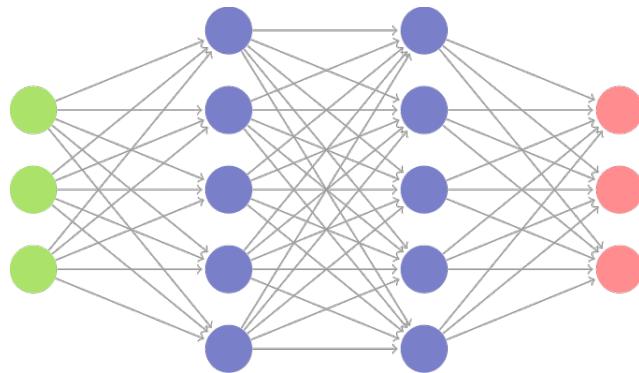




به نام خدا

دانشگاه تهران  
دانشکده مهندسی برق و کامپیوتر



Neural Networks

Project 3

نام و نام خانوادگی	علیرضا محمدی	محمدعلی شاکردرگاه
شماره دانشجویی	۸۱۰۱۹۹۳۶۵	۸۱۰۱۹۶۴۸۷
تاریخ ارسال گزارش	۱۴۰۰/۰۵/۰۲	

## فهرست مطالب

۱	سوال اول	
۴	پیاده‌سازی VAE	۱.۱
۴	۱.۱.۱ انکودر	۱.۱.۱
۵	۲.۱.۱ دیکودر	۲.۱.۱
۶	۳.۱.۱ مدل کلی	۳.۱.۱
۶	۴.۱.۱ نتایج آموزش شبکه	۴.۱.۱
۷	تابع خطأ	۲.۱
۸	Reparameterization Trick	۳.۱
۹	بررسی دیتا در فضای latent	۴.۱
۱۰	ایجاد دیتا بر اساس نمونه‌های فضای latent	۵.۱
۱۰	بررسی کیفیت شبکه	۶.۱
۱۱	پیاده‌سازی CVAE	۷.۱
۱۲	۱.۷.۱ ساختار انکودر	
۱۲	۲.۷.۱ ساختار دیکودر	
۱۳	۳.۷.۱ مدل کلی	
۱۳	۴.۷.۱ نتیجه آموزش	
۱۳	بررسی دیتا در فضای latent	۸.۱
۱۵	ایجاد دیتا بر اساس نمونه‌های فضای latent	۹.۱
۱۵	بررسی کیفیت شبکه	۱۰.۱
۱۷		سوال دوم
۱۷	ساختار شبکه	۱.۲
۱۸	PatchGAN	۲.۲
۱۸	پیاده‌سازی با استفاده از unet	۳.۲
۱۸	generator	۱.۳.۲
۱۹	discriminator	۲.۳.۲
۲۰	۳.۳.۲ ساختار کلی شبکه و آموزش آن	
۲۱	۴.۳.۲ آموزش شبکه	
۲۵	۵.۳.۲ نمودار خطای شبکه	
۲۸		سوال سوم
۲۸	خلاصه کلی مقاله	۱.۳
۲۸	۱.۱.۳ تحلیل ساز و کار شبکه	
۳۰	تابع هزینه	۲.۳
۳۰	۱.۲.۳ توابع هزینه مرحله اول StackGAN برای Generator و Discriminator	
۳۱	۲.۲.۳ توابع هزینه مرحله دوم StackGAN برای Generator و Discriminator	
۳۱	دیتاست استفاده شده	۳.۳
۳۲	پیشینه مقاله	۴.۳
۳۲	دستاوردهای نویسنده‌گان مقاله	۵.۳

۳۲	مسیر پیشنهادی برای ادامه تحقیقات . . . . .	۶.۳
۳۴	تست شبکه . . . . .	۷.۳
۳۴	۱.۷.۳ آموزش مرحله اول شبکه تا ۱۰ ایپاک . . . . .	۱۰
۳۴	۲.۷.۳ ادامه آموزش شبکه . . . . .	۲۰

## ۱ سوال اول

### ۱.۱ پیاده‌سازی VAE

در این بخش می‌خواهیم به جزئیات پیاده‌سازی این شبکه بپردازیم. همانطور که در صورت سوال هم ذکر شده است این شبکه از دو بخش انکودر و دیکودر تشکیل می‌شود که از این حیث مشابه با اتوانکودرهای معمولی است، با این تفاوت که به جای ایجاد یک فضای بعد کوچک ثانویه به صورت گسسته، دیتاها را با استفاده از انکودر به یک فضای توزیع منتقل می‌کنیم که این فضا برای هر کلاس با یک میانگین و واریانس تعریف می‌شود.

#### ۱.۱.۱ انکودر

با توجه به آنکه می‌خواهیم از این شبکه برای داده‌های mnist استفاده کنیم، ساختار شبکه را به صورت کانولوشنی طراحی می‌کنیم. ورودی این شبکه یک تصویر سیاه‌وسفید با ابعاد ۲۸ در ۲۸ است و لایه‌های انکودر به صورت زیر می‌باشد:

۱. یک لایه کانولوشنی با ۳۲ فیلتر و اندازه‌ی کرنل ۳ در ۳ و استراید ۲ که در آن از تابع فعال‌ساز ReLU استفاده شده است. پس از این لایه، یک لایه dropout قرار گرفته است.
۲. یک لایه کانولوشنی با ۶۴ فیلتر و اندازه‌ی کرنل ۳ در ۳ که در آن از تابع فعال‌ساز ReLU استفاده شده است. پس از این لایه، یک لایه dropout قرار گرفته است.
۳. یک لایه کانولوشنی با ۱۲۸ فیلتر و اندازه‌ی کرنل ۳ در ۳ و استراید ۲ که در آن از تابع فعال‌ساز ReLU استفاده شده است. خروجی این لایه با عبور از یک لایه Flatten به یک بردار ۶۲۷۲ عضوی تبدیل می‌شود.

۴. یک لایه Fully Connected با ۶۴ نورون و تابع فعال‌ساز ReLU

۵. یک لایه Fully Connected با ۱۶ نورون و تابع فعال‌ساز ReLU

۶ در این بخش دو لایه Fully Connected به موازات یکدیگر، یکی به منظور ایجاد میانگین در فضای ثانویه و دیگری برای ایجاد واریانس استفاده شده است که ابعاد آن‌ها با توجه به خواسته‌ی سوال برابر با ۲ انتخاب شده است.

با توجه به ملاحظاتی که به منظور سادگی فرآیند Back Propagation وجود دارد (در ادامه این موضوع را شرح خواهم داد) از یک لایه نمونه‌برداری استفاده می‌کنیم که با توجه به میانگین و واریانس ایجاد شده در خروجی انکودر، خروجی سومی را به صورت یک دیتای نمونه‌برداری شده از این توزیع تعریف می‌کند.

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[None, 28, 28, 1]	0	
conv2d (Conv2D)	(None, 14, 14, 32)	320	input_1[0][0]
dropout (Dropout)	(None, 14, 14, 32)	0	conv2d[0][0]
conv2d_1 (Conv2D)	(None, 14, 14, 64)	18496	dropout[0][0]
dropout_1 (Dropout)	(None, 14, 14, 64)	0	conv2d_1[0][0]
conv2d_2 (Conv2D)	(None, 7, 7, 128)	73856	dropout_1[0][0]
flatten (Flatten)	(None, 6272)	0	conv2d_2[0][0]
dense (Dense)	(None, 64)	401472	flatten[0][0]
dense_1 (Dense)	(None, 16)	1040	dense[0][0]
z_mean (Dense)	(None, 2)	34	dense_1[0][0]
z_log_var (Dense)	(None, 2)	34	dense_1[0][0]
sampling (Sampling)	(None, 2)	0	z_mean[0][0] z_log_var[0][0]

Total params: 495,252  
Trainable params: 495,252  
Non-trainable params: 0

شکل ۱: ساختار شبکه انکودر

### ۲.۱.۱ دیکودر

بخش دیکودر شبکه از خروجی نمونه‌برداری شده از فضای latent استفاده می‌کند. در نتیجه ورودی این بخش یک بردار در فضای دوبعدی است. لایه‌های شبکه دیکودر به صورت زیر است:

۱. یک لایه Fully Connected با ۶۴ نورون و تابع فعال‌ساز ReLU.
۲. یک لایه Fully Connected با ۶۲۷۲ نورون و تابع فعال‌ساز ReLU که خروجی آن به یک ماتریس ۷ در ۷ با عمق ۱۲۸ تبدیل می‌شود.
۳. یک لایه کانولوشنی معکوس با ۱۲۸ فیلتر و اندازه کرنل ۳ در ۳ و استراید ۲ که در آن از تابع فعال‌ساز ReLU استفاده شده‌است.
۴. یک لایه کانولوشنی معکوس با ۶۴ فیلتر و اندازه کرنل ۳ در ۳ که در آن از تابع فعال‌ساز ReLU استفاده شده‌است.
۵. یک لایه کانولوشنی معکوس با ۳۲ فیلتر و اندازه کرنل ۳ در ۳ و استراید ۲ که در آن از تابع فعال‌ساز ReLU استفاده شده‌است.
۶. برای لایه خروجی از یک لایه کانولوشنی معکوس با ۱ فیلتر و اندازه کرنل ۳ در ۳ و تابع فعال‌سازی Sigmoid استفاده شده‌است که نهایتاً یک تصویر سیاه‌وسفید با ابعاد ۲۸ در ۲۸ را ایجاد می‌کند.

Model: "decoder"		
Layer (type)	Output Shape	Param #
input_23 (InputLayer)	[None, 2]	0
dense_58 (Dense)	(None, 64)	192
dense_59 (Dense)	(None, 6272)	407680
reshape_13 (Reshape)	(None, 7, 7, 128)	0
conv2d_transpose_50 (Conv2DT)	(None, 14, 14, 128)	147584
conv2d_transpose_51 (Conv2DT)	(None, 14, 14, 64)	73792
conv2d_transpose_52 (Conv2DT)	(None, 28, 28, 32)	18464
conv2d_transpose_53 (Conv2DT)	(None, 28, 28, 1)	289
Total params:	648,001	
Trainable params:	648,001	
Non-trainable params:	0	

شکل ۲: ساختار شبکه دیکودر

### ۳.۱.۱ مدل کلی

اکنون که مدل شبکه‌های انکودر و دیکودر را طراحی کردیم می‌توانیم با استفاده از این دو شبکه یک مدل کلی VAE را طراحی کنیم. برای این منظور یک کلاس را بر اساس مدل‌های کتابخانه keras تعریف می‌کنیم و متدهای مربوط به مسیر back prop و feed forward را برای آن پیاده‌سازی می‌کنیم. برای آموزش شبکه باید متدهای train step مربوط به این کلاس را پیاده‌سازی کنیم. در این بخش ابتدا دیتا را به انکودر می‌دهیم و سپس خروجی مربوط به میانگین، واریانس و دیتای نمونه‌برداری شده از این فضای دریافت می‌کنیم. سپس دیتای نمونه‌برداری شده را به دیکودر می‌دهیم و خروجی کلی شبکه را محاسبه می‌کنیم. در قدم بعدی، خطای شبکه را محاسبه می‌کنیم و سپس با محاسبه گرادیان، روند بروزرسانی وزن‌ها را به صورت زیر انجام می‌دهیم:

```
grads = tape.gradient(total_loss, trainable_weights)
self.optimizer.apply_gradients(zip(grads, trainable_weights))
self.total_loss_tracker.update_state(total_loss)
self.reconst_loss_tracker.update_state(reconst_loss)
self.kl_loss_tracker.update_state(kl_loss)
```

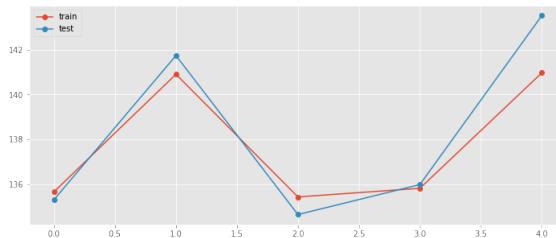
همچنین در این بخش خطای مربوط به دیتای آموزشی را بروزرسانی می‌کنیم تا این دیتا در روند آموزش شبکه ذخیره شود.

به منظور تست شبکه بر روی دیتای تست، باید متدهای test step را پیاده‌سازی کنیم. این روند تقریباً مشابه مرحله آموزش است با این تفاوت که محاسبه گرادیان و بروزرسانی وزن‌ها در این مرحله انجام نمی‌شود. در این بخش پس از محاسبه خروجی انکودر و دیکودر، خطای شبکه برای دیتای تست را محاسبه می‌کنیم و تاریخچه این مقادیر را بروزرسانی می‌کنیم.

برای اینکه بتوانیم خروجی شبکه را محاسبه کنیم، باید تابع call را پیاده‌سازی کنیم. در این تابع، دیتا به شبکه انکودر داده می‌شود و سپس با استفاده از دیتای نمونه‌برداری شده از فضای ثانویه انکودر، خروجی نهایی را با دیکودر محاسبه می‌کنیم. با استفاده از این تابع می‌توانیم خروجی شبکه را به ازای هر دیتا محاسبه کنیم.

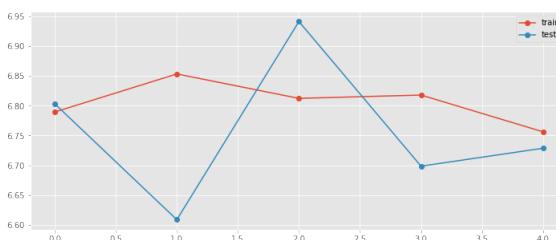
### ۴.۱.۱ نتایج آموزش شبکه

خطای بازسازی برای دیتای تست و آموزشی به صورت زیر است:



شکل ۳: خطای بازسازی

خطای kl divergence برای دیتای تست و آموزشی به صورت زیر است:



شکل ۴: خطای kl divergence

## ۲.۱ توابع خطا

این شبکه همانند شبکه‌ای اتوانکودر معمولی به دنبال این است که دیتایی را از فضای اصلی با استفاده از انکودر، به فضای latent ببرد و سپس با استفاده از دیکودر آن را مجدداً به فضای اصلی برگرداند. از این رو برای این شبکه نیز اولین خطایی که تعریف می‌شود، خطای reconstruction می‌باشد. به این صورت که دیتای  $x$  را با استفاده از انکودر به یک توزیع تصادفی منتقل می‌کنیم و سپس یک نمونه از این توزیع را با استفاده از دیکودر به فضای اصلی بر می‌گردانیم که آن را با  $\tilde{x}$  نشان می‌دهیم. خطای مربوط به بازسازی دیتا به صورت زیر تعریف می‌شود:

$$\ell_{reconstruction} = \sum x \log(\tilde{x}) + (1 - x) \log(1 - \tilde{x})$$

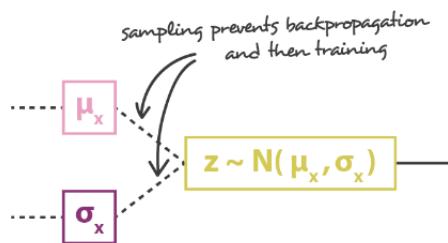
در این پروژه، ما از خطای binary crossentropy استفاده کردیم، همچنین می‌توان از خطای MSE یا سایر خطاهای مناسب برای بررسی میزان بازیابی تصویر استفاده کرد. در این مدل ما به نوعی به دنبال regularize کردن فضای latent هستیم، به نحوی که مدل دیتاهای را به نوعی در این فضا انکود کند تا شرط پیوستگی و کامل بودن برقرار شود و توزیع‌ها با یکدیگر همپوشانی داشته باشند. البته واضح است که این regularize کردن فضای latent هرچه بیشتر باشد، هزینه‌ی زیاد شدن خطای بازسازی را به دنبال دارد. خطای دومی را که به منظور regularize کردن فضای latent استفاده می‌شود، خطای Kullback-Leibler نام دارد. این خطای در واقع میزان تفاوت بین دو توزیع احتمالاتی را کمینه می‌کند. برای مثال اگر توزیع دیتا در فضای latent را به صورت  $P(z|x)$  در نظر بگیریم، تفاوت این توزیع با توزیع نرمال بر اساس تعریف تابع خطای kl زیر است:

$$\ell_{kl} = \frac{1}{2} \left( \sum_k \Sigma(X) + \sum_k \mu^2(X) - \sum_k 1 - \log \prod_k \Sigma(X) \right)$$

تعریف این تابع خطای برای شبکه باعث می‌شود تا بتوانیم فضای latent را به گونه‌ای regularize کنیم تا به هدفی که برای تولید دیتا بر اساس این فضا داشتیم، برسیم.

### Reparameterization Trick ۳.۱

ساختار کلی شبکه همانطور که تاکنون بحث کردیم از تجمعی یک انکودر و دیکودر تشکیل شده است. آنچه اهمیت دارد این است که این شبکه در نهایت باید به روش‌های متداول از جمله GD قابل آموزش باشد. نکته‌ی مهم که مشکل اساسی را در مسیر back propagation برای ما ایجاد می‌کند این است که در لایه نهایی انکودر ما یک توزیع تصادفی خواهیم داشت. بنابراین در نمونه برداری از این توزیع باید دقت کنیم زیرا در مسیر برگشت باید بتوانیم خطای از این لایه عبور بدھیم و وزن‌ها را بروز کنیم.



شکل ۵: نمونه برداری بدون Reparameterization Trick

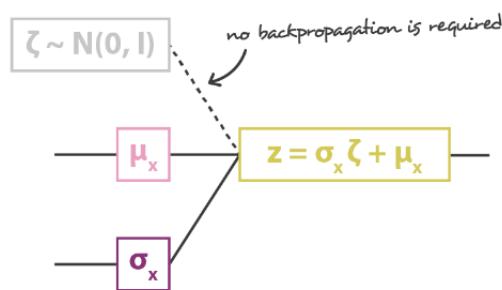
برای آنکه بتوانیم خطای از این لایه عبور بدھیم و وزن‌ها را بروز کنیم از تکنیکی استفاده می‌کنیم که آن را Reparameterization می‌نامد، به این صورت که به جای نمونه برداری از این توزیع، خروجی را به صورت زیر تعریف می‌کنیم:

$$z = \sigma_x \zeta + \mu_x$$

در عبارت فوق،  $\mu_x$  و  $\sigma_x$  به ترتیب میانگین و واریانس خروجی از انکودر هستند و ترم  $\zeta$  یک دیتای نمونه برداری شده از توزیع نرمال با میانگین صفر و واریانس ۱ است. با اینکار، علاوه بر آنکه از میانگین و واریانس خروجی انکودر استفاده کرده‌ایم و نمونه‌ی ایجاد شده نهایتاً بر اساس این پارامترها می‌باشد، ولی بخش تصادفی آن را جدا کرده و از یک توزیع نرمال استفاده می‌کنیم. در این صورت بخش تصادفی که با پارامتر

$$\zeta \sim N(0, 1)$$

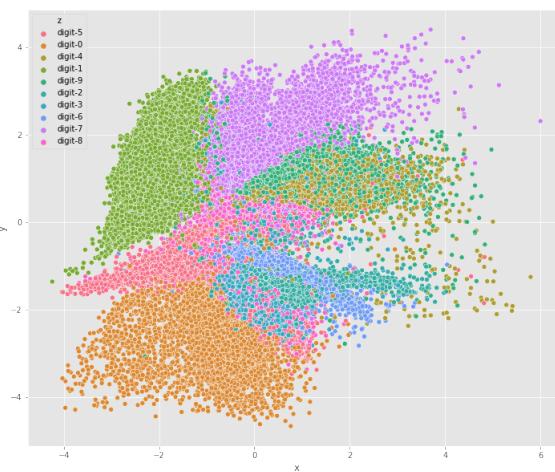
نیازی به بروزرسانی در مسیر برگشت ندارد. بنابراین تنها مسیرهای مربوط به میانگین و واریانس باقی می‌ماند که هیچ نمونه برداری تصادفی از آن‌ها رخ نمی‌دهد و می‌توانیم به سادگی مسیر برگشت را در محاسبه خطای بروزرسانی وزن‌ها طی کنیم.



شکل ۶: Reparameterization Trick

#### ۴.۱ بررسی دیتا در فضای latent

اکنون می‌خواهیم با استفاده از بخش انکودر شبکه آموزش دیده شده، دیتا را به فضای latent منتقل کنیم. چون این فضای دوبعدی است، می‌توانیم این فضا را به سادگی ترسیم کرده و بررسی کنیم. برای مشاهده توزیع دیتاهای هر کلاس در این فضای دو بعدی، هر دیتا را بر اساس برچسب واقعی آن و با یک رنگ منحصر به فرد ترسیم کردایم:



شکل ۷: دیتا در فضای latent

برای ترسیم دیتا در فضای latent از مجموعه دیتاهای آموزشی و تست استفاده کردایم. همانگونه که مشاهده می‌شود این ساختار مطرح شده در شبکه VAE باعث شده است تا گستینگی فضای ثانویه در اتوانکودر بر طرف شود. از همین رو برای هر کلاس از دیتا، یک ناحیه پیوسته در فضای ثانویه شکل می‌گیرد که می‌توانیم با انتخاب نمونه‌ای از این فضای مربوط به هر کلاس، یک دیتا را از همان کلاس ایجاد کنیم. همچنین با انتخاب یک نمونه از این فضا که در توزیع چند کلاس باشد، می‌توانیم دیتایی را ایجاد کنیم که ویژگی‌هایی را از تمامی آن کلاس‌ها شامل می‌شود. اما همانگونه که مشاهده می‌شود در فضای latent بیان دیتاهایی که برچسب یکسانی دارند نزدیک یکدیگرند. هرچند که در روند آموزش هیچ اطلاعاتی در مورد این دیتاهای به شبکه نداده‌ایم.

## ۵.۱ ایجاد دیتا بر اساس نمونه‌های فضای latent

در بخش قبل حدودی را از توزیع دیتای هر کلاس در فضای ثانویه مشاهده کردیم. حال در این بخش می‌خواهیم با ایجاد یک grid با ابعاد  $10 \times 10$  نمونه‌هایی را در فضای ثانویه ایجاد کنیم و سپس این نمونه‌ها را با استفاده از دیکودر آموزش دیده به دیتای تصویر تبدیل کنیم. برای این کار یک grid در محدوده‌ی  $-1 \leq z \leq 1$  را در دو بعد ایجاد می‌کنیم. نتیجه خروجی دیکودر برای این grid به صورت زیر است:



شکل ۸: تبدیل فضای ثانویه به تصویر

با توجه به شکل ۱۵ و محدوده‌ی تعریف شده در این بخش انتظار داشتیم که نمونه‌های ایجاد شده، بیشتر اعداد ۶ و ۷ باشند که مشاهده فوق بر این استدلال منطبق است.

## ۶.۱ بررسی کیفیت شبکه

برای بررسی عملکرد شبکه، دیتای تست را به انکودر و سپس دیکودر می‌دهیم و نمونه‌ی ایجاد شده توسط مدل را بررسی می‌کنیم. نمونه‌های واقعی از دیتای تست به صورت زیر است:

۰	۰	۴	۸	۹	۶	۸	۱	۴	۹
۲	۶	۹	۴	۳	۲	۷	۲	۵	۵
۶	۱	۶	۹	۰	۱	۸	۷	۳	۴
۳	۶	۴	۸	۱	۳	۴	۰	۶	۹
۸	۹	۲	۱	۲	۴	۱	۱	۵	۶
۶	۹	۴	۰	۴	۸	۹	۳	۶	۶
۹	۱	۵	۲	۴	۷	۹	۵	۳	۱
۸	۸	۸	۸	۴	۳	۰	۱	۲	۱
۸	۷	۴	۲	۱	۵	۷	۲	۰	۹
۷	۸	۱	۰	۱	۰	۸	۱	۳	۸

شکل ۹: تعدادی نمونه از دیتای تست

۰	۰	۹	۹	۹	۶	۹	۱	۹	۹
۲	۶	۹	۹	۳	۸	۹	۲	۵	۸
۵	۱	۶	۹	۵	۱	۳	۷	۳	۹
۳	۶	۹	۸	۱	۳	۴	۸	۶	۹
۸	۹	۲	۱	۲	۹	۱	۱	۵	۶
۳	۹	۹	۳	۹	۵	۹	۳	۶	۶
۹	۱	۹	۲	۴	۷	۹	۲	۳	۱
۳	۸	۵	۸	۶	۳	۰	۱	۲	۱
۵	۷	۴	۳	۱	۵	۷	۲	۳	۹
۷	۸	۱	۰	۱	۰	۸	۱	۳	۸

شکل ۱۰: پیش‌بینی مدل برای دیتای تست

کیفیت پیش‌بینی مدل با توجه به تعداد اپیاک کم و فضای ثانویه خیلی کوچک، تقریباً قابل قبول است. اما همانطور که مشاهده می‌شود کنترلی بر روی دیتای تولیدی شبکه نداریم و از این رو ممکن است پیش‌بینی مدل برای یک عدد، عدد دیگری یا حتی ترکیبی از چند عدد باشد.

## ۷.۱ پیاده‌سازی CVAE

حال می‌خواهیم برای حل مشکل دوم اتوانکودر و کنترل بیشتر برای ایجاد نمونه‌های جدید بر اساس دیتای ورودی، انکودر و دیکودر مطرح شده در بخش‌های قبلی را به برچسب واقعی دیتا مشروط کنیم و از این رو شبکه را به گونه‌ای آموزش دهیم تا بتوانیم با مشخص کردن برچسب دلخواه، نمونه‌ای را مربوط به همان عدد ایجاد کنیم. بنابراین باید به نحوی ساختار انکودر و دیکودر را مشروط به برچسب دیتا کنیم.

### ۱.۷.۱ ساختار انکودر

ساختار شبکه انکودر تقریبا مشابه با انکودر قبل است، با این تفاوت که در ابتدا بردار  $10 \times 10$  عضوی برچسب دیتا که ساختار one hot دارد را با استفاده از یک لایه fully connected و سپس reshape به یک ماتریس  $28 \times 28$  در  $28 \times 28$  تبدیل می‌کنیم و سپس این ماتریس را با ماتریس تصویر ورودی ترکیب می‌کنیم تا نهایتاً به یک دیتا  $28 \times 28$  با عمق ۲ برسیم. باقی شبکه مطابق با همان چیزی است که در پیاده‌سازی انکودر VAE داشتیم. تفاوت دیگر این است که ورودی بردار  $10 \times 10$  تابی مربوط به برچسب دیتا را به عنوان خروجی چهارم تعریف می‌کنیم تا بتوانیم از این برچسب در بخش دیکودر استفاده کنیم. ساختار انکودر مطابق با شکل زیر است:

Model: "encoder"			
Layer (type)	Output Shape	Param #	Connected to
input_3 (InputLayer)	[None, 10]	0	
dense_2 (Dense)	(None, 784)	6824	input_3[0][0]
input_2 (InputLayer)	[None, 28, 28, 1]	0	
reshape (Reshape)	(None, 28, 28, 1)	0	dense_2[0][0]
concatenate (Concatenate)	(None, 28, 28, 2)	0	input_2[0][0]; reshape[0][0]
conv2d_3 (Conv2D)	(None, 14, 14, 32)	668	concatenate[0][0]
dropout_2 (Dropout)	(None, 14, 14, 32)	0	conv2d_3[0][0]
conv2d_4 (Conv2D)	(None, 14, 14, 64)	18496	dropout_2[0][0]
dropout_3 (Dropout)	(None, 14, 14, 64)	0	conv2d_4[0][0]
conv2d_5 (Conv2D)	(None, 7, 7, 328)	73856	dropout_3[0][0]
flatten_1 (Flatten)	(None, 6272)	0	conv2d_5[0][0]
dense_3 (Dense)	(None, 64)	401472	flatten_1[0][0]
dense_4 (Dense)	(None, 16)	1040	dense_3[0][0]
z_mean (Dense)	(None, 2)	34	dense_4[0][0]
z_log_var (Dense)	(None, 2)	34	dense_4[0][0]
sampling_1 (Sampling)	(None, 2)	0	z_mean[0][0]; z_log_var[0][0]

Total params: 584,164  
Trainable params: 584,164  
Non-trainable params: 0

شکل ۱۱: ساختار شبکه انکودر

### ۲.۷.۱ ساختار دیکودر

همانطور که در بخش قبل به آن اشاره شد، خروجی چهارم دیکودر، برچسب حقیقی دیتا در یک ساختار one hot می‌باشد، این خروجی را به عنوان ورودی دوم دیکودر تعریف می‌کنیم و آن را با نمونه‌ی خروجی از لایه انکودر که در فضای دوبعدی است concatenate می‌کنیم. بنابراین نهایتاً به یک بردار  $12 \times 12$  عضوی می‌رسیم که پس از عبور آن از لایه‌های مشابه دیکودر در VAE به یک تصویر  $28 \times 28$  در  $28 \times 28$  تبدیل می‌کنیم.

Model: "decoder"			
Layer (type)	Output Shape	Param #	Connected to
input_4 (InputLayer)	[None, 2]	0	
input_5 (InputLayer)	[None, 10]	0	
concatenate_1 (Concatenate)	(None, 12)	0	input_4[0][0]; input_5[0][0]
dense_5 (Dense)	(None, 6272)	81536	concatenate_1[0][0]
reshape_1 (Reshape)	(None, 7, 7, 128)	0	dense_5[0][0]
conv2d_transpose_1 (Conv2DTranspose)	(None, 14, 14, 128)	147584	reshape_1[0][0]
conv2d_transpose_2 (Conv2DTranspose)	(None, 28, 28, 32)	73792	conv2d_transpose_1[0][0]
conv2d_transpose_3 (Conv2DTranspose)	(None, 28, 28, 1)	289	conv2d_transpose_2[0][0]

Total params: 321,665  
Trainable params: 321,665  
Non-trainable params: 0

شکل ۱۲: ساختار شبکه دیکودر

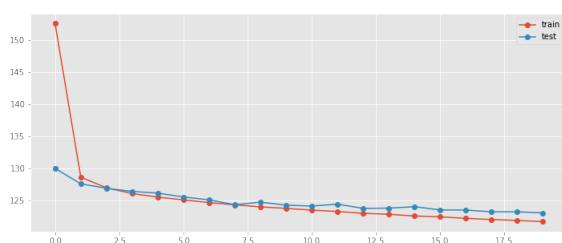
همانگونه که مشاهده می‌شود تنها تفاوت این ساختار با دیکودر استفاده شده در VAE در ورودی آن است که به جای استفاده از تنها نمونه‌ی برداشته شده از فضای ثانویه، از برچسب حقیقی دیتا نیز استفاده می‌شود.

### ۳.۷.۱ مدل کلی

مشابه آنچه برای مدل VAE داشتیم، اینجا نیز یک مدل را بر اساس ساختار کلاس مدل در کتابخانه keras تعریف می‌کنیم و متدهای مربوط به آموزش، تست و ایجاد خروجی مدل را پیاده‌سازی می‌کنیم.  
برای پیاده‌سازی متدهای مربوط به آموزش شبکه ابتدا، دیتا و برچسب حقیقی دیتا را به انکودر می‌دهیم و سپس خروجی آن را از طریق دیکودر به یک تصویر تبدیل می‌کنیم. در قدم بعدی، مقادیر خطای شبکه و گرادیان وزن‌های قبل آموزش را محاسبه کرده و وزن‌ها را بر اساس آن بروزرسانی می‌کنیم.  
متدهای مربوط به تست نیز مشابه همین بخش است، اما محاسبه گرادیان و بروزرسانی وزن‌ها در این بخش انجام نمی‌شود و تنها خطای مربوط به دیتای تست را محاسبه می‌کنیم.

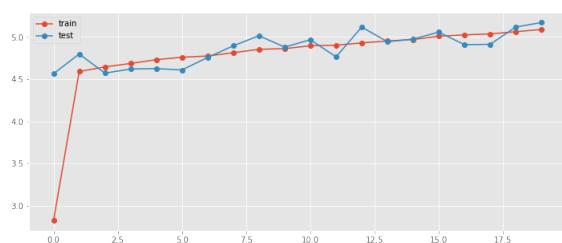
### ۴.۷.۱ نتیجه آموزش

خطای بازسازی برای دیتای تست و آموزشی به صورت زیر است:



شکل ۱۳: خطای بازسازی

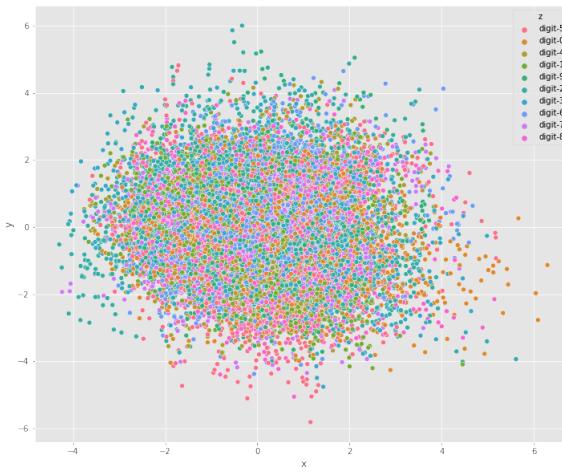
خطای kl divergence برای دیتای تست و آموزشی به صورت زیر است:



شکل ۱۴: خطای kl divergence

### ۸.۱ بررسی دیتا در فضای latent

برای مشاهده توزیع دیتاهای هر کلاس در این فضای دو بعدی، هر دیتا را بر اساس برچسب واقعی آن و با یک رنگ منحصر به فرد ترسیم کردہایم:



شکل ۱۵: دیتا در فضای latent

برخلاف آنچیزی که در شبکه‌ی vae داشتیم، توزیع دیتا در فضای latent علاوه بر ورودی، به کلاس دیتا نیز مشروط است:

$$Q(z|X, c)$$

از این رو توزیع هر یک از کلاس‌ها با توجه به خطای k1 تعریف شده برای مدل، به یک توزیع نرمال با میانگین صفر و کواریانس واحد نزدیک می‌شود. برخلاف آنچیزی که در بخش vae داشتیم و توزیع دیتا به صورت  $Q(z|X)$  بود و دیتا تقریباً به شکل مناسبی خوشبندی شده بود، اینجا به ازای هر برچسب دیتا ( $y = c$ ) توزیع  $Q(z|X, c)$  نزدیک به یک توزیع نرمال با میانگین صفر و کواریانس واحد است. زیرا اگر بخواهیم بر اساس آنچیزی که از آموزش شبکه انتظار داشتیم موضوع را بررسی کنیم، اکنون با توزیع  $Q(z|c)$  سروکار داریم.

## ۹.۱ ایجاد دیتا بر اساس نمونه‌های فضای latent

۰	۰	۰	۰	۰	۰	۰	۰	۰	۰
۱	۱	۱	۱	۱	۱	۱	۱	۱	۱
۲	۲	۲	۲	۲	۲	۲	۲	۲	۲
۳	۳	۳	۳	۳	۳	۳	۳	۳	۳
۴	۴	۴	۴	۴	۴	۴	۴	۴	۴
۵	۵	۵	۵	۵	۵	۵	۵	۵	۵
۶	۶	۶	۶	۶	۶	۶	۶	۶	۶
۷	۷	۷	۷	۷	۷	۷	۷	۷	۷
۸	۸	۸	۸	۸	۸	۸	۸	۸	۸
۹	۹	۹	۹	۹	۹	۹	۹	۹	۹

شکل ۱۶: تبدیل فضای ثانویه به تصویر

## ۱۰.۱ بررسی کیفیت شبکه

برای بررسی عملکرد شبکه، دیتای تست را به انکودر و سپس دیکودر می‌دهیم و نمونه‌ی ایجاد شده توسط مدل را بررسی می‌کنیم. نمونه‌های واقعی از دیتای تست به صورت زیر است:

۵	۱	۱	۸	۴	۳	۶	۰	۳	۸
۷	۱	۳	۵	۳	۷	۹	۱	۰	۱
۴	۸	۵	۱	۱	۷	۴	۹	۴	۴
۳	۰	۱	۴	۲	۶	۹	۹	۲	۷
۵	۵	۳	۶	۰	۷	۴	۱	۷	۰
۷	۳	۶	۷	۸	۵	۱	۱	۳	۹
۹	۶	۶	۶	۰	۵	۰	۱	۰	۲
۰	۴	۶	۵	۴	۳	۷	۰	۸	۳
۴	۲	۶	۵	۴	۲	۷	۳	۵	۳
۹	۹	۳	۲	۳	۱	۱	۶	۳	۹

شکل ۱۷: تعدادی نمونه از دیتای تست

5	1	1	8	4	3	6	0	3	8
7	1	3	5	3	7	9	1	0	1
4	8	5	1	1	7	7	9	4	4
3	0	7	4	2	6	9	9	2	7
5	5	3	6	0	7	4	4	7	0
7	3	6	7	8	5	4	1	3	4
9	6	6	6	0	5	0	1	0	2
0	4	6	5	4	3	7	0	8	3
4	2	6	5	4	2	7	3	5	3
9	9	3	2	3	2	1	6	3	9

شکل ۱۸: پیش‌بینی مدل برای دیتای تست

کیفیت پیش‌بینی مدل با توجه به فضای ثانویه خیلی کوچک، تقریباً قابل قبول است. برتری خروجی این شبکه نسبت به VAE آن است که پیش‌بینی مدل برای هر عدد، نمونه‌ای از همان عدد است که بعضی ویژگی‌های آن تغییر کرده است. در حالی که در مدل قبل مشاهده کردیم که خروجی مدل برای هر دیتا می‌تواند ترکیبی از ویژگی‌های کلاس‌های متفاوت باشد.

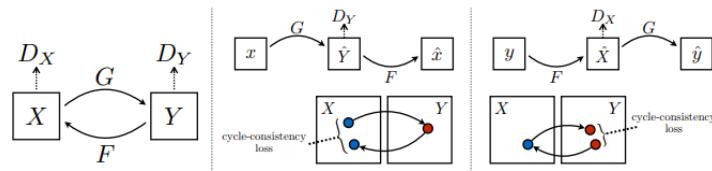
## ۲ سوال دوم

### ۱.۲ ساختار شبکه

با توجه به مقاله‌ی شبکه‌ی CycleGAN هدف کلی این شبکه آن است که تصاویر را از یک فضای به فضای دیگری تبدیل کند. اما این مساله می‌تواند با چالش‌های زیادی همراه باشد. برای مثال اگر بخواهیم با استفاده از این شبکه، تصاویر دنیای واقعی را به نقاشی‌های یک نقاش معروف و قدیمی تبدیل کنیم، پیدا کردن یک دیتاست که نقاشی‌ها و تصاویر دنیای واقعی متناسب با آن‌ها را در کنار یکدیگر داشته باشد، عملای غیرممکن است.

هدف ما این است تا تصاویر را از یک فضای دیگری تبدیل کنیم اما باید توجه داشته باشیم که الزاماً یک دیتاست که جفت تصاویر مربوط به این دو فضای در مقابل یکدیگر قرار داده باشد، وجود ندارد. این شبکه به دنبال آن است که ویژگی‌های خاص را در یک دسته از تصاویر دریافت کرده و آن‌ها را به دسته تصاویر دیگری تبدیل کند و تلاش می‌کند تا روشی را ارائه دهد تا این تبدیل را بدون داشتن یک دیتاست با جفت تصاویر ورودی و خروجی، یاد بگیرد.

دو فضای تصویری که در این شبکه می‌خواهیم استفاده کنیم یک رابطه وجود دارد، مثلاً این دو تصویر، می‌توانند بیان متفاوتی از یک صحنه‌ی یکسانی باشند. یک راهکار این است تا شبکه‌ای را آموزش دهیم که بتواند تصاویر فضای ۱ را به فضای ۲ تبدیل کند. در این ساختار ممکن است حتی شبکه بتواند تصاویری را ایجاد کند که تمامی ویژگی‌های فضای ۲ را دارا باشد و تمیز دادن آن از میان تصاویر فضای دوم غیرممکن شود. اما این ساختار تضمینی برای اینکه ارتباط معناداری بین تصویر ایجاد شده و تصویر ورودی باشد، ندارد.



شکل ۱۹: ساختاری توصیفی از شبکه CycleGAN

برای این منظور ساختاری که در این مقاله ارائه شده است متشکل از دو شبکه generator می‌باشد که یکی تصاویر را از فضای ۱ به ۲ و دیگری از فضای ۲ به ۱ تبدیل می‌کند و این دو شبکه را به صورت همزمان آموزش می‌دهد. همچنین دو شبکه discriminator نیز در این ساختار استفاده شده است. برای مثال شبکه‌ی  $D_y$  به عنوان discriminator تلاش می‌کند تا با تعریف یک adversarial loss شبکه‌ی  $G_x$  را به سمتی هدایت کند تا این شبکه بتواند تصاویری را در فضای  $y$  ایجاد کند به نحوی که تمیز دادن آن از دیتای واقعی در فضای  $y$  غیرممکن شود. در واقع در این منازعه، شبکه‌ی  $D_y$  در تشخیص تصاویر تقلیلی ایجاد شده در فضای  $y$  تخصص پیدا می‌کند و متقابلاً، شبکه‌ی  $G_x$  نیز ویژگی‌های بیشتری را از فضای ثانویه می‌آموزد و تلاش می‌کند تا با یادگیری تبدیل بهتری از  $x$  به  $y$  شبکه‌ی  $D_y$  را اشتیاه بیاندازد.

این ساختار می‌تواند امکانی را برای ما فراهم کند تا بتوانیم مفهومی به نام "پایداری چرخه" را در تبدیل بین دو فضای رعایت کنیم. این مفهوم به این صورت است که مثلاً وقتی یک جمله را از انگلیسی به فرانسه ترجمه می‌کنیم و سپس مجددآ آن را به انگلیسی بر می‌گردانیم، جمله‌ی نهایی با جمله‌ی اولیه یکسان است. برای آنکه بتوان رابطه‌ی معناداری را بین دو فضای یافت، یک خطای مربوط به پایداری چرخه به این شبکه اضافه شده است.

خطای مربوط به پایداری چرخه به صورت زیر تعریف می‌شود:

$$\ell_{eye}(G, F) = \mathbb{E}_{x \sim p_{data}(x)}[\|F(G(x)) - x\|_1] + \mathbb{E}_{y \sim p_{data}(y)}[\|G(F(y)) - y\|_1]$$

بر اساس این خطا، شبکه تلاش می‌کند تا تبدیل‌های  $F$  و  $G$  را به گونه‌ای بیاموزد که تصویری که از فضای ۱ به ۲ و سپس مجدداً به ۱ تبدیل می‌شود، همان تصویری باشد که در ابتدا به شبکه داده شده است. در واقع کل شبکه را می‌توان به صورت دو اتوانکودر دید که وابسته به یکدیگر آموزش می‌بینند. فضای latent یکی از انکودرها، تصاویر فضای اول و فضای latent دیگری، تصاویر فضای دوم است.

اتوانکودر اول، تصاویر را از فضای ۱ دریافت می‌کند و با استفاده از خطای adversarial تلاش می‌کند تا تصویری را در فضای ۲ ایجاد کند و به صورت همزمان معکوس این تبدیل را گونه‌ای بیابد تا تصویر ایجاد شده در فضای دوم با توجه به خطای پایداری چرخه، مجدداً به همان تصویر در فضای ۱ تبدیل شود.

خطای کلی شبکه به صورت زیر است:

$$\ell(G, F, D_x, D_y) = \ell_{GAN}(G, D_y, X, Y) + \ell_{GAN}(F, D_x, Y, X) + \lambda \ell_{eye}(G, F)$$

خطای دیگری در کاربرد تبدیل نقاشی به عکس به این شبکه اضافه می‌شود که باعث می‌شود تا تبدیل بین فضاهای ترکیب رنگی بین تصاویر ورودی و خروجی را حفظ کند. این تابع خطا، Identity نام دارد و به صورت زیر تعریف می‌شود:

$$\ell_{identity}(G, F) = \mathbb{E}_{x \sim p_{data}(x)} [\|F(x) - x\|_1] + \mathbb{E}_{y \sim p_{data}(y)} [\|G(y) - y\|_1]$$

بدون این خطا، شبکه‌های generator آزاد هستند تا ترکیب رنگی تصویر ورودی را تغییر دهند.

## PatchGAN ۲.۲

در این بخش، به بررسی PatchGAN در می‌پردازیم. همانطور که نویسنده‌گان مقاله اشاره کرده‌اند، Discriminator طراحی شده از PatchGAN های  $70 * 70 * 70$  بهره می‌برد. که هدف آن این است که بتواند با patch overlapping  $70 * 70 * 70$  تشخیص دهد تصویر جعلی یا واقعی است. که به این ترتیب هم پارامترهای کمتری را در خود جای می‌دهد. ایده PatchGAN این است که تصویر ورودی خام را به برخی از Patch های کوچک محلی تقسیم کرده، یک Discriminator کلی را به صورت کانولوشن در هر Patch اجرا کنیم و به طور متوسط تمام پاسخ‌ها را بدست بیاوریم تا خروجی نهایی نشان داده شود که آیا تصویر ورودی جعلی است یا خیر. حال تفاوت اصلی بین PatchGAN و یک GAN Discriminator معمولی در این است که PatchGAN تصویر ورودی را به یک خروجی اسکالر در محدوده  $[0, 1]$  نشان می‌دهد که همان احتمال واقعی بودن یا جعلی بودن تصویر است این در حالی است که PatchGAN یک آرایه را ارائه می‌دهد به عنوان خروجی با هر ورودی که واقعی یا جعلی بودن Patch مربوطه را نشان می‌دهد. دلیل اینکه مقاله CycleGAN از PatchGAN به عنوان تمایز دهنده خود استفاده می‌کند این است که پارامترهای کمتری نسبت به یک تمایز دهنده تصویر کامل دارد و بنابراین بسیار سریع اجرا می‌شود و می‌تواند روی تصاویر بزرگ بہتر کار کند.

## ۳.۲ پیاده‌سازی با استفاده از unet

حال در این بخش می‌خواهیم به جزئیات پیاده‌سازی شبکه پردازیم و نهایتاً با آموزش شبکه بر روی دیتابست monet عملکرد آن را مورد بررسی قرار دهیم.

## generator ۱.۳.۲

برای پیاده‌سازی این بخش از شبکه از معماری unet پیروی می‌کنیم. طبق این معماری، شبکه از دو بخش انقباضی و انبساطی تشکیل می‌شود که یک مسیر فرعی نیز برای انتقال دیتا از مسیر انقباضی به لایه‌های متضاظر در لایه انبساطی ایجاد می‌شود.

در مسیر انقباضی از لایه‌های کانولوشنی مختلف با اندازه‌ی کرنل ۳ در ۳ و استراید ۲ استفاده می‌کنیم که در هر مرحله بعد فضایی تصویر نصف می‌شود و متناظر، تعداد فیلترهای کانولوشنی دو برابر می‌شود. تعداد فیلتر از ۶۴ فیلتر در لایه اول شروع شده و نهایتاً به ۵۱۲ فیلتر می‌رسد. در این بخش اصلاحاتی در ساختار اصلی unet ایجاد شده است تا این شبکه برای کاربرد موردنظر مناسب باشد.

مسیر انساطی نیز در هر لایه از یک لایه کانولوشنی تشکیل شده است که با استفاده از کانولوشن معکوس، ابعاد فضایی تصویر دو برابر شده و تعداد کانال‌های ویژگی نصف می‌شود. هر لایه در این مسیر دیتایی را از لایه‌های پایینی و دیتایی را از لایه متناظر در مسیر انقباضی دریافت می‌کند. در لایه نهایی این شبکه نیز تغییری اعمال شده است به این صورت که برای ایجاد یک تصویر RGB در لایه خروجی، یک لایه کانولوشنی معکوس با ۳ فیلتر را تعریف کرده‌ایم. در تمامی این لایه‌ها از تابع فعال‌سازی ReLU استفاده شده است.

### discriminator ۲.۳.۲

ساختار این بخش از شبکه بر اساس PatchGAN و جزئیاتی که در مقاله ارائه شده است به صورت زیر است:

۱. یک لایه کانولوشنی با ۶۴ فیلتر و اندازه کرنل ۴ در ۴ و استراید ۲ که در آن از نسخه اصلاح شده‌ی تابع فعال‌سازی ReLU استفاده شده است.
۲. یک لایه کانولوشنی با ۱۲۸ فیلتر و اندازه کرنل ۴ در ۴ و استراید ۲.
۳. یک لایه کانولوشنی با ۲۵۶ فیلتر و اندازه کرنل ۴ در ۴ و استراید ۲.
۴. یک لایه کانولوشنی با ۵۱۲ فیلتر و اندازه کرنل ۴ در ۴ و استراید ۲.
۵. یک لایه کانولوشنی با ۱ فیلتر و اندازه کرنل ۴ در ۴.

برای آموزش این مدل از تابع خطای mse و بهینه‌ساز adam استفاده شده است. ساختار شبکه به صورت زیر است:

Model: "Dx"		
Layer (type)	Output Shape	Param #
input_27 (InputLayer)	[ (None, 256, 256, 3) ]	0
conv2d_189 (Conv2D)	(None, 128, 128, 64)	3136
leaky_re_lu_40 (LeakyReLU)	(None, 128, 128, 64)	0
conv2d_190 (Conv2D)	(None, 64, 64, 128)	131200
instance_normalization_232 (	(None, 64, 64, 128)	256
leaky_re_lu_41 (LeakyReLU)	(None, 64, 64, 128)	0
conv2d_191 (Conv2D)	(None, 32, 32, 256)	524544
instance_normalization_233 (	(None, 32, 32, 256)	512
leaky_re_lu_42 (LeakyReLU)	(None, 32, 32, 256)	0
conv2d_192 (Conv2D)	(None, 16, 16, 512)	2097664
instance_normalization_234 (	(None, 16, 16, 512)	1024
leaky_re_lu_43 (LeakyReLU)	(None, 16, 16, 512)	0
conv2d_193 (Conv2D)	(None, 16, 16, 1)	8193
Total params: 2,766,529 Trainable params: 2,766,529 Non-trainable params: 0		

شکل ۲۰: ساختار Discriminator

### ۳.۳.۲ ساختار کلی شبکه و آموزش آن

حال با توجه به تعریفی که برای دو بخش generator و discriminator شبکه داشتیم ساختار کلی شبکه را تعریف می‌کنیم. در این ساختار دو شبکه‌ی generator و discriminator معرفی شود. شبکه‌ی  $G$  دیتا را از فضای  $x$  گرفته و آن را به فضای  $y$  می‌برد و شبکه‌ی  $F$  هم تبدیل معکوسی را می‌آموزد. همچنین دو شبکه‌ی discriminator استفاده می‌کنیم که یکی از آن‌ها در رقابت با شبکه‌ی  $G$  اصالت دیتا را بررسی می‌کند و شبکه‌ی دوم هم به همین صورت اصالت دیتا در فضای  $x$  را بررسی می‌کند.

روند آموزش شبکه به این صورت است که به ازای یک جفت تصویر  $x$  و  $y$ ، ابتدا با استفاده از شبکه‌ی  $G$  دیتا  $\hat{y}$  را ایجاد می‌کنیم و در مرحله دوم خطای شبکه‌ی  $y$  را با استفاده از تصویر  $\hat{y}$  و  $y$  محاسبه کرده و وزن‌های این شبکه را بروز می‌کنیم. همین روند برای شبکه‌های  $F$  و  $D_x$  نیز انجام می‌شود. در ادامه adversarial loss را محاسبه می‌کنیم:

```
gen_g_validity = discriminator_y(fake_y, training=True)
gen_f_validity = discriminator_x(fake_x, training=True)
gen_g_adv_loss = gen_loss(gen_g_validity)
gen_f_adv_loss = gen_loss(gen_f_validity)
```

خطای بعدی که باید محاسبه شود خطای پایداری چرخه می‌باشد:

```
cyc_x = generator_f(fake_y, training=True)
cyc_x_loss = image_similarity(real_x, cyc_x)
cyc_y = generator_g(fake_x, training=True)
cyc_y_loss = image_similarity(real_y, cyc_y)
```

تابع مربوط به محاسبه خطای پایداری چرخه به صورت زیر پیاده‌سازی شده است:

```
tf.reduce_mean(tf.abs(image1 - image2))
```

برای محاسبه این خطای پایداری چرخه از  $x$  تصویر  $\hat{y}$  را ایجاد می‌کنیم و مجدداً این تصویر را به فضای اصلی بر می‌گردانیم تا تصویر  $\hat{x}$  ایجاد شود. این تابع میزان شباهت تصویر  $x$  و  $\hat{x}$  را می‌سنجد و خطای پایداری چرخه را محاسبه می‌کند.

خطای سومی که برای شبکه محاسبه می‌کنیم، خطای identity است که برای حفظ ترکیب رنگی در تصاویر تولیدی است:

```
id_x = generator_f(real_x, training=True)
id_x_loss = image_similarity(real_x, id_x)

id_y = generator_g(real_y, training=True)
id_y_loss = image_similarity(real_y, id_y)
```

این تابع میزان شباهت خروجی شبکه‌ی generator را با ورودی آن محاسبه می‌کند که البته این خطای پایداری در خطای کلی هر یک از شبکه‌های generator اثر می‌دهیم تا شبکه در عین حفظ ترکیب رنگی، تلاش زیادی برای شبیه‌کردن تصاویر به یکدیگر نداشته باشد تا توانایی شبکه برای ایجاد تصاویر جدید همچنان حفظ شود.

#### ۴.۳.۲ آموزش شبکه

حال در این بخش شبکه را به ازای ایپاک آموزش می‌دهیم و عملکرد آن را در هر ایپاک مورد بررسی قرار می‌دهیم:<sup>۱</sup>



شکل ۲۱: خروجی شبکه بعد از ایپاک ۱



شکل ۲۲: خروجی شبکه بعد از ایپاک ۲

<sup>۱</sup> در تصاویری که در ادامه برای بررسی عملکرد شبکه نشان داده می‌شود، تصاویر واقعی در سمت چپ و تصاویر ایجاد شده توسط شبکه در سمت راست نمایش داده می‌شود.



شکل ۲۳: خروجی شبکه بعد از اپاک ۳



شکل ۲۴: خروجی شبکه بعد از اپاک ۵



شکل ۲۵: خروجی شبکه بعد از ایپاک ۹

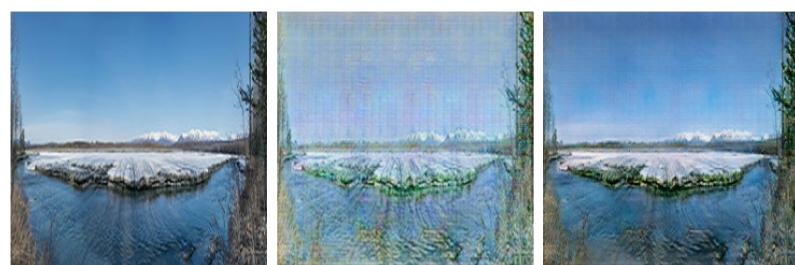


شکل ۲۶: خروجی شبکه بعد از ایپاک ۱۶

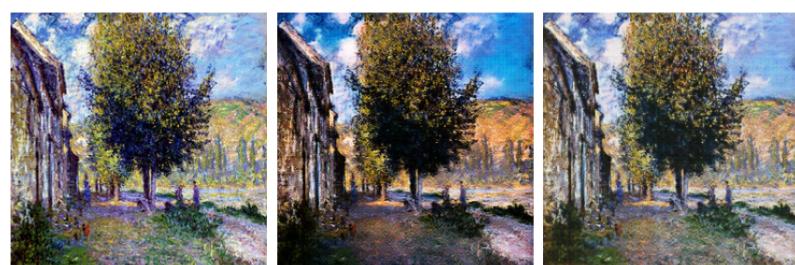


شکل ۲۷: خروجی شبکه بعد از ایپاک ۲۰

اکنون می‌خواهیم عملکرد شبکه را در پایداری چرخه بررسی کنیم. از این رو تصویری را به شبکه می‌دهیم و تبدیل آن را به فضای ثانویه حساب می‌کنیم و مجدداً به فضای اولیه برمی‌گردانیم.



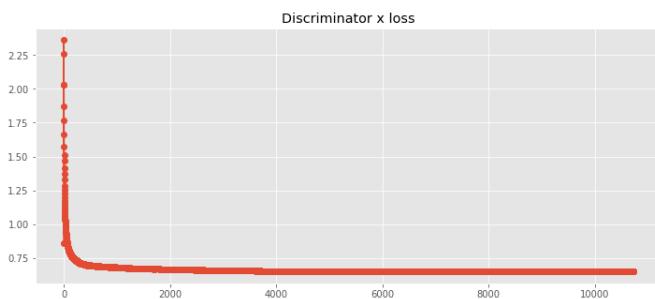
شکل ۲۸: به ازای یک تصویر واقعی



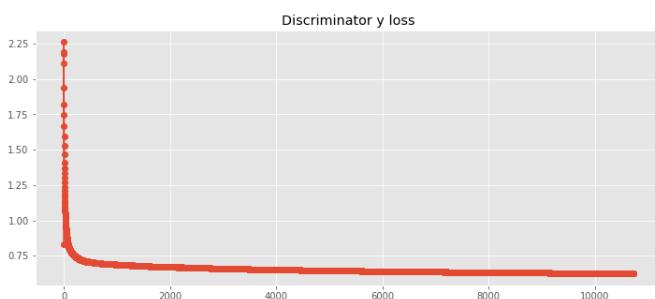
شکل ۲۹: به ازای یک نقاشی

### ۵.۳.۲ نمودار خطای شبکه

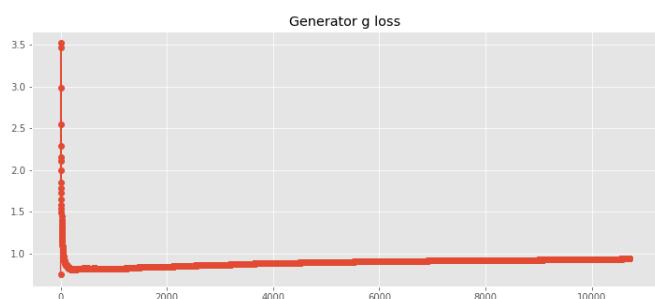
مقادیر خطاهای مختلف شبکه به صورت زیر است<sup>۳</sup> :



شکل ۳۰: خطای discriminator x loss

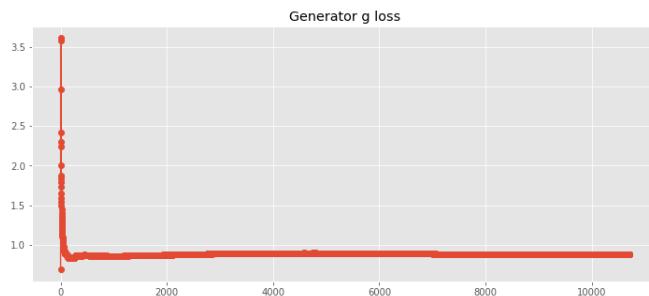


شکل ۳۱: خطای discriminator y loss

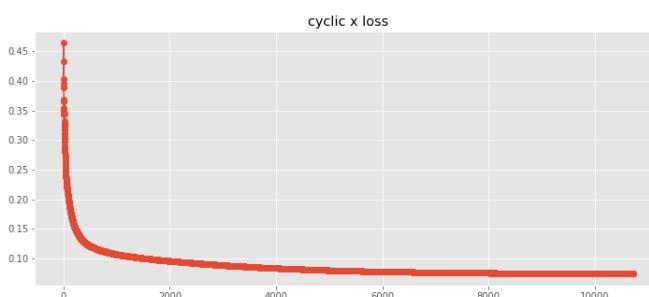


شکل ۳۲: خطای generator g loss

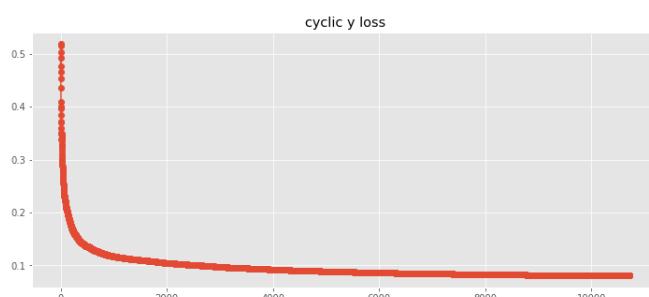
<sup>۳</sup>محور افقی نمودارهای خطای بر حسب تعداد iteration می‌باشد.



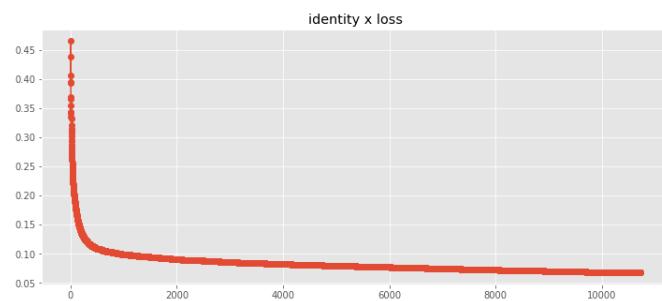
شکل ۳۳: خطای generator f loss



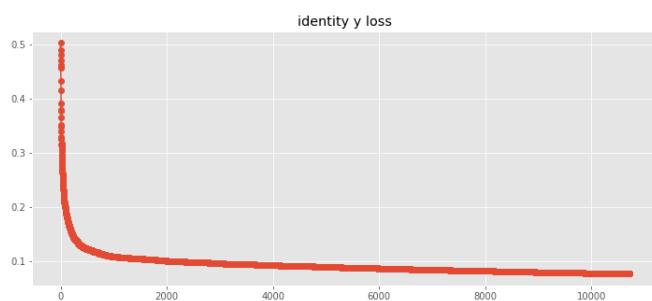
شکل ۳۴: خطای cyclic x loss



شکل ۳۵: خطای cyclic y loss



شکل ۳۶: خطای identity x loss



شکل ۳۷: خطای identity y loss

### ۳ سوال سوم

در این سوال با شبکه StackGAN کار خواهیم کرد که یکی از افزونه‌های شبکه Network است. این شبکه قادر است با استفاده از متن، تصویر سنتز کند. این شبکه از معروف‌ترین و محبوب‌ترین شبکه‌های GAN شناخته می‌شود.



شکل ۳۸: نمونه تصویر ایجاد شده توسط این شبکه

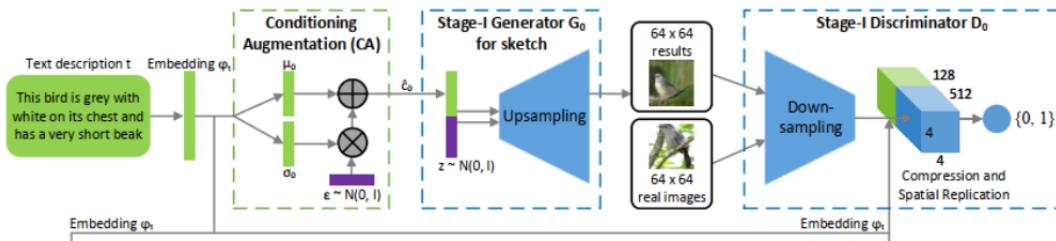
### ۱.۳ خلاصه کلی مقاله

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

#### ۱.۱.۳ تحلیل ساز و کار شبکه

برای ساخت تصاویر با رزولوشن بالا و همچنین باورپذیر، نیازمند یک شبکه قدرتمند در تولید و بازبینی تصویر میباشیم که بتواند متن را به تصاویر باورپذیر تبدیل نماید این کار را با شبکه **StackGAN** و در ۲ گام انجام میدهیم.

۱. گام اول: در این مرحله، شکل ابتدایی و رنگ‌های اصلی جسم توصیف شده ما در متن ترسیم می‌شود. همچنین طرح پس زمینه را از یک بردار نویز تصادفی ترسیم می‌کند و در نهایت نیز یک تصویر با وضوح پایین ارائه می‌دهد. به عبارتی ما به جای تولید مستقیم یک تصویر با وضوح بالا که به توصیف متن مشروط است، کار را برای شبکه در ابتدا ساده می‌کنیم تا ابتدا با استفاده از Stage-I GAN خود، تصویری با وضوح پایین ایجاد کند، این تمرکز بر روی ترسیم شکل نچندان واضح و رنگ درست برای جسم است.

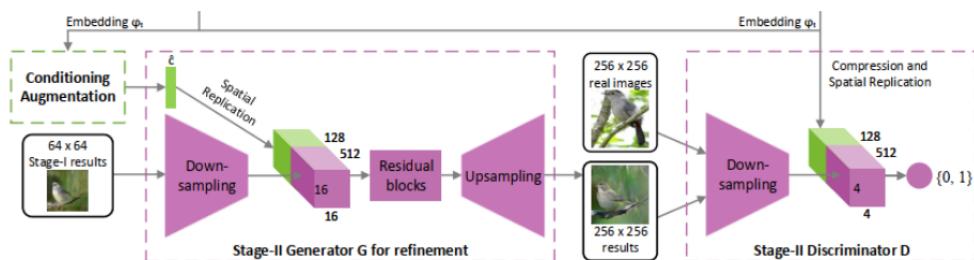


شکل ۳۹: ساختار بخش اول شبکه

همانگونه که مشاهده می‌شود، Generator بخش اول، یک تصویر خشن با رنگ‌های اساسی مهمن جسم از با استفاده از متن داده شده، تولید می‌کند همچنین قابل ذکر است که رنگ آمیزی پس زمینه، با استفاده از یک بردار نویز تصادفی ترسیم می‌شود. در ابتدا متن بردار شده  $\phi$ ، به یک شبکه Fully connected داده می‌شود که به وسیله آن، "میانگین و انحراف معیار" مربوط به نویز گوسی، فراهم شود.

برای قسمت Discriminator هم، در ابتدا متن بردار شده  $\phi$  را به وسیله یک لایه Fully-connected کاهش میدهیم و به فرم تنسور  $Md * Md * Md$  در می‌آوریم، سپس تصویر از لایه‌های Down-sampling عبور می‌کند تا به بعد  $Md * Md$  برسد. سپس، Filter Map تصویر در امتداد بعد کانال تنسور متن، بهم پیوسته می‌شود. تنسور حاصل به یک لایه کانولوشن  $1 * 1$  منتقل می‌شود تا به طور مشترک ویژگی‌های تصویر و متن را بیاموزد. سرانجام، از یک لایه Fully connected مرحله دوم نقص‌ها را تصحیح می‌کند تصمیم استفاده می‌شود. حال مشروط به نتایج همین گام، Generator جذاب‌تری را به نتایج Stage-I اضافه می‌کند، و یک تصویر با وضوح بالا واقعی‌تر ارائه می‌دهد.

۲. گام دوم: در این مرحله، نقص‌های موجود در تصویر با وضوح پایین از مرحله  $I$  تصحیح می‌شوند و با خواندن دوباره توضیحات متن، تولید یک عکس واقع گرایانه با وضوح بالا، جزئیات شی را کامل می‌کند. مرحله دوم مرحله حساس طراحی می‌باشد، که ما با استفاده از نتایج نویزی مرحله قبل، به یک تصویر با رزو لوشن بالا بررسیم.

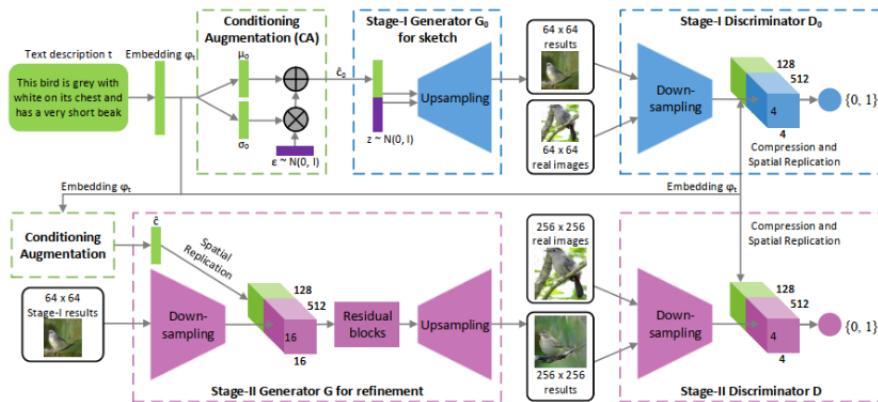


شکل ۴۰: ساختار بخش دوم شبکه

شبکه‌ی Generator مرحله دوم را به عنوان یک شبکه رمزگذار-رمزگشای (Encoder-Decoder) با بلک‌های باقیمانده متناسب طراحی می‌کنیم. مشابه مرحله قبلی، متن مورد نظر را استفاده می‌کنیم تا به

تولید برداری از متن در بعد مورد نظرمان برسیم که در نهایت یک تنسور  $Mg * Mg * Ng$  را تشکیل می‌دهد. در همین حال، نتیجه تولیدی از مرحله اول StackedGAN در چندین بلوک Down-sampling (به عنوان مثال رمزگذار) تقدیم می‌شود تا اندازه فضایی  $Mg * Mg$  بشود. ویژگی‌های تصویر و متن در امتداد بعد کانال بهم پیوسته می‌شوند. ویژگی‌های تصویر رمزگذاری شده همراه با ویژگی‌های متن در چندین بلوک باقیمانده قرار می‌گیرند، که برای یادگیری نمایش‌های چند حالت، در میان ویژگی‌های تصویر و متن طراحی شده‌اند. سرانجام، یک سری از لایه‌های Up-sampling (به عنوان مثال رمزگشایی) استفاده شده است که بتواند تصویری با ابعاد  $W * H$  با رزو لوشن بالا تولید نماید. چنین ژنراتوری می‌تواند ضمن اصلاح جزئیات برای ایجاد تصویر واقع گرایانه با کیفیت بالا، به رفع نقص در تصویر ورودی هم کمک کند.

شبکه‌ی Discriminator مرحله دوم هم مشابه با مرحله اول صورت می‌گیرد که در صفحه قبل توضیح داده شد، با این تفاوت که یک مرحله Down-sampling بیشتر اینجا اتفاق می‌افتد به دلیل آنکه تصویر اینجا ابعاد بزرگتری دارد. در نهایت ساختار نهایی شبکه همانند زیر بدست می‌آید.



شکل ۴۱: ساختار کلی شبکه

### ۲.۳ توابع هزینه

به طور کلی، در شبکه‌های GAN، به روز رسانی به شیوه خصم‌مانه صورت می‌گیرد، و تابع هزینه به صورتی طراحی می‌شود که بتواند این هدف که قوی‌تر شدن Generator برای به وجود آورن تصاویر واقعی تر است و قوی‌تر شدن برای تشخیص اصلی یا جعلی بودن تصاویر تولید شده به وسیله Discriminator در مسیر آموزش طی می‌شود را براورد سازد. تابع هدف کلی شبکه‌های GAN به صورت زیر طراحی می‌شود:

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

#### ۱.۲.۳ توابع هزینه مرحله اول StackGAN برای Generator و Discriminator

تابع هزینه این مرحله برای Generator و Discriminator ما به شکل زیر بدست می‌آید:

$$\ell_{D_0} = \mathbb{E}_{(I_0, t) \sim p_{data}} [\log D_0(I_0, \phi_t)] + \mathbb{E}_{z \sim p_z, t \sim p_{data}} [\log 1 - D_0(G_0(z, \hat{c}_0), \phi_t)]$$

$$\ell_{G_0} = \lambda D_{KL}(N(\mu_0(\phi_t), \Sigma_0(\phi_t)) \| N(0, I)) + \mathbb{E}_{z \sim p_z, t \sim p_{data}} [\log 1 - D_0(G_0(z, \hat{c}_0), \phi_t)]$$

در این فرمول فرض میکنیم ما یک Embedded-text حاصل از توضیحات متن اولیه باشد، که توسط یک رمز نگار از پیش آموزش داده شده بوجود آمده است. حال  $C$  که در بخش قبل درباره آن صحبت شد، توسطتابع نرمال ای که میانگین و انحراف معیار خود را از لایه اول Fully-connected شبکه می‌گیرد، به عنوان نمونه استفاده می‌شود. حال بسته به این متغیر  $C$ ، و  $\phi$  شبکه‌های Generator و Discriminator سعی میکنند در راستای قوی تر شدن به ترتیب در تولید محتواهی واقع گرایانه تر و تشخیص اصلی یا جعلی بودن تصویر، با استفاده از این توابع هزینه قدم بردارند.

### ۲.۲.۳ توابع هزینه مرحله دوم StackGAN برای Generator و Discriminator

توابع هزینه این مرحله برای Generator و Discriminator ما به شکل زیر بدست می‌آید:

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

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

اینجا فرض مینماییم که  $S_0$  همان تصویر low-resolution تشکیل شده از شبکه Generator این مرحله میباشد پس:

$$S_0 = G_0(z, \hat{c}_0)$$

حال اگر  $C$  را متغیر نهفته گویی بدانیم و  $D$  و  $G$  را به ترتیب Generator و Discriminator این مرحله (مرحله دوم) نامگذاری کنیم، به روابط بالا میرسیم. نکته حائز اهمیت این مرحله آن است که در فرمول تابع هزینه، اثری از نویز تصادفی  $Z$  نمیباشد چرا که ذات تصادفی بودن، در  $S_0$  موجود نمیباشد. در نهایت این نکته تأکید می‌شود که مرحله اول و دوم Fully-connected Conditioning Augmentation از لایه‌های StackGAN برای تولید متفاوت می‌باشند، برای میانگین و انحراف معیار متفاوت بهره میبرند، که بدین صورت مرحله دوم یاد می‌گیرد اطلاعات متفاوت تری که در Embedded text بوده است را بگیرد که در مرحله اول حذف شده بودند و ما به یک تصویر با وضوح بالا برسیم.

### ۳.۳ دیتاست استفاده شده

برای این شبکه، سه مجموعه داده COCO و Oxford-102 و CUB استفاده شده است که عملکرد شبکه ما یعنی StackGAN را با دو شبکه دیگر یعنی GAWWN و GAN-INT-CLS مقایسه خواهیم کرد. ابتدا به صورت شهودی این عملکرد را بررسی می‌کنیم:



شکل ۴۲: دیتاست

همانطور که مشاهده می‌شود، عملکرد شبکه StackGAN بسیار مطلوب می‌باشد. در قسمت پنجم این گزارش به سراغ نتایج عددی و دستاورد نویسنده‌گان می‌رویم.

#### ۴.۳ پیشینه مقاله

سترنز تصاویر با کیفیت بالا با استفاده از توصیف متن، یک مشکل چالش برانگیز در بینایی رایانه است و کاربردهای عملی بسیاری دارد. نمونه‌های تولید شده توسط رویکردهای تصویری موجود در متن می‌توانند تقریباً معنای توصیفات داده شده را منعکس کنند، اما حاوی جزئیات لازم و قسمت‌های زنده آن نیستند. در این مقاله، نویسنده‌گان پیشنهاد کردند شبکه‌های خصم‌مانه تولید انباسته (StackGAN) برای ایجاد تصاویر عکس واقعی  $256 * 256$  متریک متن برای اینکار استفاده شود. از این طریق یک فرآیند اصلاح، مشکلات رویکرد های قبلی برای تولید تصویر را به زیرمجموعه‌های قابل کنترل تری تجزیه می‌کند و با استفاده از آن به تصاویر باور پذیرتر و حاوی جزئیات لازم و زنده‌تر می‌رسد.

#### ۵.۳ دستاورد نویسنده‌گان مقاله

در این مقاله، نویسنده‌گان توانستند شبکه عصبی StackGAN را بنا نهند، شبکه‌ای که می‌تواند برای تولید تصویر از آن استفاده شود، در زیر نتایج عددی و دستاورد نویسنده می‌باشد که با استفاده از سه مجموعه داده COCO و Oxford-102 و CUB عملکرد شبکه StackGAN را با دو شبکه دیگر یعنی GAWWN مقایسه کرده‌اند:

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	$1.89 \pm .04$	/	$1.11 \pm .03$

شکل ۴۳: نتایج

#### ۶.۳ مسیر پیشنهادی برای ادامه تحقیقات

با توجه به شکل ۴۱ همانگونه که مشاهده می‌شود، برای تولید تصویر واقع گرایانه و با رزولوشن بالا توسط شبکه StackGAN، مسیر و فرآیند پیچیده‌ای طی می‌شود، حال تغییر یا حذف بخشی از این مراحل باعث تغییر در نتیجه ما می‌شود، همچنین در بعضی موارد به بهبود عملکرد شبکه ما منجر می‌شود. در مقاله اشاره شده است، به طور مثال وجود یا عدم وجود Text Conditioning Augmentation یا Conditioning Augmentation twice باعث شده نتیجه امتیاز تغییر کند:

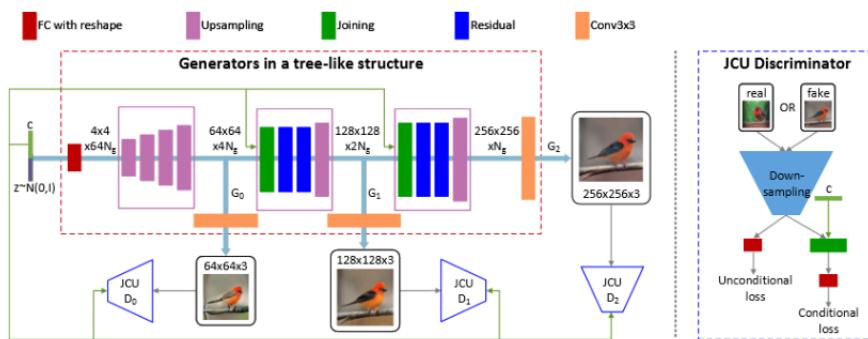
Method	CA	Text twice	Inception score
64×64 Stage-I GAN	no	/	2.66 ± .03
	yes	/	2.95 ± .02
256×256 Stage-I GAN	no	/	2.48 ± .00
	yes	/	3.02 ± .01
128×128 StackGAN	yes	no	3.13 ± .03
	no	yes	3.20 ± .03
	yes	yes	3.35 ± .02
256×256 StackGAN	yes	no	3.45 ± .02
	no	yes	3.31 ± .03
	yes	yes	3.70 ± .04

شکل ۴۴: نتیجه امتیاز شبکه StackGAN با ۳۰۰۰۰ داده تولید شده در حالت های اشاره شده شبکه

پس مشاهده میشود که میتوان حتی با تغییر ویژگی های این شبکه نیز به دقت های متفاوت رسید و میتوان در این زمینه تحقیقات لازم را به عمل آورد. در نهایت مقاله ای با عنوان

### StackGAN++: Realistic Image Synthesis with Stacked Generative Adversarial Networks

منتشر شد که در آن از Generator ها و Discriminator استفاده میشود و ساختار کلی این شبکه را میتوان در شکل زیر مشاهده نمود:



شکل ۴۵: ساختار نهایی StackGAN V2

در شکل زیر هم میتوان تفاوت عملکرد دو نسخه StackGAN V2 و StackGAN V1 را مشاهده نمود:

Dataset		CUB	Oxford-102	COCO	LSUN-bedroom	LSUN-church	ImageNet-dog	ImageNet-cat
FID ↓	StackGAN-v1	51.89	55.28	<b>74.05</b>	91.94	57.20	89.21	58.73
	StackGAN-v2	<b>15.30</b>	<b>48.68</b>	81.59	<b>35.61</b>	<b>25.36</b>	<b>44.54</b>	<b>28.59</b>
IS ↑	StackGAN-v1	$3.70 \pm .04$	$3.20 \pm .01$	$8.45 \pm .03$	$3.59 \pm .05$	$2.87 \pm .05$	$8.84 \pm .08$	$4.77 \pm .06$
	StackGAN-v2	<b><math>4.04 \pm .05</math></b>	<b><math>3.26 \pm .01</math></b>	<b><math>8.30 \pm .10</math></b>	<b><math>3.02 \pm .04</math></b>	<b><math>2.38 \pm .03</math></b>	<b><math>9.55 \pm .11</math></b>	<b><math>4.23 \pm .05</math></b>
HR ↓	StackGAN-v1	$1.81 \pm .02$	$1.70 \pm .03$	<b><math>1.45 \pm .04</math></b>	$1.95 \pm .01$	$1.86 \pm .02$	$1.90 \pm .01$	$1.88 \pm .02$
	StackGAN-v2	<b><math>1.19 \pm .02</math></b>	<b><math>1.30 \pm .03</math></b>	$1.55 \pm .05$	<b><math>1.05 \pm .01</math></b>	<b><math>1.14 \pm .02</math></b>	<b><math>1.10 \pm .01</math></b>	<b><math>1.12 \pm .02</math></b>

شکل ۴۶: عملکرد دو ورژن شبکه StackGAN

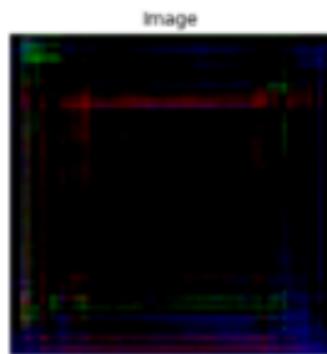
همانگونه که مشاهده میکنیم، نتایج بسیار بهبود یافته اند.

### ۷.۳ تست شبکه

برای اینکار شبکه را میسازیم و مرحله اول آن را در اپیاک های مشخصه و مرحله دوم آن را در ۲ اپیاک انجام میدهیم و به نتایج مطلوب میرسیم. برای این تمرین از داده ها Birds و CUB 200 2011 استفاده نمودیم. در ابتدا چهار پوشه میسازیم با نام ها results1 و results2 و results3 و results4 تا در آن ها به ترتیب نتایج آموزش مرحله اول StackGAN بعد از ۱۰ اپیاک و نتایج آموزش مرحله اول شبکه بعد از اپیاک مشخصه اپیاک و نتایج آموزش مرحله دوم شبکه و همچنین در نهایت برای بدست آوردن Log خروجی ازتابع خطا برای Generator و Discriminator.

#### ۱.۷.۳ آموزش مرحله اول شبکه تا ۱۰ اپیاک

نتایج اصلا مطلوب نمیباشند و میتوان گفت طقریبا شاهد نویز بی معنی شده ایم نه یک تصویر بیکیفیت اما در راستای صحیح. نمونه ای از خروجی مرحله اول StackGAN با ۱۰ اپیاک:



شکل ۴۷: نتیجه مرحله اول با ۱۰ اپیاک

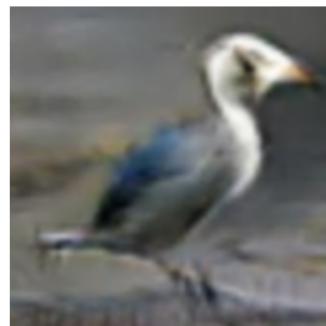
#### ۲.۷.۳ ادامه آموزش شبکه

نتیجه تابع خطا برای آخرین اپیاک و نتیجه تابع خطا کلی:

```
d_loss_real:0.3375095725059509
d_loss_fake:1.326506549048645e-06
d_loss_wrong:0.0036393743939697742
d_loss:0.16966496147810517
g_loss:[0.32757434248924255, 0.3275129497051239, 3.0702831281814724e-05]
```

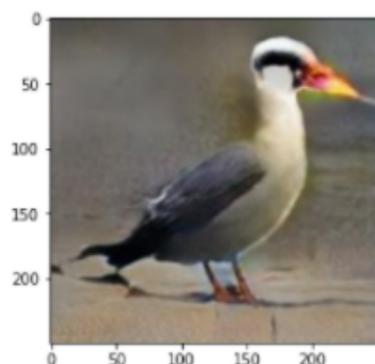
شکل ۴۸: نتیجه آموزش شبکه

همانگونه که مشاهده میشود، بعد از آخرین اپیاک مشخصه به عدد مناسبی رسیده ایم و به نتایج دلخواه شبکه هم میرسیم، نمونه ای از خروجی مرحله اول StackGAN در شکل زیر مشاهده میشود:



شکل ۴۹: نمونه خروجی از مرحله اول شبکه

حال کافیست کد مربوط به مرحله دوم را برای رسیدن به یک تصویر با وضوح بالا و با کیفیت اجرا نماییم. نتیجه حاصل شده از مرحله دوم شبکه StackGAN در شکل زیر قابل مشاهده میباشد:



شکل ۵۰: نمونه خروجی از مرحله دوم شبکه

همانگونه که مشاهده میشود، در این مرحله، نقص های موجود در تصویر با وضوح پایین از مرحله ۱ تصحیح شده است و با خواندن دوباره توضیحات متن ، تولید یک عکس واقع گرایانه با وضوح بالا، جزئیات شی را کامل کرده است.