به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

شبکه های عصبی و یادگیری عمیق

تمرین اکسترا سری یک

نام و نام خانوادگی	علی عدالت
شماره دانشجویی	۸۱۰۱۹۹۳۴۸
تاریخ ارسال گزارش	

فهرست گزارش سوالات (لطفاً پس از تکمیل گزارش، این فهرست را به‌روز کنید.)

سوال 1 – Object Detection with YOLOv5

۱

Yolov4 که توسط الکسی بوچکوفسکی منتشر شده است. تعداد زیادی ویژگی در آن وجود دارد که گفته می‌شود دقت شبکه عصبی کانولوشن (CNN) را بهبود می‌بخشد. آزمایش عملی ترکیبات چنین ویژگیهایی بر روی مجموعه دادههای بزرگ برای توجیه نظری نتیجه لازم است. برخی از ویژگی ها در مدل های خاص منحصرا و فقط برای برخی از مجموعه های داده در مقیاس کوچک کار می کنند. در حالی که برخی از ویژگی‌ها مانند نرمال سازی دسته‌ای و اتصالات باقیمانده (residual-connections) برای اکثر مدل‌ها، tasks و مجموعه داده‌ها قابل استفاده است. ویژگی‌های جدید در این مدل در Figure 1 آمده است.

New Features:

1. Weighted-Residual-Connections (WRC)
2. Cross-Stage-Partial-connections (CSP)
3. Cross mini-Batch Normalization (CmBN)
4. Self-adversarial-training (SAT)
5. Mish activation
6. Mosaic data augmentation
7. DropBlock regularization
8. CIoU loss

Figure 1 ویژگی‌های مورد استفاده در ورژن ۴

در مقایسه با YOLOv3 ، نسخه جدید متریک‌های AP (دقت) و FPS (نرخ فریم در ثانیه) را به ترتیب 10 و 12 درصد بهبود داده است. همچنین لایه بیشتری از شبکه عصبی کانولوشن (CNN) در YOLOv4 اضافه شده است. object detection از چندین قسمت تشکیل شده است. این قسمت‌ها در Figure 2 آمده است. تفاوت yolov4 با yolov3 فقط backbone است. زیرا yolov3 از Darknet53 در backbone استفاده می‌کند. اما yolov4 از CSPDarknet53 در backbone استفاده می‌کند. همه فکرهای دیگر استفاده شده در این مدل در مقایسه با yolov3 یکسان هستند.

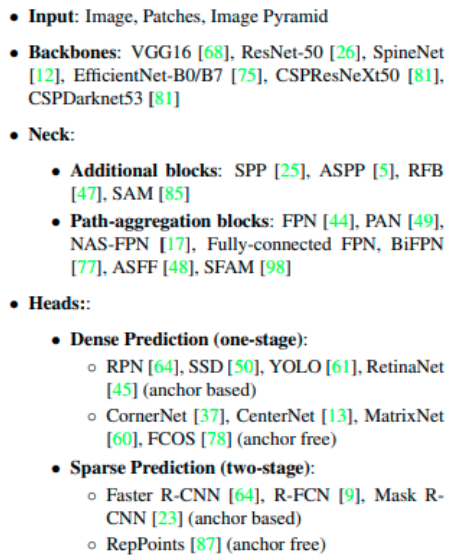

Figure 2 قسمت های تشکیل دهنده **object detection**

head های Yolov4 مانند yolov3 است. Heads قسمت پیش بینی است و دو نوع دارد. یکی پیش بینی متراکم (آشکارساز یک مرحله ای) و دیگری پیش بینی پراکنده (آشکارساز دو مرحله ای). آموزش custom object در yolov4 مانند yolov3 است. مواردی که در ورژن چهارم در بخش‌های مختلف برای تشخیص انجام می‌شود، در Figure 3 آمده است. کارهای که در این ورژن جدید در backbone انجام می‌شود نیز آمده است.

YOLO v4 uses:

- Bag of Freebies (BoF) for backbone: CutMix and Mosaic data augmentation, DropBlock regularization, Class label smoothing
- Bag of Specials (BoS) for backbone: Mish activation, Cross-stage partial connections (CSP), Multiinput weighted residual connections (MiWRC).
- Bag of Freebies (BoF) for detector: CIoU-loss, CmBN, DropBlock regularization, Mosaic data augmentation, Self-Adversarial Training, Eliminate grid sensitivity, Using multiple anchors for a single ground truth, Cosine annealing scheduler [52], Optimal hyperparameters, Random training shapes.
- Bag of Specials (BoS) for detector: Mish activation, SPP-block, SAM-block, PAN path-aggregation block, DIoU-NMS.

Figure 3 مواردی که در ورژن چهارم در بخش‌های مختلف برای تشخیص انجام می‌شود

پیشرفت های عمده در YOLO v4 : YOLO v4 از پیشرفته ترین BoF ها (bag of freebies) و چندین BoS (bag of specials) تأثیر می گیرد. BoF بدون افزایش زمان استنباط ، دقت ردیاب (object detector) را بهبود می بخشد. آنها فقط هزینه آموزش را افزایش می دهند. از طرف دیگر ، BoS هزینه استنباط را مقدار کمی افزایش می دهد ، اما به طور قابل توجهی دقت تشخیص اشیا را بهبود می بخشد. عملکرد YOLO v4 : YOLO v4 نیز بر اساس Darknet ساخته شده است و مقدار AP برابر 43.5 درصدی روی مجموعه COCO همراه با سرعت واقعی 65 FPS روی Tesla V100 بدست آورده است که از نظر سرعت و دقت سریعترین و دقیق ترین ردیاب های قبلی را شکست می دهد.

پیشرفت های عمده در YOLO v5 : YOLO v5 با سایر نسخه های قبلی متفاوت است ، زیرا این یک پیاده سازی PyTorch است و نه نسخه‌ای گرفته شده از Darknet اصلی. همانند YOLO v4، YOLO v5 دارای ستون فقرات CSP و گردن PA-NET است. پیشرفت های عمده شامل افزایش داده های (data augmentation) موزاییکی و یادگیری خودکار bounding box anchors است. در زیر جملاتی که پیاده سازان در باره عملکرد بیان کرده‌اند، مدل آمده است.

با اجرای ورژن ۵ روی تسلا P100 ، شاهد زمان استنباط تا 0.007 ثانیه در هر تصویر بودیم ، یعنی 140 فریم در ثانیه (FPS). در مقابل ، YOLO v4 پس از تبدیل به همان کتابخانه PyTorch، 50 FPS را بدست آورد. YOLO v5 از نظر تعداد وزن کوچک است. به طور خاص ، فایل وزن برای YOLO v5 برابر 27مگابایت است. فایل وزن ما برای YOLO v4 (با معماری Darknet) 244 مگابایت است. YOLO v5 تقریباً 90 درصد کوچکتر از YOLO v4 است.

بنابراین ، گفته می شود که YOLO v5 بسیار سریع و سبک تر از YOLO v4 است ، در حالی که دقت آن با معیار YOLO v4 برابر است. اما سوال اصلی مطرح شده توسط جامعه علمی این است: آیا این معیارها دقیق و قابل تکرار هستند؟

۲

برای آموزش مدل yolov5 از نوت بوک استفاده می‌کنیم. در ادامه گام‌های مورد نیاز برای آموزش این مدل را از نوت بوک بررسی می‌کنیم. ابتدا repository مربوط به مدل را download می‌کنیم که شامل ساختار شبکه و نخوه آموزش دادن شبکه و تست آن است. برای استفاده از مدل باید کتابخانه‌های مورد

نیاز را نصب کنیم که با توجه به فایل «requirements» در repository این کار را انجام می‌دهیم. برای استفاده از فایل zip مجموعه داده فقط باید بخش « Download Correctly Formatted Custom Dataset» را تغییر دهیم. در این بخش فایل zip را از حالت zip در می‌آوریم. برای این که تعداد کلاس داده‌ها و آدرس داده‌های تست و آموزش را بدست آوریم، باید فایل data.yaml را بخوانیم. تعداد کلاس که در مقابل کلید ns آمده است. داده‌های داخل این فایل در Figure 4 آمده است. در Figure 5 روش خواندن تعداد کلاس داده آمده است.

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

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

Figure 4 فایل **data.yaml**

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

Figure 5 روش خواندن تعداد کلاس داده

برای استفاده از مدل باید کانفیگ آن را تعیین کنیم و پارامترهای ساختار مدل را مشخص کنیم. یک فایل به نام «yolov5/models/yolov5s.yaml » وجود دارد که کانفیگ مدل و پارامترهای مدل را دارد. فقط در این فایل پارامتری مانند تعداد کلاس داده وجود دارد که باید با توجه به تعداد کلاس داده تغییر دهیم. دیگر پارامترهای فایل yolov5s.yaml مدبوط به anchor ها و ساختار بخش‌های مختلف مدل مانند اندازه کرنل و تعداد فیلترهای کانولوشنی است. در این فایل تنها باید تعداد کلاس را به تعداد کلاس داده تغییر دهیم. بعد از این تغییر، مقادیر پارامترهای جدید را در «yolov5/models/custom_yolov5s.yaml » می‌نویسیم. در Figure 6 داده‌های داخل فایل yolov5s آمده است.

بعد از تعیین پارامترهای مدل بر اساس مجموعه داده، می‌توانیم به آموزش مدل بر روی داده‌های آموزش بپردازیم. برای آموزش کافی است فایل train.py را با دادن پارامترهای مناسب اجرا کنیم. پارامترهایی که باید برای آموزش تعیین کنیم، تعداد epoch، اندازه batch ، اندازه تصویر ، مسیر فایل yaml برای تعیین تعداد کلاس ها و مسیر داده‌های آموزش و ارزیابی و مسیر فایل yaml برای تعیین پارامترهای ساختاری مدل است.

```
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, 9, C3, [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
  ]

# 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, C3, [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, C3, [256, False]],  # 17 (P3/8-small)

   [-1, 1, Conv, [256, 3, 2]],
   [[-1, 14], 1, Concat, [1]],  # cat head P4
   [-1, 3, C3, [512, False]],  # 20 (P4/16-medium)

   [-1, 1, Conv, [512, 3, 2]],
   [[-1, 10], 1, Concat, [1]],  # cat head P5
   [-1, 3, C3, [1024, False]],  # 23 (P5/32-large)

   [[17, 20, 23], 1, Detect, [nc, anchors]],  # Detect(P3, P4, P5)
  ]
```

Figure 6 داده‌های داخل فایل **yolov5s**

در Figure 7 نحوه آموزش مدل و دادن اطلاعات لازم برای آموزش آمده است.

Next, we'll fire off training!

Here, we are able to pass a number of arguments:

- **img:** define input image size
- **batch:** determine batch size
- **epochs:** define the number of training epochs. (Note: often, 3000+ are common here!)
- **data:** set the path to our yaml file
- **cfg:** specify our model configuration
- **weights:** specify a custom path to weights. (Note: you can download weights from the Ultralytics Google Drive fo
- **name:** result names
- **nosave:** only save the final checkpoint
- **cache:** cache images for faster training

```
# 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 ./models/custom_yolov5s.yaml
```

Figure 7 نحوه آموزش مدل

بعد از آموزش مدل به ارزیابی آن می‌پردازیم. نتایج آموزش در epoch آخر در Figure 8 آمده است.

```
     Epoch   gpu_mem       box       obj       cls     total   targets  img_size
     99/99     1.81G   0.04895   0.04021  0.008384   0.09754       270       416: 100% 99/99 [00:42<00:00,  2.35it/s]
                Class      Images     Targets           P           R      mAP@.5  mAP@.5:.95: 100% 2/2 [00:01<00:00,  1.01it/s]
                  all          58    1.00e+03       0.932        0.92        0.92       0.589
                 blue          58         115       0.988       0.991       0.995         0.6
                green          58         290       0.989        0.96       0.985       0.583
                  red          58         290       0.987       0.997       0.996       0.647
                vline          58         136       0.953       0.978       0.979       0.897
                white          58          58       0.673       0.603       0.569       0.175
               yellow          58         116           1       0.991       0.995       0.629
Optimizer stripped from runs/train/yolov5s_results/weights/last.pt, 14.8MB
Optimizer stripped from runs/train/yolov5s_results/weights/best.pt, 14.8MB
100 epochs completed in 1.215 hours.

CPU times: user 52.6 s, sys: 7.01 s, total: 59.6 s
Wall time: 1h 13min 32s
```

Figure 8 نتایج آموزش در **epoch** آخر

نمودارهای mAP0 .5، mAP0 95.0:5،.Percision و Recall برای آموزش در Figure 9 آمده است.

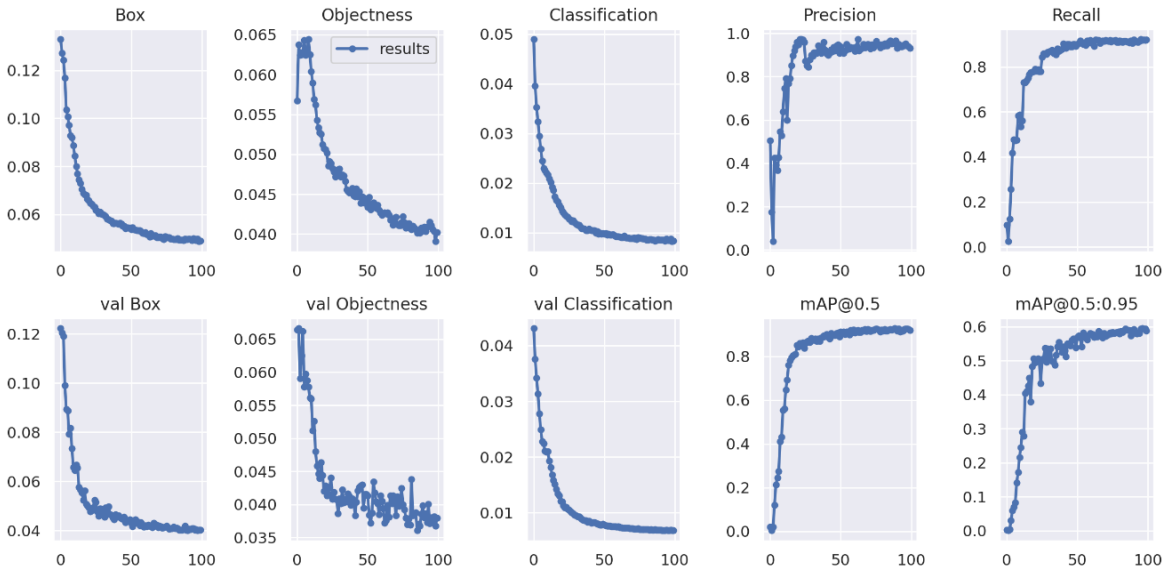

Figure 9 نمودارهای mAP0 .5، mAP0 95.0:5،.Percision و **Recall** در آموزش

همانطور که در Figure 9 دیده می‌شود، با انجام epoch بیشتر مقدار precision به طور کلی و recall افزایش یافته است. این روند نشان می‌دهد با انجام epoch های بیشتر یادگیری بیشتری انجام شده و به همین دلیل مقادیر این دو پارامتر افزایش یافته است. روند نمودار این دو پارامتر اینگونه است که در ابتدا افزایش می‌یابند و در آخر تقریبا ثابت می‌شوند و میل می‌کنند. این نشان می‌دهد که با انجام این تعداد یادگیری تقریبا کامل شده است و epoch بیشتر باعث یادگیری بهتری نمی‌شود. در Figure 8 می‌بینیم که بعد از انجام این epoch ها در تمام دسته‌ها به جز سفید مقدار دو پارامتر precision و recall بالای ۹۰ است. یعنی این دسته‌ها را به خوبی توانسته ایم از دیگر موارد مجزا کنیم. در دسته سفید مقادیر این دو

پارامتر بین ۶۰ تا ۷۰ درصد است. این یعنی یادگیری و مجزا سازی این دسته به خوبی انجام نشده است. روند دو نمودار mAP نیز صعودی است و در آخر میل می‌کند. مقادیر میل شده نسبتا بالا است و نشان می‌دهد که مدل یادگیری خوبی داشته است.

۳

در مجموعه داده فقط داده آموزش و ارزیابی داریم. از داده ارزیابی به عنوان تست استفاده می‌کنیم. در Figure 10 و Figure 11 دو نمونه از پاسخ مدل بر روی دو عکس از داده ارزیابی آمده است.

Figure 10 پاسخ مدل بر روی عکس اول از داده ارزیابی

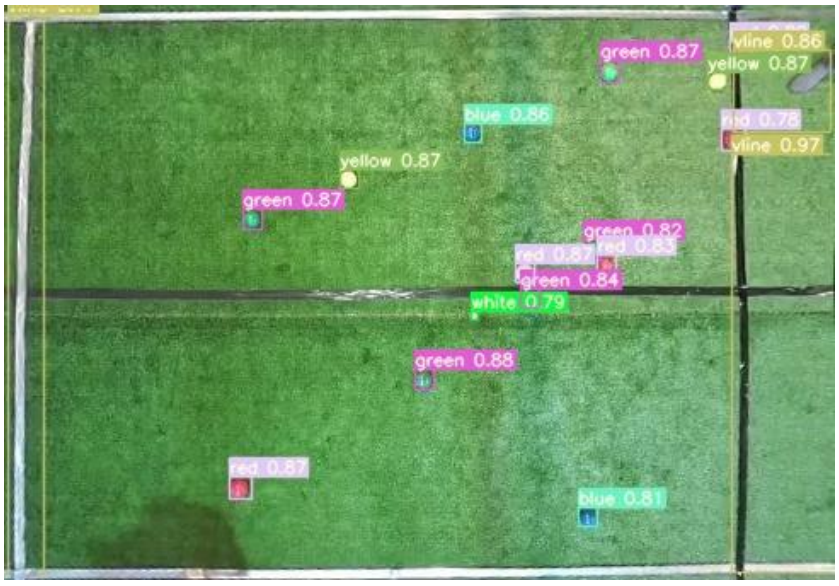

Figure 11 پاسخ مدل بر روی عکس دوم از داده ارزیابی

عکس‌های داده شده به مدل برای Object Detection در Figure 12 و Figure 13 آمده است.

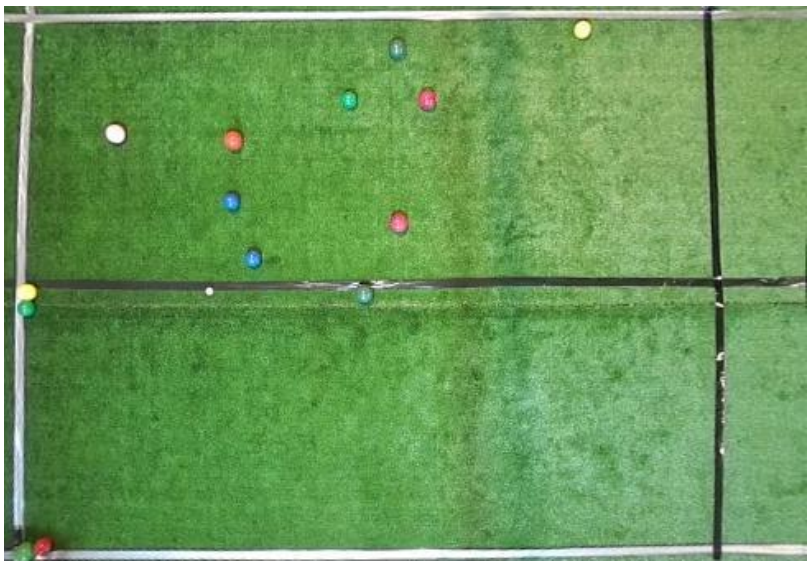

Figure 12 عکس اول ورودی

Figure 13 عکس دوم ورودی

همانطور که در Figure 12 و Figure 10 دیده می‌شود، یک گوی سفید به اشتباه قرمز تشخیص داده شده شده است. این موضوع ضعف مدل در تشخیص سفید که در بالا گفتیم را نشان می‌دهد. این مشاهده با مقدار نسبتا پایین و نزدیک ۶۰ درصد precision و recall که در قسمت قبل دیدیم، نیز مطابقت دارد. به همین دلیل ضعف است که یک گوی سفید درست و دیگری به اشتباه قرمز تشخیص داده شده است. در این ورودی دو گوی پر گوشه پایین تصویر است که مدل تشخیصی برای آن نداشته است. در مقایسه Figure 11 و Figure 13 نیز مشکل مدل در تشخیص گوی سفید را مشاهده می‌کنیم. باز یک گوی سفید به اشتباه قرمز تشخیص داده شده است. گوی های با رنگ غیر سفید در این دو حالت به درستی و با

اطمینان بالایی درست تشخیص داده شده اند. نکته درباره گوی سفید این است که گوی با حجم بیشتر اشتباه تشخیص داده شده و اطمینان از تصمیم اشتباه برای آن بالا است. برای گوی کوچکتر سفید اطمینان از تصمیم بالا بوده است. در دیگر تشخیص‌های مدل برای داده‌های ارزیابی نیز همین موارد را می‌بینیم.

برای خواناتر کردن تصویرهای خروجی مدل باید ضخامت box های دور موارد تشخیص داده شده را تغییر دهیم. به صورت پیش فرض ضخامت جعبه دور هر شی برابر ۳ بود. ما یک پارامتر ضخامت اضافه کردیم که مقدار این پارامتر از طریق ورودی دادن در هنگام اجرای detect.py تعیین می‌شود. با تغییر این پارامتر، دریافتیم که ضخامت کمتر یعنی ضخامت ۱ باعث می‌شود که خروجی خواناتر شود. خروجی های نشان داده شده در این قسمت با همین ضخامت یک هستند. فایل detect.py با این تغییرات انجام شده در پوشه Q1 و در پوشه «1-3» مربوط به این بخش در کنار گزارش ارسال شده است. تغییرات انجام شده در Figure 14 آمده است.

```
if save_img or view_img:  # Add bbox to image
    label = f'{names[int(cls)]} {conf:.2f}'
    plot_one_box(xyxy, im0, label=label, color=colors[int(cls)], line_thickness=opt.line_thickness)
```

```
parser.add_argument('--exist-ok', action='store_true', help='existing project/name ok, do not increment')
parser.add_argument('--line-thickness', default=3, type=int, help='bounding box thickness (pixels)')
```

Figure 14 تغییرات انجام شده برای خواناتر کردن خروجی **detect.py**

۴

مدل به ازای تشخیص هر شی، یک box یا جعبه برای دور آن تعیین می‌کند که شی در وسط آن قرار دارد. مدل برای تعیین این جعبه تنها دو نقطه از ابتدا و انتهای یک قطر از این box را تعیین می‌کند. با توجه به این موضوع برای تشخیص وسط شی تشخیصی کافی است که نقطه وسط دو نقطه تعیین کننده جعبه را پیدا کنیم. در اینجا می‌خواهیم که فاصله مراکز گوی‌ها از گوی سفید را بدست آوریم. برای اینکار، اولین تشخیص گوی سفید را به عنوان معیار تعیین می‌کنیم و فاصله مراکز دیگر گوی‌ها از این معیار را بدست می‌آوریم. گوی‌ها را بر اساس این فاصله مرتب می‌کنیم و خروجی می‌دهیم. در Figure 15 تغییرات انجام شده در detect.py برای این تعیین فاصله و مرتب کردن و تهیه detect2.py آمده است.

در این بخش ۵ تصویر اول در مجموعه داده‌های ارزیابی را انتخاب می‌کنیم و detect2.py را روی آنها اعمال می‌کنیم. در ادامه خروجی‌ها برای این عکس‌ها را می‌بینیم. فایل وزن‌های مدل آموزش دیده و detect2.py در پوشه Q1 و «4-1» به همراه عکس‌های خروجی ورودی‌ها آمده است.

```
# Write results
import numpy as np
refw = -1
pps = []
for *xyxy, conf, cls in reversed(det):
    cpm1, cpm2 = np.array([int(xyxy[0]), int(xyxy[1])]),\
      np.array([int(xyxy[2]), int(xyxy[3])])

    if names[int(cls)] != 'vline':
      if names[int(cls)] == 'white' and refw == -1:
        refw = (cpm1+cpm2)/2
      else:
        cenop = ((cpm1+cpm2)/2)
        pps.append([np.linalg.norm(cenop-refw), cls])


pps = sorted(pps, key=lambda x: x[0])
for i in pps:
  print(f'{names[int(i[1])]} distance: {i[0]}')
```

Figure 15 تغییرات انجام شده در **detect.py** برای این تعیین فاصله و مرتب کردن و تهیه **detect2.py**

در Figure 16 عکس خروجی برای عکس اول ارزیابی و در Figure 17 خروجی رنگ گوی‌ها به همراه فاصله از گوی سفید آمده است.

Figure 16 عکس خروجی برای عکس اول ارزیابی

```
image 1/58 /content/yolov5/../valid/images/104_jpg.rf.922cc5f61fc87ab619f35377de76820c.jpg: blue distance: 33.503731
blue distance: 56.632587791835896
red distance: 93.70832406995656
green distance: 95.01184136727379
yellow distance: 111.00112612041376
red distance: 112.07698247187065
red distance: 122.60199835239229
green distance: 144.91117969294157
red distance: 176.76325975722443
green distance: 184.11205826887058
green distance: 186.81541692269406
yellow distance: 277.02436715928076
288x416 2 blues, 4 greens, 4 reds, 3 vlines, 1 white, 2 yellows, Done. (0.010s)
```

Figure 17 خروجی رنگ گوی‌ها به همراه فاصله از گوی سفید برای عکس اول

در Figure 18 عکس خروجی برای عکس دوم ارزیابی و در Figure 19 خود عکس ورودی و در Figure 20 خروجی رنگ گوی‌ها به همراه فاصله از گوی سفید آمده است.

Figure 18 عکس خروجی برای عکس دوم ارزیابی

Figure 19 عکس دوم ارزیابی

```
image 2/58 /content/yolov5/../valid/images/106_jpg.rf.741aa61c5acfb884a2e5d91e5951d70f.jpg: green distance: 396.7581
red distance: 397.9610533708041
red distance: 403.1330425554323
red distance: 408.5979074836287
yellow distance: 412.48151473732736
green distance: 419.48927280682636
green distance: 424.2549351510245
red distance: 431.6167281280928
yellow distance: 436.4152265904571
green distance: 441.624840786838
red distance: 442.900101603059
blue distance: 446.29278506379643
288x416 1 blue, 4 greens, 5 reds, 3 vlines, 2 yellows, Done. (0.008s)
```

Figure 20 خروجی رنگ گوی‌ها به همراه فاصله از گوی سفید برای عکس دوم ارزیابی

در Figure 21 عکس خروجی برای عکس سوم ارزیابی و در Figure 22 خروجی رنگ گوی‌ها به همراه فاصله از گوی سفید آمده است.

Figure 21 عکس خروجی برای عکس سوم ارزیابی

```
image 3/58 /content/yolov5/../valid/images/110_jpg.rf.c3404cd59249da5dd4262697cf06778d.jpg: red distance: 8.90224690
yellow distance: 39.092838218783754
green distance: 42.08622102303793
red distance: 55.80322571321482
red distance: 61.56500629416032
red distance: 69.6419413859206
green distance: 71.56814934033156
green distance: 114.11945495838998
blue distance: 119.32413837945782
yellow distance: 121.43002099975112
red distance: 131.07726728918328
green distance: 140.75865870347016
blue distance: 142.79005567615695
green distance: 146.91919547833086
288x416 2 blues, 5 greens, 5 reds, 2 vlines, 1 white, 2 yellows, Done. (0.008s)
```

Figure 22 خروجی رنگ گوی‌ها به همراه فاصله از گوی سفید برای عکس سوم ارزیابی

در Figure 23 عکس خروجی برای عکس چهارم ارزیابی و در Figure 24 خروجی رنگ گوی‌ها به همراه فاصله از گوی سفید آمده است.

Figure 23 عکس خروجی برای عکس چهارم ارزیابی

```
image 4/58 /content/yolov5/../valid/images/114_jpg.rf.a67c6b727ec53130ac57c7380c9b7071.jpg: green distance: 48.31407
red distance: 131.87304500920573
red distance: 137.55453463990202
red distance: 186.81541692269406
red distance: 190.41074024329615
green distance: 200.19115864593022
green distance: 213.40630262482878
blue distance: 248.13554763475545
yellow distance: 298.5339176710077
red distance: 319.5473360865335
green distance: 342.4616766880639
green distance: 394.46799616698945
yellow distance: 426.7718945760135
288x416 1 blue, 5 greens, 5 reds, 2 vlines, 1 white, 2 yellows, Done. (0.008s)
```

Figure 24 خروجی رنگ گوی‌ها به همراه فاصله از گوی سفید برای عکس چهارم

در عکس Figure 25 خروجی برای عکس پنجم ارزیابی و در Figure 26 خود عکس ورودی و در Figure 27 خروجی رنگ گوی‌ها به همراه فاصله از گوی سفید آمده است.

Figure 25 خروجی برای عکس پنجم ارزیابی

Figure 26 عکس پنجم

```
image 5/58 /content/yolov5/../valid/images/117_jpg.rf.13a5d6ae58d59d25813cb7882f9a924c.jpg: red distance: 24.1919408
green distance: 24.520399670478458
red distance: 43.8092455995307
red distance: 44.30011286667337
green distance: 58.077534382926416
green distance: 65.07111494357538
red distance: 74.92829905983453
green distance: 76.9041611357929
yellow distance: 91.22773701018787
blue distance: 407.31560245097415
red distance: 430.8137648683013
blue distance: 435.90193851369827
green distance: 444.6484566486203
red distance: 445.52497124179246
288x416 2 blues, 5 greens, 6 reds, 3 vlines, 1 white, 1 yellow, Done. (0.008s)
```

Figure 27 خروجی رنگ گوی‌ها به همراه فاصله از گوی سفید برای عکس پنجم

سوال ۲ – Semantic Segmentation

در اینجا ابتدا مجموعه داده را دانلود می‌کنیم. داده‌ها در یک پوشه و لیبل‌های داده‌ها در یک پوشه دیگر است. لیبل هر عکس داده هم نام عکس است و فقط «_L» در انتهای نام لیبل داده است. برای لود مجموعه داده تمام عکس‌ها و لیلبل‌ها را در پوشه قرار می‌دهیم. تمام لیبل‌ها را با «_L» تشخیص می‌دهیم و از داده‌ها جدا می‌کنیم. داده‌ها را بر اساس نام فایل مرتب می‌کنیم تا در لیست نام لیبل‌ها در هر خانه مانند i لیبل متناظر داده در لیست داده‌ها و در خانه i وجود داشته باشد. بعد از این تشخیص داده‌ها و لیبل‌های متناظر، داده‌ها را لود می‌کنیم. برای لود عکس‌ها آنها را به اندازه 256 × 256 در می‌آوریم و به صورت tensor لود می‌کنیم تا بتوانیم از gpu استفاده کنیم. در Figure 28 تعدادی داده در کنار لیبل متناظر آمده است.

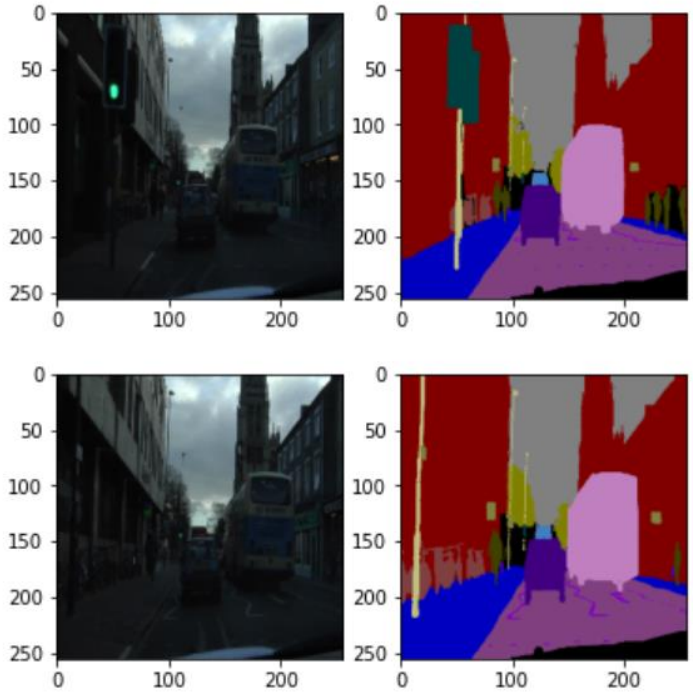

Figure 28 تعدادی داده در کنار لیبل متناظر

بعد از لود داده‌ها باید داده های تست و آموزش را از هم جدا کنیم و بر روی هر دسته داده iterator قرار دهیم. برای جدا سازی این دو دسته، 20 درصد داده‌ها را به عنوان تست در نظر می‌گیریم. ۲۰ درصد داده‌های بر خورده از انتها را به عنوان تست و ما بقی را به عنوان آموزش انتخاب می‌کنیم. برای لود عکس‌ها آنها را نرمالیزه می‌کنیم. هر خانه را تقسیم بر ۲۵۵ می‌کنیم. لیبل‌های ما عکس رنگی است. برای یادگیری باید این عکس‌های لیبل را به صورت بردار one-hot در بیاوریم. به همین دلیل تمام رنگ‌های ممکن در

عکس لیبل را به همراه نام هر شی با با هر رنگ را از فایل txt کنار داده می‌خوانیم. در این اینجا برای هر خانه از عکس لیبل با توجه به رنگ، یک بردار one-hot در نظر می‌گیریم. این بردار به ازای خانه تعیین کننده رنگ خانه لیبل برابر یک است. در بقیه خانه‌های بردار صفر است. پس لیبل هر عکس به صورت یک ماتریس عددی در آمد. برای یادگیری از ساختار UNet مطابق ساختار توضیح داده شده در مقاله استفاده می کنیم. برای تولید داده با ابعاد دو برابر از upsampling با متد nearest استفاده می‌کنیم. با روش نزدیک ترین همسایه سعی می‌کند خانه‌های جدید در اثر دو برابر کردن ابعاد ورودی را پر کند. بعد از ساخت مدل با تابع هزینه categorical_crossentropy به یادگیری می‌پردازیم.

۱

در Figure 29 نمودار تغییرات loss و accuracy برای داده‌های تست و آموزش در طی epoch ها آمده است. در اینجا از اندازه batch برابر یک برای آموزش استفاده کردیم و از ۵۰ epoch برای تکرار دیدن کل داده‌های آموزش و ۳۰۰ گام برای بررسی داده‌های آموزش در هر epoch و ۷۰ گام برای بررسی داده‌های تست بعد از هر epoch استفاده کردیم.

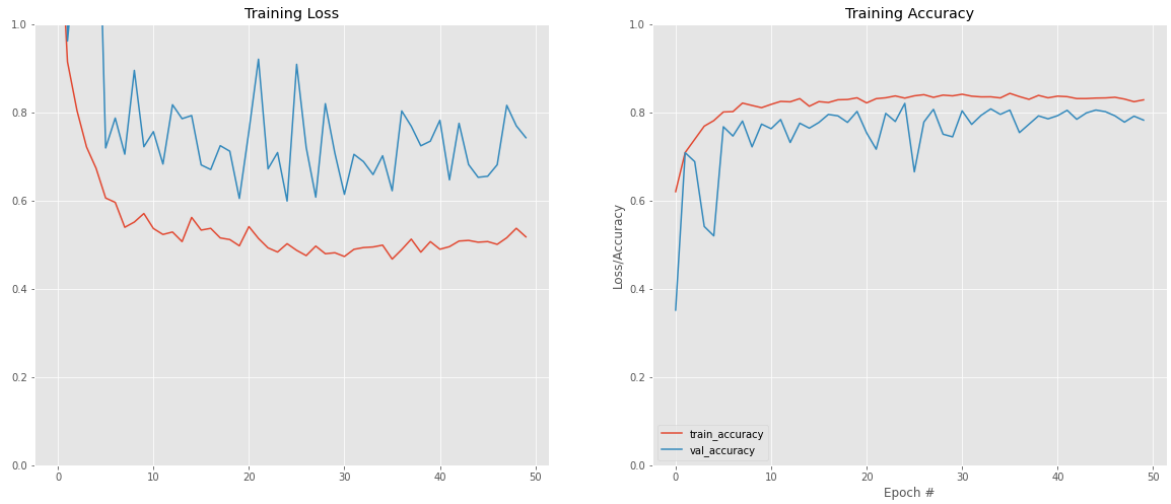

Figure 29 نمودار تغییرات **loss** و **accuracy** برای داده‌های تست و آموزش در طی **epoch** ها

همانطور که دیده می‌شود، با افزایش تعداد epoch ها مقدار accuracy در داده‌های آموزش افزایش یافته و بعد از ۱۵ epoch نمودار دقت داده آموزش به مقدار نزدیک ۸۳ درصد میل کرده است. این نشان می‌دهد که روند یادگیری خوبی داشته ایم و بعد ۱۵ تا epoch یادگیری بیشتری انجام نشده است. در طی این epoch ها دقت مدل روی داده‌های تست هم آمده است. نمودار دقت روی تست نیز ابتدا روندی صعودی با نوسان داشته است و حدودا بعد از ۱۵ تا epoch روند کلی تغییر آن به یک مقدار ۸۰ درصد

میل کرده است. در نمودار تغییرات loss در Figure 29 می‌بینیم که نمودار loss داده آموزش روندی کاهشی داشته و بعد از ۱۵ تا epoch به مقدار نزدیک 0.5 میل کرده است. بعد از آن نتوانسته هزینه را کمتر کند و یادگیری بیشتری نداشته اینم. نمودار loss روی داده تست نیز آمده است. مقدار کمینه loss روی داده تست در epoch شماره 25 رخ داده است. در این جا ما بهترین مدل با بهترین تعمیم را داریم که عملکرد خوبی روی داده دیده نشده تست دارد. در این جا دقت مدل روی داده تست 82.06 درصد است.

۲

یک عکس از داده تست به همراه پیش بینی مدل برای آن و لیبل اصلی آن در Figure 30 آمده است.

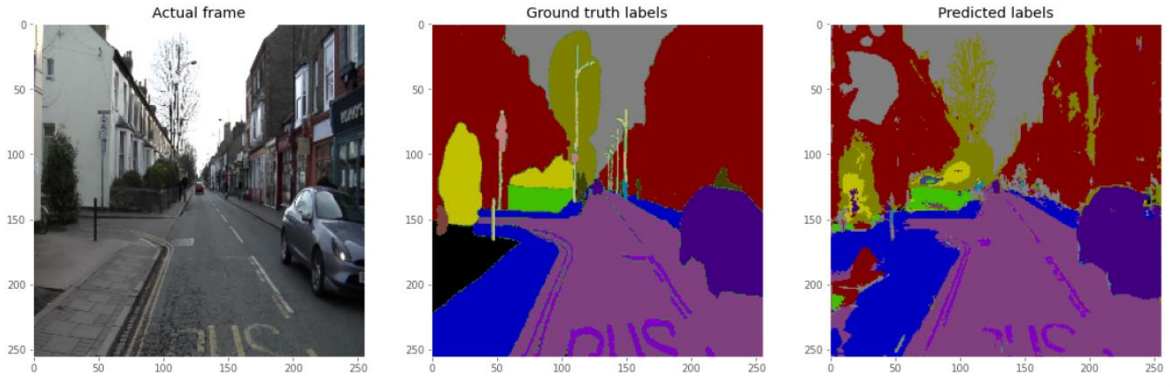

Figure 30 عکس از داده تست به همراه پیش بینی مدل برای آن و لیبل اصلی آن

همانطور که دیده می‌شود کلیت پیش بینی مانند لیبل اصلی است ولی در برخی جزئیات تفاوت دارد. این تفاوت در جزئیات نشان می دهد که دقت مدل روی داده تست صد درصد نیست. شباهت کلیت و تفاوت در برخی جزئیات نشان می‌دهد که یادگیری خوبی داشته ایم و نزدیک دقت و پیش بینی ۱۰۰ درصد هستیم. در اینجا از بهترین مدل با کمترین loss روی داده تست برای پیش بینی استفاده شده است. یک اشتباه در پیش بینی، سمت چپ بالا رخ داده که به جای رنگ قرمز رنگ آسمان خاکسری در نظر گرفته است. دلیل آن نزدیک بودن رنگ خانه در این ناحیه از تصویر با رنگ آسمان بوده است.

۳

برای بررسی تعداد epoch مورد نیاز برای یادگیری خوب از ModelCheckpoint استفاده می‌کنیم. ModelCheckpoint مقدار loss روی داده تست را مانیتور می‌کند. هر بار که این مقدار کمتر شود، مدل

بهتر از حالت قبل است و وزن‌های مدل را نگه می‌دارد. بهترین مدل، مدلی است که کمترین loss روی تست را باعث شود. پس تعداد ۵۰ epoch آموزش انجام می‌دهیم و بعد از هر epoch مقدار loss روی داده تست را بدست می‌آوریم. هر بار که مقدار loss از کمترین مقدار loss قبلی کمتر شود، وزن‌های مدل را نگه می‌داریم. آخرین وزنی را که نگه می‌داریم مربوط به مدل با کمترین loss روی تست و بهترین مدل است. تعداد epoch ای که لازم است تا این کمترین loss و بهترین مدل بدست آید، تعداد epoch لازم مناسب برای آموزش است. در اینجا بعد از 25 تا epoch دیگر مقدار loss روی داده تست کمتر نمی‌شود و کمینه مقدار loss روی تست در 25 امین epoch رخ می‌دهد. آخرین باری که وزن بهترین مدل به روز می‌شود مربوط به epoch شماره 25 است. پس 25 epoch برای آموزش مناسب است.

بعد از لود داده‌ها باید داده های تست و آموزش را از هم جدا کنیم و بر روی هر دسته داده iterator قرار دهیم. برای جدا سازی این دو دسته، 20 درصد داده‌ها را به عنوان تست در نظر می‌گیریم. ۲۰ درصد داده‌های بر خورده از انتها را به عنوان تست و ما بقی را به عنوان آموزش انتخاب می‌کنیم.

شبکه V-Net

در اینجا V-Net به طور خلاصه بررسی می شود. بیشتر داده های پزشکی مورد استفاده در عمل بالینی شامل حجم سه بعدی است. برای مثال MRI که به صورت حجمی پروستات را به تصویر می کشد. در حالی که اکثر رویکردها فقط قادر به پردازش segmentation تصاویر دو بعدی هستند. segmentation تصویر سه بعدی بر اساس حجم با یک شبکه عصبی کاملا کانولوشنال، در این کار ارائه شده است.

یک چالش اساسی که این نوع شبکه برای حل آن بررسی شده، بررسی بیماری در پروستات بر اساس عکس MRI است. segmentation تصاویر MRI پروستات به دلیل طیف گسترده ای از ظاهر ، همچنین رویکردهای مختلف اسکن ، یک کار چالش برانگیز است. تشخیص بر اساس segmentation این تصاویر بسیار از نظر هزینه تشخیص بیماری می‌تواند کمک کننده باشد.

معماری این شبکه در Figure 31 آمده است. V-Net در Figure 31 نشان داده شده است. قسمت سمت چپ شبکه از یک مسیر فشرده سازی تشکیل شده است ، در حالی که قسمت سمت راست سیگنال را از حالت فشرده خارج می کند تا به اندازه اصلی آن برسد. همانطور که مشاهده می کنید، شبیه U-Net است اما تفاوت هایی با آن دارد.

سمت چپ شبکه به مراحل مختلفی تقسیم شده است که در رزولوشن های مختلف کار می کنند. هر مرحله شامل یک تا سه لایه کانولوشن است. در هر مرحله ، یک عملکرد باقی مانده (residual function) یاد گرفته می شود. ورودی هر مرحله در لایه های کانولوشن استفاده می شود و از طریق بخش غیرخطی نیز پردازش می شود و به منظور ایجاد امکان عملکرد باقیمانده (residual function)، به خروجی آخرین لایه کانولوشن آن مرحله اضافه می شود. این معماری همگرایی را در مقایسه با شبکه یادگیری غیر باقیمانده مانند U-Net تضمین می کند.

در هر مرحله کانولوشنی از هسته های (kernel) حجمی با اندازه $5 \times 5 \times 5$ وکسل استفاده می کنند. در اینجا وکسل یک واحد یا خانه در grid در فضای سه بعدی است. یک مکعب با طول یک است. در طول مسیر فشرده سازی ، رزولوشن با کانولوشن با هسته های $2 \times 2 \times 2$ وکسل گسترده ای با گام 2 کاهش می یابد ، بنابراین ، اندازه feature maps حاصل به نصف کاهش می یابد هدف این کار مانند لایه pooling است. تعداد کانال های ویژگی در هر مرحله از مسیر فشرده سازی V-Net دو برابر می شود.

جایگزینی عملیات pooling با عملیات کانولشن به داشتن حافظه کمتری در طول آموزش کمک می کند. به این دلیل که برای پخش مجدد و رسیدن از خروجی لایه های استخر به ورودی های آنها، نیازی به نگه داری نقشه سوئیچ نیست. Down sampling به افزایش receptive field کمک می کند. PReLU به عنوان تابع فعال سازی غیر خطی استفاده می شود.

سمت راست شبکه شبکه ویژگی‌ها را استخراج می‌کند و پشتیبانی مکانی از نقشه های ویژگی با وضوح پایین را به منظور جمع آوری و سرهم کردن اطلاعات لازم برای خروجی دادن تقسیم بندی (segmentation) حجمی دو کاناله گسترش می دهد. از اطلاعات با رزولوشن کم بدست آمده از سمت چپ برای استخراج ویژگی استفاده می‌کند. در هر مرحله ، برای افزایش اندازه ورودی ها از عملیات deconvolution استفاده می‌شود که بعد از آن یک تا سه لایه کانولوشنی استفاده می‌شود. با استفاده از این عملیات‌ها در هر مرحله سعی می‌کنیم که feature map های با اندازه بزرگتر بسازیم و در انتها به اندازه ورودی برسیم. در طی این افزایش اندازه بر اساس اطلاعات مربوط به رزولوشن پایین ویژگی‌ها اسخراج می‌شوند. در این طرف نیز ساختار residual مانند طرف دیگر یاد گرفته می‌شود. در نهایت در سمت راست دو feature map با اندازه برابر ورودی داریم. این feature map ها segmentation احتمالاتی برای پیش زمینه و پس زمینه در تصویر با توجه به نگاه در سطح وکسل است.

مشابه U-Net ، اطلاعات مکان در مسیر فشرده سازی (چپ) از بین می رود. بنابراین ، ویژگی های استخراج شده از مراحل اولیه قسمت چپ CNN از طریق اتصالات افقی به قسمت راست هدایت می شوند.

این می تواند به ارائه اطلاعات موقعیت مکانی در قسمت مناسب و بهبود کیفیت پیش بینی کانتور نهایی کمک کند. و این اتصالات باعث بهبود زمان همگرایی مدل می شوند.

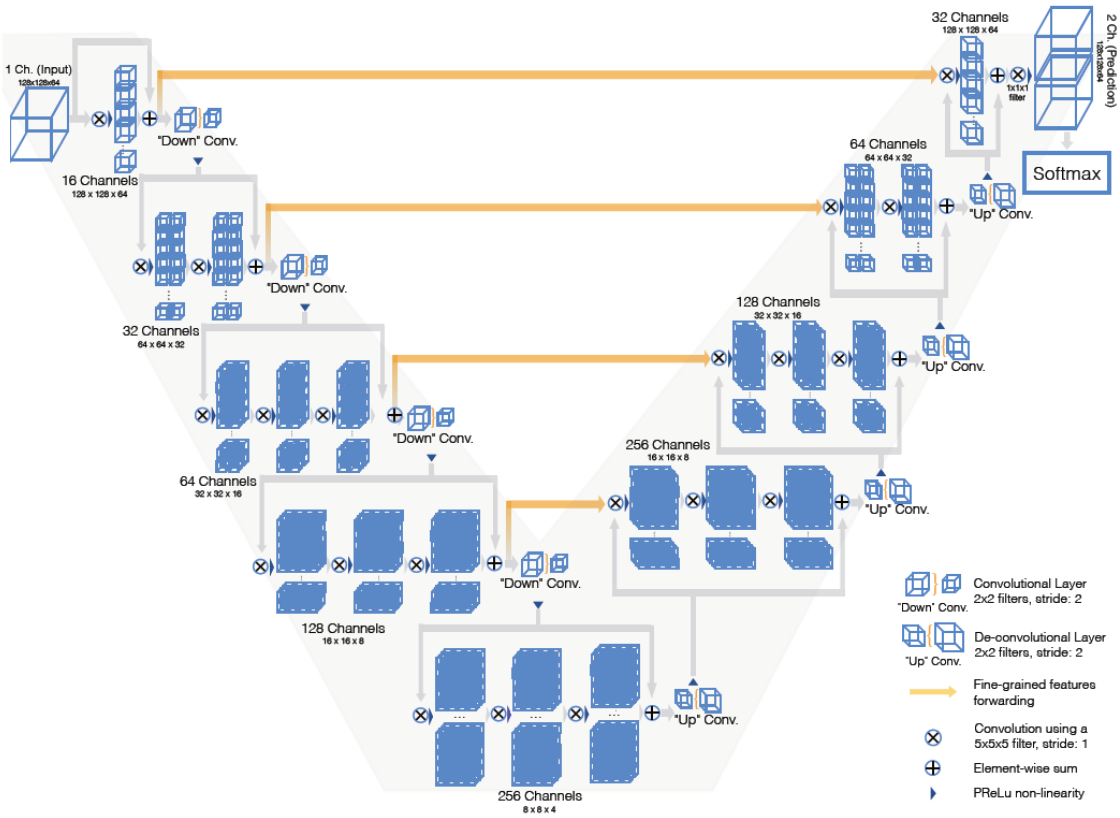

Figure 31 معماری شبکه **v-net**

تابع هزینه استفاده شده برای آموزش و بررسی عملکرد این شبکه در Figure 32 آمده است که Dice Loss است. این ضریب بین دو حجم دو دویی است. در اینجا N تعداد کل وکسل‌های مورد بررسی و Pi پیش بینی برای وکسل و gi مقدار واقعی برای وکسل است. ما در پیش بینی احتمال تعلق هر وکسل به پیش زمینه و پس زمینه را پیش بینی می‌کنیم.

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

Figure 32 تابع خطا استفاده شده

با استفاده از این تابع، به وزن نمونه های کلاس های مختلف برای ایجاد تعادل مناسب بین واکسل های پیش زمینه و پس زمینه نیازی نیست. چون این تابع مانند معیار f1-score می‌تواند برای مجموعه داده غیر متعادل بین دو کلاس استفاده شود. پیش بینی های شبکه، که از دو جلد تشکیل شده است با وضوح مشابه داده های ورودی اصلی هستند از طریق یک softmax پردازش می شود که احتمال متعلق بودن هر وکسل به پیش زمینه و پس زمینه را تولید می کند. در حجم های پزشکی مانند مواردی که ما بررسی می‌کنیم، آناتومی مورد علاقه فقط یک منطقه بسیار کوچک از اسکن را اشغال می کند. این امر غالباً باعث می شود که فرایند یادگیری در حداقل محلی از تابع هزینه گیر کند. پیش بینی ها به شدت نسبت به پس زمینه تعصب دارند. در نتیجه منطقه پیش زمینه غالباً گم شده است یا فقط تا حدی شناسایی می شود. در چندین رویکرد قبلی با دادن وزن متفاوت به اشتباه درباره پیش زمینه و پس زمینه اهمیت متفاوت داده اند. در یادگیری جایی که پیش زمینه است نسبت به مناطق پس زمینه اهمیت بیشتر داده می شود. در این کار یک تابع هدف جدید پیشنهاد شده است. کمیتی که بین 0 تا 1 است و ما آن را حداکثر می کنیم. در این تابع هزینه بالانس نبودن پیش زمینه و پس زمینه در نظر گرفته شده و این موضوع در یادگیری تاثیری نمی‌گذارد. این تابع مشخص می‌کند احتمال پس زمینه و پیش زمینه بودن هر وکسل چه قدر به واقعیت نزدیک است. به همین دلیل دوست داریم تابع را حداکثر کنیم تا پیش بینی شباهت بیشتری به واقعیت پیداکند. با گرادیان این تابع سعی می‌کنیم در جهتی حرکت کنیم که این هدف محقق شود.

نتایج این شبکه در Figure 33 آمده است. 30 حجم MRI دیده نشده به عنوان تست بررسی می شود. واکسل های بعد از سافت مکس، با داشتن احتمال بیشتر (< 0.5) مربوط به پیش زمینه نسبت به پس زمینه ، بخشی از آناتومی محسوب می شوند. ضریب dice و فاصله hausdorff اندازه گیری می شود. فاصله Hausdorff برای اندازه گیری شباهت شکل است. فاصله Hausdorff برای بدست آوردن حداکثر فاصله بین دو شکل است. از این دو معیار برای مقایسه پیش بینی و واقعیت استفاده می‌شود. می‌خواهیم شکل آناتومی پیش بینی شده مانند واقعیت باشد و آناتومی به درستی در تصویر مشخص شود.

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

Figure 33 نتایج شبکه

همانطور که در Figure 33 نشان داده شده است، V-Net با استفاده از تابع هزینه dice از V-Net با تابع لجستیک بهتر عمل می کند. V-Net از بسیاری از روش های قبلی بهتر عمل می کند.