلی مدل مناسب است.
- **Dice Coefficient**: تاکید بیشتری بر دقت ناحیه مشترک دارد و برای نواحی کوچک‌تر یا حساس‌تر (مانند تصاویر پزشکی) ارجحیت دارد.

شکل 1-7 شهود خوبی از این دو معیار نشان می‌دهد.



شکل 1-7. نحوه محاسبه **Dice Coefficient** و **IoU** در ظاهر

بهینه‌ساز

در این بخش از بهینه‌ساز Adam با learning rate = 0.001 استفاده شده است. این بهینه‌ساز میانگین متحرک gradients (First Moment) و مربع gradients (Second Moment) را نگه می‌دارد و با استفاده از آن‌ها نرخ یادگیری را برای هر پارامتر به صورت تطبیقی تنظیم می‌کند. این روش ترکیبی از مزایای Momentum (برای سرعت‌بخشی به همگرایی) و RMSprop (برای نرخ یادگیری تطبیقی) است. Adam معمولا در مسائل مختلف به دلیل پایداری و سرعت همگرایی بالا عملکرد خوبی دارد و به دلیل تنظیم خودکار نرخ یادگیری، نیاز کمتری به تنظیم دستی هایپرامترها دارد.

تابع هزینه

برای تابع هزینه از Dice Loss استفاده شده است. Dice Loss یک تابع هزینه است که برای مدل‌های سگمنتیشن تصویر به کار می‌رود و هدف آن به حداقل رساندن اختلاف بین ناحیه پیش‌بینی‌شده و ناحیه واقعی (Ground Truth) است. این تابع بر اساس Dice Coefficient ساخته شده که درباره آن گفته شد. هدف مدل این است که Dice Loss را به حداقل برساند، که نشان‌دهنده همپوشانی بیشتر بین پیش‌بینی و داده‌های واقعی است. Dice Loss برای داده‌های نامتوازن (مانند تومورها در تصاویر پزشکی) مناسب است زیرا تمرکز بیشتری بر نواحی کوچک‌تر و مهم‌تر دارد. فرمول Dice Loss به صورت زیر است:

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

مقدار این تابع هزینه بین 0 و 1 است که هرچقدر به 0 نزدیک‌تر باشد، نشان‌دهنده همپوشانی بیشتر بین پیش‌بینی مدل و Ground Truth است.

۱-۵. پیاده‌سازی مدل

ابتدا مدل VGG16 از پیش آموزش شده را از لایبرری keras لود کرده، سپس همه لایه‌های آن را freeze می‌کنیم تا پارامتر های trainable کاهش یابند و از یادگیری انتقالی با استفاده از وزن‌های مدل روی imagenet استفاده کنیم.

```
self.vgg16 = VGG16(include_top=False, weights="imagenet",
input_tensor=self.inputs)
for layer in self.vgg16.layers:
    layer.trainable = False
```

حال باید Skip Connection های مورد نیاز در UNet-VGG16 را از مدل VGG16 استخراج کنیم. لایه‌های مورد نظر که قرار است به عنوان Skip Connection استفاده شوند را بر اساس شکل 1-3 پیدا می‌کنیم.

```
# VGG16 as Encoder
skip1 = self.vgg16.get_layer("block1_conv2").output  ## (256 x 256)
skip2 = self.vgg16.get_layer("block2_conv2").output  ## (128 x 128)
skip3 = self.vgg16.get_layer("block3_conv3").output  ## (64 x 64)
skip4 = self.vgg16.get_layer("block4_conv3").output  ## (32 x 32)
```

بخش بعدی، ایجاد پل برای اتصال انکودر (لایه‌های VGG16) با بخش دیکودر (لایه های UNet) است.

```
# Bridge
bridge = self.vgg16.get_layer("block5_conv3").output  ## (16 x 16)
```

در این قسمت بخش دیکودر (همان لایه‌های UNet)را طراحی می‌کنیم. ساختارهای decoder_block و conv_block را ایجاد می‌کنیم تا پیاده‌سازی تمیزتر انجام شود. decoder_block لایه Skip Connection را به لایه کانوولوشنی مطابق متصل می‌کند تا با استفاده از آن تصاویر بازسازی شوند. همچنین conv_block در واقع دو لایه کانوولوشنی با RelU است که بعد از آنها Batch Normalization قرار دارد.

```
def conv_block(self, input, num_filters):
    x = Conv2D(num_filters, 3, padding="same")(input)
    x = BatchNormalization()(x)
    x = Activation("relu")(x)
    x = Conv2D(num_filters, 3, padding="same")(x)
    x = BatchNormalization()(x)
    x = Activation("relu")(x)
    return x
def decoder_block(self, input, skip_features, num_filters):
    x = Conv2DTranspose(num_filters, (2, 2), strides=2,
padding="same")(input)
    x = Concatenate()([x, skip_features])
    x = self.conv_block(x, num_filters)
    return x
```

حال با استفاده از این موارد، بخش دیکودر را پیاده‌سازی می‌کنیم که در آن Skip Connection ها به لایه‌های decoder block متصل می‌شوند.

```
# Rest of U-Net as Decoder
decoder1 = self.decoder_block(bridge, skip4, 512)    ## (32 x 32)
decoder2 = self.decoder_block(decoder1, skip3, 256) ## (64 x 164)
decoder3 = self.decoder_block(decoder2, skip2, 128)  ## (128 x 128)
decoder4 = self.decoder_block(decoder3, skip1, 64)   ## (256 x 256)
```

در نهایت لایه خروجی را ایجاد می‌کنیم که از تابع فعال‌سازی sigmoid بهره برده است و مدل را بر اساس معیارهای گفته شده در بخش قبل، نهایی می‌کنیم.

```
# Output
outputs = Conv2D(1, 1, padding="same", activation="sigmoid")(decoder4)
self.model = Model(self.inputs, outputs, name="UNet_VGG16")
self.model.compile(Adam(learning_rate=0.001), loss=dice_loss,
   metrics=['accuracy', iou_score, dice_coef])
```

جدول 1-2. لایه‌های مدل نهایی **UNet-VGG16**

Layer (type)	Output Shape	Param #	Connected to	Trai…
input_layer (InputLayer)	(None, 256, 256, 3)	0	-	-
block1_conv1 (Conv2D)	(None, 256, 256, 64)	1,792	input_layer[0…	N
block1_conv2	(None, 256,	36,928	block1_conv1[…	N

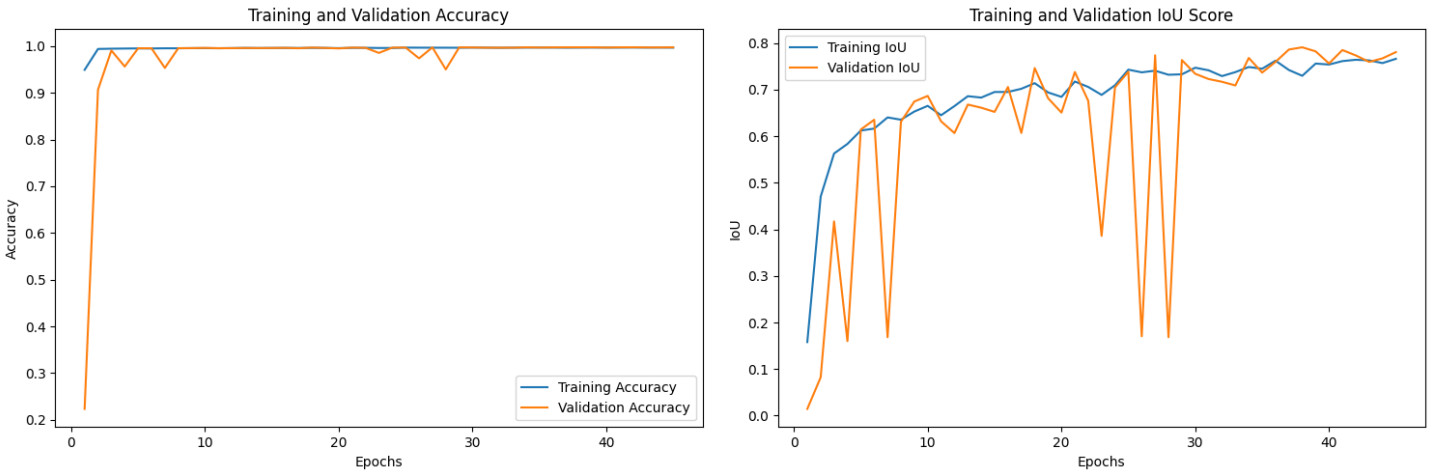
(Conv2D)	256, 64)			
block1_pool (MaxPooling2D)	(None, 128, 128, 64)	0	block1_conv2[…	-
block2_conv1 (Conv2D)	(None, 128, 128, 128)	73,856	block1_pool[0…	N
block2_conv2 (Conv2D)	(None, 128, 128, 128)	147,584	block2_conv1[…	N
block2_pool (MaxPooling2D)	(None, 64, 64, 128)	0	block2_conv2[…	-
block3_conv1 (Conv2D)	(None, 64, 64, 256)	295,168	block2_pool[0…	N
block3_conv2 (Conv2D)	(None, 64, 64, 256)	590,080	block3_conv1[…	N
block3_conv3 (Conv2D)	(None, 64, 64, 256)	590,080	block3_conv2[…	N
block3_pool (MaxPooling2D)	(None, 32, 32, 256)	0	block3_conv3[…	-
block4_conv1 (Conv2D)	(None, 32, 32, 512)	1,180,160	block3_pool[0…	N
block4_conv2 (Conv2D)	(None, 32, 32, 512)	2,359,808	block4_conv1[…	N
block4_conv3 (Conv2D)	(None, 32, 32, 512)	2,359,808	block4_conv2[…	N
block4_pool (MaxPooling2D)	(None, 16, 16, 512)	0	block4_conv3[…	-
block5_conv1 (Conv2D)	(None, 16, 16, 512)	2,359,808	block4_pool[0…	N
block5_conv2 (Conv2D)	(None, 16, 16, 512)	2,359,808	block5_conv1[…	N
block5_conv3 (Conv2D)	(None, 16, 16, 512)	2,359,808	block5_conv2[…	N
conv2d_transpose (Conv2DTranspose)	(None, 32, 32, 512)	1,049,088	block5_conv3[…	Y
concatenate (Concatenate)	(None, 32, 32, 1024)	0	conv2d_transp… block4_conv3[…	-
conv2d (Conv2D)	(None, 32, 32, 512)	4,719,104	concatenate[0…	Y
batch_normalizat… (BatchNormalizat…	(None, 32, 32, 512)	2,048	conv2d[0][0]	Y
activation (Activation)	(None, 32, 32, 512)	0	batch_normali…	-
conv2d_1 (Conv2D)	(None, 32, 32, 512)	2,359,808	activation[0]…	Y
batch_normalizat… (BatchNormalizat…	(None, 32, 32, 512)	2,048	conv2d_1[0][0]	Y
activation_1 (Activation)	(None, 32, 32, 512)	0	batch_normali…	-
conv2d_transpose… (Conv2DTranspose)	(None, 64, 64, 256)	524,544	activation_1[…	Y
concatenate_1 (Concatenate)	(None, 64, 64, 512)	0	conv2d_transp… block3_conv3[…	-
conv2d_2 (Conv2D)	(None, 64, 64, 256)	1,179,904	concatenate_1…	Y
batch_normalizat… (BatchNormalizat…	(None, 64, 64, 256)	1,024	conv2d_2[0][0]	Y
activation_2 (Activation)	(None, 64, 64, 256)	0	batch_normali…	-
conv2d_3 (Conv2D)	(None, 64, 64, 256)	590,080	activation_2[…	Y
batch_normalizat… (BatchNormalizat…	(None, 64, 64, 256)	1,024	conv2d_3[0][0]	Y
activation_3 (Activation)	(None, 64, 64, 256)	0	batch_normali…	-
conv2d_transpose… (Conv2DTranspose)	(None, 128, 128, 128)	131,200	activation_3[…	Y

concatenate_2 (Concatenate)	(None, 128, 128, 256)	0	conv2d_transp… block2_conv2[…	-
conv2d_4 (Conv2D)	(None, 128, 128, 128)	295,040	concatenate_2…	Y
batch_normalizat… (BatchNormalizat…	(None, 128, 128, 128)	512	conv2d_4[0][0]	Y
activation_4 (Activation)	(None, 128, 128, 128)	0	batch_normali…	-
conv2d_5 (Conv2D)	(None, 128, 128, 128)	147,584	activation_4[…	Y
batch_normalizat… (BatchNormalizat…	(None, 128, 128, 128)	512	conv2d_5[0][0]	Y
activation_5 (Activation)	(None, 128, 128, 128)	0	batch_normali…	-
conv2d_transpose… (Conv2DTranspose)	(None, 256, 256, 64)	32,832	activation_5[…	Y
concatenate_3 (Concatenate)	(None, 256, 256, 128)	0	conv2d_transp… block1_conv2[…	-
conv2d_6 (Conv2D)	(None, 256, 256, 64)	73,792	concatenate_3…	Y
batch_normalizat… (BatchNormalizat…	(None, 256, 256, 64)	256	conv2d_6[0][0]	Y
activation_6 (Activation)	(None, 256, 256, 64)	0	batch_normali…	-
conv2d_7 (Conv2D)	(None, 256, 256, 64)	36,928	activation_6[…	Y
batch_normalizat… (BatchNormalizat…	(None, 256, 256, 64)	256	conv2d_7[0][0]	Y
activation_7 (Activation)	(None, 256, 256, 64)	0	batch_normali…	-
conv2d_8 (Conv2D)	(None, 256, 256, 1)	65	activation_7[…	Y

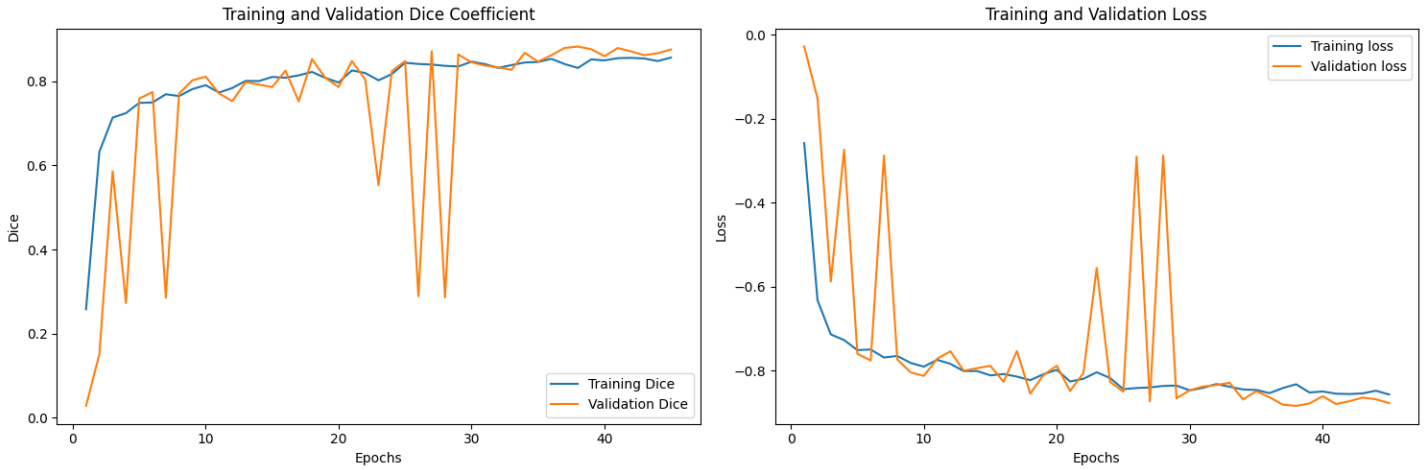
۱-۶. آموزش مدل

حال مدل را با هایپرپارامترهای ذکر شده در جدول 1-3 آموزش می‌دهیم و نتایج را در تصاویر 1-8 و 1-9 گزارش می‌کنیم.

Epochs	45
Batch Size	40
Optimizer	Adam
Learning Rate	0.001
Loss Function	Dice Loss
Metrics	Accuracy, IoU, Dice Coef



شکل 1-8. **IoU** و **Accuray** مدل بر روی داده‌های آموزش و اعتبارسنجی در حین آموزش



شکل 1-9. **Loss** و **Dice Coefficient** مدل روی داده‌های آموزش و اعتبارسنجی

مشاهده می‌شود که مدل پس از 30 epoch به قدرت خوبی رسیده است و Dice Coefficient حدودا 0.85 و IoU Score حدودا 80 می‌دهد. همچنین مدل در دیتای اعتبارسنجی عملکردی کمی بهتری از آموزش داشته است که این به معنای رخ ندادن Overfitting و تعمیم‌پذیری بسیار خوب مدل است که یکی از دلایل آن میتواند Data Augmenation مناسب باشد.

در آخرین ایپاک همچنان مدل درحال افزایش IoU Score و Dice Coefficient است که نشان می‌دهد که شاید بتواند به قدرت بیشتری برسد. اما به دلیل بسیار کم بودن تغییرات، به همین مقدار بسنده شد.

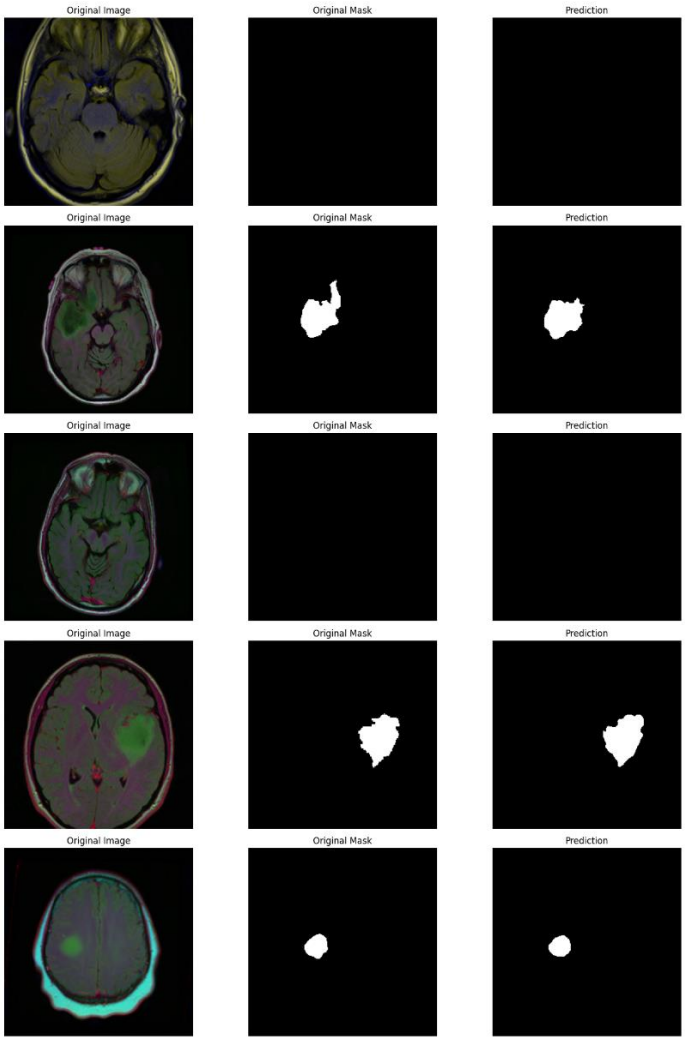
همچنین معیار accuracy پس از چند epoch اول به مقدار بسیار بالای 99 یا 100 میرسد. با بررسی انجام شده، مشخص شد که معیار accuracy در مدل به صورت پیکسل به پیکسل عمل می‌کند (در تسک سگمنتیشن) و مربوط به لیبل تصاویر نیست. به همین دلیل است که accuracy بسیار بالا دریافت کرده‌ایم.

در کل استفاده از معیار accuracy در تسک سگمنتیشن رایج نیست و معمولا از معیارهای دیگری استفاده می‌شود تا عملکرد مدل بررسی شود.

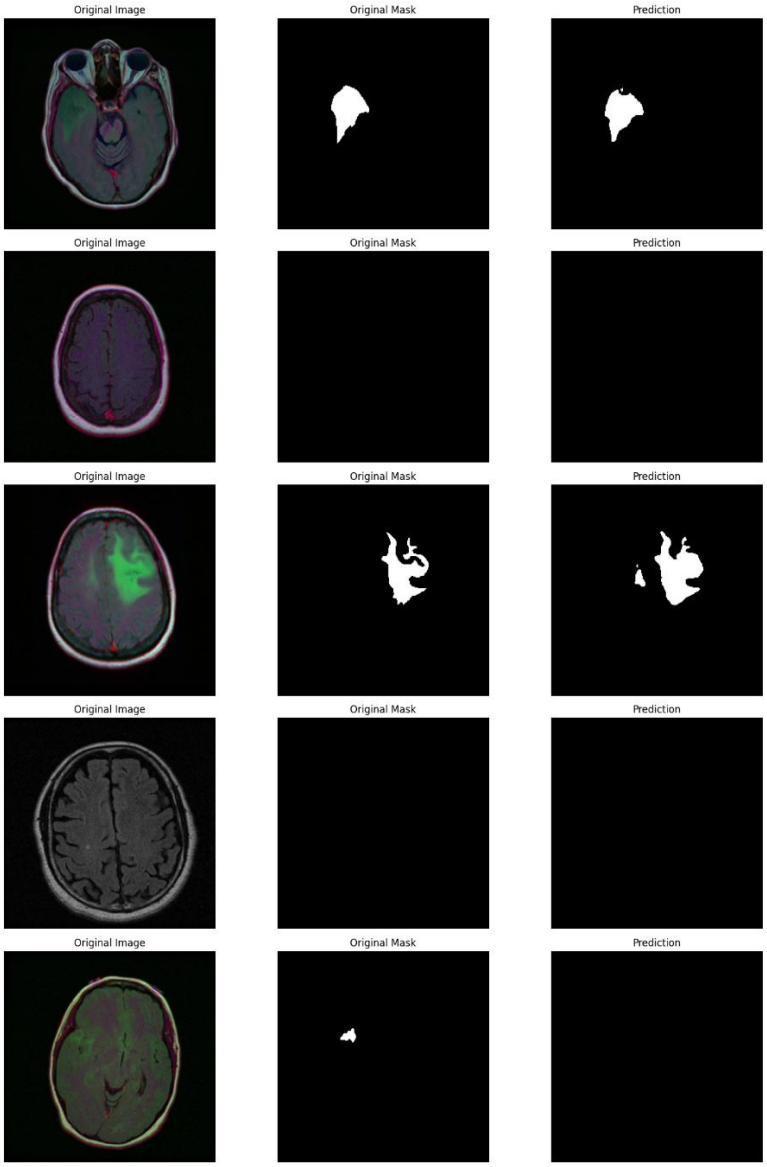
در برخی از epochها، ناگهان چند peak خطای بالا می‌دهند که باعث افت شدید معیارها در آن epochها می‌شود. دلیل واضحی برای این رفتار پیدا نشد و با چندین بار آزمایش متعدد با پارامترهای مختلف، ممکن بود که باز این peakها رخ دهند یا خیر. در کل الگویی در پیدایش این peak ها دیده نشد و باید بررسی‌های دقیق‌تر و حرفه‌ای‌تری بر روی ساختار شبکه و وزن‌ها انجام شود تا دلیل را متوجه شویم.

۱-۷. ارزیابی مدل

در این بخش 10 نمونه از دیتای ارزیابی را به مدل آموزش دیده می‌دهیم تا عملکرد آن را بررسی کنیم. نتایج در شکل‌های 1-10 الف و 1-10ب مشاهده می‌شوند.



شکل 1-10 الف. خروجی مدل بر روی نمونه‌هایی از دیتای ارزیابی



شکل 1-10 ب. خروجی مدل بر روی نمونه‌هایی از دیتای ارزیابی

همانطور که مشاهده می‌شود، مدل به خوبی قسمت های تومور را تشخیص و آن را تولید کرده است. البته در هر کدام از موارد جزییات کمی وجود دارد که در پیش‌بینی مدل اشتباه است که با توجه به مقادیر معیار های بدست آمده، قابل توجیه است. همچنین عملکرد مدل در تشخیص درست نمونه‌هایی که دچار تومور نیستند نیز قابل ملاحظه است.

یکی از اشتباهات مدل، در پیش‌بینی آخرین نمونه رخ داده که تومور بسیار کوچکی را تشخیص نداده است. این مورد نشان می‌دهد که احتمالا هنوز مدل در تشخیص تومورهای کوچک ضعیف است و شاید مدل قابل اتکایی در تشخیص زودهنگام بیماری نباشد. برای افزایش قدرت مدل در تشخیص این موارد،

میتوان دیتای تومورهای کوچک را با انواع روش‌های Over-Sampling و Data Augmentation متنوع‌تر و متعددتر کرد تا مدل این نقطه ضعف را نیز پوشش دهد. در صورتی که با انجام این روش‌ها نیز در تشخیص تومورهای خیلی کوچک ضعف داشت، میتوانیم به تنظیم مجدد هایپرپارامترها از جمله تابع هزینه، بهینه‌ساز و تعداد epoch بپردازیم و راه‌های مختلفی را بررسی کنیم تا بالاخره مدل هم جزییات تومورها را به طور پیشرفته‌تر و دقیق‌تر پیش‌بینی کند، و هم اینکه تومورهای خیلی کوچک را miss نکند.

همچنین با بررسی نمونه سوم در شکل 1-10ب، می‌بینیم که مدل به اشتباه قسمت سبز سمت چپ مغز را نیز به عنوان تومور تشخیص داده است که اشتباه است. با توجه به شباهت قسمت سبز سمت چپ با تصاویر تومور، این امر تقریبا طبیعی است و مدل باید به قدرت بسیار بالایی برسد تا بتواند این موارد را به درستی تشخیص دهد.

پرسش ۲ – تشخیص تابلوهای راهنمایی و رانندگی

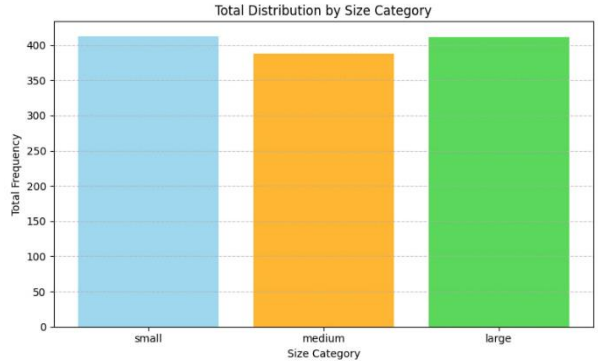
۲-۱. آماده‌سازی مجموعه داده

مجموعه داده‌ی German Traffic Sign Detection Benchmark (GTSDB) یکی از مجموعه داده‌های پرکاربرد در حوزه‌ی تشخیص و شناسایی علائم راهنمایی و رانندگی است. این مجموعه شامل 902 تصویر واقعی از 1206 علائم راهنمایی و رانندگی در شرایط مختلف محیطی است، از جمله تغییرات نوری، پوشش جزئی علائم، و ابعاد کوچک در فواصل دور که در شکل 2-1 نمونه‌هایی از آن نشان داده شده است. تصاویر موجود در این مجموعه همراه با برچسب‌های دقیق هستند که شامل اطلاعاتی مانند مختصات Bounding Box و نوع علامت (مانند محدودیت سرعت یا علائم هشداردهنده) می‌باشند.

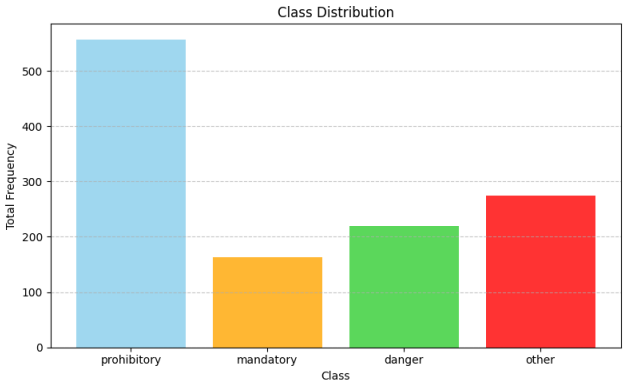
فرمت فایل‌های تصاویر این مجموعه PPM است که یک فرمت ساده و غیر فشرده برای تصاویر رنگی محسوب می‌شود. این فرمت به دلیل ساختار آسان و پشتیبانی گسترده در زبان‌های برنامه‌نویسی و کتابخانه‌های گرافیکی، به‌ویژه در تحقیقات بینایی کامپیوتری، محبوب است.

شکل 2-1. نمونه‌هایی از دیتاست به همراه برچسب‌ها

در مقاله مورد نظر، سه آستانه مختلف برای سایز اشیا تعیین شده است. گروه small با آستانه عرض 32 پیکسل، گروه medium با آستانه عرض 45 پیکسل و گروه large با عرض های بیشتر از 45 پیکسل. شکل 2-2، فراوانی هر یک از این گروه‌ها را نمایش می‌دهد. همچنین شکل 2-3، فراوانی کلاس‌ها در مجموعه داده را نمایش می‌دهد. توجه کنید در اصل دیتا 43 زیر کلاس مختلف دارد که در 4 کلاس اصلی prohibitory، danger، mandatory و other قرار دارند.

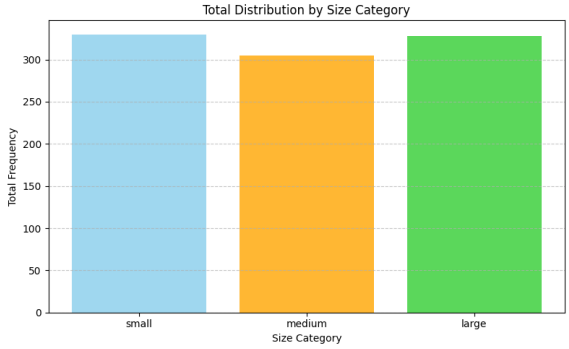


شکل 2-2. نمودار فراوانی دیتا در هر دسته سایز

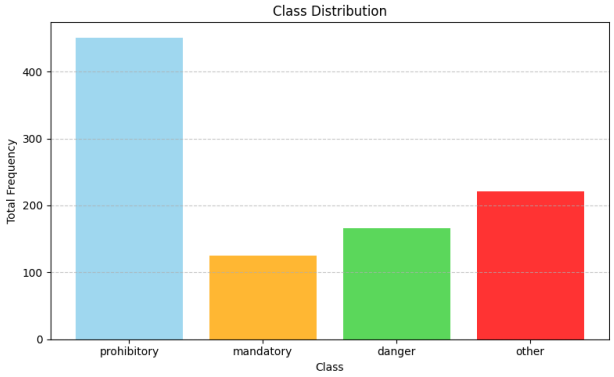


شکل 2-3. نمودار فراوانی دیتا در هر کلاس

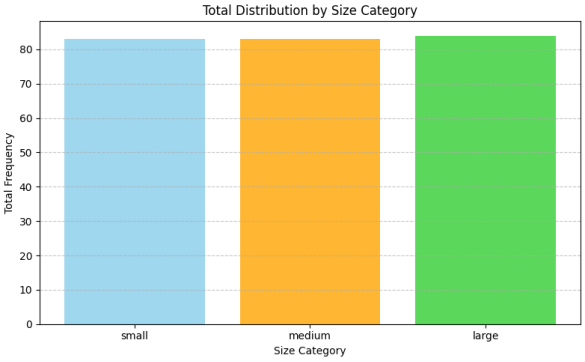
حال مجموعه داده را با نسبت 80-20 به دو دسته آموزش (train) و ارزیابی (test) تقسیم می‌کنیم. شکل های 2-4 و 2-5، نمودارهای فراوانی داده‌های آموزش و ارزیابی در هر دسته سایز و کلاس را نمایش می‌دهند. مشاهده می‌شود که توزیع فراوانی‌های داده‌های آموزش بسیار نزدیک به کل داده است و این نشان می‌دهد که تقسیم دیتا به خوبی انجام شده و توزیع اصلی داده را حفظ کرده است. توزیع داده ارزیابی اهمیت چندانی ندارد و اینکه توزیع یکسانی با دیتای آموزش نداشته باشد میتواند از این نظر بهتر باشد که با توزیع‌های جدیدی که مدل آنها را ندیده است ارزیابی انجام می‌شود تا تعمیم‌پذیری مدل بررسی دقیق‌تری شود.



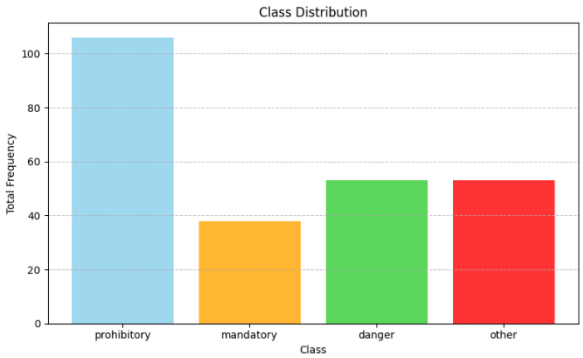
شکل 2- 4 الف. نمودار فراوانی دیتای آموزش در هر دسته سایز



شکل 2-4 ب. نمودار فراوانی دیتای آموزش در هر کلاس



شکل 2-5 الف. نمودار فراوانی دیتای ارزیابی در هر دسته سایز



شکل 2-5 ب. نمودار فراوانی دیتای ارزیابی در هر کلاس

۲-۲. تنظیم دقیق و ارزیابی مدل تشخیص شی دو مرحله‌ای

مدل دو مرحله‌ای Faster R-CNN با شبکه پشتیبان ResNet50-FPN

مدل Faster R-CNN با شبکه پشتیبان ResNet50-FPN از معماری دو مرحله‌ای برای شناسایی و طبقه‌بندی اشیا استفاده می‌کند. در مرحله اول، این مدل با استفاده از یک ماژول به نام Region Proposal Network (RPN) نواحی مستعد حضور اشیا (Region Proposals) را شناسایی می‌کند. شبکه پشتیبان ResNet50-FPN نقش استخراج ویژگی‌های تصویر را در این مدل ایفا می‌کند. ResNet50 یک شبکه عصبی عمیق با 50 لایه است که از بلوک‌های ResNet برای جلوگیری از مشکل ناپدید شدن گرادیان استفاده می‌کند و به استخراج ویژگی‌های قدرتمند کمک می‌کند. از طرف دیگر، Feature Pyramid Network (FPN) به ترکیب ویژگی‌های چند مقیاسی از سطوح مختلف شبکه کمک می‌کند، که این امر برای تشخیص اشیاء در اندازه‌های مختلف بسیار حیاتی است. ترکیب ResNet50 و FPN باعث می‌شود که مدل توانایی خوبی در تشخیص اشیاء کوچک و بزرگ داشته باشد و به بهبود دقت و کارایی تشخیص کمک کند.

دو مرحله‌ای بودن Faster R-CNN از این جهت اهمیت دارد که هر مرحله یک وظیفه خاص را با دقت بالا انجام می‌دهد. مرحله اول (RPN) مسئول یافتن نواحی مستعد حضور اشیاء است و به کاهش تعداد نواحی بررسی شده کمک می‌کند. در مرحله دوم، نواحی انتخاب‌شده به‌طور جداگانه پردازش و طبقه‌بندی می‌شوند، که این امر باعث افزایش دقت تشخیص نهایی می‌شود.

آماده‌سازی مدل

```
from torchvision.models.detection import fasterrcnn_resnet50_fpn
from torchvision.models.detection.faster_rcnn import FastRCNNPredictor

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

num_classes = 4 + 1
model = get_model(num_classes).to(device)
```

برای آماده‌سازی مدل، تابع get_model را تعریف می‌کنیم. ابتدا از مدل از پیش‌آموزش‌دیده شده‌ی fasterrcnn_resnet50_fpn که روی مجموعه داده COCO آموزش دیده است، از کتابخانه torch استفاده می‌شود. وزن‌های از پیش‌آموزش‌دیده شده به مدل بارگذاری می‌شود تا از دانش قبلی مدل در تشخیص اشیا بهره گرفته شود. همچنین در آزمایش ها مشخص شد که اگر وزن‌ها را freeze نکنیم، با اینکه هر

epoch زمان بیشتری نیاز دارد، اما به نتایج بهتری میرسیم نسبت به موقعیتی که freeze کنیم. در بخش بعدی، خروجی‌های پیش‌بینی‌کننده‌ی نهایی مدل برای هماهنگی با تعداد کلاس‌های جدید تنظیم می‌شود. ابتدا تعداد ویژگی‌های ورودی لایه‌ی cls_score (که وظیفه‌ی پیش‌بینی دسته‌بندی کلاس‌ها را دارد) با متغیر in_features ذخیره می‌شود. سپس این لایه با پیش‌بینی‌کننده‌ی FastRCNNPredictor جایگزین می‌شود که تعداد کلاس‌های آن را بر اساس دیتا، ست می‌کنیم. زیرا مدل اصلی برای تعداد کلاس‌های دیتاست COCO که 80 + 1 کلاس پس‌زمینه است طراحی شده و برای کاربرد کنونی، نیاز به تنظیم خروجی مدل متناسب با تعداد کلاس‌های دلخواه داریم. در آخر مدل برای fine tune شدن تنظیم شده است.

آماده‌سازی دیتا برای آموزش

حال باید دیتا را برای آموزش مدل آماده کنیم. ابتدا نیاز است هر کدام از زیر کلاس‌ها را به یکی از چهار کلاس اصلی مپ کنیم.

```
LABEL_TO_CLASS = {
    **dict.fromkeys([0, 1, 2, 3, 4, 5, 7, 8, 9, 10, 15, 16], 1),
    **dict.fromkeys([11, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30,
    31], 2),
    **dict.fromkeys([33, 34, 35, 36, 37, 38, 39, 40], 3),
    **dict.fromkeys([6, 12, 13, 14, 17, 32, 41, 42], 4)
}

CLASS_TO_CATEGORY = {
    1: "prohibitory",
    2: "danger",
    3: "mandatory",
    4: "other"
}
```

سپس کلاس با نام TrafficSignDataset ایجاد می‌کنیم. این کلاس از ماجول Dataset کتابخانه torch استفاده میکند. با استفاده از آن به راحتی میتوان دیتایی که باید به مدل داده شود را از طریق DataLoader تنظیم و آماده کرد. کلاس TrafficSignDataset فایل gt.txt که شامل مختصات Bounding Box های تصاویر و کلاس آنها است را که خود دیتاست فراهم کرده، می‌خواند و براساس آن، برچسب های مورد نیاز هر تصویر دیتاست را آماده می‌کند تا در حین آموزش از آنها استفاده شود. همچنین هر transform ای که نیاز باشد نیز بر روی تصاویر انجام می‌دهد. فرایند نرمال‌سازی نیز یکبار انجام شد که مشاهده شد مدل عملکرد بسیار ضعیفی با آن دارد. به همین دلیل در اجراهای بعدی، نرمال‌سازی بر روی داده‌ها انجام نشده است.

IoU Score (Intersection over Union)

$$IoU = \frac{|A \cap B|}{|A \cup B|}$$

- A: ناحیه پیش‌بینی‌شده (Predicted Region)
- B: ناحیه واقعی یا Ground Truth

- IoU نسبت اشتراک ناحیه پیش‌بینی‌شده با ناحیه واقعی به اجتماع این دو ناحیه را اندازه‌گیری می‌کند. در واقع نشان می‌دهد که چه میزان از ناحیه پیش‌بینی‌شده با ناحیه واقعی همپوشانی دارد.
- به عنوان مثال، اگر مدل بخشی از یک تومور را به اشتباه در نظر نگیرد یا نواحی غیرمرتبط را به عنوان تومور پیش‌بینی کند، IoU کاهش می‌یابد.
- مقادیر این معیار بین 0 و 1 است؛ هرچه مقدار به 1 نزدیک‌تر باشد، دقت پیش‌بینی بالاتر است.

mAP (Mean Average Precision)

- mAP معیار میانگین دقت (Precision) را در سطوح مختلف Recall برای همه‌ی کلاس‌های موجود در داده محاسبه می‌کند. محاسبه‌ی mAP ابتدا با محاسبه‌ی مقدار Average Precision (AP) برای هر کلاس انجام می‌شود. AP میانگین دقت در سطوح مختلف فراخوانی است که با رسم نمودارPrecision-Recall Curve و محاسبه‌ی مساحت زیر این منحنی به دست می‌آید. سپس مقدار AP برای تمام کلاس‌ها میانگین‌گیری می‌شود تا مقدار نهایی mAP به دست آید.
- مقادیر mAP بین 0 و 1 است؛ مقدار بالاتر نشان‌دهنده‌ی عملکرد بهتر مدل است. اگر مدل اشیارا با دقت پایین شناسایی کند یا اشتباه دسته‌بندی کند، مقدار mAP کاهش می‌یابد.

بهینه‌ساز و تابع هزینه

برای بهینه‌ساز از تابع AdamW استفاده شده است. AdamW یک نسخه بهبودیافته از Adam است که به‌طور خاص برای رفع برخی از مشکلات مربوط به Weight Decay طراحی شده است. در بهینه‌ساز Adam، وزن‌ها و مقادیر گرادیان‌ها به‌روزرسانی می‌شوند، اما معمولا اثر نرخ یادگیری تطبیقی و اضافه شدن وزن‌ها در محاسبات، تاثیر منفی بر عملکرد مدل می‌گذارد. AdamW این مشکل را با جدا کردن فرآیند وزن‌دهی و به‌روزرسانی پارامترها حل می‌کند.AdamW نیز مانند Adam از میانگین متحرک گرادیان‌ها (First Moment) و مربع گرادیان‌ها (Second Moment) برای تطبیق نرخ یادگیری استفاده می‌کند. تفاوت اصلی این است که در AdamW، نرخ جریمه‌ی وزن‌ها به صورت مستقیم و مستقل از به‌روزرسانی گرادیان‌ها اعمال می‌شود. این بهبود منجر به نتایج بهتر در مسائل با fine tune مدل‌های پیچیده می‌شود.

برای تابع هزینه در بخش Classification از Negetive Log Likelihood Loss و برای Regression از تابع Smooth L1 Loss استفاده شده است. (کلاس fasterrcnn_resnet50_fpn در torch به طور دیفالت از این توابع استفاده می‌کند و در پیاده‌سازی ما آنها را تغییر ندادیم.)

NLL Loss به مدل کمک می‌کند احتمال پیش‌بینی صحیح کلاس هدف (Ground Truth) را حداکثر کند. فرمول آن به صورت $NLLLoss = -log(p_y)$ است، که در آن p_y احتمال پیش‌بینی‌شده برای کلاس صحیح است. هرچه این احتمال بزرگ‌تر باشد، مقدار NLL Loss کمتر خواهد بود. در Object Detection، NLL Loss برای بخش Classification استفاده می‌شود. مدل باید برای هر ناحیه پیش‌بینی‌شده، کلاس درست را شناسایی کند و در اینجا NLL Loss به کاهش خطای پیش‌بینی کلاس‌ها کمک می‌کند.

تابع Smooth L1 Loss برای بهبود پیش‌بینی مختصات Bounding Box استفاده می‌شود. مدل باید موقعیت دقیق جعبه‌ها را اصلاح کند تا با جعبه‌های واقعی تطابق داشته باشد. Smooth L1 Loss ترکیبی از L1 Loss و L2 است و نسبت به خطاهای بزرگ حساسیت کمتری دارد.

خطای نهایی مدل از فرمول زیر استفاده می‌کند:

$$Total\ Loss = Classification\ Loss + \lambda \times Regression\ Loss$$

که در اینجا λ معمولا 1 است.

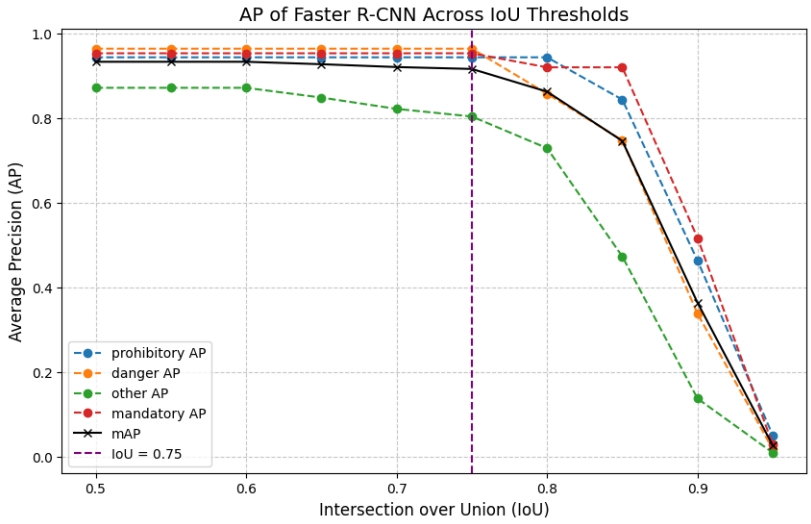
تنظیم دقیق مدل (fine tune)

حال مدل را باید آموزش دهیم. تعداد epoch را برابر 10 قرار می‌دهیم. برای ارزیابی مدل، از آستانه‌های IoU مختلف (از 0.5 تا 0.95) استفاده می‌شود که به مدل امکان می‌دهد در سطوح مختلف دقت در همپوشانی ارزیابی شود. هر چه این آستانه بالاتر باشد، مدل باید همپوشانی دقیق‌تری داشته باشد تا پیش‌بینی آن به عنوان مثبت واقعی (True Positive) در نظر گرفته شود. خطا و mAP-50 و mAP-90 مدل در شکل 2-6 قابل مشاهده است.

```
100%|██████████| 296/296 [01:43<00:00,  2.86it/s]
Epoch 1/10
Training Loss: 0.1873
Validation Metrics: {'mAP50': 0.3325891279536645, 'mAP50-95': 0.23445813000302188}
100%|██████████| 296/296 [01:43<00:00,  2.86it/s]
Epoch 2/10
Training Loss: 0.0878
Validation Metrics: {'mAP50': 0.31678906402265905, 'mAP50-95': 0.25617676853333543}
100%|██████████| 296/296 [01:43<00:00,  2.86it/s]
Epoch 3/10
Training Loss: 0.0610
Validation Metrics: {'mAP50': 0.5873697505556645, 'mAP50-95': 0.4909892866775494}
100%|██████████| 296/296 [01:43<00:00,  2.86it/s]
Epoch 4/10
Training Loss: 0.0538
Validation Metrics: {'mAP50': 0.7415705055318895, 'mAP50-95': 0.5910085547690423}
100%|██████████| 296/296 [01:43<00:00,  2.86it/s]
Epoch 5/10
Training Loss: 0.0432
Validation Metrics: {'mAP50': 0.8295570016170326, 'mAP50-95': 0.6864882504580004}
100%|██████████| 296/296 [01:43<00:00,  2.86it/s]
Epoch 6/10
Training Loss: 0.0426
Validation Metrics: {'mAP50': 0.8826928876799824, 'mAP50-95': 0.7669872694567546}
100%|██████████| 296/296 [01:43<00:00,  2.86it/s]
Epoch 7/10
Training Loss: 0.0377
Validation Metrics: {'mAP50': 0.8222423753593832, 'mAP50-95': 0.7003844453384412}
100%|██████████| 296/296 [01:43<00:00,  2.86it/s]
Epoch 8/10
Training Loss: 0.0337
Validation Metrics: {'mAP50': 0.8268551109805223, 'mAP50-95': 0.6950258914656287}
100%|██████████| 296/296 [01:43<00:00,  2.86it/s]
Epoch 9/10
Training Loss: 0.0318
Validation Metrics: {'mAP50': 0.8902352094109554, 'mAP50-95': 0.7747137560540398}
100%|██████████| 296/296 [01:43<00:00,  2.86it/s]
Epoch 10/10
Training Loss: 0.0299
Validation Metrics: {'mAP50': 0.9021985147301473, 'mAP50-95': 0.7826386432119664}
Training and evaluation completed.
```

شکل 2-6. خطای داده آموزشی و **mAP-50** و **mAP-95** مدل دو مرحله ای بر روی داده ارزیابی

برای محاسبه mAP با آستانه‌های مختلف IoU، AP برای هر آستانه IoU محاسبه می‌شود که از منحنی‌های (Precision-Recall) به دست می‌آید. سپس میانگین این APها در سطوح مختلف IoU محاسبه می‌شود تا mAP نهایی به دست آید. استفاده از آستانه‌های مختلف IoU به ارزیابی دقیق‌تر مدل کمک می‌کند و می‌توان فهمید که مدل در پیش‌بینی‌های عمومی و همچنین در دقت بالای محلی‌سازی چقدر موفق بوده است. در شکل 2-7، نمودار AP به ازای هر آستانه برای هر کلاس دیتا به طور جداگانه و همچنین mAP رسم شده است.



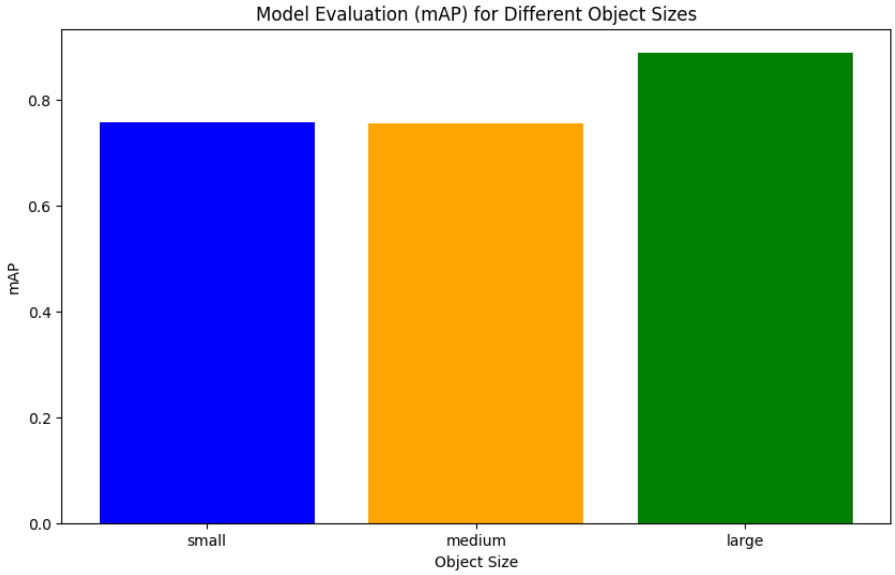
شکل 2-7. نمودار **AP** بر اساس آستانه **IoU** های مختلف برای هر کلاس دیتا در مدل دو مرحله ای

با شروع از IoU های پایین، مشاهده می‌شود که AP برای همه کلاس‌ها بسیار بالا است و تقریبا به مقدار 1 نزدیک می‌شود. این موضوع نشان‌دهنده این است که مدل در آستانه‌های پایین به‌خوبی می‌تواند اشیا را تشخیص داده و با واقعیت تطبیق دهد. با این حال، این عملکرد ایده‌آل ممکن است به این دلیل باشد که IoU پایین به مدل اجازه می‌دهد پیش‌بینی‌های نسبتا نادقیق را نیز صحیح در نظر بگیرد.

وقتی IoU افزایش می‌یابد (به سمت 0.75 و بالاتر)، نمودارها نشان‌دهنده کاهش تدریجی دقت هستند. این کاهش بیانگر سخت‌تر شدن معیارها است، زیرا مدل باید پیش‌بینی‌هایی دقیق‌تر و با همپوشانی بالاتر با جعبه‌های واقعی ارائه دهد. این روند کاهش نشان می‌دهد که مدل در IoU های بالاتر برای برخی کلاس‌ها مانند "danger" یا "other" با چالش‌های بیشتری مواجه می‌شود، در حالی که کلاس‌هایی مانند "prohibitory"و "mandatory" عملکرد پایدارتر و قوی‌تری دارند.

عملکرد کلی مدل (mAP) نیز نشان می‌دهد که اگرچه مدل به‌طور کلی در IoU های پایین‌تر قوی عمل می‌کند، اما کارایی آن با افزایش IoU کاهش می‌یابد. این موضوع نشان می‌دهد که مدل ممکن است برای تشخیص‌های دقیق‌تر نیاز به بهبودهایی در ساختار شبکه یا تنظیمات داده داشته باشد. همچنین تفاوت رفتار بین کلاس‌ها احتمالا به پیچیدگی یا شباهت ظاهری برخی کلاس‌ها در داده‌های آموزشی مرتبط است.

در شکل 2-8، عملکرد مدل با ارزیابی نسبت به سایزهای مختلف قرار داده شده است.



شکل 2-8. **mAP** مدل دو مرحله‌ای برای اشیا با اندازه‌های متفاوت

این ارزیابی mAP نسبت به آستانه با IoU=0.5 انجام شده است. همانطور که در شکل 2-6 هم مشاهده شد، مدل با این آستانه عملکرد بسیار خوبی دارد، به همین دلیل در هر سه اندازه اشیا، به mAP های بالایی رسیده است. نتایج این ارزیابی نشان می‌دهد که مدل روی اشیای بزرگ عملکرد بسیار بهتری نسبت به اشیای متوسط و کوچک دارد. دلیل این تفاوت را می‌توان در چند عامل جستجو کرد. اول اینکه اشیای بزرگ‌تر جزئیات بیشتری را به مدل ارائه می‌دهند، بنابراین استخراج ویژگی‌ها برای شبکه ساده‌تر است. دوم، هرچند معماری‌هایی مثل ResNet50-FPN برای پردازش اشیای کوچک تا بزرگ طراحی شده‌اند، اما در عمل، اشیای کوچک ممکن است در مراحل پایین‌تر شبکه و به‌ویژه در لایه‌های با رزولوشن کمتر، بخشی از اطلاعات خود را از دست بدهند. علاوه بر این، در بسیاری از مجموعه‌داده‌ها، تعداد اشیای کوچک کمتر از متوسط یا بزرگ است. این عدم تعادل داده‌ها می‌تواند یکی از دلایلی باشد که مدل برای یادگیری و تشخیص این اشیا دقت کمتری داشته باشد. همچنین، نویزهای پس‌زمینه یا وجود اشیای مشابه در تصاویر می‌توانند روی عملکرد مدل در تشخیص اشیای کوچک اثر منفی بیشتری بگذارند.

ارزیابی مدل

در این بخش یک نمونه از دیتای ارزیابی را به عنوان ورودی مدل دو مرحله‌ای داده و خروجی پیش‌بینی شده را نمایش می‌دهیم.

شکل 2-9. نمونه پیش‌بینی شده توسط مدل تنظیم شده دو مرحله‌ای

مشاهده می‌شود که مدل با confidence عالی هر سه تابلو را تشخیص داده است.

۲-۳: تنظیم دقیق و ارزیابی مدل تشخیص شی تک مرحله‌ای

مدل تک مرحله‌ای SSD300 با شبکه پشتیبان VGG16

مدل Single Shot Multibox Detector (SSD300) با شبکه پشتیبان VGG16 یکی از الگوریتم‌های پرکاربرد در تشخیص اشیاء است که به صورت تک‌مرحله‌ای عمل می‌کند. این مدل به دلیل سرعت بالا و معماری ساده‌اش در کاربردهای Real-Time محبوب است. SSD بر خلاف مدل‌های دومرحله‌ای مانند Faster R-CNN، تمام مراحل تولید Region Proposals و طبقه‌بندی اشیا را به‌طور همزمان در یک مرحله انجام می‌دهد.

شبکه پشتیبان VGG16 در SSD300 وظیفه استخراج ویژگی‌های تصویر را بر عهده دارد. VGG16 یک شبکه عصبی عمیق با ساختار ساده و لایه‌های کانولوشن متوالی است که به دلیل دقت بالا در استخراج ویژگی‌ها، همچنان در بسیاری از مدل‌ها مورد استفاده قرار می‌گیرد. در SSD، لایه‌های پایانی VGG16 با لایه‌های اضافی برای استخراج ویژگی‌ها جایگزین می‌شوند. این لایه‌های اضافی به مدل امکان می‌دهند که اشیا را در اندازه‌های مختلف شناسایی کند و در هر مقیاس طبقه‌بندی دقیقی انجام دهد.

مزیت اصلی SSD300 در مقایسه با مدل‌های دومرحله‌ای، سرعت بالای آن است، زیرا نیازی به مراحل مجزا برای پیشنهاد و طبقه‌بندی نواحی ندارد. با این حال، دقت SSD در شناسایی اشیا کوچک معمولا کمتر از مدل‌های دومرحله‌ای است، زیرا این مدل به اندازه آنها در مدیریت اشیا کوچک دقیق نیست. با این

وجود، در بسیاری از موارد، SSD با شبکه پشتیبان VGG16 انتخاب مناسبی برای کاربردهایی است که زمان پاسخ‌دهی سریع اولویت دارد.

همچنین تصاویر ورودی باید سایز 300 * 300 باشند.

آماده‌سازی مدل

```
def create_SSD_model(num_classes=5, size=300):
    model = torchvision.models.detection.ssd300_vgg16(
        weights=SSD300_VGG16_Weights.COCO_V1
    )
    in_channels = _utils.retrieve_out_channels(model.backbone,
(size, size))
    num_anchors = model.anchor_generator.num_anchors_per_location()
    model.head.classification_head = SSDClassificationHead(
        in_channels=in_channels,
        num_anchors=num_anchors,
        num_classes=num_classes,
    )
    model.transform.min_size = (size,)
    model.transform.max_size = size
    return model
```

با این کد مدل ssd300_vgg16 از کتابخانه torch بارگذاری می‌شود که با وزن‌های از پیش آموزش دیده بر روی دیتاست COCO است. سپس تعداد چنل ورودی و anchor های مورد نیاز را استخراج کرده و به head جدید که از کلاس SSDClassificationHead است میدهیم. همچنین تعداد کلاس‌ها با توجه به پروژه قرار میدهیم. در این قسمت نیز freeze انجام نمیدهیم چون با انجام آن، نتایج به شدت افت می‌کردند و نیاز است که وزن‌ها باز fine tune شوند.

معیارها، بهینه‌ساز و تابع هزینه

معیارهای استفاده شده و بهینه‌ساز مانند بخش قبل هستند. برای معیار از IoU و mAP، بهینه ساز از AdamW.

برای تابع هزینه، ssd300-vgg16 به طور دیفالت از یک تابع هزینه چندمنظوره استفاده می‌کند که شامل دو بخش اصلی است: هزینه مکان‌یابی و هزینه اطمینان.

بخش هزینه مکان‌یابی (Localization Loss) هزینه دقت پیش‌بینی مختصات Bounding Boxها را نسبت به مختصات واقعی اندازه‌گیری می‌کند. برای هر تطابق مثبت (که در آن جعبه پیش‌بینی‌شده به‌اندازه کافی با جعبه حقیقت زمینی همپوشانی دارد)، این تابع اختلاف بین مختصات پیش‌بینی‌شده و مختصات واقعی را جریمه می‌کند. از Smooth L1 Loss برای این بخش استفاده می‌شود

بخش هزینه اطمینان (Confidence Loss) هزینه دقت پیش‌بینی کلاس‌ها برای هر جعبه را اندازه‌گیری می‌کند. مدل برای هر جعبه پیش‌بینی‌شده، احتمال وجود یک شیء (کلاس مثبت) یا پس‌زمینه (کلاس منفی) را پیش‌بینی می‌کند. برای این منظور از Cross Entropy Loss استفاده می‌شود.

فرمول نهایی تابع هزینه SSD300_VGG16 به صورت زیر است

$$Total\ Loss = \frac{1}{N}\left(\alpha \times L_{loc} + L_{conf}\right)$$

که α وزن‌دهی میان دو بخش هزینه است که معمولا 1 داده می‌شود.

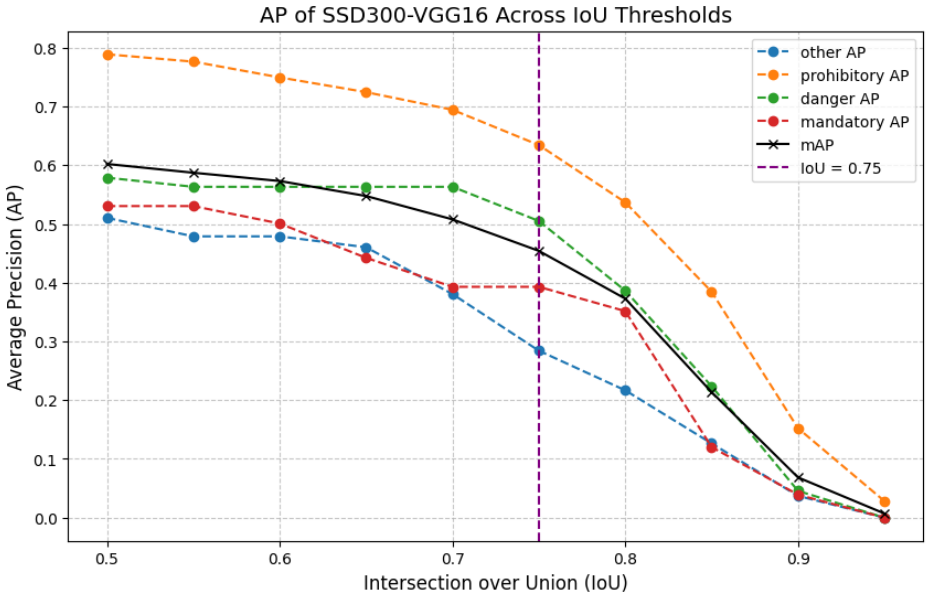
تنظیم دقیق مدل (fine tune)

در این بخش مانند بخش به fine tune مدل میپردازیم. در اینجا به دلیل کوچکتر بودن مدل نیاز است که تعداد epoch را بیشتر بگذاریم تا مدل به دقت بهتری برسد. نتایج خطا و mAP ها در چند epoch نهایی در شکل 2-10 آورده شده است.

```
100%|██████████| 296/296 [00:22<00:00, 12.93it/s]
Epoch 32/40
Training Loss: 0.0454
Validation Metrics: {'mAP50': 0.65026002161135, 'mAP50-95': 0.46237344621061877}
100%|██████████| 296/296 [00:22<00:00, 12.92it/s]
Epoch 33/40
Training Loss: 0.0484
Validation Metrics: {'mAP50': 0.6511641731814415, 'mAP50-95': 0.4574592038113999}
100%|██████████| 296/296 [00:23<00:00, 12.85it/s]
Epoch 34/40
Training Loss: 0.0534
Validation Metrics: {'mAP50': 0.6597212093755259, 'mAP50-95': 0.468917048329975}
100%|██████████| 296/296 [00:23<00:00, 12.76it/s]
Epoch 35/40
Training Loss: 0.0652
Validation Metrics: {'mAP50': 0.658882297812296, 'mAP50-95': 0.46049518665711353}
100%|██████████| 296/296 [00:22<00:00, 12.92it/s]
Epoch 36/40
Training Loss: 0.0784
Validation Metrics: {'mAP50': 0.632623947205422, 'mAP50-95': 0.4455045240387421}
100%|██████████| 296/296 [00:22<00:00, 12.89it/s]
Epoch 37/40
Training Loss: 0.1057
Validation Metrics: {'mAP50': 0.6711132843336138, 'mAP50-95': 0.46956449134537837}
100%|██████████| 296/296 [00:22<00:00, 13.04it/s]
Epoch 38/40
Training Loss: 0.1106
Validation Metrics: {'mAP50': 0.6066638974911941, 'mAP50-95': 0.4244410003966733}
100%|██████████| 296/296 [00:22<00:00, 12.96it/s]
Epoch 39/40
Training Loss: 0.8185
Validation Metrics: {'mAP50': 0.40908940324018994, 'mAP50-95': 0.27656109544179786}
100%|██████████| 296/296 [00:22<00:00, 12.90it/s]
Epoch 40/40
Training Loss: 0.1955
Validation Metrics: {'mAP50': 0.6526649322119941, 'mAP50-95': 0.4503174782949406}
Training and evaluation completed.
```

شکل 2-10. خطای داده آموزشی و **mAP-50** و **mAP-95** مدل تک مرحله ای بر روی داده ارزیابی

در شکل 2-11، نمودار AP برای هر کلاس آورده شده است.

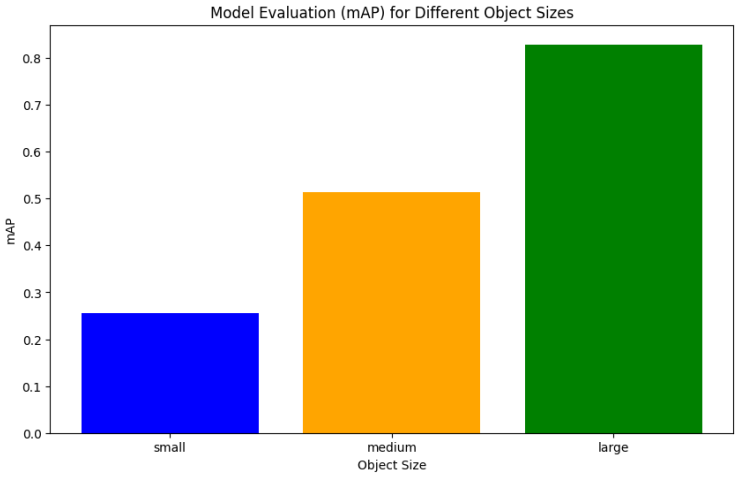


شکل 2-11. نمودار **AP** بر اساس آستانه **IoU** های مختلف برای هر کلاس دیتا در مدل تک مرحله ای

مشاهده می‌شود که AP ها و به طور کلی mAP با افزایش آستانه IoU، کاهش میابند که طبق تحلیل بخش قبل، طبیعی است. البته کلاس "danger" عملکرد با ثبات‌تری نسبت به به بقیه دارد.

برای مثال در IoU=0.65 مشاهده میشود که AP کلاس danger از mAP بیشتر میشود، که این نشاندهنده این است که بقیه کلاس در حال ضعیف شدن عملکرد هستند، در صورتی که این کلاس هنوز عملکرد خود را حفظ کرده است.

در تصویر 2-12، عملکرد مدل بر اساس اندازه های اشیا قرار داده شده است.



شکل 2-12. **mAP** مدل تک مرحله‌ای برای اشیا با اندازه‌های متفاوت

در این ارزیابی مشخص است که مدل SSD300 با شبکه پشتیبان VGG16 روی اشیای کوچک و حتی متوسط عملکرد قابل قبولی نداشته است. این ضعف به معماری مدل و محدودیت‌های آن برمی‌گردد.

عملکرد پایین روی اشیای متوسط می‌تواند دلایل مختلفی داشته باشد. اول اینکه، Anchorهای پیش‌فرض در SSD ممکن است به‌درستی برای مقیاس اشیای متوسط تنظیم نشده باشند. دوم، اشیای متوسط معمولاً حالتی بینابینی دارند، نه به اندازه اشیای بزرگ ویژگی‌های آشکاری دارند و نه به اندازه اشیای کوچک چالش‌برانگیز هستند، اما در عین حال ممکن است به دلیل اشتراک با پس‌زمینه یا اشیای مشابه، مدل را به اشتباه بیندازند.

البته عملکرد SSD روی اشیای بزرگ بسیار بهتر بوده است. این اشیا به دلیل اندازه بزرگ‌ترشان اطلاعات بیشتری در اختیار مدل قرار می‌دهند و مدل می‌تواند با دقت بالاتری آنها را تشخیص دهد. این موضوع نشان می‌دهد که مدل SSD در مواجهه با اشیای بزرگ عملکرد خوبی دارد، اما برای بهبود دقت در اشیای کوچک و متوسط نیاز به اصلاحات و بهینه‌سازی دارد.

ارزیابی مدل

در این بخش یک نمونه از دیتای ارزیابی را به عنوان ورودی مدل تک مرحله‌ای داده و خروجی پیش‌بینی شده را نمایش می‌دهیم.

شکل 2-13. نمونه پیش‌بینی شده توسط مدل تنظیم شده دو مرحله‌ای

مشاهده می‌شود که مدل با confidence عالی هر سه تابلو را تشخیص داده است.

۲-۴: ارزیابی نتایج و مقایسه مدل‌ها

در ابتدا مقایسه‌ای بین مدل‌های دو مرحله‌ای و تک مرحله‌ای انجام می‌دهیم تا نتایج موجه‌تر باشند.

روش‌های دو مرحله‌ای و تک مرحله‌ای در تشخیص اشیا دو رویکرد متفاوت با مزایا و معایب خاص خود ارائه می‌دهند. در روش‌های دو مرحله‌ای، مانند Faster R-CNN، فرآیند شناسایی به دو بخش مجزا تقسیم

می‌شود: ابتدا Region Proposals شناسایی می‌شوند و سپس این نواحی برای طبقه‌بندی و مکان‌یابی دقیق پردازش می‌شوند. این ساختار باعث می‌شود که مدل بتواند با تمرکز بر روی نواحی مهم، دقت بالاتری در تشخیص و مکان‌یابی اشیا داشته باشد. مدل‌های دو مرحله‌ای برای داده‌هایی با اشیا کوچک، پیچیدگی بالا یا نواحی با همپوشانی زیاد مناسب هستند، زیرا هر مرحله از فرآیند به‌طور خاص برای بهبود دقت طراحی شده است. با این حال، این روش‌ها به دلیل نیاز به دو مرحله پردازش، معمولا زمان‌برتر و سنگین‌تر از نظر محاسباتی هستند.

در مقابل، روش‌های تک مرحله‌ای مانند SSD تمام فرآیند شناسایی، از تولید نواحی پیشنهادی تا طبقه‌بندی و مکان‌یابی، را به صورت یکپارچه و همزمان انجام می‌دهند. این رویکرد باعث افزایش سرعت پردازش و کاهش پیچیدگی معماری می‌شود و آنها را به گزینه‌ای ایده‌آل برای کاربردهای Real-Time تبدیل می‌کند. با این حال، از آنجا که این روش‌ها به جای تمرکز بر نواحی خاص، تمام تصویر را پردازش می‌کنند، دقت آنها معمولا کمتر از مدل‌های دو مرحله‌ای است، به‌ویژه در شناسایی اشیا کوچک یا زمانی که تصویر دارای پیچیدگی زیادی باشد.

مقایسه مدل‌ها در شناسایی علائم در کلاس‌های مختلف

در این آزمایش، دو مدل Faster R-CNN-ResNet-FPN و SSD300-VGG16 روی یک دیتاست مشخص ارزیابی شدند. نتایج نشان داد که مدل دو مرحله‌ای Faster R-CNN عملکرد بهتری از نظر دقت در تشخیص و مکان‌یابی اشیا نسبت به مدل تک مرحله‌ای SSD300 داشته است. همانطور که در شکل 2-6 مشاهده شد، مدل دو مرحله‌ای تا آستانه IoU بالایی، دارای mAP بسیار خوبی است (0.89) که این نشان‌دهنده قدرت مدل در تشخیص است. اما خب به دلیل دو مرحله‌ای بودن مدل، همانطور که گفته شد، بسیار کندتر عمل می‌کند و آموزش دیدن آن با پیچیدگی زیادی همراه است.

در مقابل، مدل SSD300 با ساختاری تک مرحله‌ای تمام مراحل پیش‌بینی را به‌صورت یکپارچه و سریع انجام می‌دهد. همانطور که در شکل 2-9 مشاهده می‌شود، به دقت کمتری نسبت به مدل دو مرحله‌ای رسید که دلایل آن گفته شد. اما از نظر سرعت پردازش عملکرد بهتری داشت و فرایند آموزش آن سریع‌تر انجام شد. معماری ساده‌تر SSD300 آن را به گزینه‌ای مناسب برای کاربردهایی با نیاز به پاسخ‌دهی سریع، مانند سیستم‌های بلادرنگ یا پروژه‌هایی با محدودیت منابع محاسباتی، تبدیل می‌کند.

البته یکی از موارد قابل توجه این است که ترتیب مقدار AP درکلاس های مختلف بین دو مدل دو مرحله‌ای و تک مرحله‌ای یکسان است که این نشان دهنده روند اموزش تقریبا یکسان مدل ها بر روی این کلاس ها میباشد.

همچنین در مدل تک مرحله ای، AP ها به جز کلاس "danger" از همان ابتدای افزایش آستانه در حال کاهش یافتن هستند، اما در مدل دو مرحله‌ای تا مقدار خوبی از آستانه IoU ثابت بودند و مقدار بالایی داشتند که این باز هم ثبات و قدرت بیشتر مدل دو مرحله ای را نشان میدهد.

مقایسه مدل‌ها در شناسایی علائم با اندازه‌های مختلف

مدل‌های Faster R-CNN با ResNet50-FPN و SSD300 با VGG16 در برخورد با اشیای کوچک، متوسط و بزرگ عملکردهای متفاوتی از خود نشان می‌دهند که این تفاوت‌ها به طراحی معماری آنها بازمی‌گردد. مدل Faster R-CNN به دلیل معماری دو مرحله‌ای و استفاده از شبکه FPN عملکرد یکنواخت‌تری روی همه مقیاس‌ها دارد، در حالی که SSD300 به دلیل معماری تک‌مرحله‌ای خود، در اشیای کوچک و متوسط ضعف قابل توجهی نشان می‌دهد.

در مواجهه با اشیای کوچک، Faster R-CNN عملکرد بسیار بهتری دارد. فرآیند Region Proposal به این مدل کمک می‌کند تا اشیای کوچک را شناسایی کند، زیرا این فرآیند مناطقی با احتمال بالاتر از وجود اشیا را مشخص کرده و دقت شناسایی را افزایش می‌دهد. همچنین، استفاده از FPN باعث تقویت ویژگی‌های مقیاس کوچک می‌شود. از سوی دیگر، SSD300 به دلیل نداشتن چنین مکانیسمی و استفاده از Anchorهای پیش‌فرض که برای اندازه‌های بزرگ‌تر بهینه شده‌اند، در شناسایی اشیای کوچک با ضعف جدی مواجه است. همین امر باعث می‌شود که عملکرد آن در این بخش بسیار پایین‌تر از Faster R-CNN باشد.

برای اشیای متوسط، Faster R-CNN باز هم برتری خود را نشان می‌دهد. ترکیب شبکه ResNet50 با FPNباعث می‌شود که ویژگی‌های اشیای متوسط نیز به خوبی استخراج شوند و مدل بتواند این اشیا را با دقت بالایی شناسایی کند. در مقابل، SSD300 اگرچه نسبت به اشیای کوچک عملکرد بهتری دارد، اما همچنان نتوانسته است به دقت کافی در این بخش دست یابد. معماری تک‌مرحله‌ای SSD وAnchor های ثابت آن باعث شده‌اند که در مقیاس متوسط نیز عملکردی ضعیف‌تر از Faster R-CNN داشته باشد.

اما در شناسایی اشیای بزرگ، هر دو مدل عملکرد خوبی دارند. اشیای بزرگ به دلیل ویژگی‌های آشکارتری که دارند، برای هر دو مدل آسان‌تر هستند. با این حال، Faster R-CNN همچنان عملکرد یکنواخت‌تری در این مقیاس ارائه می‌دهد، در حالی که SSD300 نیز به دلیل اندازه بزرگ اشیا در این بخش، دقت مناسبی نشان می‌دهد و فاصله عملکردی کمتری با Faster R-CNN دارد.

عوامل تاثیرگذار بر عملکرد مدل‌ها

در این پیاده‌سازی هیچگونه preprocess ای بر روی تصاویر انجام نشد. در صورتی که در بسیاری از موارد با انجام preprocess عملکرد مدل‌ها ممکن است تغییر کند. این preprocess ها میتوانن شامل موارد مختلفی مثل Down-Sampling یا Over-Sampling کلاس‌های اکثریت و اقلیت، به ترتیب، و یا نرمال‌سازی مناسب برای مدل‌ها یا موارد گوناگون دیگری باشند.

همچنین برای جلوگیری از Overfit کردن و یا متنوع شدن دیتاست آموزش، میتوان از روش‌های Data-Augmentation استفاده کرد.

در آموزش مدل‌ها با روش یادگیری انتقالی، freeze کردن یا نکردن وزن‌های آموزش دیده شده موضوع مهمی است. در برخی موارد که دیتاستی که بر روی آن آموزش انجام شده است با دیتاست مورد آزمایش شبیه باشد یا موارد مختلف دیگر، میتوان شبکه را freeze کرد تا بخش‌های بعدی شبکه آموزش ببینند و هم پیچیدگی آموزش کم شود و هم از وزن‌های آماده به خوبی استفاده شود. اما در این پروژه، وزن‌های آماده از دیتاست COCO بنظر خیلی مناسب نبودند و درصورت freeze کردن شبکه، نتایج بسیار بدی گرفته میشد و نیاز بود تا آن وزن‌ها هم fine tune شوند تا به نتایج خوبی برسیم.

همچنین در صورت امکان، انتخاب وزن‌هایی برای مدل از پیش آموزش شده که از آموزش مدل بر روی دیتاست‌های دیگری (به جز COCO در این پروژه) بدست آمده است و بررسی نتایج آن نیز می‌تواند در عملکرد مدل تاثیرگذار باشند.

تنظیم هایپرپارامترهای مختلف، از جمله تابع هزینه، بهینه‌ساز، نرخ یادگیری و تعداد epoch نیز میتوانند تاثیر بسیار زیادی بر روی عملکرد نهایی مدل بگذارند. Fine tune کردن این هایپرپارامترها فقط با آزمایش‌های متعدد امکان دارد تا بتوانیم بهترین مجموعه از آنها را پیدا و استفاده کنیم.