به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

شبکه های عصبی و یادگیری عمیق

تمرین Extra 1

نام و نام خانوادگی	مهسا مسعود
شماره دانشجویی	810196635
تاریخ ارسال گزارش	00/02/17

فهرست گزارش سوالات

سوال 1 – Object Detection with YOLOv5

در این سوال می خواهیم با شبکه YOLOv5 آشنا شویم و بتوانیم با استفاده از آن Object detection انجام دهیم.

دیتاست این مسئله شامل تصاویری میشود که مربوط به بازی bocco ball است که کلاس های ما در این Dataset میشوند:

1- توپ های سفید
2- توپ های قرمز
3- توپ های سبز
4- توپ های آبی
5- توپ های زرد
6- خط های عمودی زمین

1)

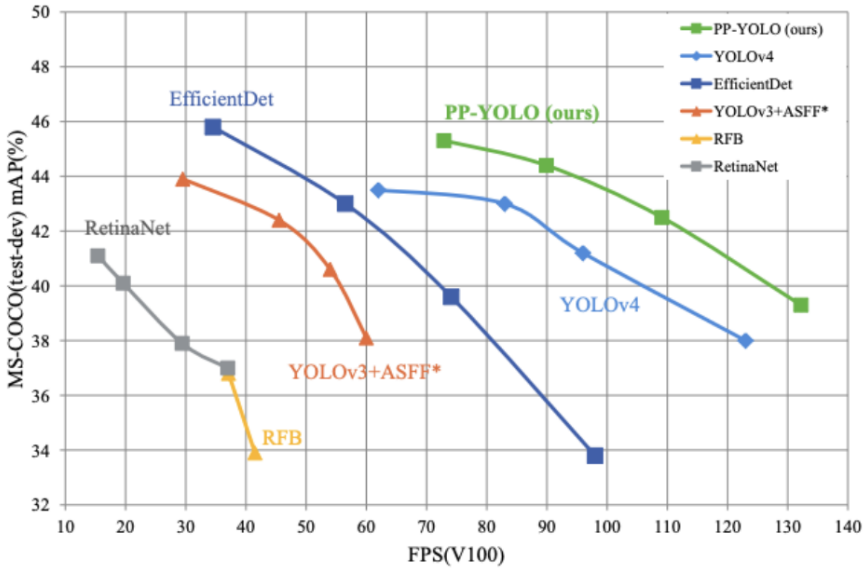
ابتدا به تعریف مختصری از YOLO میپردازیم و سپس ویژگی های کلی نسخه های 1و2و3 آن را بیان میکنیم و سپس به پیشرفت های ورژن 4 و 5 میپردازیم.

YOLO به منظور تشخیص Object استفاده میشود و میتواند با دقت مناسبی این امر را انجام دهد، در سال 2016، YOLOv1 با عنوان You Only Look Once (با این مفهوم که همانند چشم های ما که در لحظه میبینند و پردازش توسط مغز ما صورت میگیرد عمل میکنند) معرفی شد که توانایی Real time برای تشخیص اجسام داشت، بعد از آن در سال 2017 ورژن دوم آن با نام YOLOv2 معرفی شد که نه تنها سریع تر بود بود بلکه دقت Robustness بیشتری نیز داشت، در سال 2018 نیز نسخه جدید آن با نام YOLOv3 معرفی شد که Incremental Improvement در آن ایجاد شد.

در سال های 2019 توسط Alexey Bochkovskiy و 2020 توسط Glenn Jocher ورژن های بعدی یعنی YOLOv4 و YOLOv5 معرفی شدند که در آن ها شاهد تغییرات بسیار زیادی بودیم.

YOLOv4 امکان Train کردن با دقت بالا را روی GPU با 1080 TI یا 2080 TI را فراهم میکرد. همچنین تاثیر "bag-of-freebies" state of art و "bag-of-specials" که متد هایی برای تشخیص object هستند verified شد و این متد ها که شامل CBN و PAN با این ورژن هم efficient تر عمل میکنند هم برای کار با gpu میتوان آن ها را suitable تر دانست.

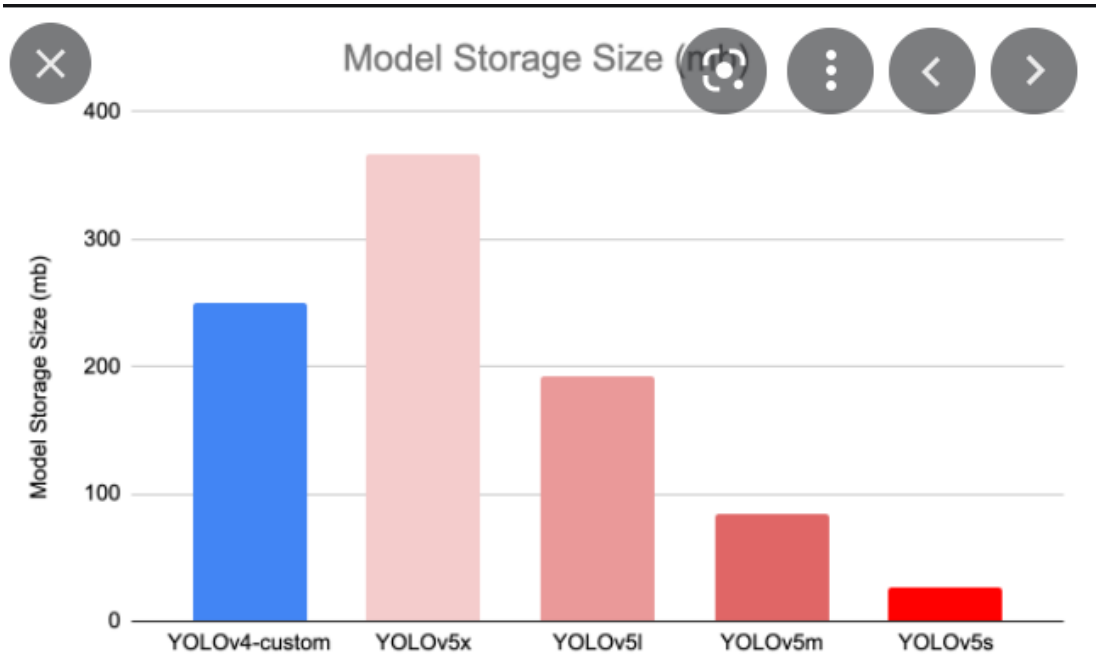
در تصویر نمونه ای از تشخیص object را مشاهده میکنیم که تفاوت YOLOv4 و YOLOv3 کاملا مبین و مشخص شود



شکل 1-1-1 عملکرد دو ورژن YOLO روی داده MS COCO

همانگونه که قابل مشاهده است، YOLOv5 با حدود 65FPS هم دقت بالاتری (43درصد) نسبت به ورژن قبلی خود داشته است که این عدد 33درصد است.

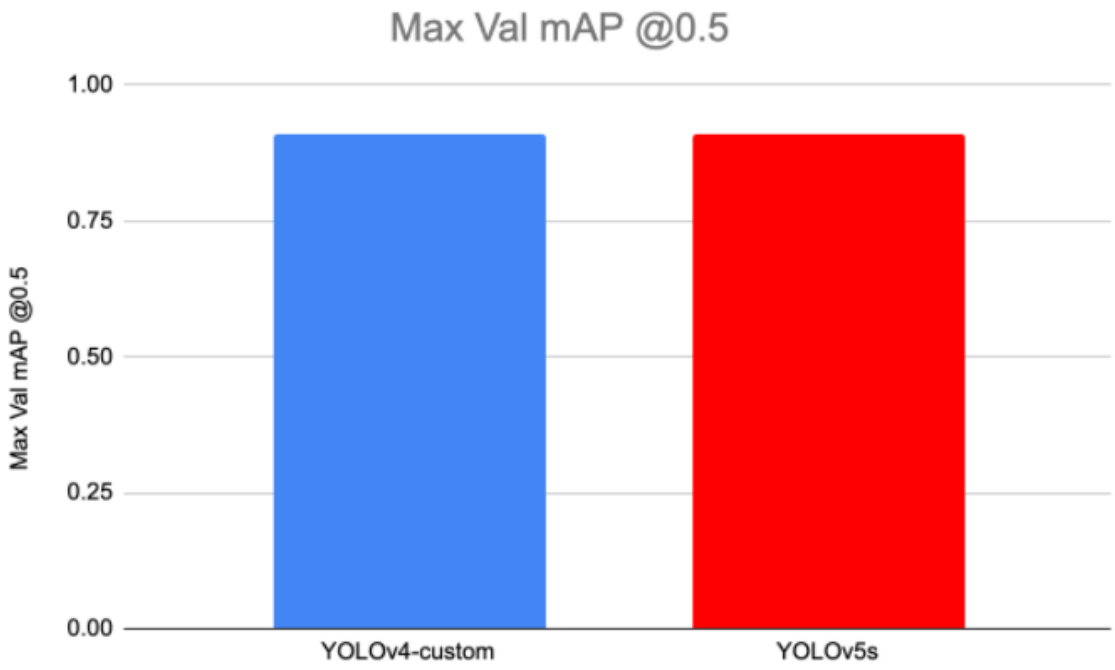
YOLOv5 در سوی دیگر، مزیت های YOLOv4 را دارد و سریعتر نیز عمل میکند و نتایج به صورت زیر شده است:



شکل 1-1-2 مقایسه زمانی YOLOv5 و YOLOv4

مشاهده میشود که YOLOv5 سریعتر بوده است.

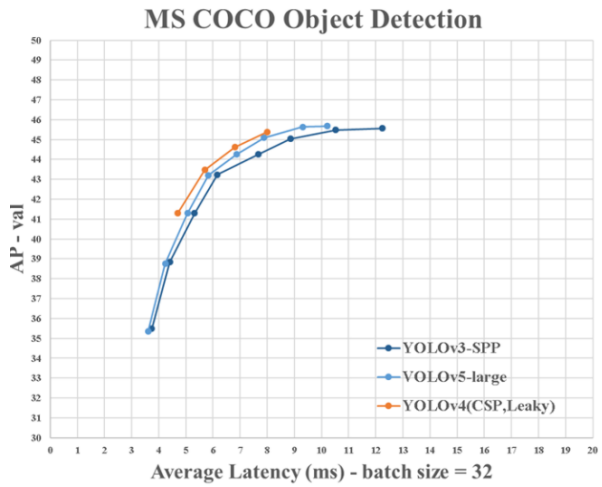
در همین آزمایش، دقت این دو ورژن هم سنجیده شده است، که نتایج به صورت زیر بوده است:



شکل 1-1-3 مقایسه دقت YOLOv5 و YOLOv4

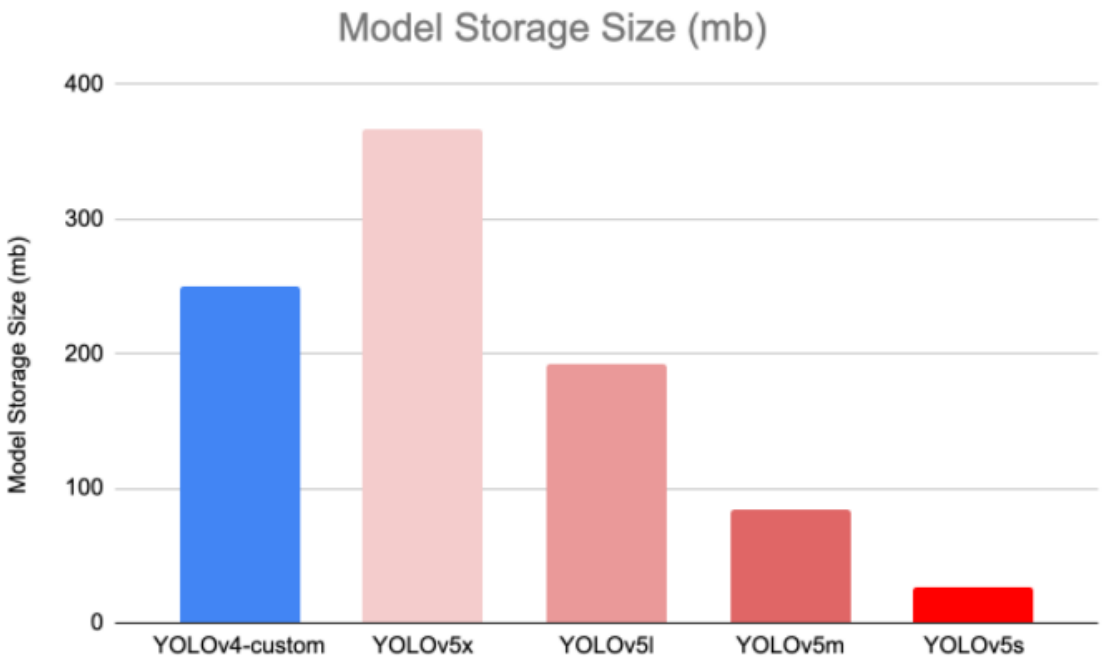
همانطور که قابل مشاهده میباشد هر دو ورژن دقت خوبی دارند.

همچنین گاهی نیز دقت با YOLOv4 بهتر نیز بدست می آید به عنوان مثال، در دیتاست MS COCO شاهد این قضیه هستیم:



شکل 1-1-4 مقایسه دقت YOLOv5 و YOLOv4 و YOLOv3 برای دیتاست MS COCO

میتوان از لحاظ حجم اطلاعات ذخیره شونده نیز این دو ورژن را مقایسه نمود،



شکل 1-1-5 مقایسه حافظه YOLOv5 و YOLOv4

مشاهده میشود که حافظه مورد نیاز برای ذخیره سازی وزن نیز برای YOLOv4 بسیار بیشتر از YOLOv5 بوده است.

به طور کلی برای سرعت بیشتر و دقت قابل قبول و در نظر گرفتن رابط کاربری آسان تر، از YOLOv5 استفاده میشود و برای task های با دقت ملزوم بالا، از YOLOv4 در darknet استفاده میشود.

(2

1- Install Dependencies and import libraries:

در این مرحله میبایست مخزن (Repository) YOLOv5 Github را clone کنیم و بعد از آن پوشه yolov5 به قسمت Google colab Files اضافه میشود، همچنین برای پاک کردن messy index files و working tree از git reset --hard استفاده میشود.

سپس کتابخانه torch و تعدادی Method را به محیط کار اضافه میکنیم

2- Load and unzip data:

فایل های زیر در صورت سوال آورده است:

1- فایل train که حاوی فایل تصاویر و label های آن ها است
2- فایل valid که حاوی فایل تصاویر و label های آن ها است
3- فایل data.yaml
4- فایل txt. که حاوی اطلاعات مخصوص preprocess هایی است که روی این dataset انجام شده است.

با استفاده از train.py با محتویات فایل train آموزش خواهیم داد و سپس از طریق detect.py محتویات فایل valid را به آزمون میگذاریم و نتایج را روی آن تست میکنیم.

3)

در این بخش با استفاده از فایل data.yaml تعداد کلاس ها و label مخصوص آن ها را در میبابیم.

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

شکل 1-2-2 اطلاعات کلاس ها برای آموزش و تست دیتا ست

6 کلاس داریم که به ترتیب آبی، سبز، قرمز، خط، سفید و زرد هستند.

Train:

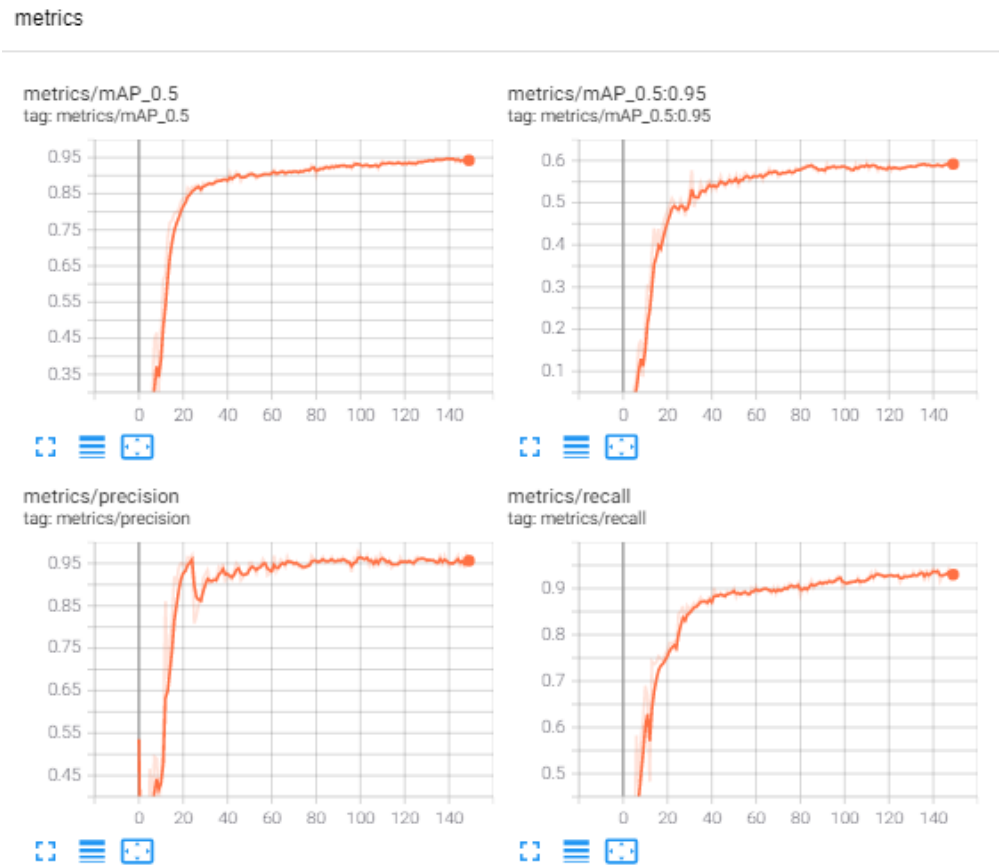
برای تمرین داده، از دستور مربوطه استفاده میکنیم. برای این دستور میتوان پارامتر هایی را لحاظ نمود که به تشریح در زیر آورده شده اند:

1- img که در واقع سایز تصویر را میگیرد که در پروژه 420 است
2- Batch که سایز batch را میگیرد که در پروژه 16 است
3- Epoch که تعداد ایپاک را میگیرد، در پروژه 150 لحاظ شد است
4- Data که مسیر data.yaml را میگیرد
5- Cfg که configuration مدل ما را مشخص میکند
6- Weights که مسیر مشخصه وزن است
7- Name نام نتیجه مدل است
8- No-save که نقطه نهایی چک-پوینت را ذخیره میکند
9- Cache برای استفاده از تصاویر cache برای یادگیری هر چه سریعتر

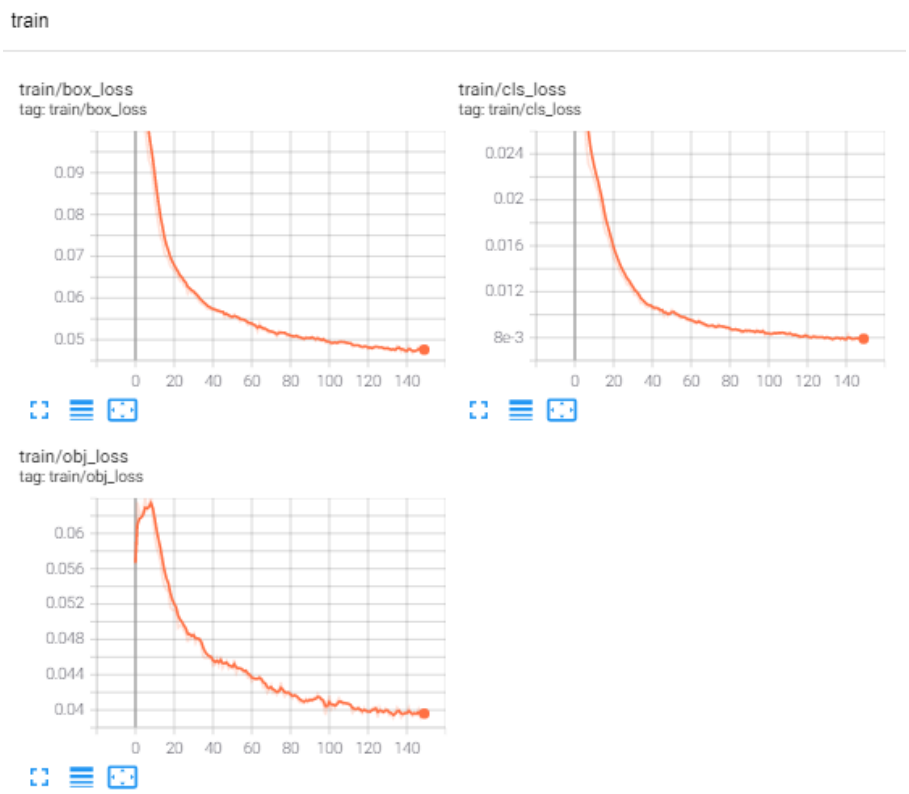
پس از 150 ایپاک آموزش داریم:

- مقدار Precision برابر با 0.956
- مقدار Recall برابر با 0.93
- مقدار mAP0.5 برابر با 0.941
- مقدار mAP0.5:0.95 برابر با 0.594

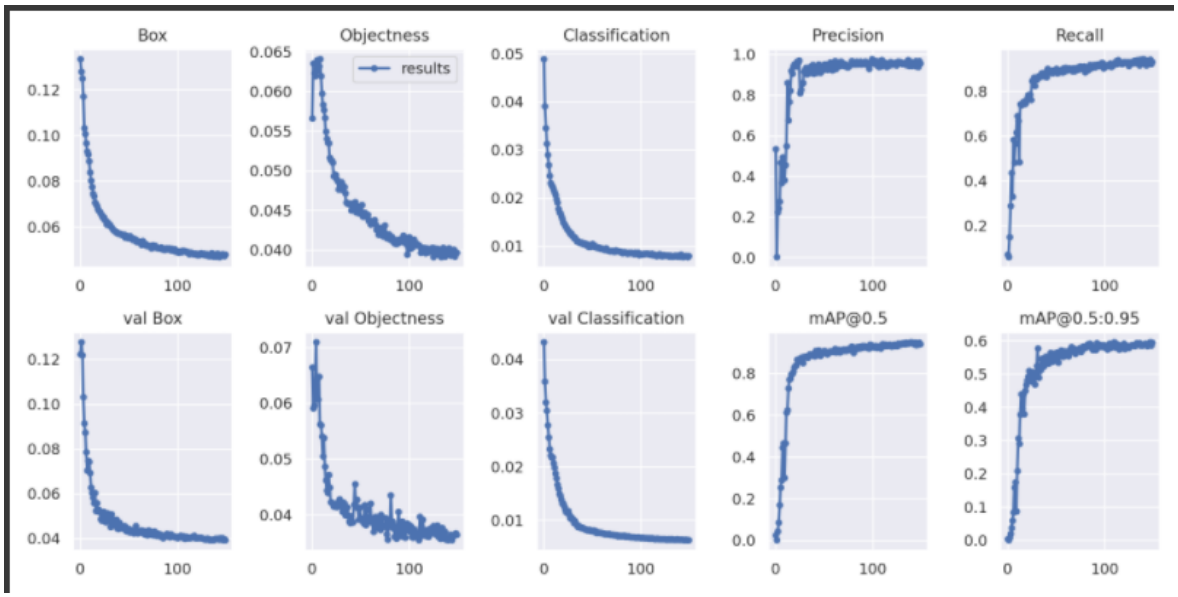
نمودار های metric ها به صورت زیر است.



شکل 1-2-4 نتایج نموداری مدل با تمرین در 150 ایپاک

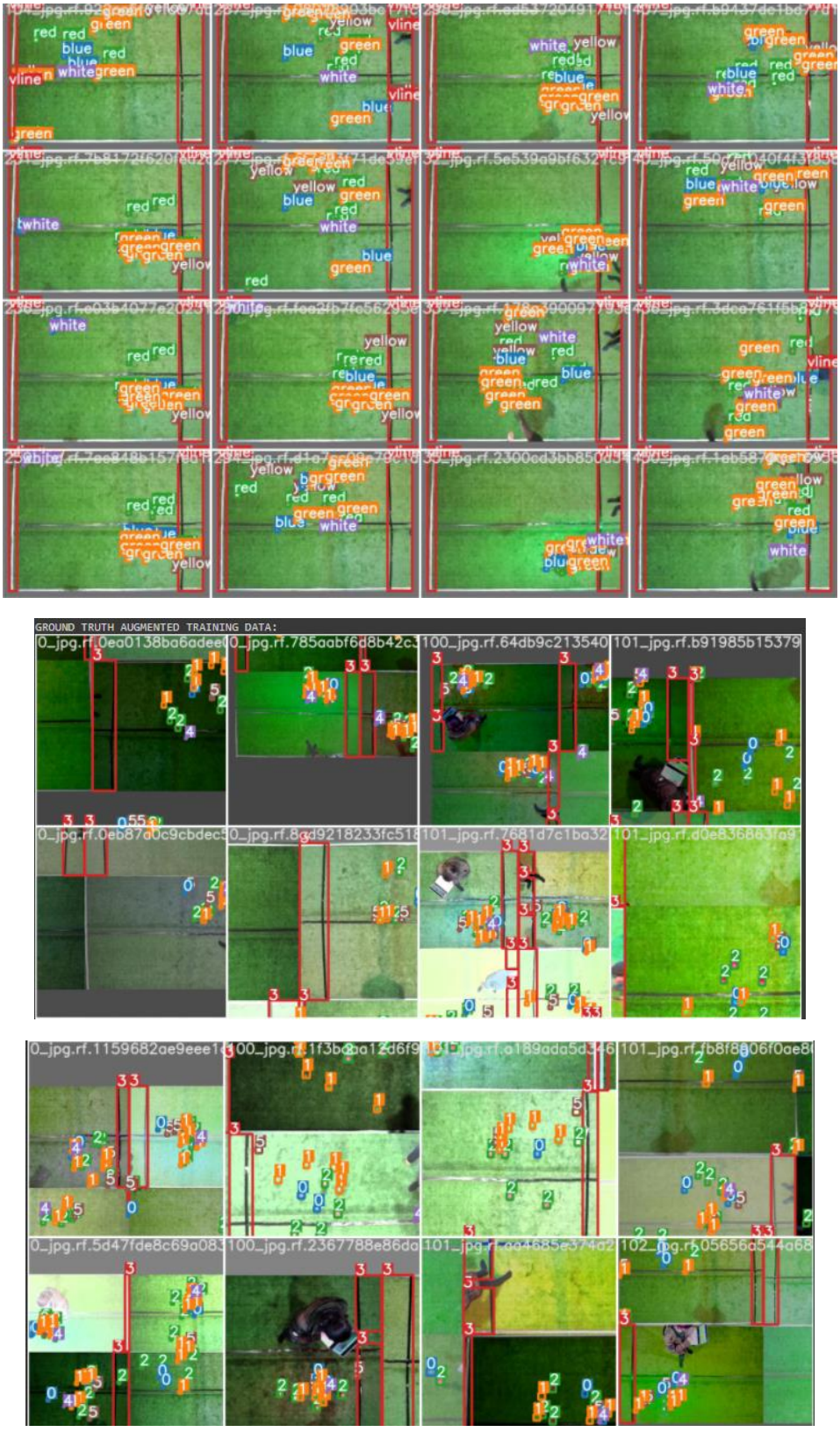


شکل 1-2-5 نتایج نموداری مدل با تمرین در 150 ایپاک



شکل 1-2-6 نتایج نموداری به سبک ابتدایی مدل با تمرین در 150 ایپاک

همچنین میتوان نتایج را برای مدل های تمرین مشاهده نمود، هرچند هم اکنون ناواضح هستند، در بخش بعد، line-thickness از 3 به 1 تغییر پیدا میکند و خوانا تر میشود.

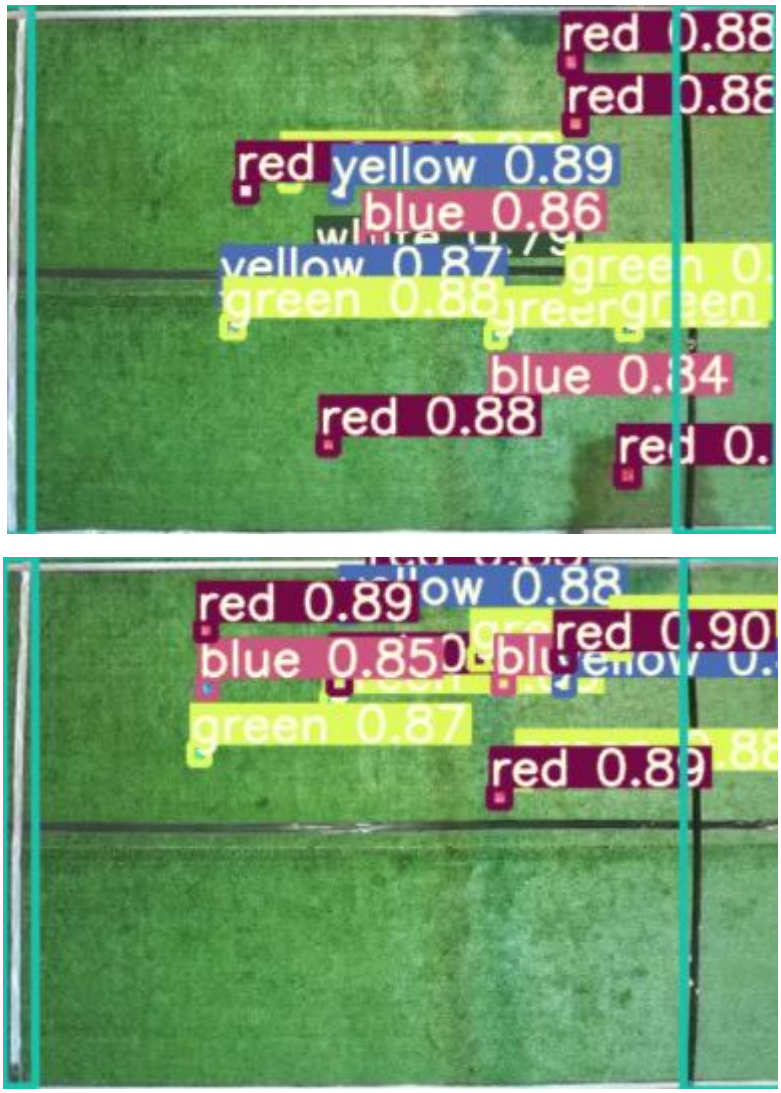


شکل 1-2-7 نتایج تشخیص مدل به داده های تمرین

1) خروجی مدل به ازای ورودی تست(Valid) بعد از آموزش، Enhancement فایل detect.py

بعد از تمرین داده های آموزش داده های پوشه valid را ارزیابی کردیم و تشخیص object را روی آن بررسی کردیم.

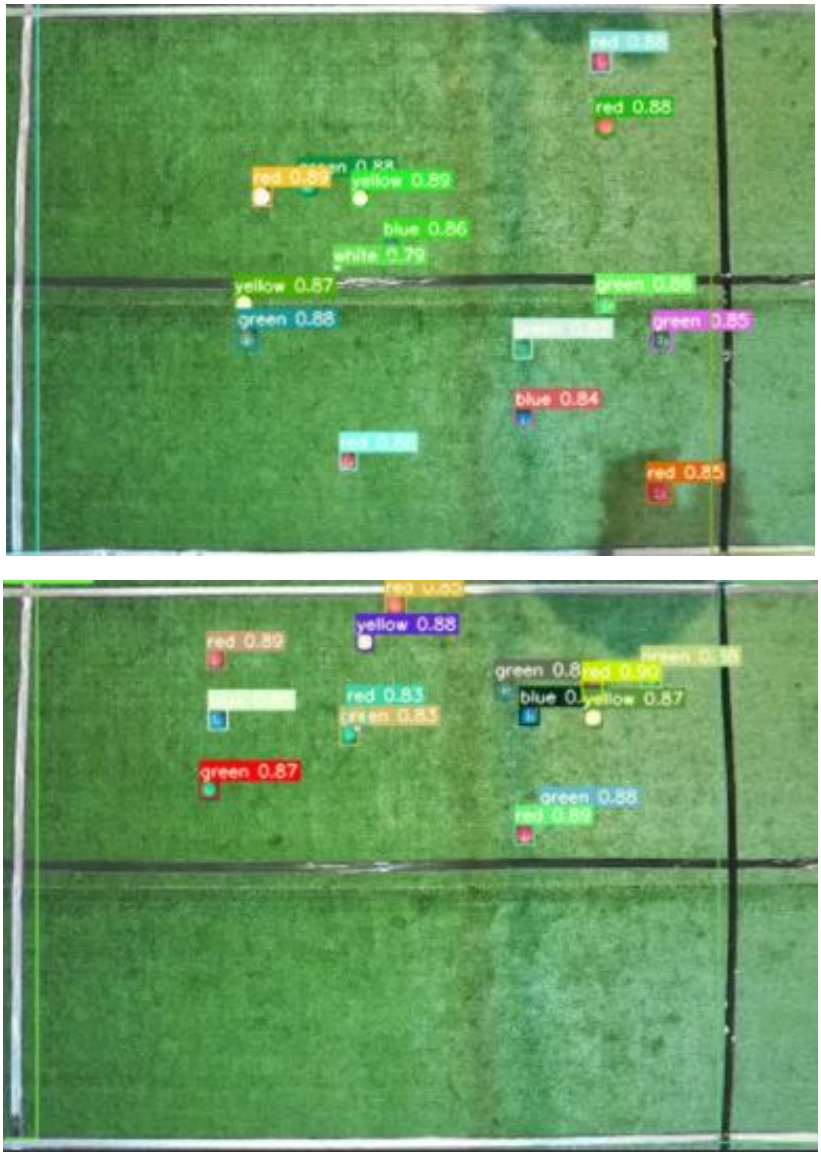
بعد از ران کردن detect.py نتایج در فایل exp ذخیره شده اند و قابل نمایش اند، در زیر، نتیجه دو خروجی آورده شده است.



شکل 1-2-8 نتایج تشخیص مدل به داده های ارزیابی

برای خواناتر کردن تشخیص مقدار Thickness لاین ها را کاهش میدهیم (از 3 به 1 میرسانیم) و در قسمت Save image اعمال کردیم.

همان دو نتیجه تصویر بالا، اکنون کاملا واضح شده اند:



شکل 1-2-9 نتایج تشخیص مدل به داده های ارزیابی

4)

در این بخش با استفاده از مرکز توپ ها می خواهیم فاصله آن ها با توپ سفید را بسنجیم و آن ها را سورت کنیم.

تغییری که در تا بع detect.py می دهیم این است که متغیر xyxy_pos دو گوشه توپ را نشان می دهد که میانگین این دو را به عنوان مرکز اعلام میکنیم. سپس متغیر cls که کلاس هر توپ را دارد و شامل 6 نوع هست را داریم.

ابتدا مرکز توپ سفید را می یابیم و بعد به objrct نگاه می کنیکم و اگر خط نبود این مقدار را برای بقیه توپ ها انجام می دهیم.(با تابع abs و لیستی از مراکز بقیه توپ ها(جز توپ سفید))

بعد با distance ای که اولین بار دیده شده و برای هر توپ و نوع کلاسش را می نویسیم و آن ها را با هم مقایسه میکنیم.

در زمانی که هیچ توپ سفیدی دیتکت نشده و کد نمیداند با چه توپی مقایسه کند، در ابتدا یک مکان (0و0) به سنتر توپ سفید اد میکنیم.

خروجی تصویر به صورت فاصله از توپ سفید به صورت زیر است:

[179 , 258, 274.5 ,289.5 ,312.5 ,338 ,432.5 , 478.5 , 511, 369]

کلاس های متناظر:

[2 , 2 , 1 , 1 , 1 , 0 , 2 , 1 , 2 , 5]

سوال ۲ – Semantic Segmentation

در این تمرین برای انجام عملیات semantic segmentationکه کاربرد های زیادی در تصاویر پزشکی ، object detection، کنترل سرعت ماشین ها، autonomous driving و.. دارد ، شبکه Unet را با توجه به مقاله ذکر شده در صورت سوال و با 23 لایه کانولوشنی ساختیم و از دیتاست CamVid برای آموزش و تست این شبکه استفاده کردیم.

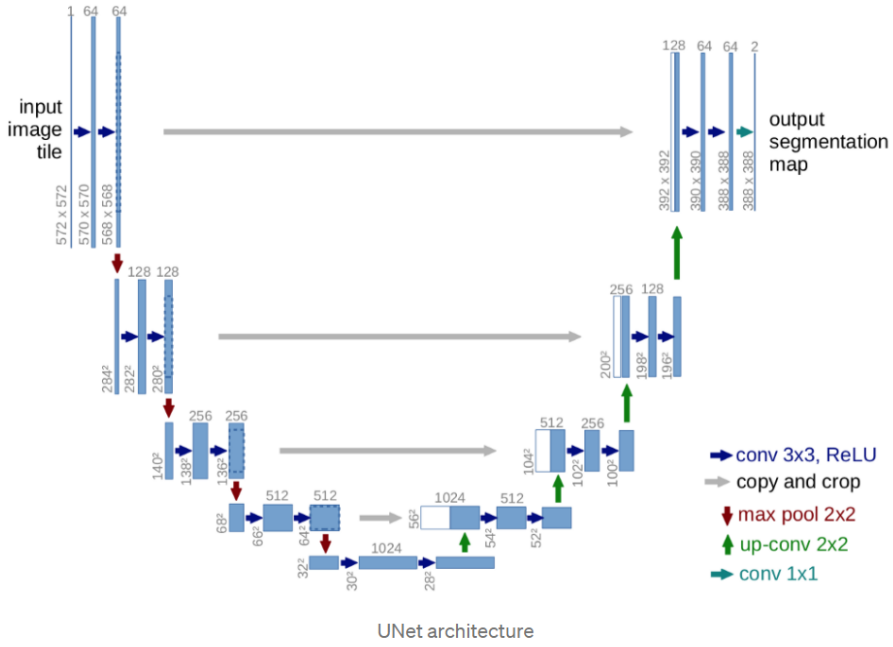
برای ساخت شبکه از کتابخانه keras , tensorflow استفاده شده است. همچنین از تابع cross entropy برای محاسبه خطا استفاده گردیده است.

این شبکه به این صورت کار می کند که در pixel level ، ویژگی های عکس را با توجه به رنگ و شکل آنها و آموزش مدل با لایه های کانولوشنی را استخراج کرده و با استفاده از up-sampling ، ویژگی های استخراج شده در هر مرحله را به دست می آورد.

در دیتا ست داده شده ، یک سری عکس از خیابان ها در حالت های مختلف داده شده و در کنار هر کدام از این عکس ها، ورژن لیبل زده شده آن نیز موجود است تا یادگیری با توجه به کلاس بندی مذکور(که با رنگ ها و طیف های مختلف نمایش داده شده است)، انجام گیرد. برای اینکه شبکه متوجه شود که هر طیف رنگی متعلق به چه جسمی در خیابان است ، یک فایل text ، شامل نام کلاس ها برای هر جسم و مقدار عددی RGB هر کدام وجود دارد.

هر کدام از عکس ها را با عکس labeled شده ی آن به صورت یک جفت در می آوریم و دیتا فریمی که از نام کلاس ها و مقدار RGB آن ها داریم(class_map) را با استفاده از تابع form_2D_label ، برچسب گزاری می کنیم.

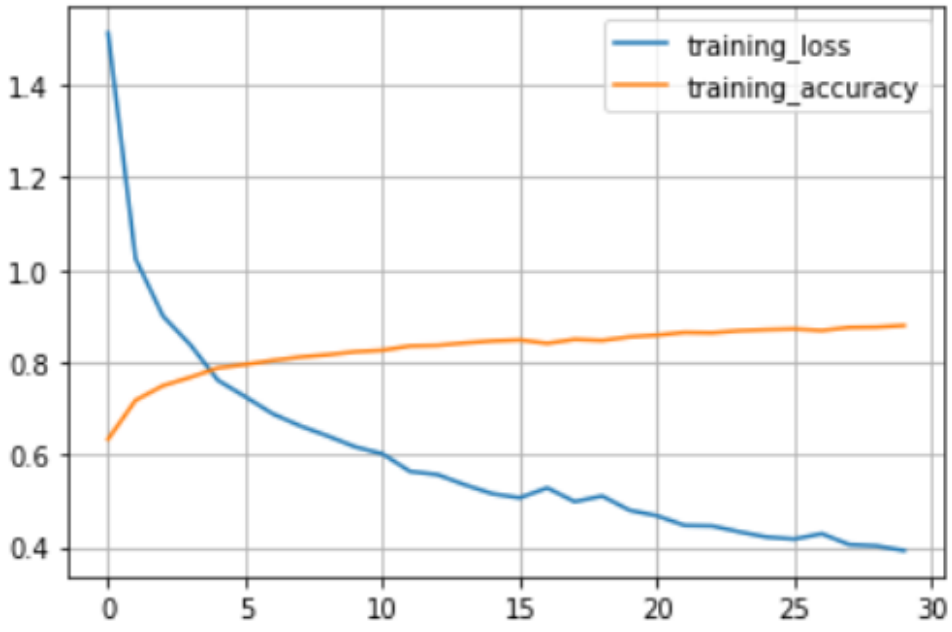
شبکه U-net به شکل زیر است:



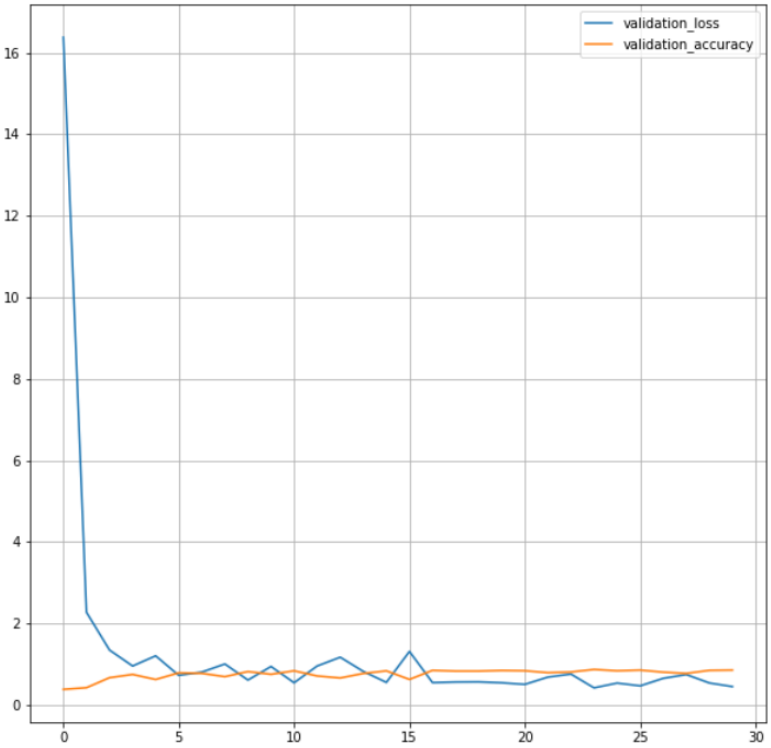
شکل 2-1 – شبکه U-net

.1

پس از گذشت 1.5 ساعت training ، نتایج به صورت زیر است:



شکل 2-2 – مقدار accuracy و loss برای داده های آموزش



شکل 2-3 – مقدار accuracy و loss برای داده های test

.2

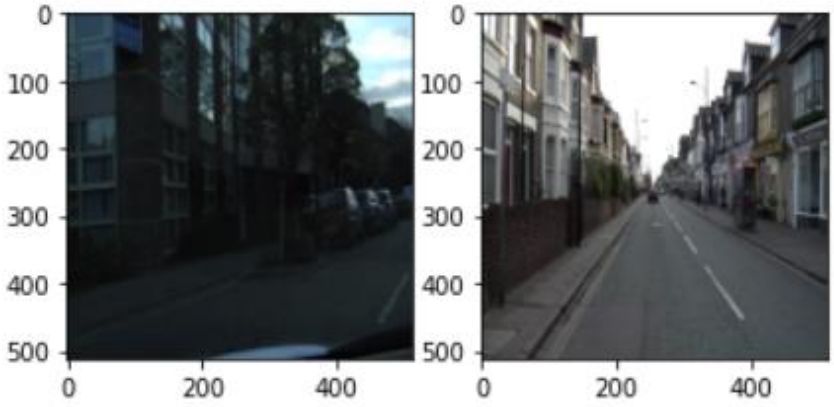
شبکه روی داده های تست اعمال شده و به نتیجه زیر رسیدیم:

```
model.evaluate_generator(test_generator)
```

```
[0.6948401927947998, 0.8018593192100525]
```

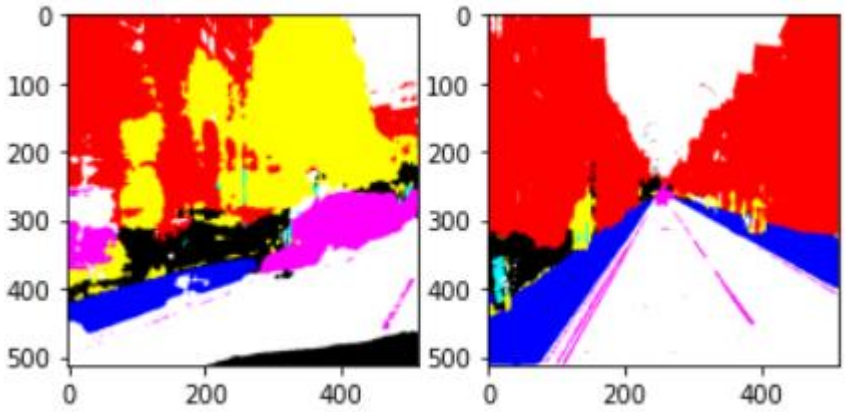
مقدار 80 درصدد accuracy و 0.69 ، loss متناظر آن است.

دو عکس زیر در مجموعه train هستند :



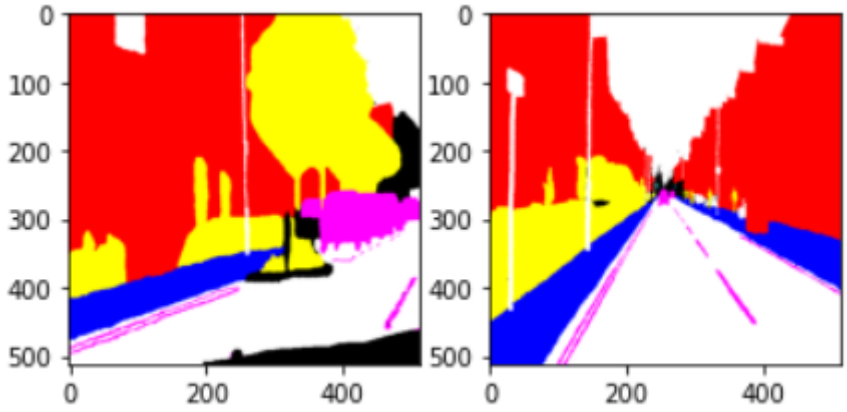
شکل 2-4– عکس های آموزشی

خروجی پیش بینی شده شبکه:



شکل 2-5 – خروجی labeled شبکه

برچسب واقعی:



شکل 2-6– عکس برچسب گذاری شده واقعی

.3

تعداد ایپاک مناسب برا شبکه از روی مقدار validation loss که در سوال 1 نمایش داده شد می تواند حاصل شود که حدودا در 23 ایپاک ، این مقدار مینیمم است و می تواند انتخاب بهینه ما باشد.

برای جدا سازی داده ها از 701 عکس موجود ، 55 درصد را برای داده های آموزش، 30 درصد برای تست و 15 درصد برای داده های validation اختصاص داده شد.

شبکه V-net :

شبکه های کانوولوشنی در سال های اخیر در زمینه های پزشکی مورد استفاده قرار گرفته اند ولی این شبکه ها معمولا توانایی بررسی و یادگیری و در نهایت segmentation عکس های پزشکی با دو بعد را دارد ولی اکثر عکس ها در این زمینه سه بعدی اند و حجم دارند پس به سراغ استفاده از V-net می رویم که با سرعتی بالا از پس این عکس های سه بعدی بر می آید. در مقاله ی مورد بررسی عکس های MRI پروستات را به عنوان ورودی به مدل داده و قسمت های مشکوک به سرطان آن را جدا کرده و segment کرده است.

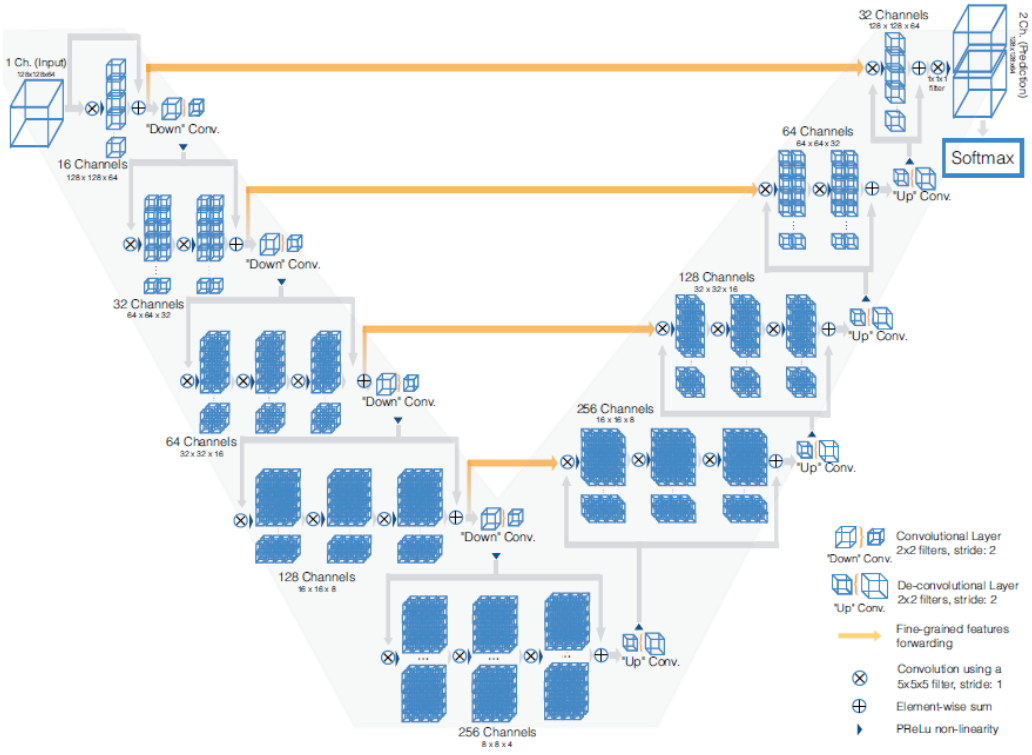
این شبکه از کانوولوشن سه بعدی استفاده می کند و objective function یا همان تابع خطای آن بر مبنای ضریب dice است که ضریبی بین 0 و 1 می باشد و برای حداکثر کردن آن، تلاش می کنیم. این ضریب برای مقادیر باینری به صورت زیر محاسبه می شود به صورتی که N تعداد حجم های کوچکی که لیبل زده می شوند و p لیبل حدس زده شده و g لیبل واقعی است:

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

از این تابع خطا می توان نسبت به هر حجم مشتق گرفت که معادل فرمول زیر است:

$$\frac{\partial D}{\partial p_j} = 2\left[\frac{g_j\left(\sum_i^N p_i^2 + \sum_i^N g_i^2\right) - 2p_j\left(\sum_i^N p_i g_i\right)}{\left(\sum_i^N p_i^2 + \sum_i^N g_i^2\right)^2}\right]$$

معماری این شبکه نیز به صورت زیر خواهد بود:



شکل 2 – 1 :

سمت چپ فشرده سازی و سمت راست بازسازی داده را انجام می دهد. سمت چپ از چند stage تشکیل شده است که هر یک بین 1 تا 3 لایه کانوولوشنی دارند. ابتدا از kernel های 5*5*5 و جلوتر از 2*2*2

با stride دو استفاده می شود همچنین down-sampling نیز در مسیر سمت چپ دیده می شود. به ازای هر لایه convolutional یک لایه deconvolution در سمت راست موجود است. در جدول زیر تعداد ورودی و سایز kernel ها ذکر شده است:

جدول 2 – 1 : مشخصات شبکه

Layer	Input Size	Receptive Field
L-Stage 1	128	$5 \times 5 \times 5$
L-Stage 2	64	$22 \times 22 \times 22$
L-Stage 3	32	$72 \times 72 \times 72$
L-Stage 4	16	$172 \times 172 \times 172$
L-Stage 5	8	$372 \times 372 \times 372$
R-Stage 4	16	$476 \times 476 \times 476$
R-Stage 3	32	$528 \times 528 \times 528$
R-Stage 2	64	$546 \times 546 \times 546$
R-Stage 1	128	$551 \times 551 \times 551$
Output	128	$551 \times 551 \times 551$

این مدل روی دیتاست PROMISE 2012 تست شده اند و در مقایسه با باقی مدل ها نتایج زیر به دست آمده اند:

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.

شکل 2 – 2 : بررسی نتایج مدل های مختلف روی دیتاست PROMISE 2012

مدل روی 50 داده augment شده این دیتاست آموزش داده شده است و هر batch دو حجم در نظر گرفته شده است زیرا مدل از لحاظ مموری هزینه بر است سپس بر ری 30 عکس MRI تست شده است که نتایج بالا به دست آمده است.