به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

شبکه های عصبی و یادگیری عمیق

مینی پروژه ۳

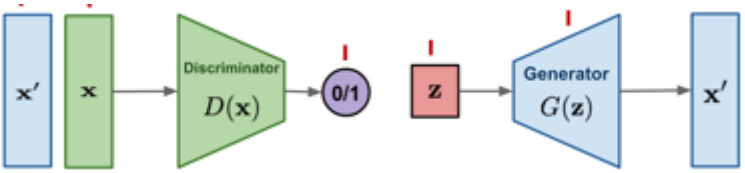
نام و نام خانوادگی	فاطمه سلیقه – زهرا نصرالهی
شماره دانشجویی	
تاریخ ارسال گزارش	۹۹/۰۵/۱۴

فهرست گزارش سوالات (لطفاً پس از تکمیل گزارش، این فهرست را به‌روز کنید.)

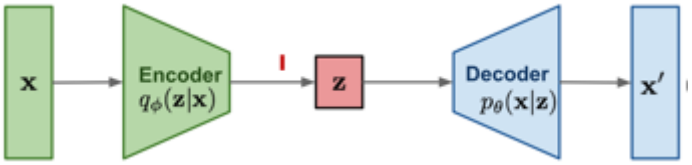
سوال ۱ – Variational AutoEncoder

الف) دو روش GANs و VAE هر دو روش های generative هستند . معماری کلی این دو روش به صورت زیر است .

در روش GANS از دو بخش discriminator و generator استفاده می کنیم . این مدل سعی می کند تا یک بردار نویز را دریافت نموده و عکسی را تولید کند که جدید است . این روش ، روش موفق تری در تولید الگوهای شبه واقعی دارد و لی نقطه قوت تفسیر آماری مانند روش VAE را ندارد.

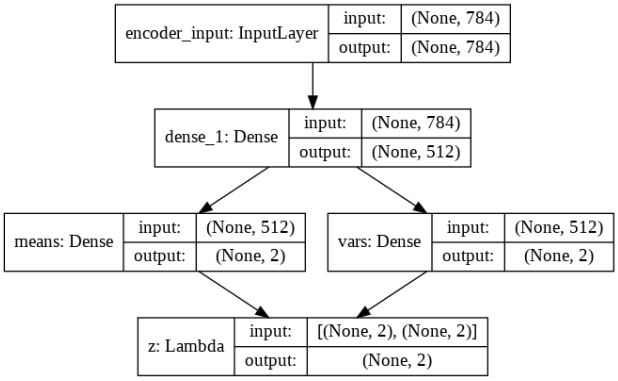


اما روش VAE به این صورت عمل می کند که ابتدا عکسی را دریافت می کند سپس سعی می کند تا ابعاد آن را کاهش دهد و سپس با تولید داده سعی می کند عکسی را تولید کند که شبیه عکس ورودی است . این روش به صراحت الگوهای تولیدی را با لحاظ مفهوم تابع چگالی احتمالاتی داده های ورودی ایجاد می کند ولی مشکل بهینه سازی در آن وجود دارد. از مزایای دیگر این روش این است که روی داده خرجی کنترل داریم . یعنی گاهی اوقات نمی خواهیم که داده خروجی کاملا رندوم باشد و می خواهیم که تنها تفاوت هایی با داده ورودی داشته باشد. از مزایای دیگر می توان به این مورد اشاره نمود که برای مقایسه دو VAE راهکارهایی وجود دارد اما نمی توانیم دو مدل GAN رو با هم مقایسه کنیم

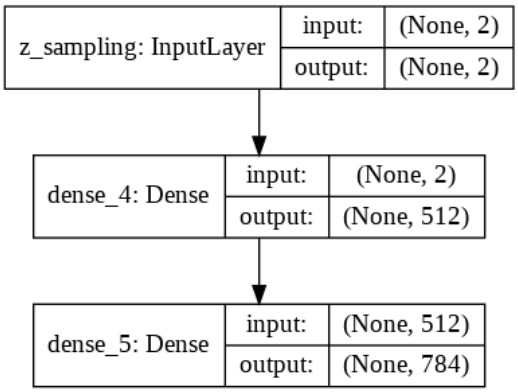


ب)

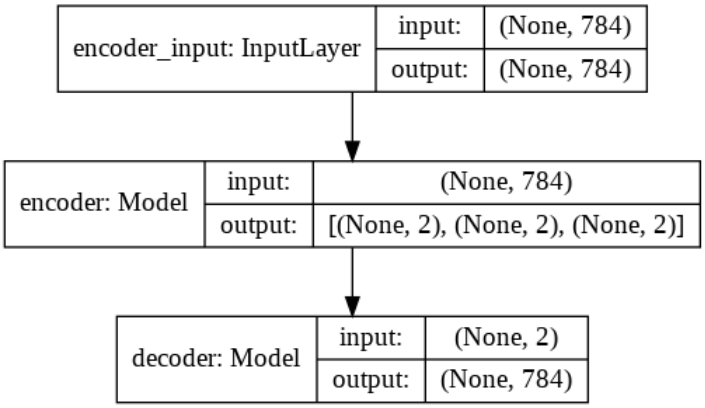
برای پیاده سازی vae از دو بخش انکودر و دیکودر تشکیل شده است که در نهایت مدل نهایی را می سازند . انکودر به صورت زیر تعریف شده است :



دیکودر هم به صورت زیر ایجاد شده است :



و ترکیب آنها که در نهایت مدل vae را می شازد نیز به صورت زیر است :



دیتاست mnist از ۱۰ کلاس شامل ارقام صفر تا ۹ تشکیل شده است .

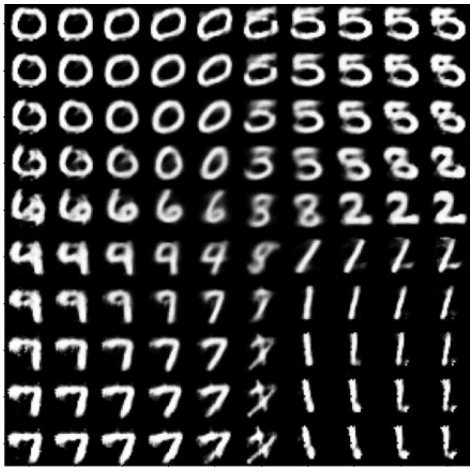
باتوجه به شکل های زیر مشاهده می کنیم هر چه جلوتر می رویم تصاویر واضح تر می شود . البته تصاویر مربوط به به گوشه های شکل واضح تر هستند و تصاویر مربوط به وسط های تصویربه مرور با گذشت چندایپاک واضح تر می شود .

ایپاک 0

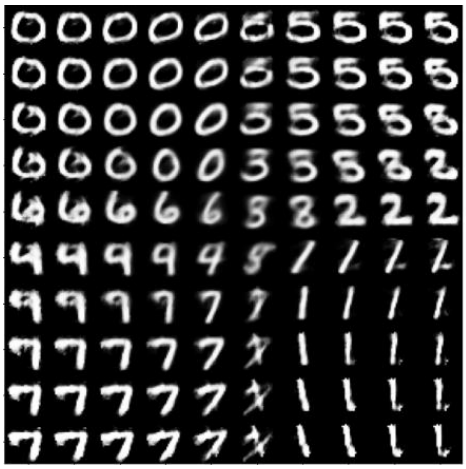
ایپاک ۱۰

ایپاک ۲۰

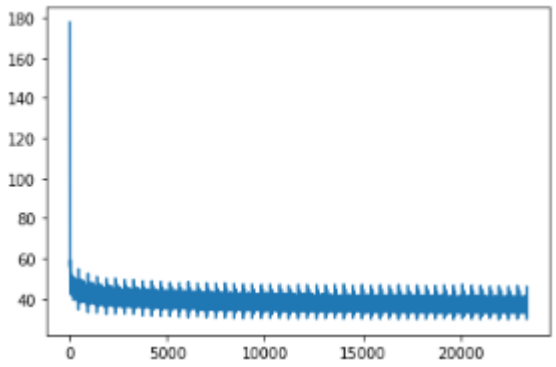
ایپاک ۳۰



ایپاک ۴۰



نمودارلاس که در زیر مشاهده می کنیم ابتدا بسیار بالاست و با گذشت زمان کوتاهی بسیار کاهش می یابد .



ج)

تابع loss مربوط به vae از دو بخش تشکیل شده است یک بخش مربوط به تولید خروجی است که باید به ورودی نزدیک باشد . بنابراین می توان از mse استفاده نمود برای اینکه خروجی به ورودی نزدیک باشد . بخش دوم از واگرایی KL استفاده می کنیم . به این مفهوم که اگر ما ورودی X را داشته باشیم و بخواهیم در فضای latent متغیر Z را بدست آوریم نیاز داریم تا p(Z) را از روی p(Z|X) به دست آوریم .بنابراین لازم است تا P(Z|X) را داشته باشیم اما چون این توزیع را نداریم سعی می کنیم تا آن را تخمین بزنیم . بنابراین یک توزیع ساده تر مثلا Q را درنظر می گیریم تا P را از روی آن تخمین بزنیم . بنابراین لازم است تا انکودر توزیع Q را یاد بگیرد . بنابرین تابع زیر را در نظر می گیریم و سعی داریم تا آن را مینیمم کنیم :

$$D_{KL}[Q(z|X)||P(z|X)] = E[\log Q(z|X) - \log P(z|X)]$$

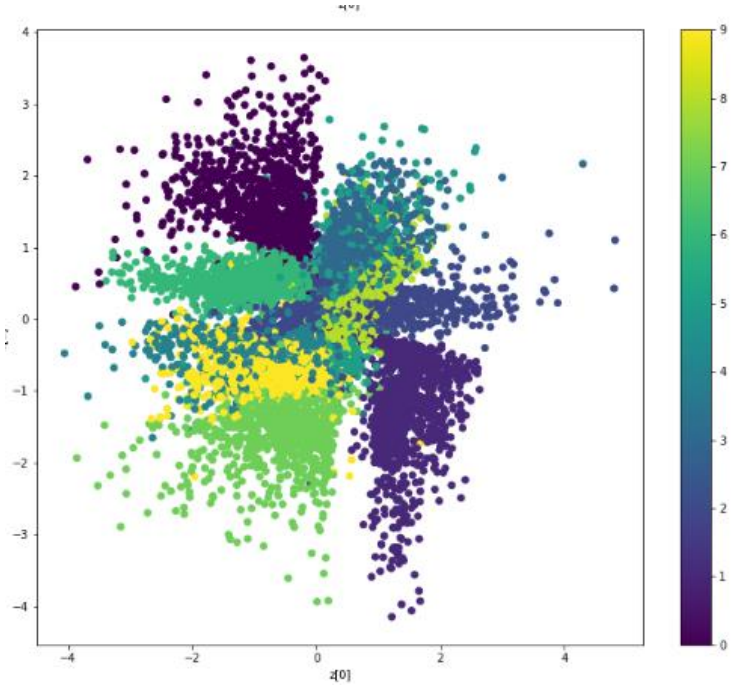
برای توزیع Q توزیع نرمال با میانگین صفر و واریانس یک را در نظر میگیریم .

به صورت کلی تابع لاس به صورت زیر است :

$$vae_{loss} = ||x - f(z)||^2 + KL[N(\mu(x), \sum(x))||N(0,1)]$$

د)

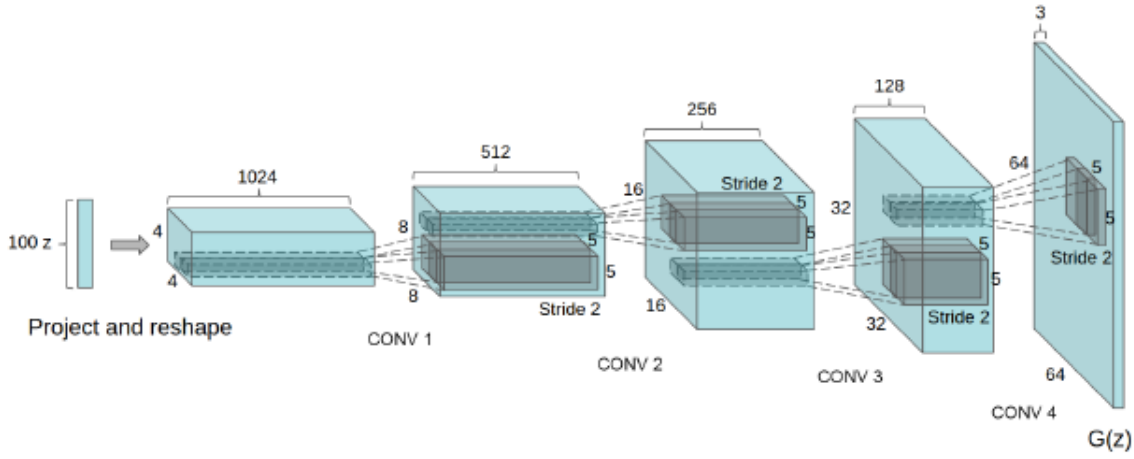
همانطور که در شکل زیر می بینیم داده ها را در ۱۰ کلاس مختلف دسته بندی نموده ایم (classification) و همان طور که در شکل می بینیم کلاس ها کاملا از همدیگر مستقل نیستند و در یک ناحیه وسط در هم فرورفتگی دارند و نمونه هایی که در ناحیه وسط قرار گرفته اند مانند انچه در شکل های بخش ب دیدیم سخت تر تشخیص داده می شوند و وضوح کمتری دارند



سوال ۲ – DCGAN

الف) DCGAN یک از انواع الگوریتم های GANs (Generative Adversarial Network) است و تفاوتی که با GAN دارد این است که در GAN لایه generator یک لایه fully connected است اما DCGAN برای generator از convolution جابه جا شده یا به عبارتی همان fractionally-strided convolute استفاده می کند به منظور انجام عمل up_sampling برای تولید تصاویر دو بعدی . DCGAN از لایه های کانولوشنی تشکیل شده بدون maxpooling و لایه های fully connected . بنابراین برای طراحی DCGAN لازه است تا لایه های fully connected را حذف کنیم و از fractionally-strided convolute برای افزایش بعد و upsampling استفاده کنیم . از batch normalization استفاده می کنیم به جز در لایه اخر generator و لایه اول discriminator . هم چنین لایه های pooling را در generator با fractional_strided convolution و در discriminator با strided convolution جایگزین می کنیم . برای activation function هم در generator از تابع RELU استفاده می کنیم به جز لایه آخر که از tanhاستفاده می کند و در discriminator هم از LeakyRELU استفاده می کنیم . ابعاد مورد نظر و اندازه فیلتر های لایه کانولوشنی را می توان در شکل پایین صفحه نیز ملاحظه نمود . در این شبکه ابتدا با استفاده از تصاویر حقیقی و تصاویر تولید شده توسط generator سعی می کنیم تا discriminator را آموزش دهیم سپس یک فضای latent تولید می کنیم و سعی می کنیم تا generator را آموزش دهیم .

ب)برای تولید عکس دو بعدی مطابق شکل زیر ابتدا یک بردار latent ایجاد می کنیم با بعد ۱۰۰ به این صورت که از توزیع نرمال یک عدد رندوم از بعد ۱۰۰ تولید می کنیم . سپس آن را به فضای 1024*4*4 نگاشت کرده و سپس از لایه های fractionally-strided convolute برای افزایش بعد به 512*8*8 و سپس 256*16*16 و سپس 128*32*32 برده و در نهایت در لایه آخر یک تصویر RGB دو بعدی با ابعاد 64*64 ایجاد می نماییم .



ج)

دلیل استفاده از strided convolution در discriminator آن است که می خواهیم عمل down sampling (کاهش ابعاد)را انجام دهیم . و دلیل استفاده از fractional strided convolution در generator آن است که می خواهیم عمل upsampling (افزایش ابعاد) را انجام دهیم .

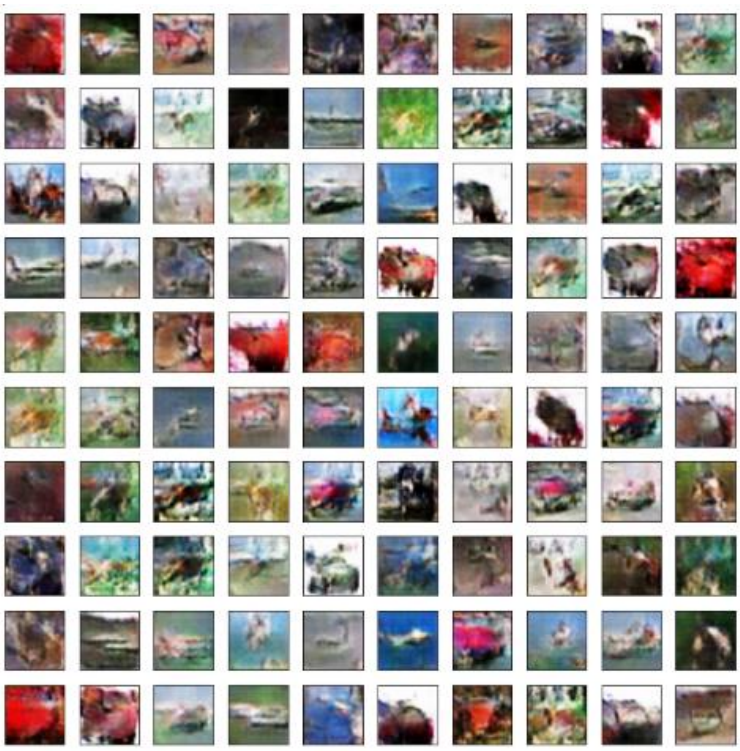
د)

شبکه GAN از دو بخش generator و discriminator تشکیل شده است . هر بخش را به صورت جداگانه تعریف می کنیم و سپس دو مدل را با هم ترکیب نموده و مدل DCGAN نهایی را ایجاد می کنیم . برای آموزش شبکه به این صورت عمل می شود که ابتدا تعدادی داده واقعی به discriminator داده تا بتواند تصاویر واقعی را یاد بگیرد سپس generator تعدادی تصاویر fakeتولید می کند و به discriminator می دهیم تا یاد بگیرد . در طول این زمان generator عمل یادگیری را انجام نمی دهد . پس از این مرحله نوبت به یادگیری generator می رسد . البته generator هیچ گاه به صورت منحصر به فرد آموزش نمی بیند بلکه کل شبکه در واقع آموزش می بیند به غیر از discriminator در واقع discriminator است که generator را آموزش می دهد.

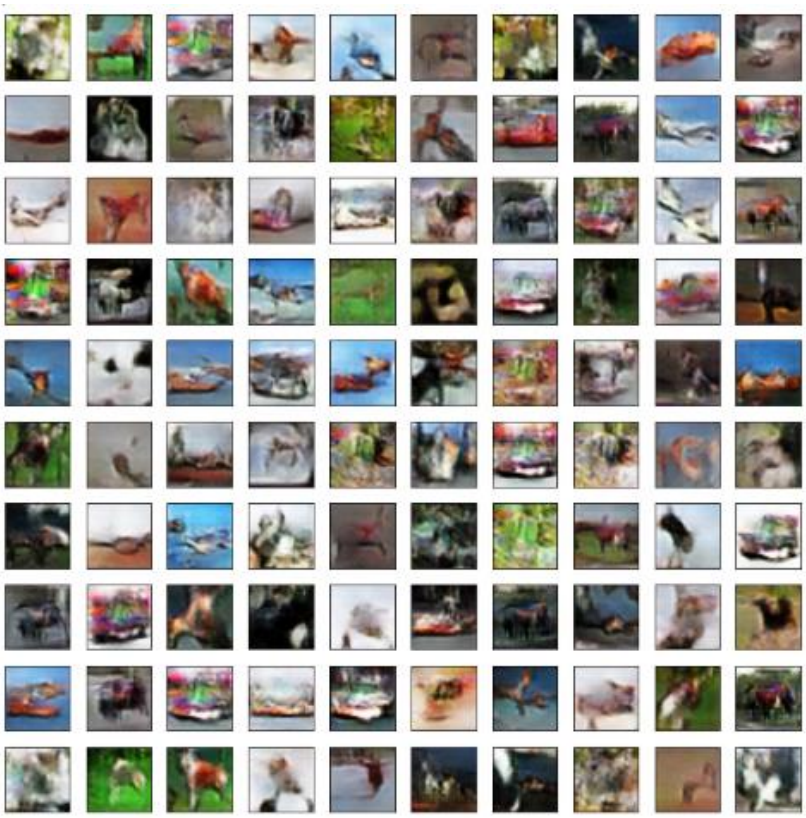
خروجی به صورت های زیر است که در ۵۱ ایپاک آموزش داده شده است . ابتدا تصاویر اصلا واضح نیستند و به مرور تصاویر واضح تر شده و در نهایت تصاویر قابل تشخیص می شوند .

ایپاک ۱

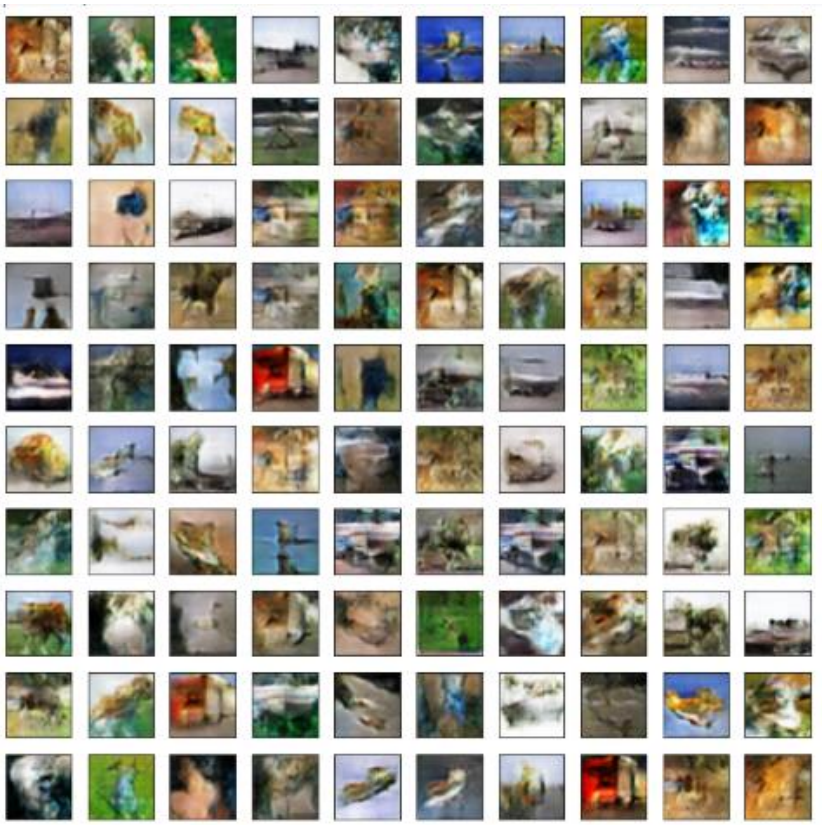
ایپاک ۱۱



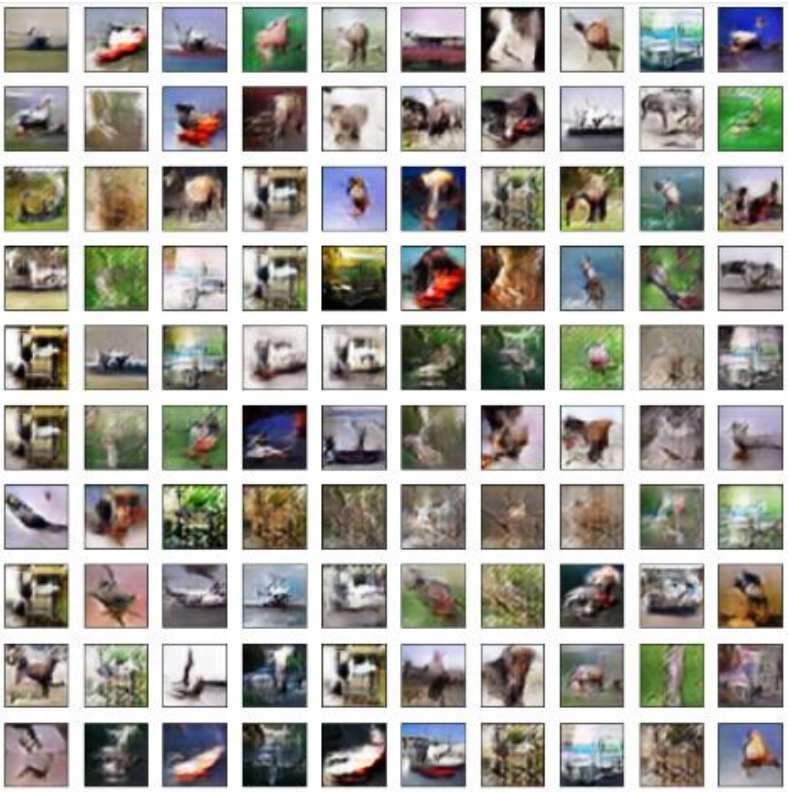
ایپاک ۲۱



ایپاک ۳۱

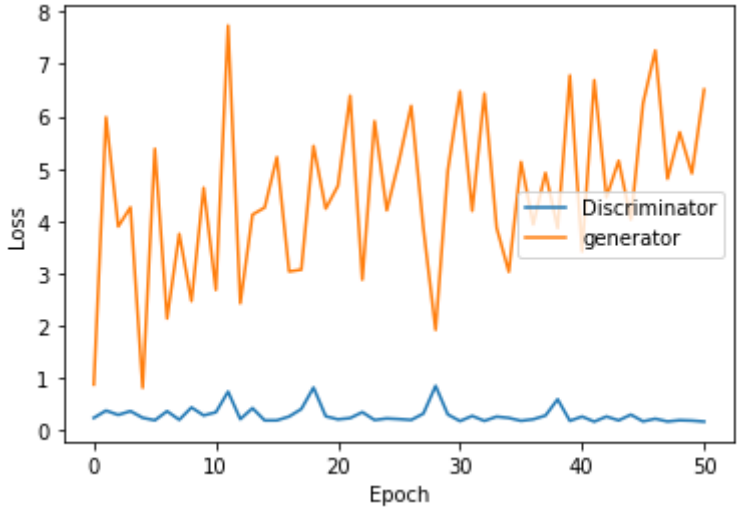


ایپاک ۴۱



ایپاک ۵۱

نمودار لاس بخش آموزش discriminator و بخش اموزش generator به صورت زیر است لاس مربوط به discriminator کاهشی است ام لاس مربوط به آموزش generator متغیر است و نوسانات زیادی دارد .



ه)

همان طور که در قسمت د توضیح داده شد دو بخش discriminator و generator به صورت جداگانه آموزش می بینند . یکی از چالش ها استفاده از تابع loss مناسب برای آموزش هر دو بخش می باشد .

۴ نوع تابع loss وجود دارد که در دو دسته مختلف standard GAN Loss Functions و alternate GAN Loss Functions معرفی شده اند که در ادامه توضیح داده خواهند شد :

standard GAN Loss Functions :

۱) Minimax GAN Loss :

این روش مانند یک بازی دو نفره می ماند که generator یک بازیکن و discriminator یک بازیکن هستند . هدف از این تابع آن است تا تابع هزینه generator را min و تابع هزینه discriminator را max کند:

Discriminator:

$$\text{maximize } \log D(x) + \log(1 - D(G(z)))$$

در واقع Discriminator می خواهد احتمال تشخیص real تصاویر واقعی و احتمال تشخیص fake بودن تصاویر تولید شده توسط generator را max کند .

Generator:

$$\text{minimize } \log(1 - D(G(z)))$$:

Generator هم در واقع می خواهد تا تصاویر را به گونه ای ایجاد کند که احتمال اینکه discriminator آن عکس ها را fake تشخیص دهد کاهش یابد .

اما در این روش تابع loss مربوط به generator می تواند saturate شود به عبارت دیگر اگر generator نتواند به اندازه discriminator سریع یاد بگیرد discriminator می برد زیرا عکس های تولید شده تفاوت زیادی با عکس های واقعی دارند discriminator می تواند با اطمینان بالا عکس های fake را تشخیص دهد .

۲) Non-Saturating GAN Loss:

این روش modify شده روش minimax است که می خواهد از saturate شدن تابع خطای generator جلوگیری کند . تابع خطای مربوط به generator را به صورت زیر تغییر می دهیم .

Generator :

$$\text{maximize } \log(D(G(z)))$$

در این روش generator در واقع می خواهد تا احتمال اینکه discriminator تصاویر تولید شده را real تشخیص دهد ماکزیمم کند .

Alternate GAN Loss Functions :

۳) Least Square GAN Loss :

Discriminator :

$$\text{minimize } (D(x) - 1)^2 + (D(G(z)))^2$$

: Generator

$$\text{minimize } (D(G(z)) - 1)^2$$

از توابع بیان شده می توان دید که generator و discriminator هر دو می خواهند که فاصله بین خروجی discriminator و مقدار حقیقی خروجی را کاهش دهند . بنابراین لازم است تا مقدار خروجی حقیقی را هم در نظر بگیریم . در عمل تابع خطای least square به صورت زیر است :

$$\text{l2 loss} = \text{sum } (y\_predicted - y\_true)^2$$

طبق این تابع هر چه خطا بیشتر باشد میزان مجازات بیشتر است و از vashing gradient جلوگیری می شود . این روش به این صورت است که در هنگام آموزش discriminator داده های حقیقی با y_true برابر یک و داده های تولید شده با y_true برابر صفر داده میشوند . در هنگام آموزش y_true generator برابر یک در نظر گرفته می شود .

۴) Wasserstein GAN Loss :

GAN ها پیش از WGAN سعی داشتند تا فاصله بین توزیعخروجی حقیقی و پیش بینی شده را برای تصاویر real و fake مینیمم کنند که به آنها kullback-Leibler و Jensen-Shannon گفته میشد . اما در WGAN مدل جدیدی از محاسبه فاصله به نام Earth-Mover's distance برای مدل سازی مسئله استفاده شد که به آن فاصله Wasserstein distance گویند .

در این مدل به discriminator، Critic گویند که به جای تشخیص real یا fake بودن تصویر یک score میدهد که میزان real و fake بودن را مشخص می کند . میزان به روز رسانی critic تقریبا ۵ برابر generator است و باید وزن های مدل مقادیر کم مثلا بین 0.01- و 0.01 باشند .

در این مدل critic آموزش داده می شود تا بتواند Wasserstein distance را تخمین بزند سپس از critic برای آموزش generator استفاده می شود .

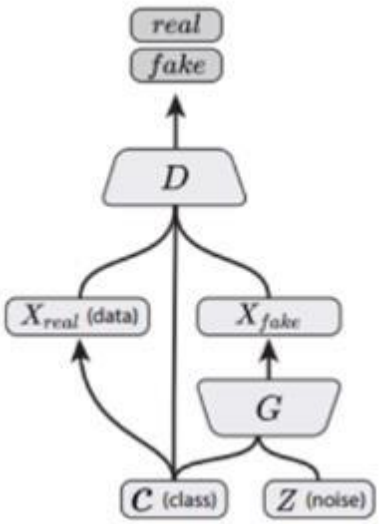
Score به این صورت محاسبه می شود که score بین real , fake بودن تصاویر ماکزیمم فاصله را داشته باشد.

$$\text{Critic Loss} = [\text{average critic score on real images}] - [\text{average critic score on fake images}]$$
$$\text{Generator Loss} = -[\text{average critic score on fake images}]$$

سوال ۳ – Conditional GAN

الف)

شبکه های GAN از جمله فریم ورک های مدل های تولیدی هستند که در جهت ساخت نمونه ای غیر واقعی و غیرقابل تشخیص از نمونه های واقعی کار شده اند. نقطه عطف این شبکه ها، این است که زنجیره ی مارکوف نیاز ندارد و با استفاده از backpropagation گرادیان را آپدیت می کند. در مدل های غیر شرطی، کنترلی روی داده های تولید شده قرار نمی دهد، در حالی که در مدل های شرطی آن، اطلاعاتی را در جهت ساخت مدل، اضافه می کند و این شرط ها می تواند در label های کلاس باشند. مدل CGAN به صورت زیر تعریف می شود:



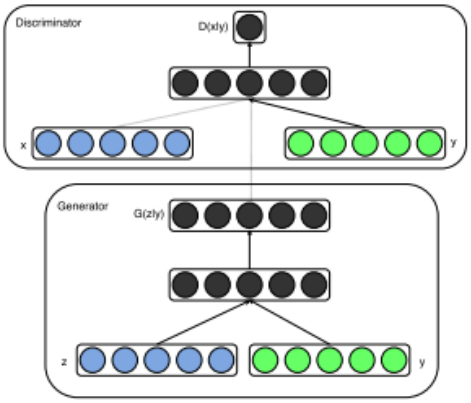
همانند دیگر GAN ها، دارای دو بخش Generator و Discriminator است. Generator توزیع داده را به عهده دارد به این صورت که از توزیع نویزی احتمالی، یک تابع نگاشت می سازد و Discriminator مسئول کنترل و بررسی داده ی تولید شده با نمونه های واقعی است. هر دو بخش همانند شبکه عصبی پرسپترون، تابع های non-linear mapping دارند. و پارامترهای آن ها را با یادگیری، تنظیم می کنیم. پارامترهای G را به گونه ای تنظیم می کنیم که مقدار $\log(1 - D(G(z)))$ را مینیمم کند و پارامترهای D را به گونه ای تنظیم می کنیم تا مقدار $logD(X)$ را مینیمم کند.

$$\min G \max D V(D, G) = Ex\sim pdata(x) [\log D(x)] + Ez\sim pz(z) [\log(1 - D(G(z)))]$$

در CGAN هر دو بخش، روی اطلاعات اضافه y شرط قرار می دهند مثلا نوع کلاس مربوطه. در generator، p(z) و y در hidden representation با هم ترکیب می شوند و انعطاف بیشتری را در چگونگی این hidden representation ساخته شده، ایجاد می کنند. در Discriminator، x و y به عنوان ورودی قرار می گیرند و سعی در برقراری رابطه زیر می کنند:

$$\min G \max D\ V(D, G) = E_{x\sim pdata(x)}[\log D(x|y)] + E_{z\sim pz(z)}[\log(1 - D(G(z|y)))].$$

در حالت کلی ساختار شبکه CGAN به صورت زیر است:



ب)

ابتدا داده های CIFAR-10 را با تابع کتابخانه ای `cifar10.load_data()` در دو دسته ی داده های آموزش به تعداد ۵۰۰۰۰ و داده های تست به تعداد ۱۰۰۰۰ تا دانلود کردیم. نمونه ای از تصاویر به صورت زیر است:

با استفاده از تابع کتابخانه keras به نام ()concatenate سعی شده است یک Genator بسازیم که ورودی آن فضای Z latent و کلاس ها هستند و به یکدیگر concate شده و به عنوان ورودی وارد generator می شوند. سپس یک fully connected با تابع dense و ۴ شبکه Conv transpos پیاده کردیم که هرکدام دارای لایه های batchNormalization و LeakyRelu هستند.

پارامترهای هر لایه شبکه به صورت زیر است:

پارامترها	مقادیر
Dense(fully connected)	2,2,512
Num of Conv2DTranspose layers	4
stride	2
Activation function	leakyrelu

حال ساختارDiscriminator را ایجاد می کنیم که شامل ۴ لایه Conv و دو لایه dense برای fully connected است. که هر کدام دارای لایه های batchNormalization و LeakyRelu هستند. بعد از ساختار لایه های conv ، این مدل را با کلاس های ورودی concatenate می کنیم و از دو لایه dense با تعداد نرون های ۵۱۲ و ۱ عبور می دهیم. در واقع مدلی ایجاد می کنیم که ورودی آن عکس های واقعی با ابعاد ۳۲*۳۲ و کلاس ها هستند و خروجی ۰و ۱ است.

مقادیر پارامترها به صورت زیر است:

پارامترها	مقادیر
Num of CNNs	4
stride	2
Activation function	relu
Dense1	512
Activation funcrion dense1	relu
Dense2 (output layer)	1
Activation function dense2	sigmoid

optimizer	adam
Learning rate	0.0002
Loss function	Binary cross entropy

حال آنچه generator تولید می کند را وارد Discriminator می کنیم و این مدل ترکیب شده را کامپایل می کنیم تا ساختار یک CGAN ایجاد شود.

پارامترهای مدل CGAN به صورت زیر است:

optimizer	Adam
Learning rate	0.0004
Loss function	Binary_cross_entropy
Latent dim	100

پارامترهای لازم جهت آموزش مدل:

#epochs	100
Batch_size	64

در ابتدا مدل discriminator را با داده های اصلی آموزش می دهیم و سپس با استفاده از generator تصاویر را ایجاد می کنیم. برای ساخت تصویر fake نیز با استفاده از تابع

```
z = np.random.normal(loc=0, scale=1, size=(samples, latent_dim))
```

نویز با توزیع نرمال ایجاد کرده و برای کلاس های داده ها، با تابع کتابخانه ای predict یک تصویر جدید می سازیم.

حال تصاویر ایجاد شده وارد discriminator می شوند و میزان خطای آن ها را براساس کلاس داده های اصلی به دست می آورد و دوباره پارامتر های خود را آپدیت کرده و آموزش می بینید. حال generator بعد از آموزش discriminator، باید تصاویر بهتر از قبل ارائه دهد. در نتیجه شبکه تولید شده از ترکیب این دو

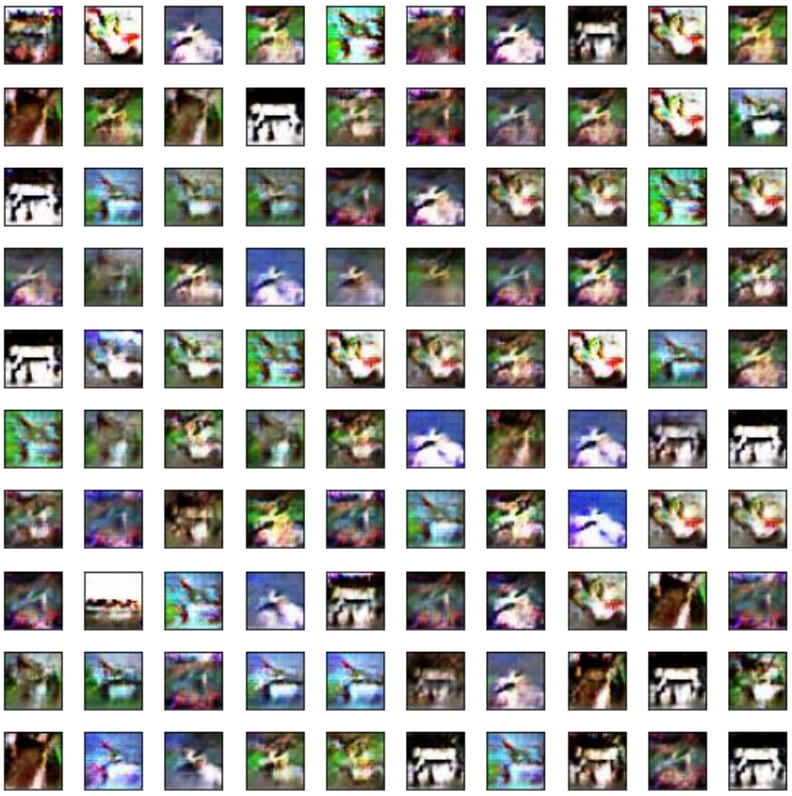
را آموزش می دهیم تا میزان یادگیری generator را بهبود بخشیم و آن را آپدیت کند. این چرخه را ادامه می دهیم تا هربار تصاویر بهتری تولید کنیم و داوری بهتری در تشخیص تصاویر فیک از تصاویر اصلی ایجاد شود. در واقع سعی شده است تا فرمول زیر نمود پیدا کند.

$$\min G \max D\ V(D, G) = E_{x\sim pdata(x)}[\log D(x|y)] + E_{z\sim pz(z)}[\log(1 - D(G(z|y)))].$$

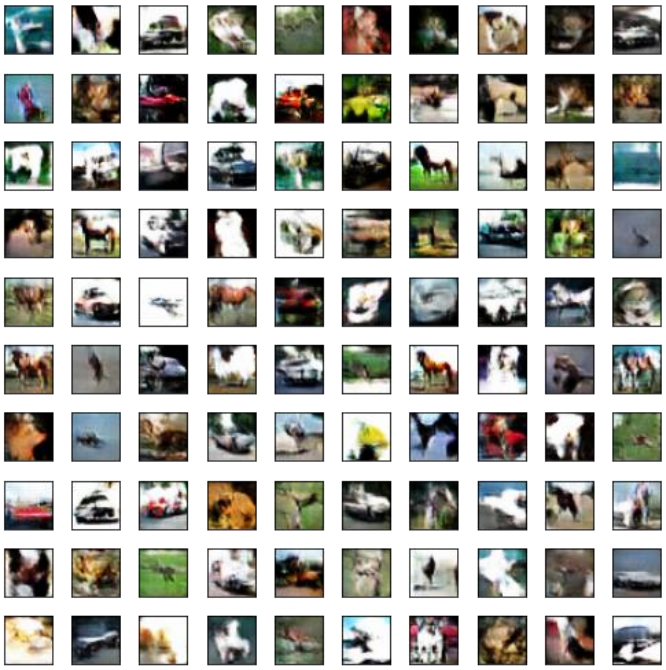
در هر اپاک میزان خطای generator و discriminator را نمایش داده ایم.

در هر ۱۰ اپاک یکبار به وسیله ی generator آپدیت شده، تصاویر جدید را ساخته و نمایش می دهیم:

```
epoch = 1/100, d_loss=0.166, g_loss=1.896
```

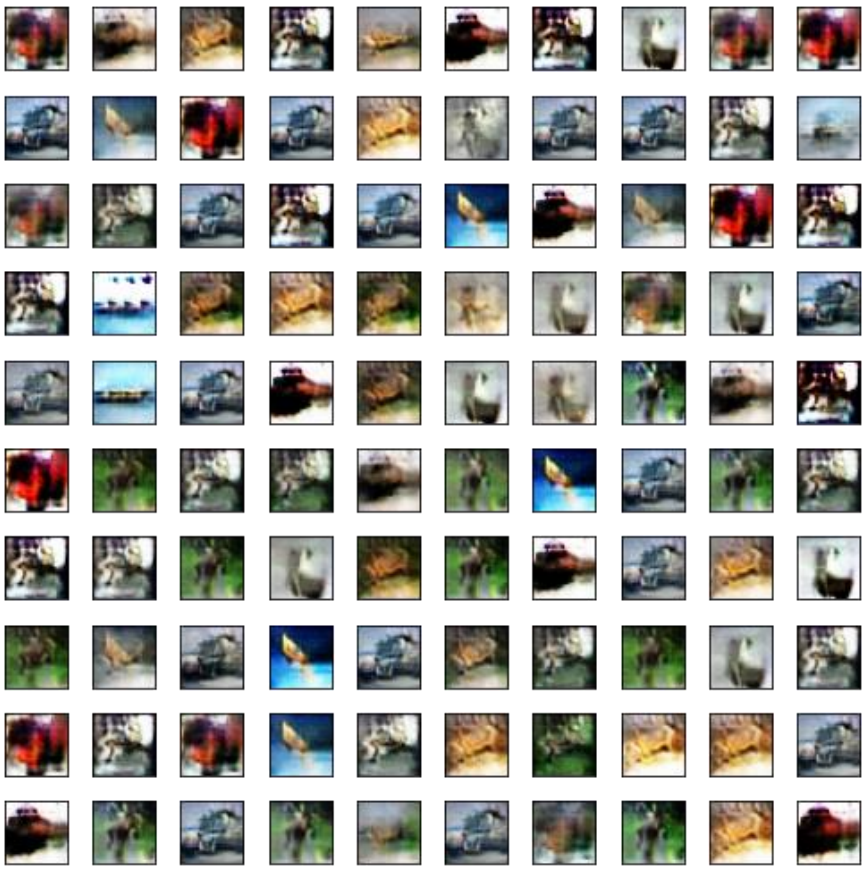


```
epoch = 21/100, d_loss=0.165, g_loss=7.445
```

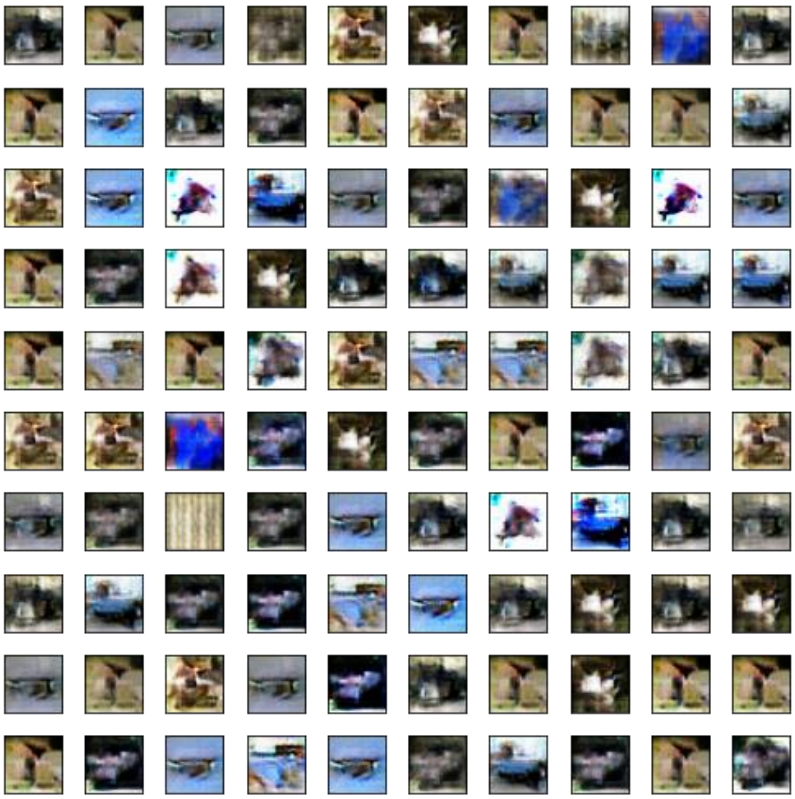


این تصاویر، شفافیت بهتری نسبت به تصاویر قبل داشته اند، حتی در بعضی عکس ها به وضوح می توان شیء رسم شده را تشخیص داد.

```
epoch = 41/100, d_loss=0.166, g_loss=6.710
```

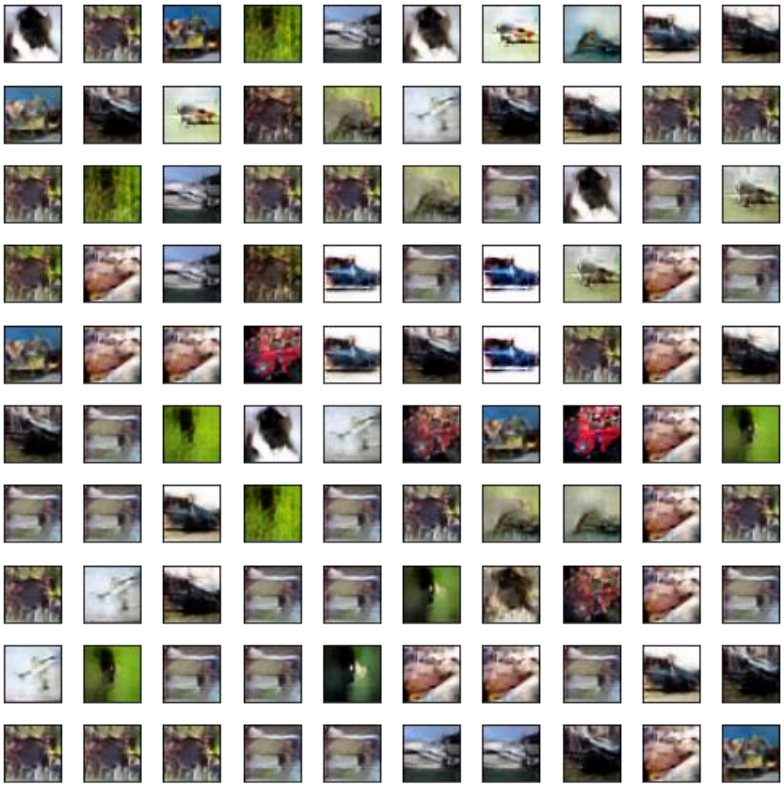


epoch = 61/100, d_loss=0.170, g_loss=5.275



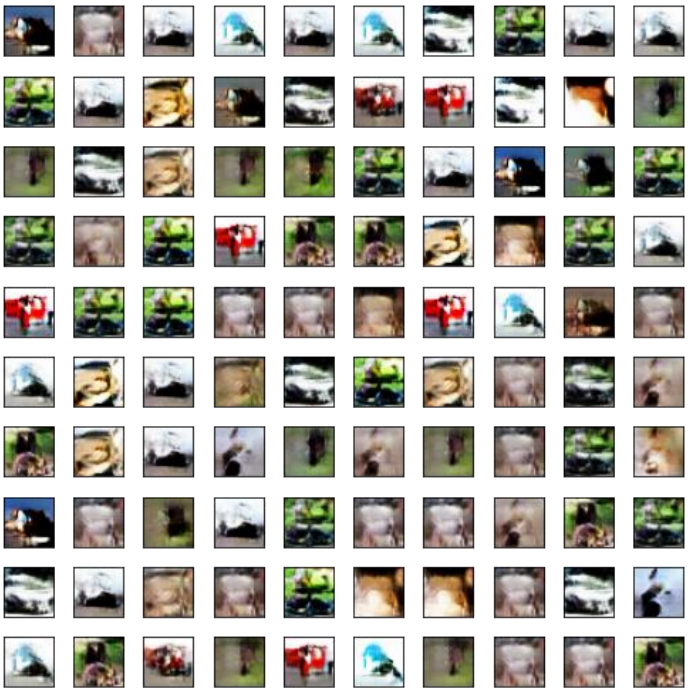
epoch = 81/100, d_loss=0.163, g_loss=9.102

```
epoch = 101/100, d_loss=0.172, g_loss=6.755
```



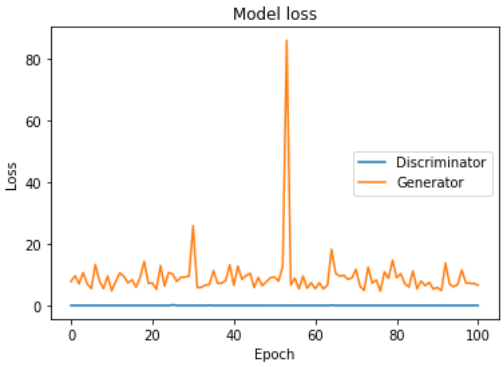
در اپاک های اولیه، تصاویر به صورت نامفهوم بوده و کم کم با آموزش شبکه، تصاویر به تصاویر واضحی تری از اشیا موجود در CIFAR-10 نزدیک تر شده است.

```
epoch = 81/100, d_loss=0.242, g_loss=5.855
```

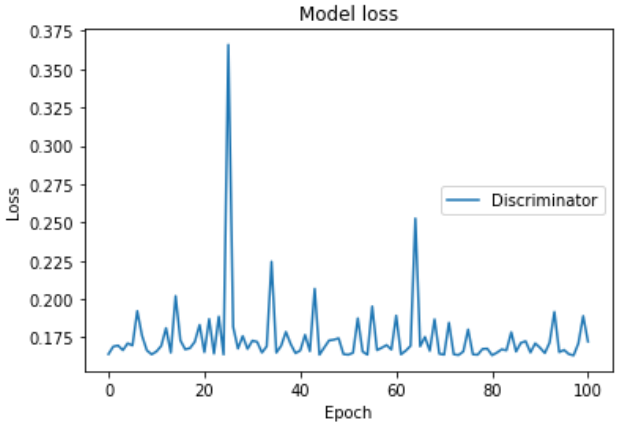


میزان نزدیک بودن تصاویر به اشیای واقعی، بستگی به پارامترهای مختلفی دارد، از جمله batch_size که برای آپدیت کردن پارامترها استفاده می شود و نیز تعداد اپاک هایی که الگوریتم تکرار می شود. برای رسیدن به بهترین عملکرد، لازم است شبکه با مقادیر مختلف پارامترها، اجرا و امتحان شود.

میزان خطای discriminator و generator را در هر اپاک به صورت نمودار نمایش داده ایم:



Discriminator توانایی بهتری را در تشخیص داده های فیک از خود نشان داده است، به طوری که در هر اپاک، مقادیر خطای بین ۱و ۰ را داشته است. در تصویر بالا به علت بازه ی نوسانی جنریتور، نوسانات discriminator واضح نبوده است، که در تصویر زیر به صورت جداگانه رسم شده است:



همان طور که دیده می شود، بین discriminator و generator رقابت وجود دارد و ممکن است در ایپاکی هر کدام بهتر عمل کند؛ به طور کلی نمودار خطای آن ها دچار نوسانات شود. زیرا وقتی خطای generator کاهش پیدا میکند ، خطای discriminator افزایش پیدا می کند و بالعکس. اگر generator صعودی باشد و discriminator نزولی باشد، به معنای عملکرد ضعیف discriminator است و اگر discriminator صعودی باشد و generator نزولی باشد، به معنای عملکرد ضعیف generator خواهد بود و تصاویر تولیدی کیفیت پایینی خواهد داشت.

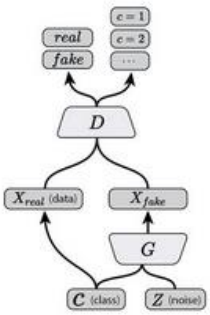
کدهای مربوط به این قسمت در فایل CGAN.ipynb ذخیره شده است.

ج)

سه شبکه معرفی شده در این سوال، از نوع شبکه های GAN ای هستند که discriminator آن، Latent variables را نیز پیش بینی می کند. ما AC-GAN را انتخاب کرده ایم. پیش بینی می شود در شرایط یکسان (شبکه طراحی شده و پارامترهای یکسان) این مدل بهتر از CGAN عمل کند.

Auxiliary Classifier GAN (AC-GAN)

این نوع از GAN ها، همانند CGAN عمل می کند، با این تفاوت که discriminator علاوه بر تشخیص تصاویر فیک از تصاویر واقعی، نوع کلاس هر تصویر را نیز می آموزد و سعی میکند لیبل کلاس تصویر را پیش بینی کند. طراحی شبکه آن را به صورت زیر نمایش می دهیم:



در رابطه ی زیر ، discriminator سعی می کند تا احتمال تخصیص داده شده به منبع درست را ماکزیمم کند و generator نیز سعی می کند ترم دوم این رابطه را مینیمم کند:

$$L = E[\log P(S = real \mid X_{real})] + E[\log P(S = fake \mid X_{fake})]$$

در generator هر نمونه تولید شده، علاوه بر نویز z (نقطه ی رندوم از فضای latent)، لیبل کلاس مربوطه اش را دارد. Generator تصویر فیک خود را با استفاده از این دو ایجاد می کند G(c,z) و discriminator تصویر را می گیرد و توزیع احتمالی براساس مرجع و توزیع احتمالی براساس لیبل های کلاس میدهد.P(S|X) , P(C|X) = D(X)

تابع هدف دو قسمت و به صورت زیر تعریف می شود:

$$L_S = E[\log P(S = real \mid X_{real})] + E[\log P(S = fake \mid X_{fake})]$$

$$L_C = E[\log P(C = c \mid X_{real})] + E[\log P(C = c \mid X_{fake})]$$

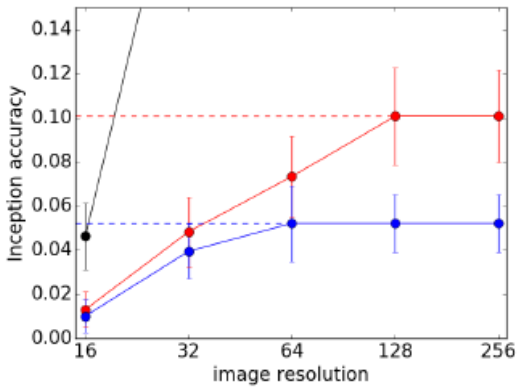
Discriminator آموزش می بیند تا مقدار $L_C + L_S$ را ماکزیمم کند در حالیکه generator آموزش می بیند تا مقدار $L_C - L_S$ را ماکزیمم کند.

معماری discriminator و generator شامل مجموعه از لایه های کانولوشن (شبکه عمیق) به صورت زیر طراحی شده است:

Operation	Kernel	Strides	Feature maps	BN?	Dropout	Nonlinearity
$G_x(z)$ – 110 × 1 × 1 input						
Linear	N/A	N/A	384	×	0.0	ReLU
Transposed Convolution	5 × 5	2 × 2	192	√	0.0	ReLU
Transposed Convolution	5 × 5	2 × 2	96	√	0.0	ReLU
Transposed Convolution	5 × 5	2 × 2	3	×	0.0	Tanh
$D(x)$ – 32 × 32 × 3 input						
Convolution	3 × 3	2 × 2	16	×	0.5	Leaky ReLU
Convolution	3 × 3	1 × 1	32	√	0.5	Leaky ReLU
Convolution	3 × 3	2 × 2	64	√	0.5	Leaky ReLU
Convolution	3 × 3	1 × 1	128	√	0.5	Leaky ReLU
Convolution	3 × 3	2 × 2	256	√	0.5	Leaky ReLU
Convolution	3 × 3	1 × 1	512	√	0.5	Leaky ReLU
Linear	N/A	N/A	11	×	0.0	Soft-Sigmoid
Generator Optimizer	Adam ($\alpha = [0.0001, 0.0002, 0.0003]$, $\beta_1 = 0.5$, $\beta_2 = 0.999$)					
Discriminator Optimizer	Adam ($\alpha = [0.0001, 0.0002, 0.0003]$, $\beta_1 = 0.5$, $\beta_2 = 0.999$)					
Batch size	100					
Iterations	50000					
Leaky ReLU slope	0.2					
Activation noise standard deviation	$[0, 0.1, 0.2]$					
Weight, bias initialization	Isotropic gaussian ($\mu = 0$, $\sigma = 0.02$), Constant(0)					

AC-GAN Generator and Discriminator Model Configuration Suggestions.
Take from: Conditional Image Synthesis With Auxiliary Classifier GANs.

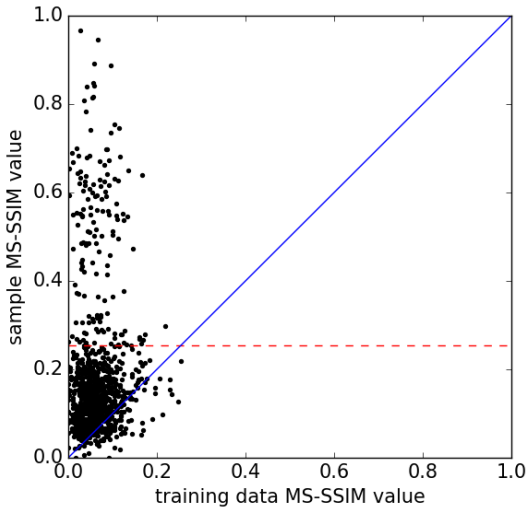
در زیر نشان داده شده است که تولید تصاویر با رزولوشن بالا، داوری را بهبود می دهد:



متد MS-SSIM : متدی برای اندازه گیری شباهت ادراکی است که سعی می کند قضاوت های ادراکی انسان را پیش بینی کند و تلاش می کند به جنبه هایی از عکس توجه کند که برای ادراک انسان مهم نبوده است. مقدار آن بین ۰ و ۱ تعریف می شود. هر چه مقدار شباهت تصاویر بیشتر باشد، مقدار بیشتری دریافت میکنند.

برای ایجاد تصاویر متنوع، یک سیاستی که اتخاذ می شود این است کهمقدار MS-SSIM بین ۱۰۰ تا از زوج تصویر انتخاب شده به صورت رندوم در کلاس داده شده، اندازه می گیرند. نمونه های کلاس ها که نتایج متنوعی در MS-SSIM میانگین پایین دارد و نمونه هایی از کلاس ها با تنوع پایین و میانگین امتیاز MS-SSIM بالا.داده های آموزش از دیتاست ImageNet که شامل تصاویر با میانین امتیاز MS-SSIM در کلاس های مختلف انتخاب شده است. بیشترین میزان امتیاز MS-SSIM برای داده های آموزش ۰,۲۵ بوده است.

برای همه ی ۱۰۰۰ داده ی تولید شده بوسیله ی generator مقدرا MS-SSIM آن را حساب کردیم تا از ناکارآمدی generator مطلع شویم. نمودار مقادیر به دست آمده هر زوج تصویر در کلاس داده شده برای داده های آموزش ImageNet و نمونه های GAN



از میان ۱۰۰۰ کلاس مطرح شده در مقاله، ۸۴۷ تای آن، مقدار میانگین امتیاز MS-SSIM آن ها کمتر از ماکزیمم مقدار MS-SSIM داده های آموزش هستند.

پیاده سازی شبکه:

معماری شبکه همانند آنچه که در بالا توضیح داده شده است، را پیاده سازی کردیم. پارامترهای مورد استفاده به صورت زیر است:

پارامترهای generator:

پارامتر	مقادیر
Num of transposed CNN	3
Input shape	Latent space =100, class size = 10
Activation function	RelU
Output :Dense layer	16
Embedding	10(class_size),50

پارامترهای discriminator:

پارامتر	مقادیر
Num of CNN	3
activation function	RelU
Dropout	0.5
Input shape	32,32,3
Output shape	1(real/fake), 10(class_size)
Activation function Output1: dense	sigmoid
Activation function Output2: dense	softmax
optimazer	adam
learning rate	0.0002
loss function	binary_crossentropy,sparse_categorical_crossentropy

پارامترهای GAN:

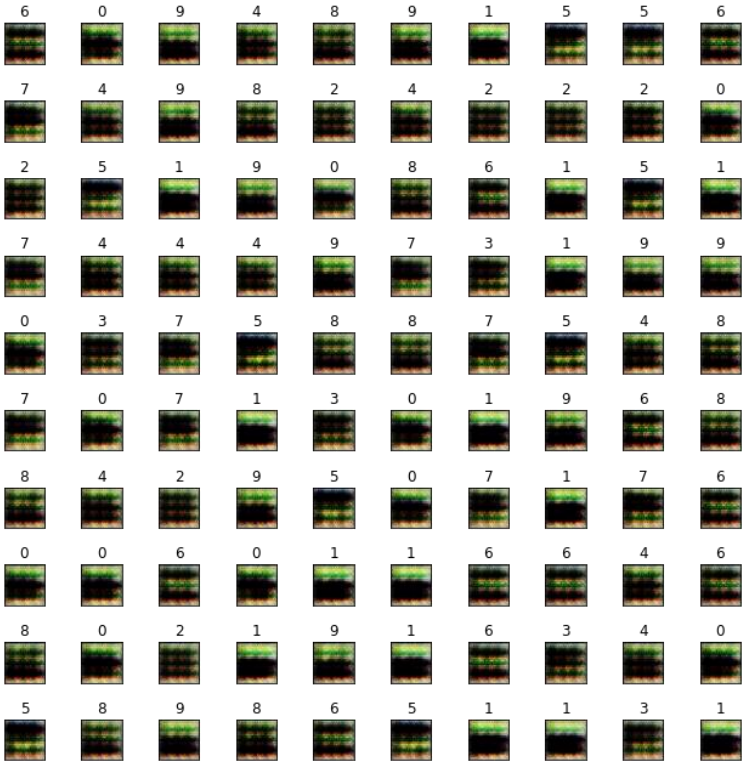
پارامتر	مقادیر
epoch	100

Batch size	64
Latent dim	100
optimazer	adam
learning rate	0.0002
loss function	binary_crossentropy,sparse_categorical_crossentropy

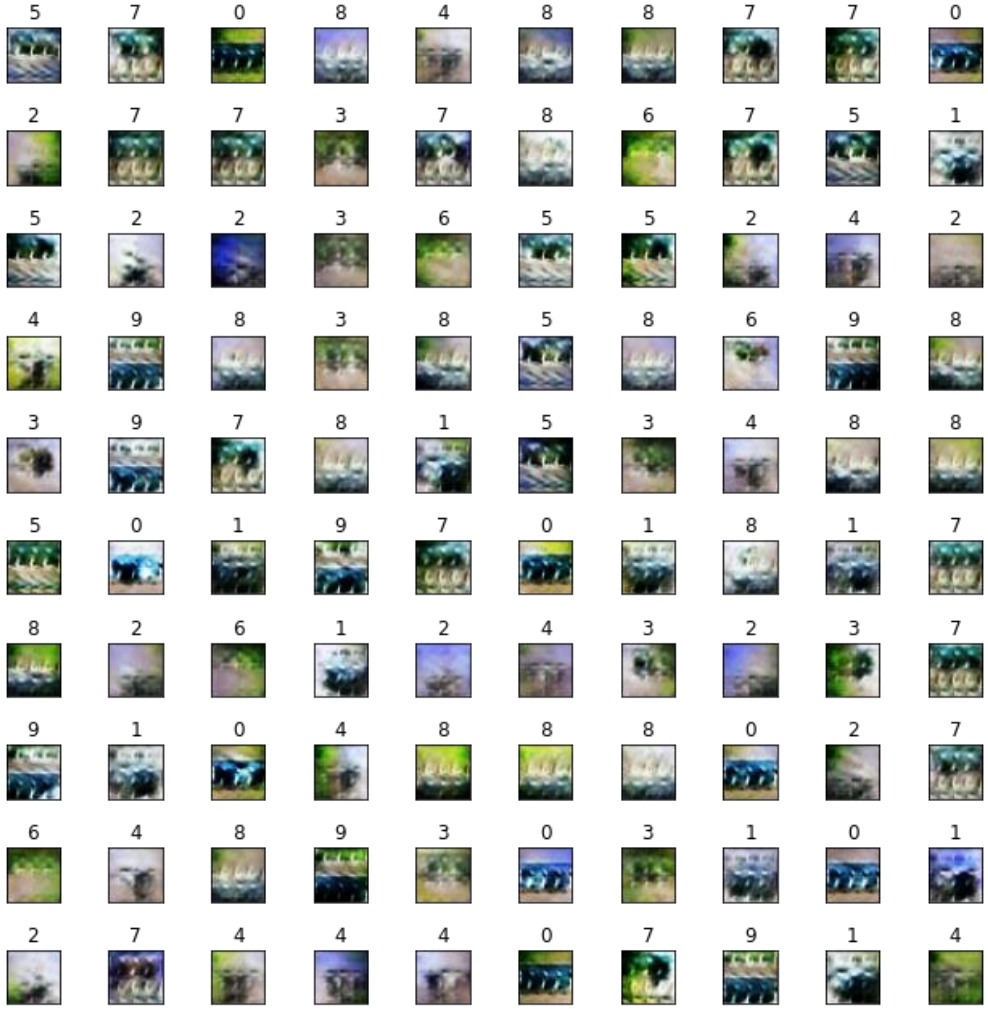
در هر ۱۰ اپاک تصاویر تولید شده توسط AC-GAN را نمایش دادیم:

در اپاک های اولیه، تصاویر به صورت نامفهوم بوده و کم کم با آموزش شبکه، تصاویر به تصاویر واضحی تری از اشیا موجود در CIFAR-10 نزدیک تر شده است. بالای هر تصویر شماره کلاس مربوطه اش را نمایش داده ایم.

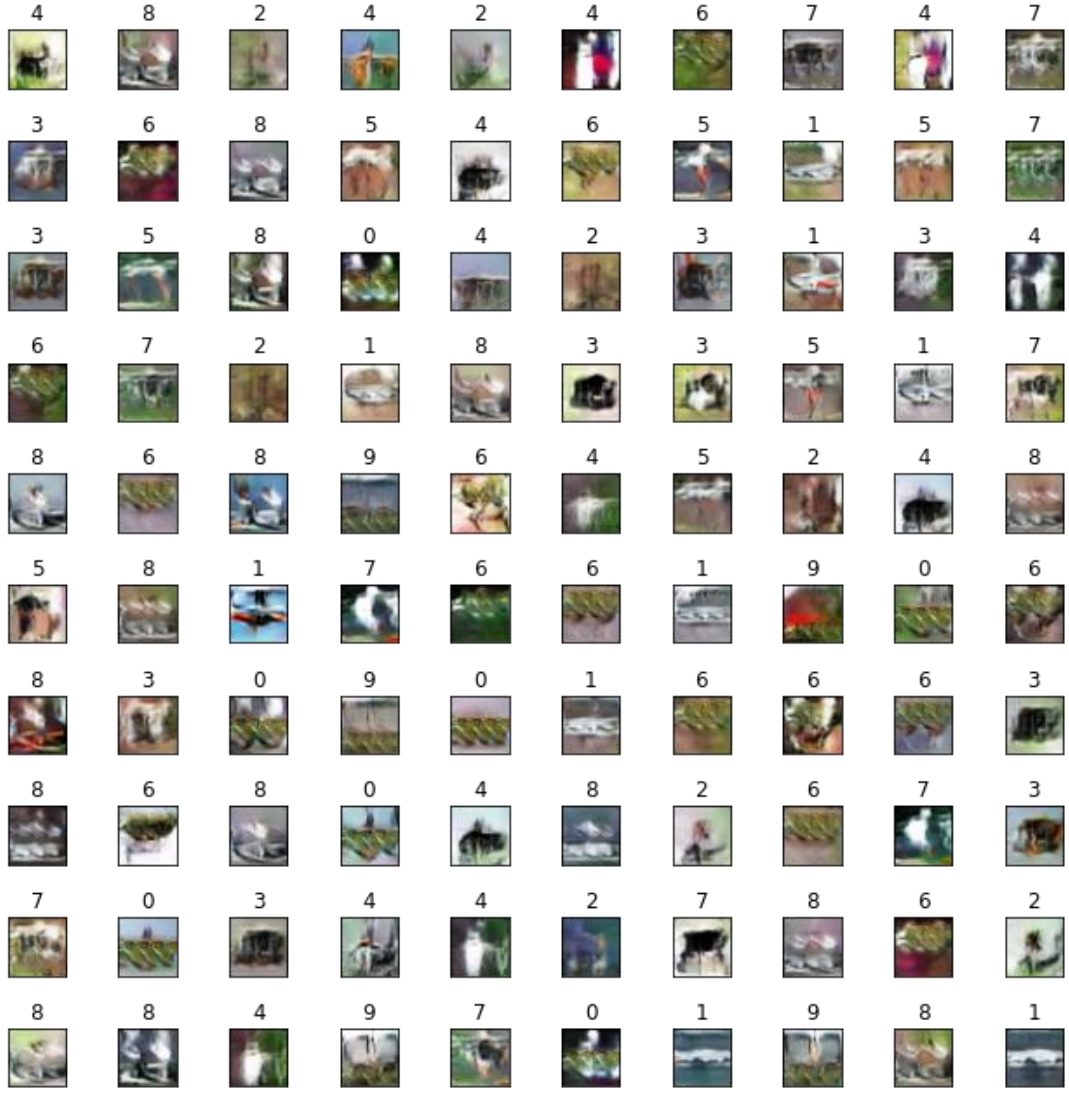
```
epoch = 1/100, d_loss=0.003,d_loss_label=1.208, g_loss=0.051,
g_loss_label=0.383
```



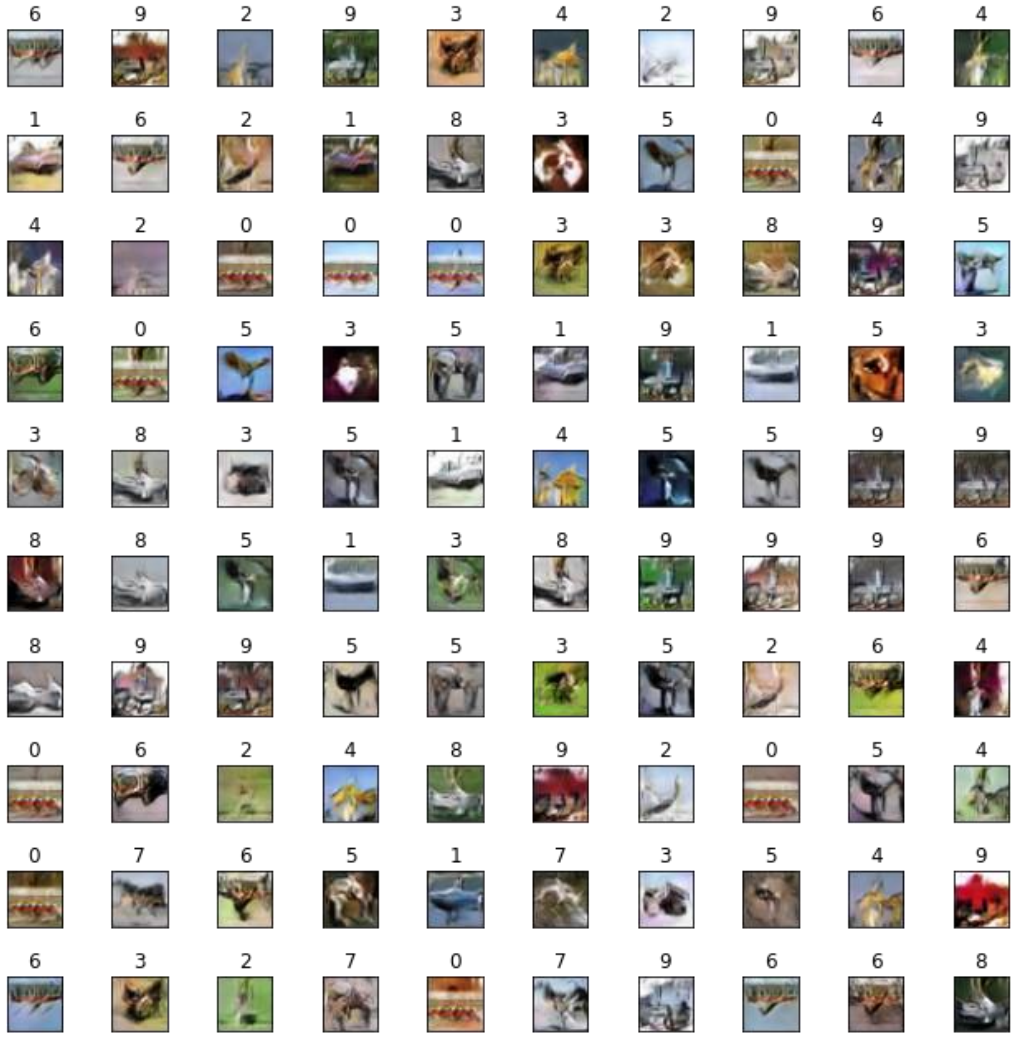
epoch = 11/100, d_loss=0.010,d_loss_label=0.757, g_loss=0.086,
g_loss_label=0.020



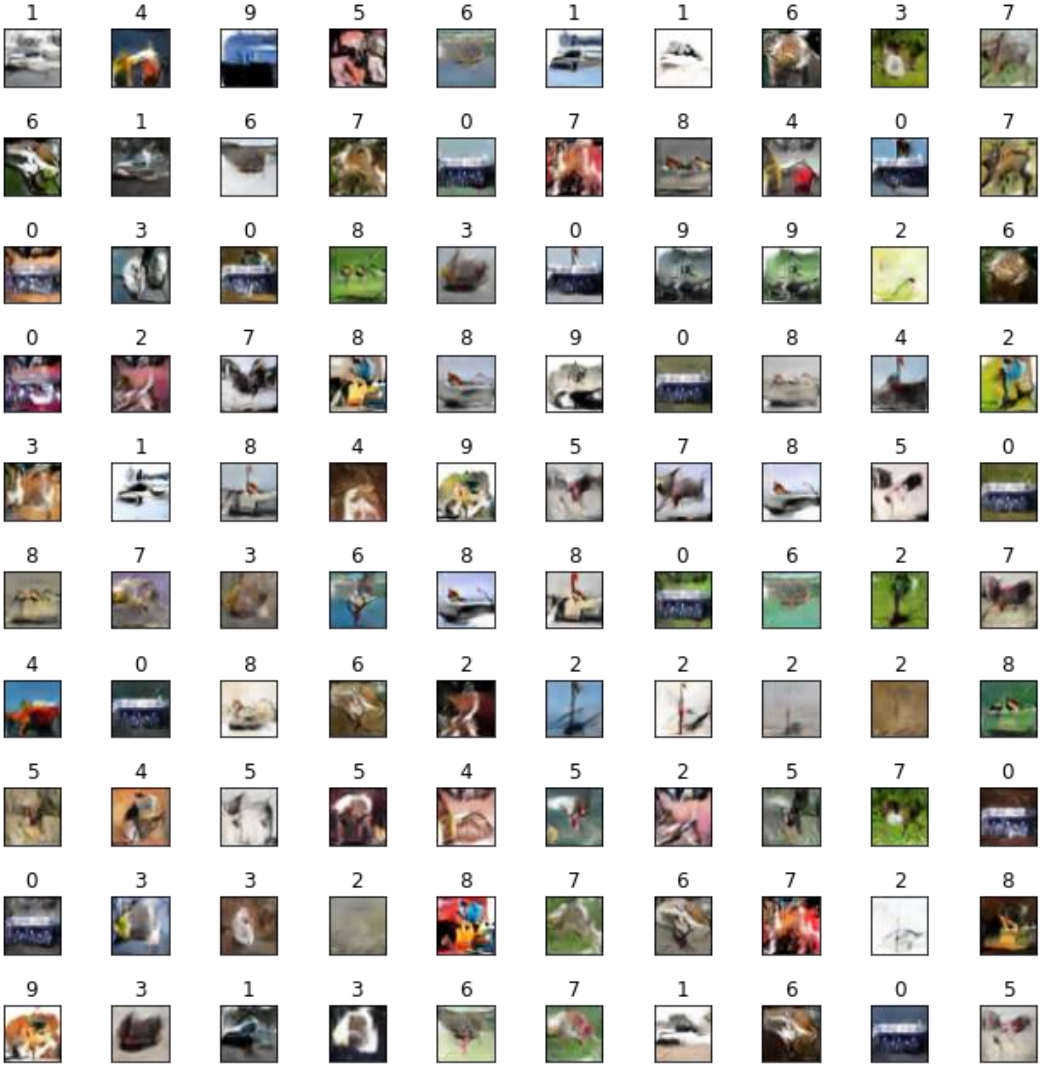
epoch = 41/100, d_loss=0.298,d_loss_label=0.252, g_loss=0.747,
g_loss_label=0.049



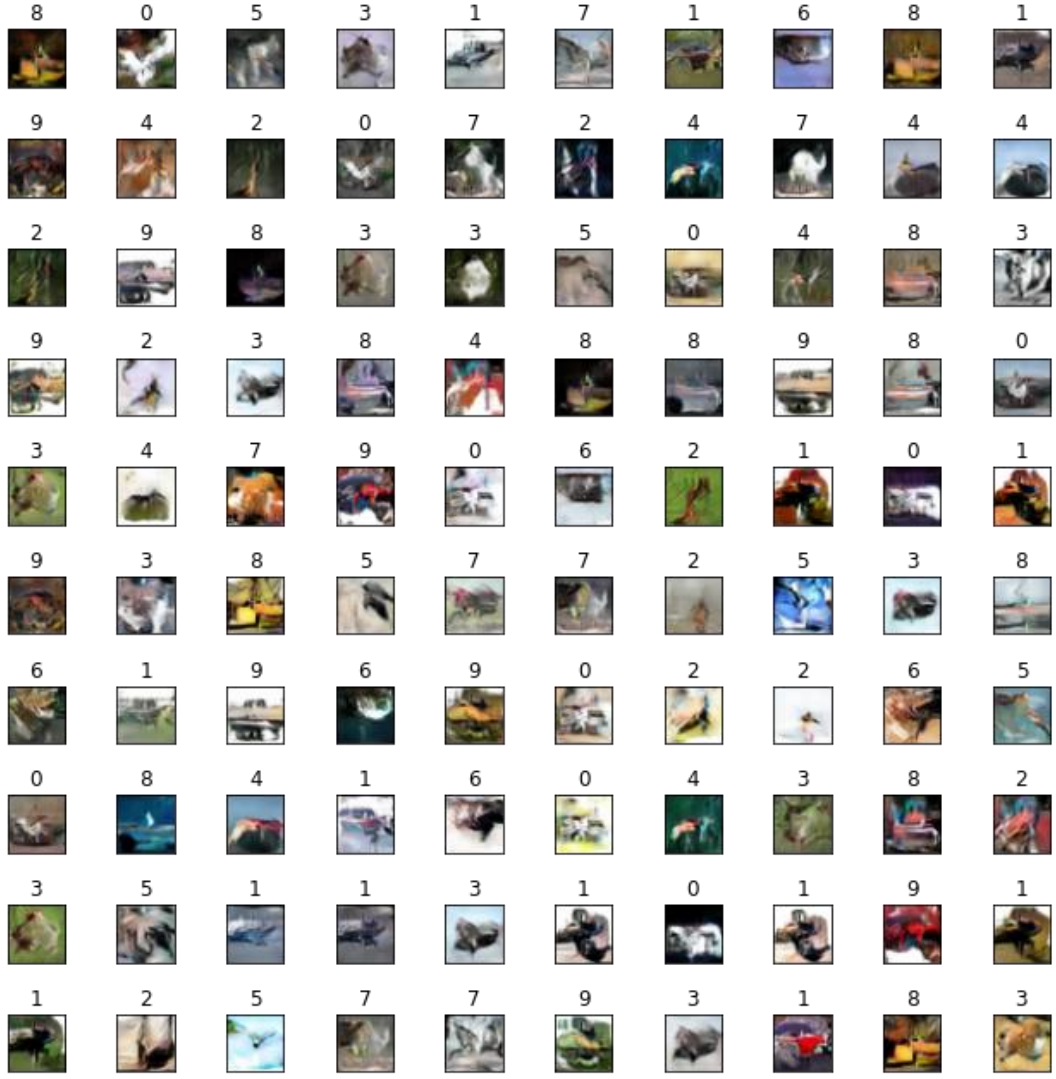
```
epoch = 61/100, d_loss=0.524,d_loss_label=0.455, g_loss=1.026,
g_loss_label=0.055
```



epoch = 91/100, d_loss=0.351,d_loss_label=0.290, g_loss=2.537,
g_loss_label=0.062

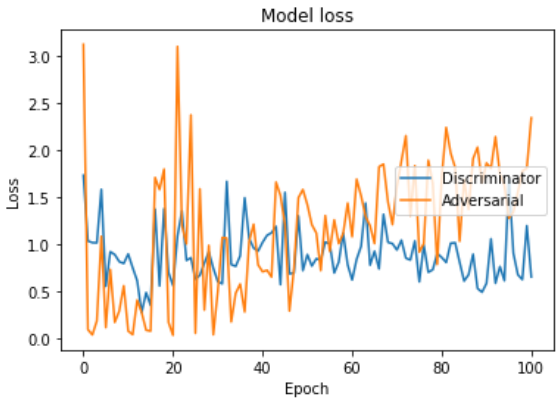


```
epoch = 101/100, d_loss=0.215,d_loss_label=0.489, g_loss=4.781,
g_loss_label=0.082
```

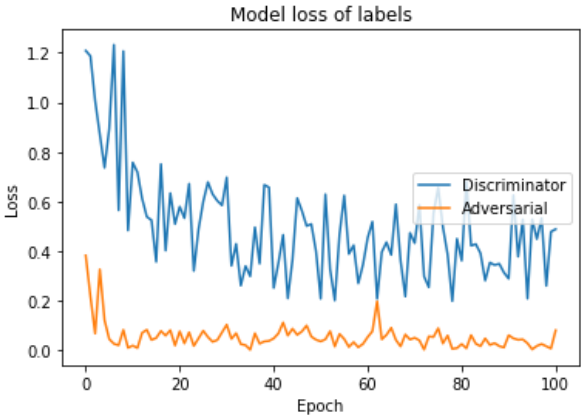


میزان خطای discriminator و generator را در هر اپاک به صورت نمودار نمایش داده ایم:

خطای تولید تصاویر فیک و تشخیص آن از تصاویر واقعی:



خطای لیبل کلاس ها (تشخیص و تولید):



همان طور که دیده می شود، بین discriminator و generator رقابت وجود دارد و ممکن است در ایپاکی هر کدام بهتر عمل کند؛ به طور کلی نمودار خطای آن ها دچار نوسانات شود. زیرا وقتی خطای generator کاهش پیدا میکند ، خطای discriminator افزایش پیدا می کند و بالعکس. اگر generator صعودی باشد و discriminator نزولی باشد، به معنای عملکرد ضعیف discriminator است و اگر discriminator صعودی باشد و generator نزولی باشد، به معنای عملکرد ضعیف generator خواهد بود و تصاویر تولیدی کیفیت پایینی خواهد داشت.

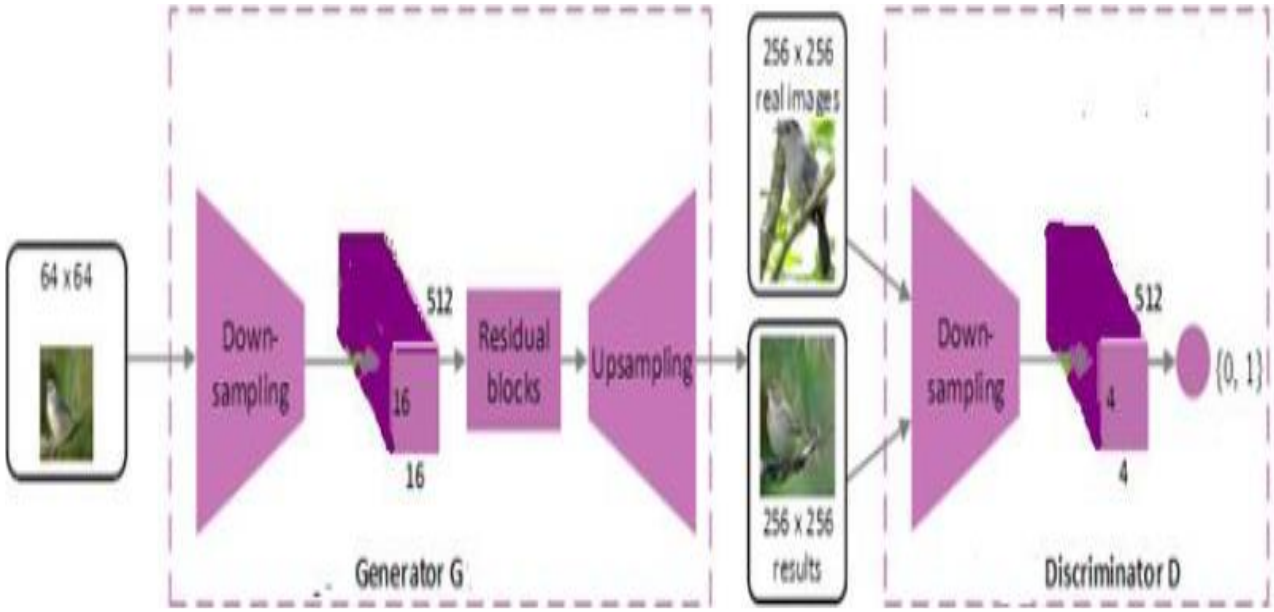
کدهای مربوط به این قسمت در فایل ACGAN.ipynb ذخیره شده است.

سوال ۴ –نمونه۳ SRGAN

الف)

شبکه super resolution GAN برای بالا بردن رزولوشن تصویر استفاده می شود. این شبکه از شبکه های توسعه یافته GAN ها هستند که خروجی آن تصویر با رزولوشن بالا هستند که کیفیت و شفافیت بهتری نسبت به الگوریتم های رقیب خود مانند Nearest Neighbor ایجاد می کند.

ساختار شبکه به صورت زیر است:

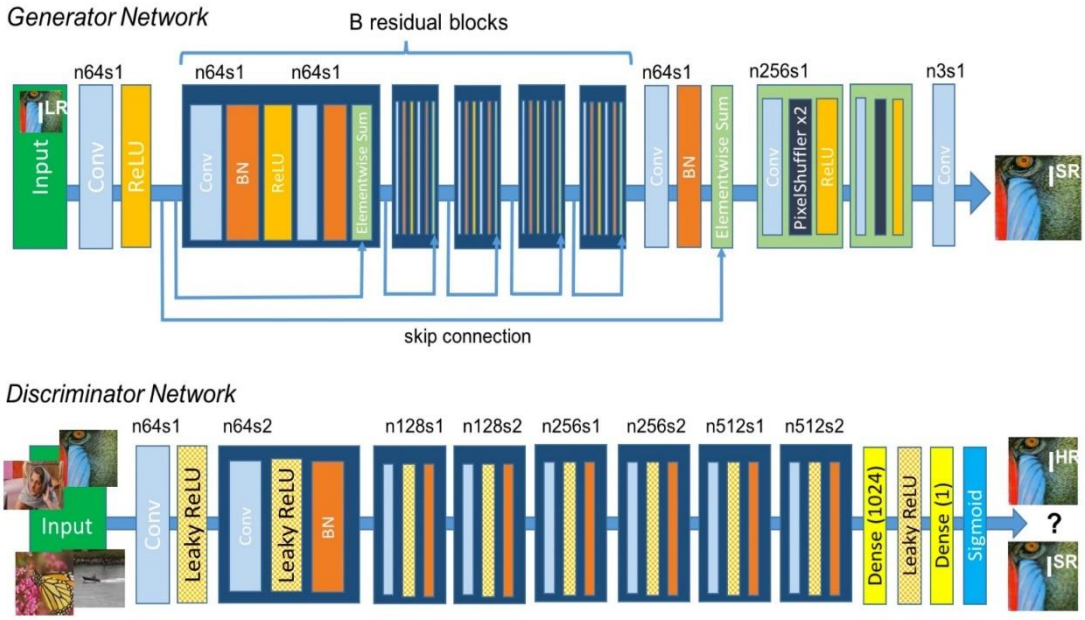


Discriminator مسئول تشخیص تصاویر تولید شده (fake) از تصاویر اصلی است و Generator مسئول تولید تصاویر high resolution از روی تصاویر با رزولوشن پایین است.

همان طور که در بالا دیده می شود، تصویر ورودی که همان تصویر low resolusion ماست، down sample می شود. حال اگر وارد بلاک های residual کنیم و از downsample عبور دهیم، تصویری با ابعاد تصویر واقعی مورد نظر ایجاد می کنیم (تصویر با رزولوشن بالا و فیک) که discriminator آن را با تصویر واقعی با همان ابعاد مقایسه می کند(با عبور از لایه های کانولوشنی و downsample کردن) در آخر خروجی یک کلاسیفایری است که می گوید آیا تصویر داده شده، واقعی است و یا فیک است (خروجی ۰ و ۱) اگر generator کارش را به خوبی انجام داده باشد، discriminator گول می خورد و تصویر تولیدی را تصویر واقعی تشخیص می دهد و اگر generator با میزان خطای بالایی تصویر را

تولید کرده باشد، discriminator ، آن را غیر واقعی تشخیص می دهد و خطایش کم می شود. در واقع یک بازی min – max ایجاد می شود که میزان خطای هر کدام از این دوشبکه بسته به نحوه ی عملکرد دیگری ، به صورت نوسانی بالا و پایین می رود.

تعداد و نوع لایه های discriminator و generator به صورت زیر است:



برای آموزش شبکه، ابتدا برای هر تصویر، یک تصویر با رزولوشن پایین برای ورودی generator و یک تصویر با رزولوشن بالا، برای مقایسه در discriminator ایجاد کرده ایم. هر تصویر دارای ۳ اندازه طول، عرض و تعداد کانال رنگ ها (سبز و قرمز و آبی) است.

شبکه سعی می کند با روش زیر یادگیری را انجام دهد و پارامترها و وزن هایش را با استفاده از بهینه کردن مقادیر خطای به دست آمده، آپدیت کند:

$$\hat{\theta}_G = \arg\min_{\theta_G} \frac{1}{N} \sum_{n=1}^{N} l^{SR}(G_{\theta_G}(I_n^{LR}), I_n^{HR})$$

I^{HR} تصویر با رزولوشن بالا و I^{HR} همان تصویر با رزولوشن پایین است. l^{SR} تابع خطایی است که سعی می شود، میان تصاویر تولید شده و تصاویر واقعی مقادیر کمتری پیدا کند.

در واقع شبکه سعی می کند مقدار زیر را بهینه کند:

discriminator: D و generator : G

$$\min_{\theta_G}\max_{\theta_D}\ \mathbb{E}_{I^{HR}\sim p_{train}(I^{HR})}[\log D_{\theta_D}(I^{HR})]+$$
$$\mathbb{E}_{I^{LR}\sim p_G(I^{LR})}[\log(1-D_{\theta_D}(G_{\theta_G}(I^{LR}))]$$

معیار خطای اندازه گیری شده در این شبکه ها معمولا MSE است.

Generator سعی می کند تصویر بارزولوشن بالا را به گونه ای طراحی کند که قابل تشخیص برای discriminator نباشد و نتواند تصویر ایجاد شده را از تصویر با رزولوشن بالای واقعی اش تشخیص دهد.

در generator از دو سری لایه های کانولوشن ها با کرنل های ۳*۳ و فیپرمپ های ۶۴ تایی به همراه لایه های batch normalization ایجاد شده است. تابع فعال ساز آن relu است. discriminator از ۸ لایه کانولوشنی بدون استفاده از maxpooling با افزایش کرنل ۳*۳ استفاده شده است. با stride کردن کانولوشن ها، رزولوشن تصاویر هر زمان که تعداد فیچرها دوبرابر شود، کاهش پیدا می کند. نتیجه ی ۵۱۲ فیچر مپ با دو لایه dense و تابع فعال ساز sigmoid جهت بهینه کردن احتمال کلاس بندی نمونه ها خواهد بود.

Perceptual loss function:

تابع loss برای بررسی عملکرد شبکه generator به صورت زیر تعریف می شود:

$$l^{SR} = \underbrace{\underbrace{l_{\mathrm{X}}^{SR}}_{\text{content loss}} + \underbrace{10^{-3}l_{Gen}^{SR}}_{\text{adversarial loss}}}_{\text{perceptual loss (for VGG based content losses)}}$$

Content loss:

خطای MSE به صورت pixel wise اینگونه تعریف می شود:

$$l_{MSE}^{SR} = \frac{1}{r^2WH}\sum_{x=1}^{rW}\sum_{y=1}^{rH}(I_{x,y}^{HR} - G_{\theta_G}(I^{LR})_{x,y})^2$$

می توان به جای تابع خطای تعریف شده که نزدیک به شباهت ادراکی است؛ از VGG loss براساس تابع فعال ساز Relu با شبکه از پیش آموزش دیده شده VGG با ۱۹ لایه کانولوشن استفاده کنیم.

تابع خطای استفاده شده برای تصویر ساخته شده با G و تصویر مرجع I^{HR} است.

$$l_{VGG/i.j}^{SR} = \frac{1}{W_{i,j}H_{i,j}}\sum_{x=1}^{W_{i,j}}\sum_{y=1}^{H_{i,j}}(\phi_{i,j}(I^{HR})_{x,y}$$
$$-\phi_{i,j}(G_{\theta_G}(I^{LR}))_{x,y})^2$$

$\phi_{i,j}$ فیچر مپ به دست آمده با کانولوشن j ام قبل از i امین لایه *maxpooling* در شبکه *VGG19* است.

$W_{i,j}\ and\ H_{i,j}$ ابعاد فیچر مپ های داخل شبکه *VGG* را نشان می دهد.

تابع خطایی که برای *generator* استفاده می شود، براساس احتمالات ساخته شده است و به صورت زیر است:

$$l_{Gen}^{SR} = \sum_{n=1}^{N} -\log D_{\theta_D}(G_{\theta_G}(I^{LR}))$$

$D(G(L^{LR}))$ احتمال این است که تصویر ساخته شده برابر تصویر اصلی *HR* می شود.

در این مقاله، برای اجرای شیکه از سه دیتاست استفاده کرده است و از مجموعه *BSD300* برای تست آن بکار می گیرد. تصویر با رزولوشن بالا، ۴ برابر بهتر از همان تصویر در رزولوشن پایینش است. تصاویر *low resolution* را می توان با *downsample* کردن تصویر*HR* به دست آورد. برای *mini batch*، ۱۶ تا از تصاویر با رزولوشن بالا ۹۶*۹۶ را به صورت رندوم *crop* می کنیم.

برای تشخیص میزان کیفیت شبکه در ایجاد تصاویر با رزولوشن بالا از تست Mean opinion score استفاده می کنیم که از رنج ۱ (کیفیت پایین) تا رنج ۵ (کیفیت بالا) امتیاز دهی شده است. این امتیاز دهنده روی bicubic, Set5, Set14 and BSD100: nearest neighbor (NN), SRCNN ,SelfExSR ,DRCN, ESPCN, SRResNet-MSE, SRResNet-VGG22_ (_not rated on BSD100), SRGAN-MSE_, SRGAN-VGG22_, SRGANVGG54 و تصاویر HR متیاز دهی شده است. بررسی ها نشان می دهد که reliability خوبی داشته و تفاوت زیادی میان امتیاز ها و تصاویر شناخته شده، دیده نمی شود. نمونه ای از امتیاز دهی ها به صورت زیر است:

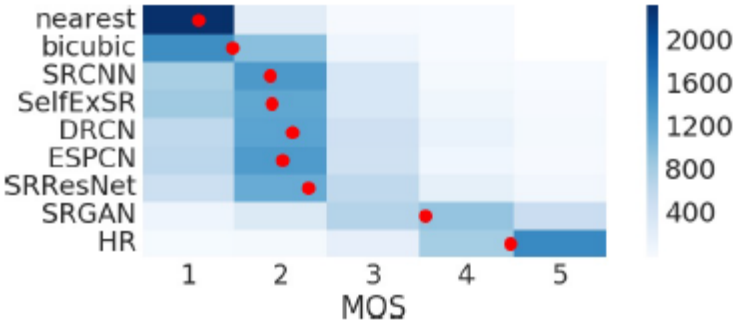


Figure 5: Color-coded distribution of MOS scores on **BSD100**. For each method 2600 samples (100 images × 26 raters) were assessed. Mean shown as red marker, where the bins are centered around value i. [4× upscaling]

ب)

برای انجام این پروژه از دو روش استفاده کردیم. روش اول، خود شبکه را با استناد به مقاله پیاده سازی و اجرا کردیم. در روش دوم از مدل پیش آموزش دیده استفاده کردیم. فایل وزن های آن در لینک

https://drive.google.com/open?id=1u9ituA3ScttN9Vi-UkALmpO0dWQLm8Rv

موجود است.

در روش اول لایه ها، داده ها، پارامترهای شبکه آن طور با استناد به مقاله، پیاده سازی شد که به علت حجم بالای آن ها و محاسبات پیچیده، از توانایی سیستم در اجرای آن، ساقط شده و با ارور OOM روبرو می شود. کدهای این قسمت در فایل SRGAN-implement.ipynb موجود است.

در روش دوم، از VGG54 با ۱,۵۵ میلیون پارامتر استفاده کردیم. در این روش آموزش شبکه SR-GAN قبلا انجام شده است و ما صرفا با استفاده از وزن های داده شده، شبکه خود را با استفاده از آن ها fine tune کرده و داده های DIV2K را آموزش دادیم. سپس با دادن تصویر با رزولوشن پایین، تصویر را با رزولوشن بالا دریافت کنیم. این روش کیفیت بهتری نسبت به روش قبل دارد؛ چرا که شبکه قبلا با حجم زیادی از داده ها، آموزش دیده و دقتی مشابه دقت مطرح شده در مقاله را ایجاد کرده است.

می توان با استفاده از تابع تعریف شده در train.py شبکه را با داده های موردنظر خودمان آموزش بدهیم و از وزن های ایجاد شده استفاده کنیم. و نیز می توان با استفاده از ایمپورت کردن فایل model.srgan در برنامه، یک generator() و یک discriminator() ایجاد کنیم و وزن های ذخیره شده در

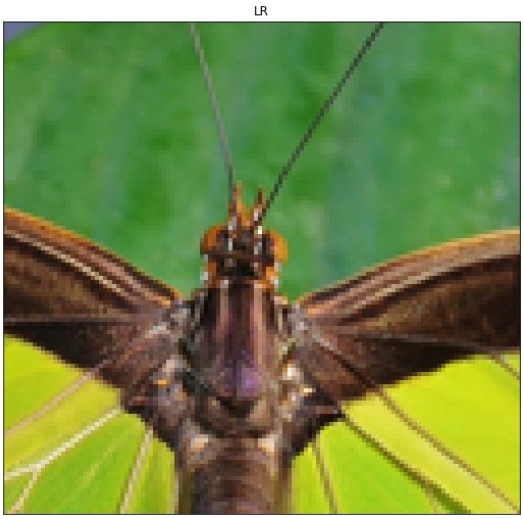
pre_generator.h5 را برای مدل generator بارگذاری کرد. با تابع SrganTrainer() که در تابع train.py تعریف شده است. شبکه GAN حاصل از ترکیب discriminator و generator ایجاد می کنیم. حال این شبکه را با داده های خود آموزش دهیم و به نحوی براساس آن ها fine tune می کنیم و وزن های tune شده ی آن را ذخیره و مورد استفاده قرار دهیم.

وزن های ذخیره شده در pre_generator.h5 و gan_generator.h5 به ترتیب به عنوان وزن های pre_generator و وزن های gan_generator مورد استفاده قرار می گیرند و با وارد کردن تصویری با رزولوشن پایین، می توان تصویر با رزولوشن بالای قبلی و همین تصویر ایجاد شده با gan مورد نظر را به دست آورد.

کد آن در فایل SRGAN-pretrained.ipynb موجود است.

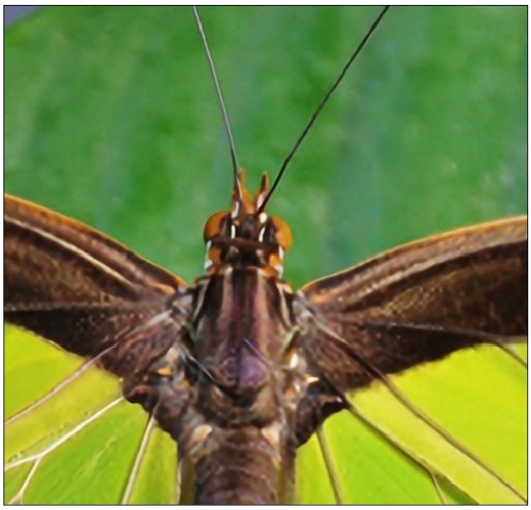
نمونه ای از خروجی SR-GAN:

تصویر با رزولوشن پایین:



تصویر اصلی با رزولوشن بالا:

SR (PRE)



تصویر تولید شده :

SR (GAN)