

به نام خدا



دانشگاه تهران

پردیس دانشکده‌های فنی

دانشکده برق و کامپیوتر



مینی پروژه سوم شبکه های عصبی

نام و نام خانوادگی:

يلدا فروتن

محمد دهقان روزي

شماره دانشجویی:

810196265

810197243

زمستان ۱۳۹۸

چکیده

هدف از مینی پروژه سوم، بررسی مفاهیم و معماری های مربوط به شبکه های Generative Adversarial Networks یا GANs است؛ به گونه ای که مفاهیم مربوط به معماری های مختلف GAN و Varitional Auto Encoder بررسی و معماری هایی همچون ACGAN و DCGAN و WGAN و پیاده سازی می شوند. در این پروژه از دو دیتاست CIFAR10 و Fashion MNIST استفاده شده است.

سوال 1

الف) شبکه SAGAN

استفاده از شبکه های SAGAN یا همان Self Attention GAN برای تولید عکس هایی با Long dependencies کمک کند. چون لایه های کانولوشنی به صورت لوکال هستند، بنابراین می بایست به تعداد زیادی لایه کانولوشنی پشت سرهم قرار بگیرد تا ایجاد long dependencies کند. افزایش سایز کانولوشن ممکن است capacity نتورک را افزایش دهد، ولی باعث از دست رفتن Statistical efficiency و Computation efficiency می شود. این درحالی است که SAGAN با داشتن محاسبات بسیار کمتر، در زمان مناسب با دقت مناسب long dependencies ایجاد خواهد کرد.

برای محاسبه attention به این صورت عمل کرده اند که در ابتدا image features از لایه مخفی قبل وارد دو فضای ویژگی f و g می شوند تا آن ها محاسبه شود:

$$f(x) = W_f x;$$

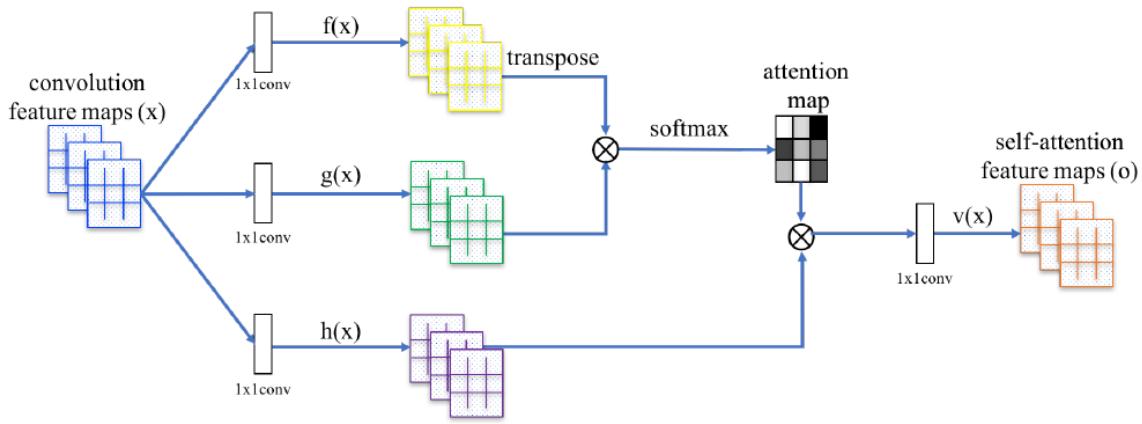
$$g(x) = W_g x;$$

$$\beta_{j,i} = \frac{\exp(s_{ij})}{\sum_{i=1}^N \exp(s_{ij})}, \text{ where } s_{ij} = f(x_i)^T g(x_j), \quad (1)$$

و سپس خروجی لایه attention محاسبه می شود:

$$o_j = v \left(\sum_{i=1}^N \beta_{j,i} h(x_i) \right), \quad h(x_i) = W_h x_i, \quad v(x_i) = W_v x_i. \quad (2)$$

به صورت کلی نیز به صورت شماتیکی به صورت شکل زیر می باشد:



ب) تفاوت Variational Autoencoder و Autoencoder

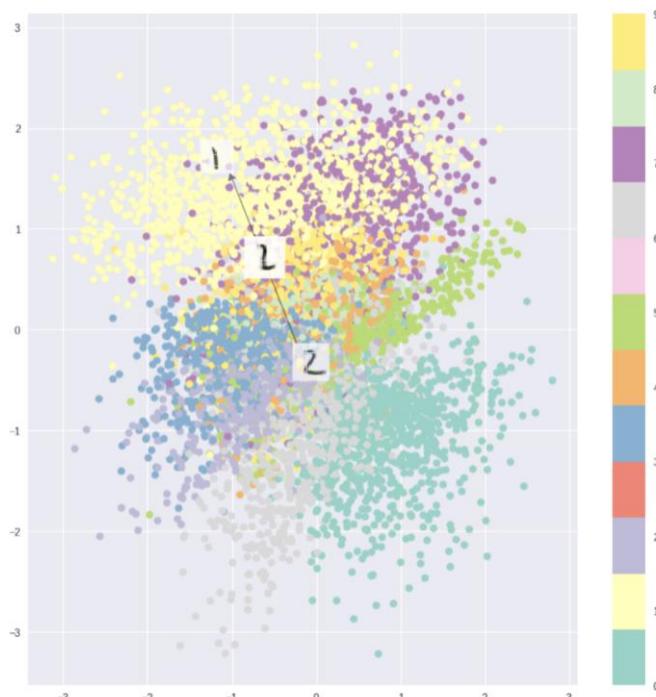
استفاده از آتوانکودر در راستای کاهش ابعاد داده‌های ورودی است. آتوانکودر دارای دو بخش انکودر و دیکودر است. بخش انکودر وظیفه down-sampling و حذف correlation های بین ابعاد را داشته و داده‌ها را به فضا latent می‌برد. از طرفی دیکودر وظیفه up-sampling دارد و داده‌ها را از فضا latent به ابعاد اولیه آن‌ها می‌برد. در هر ایپاک خروجی آتوانکودر با ورودی اولیه متناظر مقایسه و خطا شبکه محاسبه می‌گردد. بنابراین انکودر داده‌ها را به فضای latent برده و دیکودر اطمینان می‌دهد که اطلاعات اصلی همان داده‌ها را بازیابی می‌کند.

یکی از اهداف مدل‌های تولیدکننده آن است که خروجی جدید تولید شود به‌گونه‌ای که شبیه به داده‌های یادگیری باشند. بدیهی است که آتوانکودر توانایی تولید داده‌های جدید را نخواهد داشت. مشکل اصلی آتوانکودر برای تولید داده‌های جدید آن است که فضای latent داده‌ها را به صورت گستته می‌سازد و امکان درون‌یابی وجود نخواهد داشت. بنابراین اگر هدف بازیابی همان اطلاعات اولیه باشد آتوانکودرها مناسب هستند؛ اما اگر هدف تولید داده‌های جدید و ساخت یک شبکه generative است دیگر فضای گستته‌ای که آتوانکودر می‌سازد مناسب نخواهد بود. زیرا ممکن است به صورت رندوم یک نقطه از فضا latent برداشته شود که به هیچ دسته‌ای مربوط نباشد، بنابراین دیکودر نمونه‌ای بی‌معنی تولید خواهد کرد. در این حالت استفاده از Variational Autoencoder (VAE) یا به اختصار VAE پیشنهاد می‌شود؛ چراکه فضای latent آن‌ها پیوسته بوده و می‌توان به صورت رندوم نمونه‌برداری کرد و درون‌یابی کرد. اگر فضای latent قرار است n بعدی باشد، VAE دو بردار برای میانگین و انحراف معیار هریک به ابعاد n می‌سازد. پس از ترکیب این بردار در فضای latent یک توزیع احتمالاتی پدید می‌آورد میانگین نشان دهنده نقطه در فضا و انحراف معیار محدوده اطراف آن نقطه خواهد بود. بنابراین دیکودر یاد می‌گیرد که نه تنها هر نقطه در فضا latent به یک نمونه مشخص تبدیل می‌شود، بلکه نقاط اطراف آن نیز به نمونه‌های معنی‌دار تبدیل خواهند شد. البته ایده‌آل آن است که کلاس‌های مختلف در فضای latent، علاوه بر قابل تفکیک بودن، با یکدیگر همپوشانی نیز داشته باشند تا درون‌یابی معنی پیدا کند. بنابراین لازم است که توزیع به صورت متمرکز

باشد. در این حالت از مقیاس Kullback-Leibler Divergence در تابع هزینه استفاده می‌شود تا میزان واگرایی هر کلاس از کلاس دیگر محاسبه گردد. بنابراین با کاهش KL divergence پارامترهای میانگین و انحراف معیار بهینه می‌شوند تا داده‌های مختلف در محدوده توزیع هدف قرار بگیرند. در ادامه رابطه KL loss آمده است.

$$\sum_{i=1}^n \sigma_i^2 + \mu_i^2 - \log(\sigma_i) - 1$$

بنابراین فضا latent ساخته شده توسط یک آتوانکودر و استفاده از تابع هزینه و KL loss به صورت زیر خواهد بود.

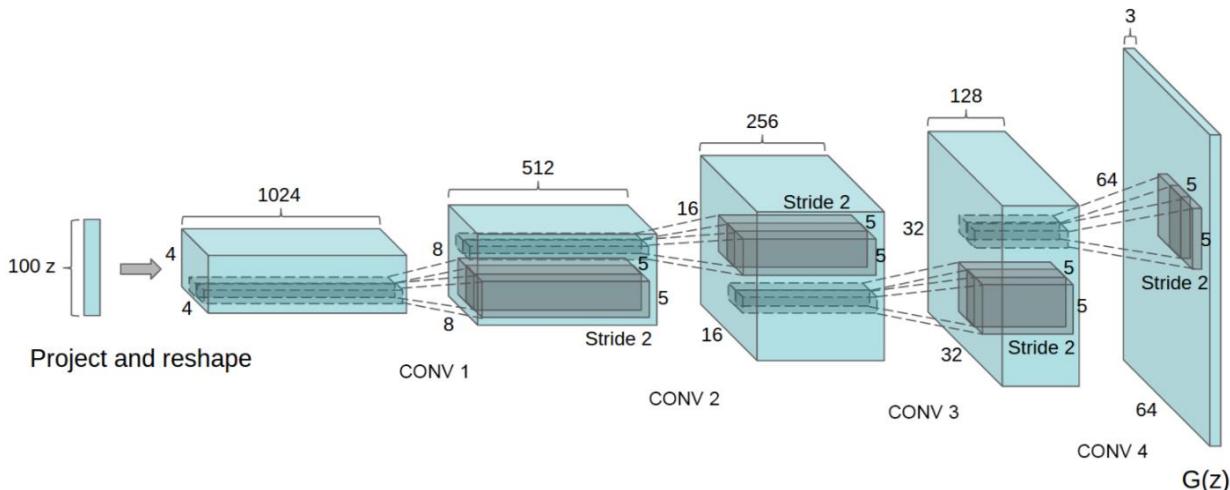


شكل – فضای latent ساخته شده توسط VAE

سوال 2 DCGAN

الف) شبکه DCGAN

نوعی GAN است که از لایه‌های کانولوشنی برای تولید تصویر استفاده می‌کند. همانند DCGAN دارای دو بخش generator و discriminator است که هر بخش از ۴ لایه به ترتیب deconv و conv استفاده می‌کند. درواقع DCGAN سعی دارد از دل نویز، تصویر تولید کند. بنابراین یک نویز با توزیع یکسان با ابعاد ۱۰۰ وارد generator می‌شود. در ابتدا نویز reform و به فرم یک تنسور با ابعاد $1024 \times 4 \times 4$ می‌شود. حال اگر در ابتدا از نویزی با ابعاد $1024 \times 4 \times 4$ استفاده می‌کرد، تنوع بالا رفته و احتمال همگرایی کم می‌شد. در نهایت generator با up-sampling، به تصاویری با ابعاد $64 \times 64 \times 3$ درآمده و با برچسب down- discriminator به داده می‌شوند تا با نمونه‌های واقعی قیاس شوند. در عمل fake sampling صورت گرفته و ویژگی‌های نمونه‌های واقعی و ساخته شده را در فضای latent برده و طبقه‌بندی می‌کند. در ادامه ساختار generator آمده است.



شکل - بخش generator مربوط به ساختار DCGAN

- استفاده از Strided Convolution

در شبکه‌های کانولوشنی، با فرض استفاده از max-pooling، padding و یا mean-pooling هستند که وظیفه down-sampling یعنی کاهش ابعاد spatial و افزایش عمق نمونه‌ها را دارند. از طرفی میتوان لایه‌های convolution را به صورت همان deconvolution فرض کرد، با این تفاوت که عمل up-sampling باید اتفاق بیفت؛ یعنی از عمق نمونه‌ها کم شود و به spatial آنها افزوده شود. حال در ساختار DCGAN از لایه‌های deconv برای generator و لایه‌های conv برای discriminator استفاده می‌شود. در DCGAN وظیفه up/down-sampling از لایه‌های pooling گرفته شده و با stride مربوط به لایه‌های conv کاهش یا افزایش ابعاد spatial صورت می‌گیرد. در discriminatior، لایه‌های کانولوشنی با stride بزرگتر از یک، down-sample می‌شوند. در generator نیز از لایه‌های conv استفاده

می‌شود، اما اینبار stride مقدار کسری خواهد داشت؛ بدین دلیل است که از این لایه‌ها به عنوان deconv نام برده می‌شود زیرا منجر به افزایش ابعاد spatial و کاهش عمق می‌شوند.

- استفاده از Batch Normalization

با اعمال batch normalization، نمونه‌ها در هر بعد دارای میانگین صفر و یک واریانس مشخص خواهند بود. استفاده از batch normalization در راستای مقابله با مشکلات یادگیری که به دلیل initialization بد اتفاق می‌افتد، است. همچنین batch normalization به گرادیان کمک می‌کند تا یادگیری عمیق‌تری را بسازد. بنابراین استفاده از این تکنیک منجر می‌شود که generator در یک local optimum گیر نکند و متنوع‌تری داشته باشد. هرچند که اعمال batch normalization به همه لایه‌ها منجر به نوسان و کاهش پایداری می‌شود. بنابراین از batch normalization در لایه آخر generator و discriminator استفاده نمی‌شود.

- استفاده از تابع فعال‌ساز Leaky ReLU

در لایه‌های generator، به جز لایه آخر، از تابع فعال‌ساز ReLU استفاده می‌شود. استفاده از یک تابع فعال‌ساز با برد محدود، منجر به همگرایی سریعتر یادگیری می‌شود. بدین جهت از تابع فعال‌ساز Tanh در لایه آخر generator استفاده شده است. در discriminator از تابع فعال‌ساز Leaky ReLU استفاده شده است، چرا که برای مدل‌های خصوصاً رزوشن بالا، به خوبی عمل می‌کند. این در حالی اسن که در GAN استاندارد از تابع maxout استفاده شده است. لازم به ذکر است، ضریب نشت تابع leaky ReLU برای مقادیر زیر صفر، 0.2 است.

- استفاده از بهینه‌ساز ADAM

در کارهای گذشته بر GAN از بهینه‌ساز momentum به دلیل ایجاد سرعت بالای یادگیری استفاده شده است. اما در DCGAN از بهینه‌ساز ADAM و با امکان تنظیم کردن پارامترها استفاده شده است. نرخ یادگیری 0.001 بزرگ بوده و از نرخ یادگیری 0.0002 استفاده شده است. همچنین ترم β_1 در momentum منجر به نوسان و ناپایداری می‌شود؛ در حالی که با کاهش آن، $\beta_1=0.5$ پایداری یادگیری را بهبود می‌بخشد.

ب) پیاده‌سازی DCGAN

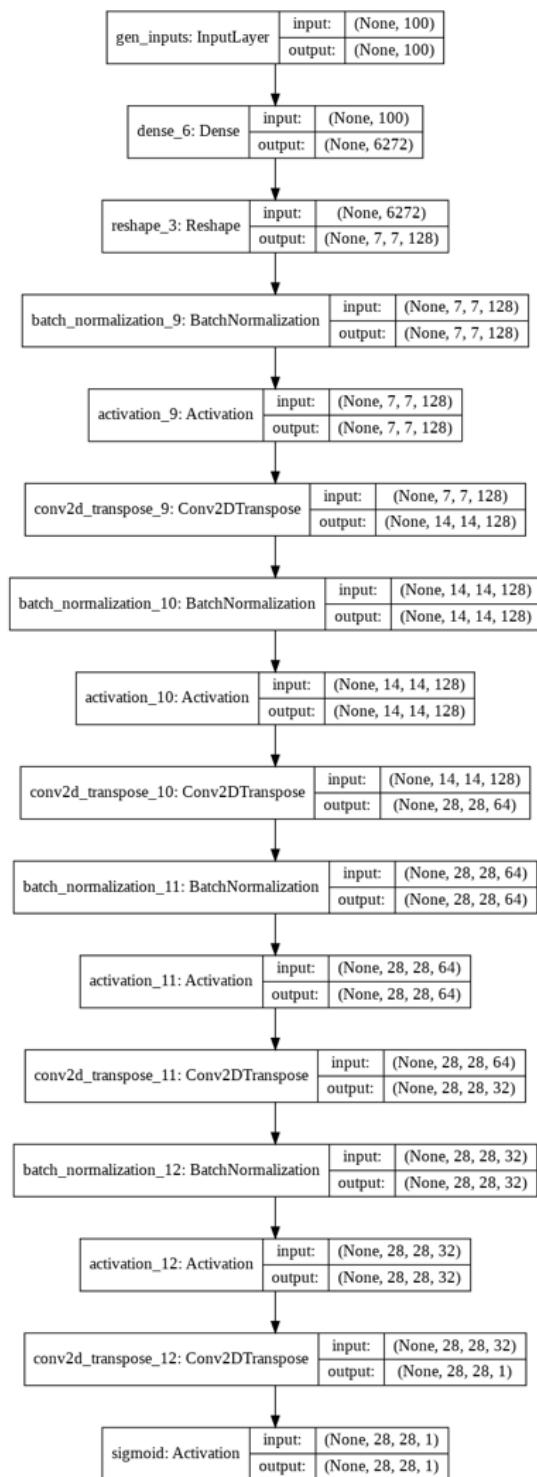
در این بخش یک شبکه DCGAN بر روی دیتاست‌های Fashion MNIST و Cifar10 پیاده‌سازی می‌شود. در ابتدا بخش‌های مختلف شبکه برای دیتاست Fashion MNIST توضیح داده شده است.

Fashion MNIST ♦

در ابتداء دیتاست Fashion MNIST از فریمورک keras فراخوانی شده است. این دیتاست شامل ۶۰۰۰۰ عکس با ابعاد ۲۸ در ۲۸ برای داده‌های train است. لازم به ذکر است که این داده‌ها grayscale بوده و ابعاد هر یک به صورت $1 \times 28 \times 28$ است. هر پیکسل دارای مقادیری بین ۰ تا ۲۵۵ است که در این بخش بین ۱ تا ۱ نرمالایز شده‌اند. این کار با تقسیم بر ۱۲۷,۵ و منهای ۱ کردن اعمال شده است.

- تپولوزی لایه‌های مولد و تفکیک‌کننده

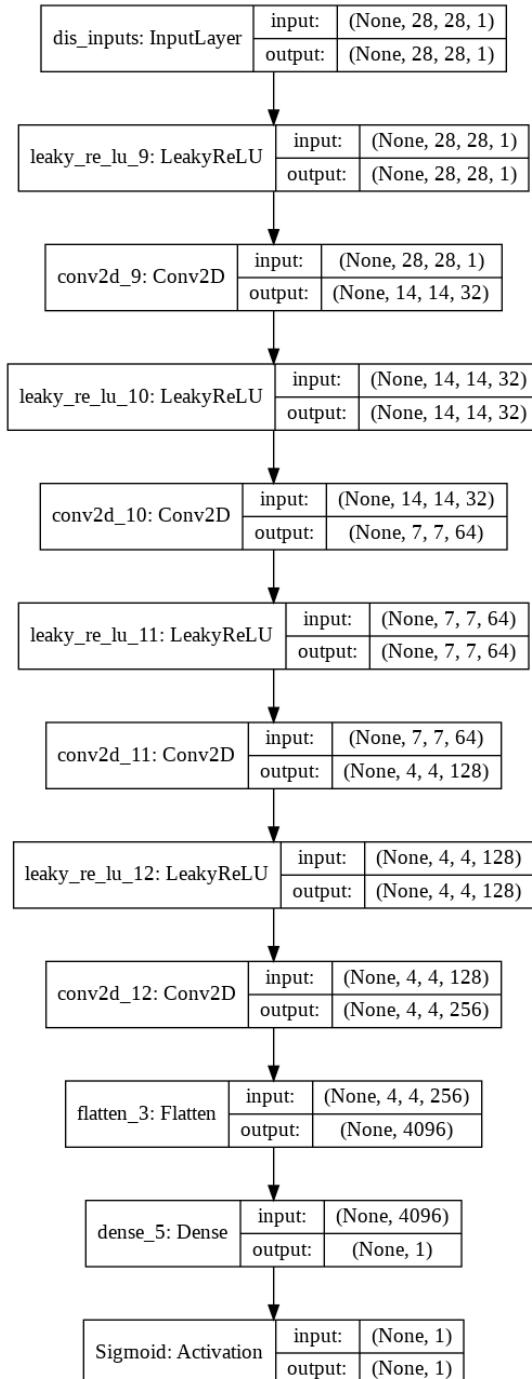
همانطور که پیشتر گفته شد، شبکه‌های GAN دارای دو بخش generator و discriminator هستند. بنابراین در ابتداء لازم است این دو بخش طراحی گردند. در ابتداء بخش generator طراحی گردید. ورودی generator درواقع یک نویز با ابعاد 100×1 است که به ابعاد $128 \times 128 \times 7 \times 7$ تبدیل می‌گردد. اگر از همان ابتداء نویزی با ابعاد $128 \times 7 \times 7$ تولید می‌شود، نویز سخت می‌گردد. بنابراین در لایه اول generator، نویز reshape می‌گردد و به ابعاد $7,7,128$ تبدیل می‌شود. پس 4×4 بار عمل batchNormalization، ReLU تابع فعال‌ساز و یک لایه Conv2DTranspose اعمال شده است. از آنجایی که هدف generator، up-sampling است، از تعداد کانال یا عمق تصاویر کم کرده و به فضای spatial آنها اضافه می‌کند. به همین دلیل از لایه Conv2DTranspose استفاده شده است، چراکه همچون deconvolution عمل می‌کند. قرار داده padding و استفاده از $\text{stride} = (2,2)$ ابعاد تصاویر را دقیقاً دو برابر می‌کند به گونه‌ای که ابعاد نویز reshape شده، از $128 \times 7 \times 7$ به $28 \times 28 \times 1$ می‌رسند که معادل ابعاد تصاویر fashion mnist است. درنهایت بر خروجی لایه deconv چهارم تابع فعال‌ساز sigmoid اعمال شده است. درادامه لایه‌های generator و ورودی و خروجی آن مشخص شده است. درواقع یک نویز با ابعاد 100×1 گرفته و یک تصویر به ابعاد تصاویر fashion mnist یعنی $1 \times 28 \times 28$ تولید می‌کند.



شکل – Fashin MNIST طراحی شده برای دیتاست generator

در ادامه نحوه طراحی discriminator توضیح داده شده است. از آنجایی که هدف discriminator است، از چهار لایه‌های کانولوشنی با تابع فعال ساز LeakyReLU استفاده شده است. از آنجایی که discriminator تصاویر real و fake را می‌گیرد، ابعاد ورودی 28×28 است. همچنین خروجی

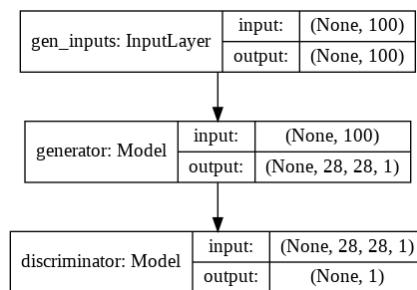
یک discriminator می‌شود که پس از binary شدن به یه لایه flat با یک نورون داده می‌شود. از آنجایی که لایه آخر عمل classifier را انجام می‌دهد، تابع فعال‌ساز لایه آخر، سیگموید است. در نهایت مدل discriminator است.



شکل – طراحی شده discriminator

حال به discriminator باید ورودی‌های مدنظر یا همان تصاویر واقعی و ساخته شده اعمال گردد. به generator نیز باید نویز با ابعاد ۱۰۰ اعمال گردد. لازم به ذکر است که در هنگام یادگیری generator وزن‌های discriminator باید به روز گردند.

در هنگام compile کردن، از تابع هزینه binary_crossentropy و اپتیمایزر RMSProp استفاده شده است. همچنین علاوه بر loss، دقت آن نیز بررسی خواهد شد. در ادامه شبکه adversial ساخته شده نیز آمده است. منظور از این شبکه، همان training مولد است. ورودی adversial، همان ورودی generator یا همان نویز است و خروجی آن خروجی discriminator است هنگامی که خروجی generator به عنوان ورودی به آن اعمال شده است. $(D(G(z)))$



شکل – طراحی adversial شده

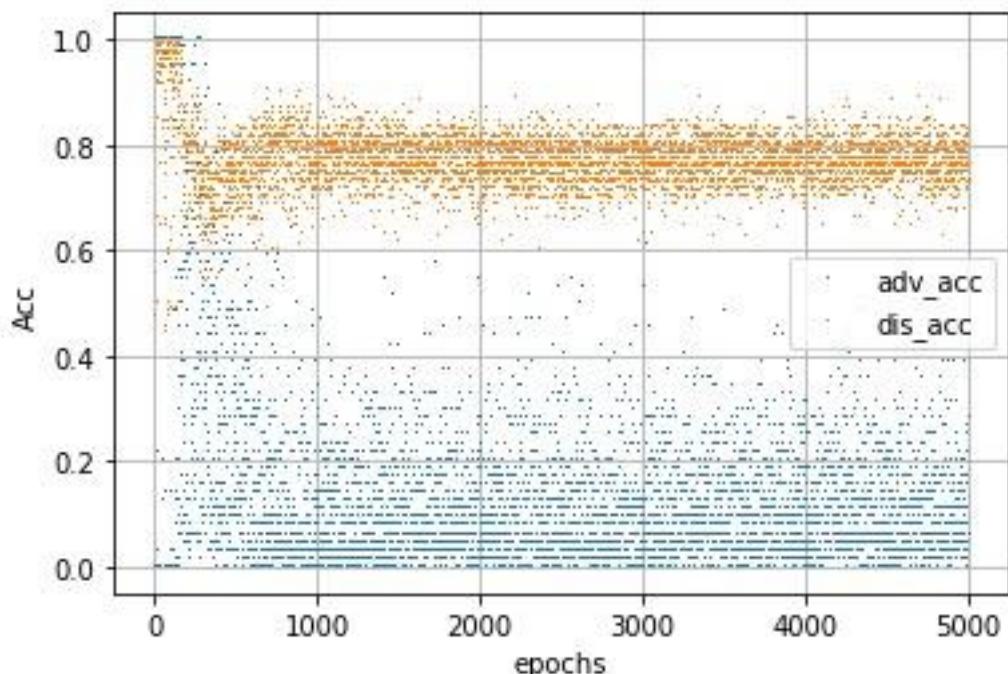
- نحوه ایجاد نویز

در این بخش لازم است یک batch از داده‌های دیتابست به discriminator اعمال شود. از طرفی به همین تعداد باید داده fake نیز وجود داشته باشد. بنابراین generator باید به تعداد batch، نویز به عنوان ورودی بگیرد. در این حالت نویز uniform با ابعاد $batch_size * 100$ با مقادیر منیمم و ماکزیمم -۱ و ۱ ساخته شده است. بدیهی است که generator برای ساخت هر تصویر به یک نویز با ابعاد ۱۰۰ نیاز داشته و تعداد تصاویر ساخته شده نیز به اندازه $batch_size = 64$ خواهد بود. در اینجا $batch_size = 64$ درنظر گرفته شده است و ابعاد نویز ورودی $100 * 64$ خواهد بود. لازم به ذکر است batch انتخاب شده از دیتابست به صورت رندوم برداشته می‌شود.

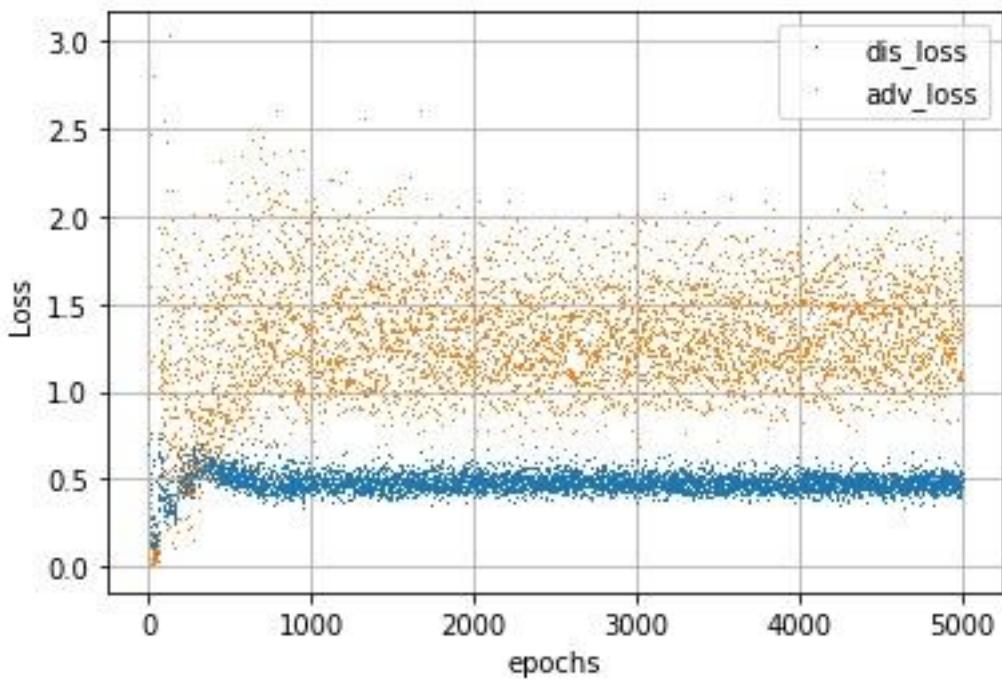
حال لازم است خروجی‌های generator برچسب real و خروجی‌های discriminator برچسب fake بگیرند. برای این کار دو بردار با ابعاد batch و درایه‌های صفر یا یک ساخته شده است. از آنجایی که برای ساخت ورودی و خروجی‌های discriminator، دیتابهای واقعی با دیتابهای ساخته شده، کانکت می‌شوند، بهتر است که ورودی discriminator به صورت شافل شده به آن اعمال گردد.

- نمودارهای loss و accuracy

در ادامه نمودار خطای دقت مربوط به دو بخش generator و discriminator که مشاهده می‌شود در نمودار مربوط به خطای خطای هر دو بخش به صورت نوسانی است ولی در کل روند نزولی دارند. در نمودار مربوط به دقت، دقت discriminator در حدود ۸۰ درصد است اما دقت MRF مربوط به generator کم می‌باشد. لازم به ذکر است، وظیفه discriminator، طبقه‌بندی دو کلاس real و fake است و معیار accuracy برای آن قابل تعریف است. حال برای generator که وظیفه آن تولید تصاویر جدید از نوبیت است، معیار دقت مناسب نمی‌باشد.



شکل - نمودار دقت مربوط به بخش generator و discriminator برای دیتابست Fashion MNIST



شکل - نمودار خطا مربوط به بخش generator و discriminator برای دیتاست Fashion MNIST

- نمونه نتایج شبکه generator

درنهایت شبکه DCGAN به تعداد ۵۰۰۰ ایپاک learn شد و به ازای هر ۵۰۰ عکس، یک ماتریس ۱۰۰×۱۰۰ از تصاویر دیتاست مشاهده گردید. لازم به ذکر است برای مشاهده بهبود تصاویر، از یک نویز با ابعاد ۱۰۰×۱۰۰ استفاده شده است و به generator داده می‌شود؛ حال در هر ایپاک مقادیر generator بهروز شده و تصاویر از همان نویز اولیه ساخته می‌شوند.



شکل - نمونه‌های تولیدشده توسط generator در ایپاک ۵۰۰



شکل - نمونه‌های تولیدشده توسط generator در ایپاک ۱۰۰۰



شکل - نمونه‌های تولیدشده توسط generator در ایپاک ۱۵۰۰



شکل - نمونه‌های تولیدشده توسط generator در ایپاک ۲۰۰۰



شکل - نمونه‌های تولیدشده توسط generator در ایپاک ۲۵۰۰



شکل - نمونه‌های تولیدشده توسط generator در ایپاک ۳۰۰۰



شکل - نمونه‌های تولیدشده توسط generator در ایپاک ۳۵۰۰



شکل - نمونه‌های تولیدشده توسط generator در ایپاک ۴۰۰۰



شکل - نمونه‌های تولیدشده توسط generator در ایپاک ۴۵۰۰



شکل - نمونه‌های تولیدشده توسط generator در ایپاک ۵۰۰

CIFAR-10 ♦

- توبولوژی لایه‌های مولد و تفکیک‌کننده

در این بخش شبکه DCGAN طراحی شده در بخش قبل، بر روی دیتاست CIFAR-10 اعمال می‌گردد. این دیتاست برای داده‌های Train شامل ۵۰۰۰۰ عکس با ابعاد $32 \times 32 \times 3$ است. درواقع معماری استفاده شده برای دیتاست cifar10 برای دیتاست fashion mnist جوابگو نبود. دلیل آن پیچیده‌تر بودن عکس‌های مربوط به این دیتاست است، چراکه سه کاناله بوده و به صورت رنگی می‌باشد.

درواقع ورودی generator یک نویز با ابعاد ۱۰۰ می‌باشد که به یک لایه fully connected با تعداد نورون $2^2 \times 512$ برد و در آنجا به استفاده از reshape کردن، ابعاد به صورت $2^2 \times 512$ درمی‌اید. حال از ۴ لایه Cinv2DTranspose استفاده شده است که همانند deconv بوده و عمل upsampling را انجام می‌دهند و ابعاد تصویر را از $2^2 \times 512$ به $32 \times 32 \times 3$ می‌رسانند. در لایه‌های deconv از padding=same و stride=2 و kernel_size=5 و رسانید و با استفاده از تعداد فیلترها هر بار عمق آن‌ها نیز نصف گردد. در همه لایه‌ها، به جز لایه آخر از تابع فعال‌ساز leaky relu با ضریب 0.2 استفاده شده است. در لایه آخر از تابع tanh استفاده شده است. همچنانی بعد از هر لایه از batch normalization نیز استفاده شده است.

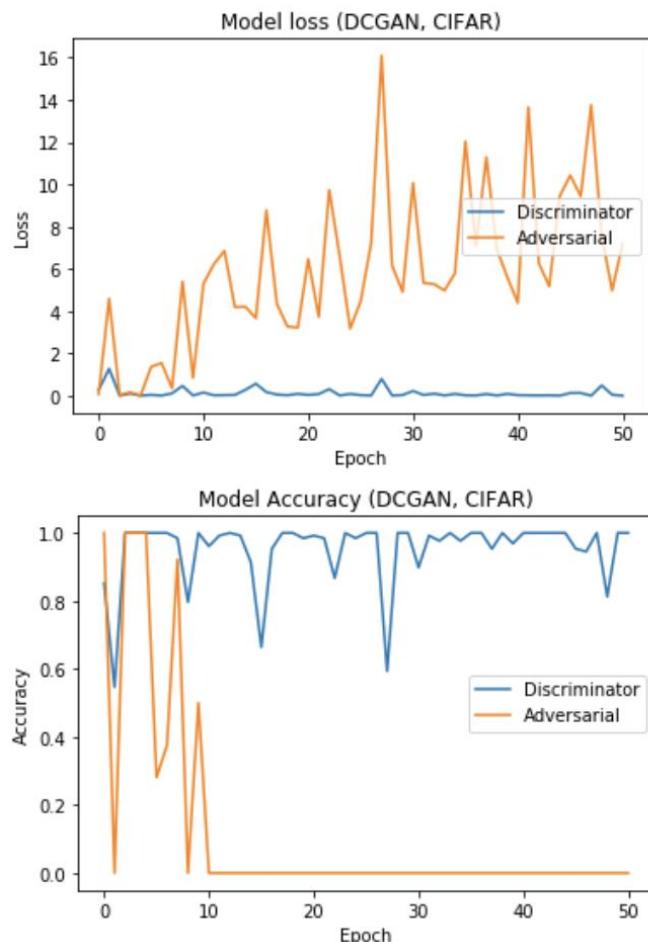
در بخش discriminator، ورودی‌های شبکه (دیتاست و ساخته شده توسط generator) باید دارای ابعاد $32 \times 32 \times 3$ باشند. معماری مربوط به لایه‌های کانولوشنی generator بر عکس discriminator است. بنابراین ابعاد $32 \times 32 \times 3$ طی ۴ لایه کانولوشنی با padding=same و stride=2 و kernel_size=5 به ابعاد $2^2 \times 512$ رسیده و پس از یک flat کردن از لایه dense با یک نورون استفاده شده است. تابع فعال‌ساز همه لایه‌ها به جز لایه آخر leaky relu است و در لایه آخر از sigmoid استفاده شده است.

- نحوه ایجاد نویز

نحوه ایجاد نویز همانند دیتاست fashion mnist است. بنابراین از نویز uniform با ابعاد $batch_size \times 100 \times 1 \times 1$ استفاده شده است. با مقادیر منیمم و ماکزیمم $1 - 0$.

- نمودار Loss و Accuracu

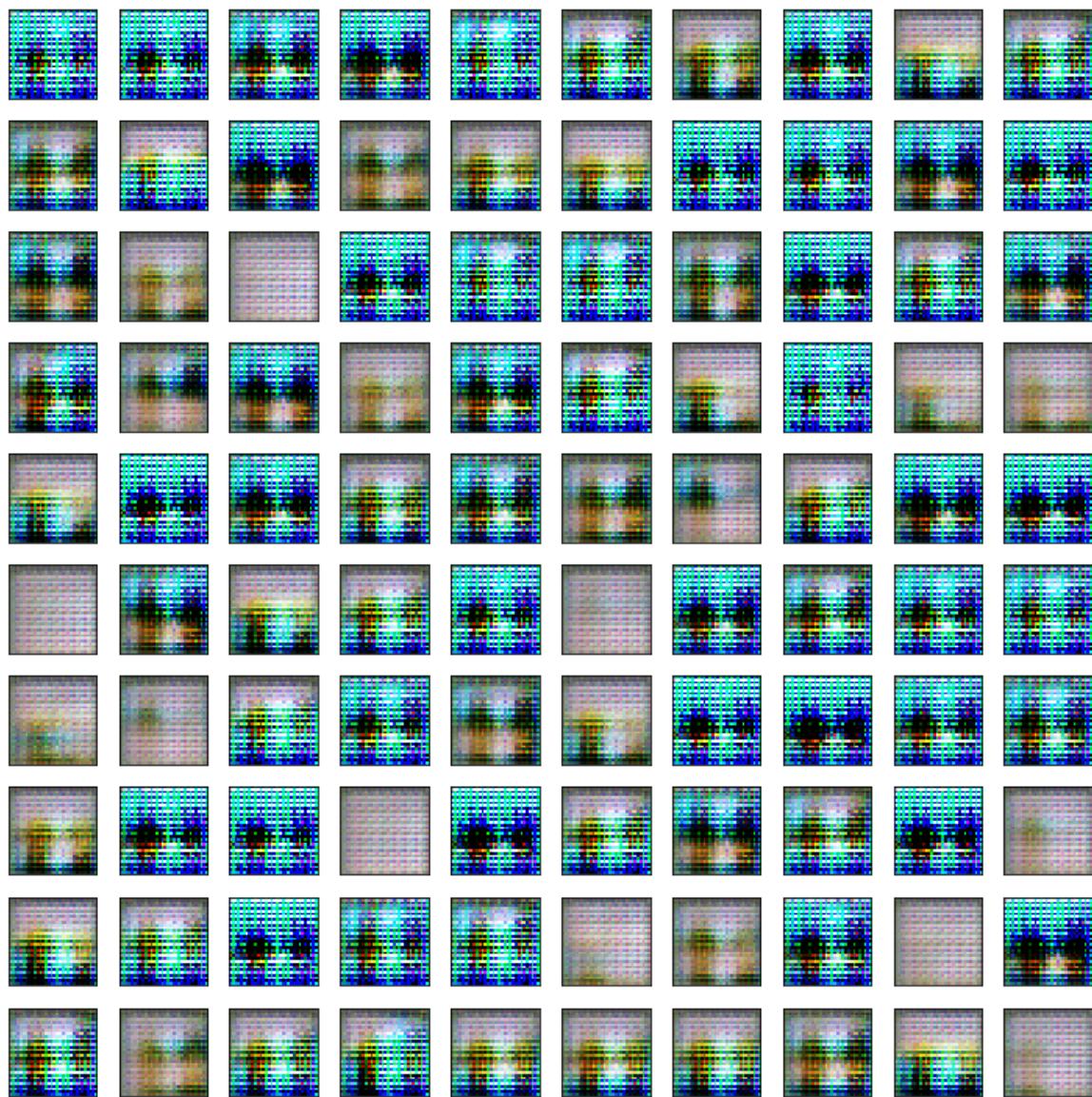
در ادامه نمودار مربوط به خطا و دقت برای DCGAN طراحی شده در ۵۰ ایپاک و batch-size=32 آمده است. همانطور که مشاهده می شود، نمودا به صورت نوسانی بوده و برای هر دو متغیر افزایش و کاهش می یابد.



شکل - نمودار خطا و دقت برای دو بخش **adversarial** و **discriminator** در دیتاست cifar10

- نمونه نتایج شبکه

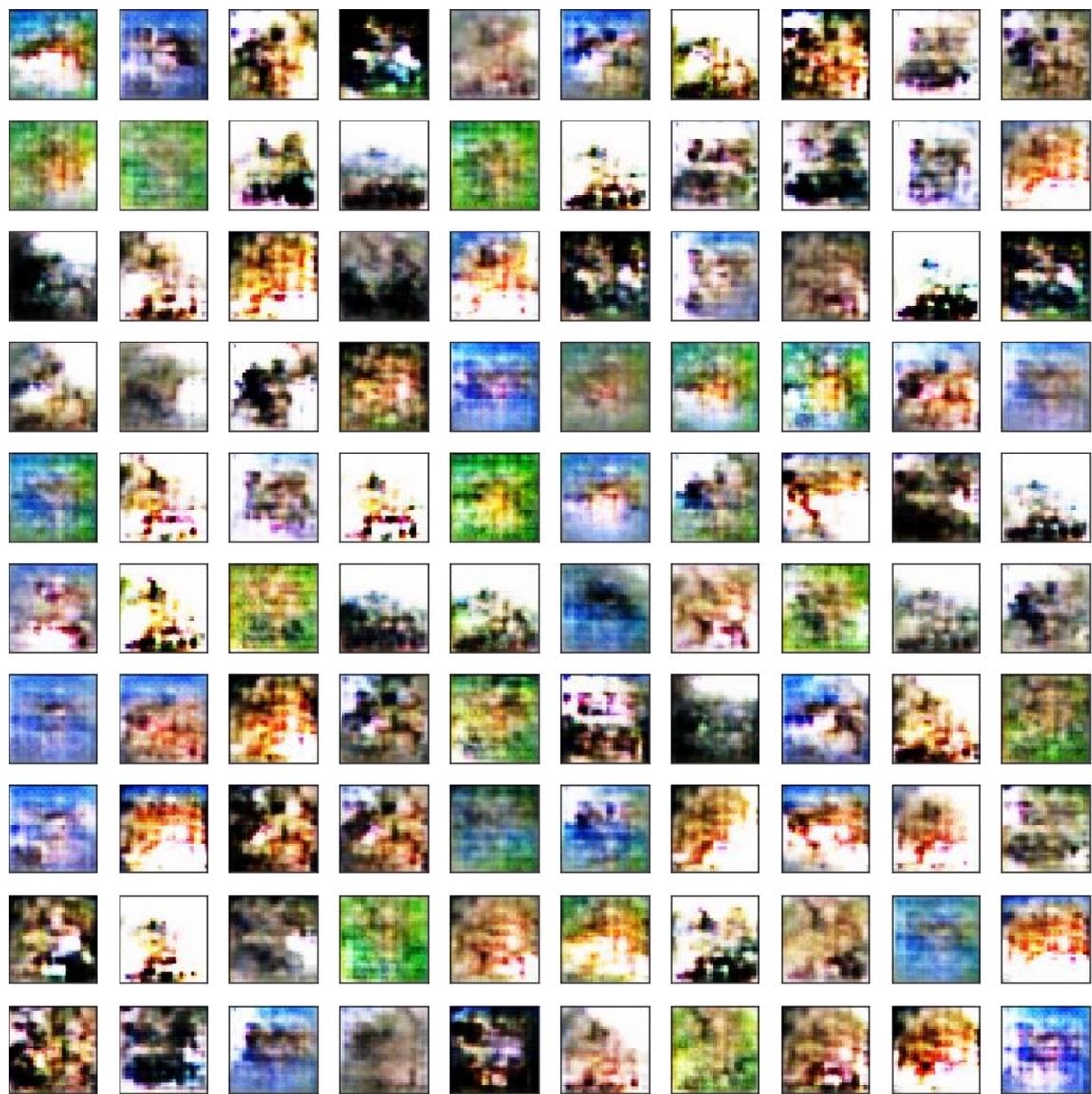
در ادامه ۱۰ نمونه از تصاویر تولید شده توسط generator برای دیتاست cifar10 قابل مشاهده است.



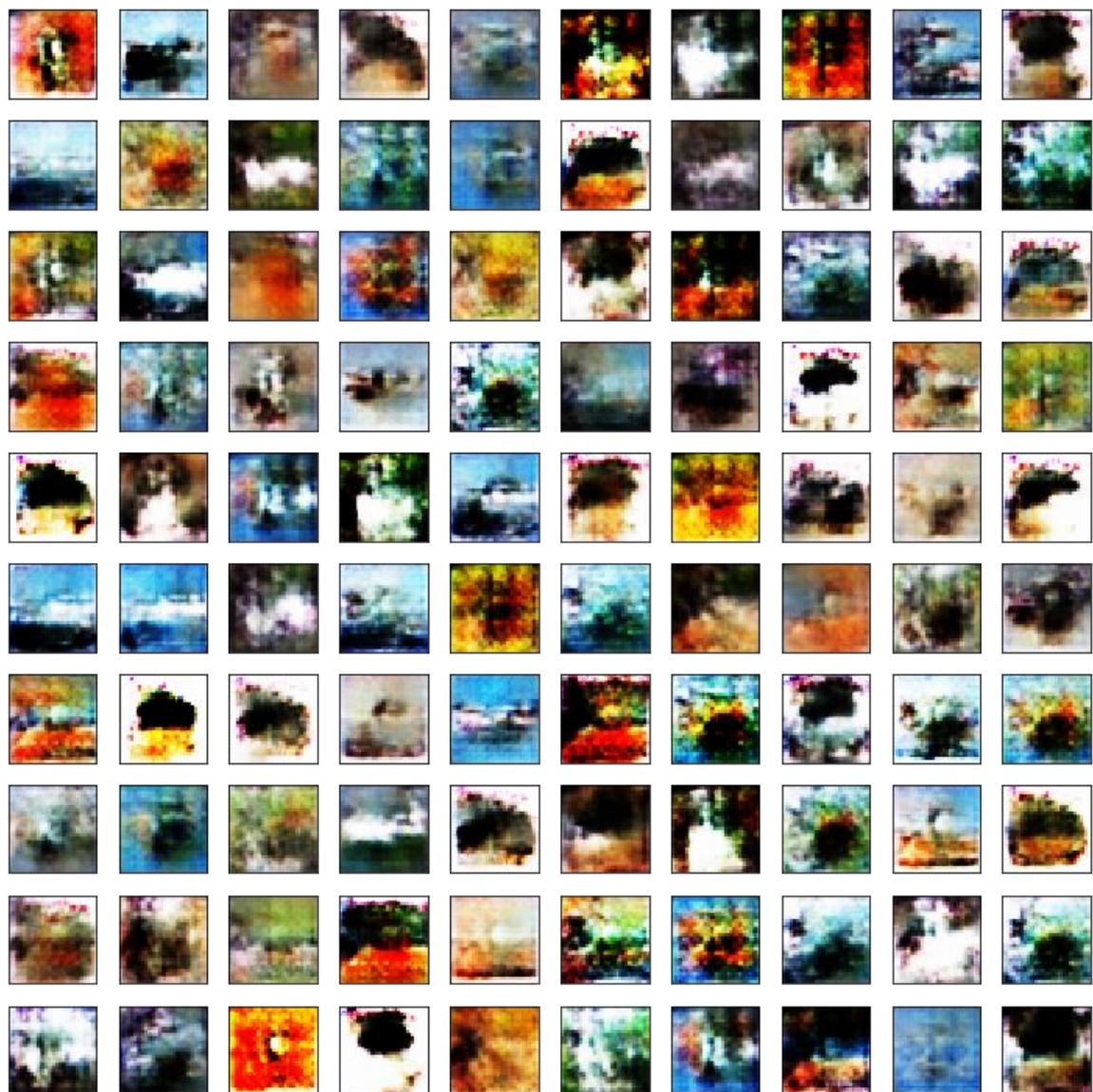
شکل - نمونه تصاویر ساخته شده توسط generator با DCGAN در ایپاک ۱



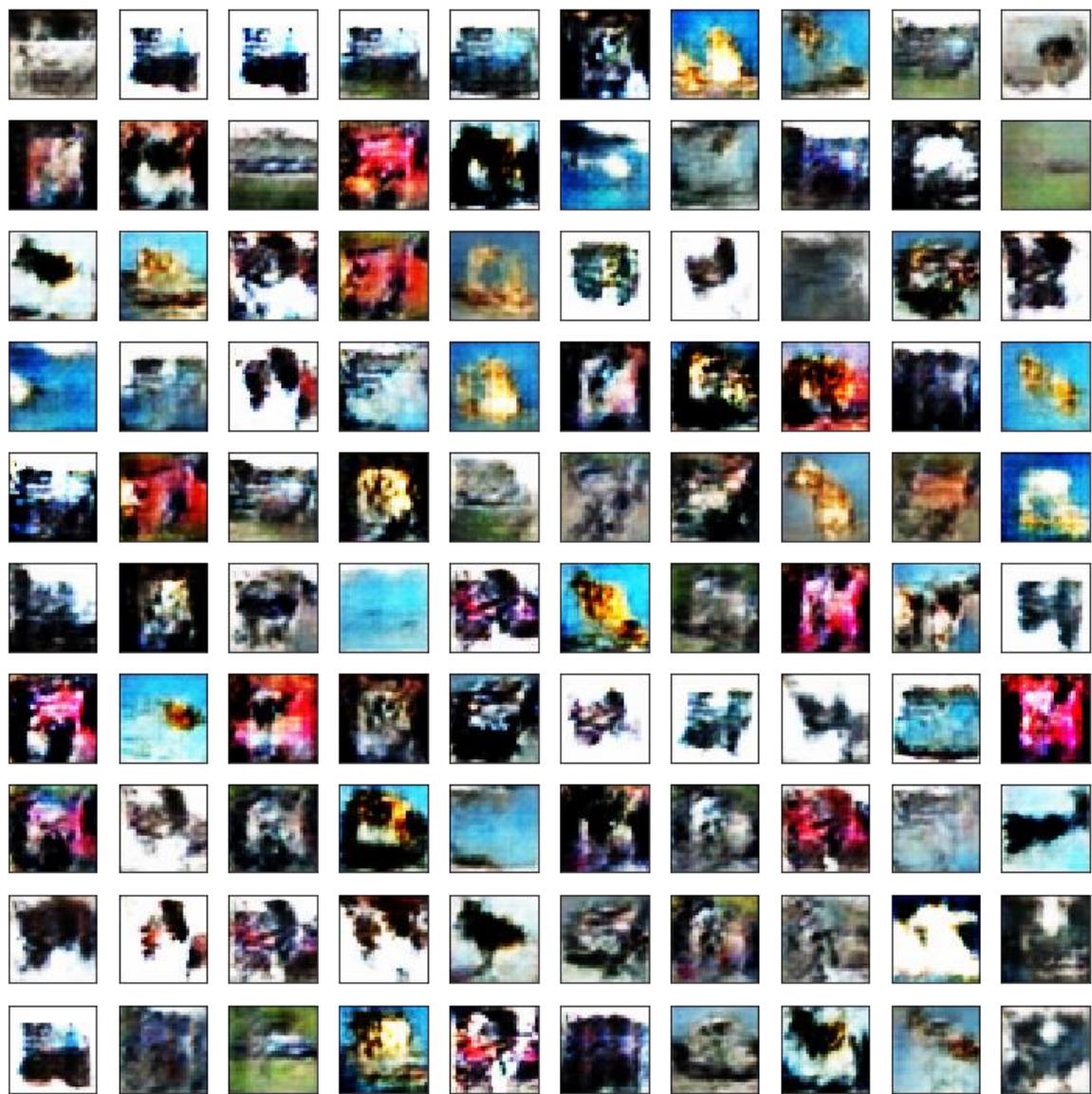
شکل - نمونه تصاویر ساخته شده توسط generator با DCGAN در ایپاک ۶



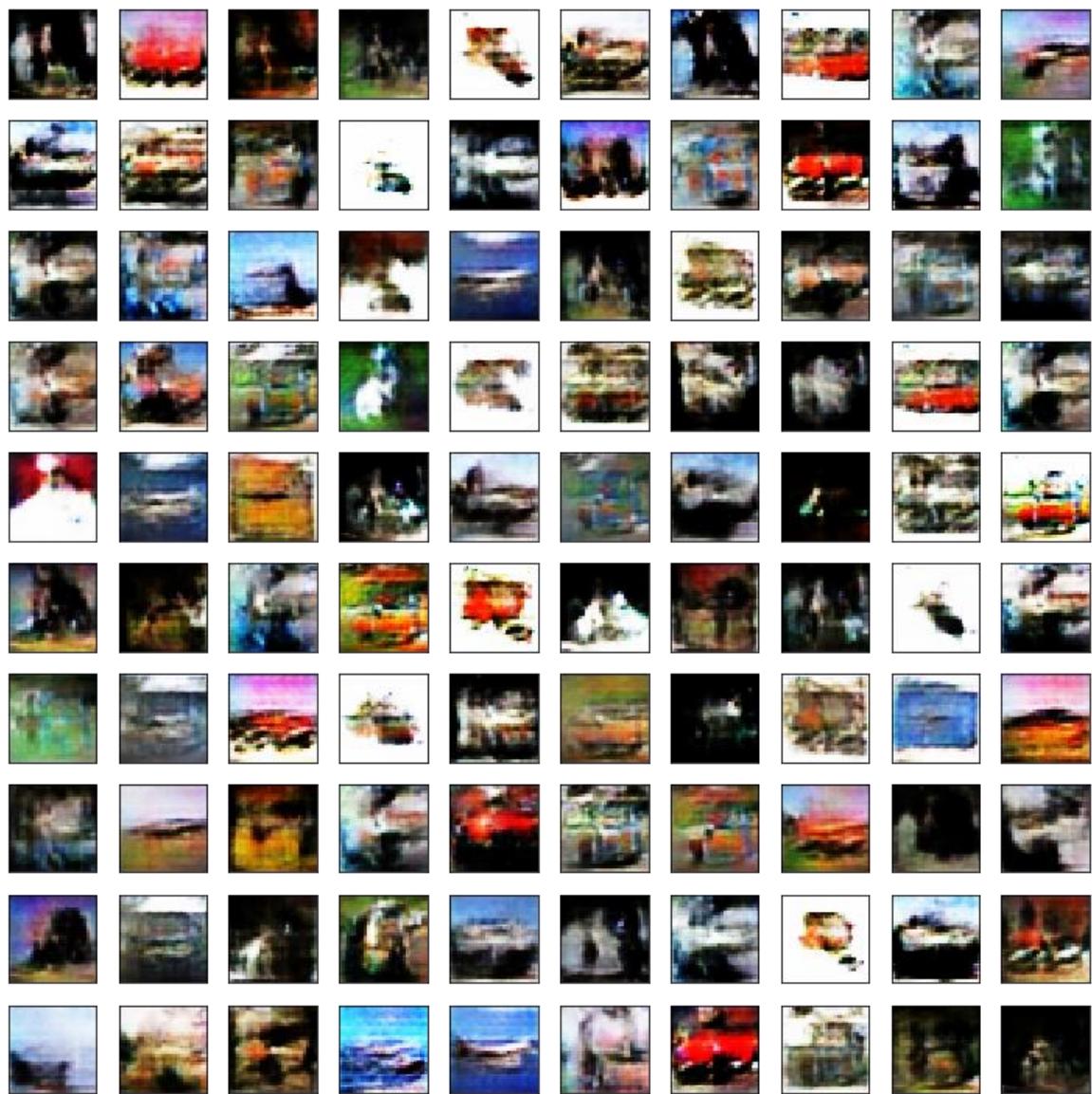
شکل - نمونه تصاویر ساخته شده توسط generator با DCGAN در ایپاک ۱۱



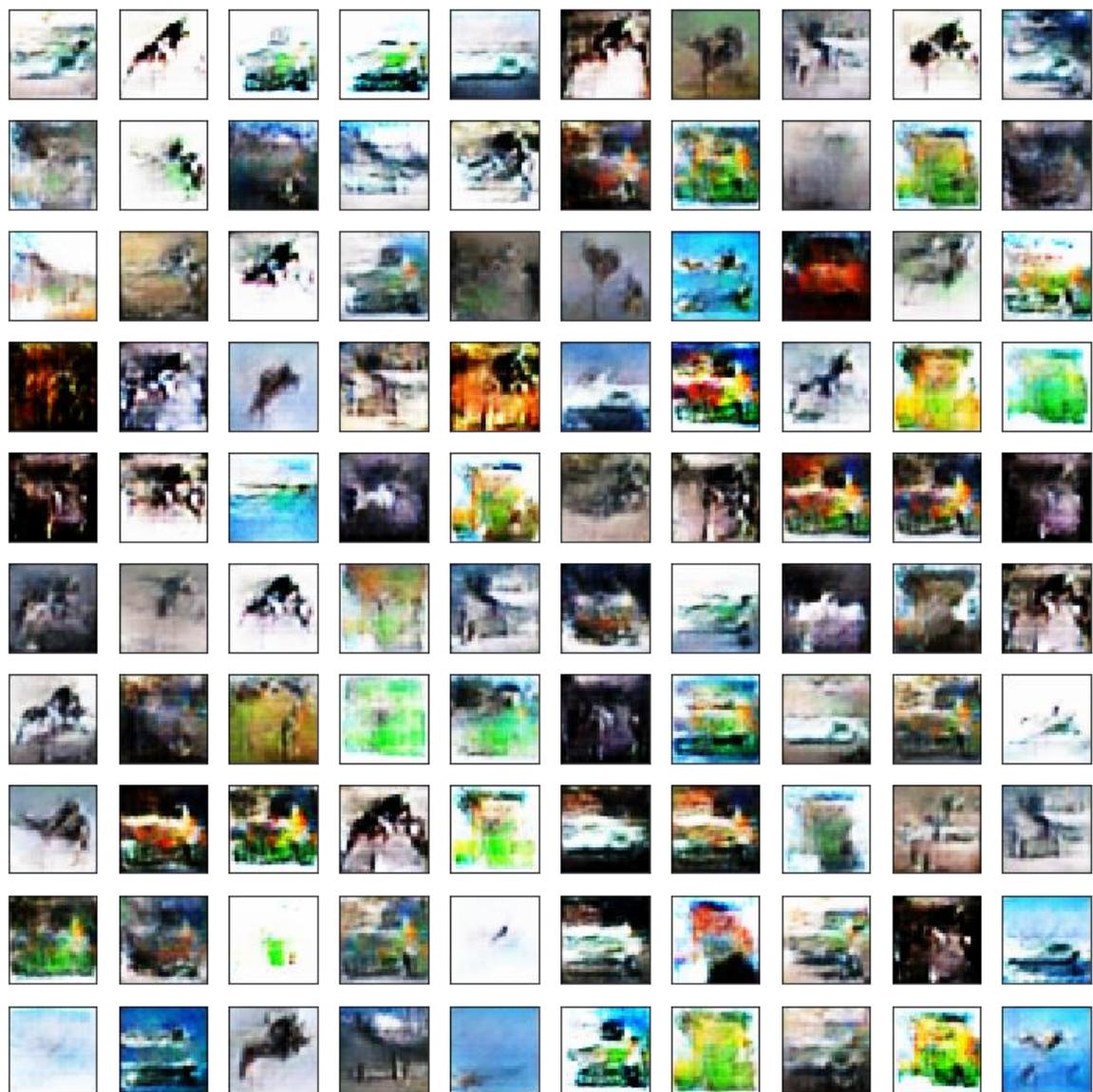
شکل - نمونه تصاویر ساخته شده توسط generator با DCGAN در ایپاک ۱۶



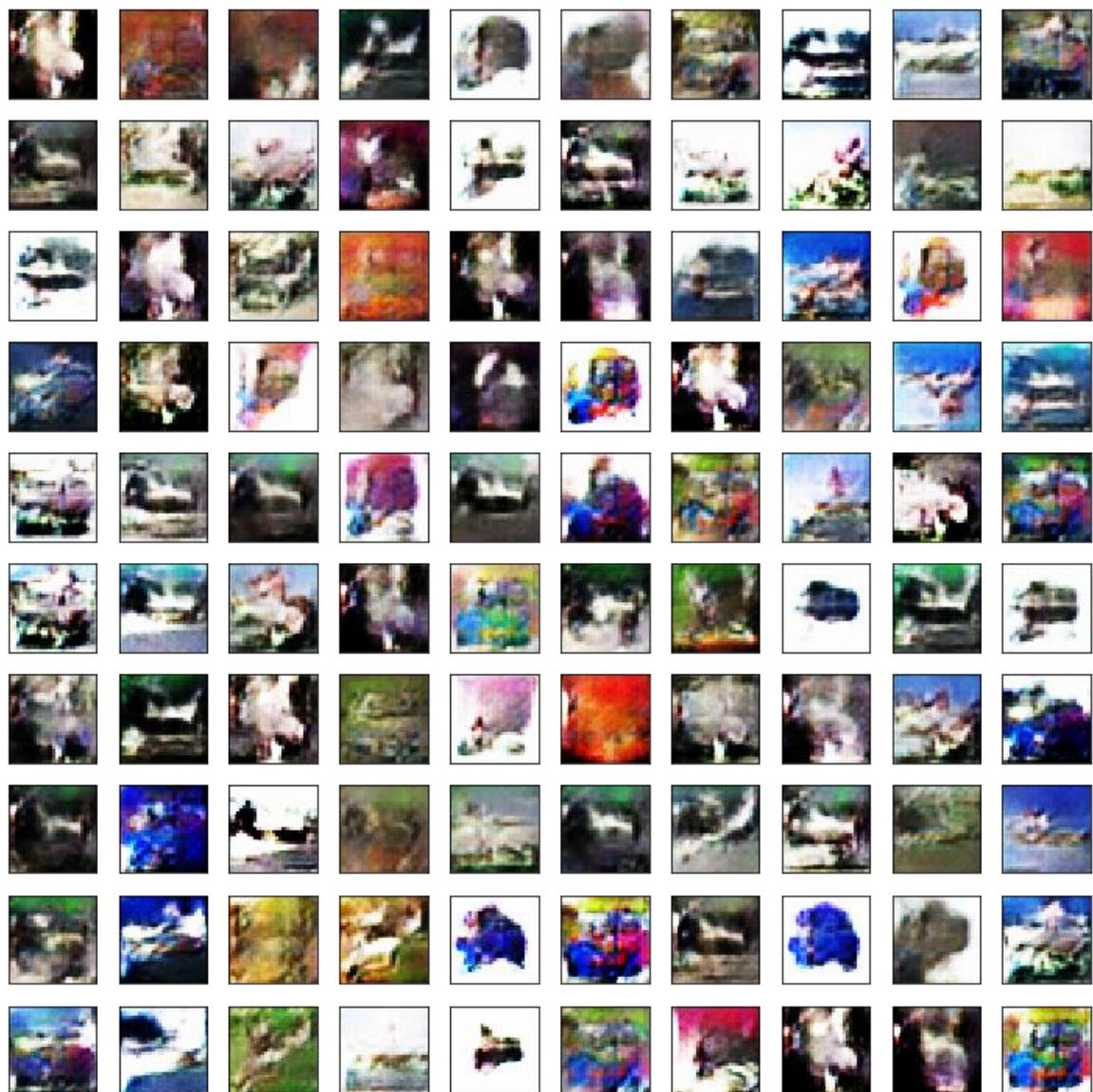
شکل - نمونه تصاویر ساخته شده توسط generator با DCGAN ایپاک ۲۱



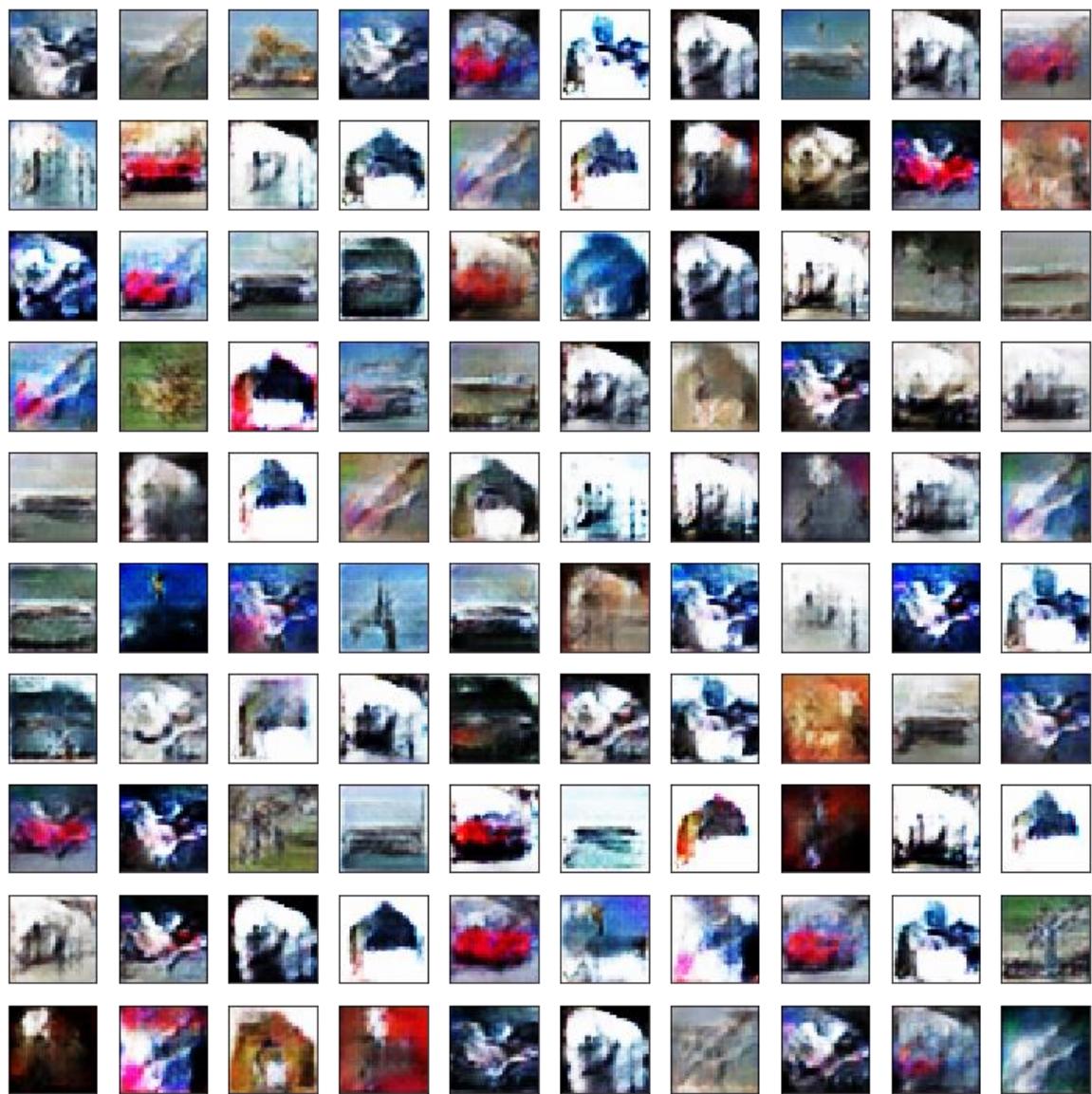
شکل - نمونه تصاویر ساخته شده توسط generator با DCGAN در ایپاک ۲۶



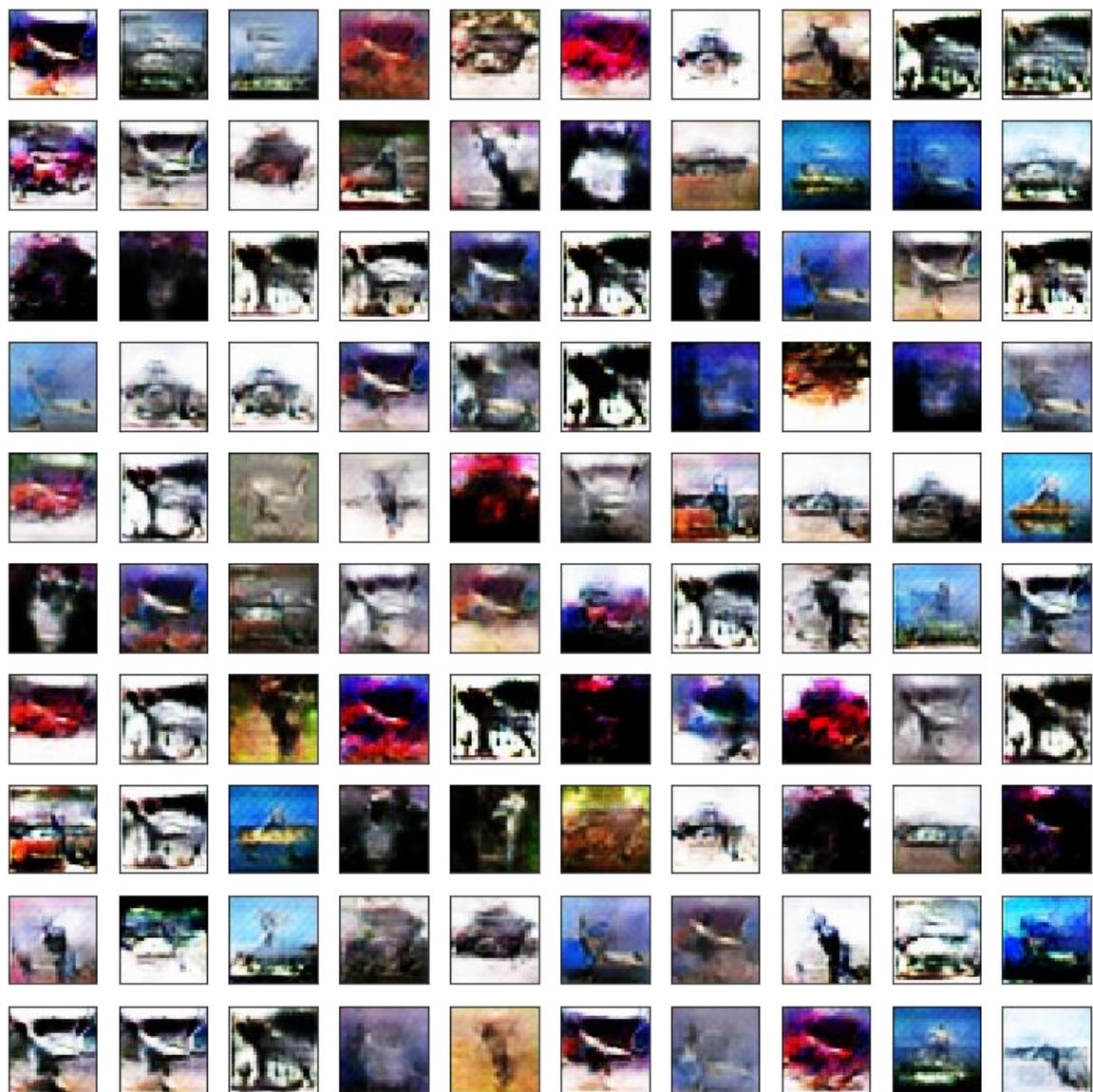
شکل - نمونه تصاویر ساخته شده توسط generator با DCGAN در ایپاک ۳۱



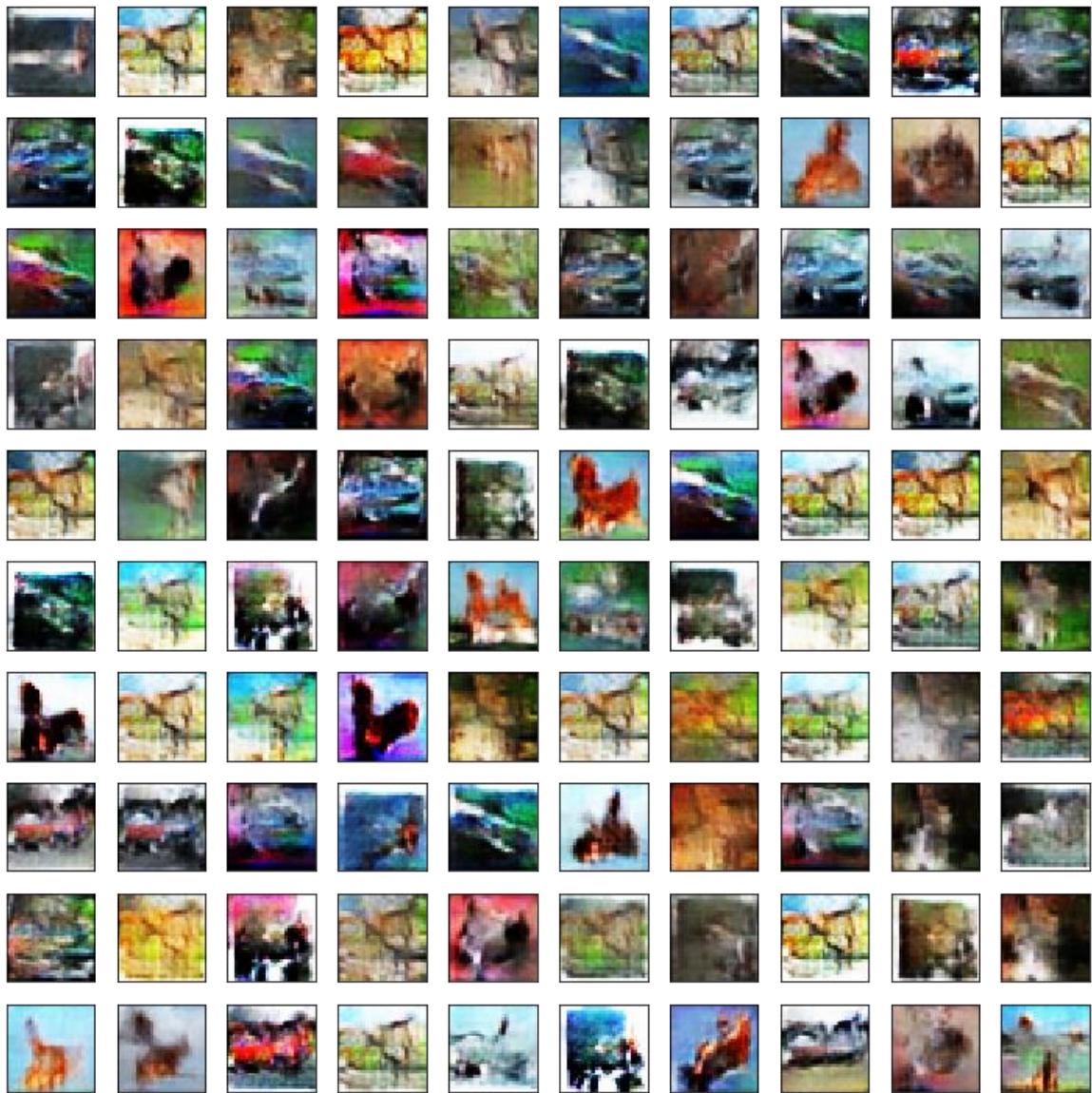
شکل - نمونه تصاویر ساخته شده توسط generator با DCGAN در ایپاک ۳۶



شکل - نمونه تصاویر ساخته شده توسط generator با DCGAN در ایپاک ۴۱



شکل - نمونه تصاویر ساخته شده توسط generator با DCGAN در ایپاک ۴۶



شکل - نمونه تصاویر ساخته شده توسط generator با DCGAN در ایپاک ۵۱

سوال ۳ - راههای بهبود GAN

الف) راههایی برای بهبود GAN

• مقاله اول: one-sided label smoothing

در این بخش راههایی برای بهبود همگرایی GAN پیشنهاد شده است. در راستای یادگیری شبکه‌های GAN لازم است یک تعادل بین دو بخش رقابت‌کننده discriminator و generator ایجاد شود. درواقع هر بخش یک تابع هزینه داشته و می‌خواهد آن را کاهش دهد. اگرچه کاهش تابع هزینه، استفاده از گرادیان کاهشی را تداعی می‌کند؛ اما استفاده از گرادیان برای کاهش تابع هزینه هر بخش، ممکن است loss

بخش دیگر را افزایش دهد. بنابراین یافتن تعادل بین دو بخش و همگرايی راحت نخواهد بود. در ادامه تکنيک‌هایی در راستای افزایش همگرايی شبکه‌های GAN معرفی می‌شود.

- تکنيک Feature Matching

در تکنيک generator، discriminator به جای feature mapping up-sample کردن مستقيم خروجي generator، discriminator برای توليد داده fake، تنها از مشخصه‌های آماري خروجي discriminator استفاده می‌کند. بنابراین در هنگام يادگيری discriminator، ويژگی‌هایی که بيشترین نقش را در تفكيك داده‌های واقعی و غيرواقعي دارند، استخراج می‌شوند. در يادگيری GAN استاندارد، نمونه‌های ساخته شده با توزيع داده‌های واقعی تطابق دارند؛ بنابراین رسیدن به آن نقطه ممکن با مشکل روبرو شود.

- تکنيک Mini-batch Discrimination

يکی از مشکلات GAN آن است که generator در تنظيم کردن پaramترها به مشکل می‌خورد و همواره یک خروجي توليد می‌کند. بنابراین گراديان زدن برای discriminator همواره یک مسیر يکسان را طی می‌کند و نمی‌توان به generator گفت که توليدات متنوعتری داشته باشد. بنابراین همگرايی به یک توزيع mini-batch با انتروپي درست، سخت خواهد شد. استراتژي مقابله با اين مشکل، استفاده از تکنيک discrimination است. اين تکنيک به discriminator کمک می‌کند تا به نمونه‌های مختلف نگاه کند و generator را از توليد نمونه‌های تكراري نجات دهد. اگر بردار ويژگي ساخته شده توسط discriminator داراي A بعد باشد و در يک تنسور با ابعاد A^*B^*C ضرب شود، يک ماترييس با ابعاد B^*C مي‌دهد. حال نرم L1 درايدها هر ستون از ماترييس با درايدهای ديگر همان ستون محاسبه می‌شود و به صورت منفي نمایي می‌گردد. مجموع آنها خروجي laie mini-batch را می‌سازد. حال اين خروجي با بردار ويژگي تركيب می‌شود و به لايه بعدی discriminator داده می‌شود. حال برای تفكيك داده‌های واقعی از داده‌های ساخته شده توسط generator، از ساير نمونه‌های موجود در mini-batch بنيز استفاده می‌کند.

- تکنيک Historical Averaging

در هنگام پياده‌سازی اين تکنيک،تابع هزينه هر بخش داراي ترم زير خواهد بود که $\theta[i]$ مقدار پaramترها در زمان گذشته i است. در الواقع historical average برای پaramترها در طول يادگيری بهروزرسانی می‌شود. درنهایت گراديان بر روی نقاط تعادل شبکه تمرکز می‌کند.

$$\|\theta - \frac{1}{t} \sum_{i=1}^t \theta[i]\|^2$$

- تکنيک One-sided Label Smoothing

می‌توان به جای استفاده از بروچسب صفر و یک برای خروجی binary classifier، از مقادیر نرم ۰.۱ و ۰.۹ استفاده می‌شود تا آسیب‌پذیری شبکه‌ها کاهش یابد. درواقع اگر α و β مقادیر متناظر با تارگت مثبت و منفی باشند، خروجی discriminator به صورت زیر می‌شود.

$$D(\mathbf{x}) = \frac{\alpha p_{\text{data}}(\mathbf{x}) + \beta p_{\text{model}}(\mathbf{x})}{p_{\text{data}}(\mathbf{x}) + p_{\text{model}}(\mathbf{x})}$$

وجود Pmodel در رابطه بالا مشکل‌ساز خواهد بود؛ زیرا که اگر Pdata تقریباً صفر باشد، مقدار Pmodel بزرگ خواهد بود و انگیزه‌ای برای کم کردن مقدار خود و رسیدن به داده‌ها را نخواهد داشت. بنابراین در این تکنیک فقط مقدار مثبت بروچسب‌ها (α) نرم می‌شود و مقدار منفی همان صفر باقی می‌ماند.

- Virtual Batch Normalization

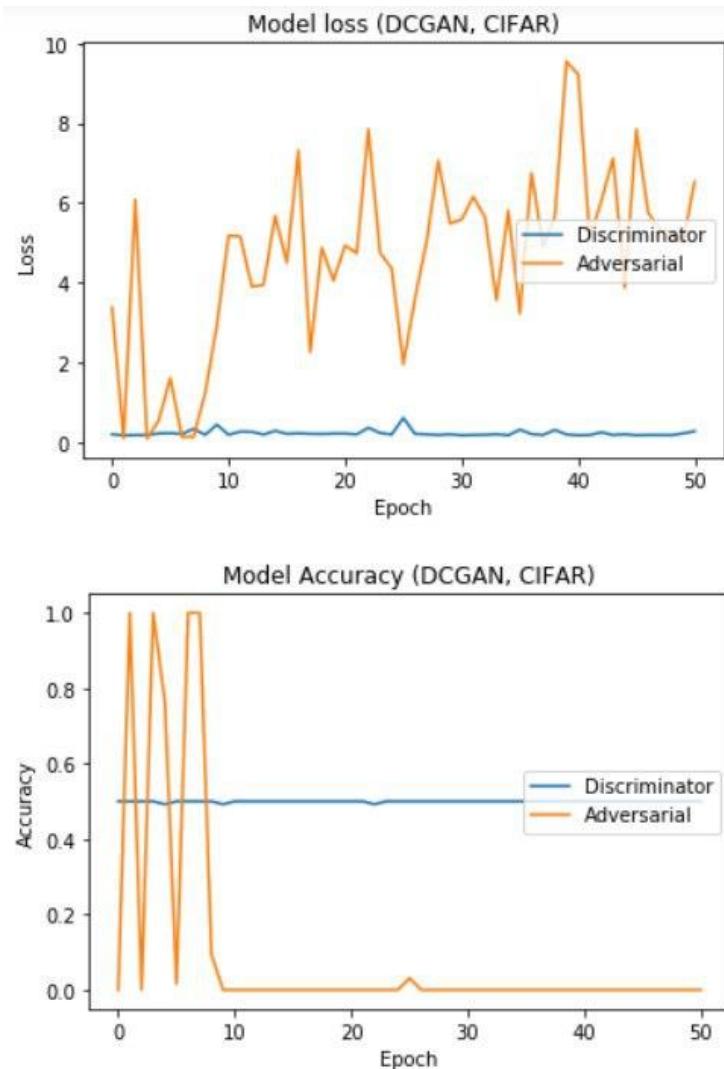
استفاده از تکنیک batch normalization منجر به بهبود بهینه‌سازی شبکه‌های عصبی می‌شود. هرچند که BN منجر به وابستگی زیاد هر نمونه به نمونه‌های دیگر موجود در mini-batch می‌شود. بنابراین در تکنیک VBN هر نمونه با مقادیر آماری batch مرجع نمایانی می‌شود. البته استفاده از تکنیک VBN از نظر محاسباتی سنگین بوده و فقط در بخش generator از آن استفاده می‌شود.

• noise : مقاله دوم

استفاده از نویز با توزیع نرمال از لحاظ پیاده‌سازی و simplicity تولید آسانتری نسبت به نویز یونیفرم دارد. از آنجایی که concentration را در حوالی صفر قرار می‌دهد؛ باعث parameter regularization می‌گردد. همچنین نویز نرمال باعث بهتر شدن انتروپی برای واریانس داده شده می‌شود.

❖ استفاده از تکنیک one-sided label smoothing در دیتابست cifar10

در این بخش همان معماری استفاده شده برای cifar10 به کار رفته است. با این تفاوت که در هنگام تعریف label های مربوط به داده‌های real، از بروچسب یک استفاده نشده و به جای آن از ۰.۹ label استفاده شده است. در ادامه نمودار مربوط به خطای دقت برای ۵۰ ایپاک و batch-size=32 آمده است.

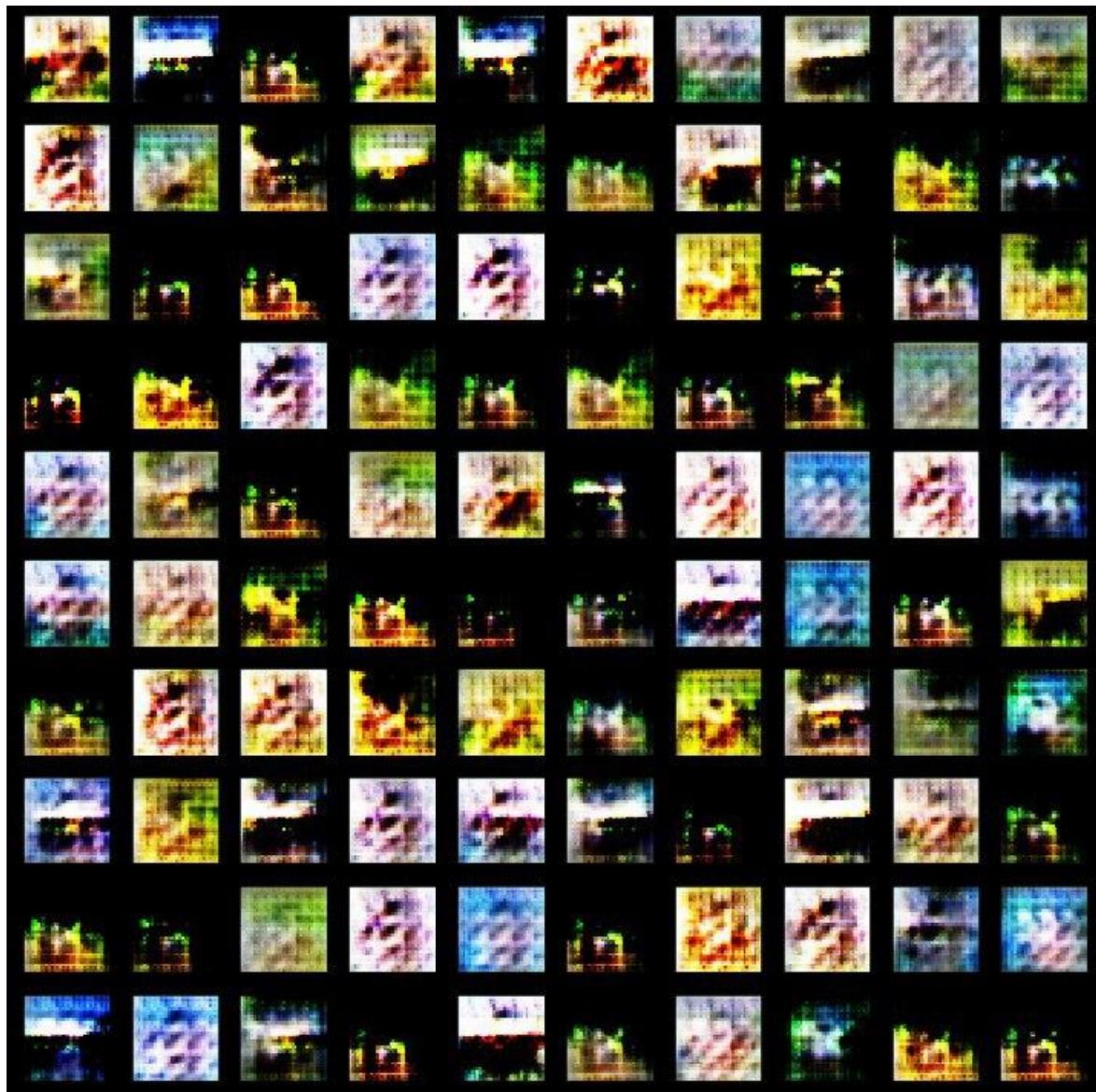


شکل - نمودار خطا و دقت برای cifar10 با استفاده از تکنیک one-sided label smoothing

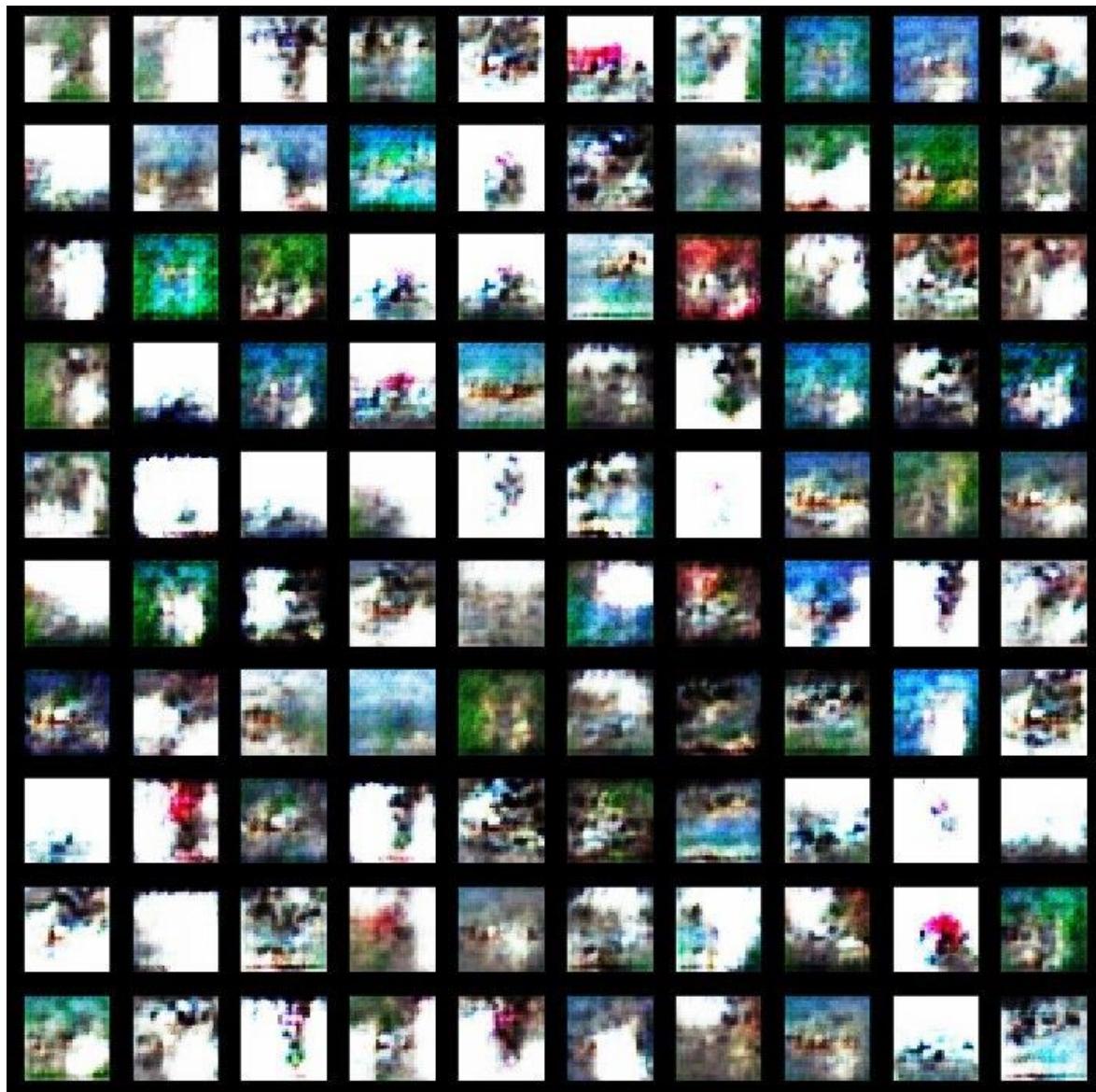
در ادامه نمونه تصاویر ساخته شده توسط generator شبکه DCGAN با استفاده از تکنیک one-sided label smoothing برای بهبود کیفیت تصاویر آمده است.



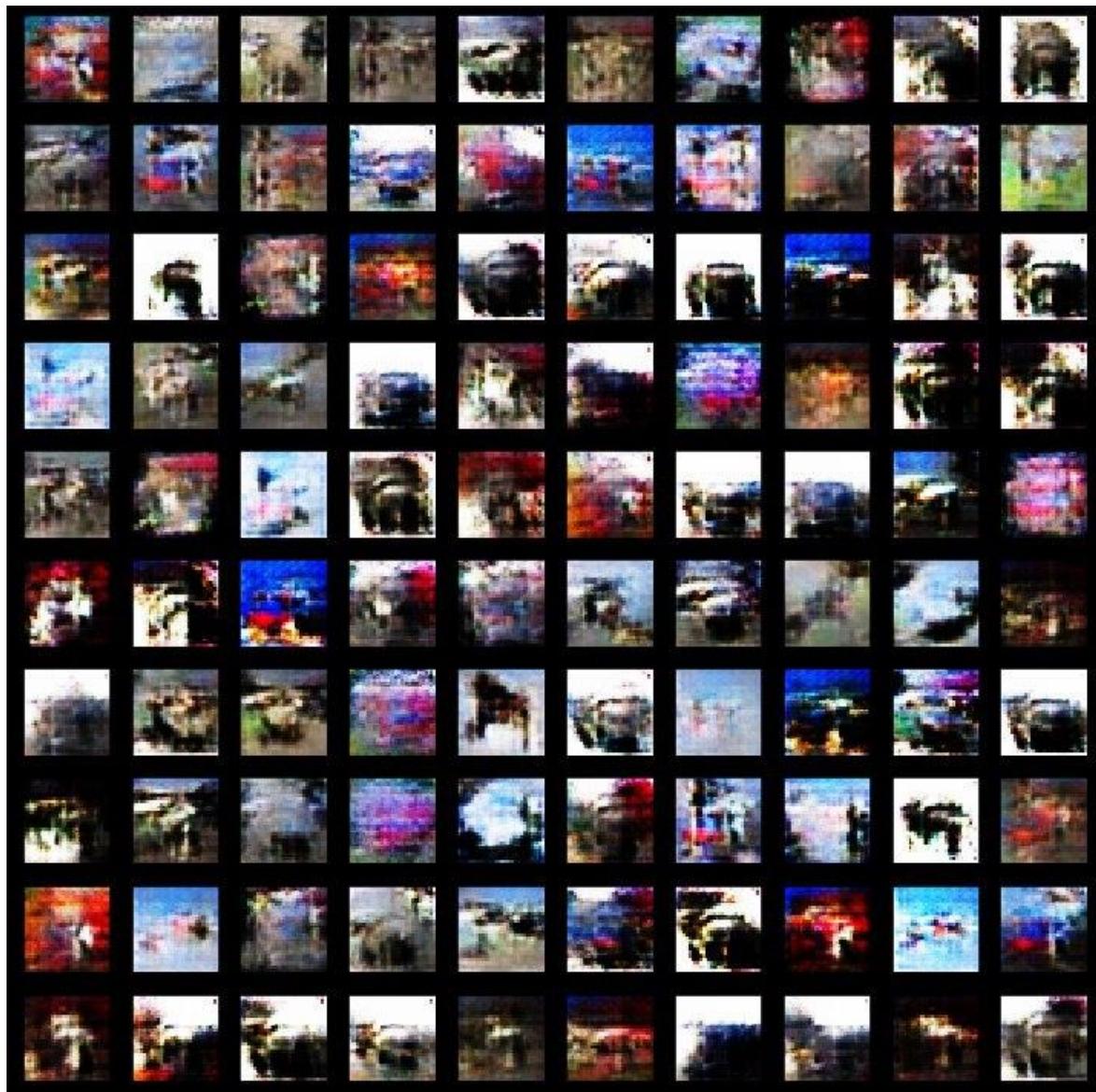
شکل - نمونه تصاویر ساخته شده توسط smoothing generator با تکنیک generator در ایپاک ۱



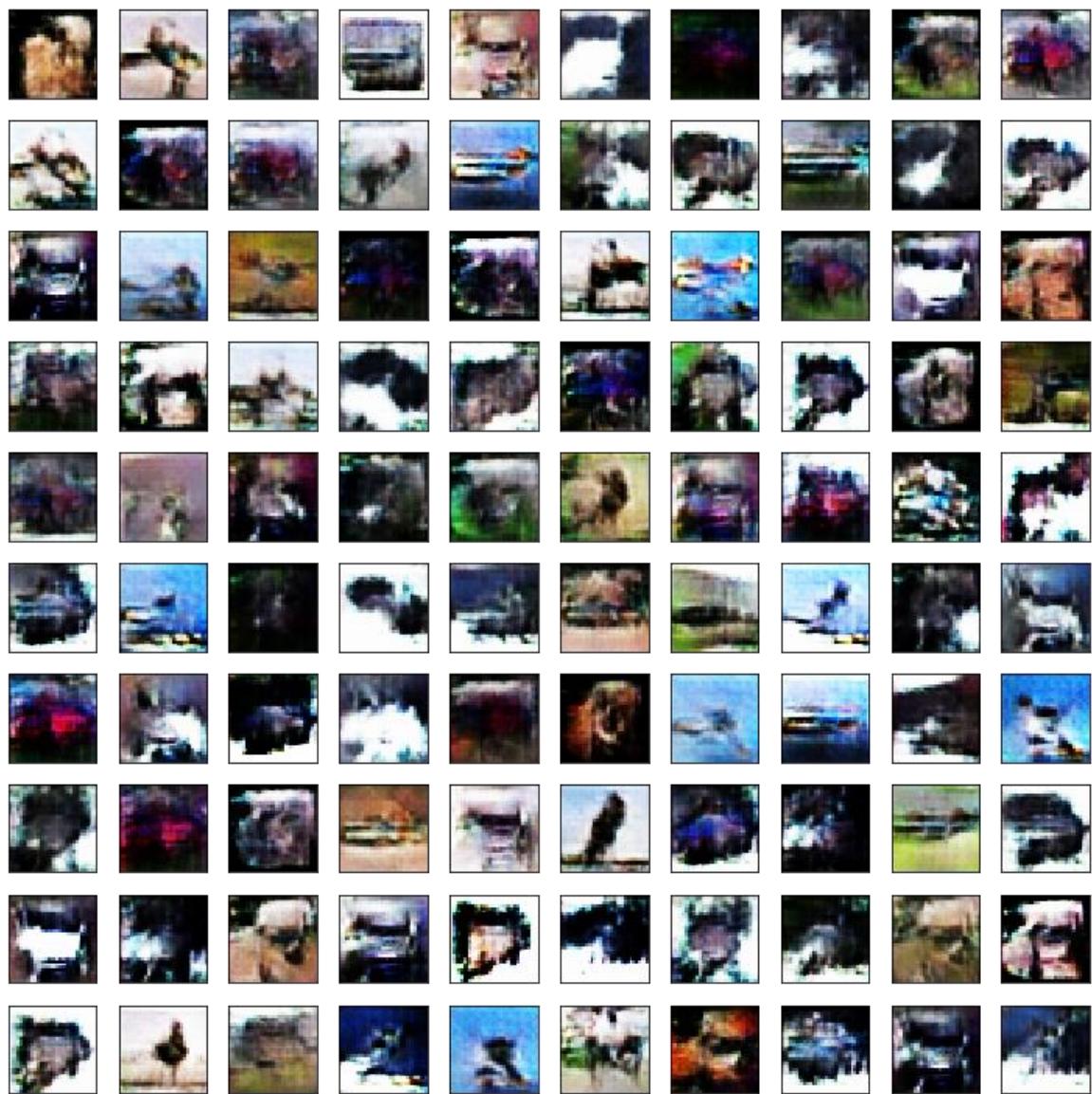
شکل - نمونه تصاویر ساخته شده توسط smoothing با تکنیک generator در ایپاک ۶



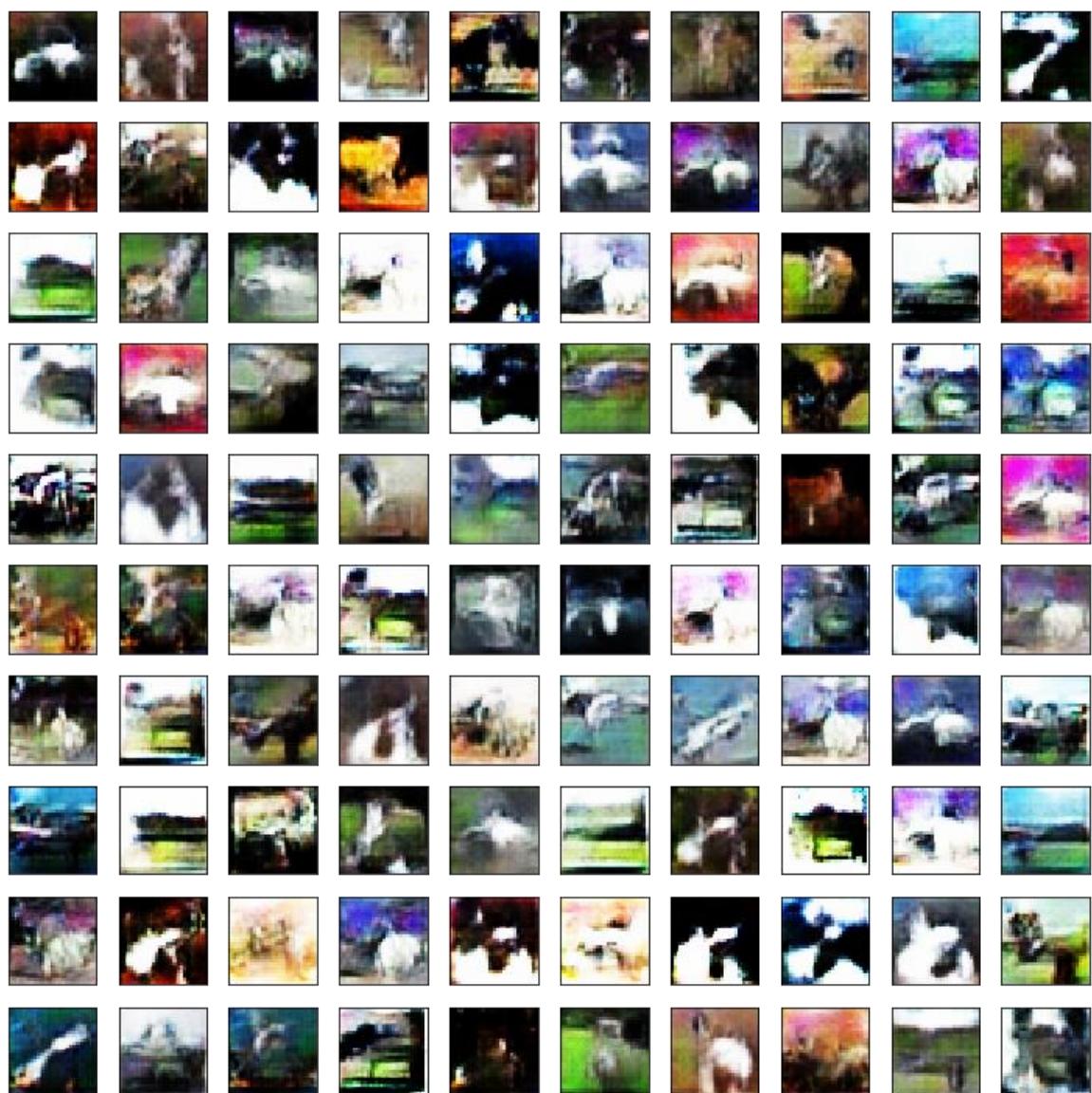
شکل - نمونه تصاویر ساخته شده توسط generator با تکنیک smoothing در اپاک ۱۱



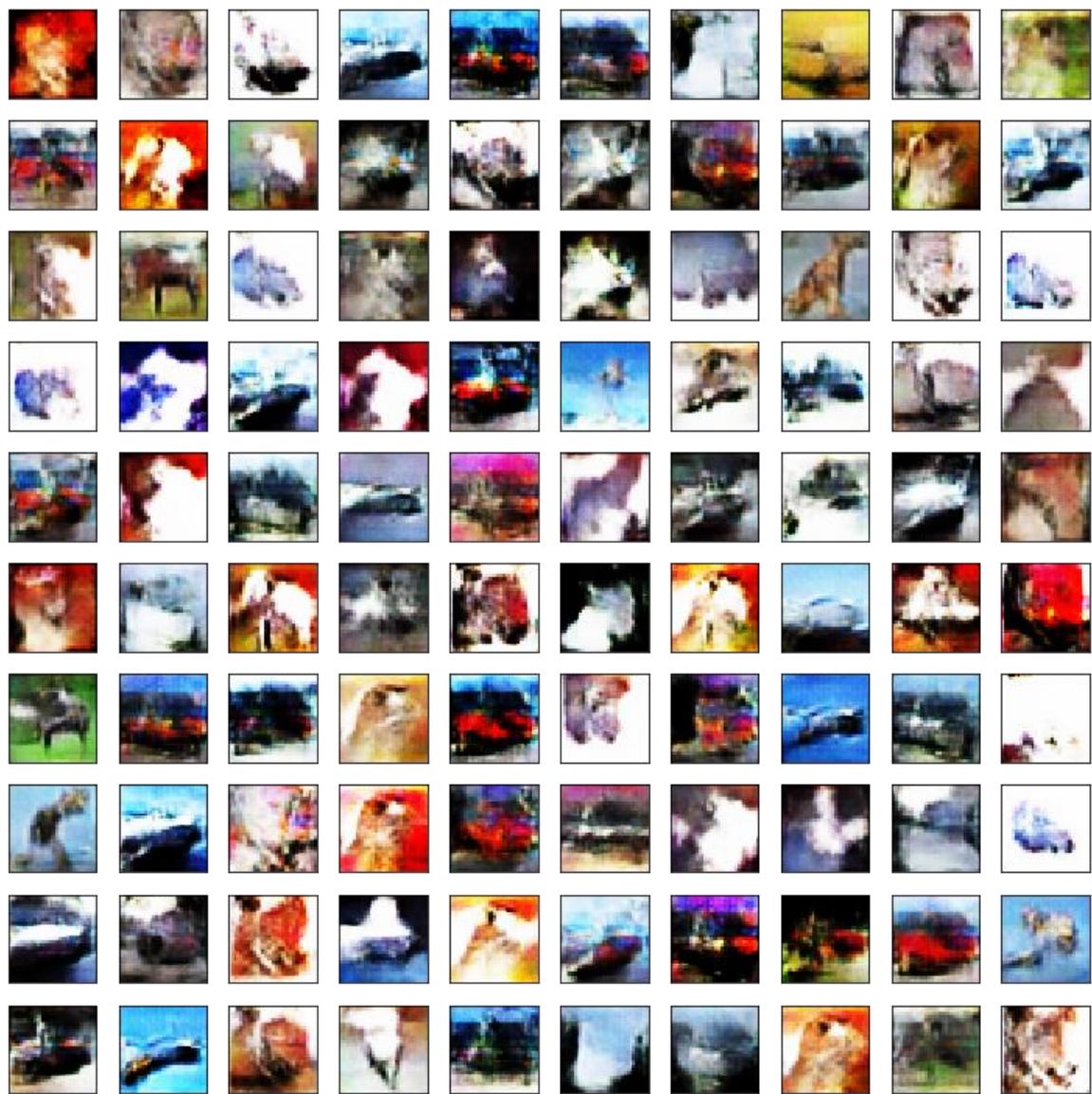
شکل - نمونه تصاویر ساخته شده توسط generator با تکنیک smoothing در اپاک ۱۶



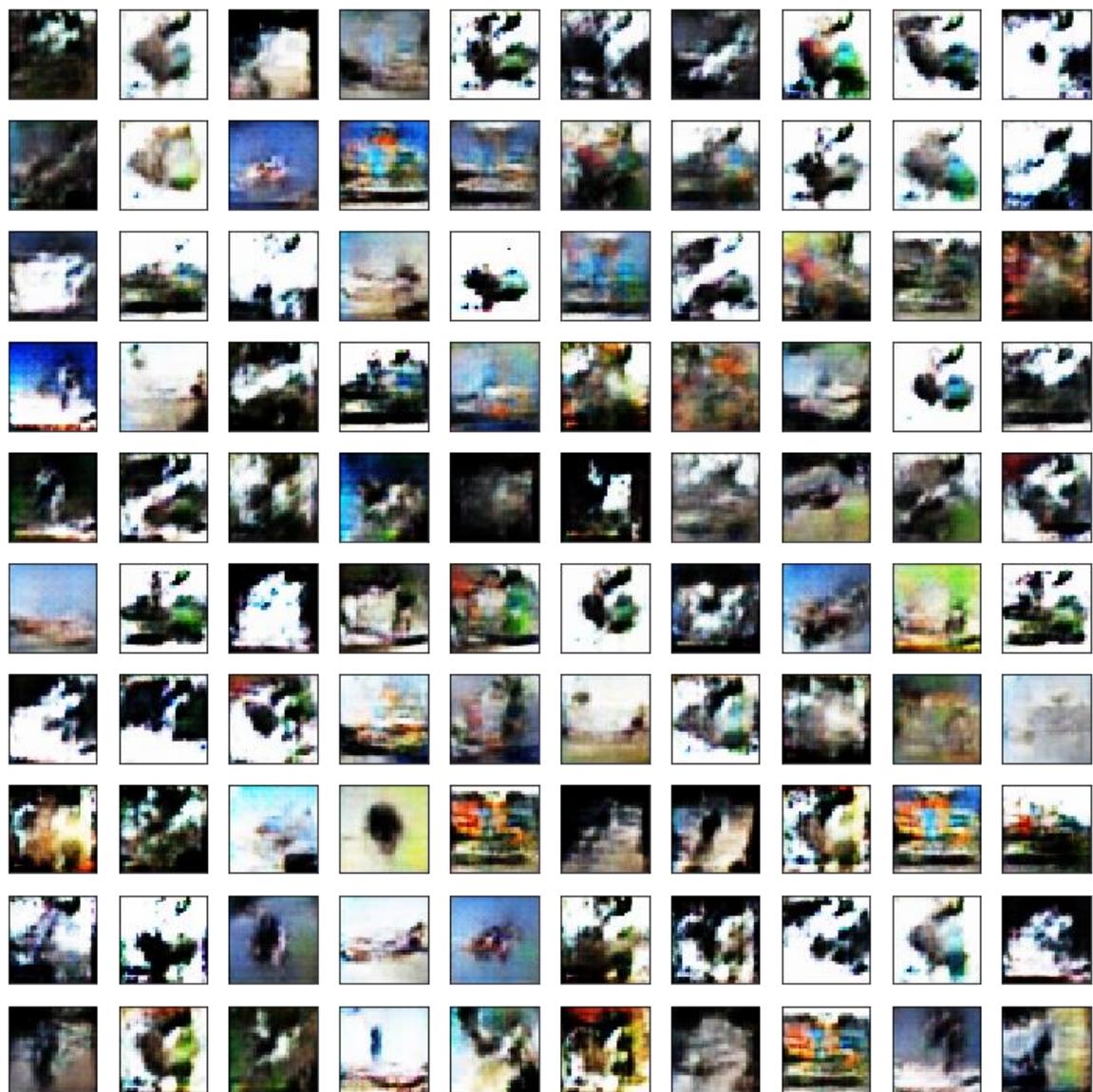
شکل - نمونه تصاویر ساخته شده توسط **generator** با تکنیک **smoothing** در ایپاک ۲۱



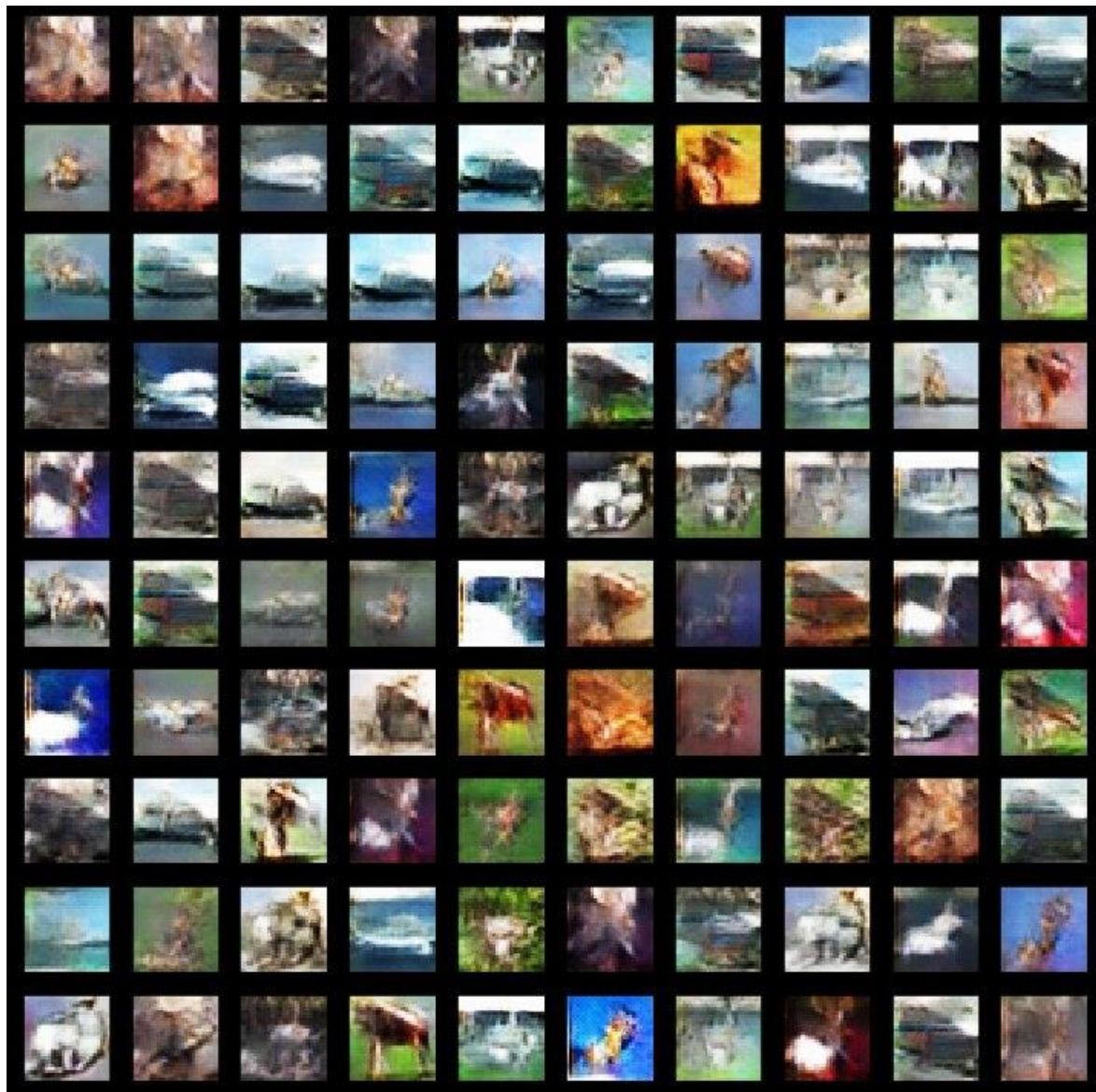
شکل - نمونه تصاویر ساخته شده توسط generator با تکنیک smoothing در ایپاک ۲۶



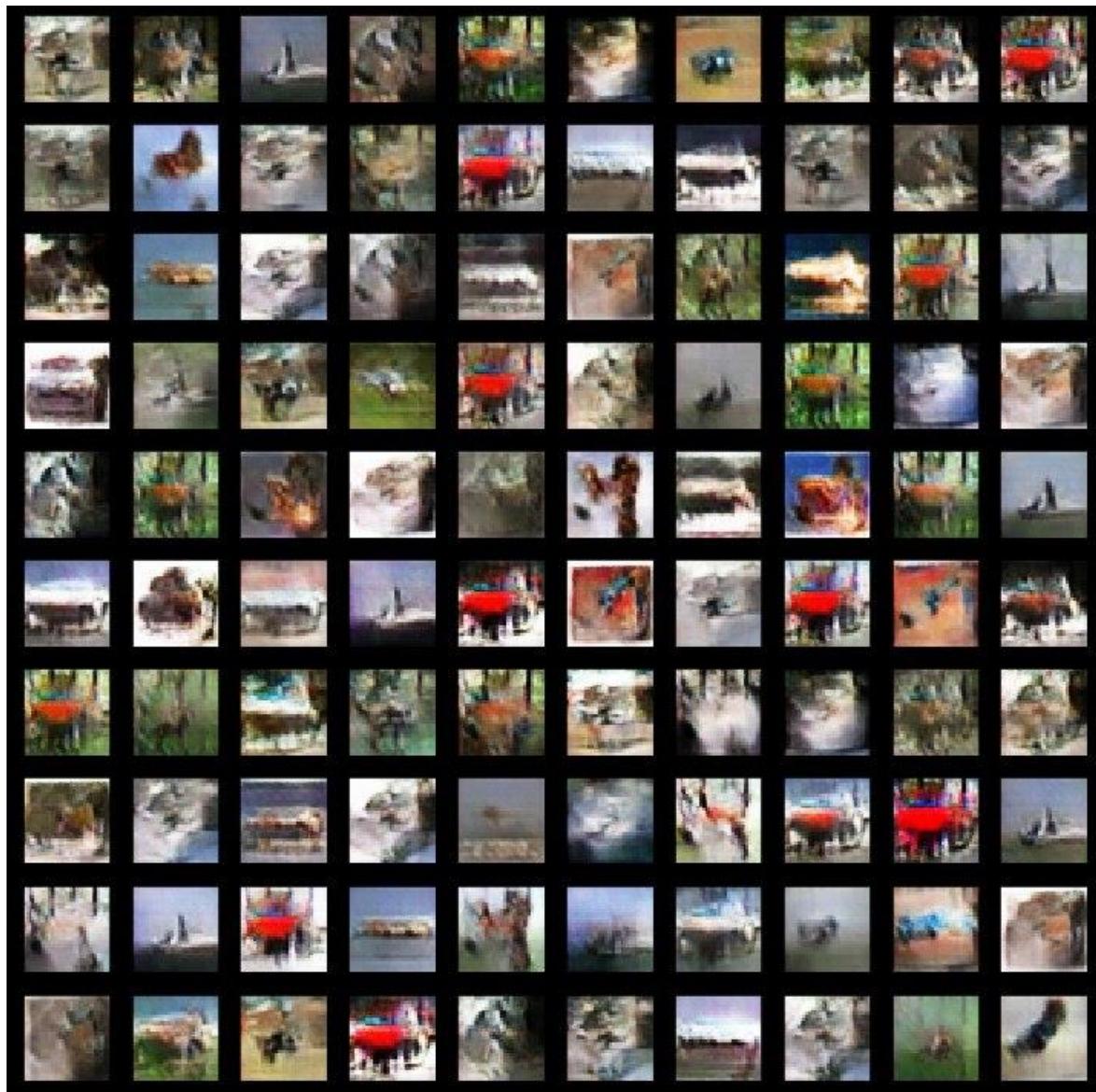
شکل - نمونه تصاویر ساخته شده توسط **generator** با تکنیک **smoothing** در ایپاک ۳۱



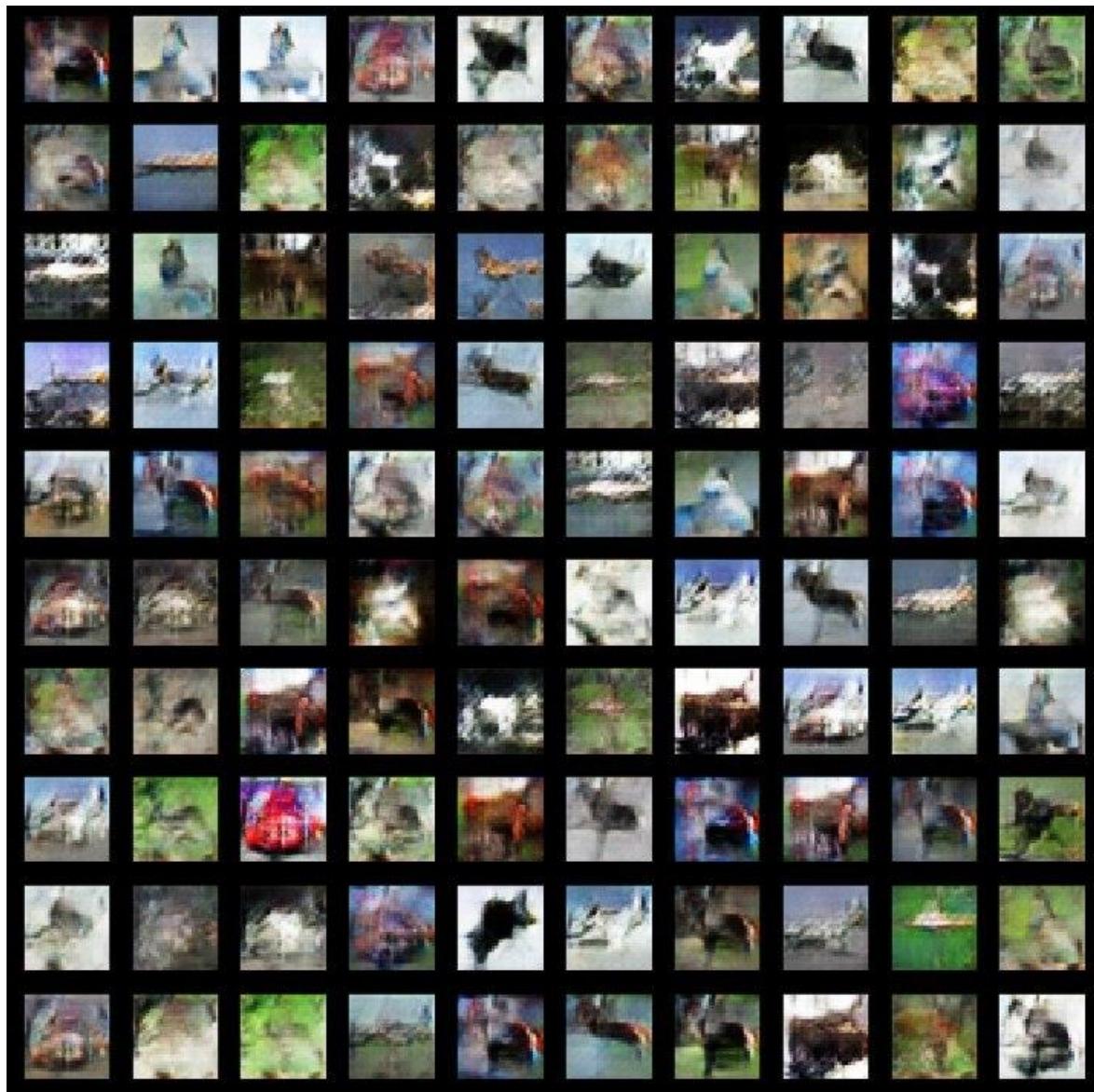
شکل - نمونه تصاویر ساخته شده توسط **generator** با تکنیک **smoothing** در ایپاک ۳۶



شکل - نمونه تصاویر ساخته شده توسط **generator** با تکنیک **smoothing** در اپاک ۴۱



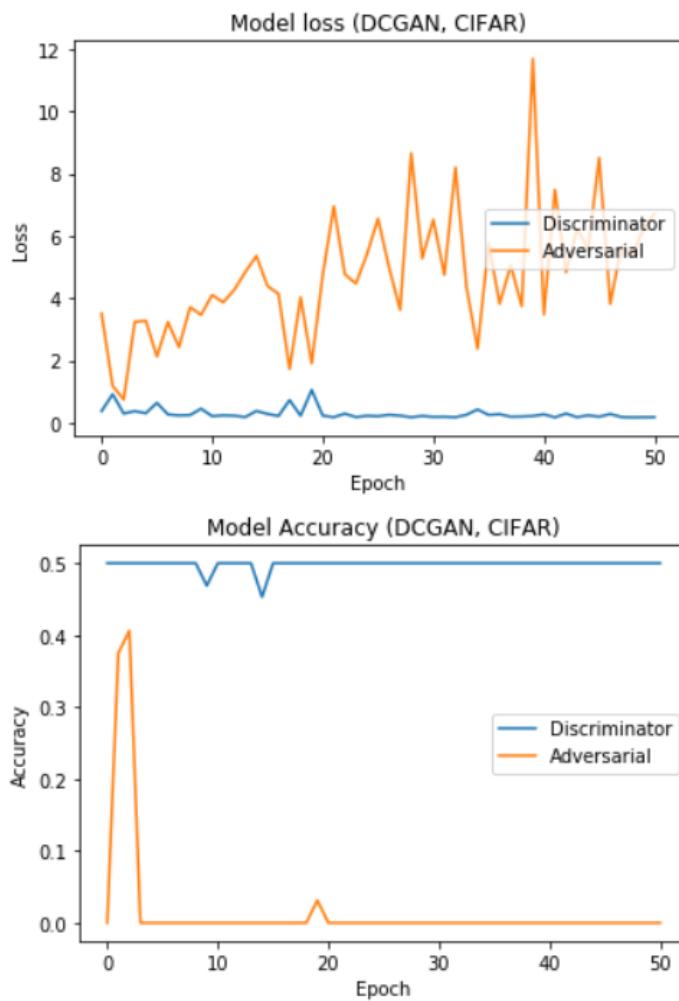
شکل - نمونه تصاویر ساخته شده توسط generator با تکنیک smoothing در ایپاک ۴۶



شکل - نمونه تصاویر ساخته شده توسط **generator** با تکنیک **smoothing** در ایپاک ۵۱

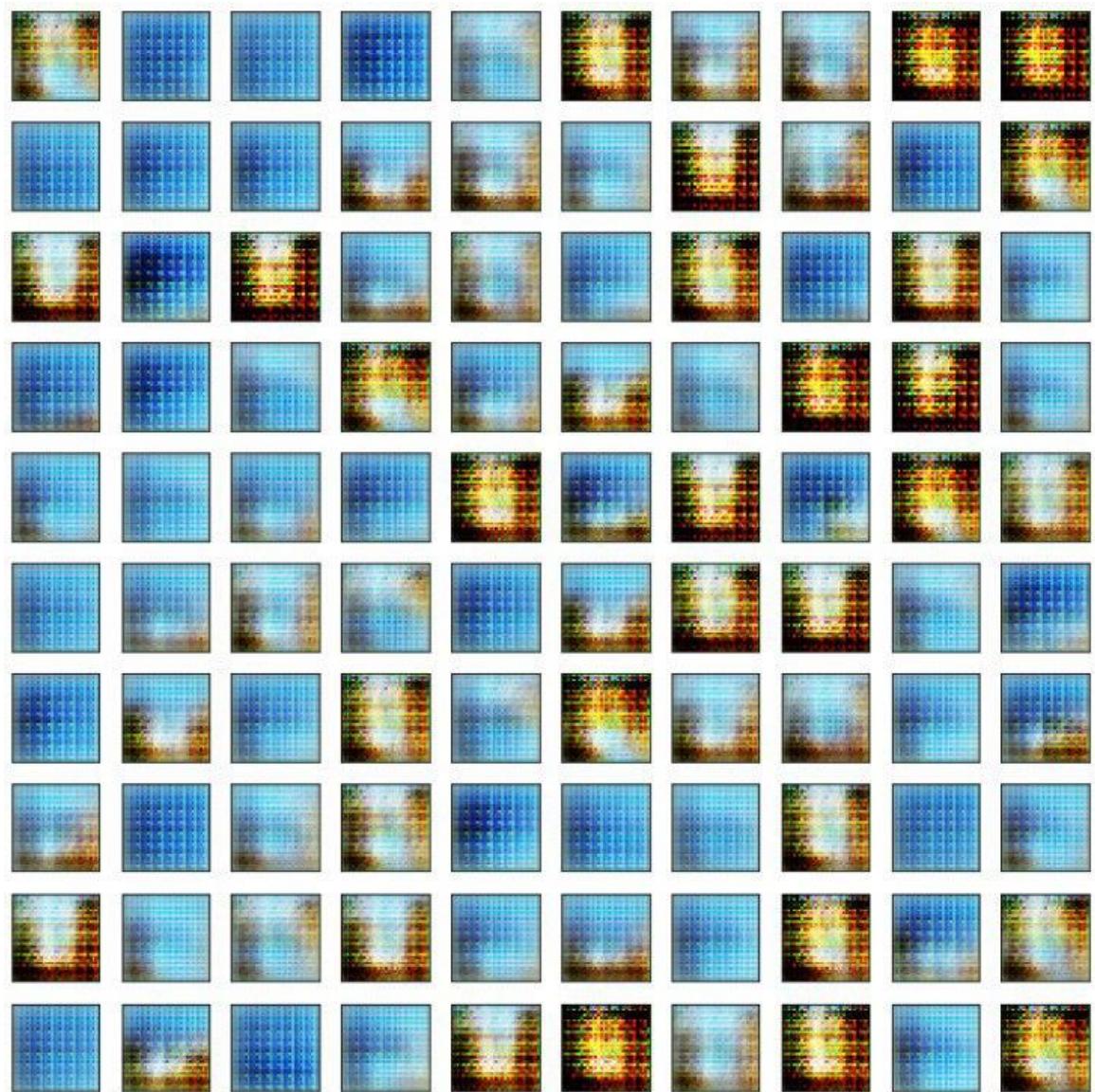
❖ استفاده از تکنیک noise در دیتاست cifar10

در این بخش همان معماری استفاده شده برای cifar10 به کار رفته است. با این تفاوت که در هنگام تعریف نویز، از نویز normal استفاده شده است. میانگین و انحراف معیار نویز به ترتیب 0 و 1 است. در ادامه نمودار مربوط به خطای دقت برای ۵۰ ایپاک و batch-size=32 آمده است. همانطور که مشاهده می شود، Loss پیوسته برای هر دو متغیر افزایش و کاهش می یابد.



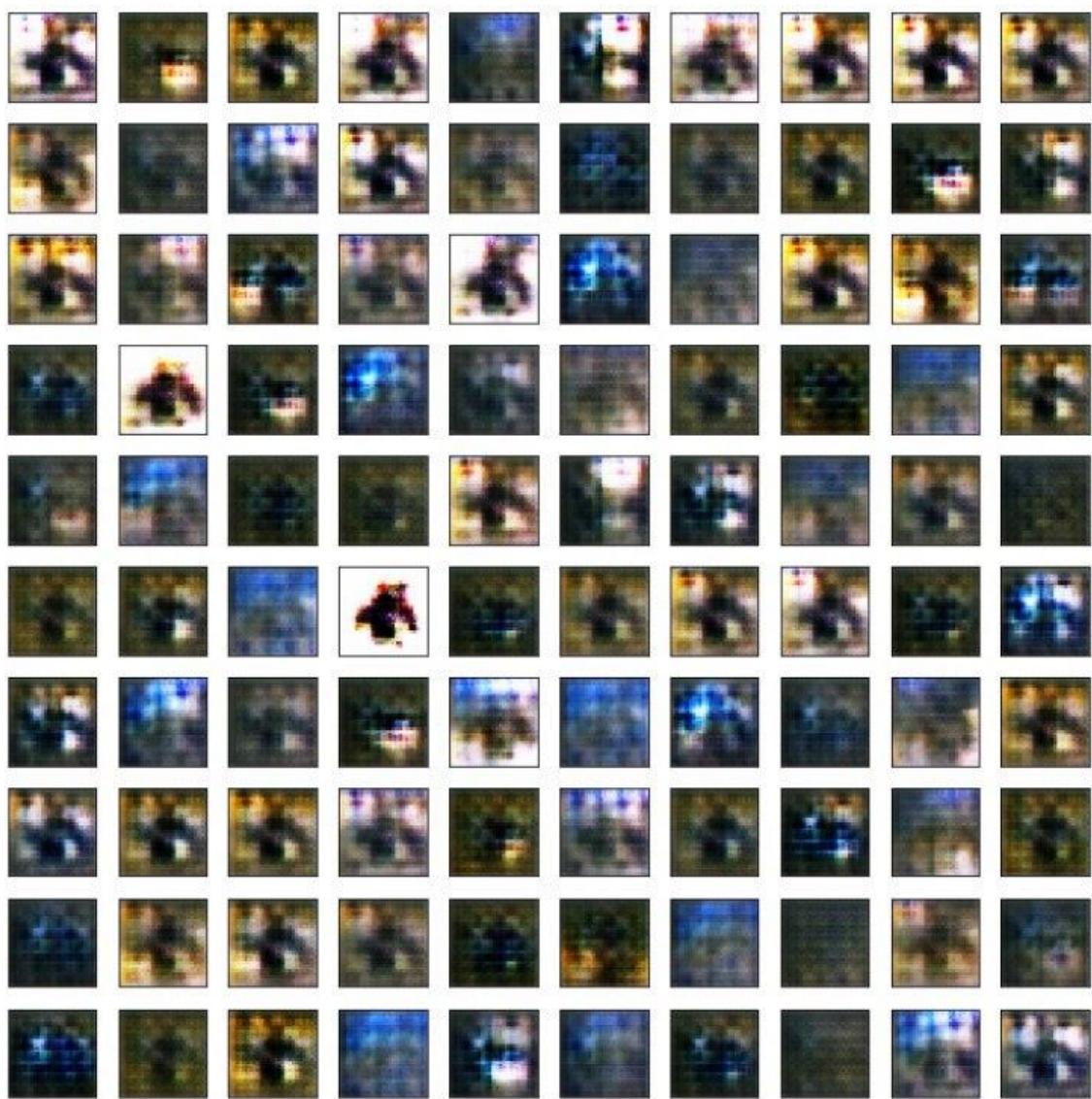
شکل - نمودار خطأ و دقت برای cifar10 با استفاده از تکنیک نویز

در ادامه ۱۰ نمونه از تصاویر ساخته شده برای دیتاست cifar10 طی ۵۰ ایپاک و روند بهبود آنها قابل مشاهده است.

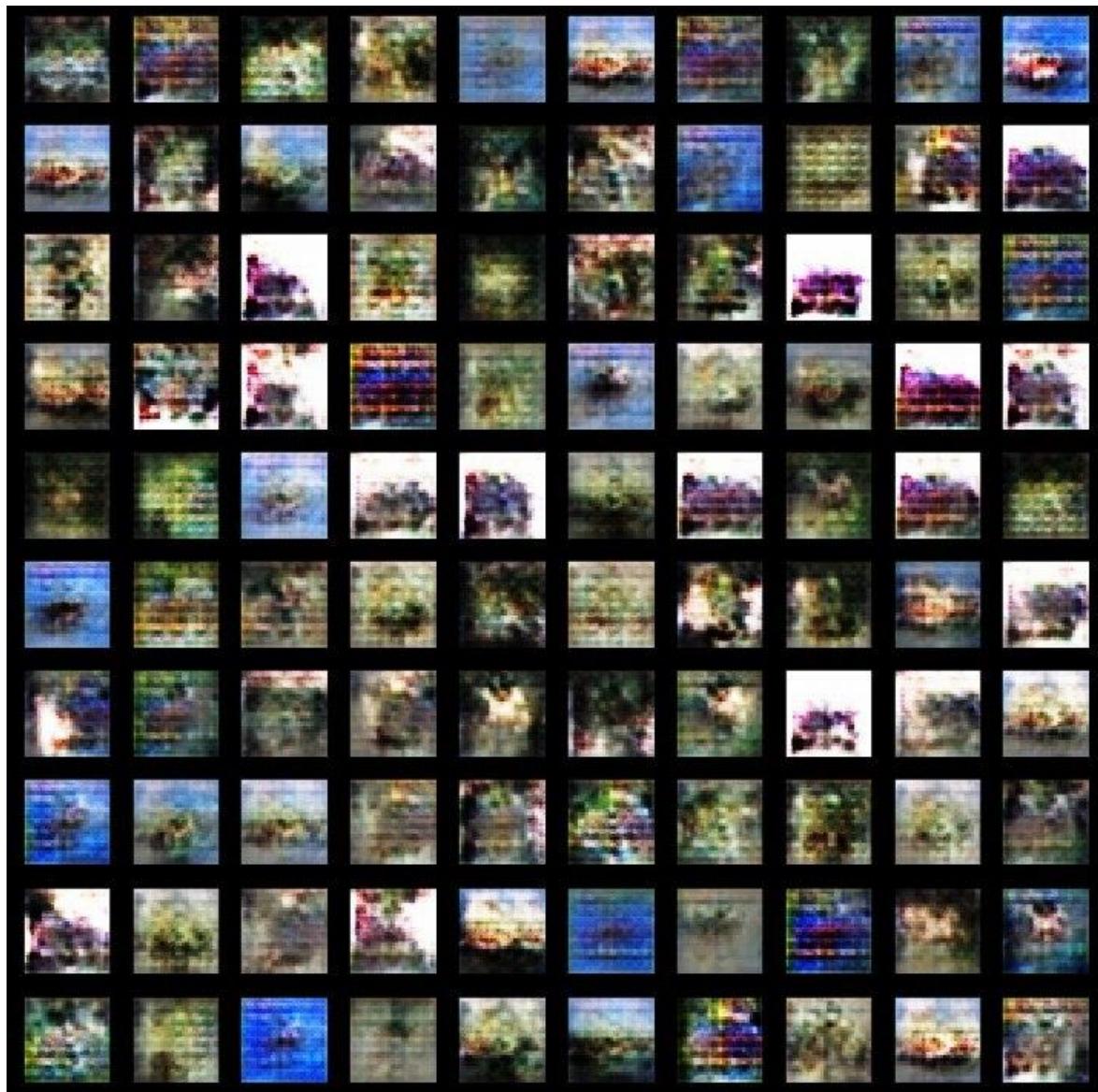


شکل - نمونه تصاویر ساخته شده توسط generator با تکنیک نویز در ایپاک ۱

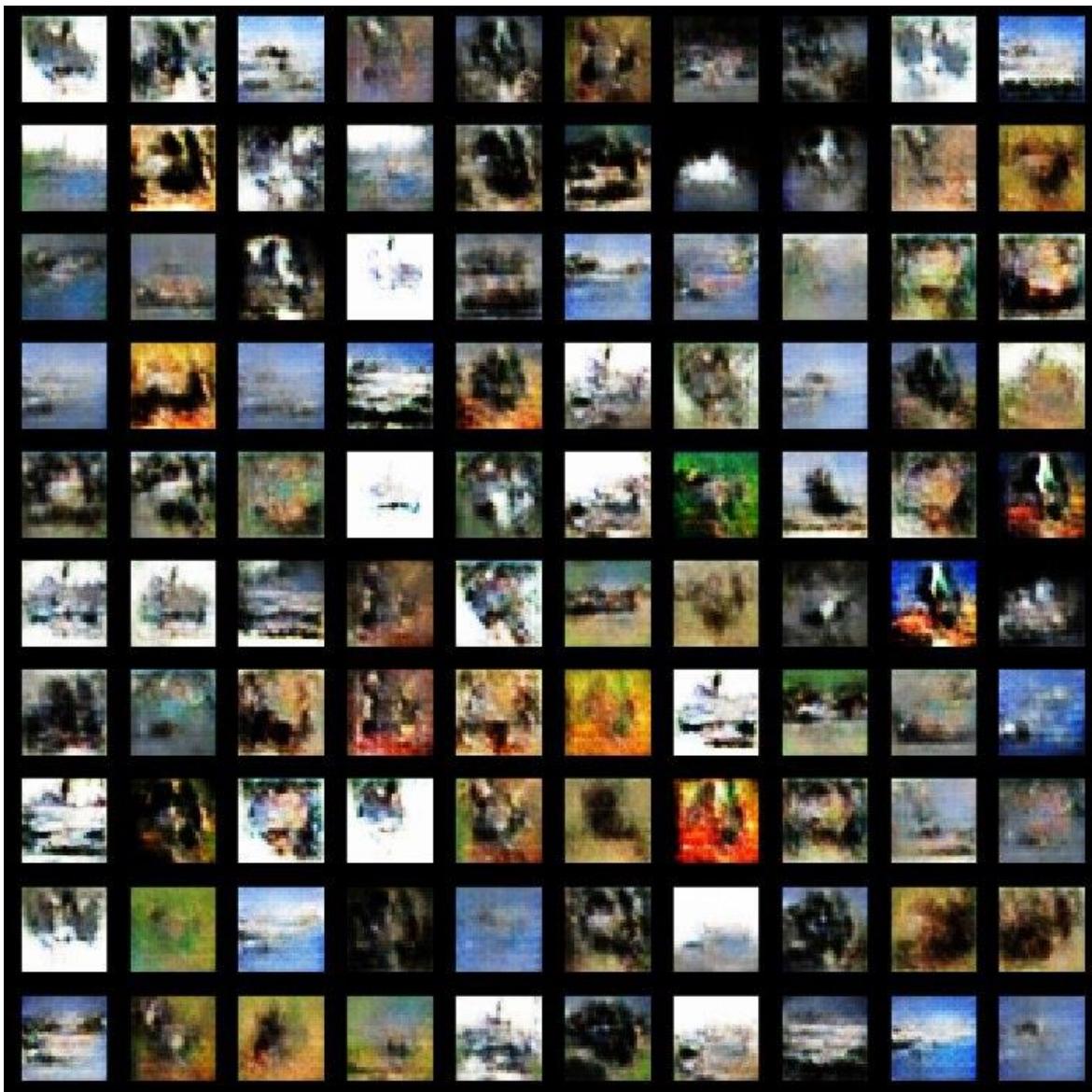
```
epoch = 6/50, d_loss=0.230, g_loss=1.614
```



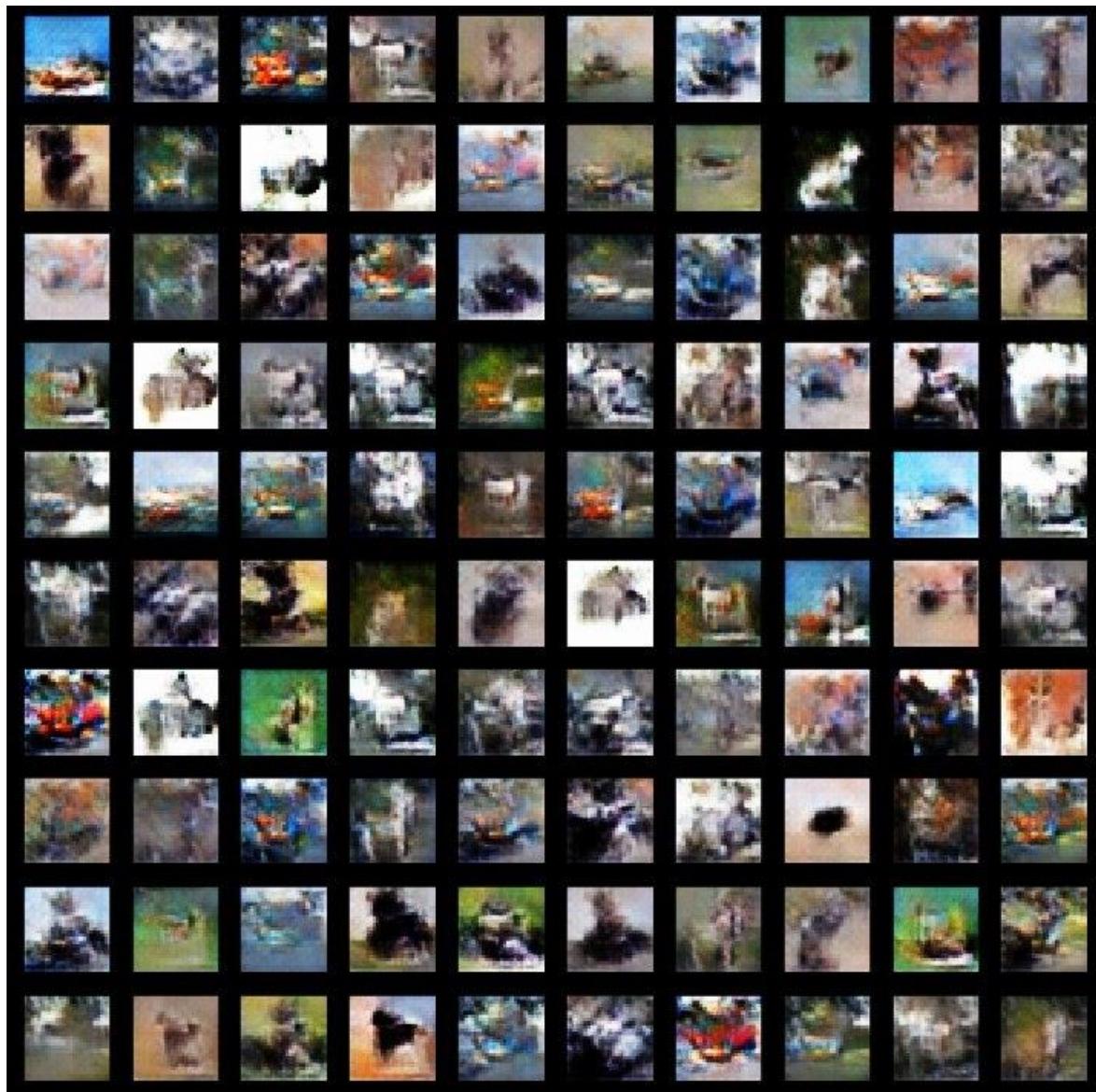
شکل - نمونه تصاویر ساخته شده توسط generator با تکنیک نویز در ایپاک ۶



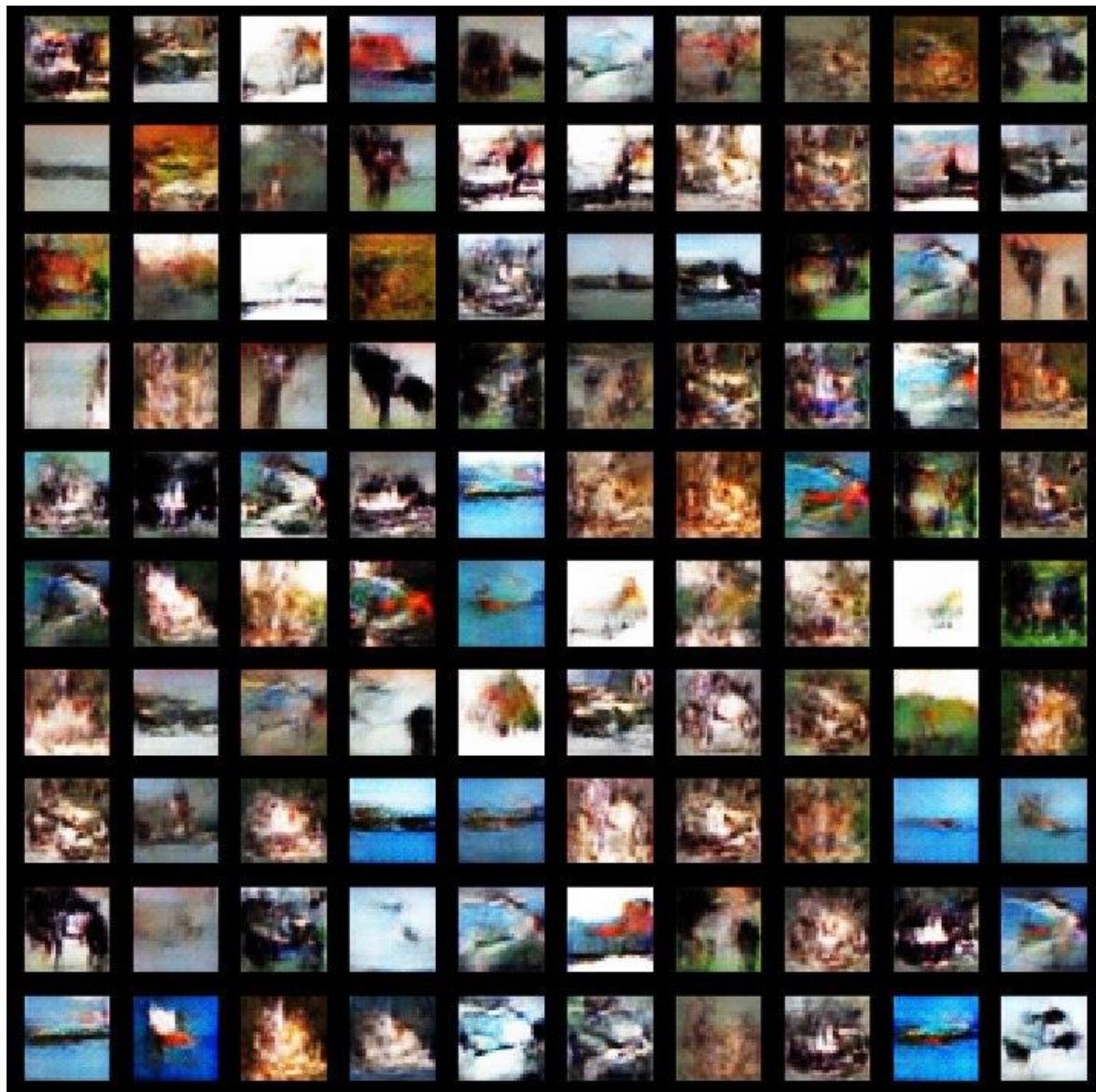
شکل - نمونه تصاویر ساخته شده توسط generator با تکنیک نویز در ایپاک ۱۱



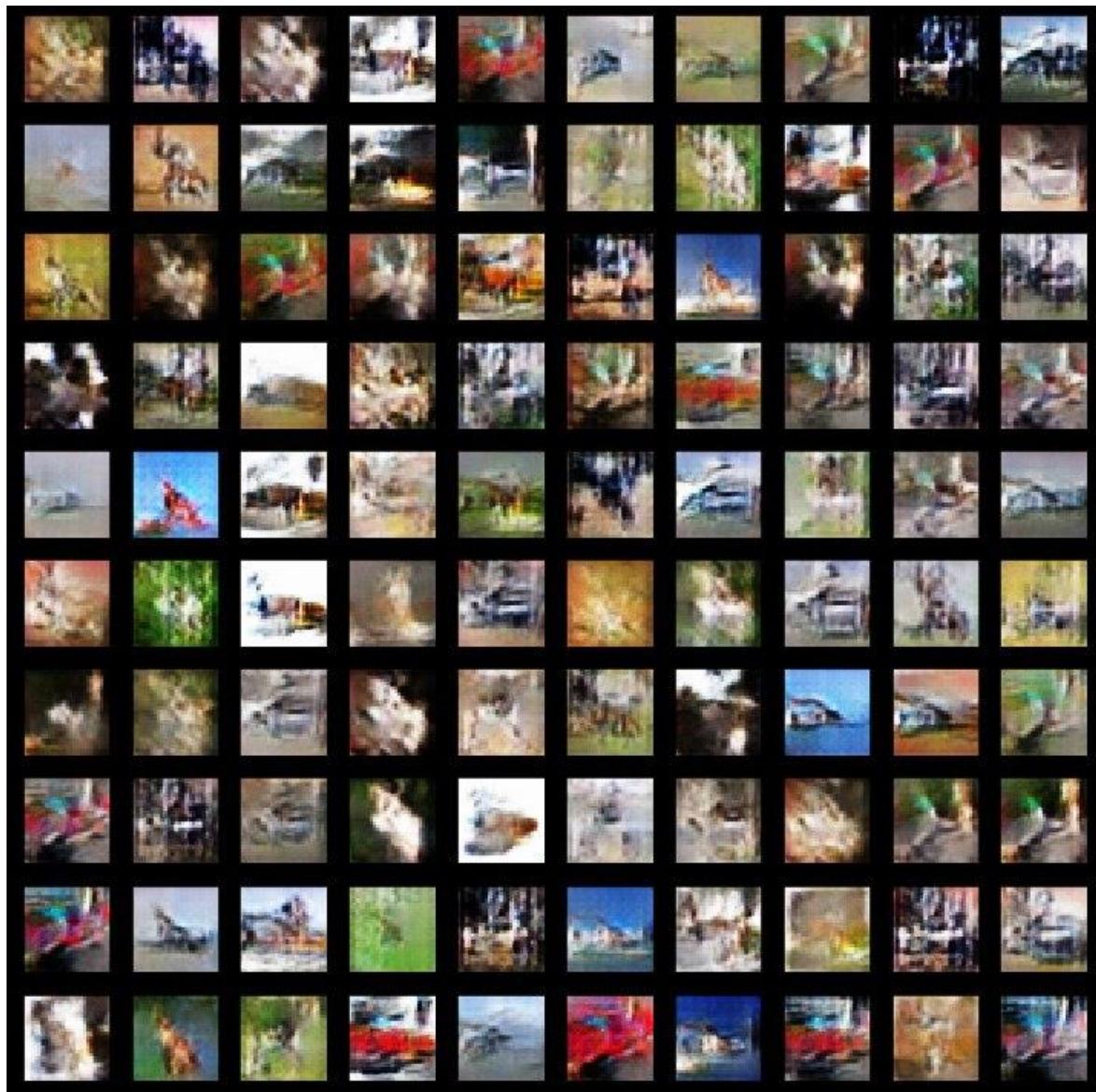
شکل - نمونه تصاویر ساخته شده توسط generator با تکنیک نویز در ایپاک ۱۶



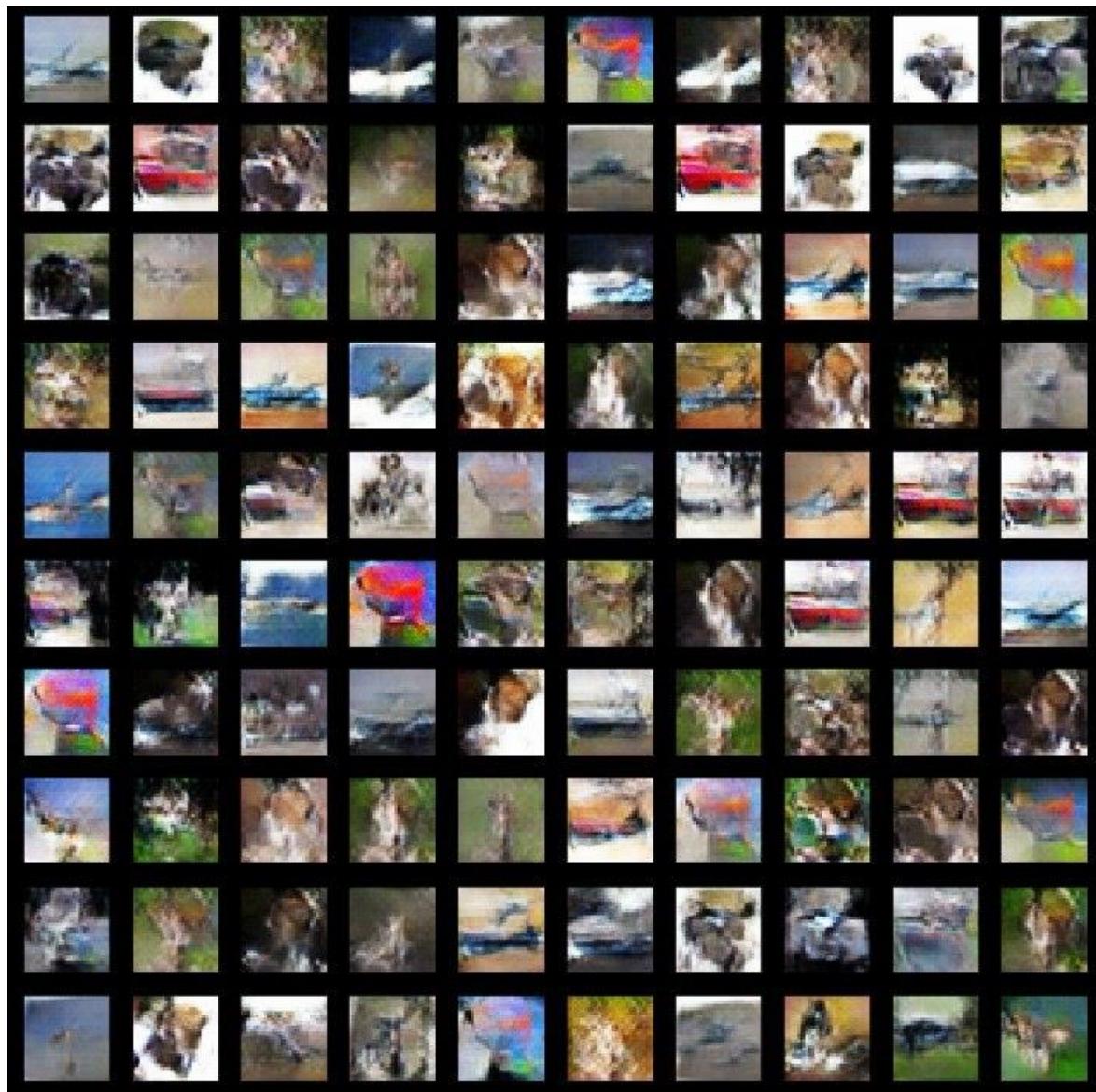
شکل - نمونه تصاویر ساخته شده توسط generator با تکنیک نویز در ایپاک ۲۱



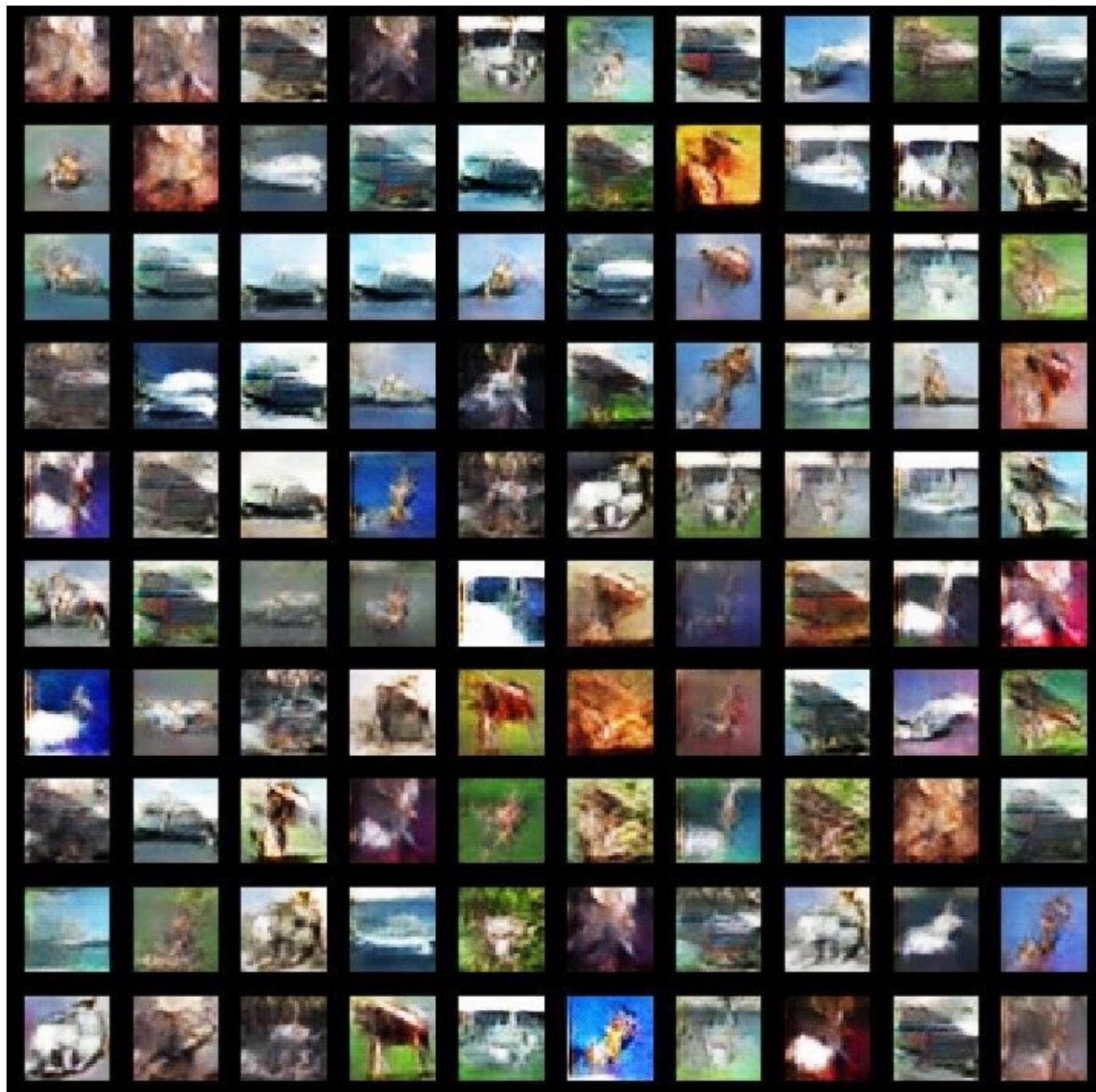
شکل - نمونه تصاویر ساخته شده توسط generator با تکنیک نویز در ایپاک ۲۶



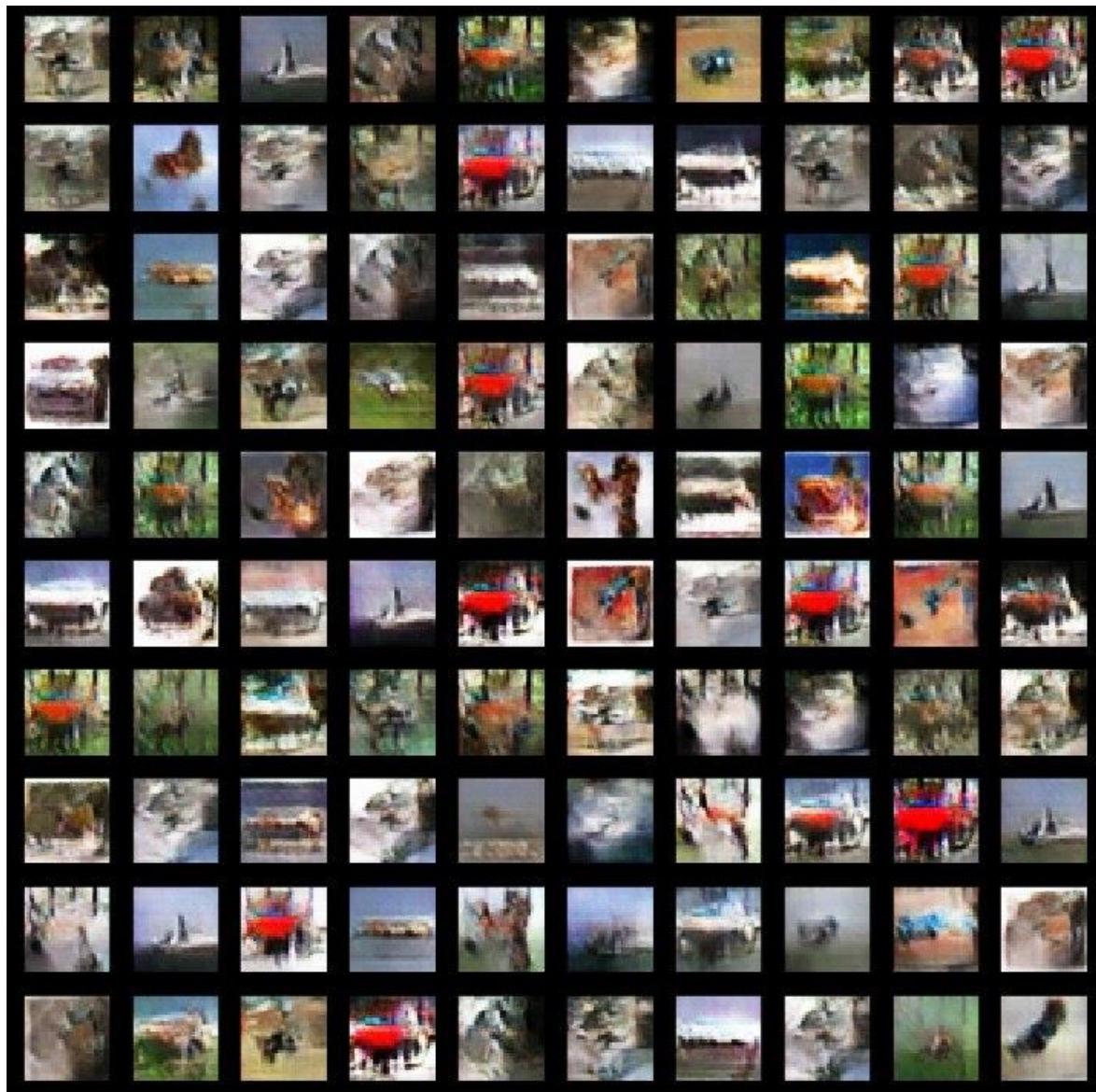
شکل - نمونه تصاویر ساخته شده توسط generator با تکنیک نویز در ایپاک ۳۱



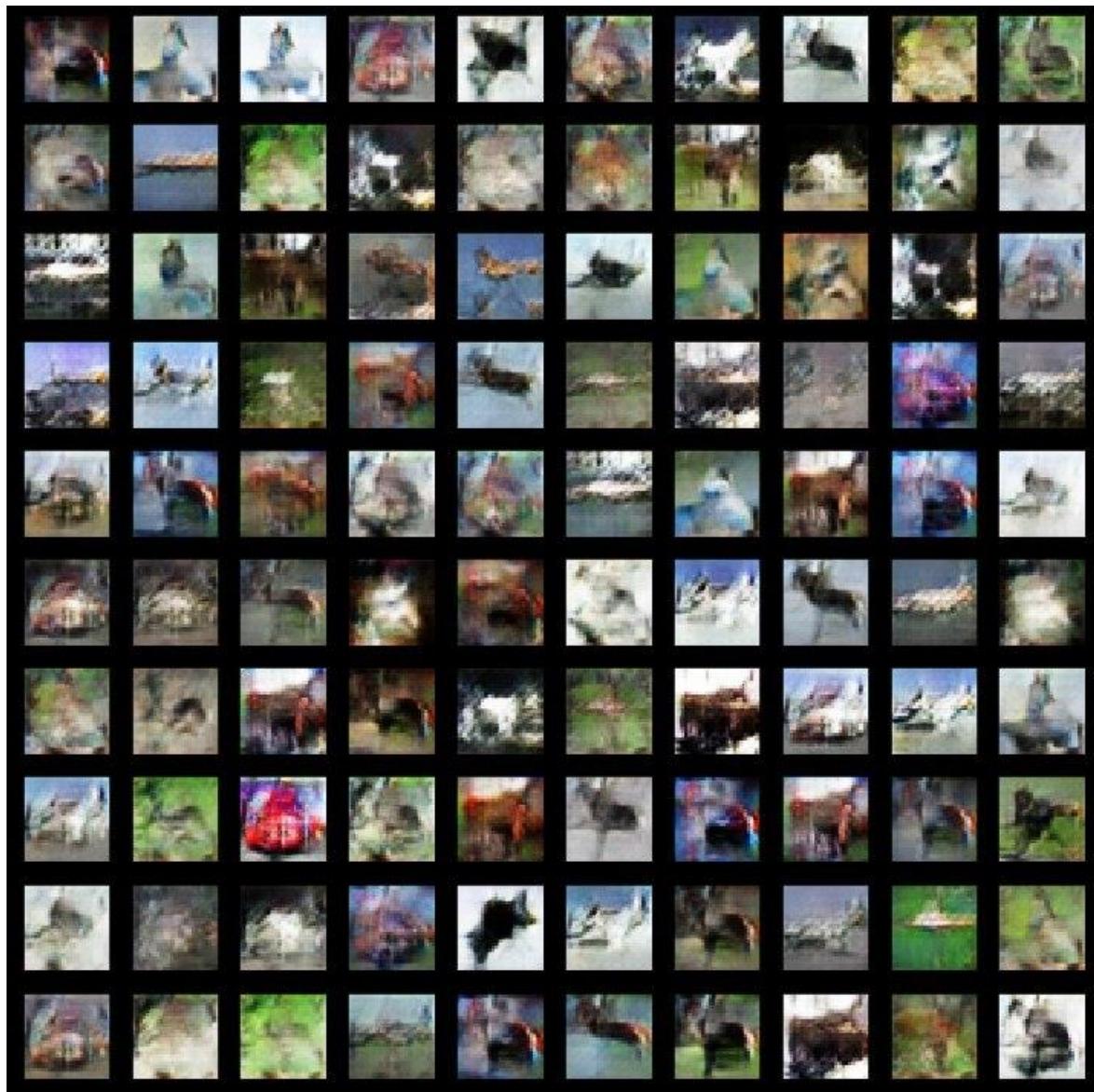
شکل - نمونه تصاویر ساخته شده توسط generator با تکنیک نویز در ایپاک ۳۶



شکل - نمونه تصاویر ساخته شده توسط generator با تکنیک نویز در ایپاک ۴۱



شکل - نمونه تصاویر ساخته شده توسط generator با تکنیک نویز در ایپاک ۴۶



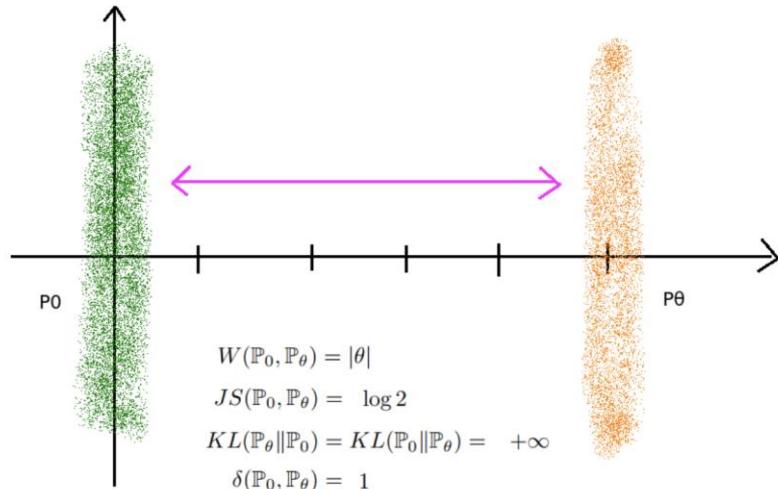
شکل - نمونه تصاویر ساخته شده توسط generator با تکنیک نویز در ایپاک ۵۱

ب) شبکه WGAN

❖ تعریف تابع خطا Wasserstein Loss

در شبکه‌های generative، فاصله بین توزیع داده‌های دیتاست و داده‌های ساخته شده توسط generator را کاهش می‌یابد. معیارهای اندازه‌گیری فاصله دو توزیع مختلف است. مثلاً در شبکه‌های VAE از معیار Kullback-Leibler و در شبکه‌های GAN از Jenson-Shannon استفاده می‌شود. تفاوت عمدی WGAN با GAN استاندارد تابع خطا است که در WGAN از Earth-Mover distance یا Wasserstein distance استفاده می‌شود. حال استفاده از تابع خطا Wasserstein منجر به داشتن

گرادیان خطی و پیوسته می‌شود. درواقع مفهوم فاصله EM، میزان کاری است که لازم است انجام شود تا یک توزیع جابه‌جا گردد بنابراین مقادیر فاصله مثبت خواهد بود.



شکل - نحوه محاسبه فواصل گفته شده

حال برای محاسبه تابع هزینه Wasserstein برای discriminator (در WGAN معمولاً از لفظ critic به جای discriminator استفاده می‌شود) و generator به صورت زیر عمل می‌شود:

critic loss = [average critic score on real images] – [average critic score on fake images]
generator loss = – [average critic score on fake images]

حال برچسب خروجی critic برای داده‌های fake و real به صورت 1- و 1 است. بنابراین تابع خطابه صورت زیر محاسبه می‌گردد.

wasserstein loss = Label * average critic score

❖ پیاده‌سازی شبکه WGAN

- تپیلوژی لایه‌های مولد و تفکیک‌کننده

تپیلوژی استفاده شده در WGAN برای دو بخش generator و critic همانند DCGAN است. بنابراین ورودی generator یک نویز با ابعاد 100 می‌باشد که به یک لایه fully connected با تعداد نورون 2*2*512 برد و در آنجا به استفاده از reshape کردن، ابعاد به صورت 2*512*2*2 می‌اید. حال از 4 لایه استفاده شده است که همانند Cinv2DTranspose upsampling بوده و عمل deconv را انجام می‌دهند و ابعاد تصویر را از 32*32*3 به 2*2*512 می‌رسانند. در لایه‌های deconv از stride=2 و kernel_size=3 استفاده شده است که منجر می‌شود ابعاد تصاویر طی هر لایه دوبرابر شوند و با استفاده padding=same

از تعداد فیلترها هر بار عمق آن‌ها نیز نصف گردد. در همه لایه‌ها، به جز لایه آخر از تابع فعال‌ساز leaky relu با ضریب 0.2 استفاده شده است. در لایه آخر از تابع tanh استفاده شده است. همچنین بعد از هر لایه از normalization نیز استفاده شده است.

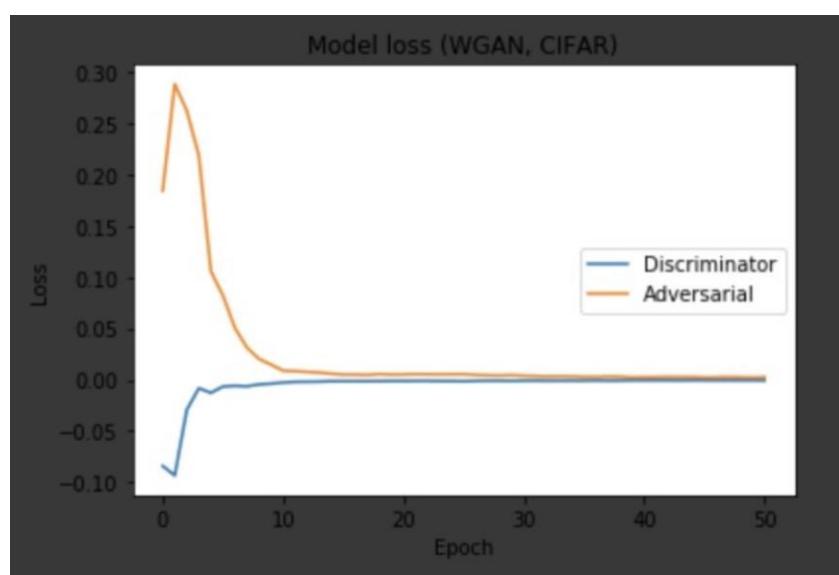
در بخش discriminator، ورودی‌های شبکه (دیتاست و ساخته‌شده توسط generator) باید دارای ابعاد 32*32 باشند. معماری مربوط به لایه‌های کانولوشنی discriminator بر عکس generator است. بنابراین ابعاد 32*32*3 طی ۴ لایه کانولوشنی با padding=same و stride=2 و kernel_size=3 و dense با ابعاد 512*2*2*32*32*3 طی ۴ لایه flat رسیده و پس از کردن از یک لایه mapping به صورت خطی بوده و از توابع فعال‌ساز leaky relu به جز لایه آخر است و در لایه آخر از leaky relu استفاده نشده است.

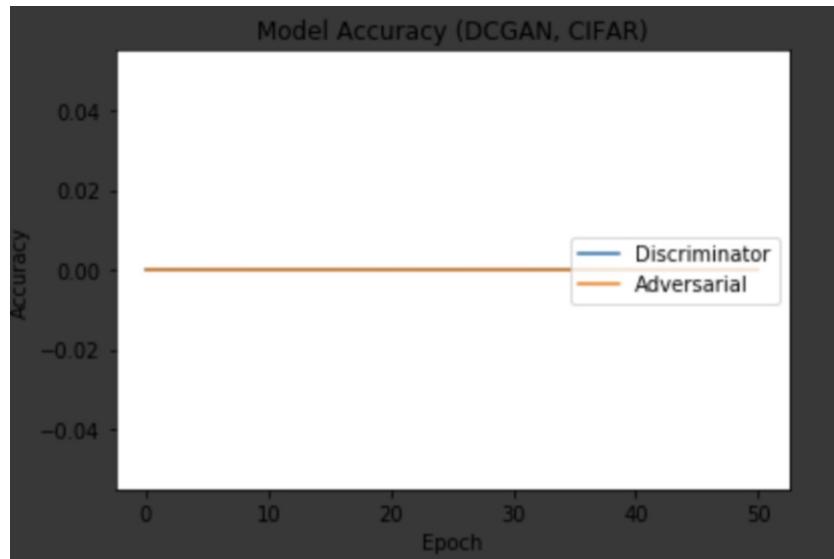
- نحوه ایجاد نویز

در این بخش از نویز random با میانگین ۰ و انحراف معیار ۱ استفاده شده است.

- نمودار Accuracy و Loss

در ادامه نمودار مربوط به خطا و دقت برای DCGAN طراحی‌شده در ۵۰ ایپاک و batch-size=64 آمده است. همانطور که مشاهده می‌شود، نمودار خطا به صورت نوسانی بوده و برای هر دو متغیر افزایش و کاهش می‌یابد.

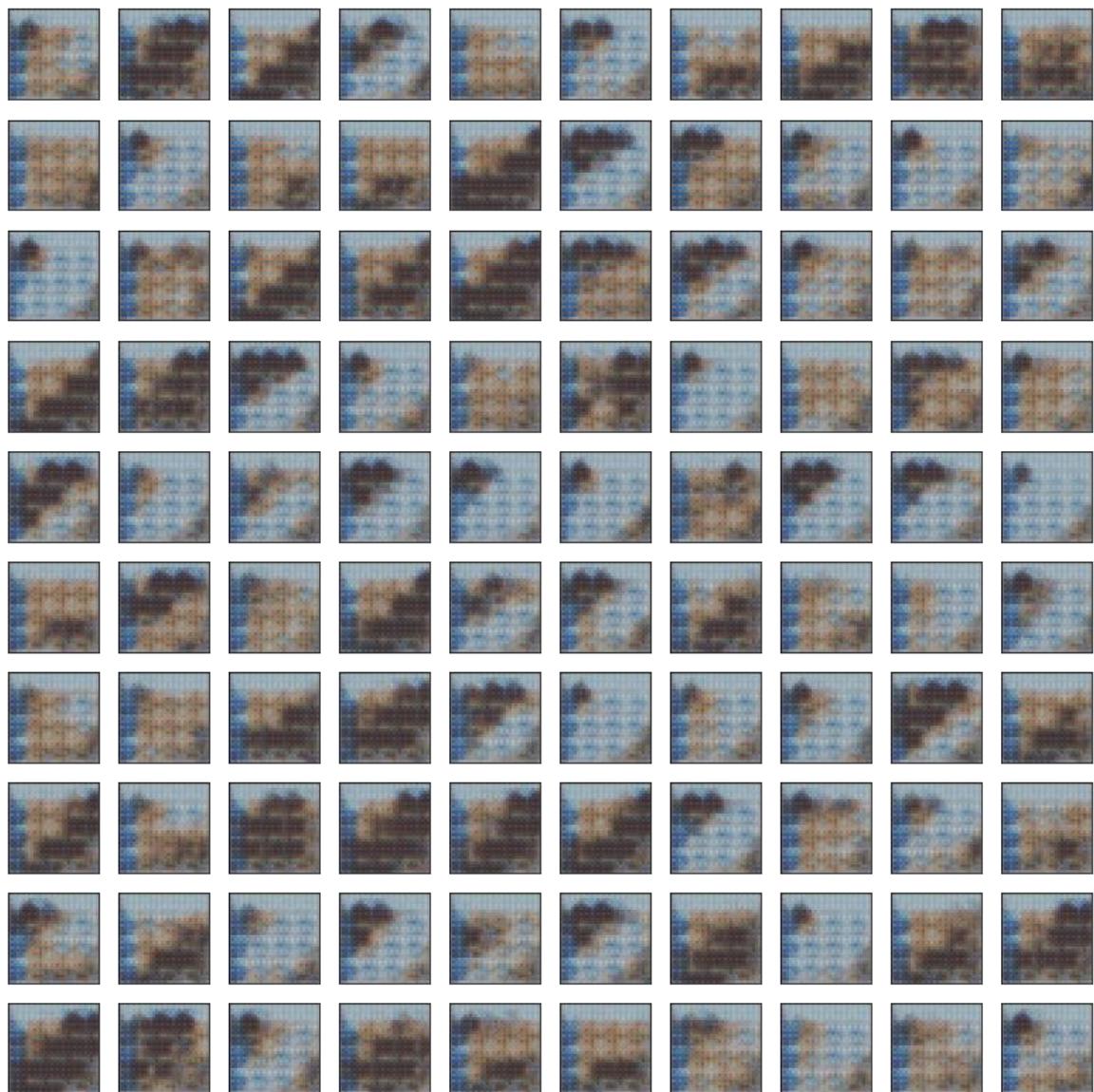




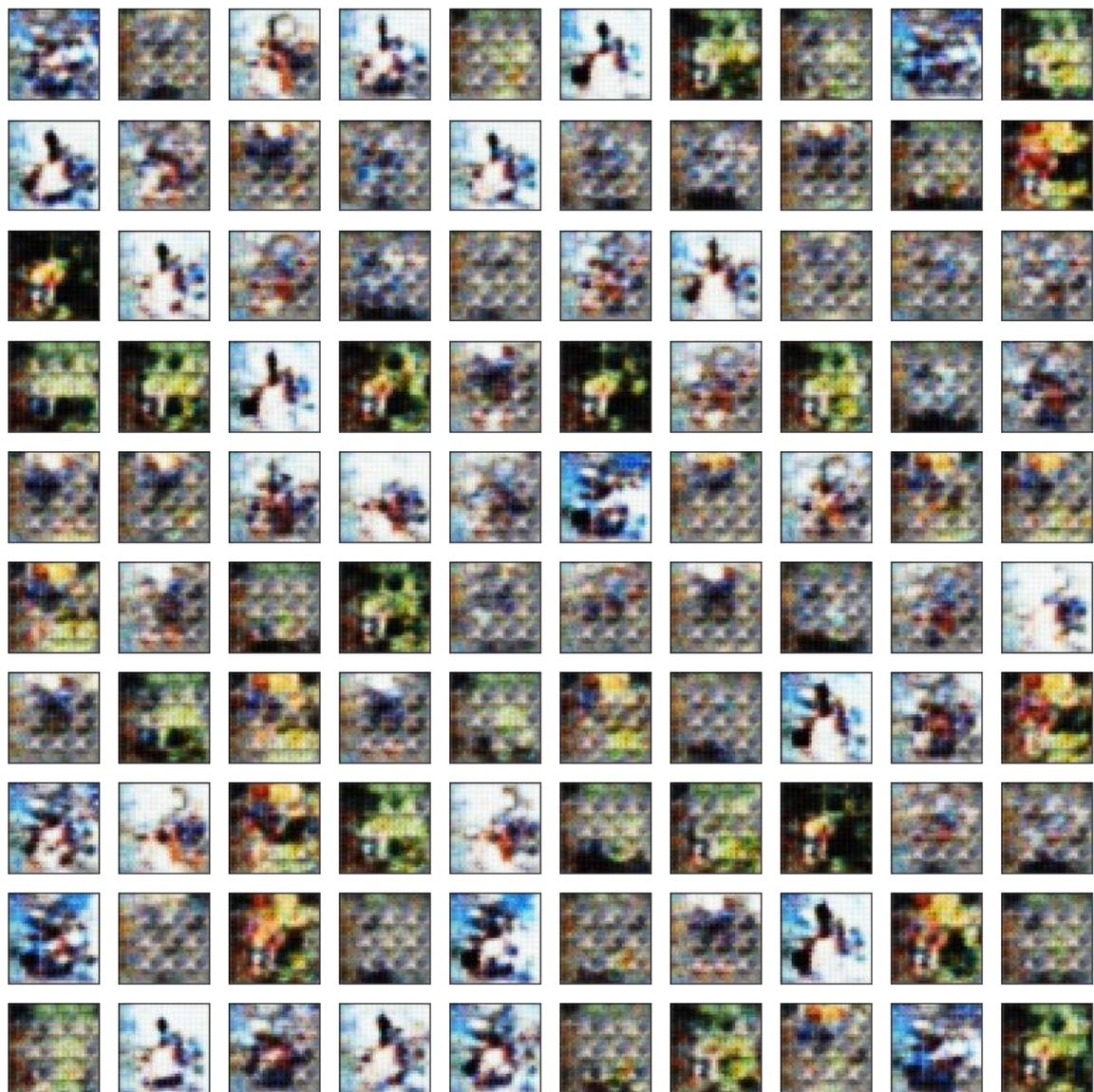
شکل - نمودار خطای دقت برای شبکه cifar10 برای WGAN

- نمونه نتایج شبکه

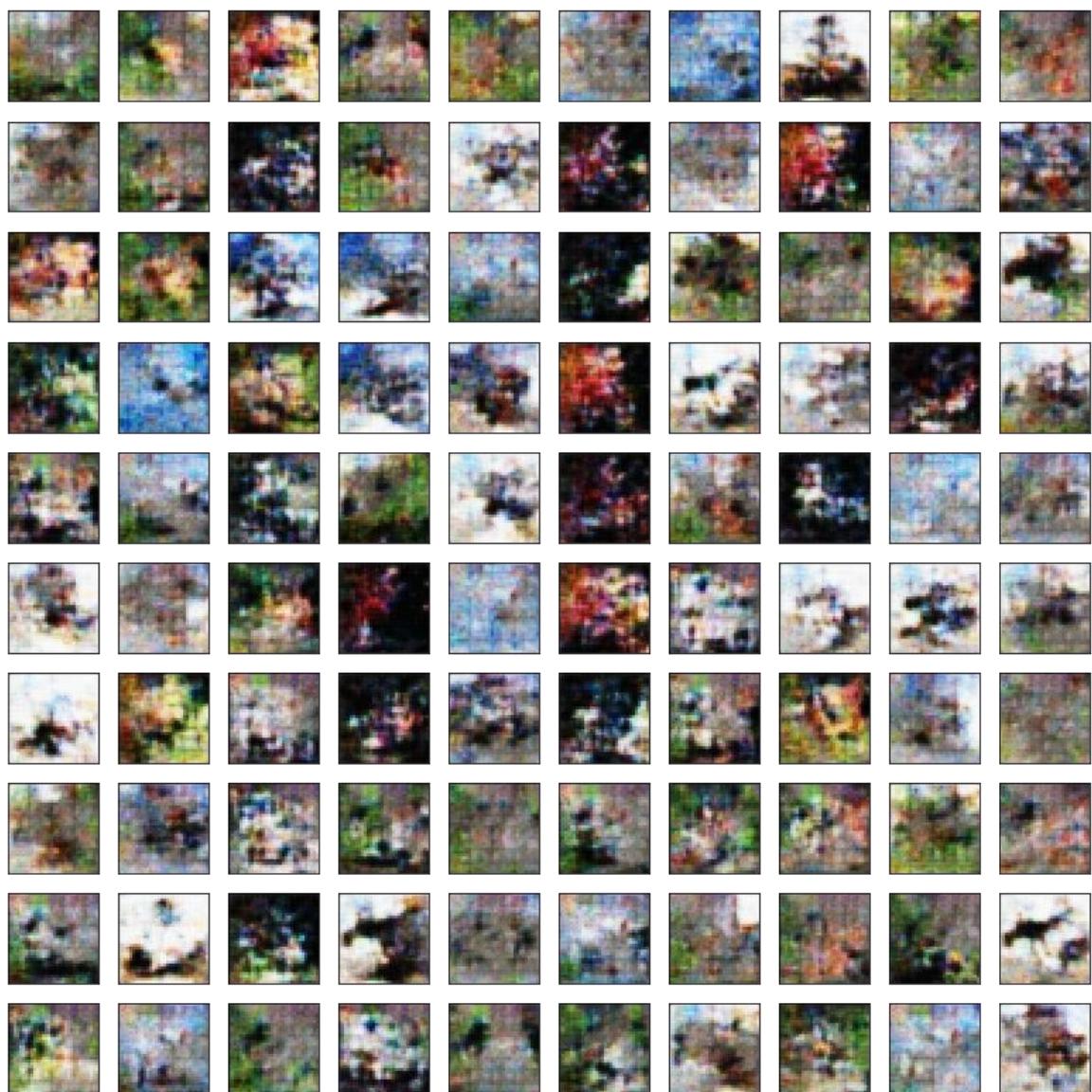
در ادامه ۵ نمونه از تصاویر ساخته شده توسط generator مربوط به شبکه WGAN برای دیتابست cifar10 طی ۵۰ ایپاک و روند بهبود آنها قابل مشاهده است.



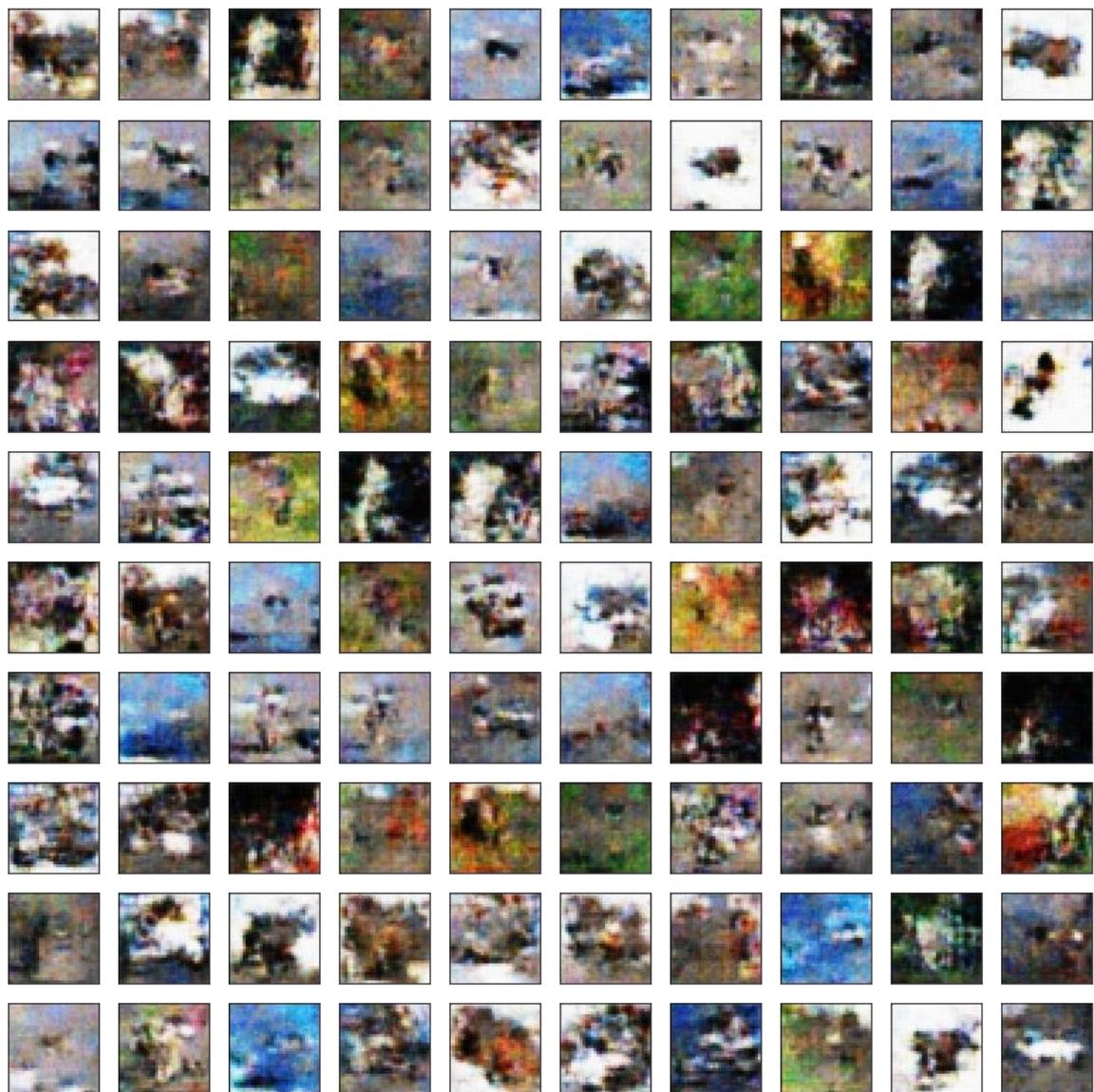
شکل - نمونه تصاویر ساخته شده توسط generator شبکه WGAN در ایپاک ۱



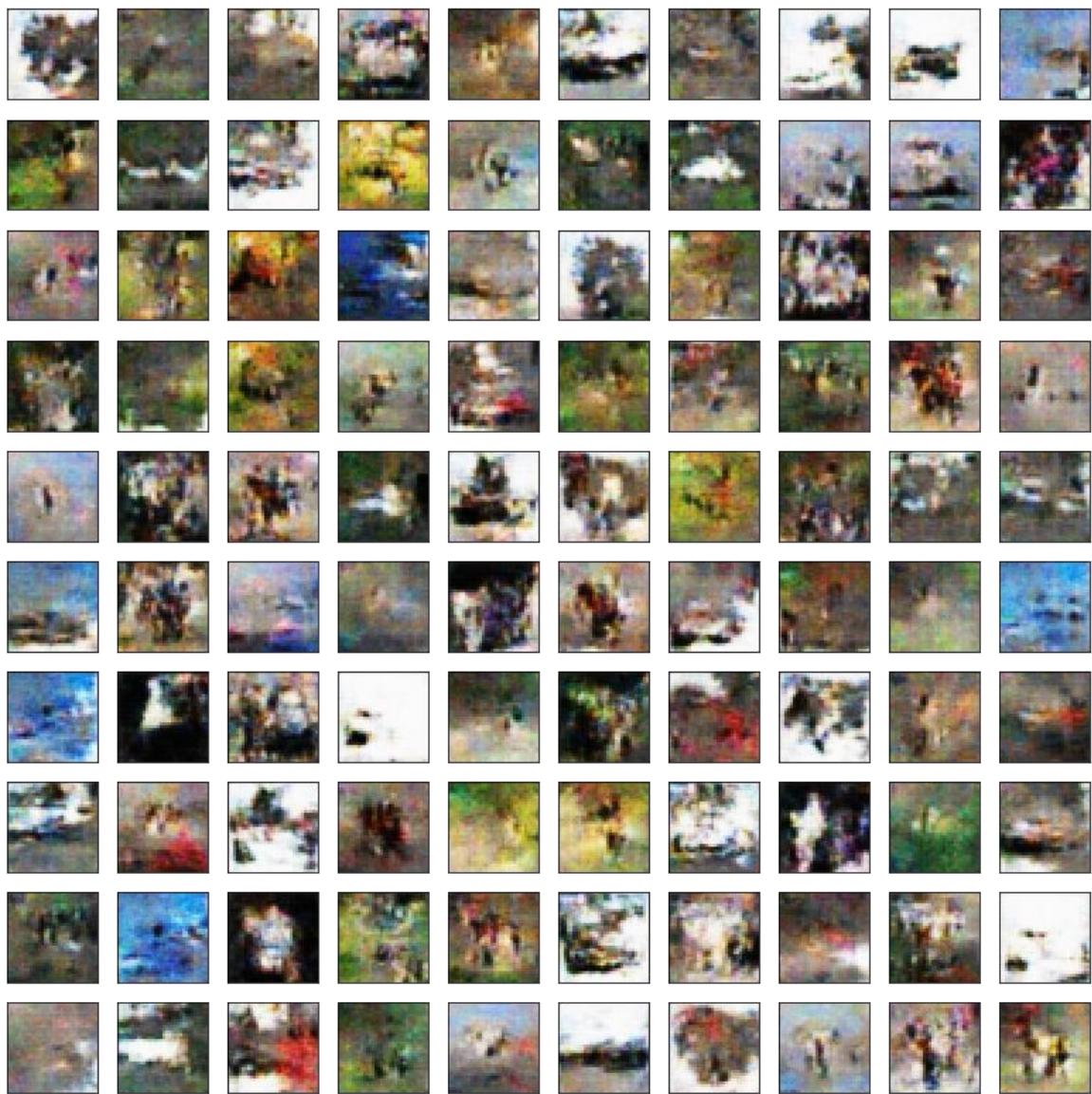
شکل - نمونه تصاویر ساخته شده توسط generator شبکه WGAN در ایپاک ۱۱



شکل - نمونه تصاویر ساخته شده توسط شبکه generator در ایپاک ۲۱



شکل - نمونه تصاویر ساخته شده توسط generator شبکه WGAN در ایپاک ۳۱



شکل - نمونه تصاویر ساخته شده توسط generator شبکه WGAN در ایپاک ۴۱

ج) شبکه ACGAN

- شبکه ACGAN

یکی از مشکلاتی که GAN با آن مواجه می‌شود آنست که امکان ساخت برخی کلاس‌های موجود در دیتاست بیشتر بوده و برخی کلاس‌ها کمتر. بدین صورت می‌توان label کلاس مدنظر را به عنوان ورودی به ACGAN اعمال کرد. همچنین از discriminator انتظار داشت که علاوه بر داده خروجی صفر یا یک

برای real یا fake بودن، کلاس مربوطه به تصاویر بدهد. در این حالت پایداری شبکه نیز افزایش یافته و می‌توان کنترل دقیق‌تری بر تولیدات GAN داشت.

- پیاده‌سازی یک شبکه ACGAN

متاسفانه پیاده‌سازی ACGAN بر روی دیتابست cifar10 همگرا نشده و در این بخش شبکه ACGAN بر روی fashion mnist پیاده‌شده است.

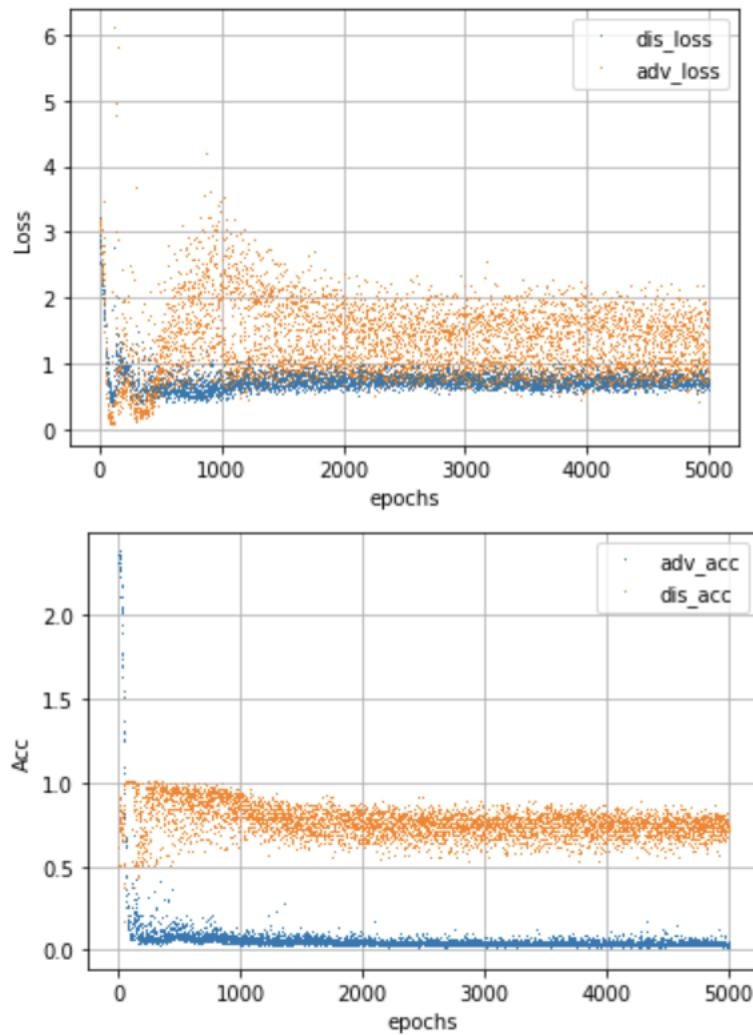
پس از فراخوانی دیتابست، توابع مربوط به generator و discriminator ساخته شده است. همانطور که پیشتر گفته شد، در acgan، شبکه generator دارای دو ورودی است. ورودی اول همان نویز با ابعاد فضای latent (در اینجا ۱۰۰) است. ورودی دوم generator برچسب‌هایی است که در انتهای generator باید کلاس مربوط به تصاویر ساخته شده توسط generator را نشان دهند. بنابراین برای این بخش از یک ماتریس همانی استفاده شده است.

در acgan discriminator شامل دو خروجی است؛ یکی برچسب real یا fake بودن، دیگری کلاس داده‌ها. بنابراین پس از اعمال لایه‌ها کانولوشنی، خروجی لایه flatten به دو لایه fully connected به ترتیب ۱۲۸ و ۱۰ نورون داده می‌شود. لازم به ذکر است تعداد کلاس‌های موجود در دیتابست cifar-10 ۱۰ کلاس است. در لایه dense با ۱۰ نورون از تابع فعال‌ساز softmax استفاده شده است. خروجی دیگر مثل dcgan بوده و یک لایه dense تکنرونی با تابع sigmoid است.

در هنگام compile کردن شبکه، از آنجایی که discriminator دارای دو خروجی است، از دو تابع هزینه برای خروجی‌های آن استفاده شود. بنابراین از categorical_crossentropy و binary_crossentropy برای خروجی‌های آن استفاده شده است. برای اعمال تابع هزینه categorical labelهای موجود در دیتابست به صورت one-hot شوند.

برای بهروزسازی generator لازم دو ورودی (نویز و برچسب‌های اولیه) به آن اعمال گردند و برای طراحی generator همانند dcgan از ۴ لایه deconv استفاده شده است.

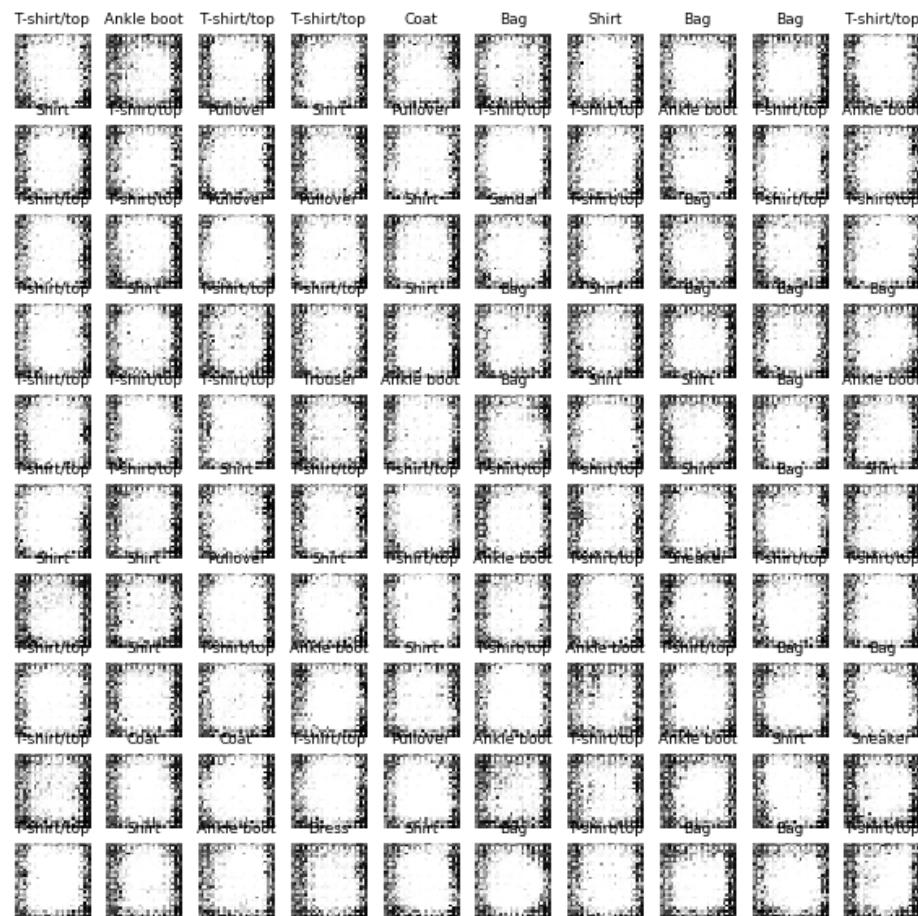
در ادامه نمودار مربوط به خطای دقت برای ACGAN طراحی شده در 5000 ایپاک و batch-size=64 آمده است.



شکل - نمودار خطا و دقت برای شبکه ACGAN برای Fashin MNIST

- نمونه نتایج شبکه

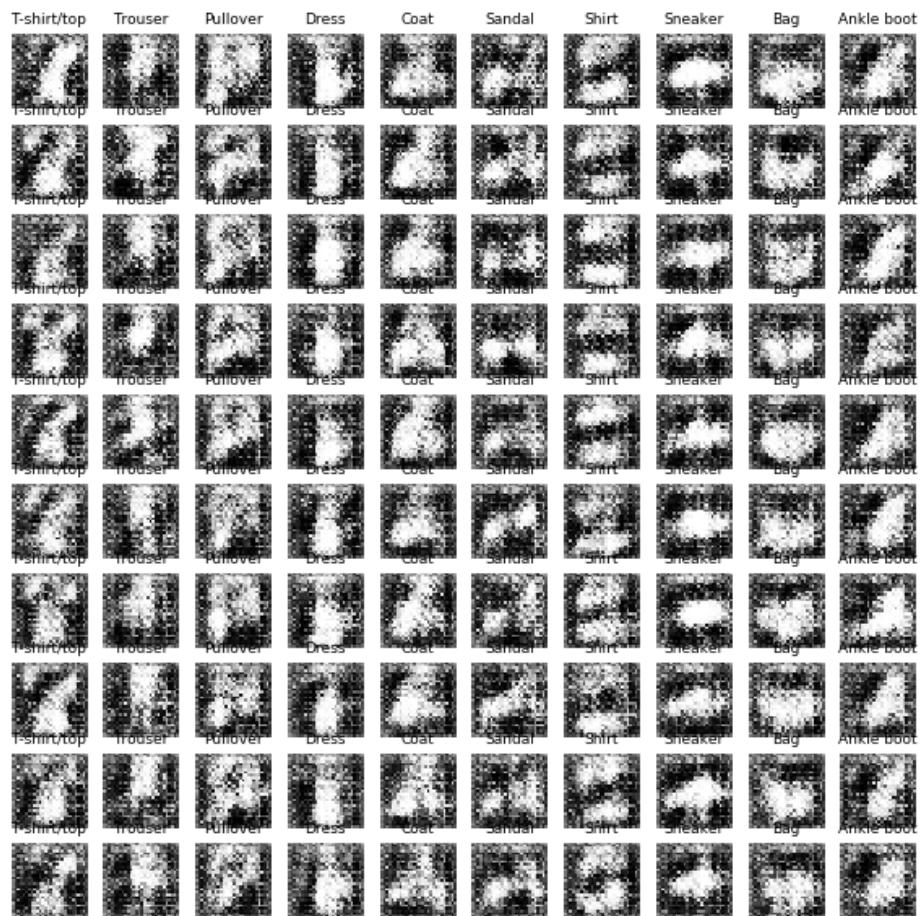
در ادامه ۵ نمونه از تصاویر ساخته شده توسط generator مربوط به شبکه ACGAN برای دیتابست Fashion MNIST طی ۴۰۰۰ ایپاک و روند بهبود آنها قابل مشاهده است.



شکل - نمونه تصاویر ساخته شده توسط شبکه generator ACGAN در ایپاک ۳۰



شکل - نمونه تصاویر ساخته شده توسط شبکه generator ACGAN در ایپاک ۵۰۰



شکل - نمونه تصاویر ساخته شده توسط generator شبکه ACGAN در ایپاک 1000



شکل - نمونه تصاویر ساخته شده توسط generator شبکه ACGAN در ایپاک ۲۰۰۰



شکل - نمونه تصاویر ساخته شده توسط generator شبکه ACGAN در ایپاک ۳۰۰۰



شکل - نمونه تصاویر ساخته شده توسط generator شبکه ACGAN در ایپاک ۴۰۰۰

د) مقایسه شبکه‌های طراحی شده

اگر خروجی شبکه های DCGAN و WGAN را در ایپاک 51 مقایسه کنیم، متوجه میشویم که عکس های خروجی از WGAN به مراتب بهتر از DCGAN تولید شده اند. چون در WGAN پارامتر loss تعريف شده است و بنابراین در خروجی نیز نتایج بهتری را نشان می دهد.

همچنین داشتن smoothing یا همان الگوریتم one-sided به بهتر شدن نتایج نیز کمک می کند که در نمودار های Loss و Accuracy قابل مشاهده می باشد.

همچنین اگر تصاویر fashion mnist برای DCGAN و برای ACGAN را مقایسه کنیم، متوجه می شویم که چون در ACGAN دو عدد loss داریم و از label نیز برای تولید کننده استفاده می کنیم، لذا خروجی های بهتری با وضوح بeter تولید می شوند که قبل از خروجی های آن ها قرار داده شد.

مراجع

1. <https://medium.com/@sunnerli/the-story-about-wgan-784be5acd84c>
2. <https://arxiv.org/abs/1805.08318>
3. <https://arxiv.org/abs/1511.06434>
4. <https://github.com/zalandoresearch/fashion-mnist>
5. <https://www.cs.toronto.edu/~kriz/cifar.html>
6. <https://arxiv.org/pdf/1701.04862.pdf>
7. <http://papers.nips.cc/paper/6125-improved-techniques-for-training-gans.pdf>
8. <https://arxiv.org/abs/1610.09585>
9. <https://arxiv.org/abs/1701.07875>
10. <https://machinelearningmastery.com/how-to-implement-wasserstein-loss-for-generative-adversarial-networks/>