به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

شبکه های عصبی و یادگیری عمیق

استاد

دکتر احمدکلهر

مینی پروژه سوم

نام و نام خانوادگی	شقایق طرب خواه - سحر رجبی
شماره دانشجویی	۸۱۰۶۹۸۱۹۳ – ۸۱۰۱۹۹۱۶۵
تاریخ ارسال گزارش	۱۴۰۰/۰۴/۳۰

فهرست گزارش سوالات

سوال ۱ – Variational Autoencoder

در این سوال به دنبال بررسی عملکرد الگوریتم Variational Autoencoder می باشیم. به این منظور ابدا به بررسی ساختار و عملکرد برخی از بخش های این روش پرداخته و سپس بر روی داده های **MNIST** پیاده سازی می کنیم.

الف) شبکه **VAE** را بر روی داده های **MNIST** پیاده سازی کنید. برای ساختار های **decoder** و **encoder** نیازی نیست از معماری خاصی پیروی کنید. تنها شرطی که در معماری شبکه باید رعایت کرد آن است که بعد فضای ثانویه باید ۲ باشد.

ب) از چه تابع هزینه ای استفاده کرده اید؟ لزوم استفاده از تابع هزینه **KL divergence** را توضیح دهید.

ج) نقش **reparameterization trick** را در تشکیل گراف محاسباتی و اعمال **back propagation** توضیح دهید.

د) مجموعه داده های **MNIST** را با استفاده از شبکه **VAE**ای که آموزش داده اید به فضای ثانویه انتقال دهید و **Scatter plot** آنها را رسم کنید.

ه) پس از آنکه داده ها را به فضای ثانویه انتقال دادید حدود پراکندگی داده ها را به دست آورید و خروجی بخش **decoder** را در یک گرید نشان دهید.

و) برای بررسی کیفیت تعدادی از داده های مسئله به عنوان خروجی شبکه را نمایش دهید.

ز) شبکه **conditional VAE** را بر روی داده های **MNIST** پیاده سازی کرده ومشروط سازی بخش **decoder** و **encoder** را انجام دهید.

ح) با انتقال داده ها به فضای ثانویه در این بخش نیز **Scatter plot** آنها را رسم کنید.

ط) به دلخواه یکی از برچسب ها را انتخاب کرده و خروجی بخش **decoder** را در یک گرید نشان دهید.

ی) برای بررسی کیفیت تعدادی از داده های مسئله به عنوان خروجی شبکه را نمایش دهید.

پاسخ بخش الف :

در بخش اول مطابق خواسته سوال شبکه variational Auto encoder بر روی داده های MNIST پیاده سازی شده است. به این ترتیب در ادامه نتایج مشاهده می شود.

پاسخ بخش ب :

از دو تابع هزینه مورد کاربرد در VAE می‌توان به Mean-Squared-Error و Cross-entropy در بیان اختلاف بین داده‌های جدید و ترین تحت عنوان Reconstruction-loss اشاره کرد. علاوه بر تابع هزینه بالا، یک تابع هزینه دیگر برای مقایسه میانگین و انحراف معیار داده‌های اصلی با میانگین و انحراف معیار تولیدی از تابع چگالی احتمال به نام Kullback-leibler divergence تعریف خواهد گردید. استفاده از تابع خطای اول به صورت مستقل منجر به بازیابی اصل ورودی بدون تغییر خواهد شد در صورتی که ویژگی‌های اصلی در فرایند انکودینگ حذف نشده باشند از طرفی استفاده تنها از تابع هزینه دوم منجر به تولید داده‌ها در فضای نهان ویژگی‌ها بدون هیچ محدودیتی خواهد شد و از آنجا که تمام دامنه مربوط به کلاس خاصی نمی‌باشد، خروجی تولیدی شانس کمتری برای قرارگیری در کلاس خاصی پیدا خواهد کرد بنابراین استفاده همزمان از دو تابع هزینه در کنار هم باعث تولید داده‌های واقعی اطراف میانگین داده‌های ترین و با پراکندگی خاصی متناسب با انحراف معیار ترین شده اطراف آن خواهد شد.

استفاده از تابع خطای Reconstruction-loss در خوشه بندی کلاس‌ها نقش خواهد داشت و تابع هزینه Kullback-leibler divergence در تولید نمونه‌های تصادفی چگال اطراف مرکز خوشه‌ها به کار گرفته خواهد شد و همچنین در این روش VAE سعی در نزدیکی کلاس‌ها به یکدیگر است تا بتوان با درون‌یابی بین کلاس‌ها، تولید داده جدید با خلاقیت نسبتا خوب دست یافت. از این تابع هزینه برای بیان میزان واگرایی بین هر کلاس استفاده خواهد شد که با کاهش این تابع هزینه پارامترهای میانگین و انحراف معیار لرن خواهند شد. این تابع در زیر آورده شده است.

$$KL\ loss = \sum_{i=1}^{n} \sigma_i^2 + \mu_i^2 - \log(\sigma_i) - 1$$

پاسخ بخش ج :

هسته اصلی روابط مورد استفاده در روش VAE در زیر نشان داده شده است:

$$\log P(X) - \mathcal{D}\left[Q(z|X)\|P(z|X)\right] = E_{z\sim Q}\left[\log P(X|z)\right] - \mathcal{D}\left[Q(z|X)\|P(z)\right]$$

هدف اصلی در پیشبرد این الگوریتم بیشینه کردن سمت راست معادله و بهینه کردن سمت چپ معادله با روش stochastic gradient descent می باشد. بخشی از این رابطه که می توان از با روش های هوشمندانه ای مقدار آن را محاسبه کرد جمله اول سمت راست می باشد. در هنگام پیاده سازی این روش، برای رسیدن به پیشبینی نسبتا دقیقی از جمله $E_{z\sim Q}[logP(X/z)]$ لازم است تا تعداد زیادی از داده های z از تابع f در داخل جمله مربوط به امید ریاضی عبور کند. اما این عمل نیاز به صرف هزینه های محاسباتی زیادی دارد که بهره گیری از آن معقول نمی باشد. حال اگر برای رسیدن به مقدار این امید ریاضی از روش stochastic gradient descent استفاده کنیم که در بالا به آن اشاره شد، به این ترتیب در هر بار محاسبه مقدار امید ریاضی تنها یک داده از z گرفته می شود که مقدار به دست آمده از آن داده برابر با مقدار امید ریاضی آن خواهد بود. به این ترتیب با استفاده از این روش و هدفی که در بالا گفته شده به دنبال بهینه کردن رابطه زیر هستیم:

$$E_{X\sim D}\left[\log P(X) - \mathcal{D}\left[Q(z|X)\|P(z|X)\right]\right] = E_{X\sim D}\left[E_{z\sim Q}\left[\log P(X|z)\right] - \mathcal{D}\left[Q(z|X)\|P(z)\right]\right]$$

که اگر از این رابطه گرادیان بگیریم، اپراتور گرادیان از امید ریاضی عبور کرده و تنها نیاز است که گرادیان عبارت داخل امید را محاسبه کنیم و پس از میانگین گیری بر روی تک تک داده های z به مقدار اصلی گرادیان نزدیک می شود که نتیجه علاوه بر پارامتر های P به پارامتر های Q نیز وابسته است. اما مسئله اصلی آن است که با توجه به این که در این الگوریتم از back propagation استفاده می شود نیاز است تا خطا به لایه های قبلی نیز منتقل شود. روش stochastic gradient descent تنها قادر است تا ورودی ها را به صورت تک به تک مدیریت کند و نه واحد های دردونی شبکه را .برای حل این مشکل از reparametrization trick استفاده شده است. با به کارگیری این روش، عمل داده گیری به یکی از لایه های ورودی منتقل می شود. به این ترتیب با $\mu(x)$ و $\Sigma(x)$ داده شده عمل داده برداری با استفاده از تابع نرمال $N(\mu(x), \Sigma(x))$صورت می گیرد که با داده گیری اولیه $\varepsilon \sim N(0, I)$انجام می شود و سپس z به صورت زیر محاسبه می شود:

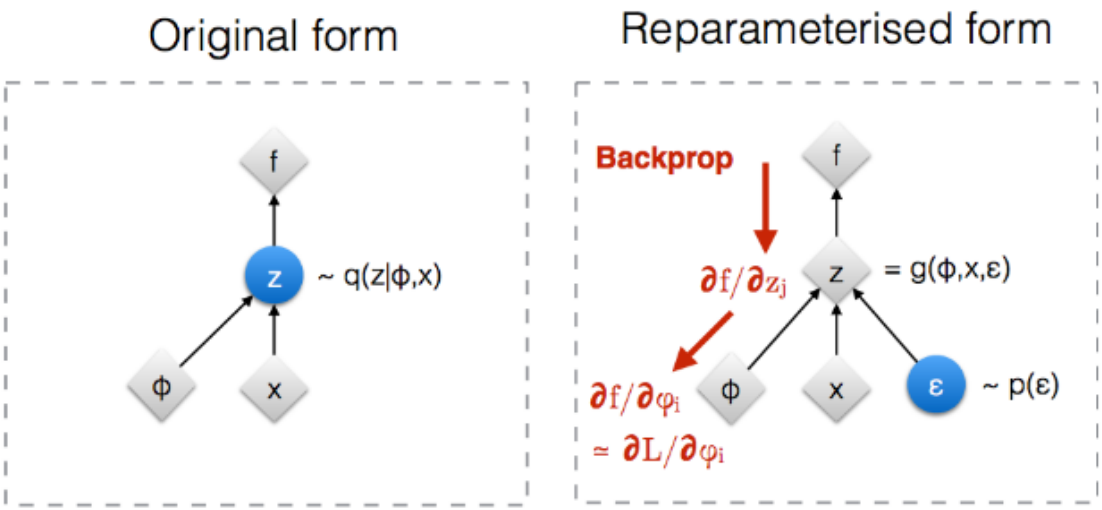
$$z = \mu(x) + \Sigma^2(x) * \varepsilon$$

با این جایگذاری در واقع تابعی که می خواهیم از آن گرادیان بگیریم به صورت زیر خواهد بود:

$$E_{X\sim D}\left[E_{\epsilon\sim\mathcal{N}(0,I)}[\log P(X|z=\mu(X)+\Sigma^{1/2}(X)*\epsilon)]-\mathcal{D}\left[Q(z|X)\|P(z)\right]\right]$$

در این معادله هیچ‌یک از امیدهای ریاضی وابسته به توزیع‌هایی که به مدل ما بستگی دارند نخواهد بود و الگوریتم stochastic gradient descent می‌تواند طبق توضیحات ارائه‌شده به کار گرفته شود و backpropagationاجرا شود.

در واقع مشکل از اینجا ناشی می‌شود که نودی که مسئولیت sampling را دارد در داخل شبکه‌ی ما قرار دارد و back propagation نمی‌تواند برای یک نودی که عملکرد رندم دارد اجرا شود.
راه ارائه شده در این بخش، به اختصار به این صورت بود: یک پارامتر اپسیلون به مدل ما اضافه می‌شود و در واقع sampling ما در لایه‌ی ورودی به کمک این پارامتر انجام می‌شود. به این صورت می‌توانیم سمپل‌های رندم را با کمک این پارامتر بدست‌بیاوریم اما در عین حال نودی با عملکرد رندم در شبکه نداشته‌باشیم تا backpropagation به مشکل نخورد. در واقع در تصویر زیر می‌تواند این راه را بهتر نشان بدهند.

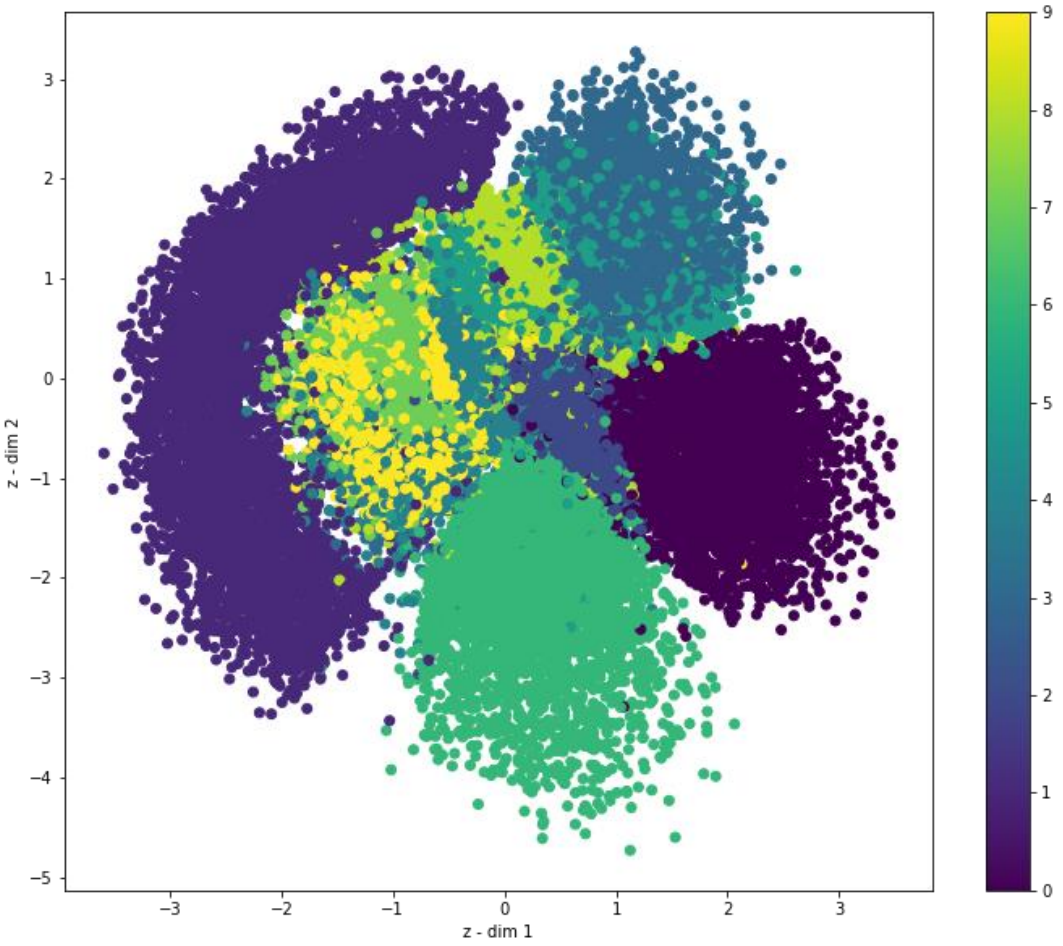


شکل ۱-۱: مقایسه ساختار استفاده از reparameterizatio trick و بدون استفاده از این روش

در تصویر سمت چپ، نود z یک نود رندم است که برای الگوریتم ما مشکل ساز است. ما با تعبیه‌ی یک ورودی رندم برای این نود، کاری می‌کنیم که این نود از حالت رندم خارج شده و سمپلینگ خارج از آن صورت بگیرد (به کمک تابعی که مقدار اپسیلون در آن تاثیرگذار است) و به این ترتیب مشکل را حل می‌کنیم.

پاسخ بخش د :

در این بخش خواسته شده است تا scatter plot داده ها را در فضای ثانویه رسم کنیم تا نحوه توزیع داده ها در استفاده از این روش مشخص گردد که این تصویر در ادامه قابل مشاهده است:

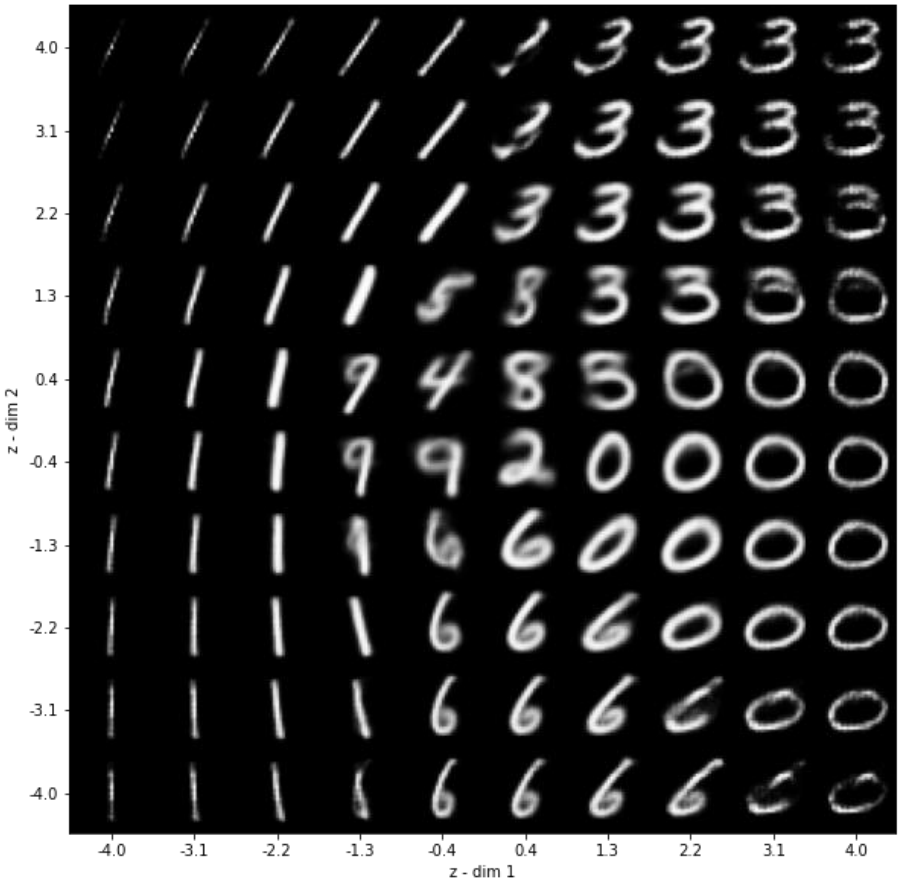


شکل ۱-۲: نمودار Scatter plot داده ها در فضای ثانویه برای روش VAE

با توجه به شکل بالا و تابع هزینه معرفی شده در قسمت ب، کلاسترها در فضای نهان علاوه بر اینکه فضای خاصی از فضای نهان را در برمی‌گیرند به یکدیگر نیز نزدیک شده‌اند که به علت تابع خطای KL می‌باشد. از طرفی ارقامی که در تولید شبیه به یکدیگر هستند در این کلاستر بندی‌های دارای خوشه‌هایی با اشتراک بیشتر هستند و این باعث تولید اعدادی با خلاقیت بالا در این فضا با اشتراک های بیشتر خواهد شد. به خوبی مشاهده می شود که با تبدیل به فضای جدید و با استفاده از دو ویژگی در نظر گرفته شده که یکی بیانگر خود عدد مد نظر و دیگری حاوی اطلاعاتی در رابطه با style عدد فوق می باشد، به نحوی قرار گرفته اند که تا حد مناسبی از یکدیگر جداپذیر باشند که بدین ترتیب مدل توانایی تولید داده های مناسب را خواهد داشت و خروجی ها نزدیک به ورودی ها می باشند.

پاسخ بخش ه :

در این بخش نیز خواسته شده است تا پس از انتقال به فضای ثانویه و یا به عبارتی دیگر پس از عبور از بخش decoder نمونه ای از داده ها را بازسازی کرده و نمایش دهیم. که این تصویر تولیدی در شکل زیر قابل مشاهده می باشد:



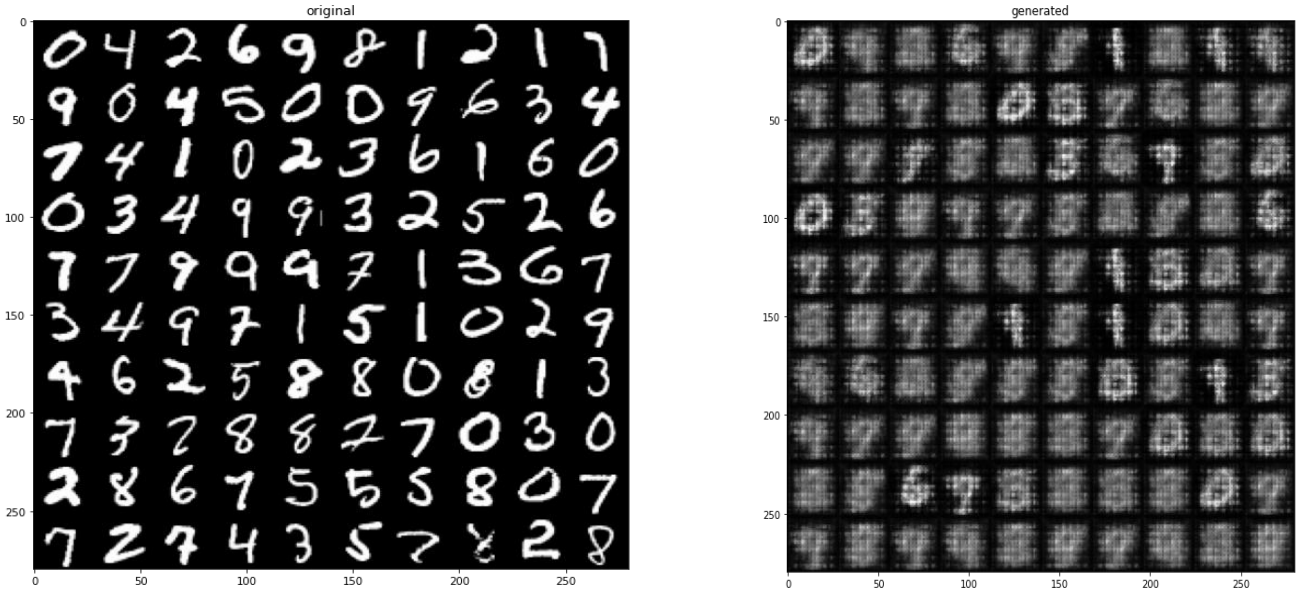
شکل ۱-۳: نمودار تصویر خروجی از بخش decoder با استفاده از روش VAE

مطابق با آنچه که در شکل بالا مشاهده می شود اعداد موجود در تصویر خروجی تا حدی قابل شناسایی از یکدیگر می باشند، به طور مثال در این شکل اعداد ۰، ۱، ۳، ۵، ۶، ۸ و ۹ به صورت چشمی به راحتی از یکدیگر قابل تشخیص می باشند هرچند که عدد ۱ به طور ویژه در زوایای مختلف قرار گرفته است که ممکن است شبکه را با مشکل مواجه سازد اما با وجود کیفیت و وضوح پایین مشکلی در رابطه با تشخیص اعداد مختلف از یکدیگر وجود ندارد.

پاسخ بخش و :

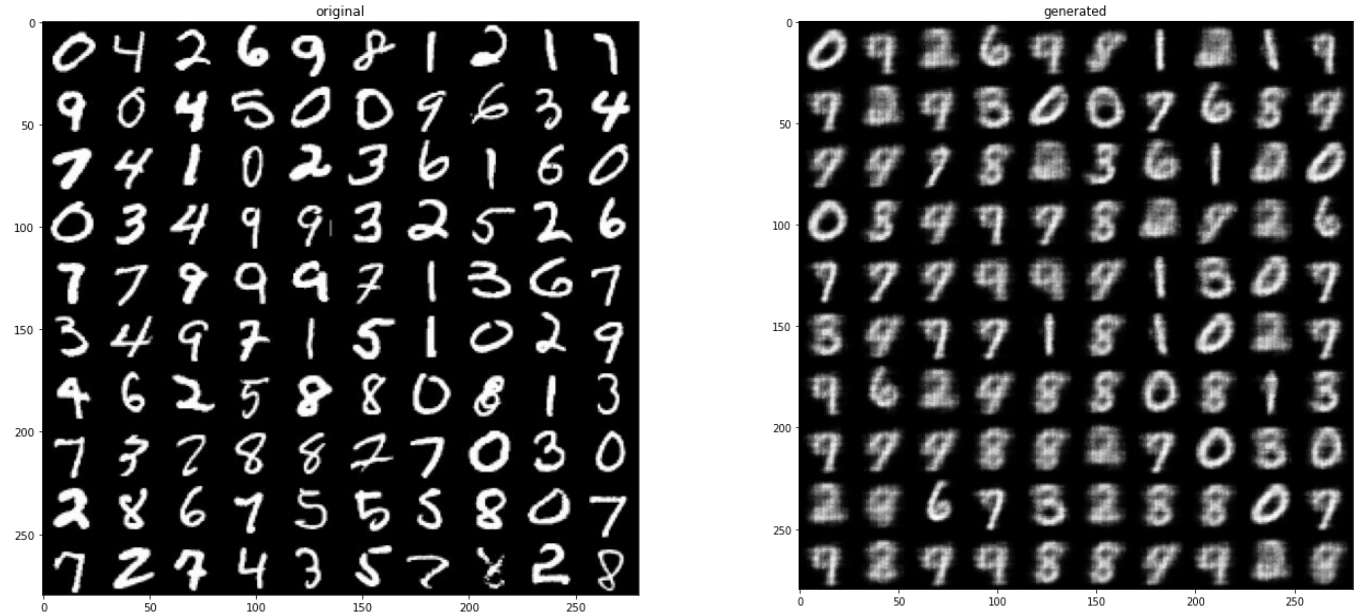
در این قسمت از سوال به دنبال آن هستیم تا نتایج حاصل از مدل را مشاهده کنیم به این منظور نتایج حاصل را در طی چند ایپاک نشان می دهیم:

خروجی از ایپاک ۲ :



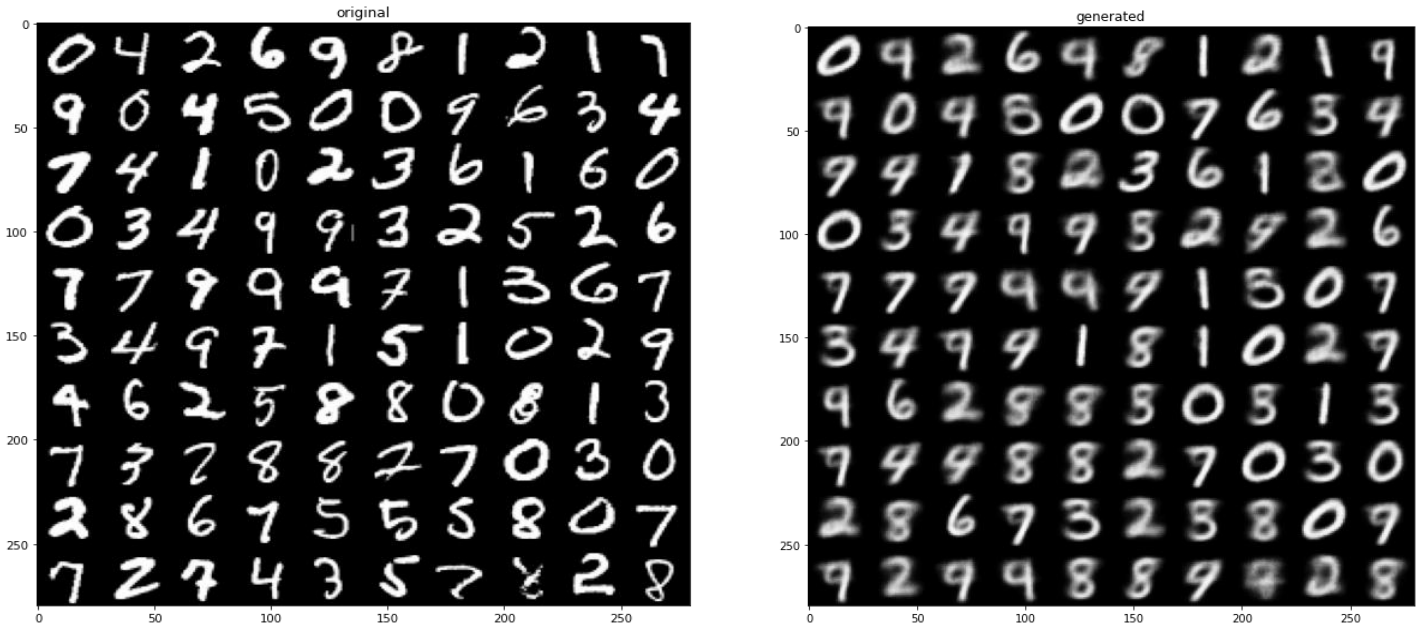
شکل ۱-۴: تصاویر اصلی داده شده و خروجی در ایپاک ۲ با روش VAE

خروجی از ایپاک ۳ :



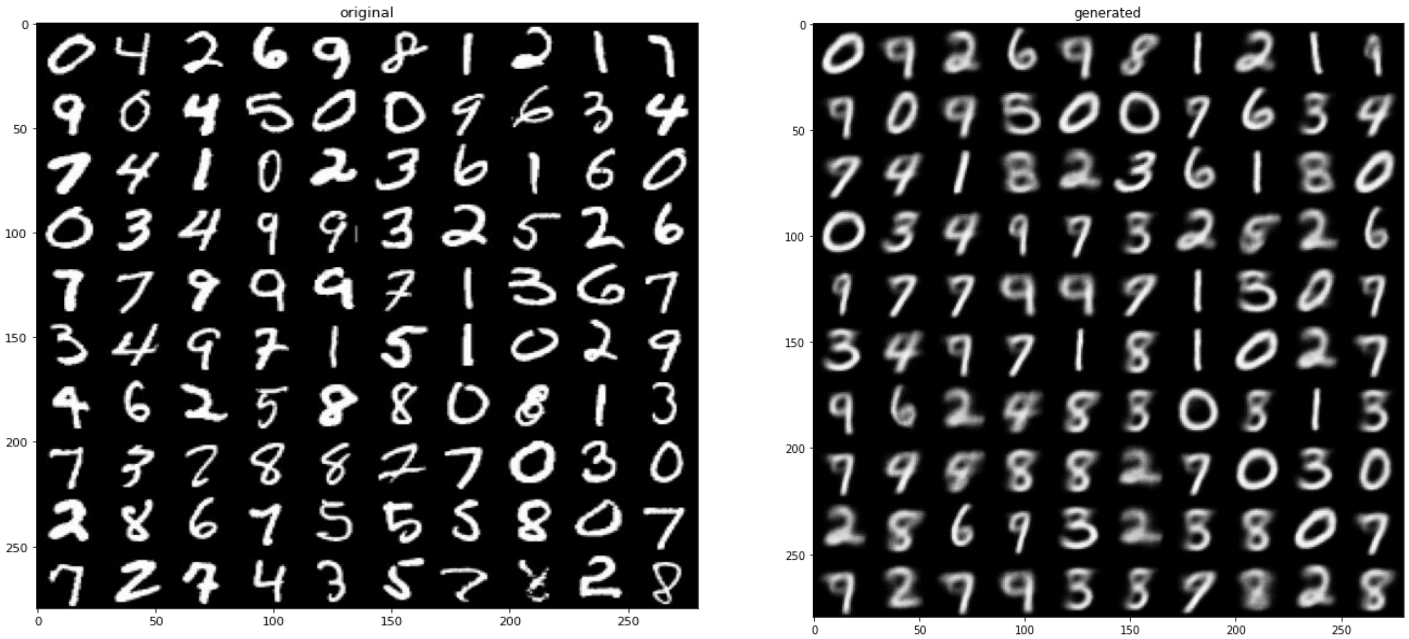
شکل ۱-۵: تصاویر اصلی داده شده و خروجی در ایپاک ۳ با روش VAE

خروجی از ایپاک ۵ :



شکل ۱-۶: تصاویر اصلی داده شده و خروجی در ایپاک ۵ با روش VAE

خروجی از ایپاک ۱۰ :

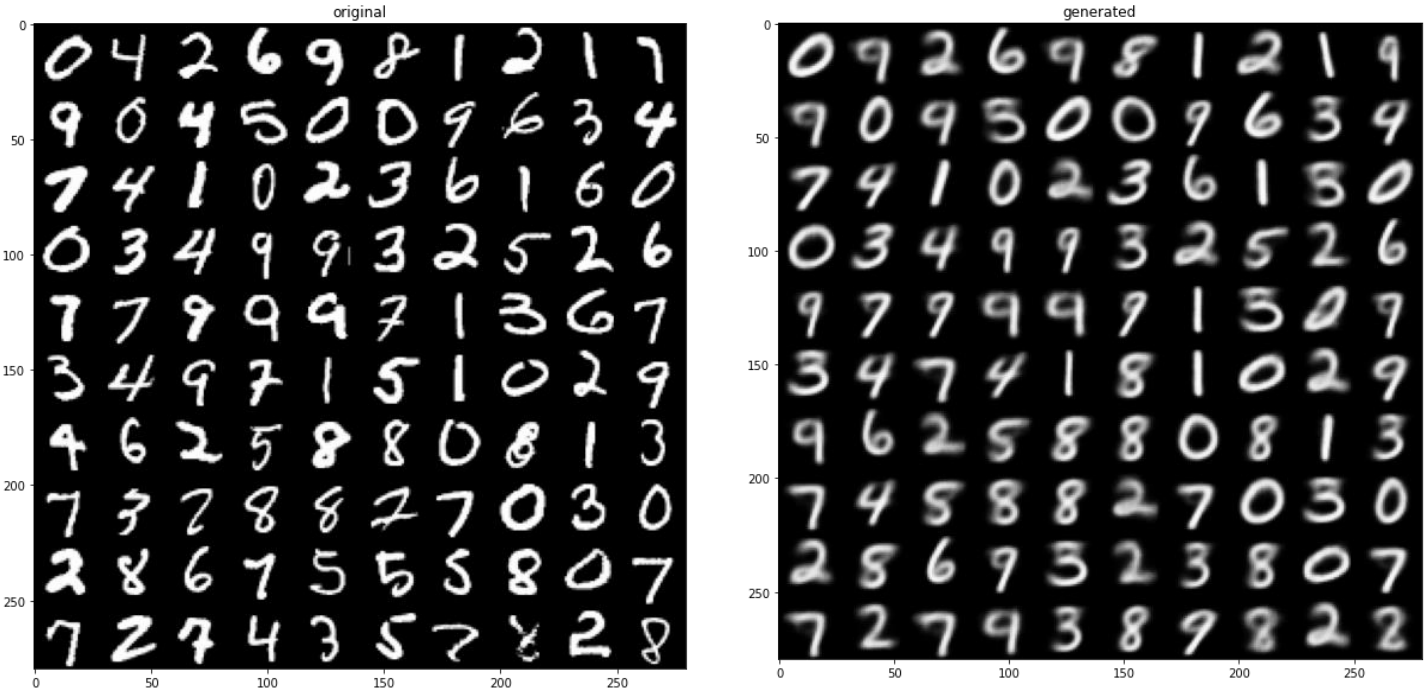


شکل ۱-۷: تصاویر اصلی داده شده و خروجی در ایپاک ۱۰ با روش VAE

خروجی از ایپاک ۲۰ :

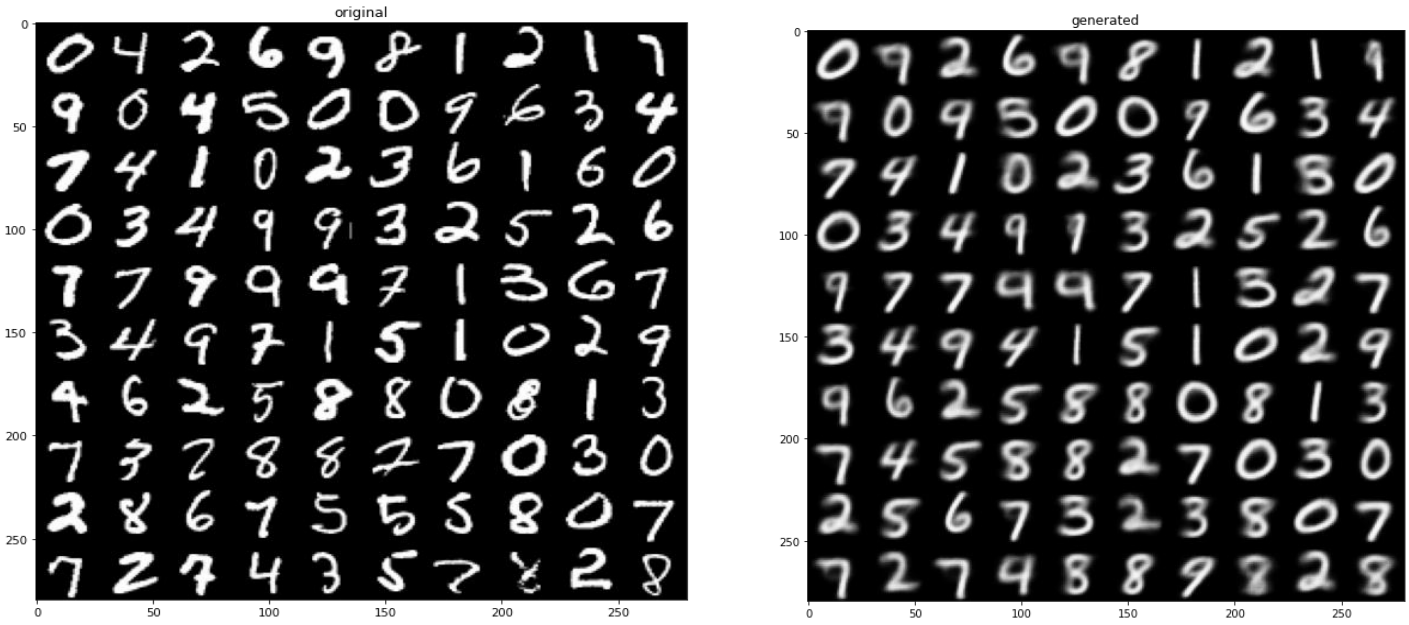
شکل ۱-۸: تصاویر اصلی داده شده و خروجی در ایپاک ۲۰ با روش VAE

خروجی از ایپاک ۴۰ :



شکل ۱-۹: تصاویر اصلی داده شده و خروجی در ایپاک ۴۰ با روش VAE

خروجی از ایپاک ۵۰ :



شکل ۱-۱۰: تصاویر اصلی داده شده و خروجی در ایپاک ۵۰ با روش VAE

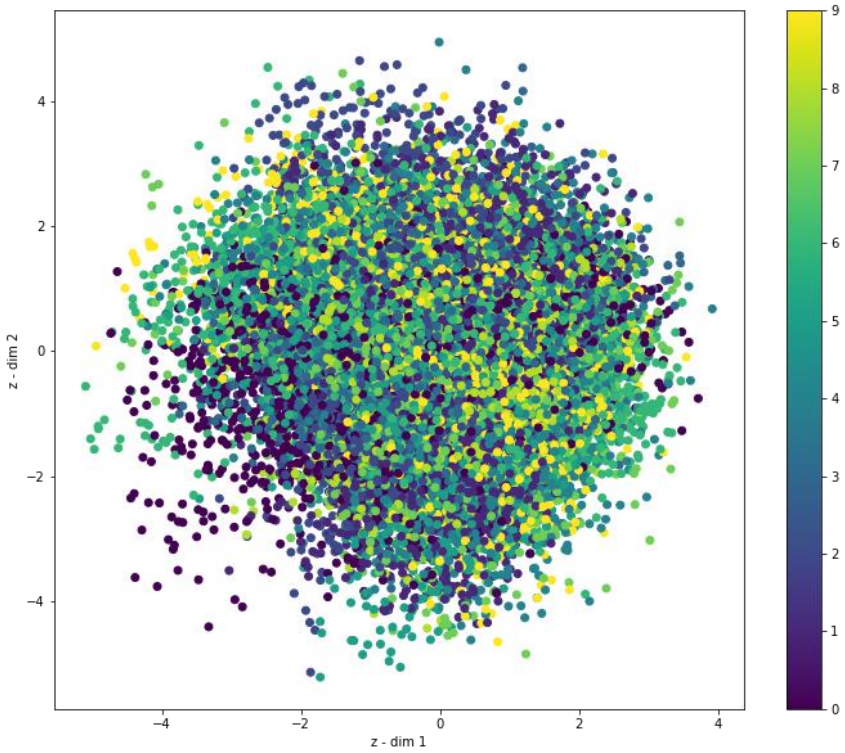
همانطور که در شکل های بالا مشاهده می شود در ابتدا تصاویر تولیدی به ویژه در ایپاک های ابتدایی دارای وضوح و کیفیت بسیار پایینی می باشد. به طور مثال در ایپاک ۲ ام شکل خروجی از شبکه بسیار در هم بوده و تشخیص اعداد از یکدیگر بسیار سخت و عملا غیر ممکن است. اما با پیش روی الگوریتم مسئله وضوح و کیفیت تصویر تولیدی توسط شبکه افزایش یافته و اعداد رفته رفته خواناتر می شود. به طوری که در ایپاک های نهایی اعداد به خوبی از یکدیگر قابل تشخیص می باشند. اما همچنان خروجی اندکی تار و با کیفیت کم باقی می ماند.

پاسخ بخش ز :

در این بخش به دنبال آن هستیم که الگوریتم Conditional VAE را بر روی داده های MNIST پیاده سازی کرده و با استفاده از مبنای این روش لیبل کلاس های مختلف را در فضای پنهان ویژگی ها لحاظ کرده و با استفاده از آنها به آموزش شبکه بپردازیم. به این ترتیب می توانیم نتایج حاصل از این بخش را با بخش قبلی که از روش خالص VAE استفاده کردیم، مقایسه کنیم که نتایج حاصل از این بخش در ادامه مشخص می شود.

پاسخ بخش ح :

در این بخش از سوال خواسته شده است که این بار شبکه conditional VAE را بر روی داده های MNIST پیاده سازی کنیم. به این ترتیب scatter plot مربوطه به صورت زیر خواهد بود:

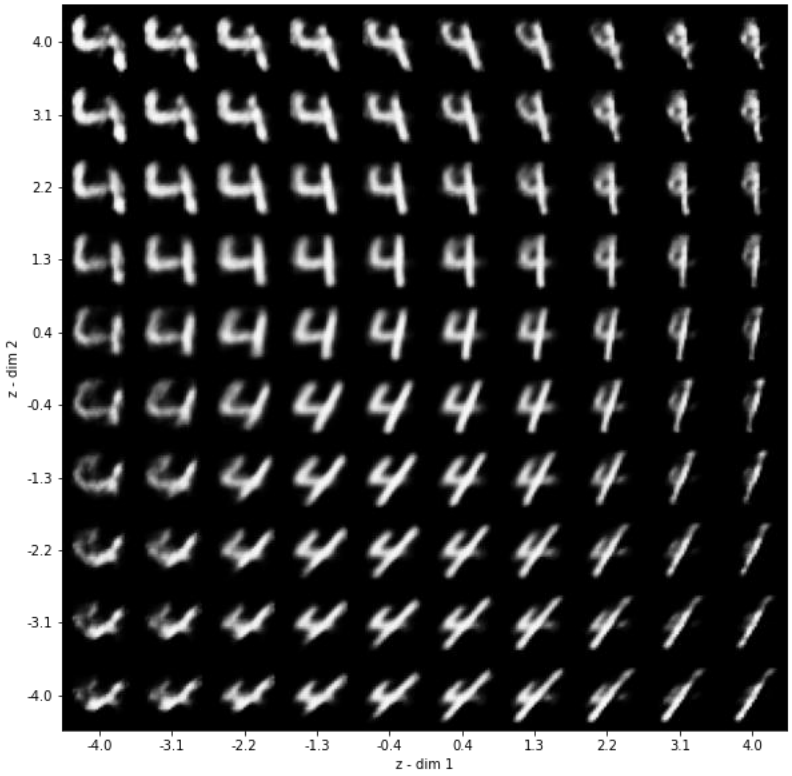


شکل ۱-۱۱: نمودار Scatter plot در فضای انتقال یافته برای ویژگی های مختلف برای روش conditional VAE

به این ترتیب در داخل شکل مشاهده می شود که نسبت به حالت قبلی داده های کلاس های مختلف در هم آمیختگی بیشتری را دارند. علت این امر را می توان از این رو دانست که در حالت VAE ساده نیاز است تا تا فضای z شامل اطلاعات بیشتری با شد به گونه ای که اطلاعات لازم برای جدا سازی و باز سازی digit های داده ها را دا شته با شد. به این ترتیب این دو ویژگی نیاز ا ست تا بیانگر خود عدد و دیگری برابر با style آن باشد تا با استفاده از آنها شبکه بتواند داده ها را به درستی بازسازی نماید. اما در روش conditional VAE در واقع لیبل های کلاس های مختلف را به صورت مستقیم در فضای مخفی ویژگی های (latent space) تولیدی اعمال می شود، شکل دیگری را انتظار داریم. به همین دلیل است که شبکه ما نیازی به داشتن اطلاعاتی که در حالت قبلی به آن نیاز است ندارد و در عوض می تواند از ویژگی های موجود در این فضای نهفته ویژگی ها اطلاعات و ویژگی های جالب تری را به دست بیاورد.

پاسخ بخش ط :

در این بخش نیز مطابق با آنچه که در حالت قبل رسم شد به دلخواه یکی از خروجی های به دست آمده برای یکی از لیبل ها را در فضای ویژگی ها را در ادامه نمایش می دهیم:



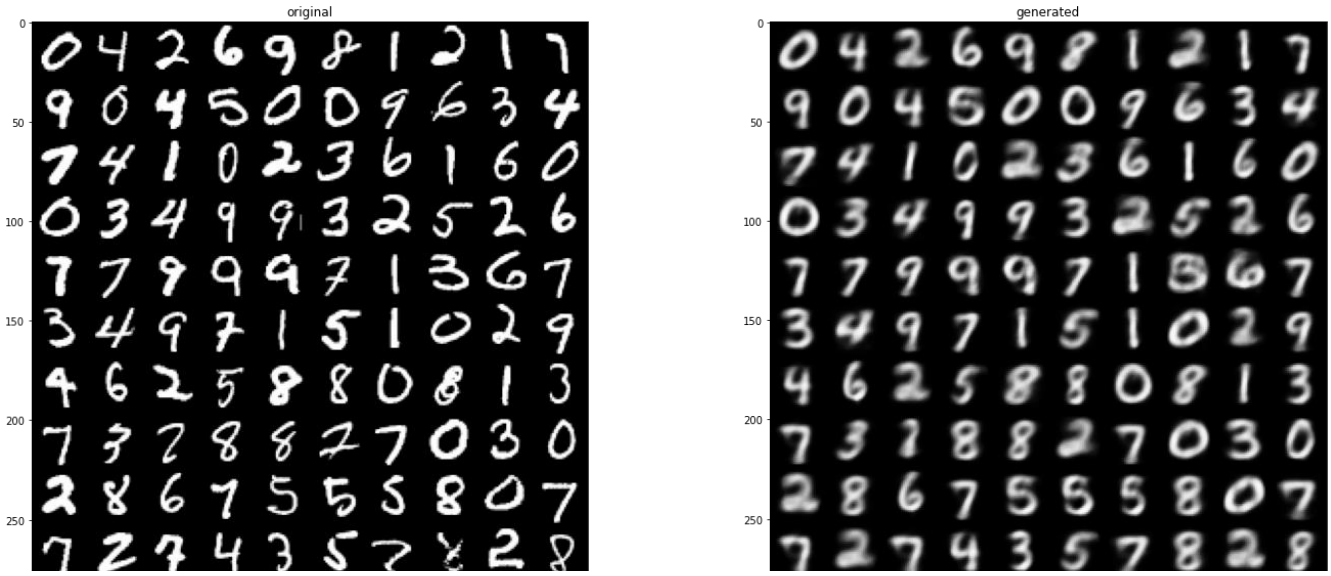
شکل ۱-۱۲: نمودار خروجی بخش encoder برای عدد ۴ برای روش conditional VAE

از آنجا که در این روش لیبل ها اعمال می شود می توان خروجی را برای یک لیبل مجزا به دست آورد که در بالا برای رقم ۴ مشاهده گردید. در داخل این شکل عدد ۴ در style های مختلف مشاهده می شود که در زوایای مختلفی قرار گرفته است.

پاسخ بخش ی :

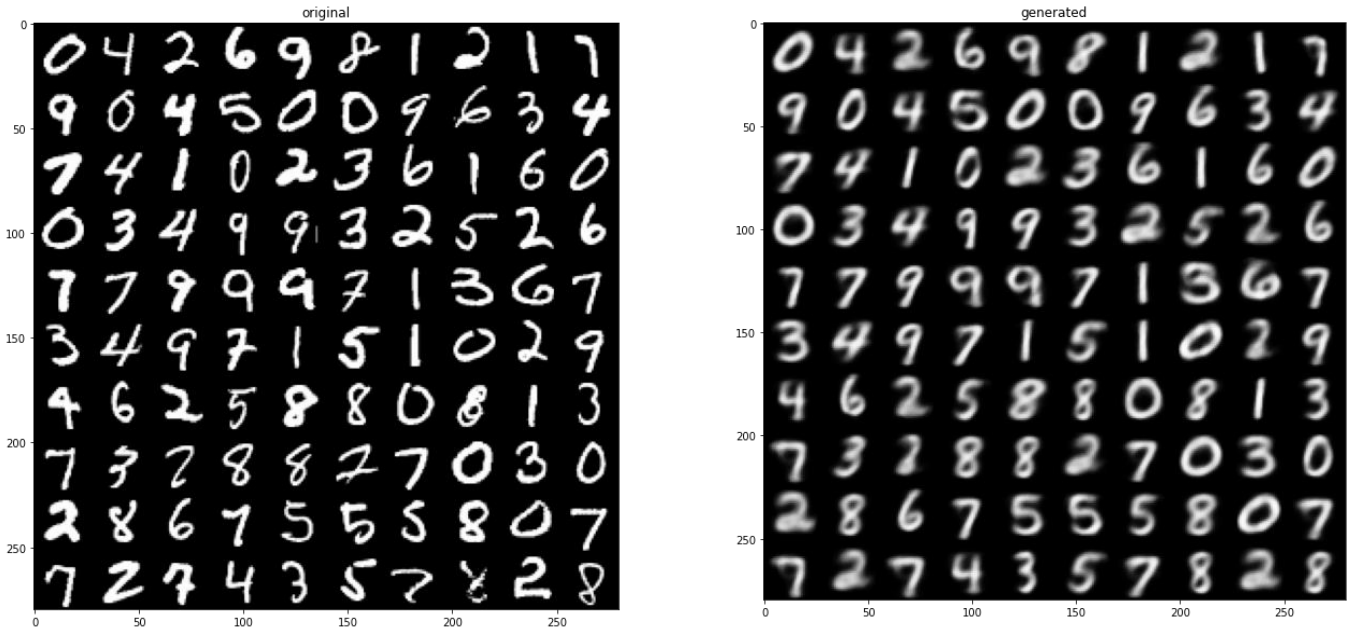
در این قسمت نیز مشابه بخش قبلی در برخی از ایپاک ها به دلخواه خروجی بازسازی شده شبکه رسم می گردد.

نتایج در ایپاک ۲ :



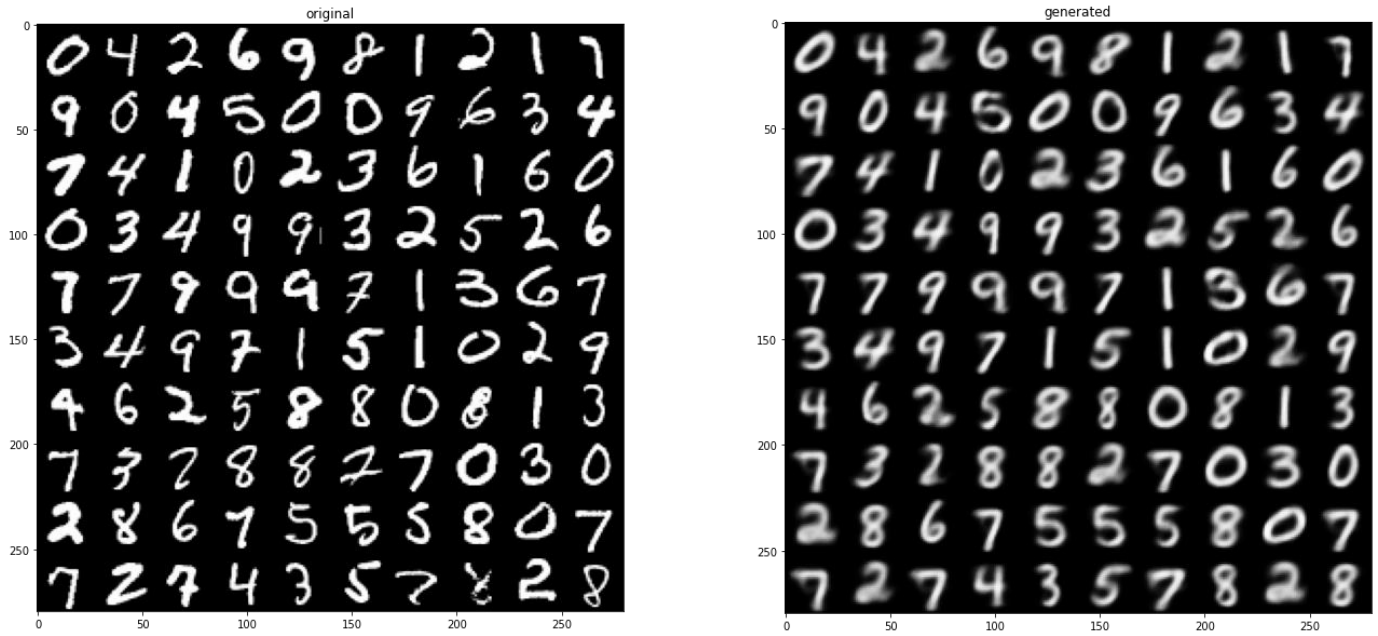
شکل ۱-۱۳ : تصاویر اصلی داده شده و خروجی در ایپاک ۲ با روش conditional VAE

نتایج در ایپاک ۳ :



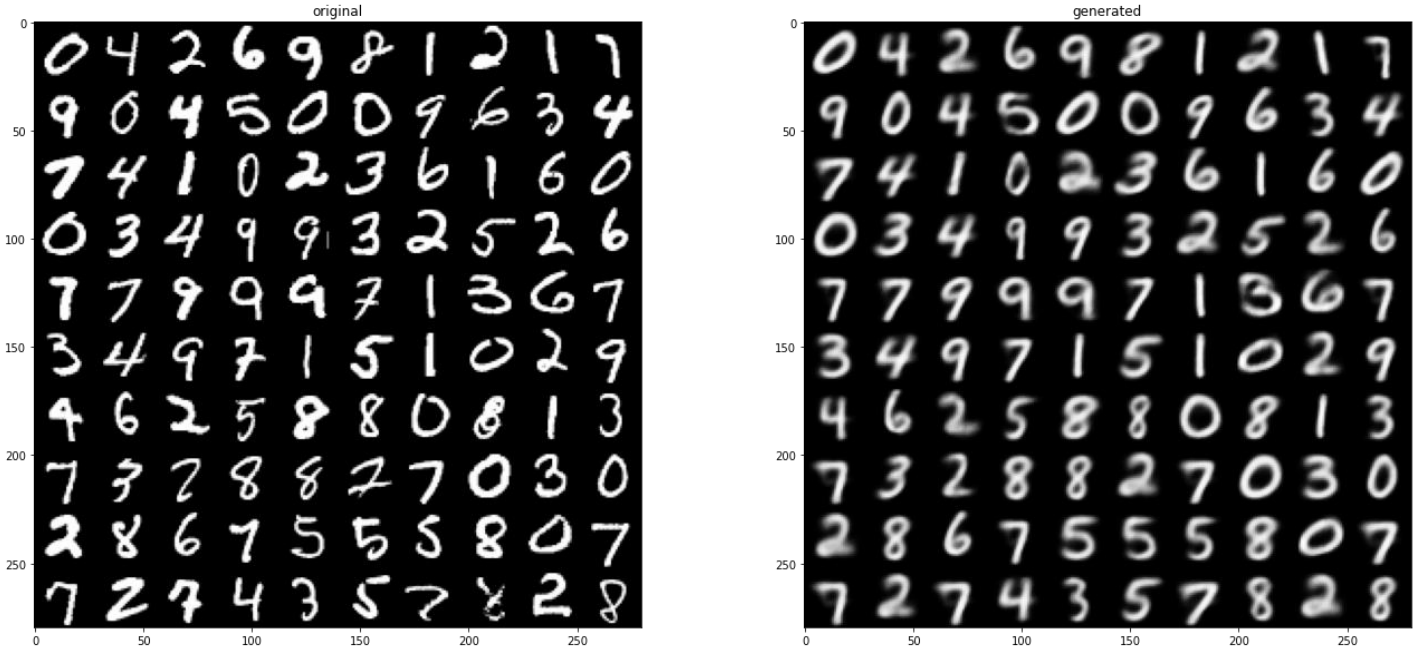
شکل ۱-۱۴: تصاویر اصلی داده شده و خروجی در ایپاک ۳ با روش conditional VAE

نتایج در ایپاک ۵ :



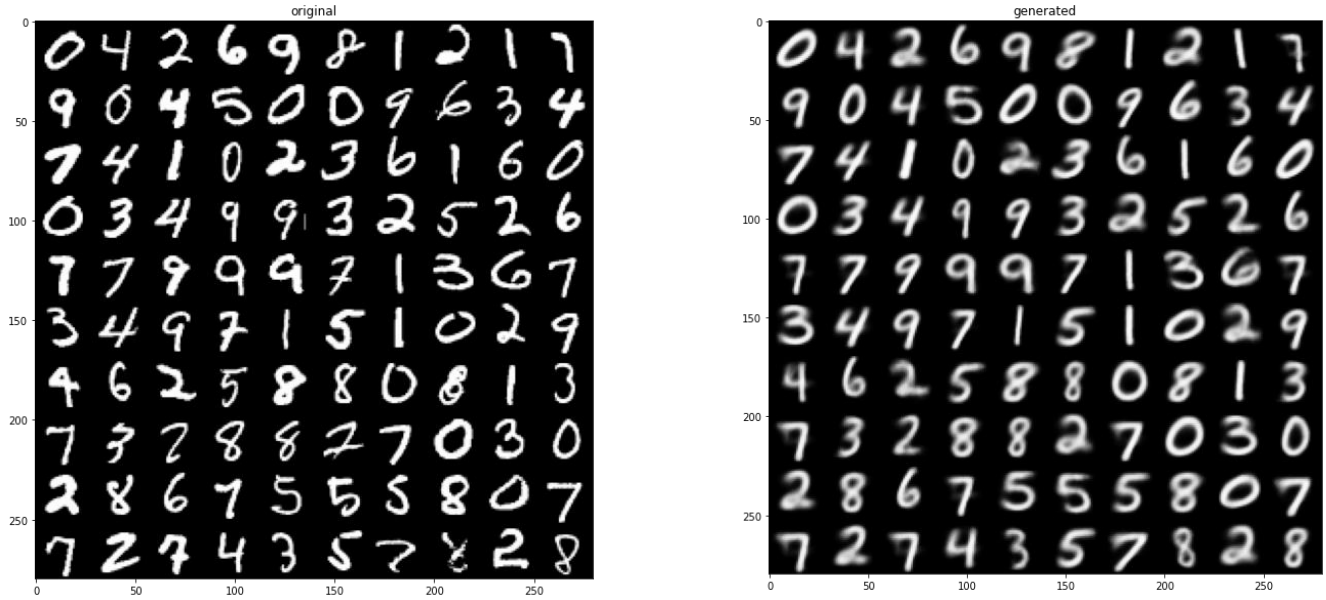
شکل ۱-۱۵: تصاویر اصلی داده شده و خروجی در ایپاک۵ با روش conditional VAE

نتایج در ایپاک ۱۰ :



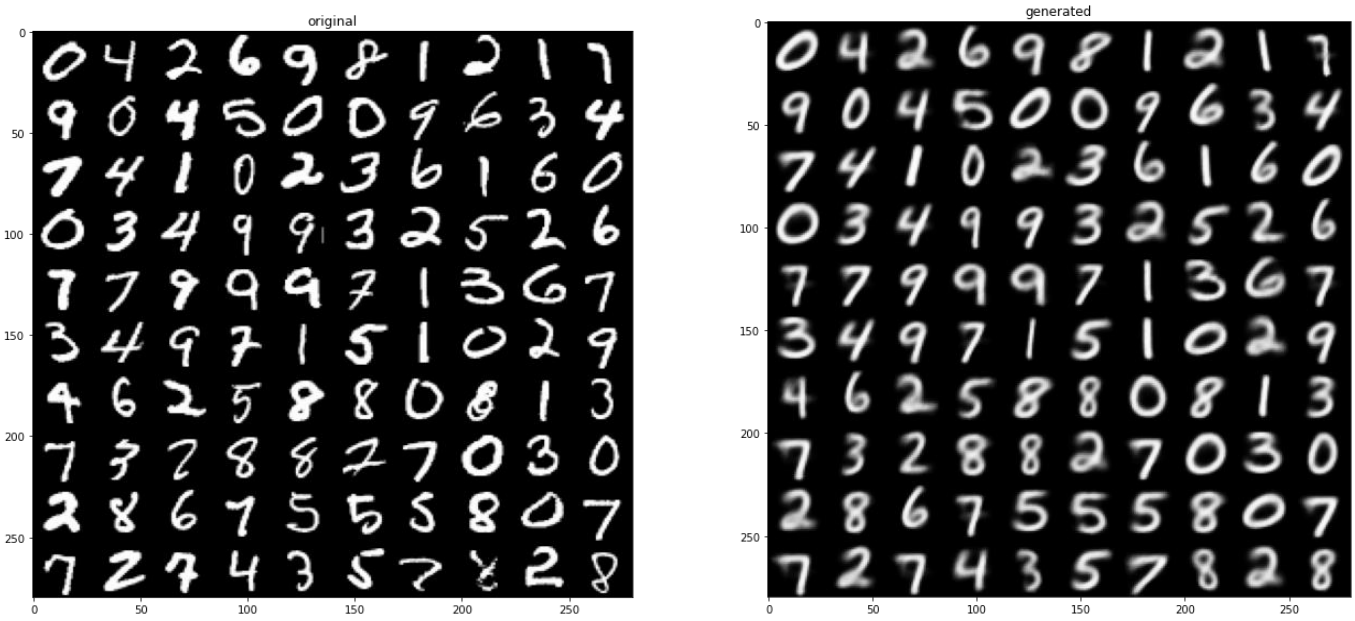
شکل ۱-۱۶: تصاویر اصلی داده شده و خروجی در ایپاک ۱۰ با روش conditional VAE

نتایج در ایپاک ۲۰ :



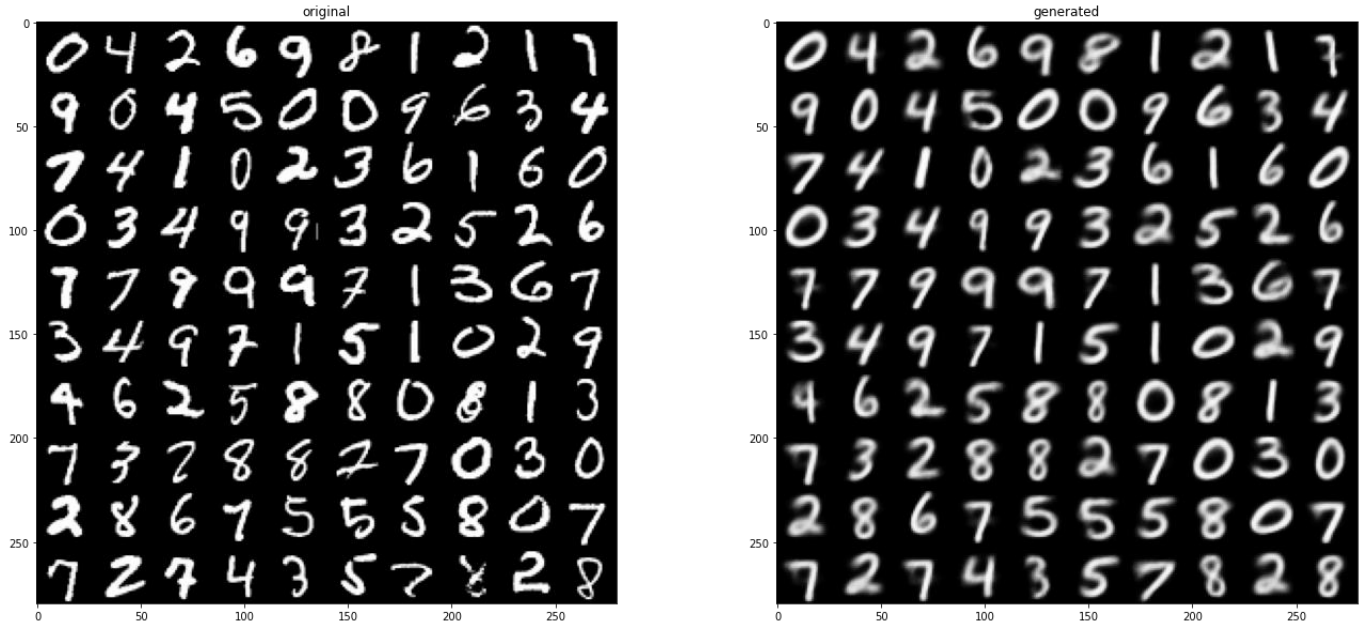
شکل ۱-۱۷: تصاویر اصلی داده شده و خروجی در ایپاک ۲۰ با روش conditional VAE

نتایج در ایپاک ۴۰ :



شکل ۱-۱۸: تصاویر اصلی داده شده و خروجی در ایپاک ۴۰ با روش conditional VAE

نتایج در ایپاک ۵۰ :

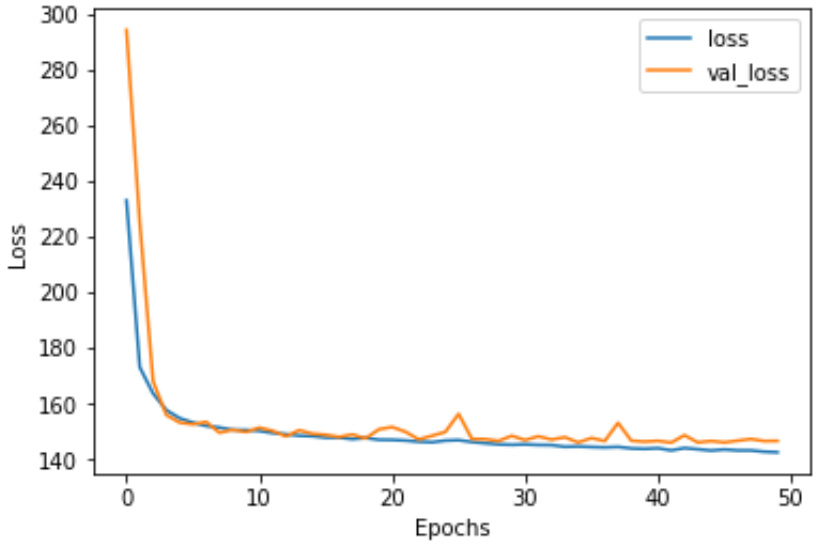


شکل ۱-۱۹: تصاویر اصلی داده شده و خروجی در ایپاک ۵۰ با روش conditional VAE

با مشاهده نتایج این بخش و مقایسه آن با نتایج به دست آمده از مدلسازی روش VAE بر روی داده های MNIST، به خوبی قابل مشاهده است که تصویر های تولید شده توسط conditional VAE دارای وضوح و دقت بیشتری از روش قبل می باشد. به طور مثال در ایپاک دوم تصویر تولید شده در روش VAE حاوی اطلاعاتی نبوده و به هیچ عنوان قابل تشخیص نمی باشد. این در صورتی است که برای ایپاک دوم در روش conditional VAE اطلاعات داخل تصویر خوانا می باشد تنها کیفیت کمتری نسبت به حالت اصلی دارد. این تفاوت در مورد ایپاک آخر نیز به سادگی قابل ملاحظه است که تصویر تولیدی در روش conditional VAE دارای وضوح و کیفیت بالاتری می باشد.

نمودار خطا برای VAE:

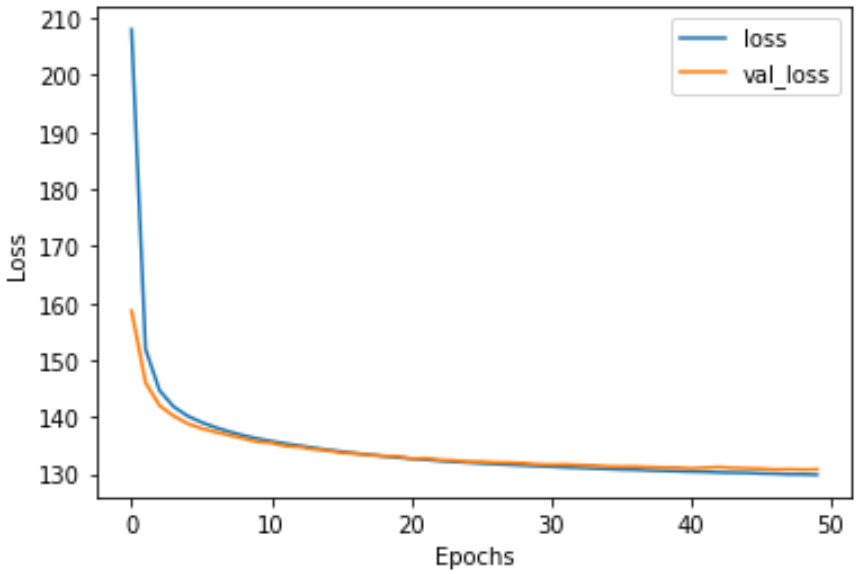
نمودار خطای کلی شبکه VAE در بازیابی و تولید تصاویر با خلاقیت بالا برای digit ها در هر ایپاک در شکل ۲۰ نمایش داده شده است. همانگونه که می‌بینید روند کلی تغییرات خطای بین ورودی و خروجی شبکه در حال کاهش می‌باشد که نشان از کلاستر بندی بهتر در عین بازیابی بهتر ورودی‌ها می‌باشد.



شکل ۱-۲۰: نمودار خطا در پیشروی ایپاک ها برای الگوریتم VAE

نمودار خطا برای conditional VAE:

همچنین این نمودار برای روش Conditional VAE نیز در زیر آورده شده است:



شکل ۱-۲۱: نمودار خطا در پیشروی ایپاک برای الگوریتم Conditional VAE

مقدار خطا در این روش پایین تر از روش قبل است.

سوال ۲- CycleGAN

در این سوال هدف آشنایی با Cycle-GAN برای تبدیل تصاویر غیر جفت و ارزیابی توانایی شما در دستکاری بخش های مختلف شبکه است. Cycle GAN تکنیکی برای تبدیل تصویر برای داده های غیر جفت است.

۱) در مورد سازو کار شبکه و خطاهای تعریف شده برای آموزش توضیح دهید.

۲) ۲ مفهوم PatchGAN استفاده شده در بخش *Discriminator* را توضیح دهید.

۳) این الگوریتم را بر روی مجموعه داده های monet2photo پیاده سازی کرده و سه خطای خواسته شده را بر روی آن پیاده سازی کرده و نتایج تصویر را نشان دهید.

۴) به جای بخش Generator در شبکه طراحی شده در بخش ۳ از یک ساختار U-net استفاده کنید.

پاسخ

پاسخ سوال ۱:

ساز و کار معماری:

در این معماری، ما دو Generator و دو Discriminator داریم. با استفاده از Generator ها دو نگاشت G: X -> Y و F: Y -> X را ایجاد می‌کنیم. در اینجا X عکس‌های طبیعی و Y نقاشی‌های مونت است. در واقع با Generator با نگاشت G سعی می‌کنیم عکس‌های طبیعی را به نقاشی‌های مونت تبدیل کنیم و با نگاشت F عکس این کار را انجام دهیم. لزوم وجود این دو معماری قرینه را در ادامه بررسی می‌کنیم.

اما در اینجا دو Discriminator هم داریم که Dx یک classifier است که تصاویر اصلی x را، از سایر تصاویر تمیز می‌دهد. و Dy هم همین کار را برای تصاویر Y می‌کند.

معماری Generatorها در واقع معماری یک encoder-decoder است که تصویر ورودی را دریافت می‌کند و سعی می‌کند از روی این تصویر تصویر دیگری را ایجاد کند و برای این کار با استفاده از شبکه‌های convolutional در ابتدا سعی می‌کند که ویژگی‌های مهم را در بخش encoder استخراج کرده و سپس در بخش decoder با استفاده از de-convolution آن‌ها را به تصویر مطلوب برساند.

در Discriminatorها هم ما با استفاده از لایه‌های convolution ویژگی‌های مهم در هر تصویر را استخراج می‌کنیم و بعد لایه‌های نهایی، کار طبقه‌بندی برای تشخیص کلاس داده‌ها را انجام می‌دهند.

آموزش Discriminator ها پیچیدگی زیادی ندارد. آن‌ها تصاویر را دریافت می‌کنند و مطابق یک طبقه‌بند عادی آموزش می‌بینند که تصاویر اصلی دامنه‌ی خودشان را تشخیص بدهند.

اما برای آموزش Generator ها ما نیاز به ایجاد یک سری حلقه داریم. در واقع می‌توان آموزش آن‌ها را در ۴ جنبه خلاصه کرد. برای آموزش Generator ای که مسئول نگاشت G است، ما ورودی از دامنه‌ی X را به آن می‌دهیم، و خروجی آن را به Dy می‌دهیم تا تشخیص دهد که خروجی تولید شده توسط این Generator حقیقی است یا تقلیدی. در بخش دوم، یک ورودی از دامنهY به این Generator داده می‌شود تا اموزش ببینید که ورودی‌های این دامنه را بدون تغییر به خروجی ارسال کند (identity mapping). در مرحله‌ی سوم، خروجی Generator در مرحله‌ی اول که تصویر X را از نگاشت عبور داده و سعی کرده به دامنه‌ی Y ببرد، به Generator متناظر با نگاشت F می‌دهیم و توقع داریم که بتواند تصویر اصلی X را بازیابی کند (forward cycle). در نهایت هم تصویری از دامنه‌ی Y که در مرحله‌ی دوم استفاده کردیم را به Generator متناظر با نگاشت F می‌دهیم تا آن را به دامنه‌ی X نزدیک کند، سپس خروجی آن را به Generator اصلی خود (متناظر با نگاشت G) وارد می‌کنیم تا بتواند تصویر اصلی از دامنه‌ی Y را بازیابی کند (backward cycle). در تمامی این مراحل، ما تنها وزن‌های Generator متناظر با نگاشت G را به‌روزرسانی می‌کنیم و وزن‌های Discriminator متناظر با آن و Generator دوم بدون تغییر باقی می‌ماند.

برای آموزش Generator دوم که متناظر با نگاشت F است هم مراحل مشابهی را طی می‌کنیم.

علت ایجاد این حلقه‌ها این است که ما بدون داشتن یک mapping مشخص در بین دو دامنه، بتوانیم نقاشی همان تصویر به‌خصوص را در خروجی ارائه کنیم و نه یک تصویر که صرفا یک نقاشی است.

جزئیات معماری استفاده شده در paper به این صورت است:

برای طراحی Generator از یک معماری با سه بلاک کانولوشنی، چندین residual block، دو بلاک کانولوشنی با stride برابر ۰,۵ و یک بلاک کانولوشن که فضای ویژگی‌ها به به رنگ‌های RGB مپ می‌کند (تعداد بلاک‌ها برای ورودی‌های ما که ۲۵۶*۲۵۶ هستند برابر ۹ است) همچنین نرمالیزیشن استفاده شده دراین لایه‌ها، InstanceNormaliztion است.

برای بخش Discriminator هم از pathcGAN های ۷۰*۷۰ استفاده می‌کنیم تا بررسی کنیم که آیا مربع‌های ۷۰*۷۰ تصویر real هستند یا fake.

بررسی‌های بیشتر این معماری در سوال ۴ که سوال پیاده‌سازی‌شده توسط ماست انجام گرفته است.

خطاهای تعریف‌شده برای آموزش:

1- Adversarial Loss:

در واقع هر Generator سعی می‌کند تصویری تولید کند که با تصاویر دامنه‌ی هدف کمترین اختلاف را داشته باشد و هر Discriminator سعی می‌کند که با دقت بیشتری بتواند تفاوت بین تصاویر تولیدی و عکس اصلی را تشخیص بدهد. در واقع یک تابع objective داریم که Generator سعی می‌کند آن را کمینه کند و در مقابل Discriminator سعی در بیشینه کردن آن دارد. این ترکیب Adversarial loss را می‌سازد. این تابع هزینه برای Generator متناظر با نگاشت G و Discriminator متناظر آن که همان Dy است به صورت زیر تعریف می‌شود.

$$\mathcal{L}_{\text{GAN}}(G, D_Y, X, Y) = \mathbb{E}_{y \sim p_{\text{data}}(y)}[\log D_Y(y)] + \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log(1 - D_Y(G(x))]$$

و فرآیند بهینه‌سازی آن در این راستاست که Genertor آن را کمینه می‌کند و Discriminator بیشینه. قابل توجه است که Discriminator در هر دو جمله‌ی آن نقش دارد اما Generator تنها در جمله‌ی دوم ظاهر می‌شود. نمایش ریاضی این بهینه‌سازی به صورت زیر است.

$$\min_G \max_{D_Y} \mathcal{L}_{\text{GAN}}(G, D_Y, X, Y)$$

2- Cycle Consistency Loss:

همانطور که قبلا هم اشاره شد، در صورتی که ما تنها از Adversarial loss استفاده کنیم، در صورتی که شبکه‌ی ما به اندازه‌ی کافی بزرگ باشد، میتواند هر ورودی از دامنه‌ی X را به یک خروجی در دامنه‌ی Y ببرد و البته تضمینی نیست که این خروجی در واقع نقاشی همان تصویر ورودی باشد. در واقع می‌تواند به هر نقاشی‌ای در آن فضا map شود. برای جلوگیری از این اتفاق، ما دو مسیر Forward cycle consistency و Backward cycle consistency را در نظر می‌گیریم؛ در اولی، یک تصویر از دامنه‌ی X را به Generator متناظر نگاشت G می‌دهیم و یک خروجی در دامنه‌ی توزیع Y خواهیم داشت، سپس این خروجی را به Generator متناظر نگاشت F می‌دهیم تا دوباره آن را به دامنه‌ی تصاویر X ببرد. در واقع توقع داریم این خروجی، همان ورودی اولیه‌ی این سیکل باشد. به همین ترتیب Backward cycle consistency هم با جابجایی Generatorها و ورودی و خروجی از جنس Y به همین ترتیب تعریف می‌شود یعنی یک ورودی از دامنه‌ی Y می‌گیرد، به Generator متناظر F میدهد و خروجی آن را هم به

Generator متناظر با G و دلخواه ما این است که دو تصویر ورودی و خروجی یکسان باشد. تابع هزینه متناظربا این هدف به صورت زیر تعریف می‌شود.

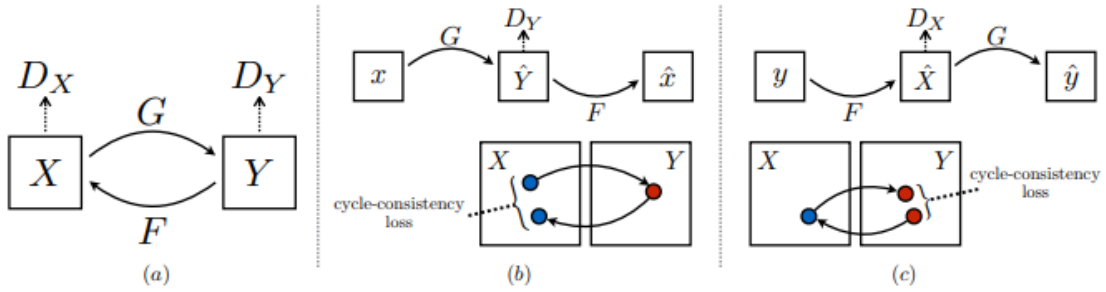
$$\mathcal{L}_{\text{cyc}}(G, F) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\|F(G(x)) - x\|_1] + \mathbb{E}_{y \sim p_{\text{data}}(y)}[\|G(F(y)) - y\|_1].$$

3- Full Objective:

تابع هزینه‌ی نهایی هم به صورت ترکیب خطی این توابع تعریف می‌شود:

$$\mathcal{L}(G, F, D_X, D_Y) = \mathcal{L}_{\text{GAN}}(G, D_Y, X, Y) + \mathcal{L}_{\text{GAN}}(F, D_X, Y, X) + \lambda \mathcal{L}_{\text{cyc}}(G, F),$$

شکل زیر شمای کلی این توابع هزینه را نمایش می‌دهد.



شکل ۲-۱: شمای کلی تابع هزینه مورد استفاده در روش Cycle GAN

3- Identity Loss:

این تابع هزینه در بخش توابع هزینه‌ی اصلی مقاله تعریف نشده، اما با استفاده از Advesarial loss و consistency loss ممکن است که generatorهای ما مثلا تم رنگی تصاویر را تغییر بدهند. برای مثال ممکن است اغلب نقاشی‌های مونت در فضای غروب آفتاب باشند و این تم رنگی بر تمامی تصاویر تولیدی قالب شود. برای جلوگیری از این مشکل، این تابع هزینه تعریف می‌شود. در واقع ما می‌خواهیم زمانی که یک ورودی از دامنه‌ی Y به Generator متناظر نگاشت G داده می‌شود، این تصویر مستقیم و با کمترین تغییرات در خروجی ظاهر شود. و عکس همین مساله برای تصاویر دامنه‌ی X. از این رو این تصاویر را بر خلاف معمول به Generator هایی که خروجی آن‌ها هم فضای این ورودی است می‌دهیم (مثلا ورودی‌ای از تصاویر X را

به Generator با نگاشت F می‌دهیم) و با تعریف تابع هزینه‌ی زیر کاری می‌کنیم که این تصاویر با کمترین میزان تغییر در خروجی ظاهرا شوند:

$$\text{erator: i.e., } \mathcal{L}_{\text{identity}}(G,F) = \mathbb{E}_{y\sim p_{\text{data}}(y)}[\|G(y)-y\|_1] + \mathbb{E}_{x\sim p_{\text{data}}(x)}[\|F(x)-x\|_1].$$

پاسخ سوال ۲:

در حالت عادی، Discriminatorها بعد از استفاده از لایه‌های convolution سعی می‌کنند که فضا را به یک فضای یک بعدی تقلیل دهند و در آن fake یا real بودن تصویر را تشخیص دهند. اما در PatchGANما patch های مختلف از تصویر را بررسی میکنیم و تشخیص می‌دهیم که هر کدام از آن‌ها fakeهستند و یا real و نهایتا خروجی یک بعدی نهایی را با توجه به خروجی این patch ها تعیین می‌کنیم. این کار به ما اجازه می‌دهد که با دقت بیشتری تصاویر را بررسی کنیم و Discriminator قدرتمندتری آموزش دهیم.

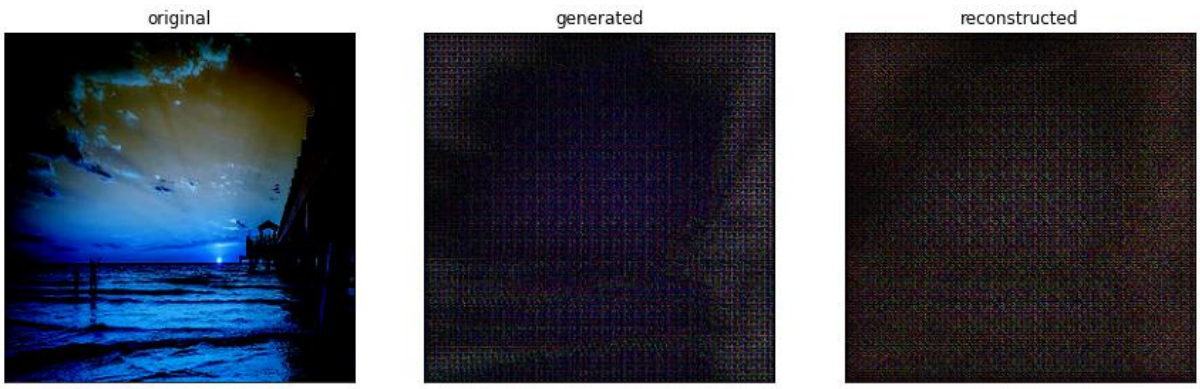
پاسخ سوال ۴:

در این بخش ما از یک Generator با معماری U-net استفاده کردیم که در واقع یک معماری encoder-decoder است.
در بخش encoder ما ۸ بلاک کانولوشنی داریم که در همه‌ی آن‌ها در انتها از leaky relu به عنوان activation function استفاده کرده‌ایم. تعداد فیلترها در لایه‌های اول تا سوم ۶۴، ۱۲۸ و ۲۵۶ است و ۵ لایه‌ی آخر همگی تعداد ۵۱۲ فیلتر دارند و اندازه‌ی stride همه‌ی این لایه‌ها برابر ۲ و سایز فیلترها هم ۴ است. در بلاک اول هم از نرمالیزیشن استفاده نکردیم.
در بخش decoder ما ۷ بلاک deconvolution داریم که خروجی هر یک از این بلاک‌ها (بعد از عبور از normalization و leaky relu) با خروجی بلاک‌های ۷ به قبل در بخش encoder ما concat می‌شوند. یعنی خروجی بلاک اول decoder با خروجی بلاک ۷ام encoder، خروجی بلاک دوم decoder با خروجی بلاک ۶ام encoder و به همین ترتیب. بعد از این بلاک‌ها هم یک لایه‌ی deconvoliution دیگر داریم که ما را به فضای RGB می‌برد. در سه بلاک اول این بخش از dropout استفاده کردیم و اندازه‌ی stride در تمامی بلاک‌ها ۲ و سایز فیلترها هم ۴ است. همچنین تعداد فیلترها در ۴ بلاک اول ۵۱۲ و بعد به ترتیب ۲۵۶، ۱۲۸× ۶۴ و ۳ است. نرمالیزیشن استفاده شده در هر دو بخش encoder و decoder همان InstanceNormalization می‌باشد.

برای بخش Discriminator هم از یک PatchGAN استفاده کردیم. در این معماری ما ۴ بلاک convolutionبرای استخراج ویژگی داریم که در هر یک از آن‌ها از InstanceNormalization و leaky reluاستفاده کردیم) بلاک اول normalization ندارد (و تعداد فیلترهای آن‌ها ۶۴، ۱۲۸، ۲۵۶ و ۵۱۲ است و سایز آن‌ها هم ۴. اندازه‌ی stride هم برای سه بلاک اول ۲ است و ما بعد از خروجی بلاک سوم و چهارم از zero padding استفاده کردیم تا سایز را تغییر ندهیم .بعد از این‌ها یک لایه‌ی کانولوشن دیگر داریم که در واقع fake یا real بودنpatch ها را تشخیص می‌دهد.

برای بهینه‌کردن تمامی این بخش‌ها از بهینه‌ساز Adam استفاده شده است. برای مقدار دهی اولیه نیزاز توزیع نرمال با $sd = 0.2$ استفاده شده است.

ما روند آموزش را در دو ایپاک تکرار کردیم و برای نشان دادن تغییرات در هر ایپاک، یک تصویر از بین عکس‌ها را به مدل می‌دهیم تا آن را به نقاشی تبدیل کند و مجددا به صورت عکس در بیاورد. همین کار را برای یک تصویر از بین نقاشی‌ها هم تکرار می‌کنیم. و نهایت در خروجی‌ها را برای یک مجموعه از هر دو دامنه را بررسی می‌کنیم. (این تصاویر از بین داده‌های **تست** انتخاب شده‌اند)
قبل از شروع آموزش، تصویر انتخاب شده از بین عکس‌های طبیعی، تبدیل شده‌ی آن به نقاشی و بازیابی‌شده‌ی آن به صورت زیر است.



شکل ۲-۲: تصویر انتخاب شده به همراه تبدیل آن و بازسازی شده آن قبل از آموزش شبکه

بعد از ایپاک اول تصاویر بدست آمده در شکل بعدی آورده شده است.

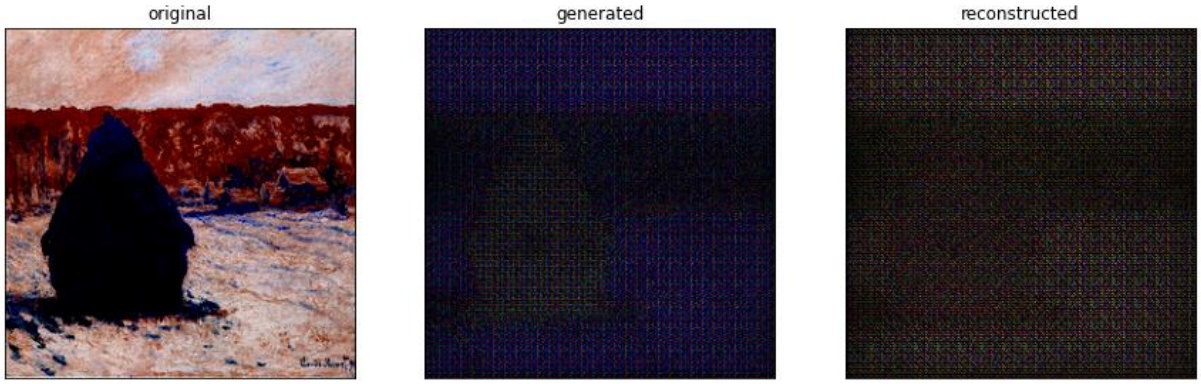
شکل ۲-۳: تصویر انتخاب شده به همراه تبدیل آن و بازسازی شده آن پس از اجرای یک ایپاک

و بعد از ایپاک دوم هم در به نتیجه‌ی زیر رسیدیم.

شکل ۲-۴: تصویر انتخاب شده به همراه تبدیل آن و بازسازی شده آن پس از اجرای دو ایپاک

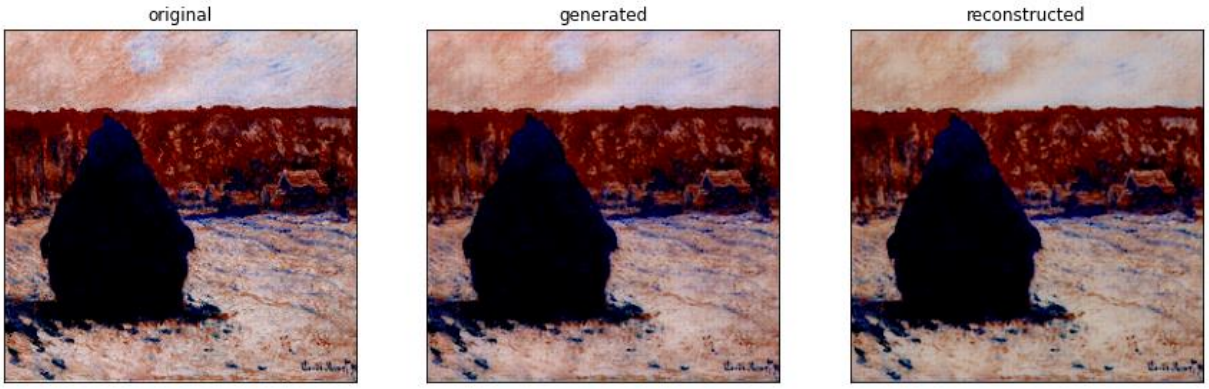
با بررسی سه تصویر بالا به نظر می‌رسد که شبکه به تدریج شروع به کم کردن رزولوشن تصاویر عکس و بردن آن‌ها به سمت تصاویر نقاشی کرده است. (در ایپاک اول در ظاهرا ما یک نگاشت ساده، بدون توانایی اعمال تغییرات پیچیده را داریم.)

همچنین تصویر انتخابی از بین نقاشی‌ها، تبدیل‌شده‌ی آن به عکس و بازیابی‌شده‌ی آن را هم در زیر مشاهده می‌کنید. مجموعه‌ی اول مربوط به قبل از شروع آموزش است.



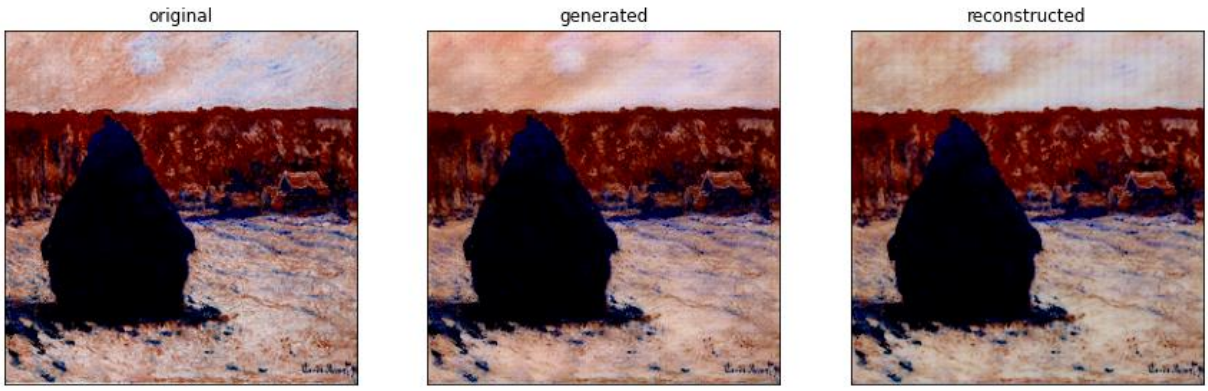
شکل ۲-۵: تصویر انتخاب شده به همراه تبدیل آن و بازسازی شده آن قبل از آموزش شبکه

تصویر بعدی بعد از اتمام ایپاک اول آموزش ثبت شده است:



شکل ۲-۶: تصویر انتخاب شده به همراه تبدیل آن و بازسازی شده آن پس از اجرای یک ایپاک

و بعد از ایپاک دوم هم تصاویر زیر بدست آمده است:



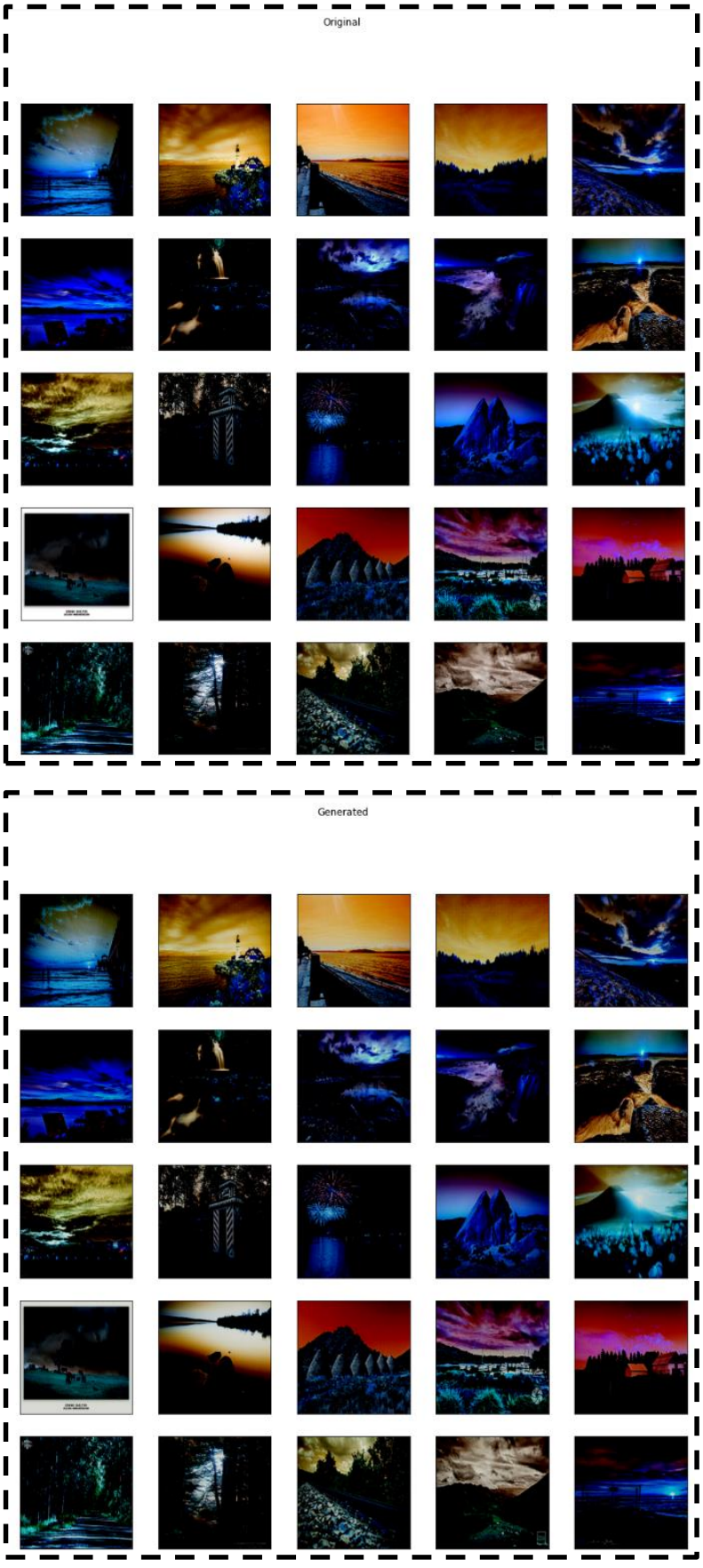
شکل ۲-۷: تصویر انتخاب شده به همراه تبدیل آن و بازسازی شده آن پس از اجرای دو ایپاک

بیشترین تغییرات این تصاویر در قسمت آسمان قابل مشاهده است. به تدریج رده‌های حاصل از قلم حذف و رنگ بیشتر به سمت یکدستی عکس‌های طبیعی پیش‌می‌رود.

همچنین ما تعداد ۲۵ تصویر از مجموعه دادگان تست را هم بعد از هر ایپاک به شبکه داده‌ایم، و برای مقایسه، هر بار آن‌ها را در کنار تبدیل‌یافته‌ی آن‌ها به دسته‌ی مقابل نمایش می‌دهیم تا تغییرات قابل مشاهده باشند.

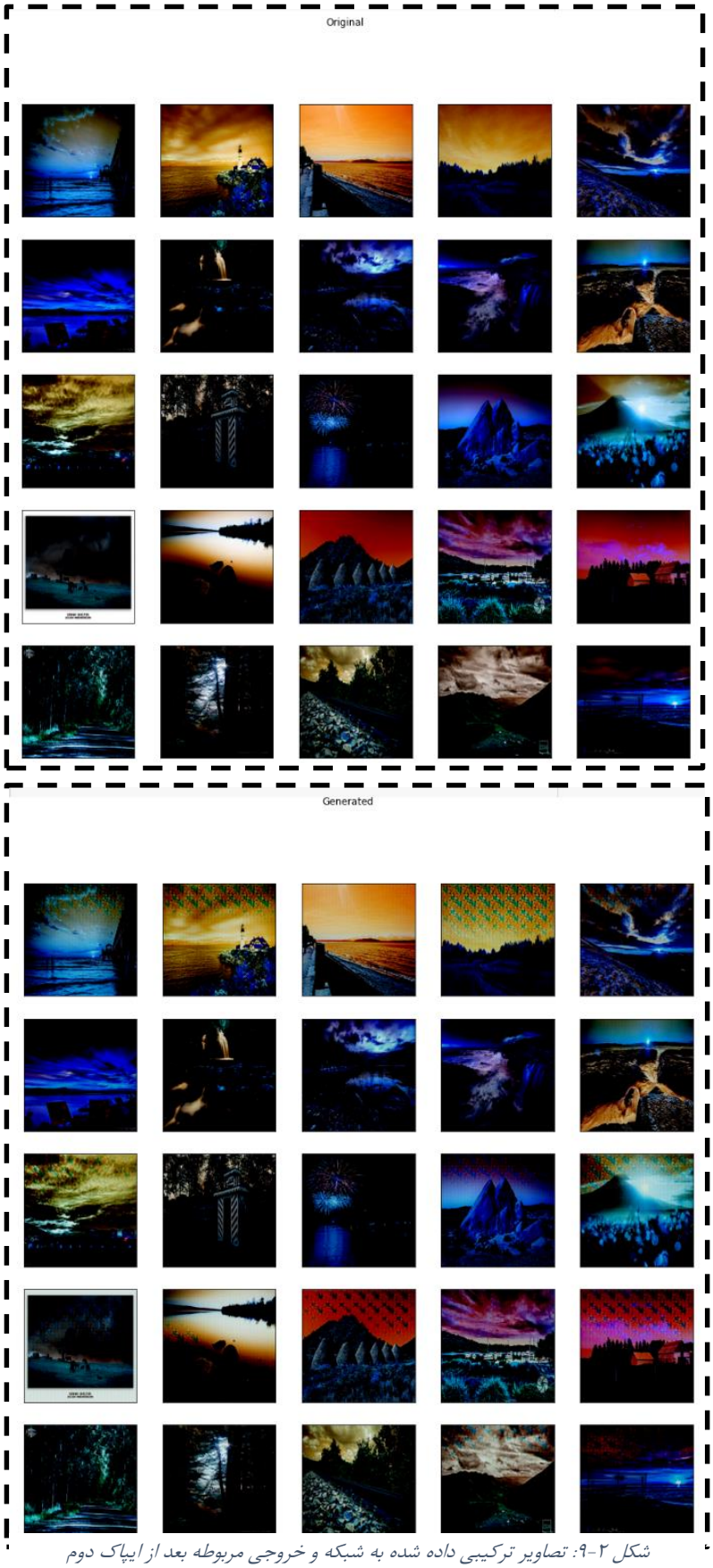
حال برخی از خروجی های مسئله را به صورت چنتایی در ادامه مشاهده می کنید.

بعد از ایپاک اول و بررسی تغییرات تبدیل عکس‌های طبیعی به نقاشی:



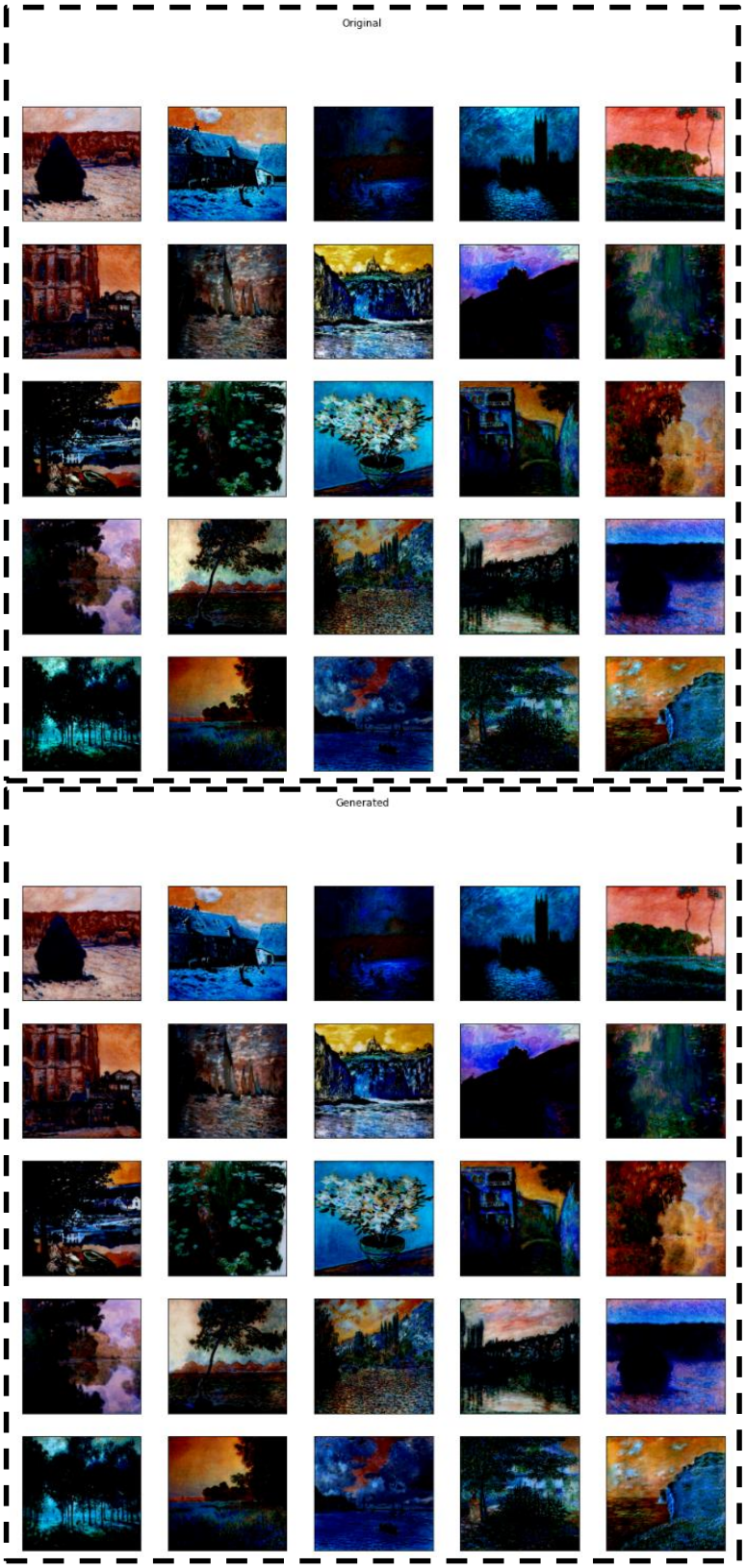
شکل ۲-۸: تصاویر ترکیبی داده شده به شبکه و خروجی مربوطه بعد از ایپاک اول

و بعد از ایپاک۲:



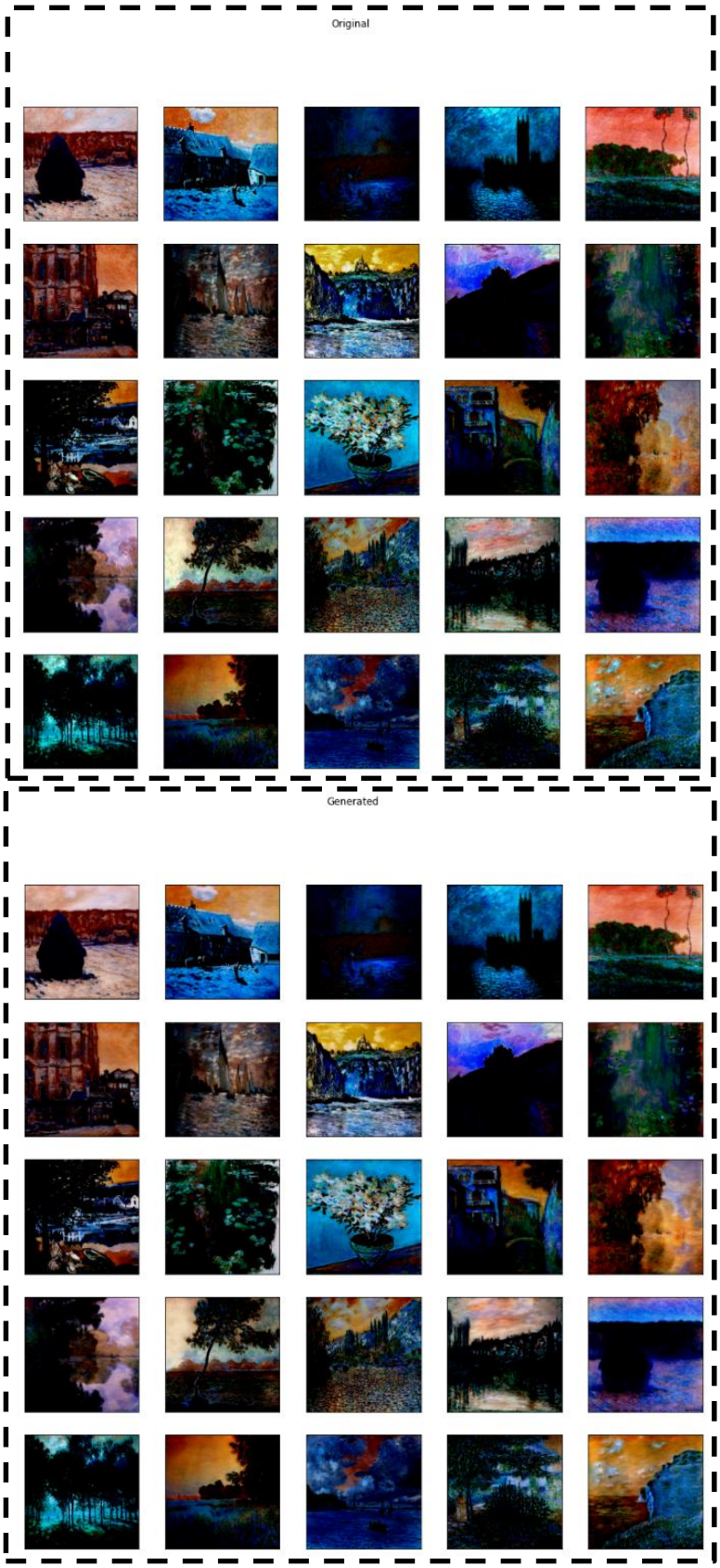
شکل ۲-۹: تصاویر ترکیبی داده شده به شبکه و خروجی مربوطه بعد از ایپاک دوم

همانند قبل می‌توانیم ظهور یک سری الگوهای تکرار شونده‌ی ریز در قسمت‌های یکرنگ تصاویر را مشاهده کنیم. همچنین برای تبدیل نقاشی‌ها به تصاویر حقیقی هم روند مشابهی را طی کردیم.



شکل ۲-۱۰: نقاشی ترکیبی داده شده به شبکه و خروجی مربوطه بعد از ایپاک اول

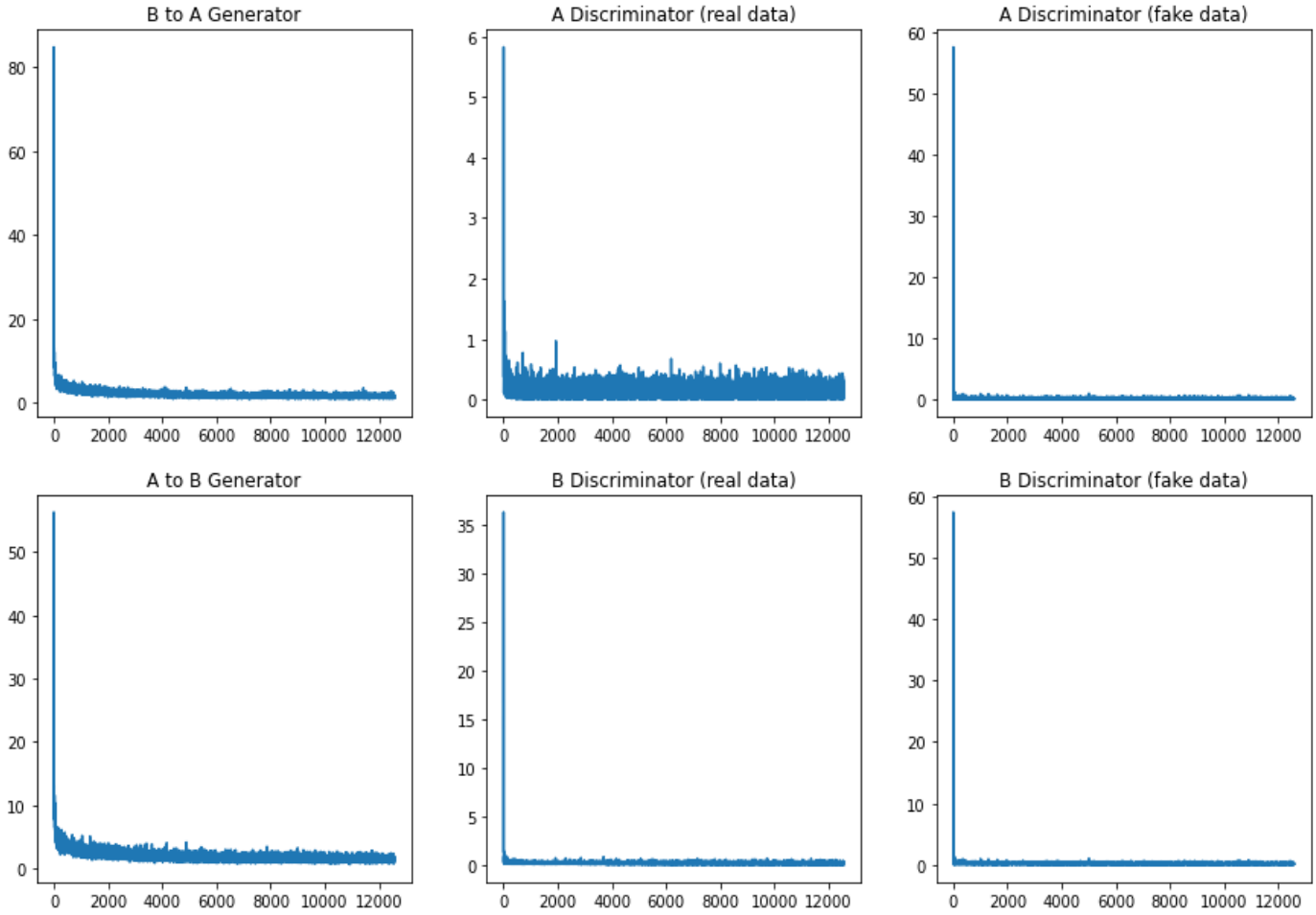
بعد از ایپاک۲:



شکل ۲-۱۱: نقاشی ترکیبی داده شده به شبکه و خروجی مربوطه بعد از ایپاک دوم

تقریبا ردهای عامدانه‌ی قلم از بین طرح‌ها به تدریج حذف می‌شوند و تصاویر به سمت واقعی‌تر شدن پیش می‌روند.

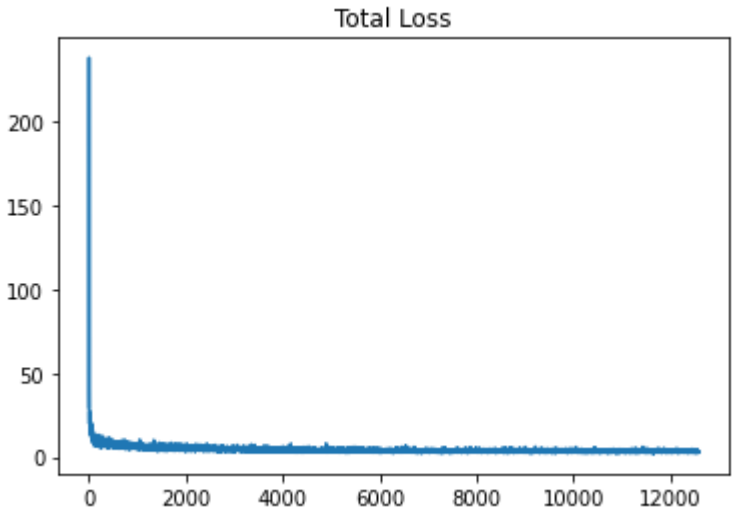
همچنین نمودار مقادیر loss برای هر یک از بلاک‌های سازنده‌ی این معماری (شامل خطای هر یک از Generatorها و خطای Discriminatorها هم برای داده‌های real و هم برای داده‌های fake) در زیر قابل بررسی است (در طی دو ایپاک)



شکل ۲-۱۲: نمودار loss بخش های مختلف معماری شبکه

در این تصویر A در واقع نماینده‌ی تصاویر نقاشی و B نماینده‌ی عکس‌های حقیقی است.

همچنین نمودار loss کلی مدل که جمعی از لاس‌های تعریف شده است، در تصویر زیر گزارش شده است.



شکل ۲-۱۳: نمودار loss کلی

سوال ۳ - StackGAN

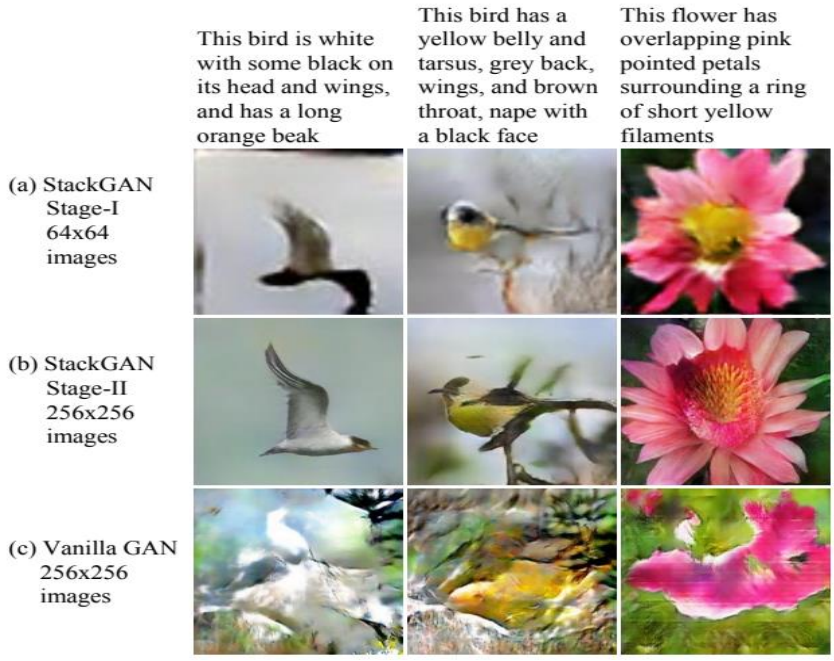
از کاربردهای خیلی معروف و جذاب GAN می توان به تبدیل متن به تصویر اشاره کرد، در این حالت یک متن به شبکه داده می شود و متناسب به متن، شبکه GAN به شما یک تصویر را به عنوان خروجی می دهد. در این بخش به بررسی مقاله مربوطه می پردازیم.

پیشینه مقاله، ساختار شبکه، تابع خطا، دستاورد های نویسندگان و ... آورده می شود.

پاسخ

مقدمه :

در ابتدای امر نیاز است تا به ضرورت فعالیت در حوزه موضوع مورد مطالعه بررسی گردد. تولید عکس هایی با کیفیت بالا از جمله اهداف مهم در زمینه های کاری شبکه های عصبی و به طور کلی تر یادگیری ماشین می باشد. به همین رو روش ها و الگوریتم های مختلفی به این منظور توسط محققان مختلف ارائه شده است. یکی از مباحث تازه در این شاخه ایجاد و تولید عکس از نوشته ها و توضیحات متفاوت می باشد. تلاش های بسیار زیادی در راستای افزایش کیفیت و وضوح تصویر های ایجاد شده در بر مبنای جملات توضیحی، صورت گرفته است که در این مقاله سعی بر معرفی یکی از این روش ها صورت گردیده است. نکته ای که در راستای تولید عکس ها حائز اهمیت است، میزان جزئیاتی است که در نتیجه نهایی از جملات مربوطه بازسازی می شوند. بدیهی است که هرچه محصول پایانی حامل جزئیات دقیق تری باشد، برای بیننده پذیرفته شده تر و به واقعیت نزدیک تر خواهد بود. به منظور انجام این فرآیند پیچیده نیاز است تا آن را به قسمت های کوچکتری تقسیم کرده و سعی بر بهبود هر یک از این بخش های کوچکتر نمود. در ادامه به بررسی دقیق تر الگوریتم stack GAN می پردازیم که با هدف تولید عکس های ۲۵۶*۲۵۶ با وضوح و جزئیات بالا، به دو بخش اصلی stage_۱ و stage_۲ تقسیم بندی می شود که در هر یک از این بخش های مختلف سعی بر به کارگیری روش های جدید برای data augmentation صورت گرفته است. در مرحله اول (stage_۱) با به دست آوردن اطلاعات کلی موجود در داخل متن های مدنظر تصاویری ابتدایی و ساده با کیفیت پایین تر تولید می شوند و در ادامه روند آموزش شبکه در مرحله دوم (stage_۲) اطلاعات خروجی از بخش قبلی و نیز بازخوانی متن های مربوطه تصاویری حاوی جزئیات بیشتر و نیز با وضوح به مراتب بالاتر نسبت به داده های تولید شده در بخش اول و نیز آنچه توسط vanilla GAN تولید می شود، می باشند. در ادامه در تصویر تفاوت خروجی های گفته شده از روش های مختلف با یکدیگر مقایسه می گردد.



شکل ۳-۱: نمونه ای از نتایج به دست آمده از روش Stack GAN

در تصاویر بالا به خوبی مشاهده می شود که تصویر ایجاد شده توسط این روش دارای کیفیت بالاتری نسبت به روش های گفته شده در بالا می باشد. به طور خلاصه میتوان گفت که استفاده از روش های هوشمندانه به منظور تولید ویژگی ها و تبدیل متن و نیز استفاده مجدد از داده های اصلی به علاوه ویژگی های استخرج شده در مرحله اول، که در مرحله دوم صورت میگیرد نکاتی است که منجر به نتایج بهتر می گردد.

پیشینه و کارهای مرتبط:

یکی از روش هایی که پیش تر به منظور این تبدیل متن به تصویر صورت می گرفته است، بهره گرفتن از Variational auto encoder بوده است که به منظور کاهش بازه مربوط به max likelihood می باشد. از دیگر موارد و روش هایی که می توان به آنها اشاره کرد، autoregressive models ها محسوب می شوند که به یاری آنها به مشروط سازی توزیع داده ها در داخل فضای ویژگی ها میسر می شود که منجر به تولید عکس های مربوطه از متن های مربوطه می باشد. درادامه راه فرآیند تولید عکس های با وضوح بالا، آشنایی و بهره گیری از Generative Adversarial Networks که به اختصار GAN نامیده می شود تحول شگرفی در این زمینه ایجاد کرد. این مدل ها توانایی رسیدن به عکس های بالا را تا حد قابل توجهی دارا بودند اما مشکل عدم قطعیت ها نکته ای بود که در آغاز راه استفاده از این مدل ها نتایج را تحت تاثیر خود قرار داد و بهبود قابل توجهی را در نتایج ایجاد نمود. به منظور حل مشکل ذکر شده راهکار های گوناگونی ارائه شد که از جمله آنها می توان به Energy Based GAN اشاره نمود که تا حد خوبی به پایداری مدل های پیشین کمک کرد. از دیگر تکنیک هایی که ارائه شد افزودن شروط مشخص در فرآیند آموزش شبکه می باشد (مانند اضافه کردن تولید داده های تصادفی مشروط به لیبل های کلاس های مختلف، انتقال دامنه)

همچنین استفاده از روش های super resolution نیز با ایجاد محدودیت هایی توانستند با کیفیت های بالاتری دست یابند. در روش های جدیدتر align DRAW ها توانستند بازم هم نسبت به گذشته تغییرات مثبتی را در نتایج به دست آمده ایجاد کنند. در ادامه مسیر در استفاده از GAN ها نوآوری هایی صورت گرفت مانند استفاده از شبکه های متوالی، تغییراتی در فرآیند یادگیری و ... نظیر آنچه که در stack GAN صورت می گیرد. اما این روش های پیشین در بهترین حالت توانستند به تولید عکس های با کیفیت به سایز ۳۲*۳۲ دستیابی کنند که در مقایسه با آنچه یا استفاده از stack GAN می توان به آن رسید (۲۵۶*۲۵۶) بسیار ضعیف تر عمل می کنند.

ویژگی ها و ساختار مدل:

همانگونه که در بخش های پیشین گفته شد این الگوریتم شامل دو بخش اصلی می باشد که در ادامه به اختصار به آنها اشاره می شود:

stage_ I.1:

در این بخش از شبکه توسط روش های تبدیل متن ویژگی های مورد نیاز برای ساخت تصاویر ساده و ابتدایی استخراج شده و با استفاده از آنها تصاویری کوچک با وضوح و کیفیت پایین ساخته می شود که در تلاش است شکل کلی و رنگ های اصلی را بسازد.

stage_ II.2:

در این بخش از ویژگی های عکس های استخراج شده در مرحله قبلی و نیز بازخوانی متن های مربوطه و سعی بر استخراج جزئیات و ویژگی های جدید که در مرحله قبلی شناسایی نشده است به تولید تصاویر با جزئیات بیشتر و با وضوح بالاتر در ابعاد ۲۵۶*۲۵۶ پرداخته می شود. در واقع ساختار این بخش شبیه دو بخش encoder و decoder عمل می کند در ابتدا با استفاده از ویژگی های به دست آمده و concat کردن آنها به فضای پایین تر رسانده و بار دیگر در بخش decoder با استفاده از upsampling های متوالی به تصاویری با وضوح بالا دست می یابد.

همانطور که می دانیم در ساختار کلی GAN ها بخش مولد (Generator) سعی بر تولید داده هایی شبیه به داده های اصلی با استفاده از نویز های تصادفی دارد بدین صورت که تشخیص داده های واقعی و تولیدی توسط این بخش برای جداساز سخت تر باشد. از طرفی دیگر این بخش جداساز (Discriminator) به دنبال آن است تا به نحوی خود را قوی تر کند که داده های واقعی و غیر واقعی را از یکدیگر تشخیص دهد. به این منظور پیشبرد الگوریتم برای روش عادی GAN از روابط زیر برای پیاده سازی الگوریتم استفاده می کنیم:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}}[\log D(x)] + \mathbb{E}_{z\sim p_z}[\log(1-D(G(z)))]$$

فرآیند گفته شده در بالا با بیشینه و کمینه کردن رابطه بالا پیاده سازی می شود. همانطور که در بخش قبلی گفته شد یکی از مهم ترین نوآوری های این روش conditional augmentation میباشد که سبب بهبود نتایج به دست آمده از این روش می گردد. ا استفاده از این روش داده های تولید شده در بخش مولد مشروط بر اطلاعات استخراج شده در بخش embedding (φ_t)، به صورت نویزهای گوسی با ماتریس های میانگین و واریانس مشخص $N(\mu_0(\varphi_t).\Sigma(\varphi_t))$ تولید میگردد که به طور طبیعی حاوی اطلاعاتی از داده های استخراج شده از متن ها میباشد که بی شک این عمل نقش به سزایی در تولید تصاویر بهتر با جزئیات مرتبط تر دارد.

در بخش قبل گفته شد که از دیگر مشکلاتی که در روش vanilla GAN مشکل ساز است بحث ناپایداری در مورد تغییر در ورود ها می باشد برای حل این مشکل با به کارگیری روش هوشمندانه و با افزودن regularizer که به صورت زیر تعریف می شود robustness مدل به صورت قابل توجهی افزایش می یابد. که این عمل نیز مشروط بر داده های φ_t می باشد که همان Kullback-Leibler divergence می باشد.

$$D_{KL}(\mathcal{N}(\mu(\varphi_t), \Sigma(\varphi_t)) \,||\, \mathcal{N}(0, I))$$

که این مقدار در واقع Kullback-Leibler divergence میان تابع توزیع گوسی استاندارد و تابع توزیع گوسی تولید شده مشروط به داده های مد نظر می باشد. استفاده از تابع خطای اول به صورت مستقل منجر به بازیابی اصل ورودی بدون تغییر خواهد شد در صورتی که ویژگی‌های اصلی در فرایند انکودینگ حذف نشده باشند از طرفی استفاده تنها از تابع هزینه دوم منجر به تولید داده‌ها در فضای نهان ویژگی‌ها بدون هیچ محدودیتی خواهد شد و از آنجا که تمام دامنه مربوط به کلاس خاصی نمی‌باشد، خروجی تولیدی شانس کمتری برای قرارگیری در کلاس خاصی پیدا خواهد کرد بنابراین استفاده همزمان از دو تابع هزینه در کنار هم باعث تولید داده‌های واقعی اطراف میانگین داده‌های ترین و با پراکندگی خاصی متناسب با انحراف معیار ترین شده اطراف آن خواهد شد.

در نهایت معادلات در I_stage به صورت زیر تغییر می یابد:

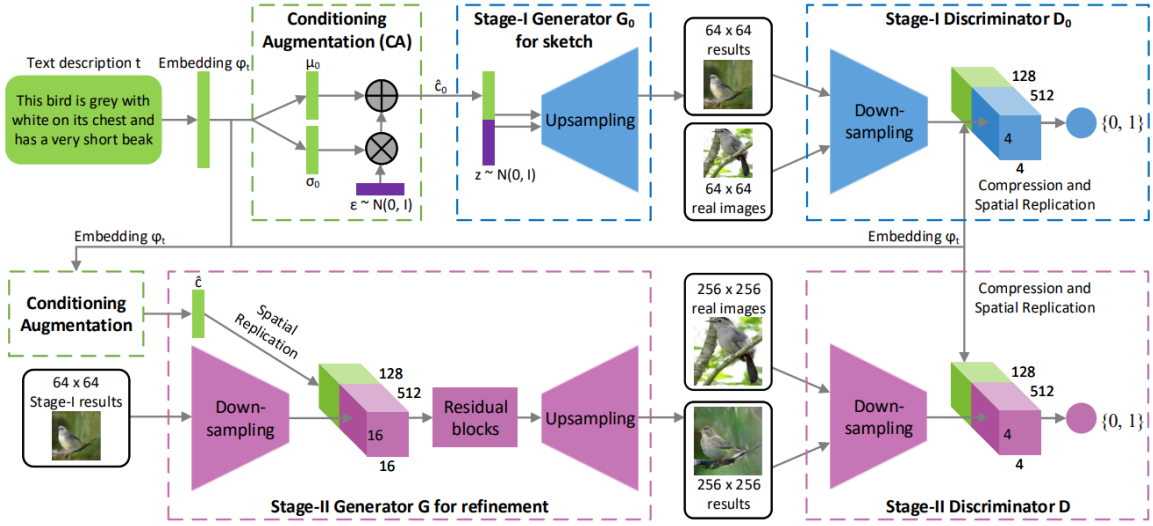
$$\begin{aligned}
\mathcal{L}_{D_0} &= \mathbb{E}_{(I_0,t)\sim p_{data}}[\log D_0(I_0, \varphi_t)] + \\
&\quad \mathbb{E}_{z\sim p_z, t\sim p_{data}}[\log(1 - D_0(G_0(z, \hat{c}_0), \varphi_t))], \\
\mathcal{L}_{G_0} &= \mathbb{E}_{z\sim p_z, t\sim p_{data}}[\log(1 - D_0(G_0(z, \hat{c}_0), \varphi_t))] + \\
&\quad \lambda D_{KL}(\mathcal{N}(\mu_0(\varphi_t), \Sigma_0(\varphi_t)) \,||\, \mathcal{N}(0, I)),
\end{aligned}$$

برای II_stage نیز خواهیم داشت:

$$\mathcal{L}_D = \mathbb{E}_{(I,t)\sim p_{data}}[\log D(I, \varphi_t)] + \mathbb{E}_{s_0\sim p_{G_0}, t\sim p_{data}}[\log(1 - D(G(s_0, \hat{c}), \varphi_t))],$$

$$\mathcal{L}_G = \mathbb{E}_{s_0\sim p_{G_0}, t\sim p_{data}}[\log(1 - D(G(s_0, \hat{c}), \varphi_t))] + \lambda D_{KL}(\mathcal{N}(\mu(\varphi_t), \Sigma(\varphi_t)) \,||\, \mathcal{N}(0, I)),$$

با این فرض های هوشمندانه صورت گرفته در نهایت ساختار مدل به صورت زیر درآمده است:



شکل ۳-۲: ساختار شبکه Stack GAN

مجموعه دادگان :

یکی از مجموعه های مورد استفاده در این پژوهش، داده CUB میباشد که شامل ۱۱۷۸۸ عکس مجزا که از ۲۰۰ گونه پرنده متفاوت است. همچنین مجموعه Oxford-۱۰۲ که شامل ۸۱۸۹ تصویر از ۱۰۲ گونه متفاوت از گربه ها نیز به کار گرفته شده است. برای اثبات کارآیی و عمویت نتایج به دست آمده از این مدل، از مجموعه MS COCO نیز که در آن تصاویر شامل عناصر متفاوت و نیز پس زمینه های متنوع می باشند نیز استفاده شده است. در این مجموعه داده ها که شامل ۵ توضیح متفاوت می باشند کار شناسایی و بازنمایی تصاویر اندکی پیچیده تر از سایر مجموعه های استفاده شده است.

نحوه ارزیابی:

به منظور ارزیابی مدل طراحی شده از یکی روش های نوین تحت عنوان inception score بهره گیری شده است که مطابق زیر تعریف می شود:

$$I = exp\ (E_x D_{KL}(p(y/x)|.|p(y)))$$

که در این رابطه x برابر با عکس تولید شده و y برابر با لیبل مورد نظر برای آن می باشد. به این ترتیب که مقدار KL divergence میان توزیع p(y) و نیز توزیع شرطی p(y/x) باید مقدار بالایی باشد. از آنجایی که این معیار نمی تواند میزان تطبیق جملات با تصاویر را بسنجد، در این پژوهش از ارزیابی افراد و متخصصین نیز بهره گرفته شده است.

نتایج و تحلیل :

در انتها برای نمایش کارایی مدل نتایجی ارائه شده است. در بخش ابتدایی نتایج حاصل از stack GAN در مقایسه با سایر روش ها در شکل زیر دیده می شود:

شکل ۳-۳: مقایسه نتایج خروجی از Stack GAN با سایر روش های مربوط

مشاهده می شود که تصاویر تولید شده در این روش،عکس ها هم شامل جزئیات بیشتری از پرندگان می باشد، همچنین علاوه بر وضوح بالا به تصاویر واقعی نزدیک تر بوده و از چشم ناظران به داده های واقعی شبیه ترند. در جدول زیر نیز برای دادگان گفته شده در بالا نتایج حاصل از معیار گفته شده و نظر ارزیابان بیرونی مشاهده می شود.

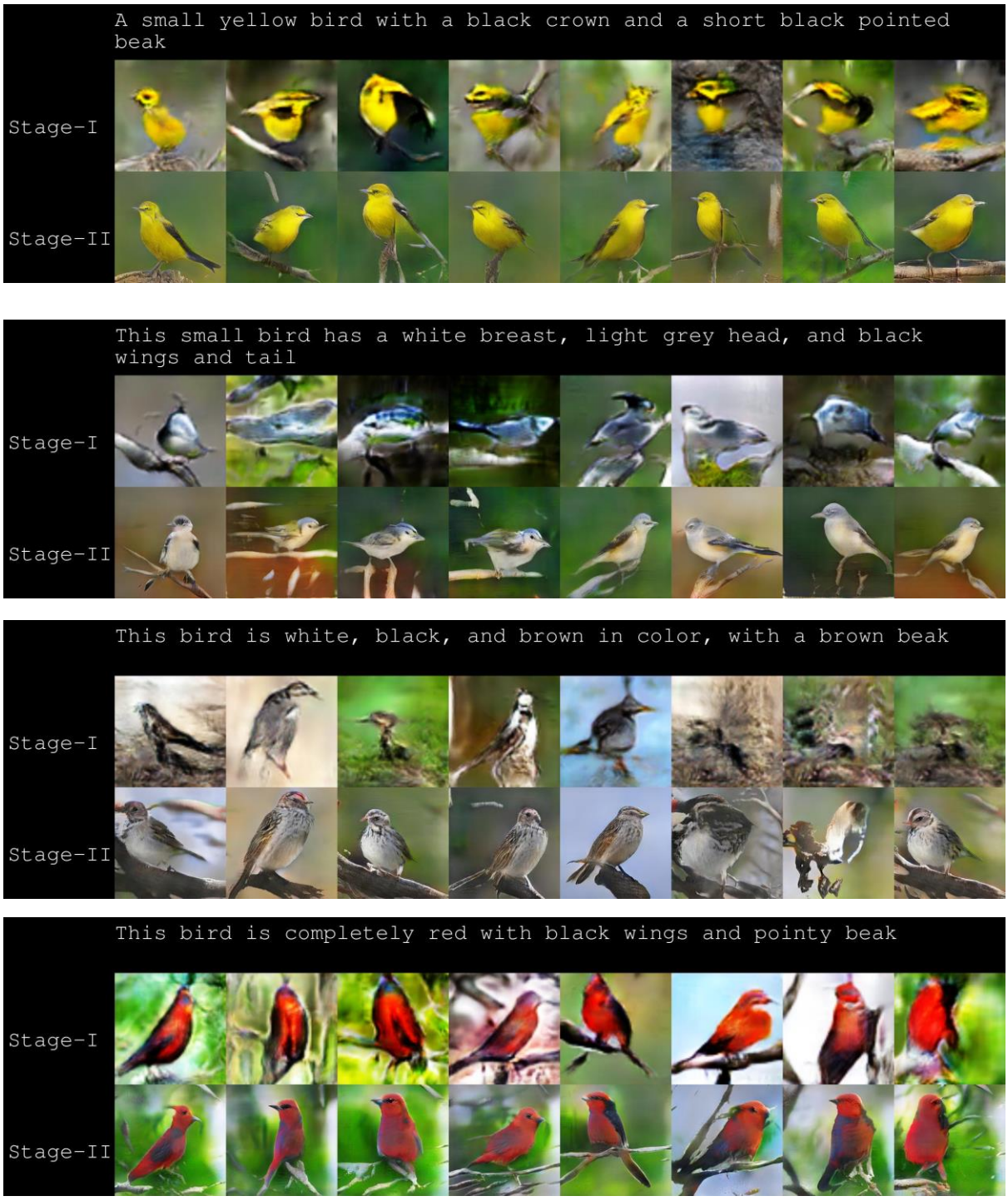
جدول ۳-۱: نتایج عددی حاصل از روش Stack GAN

Metric	Dataset	GAN-INT-CLS	GAWWN	Our StackGAN
Inception score	CUB	$2.88 \pm .04$	$3.62 \pm .07$	**$3.70 \pm .04$**
	Oxford	$2.66 \pm .03$	/	**$3.20 \pm .01$**
	COCO	$7.88 \pm .07$	/	**$8.45 \pm .03$**
Human rank	CUB	$2.81 \pm .03$	$1.99 \pm .04$	**$1.37 \pm .02$**
	Oxford	$1.87 \pm .03$	/	**$1.13 \pm .03$**
	COCO	$1.89 \pm .04$	/	**$1.11 \pm .03$**

مشاهده می شود که مقدار Inception Score برای تمامی مجموعه های گفته شده به صورت چشمگیری افزایش یافته است که از تصاویر بالا نیز به خوبی قابل انتظار است.

نتایج خروجی از الگوریتم:

در این بخش برخی دیگر از نتایج به دست آمده از الگوریتم را در ادامه بررسی و مطالعه می کنیم:



شکل ۳-۴: نتایج حاصل از شبکه Stack GAN بر روی داده های مربوط به پرندگان