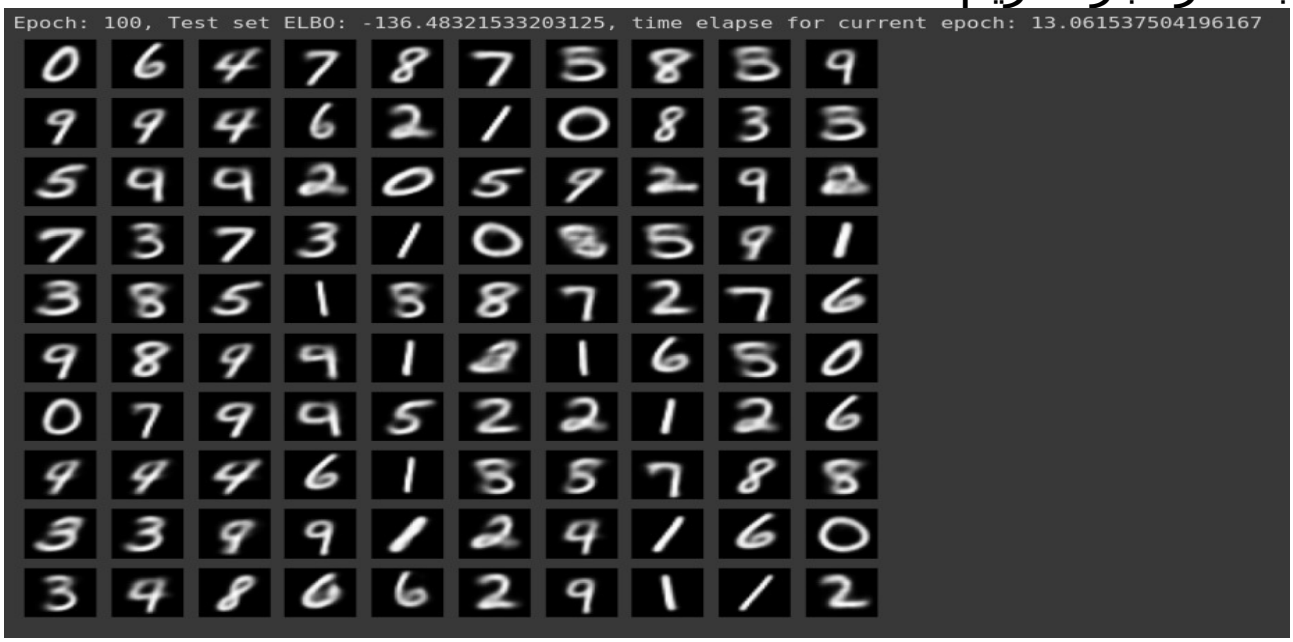
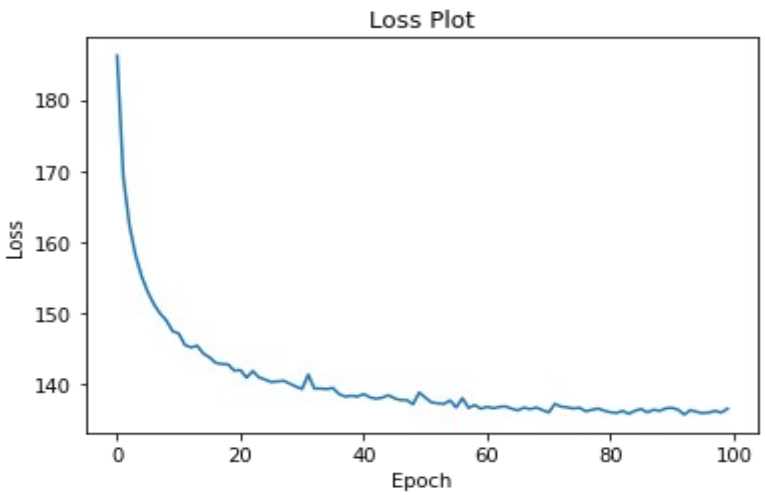
سؤال ۱)

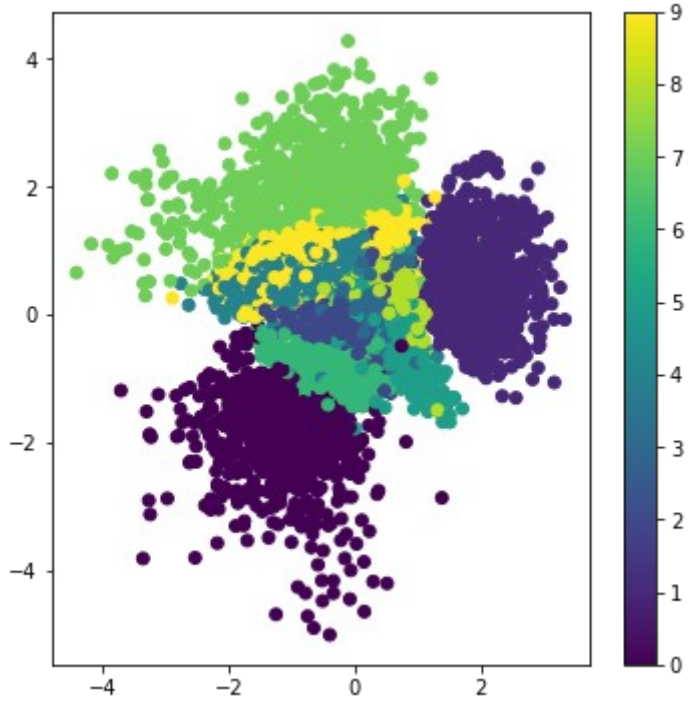
۱) تفاوت‌ها :
یکی از تفاوت‌های آن‌ها این است است در شبکه‌های VAE برای چک کردن میزان قدرت شبکه کافی است تابع لاس آن را ببینیم و ۲ شبکه VAE را با هم مقایسه کنیم ولی در شبکه GAN باید حتماً عکس‌های خروجی را ببینیم تا بتوانیم ۲ شبکه را مقایسه کنیم.
که این مزیت VAE بر GAN نیز هست
یکی دیگر از تفاوت‌های VAE این است که همانند یک AE نیز عمل می‌کند و می‌تواند ابعداد داده را نیز کمتر کند این باعث می‌شود شبکه‌های VAE برای پیدا کردن پترن های مخفی داده‌ها بهتر باشد ولی برای تولید داده‌های جدید ضعیف‌تر عمل می‌کند
شبکه‌های GAN ولی از راه دیگری برای تشخیص استفاده می‌کنند که آن تشخیص داده اصلی از فرعی توسط discriminator است که توسط generator ساخته می‌شود
مهمترین ویژگی GAN نسبت به VAE این است که می‌تواند داده‌هایی که generate می‌کند را نسبت به یک شرط انجام دهد به این معنی که داده‌های خاصی را تولید کند که با عث می‌شود در تولید داده‌های جدید و متفاوت بهتر از VAE باشد ولی VAE در تولید بهتر پترن های همان داده ورودی بهتر است

شبکه و نحوه پیاده‌سازی در کد توضیح داده شده است.
بعد از اجرا داریم:





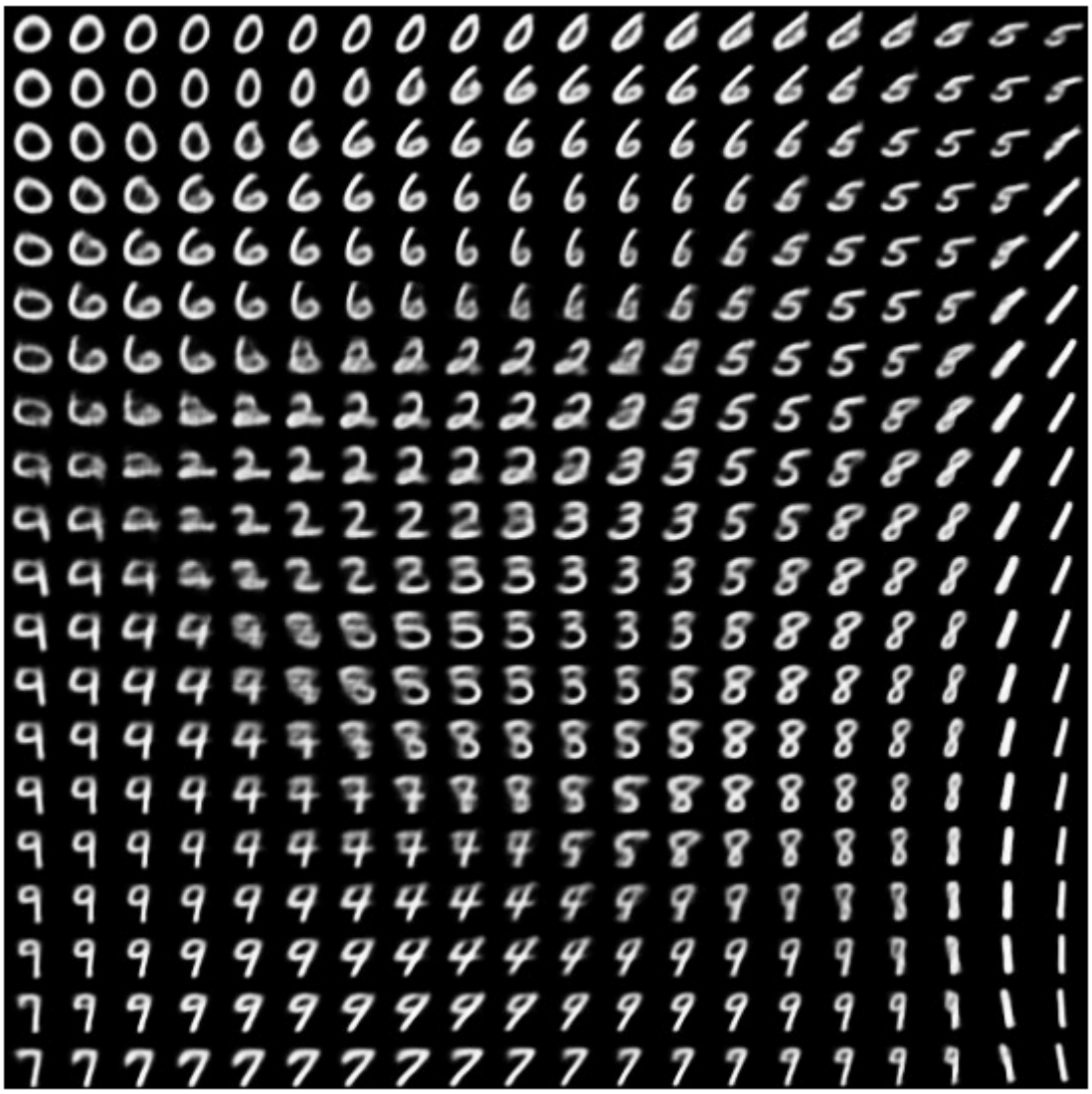
و برای کلاستر کردن آن داریم:



ج)

در این قسما ما از تابع هزینه ی ELBO یا evidance lower bound بر روی log-likelihood خروجی استفاده کرده‌ایم که به این صورت است که ابتدا تابع sigmoid_cross_entropy_with_logits حساب می‌کنیم و سعی می‌کنیم مقدار logp(x|z) + logp(z) – logq(z|x) را اپتیموم کنیم که در آن x ورودیو عکس generate شده است و z همان میانگین و log variance ما که توسط encoder تولید می‌شود است

از این تابع هزینه استفاده شده است تا با اپتیموم کردن عبارتی که در بالا گفته شد بتوانیم مقدار آن را به مقدار p(x) که درواقع همان ورودی اصلی ما می‌شود نزدیک کنیم. می‌دانیم VAE ها نیز می‌خواهند با نزدیک شدن به ورودی تصاویری شبیه به آن درست کنند.

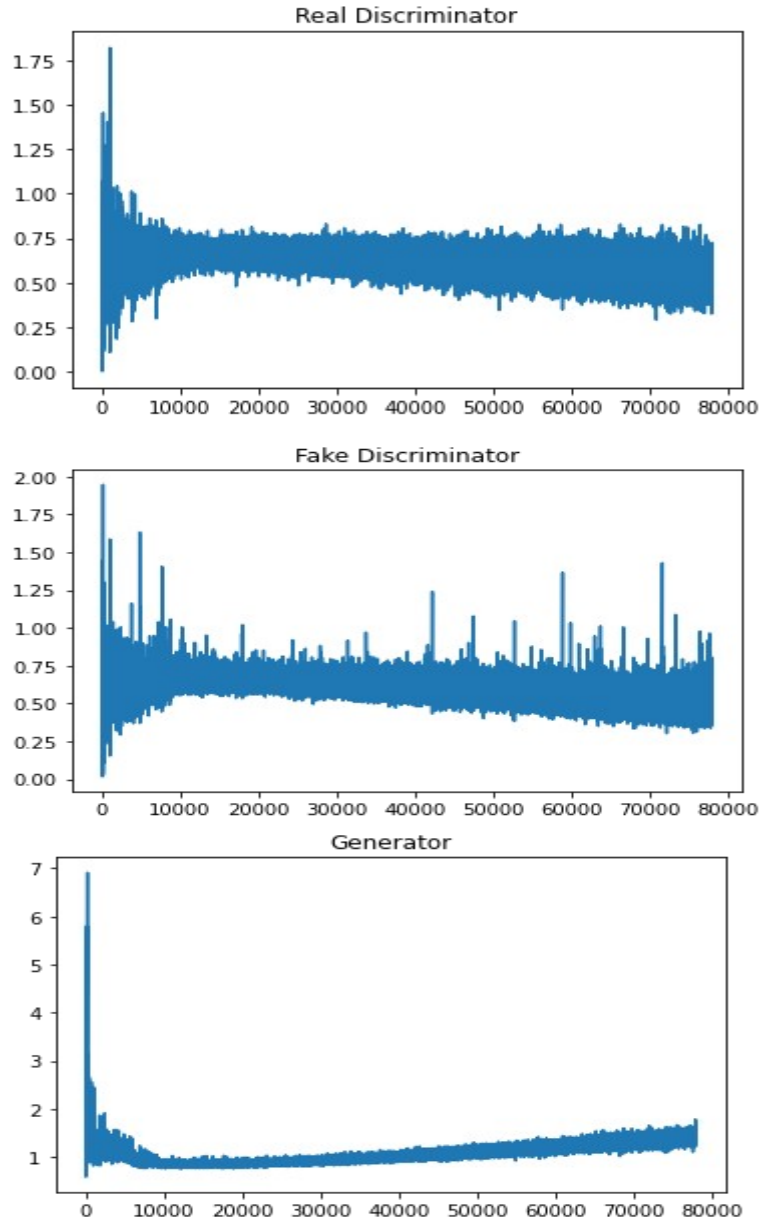


سؤال ۲)
در این قسمت می‌خواهیم یک شبکه DCGAN را پیاده‌سازی کنیم:
1)
در معماری DCGAN ما برای تولید عکس از Generative adversial network به جای متد های قدیمی تر از deep convolution ها استفاده می‌کنیم به این صورت که با تعداد لایه‌های کانولوشن پشت هم سعی می‌کنیم تا عکس هایی شبیه به عکس‌های ورودی و اضافه کردن نویز به آن‌ها تولید کنیم. در روش‌های قدیمی که تا حدی موفقیت آمیز نیز بودند تصویر های خروجی تا حدی آشکار بودند و کاملاً دیده نمی شدند ولی در روش DCGAN که بعد ها به GAN ها معرفی شد توانستند عکس هایی با وضوح بالاتر تولید کنند. در معماری DCGAN بر خلاف روش‌های شبکه عصبی دیگر به دلیل استفاده از لایه‌های کانولوشن تا حدی می‌توان ساختار درونی شبکه را ملاحضه کرد و کاربرد هر کدام از لایه‌های کانولوشن را حدس زد در حالی که در شبکه‌های عصبی معمولاً نمی‌توان کاربرد هر لایه در پیدا کردن ویژگی خاصی را فهمید

۲) در شبکه‌های GAN ما برای اینکه بتوانیم یک عکس که شبیه به ورودی است ولی با ان تفاوت دارد تولید کنیم نیاز داریم تا از یک فضایی که تعریف می‌کنیم نویز به وجود اوریم و با استفاده از آن عکس‌ها را دچار تغییر کنیم
برای این کار از فضایی که برای نویز latent space در نظر گرفتیم که در اینجا ۱۰۰ است برای هر ورودی عکس یکی را شانسی انتخاب کرده و با آن سعی می‌کنیم ورودی را تولید کنیم
شبکه انقدر این کار را انجام می‌دهد تا یادبگیرد با استفاده از همین ورودی نویز عکسی شبیه به عکس ورودی تولید کند.

3) در معماری DCGAN به جای لایه‌های Pooling می‌توانیم از strided convolution ها استفاده کنیم اینکار باعث می‌شود علاوه بر کم کردن بعد های هر لایه که pooling نیز انجام می‌داد شبکه بتواند این کم کردن را خودش نیز یاد بگیرد به این معنی که در کم کردن ابعداد ورودی هر لایه شبکه خودش یادنیگیرد که کدام قسمت‌ها از ورودی را با چه ضریبی نگه دارد این کار باعث تولید عکس‌های بهتر در generator می شود.

4) بعد از پیاده‌سازی DCGAN در خروجی داریم:
عکس‌ها در پیوست آمده‌اند بعد از epoch ۲۰۰ و نمودار ها به شکل:



5)
در معماری GAN ما ۲ شبکه برای Generator و discriminator داریم که با آموزش آن‌ها generator می‌تواند عکس هایی شبیه به ورودی تولید کند و discriminator می‌تواند فرق بین عکس‌های ساخته شده و واقعی را تشخیص دهد
یکی از مشکلات این شبکه‌ها این است که پیدا تابع loss است چرا که ما ۲ مدل داریم و مدل generator به صورت مستقیم آموزش نمی‌بیند درواقع ما مدل discriminator را طوری آموزش می‌دهیم تا بتواند تابع loss برای مدل generator را درست کند و شبکه gan سعی می‌کند تا این loss را مینیموم کند.
حال چند تابع loss را معرفی می کنیم:
۱) Minimax GAN Loss: مدل های generator و discriminator مانند ۲ بازیکن در شبکه عمل می‌کنند و این تابع باعث min کردن loss مدل generator و max کردن loss مدل discriminator است تا شبکه بتواند عکس‌های نزدیک به خروجی و در عین حال متفاوت با ورودی تولید کند
۲) Least square GAN Loss: در این تابع مدل discriminator سعی می‌کند تا جمع اختلاف مربعات بین مقادیر واقعی و مقادیر بدست آمده را مینیموم کند مانند زیر:

discriminator: minimize (D(x) – 1)^2 + (D(G(z)))^2

و در عین حال مدل generator سعی می‌کند تا جمع مربع اختلاف عکس واقعی آن با عکسی که از آن تولید کرده را مینیموم کند کانند زیر:

generator: minimize (D(G(z)) – 1)^2

۳) Wassertein GAN Loss: در این تابع مدل discriminator به تعداد بار بیشتری نسبت به مدل generator اپدیت می‌شود و اپدیت ها به این شکل است که به جای پیدا کردن یک درصد برای واقعی یا فیک بودن عکس به ما بدهد یک مقدار و امتیاز حقیقی به آن می‌دهد و این امتاز طوری محاسبه می‌شود که فاصله ی عکس واقعی با عکس تولید شده بیشترین مقدار باشد در این تابع مقادیر وزن ها نیز باید خیلی کم باشند تا مدل بتواند از این تابع استفاده کند

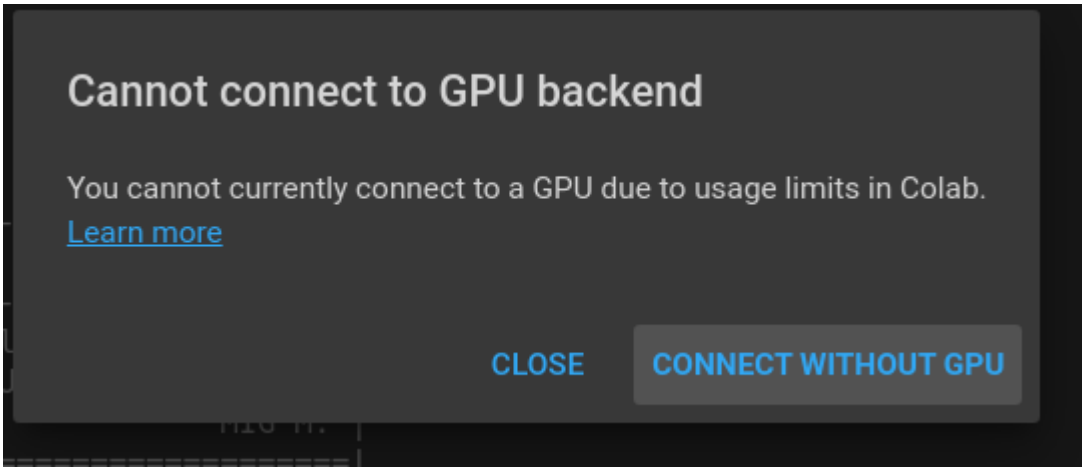
4) Cross-Entropy: این همان loss بعروف است که می‌تواند مقدار خروجی تولید شده و واقعی را محاسبه کند
این تابع با minimize کردن اختلاف تولیدی و واقعی که در اینجا همان ۰ به معنی عکس فیک و ۱ به معنی عکس واقعی است سعی می‌کند discriminator را آموزش دهد و با آموزش آن generator سعی می‌کند عکسی تولید کند که از نظر discriminator واقعی باشد و این آموزش انقدر ادامه پیدا می‌کند تا شبکه بتواند عکس هایی با وضوح بالا تولید کند

سؤال ۳)

۱) در این قسمت می‌خواهیم یک CGAN را مدل سازی کنیم CGAN نوعی دیگر از GAN ها هستند که برای درست کردن عکس هایی با توجه به عکس ورودی استفاده می‌شوند در این نوع GAN ها ما علاوه بر ورودی و خروجی در generator و discriminator یک شرط را نیز به آن‌ها اضافه می‌کنیم که می‌تواند لیبل یا یک قسما از اطلاعات عکس‌ها باشد در این صورت در generator تولید نویز و اطلاع از شرط اضافی می‌تواند در ایک لایه با هم قرار گیرند به صورتی که نویز با توجه به شرط ساخته می‌شود و به generator داده می‌شود و در discriminator هم عکس ساخته شده و هم شرط به عنوان ورودی به آن داده می‌شود با استفاده از لایه ی embed و در این صورت مدل ها به جای max و min کردن نویز و خروجی آن‌ها را به شرط داشتن اطلاع اضافی max و min می‌کنند در این صورت شبکه می‌تواند آموزش ببیند تا با داشتن یک شرط عکس هایی از همان شرط و با همان اطلاع اضافی تولید کند چرا که شبکه می‌خواهد loss را کاهش دهد و این loss شامل تولید خروجی به شرط داشتن یک اطلاع اضافی است

2) بعد از پیاده‌سازی داریم:

به دلیل ارور کولب نتوانستم دوباره به gpu متصل شوم تا نمودار ها و عکس را دوباره ران کنم ولی کد آن موجود است



ظاهراً حتی کدی که جواب درست تولید می‌کرد نویز تولید کرد عکس نیز از قبل ذخیره شده بوده است

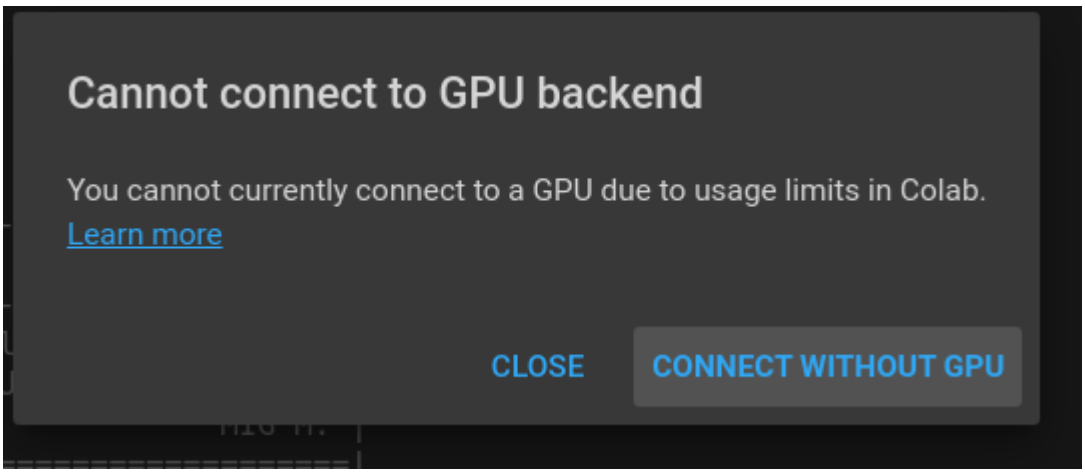
۳) در اینجا می‌خواهیم info GAN را پیاده‌سازی کنیم:
در ساختار infoGan مدل generator ما علاوه بر ورودی نویز یک فضای دیگری را نیز به عنوان ورودی می‌گیرد که شامل یک نوع اطلاعاتی از عکس‌ها است. این فضا درواقع اضافه کردن یک ترم به تابع لاس کلی GAN هاست مانند زیر:

$$\min_G \max_D V_I(D,G) = V(D,G) - \lambda I(c; G(z,c))$$

که در آن I درواقع اطلاعات مشترک بین فضای ورودی جدید generator و ورودی نویز آن است. پیدا کردن این فضا به تنهایی کاری دشوار است به همین دلیل یک کران پایین آن تعیین می‌کنیم و به همین دلیل یک ترم likelihood اضافه شده تا با محاسبه احتمال آن بتوانیم فضای جدید را بدست اوریم در این صورت برای مدل generator ما هر دفعه یک سمپل از نویز و یک سمپل از فضای جدید می‌گیریم و با محاسبه ی ترم رگولاریزیشن جدید می‌توانیم تابع loss را مینیموم کنیم و شبکه را آموزش دهیم. حال به غیر از این فضای ورودی جدید برای پیدا کردن مقدار عکس تولید شده از این فضای جدید نیاز به یک شبکه جدای دیگر نیز داریم که درواقع یک لایه fully connected است که به انتهای مدل discriminator قرار می‌گیرد و فقط زمانی مورد استفاده قرار می‌گیرد که عکس تولیدی فیک باشد چرا که فقط در این زمان است که سمپل گرفته شده از فضای جدید معلوم می شود. برای پیدا کردن آن سمپل از کد نیازی به فهمیدن خود آن نیست و باید likelihood وجود آن سمپل را در عکس تولید شده توسط generator پیدا کردن که این کار با استفاده از همین لایه جدید که به q_model معروف است استفاده کرد

بعد از پیاده‌سازی آن داریم:

به دلیل ارور کولب نتوانستم دوباره به gpu متصل شوم تا نمودار ها و عکس را دوباره ران کنم ولی کد آن موجود است



ظاهراً حتی کدی که جواب درست تولید می‌کرد نویز تولید کرد عکس نیز از قبل ذخیره شده بوده است