مقدمه

در این گزارش قصد داریم مراحل طراحی و پیاده‌سازی یک شبکه عصبی عمیق به منظور انجام دو وظیفه Image Captioning و Image Comprehension را مرور کنیم. برای این منظور، در بخش اول به توضیحات مورد نیاز به منظور آماده‌سازی دیتا خواهیم پرداخت و در بخش بعدی طراحی معماری شبکه را بررسی می‌کنیم. سپس در قسمت پایانی نمودارهای آموزش شبکه را نشان می‌دهیم و نتایج را مقایسه می‌کنیم.

توجه: همه اعداد و نمودارهای ذکر شده در این گزارش را می‌توانید در نوت بوک ضمیمه شده پیدا کنید.

آماده سازی و پردازش‌های اولیه داده‌ها

در این بخش هدف آن است که پیش‌پردازش‌های اولیه روی دیتا به درستی انجام شود تا آن‌ها آماده ورود و پردازش توسط شبکه عصبی شوند. برای این منظور، در گام نخست لازم است تا فایل csvهای مربوطه خوانده شوند و اطلاعات باکس و متون موجود در آن‌ها به درستی استخراج شود. از آن جا که تعداد تصاویر دیتاست خیلی زیاد است و جا دادن همه آن‌ها در رم ممکن نیست، لذا لود کردن آن‌ها را به داخل کلاس مربوط به دیتاست منتقل می‌کنیم. بنابراین با هربار صدا زدن یک عضو از این مجموعه، تصویر متناظر با آن هم از روی دیسک خوانده می‌شود. این مسئله متاسفانه محدودیت زیادی در سرعت آموزش ایجاد می‌کند و باعث می‌شود که قابلیت آموزش شبکه‌های متعدد با مانع مواجه شود.

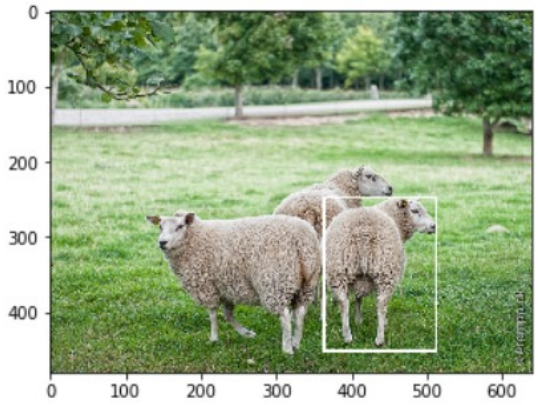
در کلاس دیتاست بعد آز آن که تصویر از روی دیسک لود شد، لازم است تا اعمال cropping، scaling و padding روی آن انجام شود. همه این پردازش‌ها توسط کتابخانه torchvision انجام می‌شود. بعد از انجام cropping، دو تصویر در اختیار خواهیم داشت که یکی تصویر کامل بوده و دیگری منطقه‌ای است که جمله در وصف آن گفته شده است. در ادامه با توجه به پیشنهاد مقاله مرجع هر دو این تصاویر scale و pad می‌شوند تا به سایز 224 در 224 برسند. (padding با مقدار صفر بوده و به این منظور استفاده می‌شود که aspect ratio به هم نریزد). یک نمونه از تصاویر اسکیل شده را می‌توان در ادامه مشاهده کرد:

از مجموعه پردازش‌های دیگری که در کلاس دیتاست انجام می‌شود، tokenize کردن و آماده‌سازی text است. برای این منظور، یک کلاس vocab (از تمرین سری ۴) استفاده شده است که کارهای مربوط به text و padding جملات و تبدیل آن‌ها به tensor را انجام می‌دهد. افزودن توکن‌های <START> و <END> نیز در این قسمت صورت می‌گیرد. ضمناً با توجه به پیشنهاد مقاله، پارامتر min_freq برابر ۳ در نظر گرفته شده است.

مجموعه فعالیت‌هایی که تا اینجا عنوان شد، همگی ملزومات انجام وظیفه Image Captioning می‌باشند اما برای آن که Image Comprehenstion هم به خوبی انجام شود، لازم است تا یک سری باکس نیز به عنوان نمونه منفی از هر تصویر تولید شود. برای تولید باکس لازم است از شبکه FasterRCNN استفاده شود و تشخیص منفی بودن باکس با استفاده از معیار IOU و مقدار آستانه 0.5 انجام می‌شود. اگر شبکه Faster نتوانسته بود باکسی با این مشخصات تولید نماید، یک باکس رندوم با IOU پایین تر از 0.5 تولید و استفاده می‌شود. البته اعمال شبکه FasterRCNN روی تصاویر خام به تعداد دفعات زیاد و با هربار فراخوانی دیتالودر کار سنگینی خواهد بود لذا شبکه Faster در ابتدای کار فقط یکبار روی کل تصاویر دیتاست اعمال می‌شود و نتایج آن را در یک فایل pickle روی دیسک ذخیره می‌کند. تا انتهای آموزش هر جا نیاز به

استفاده از شبکه Faster باشد، از این فایل ایندکس شده استفاده می‌شود و لذا دیگر احتیاجی به اجرای شبکه Faster برای بار دوم نخواهد بود. این مسئله به افزایش سرعت کمک زیادی می‌کند.

در نوت بوک ارئه شده، چندین تابع در جهت تسهیل نمایش و کار با تنسورهای تصویری پیاده‌سازی شده است که با دریافت یک باکس و تصویر اصلی، آن باکس را روی تصویر نمایش می‌دهد. یک نمونه از این نحوه نمایش را می‌توان در شکل زیر دید. چنین توابعی به دیباگ کردن مدل و ارزیابی نتایج به دست آمده کمک جدی می‌کنند و ما در بخش‌های بعدی از آن‌ها برای نمایش مناسب استفاده خواهیم کرد.



بنابراین به طور خلاصه خروجی دیتاست عبارت است از تصویر کامل resize شده، region مد نظر به صورت resize شده و متن توصیف کننده آن region. علاوه بر این سه مورد، تصاویر خام، مختصات باکس و متن پردازش نشده هم به عنوان خروجی ارسال می‌شوند که بعداً در بخش‌های comprehension و تولید فایل تست به کار می‌آیند. ضمناً اگر در مود fine tune شبکه برای وظیفه comprehension قرار داشته باشیم، علاوه بر موارد ذکر شده، یک باکس منفی و تصویر کراپ شده متناظر با آن هم ارسال می‌شود که توضیح دقیق‌تر آن را در بخش مربوطه خواهیم آورد.

در نهایت، در فاز آماده سازی لازم است تا دیتالودرها تعریف شوند. در تست‌های مختلفی که انجام شد، به نظر رسید که بزرگ کردن اندازه بچ می‌تواند هم در یادگیری بهتر و هم سرعت آموزش بیشتر، موثر باشد لذا اندازه بچ برابر ۳۲ در نظر گرفته شده است. (مقاله مرجع ۱۶ در نظر گرفته بود).

طراحی مدل

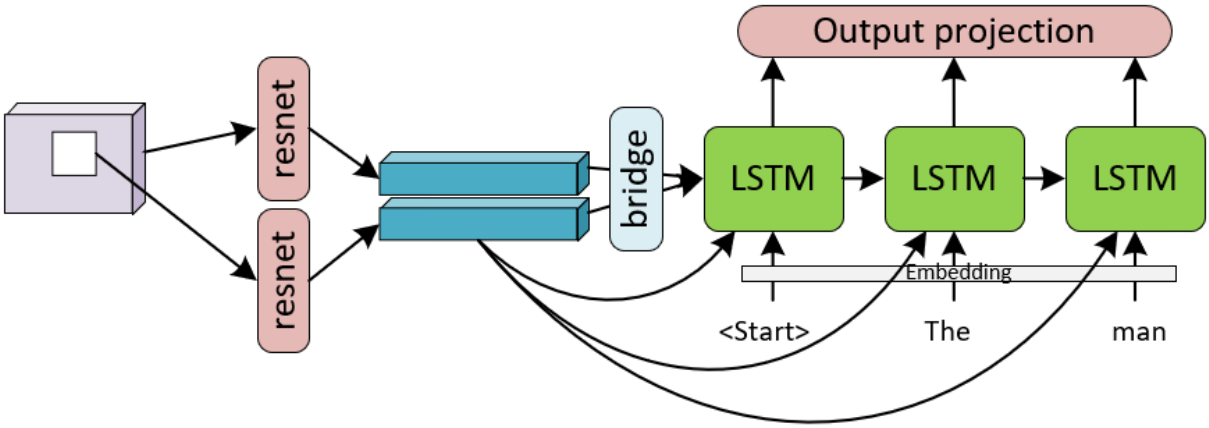
در بخش طراحی مدل ایده‌های متعددی بررسی و پیاده سازی شد و تا جایی که منابع محاسباتی اجازه می‌داد، سعی شد طیف وسیعی از مدل‌ها پوشش داده شود. عمده ایده‌هایی که در این قسمت وجود داشت، عبارت است از :

- شبکه مورد استفاده به عنوان feature extractor یکی از میان vgg یا resnet می‌تواند باشد.
- می‌توان شبکه feature extractor را فریز کرد یا آموزش آن را آزاد گذاشت.
- در ساختار دیکودر می‌توان attention به کار برد.
- در دیکودر می‌توان از LSTM به عنوان واحد بازگشتی استفاده کرد یا از GRU.
- استفاده از embedding از پیش آموزش دیده مثل Glove
- ابعاد بردارهای میانی شبکه می‌توانند در سرعت اجرا و دقت شبکه اثر جدی داشته باشند. مثلاً سایز hidden واحد بازگشتی یا اندازه embedding کلمات ورودی می‌تواند در طراحی مدل تاثیر گذار باشد.

با انجام تست‌های متفاوت سعی شده است اثر پارامترهای فوق تا جای ممکن بررسی شود. پیاده سازی برخی از این مدل‌ها به عنوان نمونه در نوت بوک پروژه قرار داده شده است. در ادامه دو تا از مدل‌هایی که که در تست‌های متفاوت بهترین نتیجه را تولید کرده‌اند معرفی می‌کنیم و نتایج آن را ذکر می‌کنیم. این مدل‌ها عبارتند از مدل مبتنی برا LSTM بدون Attention و مدل مبتنی بر GRU و به همراه Attention:

مدل LSTM

ساختار مدل LSTM که عمدتاً الهام گرفته از پیشنهاد مقاله مرجع است، به صورت زیر می‌باشد:



همانطور که دیده می‌شود، یکبار تصویر اصلی و یکبار کراپ شده آن به resnet وارد می‌شوند و لاجیت خروجی ۱۰۰۰ تایی این شبکه به عنوان فیچرهای توصیف کننده تصویر مورد استفاده قرار می‌گیرند (البته ورودی‌ها ابتدا scale می‌شوند). گرچه شبکه resnet به کار گرفته شده pretrained است اما اجاز عبور گرادیان و آپدیت وزن‌ها به تمامی لایه‌های آن داده شده است. تست‌های متفاوتی در این زمینه انجام شد و در نهایت به نظر رسید این چینش بهترین گزینه است. به عبارت دیگر مدل‌هایی مثل vgg هم تست شدند و همچنین فریز کردن یا نکردن شبکه هم مورد بررسی قرار گرفت و در نهایت به نظر می‌رسد بهترین نتیجه مربوط به همین ساختار است.

بردار فیچر به دست آمده به عنوان هیدن اولیه به شبکه بازگشتی وارد می‌شود. البته در این قسمت با توجه به این که هیدن شبکه بازگشتی ۵۱۲ تایی فرض شده است، مشکل تفاوت ابعاد وجود خواهد داشت که با یک لایه FC (به نام Bridge) این مشکل مرتفع شده است.

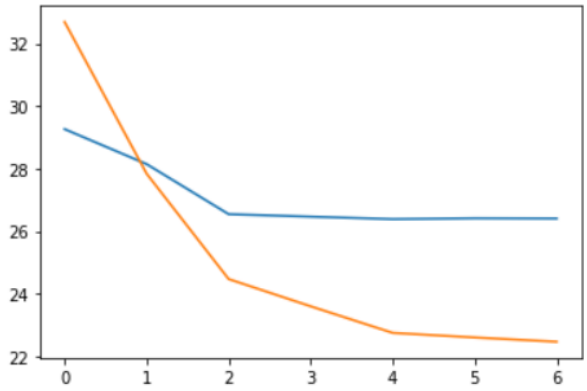
در ورودی سلول‌های LSTM کلمات وارد یک لایه امبدینگ قابل یادگیری می‌شوند و سپس بردار ۵۱۲ تایی به دست آمده با بردار ۲۰۰۰ تایی تصویر الحاق شده و به عنوان ورودی به سلول داده می‌شوند. اندازه امبدینگ کلمات در این مدل ۵۱۲ در نظر گرفته شده است.

در نهایت خروجی شبکه با کمک یک لایه FC از روی هیدن‌های هر گام به دست می‌آید. تابع ضرری که در آموزش این شبکه استفاده می‌شود، همان تابع مرسوم برای شبکه‌های بازگشتی است؛ یعنی بعد از اعمال softmax روی دنباله کلمات خروجی، کراس انتروپی توزیع به دست آمده و توزیع one-hot مورد هدف به عنوان معیار مورد استفاده قرار می‌گیرد و در نتیجه با ماکسیمم کردن log likelihood تولید جملات سعی می‌شود مدل آموزش داده شود.

در زمان آموزش از رویکرد Teacher forcing برای تزریق کلمات به داخل شبکه استفاده می‌شود اما در زمان تست و تولید جمله لازم است تا یک الگوریتم decoding معرفی شود. در اینجا دو الگوریتم Greedy Decode و Beam Search برای این منظور پیاده سازی شده است. نتایج نهایی که در اینجا ذکر می‌شود، حاصل الگوریتم Beam Search است که عملکرد نسبتاً بهتری نسبت به دیگری دارد.

در مورد بهینه ساز نیز لازم است این توضیح داده شود که دو بهینه ساز Adam و SGD (به همراه nestrov و momentum) مورد استفاده قرار گرفتند و در نهایت به نظر می‌رسد در task مورد نظر، عملکرد این دو نسبتاً مشابه یکدیگر می‌باشد. همچنین پارامتر learning rate نیز به پیشنهاد مقاله مرجع 1e-2 قرار داده شد و با گذشت یکی دو ایپاک به صورت StepLR با ضریب 0.1 آپدیت صورت گرفت.

با توصیفات ذکر شده، آموزش این شبکه انجام شد و نمودار خطای زیر برحسب تعداد epoch برای آن به دست آمد:



همانطور که مشاهده می‌شود بعد از گذشت ۷ ایپاک، خطای آموزش به چیزی حدود ۲۲ و خطای valid به حدود ۲۷ رسیده است.

معیار دیگری که برای ارزیابی این بخش استفاده شد، معیار Bleu است که کد آن در اختیار ما گذاشته شده است. کد داده شده به نوت بوک پروژه اضافه شد و معیارهای مد نظر روی داده‌های validation به صورت زیر اندازه گیری شده است:

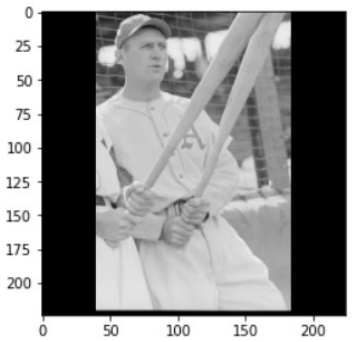
In [156]:
```
import pandas as pd
target_file = 'dataset/dataset.valid.csv'

df = pd.read_csv(target_file)
ngram = 1
print("blue-{}: {}".format(ngram,eval_bleu(df, decoded_sents, ngram)))
ngram = 2
print("blue-{}: {}".format(ngram,eval_bleu(df, decoded_sents, ngram)))
```

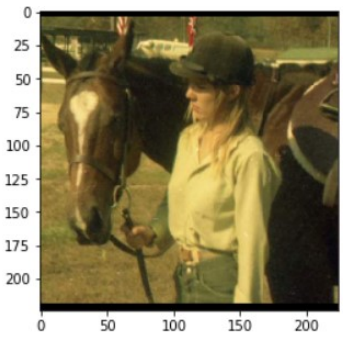
```
blue-1: 0.3878243019610898
blue-2: 0.22568921192343158
```

همچنین نمونه‌های تولید شده نیز از نظر چشمی به نظر تطابق خوبی داشتند که در ادامه دو نمونه از آن‌ها را نمایش می‌دهیم:

```
ground truth: A man with an A on his Jersey.
decoded: a baseball player holding a bat .
```



```
ground truth: The horse being led by the girl in the white shirt.
decoded: the horse on the left
```

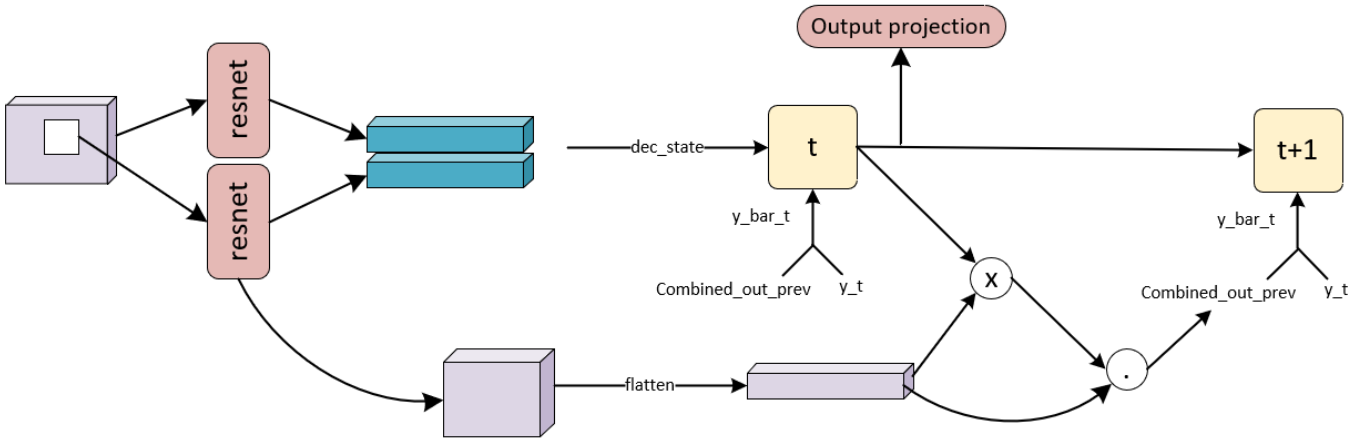


البته هنوز مقدار بلو با آن چیزی که هدف پروژه است اندکی فاصله دارد. علت این مسئله احتمالاً آن است که مدل یاد نگرفته جملات متعددی تولید کند و قالب‌های مشخصی را برای تولید متن یاد گرفته است. مخصوصاً این که بعضی از باکس‌های موجود در دیتاست با چندین جمله توصیف شده‌اند، می‌تواند به این مشکل دامن بزند. بنابراین شاید بهتر بود با روشی سعی می‌کردیم جملات متنوع‌تری تولید کنیم. البته در این راستا یکبار سعی شد تا در الگوریتم Beam Search از میان K تای باقی مانده نهایی، به جای max گرفتن، از سمپل گیری استفاده شود. با این حال این تکنیک بیشتر باعث شد تا جملات پرت‌تری تولید شوند و مقدار بلو از این هم کمتر شود.

توجه: من در ران‌های نهایی خودم در آخرین لحظات متوجه شدم تولید نقطه (.) در انتهای جملات Bleu را به مقدار یکی دو واحد کم می‌کند در صورتی که مدل آموزش یافته در انتهای جملات نقطه می‌گذارد. لذا آن را دستی از انتهای جملات حذف کردم تا این امتیاز بیشتر شود.

مدل شبکه Attention

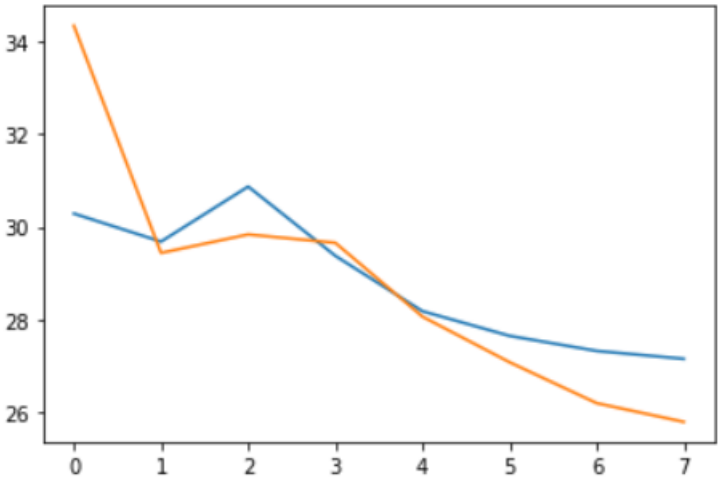
مدل دومی که در این گزارش به آن می‌پردازیم مدل مبتنی بر attention می‌باشد. برای پیاده سازی مکانیزم توجه، معماری‌های متفاوتی تست شد. در نهایت این جمع بندی حاصل شد که مدل زیر می‌تواند جواب‌های خوبی تولید نماید:



برای آن که بتوان مدل فوق را پیاده سازی نمود، در ابتدا لازم بود تا به تغییر دلخواه در ساختار resnet صورت پذیرد. با این تغییر، شبکه رزنت به جای یک خروجی، دو خروجی تولید می‌کند: اول مشابه حالت قبل یک بردار ۱۰۰۰ تایی که امتیاز تعلق تصویر به هر کلاس را نشان می‌دهد. دوم یک تصویر با ابعاد 2048 x 7 x7 که حاصل از conv لایه میانی چهارم است. بردارهای ۱۰۰۰ تایی مشابه قبل برای ساخت hidden اولیه استفاده می‌شود و تصویر دو بعدی به منظور اعمال attention استفاده می‌شود.

توجه کنید که اعمال attention روی بعد مکانی تصویر انجام می‌شود لذا با مشخصات فوق، لازم است تا ۴۹ وزن به منظور وزن‌دهی به هر یک از نواحی تصویر تولید شود. البته مسیر پایینی را فقط برای یکی از تصاویر رسم کرده‌ایم اما در پیاده‌سازی کد این مسئله در نظر گرفته شده است که هر تصویر یک attention جدا گانه برای خود نیاز دارد. به عبارت دیگر attention هم باید روی تصویر اصلی اعمال شود و هم روی تصویر ناحیه کراپ شده. سپس هر دو با هم concat شده و در ورودی سلول بعدی داده می‌شوند. توجه کنید که به دلیل ذات این معماری، امکان استفاده از torch.nn.GRU وجود نداشت و به جای آن از torch.nn.GRUCell استفاده شده است.

استفاده از سلول و for زدن روی بعد زمان باعث شد تا مدل و آموزش آن به طور جدی کند شود و رسیدن به جواب مطلوب را با مشکل مواجه کند. علی رغم این مشکلات، تا چندین ایپاک از مدل تست گرفته شد و نمودار پیشرفت آن به صورت زیر است:



همانطور که دیده می‌شود، مدل با شیب خیلی خوبی در حال کاهش تابع ضرر است. اما متاسفانه به دلیل مشکلات کولب، امکان تست کردن بیش از ۸ ایپاک وجود نداشت و به نظر می‌رسد مدل اتنشن به تعداد ایپاکی بیش از این برای حصول نتیجه لازم دارد. نمودار فوق و کد آموزش attention را می‌توانید در پوشه others در فایل‌های ارسال شده مورد بررسی قرار دهید اما توجه کنید که برای بخش‌های بعدی از این مدل استفاده نشده است.

بخش ادراک تصویر

اولین گام در بخش ادراک تصویر انجام fine tuning با کمک باکس مثبت و یک باکس منفی است که باکس دوم باید توسط شبکه Faster تولید شود. همانطور که در بخش‌های آغازین نیز عنوان شد، ابتدا باید داده‌ها یکبار از شبکه Faster عبور داده شوند تا باکس‌های آن‌ها تولید شود و سپس این باکس‌ها را تا زمان مورد نیاز در یک جای حافظه کش می‌کنیم.

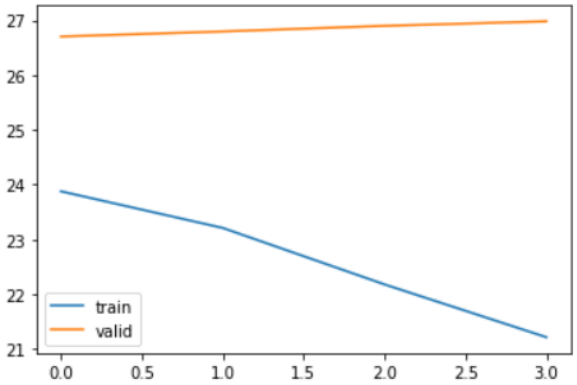
نکته‌ای که در اینجا اهمیت دارد و باید به آن پرداخته شود، نحوه انتخاب باکس منفی است. واضح است که به دلیل محدودیت محاسباتی نمی‌توان همه باکس‌های تولید شده توسط Faster را به عنوان باکس منفی در نظر گرفت. لذا اولاً در انتخاب باکس به این صورت عمل می‌شود که باکس‌هایی را که شبکه Faster خودش به آن‌ها احتمال خیلی پایینی نسبت داده را حذف می‌کنیم. سپس در میان باکس‌های باقی‌مانده می‌گردیم و باکس‌هایی که با باکس اصلی IoU کمتر از 0.5 دارند را دست چین می‌کنیم. از میان باکس‌های باقی‌مانده، در هربار فراخوانی داده، یکی را به صورت رندوم انتخاب کرده و برای fine tuning به مدل تحویل می‌دهیم.

در مورد تابع هزینه fine tuning، از رابطه زیر استفاده شده است:

$$\mathcal{J}(\theta) = -\left[\log p(S|R_{true}, I; \theta) + \lambda Max\left(0, \log p\left(S\middle|R_{false}, I; \theta\right) - \log p(S|R_{true}, I; \theta)\right)\right]$$

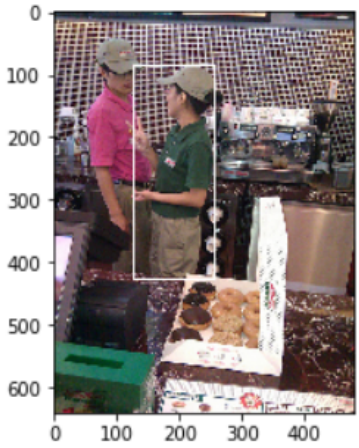
در رابطه فوق از لگاریتم احتمالات استفاده شده است چرا که خود مقادیر احتمالات مقادیر بسیار کوچکی هستند و کار با آن‌ها خیلی سخت است (توجه کنید که p احتمال جمله است که از ضرب احتمال شرطی تک تک کلمات به دست می‌آید لذا می‌تواند عدد خیلی کوچکی باشد). همچنین برای ابرپارامتر λ مقدار 1.5 در نظر گرفته شده است.

با توجه به این که در هر بار آموزش مدل با این تابع جدید لازم است تا شبکه بخش قبل را دو بار فرخوانی کنیم، لذا آموزش بسیار زمان‌گیر خواهد بود و با توجه به محدودیت منابع به این راحتی امکان پذیر نبود. به هر حال علی رغم وجود این جور مسائل fine tuning برای چهار ایپاک انجام شد (همین ۴ ایپاک هم برای من چیزی حدود ۷ ساعت طول کشید !!!) نمودار تابع ضرر برای این ایپاک‌ها به صورت زیر است:



بعد از انجام این fine tuning، نتایج خیلی جالبی روی داده‌های valid به دست می‌آید که در ادامه چند تا از آن‌ها به عنوان نمونه آورده شده است (باکس‌ها به طور خودکار توسط شبکه تولید شده‌اند)

```
Woman in Green shirt talking with her co-worker
[[126.79127502441406, 86.68758392333984, 129.4955291748047, 342.46768951416016]]
```




```
Man on skateboard performing tricks - grinding a rail
[[255.48043823242188, 22.571054458618164, 130.64913940429688, 190.75856590270996]]
```

در نهایت ارزیابی معیار precision با توجه به کدهای ارزیابی صورت گرفت و نتیجه آن روی داده‌های valid به صورت زیر است:

```
:  1 p_1, avg_IoU = eval_precision (df[0:len(predicted_boxes)],predicted_boxes)
   2 print("Precision: ", p_1)
   3 print("IoU: ", avg_IoU)
```

```
Precision:  0.533605720122574
IoU:  0.525950617201691
```

به عنوان آخرین بخش نوت بوک نیز هم برای فاز captioning و هم برای فاز comprehension فایل‌های تست تولید شد و در پوشه results قرار داده شده است.

نمونه‌ای از تولید متن روی دیتاست تست:

a black couch with a pillow on it

نمونه‌ای از تولید باکس روی دیتاست تست:

A glass of water is sitting on the table.