فهرست گزارش سوالات (لطفاً پس از تکمیل گزارش، این فهرست را به‌روز کنید.)

سوال 1 – Variational Autoencoder

توضیحات ساز و کار و معماری شبکه VAE :

همانگونه در متن مربوط به سوالات هم اشاره شده بر خلاف شبکه Autoencoder معمول که از فضای اولیه به ثانویه یک نگاشت و برای بازگشتن از فضای ثانویه به اولیه نیز یک نگاشت مشخص برای هر sample داریم ، در VAE در فضای ثانویه (Latent) که ما در این سوال دو بعدی در نظر گرفته ایم از یک distribuition به شرط مشاهده ی فضای sample ورودی استفاده می کنیم. بدین صورت که با فرض توزیع گوسی در این فضا ، به کمک نمونه گیری Mean و std deviation را برای ساخت z بکار می گیریم.

$$\text{latent} = mean + \exp(0.5 \times \text{var}\,iance) \times eps$$

که در رابطه فوق epsilon پارامتر رندوم نویزی ما را نشان می دهد.

حال در قسمت decoder در نظر گرفته می شود که متغیر رندوم z به مدل generative ما داده می شود و هدف ما ماکسیمم کردن لایکلیهود نقاط دیده شده می باشد(با توجه به سمپل های ورودی.)

در این قسمت مشاهده نویزی بصورت زیر تعریف می نماییم.

$$x' = G_\theta(z)$$

$$P_\theta(x \mid z) = N(x; G_\theta(z), \xi I)$$

با توجه به توزیع توام x و z می نویسیم.

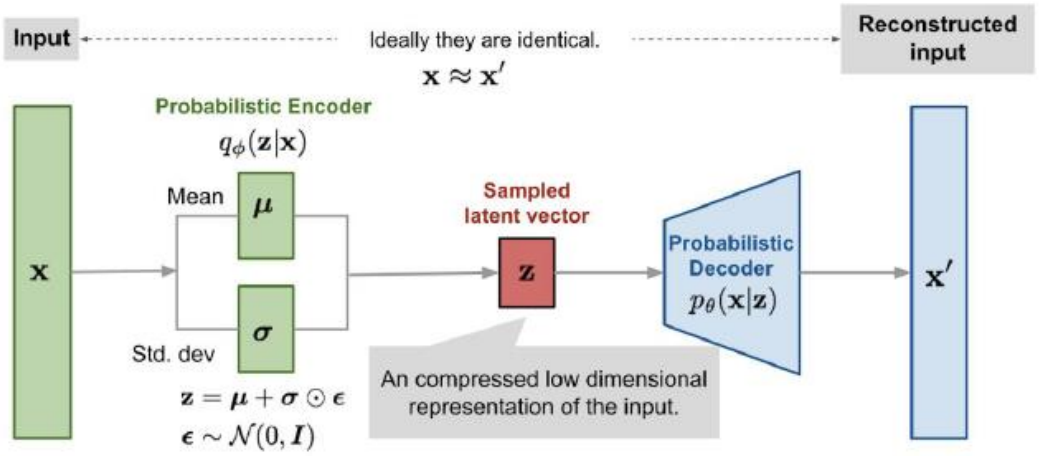
$$p(x) = \int p(z)\, p(x \mid z) dz$$

توزیع z بصورت prior گوسی فرض می شود. برای X توزیع $P_\theta(x \mid z)$ را داریم و در پروسه ترینیگ سعی داریم با بدست آوردن تقریبی از آن در قسمت Decoder به تولید داده بپردازیم. این کار توسط پیدا کردن یک حد پایین از $\log p_\theta(x)$ انجام می شود. در مقاله ی ارجاع داده شده ثابت می شود که داریم :

$$\log p_\theta(x) > D_{KL}(q \,||\, p)$$

$$L_{\theta,q}(x) = E_z . \log \frac{P_\theta(x, z)}{q_\theta(z \mid x)}$$

با توجه به رابطه فوق Loss کل را مجموع loss در قسمت reconstruction و D(KL) که همان kl می باشد در نظر گرفته می شود.(این قسمت بطور کامل تر در بخش ب و ج توضیح داده شده است.)



شکل ۱ـ۱) معماری شبکه VAE

الف) مطابق توضیحات داده شده در بخش ابتدایی در این قسمت می خواهیم ساختار شبکه به کار رفته در کد را مورد بررسی قرار دهیم.

بعد از اعمال preprocess معمول آرایش زیر را برای قسمت encoder داریم.

Layer (type)	Output Shape	Param #	Connected to
input_5 (InputLayer)	[(None, 28, 28, 1)]	0	
conv2d_5 (Conv2D)	(None, 14, 14, 32)	320	input_5[0][0]
conv2d_6 (Conv2D)	(None, 7, 7, 64)	18496	conv2d_5[0][0]
flatten_2 (Flatten)	(None, 3136)	0	conv2d_6[0][0]
dense_3 (Dense)	(None, 16)	50192	flatten_2[0][0]
mean (Dense)	(None, 2)	34	dense_3[0][0]
log_variance (Dense)	(None, 2)	34	dense_3[0][0]
lambda_2 (Lambda)	(None, 2)	0	mean[0][0] log_variance[0][0]

Total params: 69,076
Trainable params: 69,076
Non-trainable params: 0

شکل ۱ـ۲) آرایش بکار رفته در قسمت Encoder

شکل ۱ـ۲) نشان می دهد که بعد از گرفتن ورودی در قسمت input layer به وسیله ی دو لایه ی Conv2D و در نهایت flatten کردن خروجی این قسمت از یک لایه ی Dense می گذرانیم.در این مرحله توسط تابع تعریف شده sample_latent_feature مطابق توضیحات داده شده در بخش ابتدایی به ساخت بردار z ساخته شده از میانگین و واریانس در این فضا می پردازیم. خروجی encoder فضای latent ما را تشکیل می دهد.

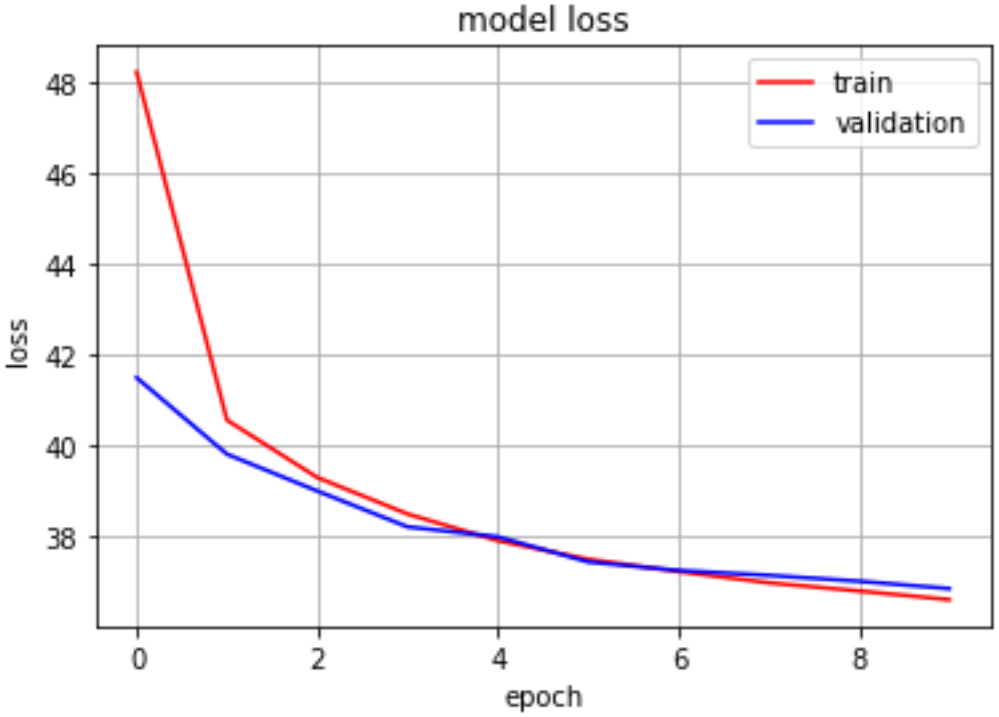
برای قسمت Decoder مطابق شکل ۱ـ۳ داریم :

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_6 (InputLayer)         [(None, 2)]               0
_________________________________________________________________
dense_4 (Dense)              (None, 3136)              9408
_________________________________________________________________
reshape_1 (Reshape)          (None, 7, 7, 64)          0
_________________________________________________________________
conv2d_transpose_4 (Conv2DTr (None, 14, 14, 64)        36928
_________________________________________________________________
conv2d_transpose_5 (Conv2DTr (None, 28, 28, 32)        18464
_________________________________________________________________
conv2d_transpose_6 (Conv2DTr (None, 28, 28, 1)         289
=================================================================
Total params: 65,089
Trainable params: 65,089
Non-trainable params: 0
_________________________________________________________________
```

شکل ۱ـ۳) آرایش بکار رفته در قسمت Decoder

مطابق شکل ۱ـ۳) توسط Conv2Dtranspose سعی در تنظیم پارامترها به نحوی داریم که به فرم ورودی بازگردیم این کار توسط reshape کردن در لایه ها صورت گرفته است.خروجی لایه ی آخر نشان می دهد که از نظر ابعاد به ورودی بازگشته ایم.در ادامه نتایج این پیاده سازی را مورد بررسی قرار می دهیم.

نمودار loss در طی ۱۰ ایپاک بصورت شکل ۱ـ۴ می باشد.



شکل ۱ـ۴) نمودار loss در هر ایپاک

ب)

$$\log p_\theta(x) = E_z(\log \frac{P_\theta(x,z)}{q_\varphi(z \mid x)}) + E(\log \frac{q_\varphi(x,z)}{P_\theta(z \mid x)})$$

تابع هزینه ی کل (در کد total_Loss) مجموع loss مربوط به $D_{KL}(q \| p)$ که نشان دهنده ی نزدیکی توزیع p و نزدیکی آن به تخمین q از آن می باشد و ترم دوم که از بخش دیکودر و مربوط به maximum کردن لایکلیهود ساخته شده می باشد تشکیل شده است. حضور ترم اول باعث می شود که تخمین بکار رفته بعنوان حد پایین برای log liklihood در قسمت انکودر Max شده و در نتیجه فضای latent در نظر گرفته شده تا حد امکان به خروجی انکودر نزدیک باشد..

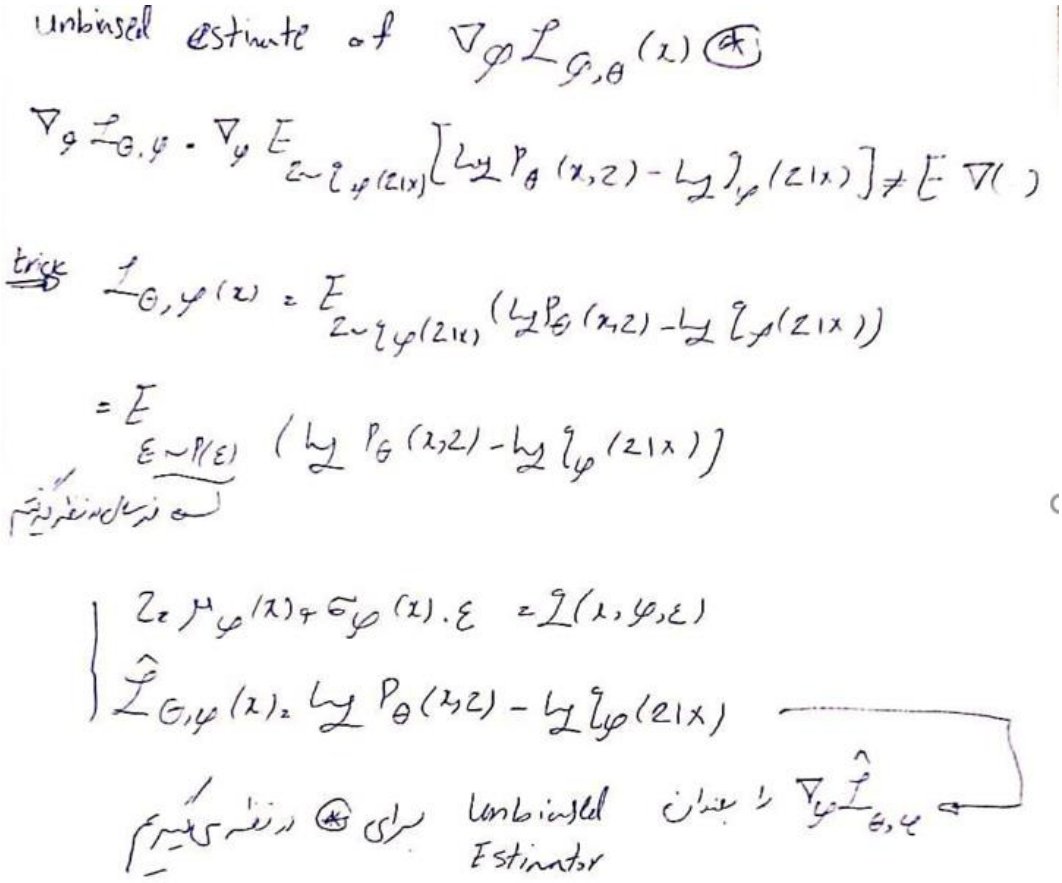
ج)

در قسمت محاسبه back propagation و آپدیت کردن وزن های شبکه ماهیت رندونس فضای z برای ما مسئله ایجاد کرده و نمی دانیم چگونه وزن های مربوط به آن را آپدیت کنیم(ترم های مشتق گیری). برای این کار از یک trick محاسباتی استفاده می شود که در ادامه شرح داده می شود که به آن parametrization trick گفته می شود.

با توجه به ترم ارور میان x, x' گرادیان را نسبت به θ, φ تشکیل می دهیم. (unbiased)

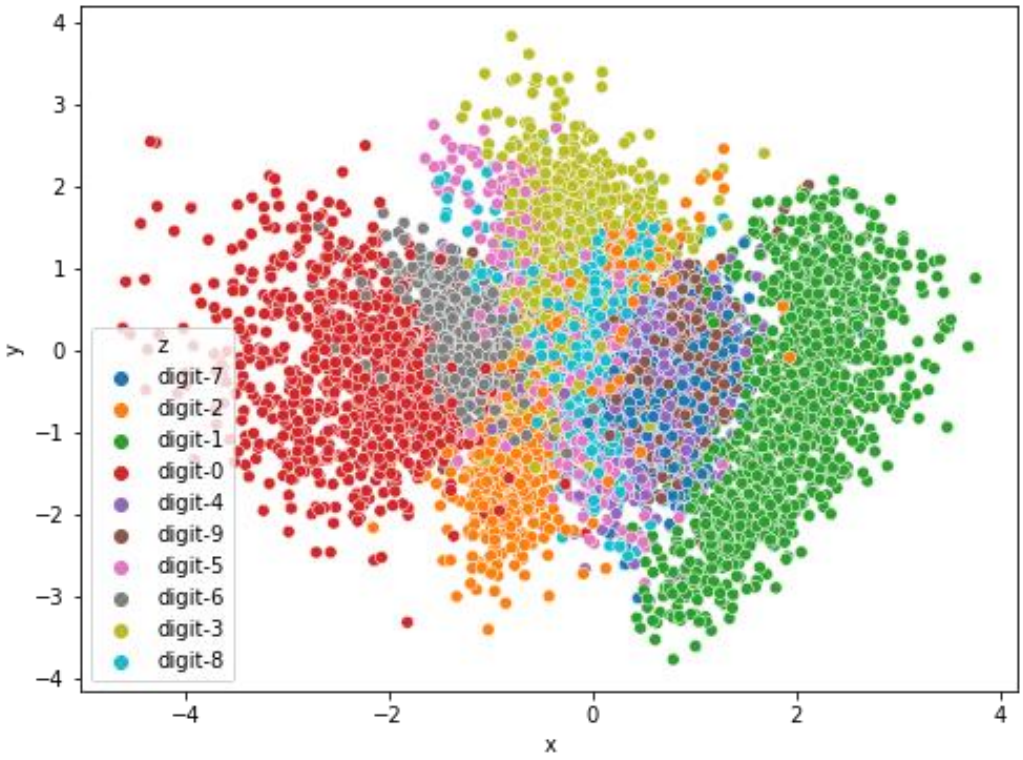
$$\nabla_\theta L_{\theta,\varphi}(x) = \nabla_\theta E[\log P_\theta(x,z) - \log q_\varphi(z \mid x)]$$

که در عبارت فوق می توان ترم گرادیان را داخل expectation نمود. با نوشتن ترم گرادیان برای φ اما این کار مجاز نیست چرا که در هر تکرار یک eps رندوم انتخاب شده است .در اینجا به کمک parametrization trick این مسئله را حل می نماییم.(این بخش بعلت طولانی بودن بصورت عکس ضمیمه می شود.)



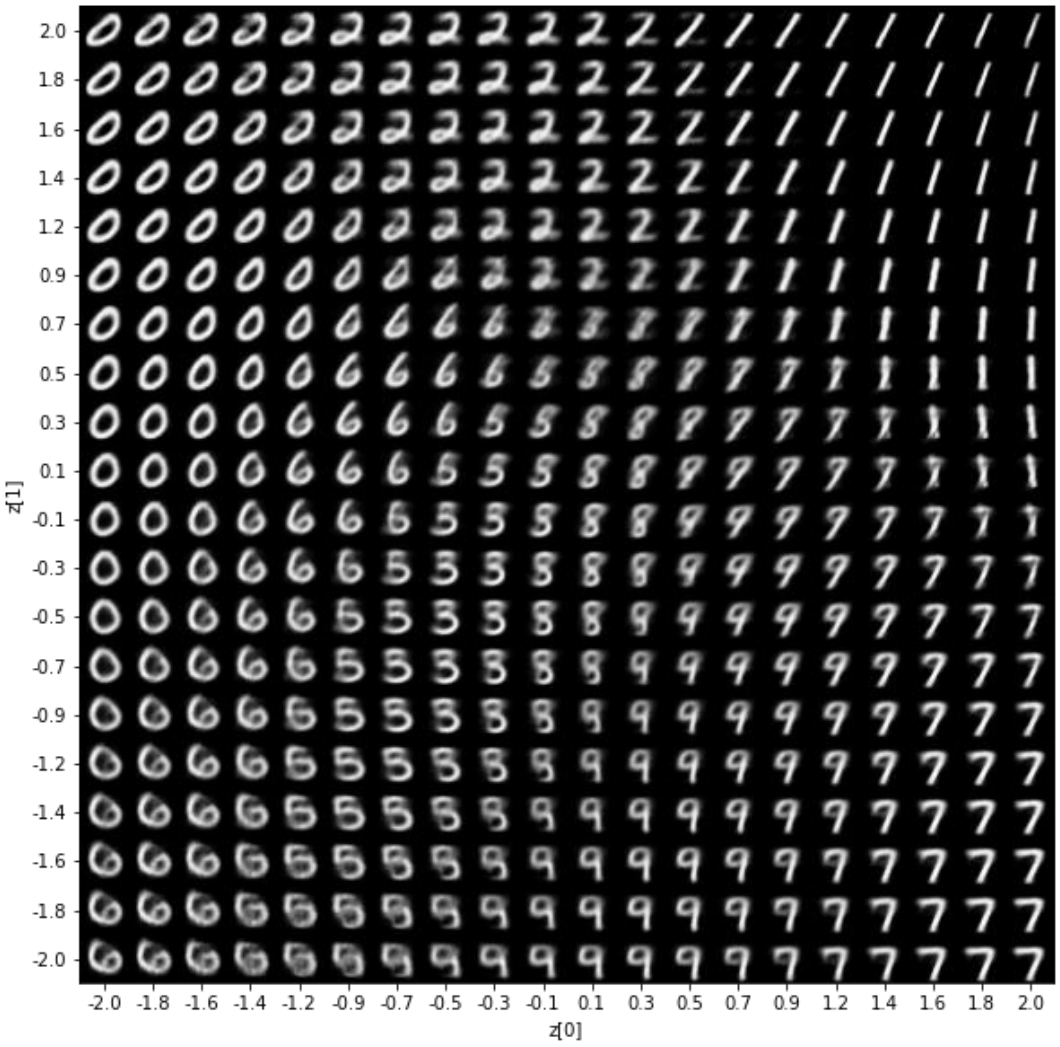
تصویر ۱) نحوه اعمال reparametrization trick برای محاسبه ی وزن ها

د) داده های ترین Mnist را که به فضای z یا latent برده شده اند توسط scatterplot با توجه به کلاس رقم مربوط به آن در شکل ۱ـ۵ آورده ایم.



شکل ۱ـ۵)نحوه ی پخش هر کلاس در فضای z

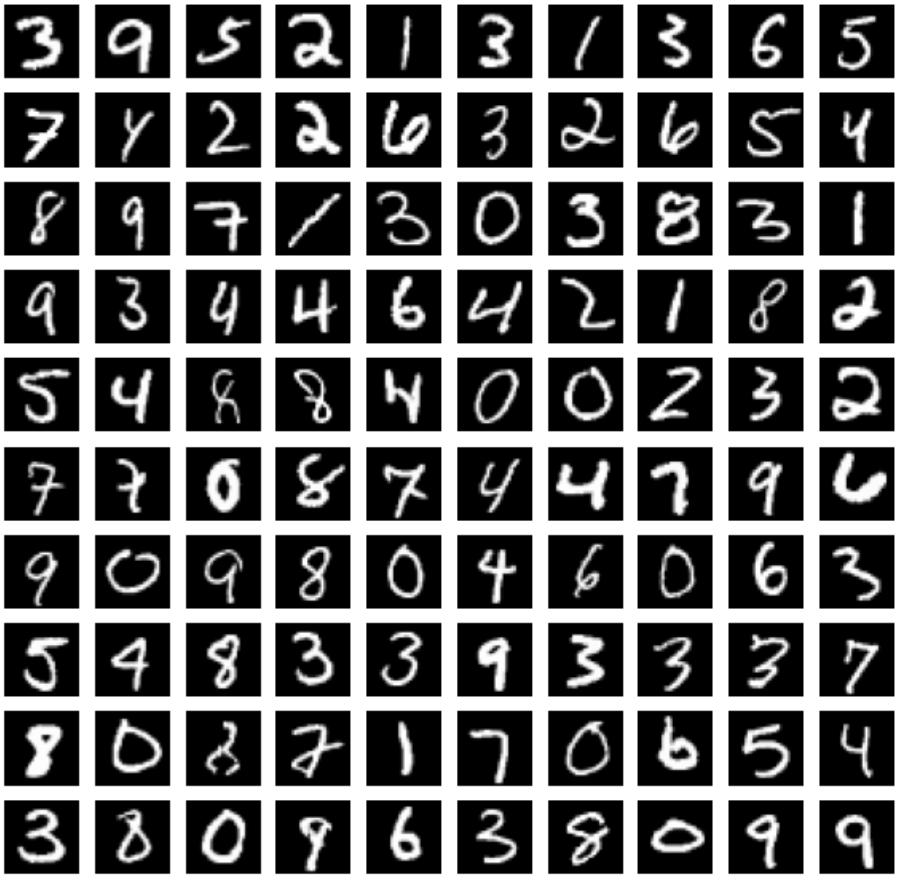
ه)با توجه به شکل ۱ـ۵)گریدی را در نظر می گیریم به نوعی که بتواند عمده ی فضای پوشش داده شده در صفحه z را پوشش دهد.برای اینکار از تابعی plot_latent_space استفاده شده است.با توجه به شکل ۱ـ۵ گرید مربعی از ۲- تا ۲+ را در نظر می گیریم.نتیجه ی آن برای یک گرید ۲۰ در ۲۰ به فرم شکل ۱ـ۶ می باشد.



شکل ۱ـ۶) نمایش یک گرید در فضای z بصورت مربعی

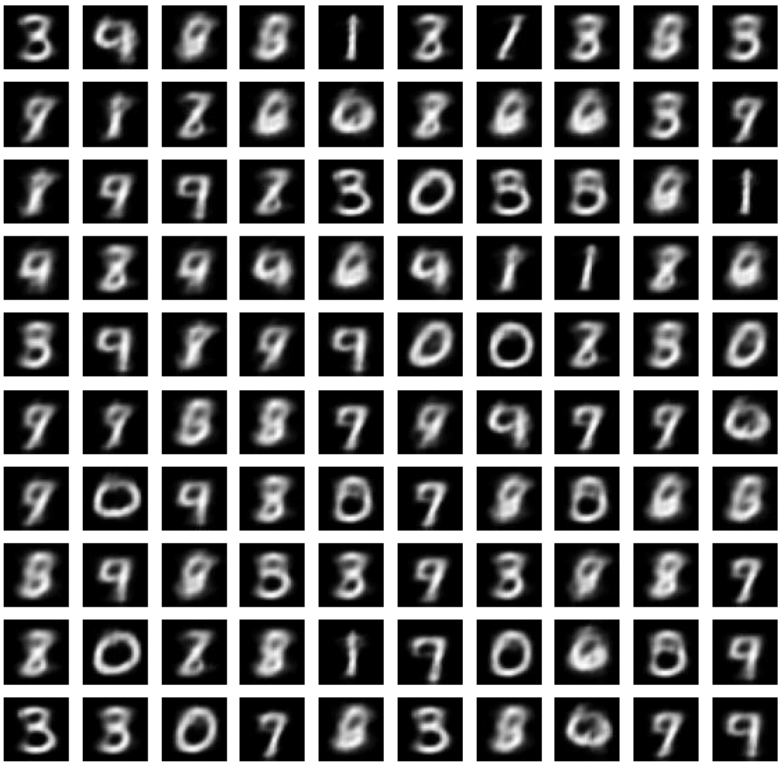
و)

ابتدا تصویر اوریجینال ۱۰ در ۱۰ استفاده شده را می آوریم.

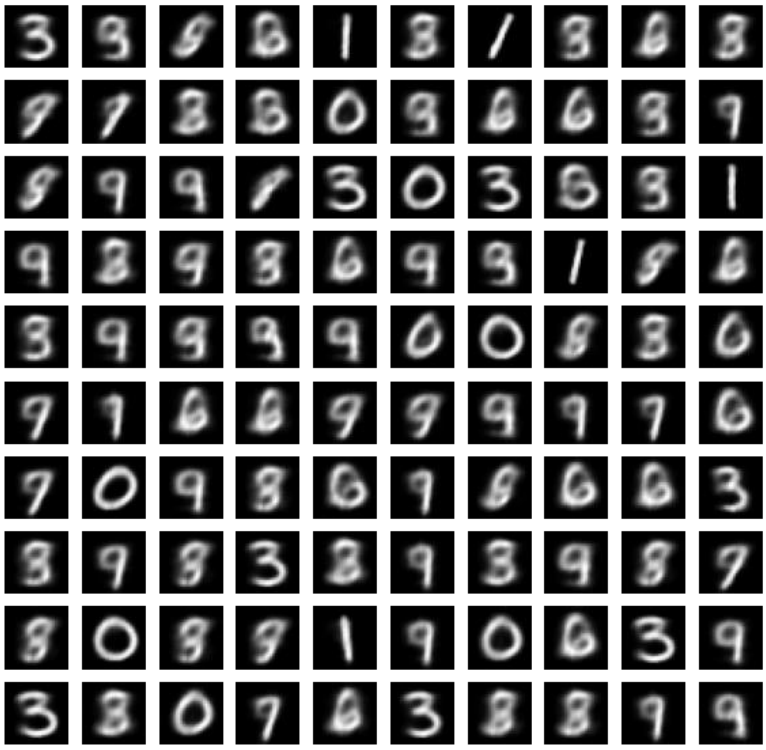


شکل ۱ـ۷) تصویر original

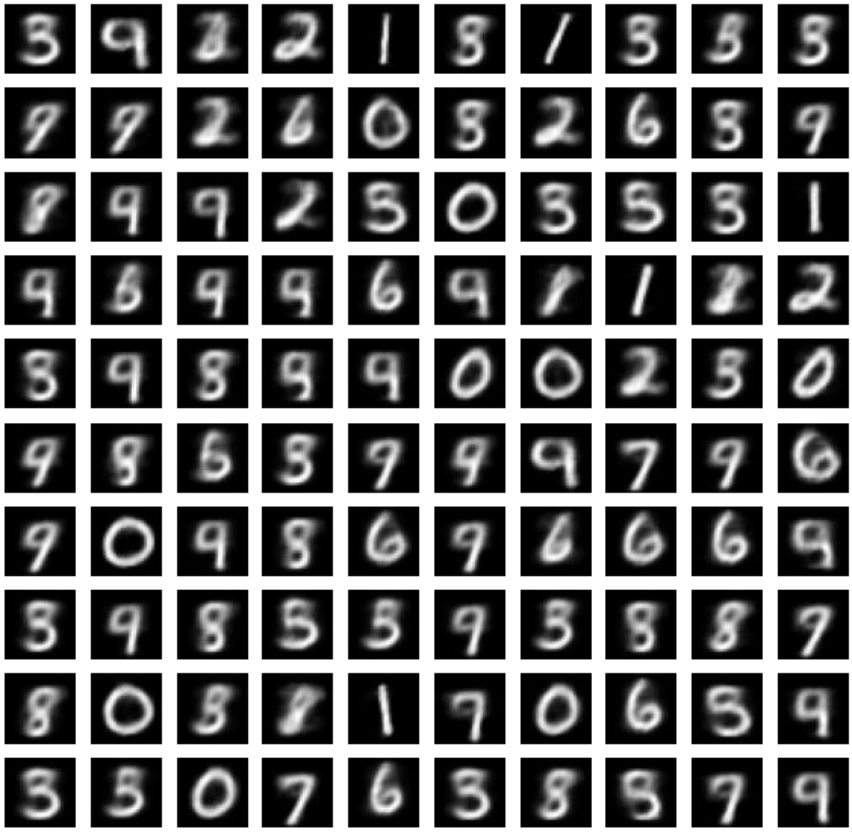
در ادامه تصاویر بازیابی شده در ایپاک های به ترتیب ۱و۲و ۴و۵و۱۰ آورده می شوند. برای این کار از یک callback برای ذخیره کردن وزن ها در ایپاک های ذکر شده استفاده شده است.



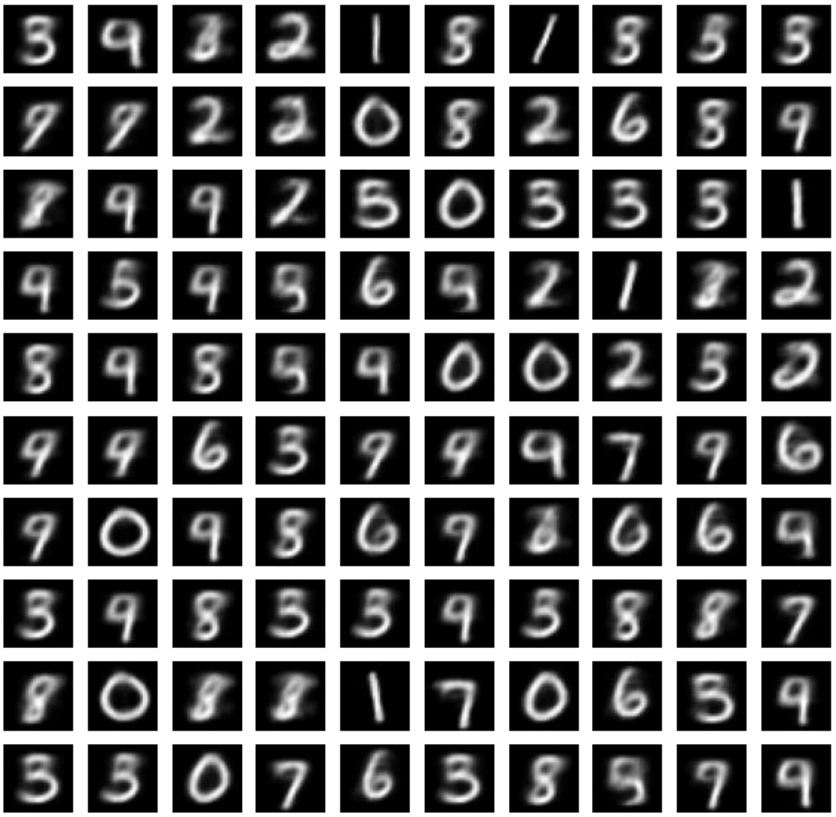
شکل ۱ـ۸) تصاویر ساخته شده بعد از اولین ایپاک



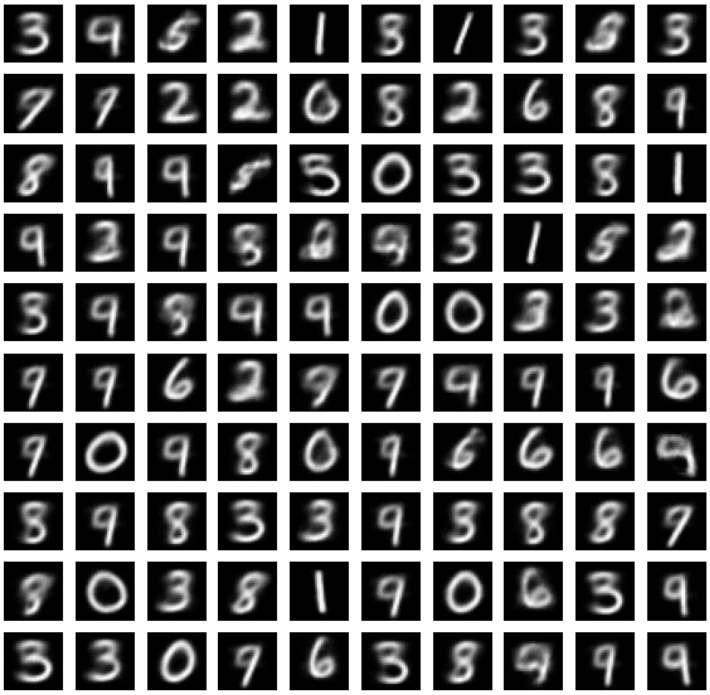
شکل ۱ـ۹) تصاویر ساخته شده بعد از دومین ایپاک



شکل ۱ـ۱۰) تصاویر ساخته شده بعد از جهارمین ایپاک



شکل ۱ـ۱۱) تصاویر ساخته شده بعد از پنجمین ایپاک

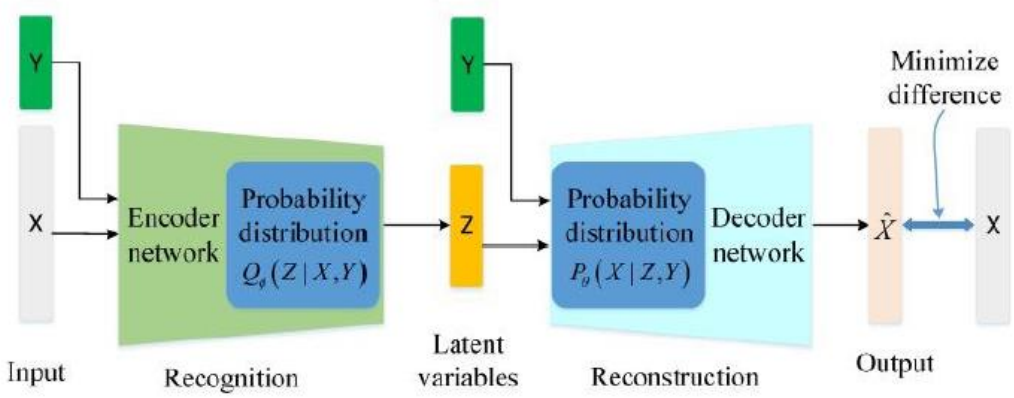


شکل ۱ـ۱۲) تصاویر ساخته شده بعد از دهمین ایپاک

بعد از دهمین ایپاک دیگر تغییر محسوسی در loss صورت نمی گیرد. با توجه به اشکال فوق روند بهبود بخصوص در بعضی از درایه ها مثل سومین درایه که در آخرین شکل به صورت خوبی به 5 تغییر یافته شده مشخص است.

Conditional VAE:

در این بخش ابتدا با در نظر گرفتن همان ساختار بکار رفته در شبکه قبلی ، به تفاوت های این دو ساختار می پردازیم. همانگونه که در شکل ۱ـ۱۳ مشخص است Y که لیبل هر سمپل هست در هر دو بخش encoder و decoder بصورت one_hot کد شده و در کنار بردار x و z اعمال می شود.این کار باعث می شود که تمامی توزیع های توصیف شده در VAE نیز بصورت شرطی درآیند که می تواند به ما در جهت داشتن کنترل بهتر بر روی خروجی generator و از بین بردن نویز کمک کند. در بخش بعد معماری بکار رفته در این قسمت را بررسی می نماییم.



شکل ۱ـ۱۳) ساختار CVAE

ز)

بعد از اعمال preprocess معمول آرایش زیر را برای قسمت encoder داریم.

Layer (type)	Output Shape	Param #	Connected to
input_16 (InputLayer)	[(None, 784)]	0	
input_17 (InputLayer)	[(None, 10)]	0	
concatenate_10 (Concatenate)	(None, 794)	0	input_16[0][0] input_17[0][0]
dense_35 (Dense)	(None, 512)	407040	concatenate_10[0][0]
dense_36 (Dense)	(None, 256)	131328	dense_35[0][0]
dense_37 (Dense)	(None, 2)	514	dense_36[0][0]
dense_38 (Dense)	(None, 2)	514	dense_36[0][0]

شکل ۱ـ۱۴) ساختار encoder بکار رفته

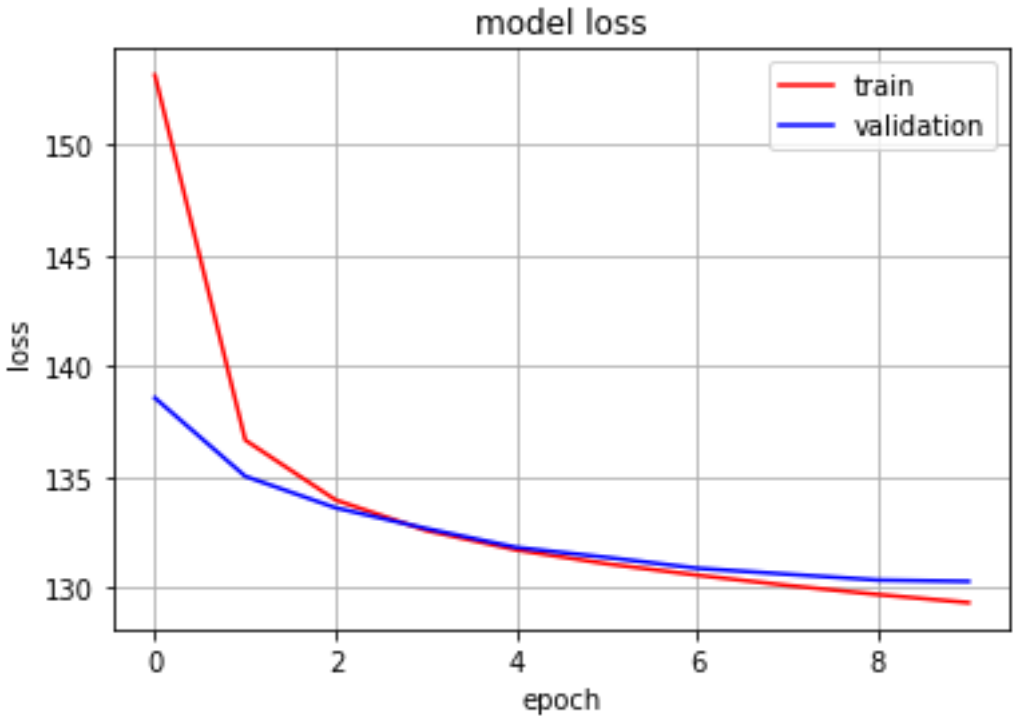
در این قسمت ابتدا ورودی flatten شده با لیبل concatenate شده و در ادامه به کمک لایه های Dense ابتدا ابعاد کاهش پیدا کرده و سپس به کمک همان sampling توضیح داده شده در قسمت قبلی داده ها را به فضای latent دو بعدی می بریم.

در بخش دیکودر نیز در ابتدا لیبل را با بردار z در برای هر بچ concatenate کرده و در ادامه توسط لایه های Dense سعی در بازسازی ورودی تغییر یافته داریم.لازم به ذکر است که از همان تابع loss بکار رفته برای VAE استفاده شده است.

tf.convert_to_tensor_15 (TFOpLa	(50, 784)	0	dense_41[0][0]
tf.math.subtract_10 (TFOpLambda	(None, 2)	0	tf.__operators__.add_10[0][0] dense_38[0][0]
tf.keras.backend.binary_crossen	(50, 784)	0	tf.cast_5[0][0] tf.convert_to_tensor_15[0][0]
tf.math.subtract_11 (TFOpLambda	(None, 2)	0	tf.math.subtract_10[0][0]
tf.math.reduce_mean_5 (TFOpLamb	(50,)	0	tf.keras.backend.binary_crossentr
tf.math.reduce_sum_5 (TFOpLambd	(None,)	0	tf.math.subtract_11[0][0]
tf.math.multiply_10 (TFOpLambda	(50,)	0	tf.math.reduce_mean_5[0][0]
tf.math.multiply_11 (TFOpLambda	(None,)	0	tf.math.reduce_sum_5[0][0]
tf.__operators__.add_11 (TFOpLa	(50,)	0	tf.math.multiply_10[0][0] tf.math.multiply_11[0][0]
add_loss_5 (AddLoss)	(50,)	0	tf.__operators__.add_11[0][0]

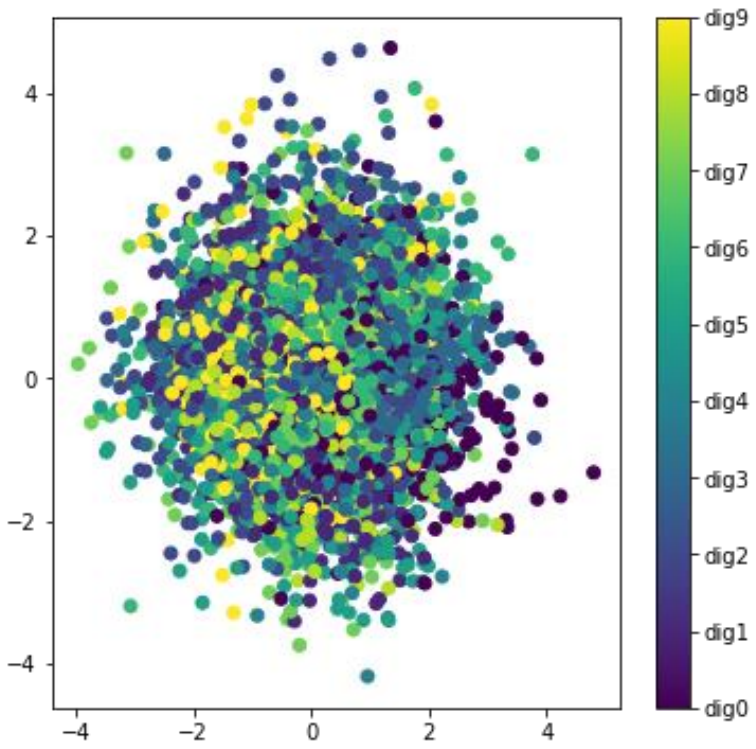
شکل ۱ـ۱۵) ساختار Decoder بکار رفته

در ادامه نتایج حاصل از پیاده سازی این شبکه در ده ایپاک بیان میگردد.



شکل ۱ـ۱۶)نمودار loss بر حسب ایپاک

ح) داده های ترین Mnist را که به فضای z یا latent برده شده اند توسط scatterplot با توجه به کلاس رقم مربوط به آن در شکل ۱ـ۱۷ آورده ایم.

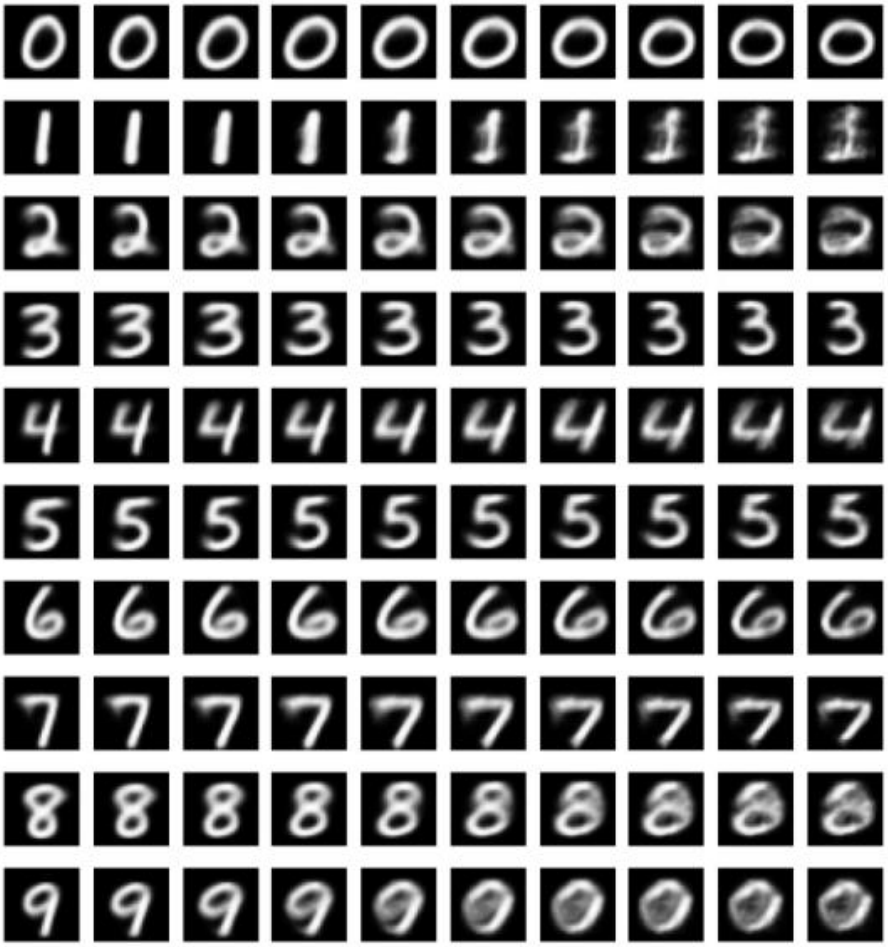


شکل ۱ـ۱۷) نحوه پخش داده های مرتبط بل هر کلاس در فضای z

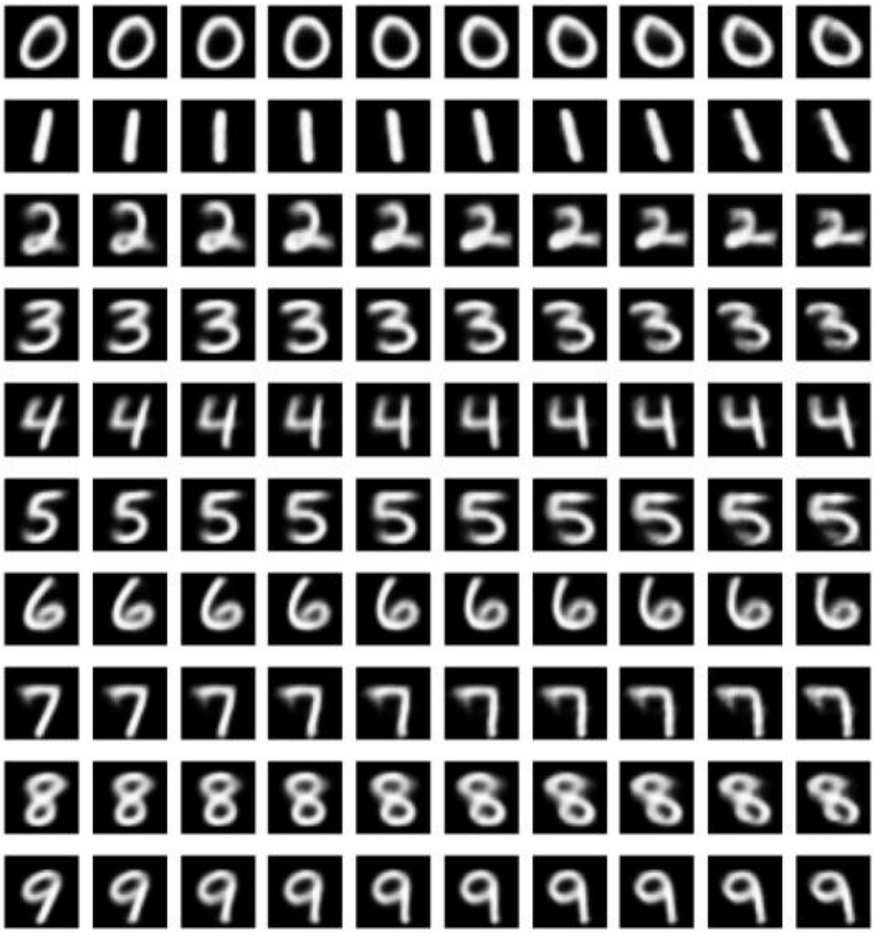
با توجه به شکل فوق همانطور که بیان شد برای داشتن کنترل بهتر بر روی خروجی تولید شده از CVAE استفاده نمودیم. با بررسی و مقایسه ی این شکل و شکل قسمت دال مشخص است که در Cvae داده های کلاس ها در هم تنیده هستند.علت آن هست که با توجه به در نظر گرفتن و وارد کردن لیبل هر عکس در ورودی انکودر و دیکودر داده های هر کلاس را جداگانه بررسی می نماییم و نیازی به جداسازی آن ها در فضای z نداریم.

در ادامه نحوه تولید داده برای هر رقم بررسی شده است. این تغییرات با توجه به شکل ۱ـ۱۷ برای هر رقم و در دو راستای x و y در نظر گرفته می شوند.

ط) برای نتیجه گیری بهتر این قسمت برای تمامی عدد ها آورده شده است.

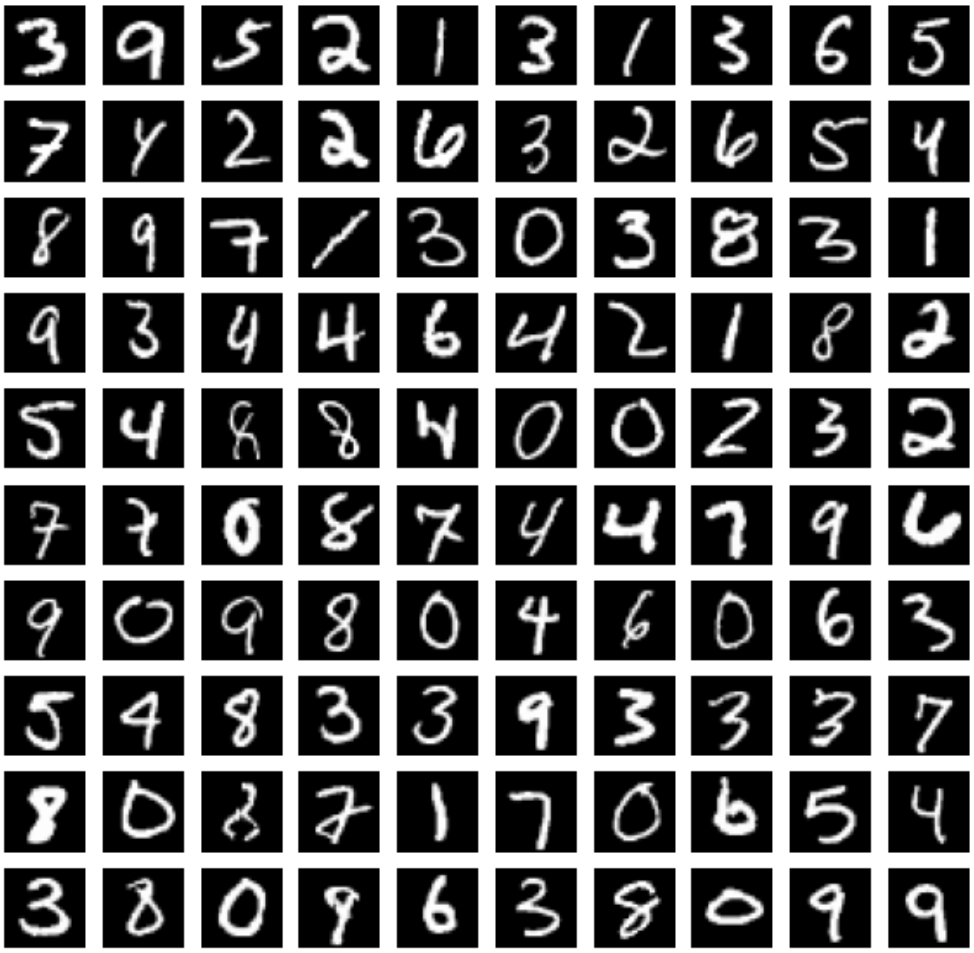


شکل ۱ـ۱۸) تغییرات در صفحه z و در راستای x برای هر رقم



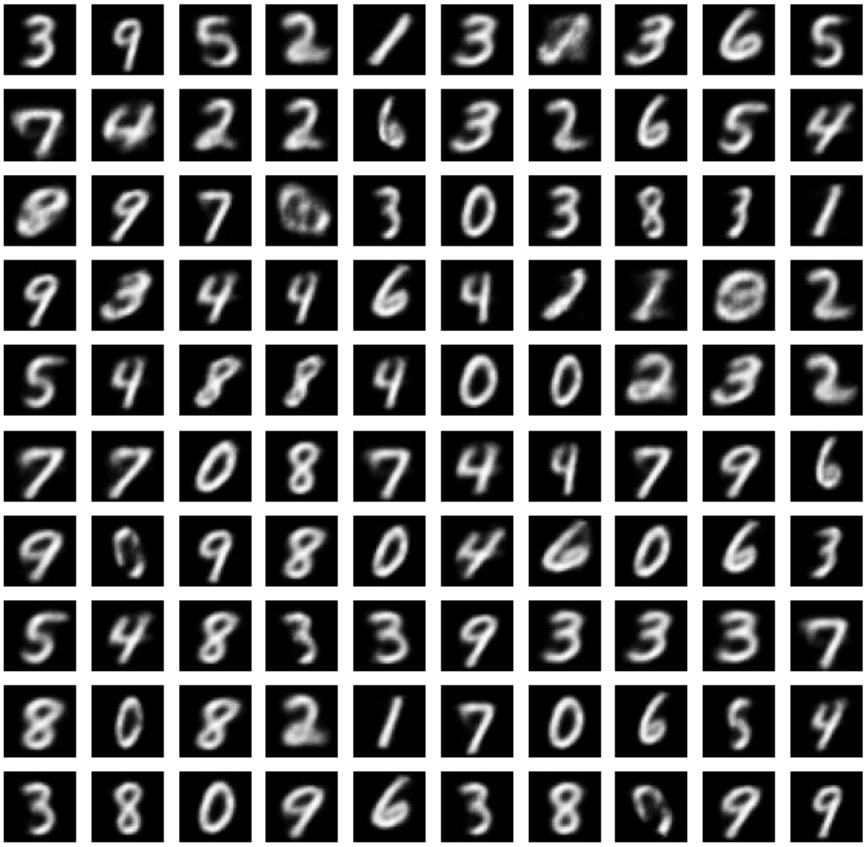
شکل ۱ـ۱۹) تغییرات در صفحه z و در راستای y برای هر رقم

ی) ابتدا تصویر اوریجینال بکار رفته در این بخش را می آوریم.

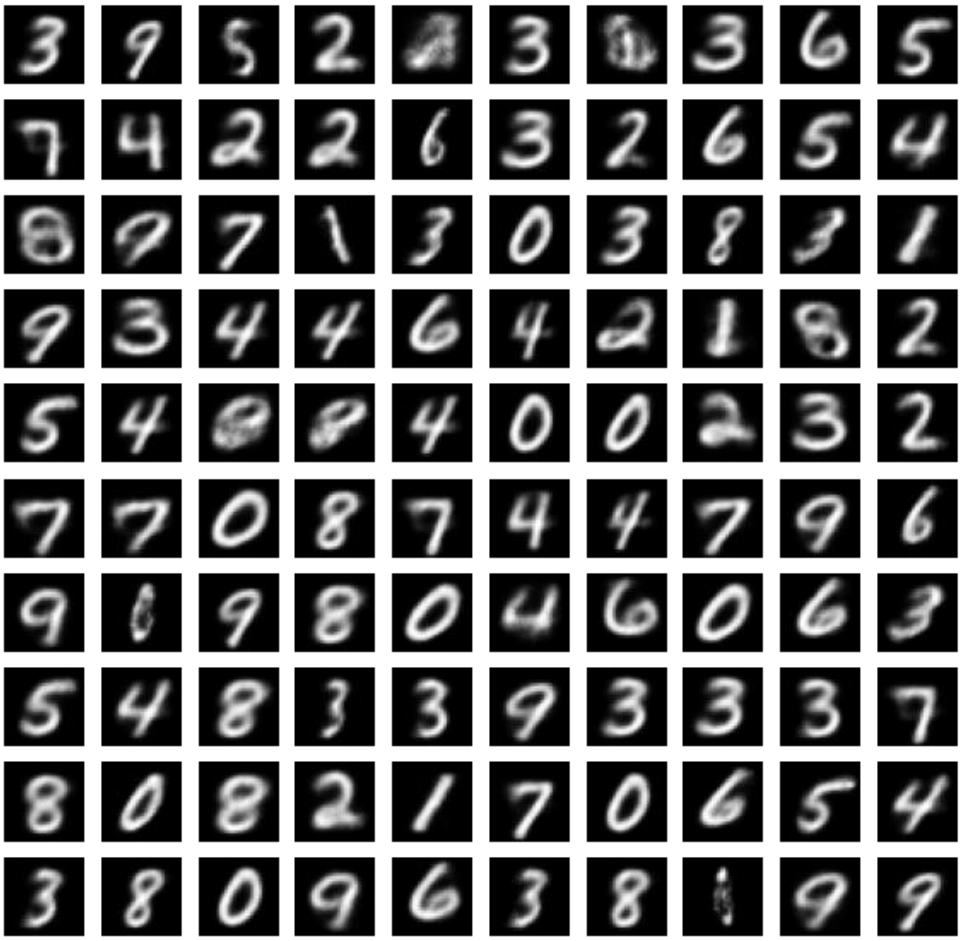


شکل ۱ـ۲۰) تصویر Original

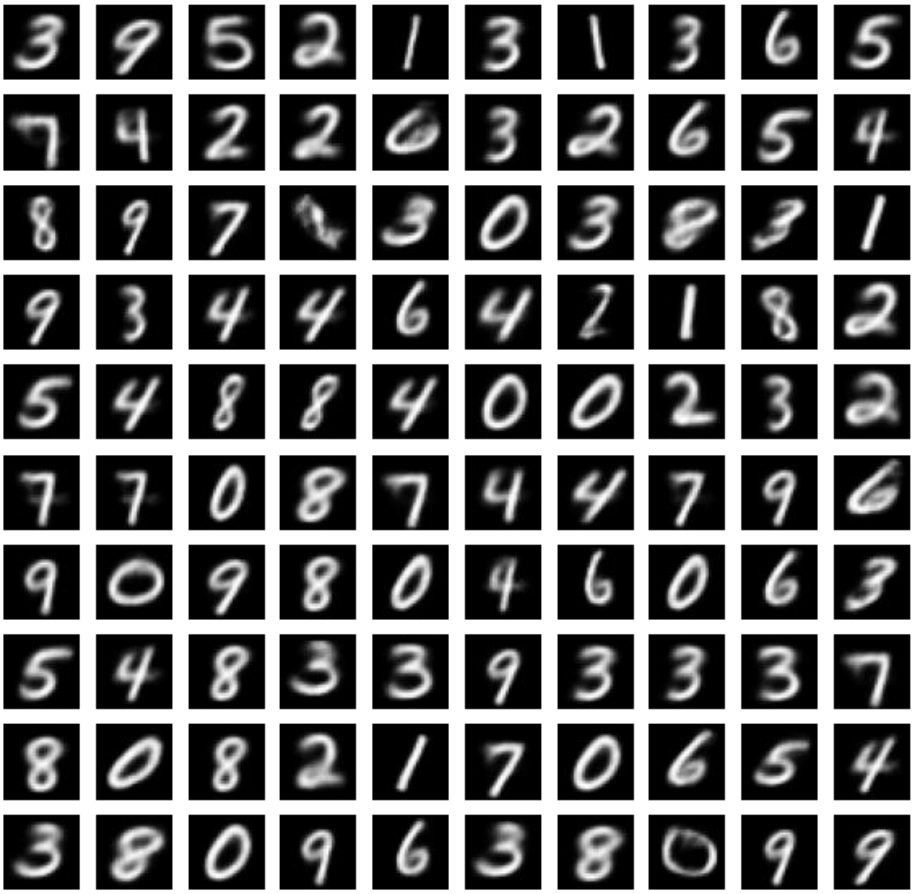
حال نتایج تصاویر ساخته شده در ایپاک های ۱و۲و۴و۵و۱۰ آورده می شوند.



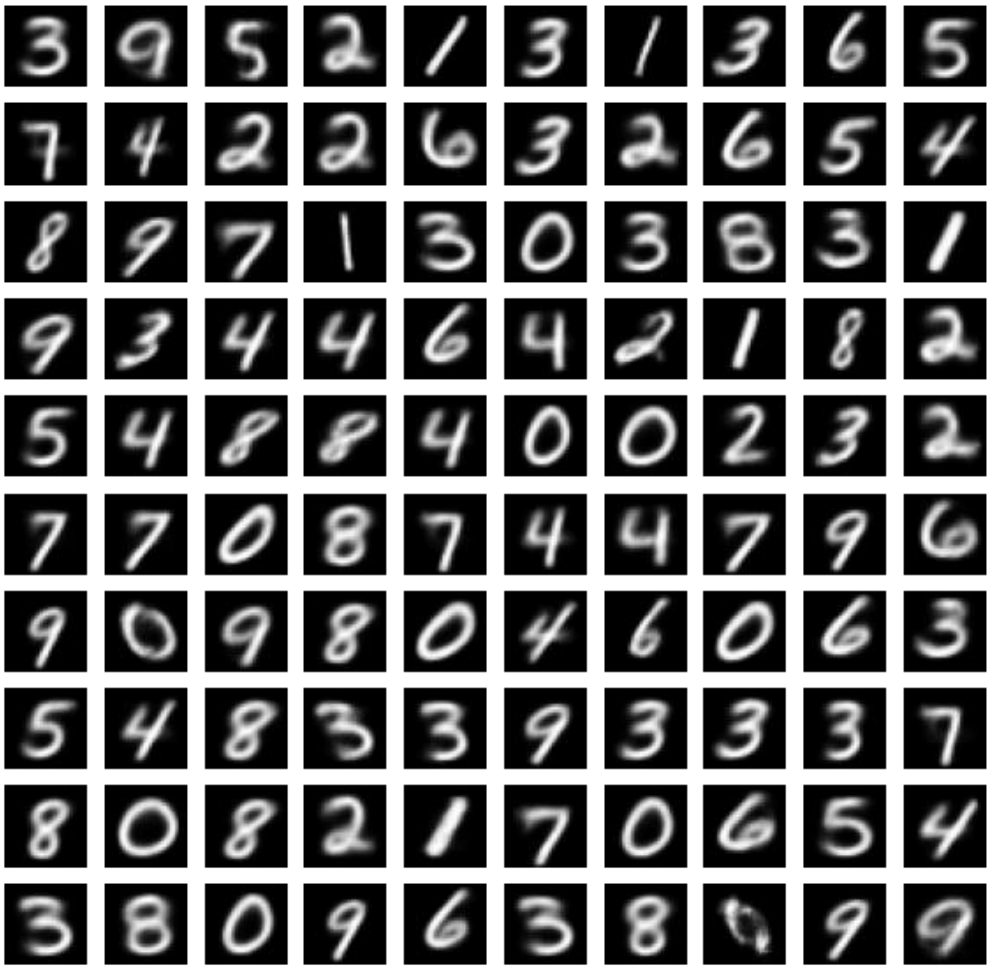
شکل ۱ـ۲۱) تصاویر ساخته شده بعد از اولین ایپاک



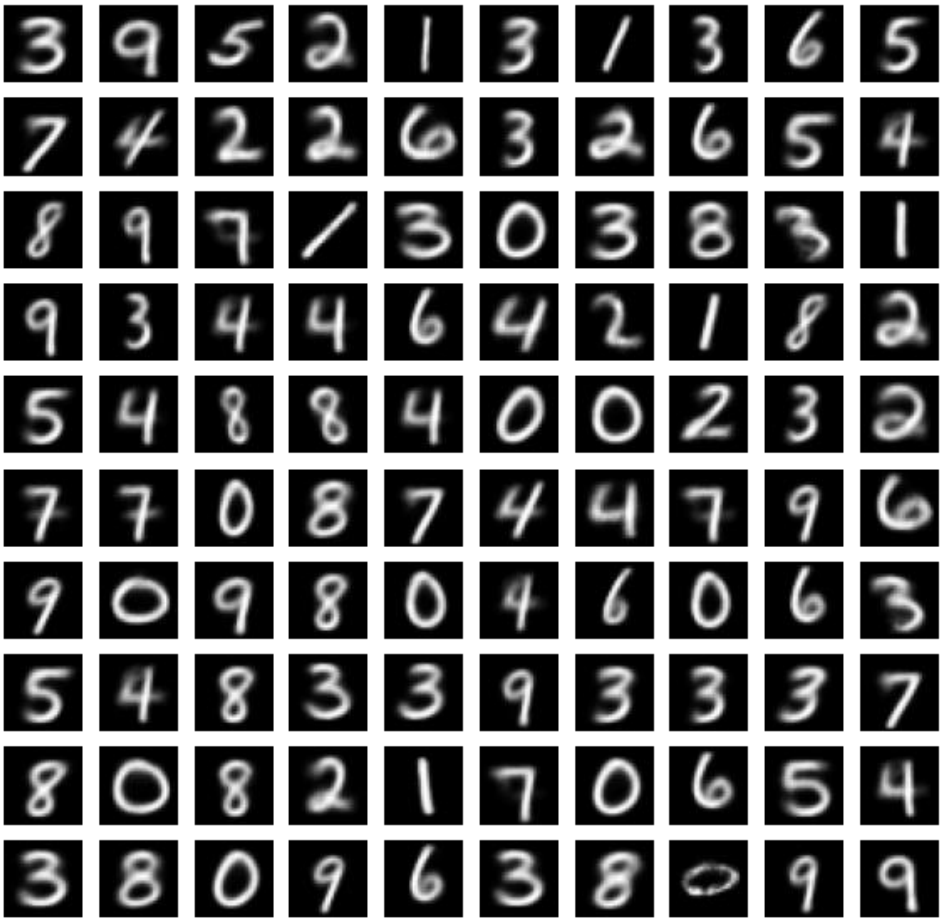
شکل ۱ـ۲۲) تصاویر ساخته شده بعد از دومین ایپاک



شکل ۱ـ۲۳) تصاویر ساخته شده بعد از چهارمین ایپاک



شکل ۱ـ۲۴) تصاویر ساخته شده بعد از پنجمین ایپاک



شکل ۱ـ۲۵) تصاویر ساخته شده بعد از دهمین ایپاک

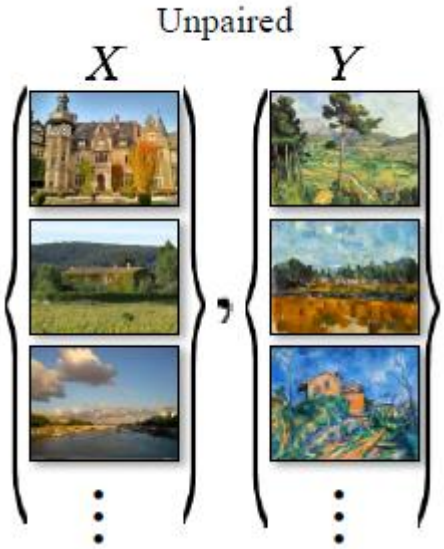
مقایسه ی نتایج این قسمت با قسمت ((و)) مشخص می کند که با توجه به مقید شدن توزیع به کلاس مرتبط با هر رقم شکل های ساخته شده از روی تصویر اصلی با آزادی عمل کمتری ساخته شده و در کلاس خود (رقم مشخص) باقی می ماند.

سوال ۲ – Cycle GAN

1) در این سوال به تحلیل و بررسی و پیاده سازی Cycle GAN میپردازیم. با توجه به مقاله هدف اصلی Cycle GAN این است که بتوانیم تصاویر از دو فضای مختلف را به یکدیگر تبدیل کنیم. معمولا در مسائل داده ها به صورت جفت هستند و آموزش روی آن ها صورت میگیرد. مثلا داده ها مانند شکل 2-1 هستند که مشاهده می شود به ازای هر داده جفتی مربوط به آن داده شده است.

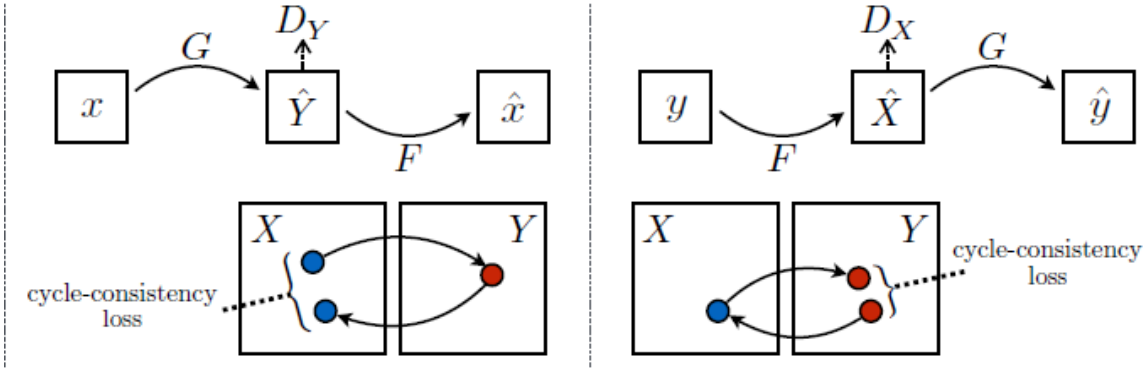
شکل 2-1: نمونه داده های جفت

اما در این مسئله داده ها به صورت جفت نیستند و صرفا تعدادی از آن ها در یک فضا و تعدادی در فضای دیگر موجود است و تناظر 1 به 1 بین آن ها نیست. در برخی مسائل ممکن است که داده ها به این صورت باشند چون حتما متناظر آن ها در فضای دیگر موجود نیست. برای مثال در همین مسئله ما عکس واقعی نقاشی هارا در اختیار نداریم و نقاشی از روی تصاویر واقعی نیز نداریم. نمونه این داده ها را در شکل 2-2 میتوان مشاهده کرد.



شکل 2-2: مجموعه داده غیر جفت در دو فضای مختلف

در این مقاله روش ارائه شده بر این پایه است که بتوانیم قابلیت تبدیل از هر دو فضا به یکدیگر را داشته باشیم. پس نیاز به دو Generator و دو Discriminator داریم، به این صورت که ورودی اگر از فضای X باشد با Generator مثلا G به فضای Y میرود و در آنجا با Dy برسی میشود. برای Y هم به همین صورت با یک تابع F به فضای X می رود و با یک Dx بررسی میشود. حال برای اینکه تصاویر به خوبی به هم تبدیل شوند و همه به یک نقطه همگرا نشوند در فضای تبدیل باید G و F معکوس هم باشند و هر نقطه را که به فضای Y میبریم با اعمال F باید تا حد امکان به نقطه اصلی خود برگردد. بر این اساس یک loss مهم در این شبکه تعریف میشود که باید آنرا کمینه کرد تا به این هدف برسیم که Cycle consistency loss می باشد. این تابع خطا را در تبدیل از فضای X به Y و برگشت مجدد X میسنجد و برعکس. در شکل 2-3 میتوان تصویری از مفهوم این خطا را دید.



شکل 2-3: Cycle Consistency Loss

دو تابع loss دیگر نیز برای این شبکه تعریف میشود که یکی از آن ها Adversial loss است. که در آن Gسعی میکند تصویری تولید کند فضای Y که مشباهت زیاد داشته باشد در حالی که Dy نیز در راستای این حرکت میکند که هر بار بهتر تشخیص دهد. Loss اخر نیز Identity loss است. این خطا به این معنا تعریف میشود که اگر ما یک تصویر از فضای Y را به تابع G بدهیم باید بتواند خود Y را به ما بدهد و تصویر دیگری تولید نکند چون خودش از همان فضا است. تمام 3 خطای گفته شده باید دو طرفه محاسبه شوند یعنی هم برای تبدیل از X به Y و هم برای Y به X.

در زیر می توان فرمول Identity Loss را مشاهده کرد که در کد پیاده سازی شده است:

$$\text{Idendity loss} = |G(y) - y| + |F(x) - x|$$

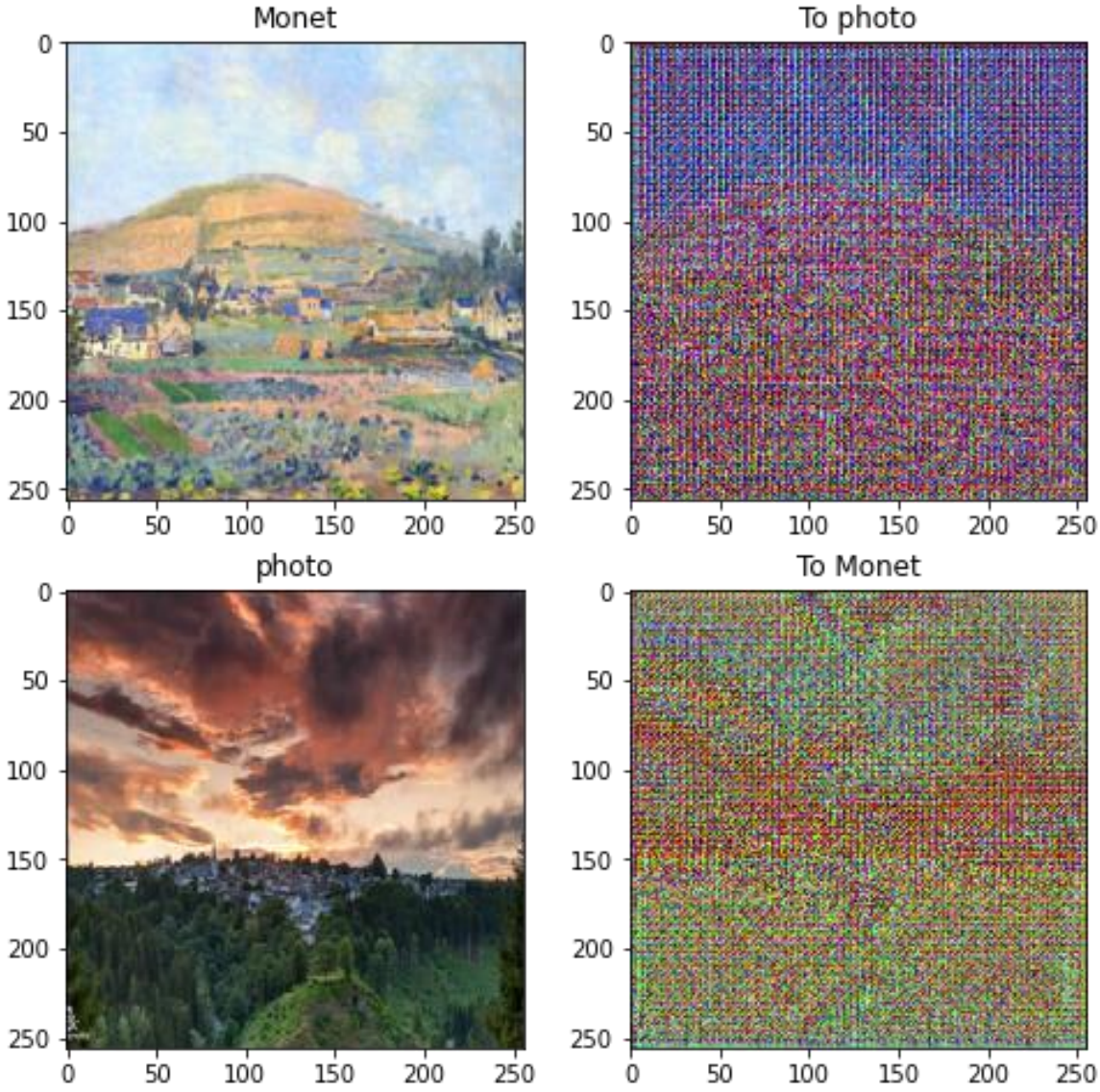
تابع خطا کلی سیستم نیز که باید آنرا کمینه کنیم به صروت زیر می شود:

$$L(G, F, Dx, Dy) = L_{Identity} + L_{Adversial} + \lambda * L_{Cycle\_Consistency}$$

2) در ایجاد Discriminator این شبکه از Patch GAN استفاده شده است. به این روش همچنین Markovian Discriminator نیز می گویند. برای جلوگیری از اینکه تصاویر تولیدی به صورت blurry نباشند از این روش استفاده میکنیم، چون در حالت عادی ممکن است فرکانس های بالا به خوبی تبدیل نشوند و تشخیص داده نشوند. در مسائلی که این مشکل وجود دارد با محدود کردن GAN Discriminator به فرکانس های بالا و تکیه بر اینکه L1 مانند قبل فرکانس های پایین را تصحیح کند. اسم PatchGAN برای این استفاده میشود که discriminator تلاش میکند که بر اساس patch های N*N تشخیص دهد هر کدام واقعی هستند یا جعلی و کل تصویر را یکجا در نظر نمیگیرد. این discriminator به خوبی تصویر را به صورت یک فضای رندوم مارکوف مدل میکند و با در نظر گرفتن استقلا بین پیکسل هایی که در خارج محدوده patch قرار دارند کارش را انجام میدهد.

4) در این بخش برای Generator از ساختار encoder – decoder unet استفاده شد. ابتدا داده ها را load کردیم. سپس تعدادی پیش پردازش بر روی آن ها انجام شد که در زمان آموزش از overfitting جلو گیری شود. مثلا قرینه کردن عکس ها و نرمال کردن و .... سپس patchGAN و Generator را تعریف کردیم.

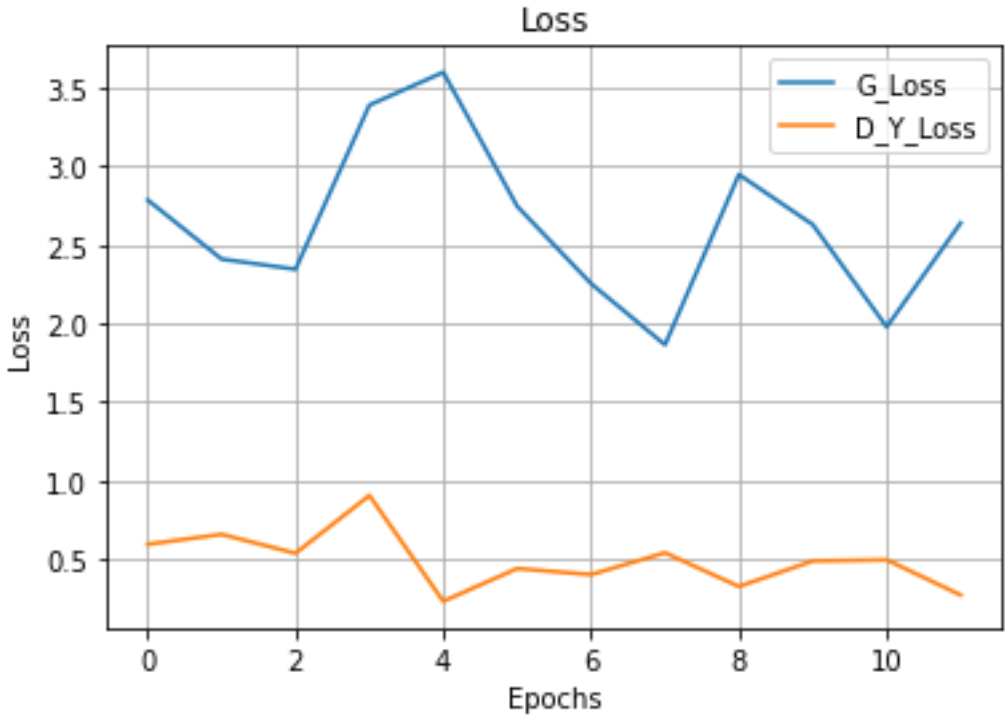
در ابتدا یک نمونه از هر فضا را به Generator ها دادیم تا نتایج اولیه قبل از آموزش را ببینیم. نتایج در شکل 2-4 آورده شده است.



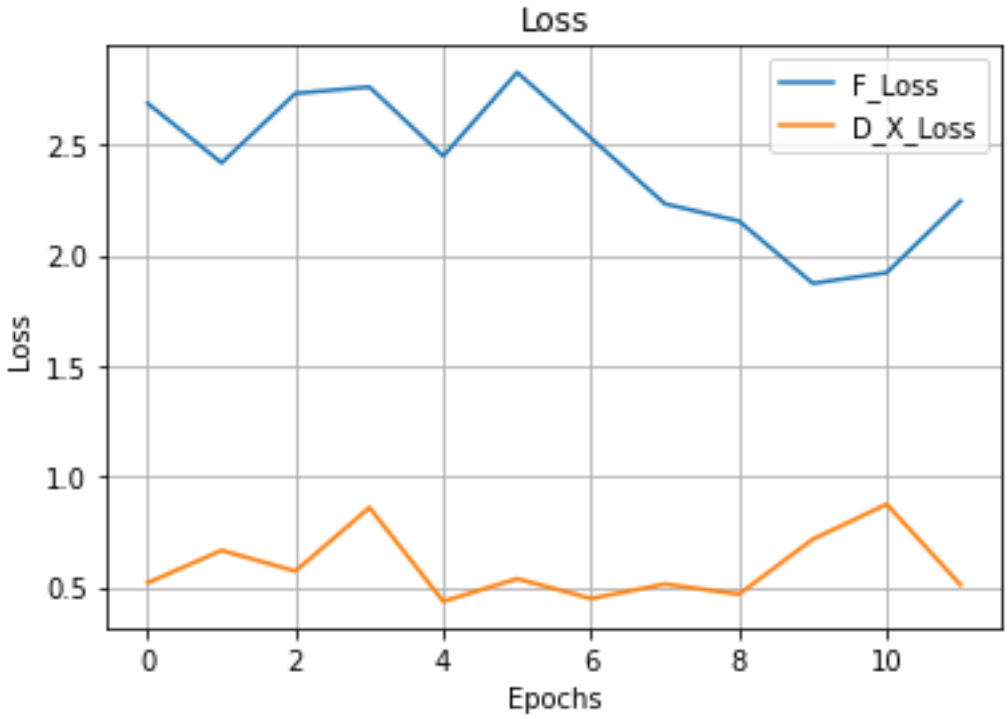
شکل 2-4: نتیجه اعمال تصاویر به generator آموزش ندیده

مشاهده میشود که نتایج بسیار بد هستند. سپس تابع های خطا را تعریف کردیم. بعد هر گام آموزش را که در آن هر بار خطا ها محاسبه شود و با Gradient Descent سعی در بهبود آن ها داشته باشیم.

شبکه برای 12 ایپاک آموزش داده شد و هر ایپاک حدود 460 ثانیه طول میکشید. روند تغییرات loss ها در شکل های 2-5 و 2-6 آورده شده است.



شکل 2-5: تغییرات loss برای G و Dy



شکل 2-6: تغییرات loss برای F و Dx

مشاهده میشود که خطا ها دائما در حال کم و زیاد شدن هستند ولی میتوان گفت روند کلی آن ها در حال کاهش است. همچنین این شبکه در تعداد ایپاک کمی آموزش دیده است و نمیتوان نتایج دقیقی از آن انتظار داشت.

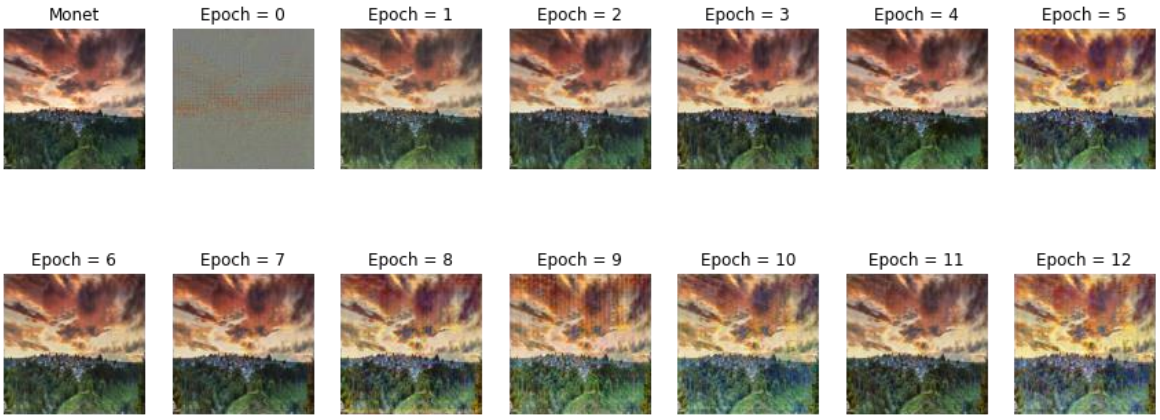
نتایج اعمال Generator تبدیل monet به photo در هر ایپاک برای تصویر نمونه اولیه در شکل 2-7 آورده شده است. همچنین برای دقیق تر مشخص بودن روند تغییرات همین تصاویر برای صرفا ایپاک اول و آخر نیز در شکل 2-8 آورده شده است.

شکل 2-7: نتایج هر ایپاک Monet2Photo

شکل 2-8: نتایج Monet2Photo برای ایپاک 1 و 12

مشاهده میشود در مقایسه با تصویر اصلی و ایپاک اول به نظر شبکه در جهت درستی دارد پیش می رود وبا افزایش تعداد ایپاک ها میتوان نتایج بهتری نیز از آن گرفت.

همین کار برای Photo2Monet نیز انجام شد ونتایج در شکل های 2-9 و 2-10 آورده شده است.

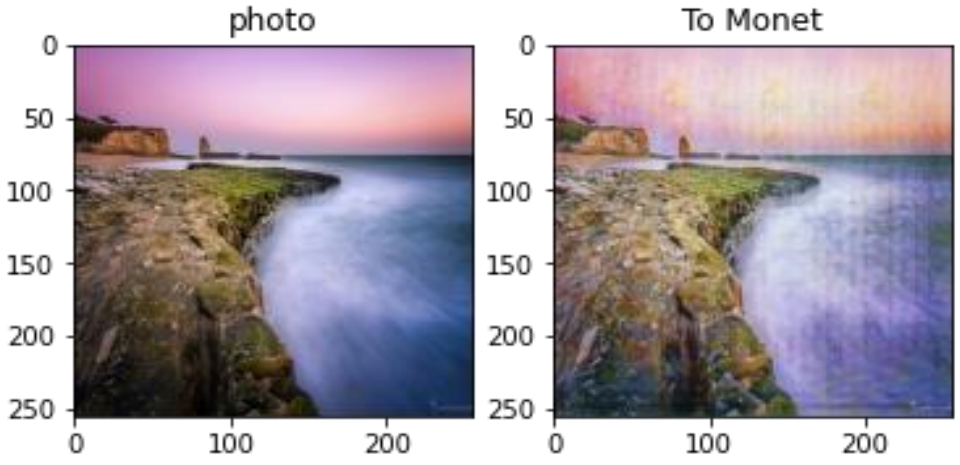


شکل2-9: نتایج تبدیل Photo2Monet در 12 ایپاک

شکل 2-10: نتایج تبدیل Photo2Monet در ایپاک 1 و 11

مشاهده می شود Generator برای تبدیل عکس به Monet به نظر بهتر کار میکند که با توجه به کمتر بودن loss آن طبیعی است. در کل هر دو generator در 12 ایپاک تصاویر قابل قبولی را دادند.

همچنین شبکه با 2 داده تست نیز بررسی شد که نتایج آن در شکل های 2-11 و 2-12 آورده شده است.



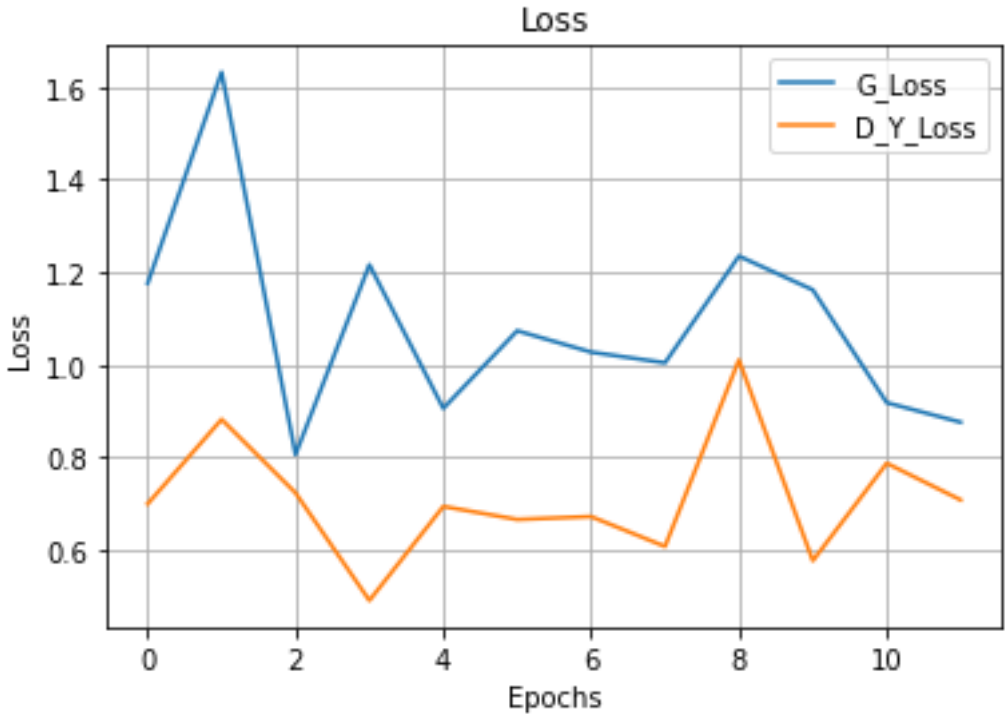
شکل 2-11: نتیجه تست Photo2Monet

شکل 2-12: نتیجه تست Monet2Photo

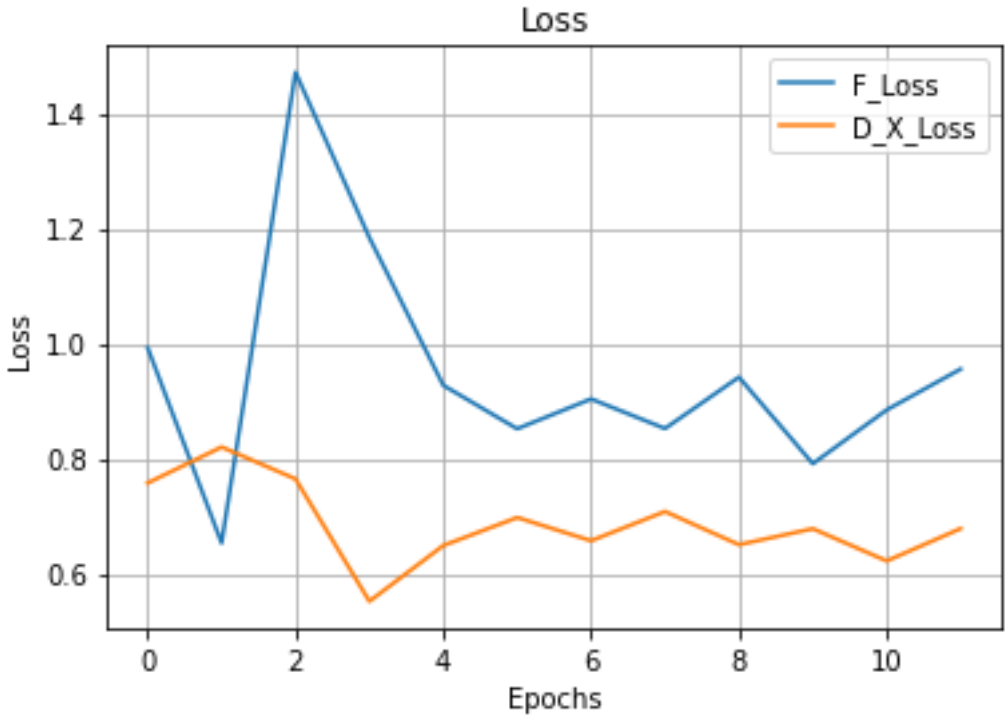
که مشاهده میشود با توجه به 12 ایپاک نتایج قابل قبولی گرفته ایم.

5) با حذف Cycle Consistency loss از loss ها الگوریتم مشابه قبل اجرا شد. این بار زمان آموزش هر ایپاک حدود 340 ثانیه شد.

نمودار مقادیر loss در هر ایپاک در شکل های 2-13 و 2-14 آورده شده است.



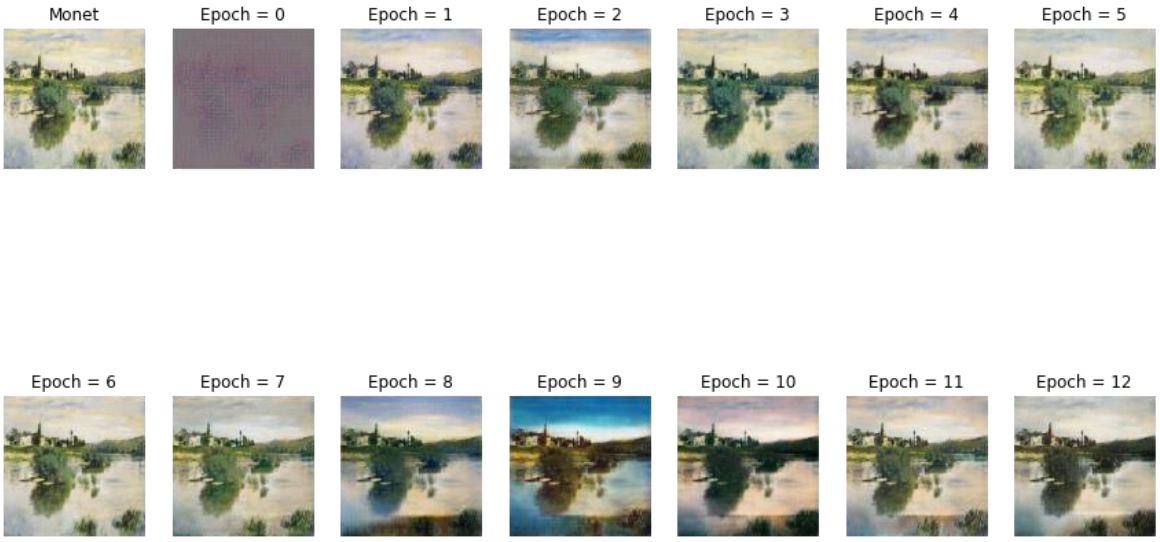
شکل 2-13: تغییرات loss برای G و Dy



شکل 2-14: تغییرات loss برای F و Dx

مشاهده می شود به نسبت قبل loss کمتری داریم ولی این به معنی بهتر بودن مدل نیست و به این دلیل است که یکی از ترم های loss را حذف کرده ایم و باید نتایج را بر اساس تصاویر تحلیل و مقایسه کرد.

نتایج تصاویر در هر ایپاک برای Monet2Photo در شکل های 2-15 و 2-16 آورده شده است.

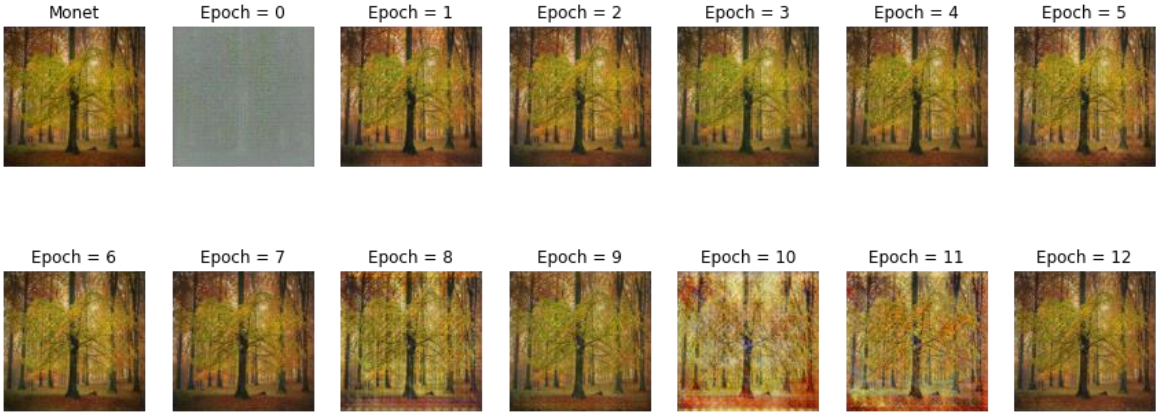


شکل 2-15: نتایج خروجی G در هر ایپاک برای Monet2Photo

شکل2-16: نتیجه خروجی Monet2Photo برای ایپاک اول و 12

مشاهده میشود در این حالت نیز خروجی قابل قبول است و تصویر خوبی داده است. اما باید تحلیل دقیقتر را برای مقایسه با حالت قبلی در تعداد ایپاک بیشتر بررسی کرد که نتایج قابل اتکا باشند.

همچنین نتایج Photo2Monet نیز به صورت شکل های 2-17 و 2-18 بدست آمد.

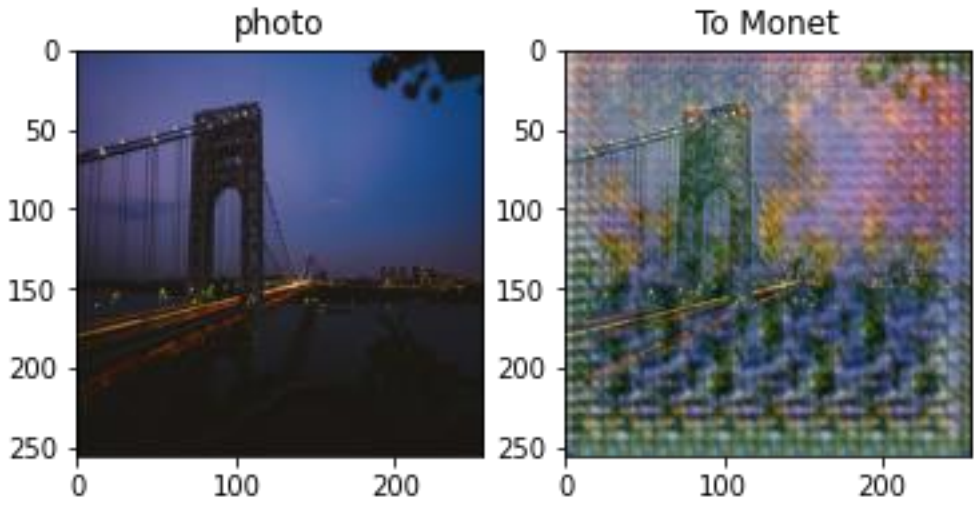


شکل2-17: نتایج خروجی F برای photo2Monet

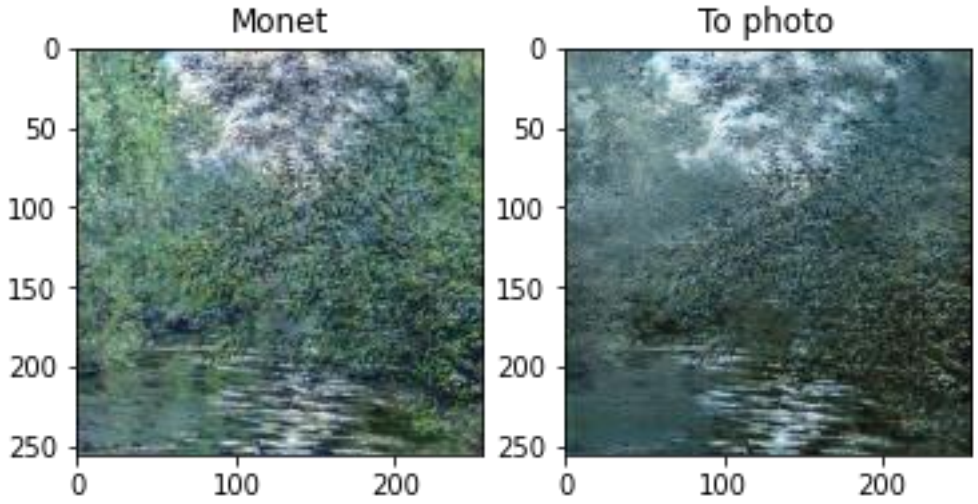
شکل 2-18: نتیجه Photo2Monet برای ایپاک 1 و 11

که در این حالت هم نتایج قابل قبول هستند.

همچنین برای 2 داده تست خروجی ها را بررسی کردیم که بصورت شکل ها 2-19 و 2-20 بدست آمد.



شکل2-19: نتیجه تست Photo2Monet



شکل2-20: نتیجه تست Monet2Photo

با حذف cycle consistency loss مشاهده میشود نتایج بر روی تست نویز زیادی دارد ولی باز هم قابل قبول هستند. اما در مقایسه با حالت سوال 4 به نظر نتایج حالت 4 روی داده تست بهتر بود.