دانشکده فنی و مهندسی
گروه مهندسی کامپیوتر

پروژه پایانی دوره کارشناسی

انتقال سبک تصاویر بوسیله تکنیک‌های یادگیری عمیق

استاد پروژه:

دکتر محرم منصوری زاده

نام و نام خانوادگی دانشجو:

امیرحسین خدابنده لو

شماره دانشجویی:

۹۵۱۲۳۱۷۰۱۹

اسفندماه۱۳۹۹

چکیده

با توجه به اهمیت روز افزون دانش هوش مصنوعی و گسترش بیش از پیش کاربردهای شبکه عصبی بر آن شدیم تا پروژه پایانی کارشناسی را بر محوریت این موضوع قرار دهیم. شبکه‌های عصبی سیستم‌ها و روش‌های محاسباتی نوین برای یادگیری ماشینی، نمایش دانش و در انتها اعمال دانش به دست آمده در جهت بیش‌بینی پاسخ‌های خروجی از سامانه‌های پیچیده هستند. ایده اصلی این گونه شبکه‌ها تا حدودی الهام‌گرفته از شیوه کارکرد سیستم عصبی زیستی برای پردازش داده‌ها و اطلاعات به منظور یادگیری و ایجاد دانش میباشد. عنصر کلیدی این ایده، ایجاد ساختارهایی جدید برای سامانه پردازش اطلاعات است.

گسترش شبکه‌های عصبی راه را برای ورود آنها به حوزه تصاویر باز کرد که نتیجه آن ظهور شبکه های عصبی کانولوشنی[۱] میباشد. پس تصمیم گرفتیم در این پروژه از یکی از کاربردهای شبکه‌های عصبی کانولوشنی استفاده کنیم و از آن‌ها برای انتقال سبک[۲] بین تصاویر استفاده کنیم. نتیجه این امر به صورت استفاده از یک فیلتر بر روی تصویر مورد نظر ما خواهد بود در صورتی که فیلتر استفاده شده، همان تصویر دارای سبک ما خواهد بود.

با توجه به ماهیت این پروژه و الگوریتم، برای ارزیابی آن از شیوه ارزیابی کیفی استفاده کرده ایم.

کلمات کلیدی

هوش مصنوعی – شبکه‌های عصبی – انتقال سبک – شبکه‌های عصبی کانولوشنی – یادگیری ماشین

[۱] Convolutional Neural Networks

[۲] Style

فهرست مطالب

فهرست تصاویر

فهرست جداول

۱ فصل اول : توضیح و بیان مسئله

۱-۱ مقدمه

هوش مصنوعی[۱] که به اختصار به آن AI نیز گفته می‌شود، به هوشمند شدن ماشین‌ها اشاره دارد. هوش مصنوعی در سال ۱۹۵۶ به عنوان یک زمینه دانشگاهی مطرح شد و از آن زمان تاکنون چندین موج بهبود و شکست را به خود دیده است. به شکست‌هایی که در زمینه ارتقا این دانش به وقوع پیوسته، اصطلاحا «زمستان هوش مصنوعی» می‌گویند. به دنبال این شکست‌ها، رویکردهای نو، موفقیت‌های چشم‌گیر و سرمایه‌گذاری‌های جدیدی نیز در این زمینه صورت پذیرفته است. در راستای گستردگی موضوعات موجود در هوش مصنوعی، آن‌ها را به زیردسته‌هایی تقسیم کرده‌اند که اغلب ارتباطی با یکدیگر ندارند. این زیربخش‌های موضوعی بر مبنای ملاحظات فنی مانند اهداف خاص (برای مثال، رباتیک و یادگیری ماشین)، استفاده از ابزارهای مشخص (منطق یا شبکه‌های عصبی)، یا تفاوت‌های عمیق فلسفی تعیین شده‌اند.

اهداف دیرینه‌ای که در پژوهش‌های هوش مصنوعی همواره قصد پرداختن به آن‌ها وجود داشته شامل استدلال، ارائه دانش، برنامه‌ریزی، یادگیری، پردازش زبان طبیعی، ادراک و توانایی حرکت دادن و دست‌کاری کردن اشیا می‌شود. رویکردهای هوش مصنوعی شامل روش‌های آماری، هوش محاسباتی و هوش مصنوعی نمادین سنتی می‌شود. ابزارهای زیادی از جمله بهینه‌ساز ریاضیاتی و جست‌وجو، شبکه‌های عصبی، روش‌های مبتنی‌بر آمار و احتمالات و اقتصاد نیز در هوش مصنوعی مطرح هستند. دانش یاد شده، در زمره علوم کامپیوتر، ریاضیات، روانشناسی، زبان‌شناسی، فلسفه و بسیاری از دیگر علوم می‌گنجد.

با توجه به اهمیت روز افزون دانش هوش مصنوعی و گسترش بیش از پیش کاربردهای شبکه عصبی تصمیم گرفتیم تا بر روی کاربرد های شبکه های عصبی متمرکز شویم. شبکه‌های عصبی سیستم‌ها و روش‌های محاسباتی نوین برای یادگیری ماشینی، نمایش دانش و در انتها اعمال دانش به دست آمده در جهت بیش‌بینی پاسخ‌های خروجی از سامانه‌های پیچیده هستند. ایده اصلی این گونه شبکه‌ها تا حدودی الهام‌گرفته از شیوه کارکرد سیستم عصبی زیستی برای پردازش داده‌ها و اطلاعات به منظور یادگیری و ایجاد دانش میباشد. عنصر کلیدی این ایده، ایجاد ساختارهایی جدید برای سامانه پردازش اطلاعات است.

گسترش شبکه‌های عصبی راه را برای ورود آنها به حوزه تصاویر باز کرد که نتیجه آن ظهور شبکه های عصبی کانولوشنی[۲] میباشد. پس تصمیم گرفتیم در این پروژه از یکی از کاربردهای شبکه‌های عصبی کانولوشنی استفاده

---

[۱] Artificial Intelligence

[۲] Convolutional Neural Networks

کنیم و از آن‌ها برای انتقال سبک بین تصاویر استفاده کنیم. نتیجه این امر، به صورت استفاده از یک فیلتر بر روی تصویر مورد نظر ما خواهد بود در صورتی که فیلتر استفاده شده، همان تصویر دارای سبک ما خواهد بود.
انتقال سبک به کاربرد شبکه های عصبی، برای تبدیل تصویر به گونه ای که از لحاظ هنری شبیه به تصویر دیگری شود، اشاره دارد، در حالی که محتوای اصلی خود را حفظ می کند. انتقال سبک عصبی[۱] به عنوان روشی برای تغییر زیبایی شناسی یک تصویر به سرعت در حال محبوب شدن است.
در قسمت بعدی ابتدا به بررسی مفاهیم مورد نیاز جهت مسئله میپردازیم و در فصل آینده مسئله را پیاده سازی میکنیم.

۲-۱ ابزارهای مورد نیاز

برای حل این مسئله از زبان برنامه نویسی پایتون وکتابخانه های آن استفاده میکنیم که در زیر برخی از این کتابخانه‌ها به صورت مختصر توضیح داده شده اند:

- PyTorch :

این کتابخانه اصلی ترین کتابخانه برای حل مسئله میباشد که یک کتابخانه متن باز میباشد و براساس کتابخانه Torch ایجاد شده است. از این کتابخانه برای کار با انواع شبکه‌های عصبی در کاربردهایی مشابه بینایی ماشین[۲] و پردازش زبان طبیعی[۳] استفاده میشود .

- PIL[4]:

از این کتابخانه جهت کار با تصاویر استفاده میشود.

- Matplotlib:

از این کتابخانه هم جهت نمایش تصاویر استفاده میشود.

۱ Neural Style Transfer
۲ Machine Vision
۳ Natural Language Processing
4 Python Imaging Library

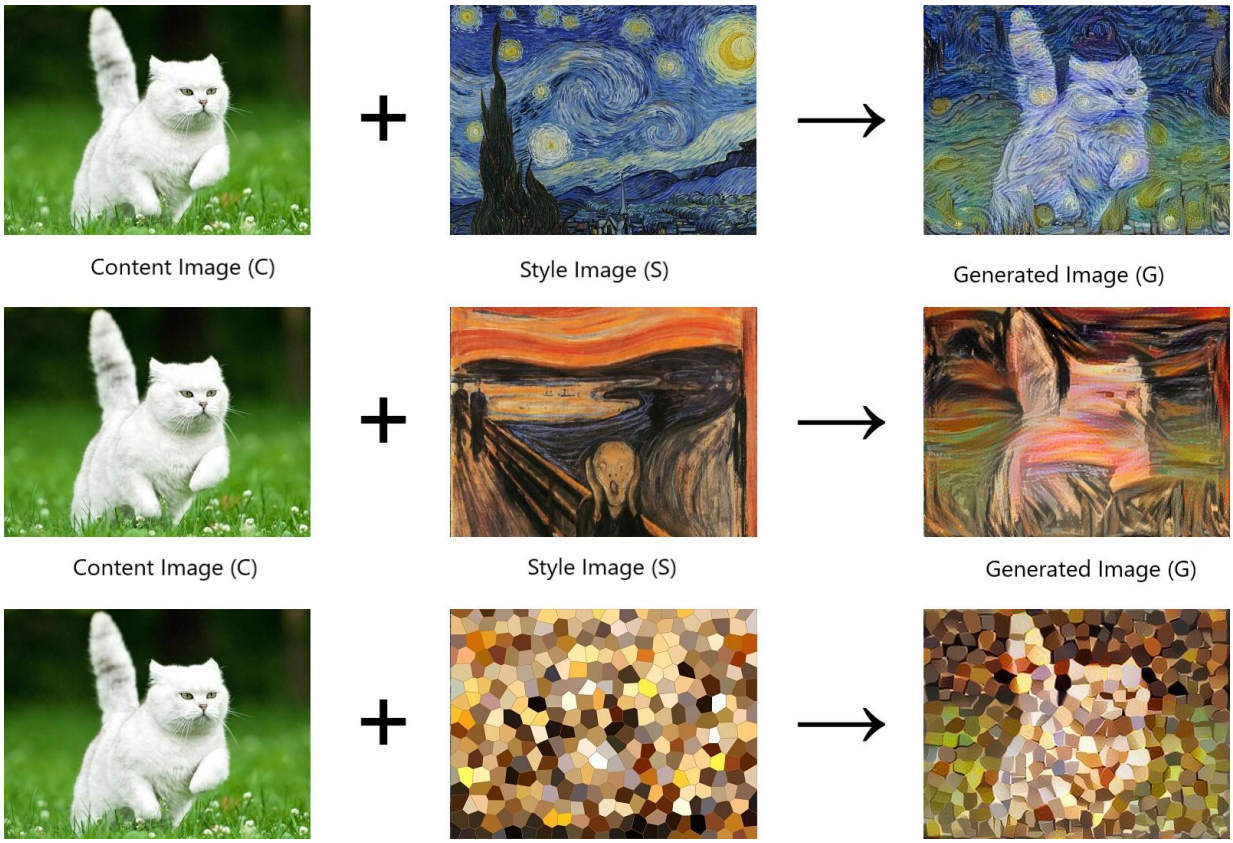
۳-۱ انتقال سبک یا Neural Style Transfer چیست؟

انتقال سبک یک تکنیک بهینه سازی است که با ۳ تصویر سر و کار دارد: یک تصویر محتوا (Content)، یک تصویر مرجع از سبک مورد نظر(Style) (تصویری که میخواهیم سبک هنری آن را استخراج کنیم) و تصویر خروجی و تولید شده از این دو تصویر.

پس ما میخواهیم دو تصویر content و style را باهم ترکیب کرده و یک تصویر خروجی بگیریم که آن تصویر مشابه تصویر محتوا است ولی به سبک تصویر style رنگ آمیزی شده است.

تصویر content را به اختصار با C، تصویر style را به اختصار با S و تصویر تولید شده را به اختصار با G نمایش میدهیم.

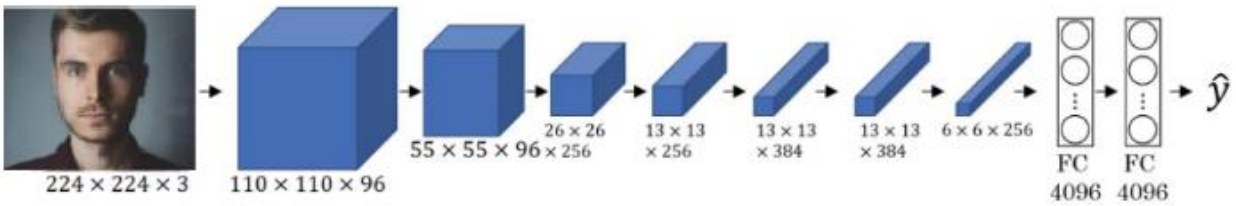
در تصویر نمونه هایی از کارکرد این الگوریتم مشاهده میکنید :



شکل ۱-۱: مثال از عملیات انتقال سبک

۱-۴ شبکه های کانولوشنی عمیق (ConvNet) چه چیزی یاد میگیرند؟

فرض میکنیم که شما یک شبکه کانولوشنی را مشابه شکل زیر آموزش داده اید و میخواهید ببینید که واحدهای پنهان[۱] آن در لایه های مختلف چه چیزی را محاسبه میکنند.



شکل ۲-۱ : شبکه عصبی کانولوشنی

فرض میکنیم که شما میتوانید در مجموعه داده های آموزشی جستجو کنید و بفهمید که کدام تصاویر یا کدام قسمت‌هایی از تصویر فعال ساز unit مورد نظر را به حداکثر مقدار خود میرساند.

برای شروع فرضا از لایه اول یک unit را انتخاب میکنیم. برای این unit باید ۹ تیکه از تصویر ورودی را که فعال ساز آن unit را به حداکثر مقدار خود میرساند پیدا کنیم. باید توجه داشته باشیم که یک unit در لایه اول فقط قسمت بسیار کوچکی از شبکه عصبی را میبیند و شامل میشود. اگر بتوانیم چیزی که فعال ساز آن unit را فعال میکند نمایش دهیم، قسمت هایی کوچک از تصویر خواهد بود چرا که آن unit فقط این مقدار از تصویر را میتواند ببیند.

پس اگر یک hidden unit را از لایه اول انتخاب کنیم و ۹ تصویر ورودی که فعال ساز آن unit را به حداکثر مقدار خود میرساند را پیدا کنیم، تصویری مشابه زیر خواهیم داشت :

شکل ۳-۱ : مثال از ۹ قسمت از تصویری که فعال ساز unit را به حداکثر مقدار خود میرسانند.

که به نظر میرسد در قسمت های پایینی تصویری که این unit مشاهده میکند، این unit به دنبال یک خط مورب با شیب منفی میباشد.

پس اینها ۹ قسمت از تصویر هستند که فعال ساز این unit را به حداکثر مقدار خود رساندند.

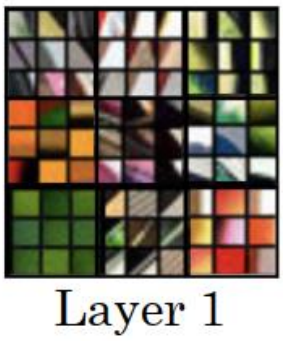
میتوان اینکار را برای unit های دیگر لایه ۱ انجام داد. به عنوان مثال فرضا یک unit دیگر به دنبال قسمت های سبز رنگ تصویر میباشد:

---

[۱] Hidden Units

شکل ۴-۱ : تصاویر مربوط به unit ای که به دنبال مناطق سبز تصویر میگردد

اگر فرضا برای لایه اول ۹ عدد unit انتخاب کنیم و بررسی کنیم که فعال ساز هر unit با چه قسمت هایی از تصویر به حداکثر مقدار خود میرسند، به تصویری مشابه شکل زیر برای لایه اول میرسیم:



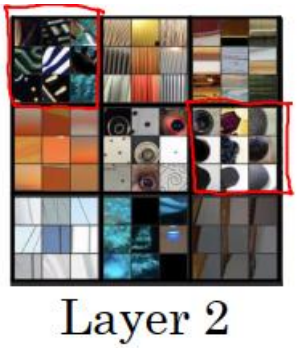
شکل ۵-۱ : نمونه خروجی از تصاویر لایه اول

نتیجه ای که از این تصویر میگیریم این است که unit های لایه اول اغلب به دنبال ویژگی های ساده مثل یک خط مورب یا یک سایه از رنگ خاصی هستند.

حالا اگر همین کار را برای لایه های عمیق تر شبکه انجام دهیم، متوجه خواهیم شد که شبکه عصبی چه چیزی را در لایه های عمیق تر یاد میگیرد و اینکه لایه های عمیق تر دنبال چه نوع ویژگی هایی از تصویر هستند.

در لایه های عمیق تر هر unit بخش بزرگتری از تصویر را میبیند و قسمت های بزرگتری از تصویر را در برمیگیرد.

اگر عملیات را قبل را برای لایه دوم انجام دهیم، به تصویر زیر میرسیم که مشخص میکند چه چیزی فعال ساز هرکدام از ۹ عدد unit را به حداکثر مقدار خود میرساند.



شکل ۶-۱ : نمونه خروجی از تصاویر لایه دوم

هر کدام از قسمت های مشخص شده به عنوان مثال، ۹ قسمتی هستند که فعال ساز یک unit را به حداکثر مقدار خود میرسانند. که این تصویر مشابه قسمت قبل، ۹ عدد hidden unit را مشخص میکند که هر کدام از آنها ۹ قسمت از تصویر را که باعث میشوند فعال ساز و خروجی بسیار بزرگی داشته باشند را مشخص میکند.

همانطور که مشخص است این لایه الگوها و شکل های پیچیده تری را نسبت به لایه اول تشخیص میدهد.

به عنوان مثال یکی از unit ها در لایه دوم همانطور که ملاحظه میکنید به دنبال یک حالت نیم دایره در تصویر میگردد.

پس ویژگی های خروجی این لایه در حال پیچیده تر شدن هستند.

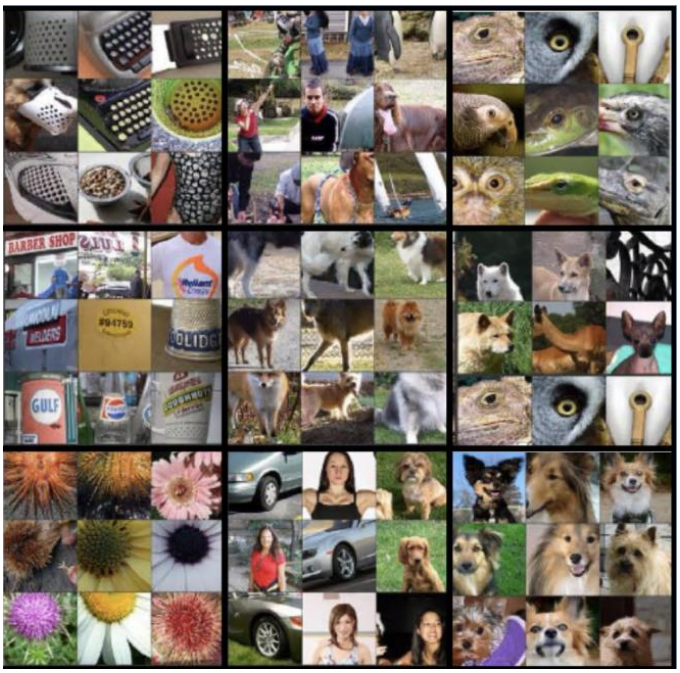
اگر همین کار را برای لایه های عمیق تر انجام دهیم به نتایج زیر میرسیم:

Layer 3

شکل ۷-۱ : نمونه خروجی از تصاویر لایه سوم

همانطور که مشخص است این لایه قسمت های بزرگتری از تصویر را میبیند و به مراتب الگوها و ویژگی های پیچیده‌تری را تشخیص میدهد، به عنوان مثال یک unit انسان ها را در تصویر تشخیص میدهد.

به عنوان نمونه آخر در تصویر زیر خروجی را برای لایه ۴ مشاهده میکنید :



شکل ۸-۱ : نمونه خروجی از تصاویر لایه چهارم

در این لایه الگوها به مراتب پیچیده شدند و هر unit بخش های بزرگتری از تصویر را در بر میگیرد و ویژگی های بسیار پیچیده تری را تشخیص میدهد مشابه چهره انواع حیوانات و پرندگان، متن روی اشیا و ...

از تشخیص ویژگی های بسیار ساده مشابه لبه ها و خطوط مورب در لایه های ابتدایی شبکه به تشخیص ویژگی ها و اشیا بسیار پیچیده در لایه های عمیق تر شبکه رسیدیم.

پس به عنوان نتیجه گیری، لایه های ابتدایی شبکه قسمت های کوچکی از تصویر را میبینند و ویژگی های ساده ای را تشخیص میدهند در حالی که لایه های عمیق تر شبکه قسمت های بزرگتری از تصویر را میبینند و ویژگی های پیچیده تری را تشخیص میدهند.

۱-۵ تابع هزینه

برای سیستم انتقال سبک باید یک تابع هزینه تعریف کنیم که با مینیمایز کردن آن، به تصویر تولید شده موردنظر برسیم.

گفتیم که یک تصویر Content (C) ، یک تصویر Style (S) داریم و هدفمان تولید تصویر جدیدی به اسم Generated Image (G) میباشد. پس برای پیاده سازی سیستم انتقال سبک یک تابع هزینه به اسم J(G) تعریف

میکنیم که مشخص میکند به چه میزان تصویر تولید شده خوب است و بوسیله یک الگوریتم بهینه سازی مشابه الگوریتم کاهش گرادیان، سعی میکنیم تا تابع J(G) را به حداقل مقدار ممکن برسانیم تا تصویر مناسبی تولید کنیم.
این تابع هزینه از دو بخش تشکیل شده است:

۱) تابع هزینه تصویر محتوا یا content که تابعی از تصویر Content وتصویر تولید شده میباشد. کاری که این تابع میکند این است که میزان شباهت محتوای تصویر تولید شده (G) به محتوای تصویر محتوا (C) را اندازه گیری میکند :

$J_{content}(C, G)$

۲) تابع هزینه سبک که تابعی از تصویر Style و تصویر تولیده شده میباشد. وظیفه این تابع اندازه گیری میزان شباهت سبک تصویر تولید شده (G) به سبک تصویر Style (S) میباشد :

$J_{style}(S, G)$

در نهایت تابع هزینه کلی تصویر تولید شده، جمع وزن دار این دو تابع است:

$$J(G) = \alpha J_{content}(C, G) + \beta J_{style}(S, G)$$

که آلفا و بتا دو hyperparameter هستند که وزن نسبی بین دو تابع هزینه محتوا و تابع هزینه سبک را مشخص میکنند. در واقع اینکه کدامیک تاثیر بیشتری بر روی تابع هزینه کلی بگذارند را مشخص میکنند.

با تعریف تابع هزینه J(G) ، برای تولید یک تصویر جدید باید کارهای زیر را انجام دهیم:

۱) مقادیر پیکسل های تصویر G را به صورت تصادفی و با ابعاد تصویر مورد نظر خود مقدار دهی میکنیم، به عنوان مثال :

G: 100* 100 * 3

۲) سپس به وسیله تابع هزینه تعریف شده J(G) ، از یک الگوریتم بهینه سازی مشابه کاهش گرادیان استفاده میکنیم تا تابع هزینه را به حداقل برسانیم و در هر مرحله پیکسل های تصویر G را آپدیت میکنیم.

۱-۶ تابع هزینه محتوا $J_{contnet}(C, G)$

فرض میکنیم که لایه l (ال) را برای محاسبه هزینه content یا محتوا انتخاب میکنیم. اگر l یک مقدار کوچک باشد، به عنوان مثال لایه اول، باعث میشود پیکسل های تصویر تولید شده نهایی بسیار مشابه پیکسل های تصویر محتوا باشند.

چرا که همانطور که در بخش های قبلی توضیح داده شد، لایه اولیه شبکه عصبی قسمت های کوچکی از تصویر را میبینند و باعث میشوند ریزترین جزئیات را به تصویر تولید شده G انتقال دهند و باعث شوند تصویر تولیدی بسیار مشابه تصویر محتوا باشد. اما اگر l یک مقدار بزرگ باشد و یک لایه عمیق را برای محاسبه هزینه استفاده کنید، باعث میشود تصویر تولیدی شباهت خود به تصویر محتوا را از دست بدهد، چرا که همانطور که توضیح داده شد لایه های عمیق تر شبکه بخش های بزرگتری از تصویر را میبینند و جزئیات زیادی از تصویر را منتقل نمیکنند. به همین دلیل لایه l باید مقداری متوسط داشته باشد و از لایه های وسط شبکه عصبی انتخاب شود.

در مرحله بعد یک شبکه عصبی کانولوشنی از پیش آموزش دیده را استفاده میکنیم. مشابه شبکه VGG.

حالا میخواهیم با داشتن یک تصویر محتوا یا content و یک تصویر تولید شده که پیکسل های آن مقادیر تصادفی دارند، مشخص کنیم که این دو تصویر از نظر محتوا چه میزان بهم شباهت دارند. سپس به کمک الگوریتم های بهینه سازی و تابع هزینه هر بار محتوای تصویر تولید شده را به محتوای تصویر محتوا مشابه تر کنیم.

با توجه به توضیحات فرض میکنیم $a^{[l](C)}$ خروجی activation لایه l بر روی تصویر محتوا و $a^{[l](G)}$ خروجی activation لایه l بر روی تصویر تولید شده (G) باشند.

حال اگر مقادیر $a^{[l](C)}$ و $a^{[l](G)}$ مشابه باشند، یعنی هر دو تصویر محتوا یکسانی دارند. پس تفاوت این دو مقدار نشان‌دهنده میزان شباهت خواهد بود.

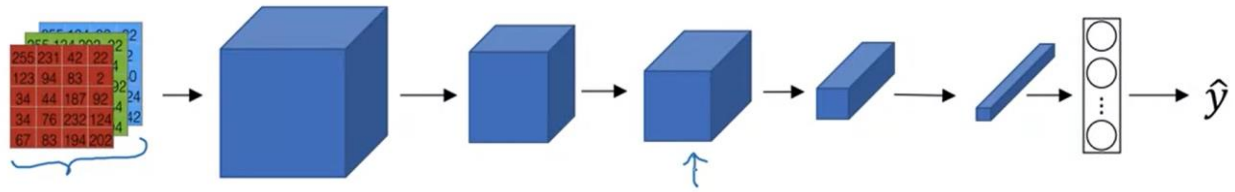
پس تابع هزینه محتوا به شکل زیر تعریف میشود :

$$J_{content}(C,G) = \left(\frac{1}{2}\right) ||a^{[l](C)} - a^{[l](G)}||^2$$

- $a^{[l](C)}$: خروجی activation لایه l شبکه عصبی کانولوشنی با ورودی تصویر C یا همان تصویر محتوا.
- $a^{[l](G)}$: خروجی activation لایه l شبکه عصبی کانولوشنی با ورودی تصویر G یا همان تصویر تولید شده.
- مقدار (۱/۲) برای نرمال سازی قرار داده شده است که قابل حذف میباشد.

۱-۷ تابع هزینه سبک $J_{style}(S, G)$

برای محاسبه تابع هزینه سبک، ابتدا باید بفهمیم که « سبک » یک تصویر به چه معناست.

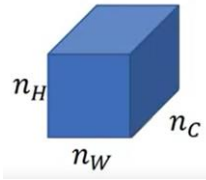


شکل ۹-۱ : نمونه شبکه عصبی کانولوشنی برای توضیح سبک یک تصویر

فرض کنید یک تصویر ورودی و یک شبکه کانولوشنی به صورت بالا داریم که در لایه های مختلف شبکه ویژگی‌های تصویر محاسبه میشوند. هم چنین فرض کنید یک لایه l را انتخاب کرده ایم تا سبک تصویر را اندازه گیری کنیم.

سبک تصویر ورودی به صورت زیر تعریف میکنیم : « سبک یک تصویر به معنی همبستگی بین activation ها یا فعال‌ساز ها بین کانال های مختلف تصویر در activation این لایه L میباشد.»

فرض کنید که activation لایه L را به صورت زیر داریم :

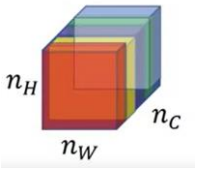


شکل ۱۰-۱ : فعال ساز لایه L

که ابعاد آن برابر است با : $n_H * n_w * n_c$

سوال این است که activation ها بین کانال های مختلف که تعداد آنها n_c عدد میباشد، به چه میزان همبستگی دارند؟

برای بیشتر مشخص شدن موضوع، این بلاک از activation ها را رنگ بندی کرده و هر کانال را با یک رنگ متفاوت نمایش میدهیم :

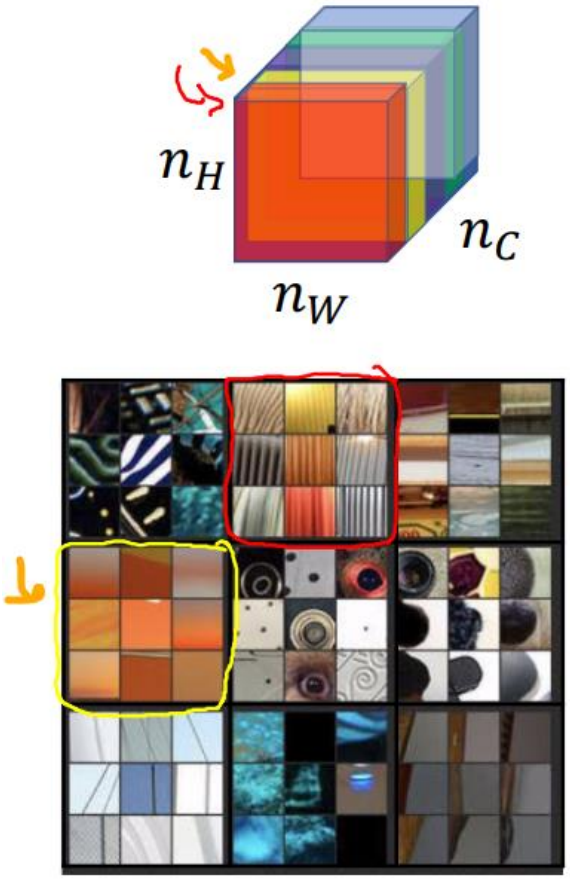


شکل ۱۱-۱ : فعال ساز لایه l به صورت رنگ بندی شده

که در این مثال فرضا ۵ کانال داریم.

برای بدست آوردن سبک یک تصویر به شکل زیر عمل میکنیم:

activation های کانال های قرمز و زرد به چه میزان بهم همبستگی دارند؟ برای این کار به اعداد این دو بردار نگاه میکنیم و هر عدد را با عدد متناظر آن از نظر موقعیت مکانی مقایسه میکنیم تا ببینیم این دو کانال چه میزان بهم همبستگی دارند. این کار را برای کانال های دیگر هم انجام میدهیم.



شکل ۱۲-۱ : فعال ساز لایه I به همراه تصاویری که مقدار آن را حداکثر مقدار ممکن میکنند.

به عنوان مثال در تصویر بالا، کانال قرمز مربوط به خروجی نورونی است که با رنگ قرمز دور آن خط کشیده ایم و به دنبال پیداکردن بافت های عمودی در تصویر میباشد و کانال زرد هم مربوط به خروجی نورونی است که با رنگ زرد دور آن خط کشیده ایم و به نظر دنبال قسمت های نارنجی رنگ در تصویر میباشد.

معنی همبستگی برای این دو کانال این است که هر وقت جایی در تصویر بافت های عمودی شکل وجود داشته باشد، آن قسمت از تصویر به رنگ نارنجی باشد.

اما معنی ناهمبستگی برای آن ها این است که هر وقت جایی در تصویر بافت های عمودی شکل وجود داشته باشد، آن قسمت از تصویر به رنگ نارنجی نباشد.

پس درجه همبستگی مشخص میکند که به چه میزان در تصویر ورودی این دو ویژگی (داشتن بافت عمودی و نارنجی رنگ بودن) اتفاق می افتد.

پس ما میتوانیم درجه همبستگی را بین کانال های مختلف به عنوان یک معیار اندازه گیری سبک استفاده کنیم.

در نتیجه در تصویر تولید شده میتوانیم درجه همبستگی را به طور مشابه اندازه گیری کنیم. به عنوان مثال کانال قرمز چه میزان با کانال زرد همبستگی یا ناهمبستگی دارد و این موضوع مشخص میکند که در تصویر تولید شده، بافت‌های عمودی چه میزان با رنگ های نارنجی ظاهر شده اند یا ظاهر نشده اند و این یک معیار اندازه گیری به ما میدهد که سبک تصویر تولید شده(G) تا چه میزان به سبک تصویر سبک ورودی(S) شباهت دارد.

پس شروع به فرموله سازی این موضوع میکنیم.

برای این کار باید ماتریسی به اسم Style Matrix را محاسبه کنیم که تمام همبستگی های توضیح داده شده را اندازه گیری میکند.

$a^{[l]}_{i,j,k}$ = activation at (i, j, k)

فرض میکنیم $a^{[l]}_{i,j,k}$ فعال ساز یا activation در مختصات i,j,k در لایه l باشد.

پس i روی ارتفاع، jروی عرض و k روی کانال های مختلف اندیس گذاری میکند.

برای محاسبه Style Matrix باید ماتریس $G^{[l]}$ را محاسبه کنیم که یک ماتریس با ابعاد $n^{[l]}_c * n^{[l]}_c$ میباشد، چرا که قرار است همبستگی دو به دوی کانال ها را اندازه گیری کنیم.($n^{[l]}_c$ تعداد کانال های لایه l را مشخص میکند.)

پس $G^{[l]}_{kk'}$ میزان همبستگی activation ها در کانال k را در مقایسه با با کانال k′ اندازه گیری میکند.

برای محاسبه $G^{[l]}_{kk'}$ داریم :

$$G^{[l]}_{kk'} = \sum_{i=1}^{n_H^{[l]}} \sum_{j=1}^{n_w^{[l]}} \left(a^{[l]}_{i,j,k} * a^{[l]}_{i,j,k'} \right)$$

پس این فرمول روی موقعیت های مختلف تصویر میچرخد که iوjروی ارتفاع و عرض بلوک ها میچرخند و اندیس گذاری میکنند و k و k′ بر روی کانالها میچرخند و اندیس گذاری میکنند و مقدار آن ها از ۱ تا تعداد کل کانالها در لایه l میباشد.

k,k′ = 1 ... $n_c^{[l]}$

این فرمول را باید برای هر مقدار k و k′ انجام دهیم تا ماتریس کلی $G^{[l]}$ یا Style Matrix را محاسبه کنیم.

نکته حائز اهمیت این است که اگر هر دو activation موجود در فرمول بزرگ باشند، پس $G^{[l]}_{kk'}$ بزرگ خواهد بود، در حالی که اگر دو کانال همبستگی نداشته باشند $G^{[l]}_{kk'}$ کوچک خواهد بود.

پس بوسیله این فرمول و برای هر مقدار k و k′ ، سبک یا style تصویر مورد نظر محاسبه میشود و باید این فرآیند را برای هر دو تصویر سبک (S) و تصویر تولیدی (G) محاسبه کنیم. یعنی :

$$G_{kk\prime}^{[l](S)} = \sum_{i=1}^{n_H^{[l]}} \sum_{j=1}^{n_w^{[l]}} \left(a_{i,j,k}^{[l](S)} \ * \ a_{i,j,k\prime}^{[l](S)} \right)$$
$$G_{kk\prime}^{[l](G)} = \sum_{i=1}^{n_H^{[l]}} \sum_{j=1}^{n_w^{[l]}} \left(a_{i,j,k}^{[l](G)} \ * \ a_{i,j,k\prime}^{[l](G)} \right)$$

پس دو فرمول داریم که فرمول اول استایل یا سبک تصویر S را اندازه گیری میکند و فرمول دوم استایل یا سبک تصویر G را اندازه گیری میکند.

در جبر خطی به این ماتریس ها، Gram Matrix هم گفته میشود و به همین دلیل با G نمایش داده میشوند.

پس تابع هزینه استایل یا سبک لایه L به صورت زیر تعریف میشود :

$$J_{style}^{[l]}(S,G) = \frac{1}{(2n_H^{[l]} n_W^{[l]} n_c^{[l]})^2} \left|\left| G^{[l](S)} - \ G^{[l](G)} \right|\right|^2$$

که به عبارت بهتر برابر است با :

$$J_{style}^{[l]}(S,G) = \frac{1}{(2n_H^{[l]} n_W^{[l]} n_c^{[l]})^2} \sum_{k} \sum_{k\prime} (G_{kk'}^{[l](S)} - G_{kk'}^{[l](G)})^2$$

این فرمول تابع هزینه سبک برای لایه L میباشد.

توجه داشته باشید که کسر اولیه برای نرمال سازی است و قابل حذف نیز میباشد.

در نهایت به نظر میرسد که اگر تابع هزینه سبک را برای چند لایه مختلف محاسبه کنیم، نتایجی بهتری خواهیم گرفت.

بنابراین تابع هزینه سبک کلی به صورت زیر قابل تعریف است :

$$J_{style}\ (S,G) = \sum_{l} \lambda^{[l]} * J_{style}^{[l]}(S,G)$$

که این فرمول تابع هزینه سبک را به ازای همه لایه ها محاسبه کرده و با هم جمع میکند.

در این فرمول تابع هزینه سبک هر لایه در یک ضریب λ ضرب میشود تا وزن دهی هر لایه برای مشخص کردن میزان تاثیر گذاری آن در تابع هزینه کلی مشخص شود. پس λ هم یک hyperparameter میباشد.

پس این فرمول کمک میکند تا لایه های مختلف یک شبکه عصبی را استفاده کنیم، هم لایه های اولیه که ویژگی های سطح پایین را اندازه گیری میکنند و هم لایه های عمیق تر که ویژگی های پیچیده تر و سطح بالا را اندازه گیری

میکنند، و باعث میشود تا شبکه عصبی هم همبستگی های سطح پایین و هم سطح بالا را هنگام محاسبه سبک در نظر بگیرد.

پس میتوانیم تابع هزینه کلی را به صورت زیر تعریف کنیم :

$$J(G) = \alpha J_{content}(C, G) + \beta J_{style}(S, G)$$

و از یک الگوریتم بهینه سازی مشابه کاهش گرادیان یا الگوریتم های بهینه سازی پیچیده تر مشابه Adam استفاده کنیم تا هزینه J(G) را به حداقل برسانیم و به تصویر مطلوبی از برسیم که همان تصویر G خواهد بود.

۲ فصل دوم : پیاده سازی عملی

۲-۱ مقدمه

برای پیاده سازی پروژه از زبان پایتون وکتابخانه PyTorch که بیشتر برای انجام عملیات های مربوط به شبکه های عصبی و یادیگری عمیق مورد استفاده قرار میگیرد استفاده میکنیم.

جهت یاد آوری قرار شد که ما دو هزینه ، تابع هزینه محتوا و تابع هزینه سبک را محاسبه کنیم. تابع هزینه محتوا میزان تفاوت محتوای دو تصویر محتوا و تصویر جدید را اندازه میگیرد و تابع هزینه سبک هم میزان تفاوت سبک یا استایل دو تصویر استایل و تصویر جدید را اندازه میگیرد.

برای اینکار یک تصویر جدید را به همراه نویز مقدار دهی میکنیم و در هر مرحله تابع هزینه محتوا و سبک را برای آن محاسبه کرده و سعی میکنیم تا تابع هزینه کلی تصویر را به حداقل مقدار ممکن برسانیم.

۲-۲ اضافه کردن کتابخانه های مورد نیاز

پس در قدم اول کتابخانه های مورد نیاز را اضافه میکنیم:

```
from __future__ import print_function

import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim

from PIL import Image
import matplotlib.pyplot as plt

import torchvision.transforms as transforms
import torchvision.models as models

import copy
import numpy as np
```

- کتابخانه های torch, torch.nn, numpy برای پیاده سازی شبکه های عصبی با PyTorch استفاده میشوند.
- کتابخانه torch.optim برای استفاده از توابع مربوط به بهینه سازی استفاده میشود.
- کتابخه PIL برای کار با تصاویر در پایتون مورد استفاده قرار میگیرد.
- کتابخانه matplotlib جهت نمایش تصاویر و نمودار استفاده میشود.
- کتابخانه torchvision.transforms برای تبدیل تصاویر به tensor ها مورد استفاده قرار میگیرد.

- کتابخانه torchvision.models برای آموزش یا استفاده از مدلهای از پیش آموزش دیده (pre-trained) استفاده میشود.
- کتابخانه copy هم یک کتابخانه سیستمی برای کپی کردن مدل ها میباشد.

در قدم بعدی برای اینکه بتوانیم از GPU در پردازشها و load مدلها استفاده کنیم، از تکه کد زیر استفاده میکنیم که اگر GPU در دسترس بود device ما GPU خواهد بود در غیر اینصورت CPU خواهد بود.
بدیهی است سرعت پردازش ها بر روی GPU بسیار سریعتر از CPU خواهد بود.

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

۳-۲ اضافه کردن تصاویر

برای اضافه کردن تصاویر از تکه کد زیر استفاده میکنیم :

```
imsize = 512 if torch.cuda.is_available() else 128  # use small size if no gpu

loader = transforms.Compose([
    transforms.Resize(imsize),
    transforms.ToTensor()])

def image_loader(image_name):
    image = Image.open(image_name)

    image = loader(image).unsqueeze(0)
    return image.to(device, torch.float)
```

متغیر imsize در صورتی GPU در دسترس باشد، مقدار ۵۱۲ و در غیر اینصورت مقدار ۱۲۸ میگیرد، به این دلیل که هرچه سایز تصویر بیشتر باشد، محاسبات سنگین تری خواهیم داشت. از این متغیر برای تغییر اندازه سایز تصاویر استفاده میشود.

در قسمت بعدی یک loader تعریف کرده ایم که از ماژول transforms استفاده کرده است و قرار است تصویر را ابتدا به وسیله تابع Resize به اندازه متغیر imsize تغییر اندازه دهد تا تصاویر سایز یکسانی داشته باشند و سپس آن تصویر را به بوسیله تابع ToTensor به tensor تبدیل کند تا در محاسبات بتوانیم استفاده کنیم. تابع Compose برای اینکه بتوانیم تبدیلات مختلف را اضافه کنیم، استفاده میشود.

نکته حائز اهمیت این است که تصاویر دارای پیکسلهایی با مقادیر 0 تا 255 هستند در حالی که زمانی که به tensor تبدیل میشوند، مقادیر آن ها به بازه 0 تا 1 تبدیل میشوند. هم چنین شبکه های عصبی کتابخانه torch بوسیله مقادیر با بازه 0 تا 1 آموزش دیده اند، برای همین باید به این نکته توجه داشته باشیم.

درتابع image_loader ابتدا نام تصویر را گرفته و تصویر را مورد نظر را در متغیر image قرار میدهیم و سپس تبدیلات مورد نظر را که گفته شد را بوسیله loader بر روی تصویر اعمال میکنیم و توسط متد unsqueeze(0) یک بعد به تصویر اضافه میکنیم، دلیل آن این است که ورودی مدل مورد نظر یک tensor ۴ بعدی است که بعد اول آن تعداد تصاویر را مشخص میکند و ۳ بعد بعدی ابعاد تصویر هستند، چرا که تصاویر را به صورت دسته ای به مدل میدهیم، برای همین در این جا یک بعد به تصویر اضافه میکنیم که مقدار ۱ دارد تا بتوانیم آن را به عنوان ورودی به مدل بدهیم. در نهایت در این تابع تصویر را به device مورد نظر که میتواند CPU یا GPU باشد ارسال میکنیم.

در نهایت دو تصویر content و style را به صورت زیر میخوانیم :

```
style_img = image_loader("picasso.jpg")
content_img = image_loader("paris.jpg")
```

در قدم بعدی میخواهیم تابعی تعریف کنیم که تصاویری که به tensor تبدیل شده اند را به تصاویر PIL تبدیل کند و آنها را نمایش دهد.

برای اینکار مشابه قسمت قبل از ماژول transforms استفاده میکنیم و تابع مربوط به تبدیل تصاویر را صدا میزنیم.

```
unloader = transforms.ToPILImage()  # reconvert into PIL image
```

سپس تابع مربوط به نمایش تصاویر را به شکل زیر تعریف میکنیم :

```
def imshow(tensor, title=None):
    image = tensor.cpu().clone()  # we clone the tensor to not do changes on it
    image = image.squeeze(0)
    image = unloader(image)
    plt.imshow(image)
    if title is not None:
        plt.title(title)
    plt.pause(0.001) # pause a bit so that plots are updated
```

این تابع یک tensor ورودی میگیرد و سپس توسط تابع cpu().clone()، کاری میکنیم تا روی tensor تغییراتی اعمال نشود. سپس توسط تابع squeez(0) بعد اضافه را حذف میکنیم. در مراحل بعدی tensor را به تصویر تبدیل کرده و توسط کتابخانه matplotlib آن را نمایش میدهیم.

در نهایت دو تصویر content و style ای را که در قسمت قبل load کرده بودیم را جهت اطمینان توسط این تابع نمایش میدهیم:

```
plt.figure()
imshow(style_img, title='Style Image')

plt.figure()
imshow(content_img, title='Content Image')
```

۲-۴ توابع هزینه

همانطور که گفته شد تابع هزینه محتوا برای اندازه گیری میزان تفاوت محتوای دو تصویر محتوا و تصویر تولیدی جدید در یک لایه به کار میرود.

Feature map ها یا activation های تصویر محتوا باید توسط تابع ما شناخته شده باشد تا تابع هزینه را محاسبه کند.

برای همین تابع مربوط به تابع هزینه را بوسیله یک module از کتابخانه torch پیاده سازی میکنیم که سازنده آن $a^{[l](C)}$ را به صورت ورودی دریافت میکند. تابع هزینه هم توسط خطای Mean Squared Error محاسبه میشود که میتوان برای محسابه آن از تابع mse_loss از ماژول torch.nn.functional استفاده کرد:

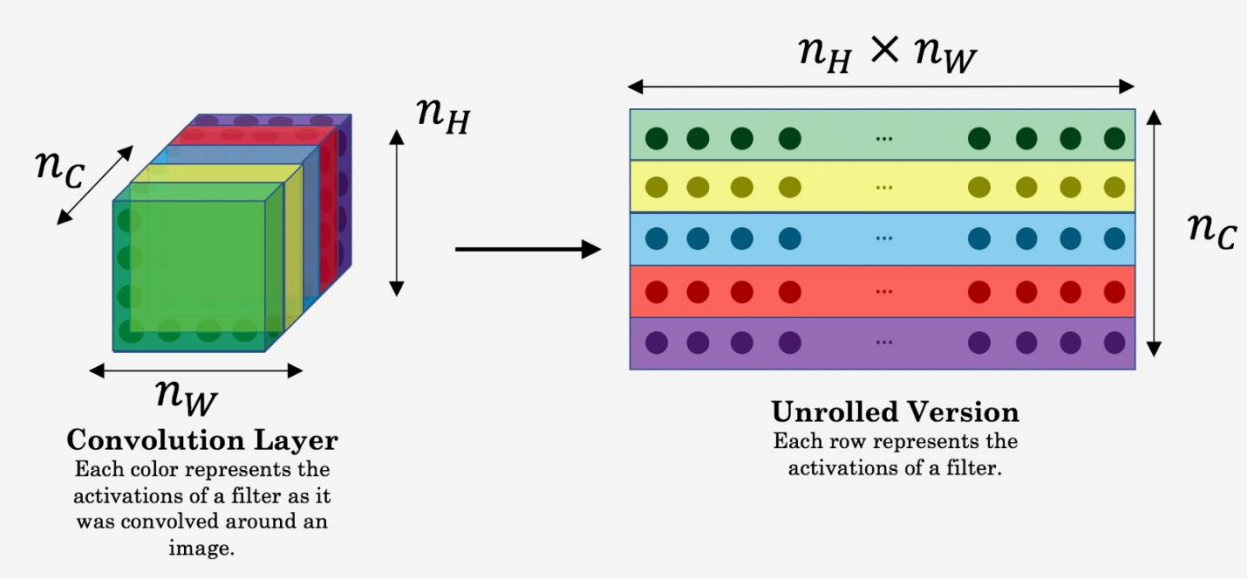
```
class ContentLoss(nn.Module):

    def __init__(self, target,):
        super(ContentLoss, self).__init__()
        self.target = target.detach()

    def forward(self, input):
        self.loss = F.mse_loss(input, self.target)
        return input
```

پس متغیر target در واقع همان $a^{[l](C)}$ میباشد. برای محاسبه تابع هزینه محتوا باید target را detach کنیم. در واقع تابع detach یک tensor جدید برمیگرداند که آن را از گراف محاسباتی جدا کرده است.

ما این ماژول ContentLoss به طور مستقیم بعد از لایه های کانولوشنی مدل قرار میدهیم تا هر بار که تصویر جدیدی وارد شبکه شد، هزینه های محتوا در لایه های مورد نظر ما حساب میشوند و به دلیل قابلیت گرادیان گیری خودکار کتابخانه PyTorch، همه گرادیان ها و مشتق ها به صورت خودکار محاسبه میشوند.

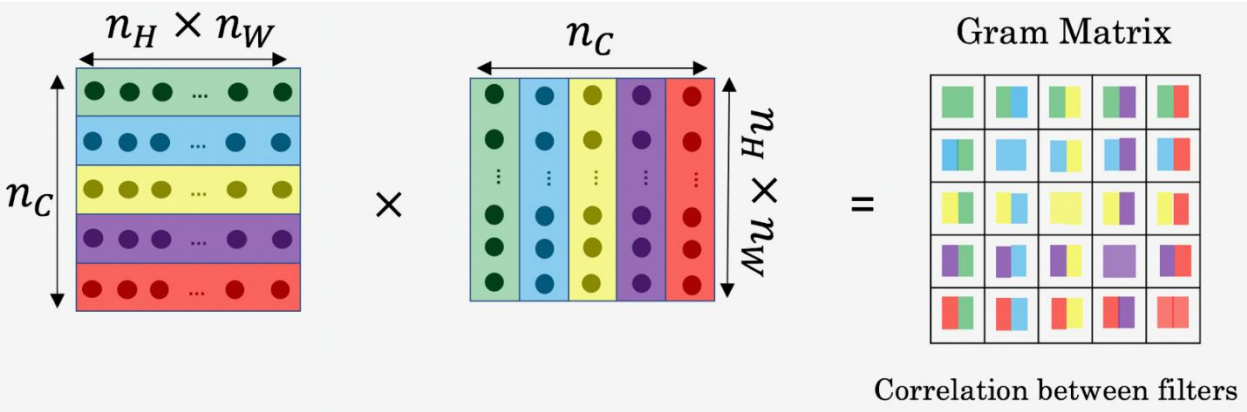
پس برای اینکه این لایه ContentLoss ما تاثیری بر روی عملکرد اصلی شبکه نگذارد، در تابع forward هزینه محتوا را حساب کرده و ورودی این لایه که متغیر input میباشد را عینا برمیگردانیم تا عملا در محتوای شبکه تغییری ایجاد نکرده باشیم و از طرفی هزینه محتوا را در متغیر loss کلاس نگه میداریم.

برای محاسبه هزینه سبک، همانطور که گفته شد نیاز به محاسبه ماتریس Style یا ماتریس Gram داریم. ماتریس Gram طبق تعریف، از ضرب یک ماتریس در ترانهاده آن بدست می آید. در اینجا ماتریس مورد نظر ما، ماتریس تغییر شکل داده شده ماتریس $G^{[l]}$ میباشد که یک ماتریس K*N خواهد بود، که K همان تعداد کانالهای تصویر یا n_c میباشد و N برابر ضرب $n_H * n_W$ میباشد. به عبارت بهتر :



شکل ۱-۲ : ماتریس تغییر شکل داده شده ماتریس $G^{[l]}$

پس ضرب ماتریس بالا در ترانهاده خودش، ماتریس Gram را برای لایه مورد نظر تولید میکند، یعنی :



شکل ۲-۲ : نحوه محاسبه ماتریس Gram

پس تابع محاسبه ماتریس Gram به شکل زیر تعریف میشود:

```
def gram_matrix(input):
    m, n_c, n_H, n_W = input.size()
    features = input.view(m * n_C, n_H * n_W)
    G = torch.mm(features, features.t())
    return G.div(m * n_C * n_H * n_W)
```

در اینجا m همان تعداد تصاویر میباشد که در این مسئله همان ۱ خواهد بود.

در این تابع ماتریس Gram باید نرمال سازی شود به صورتی هر عضو آن را تقسیم بر تعداد کل اعضای موجود در ماتریس میکنیم. دلیل آن هم این است ماتریس های که طول و عرض بزرگی دارند یا به عبارتی مقدار N بزرگی دارند باعث تولید ماتریس های Gram بزرگ میشوند که این باعث میشود لایه های اولیه قبل از لایه های pooling تاثیر زیادی هنگام بهینه سازی بگذارند. از طرفی ویژگی های استایل همانطور که قبلا توضیح داده شد در لایه های عمیق تر شبکه قرار دارند و این قضیه باعث تاثیر کم سبک و استایل بر تصویر خروجی خواهد داشت، پس این نرمال سازی تا حدی ضروری خواهد بود.

با داشتن ماتریس Gram میتوانیم کلاس مربوط به هزینه سبک را مشابه کلاس مربوط به هزینه محتوا را به شکل زیر تعریف کنیم :

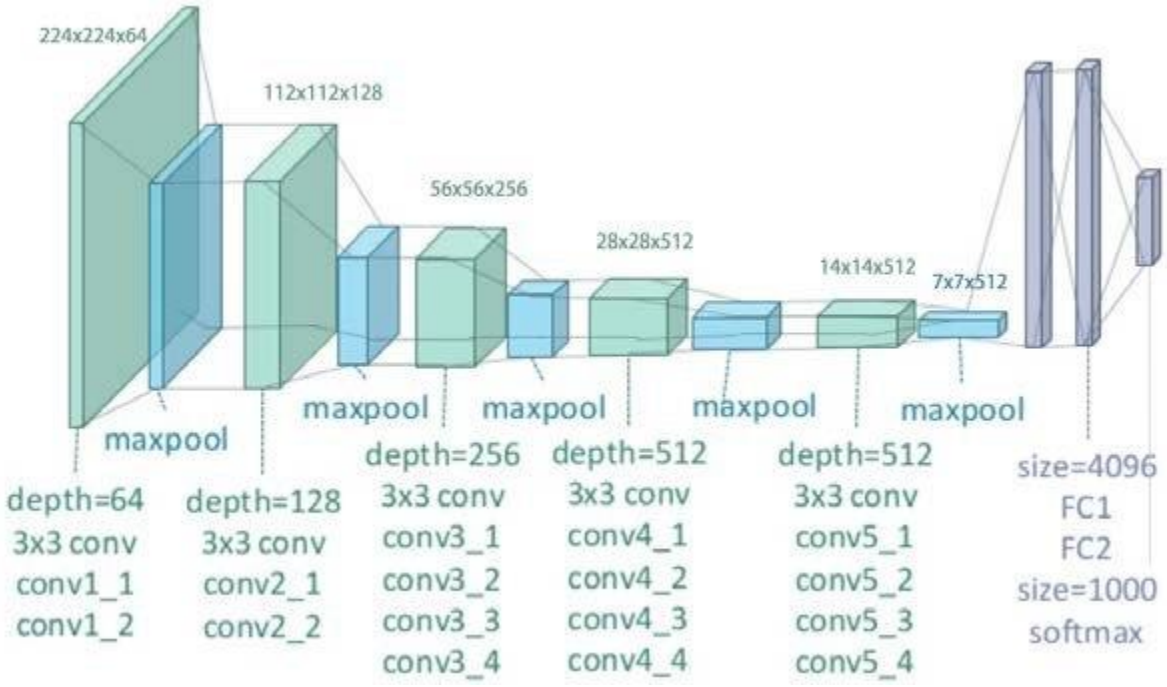
```
class StyleLoss(nn.Module):

    def __init__(self, target_feature):
        super(StyleLoss, self).__init__()
        self.target = gram_matrix(target_feature).detach()

    def forward(self, input):
        G = gram_matrix(input)
        self.loss = F.mse_loss(G, self.target)
        return input
```

مشابه لایه مربوط به هزینه محتوا، هزینه سبک مربوط لایه مورد نظر خود را بوسیله Mean Squared Error اندازه‌گیری میکنیم و ورودی این لایه که همان input میباشد را عینا خروجی میدهیم تا تاثیری بر عملکرد شبکه نگذاشته باشیم در حالی که هزینه سبک لایه مورد نظر را به شکل یک پارامتر در لایه ذخیره کرده ایم.

۲-۵ اضافه کردن مدل به کد

حالا نیاز داریم تا که یک شبکه عصبی آموزش دیده را به کد خود اضافه کنیم. مشابه مقاله، ما از مدل VGG19 که ۱۹ لایه دارد استفاده میکنیم. مدل VGG19 روی یک دیتاست بسیار بزرگ آموزش دیده است.

در تصویر زیر معماری این مدل را مشاهده میکنید:



شکل ۳-۲: معماری مدل VGG19

لایه های میانی مدل VGG19 مشابه feature detector ها عمل میکنند، برای همین میتوانیم میتوانیم از لایه های مختلف آن جهت محاسبه هزینه استفاده کنیم.

پیاده سازی ای که PyTorch از مدل VGG19 انجام داده است، از دو قسمت تشکیل میشود :

- Features : که شامل لایه های کانولوشن و pooling است.
- Classifier : که شامل لایه های fully connected میباشد.

ما در این جا از بخش features آن استفاده میکنیم، چرا که ما نیاز به خروجی لایه های کانولوشنی به صورت انفرادی داریم تا بتوانیم هزینه های محتوا و سبک را برای هر لایه محاسبه کنیم.

بعضی از لایه رفتار متفاوتی هنگام آموزش نسبت به زمان ارزیابی دارند، برای همین باید شبکه در حالت ارزیابی قرار دهیم، به کمک متد ()eval. این متد توسط PyTorch پیاده سازی شده است.

```
cnn = models.vgg19(pretrained=True).features.to(device).eval()
```

هم چنین شبکه های VGG بر روی تصاویری آموزش دیده اند که در آن ها هر کانال بوسیله مقدار میانگین و انحراف معیار زیر نرمال سازی شده است :

mean = [0.485, 0.456, 0.406]
std=[0.229, 0.224, 0.225]

پس ما هم از این مقادیر برای نرمال سازی تصویر پیش از آنکه آن را وارد شبکه کنیم استفاده میکنیم. برای اینکه یک لایه جدید به اسم Normalization به شکل زیر تعریف میکنیم تا بتوانیم آن را به راحتی وارد یک ماژول nn.Sequential بکنیم:

```
cnn_normalization_mean = torch.tensor([0.485, 0.456, 0.406]).to(device)
cnn_normalization_std = torch.tensor([0.229, 0.224, 0.225]).to(device)
class Normalization(nn.Module):
    def __init__(self, mean, std):
        super(Normalization, self).__init__()
        self.mean = torch.tensor(mean).view(-1, 1, 1)
        self.std = torch.tensor(std).view(-1, 1, 1)

    def forward(self, img):
        # normalize img
        return (img - self.mean) / self.std
```

ابتدا مقادیر گفته شده را در دو متغیر cnn_normalization_mean و cnn_normalization_std وارد میکنیم و سپس لایه مورد نظر خود را تعریف میکنیم که این ماژول میانگین و انحراف معیار را گرفته و تصویر ورودی را بوسیله این مقادیر نرمال سازی میکند و آن را برمیگرداند.

توجه کنید که بوسیله تابع view(). مقادیر tensor های میانگین و انحراف معیار را تغییر شکل میدهیم تا بتوانند با تصاویر که ۳ بعدی هستند به طور مستقیم کار کنند.

هر ماژول Sequential یک لیست مرتب از ماژول های فرزند را شامل میشود. به عنوان مثال ماژول vgg19.features شامل یک دنباله به صورت زیر میشود :

Conv2d, ReLU, MaxPool2d, Conv2d, ReLU ,...

ما باید لایه های مربوط به content loss و style loss تعریف شده را بلافاصله بعد از لایه های کانولوشن قرار دهیم.

برای همین ما باید یک ماژول Sequential جدید ایجاد کنیم که لایه های مربوط به content loss و style loss به درستی در آن قرار گرفته باشند.

```
content_layers_default = ['conv_4']
```

```
style_layers_default = ['conv_1', 'conv_2', 'conv_3', 'conv_4', 'conv_5']
```

ابتدا لایه هایی که میخواهیم در آن ها هزینه های سبک و محتوا را محاسبه کنیم مشخص میکنیم، که مطابق توضیحات قبلی برای هزینه محتوا یک لایه (لایه کانولوشن چهارم) را انتخاب کردیم و برای هزینه سبک ۵ لایه کانولوشنی ابتدا را انتخاب کرده ایم.

```
def get_style_model_and_losses(cnn, normalization_mean, normalization_std,
                               style_img, content_img,
                               content_layers=content_layers_default,
                               style_layers=style_layers_default):
    cnn = copy.deepcopy(cnn)

    # normalization module
    normalization = Normalization(normalization_mean, normalization_std).to(devic
e)

    content_losses = []
    style_losses = []

    model = nn.Sequential(normalization)

    i = 0  # increment every time we see a conv  layer
    for layer in cnn.children():
        if isinstance(layer, nn.Conv2d):
            i += 1
            name = 'conv_{}'.format(i)
        elif isinstance(layer, nn.ReLU):
            name = 'relu_{}'.format(i)
            layer = nn.ReLU(inplace=False)
        elif isinstance(layer, nn.MaxPool2d):
            name = 'pool_{}'.format(i)
        elif isinstance(layer, nn.BatchNorm2d):
            name = 'bn_{}'.format(i)
        else:
            raise RuntimeError('Unrecognized layer: {}'.format(layer.__class__.__
name__))

        model.add_module(name, layer)

        if name in content_layers:
            # add content loss:
            target = model(content_img).detach()
            content_loss = ContentLoss(target)
```

```
            model.add_module("content_loss_{}".format(i), content_loss)
            content_losses.append(content_loss)

        if name in style_layers:
            # add style loss:
            target_feature = model(style_img).detach()
            style_loss = StyleLoss(target_feature)
            model.add_module("style_loss_{}".format(i), style_loss)
            style_losses.append(style_loss)

    for i in range(len(model) - 1, -1, -1):
        if isinstance(model[i], ContentLoss) or isinstance(model[i], StyleLoss):
            break

    model = model[:(i + 1)]

    return model, style_losses, content_losses
```

کار این این تابع این است که یک ماژول Sequential جدید براساس مدل VGG19 تعریف کند و لایه های مربوط به محاسبه هزینه را به آن اضافه کند و در نهایت مدل جدید را به همراه هزینه های محتوا و سبک برگرداند.

ورودی های این تابع شامل موارد زیر میشود :

- cnn : که همان مدل ورودی میباشد که ما VGG19 انتخاب کردیم.
- normalization_mean و normalization_std : که برای نرمال سازی تصویر ورودی استفاده میشوند.
- style_img و content_img : به ترتیب تصاویر سبک و محتوا هستند.
- content_layers : لایه هایی که میخواهیم در آنها هزینه محتوا را محاسبه کنیم، مشخص میکند که مقدار پیش فرض آن را در قسمت قبل تعریف کردیم.
- style_layers : لایه هایی که میخواهیم در آنها هزینه سبک را محاسبه کنیم، مشخص میکند که مقدار پیش فرض آن را در قسمت قبل تعریف کردیم.

داخل این تابع، ابتدا یک کپی از مدل ورودی میگیریم تا مدل اصلی را تغییر ندهیم.

سپس یک ماژول برای نرمال سازی تصاویر ورودی بوسیله کلاسی که قبلا تعریف کرده بودیم، ایجاد میکنیم.

در مرحله بعد دو لیست خالی تعریف میکنیم تا به content loss ها و style loss ها به ترتیب دسترسی داشته باشیم.

سپس یک مدل به وسیله ماژول nn.Sequential ایجاد میکنیم و لایه اول آن را هم لایه نرمال سازی قرار میدهیم، تا همانطور که گفته شد بتوانیم مدل جدیدی بر اساس مدل ورودی ولی با داشتن لایه های مربوط به محاسبه هزینه ها داشته باشیم.

متغیر i که مقدار اولیه 0 دارد، داخل حلقه هر زمانی که به یک لایه کانولوشنی بر بخوریم یک واحد افزایش میابد.

یک حلقه ایجاد میکنیم که بر روی لایه های مختلف مدل اصلی میچرخد. توجه داشته باشید که تابع children() یک iterator بر روی ماژول های و لایه های مدل برمیگرداند.

داخل حلقه، اگر لایه دیده شده لایه کانولوشن بود، مقدار i را یک واحد افزایش داده و متغیر name را به وسیله string formatting طوری بوسیله i مقداری دهی میکنیم تا شماره لایه کانولوشن مورد نظر را نگه داری کند. اما اگر لایه دیده شده، لایه ReLU بود، متغیر name را براساس نام مقداردهی میکنیم و پارامتر inplace آن را False میکنیم چرا که در حالتinplace = True بخوبی با لایه های محاسبه هزینه که اضافه میکنیم کار نمی کند.

اگر هم لایه دیده شده لایه MaxPool یا لایه Batch Normalization بود، مشابه بالا متغیر name را متناسب با آن ها مقدار دهی میکنیم.

اگر هم لایه موجود هیچ کدام از لایه های بالا نبود، یک استثنا پرتاب میکنیم.

پس از مشخص شدن لایه و نام آن، آن را به مدل جدید خود بوسیله تابع add_module اضافه میکنیم.

برای اینکه لایه های هزینه خود را به مدل اضافه کنیم، از دو شرط استفاده میکنیم که اگر لایه فعلی جزو لایه هایی باشد که قرار است بلافاصله پس از آن لایه هزینه محتوا یا سبک را اضافه کنیم، ابتدا مدل را detach میکنیم تا بتوانیم لایه را اضافه کنیم و خروجی مدل تا آن مرحله را داشته باشیم، سپس خروجی مدل را به لایه هزینه محتوا یا سبک خود میدهیم و یک لایه بر همین اساس ایجاد میکنیم و در نهایت این لایه جدید محاسبه هزینه محتوا یا سبک را به مدل خود اضافه میکنیم و هم چنین شی حاصل از کلاس ContentLoss یا StyleLoss را به لیست هایی که در ابتدا تعریف کردیم اضافه میکنیم.

در مرحله بعد یک یک حلقه ایجاد میکنیم و در آن از لایه های آخر مدل به لایه های اول آن حرکت میکنیم و هرجا که لایه مورد نظر لایه محاسبه هزینه محتوا (ContentLoss) یا لایه محاسبه هزینه سبک (StyleLoss) باشد، از حلقه خارج میشویم و بر اساس اندیس حلقه که i بود مدل را تا آن قسمت نگه میداریم. به عبارت بهتر مدل را از ابتدا تا جایی که آخرین لایه هزینه محتوا یا سبک را داشته باشیم جدا میکنیم و لایه های بعد از آن را حذف میکنیم. چرا که VGG19 مدل بزرگی است و ما به همه لایه های آن احتیاجی نداریم.

در نهایت مدل جدید و لیست اشیا حاصل از لایه های هزینه و محتوا را برمیگردانیم.

در مرحله بعدی یک تصویر با مقادیر پیکسل های تصادفی که همان تصویر تولیدی ما خواهد بود ایجاد میکنیم و آن را چاپ میکنیم:

```
input_img = torch.randn(content_img.data.size(), device=device)
plt.figure()
imshow(input_img, title='Input Image')
```

۲-۶ بهینه سازی مدل

مطابق توصیه نویسنده مقاله ما هم از الگوریتم L-BFGS برای اجرای عملیات کاهش گرادیان یا Gradient Descent جهت بهینه سازی مدل استفاده میکنیم.

در اینجا بر خلاف آموزش مدل، ما میخواهیم تصویر تولیدی خود را آموزش دهیم تا هزینه آن به کمترین مقدار ممکن برسد.

```
def get_input_optimizer(input_img):
    optimizer = optim.LBFGS([input_img.requires_grad_()])
    return optimizer
```

پس تابعی تعریف میکنیم که تصویر تولیدی را میگیرد و توسط ماژول optim کتابخانه PyTorch آن تصویر را بهینه سازی میکند.

هم چنین تابع requires_grad_() را هم روی تصویر فراخوانی میکنیم تا نشان دهیم ورودی یک پارامتر است و نیاز به محاسبه گرادیان و مشتق دارد.

لازم به ذکر است منظور از input_img همان tensor حاصل از تصویر تولیدی خواهد بود.

۲-۷ جمع بندی

در نهایت میخواهیم تابعی تعریف کنیم که عمل انتقال سبک را انجام دهد.

در هر iteration ، شبکه یک تصویر به روز شده میگیرد و loss های جدید را محاسبه میکند.

متد backward از کتابخانه PyTorch را برای هر ماژول loss فراخوانی میکنیم تا به صورت پویا گرادیان آن ها توسط ماژول autograd کتابخانه PyTorch محاسبه شود.

متغیر optimizer که در تابع get_input_optimizer تعریف شد، نیاز به یک تابع closure دارد که این تابع ماژول را مجددا ارزیابی میکند و loss را برمیگرداند.

نکته حائز اهمیت این است که ممکن است شبکه تصویر ورودی را با مقادیری خارج از بازه صفر تا ۱ بهینه سازی کند، برای حل این موضوع هر بار که شبکه run و اجرا میشود مقادیر ورودی را به بازه صفر تا ۱ تبدیل میکنیم. این کار توسط تابع data.clamp_(0, 1) در کد انجام میشود.

پس تابع نهایی ما به صورت زیر خواهد بود :

```
def run_style_transfer(cnn, normalization_mean, normalization_std,
                       content_img, style_img, input_img, num_steps=300,
                       style_weight=1000000, content_weight=1):
    print('Building the style transfer model..')
    model, style_losses, content_losses = get_style_model_and_losses(cnn,
        normalization_mean, normalization_std, style_img, content_img)
    optimizer = get_input_optimizer(input_img)

    print('Optimizing..')
    run = [0]
    while run[0] <= num_steps:

        def closure():
            input_img.data.clamp_(0, 1)

            optimizer.zero_grad()
            model(input_img)
            style_score = 0
            content_score = 0

            for sl in style_losses:
                style_score += sl.loss
            for cl in content_losses:
                content_score += cl.loss

            style_score *= style_weight
            content_score *= content_weight

            loss = style_score + content_score
            loss.backward()

            run[0] += 1
            if run[0] % 50 == 0:
                print("run {}:".format(run))
                print('Style Loss : {:4f} Content Loss: {:4f}'.format(
                    style_score.item(), content_score.item()))
                print()
```

```
            return style_score + content_score

        optimizer.step(closure)

    input_img.data.clamp_(0, 1)

    return input_img
```

ورودی های این تابع شامل موارد زیر میشود :

- cnn : که همان مدل ورودی میباشد که ما VGG19 انتخاب کردیم.
- normalization_mean و normalization_std : که برای نرمال سازی تصویر ورودی استفاده میشوند.
- style_img و content_img : به ترتیب تصاویر سبک و محتوا هستند.
- input_img : تصویر ورودی ما که با مقادیر تصادفی ایجاد کردیم، میباشد.
- num_steps : مشخص کننده تعداد دفعات اجرای شبکه میباشد که مقدار پیش فرض آن ۳۰۰ بار است.
- style_weight : وزن مربوط به هزینه تابع سبک است که در فرمولی که در بخش های قبلی معرفی کردیم، همان hyperparameter ای است که با نام β در تابع هزینه کلی قرار داده بودیم و مقدار پیش فرض آن یک میلیون میباشد.
- content_weight : وزن مربوط به هزینه تابع محتوا است که در فرمولی که در بخش های قبلی معرفی کردیم، همان hyperparameter ای است که با نام α در تابع هزینه کلی قرار داده بودیم و مقدار پیش فرض آن یک میباشد.

از ورودی های این تابع، مقادیر cnn, normalization_mean, normalization_std, style_img, content_img را به تابع get_style_model_and_losses میدهیم و خروجی های آن را که شامل model, style_losses, content_losses میباشد را میگیرم.
بهینه ساز خود را توسط تابع get_input_optimizer مقداردهی میکنیم.

سپس از یک حلقه استفاده میکنیم که به تعداد مورد نظر ما میچرخد و در آن شبکه اجرا میشود و هزینه ها محاسبه شده و بهینه سازی انجام میشود.
همانطور که گفته شد درون حلقه تابع closure را تعریف میکنیم که در آن ابتدا ورودی به بازه صفر تا ۱ تبدیل میشود. سپس توسط تابع zero_grad() گرادیان ها را صفر میکنیم تا بتوانیم backpropagation را انجام دهیم.

علت آن این است که PyTorch گرادیان های محاسبه شده را بر روی عملیات های backward بعدی تجمیع میکند، برای همین باید برای هر دور اجرای شبکه ابتدا گرادیان ها را صفر کنیم تا مجددا محاسبه شوند.
در قسمت بعد تصویر ورودی را به مدل میدهیم. سپس دو متغیر برای محاسبه هزینه های سبک و محتوا با مقدار اولیه صفر میسازیم. با دادن تصویر ورودی به مدل، همانطور که در قسمت های قبل مشاهده کردید، تصویر ورودی از لایه های مختلف عبور میکند و در لایه هایی که قرار بود هزینه ها را محاسبه کنند مقدار پارامتر loss آن ها مقدار دهی میشد، پس در دو حلقه مقادیر loss مربوط به سبک و محتوا را جمع میکنیم و سپس هزینه بدست آمده برای سبک و محتوا را در وزن متناسب آنها ضرب میکنیم و در نهایت هزینه کلی که جمع این دو مقدار میباشد، بدست می آید و این تابع این مقدار را خروجی میدهد.
در نهایت تابع ()backward بر روی این هزینه بدست آمده صدا میزنیم تا عملیات backpropagation انجام شود.
در یک حلقه هم هر ۵۰ بار چرخش حلقه، مقادیر هزینه سبک و محتوا را چاپ میکنیم.
این تابع closure را هم به بهینه ساز خود پاس میدهیم تا بهینه ساز تلاش کند تا مقدار loss برگردانده شده را توسط الگوریتم L-BFGS در هر مرحله بهینه کند و کاهش دهد.
در نهایت هم مجدد تصویر ورودی را پس از بهینه سازی های انجام شده به بازه صفر تا ۱ تبدیل میکنیم و آن را به عنوان خروجی تابع برمیگردانیم.

۸-۲ اجرای مدل و گرفتن خروجی

در نهایت با تمامی تعاریف انجام شده، مدل را اجرا میکنیم و تصویر حاصل را چاپ میکنیم:

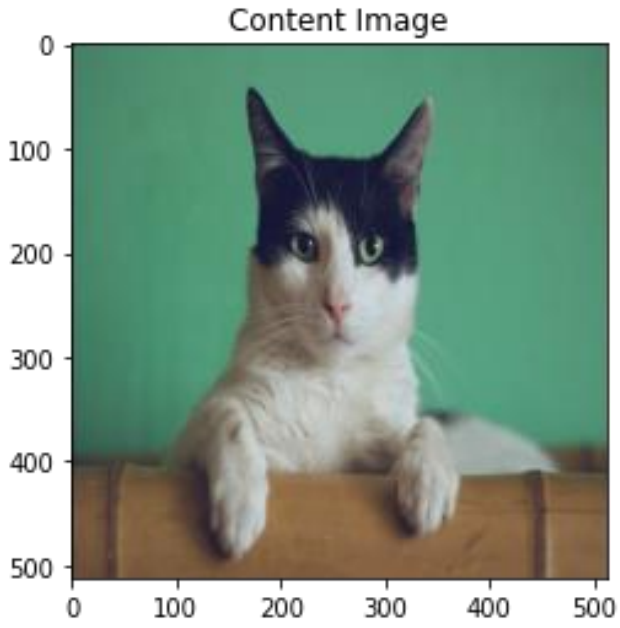
```
output = run_style_transfer(cnn, cnn_normalization_mean, cnn_normalization_std,
                            content_img, style_img, input_img)

plt.figure()
imshow(output, title='Output Image')
plt.ioff()
plt.show()
```

۳ فصل سوم: مشاهده نتایج

۳-۱ نمونه اجرای الگوریتم

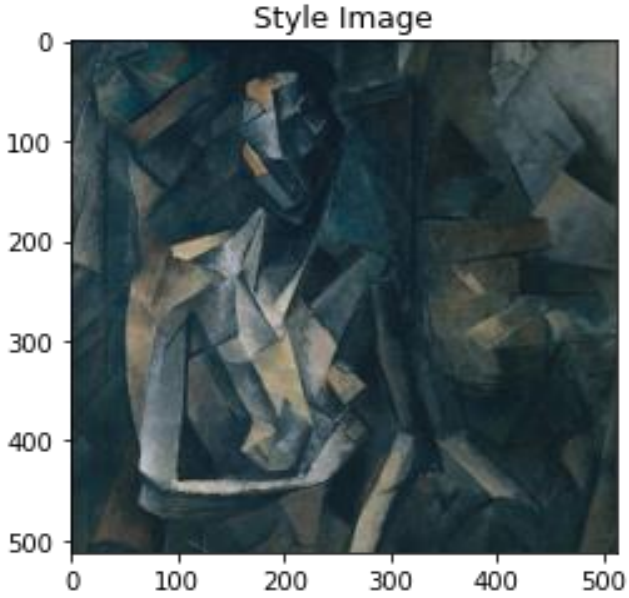
پس از پیاده سازی پروژه، حال میخواهیم عملکرد آن را بسنجیم.

برای اینکار میخواهیم روی تصویر محتوا زیر را به وسیله استایل های مختلف، الگوریتم انتقال سبک را اجرا نماییم:



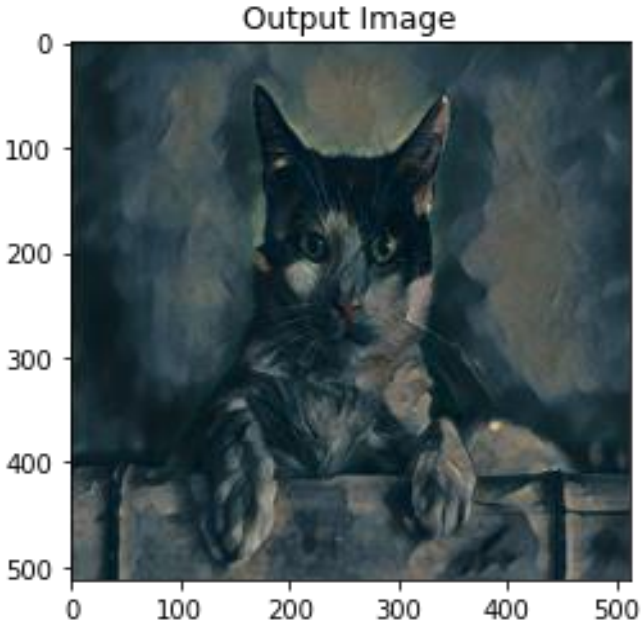
شکل ۳-۱ : تصویر محتوا جهت امتحان عملکرد الگوریتم

ابتدا تصویر زیر که یک نقاشی از پیکاسو میباشد را به عنوان تصویر سبک انتخاب میکنیم :



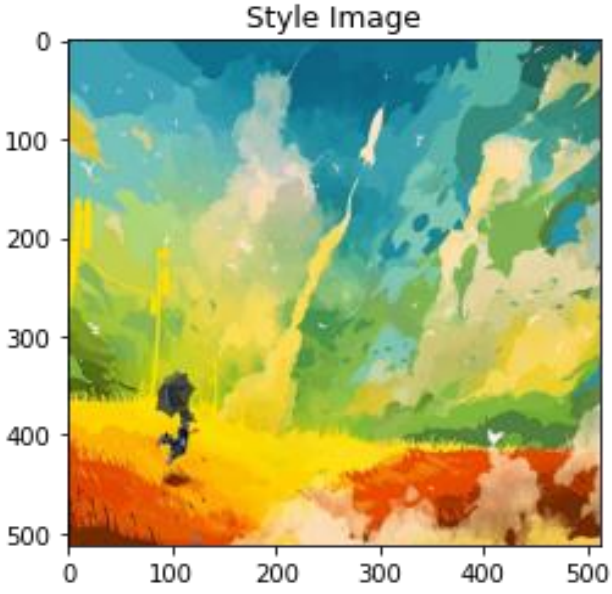
شکل ۳-۲ : تصویر سبک اول

نتیجه به صورت تصویر زیر میشود :



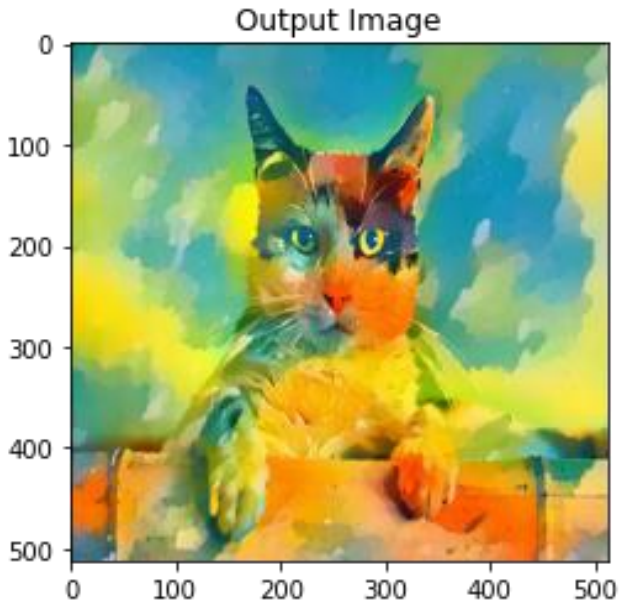
شکل ۳-۳ : خروجی اول

برا ی تصویر بعدی، از تصویر سبک زیر استفاده میکنیم:



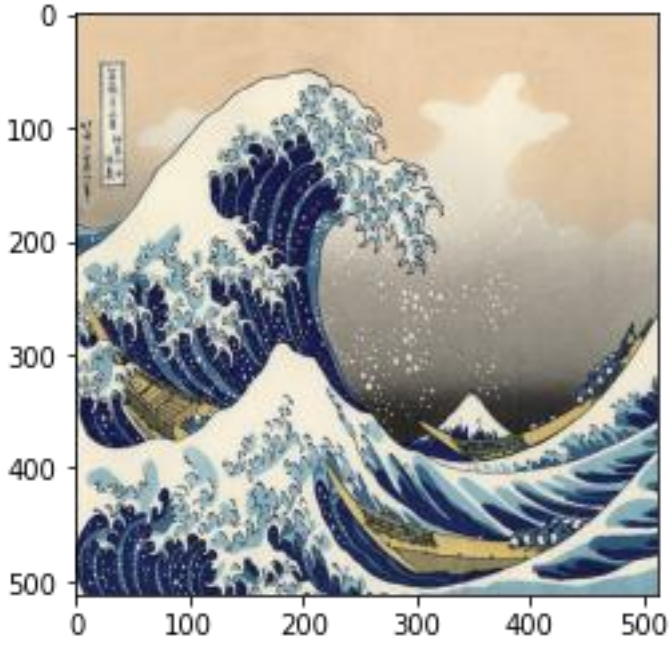
شکل ۴-۳ : تصویر سبک دوم

که تصویر حاصل به صورت تصویر زیر نمایش داده میشود :



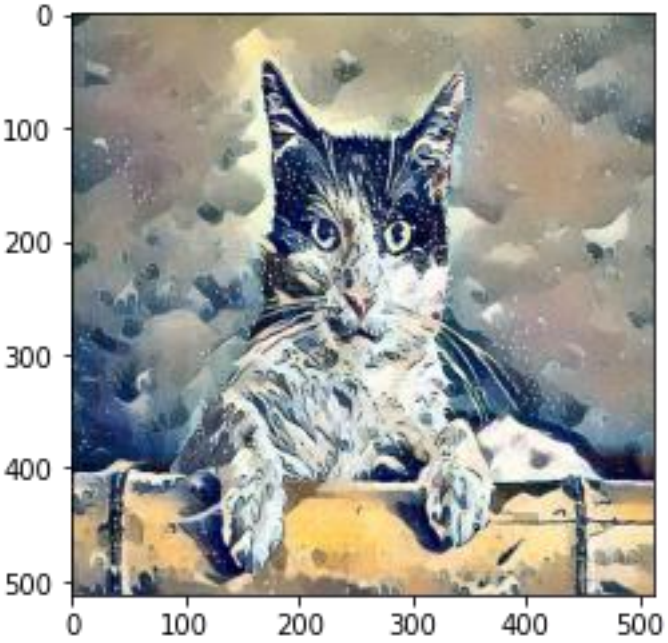
شکل ۵-۳: خروجی دوم

و در حالت بعدی از تصویر سبک زیر استفاده میکنیم :



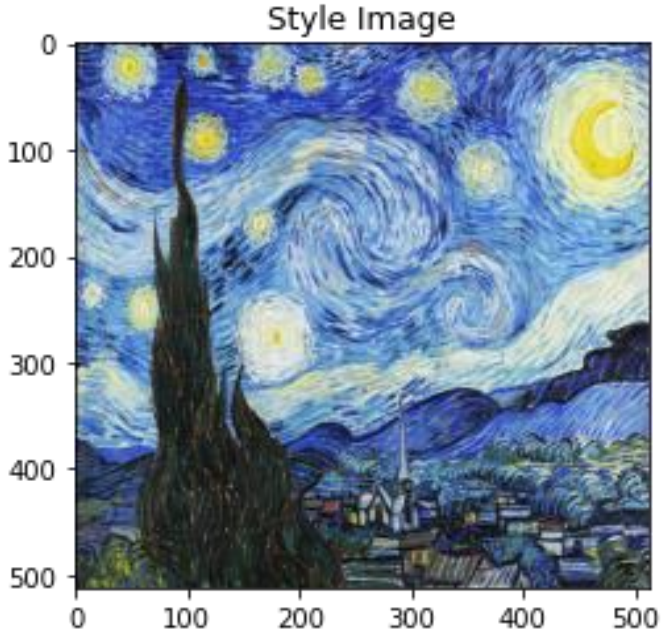
شکل ۶-۳: تصویر سبک سوم

که حاصل به صورت تصویر زیر میشود:



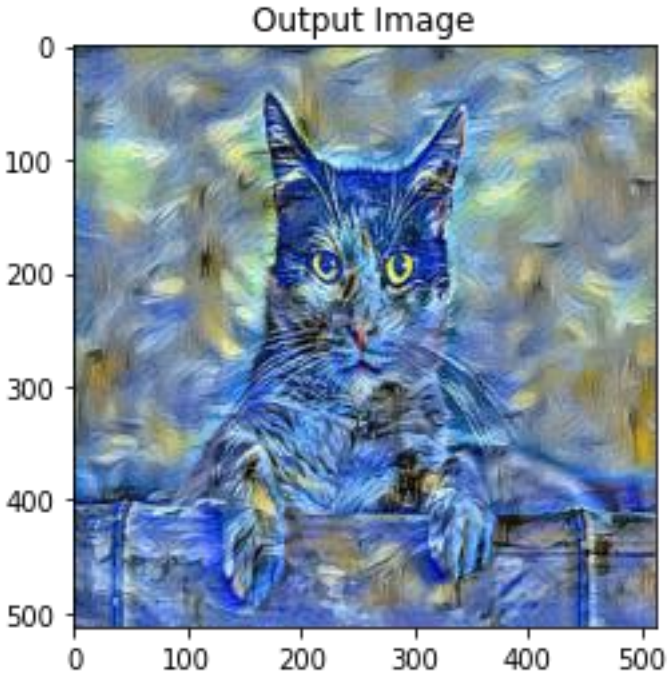
شکل ۷-۳: خروجی سوم

به عنوان مثال آخر از تصویر سبک زیرکه اثر ون گوگ میباشد استفاده میکنیم:



شکل ۸-۳: تصویر سبک چهارم

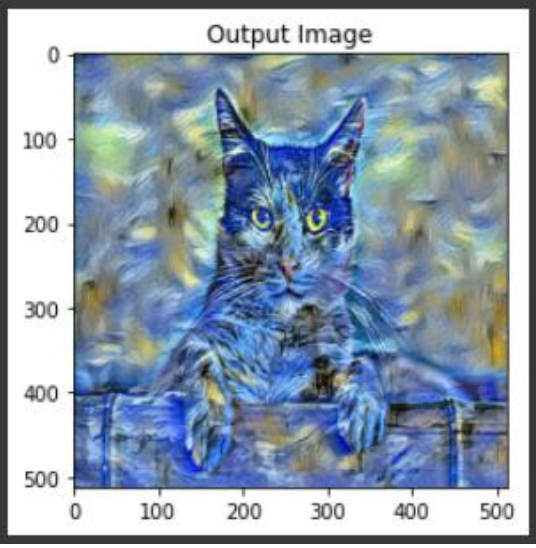
و در نهایت خروجی به صورت زیر میشود:



شکل ۹-۳: خروجی چهارم

در تصویر زیر نیز خروجی الگوریتم برای یکی از حالات فوق مشاهده میکنید :

```
Optimizing..
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12:
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:13:
  del sys.path[0]
run [50]:
Style Loss : 21.455296 Content Loss: 377.903198

run [100]:
Style Loss : 554.633301 Content Loss: 350.874878

run [150]:
Style Loss : 24.604717 Content Loss: 343.825714

run [200]:
Style Loss : 23.430822 Content Loss: 355.744629

run [250]:
Style Loss : 18.733425 Content Loss: 334.309998

run [300]:
Style Loss : 17.267527 Content Loss: 329.112427
```



شکل ۱۰-۳: نمونه خروجی کد الگوریتم

۳-۲ ارزیابی

با توجه به ماهیت مسئله، برای ارزیابی الگوریتم و پروژه خود از شیوه ارزیابی کیفی استفاده کردیم. به این صورت که ابتدا تصاویر مختلفی را به عنوان تصویر محتوا با هر تصویر سبک به عنوان ورودی دادیم و سپس این خروجی های بدست آمده را در قالب یک نظر سنجی به افراد مختلف نشان دادیم و از آنها خواستیم تا به هر تصویر از لحاظ هنری بودن یک امتیاز از ۰ تا ۱۰ بدهند. لازم بذکر است خود تصاویر سبک را هم در این بین قرار دادیم تا توسط مخاطب به لحاظ هنری بودن امتیاز به آن داده شود. پس از انجام نظر سنجی میانگین امتیازاتی که فقط تصاویر سبک اصلی گرفته بودند را محاسبه کردیم، هم چنین میانگین امتیازات تصاویر خروجی الگوریتم را هم محاسبه کردیم، در نهایت این دو عدد را بر هم تقسیم کردیم تا متوجه شویم امتیازات تصاویر تولیدی چه مقدار به امتیاز تصاویر سبک اصلی نزدیک هستند و به این وسیله معیاری برای سنجش عملکرد الگوریتم خود بدست آوردیم. این مراحل را در جدول زیر مشاهده میفرمایید:

جدول ۱-۳: جدول ارزیابی نتایج

نوع تصویر سبک	میانگین امتیاز تصویر سبک	میانگین امتیاز تصاویر حاصل از تصویر سبک	میانگین امتیاز کل تصاویر
نقاشی پیکاسو (شکل ۲-۳)	۷.۴۴	۷.۱۵	۷.۲
نقاشی ون گوگ (شکل ۸-۳)	۷.۸۸	۶.۰۲	۶.۳۳
نقاشی آبرنگ (شکل ۴-۳)	۸.۱۱	۷.۰۶	۷.۲۴
نقاشی موج دریا (شکل ۶-۳)	۸.۷۷	۷.۰۴	۷.۳۳

با توجه به این اطلاعات، نتیجه زیر حاصل میشود:

میانگین امتیاز تصاویر سبک : ۸.۰۵

میانگین امتیاز تصاویر حاصل از تصاویر سبک : ۶۸۲

میانگین امتیاز کل تصاویر: ۷.۰۲

بنابراین اگر " میانگین امتیاز تصاویر حاصل از تصاویر سبک" که نتیجه عملکرد الگوریتم را مشخص میکند، بر "میانگین امتیاز تصاویر سبک" امتیاز تصاویر اصلی سبک را مشخص میکند، تقسیم کنیم، درصدی تقریبی از نحوه عملکرد الگوریتم ما بدست می‌آید. یعنی:

$$\text{درصد عملکرد الگوریتم} : \frac{\text{میانگین امتیاز تصاویر حاصل از تصاویر سبک}}{\text{میانگین امتیاز تصاویر سبک}} = ۰.۸۴۶۸$$

میتوان به صورت تقریبی گفت، تصویر انتقال سبک داده شده الگوریتم ما نزدیک به ۸۴ درصد به عنوان یک تصویر هنری شناخته میشود.

٤ منابع

[1] Leon A. Gatys, Alexander S. Ecker, Matthias Bethge, "A Neural Algorithm of Artistic Style," 2016.

[2] Andrew Ng, "Coursera: Deep Learning Specialization," 2018. [Online].

Bu-Ali Sina University
Engineering Department
Computer Engineering

B.S.c Thesis

Neural Style Transfer

Supervisor:
Dr. Muharram Mansoorizade

Amirhossein Khodabandehlou

March, 2021