



به نام خدا



دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

## شبکه های عصبی و یادگیری عمیق

استاد

دکتر کلهر

تمرین اضافه

سحر رجبی	نام و نام خانوادگی
۸۱۰۱۹۹۱۶۵	شماره دانشجویی
۱۴۰۰/۳/۱۷	تاریخ ارسال گزارش

## فهرست گزارش سوالات

3.....	Object Detection with YOLOv5 – 1
10 .....	Semantic Segmentation – 2

# سوال ۱ Object Detection with YOLOv5 – ۱

(۱)

ابزاری برای تشخیص YOLO real-time و با دقت بالای شیهای مختلف است. در نسخه‌ی ۲، سرعت پردازش عکس‌ها به ۴۰ تا ۹۰ فریم در ثانیه رسید. در نسخه‌ی ۳ام، افراد می‌توانستند یک tradeoff بین سرعت و دقت با عوض کردن model size داشته باشند.

اما در نسخه‌ی ۴ام، تحت تاثیر BoS (bag of specials) و BoF (bag of freebies) باعث افزایش دقت BoF real-time detector شود. بدون افزایش زمان استنتاج می‌شود و BoS با اندکی افزایش زمان استنتاج، باعث رشد زیادی در دقت تشخیص اشیا می‌شود.

(۲)

برای توضیح قسمت‌های notebook، تصویر هر بخش بعد از توضیح آن آورده شده است.

- در قسمت اول، ما commit hash آن clone را از github می‌کنیم و با کمک commit hash به مدنظر می‌رویم.

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

شکل ۱- کد مربوط به clone کردن

- در بخش بعدی، کتابخانه‌ها و پیش‌نیازهای لازم برای استفاده از YOLOv5 را نصب می‌کنیم.

```
[ ] # 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).name))
import importlib
```

شکل ۲- کتابخانه‌های مورد استفاده

- در قسمت سوم، notebook اصلی، با دانلود یک دیتاست از roboflow آن را unzip می‌کند تا از آن استفاده کند. اما ما در این قسمت، داده‌ها را در درایو آپلود کردیم و سپس با وصل شدن به درایو، آن‌ها را unzip کردیم.

```
[ ] from google.colab import drive
drive.mount('/content/drive/')

Drive already mounted at /content/drive/; to attempt to forcibly remount, call drive.mount("/content/drive/", force_remount=True)

[ ] # Export code snippet and paste here
%cd /content
!unzip /content/drive/MyDrive/roboflow.zip; rm /content/drive/MyDrive/roboflow.zip
```

شکل ۳- unzip کردن داده‌ها، برای استفاده‌های بعدی

- فایل data.yaml اطلاعات مربوط به داده‌های ما برای استفاده از YOLO را دربر دارد. شامل محل داده‌های train و validation، تعداد کلاس‌های موجود در داده‌های ما و نام این کلاس‌ها.

```
# this is the YAML file Roboflow wrote for us that we're loading
%cat data.yaml

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

nc: 6
names: ['blue', 'green', 'red', 'vline', 'white', 'yellow']
```

شکل ۴- اطلاعات داخل فایل data.yaml

- در بخش بعدی، تعداد کلاس‌های موجود در داده‌های ما، با استفاده از اطلاعات موجود در data.yaml بدست می‌آید. به این ترتیب که این فایل با استفاده از ابزار yaml خوانده می‌شود و پارامتر nc آن به عنوان تعداد کلاس‌ها، خوانده می‌شود.

```
# define number of classes based on YAML
import yaml
with open("data.yaml", 'r') as stream:
    num_classes = str(yaml.safe_load(stream)['nc'])
```

شکل ۵- استخراج تعداد کلاس‌ها از فایل data.yaml

- در قسمت بعدی، اطلاعات یک configuration yaml file که در واقع configuration مربوط به مدل است نمایش داده شده است. البته این قسمت، آن config ای که ما برای آموزش شبکه استفاده می‌کنیم نیست؛ اما می‌توانیم شما کلی این مدل، شامل لایه‌ها و نحوه قرارگیری آن‌ها در کنار یکدیگر را بینیم.

```
#this is the model configuration we will use for our tutorial
%cat /content/yolov5/models/yolov5s.yaml
```

```
# parameters
nc: 80 # 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, C3, [128]],
  [-1, 1, Conv, [256, 3, 2]], # 3-P3/8
  [-1, 1, BottleneckCSP, [256]],
  [-1, 1, Conv, [512, 3, 2]], # 5-P4/16
  [-1, 9, C3, [512]],
  [-1, 1, Conv, [1024, 3, 2]], # 7-P5/32
  [-1, 1, SPP, [1024, [5, 9, 13]]],
  [-1, 3, C3, [1024, False]], # 9
  ]]
```

شکل ۶- پخشی از اطلاعات داخل یک model configuration file

- در این قسمت، یک magic function تعریف می‌کنیم که آن را در بالای یک cell در jupyter notebook استفاده کنیم؛ خطوط آن را در فایلی که به عنوان پارامتر به آن داده شده، می‌نویسند.

```
[ ] #customize iPython writefile so we can write variables
from IPython.core.magic import register_line_cell_magic

@register_line_cell_magic
def writetemplate(line, cell):
    with open(line, 'w') as f:
        f.write(cell.format(**globals()))
```

شکل ۷- تعریف magic function گفته شده

- در اینجا با استفاده از writetemplate تعریف شده در قسمت قبلی، ما یه model configuration جدید- مطابق با فرمت قبلی که در قسمت‌های پیش مشاهده کردیم- می‌سازیم. همچنین تعداد کلاس‌ها در داده‌های مورد نظر ما هم، در این بخش تنظیم می‌شود.

```
[ ] %%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, 1, 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
  ]]
```

شکل ۸- پخشی از تعریف یک custom config file

- این بخش که در واقع بخش اصلی کار ماست، در واقع فایل `train.py` را با استفاده از آرگومان‌های ورودی‌ای که در ادامه توضیح می‌دهیم؛ اجرا می‌کند. آرگومان `img` در واقع اندازه‌ی تصاویر ورودی را مشخص می‌کند. آرگومان `batch` –اندازه‌ی `batch` را مشخص می‌کند. آرگومان `epoch` –بیان می‌کند که چند دور باید داده‌های آموزش را بینیم. آرگومان `data.yaml` –مسیر فایل `data.yaml` را برای شناخت داده‌ها می‌گیرد. اطلاعاتی مثل محل ذخیره‌ی داده‌ها، تعداد کلاس‌های اشیا در دیتابست و ... در این فایل قرار دارد. همچنین پارامتر `cfg`-- را برای ساخت مدل با `config` که در قسمت‌های قبل ساختیم، دریافت می‌کند و مدل ما با آن لایه‌ها و ویژگی ساخته می‌شود. پارامتر `weights` در صورتی که بخواهیم از فایل وزن‌های خاصی استفاده کنیم؛ می‌تواند مورد استفاده قرار بگیرد که در اینجا ما هیچ مسیری به آن ندادیم. آرگومان `name` –در واقع نام فولدری است که می‌خواهیم نتایج ما در آن قرار بگیرد و در آخر `cache` –هم در صورتی استفاده می‌شود که بخواهیم تصاویر را برای آموزش سریع‌تر، `cache` کنیم.

```
# train yolov5s on custom data for 100 epochs
# time its performance
%%time
%cd /content/yolov5/
!python train.py --img 416 --batch 16 --epochs 100 --data '../data.yaml' --cfg ./
```

شکل ۹- اجرای `train.py` با پارامترهای مورد نظر

- بخش قبلی پایان کارهای لازم برای آموزش مدل بود. از این بخش به بعد، کدهای مربوط به `evaluate` کردن مدل قرار دارد. اما قبل از بررسی قسمت‌های بعدی `notebook`، خلاصه‌ی عملکرد مدل در آخرین `epoch`، `recall`، `precision`، `epoch`، شامل `mAP@.5:0.95` و ... در شکل ۱۰ قابل بررسی است.

Class	Images	Targets	P	R	mAP@.5	mAP@.5: .95: 100%
all	58	1.00e+03	0.93	0.922	0.926	0.575
blue	58	115	0.987	0.991	0.995	0.603
green	58	290	0.986	0.941	0.985	0.566
red	58	290	0.994	0.99	0.996	0.639
vline	58	136	0.96	0.978	0.98	0.85
white	58	58	0.672	0.638	0.602	0.174
yellow	58	116	0.982	0.991	0.995	0.619

شکل ۱۰- خلاصه‌ی عملکرد مدل در آخر `epoch`

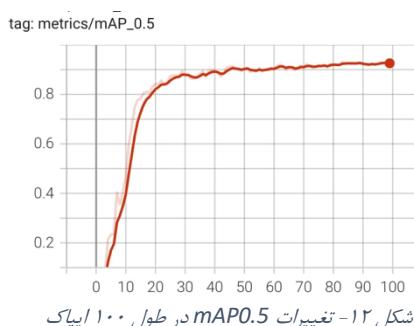
همانطور که مشخص است، در اکثر کلاس‌ها `precision` و `recall` بالای ۹۰ درصد داریم، بجز کلاس `white` که هر دوی آن‌ها در بازه‌ی ۰-۶۰ درصدی قرار دارند. با توجه به نتایج، به نظر می‌رسد که مدل، عملکرد قابل قبولی داشته است.

- در این قسمت ما با استفاده از ابزار `tensorboard` نمودارهای مختلفی که نمایانگر نحوه‌ی آموزش مدل، تغییرات `loss`ها، تغییرات دقیق و معیارهای مختلف ارزیابی و ... را رسم می‌کنیم.

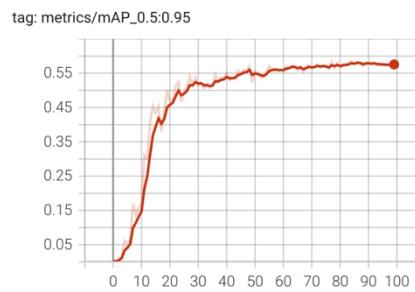
```
# Start tensorboard
# Launch after you have started training
# logs save in the folder "runs"
%load_ext tensorboard
%tensorboard --logdir runs
```

شکل ۱۱- استفاده از `tensorboard`

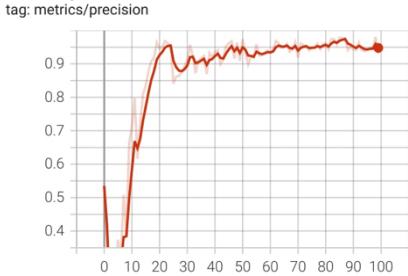
نمودارهای خواسته‌شده در صورت سوال، در این قسمت رسم می‌شوند که در ادامه، هر کدام را خواهیم دید.



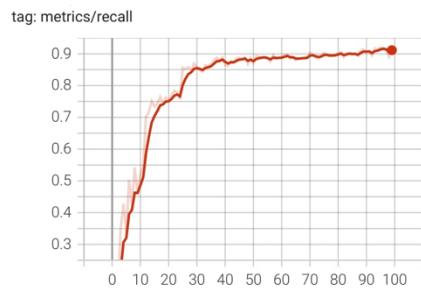
شکل ۱۲- تغییرات `mAP@0.5` در طول ۱۰۰ اپیک



شکل ۱۳- تغییرات  $mAP_{0.5:0.95}$  در طول ۱۰۰ اپیک



شکل ۱۴- تغییرات  $precision$  در طول ۱۰۰ اپیک



شکل ۱۵- تغییرات  $recall$  در طول ۱۰۰ اپیک

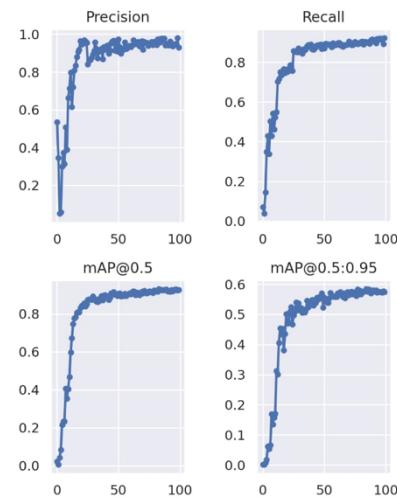
مشخصا تمامی این معیارها، در طول آموزش افزایش داشته‌اند و به مقادیری نسبتاً بالایی همگرا شده‌اند.

- این قسمت هم دقیقاً کارایی ای مشابه قبل دارد؛ اما بدون ابزار `tensorboard`. و ما می‌توانیم نمودارهای مورد نیاز را در این بخش هم ببینیم.

```
# we can also output some older school graphs if the tensor board isn't working for whatever reason
from utils.plots import plot_results # plot results.txt as results.png
Image(filename='/content/yolov5/runs/train/yolov5s_results10/results.png', width=1000)
```

شکل ۱۶- کد رسم پلات‌ها، بدون استفاده از `tensorboard`

نمودارهای رسم شده در قسمت قبل، در این بخش هم می‌توانند بدون استفاده از `tensorboard` به صورت زیر، رسم شوند.



شکل ۱۷- نمودارهای  $mAP_{0.5:0.95}$  و  $mAP_{0.5}$  بدون `tensorboard`

- در cell بعدی، ما تصاویر batch اول از دادگان تست را به همراه آن‌ها نمایش داده‌ایم.

```
# first, display our ground truth data
print("GROUND TRUTH TRAINING DATA:")
Image(filename='/content/yolov5/runs/train/yolov5s_results/test_batch0_labels.jpg', width=900)
```

شکل ۱۸ - رسم یک batch از دادگان تست، با آن‌ها

- در بخش بعدی هم مشابه قسمت قبل عمل کردیم، اما این بار با استفاده از دادگان .train

```
# print out an augmented training example
print("GROUND TRUTH AUGMENTED TRAINING DATA:")
Image(filename='/content/yolov5/runs/train/yolov5s_results/train_batch0.jpg', width=900)
```

شکل ۱۹ - رسم یک batch از دادگان آموزش

- در این قسمت، برای دادگان valid، مدل را با استفاده از وزن‌های آموزش دیده در قسمت قبل اجرا می‌کنیم (فایل detect.py)

```
# when we ran this, we saw .007 second inference time. That is 140 FPS on a TESLA P100!
# use the best weights!
%cd /content/yolov5/
!python detect.py --weights runs/train/yolov5s_results/weights/best.pt --img 416 --conf 0.4 --s
```

شکل ۲۰ - اجرای detect.py بر روی دادگان تست با وزن‌های آموزش دیده در قسمت‌های قبل

- بعد از اجرای detect.py با وزن‌های قبلی، می‌توانیم نتایج آن را در این قسمت، بر روی تصاویر مشاهده کنیم.

```
#display inference on ALL test images
#this looks much better with longer training above

import glob
from IPython.display import Image, display

for imageName in glob.glob('/content/yolov5/runs/detect/exp2/*.jpg'): #assuming JPG
    display(Image(filename=imageName))
    print("\n")
```

شکل ۲۱ - رسم تصاویر label از داده‌شده توسط وزن‌های قبلی

- و در نهایت وزن‌های استفاده شده را برای استفاده‌های بعدی، در drive، ذخیره می‌کنیم.

```
from google.colab import drive
drive.mount('/content/gdrive')
```

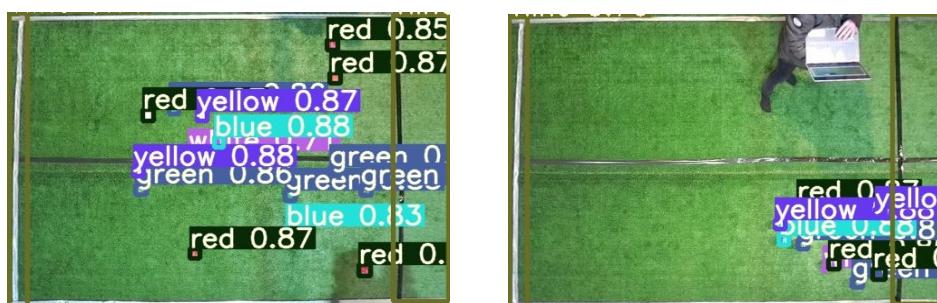
```
%cp /content/yolov5/runs/train/yolov5s_results/weights/best.pt /content/gdrive/My\ Drive
```

شکل ۲۲ - ذخیره‌ی وزن‌ها در drive برای استفاده‌های بعدی

برخی نتایج بدست آمده در این قسمت، گزارش نشده‌اند. چرا که در قسمت‌های بعدی خواسته شده؛ و ما آن‌ها را به همراه توضیحات، در ادامه خواهیم آورد.

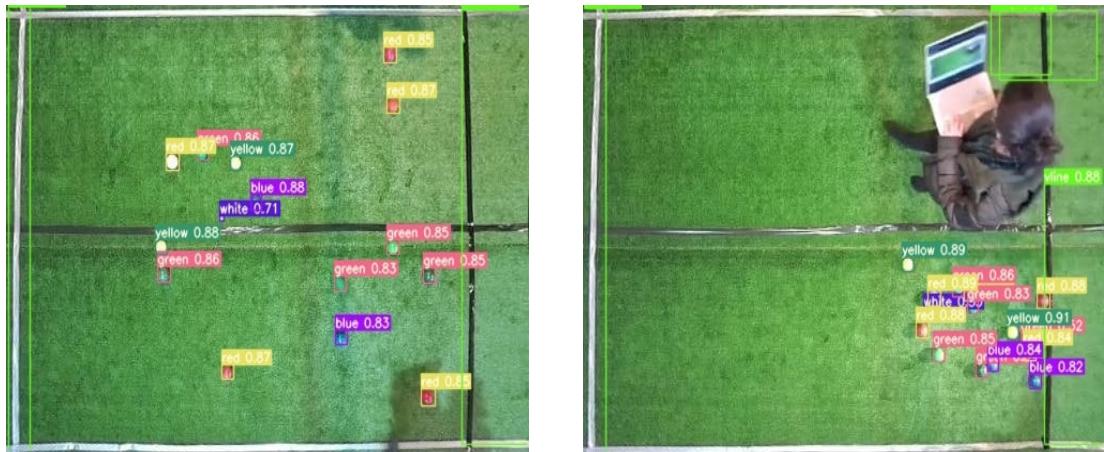
(۳)

دو نمونه از خروجی نتیجه‌ی شبکه بر روی دادگان تست، در زیر قابل مشاهده است:



شکل ۲۳ - دو نمونه از خروجی مدل روی دادگان تست

برای خواناتر شدن نتایج هم، با استفاده از پارامتر `line_tickness` می‌توانیم اندازه‌ی نوشته‌ها را کوچک‌تر کنیم تا نسبت به اندازه‌ی توپ‌ها، معقول‌تر باشد. برای مثال دو تصویر زیر نمایانگر این تغییر در فایل `detect.py` هستند.



سؤال ۲۴ - خروجی پس از کم کردن اندازه‌ی نوشته‌ها

(۴)

زمانی که عکس‌ها به مدل داده می‌شوند، ما در خروجی مختصات مشخص‌کننده‌ی مستطیل دور آن‌ها، اطمینان مربوط به `label` اختصاص‌داده شده و نام کلاس آن‌ها را خواهیم داشت.

برای انجام این بخش، ما با استفاده از مختصات مستطیل‌ها، مرکز هر توپ را بدست‌آورده‌یم. سپس مختصات توپ سفید را ذخیره کردیم و در بخشی که به عنوان `label` در کنار کلاس، اطمینان تصمیم را می‌نوشت (در بخش‌های قبلی) ما فاصله‌ی هر توپ، از توپ سفید را گزارش می‌کنیم. قطعه کدی که در زیر مشاهده می‌کنید، تنها بخش‌های تغییر کرده نسبت به قبل می‌باشد. که در فایل `detect2.py` قابل مشاهده است.

```
# Write results

def center(xyxy):
    return np.array([(xyxy[0]+xyxy[2])/2, (xyxy[1]+xyxy[3])/2])

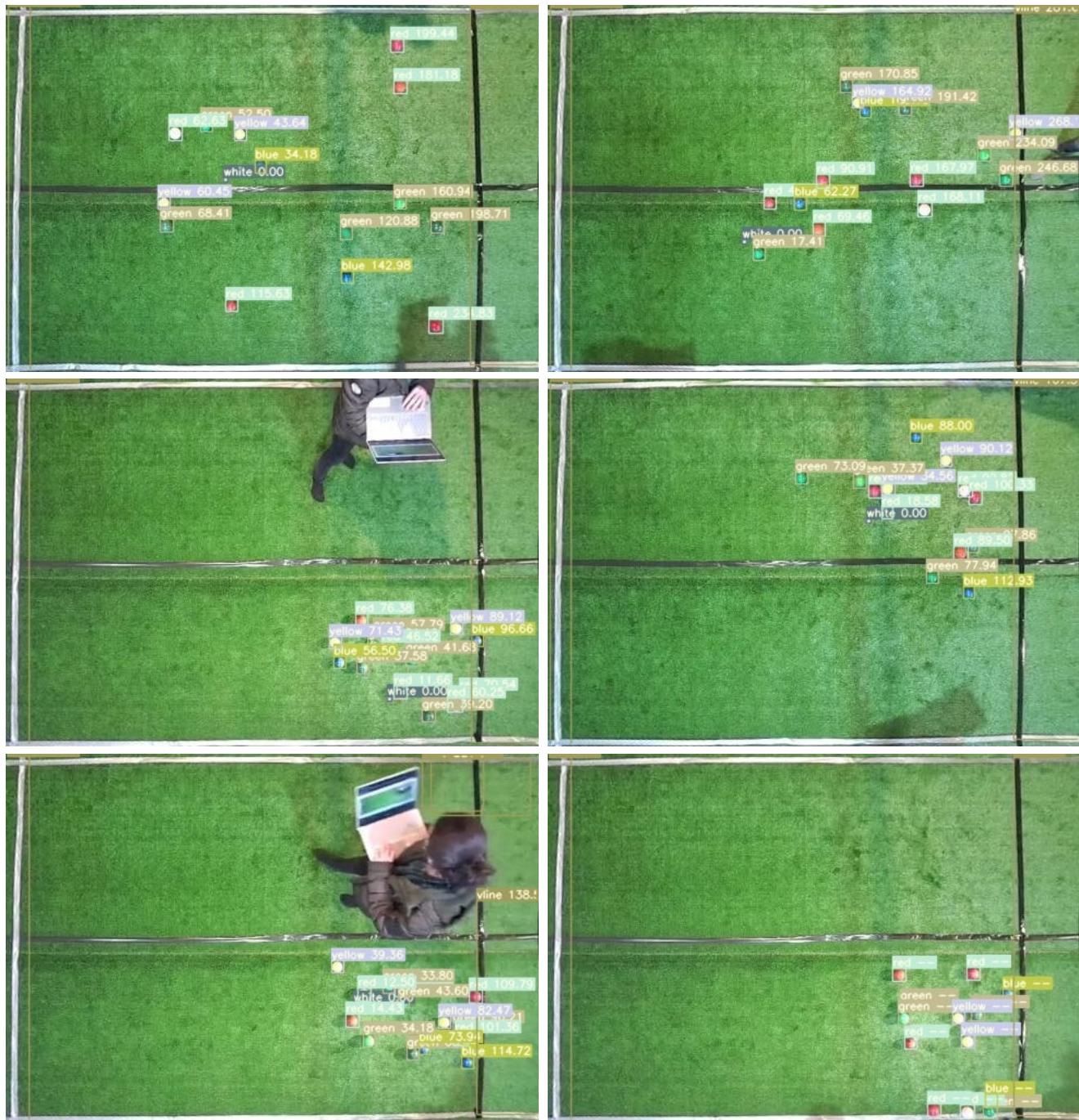
white_index = -1
white_xy = None
has_white = False
for i, (*xyxy, conf, cls) in enumerate(reversed(det)):
    if names[int(cls)] == 'white':
        white_index = i
        white_xy = center(xyxy)
        has_white = True
        break
for *xyxy, conf, cls in reversed(det):
    if save_txt: # Write to file
        xywh = (xyxy2xywh(torch.tensor(xyxy).view(1, 4)) / gn).view(-1).tolist() # normalized xywh
        line = (cls, *xywh, conf) if opt.save_conf else (cls, *xywh) # label format
        with open(txt_path + '.txt', 'a') as f:
            f.write(( '%s' * len(line)).rstrip() % line + '\n')

    if save_img or view_img: # Add bbox to image
        if has_white:
            label = f'{names[int(cls)]} {sum((center(xyxy)-white_xy)**2)**0.5:.2f}'
            plot_one_box(xyxy, im0, label=label, color=colors[int(cls)], line_thickness=1)
        else:
            label = f'{names[int(cls)]} --'
            plot_one_box(xyxy, im0, label=label, color=colors[int(cls)], line_thickness=1)
```

شکل ۲۵ - تغییرات ایجاد شده برای بدست‌آوردن فاصله از توپ سفید

در صورتی که توپ سفید در تصویر، تشخیص داده نشود، -- در کنار اسم کلاس چاپ می‌شود.

در عکس‌های زیر ۵ نمونه تصویر که توب سفید دارند، و یک تصویر بدون آن نمایش داده شده‌اند.

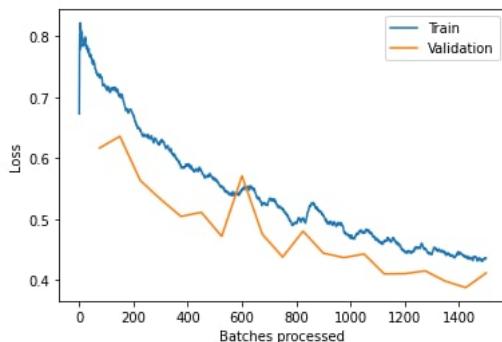


شکل ۲۶ - ۵ نمونه از فاصله‌ی توب‌ها با توب سفید، و یک نمونه که توب سفید آن تشخیص داده نشده

## Semantic Segmentation – ۲ سوال

(۱)

پیاده‌سازی این شبکه در فایل CamVid.ipynb قابل مشاهده است که با استفاده از fastAI از این مدل استفاده کرده‌ایم.  
نمودار تابع خطا برای دادگان آموزش و ارزیابی به شکل زیر است.



شکل ۲۷ - نمودار تغییرات تابع خطا برای دادگان آموزش و تست

(۲)

تصویر خواسته شده در زیر آورده شده است.

Ground truth/Predictions



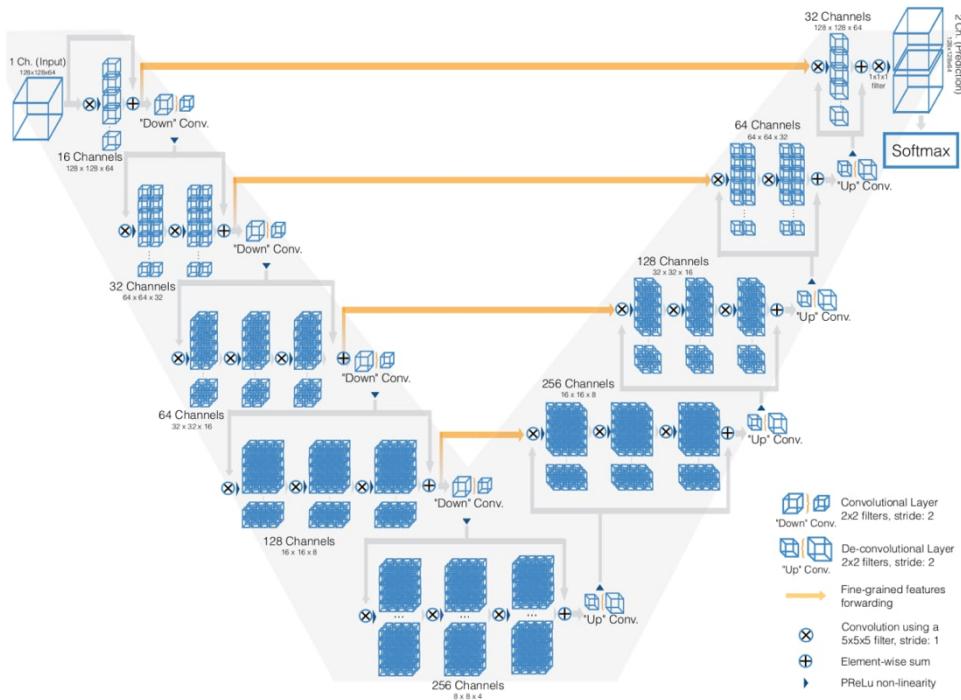
شکل ۲۸ - تصویر سگمنت های واقعی و سگمنت های تشخیص داده شده توسط شبکه

(۳)

به نظر می‌رسد که بعد از ۲۰ ایپاک، تقریباً شیب تغییرات loss نزدیک به صفر شده است و احتمالاً بیشتر از ۳۰ ایپاک نیاز به آموزش نخواهیم داشت.

شبکه‌ی (V-Net)

این شبکه یک convolutional neural network است که از convolution هم برای استخراج ویژگی و هم کاهش resolution در انتهای هر stage استفاده کرده است. سمت چپ معماری ای که در شکل ۲۷ نشان داده شده است، در واقع در حال فشرده سازی است. هر stage در این بخش در resolution های مختلفی کار می کند و یک residual function را یاد می گیرد (ورودی هر stage در لایه های کانولوشن استفاده می شود و بعد از اعمال توابعی غیرخطی، با خروجی آخرین لایه کانولوشن آن stage جمع می شود). این معماری تضمین می کند که در درصدی از زمان استفاده شده توسط شبکه های مشابه که یادگیری residual function را ندارند، همگرا می شود. شکل زیر شماتیکی کلی این معماری است.



شکل ۲۷- معماری شبکه V-Net

برای فرایند optimization، این شبکه از یک objective function بر مبنای Dice coefficient استفاده کرده است. به طور کلی در کاربردهای پزشکی، بسیار اتفاق می افتد که قسمت مورد علاقه برای بررسی، بخش خیلی کوچکی از تصویر باشد و آنگاه نتورک ها ممکن است در مینیمم محلی حاصل از توجه زیاد به قسمت های دیگر گیر بیفتند. با استفاده از dice coefficient می توان این مشکل را حل کرد.

این روش توانسته هم دقیق با اتاری بدست بیاورد و هم در زمان کوتاه تری به آن دقیق دست پیدا کند. نتایج مقایسه ای آن با سایر روش ها در جدول زیر گزارش شده است.

جدول ۱- مقایسه ای V-Net با دو logistic loss (یکی Dice-based objective function) با سایر روش ها

Algorithm	Avg. Dice	Avg. Hausdorff distance	Score on challenge task	Speed
V-Net + Dice-based loss	$0.869 \pm 0.033$	$5.71 \pm 1.20$ mm	82.39	1 sec.
V-Net + mult. logistic loss	$0.739 \pm 0.088$	$10.55 \pm 5.38$ mm	63.30	1 sec.
Imorphics [22]	$0.879 \pm 0.044$	$5.935 \pm 2.14$ mm	84.36	8 min.
ScrAutoProstate	$0.874 \pm 0.036$	$5.58 \pm 1.49$ mm	83.49	1 sec.
SBIA	$0.835 \pm 0.055$	$7.73 \pm 2.68$ mm	78.33	—
Grislies	$0.834 \pm 0.082$	$7.90 \pm 3.82$ mm	77.55	7 min.