به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

شبکه های عصبی و یادگیری عمیق

تمرین امتیازی سری ۱

نام و نام خانوادگی	امید واهب
شماره دانشجویی	۸۱۰۱۹۶۵۸۲
تاریخ ارسال گزارش	۱۴۰۰/۰۳/۱۸

فهرست گزارش سوالات

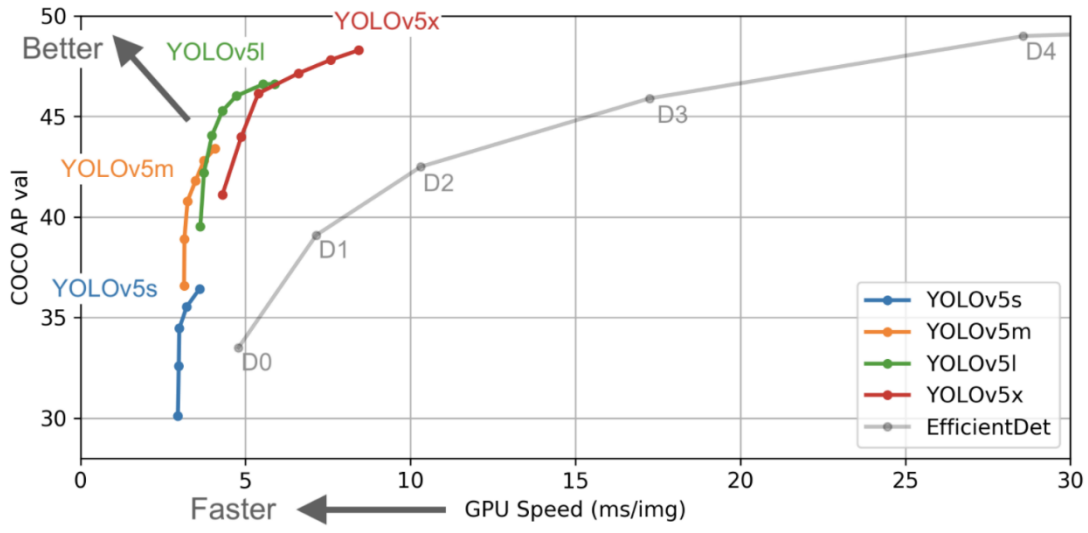
سوال ۱ – Object Detection with YOLOv5

۱- ورژن ۴ را سه برنامه نویس به نام های Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao نوشته اند و Joseph Redmon که در ورژن ها قبلی حضور داشت به دلیل احتمال استفاده های نظامی از تکنولوژی اش در این ورژن کناره گیری کرده است. درباره این ورژن جمله ی زیر گفته شده است:

YOLOv4's architecture is composed of CSPDarknet53 as a backbone, spatial pyramid pooling additional module, PANet path-aggregation neck and YOLOv3 head.

این مدل از CSPDarknet53 به عنوان ستون اصلی استفاده می کند تا مهم ترین ویژگی ها را استخراج کند سپس به جای Feature Pyramid Networks که در ورژن ۳ استفاده شده بود از PANet استفاده شده است که روشی است برای جمع بندی پارامترها برای سطوح مختلف detector. این ورژن دو برابر سریع تر از EfficientDet است که مرجع مقایسه در این مساله است و از نظر دقت نیز بهبود داشته است و میانگین Precision یا AP و Frame Per Second به ترتیب ۱۰ و ۱۲ درصد افزایش به نسبت ورژن ۳ داشته است. می توان گفت که این ورژن هم سریعتر و هم دقیق تر است.

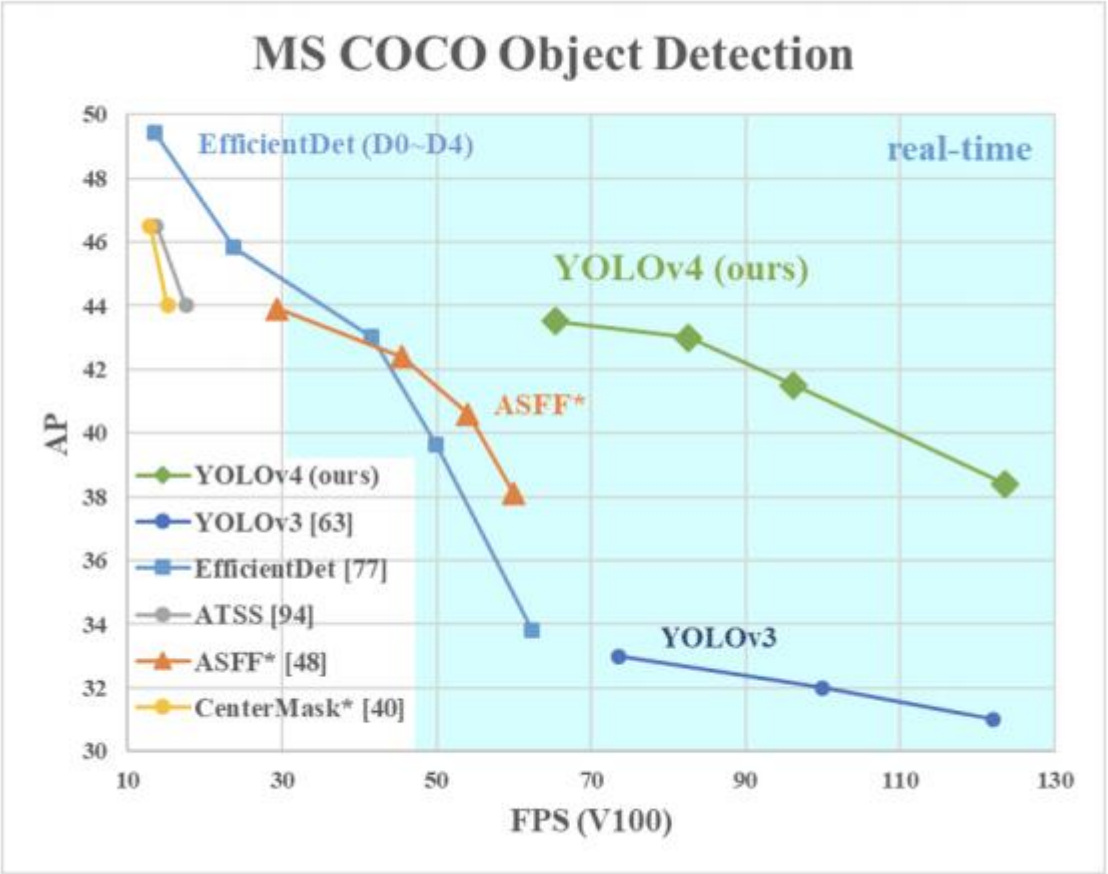
ورژن پنجم Yolo نیز ارایه شده است که سریعتر است و بهبود دقت داشته است.



شکل ۱ – ۱ : وضعیت سرعت و دقت ورژن ۵ به نسبت مدل مرجع EfficientDet

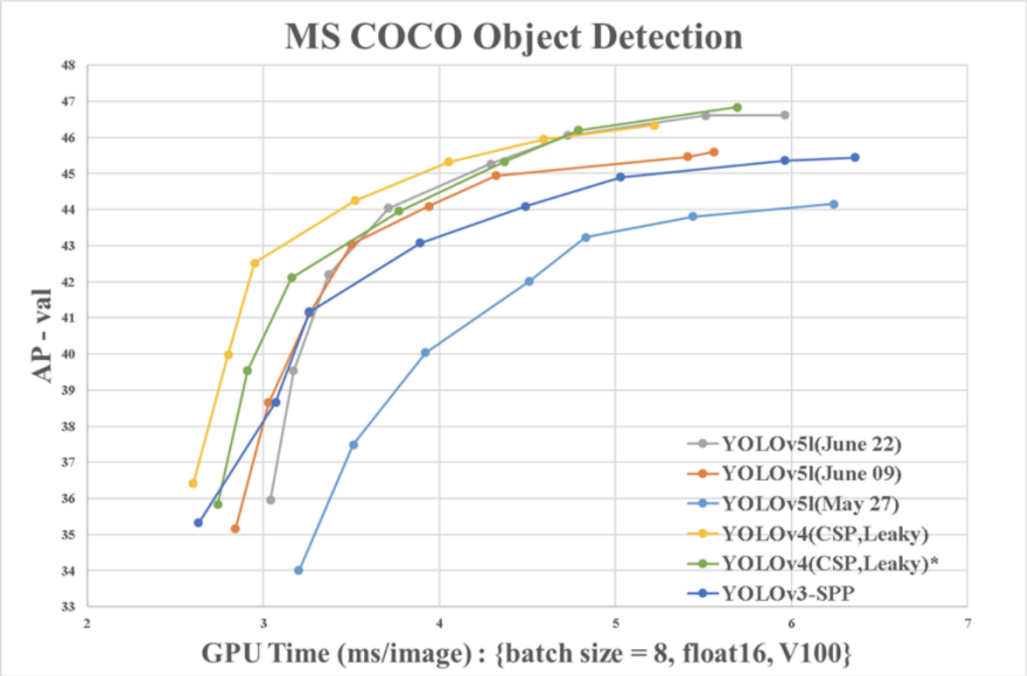
معماری این مدل تقریبا شبیه ورژن چهارم است و از تکنیک های Pytorch استفاده کرده است و هم اکنون نیز در حال ارتقا است. از تغییرات این ورژن به تغییرات در Data Augmentation می توان اشاره کرد که از سه نوع اسکیل کردن، تغییرات رنگ و mosaic augmentation تشکیل شده است.

Anchor Box ها به صورت اتوماتیک در ورژن ۵ آموزش داده می شوند. همچنین برای افزایش سرعت از دقت ۱۶ بیتی به جای ۳۲ بیتی برای رنگ ها استفاده شده است. هر دو ورژن ۴ و ۵ از Darknet و PANet به عنوان ستون اصلی مدل استفاده می کنند. عکس زیر عملکرد مدل های مختلف را نشان می دهد:



شکل ۱ – ۲ : عملکرد مدل ها روی COCO

همچنین در عکس زیر مقایسه ورژن های مختلف Yolo را می بینیم:



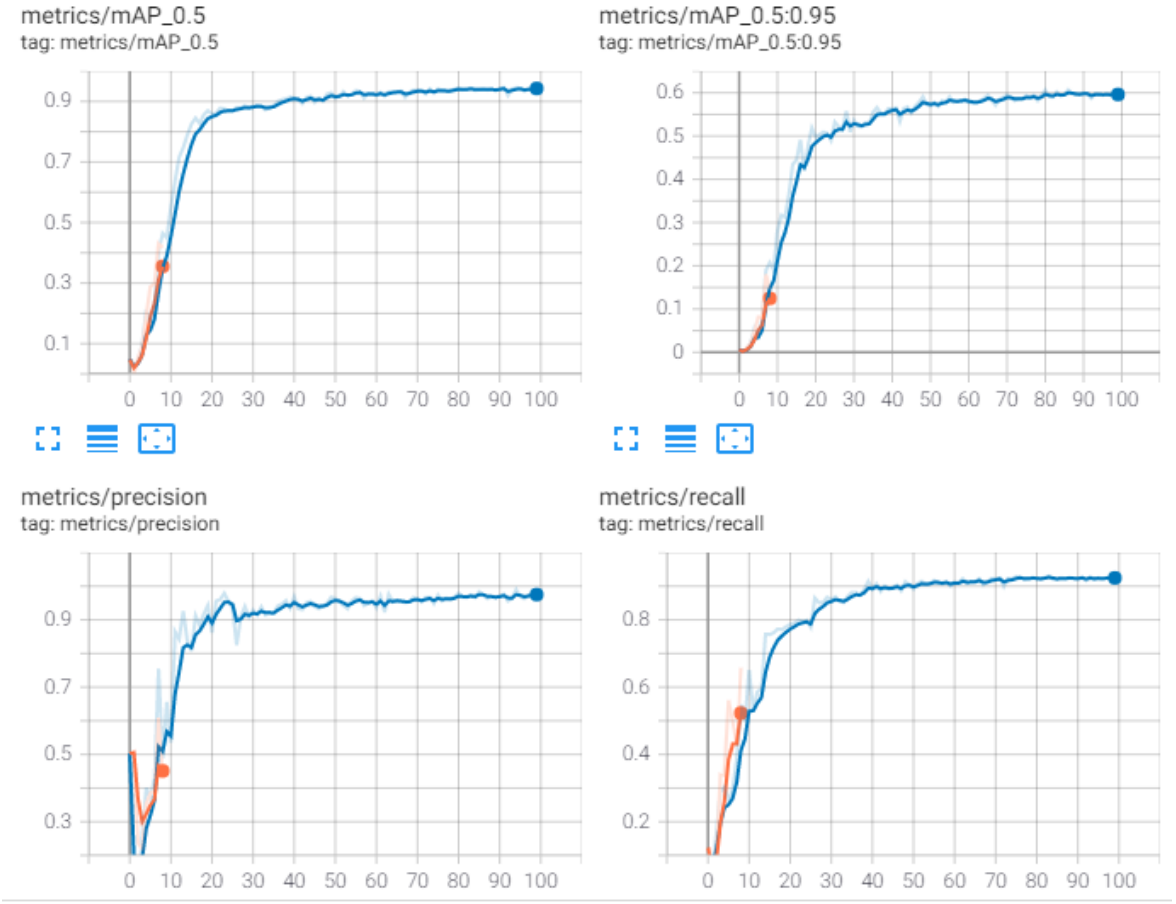
شکل ۱ – ۳ : مقایسه عملکرد ورژن های Yolo

۲- حال به سراغ نوت بوک آپلود شده می رویم و قدم به قدم آن را بررسی می کنیم. ابتدا از لینک گیت هاب Yolo کد موجود را clone می کنیم. سپس پیش نیاز های مورد نیاز را نصب کرده و دیتاست داده شده را در نوت بوک لود می کنیم. داده ها را در گوگل درایو آپلود کرده ایم و به سادگی درایو را mount کرده و فایل مدنظر را unzip می کنیم. فایل yaml را نیز بررسی کرده و کدگذاری کلاس ها را می بینیم که به ترتیب از ۰ تا ۵ را به آبی، سبز، قرمز، خط، سفید و زرد نسبت داده ایم. سپس تعداد کلاس ها را مشخص کرده و شبکه را تعریف می کنیم. همانطور که در قسمت قبل توضیح داده شد در سه استیج anchors، backbone و head شبکه تعریف می شود که در هر یک تعدادی لایه وجود دارد. در مرحله ی بعد به کمک کد از قبل نوشته شده train.py مدل را آموزش می دهیم.

```
Epoch   gpu_mem       box       obj       cls     total   targets  img_size
99/99     1.68G    0.0461   0.04124  0.007825   0.09517       272       448: 100% 99/99 [00:13<00:00,  7.08it/s]
          Class     Images     Targets         P         R     mAP@.5  mAP@.5:.95: 100% 2/2 [00:01<00:00,  1.59it/s]
            all         58   1.00e+03     0.977     0.926     0.946     0.595
           blue         58        115     0.988     0.983     0.995     0.607
          green         58        290         1     0.955     0.996     0.593
            red         58        290     0.995     0.983     0.996     0.651
          vline         58        136     0.971     0.977     0.983     0.879
          white         58         58     0.907     0.672     0.712      0.19
         yellow         58        116         1     0.987     0.995      0.65
```

شکل ۱ – ۴ : نتایج ایپاک آخر

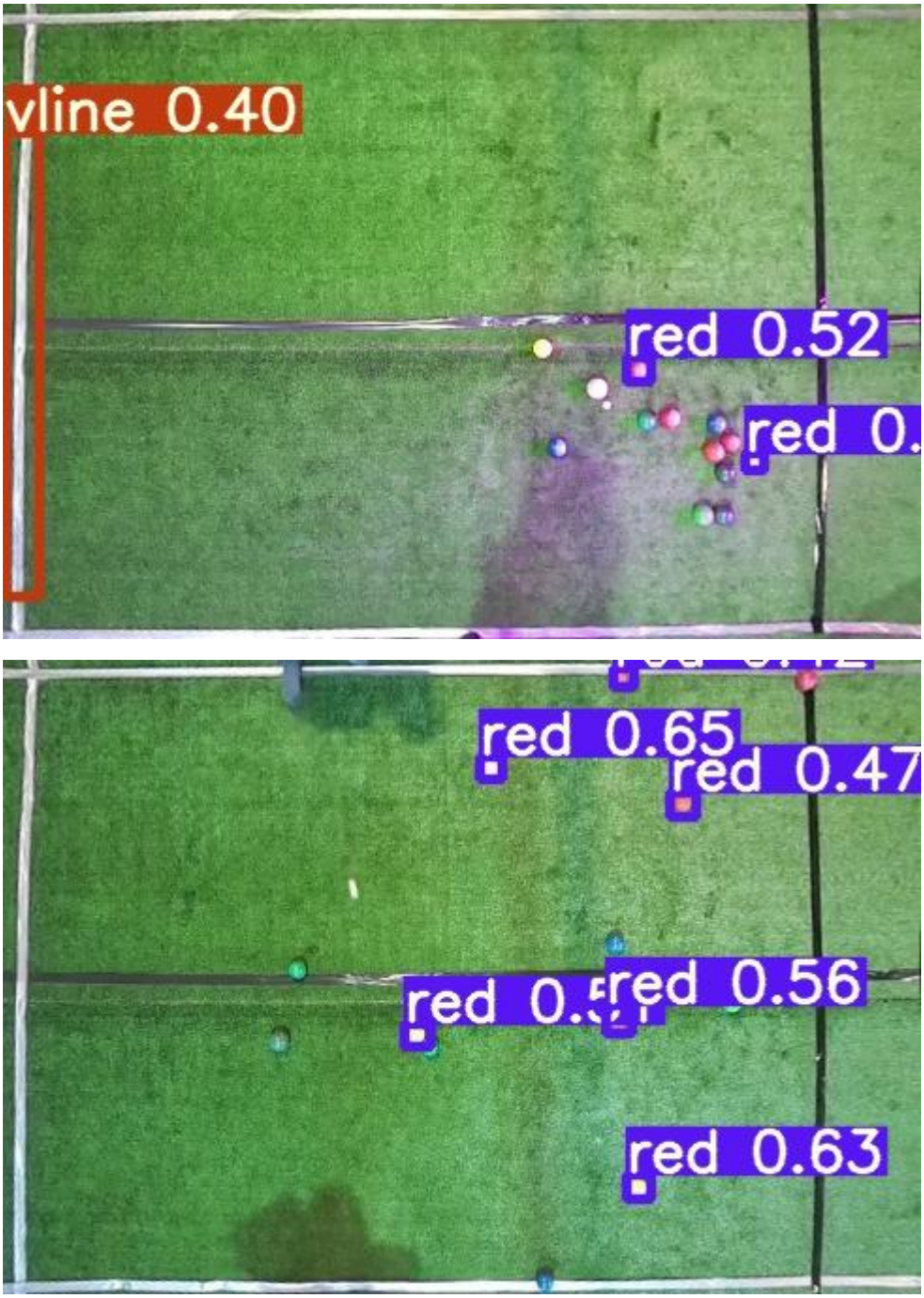
می توان نتیجه ی ایپاک آخر را در شکل بالا مشاهده کرد به صورتی که recall، precision، map0.5 و map0.5:0.95 برای کل داده ها و هر کلاس از توپ یا خطوط آمده است که واضح است به دقت بالایی رسیده ایم.



شکل ۱ – ۵ : نمودار شاخص های عملکرد در ایپاک ها

شکل ۱ – ۶ : نمونه تشخیص توپ ها و کلاس های مختلف تشخیص داده شده توسط مدل

۳- حال داده های تست را به مدل می دهیم و خروجی ها را بررسی می کنیم. نتیجه دو تا از عکس ها به شرح زیر است:



شکل ۱ – ۷ : خروجی ۲ تا از تصاویر تست

برای واضح تر شدن عکس ها اقداماتی برای کاهش line thickness انجام شد. ابتدا line_thickness را از ۳ به ۱ تغییر می دهیم که متاسفانه جواب نداد سپس با اضافه کردن پارامتر line thickness و با قرار دادن مقدار پیش فرض آن با مقدار ۱ به وضوح بالایی در عکس ها می رسیم.

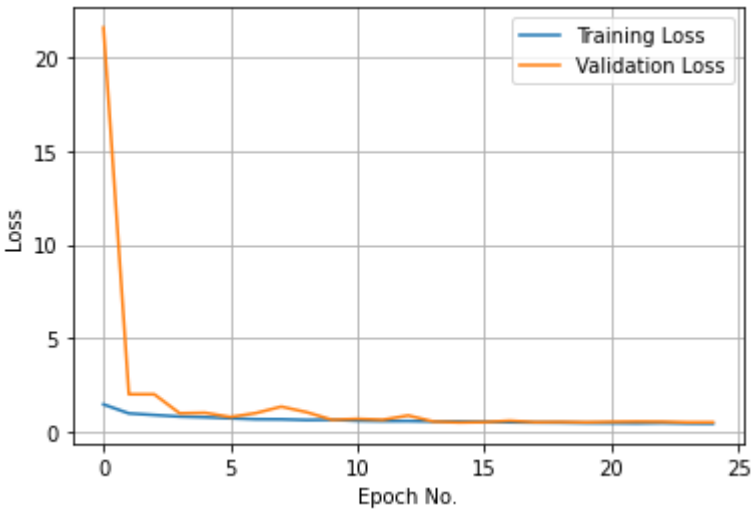
۴ – در قسمت آخر باید توسط پارامتر xyxy عملیات خواسته شده را انجام داد به این صورت که میانگین این پارامتر برای گوشه سمت چپ بالا و سمت راست پایین را گرفته تا مرکز به دست آید و در هر عکس

اگر توپ سفید موجود بود فاصله ی بقیه ی توپ ها تا این توپ را توسط جذر مجموع توان دوم دو پارامتر توپ ها و توپ سفید و سورت کردن این فواصل. متاسفانه به علت مشکلات tensorflow این قسمت کامل انجام نشد.

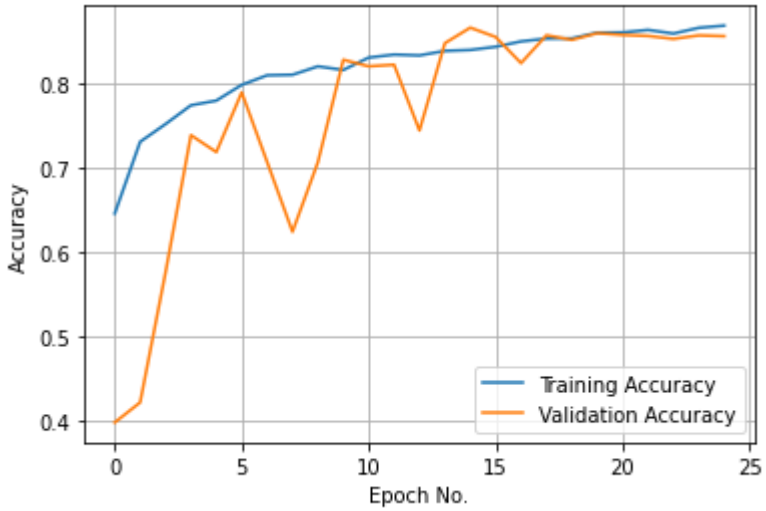
سوال ۲- Semantic Segmentation

۱- ابتدا داده Camvid را در نوت بوک آپلود می کنیم. برای این کار از طریق خود سایت Kaggle این کار را انجام می دهیم و با آپلود فایل json مربوط به اکانت خود در چند ثانیه دیتای نسبتا بزرگ این مساله را load می کنیم. این دیتاست عکس هایی از خیابان است که با هدف کمک به پیشبرد خودروهای خودران از آن استفاده می شود. این دیتاست علاوه بر عکس های خام محیط، ورژن آن عکس ها با نواحی segment شده را نیز دارد. همچنین در فایلی دیگر نیز نواحی و رنگ های RGB هر ناحیه آمده اند. در این سوال از نوت بوک موجود در Kaggle استفاده می شود زیرا کامل و جامع است و تغییرات کمی لازم دارد. ابتدا عکس اصلی و عکس ماسک شده یا همان عکس با نواحی مشخص شده را با هم جفت می کنیم و با اطلاعات فایل csv موجود نیز تطبیق می دهیم. سپس شبکه U-net را پیاده سازی می کنیم و تمامی لایه های آن شامل up sampling و dropout و لایه های کانوولوشنی را اضافه کرده و مدل را آموزش می دهیم.

نمودار خطا و دقت داده های آموزش و تست به شرح زیر است:



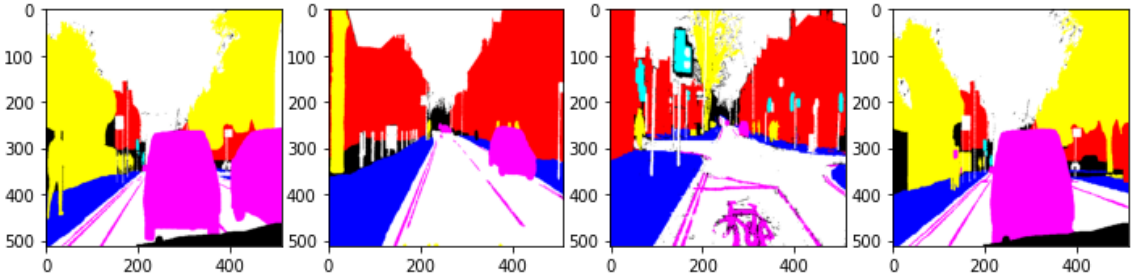
شکل ۲ – ۱ : نمودار خطا داده های آموزش و validation



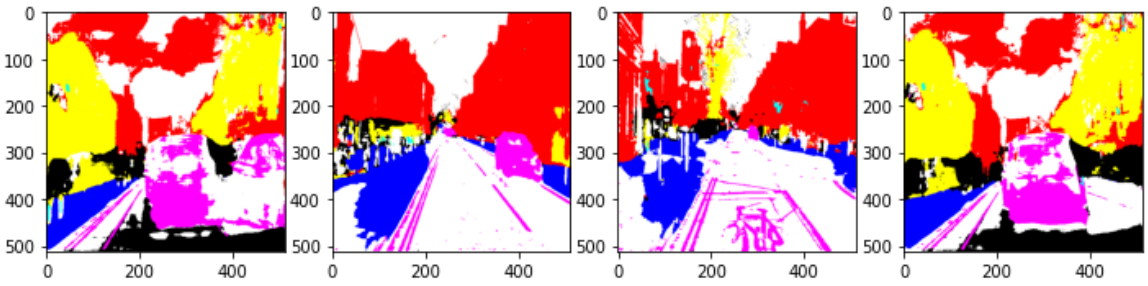
شکل ۲ – ۲ : نمودار دقت داده های آموزش و validation

۲- حال شبکه را روی چند تا از عکس های دیتای تست امتحان می کنیم و نتیجه به شرح زیر است که نشان از عملکرد خیلی خوب مدل است:

شکل ۲ – ۳ : عکس های اصلی



شکل ۲ – ۴ : عکس های از قبل segment شده



شکل ۲ – ۴ : عکس های segment شده توسط مدل

می توان مشاهده کرد که تقریبا مثل داده های لیبل زده شده توانستیم segment کنیم و مدل به خوبی توانایی جداسازی کلاس های موجود در یک عکس را دارد.

۳- با دقت در نمودار های قسمت اول می توان دید که تعداد ایپاک مناسب بین ۲۵ تا ۳۰ است زیرا بعد از این بازه دقت شبکه بهبود آنچنانی ندارد و صرفا overfit می شود و کمتر از ۲۵ نیز خوب نیست زیرا با ادامه دادن آموزی می توان به دقت بهتری رسید. در این تمرین ۲۵ ایپاک در نظر گرفته شد زیرا تفاوت چندانی بین ۲۵ و ۳۰ نبود و در زمان صرفه جویی شد. معیار جداسازی داده ها نیز به صورت تصادفی است به این صورت که عکس ها shuffle شده اند و ۳۶۷ داده برای train و ۲۳۳ داده برای test و ۱۰۱ داده برای validation جدا شده اند. به عبارتی ۳۳ درصد از داده ها برای تست و از بین ۷۵ درصد باقیمانده تقریبا ۲۰ درصد برای validation و باقی نیز برای یادگیری انتخاب شده اند.

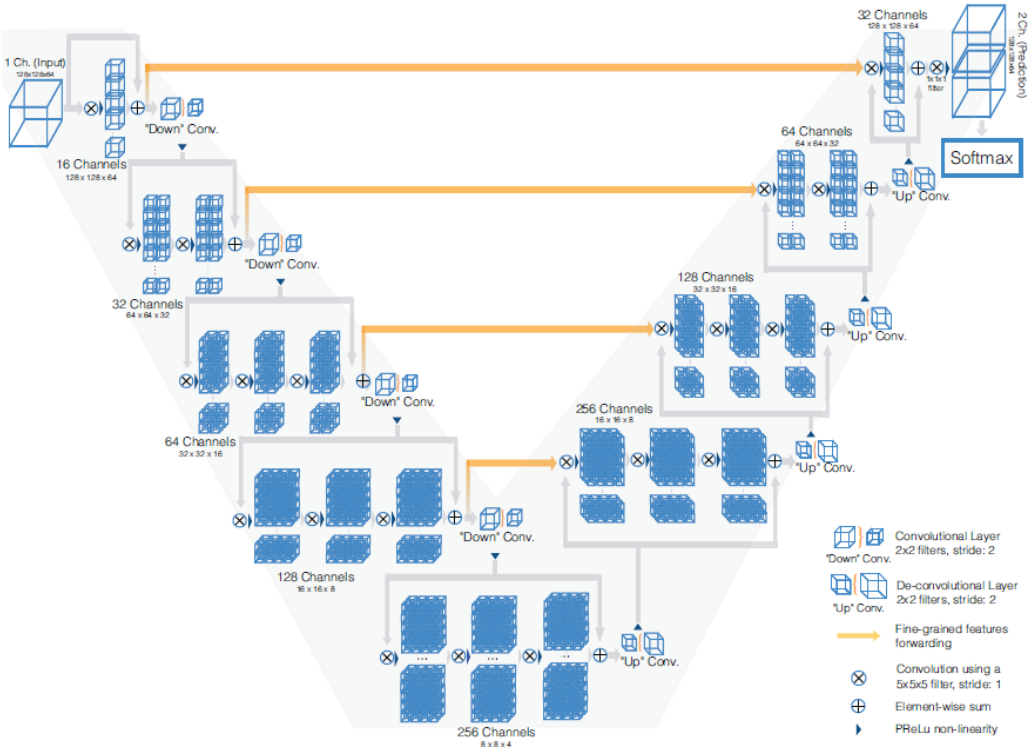
شبکه های کانوولوشنی در سال های اخیر در زمینه های پزشکی مورد استفاده قرار گرفته اند ولی این شبکه ها معمولا توانایی بررسی و یادگیری و در نهایت segmentation عکس های پزشکی با دو بعد را دارد ولی اکثر عکس ها در این زمینه سه بعدی اند و حجم دارند پس به سراغ استفاده از V-net می رویم که با سرعتی بالا از پس این 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 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 ۰ و یک است 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 ۲ – ۱ : معماری شبکه

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 ۱ تا ۳ 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 ۲ – ۱ : 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 ۲ – ۲ : بررسی نتایج مدل های مختلف روی دیتاست PROMISE 2012

مدل روی ۵۰ 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 ۳۰ عکس MRI تست شده است که نتایج بالا به دست آمده است.