



به نام خدا



دانشگاه تهران
دانشکده مهندسی برق و کامپیوتر
شبکه های عصبی و یادگیری عمیق

تمرین ویژه سری اول

امیر محمد رنجبر پازکی	نام و نام خانوادگی
۸۱۰۱۹۹۳۴۰	شماره دانشجویی
۱۴۰۰/۳/۱۸	تاریخ ارسال گزارش

فهرست گزارش سوالات

- 3 Object Detection with YOLOv5 – سوال ۱
- 13 Semantic Segmentation – سوال ۲

سوال ۱ – Object Detection with YOLOv5

۱) معماری نسخه چهارم YOLO از یک شبکه‌ی CSP با عنوان CSPDarknet53 به عنوان backbone. ماثول سر استفاده می‌کند.

شبکه‌ی CSP یک backbone جدید بود که می‌تواند توانایی یادگیری CNN را افزایش دهد. Spatial Pyramid Pooling به این backbone افزوده می‌شود تا بتواند receptive field را افزایش دهد و ویژگی‌های زمینه‌ای بر جسته‌تری را استخراج کند.

در این شبکه به جای feature pyramid network نسخه سوم از PANet برای جمع‌آوری پارامترها استفاده شده‌است.

نسخه چهارم دوباره سریع‌تر از EfficientDet (رقیب قابل مقایسه) است. همچنین، میانگین دقت کلاس‌های مختلف درصد و Frame Per Second پردازشی آن ۱۲ درصد نسبت به نسخه سوم افزایش داشته‌است.

همچنین، این نسخه از mosaix data augmentation استفاده می‌کند که یک تکنیک جدید برای داده‌افزایی است که در آن ۴ عکس آموزش به جای یک عکس با یکدیگر ادغام می‌شوند.

همچنین این شبکه از cross mini-batch normalization استفاده می‌کند که فرض می‌کند در هر batch چهار mini-batch وجود دارد.

توسعه‌دهندگان این نسخه دو دسته روش‌هایی که برای افزایش دقت تشخیص دهنده‌های اشیا را بررسی کرده‌اند و برخی از آن‌ها را در این نسخه به کار گرفته‌اند که در تصویر زیر آن‌ها را مشاهده می‌کنید.

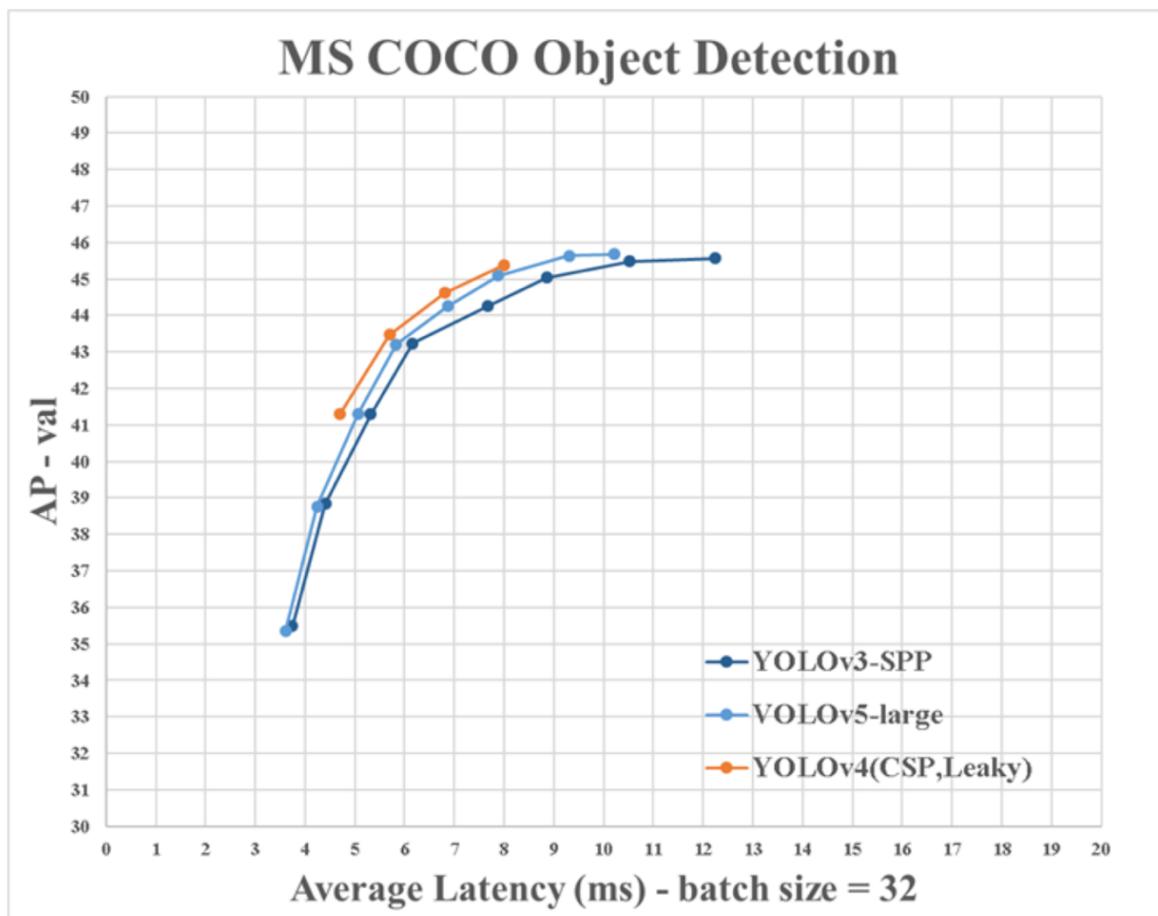
	Backbone	Detector
Bag of Freebies (BoF)	<ul style="list-style-type: none">CutMixMosaic data augmentationDropBlockClass label smoothing	<ul style="list-style-type: none">IoU-lossCross mini-Batch NormalizationDropBlockMosaic data augmentationSelf-Adversarial TrainingMultiple anchors for a single ground truthCosine annealing schedulerOptimal hyperparametersRandom training shapes
Bag of Specials (BoS)	<ul style="list-style-type: none">Mish activationCross-stage partial connections (CSP)Multi-input weighted residual connections (MiWRC)	<ul style="list-style-type: none">Mish activationSPP-blockSAM-blockPAN path-aggregation blockDIoU-NMS

شکل ۱- تصویر روش‌های استفاده شده در بخش‌های مختلف YOLOv4

معماری نسخه پنجم YOLO در بسیاری از موارد مشابه نسخه چهارم آن است. به دلیل استفاده از این نوآوری‌ها کارایی این نسخه نیز بسیار قابل قبول است. تنها تفاوت این نسخه با نسخه چهار انتخاب سازگار و در هنگام آموزش سایز anchor box‌هاست. در نسخه‌ی قبلی سایز anchor box با توجه به مجموعه‌داده و در فرآیند جدایی باید انتخاب می‌شد. برای مثال، برای تصویر زرافه anchor box باریک‌تر و درازتری در نظر گرفته شود. البته این ایده در نسخه چهار نیز مورد استفاده قرار گرفت.

این دو معماری تفاوت چندانی با یکدیگر ندارند و به دلیل عرضه کمی دیرتر نسخه پنجم این نام بر روی آن مانده است و گرنه تقریباً همزمان با نسخه چهار عرضه شده است.

زمان آموزی نسخه پنجم کوتاه‌تر از نسخه چهارم است. همچنین نسخه پنجم زمان استنباط کمتری نسبت به نسخه چهارم نیاز است. نسخه چهارم این مجموعه کمی دقت میانگین بهتری بر حسب میانگین latency MS COCO نسبت به نسخه پنجم داشته است. در وصف این نسخه آمده است که استفاده آن نسبت به نسخه چهارم راحت‌تر است و کارآیی بهتری رو داده‌ی شخصی‌سازی شده دارد.



شکل ۲- نمودار دقت میانگین نسخه‌های مختلف YOLO بر روی MS COCO

- ۲) گام به گام با نوتبوک جلو رفته و قدمها در زیر توضیح داده می‌شود.
- نصب وابستگی‌ها:

ابتدا repository مربوط به YOLOv5 clone می‌کنیم و بر روی کامیت موردنظر قرار می‌دهیم. سپس پیش‌نیازی‌های آن را که در فایل requirements.txt قرار گرفته‌اند با استفاده از pip نصب می‌کنیم. سپس کتابخانه‌های مورد نیاز را import می‌کنیم.

```
[1] # clone YOLOv5 repository
!git clone https://github.com/ultralytics/yolov5 # clone repo
%cd yolov5
!git reset --hard 886f1c03d839575afecb059accf74296fad395b6

Cloning into 'yolov5'...
remote: Enumerating objects: 6872, done.
remote: Counting objects: 100% (176/176), done.
remote: Compressing objects: 100% (107/107), done.
remote: Total 6872 (delta 103), reused 118 (delta 68), pack-reused 6696
Receiving objects: 100% (6872/6872), 9.03 MiB | 31.89 MiB/s, done.
Resolving deltas: 100% (4710/4710), done.
/content/yolov5
HEAD is now at 886f1c0 DDP after autoanchor reorder (#2421)

▶ # install dependencies as necessary
!pip install -qr requirements.txt # install dependencies (ignore errors)
import torch

from IPython.display import Image, clear_output # to display images
from utils.google_utils import gdrive_download # to download models/datasets

# clear_output()
print('Setup complete. Using torch %s %s' % (torch.__version__, torch.cuda.get_device_properties(0) if torch.cuda.is_ava
```

```
Setup complete. Using torch 1.8.1+cu101 _CudaDeviceProperties(name='Tesla T4', major=7, minor=5, total_memory=15109MB, mu
```

شکل ۳- گرفتن کد و نصب پیش‌نیازی‌های YOLOv5

البته لازم به ذکر است این پیش‌نیازی‌ها بر روی colab قرار دارد و نصب پیش‌نیازی برای local نیاز است.

- دانلود یا آماده کردن داده با فرمت مناسب:

برای کار با این کتابخانه نیاز است تا داده‌ها فرمت مناسب داشته باشند. این داده‌ها باید سازگار با Pytorch و YOLOv5 باشند. یکی از راه‌های این کار استفاده از مجموعه داده‌های سایت Roboflow است که می‌توان با فرمت موردنظر دانلود کرد.

اگر خودمان مجموعه داده‌ای داشته باشیم، می‌توان با استفاده از یک annotator روی آن label زد. سپس، آن را بر روی Roboflow آپلود کنیم تا ضمن تنظیم فرمت داده‌افزایی نیز انجام دهد و مجموعه داده غنی با فرمت مناسب برای ما بوجود آورد. همچنین این مجموعه داده‌ها یک فایل yaml باید داشته باشد که تعداد کلاس‌ها به همراه نام آن‌ها و همچنین مسیر آموزش و ارزیابی در آن قید شود.

داده‌های ما در این سوال از پیش آمده هستند. به همین دلیل، کافی است تا روی google drive آپلود شوند و سپس، با mount کردن drive و خارج کردن آن‌ها از حالت فشرده در دسترس قرار گیرند. نیاز است تا این داده‌ها در پوشه content قرار بگیرند.

- تعریف معماری مدل و تنظیمات آن:

این تنظیمات شامل تعداد کلاس‌ها، anchorها و هر لایه در فایل yolov5s.yaml آمده است. کافی است این فایل را تغییر دهیم و تنظیمات مورد نیاز را قرار دهیم.

```
%%writetemplate /content/yolov5/models/custom_yolov5s.yaml

# parameters
nc: {num_classes} # number of classes
depth_multiple: 0.33 # model depth multiple
width_multiple: 0.50 # layer channel multiple

# anchors
anchors:
  - [10,13, 16,30, 33,23] # P3/8
  - [30,61, 62,45, 59,119] # P4/16
  - [116,90, 156,198, 373,326] # P5/32

# YOLOv5 backbone
backbone:
  # [from, number, module, args]
  [[-1, 1, Focus, [64, 3]], # 0-P1/2
   [-1, 1, Conv, [128, 3, 2]], # 1-P2/4
   [-1, 3, BottleneckCSP, [128]],
   [-1, 1, Conv, [256, 3, 2]], # 3-P3/8
   [-1, 9, BottleneckCSP, [256]],
   [-1, 1, Conv, [512, 3, 2]], # 5-P4/16
   [-1, 9, BottleneckCSP, [512]],
   [-1, 1, Conv, [1024, 3, 2]], # 7-P5/32
   [-1, 1, SPP, [1024, [5, 9, 13]]],
   [-1, 3, BottleneckCSP, [1024, False]], # 9
  ]

# YOLOv5 head
head:
  [[-1, 1, Conv, [512, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 6], 1, Concat, [1]], # cat backbone P4
   [-1, 3, BottleneckCSP, [512, False]], # 13

   [-1, 1, Conv, [256, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 4], 1, Concat, [1]], # cat backbone P3
   [-1, 3, BottleneckCSP, [256, False]], # 17 (P3/8-small)

   [-1, 1, Conv, [256, 3, 2]],
   [[-1, 14], 1, Concat, [1]], # cat head P4
   [-1, 3, BottleneckCSP, [512, False]], # 20 (P4/16-medium)

   [-1, 1, Conv, [512, 3, 2]],
   [[-1, 10], 1, Concat, [1]], # cat head P5
   [-1, 3, BottleneckCSP, [1024, False]], # 23 (P5/32-large)
```

شکل ۴- تنظیمات مربوط به معماری مدل

- آموزش مدل:

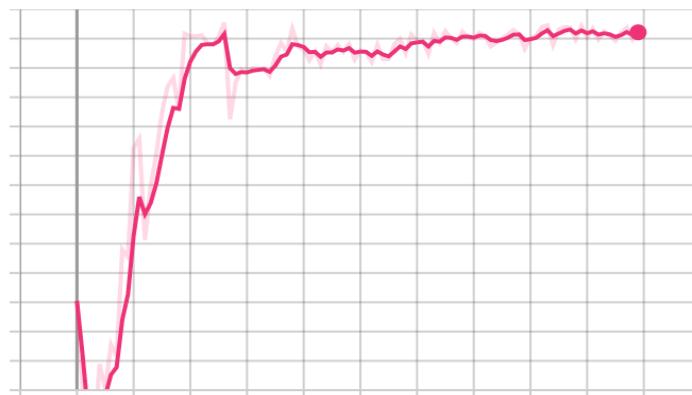
در این مرحله باید داده‌های آموزش و ارزیابی را به پوشه **content** منتقل کنیم و سپس، با اجرای کد **train.py** با استفاده از تنظیمات بالا و دادن هایپرپارامترها و همچنین مسیر داده‌ی آموزش مدل مورد نظر را آموزش می‌دهیم. تعداد epoch‌ها ۱۰۰ در نظر گرفته شده است. نتایج آموزش در epoch آخر به صورت زیر بوده است.

Epoch	gpu_mem	box	obj	cls	total	targets	img_size
99/99	1.82G	0.04907	0.04016	0.008577	0.09781	270	416: 100% 99/99 [00:41<00:00,
	Class	Images	Targets		P	R	mAP@.5 mAP@.5:.95: 100% 2/2 [00
	all	58	1.00e+03		0.967	0.908	0.924 0.587
	blue	58		115	1	0.982	0.995 0.62
	green	58		290	0.993	0.948	0.987 0.586
	red	58		290	1	0.993	0.996 0.639
	vline	58		136	0.971	0.98	0.984 0.879
	white	58		58	0.841	0.552	0.589 0.17
	yellow	58		116	1	0.991	0.995 0.626

شکل ۵- نتایج آموزش در epoch آخر

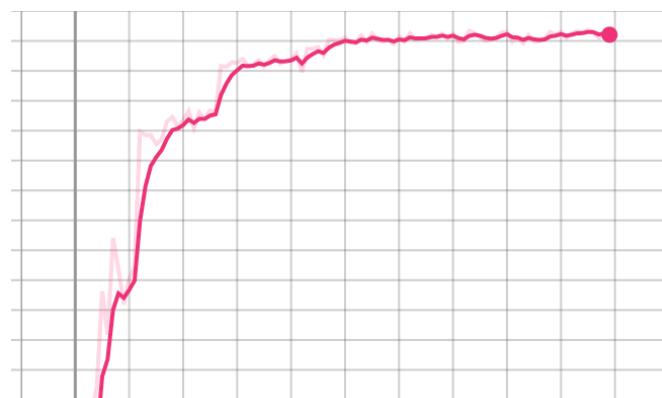
- ارزیابی عملکرد مدل: برای ارزیابی عملکرد مدل، نمودارهای recall و precision .mAP0.5:0.95 و mAP0.5 در epoch رسم شده و در زیر آورده شده است.

metrics/precision
tag: metrics/precision



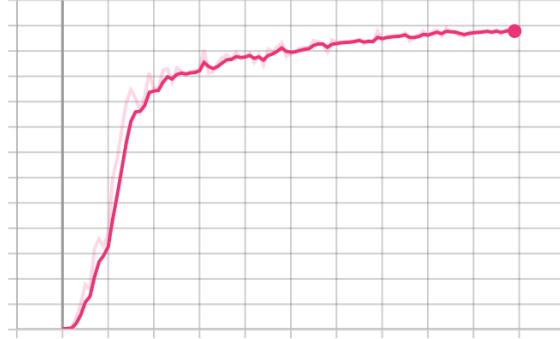
شکل ۶- نمودار precision مدل

metrics/recall
tag: metrics/recall



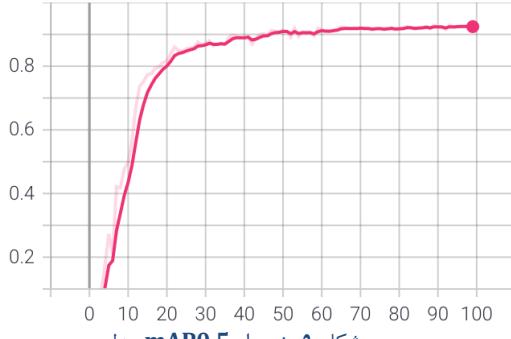
شکل ۷- نمودار recall مدل

metrics/mAP_0.5:0.95
tag: metrics/mAP_0.5:0.95



شکل ۸- نمودار mAP0.5:0.95 مدل

metrics/mAP_0.5
tag: metrics/mAP_0.5



شکل ۹- نمودار mAP0.5 مدل

- دیدن نتایج آموزش:

در تصویر زیر تعدادی از داده‌های آموزش به همراه labelهای ایشان قابل مشاهده هستند.

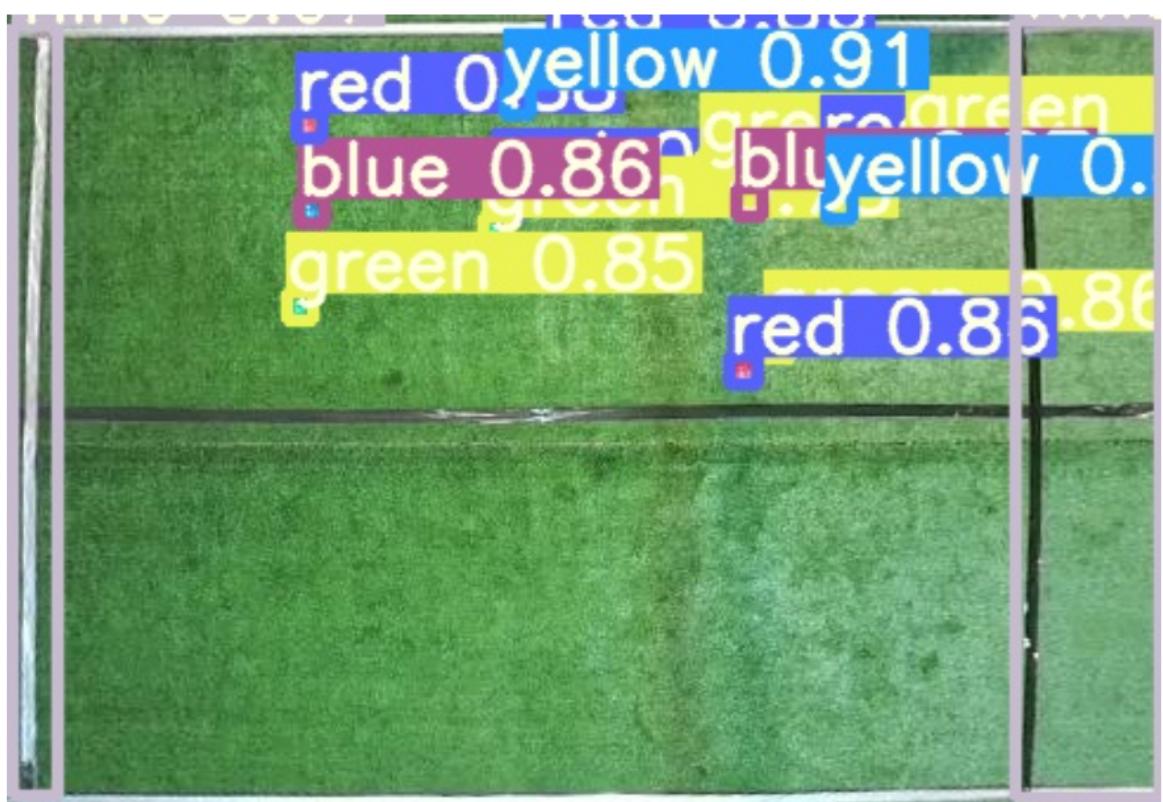


شکل ۱۰- برخی از داده‌های آموزش به همراه label

- بررسی داده‌های ارزیابی با استفاده از وزن‌های بدست آمده:

وزن‌های به دست آمده از آموزش را بر روی درایو ذخیره می‌کنیم تا بتوان از آن استفاده کرد. حال با استفاده از آن وزن‌ها می‌توان مدل قبلی را بازیابی و یا استفاده کد detect.py فرآیند تشخیص اشیا را روی داده‌ها دیده نشده اجرا کرد. نتایج این قسمت در قسمت ۳ اورده شده است.

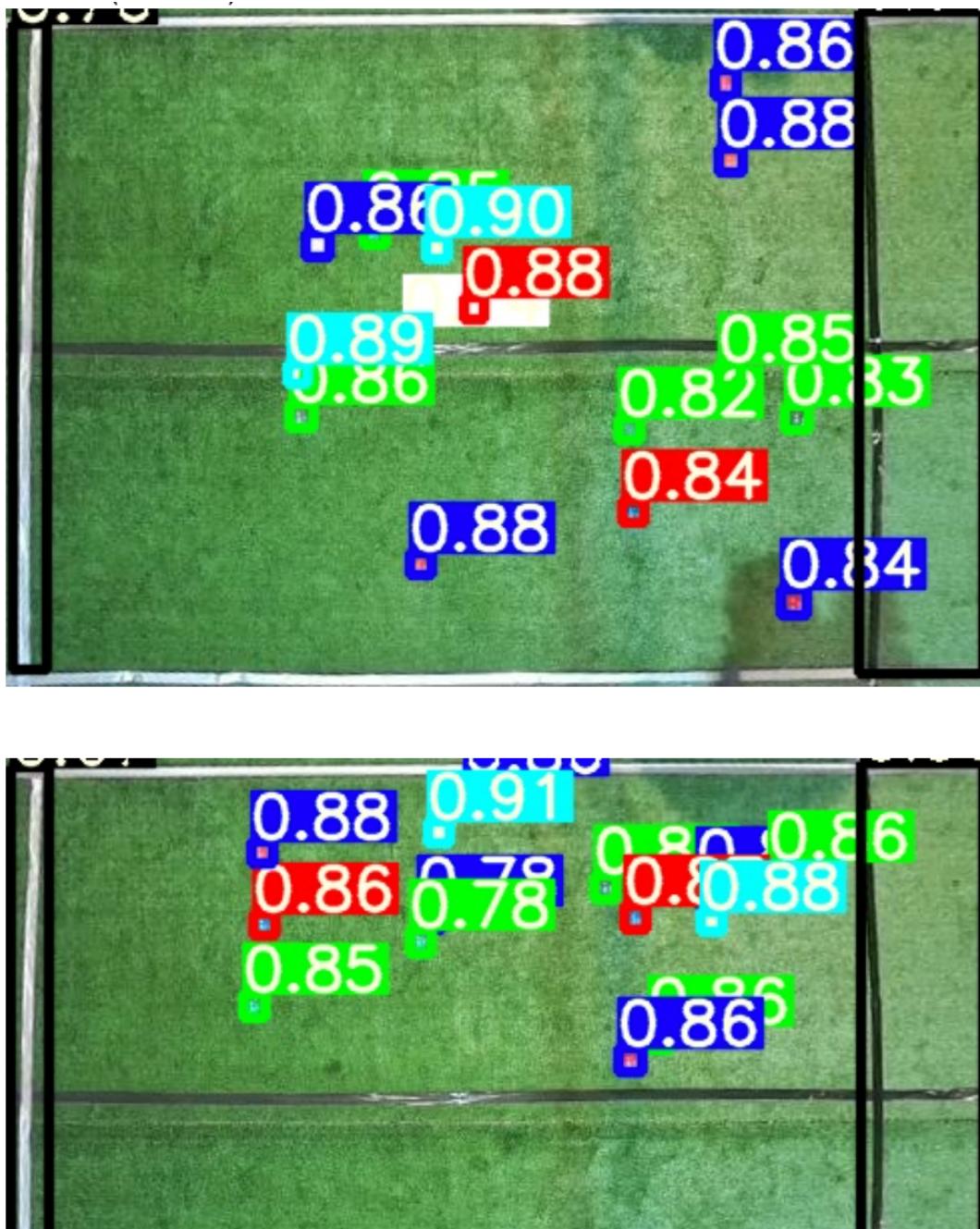
- تصاویر موجود در پوشه‌ی valid به مدل داده شد و با استفاده از detect.py خروجی تولیدی مدل گرفته شد که در زیر دو نمونه از آن‌ها آورده شده است.



شکل ۱۱- دو نمونه پیش‌بینی شده از داده‌ی **valid** با استفاده از مدل آموزش دیده

همانطور که مشاهده می‌کنید مشکل اصلی این خروجی‌ها شلوغی بیش از حد است که این به دلیل، آوردن نام به همراه میزان اطمینان از پیش‌بینی در **label** هر باکس است. این کار باعث شده است تا در یک نگاه نتوان نتیجه را متوجه شد.

به همین دلیل، ابتدا نام‌ها حذف شد و تنها میزان اطمینان label قرار گرفت. همچنین رنگ باکس‌ها به رنگ کلاس‌ها درآمد. اما باز هم مشکل شلوغی کمی پایر جا ماند. در انتها، تنها باکس‌ها به عنوان مشخص کننده کلاس قرار گرفتند و label کاملاً حذف شد.



شکل ۱۲ - label کوچک شده برای اطلاعات بیشتر



/content/yolov5/runs/detect/exp7/40.jpg.rf.50d71040f4f3f838



شکل ۱۳- نحوه گذاری نهایی label (قبل تشخیص)

۴) برای این کار کد detect.py در فایل pss از تشخیص گوی‌ها مختصات سمت چپ بالا و سمت راست پایین باکس‌ها در یک لیست ذخیره شد و همچنین، تعداد گوی‌ها سفید و مختصات آن‌ها جداگانه نگهداری شد. سپس در صورتی که تنها یک گوی سفید وجود داشت، فاصله گوی‌ها و خط عمودی از گوی سفید محاسبه شد و سپس، مرتب سازی روی آن به صورت صعودی صورت گرفت. نتایج در فایل distance.txt که در ضمیمه آمده است، برای هر عکس داده‌ی ارزیابی آمده است.

```

centers = []
white_x, white_y = 0, 0
white_counter = 0
distance_file = open("distance.txt", 'a')
for *xyxy, _, cls in reversed(det):
    x_lu, y_lu, x_rd, y_rd = [x.item() for x in xyxy]
    center_x, center_y = (x_lu + x_rd)/2, (y_lu + y_rd)/2
    centers.append((names[int(cls)], center_x, center_y))
if names[int(cls)] == "white":
    white_x, white_y = center_x, center_y
    white_counter += 1
image_name = path.split('/')[-1]
distance_file.write(image_name + "\n")
distances = []
if white_counter == 1:
    for center in centers:
        color_cls, center_x, center_y = center
        if color_cls == "white":
            continue
        distance = math.sqrt((center_x - white_x) ** 2 + (center_y - white_y) ** 2)
        distances.append((color_cls, distance))
    distances.sort(key=lambda x:x[1])
    for obj, distance in distances:
        distance_file.write("{}: {}\n".format(obj, distance))
else:
    distance_file.write("No white ball or more than one white ball has been detected.\n")
distance_file.write("\n\n")
distance_file.close()

```

شکل ۱۴- تکه کد مربوط به محاسبه فاصله گوی ها و خط عمودی به ترتیب صعودی از گوی سفید

104.jpg.rf.922cc5f61fc87ab619f35377de76820c.jpg

```

blue: 33.30540496676178
blue: 55.78530272392541
red: 92.31061694084815
green: 94.52116165176982
green: 110.59950271135942
yellow: 111.00450441310929
red: 111.13955191559843
red: 122.72937708633577
vline: 138.26152754833862
vline: 142.00352108310554
green: 144.00086805293918
red: 176.47946056127893
red: 184.0387187523321
green: 186.3310226451838
yellow: 277.1461708196597
vline: 330.0545409474016

```

106.jpg.rf.741aa61c5acfb884a2e5d91e5951d70f.jpg

No white ball or more than one white ball has been detected.

114.jpg.rf.a67c6b727ec53130ac57c7380c9b7071.jpg
green: 47.668123520860355
red: 131.0925245771093
red: 136.50091574784398
vline: 156.13535794303607
red: 185.70204629998022
red: 189.00066137450418
green: 199.465535870235
green: 211.83543140844026
blue: 247.73070056010417
yellow: 297.2490538252393
red: 318.330802782263
green: 341.5391632009424
green: 392.7228666629943
yellow: 425.3577905716551
vline: 463.9536614792473

117.jpg.rf.13a5d6ae58d59d25813cb7882f9a924c.jpg
green: 15.628499608087784
yellow: 18.741664813991314
red: 23.717082451262844
green: 25.46075411294803
red: 44.11915683691156
red: 44.30011286667337
blue: 51.32494520211395
green: 57.578207683115664
red: 58.148516748065035
green: 64.41273166075166
red: 75.85677293426079
green: 77.20103626247513
blue: 87.0933407328023
yellow: 91.22773701018787
vline: 95.32313465261201
vline: 231.9051530259731
vline: 358.79555459899444

123.jpg.rf.3f3fcde3b223aa7ba54586591d47da5d.jpg
green: 15.074813431681335
blue: 15.20690632574555
green: 21.70829334609241
yellow: 26.1725046566048
green: 29.415132160165456
red: 42.005951959216446
red: 48.399380161320245
red: 50.06246098625196
red: 67.03170891451299
blue: 75.67364138192373
red: 81.62413858657254
green: 87.78382538941898
yellow: 96.82974749528164
blue: 107.0
vline: 166.4692163734785
vline: 325.00192307123353

```

125.jpg.rf.f6f19a6e1c9873032000a8411f61de88.jpg
red: 70.14627573863064
red: 71.17583859709698
yellow: 71.18461912520148
blue: 87.17367721967452
yellow: 97.93492737527302
blue: 123.98891079447388
green: 161.45123102658584
green: 168.75796277509397
green: 183.17546233052067
red: 209.7528307317925
vline: 213.90009350161586
green: 240.05051551704696
red: 292.85662020859286
red: 304.64733709651887
green: 315.94065898519614
vline: 328.93654403243187

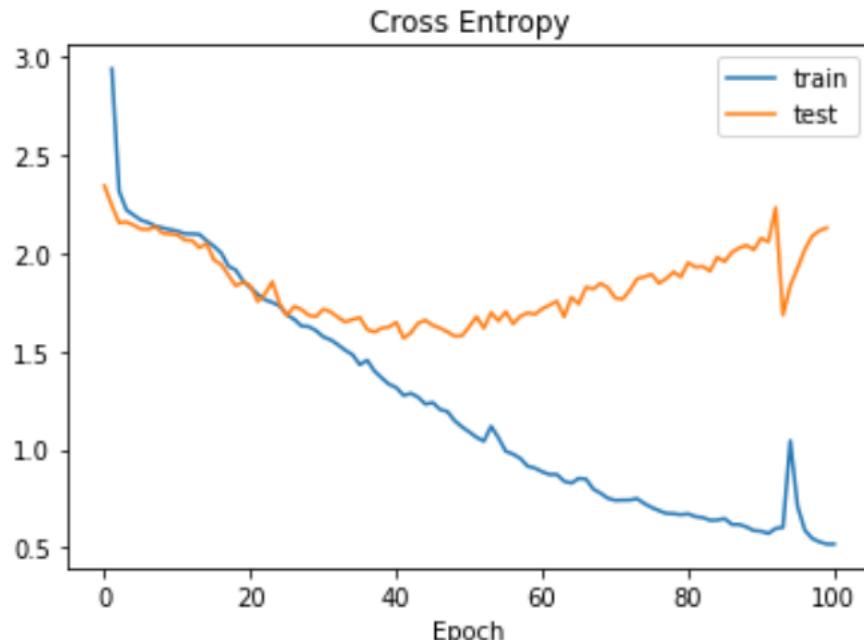
```

شکل ۱۵- نتایج مربوط به فاصله گوی‌ها به ترتیب صعودی برای پنج عکس دلخواه

در کنار گزارش فایل detect2.py به همراه بهترین وزن‌های شبکه به دست آمده بارگذاری شده‌اند.

Semantic Segmentation – ۲

۱) شبکه‌ی U-Net مطابق با مقاله گفته شده و با استفاده از pytorch پیاده‌سازی شد. نمودار خطای آموزش و آزمون در هر epoch در زیر قابل مشاهده است.



شکل ۱۶- نمودار خطای شبکه‌ی U-Net بر روی داده‌ی آموزش و آزمون

۲) اول داده‌ی تست به مدل ورودی داده شد و خروجی‌ها مطابق زیر گرفته شد.

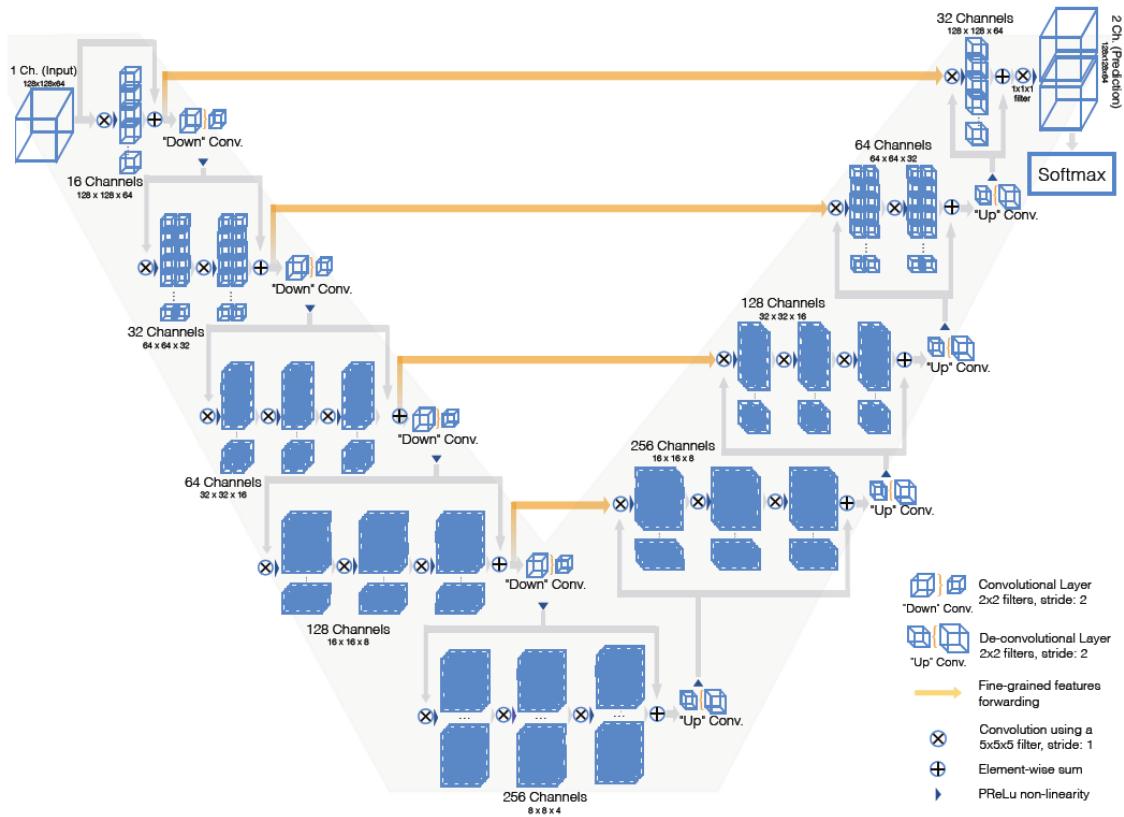


شکل ۱۷- ورودی دادن به شبکه و نمایش خروجی‌ها

۳) تعداد epoch موردنیاز برای آموزش شبکه ۴۰ است. زیرا پس از آن دقت داده‌ی آموزش بالاتر می‌رود ولی دقت داده‌ی آزمون کاهش می‌یابد یا به عبارتی، خطای آن افزایش می‌یابد. در این حالت شبکه دچار overfitting شده‌است و نویز روی داده‌ی آموزش را فراگرفته‌است و به همین دلیل، قدرت تعیین‌دهی آن کاهش یافته‌است. داده‌ی آموزش و آزمون به صورت تصادفی به نسبت ۷۰ به ۳۰ درصد تقسیم شده‌اند. این کار به این دلیل صورت گرفته‌است که سوگیری روی انتخاب عکسی در حالت خاص نداشته باشیم و انواع عکس‌ها را ببینیم. برای افزایش دقت و پیش‌بینی شبکه می‌توان از تکنیک‌های داده افزایی استفاده کرد تا مدل بتواند روی داده‌های تست که ندیده‌است عملکرد بهتری داشته باشد. یا به عبارتی دیگر، میزان robustness مدل بالاتر رود.

شبکه‌ی V-Net: این شبکه برای segmentation بر روی عکس‌های سه بعدی ساخته شده‌است. کاربرد این شبکه در عکس برداری‌های پزشکی مانند MRI است که می‌تواند در تشخیص کمک شایانی کند. در طول یادگیری یکتابع خاص بهینه‌سازی می‌شود که بر مبنای dice coefficient است. این ضریب بیانگر دو برابر نسبت هم پوشانی دو سطح به مجموع آن دو سطح به تنها‌ی است. با این کار می‌توان عدم توازن شدید حجم‌های جلویی و پشتی را تعدیل کرد.

در آموزش این شبکه به دلیل کمبود داده از روش‌های مختلف data augmentation استفاده می‌شود. ساختار این شبکه نیز مانند U-Net دو تکه است. در تکه سمت چپ فرآیند فشرده‌سازی رخ می‌دهد. در هر یک تابع residual یاد گرفته‌می‌شود. این ایده همگرایی شبکه را نسبت به U-Net تامین می‌کند. Convolution استفاده شده سه‌بعدی و دارای سایز $5*5*5$ است. به جای استفاده از pooling از کانولوشن‌های $2*2$ استفاده شده‌است که رزلوشن را کاهش جدی می‌دهد و باعث استفاده بهینه‌تر از حافظه می‌شود. در قسمت راست شبکه از deconvolution استفاده شده‌است. در این قسمت نیز یکتابع residual یاد گرفته‌می‌شود. خطوط افقی نیز برای افزودن داده‌هایی از سمت چپ به راست برای جبران داده‌هایی است که در compression گم شده‌اند.



شکل ۱۸- معماری شبکه v-net

تابع خطای گفته شده نیز فرمولی به صورت زیر دارد.

$$D = \frac{2 \sum_i^N p_i g_i}{\sum_i^N p_i^2 + \sum_i^N g_i^2}$$

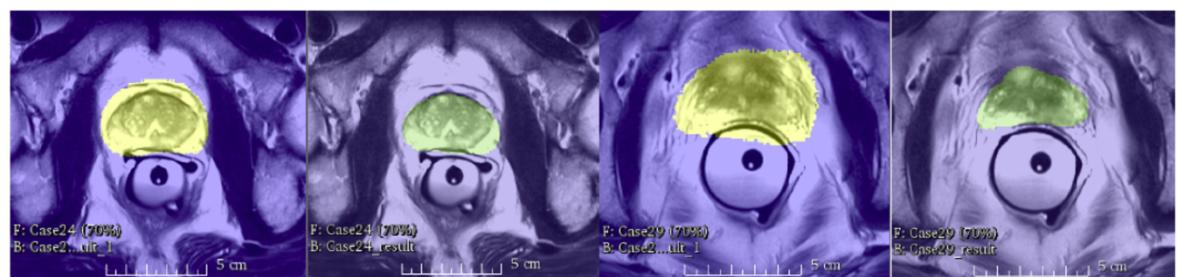
شکل ۱۹- فرمول تابع loss

استفاده از این تابع خطای باعث وزن دهنی مناسب کلاس های مختلف و حل مشکل عدم توازن در عمق اشیا می شود. استفاده از این شبکه نتایج قابل قبولی داشته است که در زیر مشاهده می شود. همانطور که می بینید، استفاده از این تابع خطای خاص باعث بهتر شدن دقت کلی و دقت تسک چالشی شده است.

Algorithm	Avg. Dice	Avg. Hausdorff distance	Score on challenge task
V-Net + Dice-based loss	0.869 ± 0.033	5.71 ± 1.20 mm	82.39
V-Net + mult. logistic loss	0.739 ± 0.088	10.55 ± 5.38 mm	63.30
Imorphics [18]	0.879 ± 0.044	5.935 ± 2.14 mm	84.36
ScrAutoProstate	0.874 ± 0.036	5.58 ± 1.49 mm	83.49
SBIA	0.835 ± 0.055	7.73 ± 2.68 mm	78.33
Grislies	0.834 ± 0.082	7.90 ± 3.82 mm	77.55

شکل ۲۰- مقایسه دقت الگوریتم های مختلف

عملکرد این شبکه با دو تابع خطای متفاوت در شکل های زیر مشخص شده است. استفاده از تابع خطای گفته شده دقت چشمگیری دارد.



شکل -۲۱- عملکرد **v-net** با استفاده از تابع خطای **logistic** (زرد) و تابع خطای **dice** (سبز)