به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

شبکه های عصبی و یادگیری عمیق

پروژه سوم

نام و نام خانوادگی	**مهسا مسعود – امید واهب**
شماره دانشجویی	810196635-810196582
تاریخ ارسال گزارش	1400/05/03

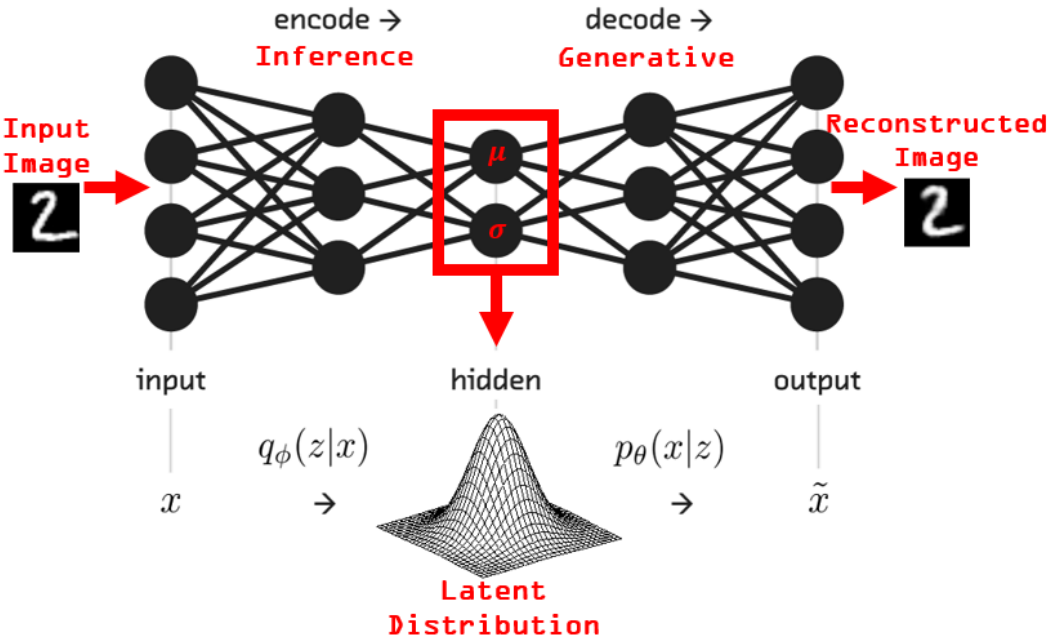
فهرست گزارش سوالات

سوال 1 – Variational Autoencoder

در این سوال به پیاده سازی شبکه variational autoencoder می پردازیم سپس به کمک ورژن conditional این شبکه مشکل کنترل روی داده های تولیدی از فضای latent را نیز حل می کنیم. داده استفاده شده MNIST است که تصاویر 28*28 اعداد است. کد این سوال در فایل Project3_Q1.ipynb موجود است.

الف) ابتدا شروع به پیاده سازی مدل خواسته شده در colab می کنیم که تمامی قدم های پیاده سازی در نوت بوک گفته شده است ولی در اینجا نیز کلیت را دوباره توضیح می دهیم. لینک مورد استفاده https://keras.io/examples/generative/vae/ است.

VAEها برای نمایش دادن داده ها با ابعاد بالا در فضایی با بعد کمتر ولی دارای معنی و بدون از دست رفتن اطلاعات استفاده می شوند. Cluster کردن داده در فضای latent وقتی لیبل را نداریم از دیگر کاربردهای این نوع شبکه عصبی است. Autoencoding روشی semi-supervised است که برای فشرده سازی داده بدون از دست رفتن اطلاعات استفاده می شود. توابع compress و decompress کردن این مدل از روی داده های موجود وزن های خود را تنظیم می کنند به عبارتی مثل روش PCA در VAE نیز هدف بردن داده به بعد کم بدون از دست رفتن اطلاعات مفید است. اتوانکودرها معمولا سه قسمت دارند که شامل انکودر، دیکودر و تابعی برای محاسبه loss است. این loss معمولا شامل خطای reconstruction است که خطای تفاوت خروجی مدل با ورودی آن است. در یادگیری این مدل ها معمولا هدف حداقل کردن این خطا به کمک گرادیان کاهشی است. نوع variational این اتوانکودرها که اینجا مورد بحث است خود داده را به خودی خود به یاد نمی سپارد بلکه سعی می کند توزیعی که داده های ورودی از آن می آیند را به دست آورد به عبارتی میانگین و انحراف از معیار را در فضای latent بیابد. این نوع از مدل جدیدا خیلی استفاده شده است و کاربردهای clustering نیز دارد.



شکل 1-1 : شکل کلی VAE

در شکل بالا معماری کلی این نوع از شبکه را مشاهده می کنیم که قسمت encoder یا اولیه مدل شکل ورودی را به فضای latent می برد و میانگین و انحراف از معیاری را در این فضا به دست می آورد و به نوعی ماهیتی استنباطی از روی داده ها دارد. قسمت دوم یا decoder هم ماهیت generative دارد و از این توزیع داده شده سمپل می گیرد و خروجی مشابه ورودی اولیه می سازد. ابتدا به سراغ ساخت انکودر می رویم.

پس از لایه ورودی دو تا لایه کانوولوشنی دوبعدی با 3 فیلتر برای RGB و تابع activation معمول این شبکه ها یعنی ReLU داریم. Padding نیز داریم تا سایز عکس ها کم نشود و stride نیز دوتایی است. ابتدا 784 داده ورودی را به 32 و بعد از آن به 64 نورون می دهیم سپس به یک لایه flatten می دهیم تا یک بعدی کند و سپس با یک لایه fully-connected با 16 نورون ReLU به فضای latent می رسیم. برای این کار دو تا خروجی از انکودر برای میانگین و برای لگاریتم انحراف از معیار تعریف می کنیم همچنین بعد فضای latent نیز طبق خواسته ی سوال 2 است.

همانطور که قبلا گفته شد تفاوت VAE با اتوانکودرهای معمول در ماهیت تصادفی تولید خروجی از latent space است یعنی sample برداشتن از متغیر های تصادفی که میانگین و انحراف از معیار آن ها را محاسبه کردیم. برای شبیه سازی این قسمت یک لایه جدید تعریف می کنیم به نام sampling که به شکل زیر تعریف می شود:

```python
class Sampling(layers.Layer):
    """Uses (z_mean, z_log_var) to sample z, the vector encoding a digit."""
    def call(self, inputs):
        z_mean, z_log_var = inputs
        batch = tf.shape(z_mean)[0]
        dim = tf.shape(z_mean)[1]
        epsilon = tf.keras.backend.random_normal(shape=(batch, dim))
        return z_mean + tf.exp(0.5 * z_log_var) * epsilon
```

شکل 1-2 : لایه sampling

در این لایه با گرفتن میانگین و انحراف از معیار متغیر z که برای latent space است به عنوان ورودی و سمپل گرفتن از متغیر نرمال با ویژگی های داده شده خروجی برای decode کردن در ادامه را تولید می کنیم. برای محاسبه خروجی نیز مقدار میانگین را با ترم اپسیلون ضرب در $e^{0.5\log(Var(Z))}$ جمع می کنیم که مقدار اپسیلون همان مقدار تصادفی حاصل از سمپل گرفتن از Z است.

```
Model: "encoder"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_3 (InputLayer)            [(None, 28, 28, 1)]  0
__________________________________________________________________________________________________
conv2d_2 (Conv2D)               (None, 14, 14, 32)   320         input_3[0][0]
__________________________________________________________________________________________________
conv2d_3 (Conv2D)               (None, 7, 7, 64)     18496       conv2d_2[0][0]
__________________________________________________________________________________________________
flatten_1 (Flatten)             (None, 3136)         0           conv2d_3[0][0]
__________________________________________________________________________________________________
dense_2 (Dense)                 (None, 16)           50192       flatten_1[0][0]
__________________________________________________________________________________________________
z_mean (Dense)                  (None, 2)            34          dense_2[0][0]
__________________________________________________________________________________________________
z_log_var (Dense)               (None, 2)            34          dense_2[0][0]
__________________________________________________________________________________________________
sampling_1 (Sampling)           (None, 2)            0           z_mean[0][0]
                                                                 z_log_var[0][0]
==================================================================================================
Total params: 69,076
Trainable params: 69,076
Non-trainable params: 0
__________________________________________________________________________________________________
```

شکل 1-3 : معماری انکودر VAE

حال به سراغ طراحی قسمت دوم یعنی decoder می رویم و اصولا مانند باقی اتوانکودرها معماری این قسمت باید برعکس معماری انکودر باشد. این قسمت مقادیر سمپل گرفته شده از latent space حاصل از لایه sampling را به یک لایه fully-connected با 7*7*64 نورون با تابع فعالساز ReLU می برد سپس با یک لایه reshape به فضای سه بعدی با ابعاد 7 و 7 و 64 می برد. سپس دو لایه deconvolution یا به عبارتی کانوولوشنی دو بعدی معکوس داریم با activation function معمول یعنی ReLU و padding برای عدم کاهش ابعاد و stride دوتایی. ابعاد این دو لایه نیز برعکس معادلشان در انکودر است یعنی به ترتیب

64 و 32 هستند و بعد از آن ها نیز یک لایه ی کانوولوشنی دوبعدی دیگر برای تولید خروجی داریم. که با تابع sigmoid مقادیر پیکسل ها را می سازد. لازم به ذکر است که لایه های کانوولوشنی دیکودر نیز سه تا فیلتر دارند برای تولید RGB و خلاصه مدل طراحی ده به شکل زیر است:

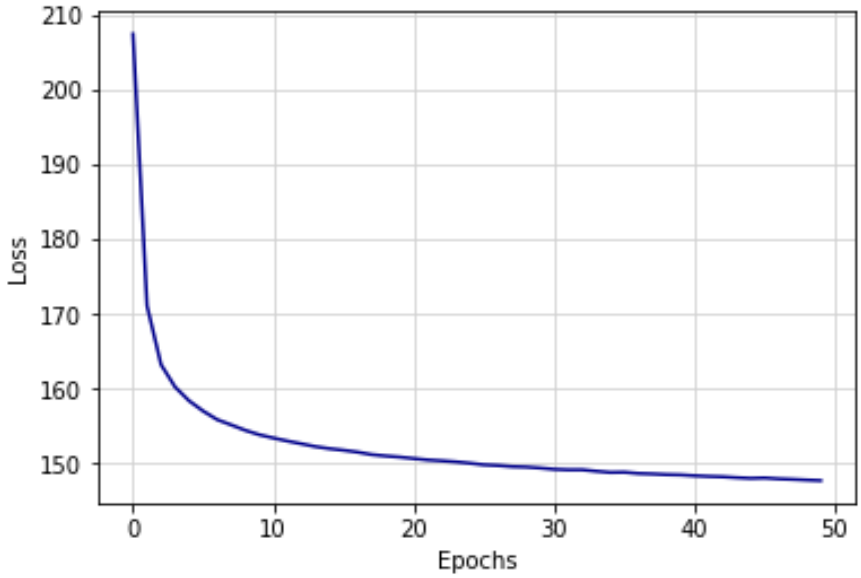
```
Model: "decoder"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_4 (InputLayer)         [(None, 2)]               0
_________________________________________________________________
dense_3 (Dense)              (None, 3136)              9408
_________________________________________________________________
reshape_1 (Reshape)          (None, 7, 7, 64)          0
_________________________________________________________________
conv2d_transpose_3 (Conv2DTr (None, 14, 14, 64)        36928
_________________________________________________________________
conv2d_transpose_4 (Conv2DTr (None, 28, 28, 32)        18464
_________________________________________________________________
conv2d_transpose_5 (Conv2DTr (None, 28, 28, 1)         289
=================================================================
Total params: 65,089
Trainable params: 65,089
Non-trainable params: 0
_________________________________________________________________
```

شکل 1-4 : معماری دیکودر VAE

در مجموع این مدل تقریبا 130 تا 140 هزار پارامتر دارد که نشان دهنده این است که خیلی مدل پیچیده ای نیست.

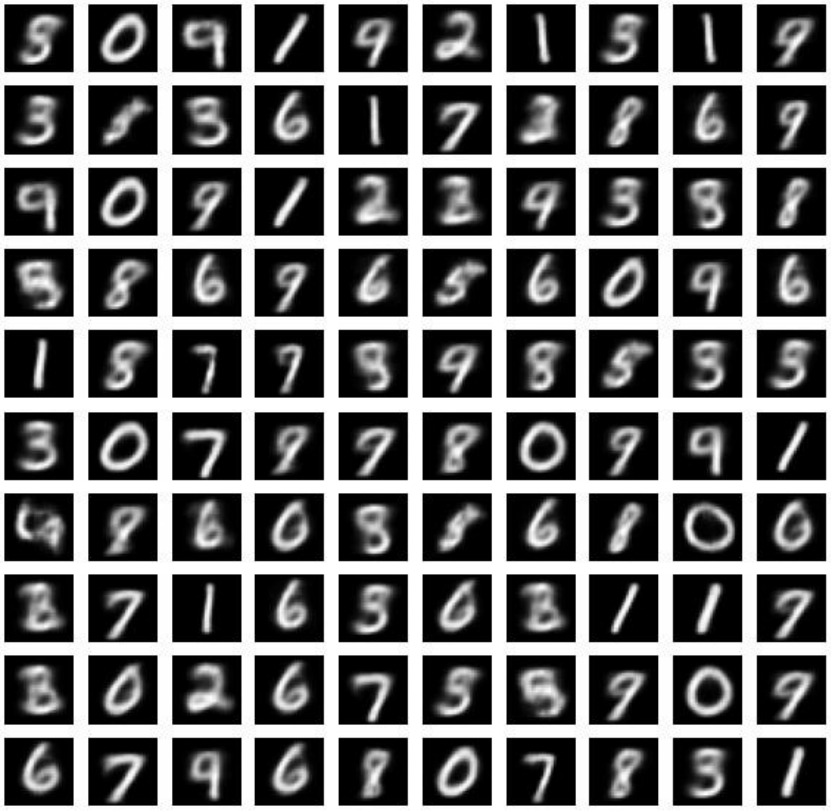
حال این دو قسمت را در کنار هم قرار می دهیم و کلاس VAE خواسته شده را می سازیم البته باید آخرین قسمت از VAE را نیز پیاده کنیم که تابع loss است. بالاتر گفتیم که عملیات آموزش توسط گرادیان کاهشی در راستای حداقل کردن تابع خطا انجام می شود که در این مدل خطا حاصل از دو نوع مختلف است. اولی reconstruction error است که cross-entropy بین خروجی مدل حاصل از سمپل های دیکود شده از latent space و ورودی مدل است و دومی هم Kullback-Liebler divergence است که فاصله بین توزیع فضای latent و توزیع واقعی داده ها است. می توان گفت که ترم دوم نقش regularization دارد. در این قسمت متدهای call و predict را نیز دوباره نوشتیم که بتوان منظم تر تولید خروجی کرد. حال دو تابع نیز برای رسم خروجی حاصل از فضای latent و cluster های موجود در این فضا طبق لیبل هر یک می نویسیم تا در ادامه استفاده کنیم. حال در آخرین مرحله داده ها را لود کرده و

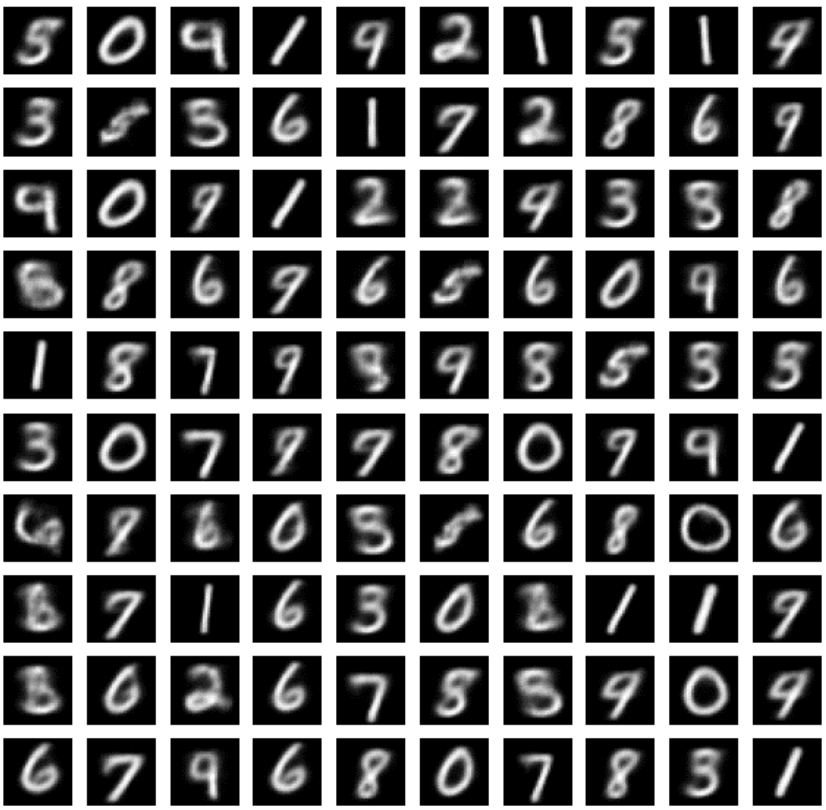
نرمال می کنیم یعنی تقسیم بر 255 می کنیم تا بین 0 تا 1 قرار گیرند سپس در 50 ایپاک مدل را آموزش می دهیم. نمودار خطا بر حسب ایپاک به شرح زیر است:

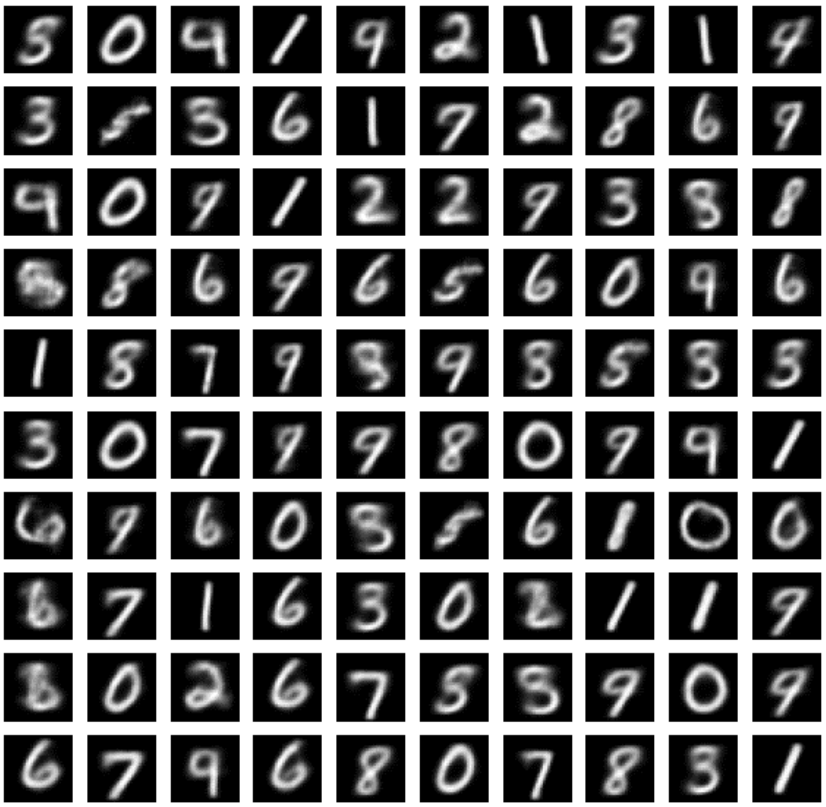


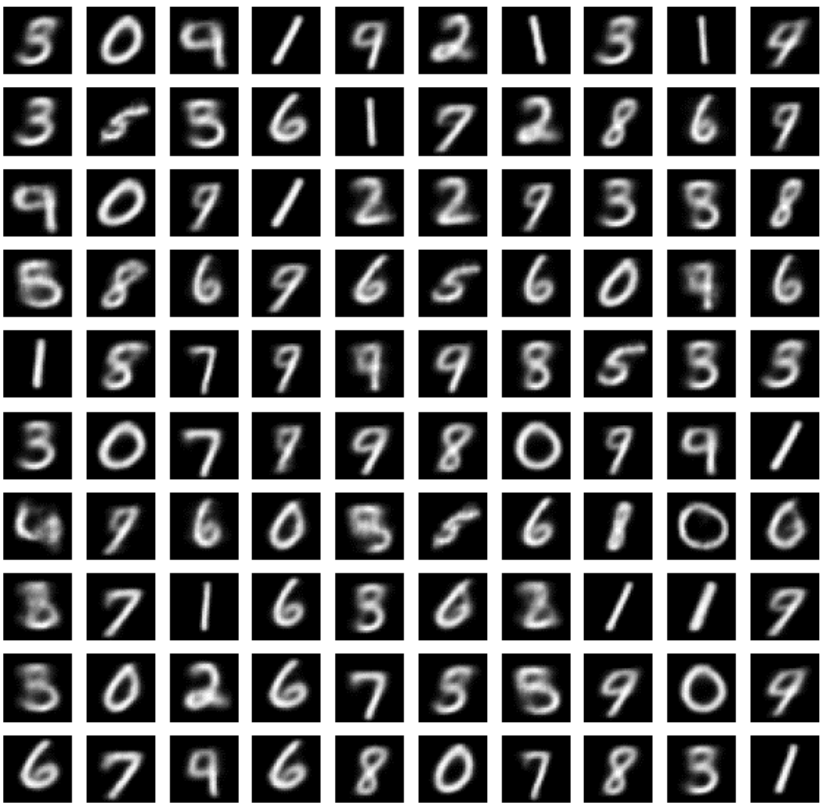
شکل 1-5 : نمودار خطا بر حسب ایپاک

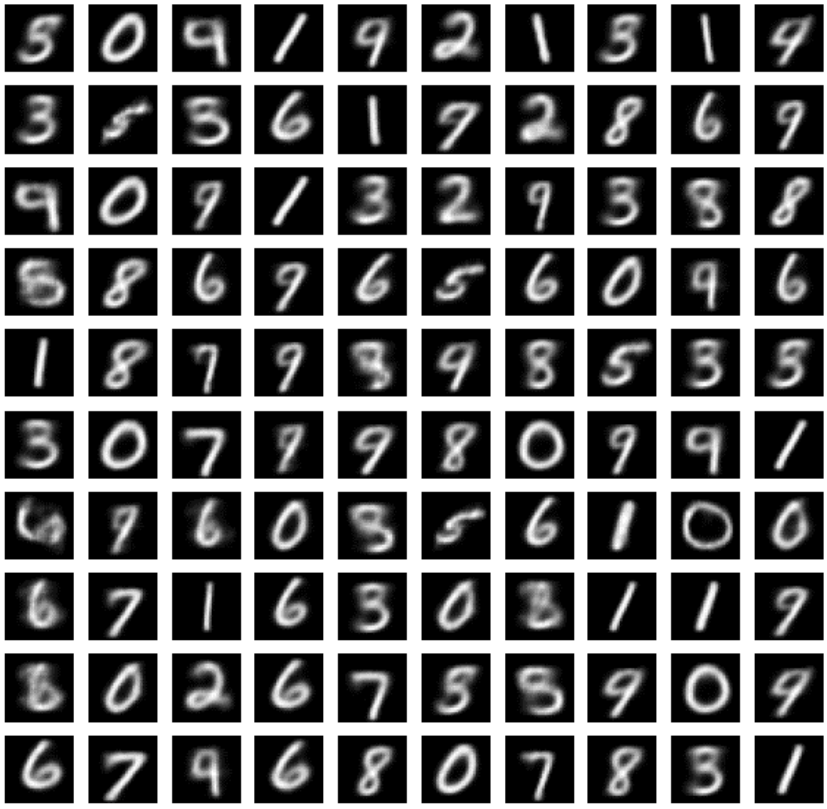
تعداد ایپاک 50 انتخاب شده چون شیب کاهش روی داده ی train کم شده است. در ادامه خروجی حاصل از 100 تا داده اول دیتاست را در ایپاک های مختلف یادگیری می بینیم تا پیشرفت مدل در تولید خروجی را ببینیم. ایپاک های 10، 20، 30، 40، 50 برای دیدن خروجی انتخاب شده اند تا پیشرفت به خوبی مشاهده شود.











شکل 1-6 : خروجی مدل به مرور زمان در ایپاک های مختلف

می توان مشاهده کرد که به مرور زمان وضوح تصاویر بالاتر می رود و کیفیت تصاویر خروجی پیشرفت می کند ولی کلاس های آنها تغییری نمی کند که انتظار هم همین است و این اتفاق که عملکرد مدل در حدی ضعیف باشد که کلاس اشتباه تولید کند نهایتا در چند ایپاک اول اتفاق می افتد.

ب) همانطور که گفته شد از تابع خطایی استفاده می کنیم که دو قسمت دارد یعنی reconstruction error و Kullback-Liebler divergence. ترم اول که فاصله ی عکس های تولید شده با عکس های واقعی را حساب می کند و عضو جدا ناپذیر اتوانکودرها است و ترم دوم نیز تفاوت فضای latent و ورودی ها را تاثیر می دهد. فرمول محاسبه ترم دوم به صورت زیر است:

$$D_{KL}[N(\mu(X),\Sigma(X))\|N(0,1)] = \frac{1}{2}\left(\mathrm{tr}(\Sigma(X)) + \mu(X)^T\mu(X) - k - \log\det(\Sigma(X))\right)$$

در این فرمول P(z) توزیع نرمال استاندارد است و میو و سیگما هم میانگین و واریانس متغیر تصادفی X است و K نیز بعد متغیر X است. می توان فرمول بالا را به شکل زیر بسط داد:

$$D_{KL}[N(\mu(X),\Sigma(X))\|N(0,1)] = \frac{1}{2}\left(\sum_k \Sigma(X) + \sum_k \mu^2(X) - \sum_k 1 - \log\prod_k \Sigma(X)\right)$$
$$= \frac{1}{2}\left(\sum_k \Sigma(X) + \sum_k \mu^2(X) - \sum_k 1 - \sum_k \log\Sigma(X)\right)$$
$$= \frac{1}{2}\sum_k \left(\Sigma(X) + \mu^2(X) - 1 - \log\Sigma(X)\right)$$

در انتها می توان گفت که از فرمول $\frac{1}{2}\sum_k (\exp(\Sigma(X)) + \mu^2(X) - 1 - \Sigma(X))$ برای این ترم از خطا استفاده می کنیم.

در انتها مجموع این دو ترم خطا را به عنوان خطای کلی در نظر می گیریم. لزوم وجود ترم دوم این است که داده های کلاس های مختلف در فضای latent از هم فاصله داشته باشند تا بتوان آنها را از هم تمییز داد و هر زمان که خواستیم با لیبل خواسته شده داده تولید کنیم البته که این کار خیلی خوب انجام نمی شود و استفاده از Conditional VAE برای این کار مناسب تر است. اگر این ترم را نمی گذاشتیم مدل طوری آموزش داده میشد که همه ی داده ها نزدیک هم قرار می گرفتند.

ج) از این روش برای گرادیان گرفتن از تابع loss و پیاده سازی gradient descent backpropagation استفاده می شود. می خواهیم عبارت زیر را محاسبه کنیم پس داریم:

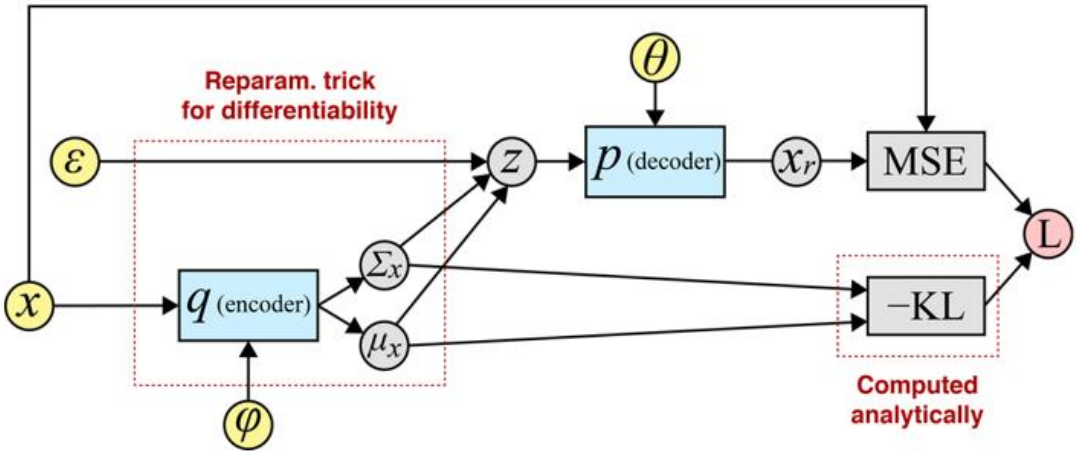
$$\nabla_\theta \mathbb{E}_{p(z)}[f_\theta(z)] = \nabla_\theta\Big[\int_z p(z) f_\theta(z) dz\Big]$$
$$= \int_z p(z)\Big[\nabla_\theta f_\theta(z)\Big]dz$$
$$= \mathbb{E}_{p(z)}\Big[\nabla_\theta f_\theta(z)\Big]$$

$$\nabla_\theta \mathbb{E}_{p_\theta(z)}[f_\theta(z)] = \nabla_\theta\Big[\int_z p_\theta(z) f_\theta(z) dz\Big]$$
$$= \int_z \nabla_\theta\Big[p_\theta(z) f_\theta(z)\Big]dz$$
$$= \int_z f_\theta(z)\nabla_\theta p_\theta(z) dz + \int_z p_\theta(z)\nabla_\theta f_\theta(z) dz$$
$$= \int_z f_\theta(z)\nabla_\theta p_\theta(z) dz + \mathbb{E}_{p_\theta(z)}\Big[\nabla_\theta f_\theta(z)\Big]$$

خود P نیز می تواند تابعی از θ باشد که کار را سخت می کند و ترم انتگرالی خط آخر را نمی توان محاسبه کرد پس به کمک تقریب پایین گرادیان خطا را محاسبه می کنیم:

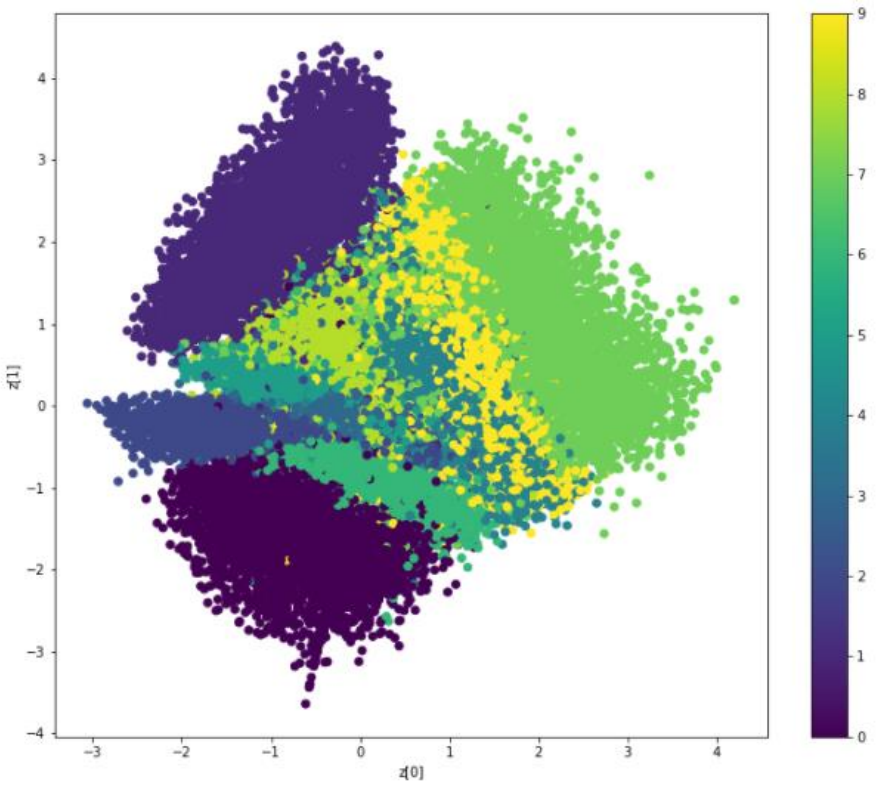
$$\begin{aligned}\nabla_\theta \mathbb{E}_{p_\theta(\mathbf{z})}[f(\mathbf{z}^{(i)})] &= \nabla_\theta \mathbb{E}_{p(\epsilon)}[f(g_\theta(\epsilon, \mathbf{x}^{(i)}))] \\ &= \mathbb{E}_{p(\epsilon)}[\nabla_\theta f(g_\theta(\epsilon, \mathbf{x}^{(i)}))] \\ &\approx \frac{1}{L}\sum_{l=1}^{L} \nabla_\theta f(g_\theta(\epsilon^{(l)}, \mathbf{x}^{(i)}))\end{aligned}$$

در رابطه بالا اپسیلون همان P است که توزیع نرمال استاندارد است و z نیز تغییر متغیر گفته شده است و L نیز مقدار خطا است. در این روش به جای محاسبه گرادیان یک امید ریاضی، امید ریاضی ن گرادیان را حساب می کنیم



شکل 1-7 : ترفند reparameterization

د) حال scatterplot داده ها در فضای latent را می کشیم که نتیجه به شکل زیر است:



شکل 1-8 : داده ها در فضای latent

می توان دید که داده های کلاس های مختلف از یکدیگر فاصله دارند و در فضای دوبعدی به خوبی از هم تفکیک شده اند.

ه) حال از شکل 8 حدود پراکندگی داده های کلاس های مختلف را پیدا می کنیم. به ترتیب محدوده بعد صفرم و بعد یکم را برای هر کلاس در زیر محاسبه می کنیم:

0: -2.5 , 1 ; -3.6 , 1.6

1: -2.6 , 0.5 ; 0.5 , 4.5

2: -3 , 1 ; -1 , 0

3: -1 , 2 ; -1 , 2

4: -2 , 1 ; -1 , 0.5

5: -1 , 2 ; 0 , -2

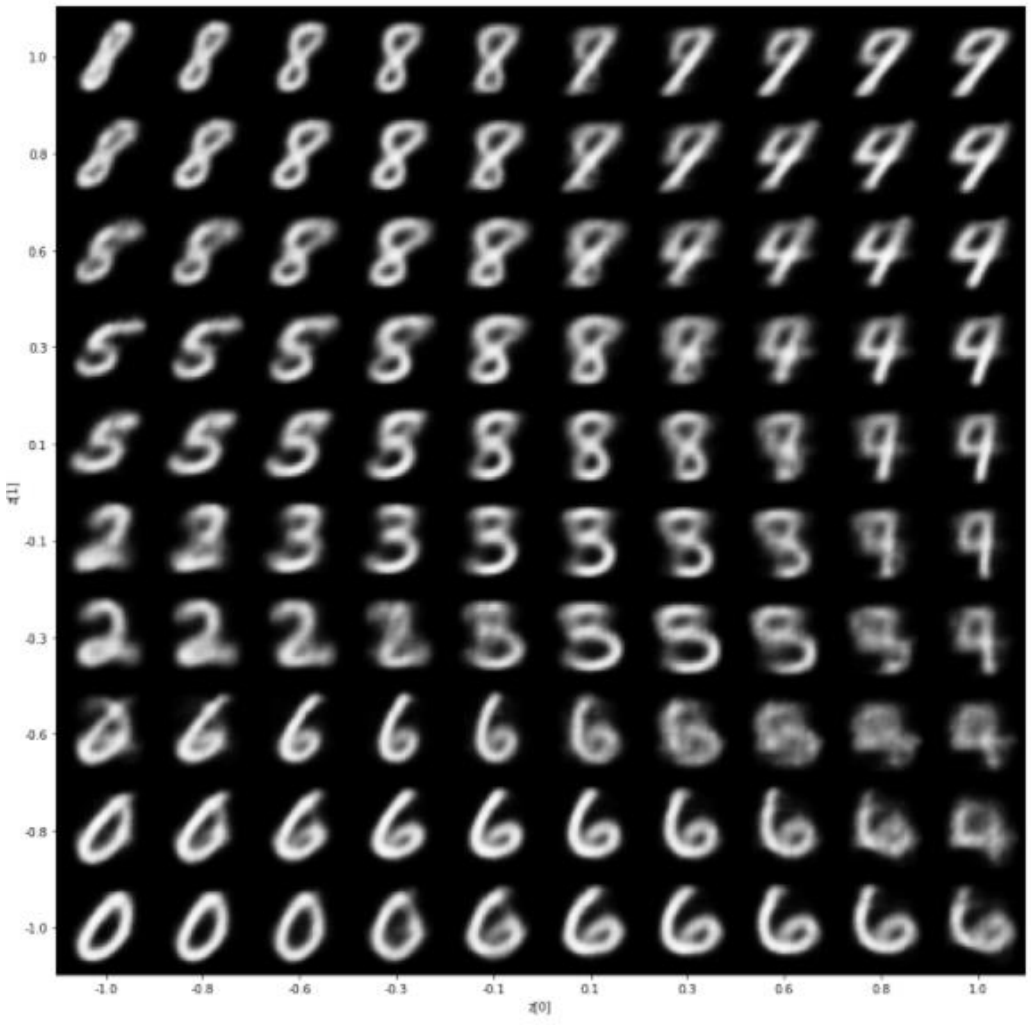
6: -1 , 1.5 ; -2.5 , 0

7: 0 , 4 ; -1.5 , 3.5

8: -2 , 1 ; -1 , 2

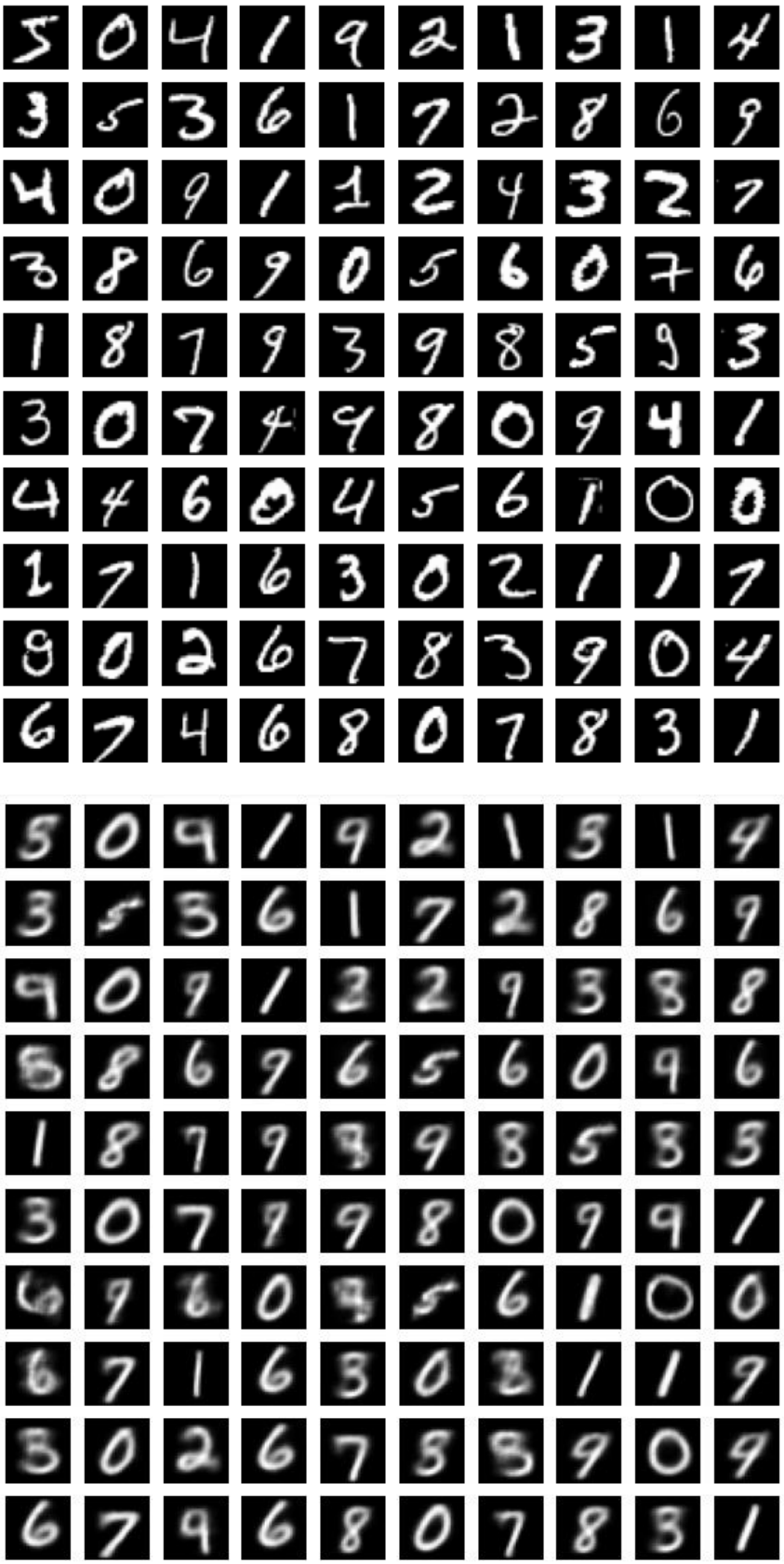
9: -1 , 2.5 ; -2 , 3

حال یک grid ده در ده از داده های موجود در latent space و خروجی حاصل از آنها می کشیم. در این قسمت در اصل می خواهیم ببینیم که با تغییر در بازه های فضای latent چه خروجی هایی تولید می شوند و تغییرات بین کلاس ها در این فضا به چه صورت است. شکل حاصل به صورت زیر است:



شکل 1-9 : خروجی حاصل از داده های موجود در فضای latent

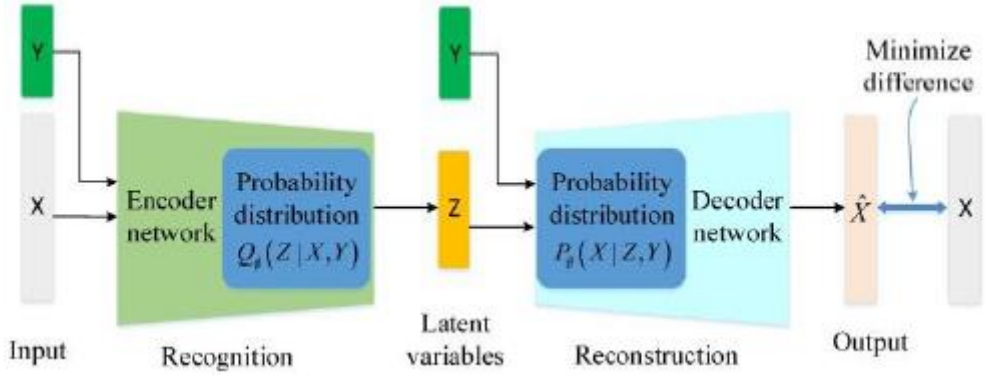
و) حال برای بررسی کیفیت داده های تولید شده 100 تا داده ی ورودی را کشیده سپس به مدل داده و خروجی حاصل را رسم می کنیم:



شکل 1-10 : ورودی و خروجی مدل

این خروجی را برای 100 تا داده ی اول مدل می کشیم که می توان دید که به خوبی مدل ما توانسته خروجی هایی با این داده ها تولید کند که شبیه ورودی ها هستند البته کمی تار هستند که می توان با یادگیری بیشتر آن را بهبود بخشید همچنین چند مورد داده تولیدی درست نیست و داده ی تولید شده با کلاس داده ی ورودی یکسان نیست البته خیلی شبیه اند که علت این مشکل همپوشانی کلاس های مختلف در فضای latent است که با یادگیری طولانی تر یا بالا بردن بعد این فضا قابل حل بود.

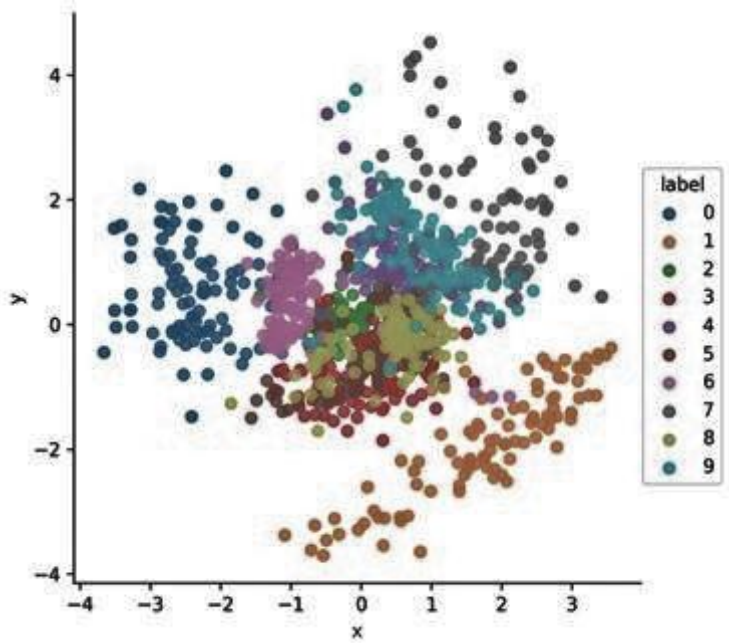
ز) حال در این قسمت conditional VAE را برای همین دیتاست پیاده می کنیم. شکل این مدل در زیر آمده است که به متغیر y مشروط شده است:



شکل 1-11 : شماتیک شبکه CVAE

معماری این شبکه تا حدودی شبیه شبکه ی قبلی است و بعد فضای latent را 2 در نظر گرفتیم و تعداد 40 اپاک هم آموزش دادیم تا از هم جدا شوند و بتوانیم به خوبی condition و شرط در فضای latent بگذاریم. قسمت انکودر از دو لایه کانوولوشنی دوبعدی با ابعاد 2 و 16 تشکیل شده که kernel size 5 دارند و stride هم 2 است. سپس یک لایه fully connected با 300 نورون برای میانگین داریم و یکی هم برای انحراف از معیار تا دو خروجی خواسته شده در فضای latent ساخته شوند. سپس قسمت دیکودر را داریم که دو لایه fully connected دیگر شامل 300 و 512 نورون دارد و پشت آنها هم سه تا لایه کانوولوشنی دوبعدی با کرنل های 5 تایی و stride دو به ترتیب با 32، 15 و 2 نورون قرار دارند.

ح) مثل قسمت های قبل scatterplot و cluster های خواسته شده را رسم می کنیم:



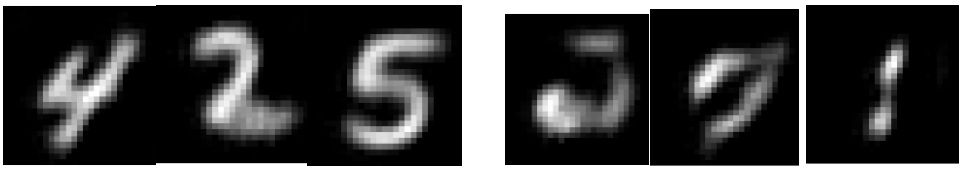
شکل 1-12 : Cluster های داده ها در فضای latent

می توان دید که به نسبت حالت قبل بهتر تفکیک شده اند که قابل انتظار هم بود البته لازم به ذکر است که همه ی داده ها رسم نشدند و سمپلی از کل کشیده شده تا واضح باشد. علت بهتر شدن تفکیک پذیری هم وارد کردن شرطی شدن نسبت به متغیر y است.

ط) حال به دلخواه یک grid ده در ده از داده های موجود در latent space را می کشیم که می توان دید از حالت قبل بهتر است که به دلیل تفکیک پذیری بهتر این مدل است:

شکل 1-13 : خروجی حاصل از داده های موجود در فضای latent

ی) مانند قسمت و تعدادی از تصاویر ورودی و خروجی آنها را رسم می کنیم. به ترتیب لیبل عکس ها 0، 2، 4 و 5، 2، 4 است:



شکل 1-14 : نمونه خروجی ها و ورودی مدل

می توان دید که وضوح خیلی خوب نیست و در یک مورد حتی تصویر تولیدی با ورودی یکی نیست پس بهتر بود بعد فضای latent را در این قسمت بالا می بردیم که وقت نشد متاسفانه.

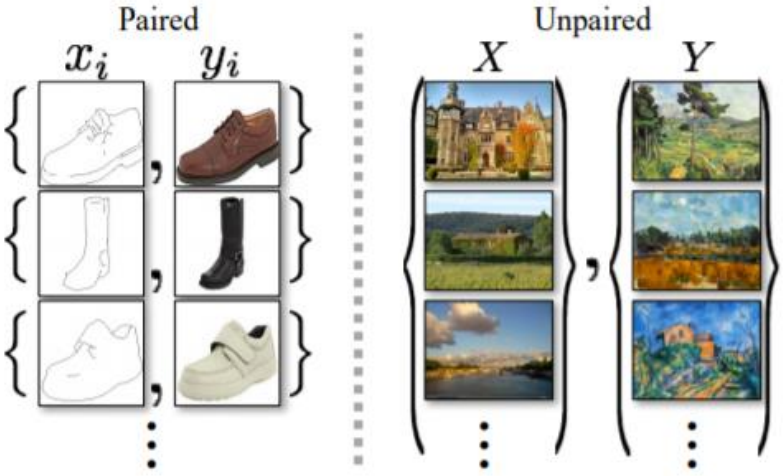
سوال 2 – CycleGAN

در این بخش با شبکه CycleGAN کار خواهیم کرد که یکی از اکستنشن های شبکه GAN است که از دو generator و دو discriminator استفاده می کند. اینجا با استفاده از این نوع ایده (CycleGAN) برای تبدیل تصاویر غیر جفت استفاده میکنیم.

1:

در این قسمت به تحلیل ساز و کار شبکه میپردازیم و خطاهای تعریف شده برای آموزش را بیان مینماییم.

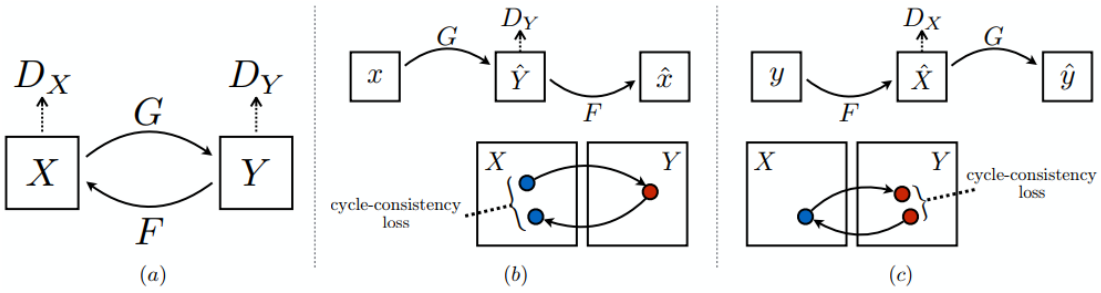
ترجمه تصویر به تصویر، یک مساله از جنس vision وgraphicمیباشد که هدف در آن رسیدن به یک نوع نگاشت از تصاویر ورودی به تصاویر خروجی میباشد که از یک یادگیری با استفاده از جفت تصاویر بهره میگیرد، اما در بسیاری از کارها، ما دسترسی ای به داده های train جفت شده نداریم، در اینجا از یک راه دیگر برای ترجمه تصویر از دامنه ورودی X به دامنه خروجی Y در غیاب نمونه های جفت شده استفاده خواهیم کرد.



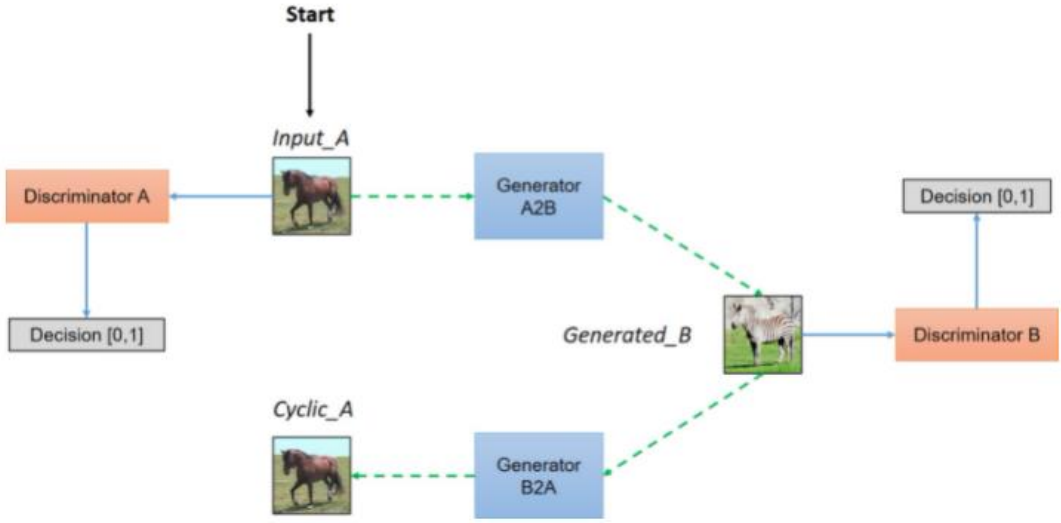
شکل 2-1 مثالی از نگاشت های جفت شده(مطلوب) و جفت نشده

هدف ،آموزش یک نگاشت کننده (تابع Generator اول)که فضای ورودی X را به فضای خروجی Y ببرد به طوری که توزیع تصاویر بدست آمده از G(X) ، از فضای خروجی توصیف شده، غیر قابل تشخیص باشد که این کار با یک adversarial loss انجام میشود. حال به این دلیل که، این نگاشت به اندازه کافی definite نیست و درجه های آزادی زیادی برای تعیین دقیق شدن دارد ، ما آن را با نگاشت معکوس یک تابع دیگر که

دومین generator ما می باشد (F) و ایندفعه فضای Y را به X میبرد، جفت میکنیم. اگر که فرض کنیم F(G(X)) = X ، داریم:



شکل2-2- Generator های شبکه CycleGAN



شکل 2-2: نمونه عملکرد شبکه CycleGAN با ورودی مشخصه

در این شبکه هدف Generator AtoB (اولین جنریتور) این است که تصویر اسب را به گورخر تبدیل کند و این کار را به گونه ای انجام دهد که Discriminator B (که مثل یک قاضی در این جا عمل میکند و باید label درست یا غلط بودن خروجی تبدیل را مشخص نماید)، تایید کند که این یک تصویر گورخر واقعی میباشد. اما طبق آزمایش مشاهده میکنیم که بدون Generator دوم، گورخر های کاملا مشابهی از Generator اول میگیریم که امکان مشخص شدن جعلی بودن آن ها زیاد است برای همین از Generator BtoA (جنریتور دوم) استفاده میکنیم که وظیفه دارد با استفاده از تصویر ساخته شده توسط Generator AtoB ، تصویر اورجینال ورودی شبکه Generator AtoB را بازسازی کند، بدین ترتیب، شبکه Generator AtoB تمام سعی خود را بکار میگیرد که تصاویر فضای خوجی Y (خروجی مطلوب) را تولید نماید که از نمایش دقیق فضای ورودی X (ورودی) ساخته شده است.

برای آموزش این شبکه دو نوع خطا (تابع خطا) معرفی شده است:

1- Adversarial Loss:

برای تابع G که داده را از فضای X به فضای Y میبرد و Discriminator اش، که آن را با D_y نمایش میدهیم، داریم:

$$L_{GAN} = E_{y \sim P_{data}}\left(LogD_y(Y)\right) + E_{x \sim P_{data}}\left(Log(1 - D_y(G(X))\right)$$

برای هر دو تابع این loss اعمال خواهد شد.

در واقع G سعی میکند خروجی هایش را به دامنه Y چنان نزدیک کند که به مرتبه مشابه برسند در همین حال D_y سعی میکند بین نمونه های y و خروجی شبکه G تفاوت ایجاد کند. G به سمت minimize شدن و D خود را maximize می کند.

2- Cycle consistency:

به صورت تئوری میتوان مشاهده نمود که با تابع خطای شماره 1 که در بالا توضیح داده شد میتوانیم به توابع Generator از G و F برسیم که شرایط مطلوب مارا در ساختن داده های مورد انتظار برآورده کنند. اما این به تنهایی کافی نیست و باید یک تابع هزینه دیگر داشته باشیم که هر x_i از ورودی را به y_i متناظر آن نگاشت کنیم.

تابع هزینه دوم:

$$L_{GAN} = E_{x \sim P_{data}}\left(\left|F\left(G(x)\right) - x\right|\right) + E_{y \sim P_{data}}\left(\left|G\left(F(y)\right) - y\right|\right)$$

حال برای هر x ای از دامنه X ، چرخه ترجمه تصویر، باید بتواند مارا به x برگرداند همانند زیر:

$$x \to G(x) \to F\left(G(x)\right) \to x$$

همچنین به صورت مشابه نیز میتوان متصور شد که برای y نیز در حالت برعکس این عملیات خواهیم داشت:

$$y \to F(y) \to G\left(F(y)\right) \to y$$

و بدین صورت از این تابع Loss استفاده میشود.

2-

در این بخش، به بررسی PatchGAN در Discriminator میپردازیم.

Discriminator های طراحی شده در این ساختار از PatchGAN های 70*70 ساخته شدند که هدف آن این است که بتواند با استفاده از overlapping تشخیص دهد patch تصویر جعلی یا واقعی است. مزیت آن این است که در این حالت، پارامتر های کمتری را هم شامل میشود. ایده PatchGAN این است که تصویر ورودی خام را به برخی از

Patch های کوچک محلی تقسیم کرده ، یک Discriminator کلی را به صورت convolve شده در هر پچ اجرا کنیم و به طور متوسط تمام پاسخ ها را بدست بیاوریم تا خروجی نهایی نشان داده شود که آیا تصویر ورودی معتبر است یا نه .

تفاوت اصلی بین PatchGAN و یک GAN Discriminator معمولی در این است که GAN Discriminator تصویر ورودی را به یک خروجی اسکالر در محدوده 0 تا 1 نشان می دهد که همان احتمال واقعی بودن یا جعلی بودن تصویر است این در حالی است که PatchGAN یک آرایه را ارائه می دهد به عنوان خروجی با هر ورودی که واقعی یا جعلی بودن Patch مربوطه را نشان می دهد. دلیل اینکه این مقاله از PatchGAN به عنوان تمایز دهنده خود استفاده می کند این است که پارامترهای کمتری نسبت به یک تمایز دهنده تصویر کامل دارد و بنابراین بسیار سریع اجرا می شود و می تواند روی تصاویر بزرگ بهتر کار کند.

3-

در این بخش سعی میکنیم با توجه به خواسته های مساله، با Loss های مناسب، شبکه CycleGAN را پیاده سازی کنیم. برای اینکار از دیتاست monet2photo استفاده مینماییم.

برای اینکار ابتدا Requirement های اولیه را نصب میکنیم:

```
Requirement already satisfied: torch>=1.4.0 in /usr/local/lib/python3.7/dist-packages (from -r requirements.txt (line 1)) (1.9.0+cu102)
Requirement already satisfied: torchvision>=0.5.0 in /usr/local/lib/python3.7/dist-packages (from -r requirements.txt (line 2)) (0.10.0+cu102)
Collecting dominate>=2.4.0
  Downloading dominate-2.6.0-py2.py3-none-any.whl (29 kB)
Collecting visdom>=0.1.8.8
  Downloading visdom-0.1.8.9.tar.gz (676 kB)
     |████████████████████████████████| 676 kB 9.0 MB/s
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.7/dist-packages (from torch>=1.4.0->-r requirements.txt (line 1)) (3.7.4.3)
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from torchvision>=0.5.0->-r requirements.txt (line 2)) (1.19.5)
Requirement already satisfied: pillow>=5.3.0 in /usr/local/lib/python3.7/dist-packages (from torchvision>=0.5.0->-r requirements.txt (line 2)) (7.1.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.7/dist-packages (from visdom>=0.1.8.8->-r requirements.txt (line 4)) (1.4.1)
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages (from visdom>=0.1.8.8->-r requirements.txt (line 4)) (2.23.0)
Requirement already satisfied: tornado in /usr/local/lib/python3.7/dist-packages (from visdom>=0.1.8.8->-r requirements.txt (line 4)) (5.1.1)
Requirement already satisfied: pyzmq in /usr/local/lib/python3.7/dist-packages (from visdom>=0.1.8.8->-r requirements.txt (line 4)) (22.1.0)
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from visdom>=0.1.8.8->-r requirements.txt (line 4)) (1.15.0)
Collecting jsonpatch
  Downloading jsonpatch-1.32-py2.py3-none-any.whl (12 kB)
Collecting torchfile
  Downloading torchfile-0.1.0.tar.gz (5.2 kB)
Collecting websocket-client
  Downloading websocket_client-1.1.0-py2.py3-none-any.whl (68 kB)
     |████████████████████████████████| 68 kB 8.2 MB/s
Collecting jsonpointer>=1.9
  Downloading jsonpointer-2.1-py2.py3-none-any.whl (7.4 kB)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-packages (from requests->visdom>=0.1.8.8->-r requirements.txt (line 4)) (2021.5.30)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib/python3.7/dist-packages (from requests->visdom>=0.1.8.8->-r requirements.txt (line 4)) (1.24.3)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packages (from requests->visdom>=0.1.8.8->-r requirements.txt (line 4)) (3.0.4)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (from requests->visdom>=0.1.8.8->-r requirements.txt (line 4)) (2.10)
Building wheels for collected packages: visdom, torchfile
  Building wheel for visdom (setup.py) ... done
  Created wheel for visdom: filename=visdom-0.1.8.9-py3-none-any.whl size=655249 sha256=0cc59f35862ff401a66a39e092a32cc4902964a4d53446e8252ab0c69508bf7e
  Stored in directory: /root/.cache/pip/wheels/2d/d1/9b/cde923274eac9cbb6ff0d8c7c72fe30a3da9095a38fd50bbf1
  Building wheel for torchfile (setup.py) ... done
  Created wheel for torchfile: filename=torchfile-0.1.0-py3-none-any.whl size=5710 sha256=88dd834e87371b86d722162c784932663c29ad69997ba5b9bea20ba3a2c05d8b
  Stored in directory: /root/.cache/pip/wheels/ac/5c/3a/a80e1c65880945c71fd833408cd1e9a8cb7e2f8f37620bb75b
Successfully built visdom torchfile
Installing collected packages: jsonpointer, websocket-client, torchfile, jsonpatch, visdom, dominate
Successfully installed dominate-2.6.0 jsonpatch-1.32 jsonpointer-2.1 torchfile-0.1.0 visdom-0.1.8.9 websocket-client-1.1.0
```

شکل ۲-۲ نصب Requirement های مورد نیاز

سپس مجموعه داده مربوطه را لود میکنیم برای اینکار از داده های Kaggle که monet2photo را نیز شامل میشود استفاده میکنیم:

```
GCS_PATH = KaggleDatasets().get_gcs_path('monet-gan-getting-started')

MONET_FILENAMES = tf.io.gfile.glob(str(GCS_PATH + '/monet_tfrec/*.tfrec'))
PHOTO_FILENAMES = tf.io.gfile.glob(str(GCS_PATH + '/photo_tfrec/*.tfrec'))

def count_data_items(filenames):
    n = [int(re.compile(r"-([0-9]*)\.").search(filename).group(1)) for filename in filenames]
    return np.sum(n)

n_monet_samples = count_data_items(MONET_FILENAMES)
n_photo_samples = count_data_items(PHOTO_FILENAMES)

print(f'Monet TFRecord files: {len(MONET_FILENAMES)}')
print(f'Monet image files: {n_monet_samples}')
print(f'Photo TFRecord files: {len(PHOTO_FILENAMES)}')
print(f'Photo image files: {n_photo_samples}')
```

لود مجموعه داده monet2photo

همچنین از آنجا که CycleGAN شبکه سنگینی است، میبایست که با استفاده از TPU آن را اجرا نمود. پس در یک Try-Except عملکرد و وضعیت TPU را میسنجیم.

```
try:
    tpu = tf.distribute.cluster_resolver.TPUClusterResolver()
    print(f'Running on TPU {tpu.master()}')
except ValueError:
    tpu = None

if tpu:
    tf.config.experimental_connect_to_cluster(tpu)
    tf.tpu.experimental.initialize_tpu_system(tpu)
    strategy = tf.distribute.experimental.TPUStrategy(tpu)
else:
    strategy = tf.distribute.get_strategy()


REPLICAS = strategy.num_replicas_in_sync
print(f'REPLICAS: {REPLICAS}')
AUTO = tf.data.experimental.AUTOTUNE
```

شکل 2-4 عملکرد و وضعیت TPU

سپس سعی میکنیم با استفاده از عملکرد گفته شده استفاده کنیم و Generator را با 3 کانال خروجی تشکیل میدهیم این شبکه یک down-stack دارد و همچنین up-stack که در کد مشاهده میشود:

```
down_stack = [
    downsample(64, 4, apply_instancenorm=False), # (bs, 128, 128, 64)
    downsample(128, 4),                          # (bs, 64, 64, 128)
    downsample(256, 4),                          # (bs, 32, 32, 256)
    downsample(512, 4),                          # (bs, 16, 16, 512)
    downsample(512, 4),                          # (bs, 8, 8, 512)
    downsample(512, 4),                          # (bs, 4, 4, 512)
    downsample(512, 4),                          # (bs, 2, 2, 512)
    downsample(512, 4),                          # (bs, 1, 1, 512)
]

up_stack = [
    upsample(512, 4, apply_dropout=True), # (bs, 2, 2, 1024)
    upsample(512, 4, apply_dropout=True), # (bs, 4, 4, 1024)
    upsample(512, 4, apply_dropout=True), # (bs, 8, 8, 1024)
    upsample(512, 4),                     # (bs, 16, 16, 1024)
    upsample(256, 4),                     # (bs, 32, 32, 512)
    upsample(128, 4),                     # (bs, 64, 64, 256)
    upsample(64, 4),                      # (bs, 128, 128, 128)
]
```

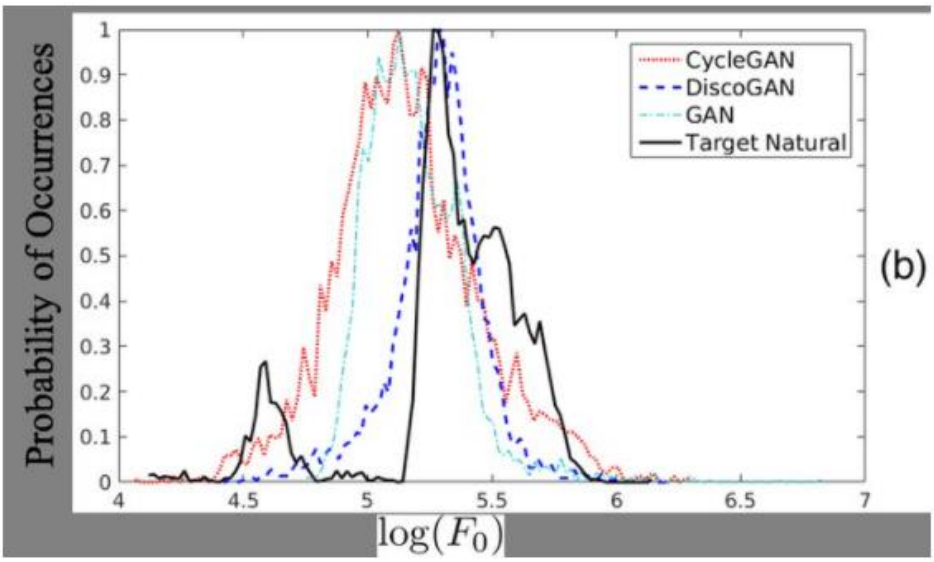
و همچنین توابع متناسب برای Initialize کردن و ساخت و پرداختن تصویر جعلی.

از طرفی ساختار Discriminator را داریم که به مانند زیر میباشد:

```
def discriminator_fn():
    initializer = tf.random_normal_initializer(0., 0.02)
    gamma_init = tf.keras.initializers.RandomNormal(mean=0.0, stddev=0.02)

    inp = L.Input(shape=[HEIGHT, WIDTH, CHANNELS], name='input_image')

    x = inp

    down1 = downsample(64, 4, False)(x) # (bs, 128, 128, 64)
    down2 = downsample(128, 4)(down1) # (bs, 64, 64, 128)
    down3 = downsample(256, 4)(down2) # (bs, 32, 32, 256)

    zero_pad1 = L.ZeroPadding2D()(down3) # (bs, 34, 34, 256)
    conv = L.Conv2D(512, 4, strides=1,
                    kernel_initializer=initializer,
                    use_bias=False)(zero_pad1) # (bs, 31, 31, 512)

    norm1 = tfa.layers.InstanceNormalization(gamma_initializer=gamma_init)(conv)

    leaky_relu = L.LeakyReLU()(norm1)

    zero_pad2 = L.ZeroPadding2D()(leaky_relu) # (bs, 33, 33, 512)

    last = L.Conv2D(1, 4, strides=1,
                    kernel_initializer=initializer)(zero_pad2) # (bs, 30, 30, 1)

    return Model(inputs=inp, outputs=last)
```

و در آن همانند مرحله گفته شده از ملزومات گفته شده استفاده شده ات و از تابع LeakyReLU استفاده میکند. سپس شبکه نهای را طراحی میکنیم که حاوی Generator 2 است و Discriminator 2 است.

این نتیجه مورد انتظار ما نیز بود، چرا که در مقاله های دیگر هم، به نتایج خوبی رسیده اند که نشان میدهد عملکرد CycleGAN بسیار متناسب است. نمونه ای از آن را در شکل زیر میبینید که در مقاله **Effectiveness of Cross-Domain Architectures for Whisper-to-Normal Speech Conversion** هم به آن اشاره شده است.



پس از گذشت 1 ایپاک و 6200 ایتریشن در این ایپاک و همچنین 1013 ایتریشن از ایپاک دوم، یکی از خروجی هایی که شبکه تولید کرده را نمایش می دهیم:

با استفاده از خطای cycle consistency

شکل 2-4-: خروجی fakeتولیدی شبکه

که تصویر اصلی به صورت زیر است:

شکل 2-5--: خروجی اصلی

سوال 3 – آشنایی با مقالات مربوط

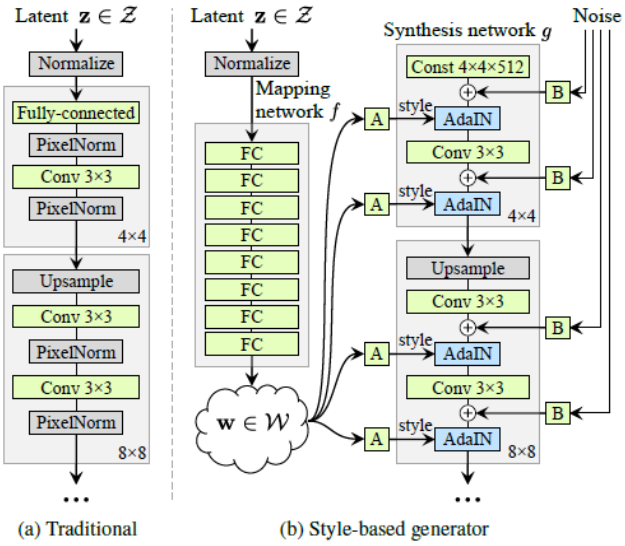
در این سوال به بررسی ساختاری و ماهیتی مقاله styleGAN می پردازیم .

پیشینه:

ابتدا باید اشاره کنیم که styleGAN از styletransfer الهام گرفته شده است با این تفاوت که معماری جدید به صورت اتوماتیک آموزش می بیند(این کار به صورت unsupervised انجام می شود.) generator جدید معیار های توزیع را بهبود بخشیده و همچنین فاکتور های پنهان در variation را بهتر جدا مینماید.

برای این جدایی دو روش جدید اتوماتیک (که برای هر generator ای قابل اعمال هستند) معرفی شده است و در نهایت یک دیتاست با کیفیت و با قدرت تغییر پذیری بالا از صورت انسان ها ندارک دیده شده است.لازم به ذکر است که تمام تغییرات ایجاد شده در generator اعمال شده اند(نسبت به GAN معمولی)

معماری شبکه:



شکل 3-1 معماری شبکه StyleGAN

بخش های معماری این شبکه را میتوان به 3 بخش اصلی تقسیم کرد:

Mapping Network-1

شبکه نگاشتی یک شبکه MLP است که از 8 لایه تشکیل شده است. نقش آن این است که ورودی مخفی را (z) به یک فضای مخفی میانی به نام W انکود کند. ورودی مخفی باید چگالی احتمال داده های آموزش را داشته باشد. بر خلاف معماری اولیه gan که در آن یک بردار مخفی به generator توسط لایه ورودی اعمال میشد، در styleGAN از یک ثابت آموزش داده شده استفاده می شود و لایه ورودی از معماری حذف می شود.. بردار W به

ماژول Affine transformationاعمال می شود که خروجی y (که استایل مطلوب است) خروجی این ماژول می باشد. این خروجی ها که شامل y و ویژگی های کانوولوشن های قبلی x هستند ، به عنوان ورودی به لایه های adaptive instance normalization داده می شوند. علاوه بر این ها یک بردار ثابت به سایز 4x4x512 داریم که بردار سنتر g را تغذیه می کند.(دارای 18 لایه)

2- The Affine Transformation (A) and adaptive instance Normalization(AdaIN)

AdaIN یک تکنیک نرمالیزیشن است که از batch normalization الهام گرفته است. این تکنیک در generative image modeling ها نیز کاربرد دارد.

در الگوریتم batch-normalizationداریم:

$$\mathbf{BN}(x) = \gamma \left(\frac{x - \mu(x)}{\sigma(x)} \right) + \beta$$

شکل 3-2 نرمال سازی به صورت Batch

این نرمال سازی به محاسبه میانگین و انحراف از معیار می پردازد و همچنین بک سری پارامتر های scaling مثل بتا و گاما دارد.

حال آنکه Instance Normalization از batch normalization برای هر نمونه استفاده می کند. این الگوریتم هر نمونه در یک batch را جداگانه در نظر گرفته و میانگین و انحراف از معیار آن را حساب میکند که باعث بالاتر رفتن توانایی شبکه در یادگیری استایل ها می شود.

AdaIN نیز یک extension از instance normalization است که به صورت زیر تعریف می شود:

$$\mathbf{AdaIN}(x, y) = \sigma(y) \left(\frac{x - \mu(x)}{\sigma(x)} \right) + \mu(y)$$

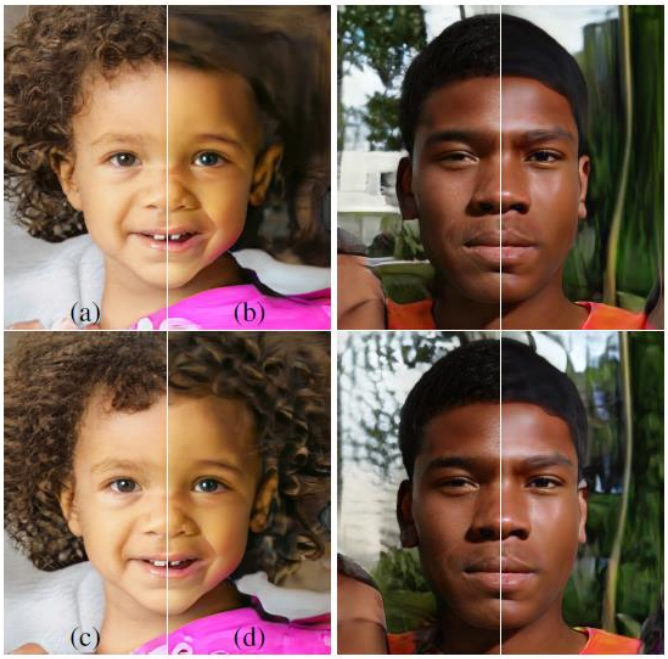
شکل 3-3 نرمال سازی با استفاده از Instance با AdaIN

الگوریتم دو ورودی x و ورودی استایلy را می گیرد. X نشانگر ویژگی های لایه قبل و y نشانگر affine transformation module است.Y نقش کنارلی برای تصاویر خروجی را دارد. قابل ذکر است که AdaIN هیچ پارامتر آموزش پذیری ندارد و به صورت تطبیقی ، پارامتر های استایل ورودی را محاسبه می کند.

3- The noise vector

این بردار توسط لایه affine transformation به شبکه اضافه می شوند که شامل یک تک کانال برای تصاویر بدون کورلیشن نویز گوسی میباشد. این بردار به خروجی هر لایله 3x3 کانوولوشنی اضافه می شود.

بردار نویز برای این است که جزییات stochastic را در تصاویر شبکه اضافه کند. برای همین استفاده از نویز دقت شبکه را برای متغیر هایی مثل موهای شانه شده، منافذ پوستی یا ریش ها بالا میبرد.



شکل 3-4 گوشه ای از عملکرد شبکه CycleGAN در زمینه پوست و حالت مو

تابع هزینه:

از دو تابع loss function در این مقاله استفاده شده است که برای دیتاست CELEBA-HQ از WGAN-GP استفاده می شود و برای دیتاست FFHQ از WGAN-GP برای configuration A که یک loss بدون اشباع است، استفاده می شود.البته این موارد برای بهترین نتیجه گیری است ولی در نتیجه گیری نهایی تغییری در lossfunction ایجاد نشده است.

دیتاست FFHQ:

این دیتا ست شامل 70000 عکس کیفیت بالا (1024x1024) ست که شامل گوناگونی های بسیار بالا از جمله سن، نژاد، نور، background و پوزیشن های مختلف است.

نمونه ای از دیتا ست:

شکل 3-5 نمونه ای از داده ها در مجموعه داده

این یک دیتاست پابلیک است که از سایت Flickr برداشته شده و ممکن است بایاس هایی در قسمت های عینک و یا کلاه ها داشته باشد. همچنین همه عکس ها aligned و cropشده هستند.

دست آورد نویسندگان(نتایج):

1-Mixing regularization

این ایده یک ایده جدید معرفی شده در این پیپر است که به جای فرستادن فقط یک لایه مخفی z ، 2 لایه z1و z2 توسط شبکه مپینگ به بردار w فرستاده شده و w1 و w2 حاصل می گردند. این خروجی ها در هر iteration رندوم هستند و اثبات می شود که این تکنیک مانع از این میشود که شبکه تصور کند که استایل های همسایه، با هم correlation دارند.

در زیر نتایج mixing regularization را به شبکه می بینیم:

Mixing regularization	Number of latents during testing			
	1	2	3	4
E 0%	4.42	8.22	12.88	17.41
50%	4.41	6.10	8.71	11.61
F 90%	**4.40**	**5.11**	6.88	9.03
100%	4.83	5.17	**6.63**	**8.40**

شکل 3-6 در زیر نتایج mixing regularization

2- تاثیر تغییر دادن نویز

تاثیر تغییر دادن نویز روی شکل های خروجی نیز بررسی شده است. به طوری که متوجه شدند با تغییر نویز، یک سری تغییر روی مثلا مدل موی افراد ایجاد شد در حالیکه بقیه ویژگی های تصویر دست نخورده باقی ماندند.

این نشان دهنده این است که تغییر نویز تاثیر خیلی بزرگتری روی تصاویر خروجی نسبت به GAN های عادی دارد که ویژگی های کلی یکسان باقی می مانند ولی ویژگی های جزیی مثل مو تغییر میکنند. این در حالیست که در GAN های عادی ، با تغییر نویز کل تصویر به طور کامل تغییر میکند.

(a) Generated image (b) Stochastic variation (c) Standard deviation

شکل 3-7 تاثیر تغییر دادن نویز و تغییر در جزییات موها (صرفا)

3- Frechet inception distance (FID) score

Method	CelebA-HQ	FFHQ
A Baseline Progressive GAN [30]	7.79	8.04
B + Tuning (incl. bilinear up/down)	6.11	5.25
C + Add mapping and styles	5.34	4.85
D + Remove traditional input	5.07	4.88
E + Add noise inputs	**5.06**	4.42
F + Mixing regularization	5.17	**4.40**

شکل 3-8 تاثیر عملکرد های متفاوت در مقادیر CelebA-HQ و FFHQ

FID score یک معیار و محک برای GAN ها است که هر چه کمتر باشد بهتر بودن مدل را نشان می دهد. در اینجا به مقایسه دیتاست CELEBA-HQ و FFHQ پرداخته شده است. متوجه شدند که هر چقدر طراحی های جدیدی به معماری GAN اضافه شود، این معیار نیز بهبود می یابد. همچنین مشخص شد که دیتاست FFHQ امتیاز FID بهتری از دیتاست CELEB-HQ دارد.

مسیر پیشنهادی برای ادامه تحقیقات:

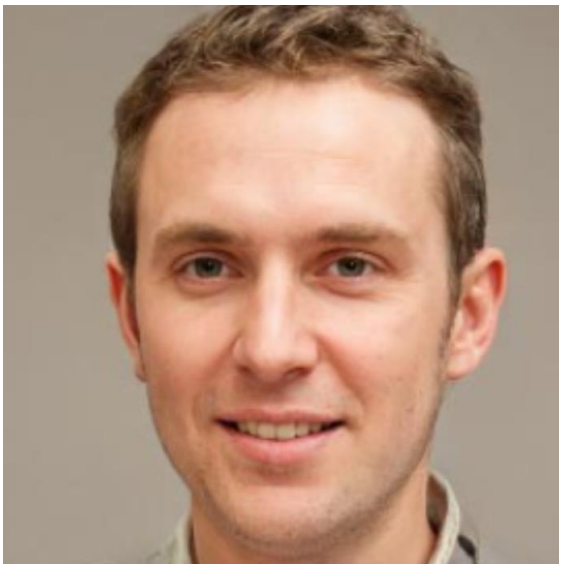
متوجه شدند که از معیار میانگین طول مسیر میتوان برای regularization در هنگام آموزش و شاید هم بتوات از بعضی از معیار های جدایی پذیری خطی به صورت یک پارچه استفاده کرد. انتظار می رود که در آینده روش هایی برای شکل دهی فضای نهان متوسط به صورت مستقیم و در طول آموزش تدارک دیده شود که می تواند راهکار های جالبی را در آینده رقم بزنند.

برای پیاده سازی از یک مدل pretrained استفاده می کنیم که نتیجه خروجی برای یک عکس به صورت زیر می شود:

در تصویر زیر ورودی به شبکه اعمال شده است، و قابل مشاهده میباشد:

شکل 3-9 ورودی به شبکه داده شده

حال با استفاده از تمامی نکات ارائه شده، سعی مینماییم Style عکس را عوض کنیم، برای اینکار ورودی منظور شده را کمی جوان تر میکنیم و عکس خروجی را در درایومان save می کنیم که نتیجه به صورت زیر است:



شکل 3-9 خروجی جوان شده

یکی دیگر ازتغییراتی که روی این عکس می دهیم ، تغییر جنسیت عکس می باشد که این را نیز در درایو ذخیره کرده و به صورت زیر نمایش می دهیم:

شکل 3-9 خروجی جوان شده