تمرین سری سوم – درس یادگیری ژرف

فاطمه براتی 4023245503

۱- برای آموزش یک شبکه GAN میتوانستیم روش گرادیان کاهشی را روی تابع خطای قسمت generator اعمال کنیم. ولی به جای آن تابع خطا را تغیر دادیم و روش گرادیان افزایشی را روی تابع تغییر یافته اعمال کردیم. این تغیر چه بود و دلیل این کار را توضیح دهید.

در شبکه‌های مولد تخاصمی (GANs)، مولد (Generator) و تفکیک‌کننده (Discriminator) به صورت رقابتی آموزش داده می‌شوند. تفکیک‌کننده سعی می‌کند داده‌های واقعی را از داده‌های جعلی به درستی تشخیص دهد، در حالی که مولد تلاش می‌کند داده‌های جعلی تولید کند که بتواند تفکیک‌کننده را فریب دهد.

فرآیند آموزش معمول شامل مراحل زیر است:

- تفکیک‌کننده: کمینه کردن تابع زیان (Loss Function) برای بهبود توانایی تشخیص داده‌های واقعی از داده‌های جعلی. این کار معمولاً با استفاده از گرادیان نزولی (Gradient Descent) انجام می‌شود.

- مولد: بیشینه کردن تابع زیان برای بهبود توانایی فریب دادن تفکیک‌کننده در تشخیص داده‌های جعلی به عنوان واقعی. این کار معمولاً با استفاده از گرادیان صعودی (Gradient Ascent) انجام می‌شود.

چرا از گرادیان صعودی برای مولد استفاده می‌کنیم؟

1. هماهنگی هدف: هدف مولد این است که احتمال اینکه تفکیک‌کننده داده‌های جعلی را به عنوان واقعی طبقه‌بندی کند، بیشینه کند. این کار را می‌توان به عنوان بیشینه کردن زیان تفکیک‌کننده در مواجهه با داده‌های جعلی در نظر گرفت که با هدف مولد هم‌خوانی دارد.

2. فرمول‌بندی ریاضی: اگر تابع زیان تفکیک‌کننده در هنگام درست طبقه‌بندی کردن داده‌های واقعی $\log(D(x))$ باشد و برای داده‌های جعلی $\log(1 - D(G(z)))$ باشد، مولد می‌خواهد $\log(1 - D(G(z)))$ را کمینه کند. کمینه کردن این زیان به طور مستقیم با گرادیان نزولی می‌تواند منجر به ناپدید شدن گرادیان‌ها شود وقتی که $D(G(z))$ بسیار کوچک است و آموزش را دشوار می‌کند. در عوض، بیشینه کردن $\log(D(G(z)))$ از طریق گرادیان صعودی تضمین می‌کند که گرادیان‌های بیشتری برای یادگیری مولد وجود دارد.

مزایا و معایب گرادیان صعودی برای مولد:

مزایا:

1. گرادیان‌های بهتر: گرادیان صعودی روی log(D(G(z))) معمولاً گرادیان‌های قوی‌تری نسبت به گرادیان نزولی روی log(1 - D(G(z))) فراهم می‌کند، به ویژه هنگامی که تفکیک‌کننده قوی است و D(G(z)) نزدیک به صفر است.

2. پایداری آموزش: هماهنگی صحیح اهداف آموزشی (تفکیک‌کننده که زیان خود را کمینه می‌کند و مولد که موفقیت خود را بیشینه می‌کند) می‌تواند منجر به پایداری بیشتر در دینامیک آموزشی و همگرایی شود.

3. بهینه‌سازی مستقیم هدف: گرادیان صعودی به طور مستقیم هدف مولد را برای فریب دادن تفکیک‌کننده بهینه می‌کند و فرآیند آموزش را شهودی‌تر و موثرتر می‌کند.

معایب:

1. پیچیدگی پیاده‌سازی: پیاده‌سازی گرادیان صعودی نیاز به مدیریت دقیق برای اطمینان از پایداری عددی و همگرایی دارد.

2. خطر ناپایداری: در حالی که گرادیان صعودی می‌تواند به گرادیان‌های بهتر منجر شود، اما اگر نرخ‌های یادگیری به درستی تنظیم نشوند، می‌تواند باعث ناپایداری شود زیرا مولد ممکن است از نقاط بهینه عبور کند.

3. تعادل آموزش: سرعت‌های آموزش مولد و تفکیک‌کننده باید متعادل باشند. اگر یکی نسبت به دیگری به طور قابل توجهی پیشی بگیرد، می‌تواند منجر به عملکرد کلی ضعیف و عدم همگرایی شود.

استفاده از گرادیان صعودی برای مولد در GANs با هدف فریب دادن تفکیک‌کننده هماهنگ است و می‌تواند به آموزش موثرتر و پایدارتر منجر شود. با این حال، نیاز به پیاده‌سازی دقیق و تعادل با آموزش تفکیک‌کننده دارد تا از ناپایداری جلوگیری کرده و اطمینان حاصل شود که همگرایی صورت می‌گیرد. کلید این کار، فراهم کردن گرادیان‌های قوی و اطلاعاتی برای مولد است که آن را به سمت تولید داده‌های واقعی‌تر هدایت کند.

۲- الف- به آدرس زیر بروید و توضیحات و کدها را به دقت مطالعه کنید. سپس این کدها را در پایتون در کامپیوتر خودتان یا در یک سرور ابری اجرا کنید و نتایج را به صورت یک گزارش کوتاه ارائه دهید.

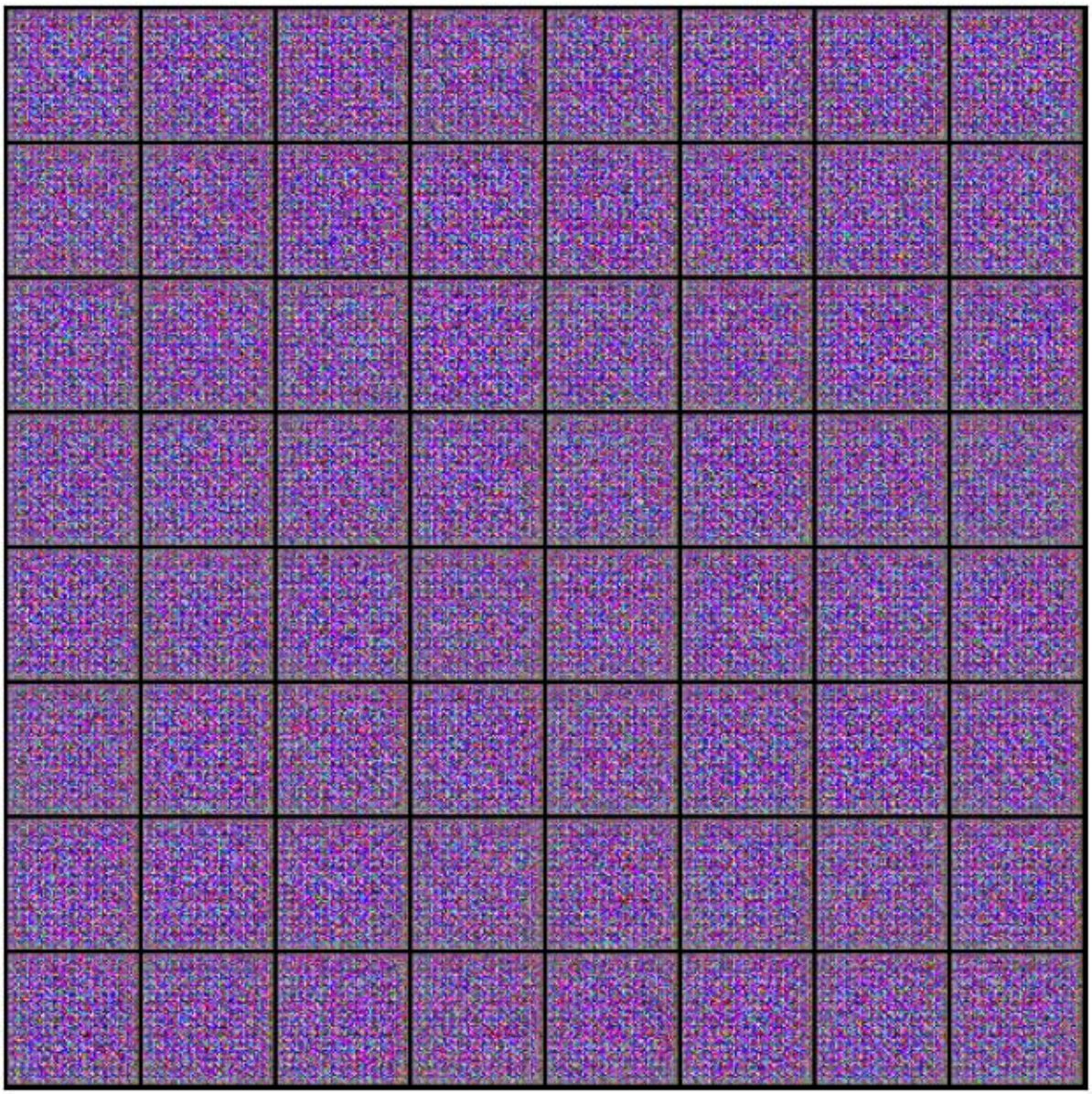
https://towardsdatascience.com/getting-started-with-gans-using-pytorch-78e7c22a14a5

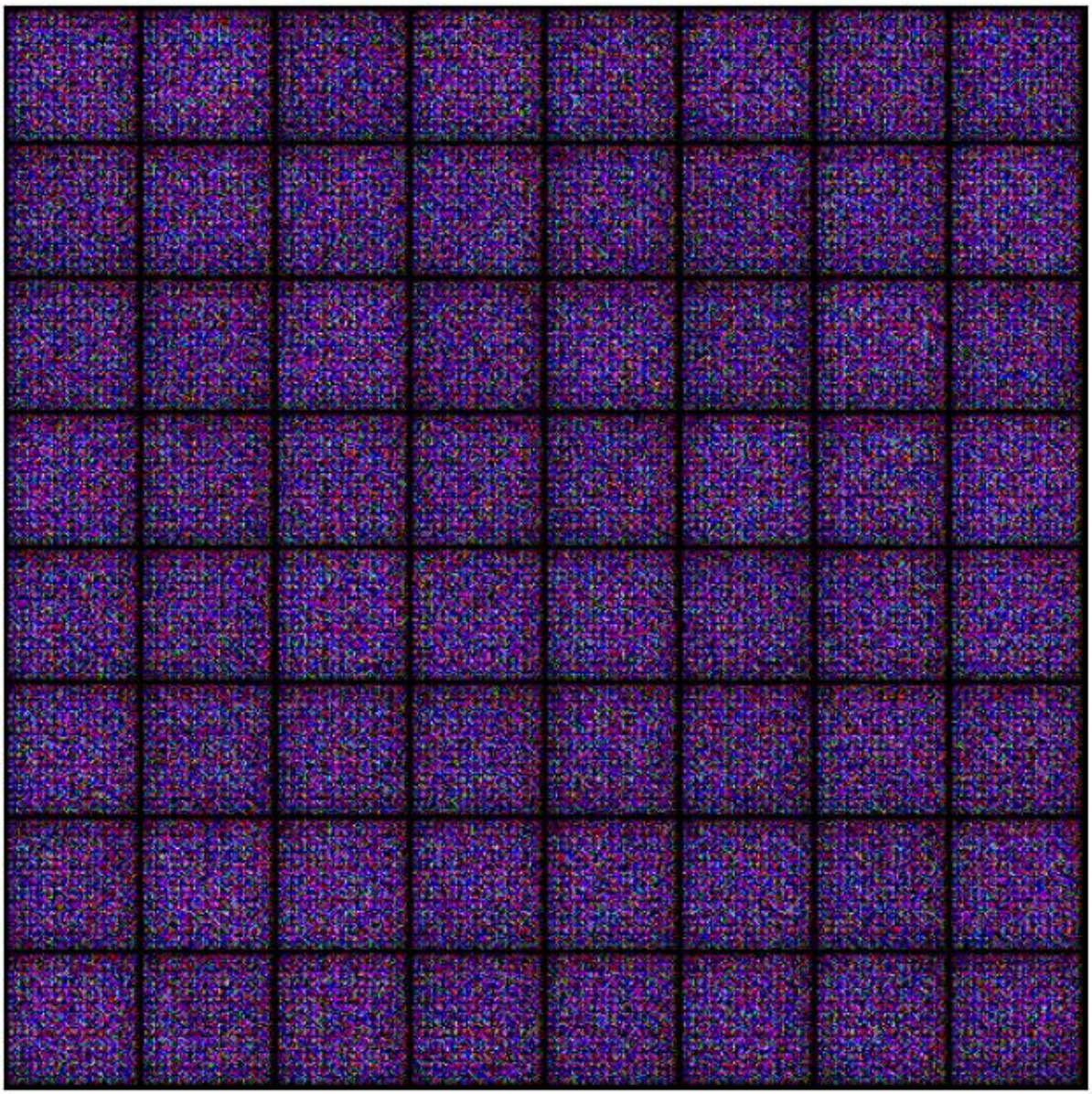
کدهای مربوط به این سوال (کد اصلی و کد تغییر داده شده) در زیپ قرار داده شده اند.

نتایج:

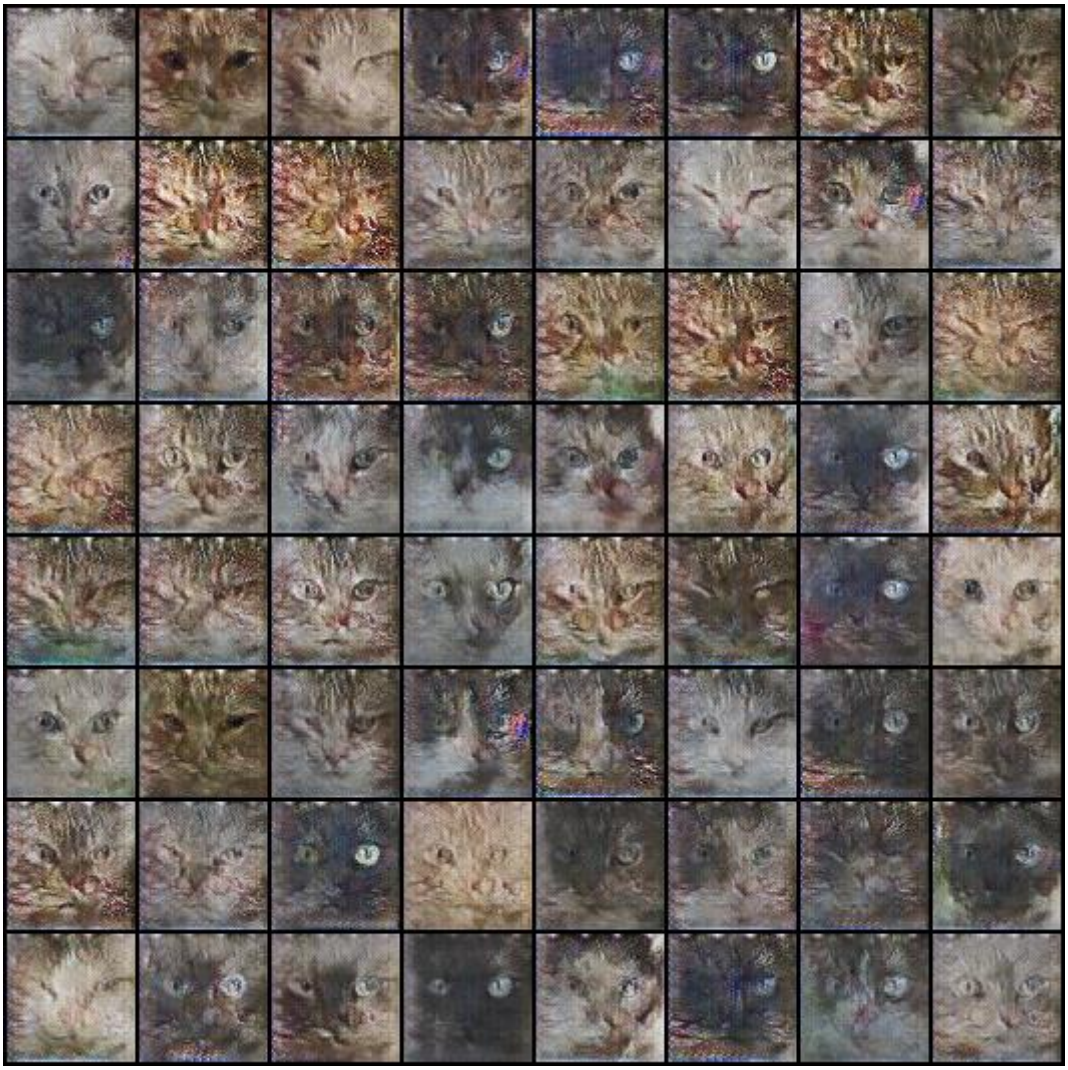
عکس های دیتاست به این صورت هستند.

Generator بدون کمک Discriminator تنها می تواند عکس های نامفهوم و نویزمانند درست کند.

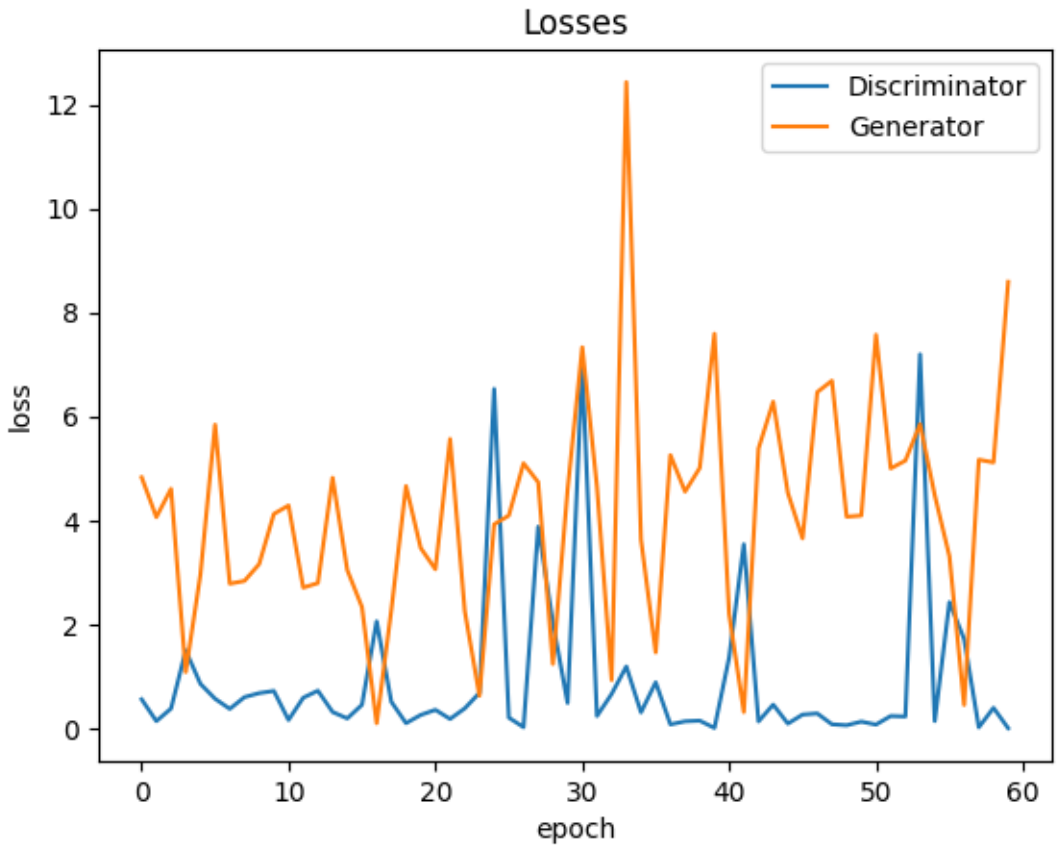




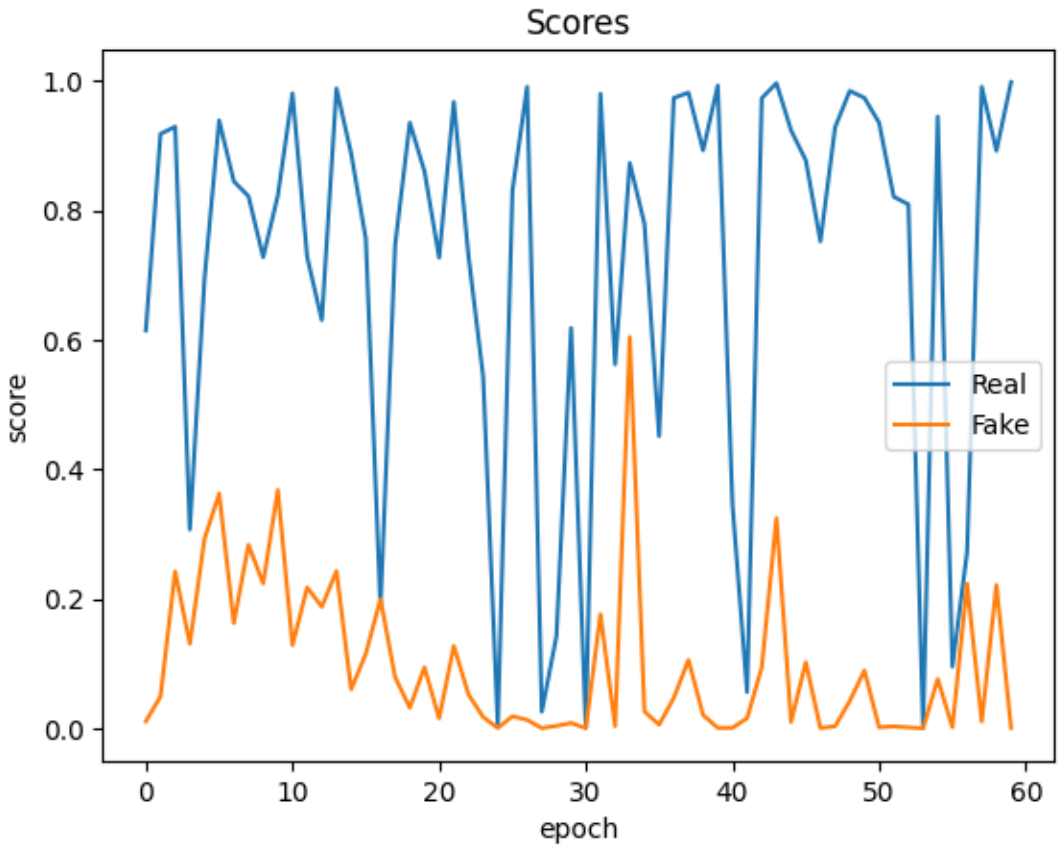
پس از آن که Generator و Discriminator به رقابت پرداختند، حال Generator قادر است عکس هایی مشابه دیتاست تولید کند که Discriminator نتواند متوجه fake بودن آن ها بشود.



نمودار Loss ها:



نمودار Score ها:



ب- توضیح مختصری راجع به دیتاست، generator و discriminator و پیاده سازی آن در کد بالا ارائه دهید.

ما از Cats Faces Dataset استفاده می کنیم که از بیش از 15700 تصویر گربه تشکیل شده است. از آنجایی که مدل سازی تولیدی یک روش یادگیری بدون نظارت است، بنابراین هیچ برچسبی روی تصاویر وجود ندارد.

مولد (Generator):

مولد یک شبکه عصبی است که برای تولید تصاویر واقعی از نویز تصادفی طراحی شده است. در این کد، معماری مولد از چندین لایه `d2ConvTranspose` برای افزایش تدریجی اندازه بردارهای تصادفی (با اندازه `latent_size`) به تصاویر 64x64 استفاده می‌کند. هر لایه کانولوشنی پس از نرمال‌سازی دسته‌ای و تابع فعال‌سازی ReLU قرار دارد، به جز لایه آخر که از تابع فعال‌سازی Tanh برای تولید تصویر نهایی استفاده می‌کند.

متمایزکننده (Discriminator):

متمایزکننده یک شبکه عصبی است که تلاش می‌کند تصاویر واقعی و جعلی را از هم تشخیص دهد. در این کد، متمایزکننده از چندین لایه `d2Conv` برای کاهش اندازه تصاویر ورودی از 64x64 به یک مقدار اسکالر استفاده می‌کند که نشان‌دهنده احتمال

واقعی بودن تصویر است. هر لایه کانولوشنی پس از نرمال‌سازی دسته‌ای و تابع فعال‌سازی LeakyReLU قرار دارد. لایه آخر از تابع فعال‌سازی Sigmoid برای خروجی یک مقدار استفاده می‌کند.

خلاصه کد:

1. تنظیمات و آماده‌سازی داده:

- فایل `kaggle.json` را برای دسترسی به API کگل آپلود می‌کنیم.

- API کگل را نصب کرده، دیتاست چهره گربه‌ها را دانلود و از حالت فشرده خارج می‌کنیم.

- دیتاست را با استفاده از `ImageFolder` آماده کرده و تبدیل‌هایی مانند تغییر اندازه، برش مرکزی و نرمال‌سازی را اعمال می‌کنیم.

- یک DataLoader برای بارگذاری تصاویر در دسته‌ها ایجاد می‌کنیم.

2. تنظیم دستگاه:

- بررسی می‌کنیم که آیا GPU موجود است و دستگاه را بر این اساس تنظیم می‌کنیم.

- توابع کمکی برای انتقال داده‌ها به دستگاه انتخاب شده تعریف کرده و DataLoader را برای انتقال خودکار داده‌ها به دستگاه فراخوانی می‌کنیم.

3. تعاریف مدل:

- معماری متمایزکننده و مولد را تعریف می‌کنیم.

- هر دو مدل را به دستگاه انتخاب شده منتقل می‌کنیم.

4. توابع آموزش:

- تابع `train_discriminator` را برای به‌روزرسانی وزن‌های متمایزکننده تعریف می‌کنیم.

- تابع `train_generator` را برای به‌روزرسانی وزن‌های مولد تعریف می‌کنیم.

5. حلقه آموزش:

- تابع `fit` را برای آموزش هر دو مدل برای تعداد مشخصی از دوره‌ها تعریف می‌کنیم.
- در هر دوره، بین آموزش متمایزکننده و مولد تناوب ایجاد می‌کنیم.
- تصاویر تولید شده را در هر دوره ذخیره کرده و ضررها و نمرات را ثبت می‌کنیم.

6. بصری‌سازی:

- نمودار ضررهای مولد و متمایزکننده در طول دوره‌ها را ترسیم می‌کنیم.
- نمودار نمرات واقعی و جعلی در طول دوره‌ها را ترسیم می‌کنیم.
- یک ویدیو از تصاویر تولید شده ایجاد می‌کنیم تا پیشرفت آموزش را بصری‌سازی می‌کنیم.

این کد یک مدل GAN کانولوشن عمیق (DCGAN) را برای تولید تصاویر چهره گربه‌ها پیاده‌سازی می‌کند. فرآیند آموزش شامل هر دو مولد و متمایزکننده است که تلاش می‌کنند یکدیگر را شکست دهند، با این که مولد تصاویر واقعی‌تر تولید می‌کند و متمایزکننده در تشخیص تصاویر واقعی و جعلی بهتر می‌شود.

ج- سعی کنید لایه کانولوشنی آخر را حذف کنید. بعد از حذف این لایه در صورت لزوم تغییرات لازم برای اجرای صحیح و بدون خطای کد را انجام دهید. نتیجه این کار را در دقت نهایی گزارش کنید.

با حذف لایه ی آخر، معماری شبکه به هم می خورد و لازم است کارایی لایه ی ما قبل آخر به گونه ای تغییر کند که بتواند کار دو لایه ی آخر را انجام دهد. به عبارت دیگر، دو لایه ی آخر باید در یک لایه خلاصه شوند.

حال به بررسی معماری های اصلی می پردازیم:

در یک شبکه مولد متخاصم عمیق (DCGAN)، مولد و متمایزکننده از طریق نقش‌های خود در فرایند آموزش متخاصم با یکدیگر ارتباط و تعامل دارند.

متمایزکننده:

متمایزکننده برای تشخیص بین تصاویر واقعی و تولید شده طراحی شده است. معماری آن به تدریج تصویر ورودی را پایین‌نمونه کرده و در هر مرحله ویژگی‌های سطح بالاتری را استخراج می‌کند. در اینجا خلاصه‌ای از لایه‌های متمایزکننده آمده است:

1. لایه ورودی:

- ورودی: 3 64x 64x (تصویر رنگی).

- d2Conv: از 3 به 64 (ویژگی‌ها)، کرنل 4x4، گام 2، پدینگ 1.

- d2BatchNorm، LeakyReLU: نرمال‌سازی و فعال‌سازی.

- خروجی: 64 x32 x32.

2. لایه‌های مخفی:

- الگوی مشابه: کانولوشن -> LeakyReLU <- BatchNorm.

- گسترش تدریجی ویژگی‌ها: از 64 به 128، 128 به 256، 256 به 512.

- کاهش اندازه فضایی: از 32 به 16، 16 به 8، 8 به 4.

3. لایه خروجی:

- d2Conv: از 512 به 1 (طبقه‌بندی باینری)، کرنل 4x4، گام 1، پدینگ 0.

- Flatten، Sigmoid: تبدیل به یک مقدار واحد، اعمال Sigmoid برای بدست آوردن احتمال.

- خروجی: 1 (واقعی/جعلی).

مولد:

مولد برای ایجاد تصاویر واقعی از نویز تصادفی طراحی شده است. این کار را با افزایش تدریجی اندازه و بهبود نویز ورودی انجام می‌دهد. در اینجا خلاصه‌ای از لایه‌های مولد آمده است:

1. لایه ورودی:

- ورودی: بردار نهفته (مثلاً 128 x1 x1).

- d2ConvTranspose: از 128 به 512 (ویژگی‌ها)، کرنل 4x4، گام 1، پدینگ 0.

- d2BatchNorm، ReLU: نرمال‌سازی و فعال‌سازی.

- خروجی: 512 x4 x4.

2. لایه‌های مخفی:

- الگوی مشابه: ConvTranspose -> BatchNorm -> ReLU.

- کاهش تدریجی ویژگی‌ها: از 512 به 256، 256 به 128، 128 به 64.

- افزایش اندازه فضایی: از 4 به 8، 8 به 16، 16 به 32.

3. لایه خروجی:

- d2ConvTranspose: از 64 به 3 (کانال‌های تصویر)، کرنل 4x4، گام 2، پدینگ 1.

- Tanh: تابع فعال‌سازی برای مقیاس‌بندی خروجی بین -1 و 1.

- خروجی: 3 64x 64x.

رابطه و تعامل

1. آموزش متخاصم:

- مولد: یک بردار نویز تصادفی (بردار نهفته) می‌گیرد و یک تصویر تولید می‌کند.

- متمایزکننده: یک تصویر (واقعی یا تولید شده) می‌گیرد و احتمالی که تصویر واقعی است را خروجی می‌دهد.

2. توابع Loss:

- Loss مولد: تشویق به تولید تصاویری که متمایزکننده آنها را به عنوان واقعی طبقه‌بندی می‌کند.

- Loss متمایزکننده: تشویق به طبقه‌بندی صحیح تصاویر واقعی و تولید شده.

3. حلقه بازخورد:

- متمایزکننده: گرادیان‌ها را به مولد ارائه می‌دهد و راهنمایی می‌کند که چگونه تصاویر واقعی‌تری تولید کند.

- مولد: تصاویر تولید شده را به متمایزکننده ارائه می‌دهد و آن را بهبود می‌دهد تا توانایی طبقه‌بندی خود را افزایش دهد.

4. فرایند آموزش:

- به طور متناوب بین آموزش متمایزکننده و مولد.

- متمایزکننده: آموزش داده می‌شود تا احتمال طبقه‌بندی صحیح تصاویر واقعی و جعلی را به حداکثر برساند.

- مولد: آموزش داده می‌شود تا احتمال طبقه‌بندی تصاویر تولید شده به عنوان جعلی را به حداقل برساند.

به‌روزرسانی Discriminator:

برای ترکیب عملکرد این دو لایه کانولوشن به یک لایه، می‌توانید از یک لایه `nn.Conv2d` استفاده می‌کنیم که مستقیماً از ۲۵۶ کانال به ۱ کانال نگاشت می‌کند. این لایه ترکیبی باید دارای اندازه کرنل و گامی باشد که اندازه خروجی مورد نظر (1*1) را ایجاد کند.

لایه‌های اصلی:

1. لایه اول `nn.Conv2d`:

- ورودی: ۲۵۶ کانال

- خروجی: ۵۱۲ کانال

- اندازه کرنل: ۴

- گام: ۲

- پدینگ: ۱

2. لایه دوم `nn.Conv2d`:

- ورودی: ۵۱۲ کانال

- خروجی: ۱ کانال

- اندازه کرنل: ۴

- گام: ۱

- پدینگ: ۰

لایه ترکیبی:

برای ترکیب اثر، باید یک پیکربندی لایه `nn.Conv2d` پیدا می‌کنیم که همان نتیجه را تولید کند. باید اثر ترکیبی کانولوشن‌ها و غیرخطی‌ها را در نظر می‌گیریم.

با توجه به تنظیمات، لایه اول ابعاد فضایی را از 8*8 به 4*4 کاهش می‌دهد و عمق را به ۵۱۲ افزایش می‌دهد. لایه دوم ابعاد فضایی را از 4*4 به 1*1 کاهش می‌دهد و عمق را به ۱ کاهش می‌دهد.

برای ترکیب آنها، ما نیاز داریم:

- ورودی: ۲۵۶ کانال

- خروجی: ۱ کانال

- اندازه کرنل: k

- گام: s

- پدینگ: p

ابعاد فضایی باید از 8*8 به 1*1 تغییر کند. گام و پدینگ موثر را می‌توان با توجه به تبدیل کلی محاسبه کرد.

Given the output dimensions $(H_{\text{out}}, W_{\text{out}})$:

$$H_{\text{out}} = \left\lfloor \frac{H_{\text{in}} + 2p - k}{s} \right\rfloor + 1$$

Setting $H_{\text{in}} = 8$, $H_{\text{out}} = 1$, and solving for k, s, and p, we get:

$$1 = \left\lfloor \frac{8 + 2p - k}{s} \right\rfloor + 1$$

ما متوجه می‌شویم که یک اندازه کرنل ۸، گام ۱، و بدون پدینگ کار می‌کند (زیرا مستقیماً 8*8را به 1*1 تبدیل می‌کند):

```
nn.Conv2d(256, 1, kernel_size=8, stride=1, padding=0, bias=False),
# out: 1 x 1 x 1

nn.Flatten(),
nn.Sigmoid()
```

این لایه اثر هر دو لایه کانولوشن، نرمال‌سازی دسته‌ای، و تابع فعال‌سازی را در یک لایه کانولوشن ترکیب می‌کند که مستقیماً یک نقشه ویژگی 1*1 را از ورودی 8*8 تولید می‌کند.

به‌روزرسانی Generator:

ترکیب این دو لایه `nn.ConvTranspose2d` به یک لایه نیازمند درک نحوه کار کانولوشن‌های معکوس (که به عنوان دکانولوشن‌ها نیز شناخته می‌شوند) در دنباله است. در اینجا توضیح داده شده است:

لایه‌های اصلی:

1. لایه اول `nn.ConvTranspose2d`:

- ورودی: ۱۲۸ کانال

- خروجی: ۶۴ کانال

- اندازه کرنل: ۴

- گام: ۲

- پدینگ: ۱

- اندازه خروجی: 32 x 32 x 64

2. لایه دوم `nn.ConvTranspose2d`:

- ورودی: ۶۴ کانال

- خروجی: ۳ کانال

- اندازه کرنل: ۴

- گام: ۲

- پدینگ: ۱

- اندازه خروجی: 64 x 64 x 3

لایه ترکیبی:

برای ترکیب اثر، باید یک پیکربندی لایه `nn.ConvTranspose2d` پیدا می‌کنیم که همان نتیجه نهایی را تولید کند.

با توجه به اینکه هر لایه `ConvTranspose2d` ابعاد فضایی را هنگام استفاده از گام ۲ و پدینگ ۱ دو برابر می‌کند، یک لایه باید همان بزرگ‌نمایی کلی از 16*16 به 64*64 را به دست آورد.

تبدیل از 16*16 به 32*32 و سپس از 32*32 به 64*64 را می‌توان به یک تبدیل واحد ترکیب کرد:

$$H_{\text{out}} = (H_{\text{in}} - 1) \times \text{stride} - 2 \times \text{padding} + \text{kernel size}$$

پیکربندی:

ترکیب دو گام ۲ منجر به یک گام موثر ۴ می‌شود. بنابراین، اندازه کرنل باید به طور مناسب انتخاب شود. از آنجا که هر مرحله شامل دو برابر شدن است، ما به اندازه کرنلی نیاز داریم که این بزرگ‌نمایی ترکیبی را فراهم کند.

ما می‌توانیم استفاده کنیم:

- ورودی: ۱۲۸ کانال

- خروجی: ۳ کانال

- اندازه کرنل: ۸ (جمع کردن مشارکت‌های فردی)

- گام: ۴

- پدینگ: ۲ (برای رسیدن به ابعاد فضایی مطلوب)

لایه جدید:

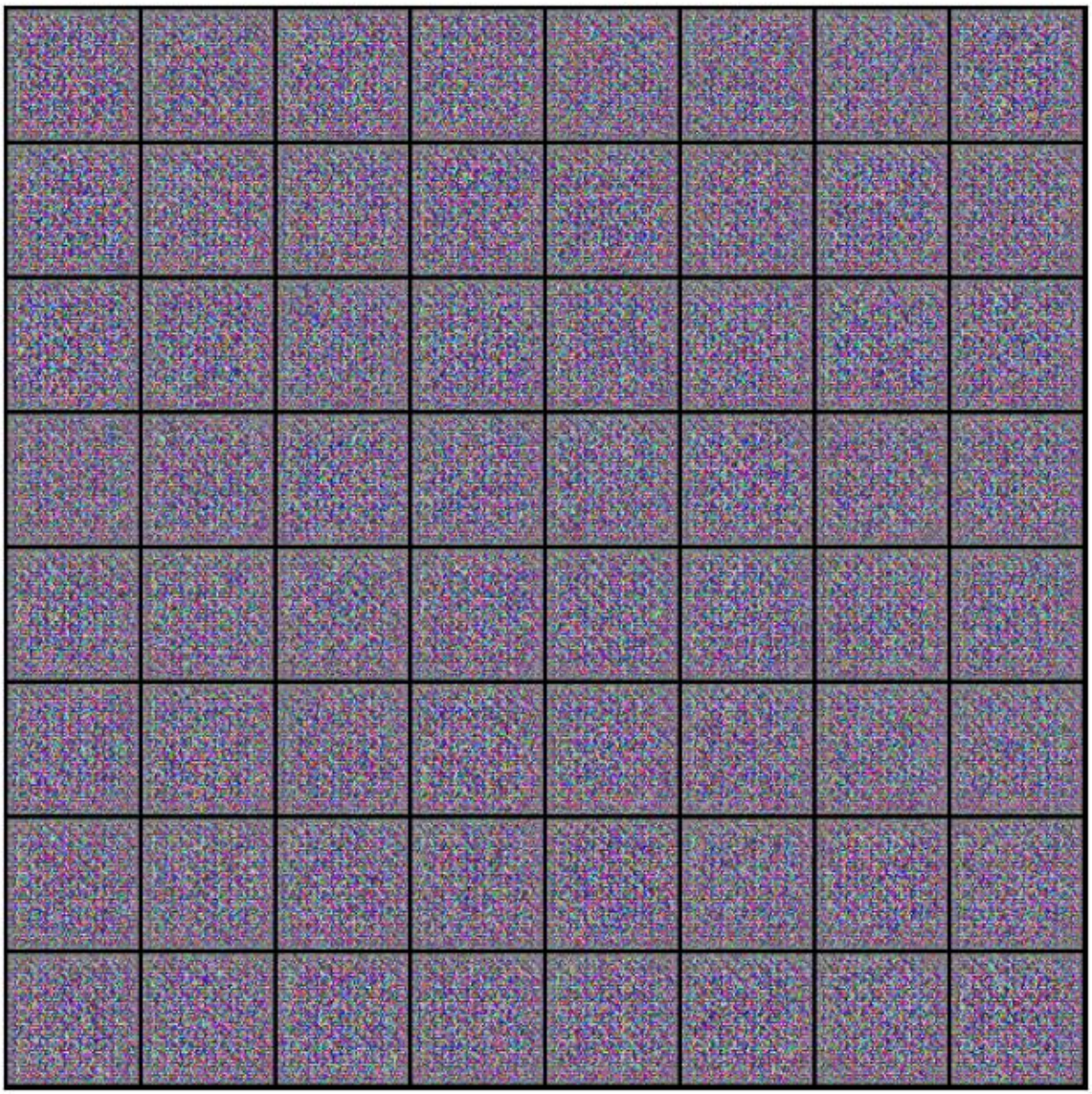
```
nn.ConvTranspose2d(128, 3, kernel_size=8, stride=4, padding=2, bias=False),
nn.Tanh()
out: 3 x 64 x 64
```

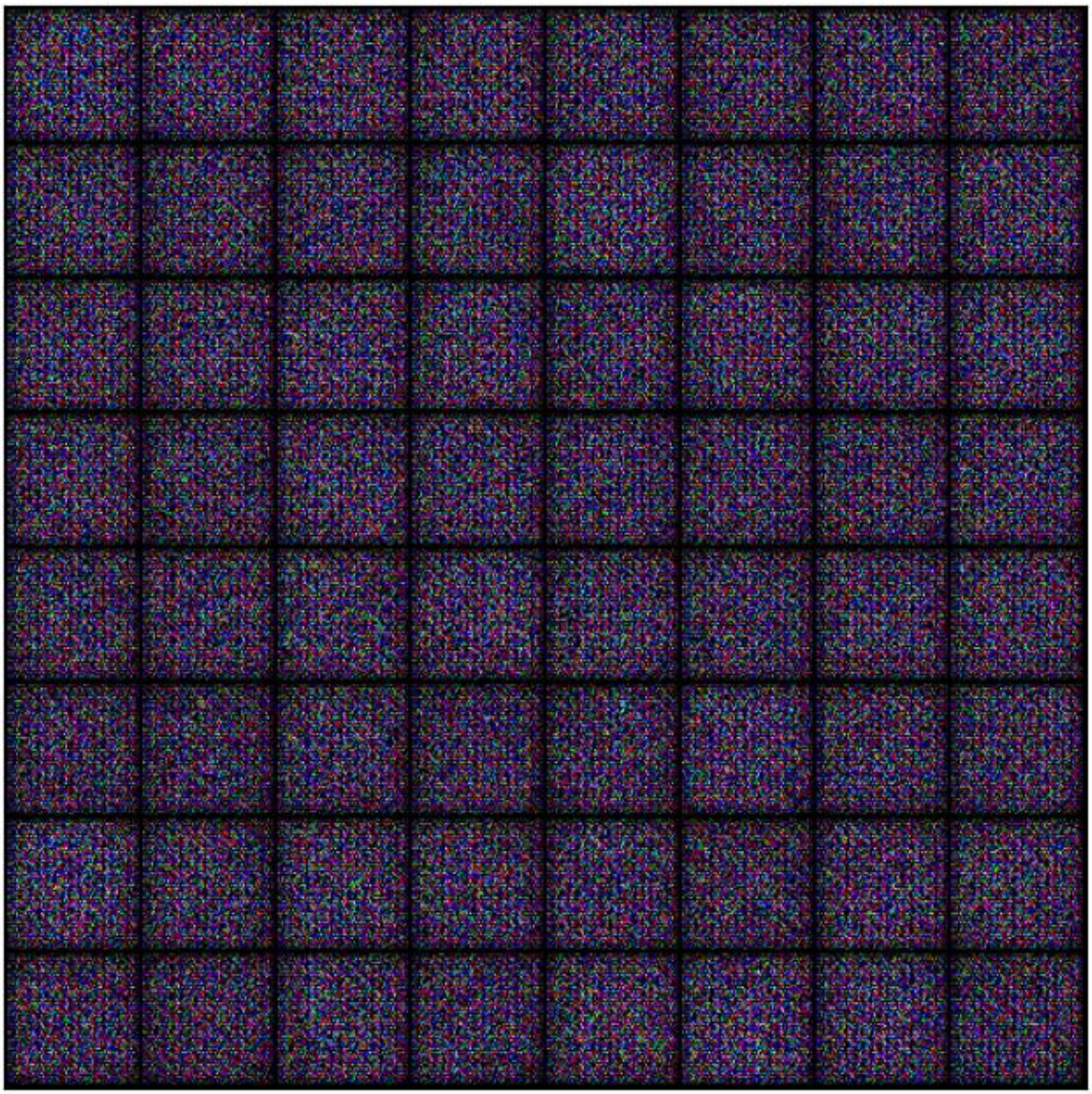
این پیکربندی لایه مستقیماً از اندازه ورودی (16*16) به اندازه خروجی (64*64) در یک مرحله، بزرگ‌نمایی می‌کند و تبدیل معادل با دو لایه کانولوشن معکوس جداگانه را انجام می‌دهد.

پس از اعمال این تغییرات و انجام آموزش، نتایج زیر به دست می‌آیند:

عکس های دیتاست به این صورت هستند.

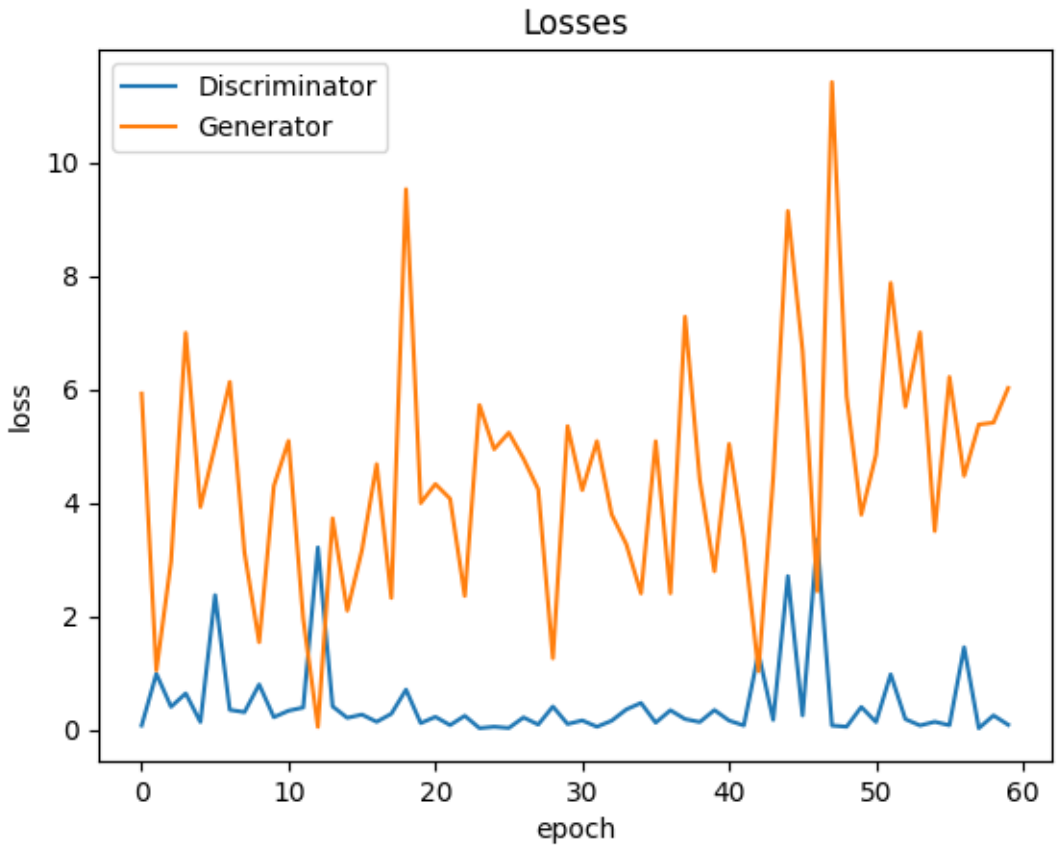
Generator بدون کمک Discriminator تنها می تواند عکس های نامفهوم و نویزمانند درست کند.



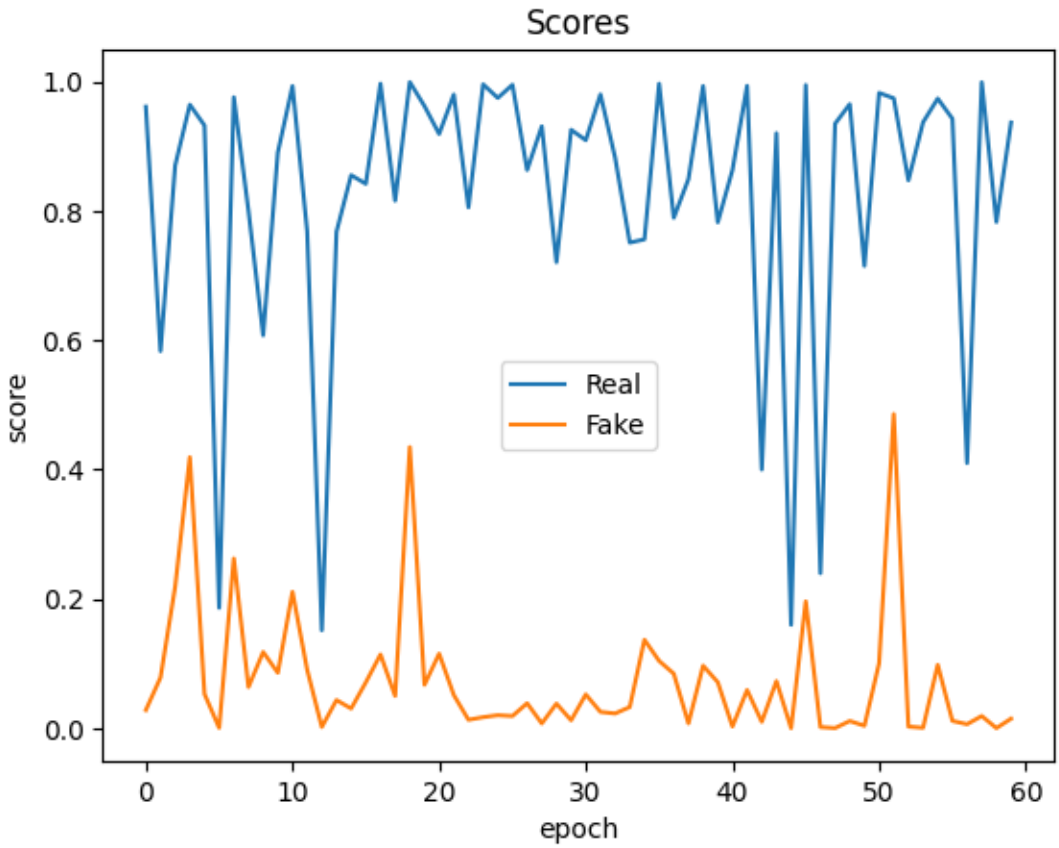


پس از آن که Generator و Discriminator به رقابت پرداختند، حال Generator قادر است عکس هایی مشابه دیتاست تولید کند که Discriminator نتواند متوجه fake بودن آن ها بشود.

نمودار Loss ها:



نمودار Score ها:



نتیجه نهایی:

از مقایسه عکس های تولید شده و نمودار های Losses و Scores پیش از حذف لایه کانولوشن آخر و پس از آن درمی‌یابیم که عملکرد DCGAN پس از حذف لایه کانولوشن آخر به وضوح بهتر شده است و مولد می تواند تصاویر نزدیک تری به دنیای واقعی تولید کند به طوری که متمایزکننده متوجه ساختگی بودن آن ها نشود.