گزارش تمرین دوم یادگیری عمیق

آرش محمد قلی نژاد

400443183

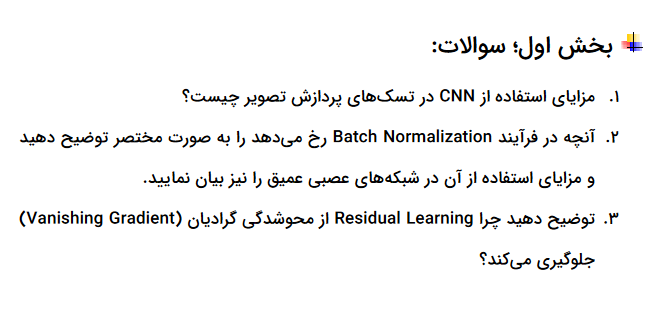
استاد درس: دکتر حامد ملک

توجه -1 :اگر در فایل های تمرین ایرادی وجود داشت کلیه ی فایل ها در [لینک](https://drive.google.com/drive/folders/1vSYUttL5utWyReT73f6PXbSvHtr-kCUE?usp=sharing) موجود است

توجه 0 :اگر نیاز به بررسی نوت بوک داخل فولدر بود و نوت بوک ایرادی داشت از [لینک](https://drive.google.com/file/d/1g0J3ccn1aL_0lqS9d8Gh6g-BP1WR0EH4/view?usp=sharing) استفاده کنید برای کولب هست

توجه 1 :متاسفانه بعضی نمودار ها به اشتباه در حلات دارک مود عکس گگرفته شده اند با بردن روشنایی به بالا میتوانید اعداد را بخوانید.

بخش اول سوالات:



سوال اول:

دلایل مختلفی باعث این میشوند که شبکه ی عصبی کانوولوشنی از شبکه ی عصبی mlp بهتر عمل کند یکی از آنها Curse of Dimentionality است در شبکه ی عصبی mlp همه ی پیکسل های تصویر را مستفیما به شبکه ی عصبی وارد میکند و تصویر با ابعاد بالا و وضوح بالاتر ابعاد بیشتری دارد و در مجموع تصاویر پیچیده تر هستند زیرا به طور کل پیکسل های زیاد باعث میشوند فیچر های ورودی تعداد زیادی داشته باشند و در مجموع باعث پیچیدگی شبکه و خود داده میشوند که باعث بیشتر شدن تعداد وزن های شبکه میشود که بایدد تمرین داده شوند.

که همین مشکل باعث میشود تا تمرین دادن شبکه با روش هایی مثل کاهش گرادیان سخت تر شود.

اما شبکه های کانوولوشنی به جای اینکه کلیه ی اطلاعات تصویر را به عنوان ورودی بدهیم به شبکه ی عصبی در لایه های کانولوشنی ویژگی های اصلی را استخراح میکنیم و با مکس پولینگ ابعاد نیز کمی کاهش میابد و علاوه بر آن کل تصویر را به fully connected نمیدهیم و سرعت یادگیری نیز به مراتب بیشتر میشود نکته ی اصلی عملیات کانوولوشن نیز همین هست کاهش ابعاد تصویر به طوری که به لایه های fully connected تصاویر با ابعاد کمتری داده شود اما تصاویر ویژگی اصلی خود را از دست ندهند.

به طور خلاصه مزایای کانوولوشن به شرح زیر هستند:

1-یادگیری الگوی دقیق و بینش عمیق از داده های ارائه شده.

2-اتصال محلی هر نورون دیگر به همه ی سلول های عصبی موجود در لایه ی قبلی متصل نیست که باعث کاهش تعداد پارامتر ها میشود.

3-نمونه گیری های لایه ی pooling میتواند با حفظ اطلاعات مفید مقدار پردازش را کاهش دهد.

سوال دوم:

این فرایند تکنیکی برای بهبود عملکرد و پایداری شبکه های عصبی دانست

ایده ی این کار نرمال کردن ورودی هر لایه با کمک میانگین و واریانس میباشد به شکلی که میانگین اعداد بردار نهایی برابر صفر و واریانس آنها برابر یک شود .

میدانیم که نرمال سازی ورودی شبکه به فرآیند یادگیری کمک میکند اما در نظر داشته باشید که شبکه یک سریال از لایه ها باشد و خروجی هر لایه هم ورودی لایه بعدی میباشد پس هر لایه میتواند برای ما حکم لایه اول یک زیر شبکه کوچکتر را داشته باشد.

بچ نرمالایزیشن به این شکل کار میکند که هر بار که لایه ی اکتیویشن اعمال میشود خروجی ها را به لایه ی نرمال سازی میدهیم تا مقادیر لایه ها نرمال شوند که اینکار توسط میانگین و واریانس این بچ ها انجام میشود.

در کل مزایای نرمال سازی بچ ها این هست که شبکه ی عصبی را سریعتر و پایدار تر میکند.

سوال سوم:

اول باید ببینیم محو شدگی گرادیان چیست؟

وقتی که ما از لایه های بیشتری با اکتیویشن فانکشن های مشخص استفاده میکنیم گرادیان تابع loss ما به سمت صفر میل میکند که این کار شبکه را برای آموزش دادن سخت میکند.

اما چرا این اتفاق میافتد.

برخی از توابع اکتیویشن مثل سیگمویید بازه ی بزرگی از مقادیر را به بازه ی کوچکی بین 0 تا 1 نگاشت میکنند پس اگر تغییر بزرگی روی ورودی رخ دهد ما در خروجی تغییر بسیار کمی مشاهده میکنیم پس مشتق ورودی ما نیز کوچک میشود.

شاید برای شبکه های کم عمق مشکل خاصی به وجود نیاید ولی وقتی لایه های بیشتری استفاده میشوند وقتی که الگوریتم بک پروپاگیشن از لایه ی اخر به اول شروع به استفاده از قانون مشتق زنچیره ای میکند و مثلا تابعی مثل سیگمویید داریم پس طبق چیزی که گفتیم این مشتق ها وقتی برای تعداد بالای لایه ها حساب میشوند مقادیر بسیار کوچکی خواهند داشت پس گرادیان به مقدار زیادی کاهش میابد

گرادیان کم یعنی که وزن ها و بایاس های ما به طور خیلی موثری کاهش نمیابند پس در نتیجه شبکه ی عصبی به خوبی اموزش داده نمیشود

ساده ترین راه حل استفاده از یک اکتیویشن فانکشن بهتر میباشد اما راه حل بهتری نیز وجود دارد استفاده از residual network ها چرا که انها با استفاده از رزیجوآل بلاک ها و اضافه کردن ورودی آن لایه از شبکه ی عصبی مستقیما به خروجی ان شبکه ی عصبی باعث میشوند تا این ورودی جمع شده با خروجی از لایه ی اکتیویشن فانکشن عبور نکند.

پس مشتق تابع خطای ما که برای اپدیت وزن استفاده میشود نیز مقدار اندک و کوچکی نخواهد بود و مقدار عددی آن بزرگتر میشود پس آپدیت وزن ها موثر تر خواهد بود.

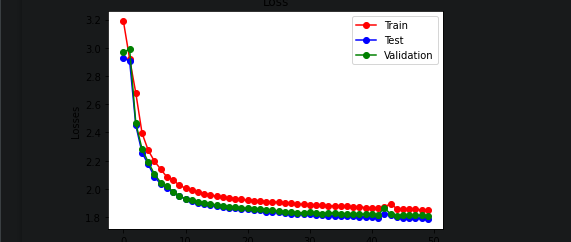
بخش دوم توضیحات خواسته شده در pdf :



در بخش basenet تست هایی روی تعداد فیتر ها انجام دادم در حالات به ترتیب 128 فیلتر 64 فیلتر و 32 فیلتر

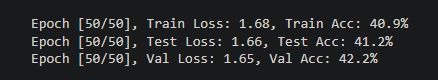
در معماری 128 فیلتر:

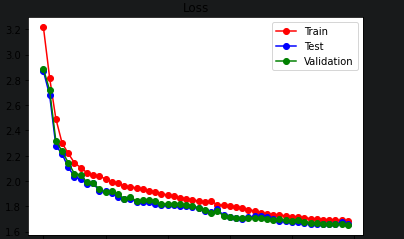




با این تعداد فیلتر مدل توانست به عدد دقت 38 برسد و لاس روی 1.81 برای Loss

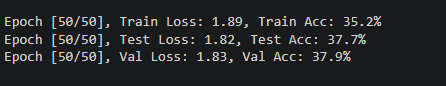
در معماری 64 فیلتر:

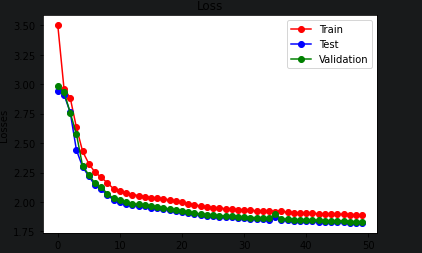




در معماری با 64 فیلتر به دقت 42 میرسیم و لاس هم کمتر از مدل 128 میباشد

معماری 32 فیلتر:





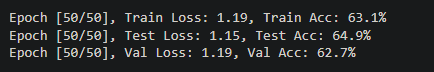
در این معماری نیز نسبت به 64 فیلتر لاس بالاتر و دقت کمی پایینتر هست.

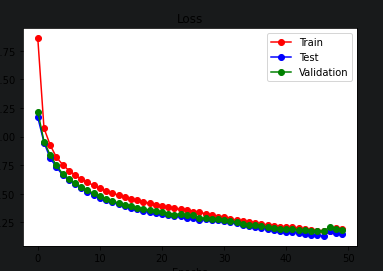
پس ما در حال حاضر با 64 فیلتر بهترین حالت ممکن را برای مدل base داریم

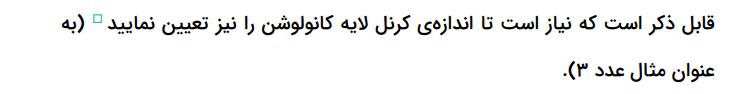
جواب سوال در مدل کاستوم نیز همین حالت هست.

در مدل کاستوم هم همین صورت هست و 64 فیلتر بهترین عدد برای مدل کاستوم من میباشد

تصاویر زیر میزان لاس و .... در این حالت هستند

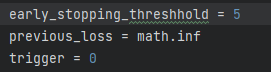


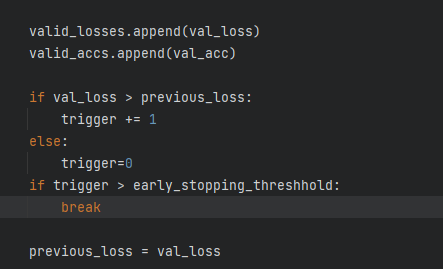




برای کرنل کانوولوشن در همه ی معماری ها عدد 3 را در نظر گرفتم کرنل های ما 3 در 3 هستند.

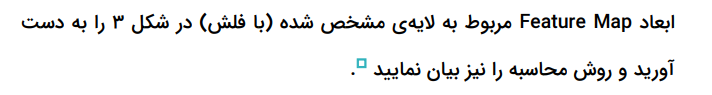






ارلی استاپینگ تکنیکیست که از اور فیت شدن جلوگیری میکند به یعنی وقتی که برای n بار مشاهده شود که لاس ولیدیشن در حال افزایش هست پس مدل در حال اورفیت شدن میباشد پس باید از این کار جلوگیری شود.

کد من هم در ابتدا چک میکند از لاس قبلی بزرگتر هست لاس فعلی یا خیر سپس اگر در حال سعود بود یک واحد به تعداد دفعاتی که سعودی بوده اضافه میشود اگر نه پس به حالت طبیعی برگشتیم و میتوانیم tigger را صفر کنیم چون نمیخواهیم اگر به طور مثال 3 بار صعودی بوده و سپس 10 ایپاک لاس کاهش یافته و 2 بار سعود کرده ارلی استاپینگ اتفاق بیافتد.



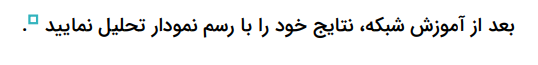
**K\*((W−F+2P)/S+1)**

K سایز عمق سایه ی کانوولوشن

W سایز ورودی

S سایز stride

P همان سایز پدینگ میباشد

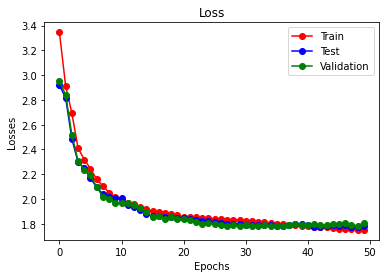


نتایح و پارامتر های ورودی من به شرح زیر بوده اند

برای مدل بیس پارامتر های ورودی به شرح زیر است:



با استفاده از این پارامتر ها و ترین مدل با حالت معماری پایه به لاس 1.6 الی 1.8 و دقت 43 درصد میرسیم.

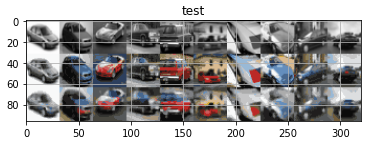
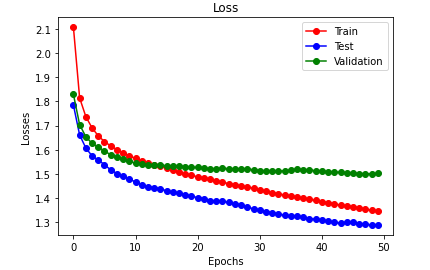


خروجی تصویر نیز به صورت زیر میباشد:

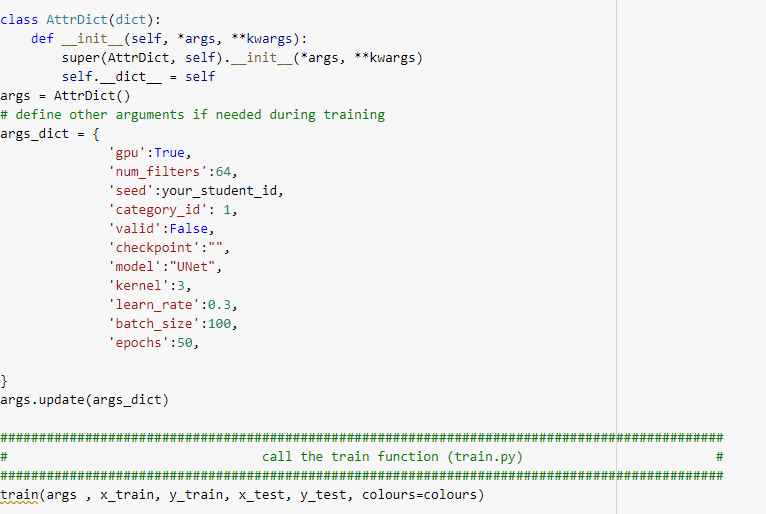


برای مدل Base با آزمایش دوم با بچ سایز 50 و learning rate 0.01 و تعداد فیلتر 32 به دقت 48 و لاس 1.5 میرسیم

خروجی و نمودار لاس به صورت زیر است.

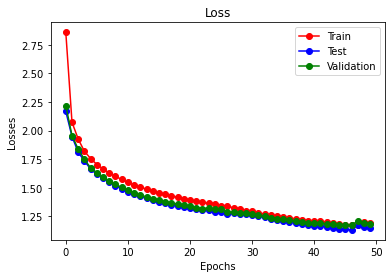
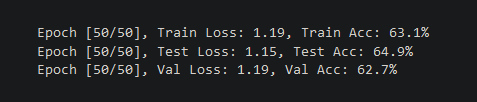
 

اما مشاهده میشود که مدل پس از 50 ایپاک در حال اور فیت شدن هست.

برای مدل CustomUnet با اسکیپ کانکشن:

با مقادیر ورودی بالا مدل با اسکیپ کانکشن موفق شده لاس کمتر یعنی 1.1 و دقت 62 درصد را بدست آورد.

خبری از اور فیت شدن نیست اما چون معماری به نسبت ضعیف هست نمیتوان از این دقت فراتر رفت.

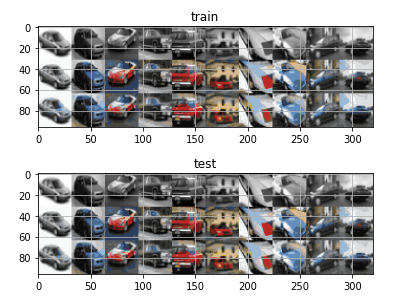
نمونه ی خروجی تصاویر رنگ شده نیز به صورت زیر میباشد.



برای این بخش نیز آزمایش دیگری میکنیم

با نرخ یادگیری 0.01 , بچ سایز 50

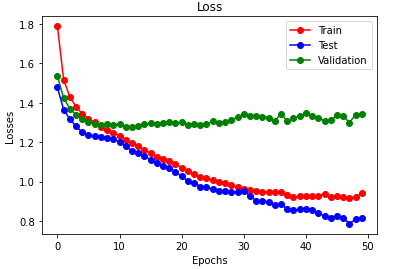
خروجی تصویر رنگ شده توسط مدل به شرح زیر میباشد.



مدل در این حالت بهتر رنگ امیزی کرده اما روی دیتای ولیدیشن دقت به جای 69 درصد در تست روی 57 میباشد

که نشانه ای از اورفیت شدن مدل میباشد.

نمودار هم به این شکل است

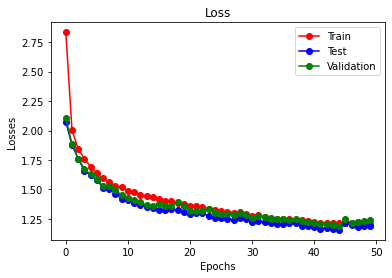


توجه:در این بخش هنوز از early stopping استفاده نشده است.

بخش امتیازی:

معماری Custome Unet با skip Connection و Residual Block :

در این حالت به معماری قبلی Residual Block اضافه میکنیم



تفاوت خاصی در این معماری با حالت قبل در مدل من ایجاد نشده لاس و دقت در حدود همان معماری بدون رزیجوآل بلاک

هست ولی تفاوت اصلی در تعداد ایپاک هایی است که نیاز است تا به دقتی معادل دقت معماری دوم برسیم اسکیپ کانکشن سبب بالاتر رفتن دقت ما میشود و لاس کمتری را مشاهده میکنیم ولی رزیجوآل بلاک ها کار مدل را کمی ساده تر کرده اند و پس از 10 الی 20 ایپاک به دقتی معادل معماری اسکیپ کانکشن میرسیم.

در رزیجوال بلاک من ما در حقیقت بخشی که MyConv نام داشت را با یک رزیجوآل بلاک تعویض کردیم

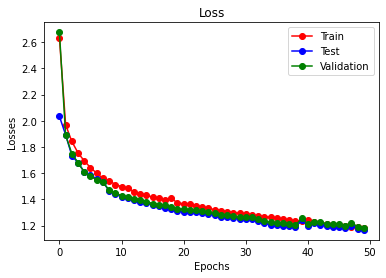
و فایل های BottleNeck و UpConv و DownConv را با رزیجوآل بلاک نوشتیم

معماری رزیچوآل بلاک به شکل زیر میباشد



خروجی تصاویر رنگ شده به شکل زیر میباشد





شکل بالا نمودار تغییرات لاس در 50 ایپاک میباشد.

Early stopping : دیتای ولیدیشن برای این هست که ببنییم مدل اور فیت میشود یا خیر این دیتا را به مدل میدهیم و عیین میکنیم وقتی مثلا 10 ایپاک لاس همینطور ر حال افزایش است فرایند یاد گیری را متوقف کنند در توضیحات کد ارلی استاپینگ را توضیح دادم .

بخش سوم :

توضیحات اجمالی کدها :

در فایل model.py یکسری مدل شبکه عصبی با معماری گفته شده در صورت تمرین یعنی basenet نوشته شده مدل دوم مدل پایه با skip connection میباشد

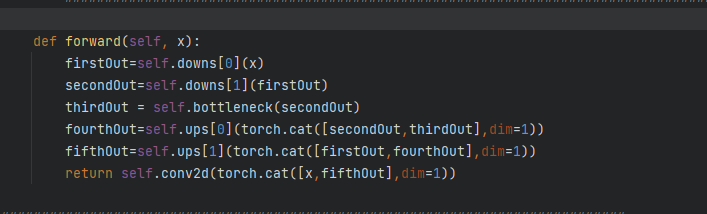
کلاس base net کلاس مدل اصلی ما هست از سه مدل bottleneck و BaseModel و UpConv و DownConv برای ساخت مدل لایه به لایه استفاده کرده است در Down Conv کلاس ابتدا کانوولوشن میگیریم و سپس با مکس پولینگ و بچ نورمالایزیشن و در آخر تابع Relu سپس در تابع فوروارد این متغیر ها را استفاده میکنیم.

مکس پولینگ تکنیکی برای کاهش ابعاد میباشد به صورتی که یک ماتریس با سایز مشخص را در نظر میگیرد و روی تصویر اعمال میکند و بزرگترین مقدار در بین پیکسل های تصویر درون آن ماتریس را در نظر میگیرد و اینگونه آن پیکسل انتخاب میشود و سپس ابعاد تصویر کاهش میابد.

در تابع فوروارد هم خروجی هر لایه به لایه ی بعدی داده میشود.

مدل Costumnet : در این مدل تنها تابع forward تفاوت دارد و از skip connection در آن استفاده شده است

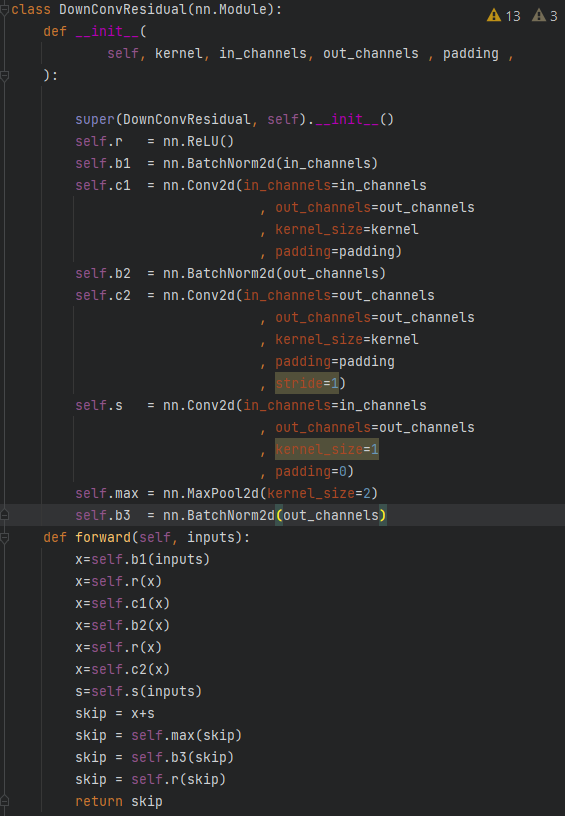
اسکیپ کانکشن به صورت زیر پیاده سازی شده است.



لایه های مختلف را درون متغیر های مختلف ذخیره میکنیم و سپس لایه ی ورودی را به اخرین لایه ی کانوولوشن و لایه ی اول با چهارم و سوم با دوم کانکت میشوند و به لایه ی بعدی داده میشوند.

Residual Block

معماری سوم معماری بیس نت با اسکیپ کانکشن هست ولی در تابع فوروارد و لایه های کانوولوشنی و ... تغییراتی ایجاد شده است برای بلاک های مختلف bottleneck , downconv , upconv سعی شده رزیجوآل بلاک استفاده شود.



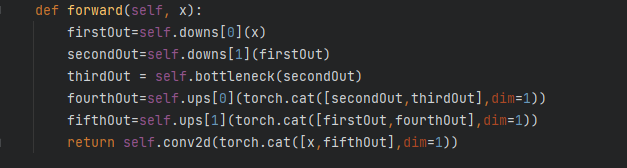
به طور مثال در این کد که مربوط به بخش downconv میباشد که رزیجوال بلاک پیاده شده است

در رزیجوآل بلاک دو لایه ی کانوولوشن داریم پس از لایه ی دوم کانوولوشن ما باید مستقیما ورودی کانوولوشن اول را به خروجی بلاک کانوولوشن جمع کنیم ولی گاهی اوقات اینکار قابل انجام نیست به علت اینکه ابعاد متفاوت هستند باید بررسی کنیم و در صورت تفاوت یک لایه کانوولوشن از ورودی خواهیم گرفت.

در بقیه ی بلاک های upconvResidual وBottleneckResidual هم به همین صورت تابع فوروارد را تغییر میدهیم.

در این رزیجوآل بلاک دو لایه ی کانوولوشنی میزسازیم اولی و سپس متغیر ورودی را با کانوولوشن تبدیل به حالت قابل جمع با خروجی رزیجوآل بلاک میکنیم یعنی کانوولوشنی با ورودی in\_channels و خروجی out\_channels روی آن اعمال میشود.

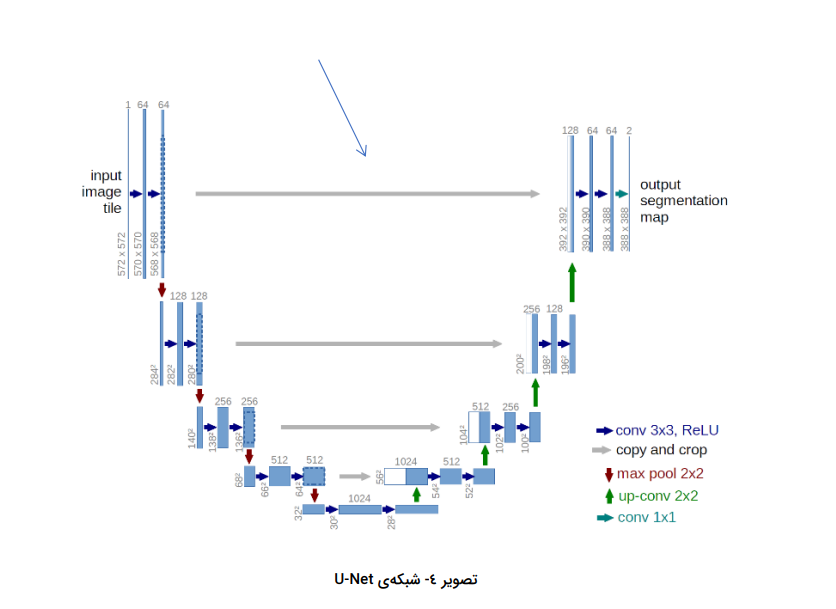
Skip connection :



کد بخش اسکیپ کانکشن به این صورت میباشد ورودی را با لایه ی 5 ام و پاسخ چهارم را با پاسخ اول و پاسخ سوم را با پاسخ دوم کان کت میکنیم اسکیپ کانکشن موجب افزایش دقت و کاهش میزان Loss میشود.

معماری Unet :

شبکه ی Unet اصلی در این بخش پیاده سازی شده است ابتدا کلاسی به نام doubleconv تعریف کردیم این کلاس در حقیقت همان بخشی از معماری Unet میباشد که در شکل زیر :



هر بار طبق تصویر دو بار کانوولوشن میگیریم و سپس با مکس پولینگ به لایه ی بعدی انتقال میدهیم و یا با upsampling به لایه ی بعدی انتقال میدهیم.

در هر صورت کلاس double conv صرفا قسمت دو بار کانوولوشن را انجام میدهد که در تمام لایه ها تکرار شده.

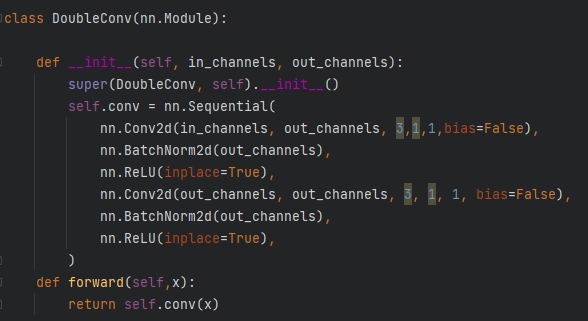
کلاس Unet هم پیاده سازی یونت میباشد.

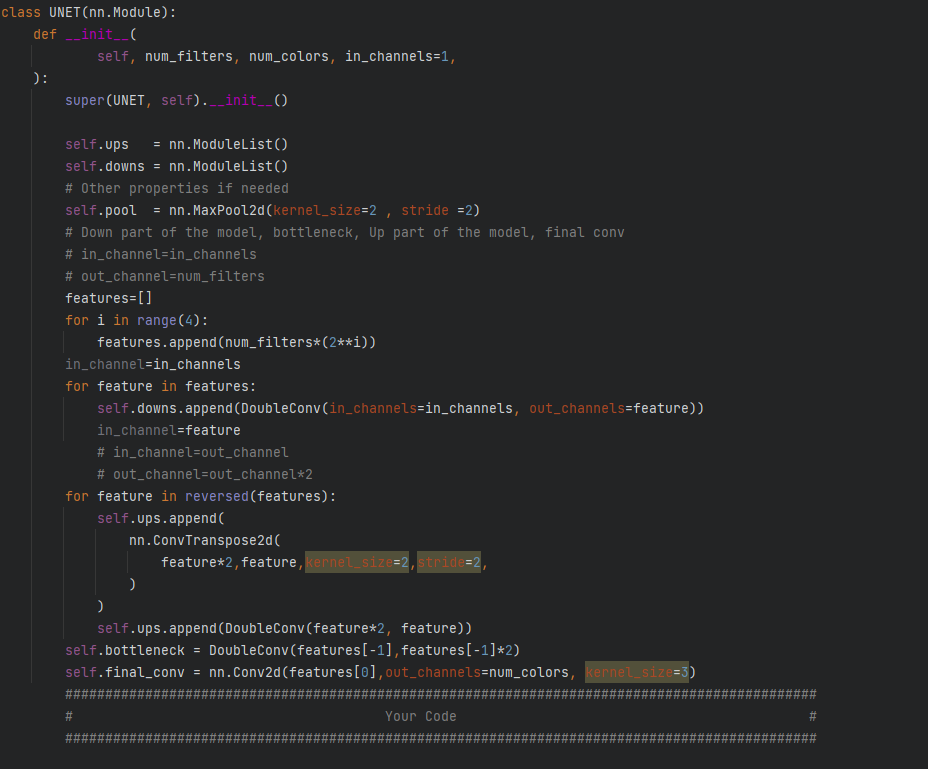
طبق معمول تعدادی لایه ی up و down وجود دارد که در تابع init نوشته شده است.

و در ادامه initialize لایه های آپ و داون صورت گرفته است.

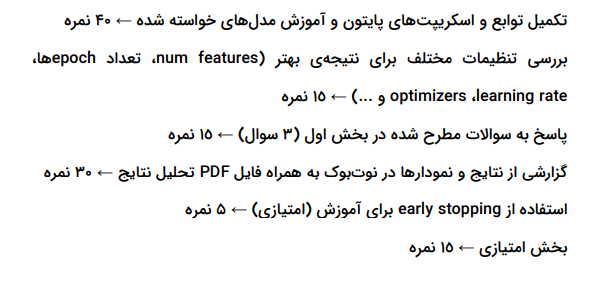
در تابع فوروارد نیز در یک لیست از اسکیپ کانکشن هایی که قرار است مستقیم به خروجی ببریم را ذخیره میکنیم و در هنگام اجرای لایه های UP ورودی اسکیپ کانکشن را با ورودی خروجی قسمت upconv کانکت میکنیم

و سپس اخرین لایه های کانوولوشنی را به روی آن اعمال میکنیم کل پیاده سازی معماری unet به این شکل میباشد.





در کل طبق شکل تمام لایه های مناسب برای یونت را پیاده کردم بعدش فهمیدم یونت اصلی اصلا شامل تمرین نمیشه برا همین تست نگرفتم.



برای بخش بررسی تنظیمات مختلف برای نتایج بهتر من بررسی کردم لرنینگ ریت مناسب 0.3 بوده است و تعداد ایپاک 50 مناسب بود

لرنینگ ریت های کوچکتر باعث اور فیت شدن مدل میشدند و لاس ولینیشن از جایی به بعد صعودی میشد.

برای بخش early stopping در فایل train.py موجود است و توضیح در گزارش وجود دارد.