



به نام خدا



دانشگاه تهران
دانشکده مهندسی برق و کامپیوتر
درس شبکه‌های عصبی و یادگیری عمیق

پروژه شماره 3

محسن فیاض - محمد رضا عظیمی	نام و نام خانوادگی
810100521 - 810100524	شماره دانشجویی
1401 17 تیر	تاریخ ارسال گزارش

فهرست گزارش سوالات

1

سوال 1 - SGAN

1

1) توضیحی مختصر از نحوه ی آموزش شبکه ارائه دهید.

2) با توجه به جزئیات موجود در مقاله شبکه Semi Supervised GAN را پیاده سازی و با داده ی MNIST آن را آموزش دهید. برای آموزش شبکه تنها 100 نمونه را برچسب دار و بقیه را بدون برچسب در نظر بگیرید.

3) بخش کلاسیفایر سوال 2 را بدون کمک GAN و به ازای 100 نمونه ی برچسب دار به صورت Supervised آموزش دهید و نتایج این 4 دو (سوال 2 و 3) را همانند جدول 1 مقاله گزارش کنید، یعنی دقت طبقه بند را گزارش کنید.

4) به جای generator در سوال دوم از VAE Variational autoencoder استفاده کنید. برای آپدیت پارامترهای VAE، خطا را مجموع سه مؤلفه ی خطای GAN، خطای بازسازی و خطای KL divergence در نظر بگیرید و با همان تنتیمیات بخش 2 شبکه را آموزش دهید و نتیجه را به جدول بخش 3 اضافه کنید. (توجه کنید که VAE به جای کل GAN استفاده نمی شود و فقط جایگزین generator می شود). شکل 1 مقاله را برای بخش 2 و 4 به ازای یک اپیک دلخواه رسم کنید.

10

سوال 2 - DCGAN

الف) مقاله ی مربوط به DCGAN را این لینک دانلود و مطالعه کنید. خلاصه ای از طرز کار این شبکه را بنویسید.

ب) با توجه به مقاله ذکر شده شبکه را پیاده سازی و به کمک دیتاست گفته شده آموزش دهید. توجه کنید که حتما باید از Data Loader برای فرستادن تصاویر به شبکه جهت آموزش استفاده کنید. ساختار شبکه ی کانولوشنی مورد استفاده و نتایج را گزارش کنید.

14 شکل 3 - نمودار لاس

ج) سه مورد از مشکلاتی که در پیاده سازی شبکه های GAN رایج است، عدم همگرایی، محو شدن گرادیان و mode collapse می باشد. در مورد هر کدام به اختصار توضیح دهید و برای حل این مشکلات راه حل هایی ارائه دهید، آنها را پیاده سازی و نتایج قبل و بعد را مقایسه کنید.

19

سوال 4 - Cycle GAN

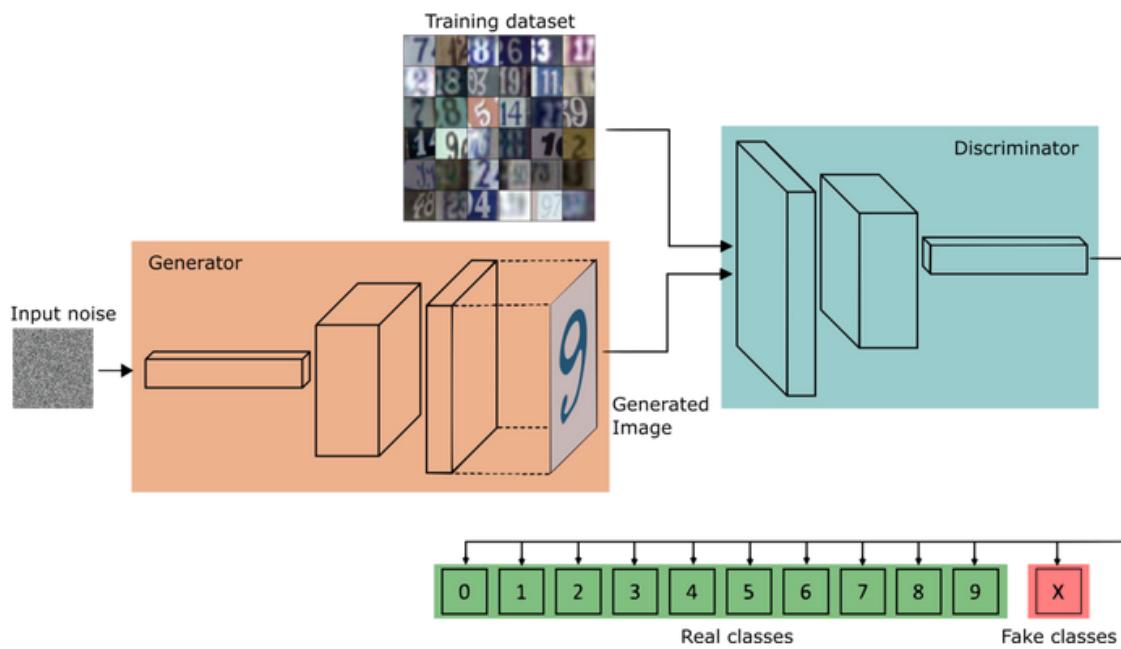
30

منابع

سوال 1 - SGAN

(1) توضیحی مختصر از نحوه آموزش شبکه ارائه دهید.

شبکه Semi-Supervised GAN سعی می کند با دیتای اندکی که لیبل دارد بهبود ایجاد کند.



شکل 1 - معماری SGAN

به این ترتیب که علاوه بر خروجی discriminator عادی در GAN که مشخص می کند ورودی واقعی یا ساختگی بوده است، کلاس آن را نیز خروجی می دهد. مثلا در mnist باید 10 خروجی بیشتر داشته باشد (که softmax زده می شود) و مشخص می کند در کدام کلاس است. به این ترتیب با استفاده از داده اندک که لیبل دارد این بخش را نیز علاوه بر real/fake آموزش می دهد. ادعای مقاله این است که این شکل آموزش discriminator می تواند آن را به یک classifier قدرتمندتر از اینکه به صورت مستقیم مدل کلسیفایر آموزش داده شود تبدیل کند که در این سوال این ادعا را بررسی می کنیم.

(2) با توجه به جزئیات موجود در مقاله شبکه Semi Supervised GAN، اپیاده سازی و با داده‌ی MNIST آن را آموزش دهید. برای آموزش شبکه تنها 100 نمونه را برچسب دار و بقیه را بدون برچسب در نظر بگیرید.

ابتدا با استفاده از کلاس DataLoader و datasets از پایتورچ دیتاست MNIST را بارگذاری کردیم و transform های لازم مانند تغییر اندازه و تبدیل به تنسور را انجام دادیم.

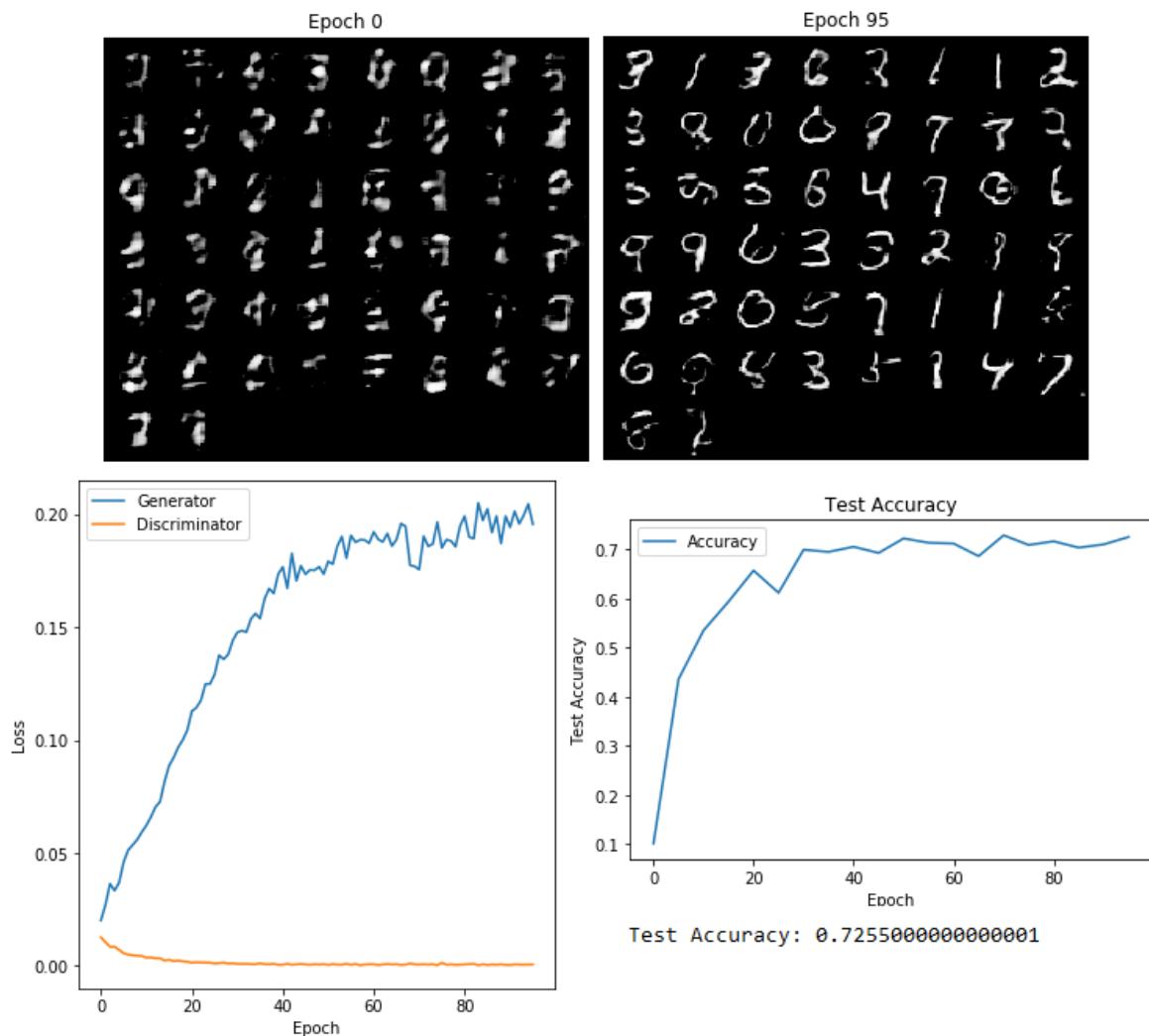
برای پیاده سازی SGAN ابتدا بخش generator را مانند یک GAN عادی پیاده سازی کردیم تا یک وکتور ورودی به عنوان نویز z بگیرد و آن را تبدیل به تصویر کند. اما برای بخش discriminator همانطور که در بخش قبل هم گفته شد یک خروجی برای real/fake در نظر گرفتیم و همچنین خروجیهایی برای کلاس بندی اعداد که اولی باتابع فعالساز sigmoid و دومی چون چندین کلاس است باتابع فعالساز softmax پیاده شد (логیست گرفته و کل خروجی را تبدیل به توزیع احتمالی می‌کند به طوری که مجموع 1 شود و این برای محاسبه loss هایی که در ادامه محاسبه می‌شوند اهمیت دارد که خروجی حتماً توزیع احتمالی باشد). باقی ساختار مانند GAN عادی بوده و با لایه‌های convolution تصویر را از عمق فیچر کم و سطح بزرگ به سطح کم و فیچر بزرگ بردیم و در انتها همان خروجی‌ها که توضیح داده شد را به دست آوردهیم.

برای آموزش، باز هم بخش generator تفاوت خاصی با GAN عادی نداشت و صرفاً باید لاس مربوط به آن که هدفش نزدیکی نتیجه discriminator روی خروجی‌های generator به real است محاسبه می‌شد و برای این کار از BCELoss استفاده کردیم.

اما برای آموزش discriminator دو بخش داریم. یک بخش تصاویر حقیقی را گرفته و انتظار داریم real دهد و دیگری خروجی generator را می‌گیرد و باید fake دهد. در این بخشها تفاوت در این است که علاوه بر loss مربوط به خروجی، جاهایی که label داریم، انتظار داریم discriminator کلاس درست عدد را نیز تشخیص دهد که آن را با CrossEntropyLoss بین خروجی و لیبل درست می‌سنجدیم. (این لاس براساس information theory است) در آخر لاس نهایی که discriminator براساس آن آموزش می‌بیند میانگین این دو است.

همانطور که خواسته شده بود لیبل ها را نیز به 100 نمونه محدود کردیم که چون بچ سایز را 50 گرفتیم می‌شد دو step اول. و در ادامه آن مدل به صورت عادی GAN آموزش می‌بیند و هدف اصلی تشخیص real/fake خواهد بود.

پس از آموزش مدل به شیوه‌ی توضیح داده شده، پیشرفت خروجی generator و همچنین پیشرفت دقت discriminator روی بخش تست mnist در زیر آمده است.



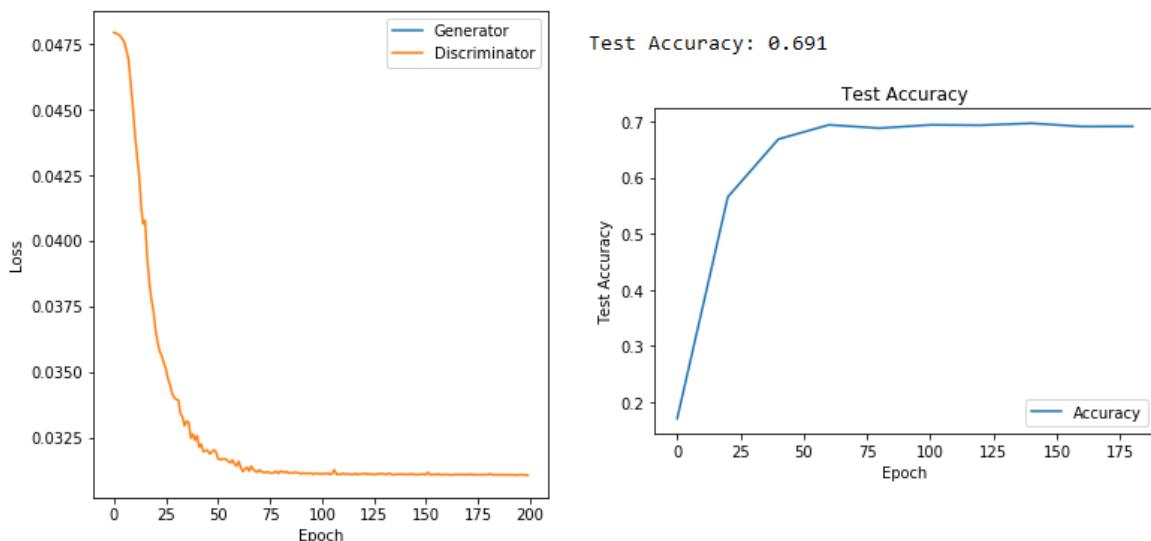
شکل 2 - پیشرفت SGAN، نمودار لاس و نمودار دقت روی بخش تست با آموزش روی فقط 100 نمونه

همانطور که دیده می‌شود مدل به خوبی آموزش دیده است و پیشرفت generator در تولید و همچنین افزایش دقت discriminator روی کلاس بندی بخش تست mnist مشهود است. باید اشاره کرد که این نتیجه تنها با استفاده از 100 نمونه لیبل زده شده به دست آمده است که خود جالب توجه است. ضمناً همانطور که دیده می‌شود هنوز جای پیشرفت هم وجود داشت، اما به علت محدودیت منابع، تست‌ها را به 100 ایپاک محدود کردیم و گرنه دقت‌های پیشتر هم می‌توانستیم بگیریم.

(3) بخش کلاسیفایر سوال 2، ا بدون کمک GAN و به ازای 100 نمونه ی برچسب دار، به صورت Supervised آموزش دهید و نتایج این دو (سوال 392)، همانند جدول 1 مقاله گزارش کنید، یعنی دقت طبقه بند، اگزارش کنید.

در این بخش فقط قسمت discriminator را که خروجی 10 کلاسه هم داشت آموزش می‌دهیم. به این ترتیب که loss مدل که بر اساس آن آموزش می‌بیند را محدود به CrossEntropyLoss بین خروجی و لیل‌ها می‌کنیم و ضمناً دیگر فقط روی همان 100 نمونه کافیست آموزش دهیم و مثل بخش قبلی نیازی نیست تا روی دیتای بدون لیل هم آموزشی داشته باشیم.

نتیجه اجرا در زیر آمده است.



شکل 3 - نمودار لاس و دقت روی تست برای آموزش فقط بخش discriminator به شکل یک کلاسیفایر

کاهش loss و افزایش دقت را شاهد هستیم. به این ترتیب آموزش این بخش discriminator به تنها به شکلی که تنها طبقه‌بندی mnist را انجام دهد موفقیت آمیز بود و به دلیل سبکی اجرا خیلی سریعتر از بخش‌های قبلی انجام شد.

جدول خواسته شده در زیر آمده است.

جدول 1 - تفاوت SGAN و آموزش کلاسیفایر

EXAMPLES	CNN	SGAN
100	69.1	72.5

همانطور که دیده می شود، دقت این روش کمتر از SGAN شد که در مقاله همین ادعا شده بود و موفق به بازتولید این نتیجه شدیم. علت این موضوع را می توان در این دانست که SGAN با اینکه داده لیل خورده کمی می بیند اما در موافقی که لیلی نیست، باز هم باید تشخیص واقعی یا fake بودن بدهد و به طور unsupervised در حال یادگیری در مورد دیتاست و اعداد است. همین یادگیری غیر مستقیم می تواند در کنار آموزش با لیل که supervised است یک مکانیزم آموزش semi-supervised را فراهم کند که دیدیم تاثیر بهتری نسبت به آموزش supervised به Semi-Supervised Learning with Generative Networks تنها دارد. نام این روش یعنی Adversarial Networks نیز از همین جهت گذاشته شده است.

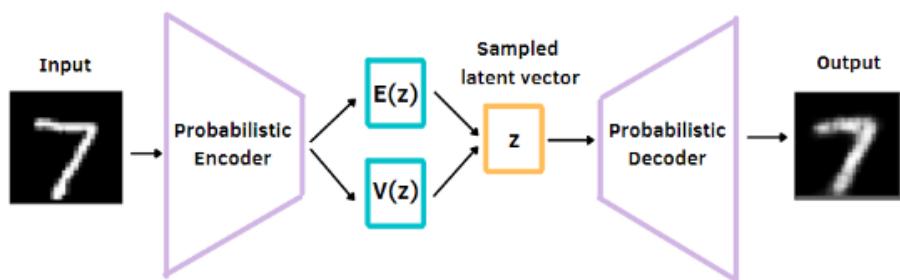
مشابه این موضوع در فیلدهای دیگر مثل NLP هم دیده شده که آموزش حتی بدون label در یک زمینه خاص می تواند نتیجه نهایی طبقه بندی برای آن دومین را تا حد خوبی افزایش دهد.

<https://aclanthology.org/2020.acl-main.740/>

در مجموع آشنایی بیشتر مدل با دیتاها مشابه چیزی که باید طبقه بندی کند، حتی بدون لیل، تاثیر مثبتی دارد.

(4) به جای **generator** دو، سوال دوم از **VAE Variational autoencoder** استفاده کنید. برای آپدیت پارامترهای VAE، خطا را مجموع سه مولفه‌ی خطای GAN، خطای بازسازی و خطای KL divergence در نظر بگیرید و با همان تنظیمات بخش 2 شبکه را آموزش دهید و نتیجه را به جدول بخش 3 اضافه کنید. (توجه کنید که VAE به جای کل استفاده نمی‌شود و فقط جایگزین **generator** می‌شود). شکل 1 مقاله را برای GAN بخش 2 و 4 به ازای یک ایپاک دلخواه، سم کنید.

ابتدا شمای کلی VAE‌ها را در زیر می‌بینیم.



شکل 4 - ساختار VAE

این روش از دو بخش encoder و decoder تشکیل شده است که ابتدا تصویر ورودی را گرفته و به یک فضای latent با ابعاد کوچک می‌برد و سپس decoder باید سعی کند از آن فضای کوچک، تصویر اصلی را بازتولید کند.

بخش encoder با استفاده از لایه‌های Conv2d و لایه‌های دیگر ابتدا این کوچک‌سازی را انجام می‌دهد و سپس decoder با استفاده از ConvTranspose2d و لایه‌های دیگر بازگردانی را انجام می‌دهد.

اما مهمترین بخش در اینجا اتصال این مدل به SGAN و آموزش آن است. برای این کار generator قبلی را به کلی کنار گذاشتیم و این مدل را جایگزین کردیم. روش VAE دو لاس مهم دارد، یکی reconstruction term نامیده می‌شود که خطای بین خروجی نهایی و ورودی اولیه را می‌سنجد که می‌توان از روش‌های مختلفی برای محاسبه آن بهره برد مثل MSE.

و بخش دوم Kullback-Leibler divergence regularization term بین توزیع فضای latent و توزیع نرمال استاندارد می‌گیرد. KL Divergence (که همان cross-entropy entropy منهای تفاوت دو توزیع را اندازه می‌گیرد. به این ترتیب

وقتی آن را به عنوان لاس فرار دهیم یعنی دوست داریم توزیع فضای latent یعنی خروجی encoder به نرمال استاندارد شبیه باشد و gradient descent در جهت کاهش آن باعث همین

$$l(x, \hat{x}) = l_{reconstruction} + KL(z, N(0, I_d))$$

موضوع می شود.

برای اضافه کردن آن به مدل چون از پایتورچ استفاده کردیم مشکل خاصی نداریم. کافی است تا این لاس ها را به لاس قبلی generator اضافه کنیم. یعنی

```
g_loss = adversarial_loss(validity, valid)
```

```
g_loss += ((gen_imgs - imgs)**2).sum() + generator_vae.encoder.kl
```

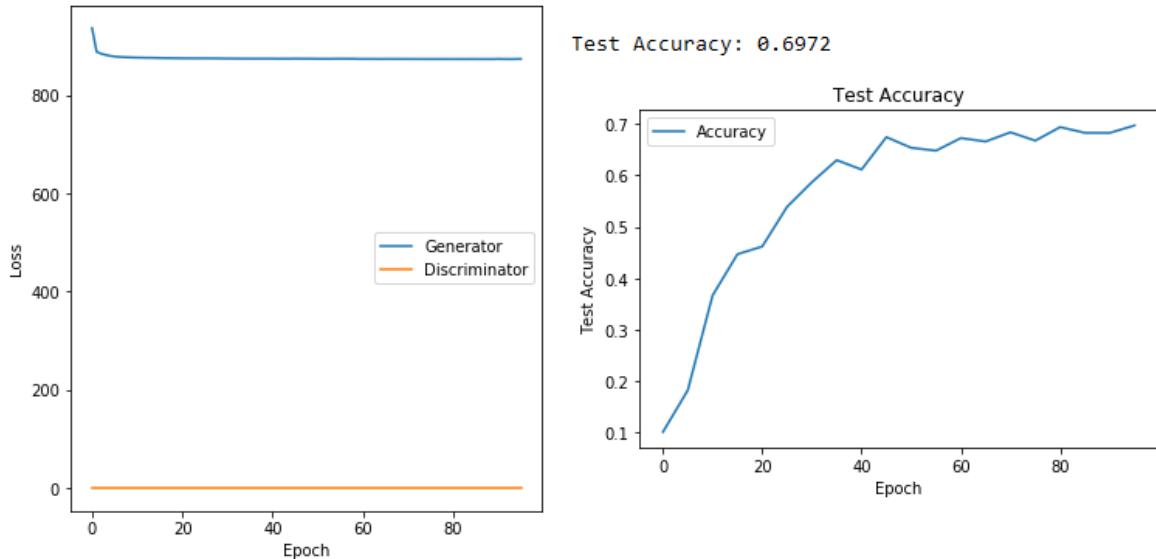
به این ترتیب 3 لاس را ترکیب کردیم. اولی لاس GAN است که مربوط به خروجی discriminator است که دوست داریم آن را به اشتباه بیاندازیم و همه را real بدهد.

دومی لاس بازسازی vae است که از MSE استفاده کردیم و توان دوی تفاوت تصویر ورودی و خروجی را می سنجیم.

و سومی خطای KL Divergence که سعی بر نرمال کردن فضای پنهان که خروجی انکورد است داریم.

و در ادامه کافیست optimizer_G.step و g_loss.backward فراخوانی کنیم تا گرادیانها روی گراف محاسبات به دست آورده شوند و سپس براساس اپتیمایزر، وزنها آپدیت شوند. به این ترتیب هر سه لاس را در نظر گرفتیم.

نتیجه اجرا در زیر آمده است.



شکل ۵ - نمودار لاس و دقت روی تست برای آموزش VAE به جای generator

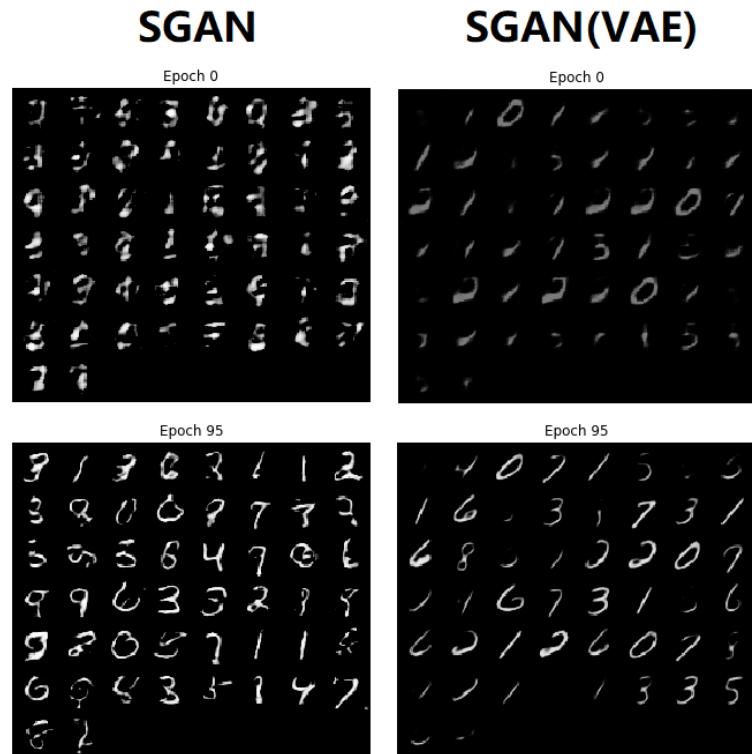
همانطور که دیده می‌شود چون لاس جنریتور مجموع سه لاس شده است در کل افزایش مقدار داشته ولی باز هم رو به کاهش است و آموزش به خوبی انجام می‌شود چون دقت روی تست هم رو به افزایش است. دقت در مقایسه با قبل در جدول زیر آمده است.

جدول ۲ - تفاوت SGAN و آموزش کلسیفایر و VAE به جای generator

EXAMPLES	CNN	SGAN	VAE Generator
100	69.1	72.5	69.7

انتظار خاصی نداشتیم اما VAE توانست بین CNN و SGAN قرار گیرد. به این ترتیب که این مدل هم مثل SGAN از آموزش semi-supervised بهره می‌برد و دیتای بیشتری می‌بیند بنابراین می‌توان دقت بهتر نسبت به CNN را توجیه کرد. اما از طرف دیگر لزوماً بهتر از GAN ها نیست و محدودیت ها و خصیت های خاص خود را دارد که نمی‌توان گفت لزوماً باید بهتر از SGAN عادی عمل کند. ضمناً اینجا هم به علت محدودیت منابع، به حداقل 100 اپیک ایندیکاتور کردیم و گرنه قطعاً اگر ادامه می‌دادیم نتایج بهتری هم امکان حاصل شدن داشت.

در آخر برای مقایسه، مشابه شکل 1 مقاله، خروجی بخش 2 و 4 در اپاک های 0 و 95 مقایسه شدند که تفاوت تولیدات GAN با VAE مشهود است و تفاوتهایی مثل نویز در SGAN و کمنگی در VAE دیده می شود و همچنین پیشرفت هر دو بین اپاکها مشخص است.

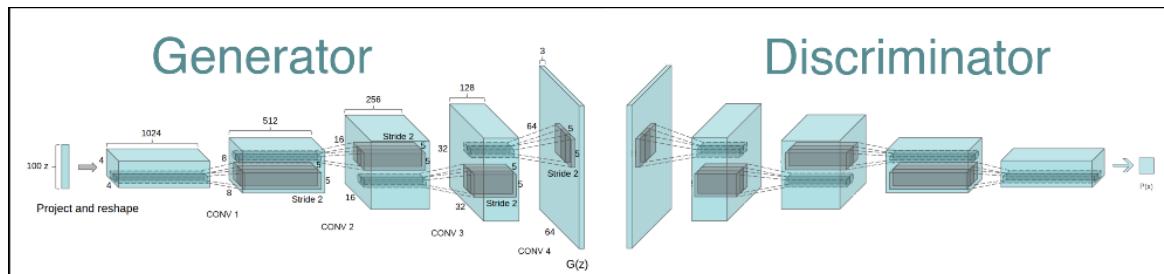


شکل 6 - بازسازی شکل 1 مقاله

سوال 2 - DCGAN

الف) مقاله‌ی مربوط به DCGAN، از این لینک دانلود و مطالعه کنید. خلاصه‌ای از طرز کار این شبکه را بنویسید.

در این شبکه در بخش discriminator و generator از شبکه‌های عصبی convolutional استفاده شده و به همین دلیل نام آن deep convolutional GAN یا DCGAN است. گذاشته شده است.



شکل 1 - معماری بخش discriminator و generator

همانطور که در تصویر هم مشخص است برای بخش generator از لایه‌های conv-transpose استفاده می‌شود (که به اشتباہ deconvolution هم گفته می‌شوند). ابتدا یک بردار نویز 100 بعدی داریم که سپس آن را به شکل کانولوشنی با ابعاد مشخص می‌بریم و پس از اعمال چندین لایه، به تصویر 64×64 با 3 کanal که همان رنگ‌ها هستند می‌رسیم. این تولیدات بخش داده‌های fake را تشکیل می‌دهند. در ادامه به یک شبکه عمیق CNN این تصاویر داده می‌شود و مدل discriminator باید تشخیص دهد که تصویر fake است یعنی از generator آمده است یا داده واقعی است و دیتاست آمده است. به این ترتیب در این تقابل، دو مدل آموزش می‌بینند.

ب) با توجه به مقاله‌ی ذکر شده شبکه را پیاده سازی و به کمک دیتاست گفته شده آموزش دهید. توجه کنید که حتما باید از Data Loader برای فرستادن تصاویر به شبکه جهت آموزش استفاده کنید. ساختار شبکه‌ی کانولوشنی مورد استفاده و نتایج، اگزارش کنید.

همانطور که خواسته شده برای داده‌ها از Data Loader بهره بردیم و آدرس فایلها و همچنین تابعی برای اعمال پیش پردازش را به آن دادیم تا کارهایی مانند resize کردن تصاویر ورودی به 64 و نرمال کردن انجام شود. تمام جزئیات در کدی که کنار این فایل است آمده است.

ساختار شبکه جنریتور به شکل زیر است.

Generator(
(cnn): Sequential(
 (0): ConvTranspose2d(100, 512, kernel_size=(4, 4), stride=(1, 1), bias=False)
 (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
 (2): ReLU(inplace=True)
 (3): ConvTranspose2d(512, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
 (4): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
 (5): ReLU(inplace=True)
 (6): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
 (7): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
 (8): ReLU(inplace=True)
 (9): ConvTranspose2d(128, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
 (10): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
 (11): ReLU(inplace=True)
 (12): ConvTranspose2d(64, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
 (13): Tanh()
)
)

و معماری بخش discriminator به شکل زیر است.

Discriminator(
(main): Sequential(
 (0): Conv2d(3, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
 (1): LeakyReLU(negative_slope=0.2, inplace=True)
 (2): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
 (3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)

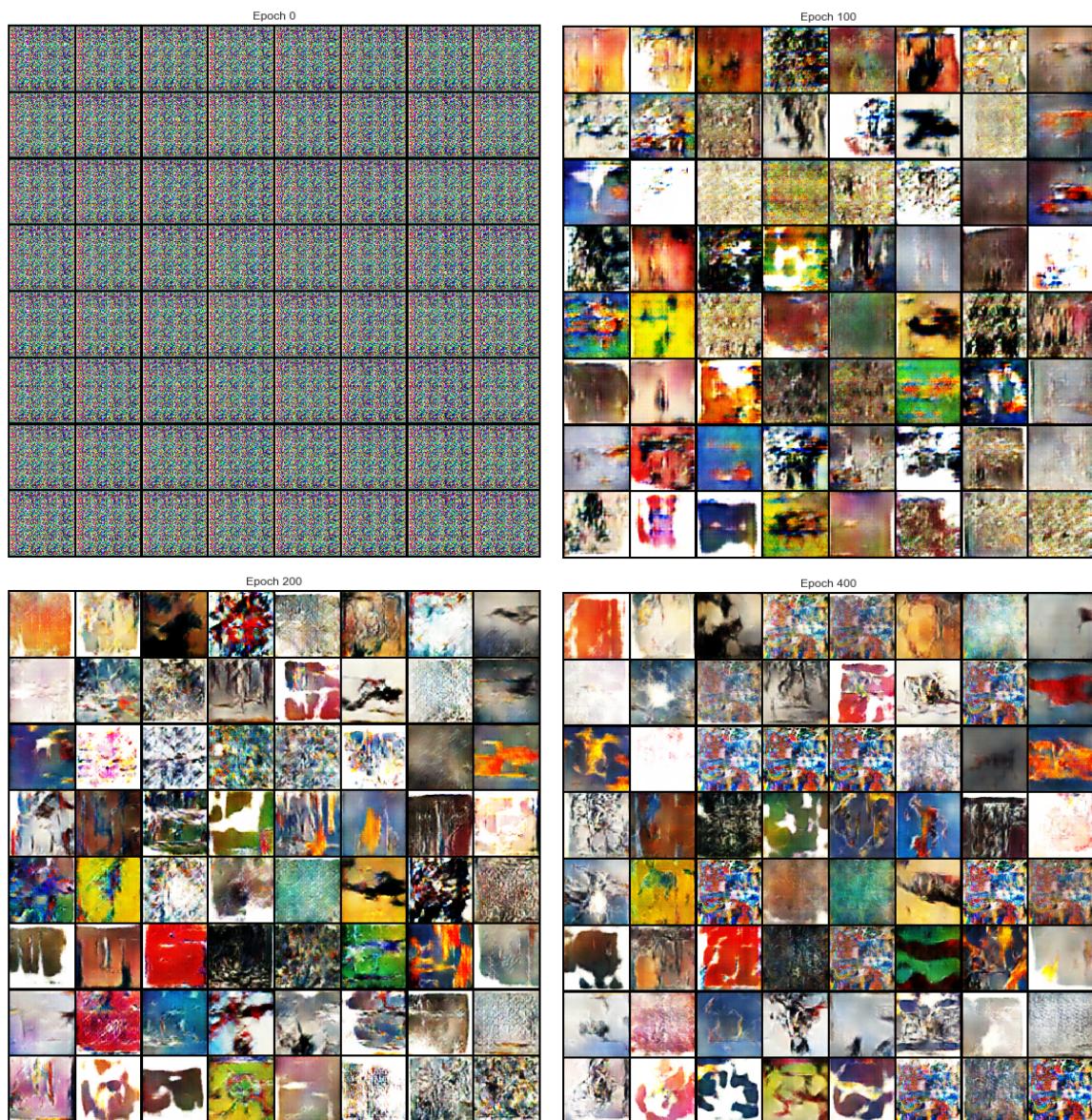
```
(4): LeakyReLU(negative_slope=0.2, inplace=True)
(5): Conv2d(128, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
(6): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(7): LeakyReLU(negative_slope=0.2, inplace=True)
(8): Conv2d(256, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
(9): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(10): LeakyReLU(negative_slope=0.2, inplace=True)
(11): Conv2d(512, 1, kernel_size=(4, 4), stride=(1, 1), bias=False)
(12): Sigmoid()
```

)

)

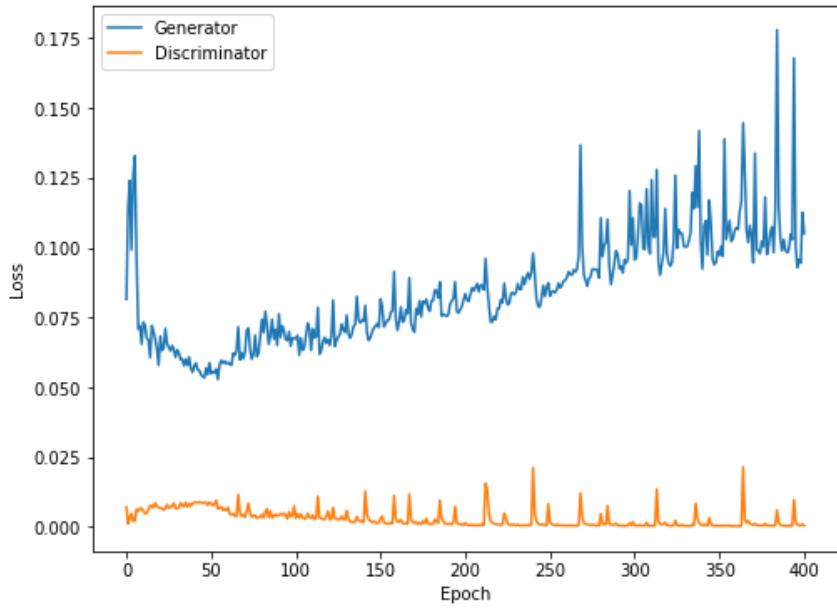
در ادامه از cross entropy به عنوان لاس و از adam به عنوان اپتیمایزر دو مدل استفاده کردیم. ابتدا discriminator را با یک سری داده real و یک سری داده که از generator به عنوان fake می‌گیریم اجرا کرده و لاس را حساب کرده و یک step حرکت می‌دهیم و سپس آموزش discriminator را بر اساس خروجی generator می‌دهیم تا هر دو در تقابل باشند. این کار را برای تعداد ایپاک بسیار تکرار می‌کنیم تا نتایج مناسبی ببینیم.

نتایج ایپاک های مختلف در زیر آمده است.



شکل 2 - پیشرفت در طول آموزش

همانطور که دیده می شود ابتدا مدل جنریتور از نویز به نویز رسیده، اما با پیشرفت در طول آموزش، کم کم یاد می گیرد تا تصاییر دقیق تری بسازد که discriminator را به اشتباه بیاندازد. نمودار لاس نیز در زیر آمده است.



شکل 3 - نمودار لاس

و دیده می شود که هم generator و هم discriminator به خوبی آموزش می بینند و لاستان کاهش می باید از طرفی discriminator بیشتر چون کار نسبتا ساده تری دارد و هر چه بهتر شود طبیعتا کار generator سخت تر می شود و به همین دلیل در کل افزایش دارد اما در هر مقطع سعی بر کاهش داشته. البته که در GAN ها این مشکل converge نشدن از مشکلات معروف است و همگرایی در تقابل بسیار سخت است، البته که در نتایج نهایی قبلی که دیدیم مشخص بود مدل به خوبی آموزش دیده است و انتظاری که داشتیم را توانسته براورده کند.

ج) سه مورد از مشکلاتی که در پیاده سازی شبکه های GAN رایج است، عدم همگرایی، محو شدن گرادیان و mode collapse می باشد. در مورد هر کدام به اختصار توضیح دهید و برای حل این مشکلات راه حل هایی ارائه دهید، آنها را پیاده سازی و نتایج قبل و بعد را مقایسه کنید.

1. مشکل عدم همگرایی یا Non-convergence

این مشکل وقتی پیش می آید که پارامترهای مدل نوسان می کنند، بی ثبات می شوند و هرگز همگرا نمی شوند.

راه حل های این مشکل شامل اضافه کردن نویز به ورودی discriminator یا Penalizing discriminator weights است. همانطور که دیده می شود این کار ها در جهت regularization است تا همگرایی تسهیل شود.

2. مشکل محو شدن گرادیان یا Diminished gradient

تحقیقات نشان داده است که اگر discriminator خیلی خوب باشد، آموزش generator به discriminator با شکست مواجه می شود. در واقع، یک discriminator بهینه که هر چیزی را به شکل کامل تشخیص می دهد واقعی یا fake است اطلاعات کافی برای پیشرفت generator ارائه نمی کند. مثلاً اگر فقط 0 و 1 بدهد مدل هیچ وقت نمی فهمد در چه جهتی باید خود را تغییر دهد تا بهبود حاصل شود چون باز هم با اطمینان کامل fake تشخیص داده می شود.

برای حل این مشکل می توان از Wasserstein loss استفاده کرد که برای همین منظور طراحی شده است. یا از Modified minimax loss استفاده کرد که آن هم این موضوع را بهبود می بخشد.

3. مشکل Mode collapse

از GAN انتظار داریم مثلاً وقتی می خواهیم اعداد 0 تا 9 را تولید کند از همه موارد آن تولید کند. با این حال، یک generator ممکن است یاد بگیرد که فقط یک خروجی خاص را تولید کند. در واقع، generator همیشه در تلاش است تا خروجی ای را بیابد که برای discriminator، محتمل ترین به نظر می رسد. اگر generator بارها و بارها شروع به تولید همان خروجی کند، بهترین استراتژی discriminator این است که یاد بگیرد همیشه آن خروجی را رد

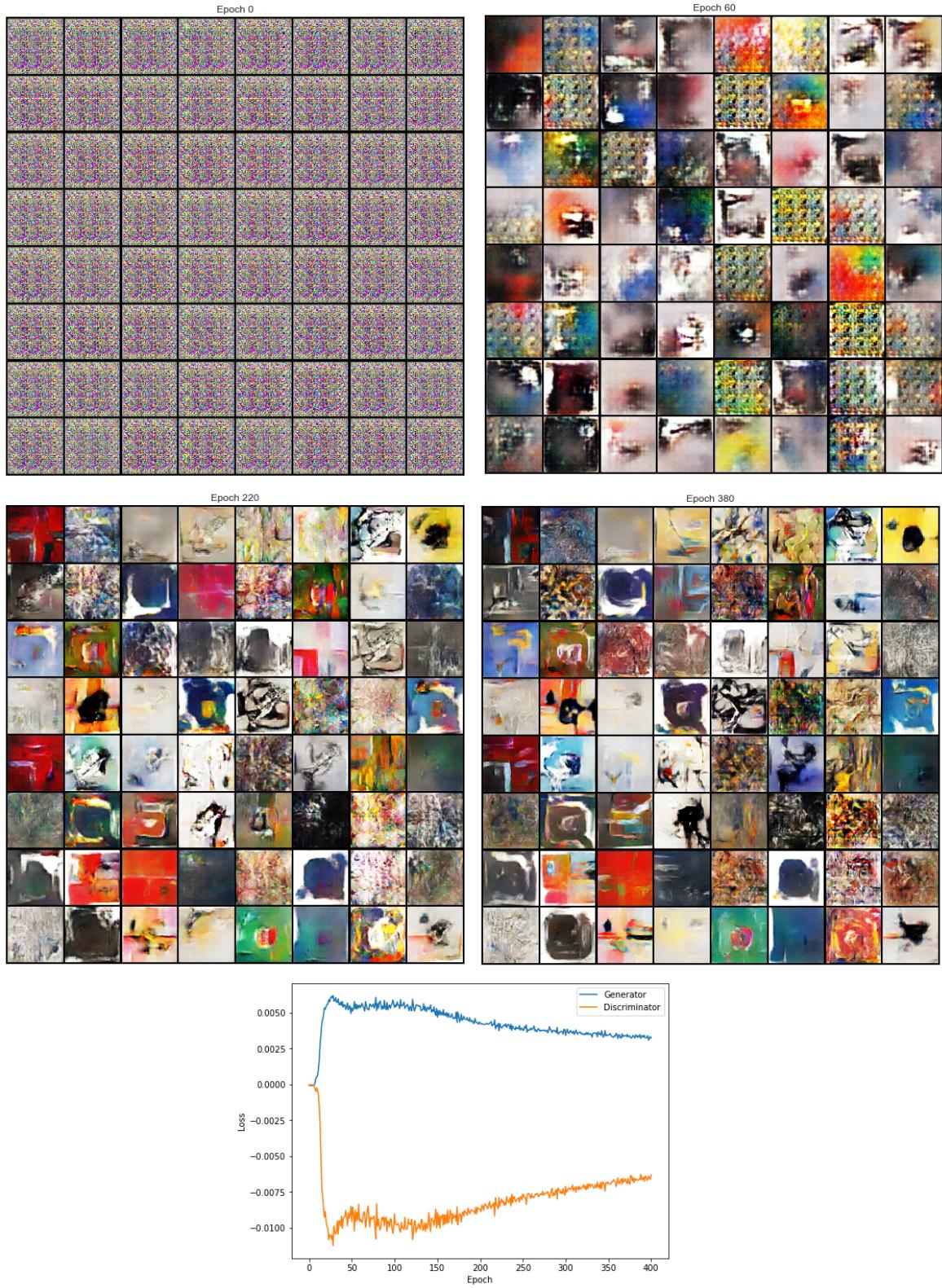
کند. اما با این کار generator در مرحله بعدی کلاس دیگری را انتخاب می‌کند. به این ترتیب هر بار generator برای یک discriminator خاص overfit می‌شود و همین حلقه ادامه می‌یابد. در نتیجه generator از طریق مجموعه کوچکی از انواع خروجی می‌چرخدن. این شکل از خرابی GAN، فروپاشی حالت یا mode collapse نامیده می‌شود.

راه حل‌های این مشکل یکی استفاده از از لاس Wasserstein است. این لاس امکان می‌دهد discriminator را بدون نگرانی آموزش دهیم و یاد می‌گیرد که خروجی هایی را که روی آنها تمرکز می‌کند را رد کند. بنابراین این مشکل برطرف می‌شود.

همچنین راه دیگر استفاده از Unrolled GAN‌ها است. این مدل‌ها از یکتابع لاس برای generator استفاده می‌کنند که نه تنها طبقه‌بندی‌های discriminator فعلی، بلکه خروجی‌های نسخه‌های discriminator آینده را نیز در نظر می‌گیرد. بنابراین generator نمی‌تواند برای یک discriminator بیش از حد بهینه‌سازی کند.

بنابر توضیحات بالا تغییراتی در کد دادیم و در فایلهای ارسالی در کنار این گزارش آمده است و به طور خلاصه دیگر لاس لگاریتم ندارد و سیگموید را در انتهای discriminator اعمال نکردیم و وزنهای discriminator را Clip می‌کنیم تا مطمئن باشیم مدل پایدار است و شروط Lipschitz برقرار باشد و به جای ADAM از RMSProp استفاده می‌کنیم و نرخ یادگیری کمتر regularization $\alpha=0.00005$ استفاده می‌کنیم. ضمناً برای بهبود مشکل عدم همگرایی از optimizer weight_decay در مشکلات ذکر شده در نظر گرفته شدند.

پیشرفت مدل و لاس آن در زیر آمده است.



شکل 4 - بیان کردن مشکلات و *WGAN*

همانطور که دیده می شود توانستیم خروجی های مناسبی بگیریم و تنوع مناسبی هم وجود دارد و دچار مشکلاتی که به دنبال برطرف کردنشان بودیم نشدیم و مدل به خوبی آموزش دید. ضمنا مسیر converge شدن هم هموارتر است نسبت به حالت قبل.

بنابراین با اصلاحاتی که بر اساس راه حل ها انجام دادیم توانستیم مشکلات موجود را برطرف کنیم و در مقایسه بهتر از حالت قبلی عمل کنیم.

سوال 4 - Cycle GAN

شبکه Cycle GAN برای تبدیل تصاویر غیرجفت به کار می رود. به این معنی که اگر بخواهیم تصویری را در فضای اول به تصویری در فضای دوم ببریم، در مجموعه دادگان آموزش خود نمونه‌ی متناظرش را نداشته باشیم.

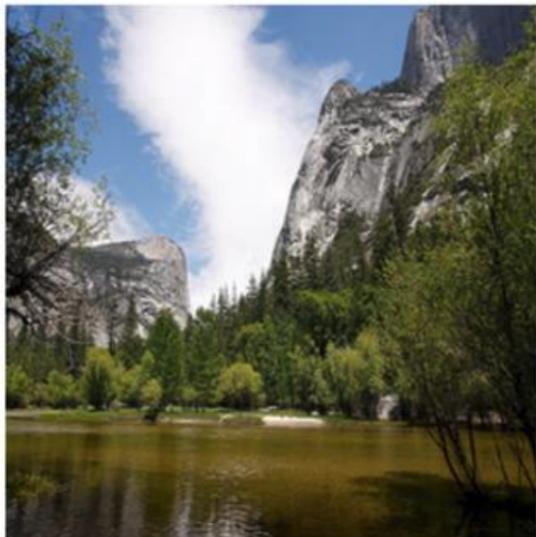
در این سوال قصد داریم تعدادی تصویر مربوط به فصل تابستان را به عکس‌هایی از فصل زمستان تبدیل کنیم، در حالی که چنین عکسی را نداریم.

این کار با Cycle GAN قابل انجام خواهد بود.

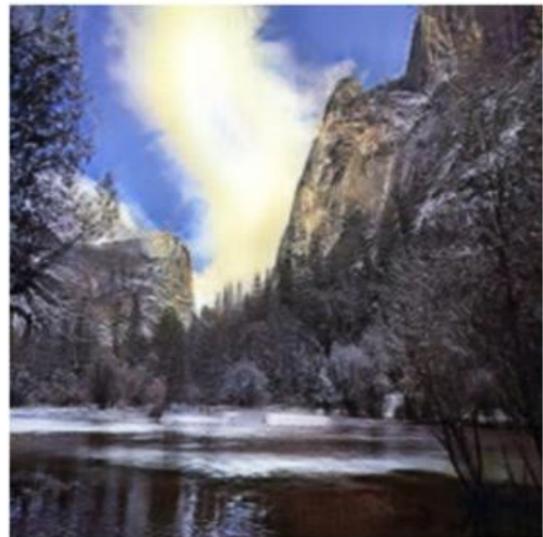
دادگان ما مجموعه summer2winter است که برای دانلود و استفاده از آن از نام summer2winter_yosemite استفاده می کنیم.

برای مثال در تصویر زیر تصویری که مربوط به فصل تابستان است، به تصویری از فصل زمستان تبدیل شده است:

Summer



Summer to Winter



شکل ۵ - نمونه‌ای از تبدیل تابستان به زمستان با استفاده از Cycle GAN

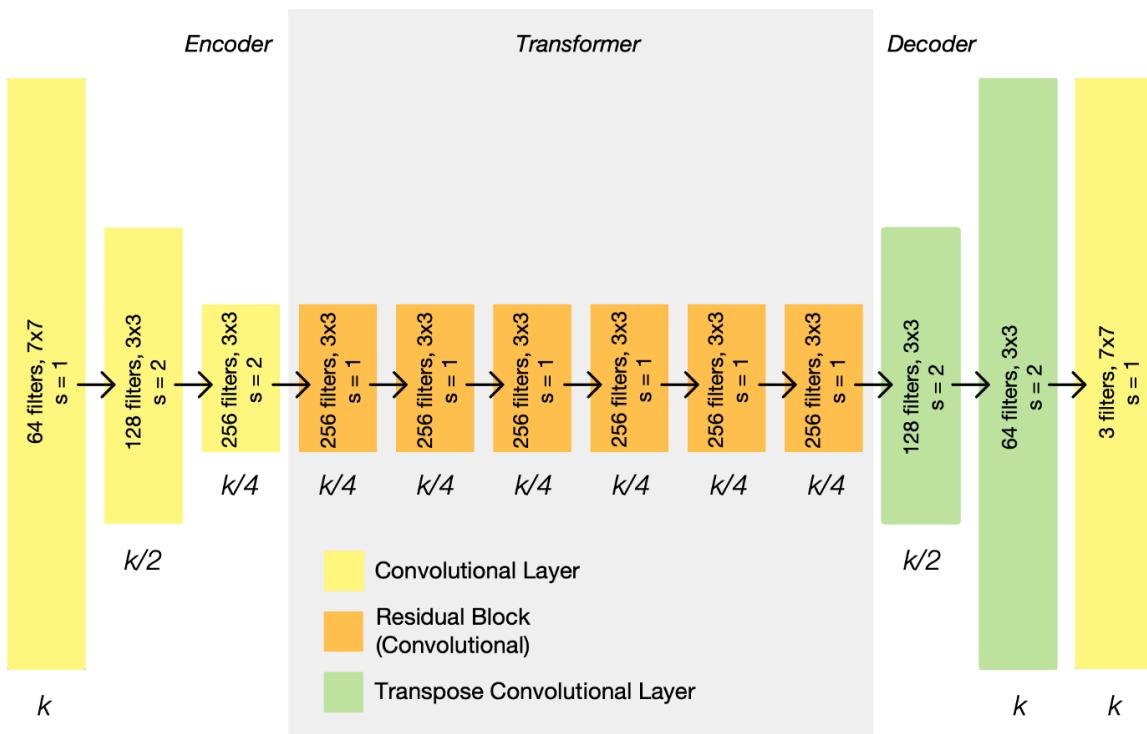
این عملیات به این صورت است که ابتدا تصویر تابستان به زمستان تبدیل شده و سپس مجدداً به تصویر تابستان باز می گردد و این عملیات باید به درستی صورت بگیرد.

Encoder، Decoder شبکه‌ی Generator از سه بخش Cycle GAN تشکیل شده است. Transformer

بخش Encoder وظیفه‌ی استخراج ویژگی‌های تصویر را دارد که از سه لایه کانولوشنی تشکیل شده است. همچنین قسمت Transformer از تعدادی Residual Block استفاده می‌کند تا با استفاده از آن‌ها ویژگی‌های بدست آمده را به گونه‌ای تغییر دهد تا به فضای جدید منتقل شود.

سپس در Decoder ساختاری کانولوشنی دارد، از ویژگی‌هایی که در بخش قبل تولید شده است استفاده می‌کنیم تا تصویر مورد نظر را به دست بیاوریم.

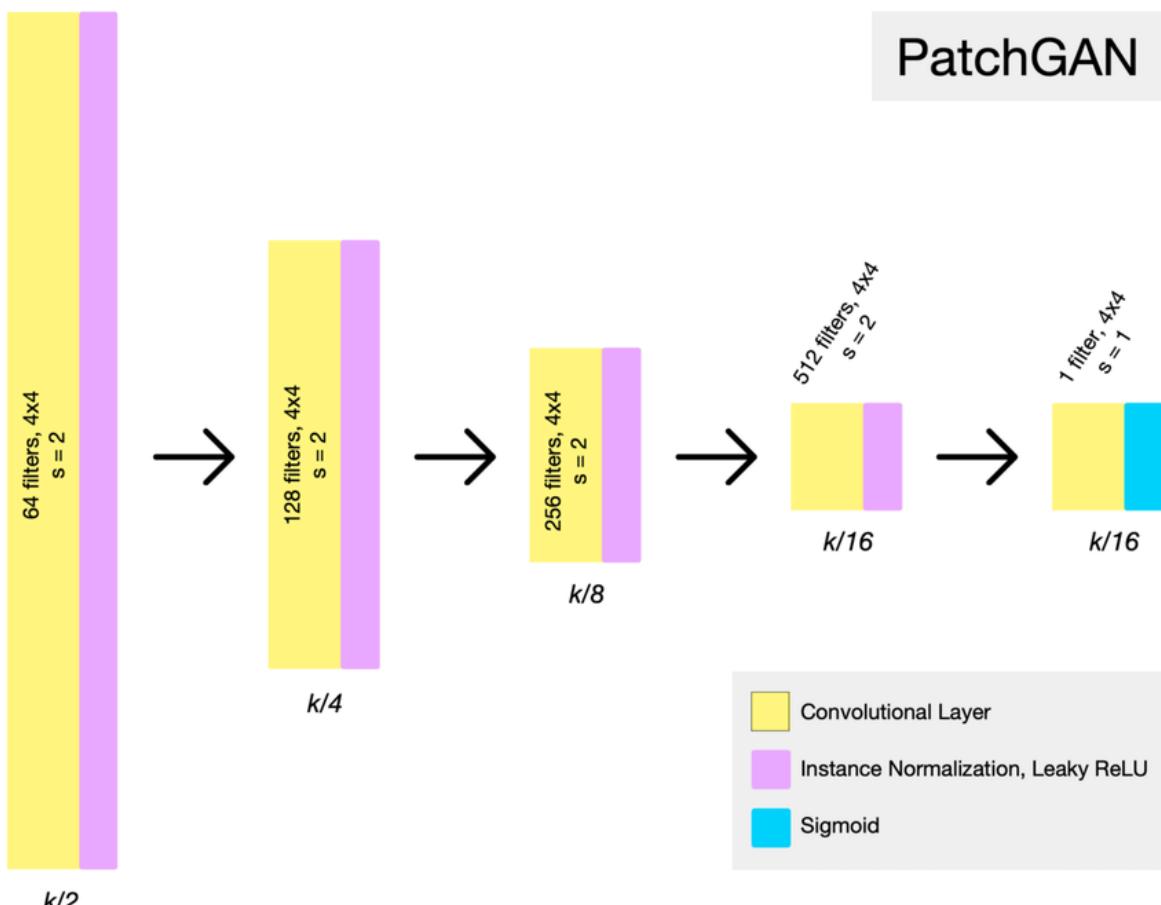
این ساختار در تصویر زیر قابل مشاهده است:



شکل ۶ - شماتیکی کلی بخش Cycle GAN Generator ساختار

همچنین این شبکه دارای قسمت‌های Discriminator می‌باشد. این بخش‌ها تلاش می‌کنند تا تشخیص دهند که آیا تصویر تولید شده واقعی است یا خیر.

PatchGAN



شکل ۷ - شمای کلی بخش Cycle GAN ساختار Discriminator

همانطور که می بینیم، در این بخش در چند مرحله ویژگی های موثر استخراج می شوند. در هر مرحله از توابع فعال ساز و نرمال سازی استفاده می شود. همچنین در لایه آخر نیز از Sigmoid برای انجام تشخیص استفاده می کنیم.

در این شبکه از دو Generator و دو Discriminator استفاده می شود. جفت اول برای تبدیل تصویر از فضای اول به فضای دوم استفاده می شود. این جفت به این صورت عمل می کنند که ابتدا Generator تصویر مورد نظر را تولید می کند و سپس Discriminator واقعی بودن تصویر تولید شده را با استفاده از تصاویر موجود در فضای دوم، تشخیص می دهد. این کار بصورت تکراری ادامه پیدا می کند و هر دو در تشخیص و تایید تصاویر آموزش می بینند.

همچنین جفت Generator و Discriminator بعدی برای تبدیل تصاویر فضای دوم به فضای اول کار می کنند. به این ترتیب گویی حلقه از این سلسله مراتب به وجود می آید و این کار بطور مداوم تکرار می شود. به همین دلیل نام Cycle GAN برای این شبکه انتخاب شده است.

خطاهای:

خطای Adversarial

این خطا که برای هر دو سمت تبدیل استفاده می‌شود، میزان واقعی بودن تصاویر تولید شده توسط Generator را ارزیابی می‌کند. در واقع این خطا مشخص می‌کند که Generator تا چه حد توانسته بخش Discriminator را در واقعی بودن تصاویر تولیدی قانع کند.

خطای Cycle Consistency

این خطا بیان می‌کند که برای مثال اگر تصویری از تابستان را به زمستان تبدیل کرده و سپس مجددآ آن را به تصویری در تابستان تبدیل کنیم، مشابه تصویر اولیه خود شود (و بالعکس). در واقع با این کار یک چرخه کامل شده و تصویر اولیه مجددآ بازسازی می‌شود.

خطای Identity

این خطا بیان می‌کند که اگر برای مثال تصویری از تابستان به Generator مربوط به تبدیل زمستان به تابستان داده شود، باید خود آن تصویر تولید شود (و بالعکس). بعبارتی باید تصویر را صرفاً متضاد نکند و سعی کند با هر ورودی فقط به خروجی مطلوب خود برسد.

(ب)

در بخش Discriminatior در شبکه‌ی Cycle GAN از ساختار Patch GAN استفاده شده است. این ساختار Discriminatior ای است که میزان واقعی بودن یا نبودن تصویر تولید شده را با یک عدد مشخص می‌کند. این عدد میزان واقعی بودن را برای هر پنجره با اندازه‌ی مشخص اندازه‌گیری کرده و معیار سنجش قرار می‌دهد. این کار مشابه عملیات فیلتر در CNN است. در نتیجه این خطا یابی با دقت بیشتری انجام می‌شود و ما تنها یک عدد نهایی به ازای هر تصویر نخواهیم داشت. بلکه به ازای هر باکس از تصویر یک مقدار برای سنجش داریم.

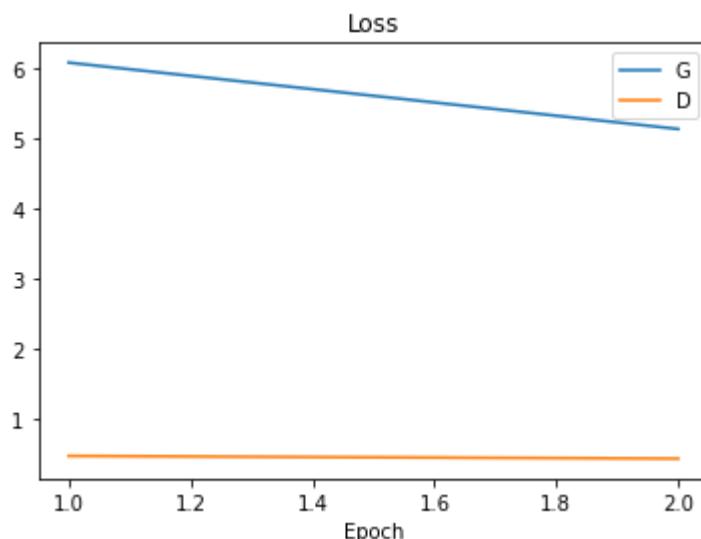
به همین دلیل بررسی ساختار Patch GAN دقیق تر است و بطور کلی عملکرد دقیق تری از Discriminstor شاهد خواهیم بود.

(ج)

برای پیاده سازی شبکه اصلی در این بخش، دو کلاس Generator و Discriminstor را طبق ساختاری که توضیح داده شد پیاده سازی می کنیم.

ابتدا دادگان summer2winter_yosemite را دریافت کرده و سپس به قسمت های آموزش و تست تقسیم می کنیم. همچنین داده های مربوط به تابستان و زمستان را نیز جدا می کنیم. از تمامی خطاهای ذکر شده در بخش الف برای آموزش این شبکه استفاده می کنیم.

نتایج بعد از ۲ اپیاک بصورت زیر است:



شکل ۸ - نمودار خطاهای Generator و Discriminator در حالت استفاده از

مقادیر این خطاهای بصورت زیر است:

خطای Generator در اپیاک اول: 6.077842940555947

خطای Generator در اپیاک دوم: 5.129397395759168

خطای Discriminator در اپیاک اول: 0.4597763809293

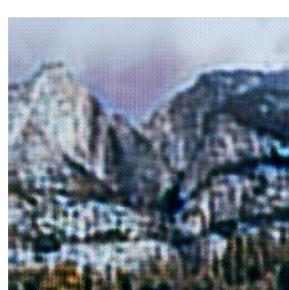
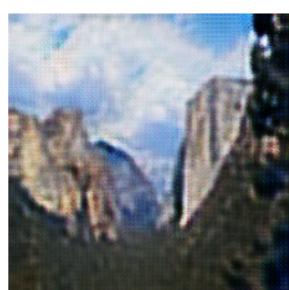
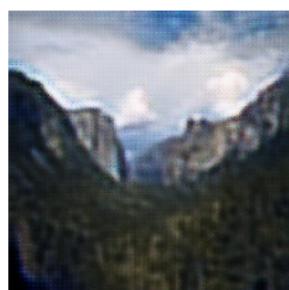
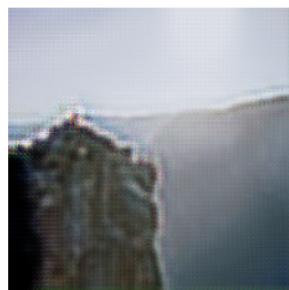
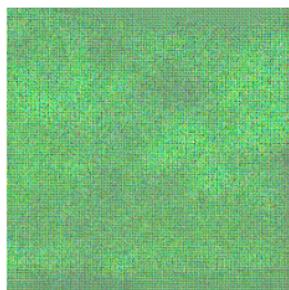
خطای Discriminator در اپیاک دوم: 0.4211327330936367

دقت ها در اپیک دوم کمتر شده اند و این نشان دهنده پیش روی خوب مرحله train است.
میزان خطای نیز مطلوب است.

در زیر نیز تعدادی از تصاویر ساخته شده در دو فصل را مشاهده می کنیم:



شکل ۹ - تصاویر تولید شدهٔ فصل تاپستان با استفاده از Discriminator



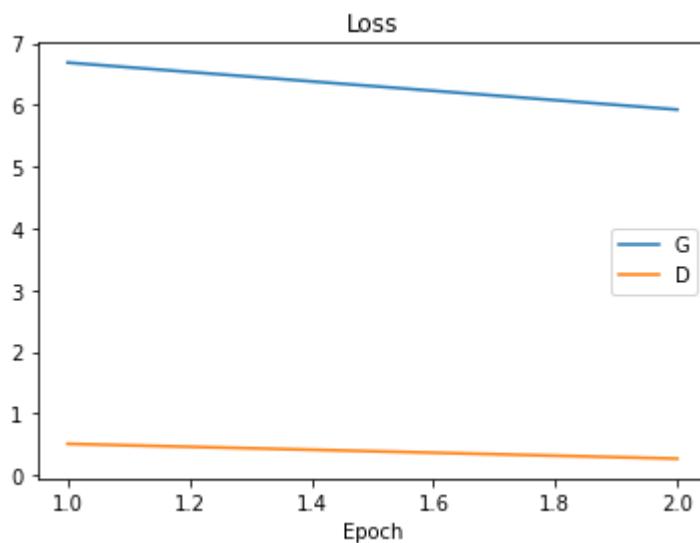
شکل ۱۰ - تصاویر تولید شدهٔ فصل زمستان با استفاده از *Discriminator*

(د)

در این قسمت بجای Discriminator از ساختار U-net استفاده می کنیم.

با این کار دقیق تولید تصاویر واقعی کاهش می یابد، چرا که در تشخیص واقعی بودن تصاویر از Discriminator اصلی استفاده نمی کنیم و استفاده از U-net باعث می شود نتوانیم با دقیق بالایی واقعی بودن یا نبودن تصاویری که تولید شده را بسنجیم.

نتایج استفاده از U-net بصورت زیر می باشد:



شکل ۱۱ - نمودار خطاهای Generator و Discriminator در حالت استفاده از U-net

مقادیر این خطاهای بصورت زیر است:

خطای Generator در اپیک اول: 6.685670790296565

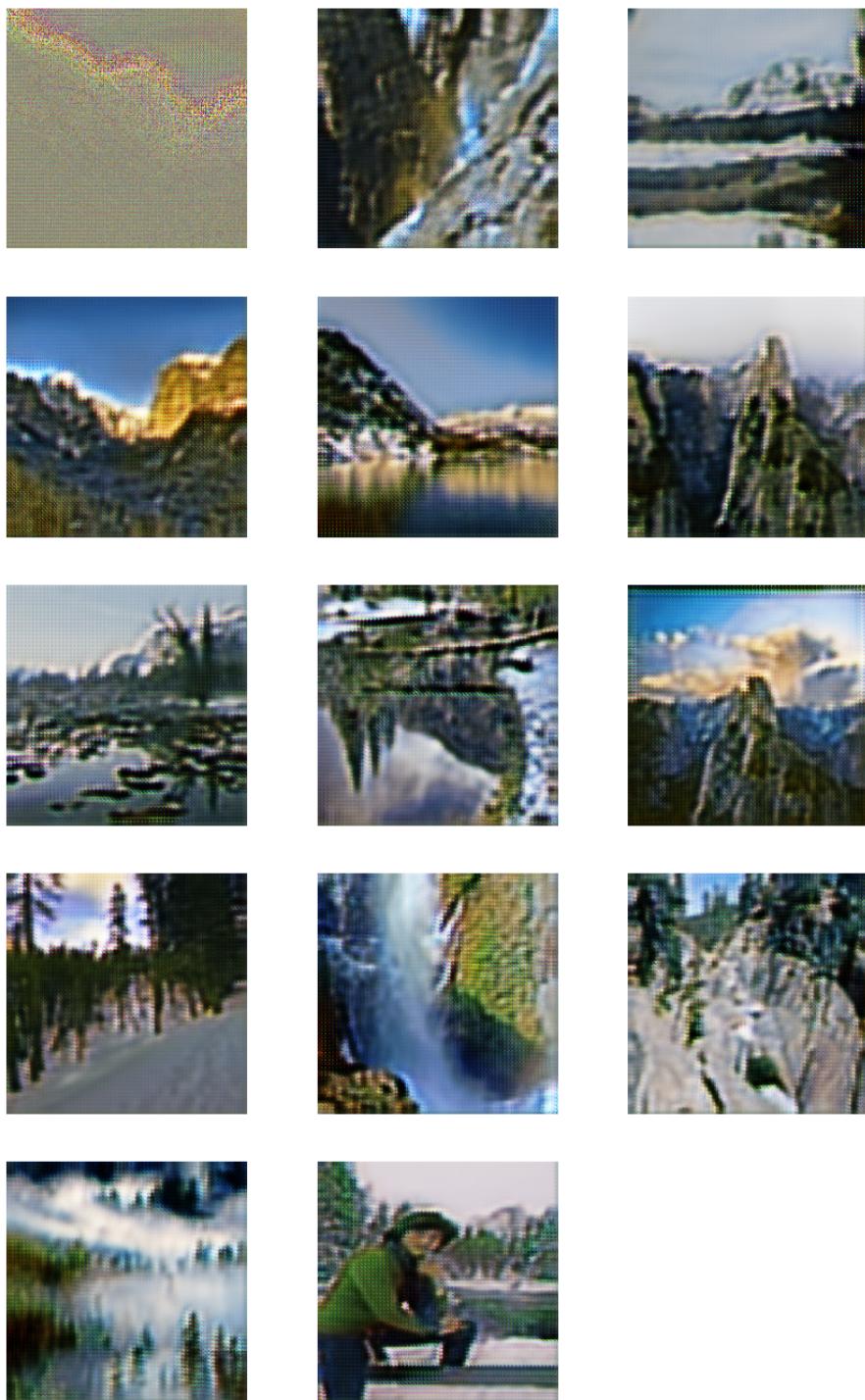
خطای Generator در اپیک دوم: 5.924521194251934

خطای Discriminator در اپیک اول: 0.5087457023361269

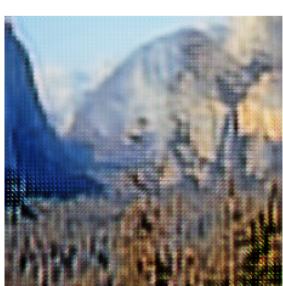
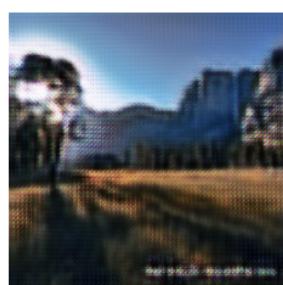
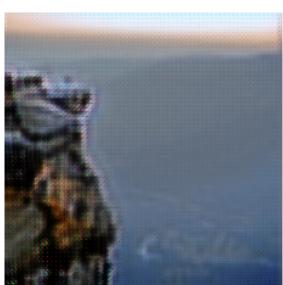
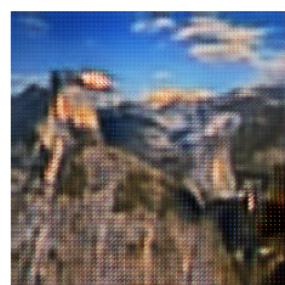
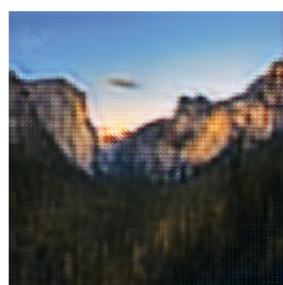
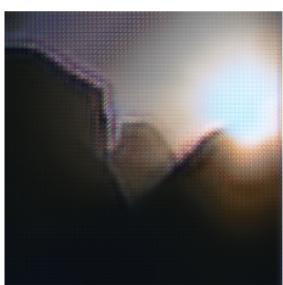
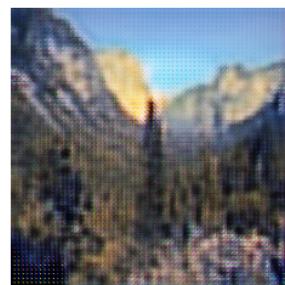
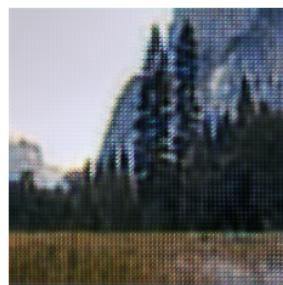
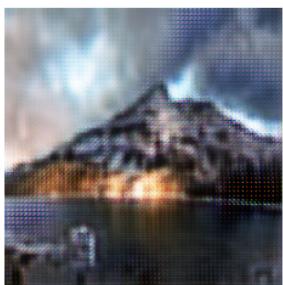
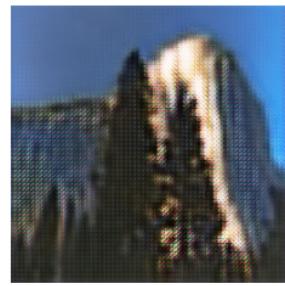
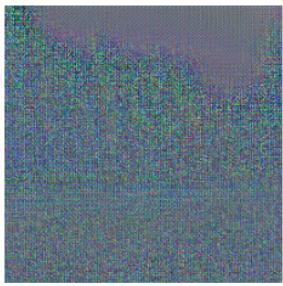
خطای Discriminator در اپیک دوم: 0.26870550709125185

همانطور که میبینیم خطاهای نسبت به حالت قبل بیشتر شده است.

تصاویر تولید شده نیز به صورت زیر می باشد:



شکل ۱۲ - تصاویر تولید شده‌ی فصل نایستان با استفاده از *U-net*



شکل ۱۳ - تصاویر تولید شده‌ی فصل زمستان با استفاده از *U-net*

همانطور که در تصاویر تولید شده مشخص است، بخاطر استفاده از U-net میزان واقعی بودن تصاویر کمتر از حالت قبل است و تصاویر خیلی واقعی به نظر نمی رسند.

در حالیکه در حالت قبل که از Discriminator استفاده کرده بودیم، تصاویر به نسبت واقعی تری تولید شده بودند.

منابع

<https://agustinus.kristia.de/techblog/2017/02/04/wasserstein-gan/>

<https://machinelearningmastery.com/how-to-implement-wasserstein-loss-for-generative-adversarial-networks/>

<https://github.com/ozanciga/gans-with-pytorch>

<https://medium.com/mlearning-ai/how-to-improve-image-generation-using-wasserstein-gan-1297f449ca75>

<https://github.com/eriklindernoren/PyTorch-GAN/blob/master/implementations/sgan/>

<https://medium.com/dataseries/variational-autoencoder-with-pytorch-2d359cbf027b>