



به نام خدا



دانشگاه تهران

دانشکده مهندسی برق و کامپیووتر

شبکه های عصبی و یادگیری عمیق

مینی پروژه سوم

علیرضا فدایکار	نام و نام خانوادگی
810195555	شماره دانشجویی
99/5/15	تاریخ ارسال گزارش

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

- 3..... سوال 1 – شبکه Variational Autoencoder
- 19..... سوال 2 – شبکه DCGAN
- 36..... سوال 3 – شبکه های conditional GANs
- 64..... سوال 4 – شبکه های Cycle GAN

سوال 1 – شبکه Variational Autoencoder

(الف)

هر دو شبکه GAN و generative Variational Autoencoder مدل‌هایی (سازنده) هستند. در واقع هر دو شبکه پس از یادگیری توزیع دیتای ورودی قادر به ساختن تصاویر هستند. تفاوت دو شبکه در نحوه آموزش آنهاست. شبکه‌های VAE یک توزیع را با مقایسه بین ورودی و خروجی شان learn می‌کنند که این کار برای آموزش representation‌های مخفی داده‌ها خوب است ولی برای ساختن تصاویر جدید مناسب نیست. به همین خاطر خروجی VAE‌ها معمولاً تار هستند. ولی شبکه‌های Generative Adversarial یا GAN‌ها به صورت متفاوتی آموزش می‌بینند. در این شبکه‌ها از یک شبکه دیگر که Discriminator نامیده می‌شود، استفاده می‌شود که این شبکه تفاوت بین تصویر تولید شده و تصویر واقعی را ارزیابی می‌کند و به طور مفهومی تصاویر تولید شده توسط generator را از تصاویر واقعی تشخیص می‌دهد. در این میان generator سعی می‌کند طوری آموزش ببیند که تصاویری که تولید می‌کند را تصویر fake تشخیص ندهد.

مزیت مهم GAN‌ها نسبت به VAE‌ها این است که می‌توانند به ورودی‌های مختلفی condition شوند. به عنوان مثال می‌توانند با دریافت ورودی عکس‌های گرفته شده توسط ماهواره‌ها، نقشه مناطق مختلف را در خروجی تولید کنند و یا اینکه می‌توانند طوری آموزش ببینند که generator کلاس‌های مختلفی از دیتا را تولید کند به عنوان مثال دیتاست MNIST

از دیگر مزایای GAN‌ها این است که می‌توانند با دیتای کمتری آموزش ببینند و خروجی شان نسبت به خروجی VAE‌ها معمولاً sharp‌تر و واضح‌تر است.

از معایب GAN‌ها این است که ساختار پیچیده‌ای نسبت به VAE‌ها دارند و زمان آموزش آنها زیاد است. در حالی که یکی از مزایای VAE‌ها این است که ساختار بسیار ساده‌تری دارند و به همین ترتیب زمان آموزش آنها به مراتب از GAN‌ها کمتر است.

(ب)

در این قسمت شبکه VAE را پیاده سازی می‌کنیم. در ادامه درباره ساز و کار و معماری این شبکه توضیح می‌دهیم:

همانطور که در صورت سوال نیز توضیح داده شده است ، شبکه VAE از دو قسمت encoder و decoder تشکیل می شود. شبکه encoder مشاهده ورودی x را در ورودی دریافت می کند و در خروجی خود بردار میانگین و واریانس و از روی آن متغیر نهفته^۱ z را در خروجی تولید می کند و بنابرین توزیع خود بردار میانگین و واریانس و از این توزیع است) را تقریب می زند. ما این توزیع را به صورت ساده با توزیع گوسی مدل می کنیم.

برای generate کردن متغیر نهفته z که ورودی شبکه decoder می باشد ، از تکنیک reparameterization که در مقاله نیز ذکر شده است استفاده می کنیم. در این تکنیک ما z را به کمک بردار میانگین μ و بردار انحراف معیار σ که خروجی encoder می باشند به صورت زیر تقریب می زنیم:

$$z = \mu + \sigma \odot \epsilon$$

که در این رابطه ϵ یک آرایه هم بعد با μ و σ بوده و مقادیرش از توزیع استاندارد نرمال گوسی نمونه برداری شده اند. منظور از \odot ضرب درایه به درایه می باشد.

در معماری شبکه encoder ما در ابتدا از یک لایه InputLayer برای دریافت ورودی استفاده می کنیم. سپس دو لایه Conv2D به ترتیب با 32 و 64 فیلتر 3×3 با استرايد 2 در نظر می گیریم. سپس یک لایه Flatten برای تبدیل کردن خروجی لایه کانولوشن به صورت بردار و در نهایت یک لایه Dense با بعد دو برابر latent_dim برای خروجی های میانگین و انحراف معیار قرار می دهیم. این معماری به صورت در شکل 1 نیز قابل مشاهده است:

```
tf.keras.layers.InputLayer(input_shape=(28, 28, 1)),
tf.keras.layers.Conv2D(
    filters=32, kernel_size=3, strides=(2, 2), activation='relu'),
tf.keras.layers.Conv2D(
    filters=64, kernel_size=3, strides=(2, 2), activation='relu'),
tf.keras.layers.Flatten(),
# No activation
tf.keras.layers.Dense(latent_dim + latent_dim),
```

شکل 1-1: معماری encoder

معماری شبکه decoder دقیقا عکس شبکه encoder است. در ابتدا یک لایه InputLayer با بعد latent_dim برای دریافت ورودی z قرار می دهیم. سپس یک لایه Dense با بعد $32 \times 7 \times 7$ قرار می دهیم (این بعد برابر بعد خروجی Conv2D دوم در encoder است). سپس یک لایه Reshape قرار می

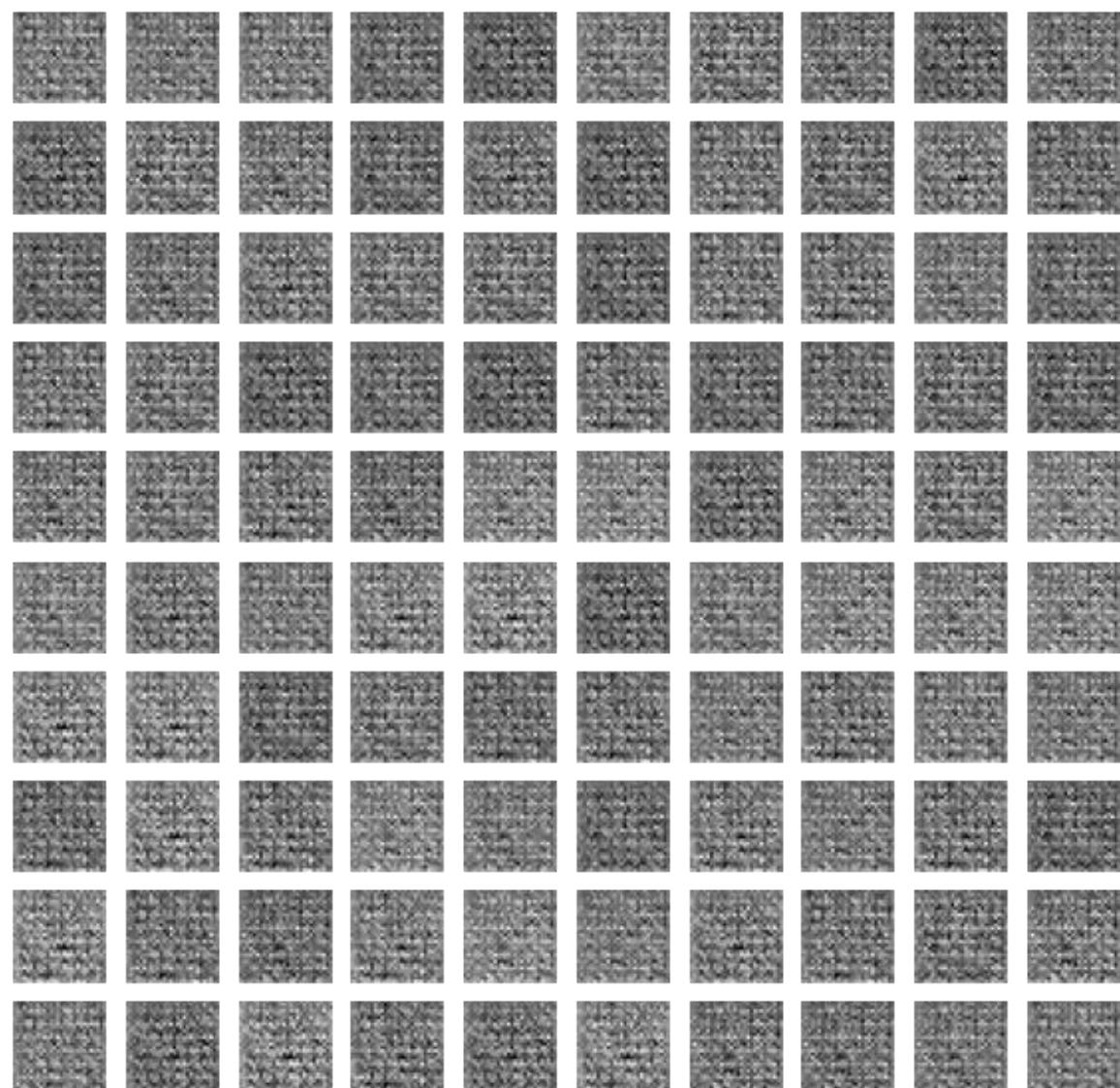
Latent representation¹

دهیم و ابعاد خروجی لایه قبل را به ابعاد (7, 7, 32) تغییر می دهیم. سپس سه لایه Conv2DTranspose قرار می دهیم. معماری decoder در قالب کد به صورت شکل 1-2 می باشد. در مورد تابع loss ای که در نظر گرفتیم در قسمت ج توضیح خواهیم داد.

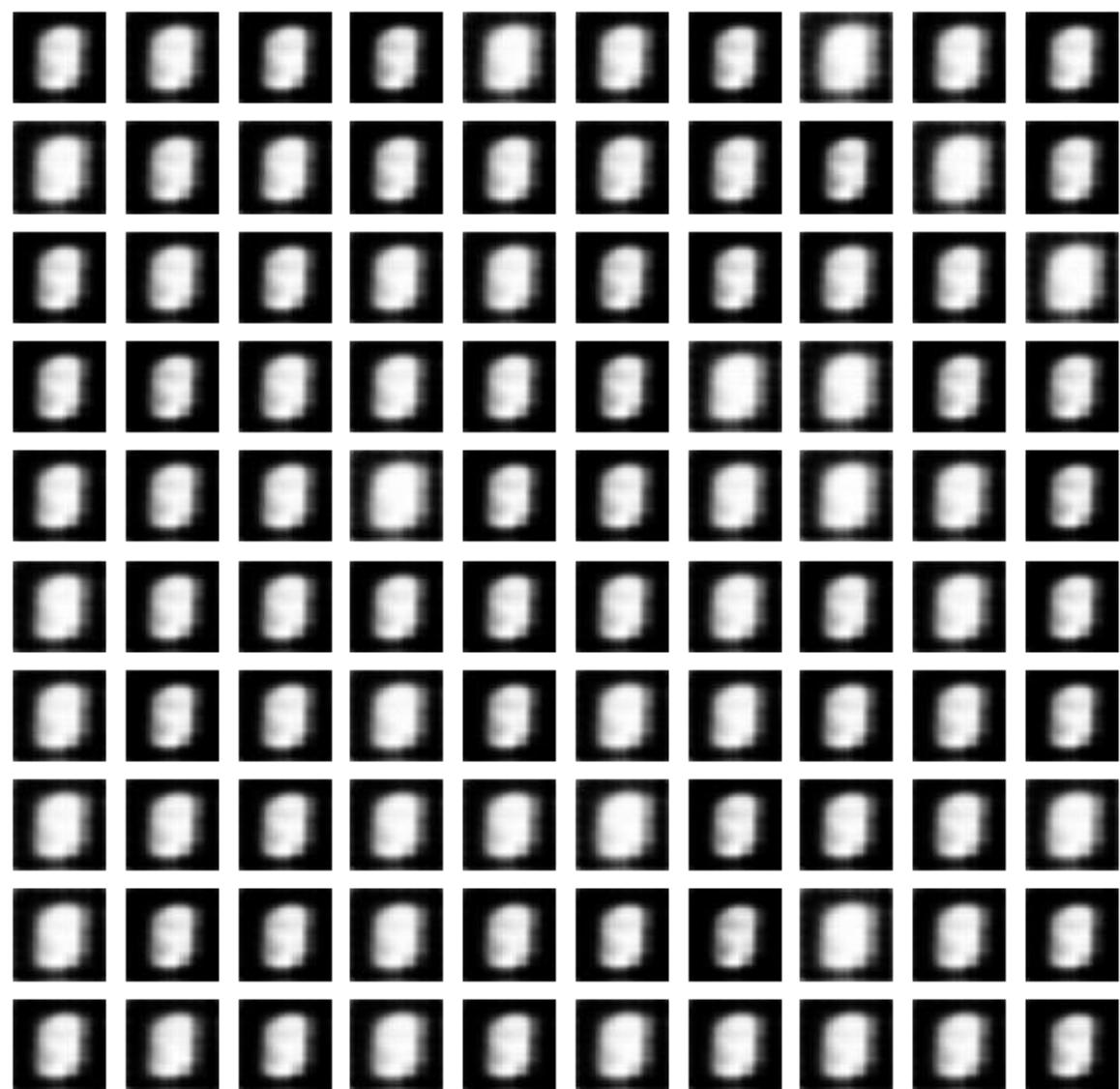
```
tf.keras.layers.InputLayer(input_shape=(latent_dim,)),
tf.keras.layers.Dense(units=7*7*32, activation=tf.nn.relu),
tf.keras.layers.Reshape(target_shape=(7, 7, 32)),
tf.keras.layers.Conv2DTranspose(
    filters=64, kernel_size=3, strides=2, padding='same',
    activation='relu'),
tf.keras.layers.Conv2DTranspose(
    filters=32, kernel_size=3, strides=2, padding='same',
    activation='relu'),
# No activation
tf.keras.layers.Conv2DTranspose(
    filters=1, kernel_size=3, strides=1, padding='same'),
```

شکل 2-1: معماری **decoder**

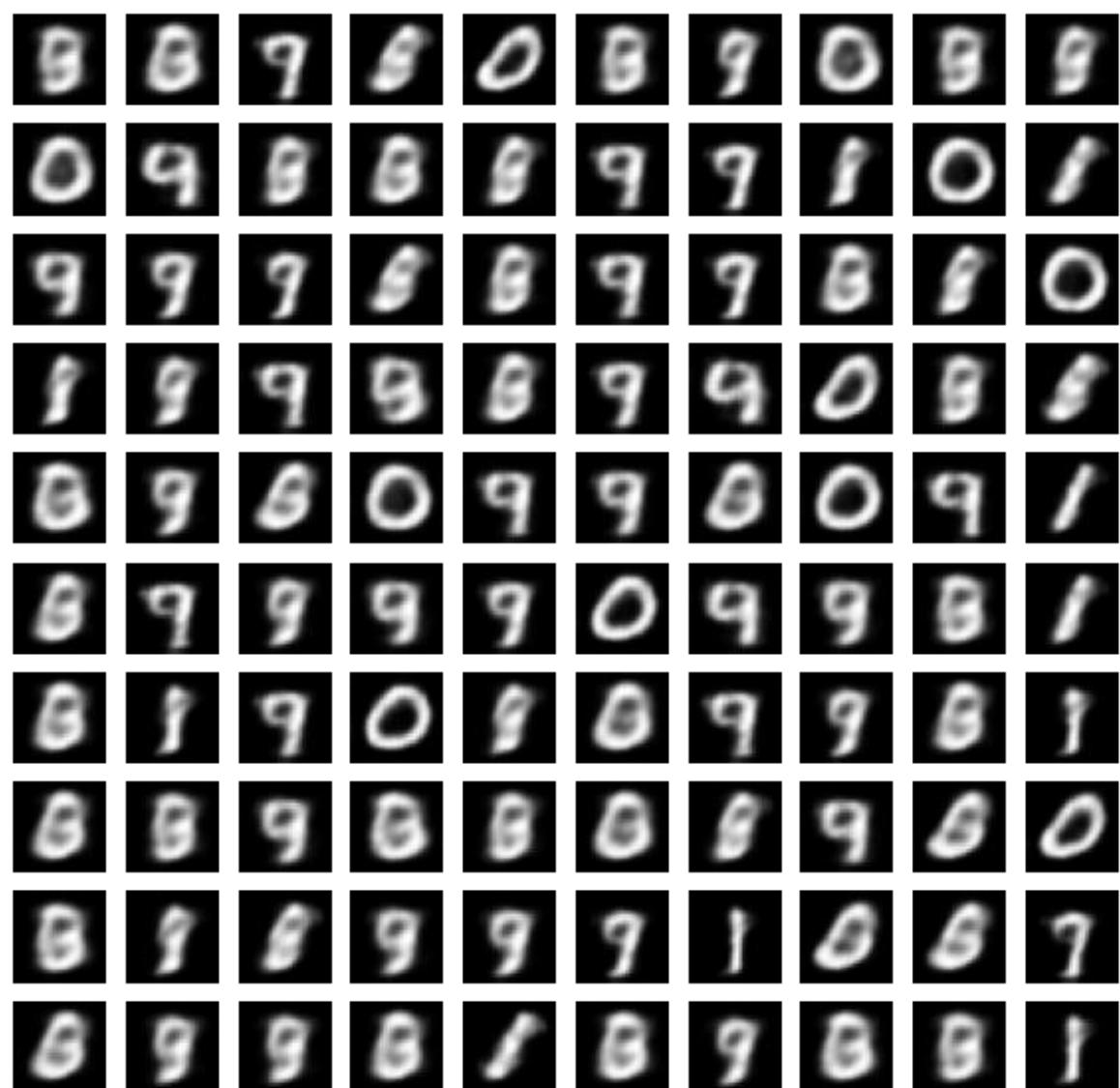
حال شبکه را train می کنیم و نتایج تصاویر تولید شده تصاویر validation را به صورت یک جدول 10×10 برای مشاهده پیشرفت در epoch های مختلف رسم می کنیم. در ادامه تصاویر 1-3 ، 1-4 ، 1-5 ، 1-6 ، 1-7 و 1-8 و 21 نشان می دهد. همان طور که مشاهده می شود نتایج به مرور بهتر می شوند.



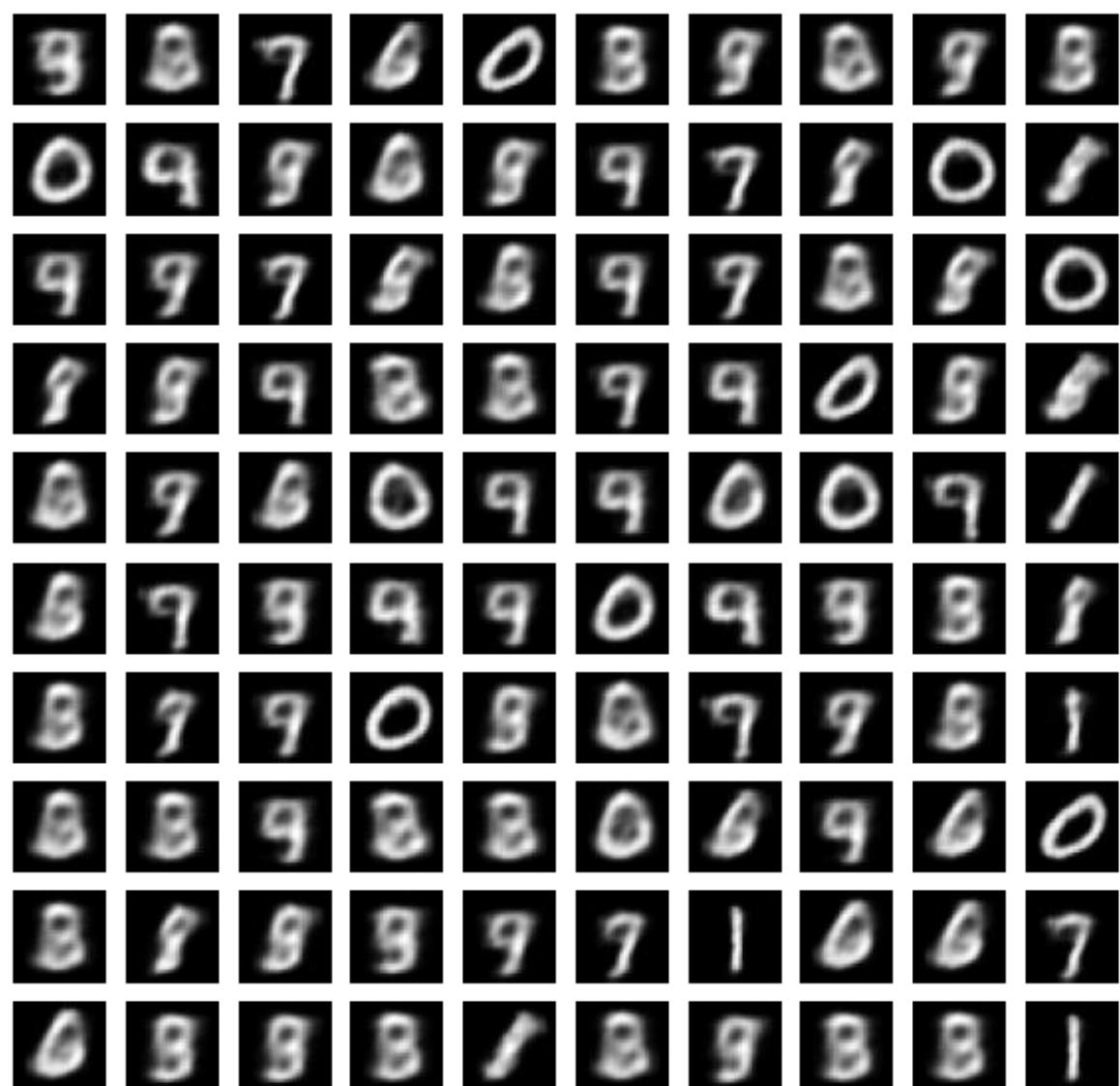
شکل 3: خروجی قبل از شروع training در epoch صفر



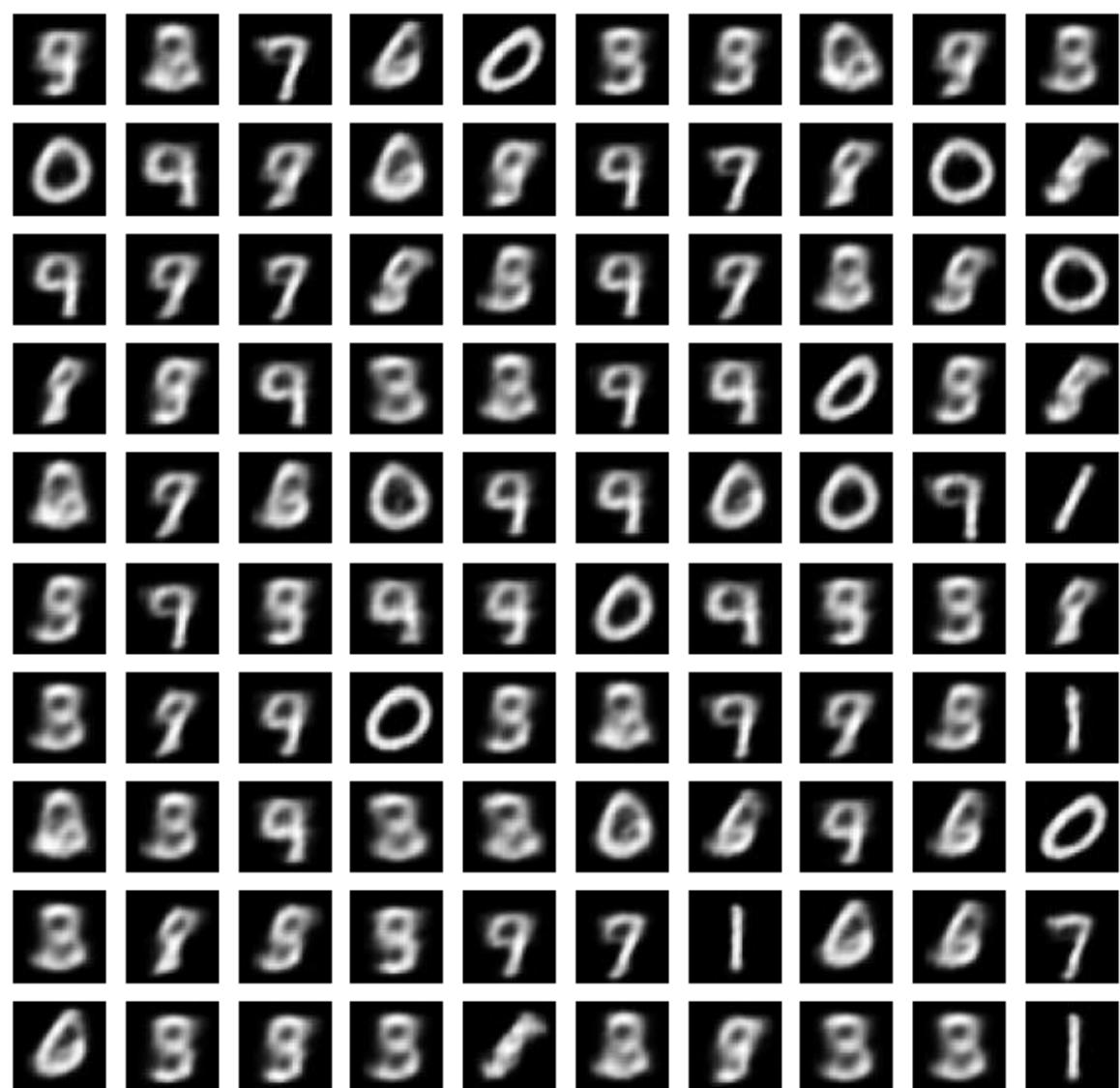
شکل 1-4: خروجی در $epoch = 1$



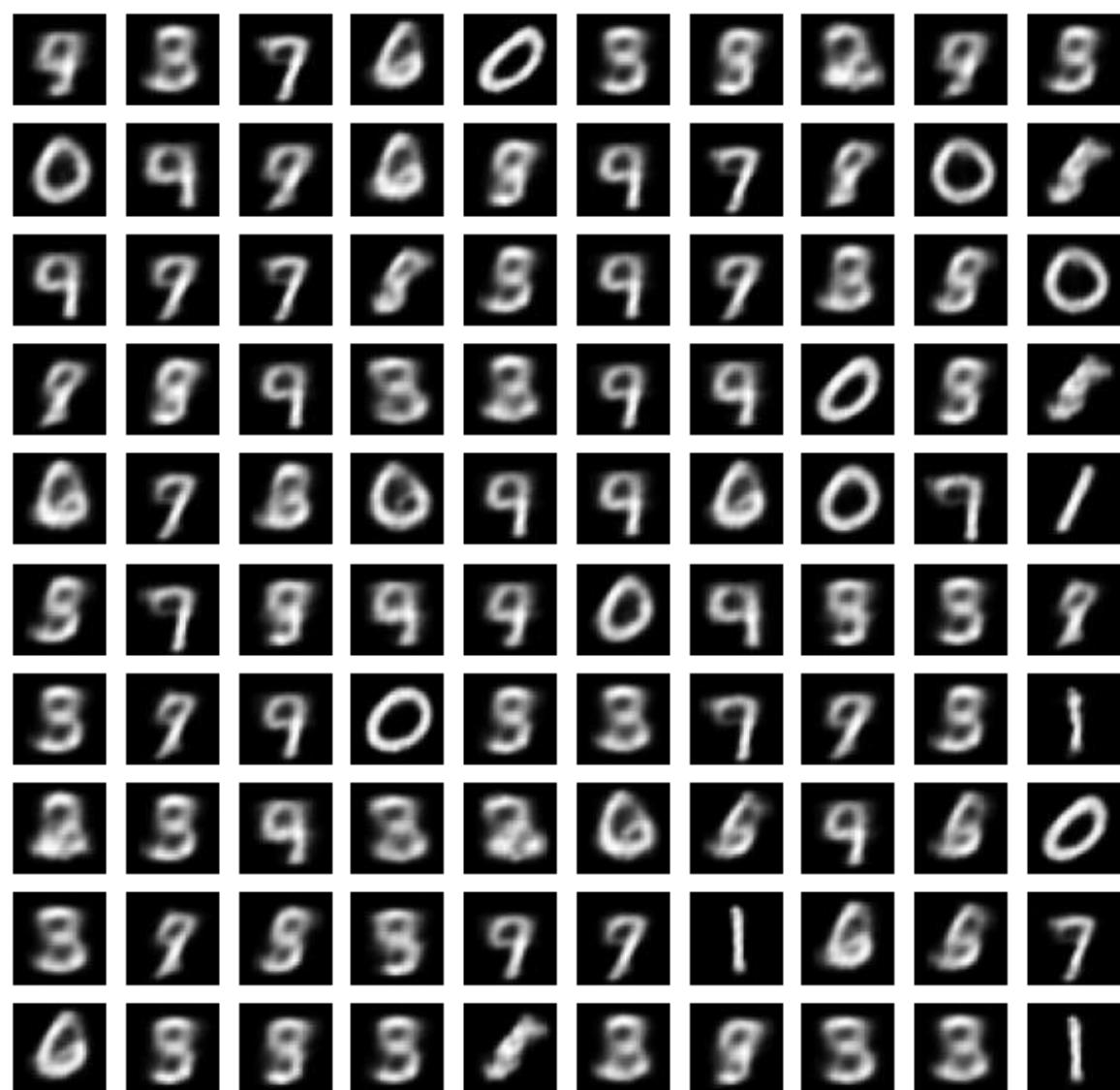
شکل 1-5: خروجی در epoch = 6



شکل 6-1: خروجی در epoch = 11

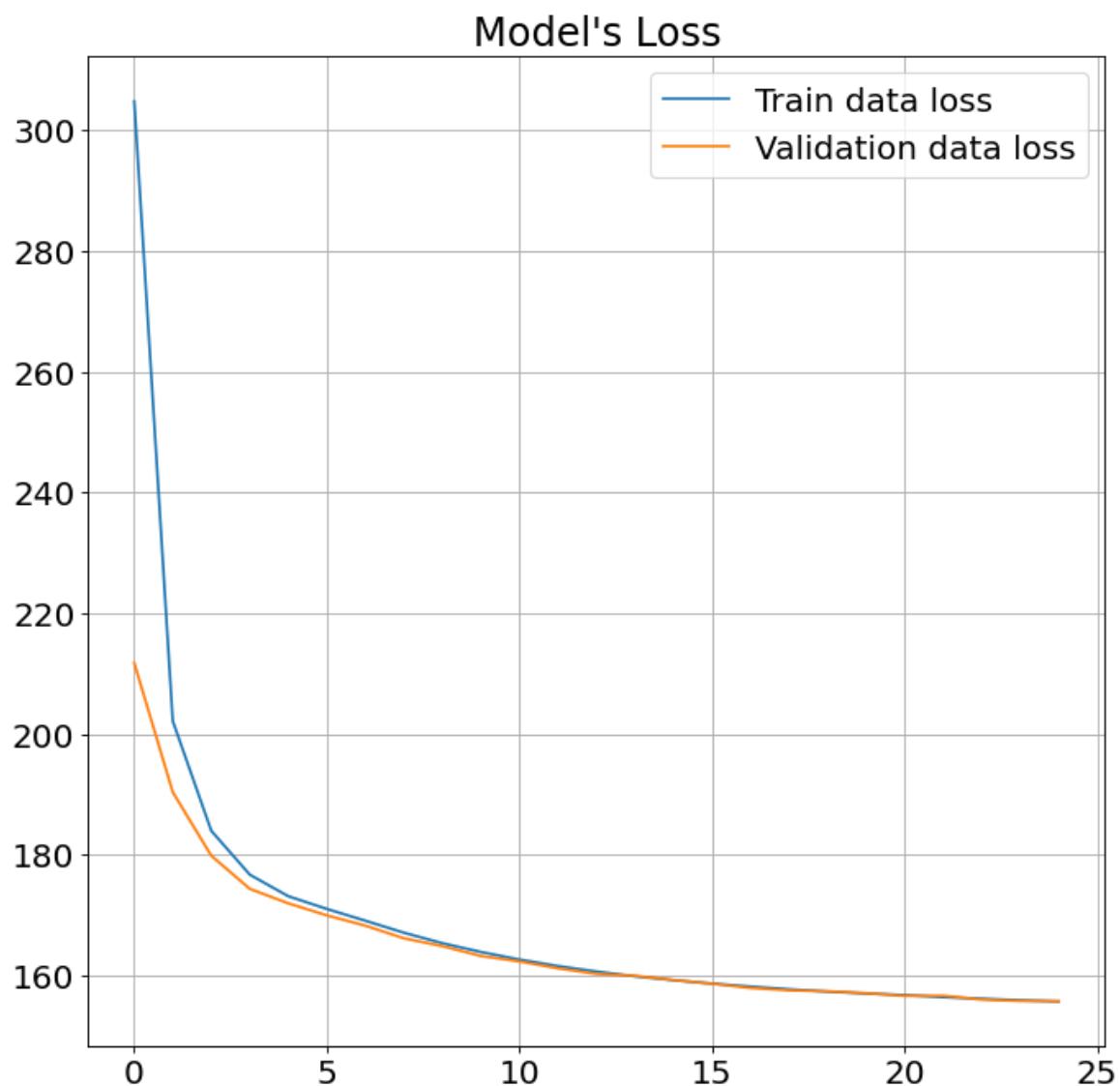


شکل 7-1: خروجی در epoch = 16



شکل ۱-۸: خروجی در $epoch = 21$

نمودار خطای loss برای دیتای train و validation در شکل ۹-۱ بر حسب epoch رسم شده است:



شکل 1-9: نمودار loss بر حسب epoch

همان طور که در شکل 1-9 مشاهده می شود دو نمودار به خوبی بر حسب epoch کاهش یافته اند.

(ج)

ما در این سوال وزنها را به منظور ماقزیم کردن variational lower bound یا evidence lower bound (که در مقاله با این نام ذکر شده) که با ELBO نیز نشان داده می شود روی marginal log-likelihood با روشنگاری gradient descent آپدیت می کنیم. ELBO به صورت زیر تعریف می شود:

$$\text{ELBO} = E_{q(z|x)} \left[\log \frac{p(x,z)}{q(z|x)} \right]$$

برای بهینه سازی ELBO همان طور که در صفحه ۳ مقاله نیز اشاره شده است از روش ساده نمونه برداری تقریب Monte Carlo برای ماقزیم کردن expectation استفاده می کنیم. به عبارت دیگر عبارت داخل expectation را برای بهینه سازی در نظر می گیریم:

$$\log \frac{p(x,z)}{q(z|x)} = \log p(x|z) + \log p(z) - \log q(z|x)$$

این تابع را با روش گرادیان کاهشی (که در فایل notebook جزئیات آن توضیح داده شده است) ماقزیم می کنیم. تابع loss را قرینه میانگین عبارت فوق تعریف می کنیم:

$$loss = -E(\log p(x|z) + \log p(z) - \log q(z|x))$$

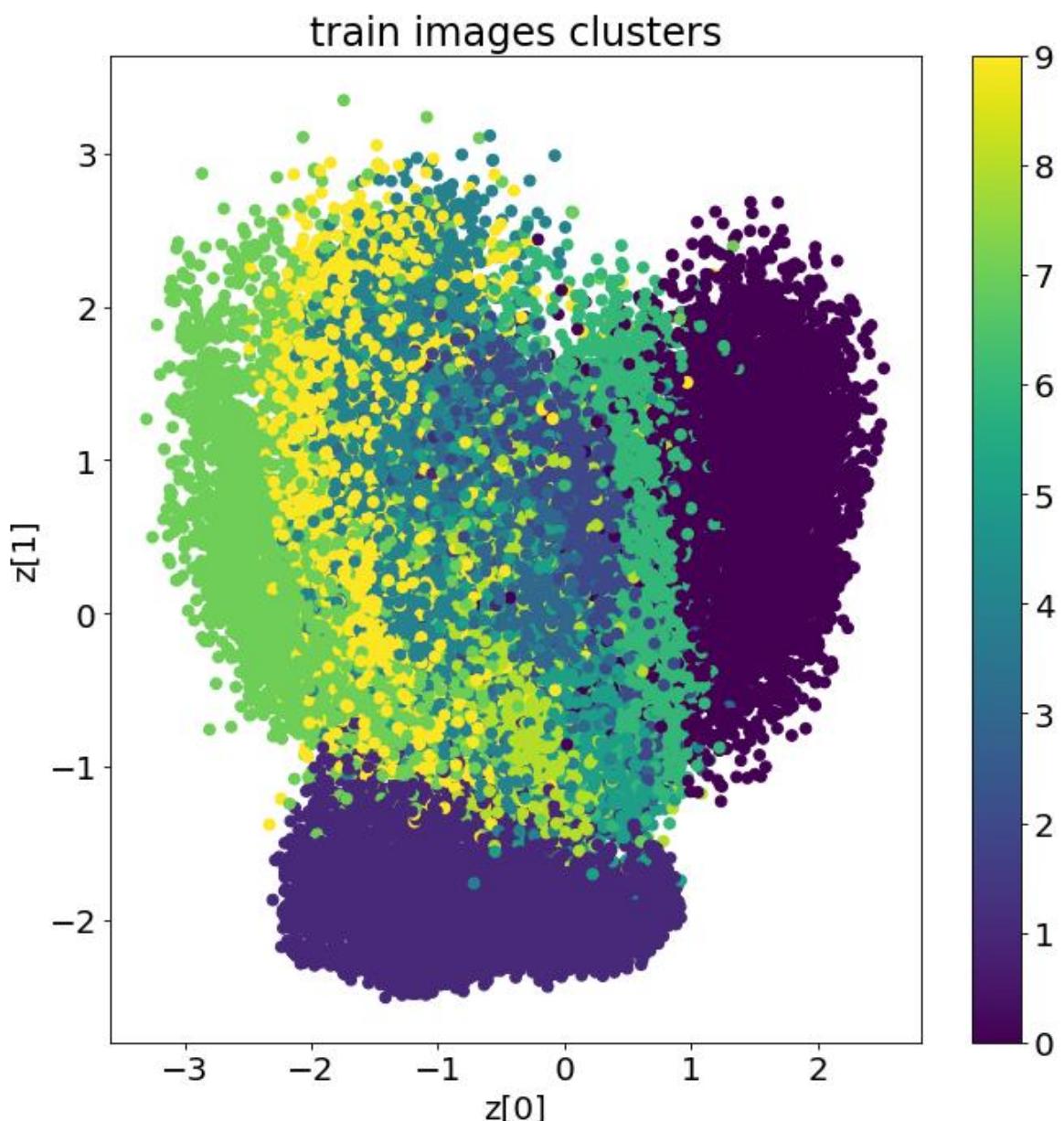
پیاده سازی این تابع loss به صورت کد در شکل 10-1 نیز قابل مشاهده است.

```
def compute_loss(model, x):
    mean, logvar = model.encode(x)
    z = model.reparameterize(mean, logvar)
    x_logit = model.decode(z)
    cross_ent = tf.nn.sigmoid_cross_entropy_with_logits(logits=x_logit, labels=x)
    logpx_z = -tf.reduce_sum(cross_ent, axis=[1, 2, 3])
    logpz = log_normal_pdf(z, 0., 0.)
    logqz_x = log_normal_pdf(z, mean, logvar)
    return -tf.reduce_mean(logpx_z + logpz - logqz_x)
```

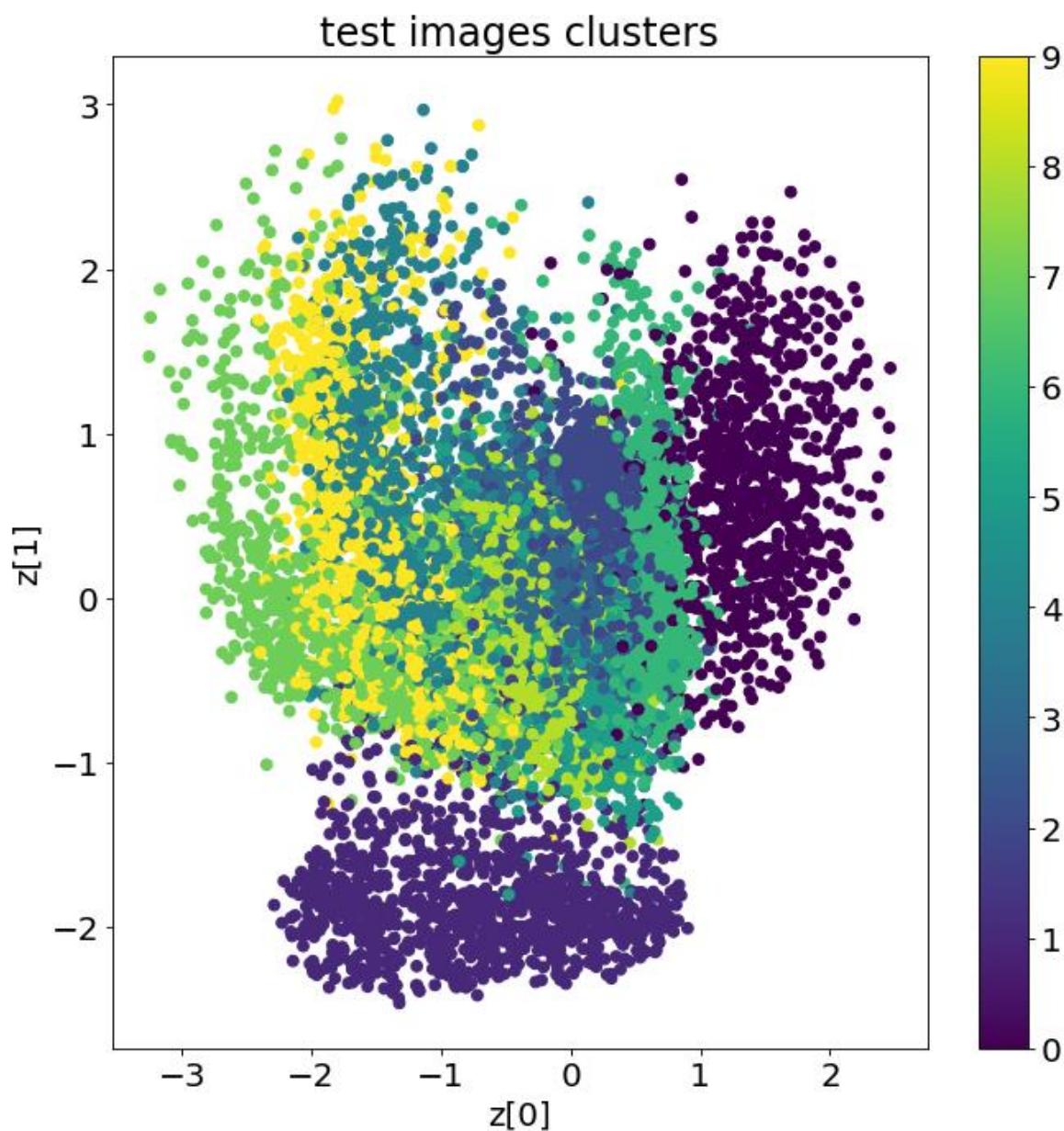
شکل 1-10: تابع loss

(د)

در این قسمت به کمک Encoder تصاویر داده های train و validation را از فضای 784 بعدی به فضای دو بعدی latent (خرجی mean انکدر) و نتایج را به صورت دو بعدی با دستور scatter رسم می کنیم. شکل 1-11 نتیجه را برای داده های train و شکل 12-1 نتیجه را برای داده های validation نشان می دهد:



شکل 1-11: خوش بندی داده های train

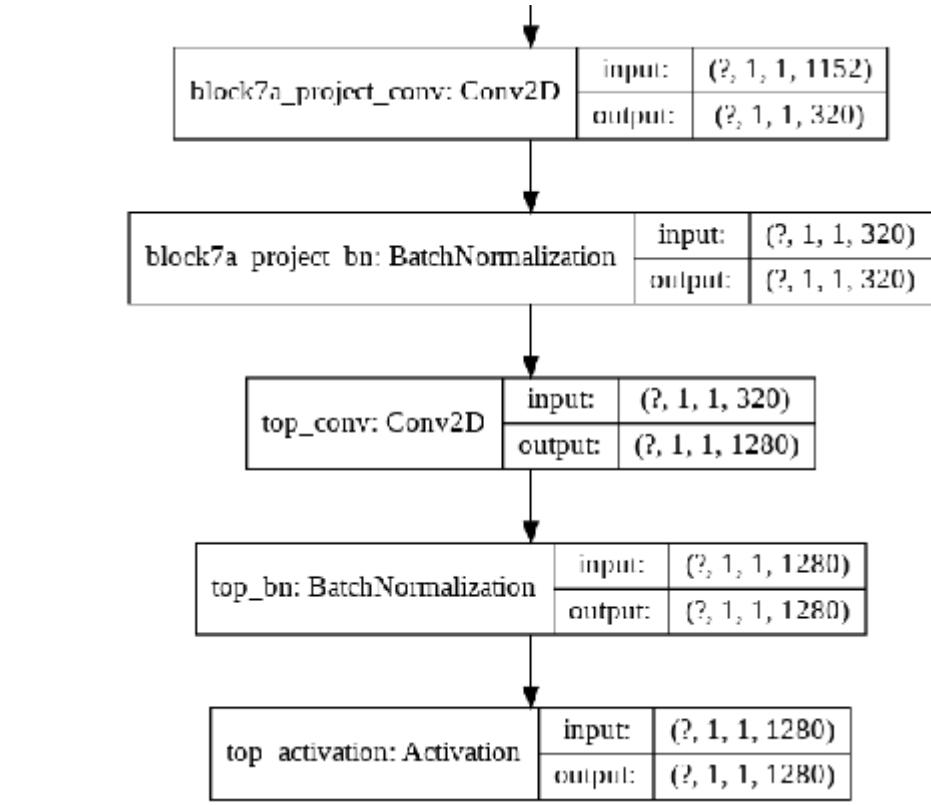


شکل 1-12: خوشه بندی داده های validation

(۵)

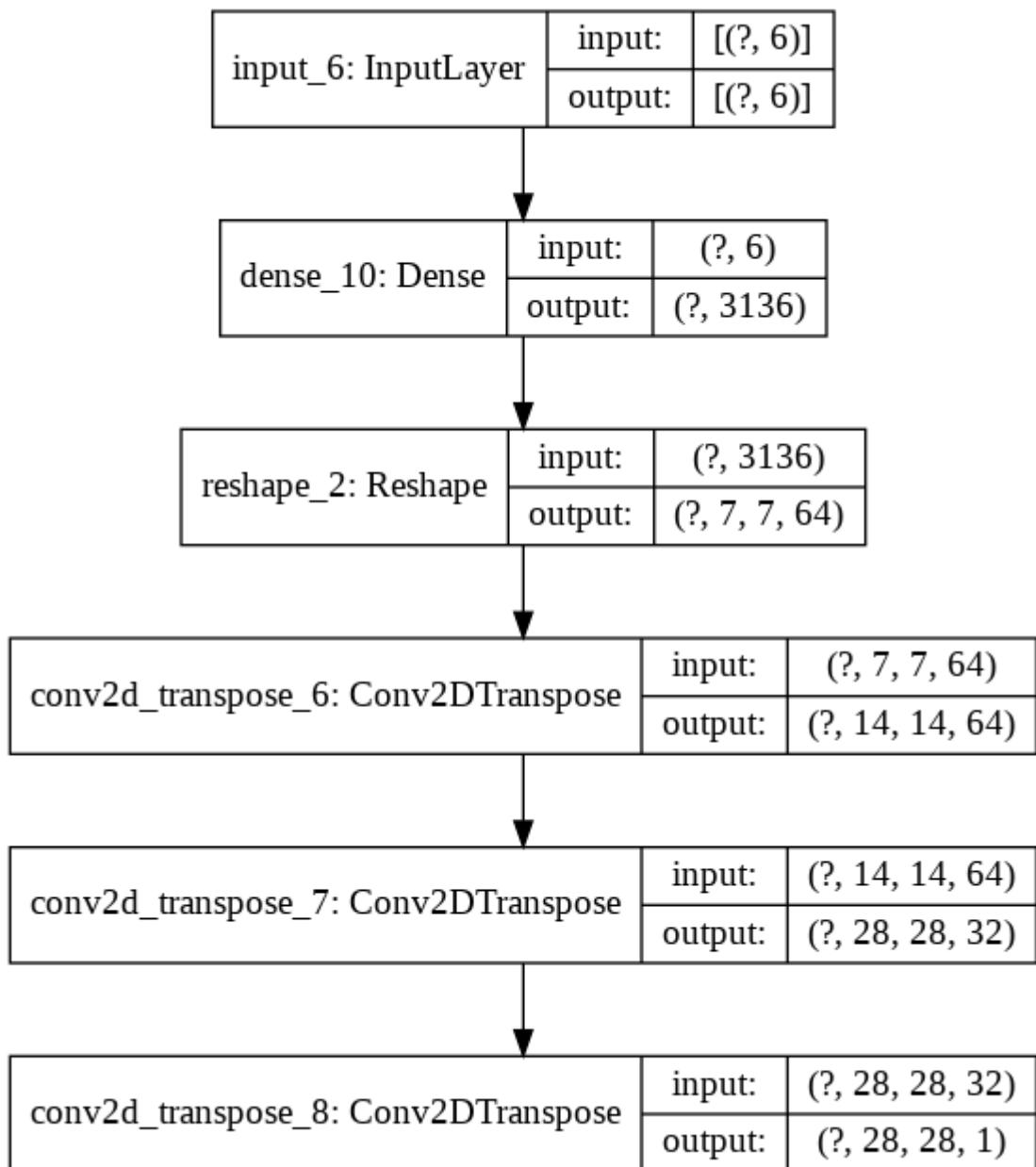
شبکه های efficientnet که از لحاظ محاسبات بسیار بهینه هستند برای اولین بار در ماه May سال 2019 توسط دو مهندس از گوگل در مقاله ای به نام “EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks” معرفی شد. این شبکه که هم از لحاظ تعداد پارامترها و هم از لحاظ محاسبات بسیار بهینه هستند موفق شد به مقدار accuracy 84.4% را روی دیتاست ImageNet دست پیدا کند. در این روش به کمک transfer learning این شبکه را روی داده های MNIST اعمال می کنیم.

پس از import مدل efficientnet B0 از پیش ترین شده ، ساختار آن را رسم می کنیم که چون ساختار بسیار بزرگی دارد فقط چند لایه آخر را رسم می کنیم که در شکل 1-13 قابل مشاهده است:



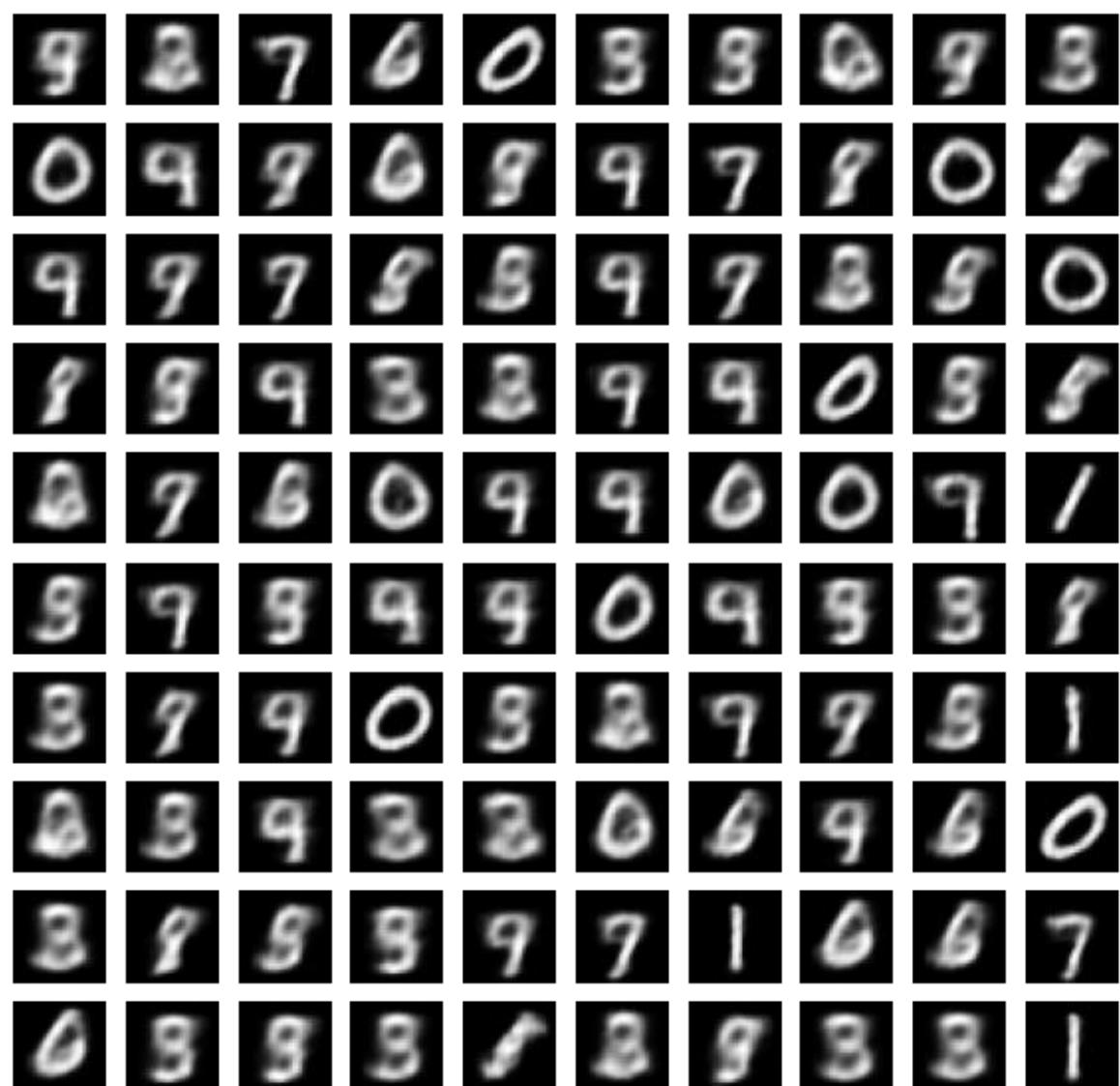
شکل 1-13

ساختار decoder به صورت شکل 1-14 است:



شکل 1-14: ساختار **decoder**

لازم به ذکر است چون مدل efficientnet روی دیتاست imagenet ترین شده است ورودی اش باید باشد بنابرین تصاویر را باید با 0 پدینگ کنیم تا ساختار ورودی را رعایت کنیم.
 شبکه را در 10 ایپاک ترین می کنیم که در شکل زیر نتیجه آخرین ایپاک قابل مشاهده است:



آخرین ایپاک

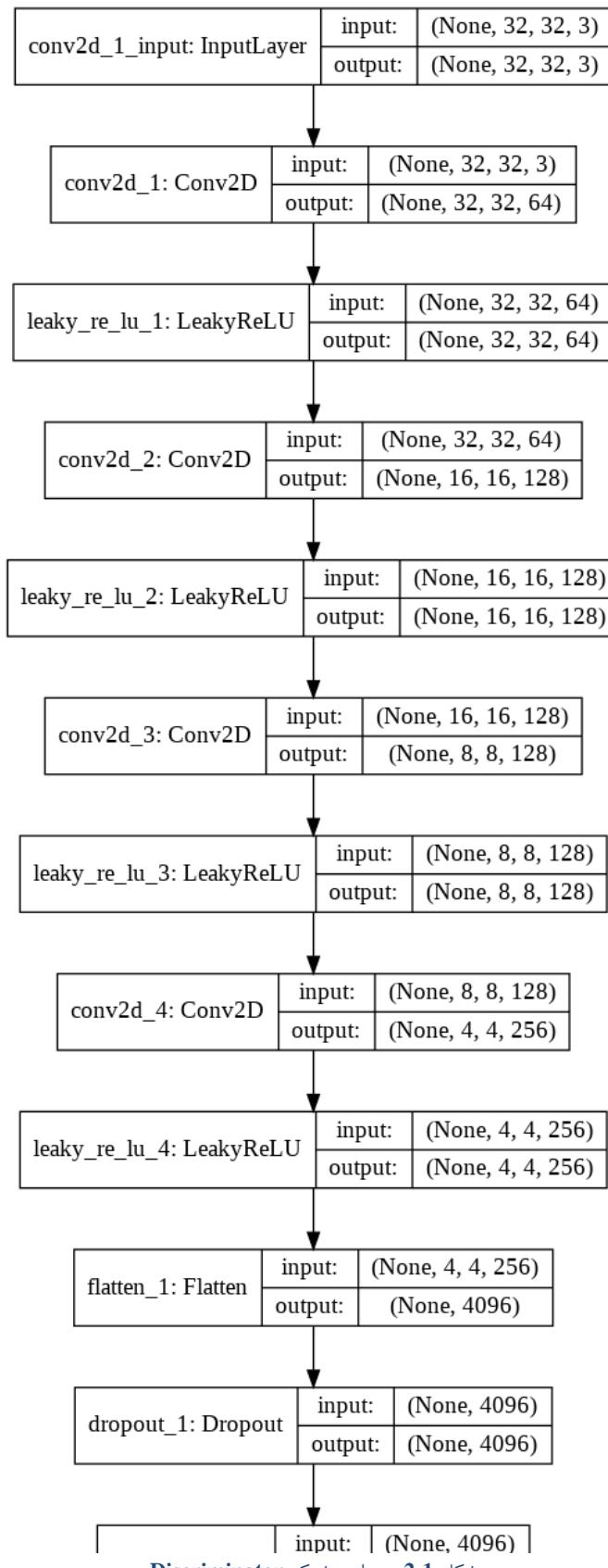
سوال ۲ - شبکه DCGAN

(الف)

شبکه همانند GAN ها از دو قسمت generator و discriminator تشکیل می شود که هر کدام از این دو شبکه از چندین لایه کانولوشن تشکیل می شود که دلیل نام گذاری آن (Deep Convolutional Generative Adversarial Networks) نیز همین است. طبق مقاله این مدل پس از آزمایشات بسیار زیادی روی ساختارهای مختلف و دیتاست های مختلف بدست آمده است. هسته اصلی این شبکه ایجاد سه تغییر در شبکه های با ساختار CNN می باشد. اول اینکه از لایه های maxpooling با طول استراید ثابت بعد از لایه های کانولوشن (Conv2D) استفاده نکنیم تا شبکه خودش قادر به یادگیری فضایی خود باشد. دوم اینکه بعد از لایه های convolution به منظور افزایش سرعت همگرایی به جز در ابتدای generator و در انتهای discriminator از لایه fully connected یا Dense استفاده نکنیم. و در نهایت نکته سوم این است که به منظور افزایش پایداری آموزش شبکه ، از لایه های Batch normalization در لایه های کانولوشن استفاده کنیم. این لایه ها باعث می شوند ورودی های هر واحد میانگین صفر و واریانس واحد داشته باشند. این عمل باعث بهبود مشکلات training که به خاطر مقدار دهی اولیه وزنها بدست می آید می شود و امکان به وجود آمدن مشکل gradient vanishing را در شبکه های عمیق را کاهش می دهد.

در شبکه generator در تمام لایه ها به غیر از لایه آخر ازتابع فعال ساز ReLU استفاده می شود. در لایه آخر ازتابع فعال ساز tanh که خروجی آن در بازه $[-1, 1]$ است و از دو طرف اشباع می شود. با استناد به مقاله ، دلیل استفاده از این تابع این است که پس از تست کردن شبکه و انجام مشاهدات و نتایج مختلف ، به این نتیجه رسیدند که استفاده از یک تابع کراندار ، به مدل اجازه می دهد که یاد بگیرد با سرعت بیشتری به اشباع بررسد و فضای رنگی توزیع داده های training را سریعتر پوشش دهد. همچنین مشاهده شده است که استفاده از تابع فعال ساز leaky ReLU با شیب 0.2 به نتایج خوبی منجر می شود. در ادامه درباره معماری این شبکه به طور دقیق تر توضیح می دهیم.

در این سوال ، از معماری شکل 1-2 برای شبکه discriminator استفاده می کنیم. در لایه اول از InputLayer استفاده می کنیم. سپس 4 بار از دو لایه Conv2D و leaky ReLU و سپس از لایه Sigmoid و در نهایت لایه Dropout (با ضریب 0.4) و در لایه آخر ، لایه Dense با بعد یک و تابع فعال ساز استفاده می کنیم که جزئیات این لایه ها در شکل 1-2 قابل مشاهده است.



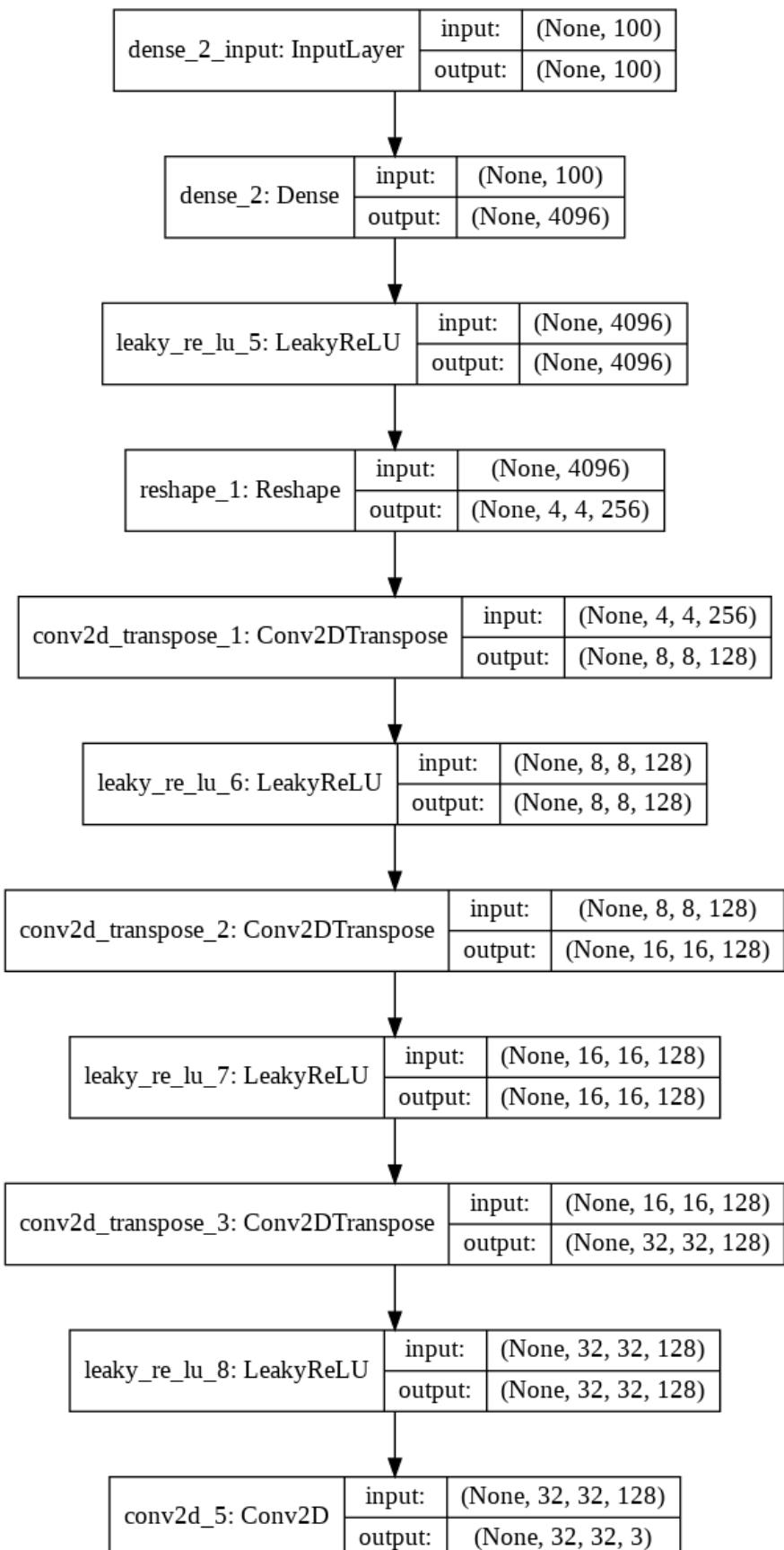
شکل 2-1: معماری شبکه

وظیفه شبکه discriminator این است که یک تصویر (مثلا تصویر $3 \times 32 \times 32$ دیتاست cifar10) را دریافت کند و واقعی بودن یا تقلیلی بودن آن را تشخیص دهد. لازم به ذکر است خروجی اینتابع با توجه به اینکه از تابع فعالساز sigmoid در لایه آخر استفاده کرده ایم عددی در بازه $[0, 1]$ است که هر چه خروجی به یک نزدیکتر باشد به معنای آن است که discriminator این عکس را واقعی تشخیص داده است و بر عکس هر چه به صفر نزدیکتر باشد، عکس را غیر واقعی تر تشخیص داده است.

نحوه آموزش discriminator به این نحو است که یک بچ عکس واقعی را از دیتاست اصلی cifar10 با لیبل 1 به عنوان ورودی به شبکه می دهیم و وزنها را آپدیت می کنیم. سپس یک بچ تصاویر غیر واقعی را توسط generator از فضای نهفته latent space (که در این سوال بردار های به طول 100 رندوم از تابع گوسی نرمال هستند) تولید می کنیم و با لیبل 0 به شبکه داده و train می کنیم.

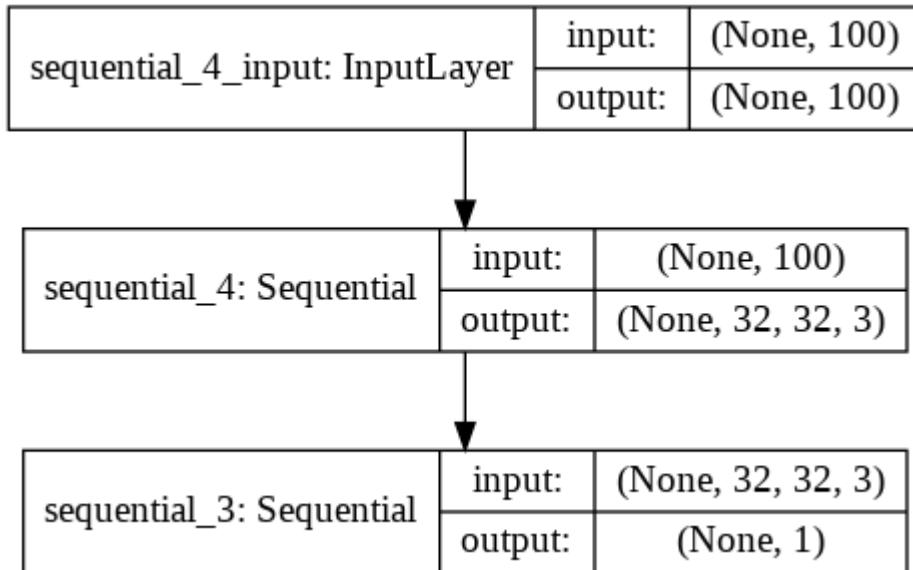
معماری شبکه generator که در این سوال آن را پیاده سازی کرده ایم در شکل 2-2 قابل مشاهده است.

همان طور که در شکل نیز به خوبی قابل مشاهده است در ورودی یک لایه InputLayer با بعد 100 قرار می دهیم. سپس از یک لایه Dense با طول $4096 = 4 \times 4 \times 256$ با تابع فعالساز leaky ReLU استفاده می کنیم. سپس از یک لایه Reshape برای تغییر ابعاد خروجی لایه Dense به ابعاد $4 \times 4 \times 256$ استفاده می کنیم. در ادامه سه بار از لایه Conv2DTranspose که جزئیات آن در شکل 2-2 قابل مشاهده است، استفاده می کنیم. در نهایت از یک لایه Conv2D با تابع فعالساز tanh که دلیل استفاده از آن را پیشتر توضیح دادیم استفاده می کنیم. توجه داشته باشید ورودی شبکه generator یک بردار به طول 100 که مقادیر آن رندوم و دارای توزیع گوسی نرمال استاندارد هستند که در نهایت در خروجی، این ورودی به یک تصویر 32×32 رنگی با سه کانال تبدیل می شود.



شکل ۲-۲ معماری شبکه generator

سپس مدل کلی DCGAN را با ترکیب دو مدل generator و discriminator که توضیح دادیم ، تعریف می کنیم که ساختار آن در شکل 2-3 قابل مشاهده است.



شکل 3-2: ساختار کلی شبکه DCGAN

حال درباره نحوه آموزش generator توضیح می دهیم. پس از اینکه در یک بچ دیتا را آموزش دادیم و وزن های آن را آپدیت کردیم ، خاصیت trainable بودن این شبکه را false کرده و در نتیجه مقدار وزنهای آن را ثابت در نظر می گیریم. سپس یک بچ بردار به طول 100 با مقادیر نرمال (فضای latent) تولید کرده و با لیبل 1 به ورودی generator می دهیم. خروجی generator همان طور که توضیح داده شد به ازای هر بردار ورودی ، یک تصویر رنگی 32×32 می باشد ، این تصویر وارد شبکه discriminator شده و با توجه به خروجی generator و لیبل 1 بردار ورودی ، وزنهای discriminator آپدیت می شوند. دلیل اینکه از لیبل 1 استفاده می کنیم این است که قدرت generator را برای تولید تصاویری که به واقعیت نزدیکتر باشد ، بیشتر کنیم. در واقع ممکن است در ابتدا discriminator به سادگی تصاویر تولید شده توسط generator را fake تشخیص دهد و خروجی نزدیک به صفر تولید کند ولی با توجه به اینکه لیبل ورودی 1 بوده ، optimizer خطای بسیار بزرگی تشخیص می دهد و در نتیجه در عملیات back propagation وزنهای را طوری آپدیت می کند که به ازای ورودی ها و بچ های بعدی خطای کمتری در خروجی ایجاد شود و به عبارتی generator ، discriminator را در تشخیص تصاویری که تولید می کند فریب دهد و به نوعی یک تنازع بین generator و discriminator رخ می دهد. پس از آموزش generator برای یک بچ ، عملیات آموزش generator و discriminator را بار دیگر به ازای بچ های مختلف تکرار می کنیم تا اینکه کل دیتاست train شود. این کار را برای 200 epoch نیز تکرار می کنیم.

(ب)

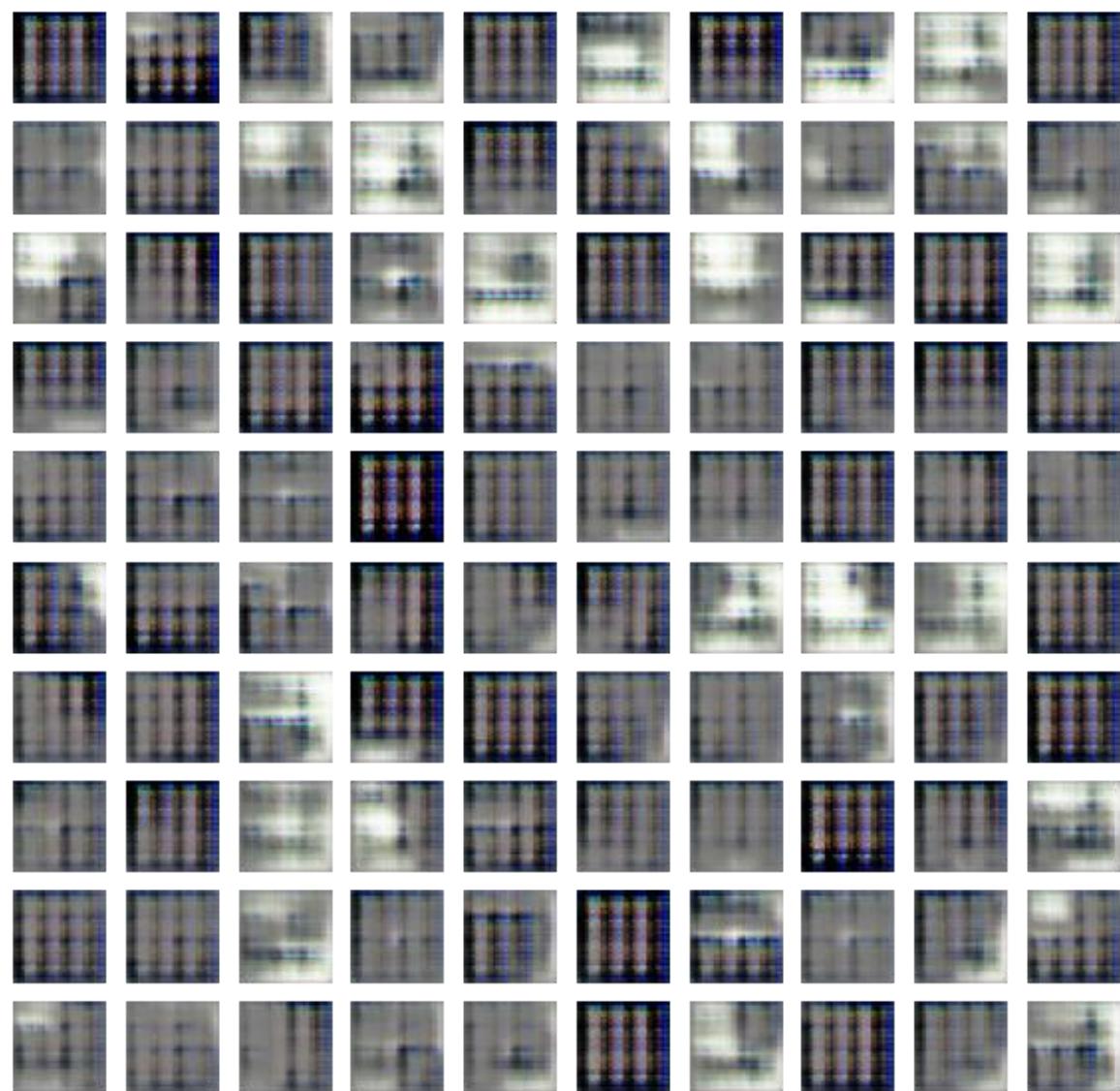
همانطور که در قسمت قبل به تفضیل توضیح دادیم ، بردار نویز را که طول آن 100 می باشد را از توزیع گوسی نرمال استاندارد با کمک تابع randn تولید می کنیم که این بردار رودی شبکه generator می باشد و در خروجی generator به یک تصویر 32×32 رنگی تبدیل می شود که جزئیات آن را در قسمت الف شرح دادیم.

(ج)

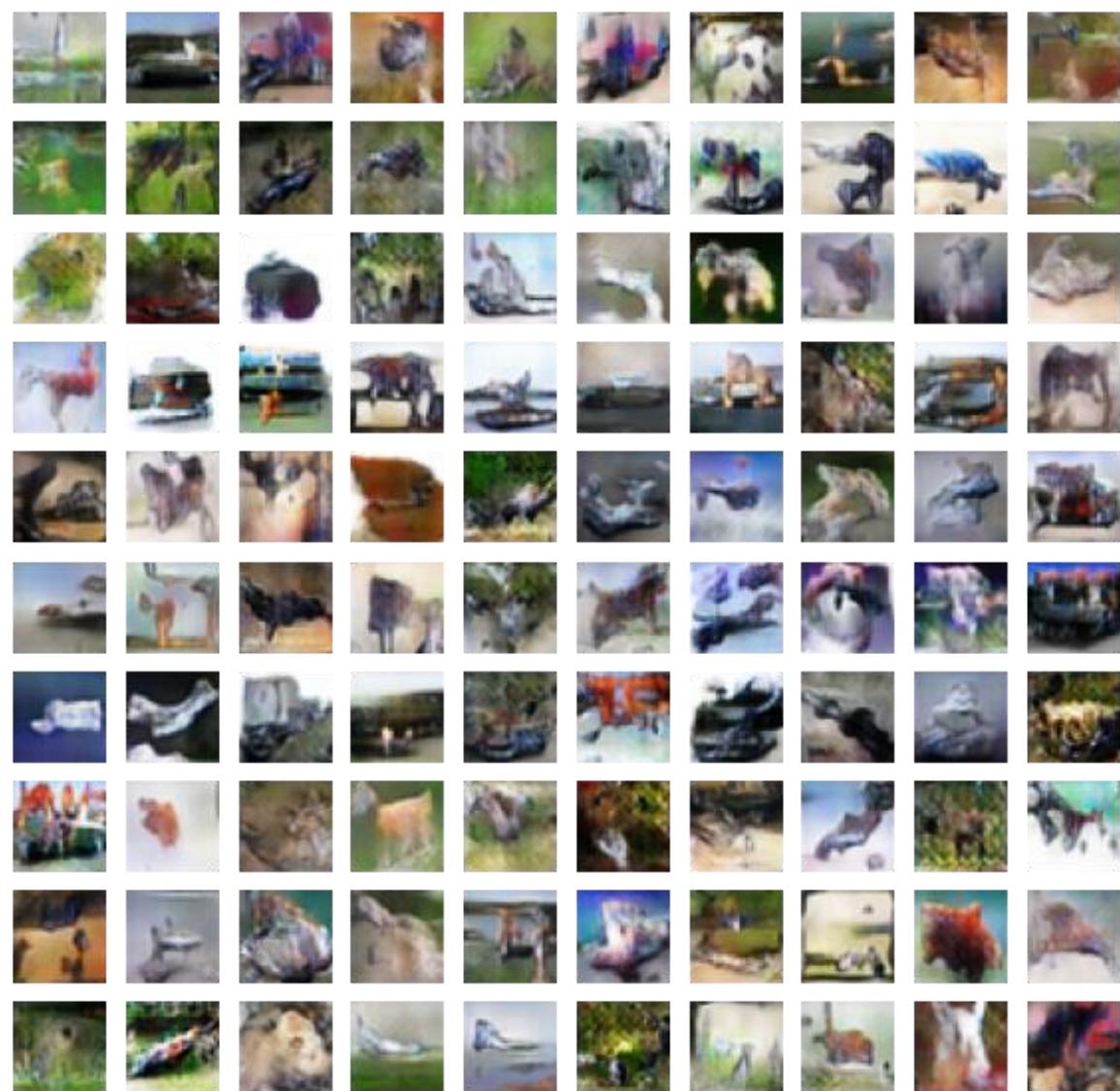
همان طور که در اوایل قسمت الف توضیح دادیم دلیل استفاده از لایه های Strided convolution و maxpooling این است که از لایه های Fractional strides convolution های کانولوشن (Conv2D) استفاده نکنیم تا شبکه خودش قادر به یادگیری upsampling فضایی خود و discriminator باشد.

(د)

معماری شبکه DCGAN را در قسمت الف (شکل های 2-1 ، 2-2 و 2-3) توضیح دادیم. در این قسمت شبکه DCGAN را پیاده سازی می کنیم که جزئیات پیاده سازی در داخل فایل نوتبوک توضیح داده شده است. در طی آموزش شبکه DCGAN در 200 epoch ، هر 20 epoch تعداد 100 تصویر را به کمک generator به صورت یک جدول 10×10 رسم می کنیم تا روند بهبود شبکه و یادگیری آن را مشاهده کنیم. در شکل های 2-4 ، 2-5 ، 2-6 ، 2-7 ، 2-8 ، 2-9 و 2-10 به ترتیب نتایج را برای epoch های 1 ، 41 ، 81 ، 121 و 200 را نشان می دهند.



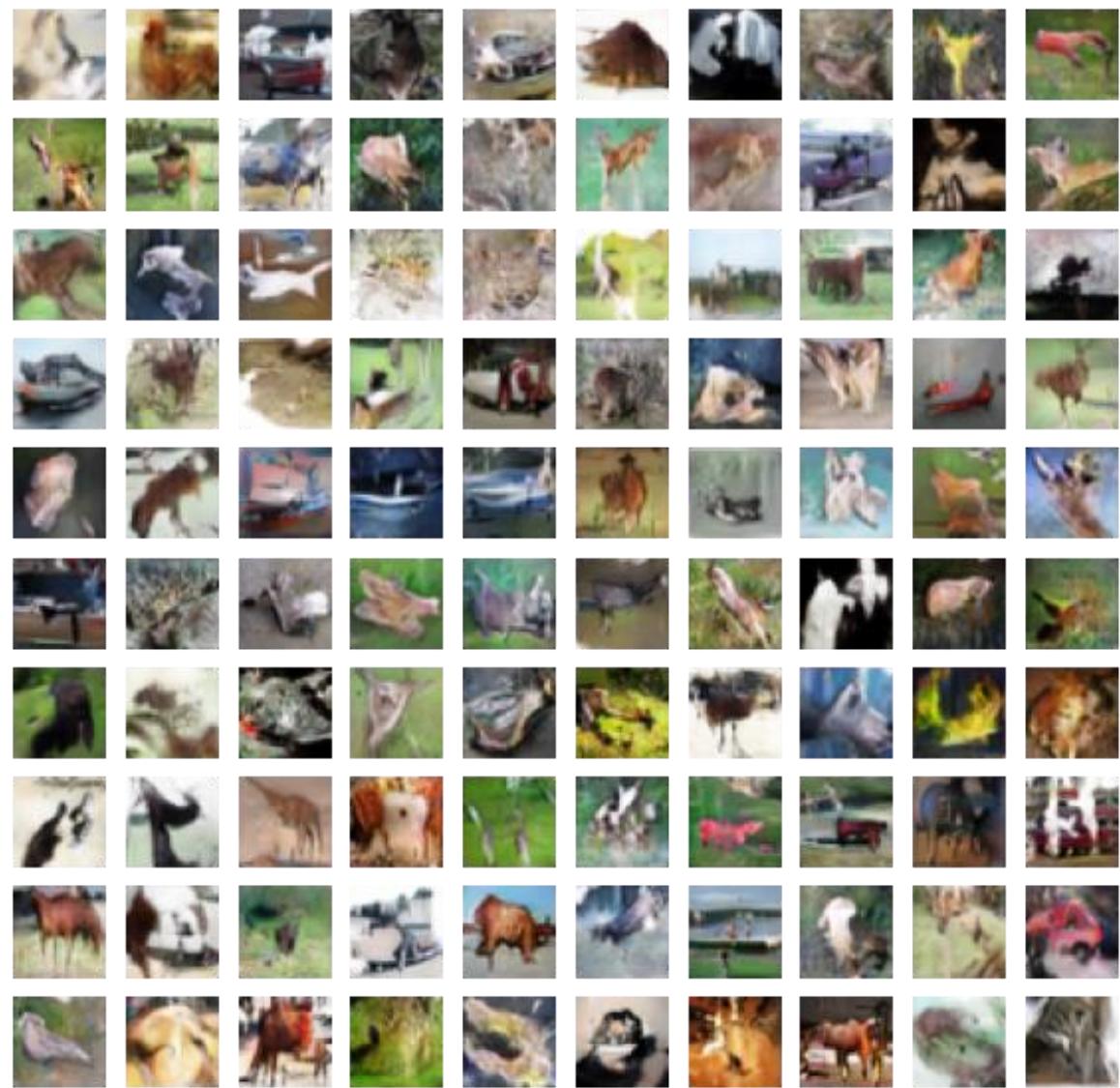
شکل ۴-۲: نتایج بعد از ایپاک ۱



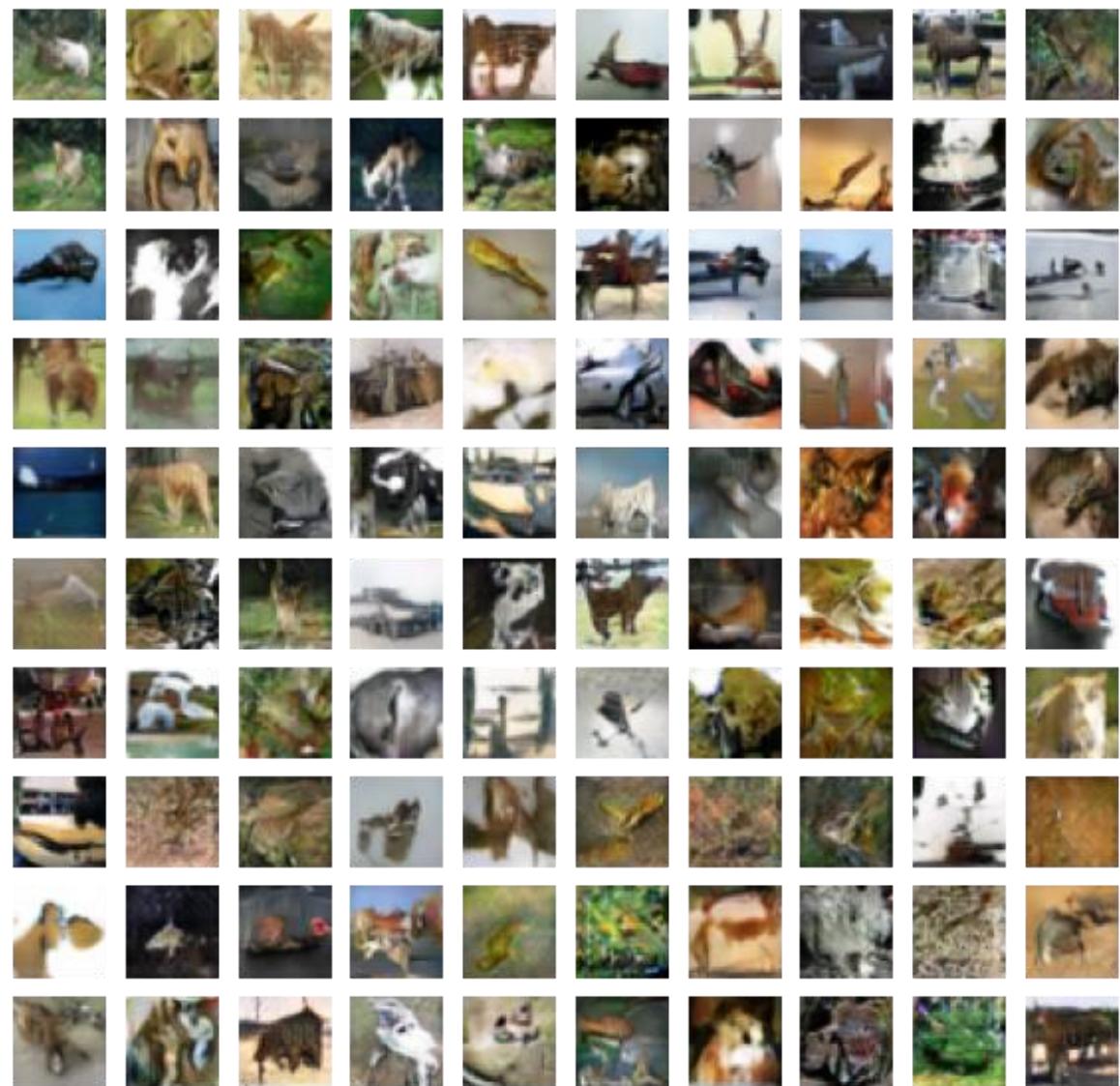
شکل ۵-۲: نتایج بعد از ایپاک ۴۱



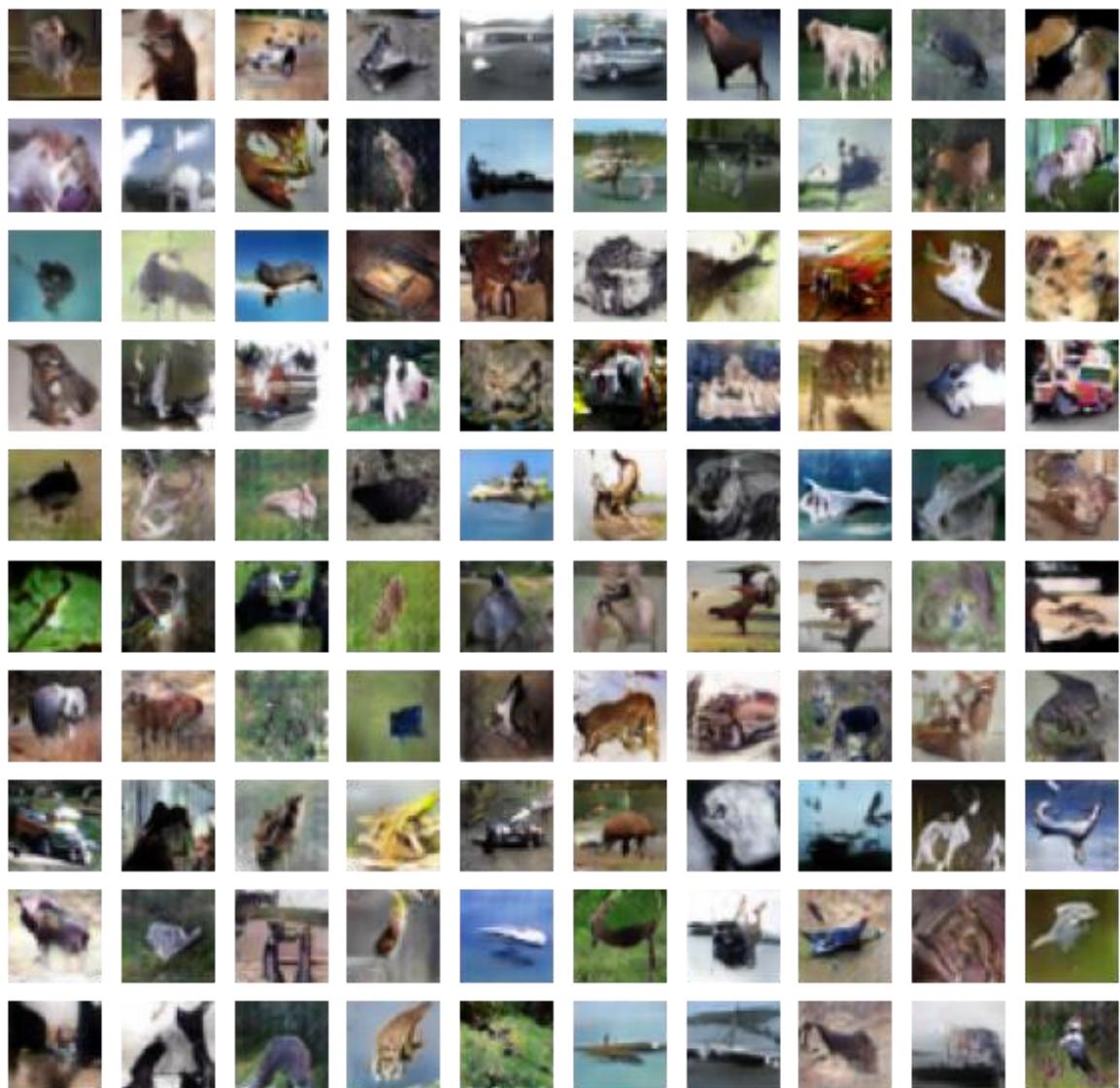
شکل 6-2: نتایج بعد از ایپاک 81



شکل 7-2: نتایج بعد از ایپاک 121



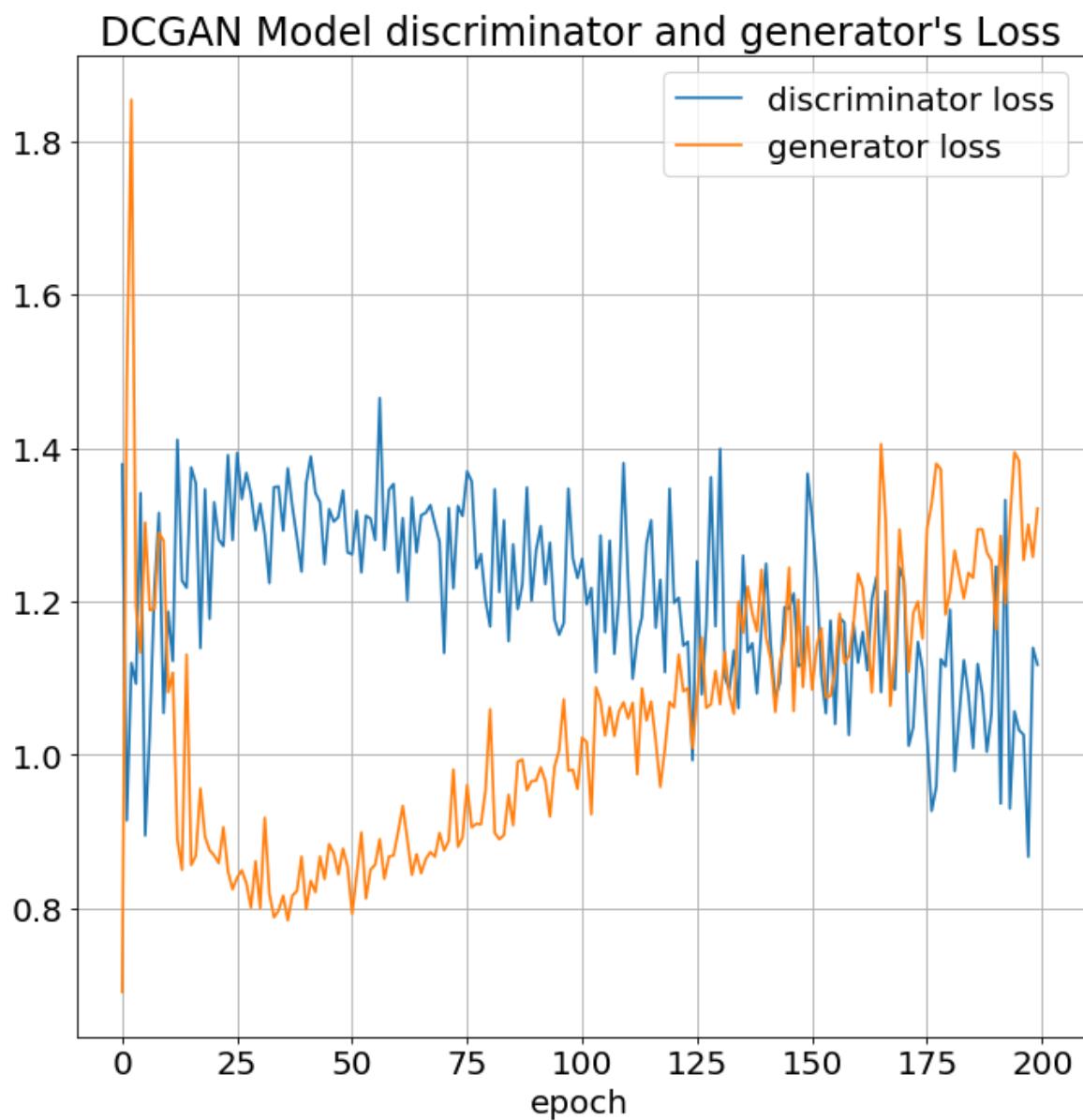
شکل ۸-۲: نتایج بعد از ایپاک ۱۶۱



شکل 9-2: نتایج بعد از اپاک 200 (آخرین ایپاک)

همان طور که در شکل های بالا مشاهده می شود ، نتیجه می گیریم به مرور تصاویر تولید شده توسط generator بهتر شده اند و در ایپاکهای آخر به تصاویر واقعی cifar10 نزدیکتر شده است.

همچنین در شکل 10-2 ، نمودار loss برای دو شبکه generator و discriminator مشاهده می شود:



شکل 2-10: نمودار loss دو شبکه generator و discriminator

طبق شکل 2-10 ، نمودار loss هر دو شبکه حالتی نوسانی داشته و در برخی epoch ، خطای discriminator کمتر است و برعکس در برخی epoch ها generator خطای کمتری دارد. این دقیقا همان چیزی است که انتظار داشتیم و به خوبی تنازع بین generator و discriminator را نشان می دهد. حال پس از آموزش شبکه می توانیم به کمک generator از فضای latent (بردار هایی به طول 100 با توزیع گوسی نرمال) ، تصاویری تولید کنیم که به دیتابیس cifar10 نزدیک باشد. در شکل 2-11 ، تعداد 100 را به کمک generator تولید کردیم که به صورت جدول 10×10 قابل مشاهده است.



شکل 2-11: تولید 100 تصویر به کمک generator

Wasserstein GAN Loss (1)

این تابع خطای Wasserstein در سال 2017 معرفی شده است. این خطای Wasserstein تحت یک مشاهده بر روی GAN های اولیه که مبتنی بر کمینه کردن فاصله بین توزیع های احتمال پیش بینی شده و واقعی (که واگرایی Jensen-Shannon یا Kullback-Leibler نامیده می شوند) بودند، به وجود آمده است.

در استفاده از خطای Wasserstein ، مسئله روی کمینه کردن فاصله Earth-Mover که به Wasserstein-1 distance نیز معروف است ، تعریف می شود. فاصله Earth-Mover فاصله بین دو توزیع احتمال را بر حسب cost یا هزینه ای که برای تبدیل یک توزیع به دیگری لازم است ، تعریف می شود. در شبکه GAN ای که از Wasserstein استفاده می کند ، critic یا تفکیک کننده به generator یا یک منتقد تبدیل می شود و بیشتر از train یا سازنده می شود (مثلا حدود پنج برابر بیشتر). در این صورت critic ، به جای اینکه میزان واقعی بودن تصویر را پیش بینی کند ، به هر عکس با یک عدد حقیقی امتیاز می دهد. همچنین در این مدل مقدار وزنها کوچک نگه داشته می شوند. مثلا در بازه $[-0.01, 0.01]$.

این امتیاز به نحوی توسط critic داده می شود که تا حد ممکن اختلاف امتیاز بین تصویر واقعی و جهلی (real and fake) ، بیشینه شود.

تابع خطای Wasserstein را می توان با محاسبه میانگین امتیاز های داده شده توسط critic به تصاویر real و fake سپس ضرب این دو میانگین به ترتیب در 1 و -1 بدست آورد:

$$L_D = E(D(x)) - E(D(G(x)))$$

$$L_G = E(D(G(z)))$$

از مزیت های خطای Wasserstein این است که در همه جا یک مقدار گرادیان کافی در آپدیت وزنها ، فراهم می کند که این باعث آموزش پیوسته شبکه می شود. همچنین کم بودن خطای Wasserstein ارتباط مستقیمی با بهتر شدن کیفیت تصاویر تولیدی generator توسط generator دارد و در نتیجه در صورت استفاده از این تابع خطای Wasserstein کمینه کردن خطای generator هستیم.

Least Squares GAN Loss (2)

این تابع Loss در مقاله ای با عنوان "Least Squares Generative Adversarial Networks" توسط Xudong Mao در سال 2016 مطرح شد.

ایده آنها مبتنی بر محدودیت های استفاده از خطای Binary cross entropy بود که در حالتی که تفاوت تصاویر تولید شده با تصاویر واقعی بسیار زیاد باشد می تواند به مقادیر بسیار کم گرادیان و در نتیجه gradient vanishing آموزش generator منجر شود.

در روش discriminator ، Least Squares سعی در کمینه کردن مجموع مربعات خطای خود (تفاوت مقادیر پیش بینی شده با مقادیر واقعی) را دارد. به طور دقیقترا اگر $D(x)$ خروجی discriminator به ازای ورودی واقعی x و $G(z)$ تصویر تولیدی generator با استفاده از بردار نویز z باشد در این صورت سعی می کند عبارت زیر را کمینه کند.

$$L_D = (D(x) - 1)^2 + (D(G(z)))^2$$

با استدلالی مشابه generator سعی در کمینه کردن عبارت زیر را دارد:

$$L_G = (D(G(z)) - 1)^2$$

کمینه کردن L2 Loss ، Least Squares نیز نامیده می شود.

$$L2 Loss = (y_{predicted} - y_{true})^2$$

مزیت روش Least Squares این است که جریمه بیشتری به ازای خطاهای بزرگ می دهد و در نتیجه وزنهای را بیشتر در جهت کمینه شدن خطای آپدیت می کند و بنابراین مشکلات gradient vanishing نیز کمتر رخ می دهد.

Minimax GAN Loss (3)

قاعده Minimax یک استراتژی بهینه سازی برای کمینه کردن همزمان خطای Discriminator و Generator است که ایده آن از نظریه بازی ها آمده است. در یک GAN در واقع Generator و Discriminator دو بازیکن هستند که در هر مرحله هر یک از آنها سعی می کنند خطای خود را کمتر کنند. به طور دقیق تر:

$$L_D = E(\log D(x) + \log(1 - D(G(z))))$$

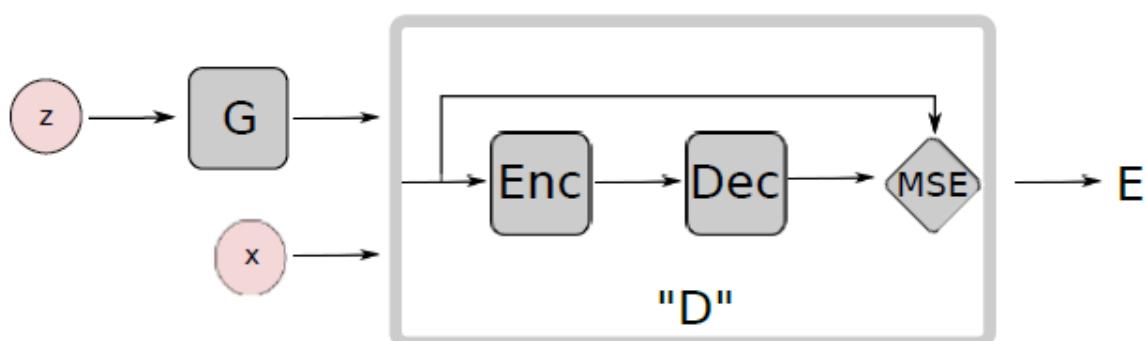
$$L_G = \log(1 - D(G(z)))$$

در واقع Discriminator سعی در بیشینه کردن L_D و generator سعی در کمینه کردن L_G دارد. این تابع خطای تحلیل های تئوری بسیار مفید است اما در عمل این خطای Generator به اشباع می رسد. یعنی نمی تواند با سرعتی که Discriminator آموزش می گیرد، یاد بگیرد و در نتیجه از یک جا به بعد همیشه برندۀ می شود و عمل یادگیری به خوبی شکل نمی گیرد.

EBGAN یا Energy Based GAN (4)

در استفاده از این تابع loss، ما به عنوان یک تابع انرژی (Energy function)، در نظر می گیریم. مقدار انرژی ای که توسط discriminator محاسبه می شود را می توان به عنوان تابع خطای generator در نظر گرفت. طوری آموزش داده می شود که به نواحی با چگالی داده بالا، انرژی کمتری assign کند و بیرون این نواحی، انرژی بیشتری assign کند. از طرف دیگر می توان generator را به عنوان یک تابع پارامتری در نظر گرفت که در نواحی ای که داده تولید می کند انرژی کمتری به آنها assign می کند.

در این روش، از یک اتوانکدر نیز برای پیش بینی و بازسازی خطای استفاده می شود. به این صورت که ابتدا اتوانکدر روی داده های اصلی train می شود. سپس داده های تولید شده توسط generator از این اتوانکدر عبور می کنند. با این کار، نویز تصاویری که ضعیف و بی کیفیت تولید شده اند، یک بازسازی بسیار بدتری در خروجی اتوانکدر خواهد داشت و بنابرین یک معیار بسیار خوبی از کیفیت تصاویر تولیدی خواهد بود. در این approach با انتخاب مناسب ضریب regularization می توان از mode collapse جلو گیری کرد (مشکل mode collapse زمانی رخ می دهد که تصور تکراری را بارها تولید کند). به صورت تجربی نشان داده شده است در صورت استفاده از این loss، سرعت GAN بیشتر، پایداری آن بیشتر و نسبت به تغییرات پارامترها robust خواهد بود. در شکل 2-12، ساختار EBGAN نشان داده شده است.



شکل 2-12: ساختار EBGAN

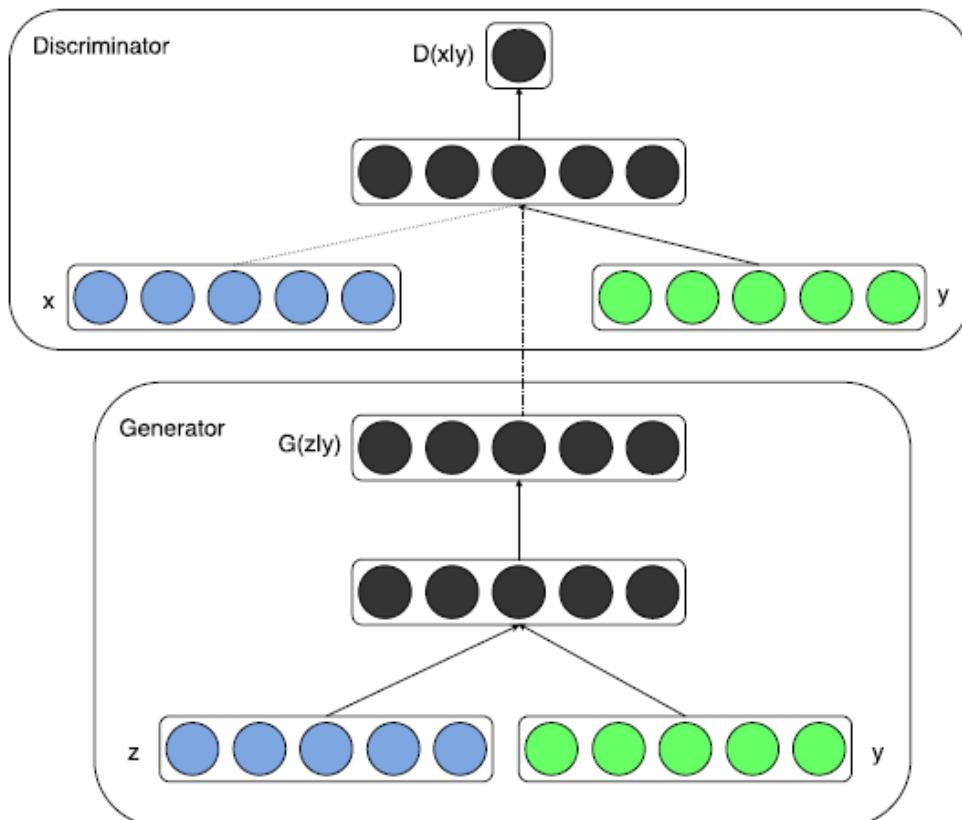
سوال 3 – شبکه های conditional GANs

(الف)

شبکه های Conditional GANs یا CGAN ها دسته خاصی از GAN ها هستند که در آنها روی اطلاعات اضافه ای مانند y مشروط (conditional) می شوند. اطلاعات Generator و Discriminator اضافه y می تواند هر نوع اطلاعات کمکی باشد. مثلا y می تواند کلاس داده ها یا اطلاعات بدست آمده از بقیه روشها باشد یا می تواند یک عکس باشد. ما می توانیم این conditioning را به عنوان یک لایه ورودی جدید به هر دو شبکه generator و discriminator اعمال کنیم. در این صورت تابع هدف^۱ یا بازی two-player minmax به صورت زیر خواهد شد:

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{\text{data}}(x)} [\log D(x|y)] + \mathbb{E}_{z \sim p_z(z)} [\log(1 - D(G(z|y)))].$$

در داخل مقاله ، ساختار یک شبکه CGAN ساده به صورت شکل ۳-۱ نشان داده شده است.



شکل ۳-۱: ساختار یک شبکه CGAN ساده

Objective function¹

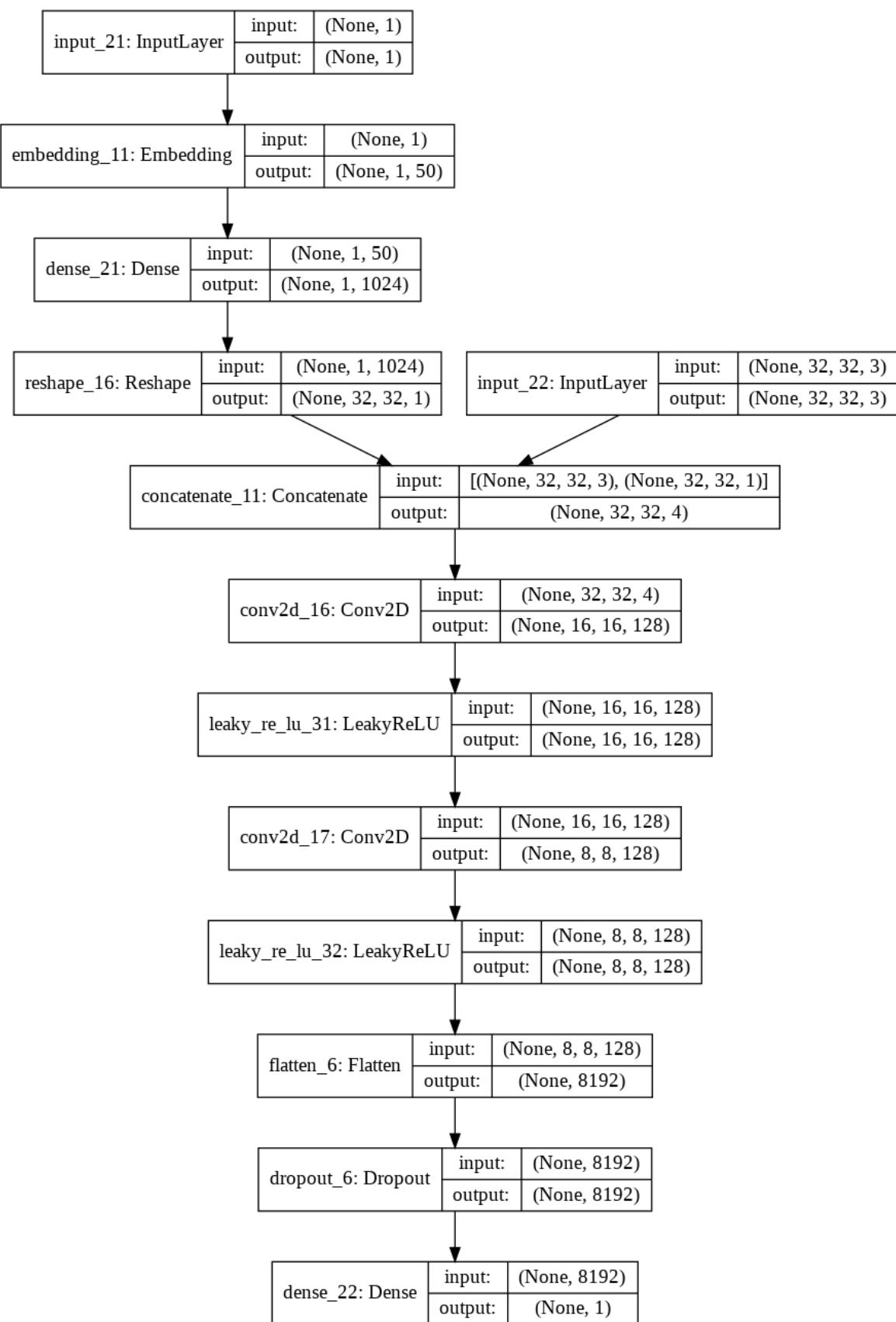
در واقع دلیل اصلی به وجود آمدن این شبکه ها این است که بر خلاف GAN های معمولی ، کنترلی بر روی تصاویر تولیدی توسط Generator داشته باشیم.

از دلایلی که از لیبل کلاسها به عنوان اطلاعات اضافه استفاده می کنیم این است که اولاً قابلیت یادگیری GAN را بهبود ببخشیم (از لحاظ پایداری و سرعت training و بهبود کیفیت تصاویر تولیدی) و دوماً پس از آموزش شبکه ، با استفاده از Generator بتوانیم تصاویر جدید با لیبل خاصی تولید کنیم.

روش های مختلفی برای اضافه کردن z به عنوان ورودی وجود دارد. به عنوان مثال اگر z لیبل تصاویر باشد می توان آن را به صورت بردار one hot در آورد و سپس از یک لایه Dense با تابع فعالساز خطی با طول برابر تعداد پیکسل های تصاویر دیتاست مورد استفاده ، عبور داده و سپس به صورت ابعاد تصاویر دیتاست ، Reshape کرده و به عنوان یک کanal (یا feature map) به عکس ورودی اضافه کنیم. ما در این سوال از این تکنیک استفاده می کنیم.

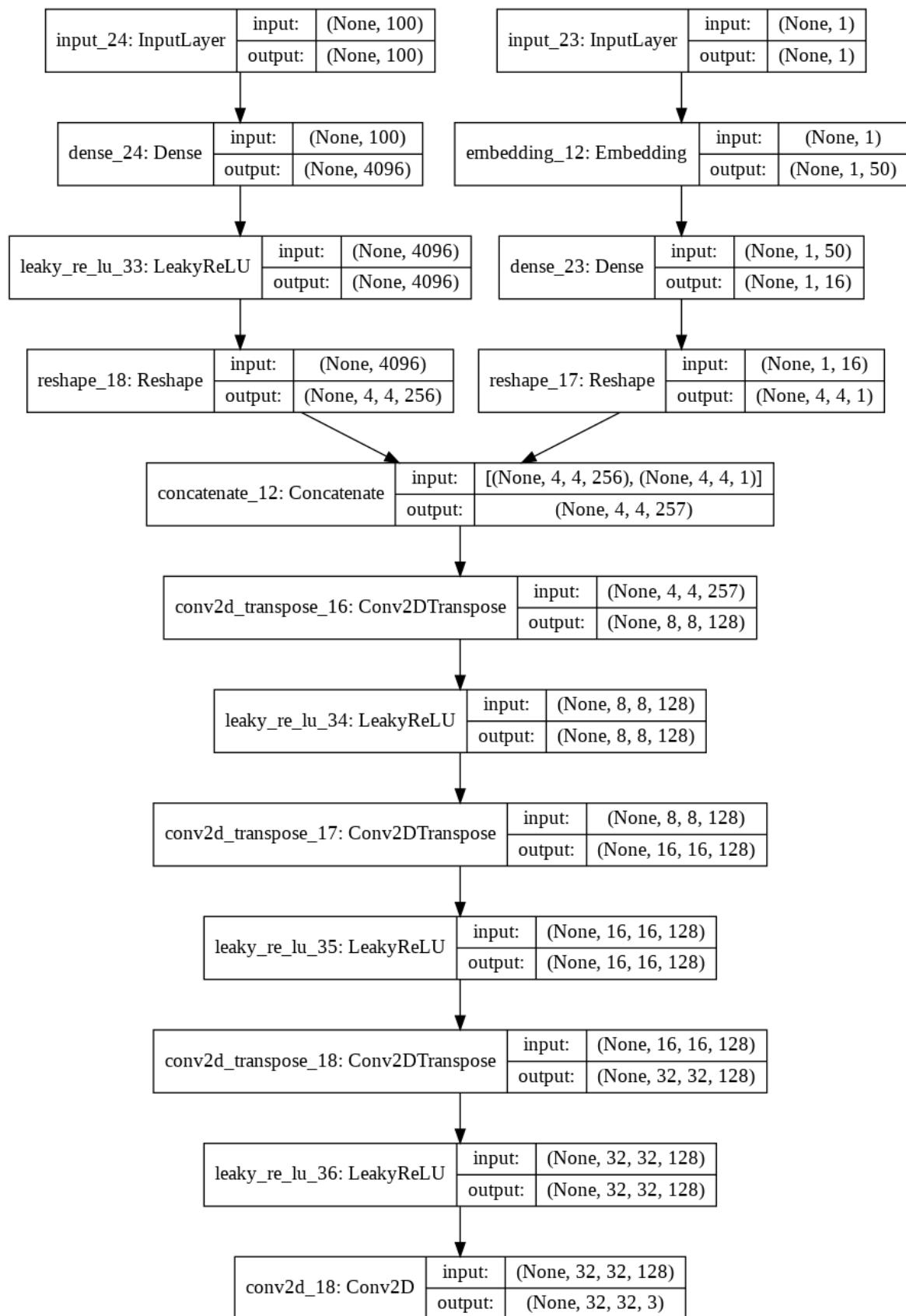
ساختار و معماری شبکه Discriminator که در این سوال آن را پیاده سازی کرده ایم به کمک دستور plot_model در شکل 2-3 نشان داده شده است:

همان طور که در شکل 2-3 مشاهده می شود ، ورودی لیبل که بعد 1 دارد را ابتدا وارد لایه Embedding با طول 50 می کنیم تا شبکه یک representation دیگر به طول بردار 50 از لیبل ها یاد بگیرد. خروجی لایه را وارد لایه Dense با طول $32 \times 32 = 1024$ می کنیم با تابع فعالساز linear می کنیم و سپس با استفاده از لایه Reshape ، ابعاد آن را به $(32, 32, 1)$ تغییر می دهیم و در نهایت به ورودی تصویر $3 \times 32 \times 32$ (یکی از تصاویر دیتاست cifar10) با استفاده از لایه Concatenate ، به عنوان کanal چهارم تصویر ، اضافه و ادغام می کنیم. خروجی این لایه ابعادی به صورت $(32, 32, 4)$ دارد. در ادامه همان طور که در شکل 2-3 مشاهده می شود ، تمامی لایه هایی که قرار می دهیم (به طور خلاصه دو لایه Conv2D با تابع فعالساز leaky ReLU و در نهایت لایه Dense و Dropout) دقیقاً مشابه با شبکه DCGAN می باشد که در قسمت الف سوال 2 با جزئیات توضیح دادیم.



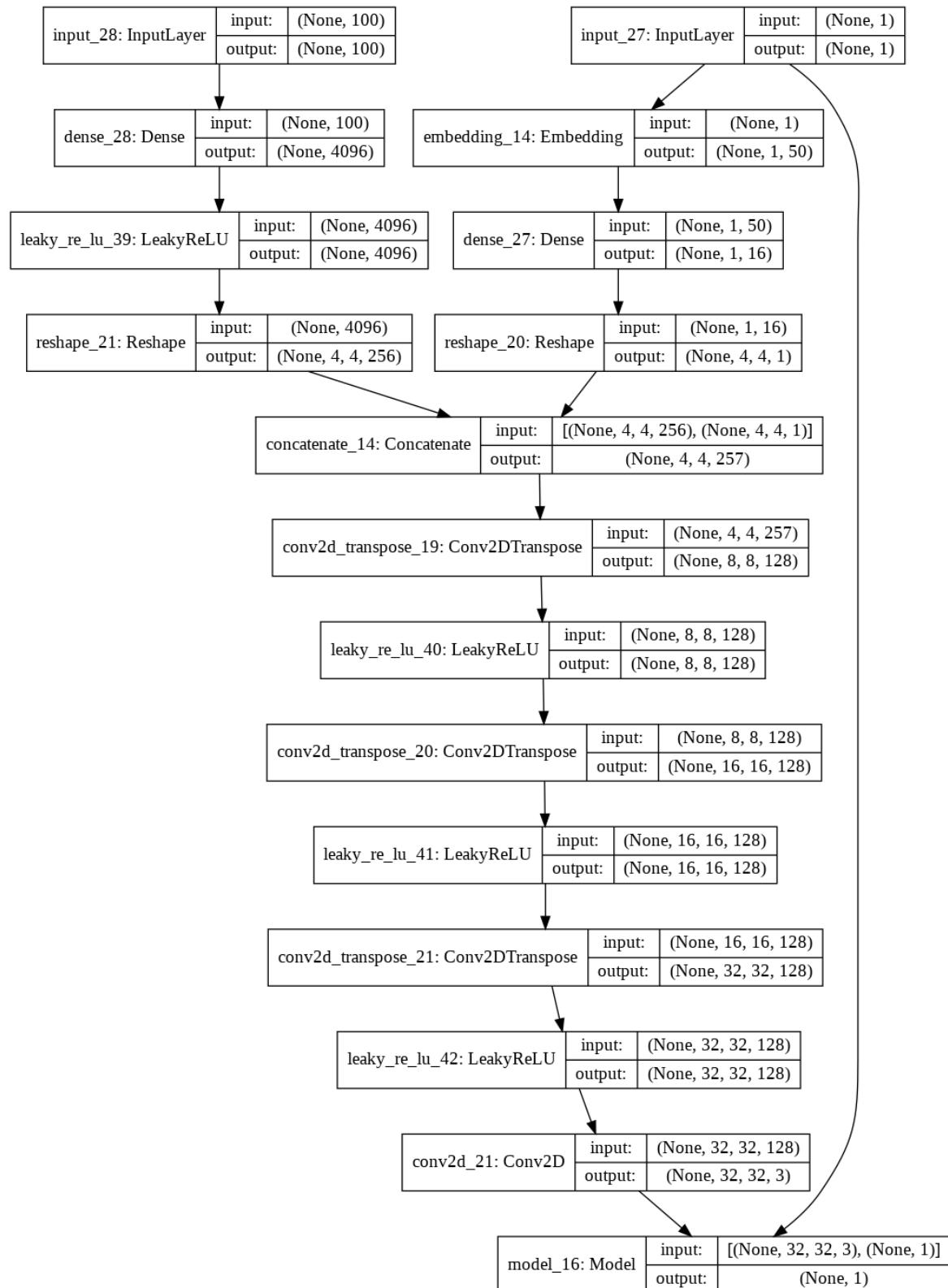
شکل ۳-۲: ساختار و معماری شبکه Discriminator

ساختار شبکه Generator که در این سوال پیاده سازی می کنیم در شکل 3-3 قابل مشاهده است. در این شبکه نیز مشابه تکنیکی که در Discriminator استفاده کردیم، عمل می کنیم. به این صورت که ورودی لیبل را از یک لایه Embedding با طول 50 و سپس از لایه Dense با طول $16 = 4 \times 4$ عبور داده و با کمک لایه Reshape به ابعاد (4, 4, 1) تغییر می دهیم. سپس این خروجی را با خروجی با ابعاد (4, 4, 256) که از فضای latent (بردار های رندوم گوسی به طول 100) بدست آمده است مطابق شکل concatenate 3-3 گرده و بقیه شبکه دقیقاً مانند شبکه DCGAN است که در سوال ۲ قسمت الف توضیح دادیم. به این صورت که سه لایه Conv2DTranspose با تابع فعالساز leaky ReLU و سپس یک لایه Conv2D قرار می دهیم که جزئیات آن در شکل 3-3 قابل مشاهده است.



شکل ۳-۳: معماری Generator

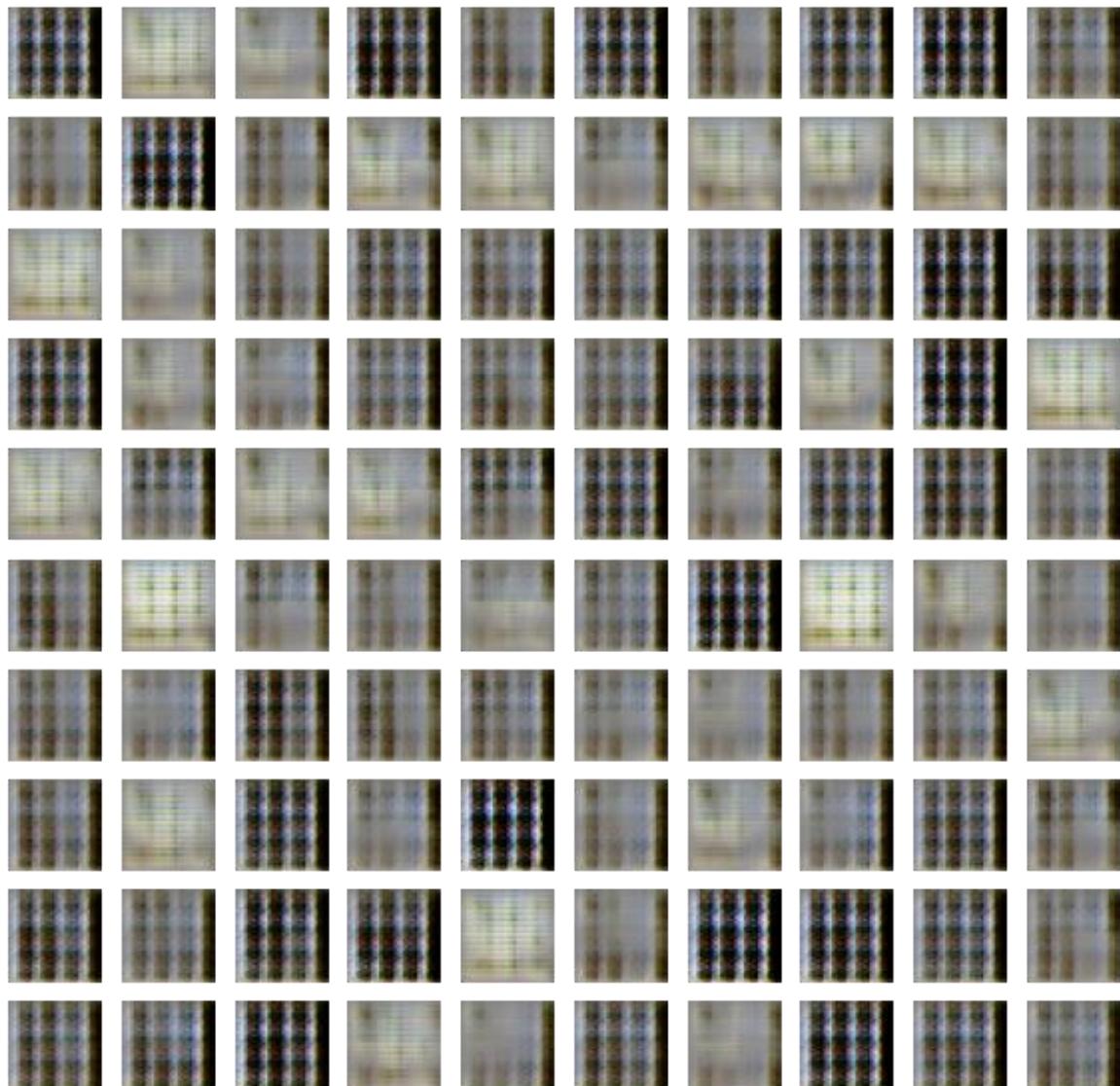
حال برای آموزش شبکه generator دو شبکه را با هم ترکیب کرده و مدل کلی CGAN را تعریف می کنیم. ساختار کلی شبکه CGAN که در این سوال پیاده سازی کرده ایم مطابق شکل ۳-۴ است.



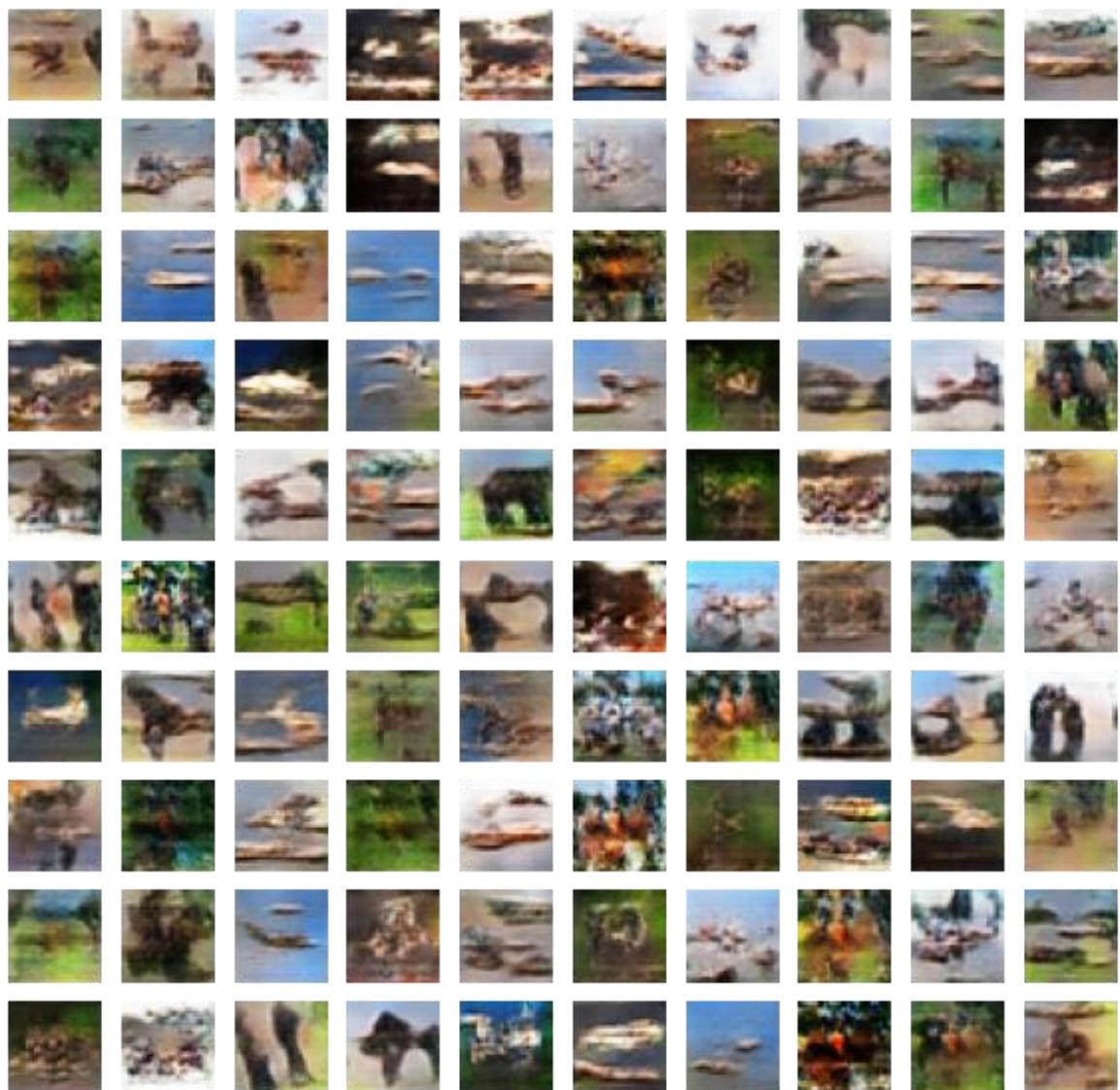
شکل ۳-۴ ساختار کلی شبکه CGAN

(ب)

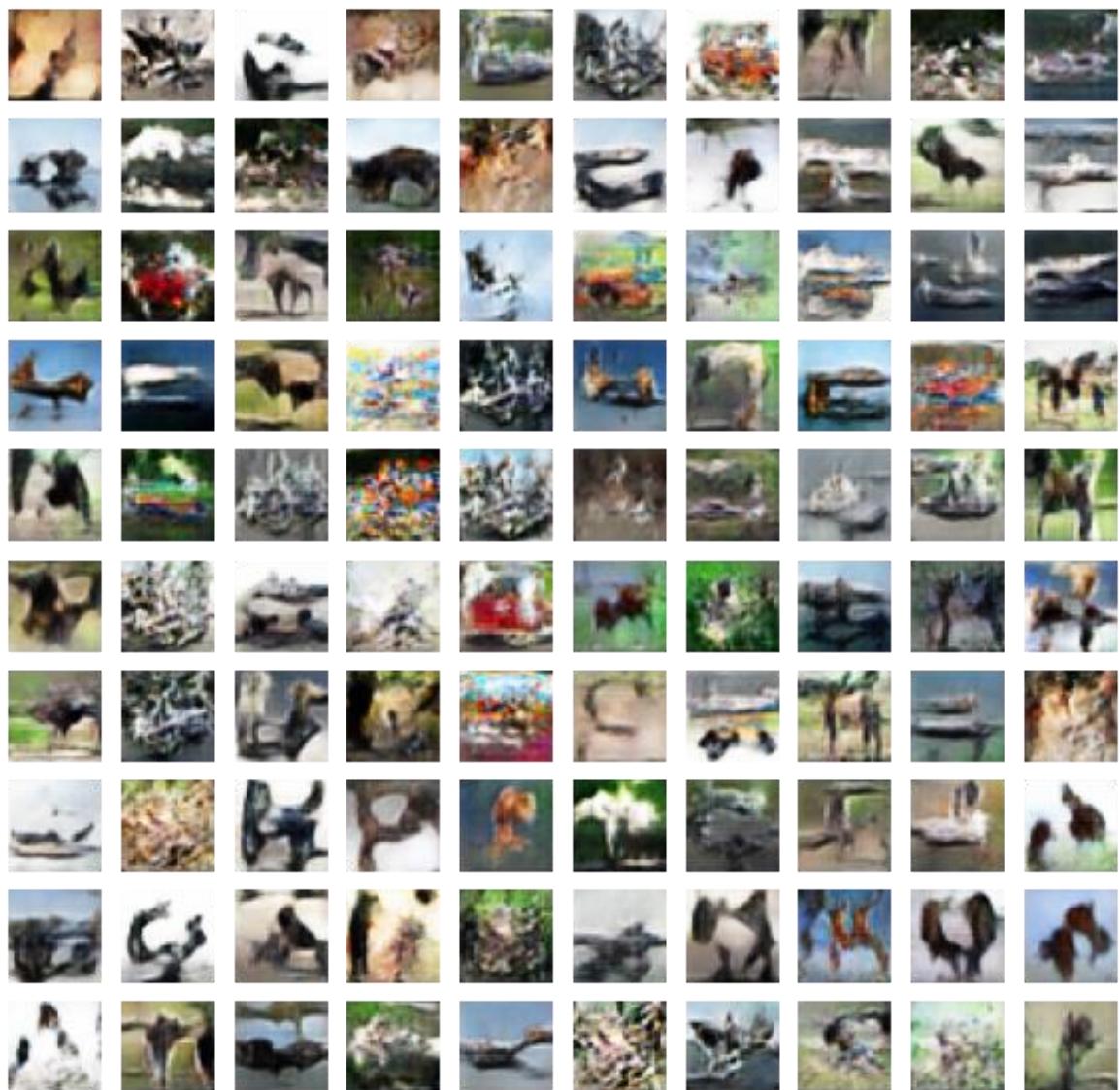
در این قسمت شبکه CGAN را مطابق شکل های 3-2 ، 3-3 و 3-4 پیاده سازی می کنیم و آن را در 201 epoch آموزش می دهیم. جزئیات کدها در داخل فایل notebook توضیح داده شده است. شکل های 3-5 ، 3-6 ، 3-7 ، 3-8 ، 3-9 و 3-10 نتایج را در epoch های 1 ، 41 ، 81 ، 121 ، 161 و 201 نشان می دهد. همانطور که مشاهده می شود کیفیت تصاویر تولید شده به مرور بهتر می شود.



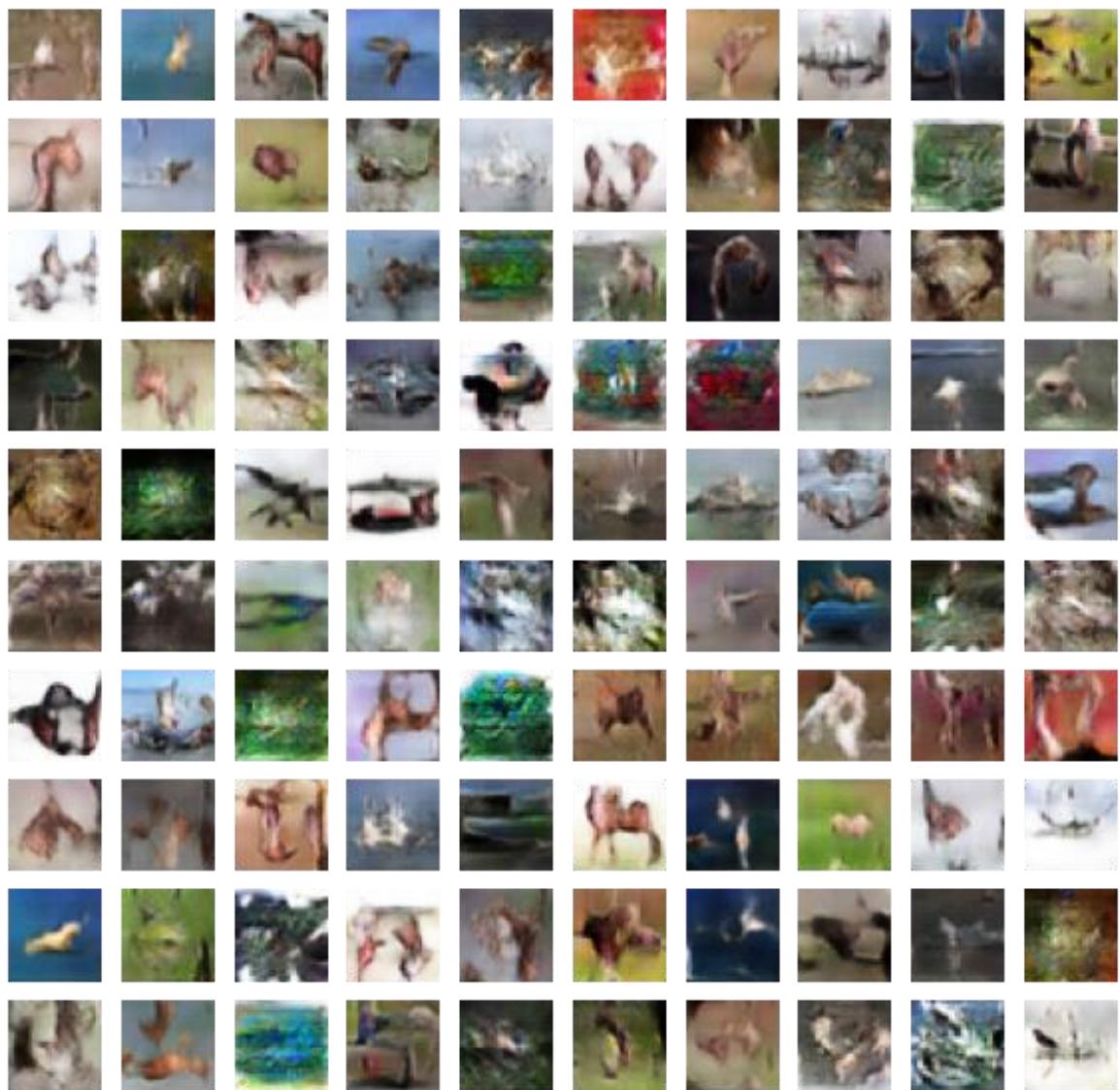
شکل 5-3: ایپاک 1



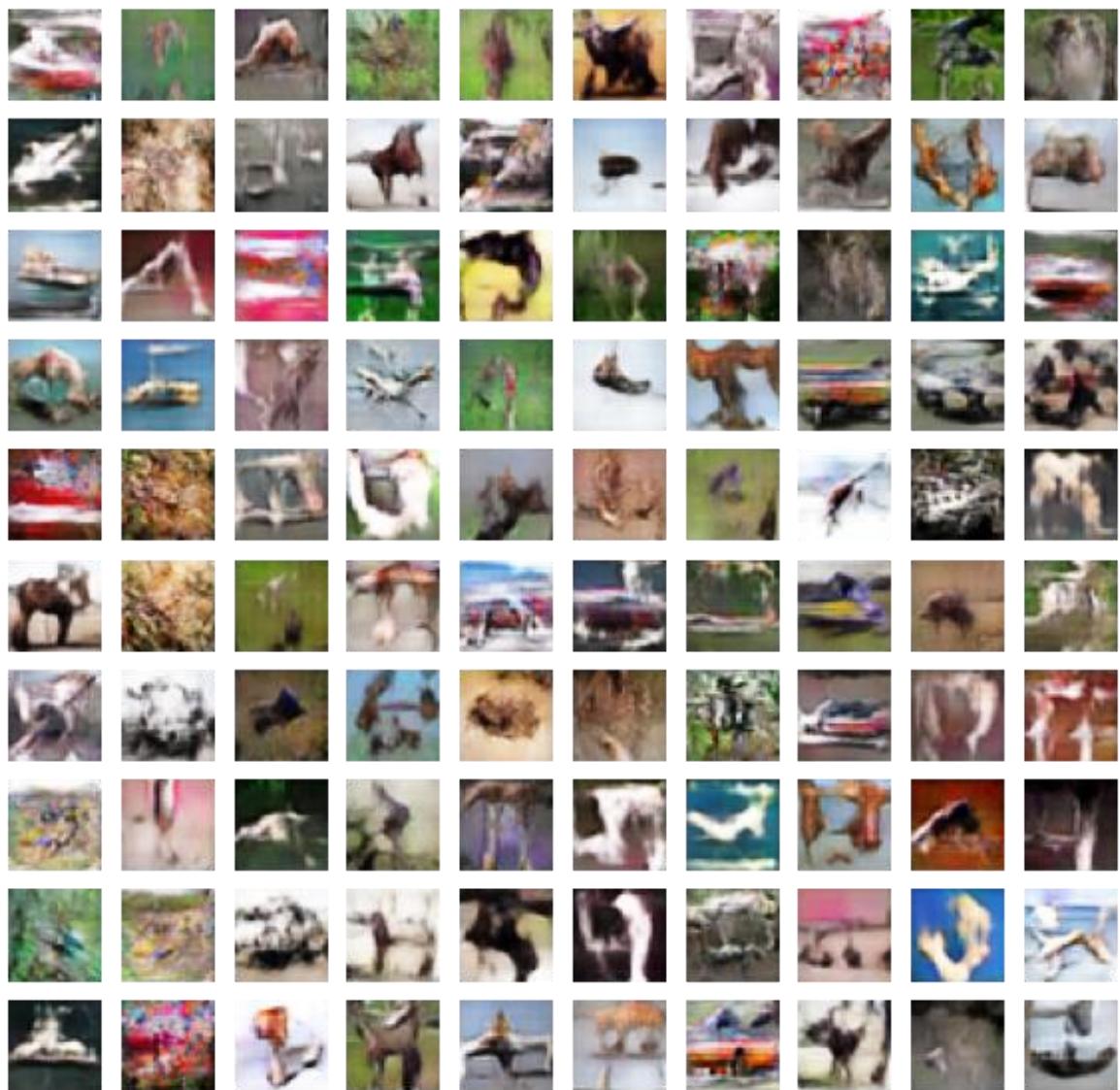
شکل ۶-۳ ایپاک 41



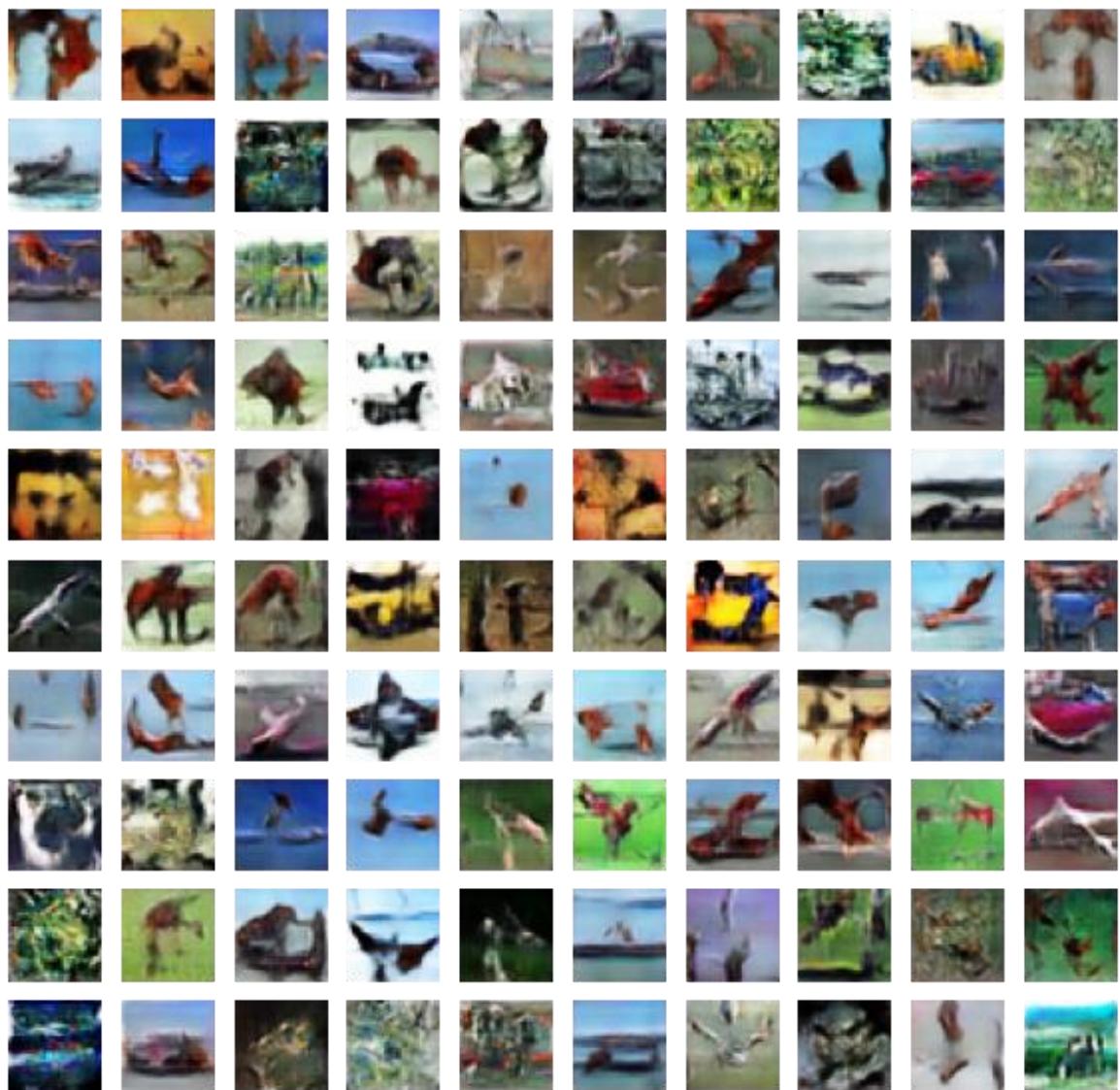
شکل 3-7: ایپاک 81



شکل 3-8: ایپاک 121

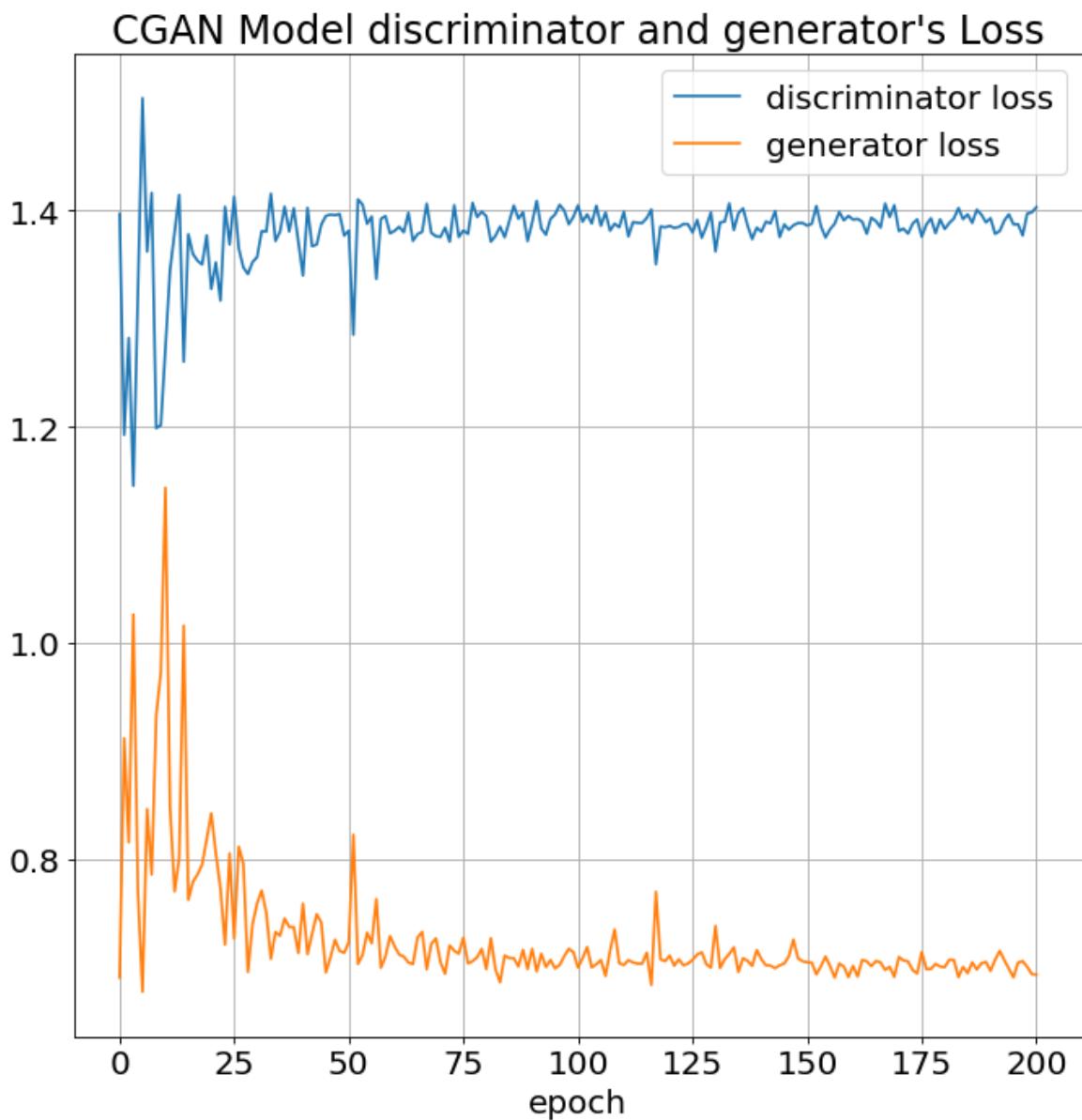


شکل 9-3: ایپاک 161



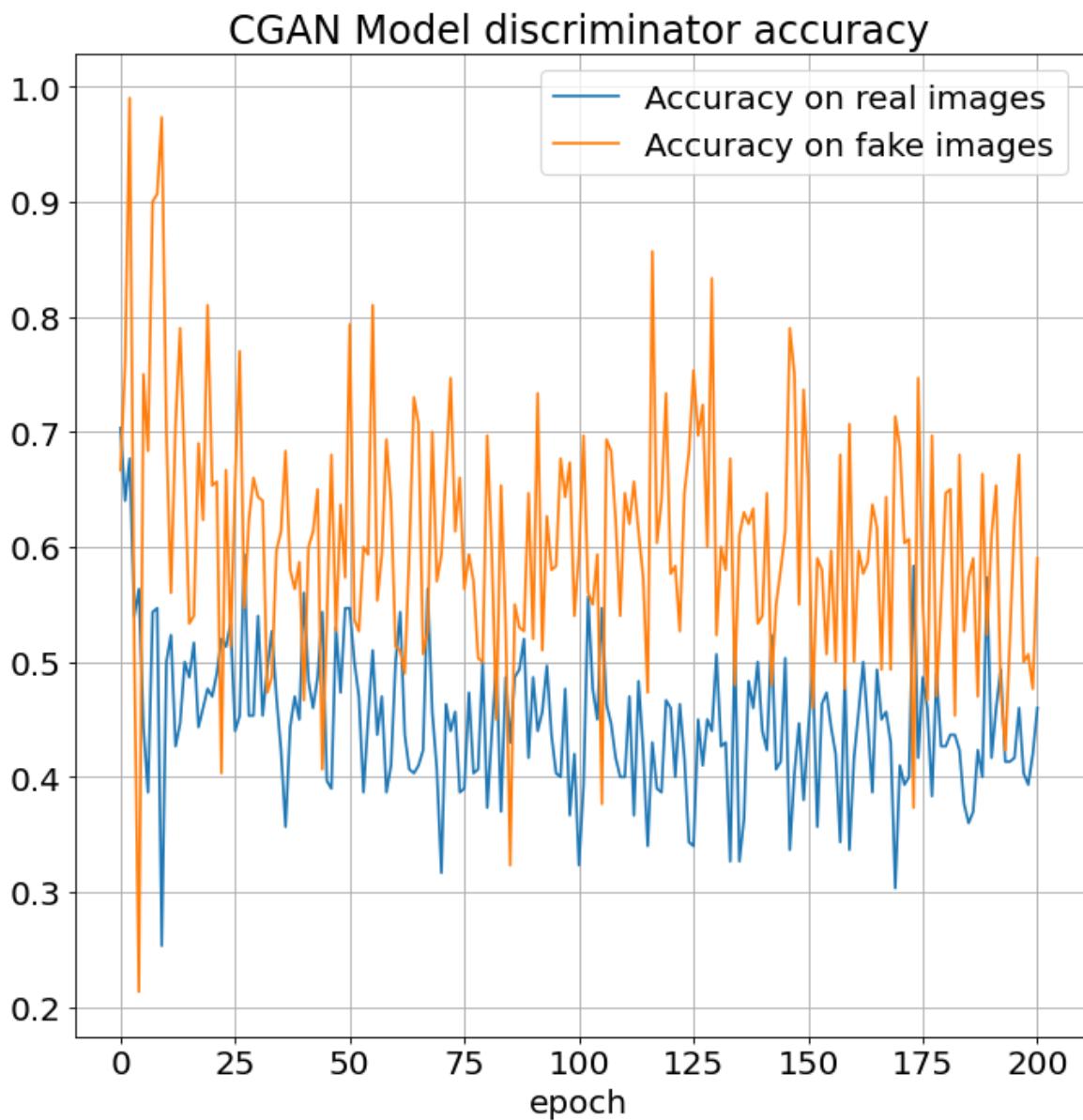
شکل 3-10: ایپاک 201 (آخرین ایپاک)

نمودار loss نیز در شکل 3-11 قابل مشاهده است:



شکل 3-11: نمودار loss شبکه generator و discriminator دو

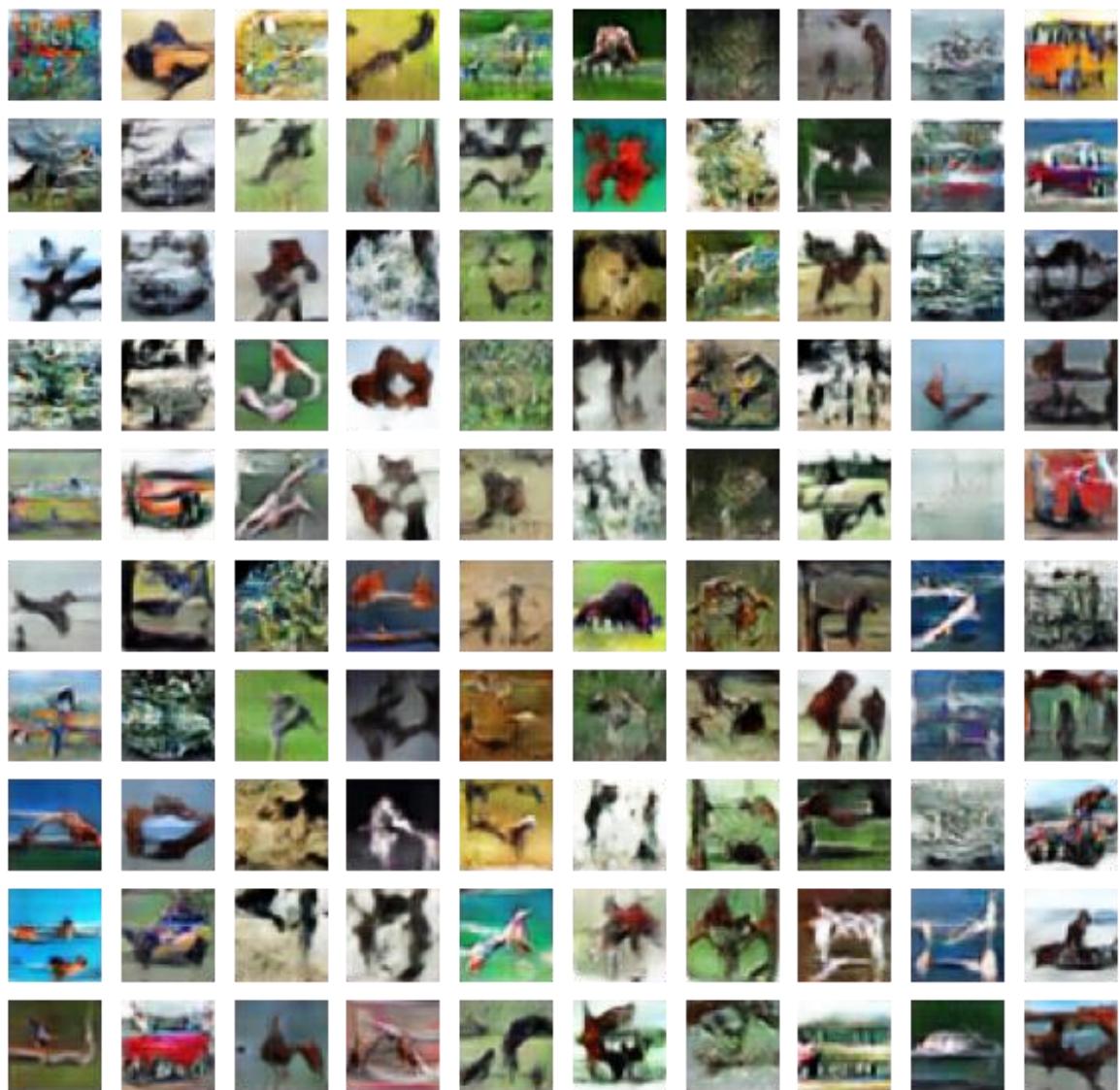
همان طور که از شکل 3-11 مشاهده می شود ، نمودار loss طبق چیزی که انتظار داشتیم نوسانی است و نشان می دهد که روند آموزش به خوبی پیش می رود. لازم به ذکر است دلیل اینکه discriminator بالاتر از generator است این است که خطای discriminator را روی بچ داده های real و fake جمع کرده ایم. همچنین شکل 3-12 ، نمودار accuracy شبکه discriminator را روی داده های real و fake بر حسب epoch نشان می دهد.



شکل 3-12: نمودار loss

نوسانات شکل 3-12 نیز دقیقاً ممتازه بین generator و discriminator را نشان می‌دهد.

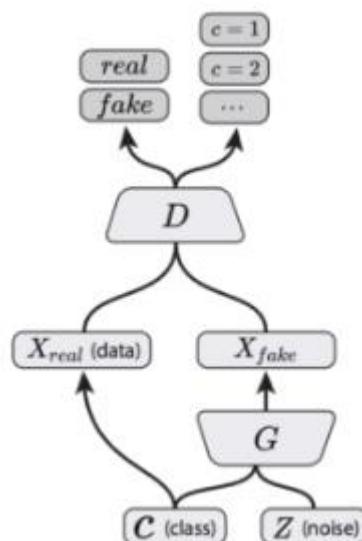
حال پس از آموزش شبکه، می‌توانیم به کمک generator از فضای latent داده‌های با لیبل خاص تولید کنیم. به عنوان مثال در شکل 3-13 تعداد 100 تصویر از هر لیبل 10 تصویر را تولید می‌کنیم که هر ستون، تصاویر مربوط به یک لیبل را نشان می‌دهد:



شکل 3-13: تصاویر تولیدی توسط generator پس از آموزش CGAN

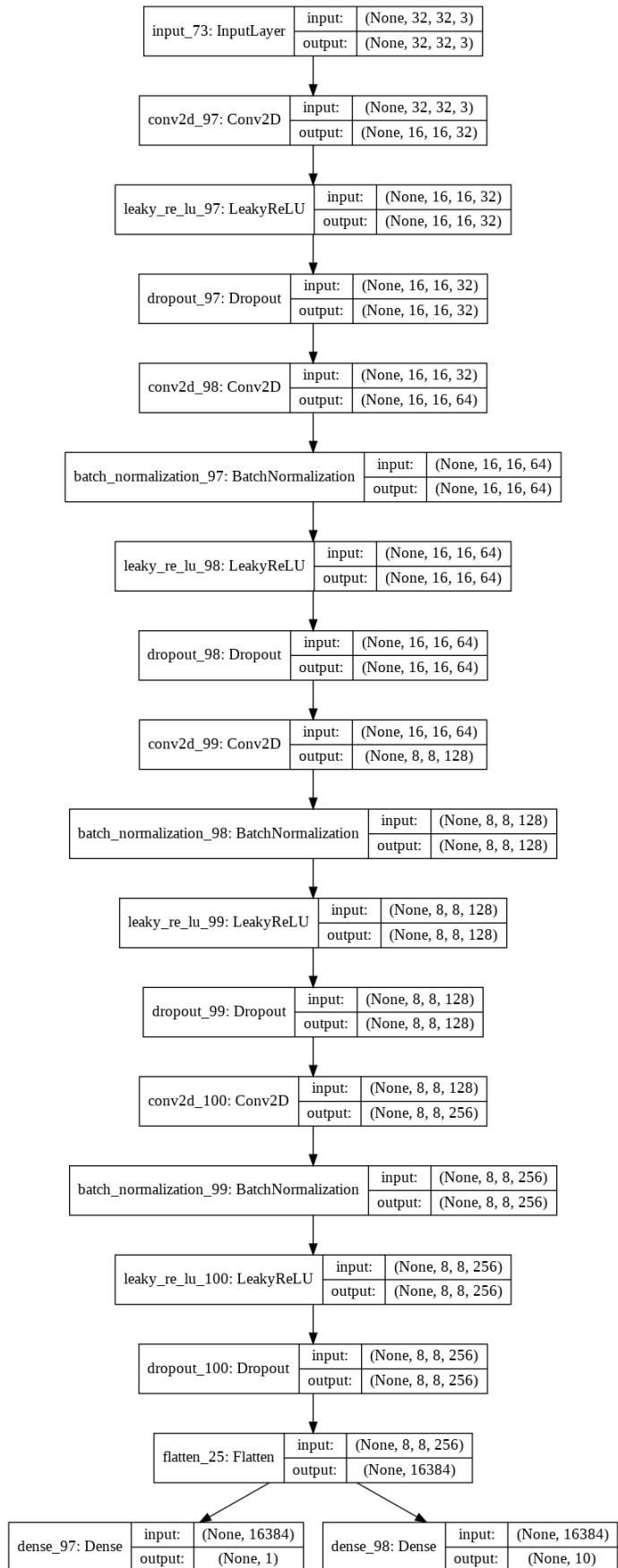
(ج)

در این قسمت شبکه AC-GAN را با اختصار Auxiliary Classifier GAN یا به اختصار AC-GAN می‌پیاده سازی می‌کنیم. ساختار این شبکه بسیار شبیه به CGAN می‌باشد با این تفاوت که در این مدل، لیبل ورودی را به عنوان ورودی به شبکه Discriminator نمی‌دهیم و بر عکس آن را توسط Discriminator، پیش‌بینی می‌کنیم. بنابرین در شبکه AC-GAN، Discriminator دو خروجی دارد. یک خروجی عددی بین صفر و یک است که احتمال واقعی بودن تصویر ورودی را نشان می‌دهد. خروجی جدید برداری به طول تعداد کلاس‌ها (در این سوال چون دیتابست cifar10 استفاده می‌کنیم طول بردار 10 است) می‌باشد که پس از عبور از لایه softmax احتمال لیبل تصویر ورودی را پیش‌بینی می‌کند. ساختار generator دقیقاً مانند شبکه CGAN است. به عبارتی در این شبکه نیز generator دارای دو ورودی label و برداری (به طول 100 دارای توزیع نرمال استاندارد گوسی و رندوم) از فضای latent می‌باشد. در نتیجه تفاوت اصلی شبکه AC-GAN با CGAN در مدل Discriminator آنهاست به طوری که در AC-GAN برخلاف Discriminator در CGAN فقط ورودی تصویر دارد و لیبل آن را در خروجی خود پیش‌بینی می‌کند. شماتیک کلی AC-GAN در شکل 3-14 قابل مشاهده است.



شکل 3-14 شبکه AC-GAN

معماری AC-GAN در discriminator که آن را پیاده سازی کرده ایم، در شکل 3-15 قابل مشاهده است.

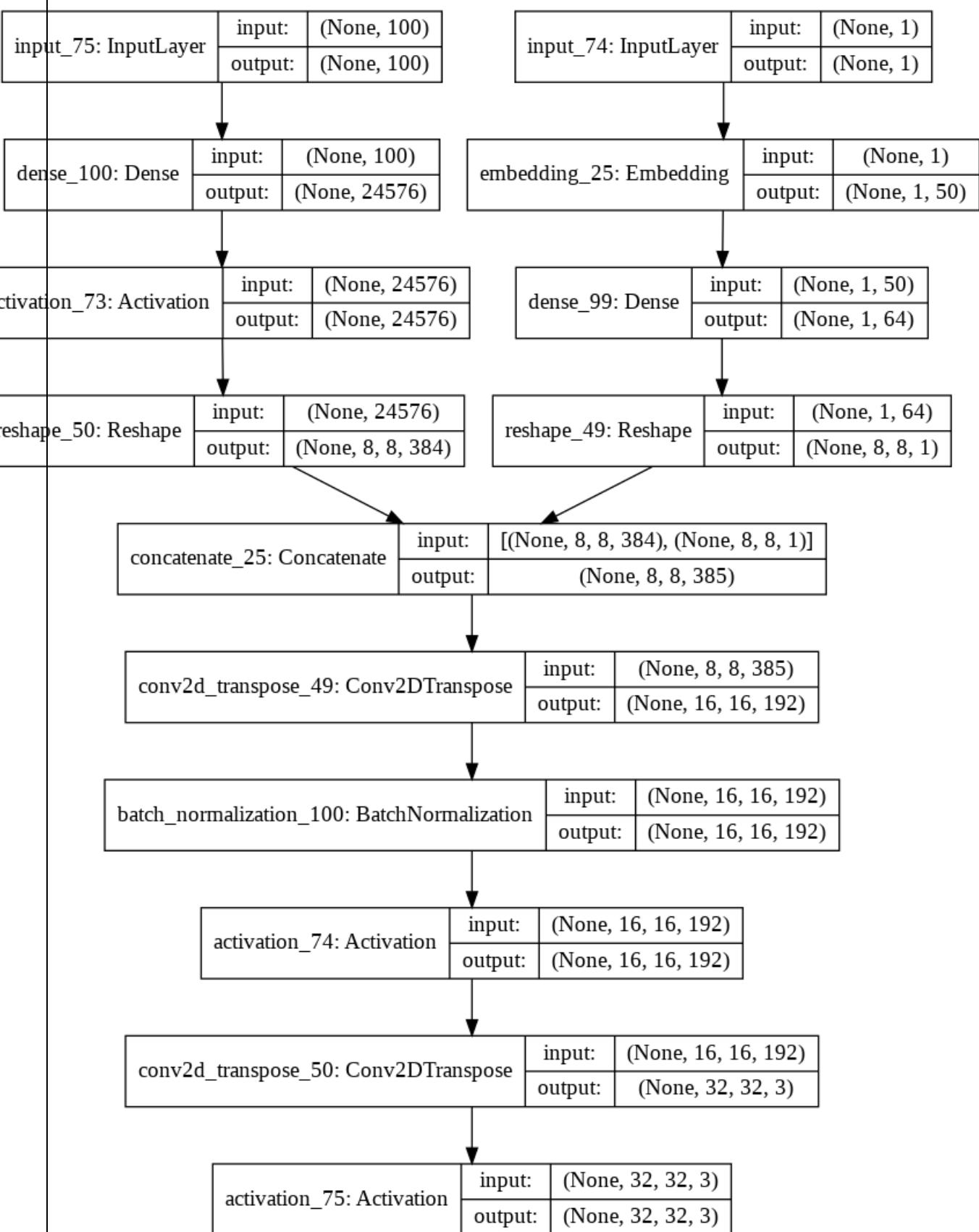


شکل 3-15: معماری **discriminator** در **AC-GAN**

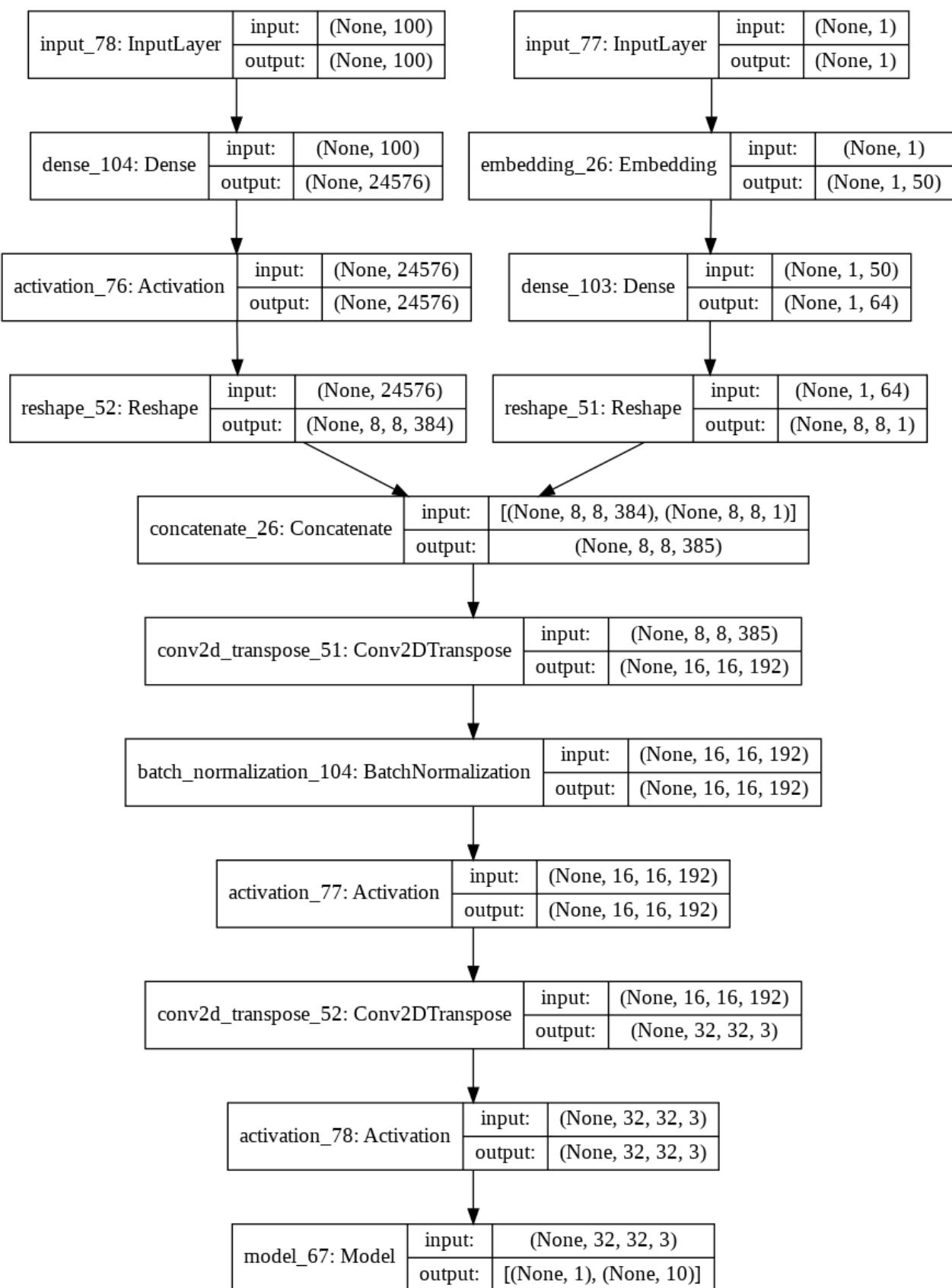
همان طور که در شکل 3-15 مشاهده می شود ، از 4 لایه Conv2D با تابع فعالساز leaky ReLU با شبکه 0.2 و همچنین لایه های Dropout و BatchNormalization به منظور افزایش پایداری شبکه هنگام آموزش و پرهیز از gradient vanishing استفاده کرده ایم. در نهایت دو خروجی خواهیم داشت که یک خروجی پس از عبور از لایه sigmoid احتمال واقعی بودن عکس را نشان می دهد و دیگری پس از عبور از softmax یک بردار احتمال به طول 10 می دهد که احتمالات لیبل تصویر ورودی را نشان می دهد.

ساختار شبکه generator که بسیار مشابه CGAN است در شکل 3-16 نشان داده شده است. همان طور که مشاهده می شود، در AC-GAN نیز Generator دو ورودی لیبل و یک بردار نویز از فضای latent به طول 100 دریافت می کند. ورودی لیبل را پس از عبور از لایه Embedding به طول 50 ، از یک لایه Dense به طول 64 عبور می دهیم و سپس به ابعاد (8, 8, 1) با لایه Reshape تبدیل می کنیم. از طرفی دیگر ورودی نویز را پس از عبور از لایه Dense و سپس Reshape به ابعاد (8, 8, 384) تغییر می دهیم. در نهایت دو خروجی تا خروجی با ابعاد (8, 8, 385) بدست آید. در آخر طبق شکل 3-16 این خروجی را از دو لایه Conv2DTranspose با تابع فعالساز relu و همچنین لایه های Batch Normalization عبور می دهیم تا در نهایت در خروجی تصویر با ابعاد (32, 32, 3) تولید کنیم. (در لایه آخر از تابع فعالساز tanh استفاده کرده ایم).

توجه داشته باشید مشابه قبل ، دو شبکه generator و discriminator را باید جدا از هم آموزش داد. به همین خاطر مانند DCGAN و CGAN که در سوالهای قبل پیاده سازی کردیم یک مدل دیگر که ترکیبی از Generator و Discriminator می باشد را به منظور آموزش generator تعریف می کنیم و در آن قابلیت trainable بودن discriminator را False می کنیم تا وزنهای discriminator هنگام آموزش ثابت بماند. ساختار و معماری این شبکه combine شده در شکل 3-17 قابل مشاهده است.

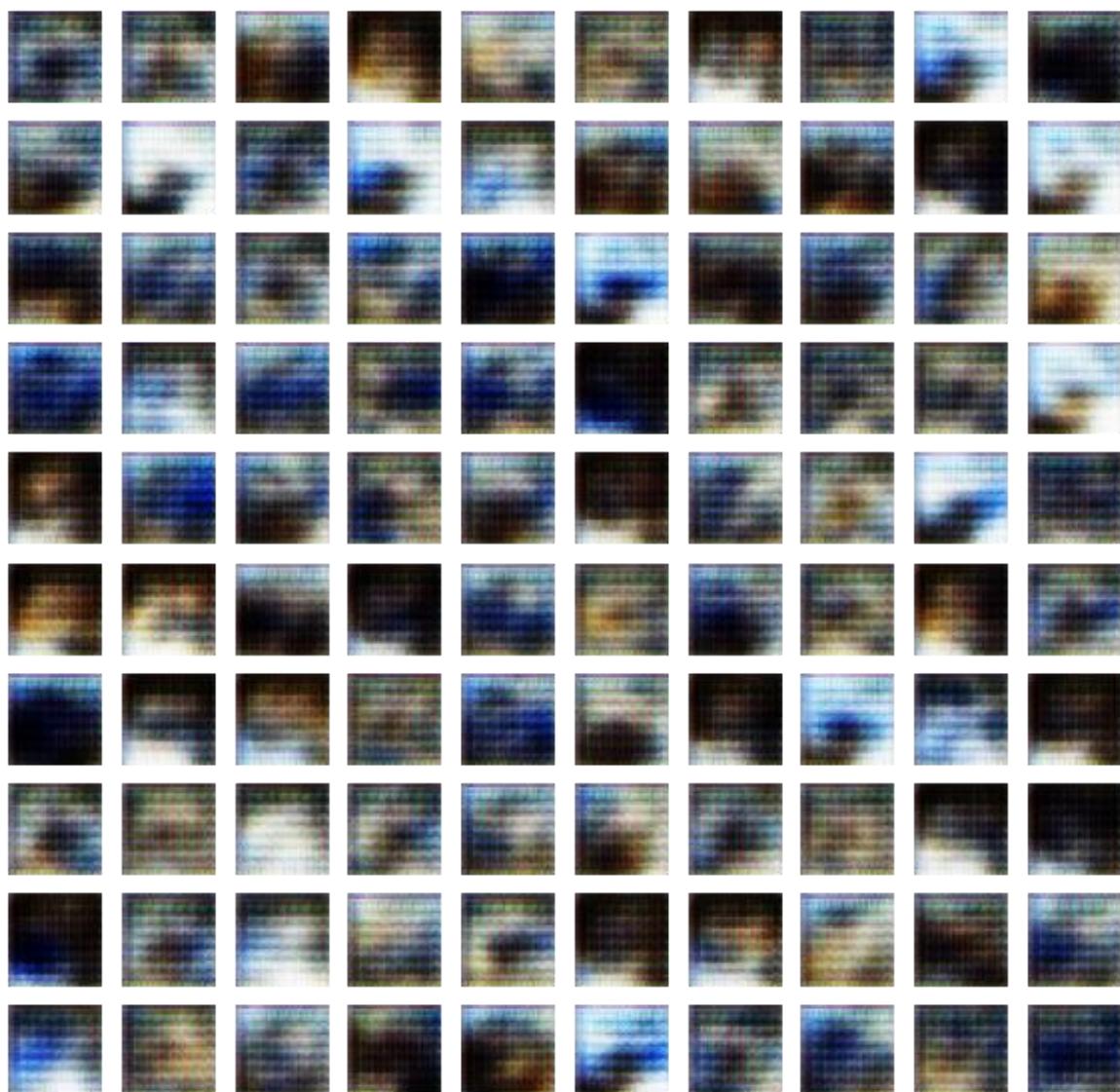


شکل 3-16: شبکه Generator در AC-GAN

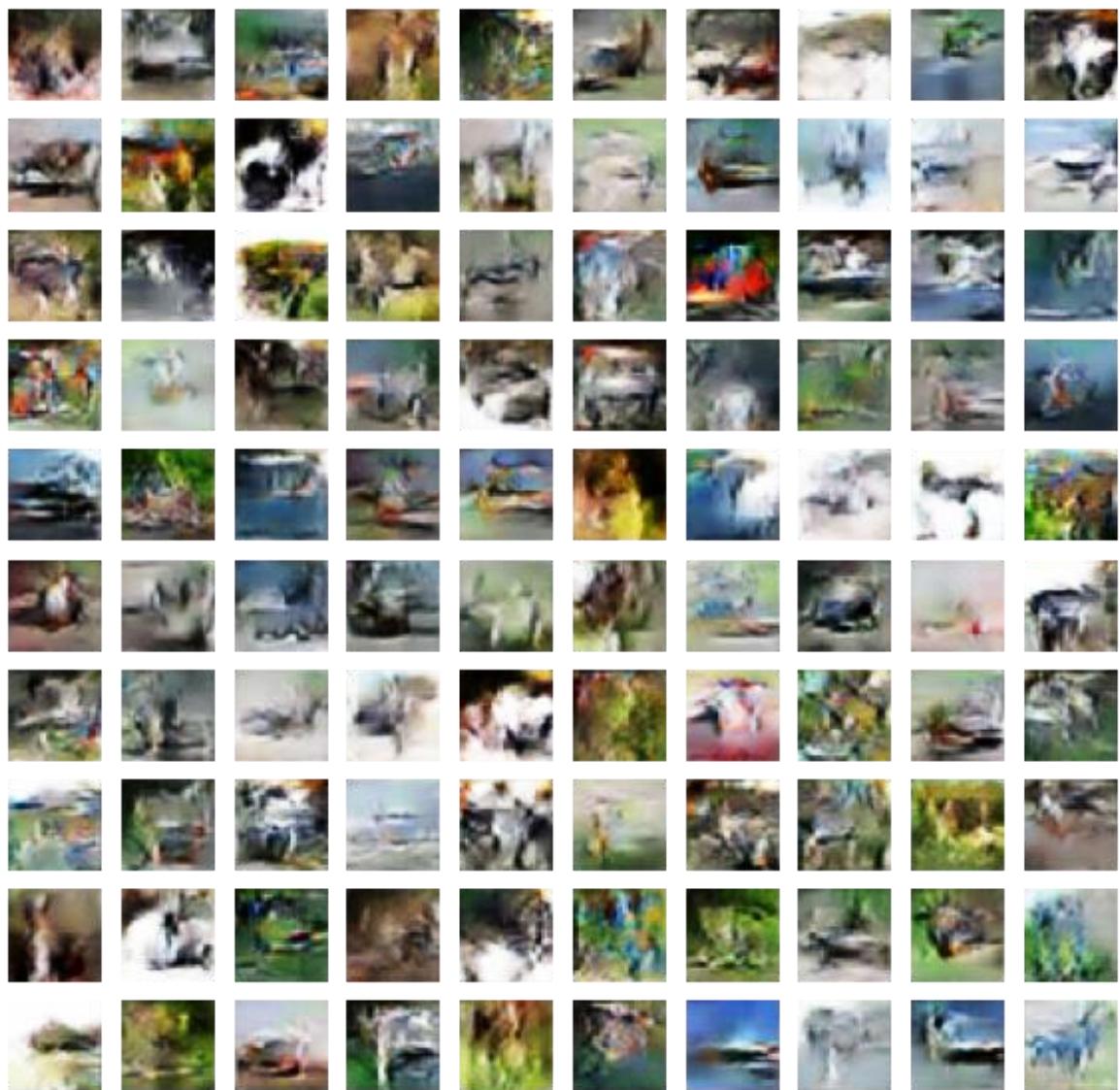


شکل ۳-۱۷ ترکیب دو شبکه generator و discriminator به منظور آموزش مستقل

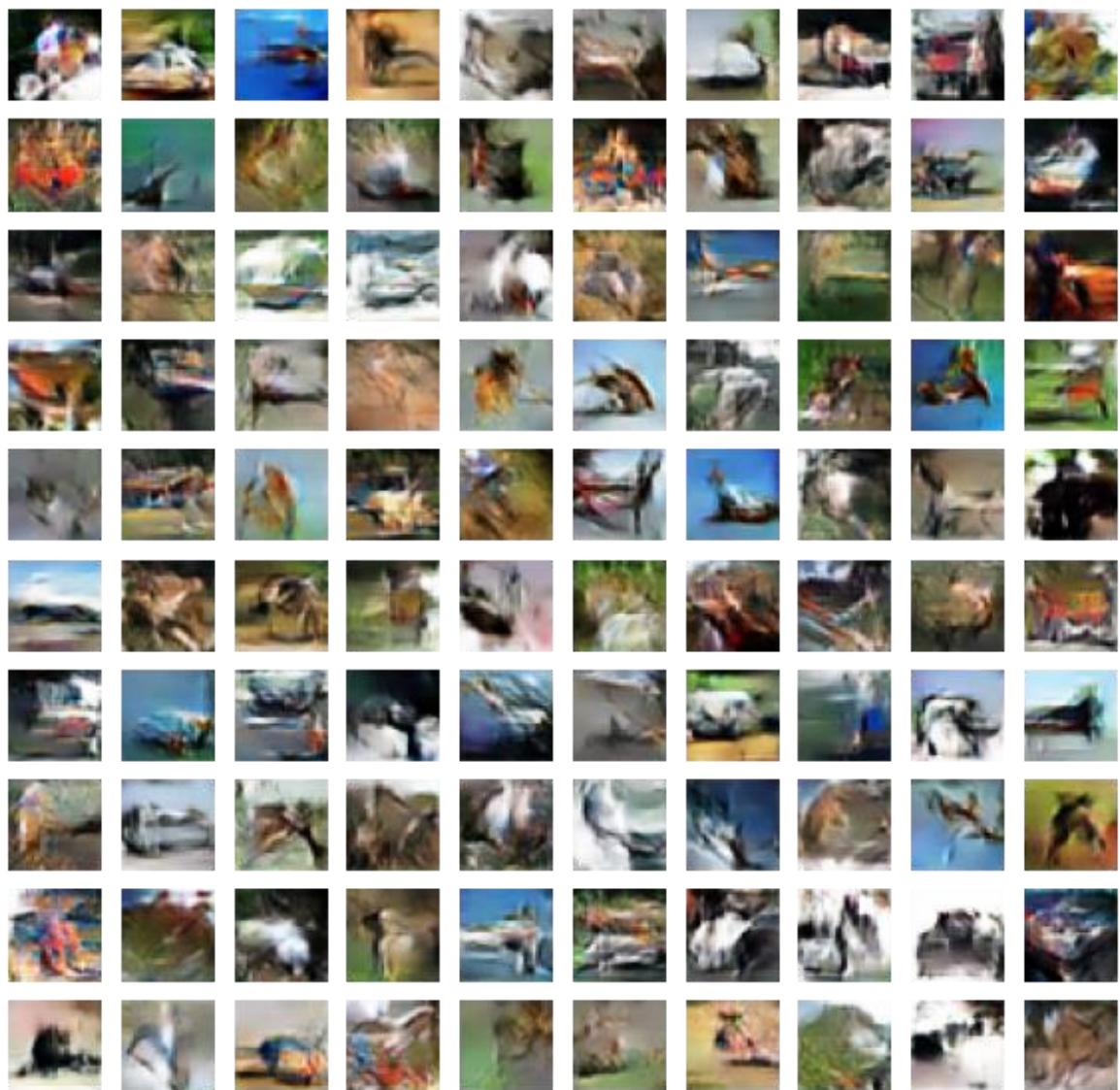
در طی آموزش شبکه در 101 ایپاک با طول بچ 64 ، در ایپاکهای 1 ، 21 ، 41 ، 81 و 101 تعداد 100 تصویر را توسط generator تولید می کنیم تا روند بهبود شبکه را مشاهده نماییم. این تصاویر تولید شده به صورت جدول های 10×10 که با استفاده از subplot رسم کرده ایم ، به ترتیب در شکلهاي 3-21 ، 3-19 ، 18 ، 3-20 ، 3-22 قابل مشاهده هستند.



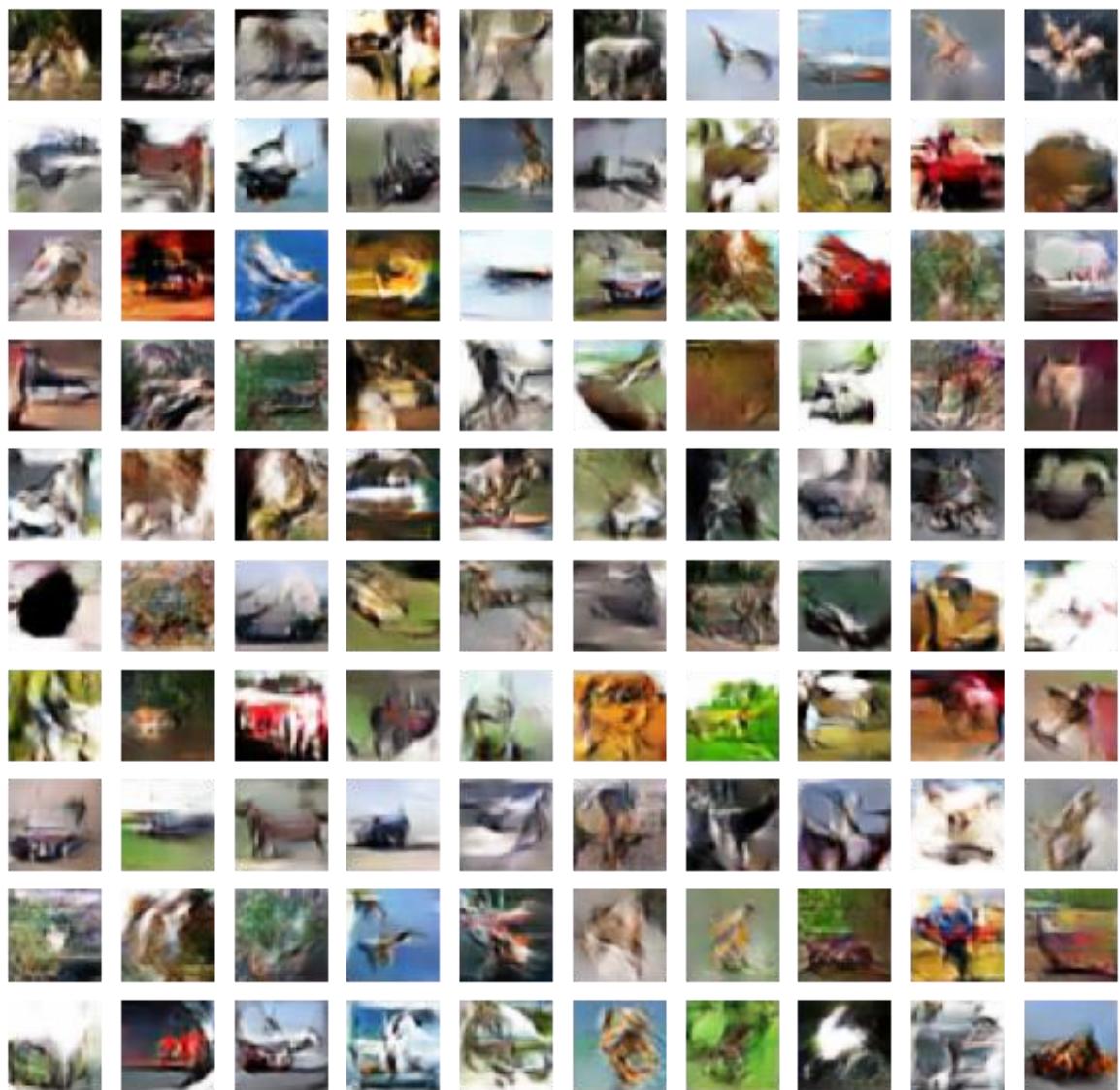
شکل 3-18: تصاویر تولید شده پس از 1 ایپاک



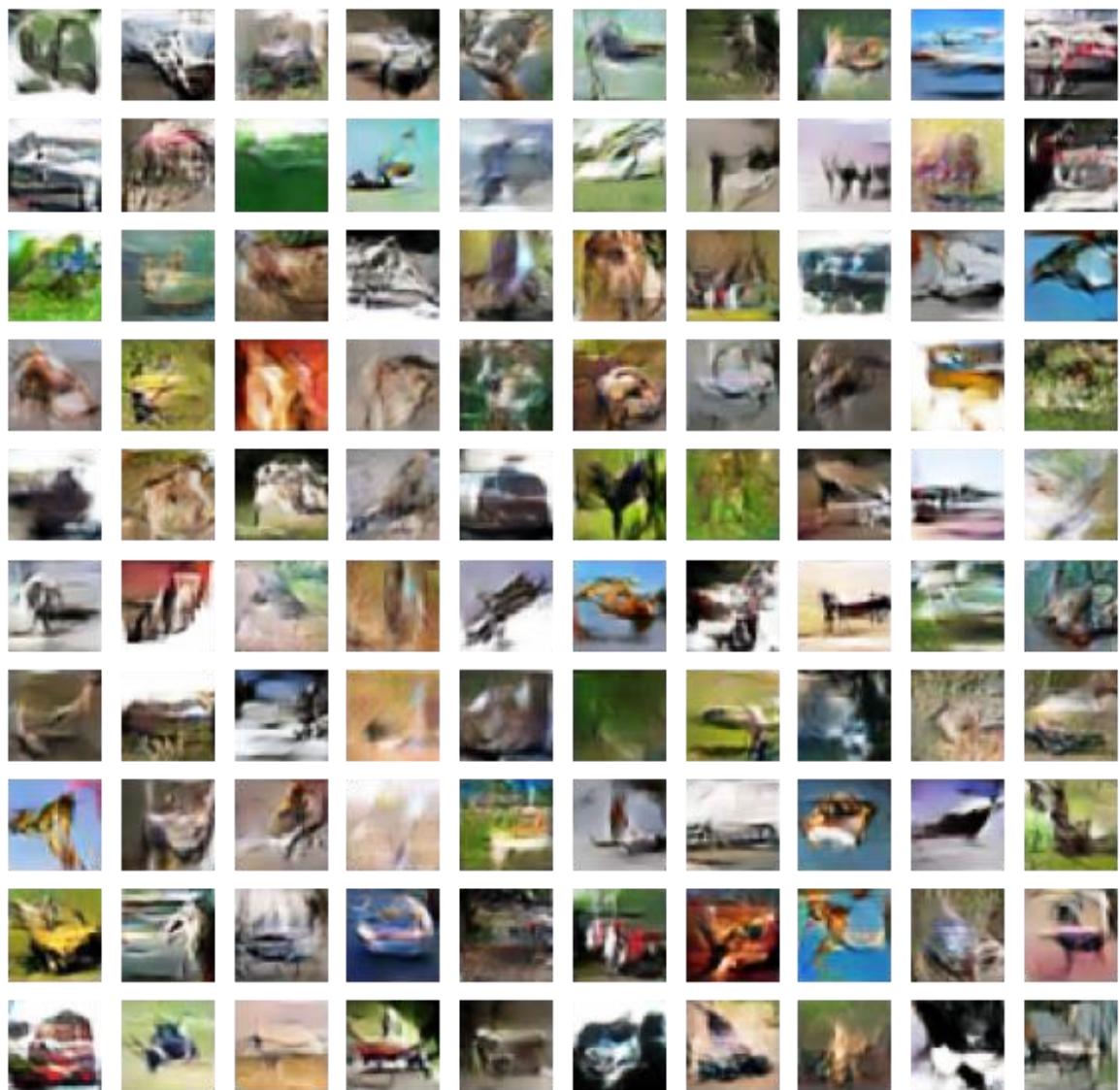
شکل ۱۹-۳: تصاویر تولید شده پس از ۲۱ ایپاک



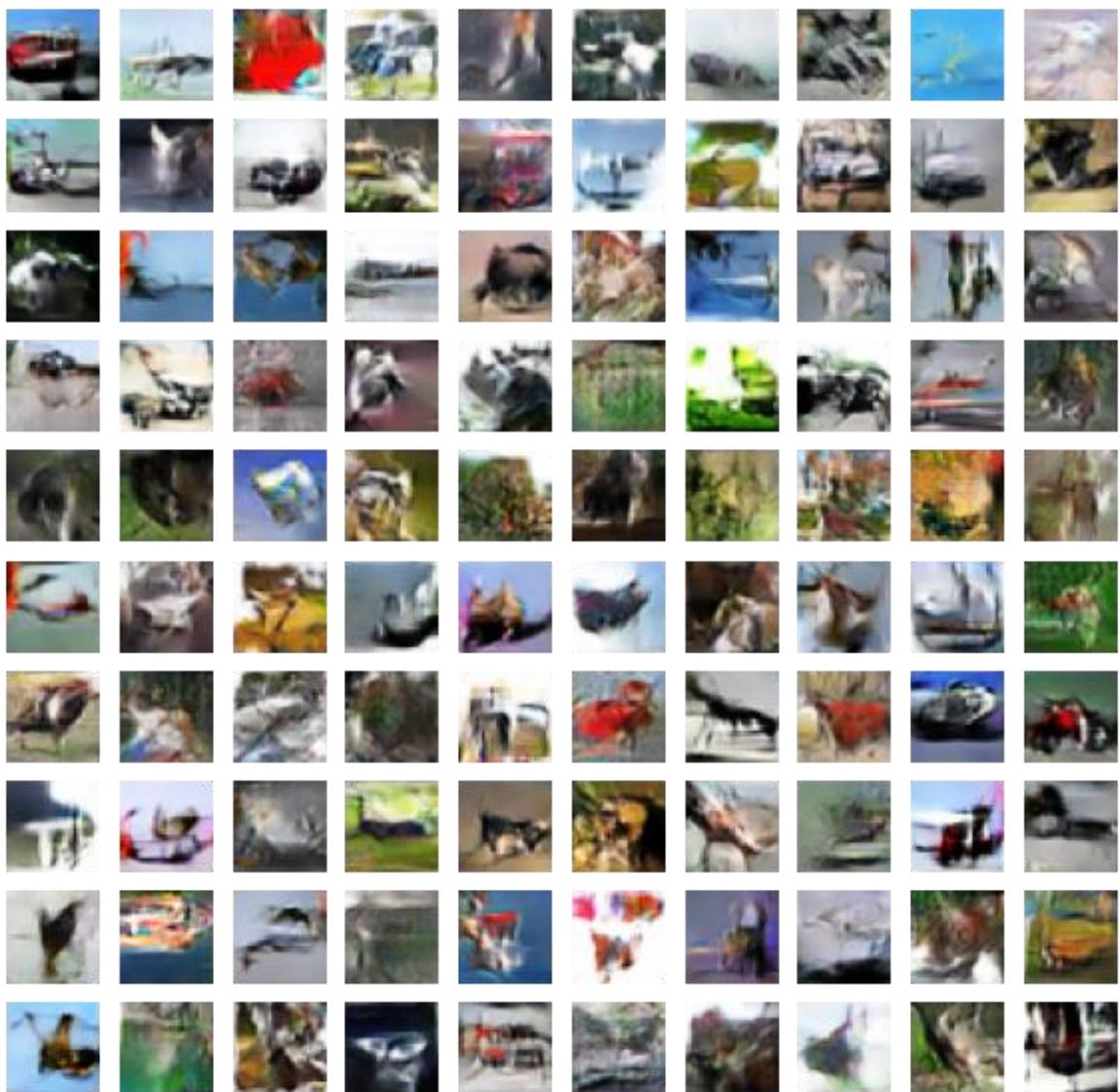
شکل 20-3: تصاویر تولید شده بعد از 41 ایپاک



شکل 3-21: تصاویر تولید شده پس از 61 ایپاک



شکل 3-22: تصاویر تولید شده پس از 81 ایپاک

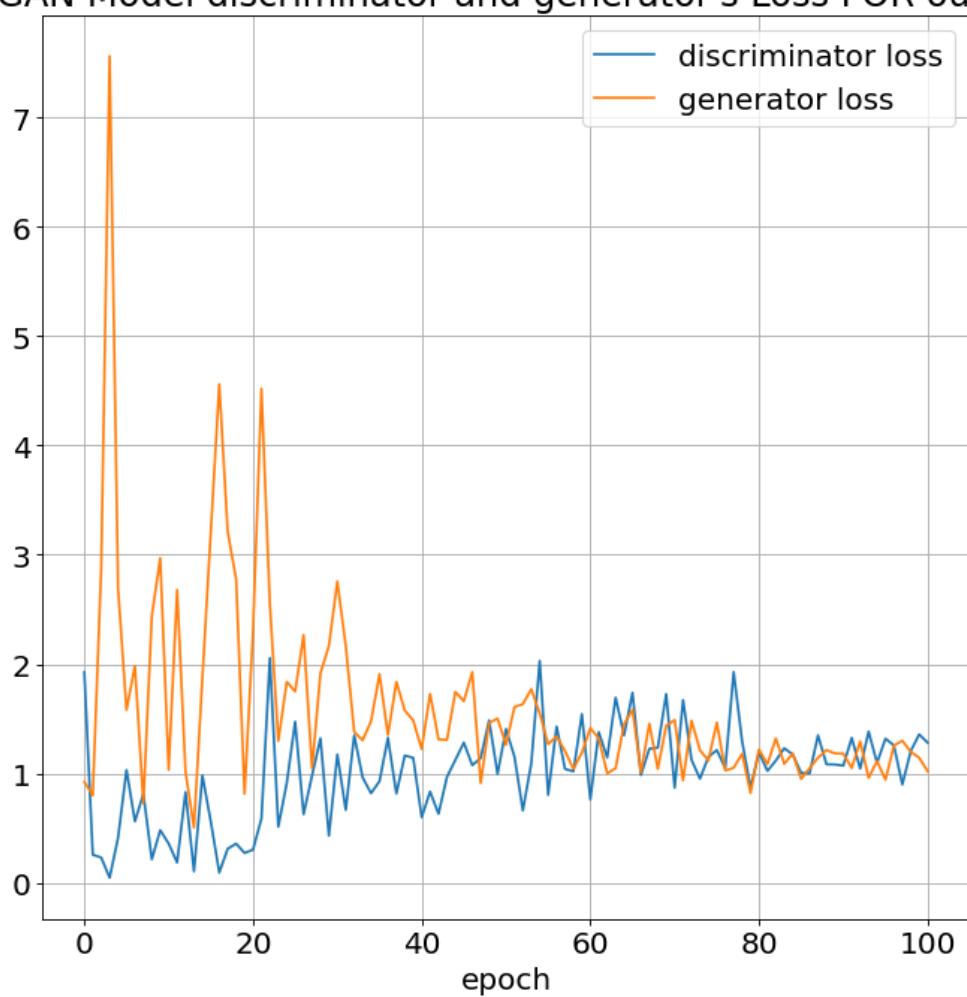


شکل 3-23: تصاویر تولید شده پس از 101 ایپاک

همان طور که در شکل‌های بالا مشاهده می‌شود با گذشت epoch ها ، کیفیت تصاویر تولیدی بهتر شده است و این نشان دهنده آن است که شبکه به خوبی کار کرده است.

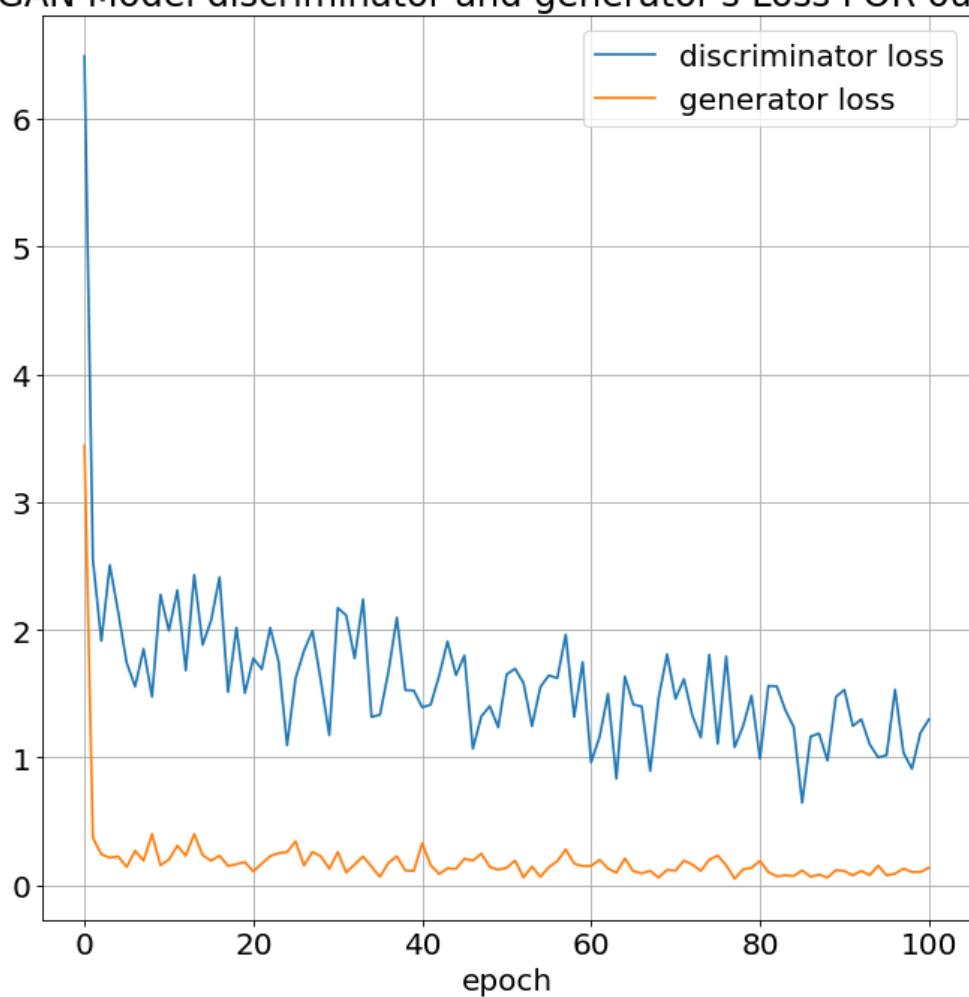
همچنین در شکل 3-24 و 3-25 به ترتیب نمودارهای loss دو شبکه generator و discriminator برای خروجی اول (میزان واقعی بودن تصویر) و خروجی دوم (بردار احتمال لیبل تصویر ورودی) نشان داده شده است.

AC-GAN Model discriminator and generator's Loss FOR output 1



شكل 3-24: نمودار loss خروجی اول

AC-GAN Model discriminator and generator's Loss FOR output 2



شکل 3-25: نمودار loss خروجی دوم

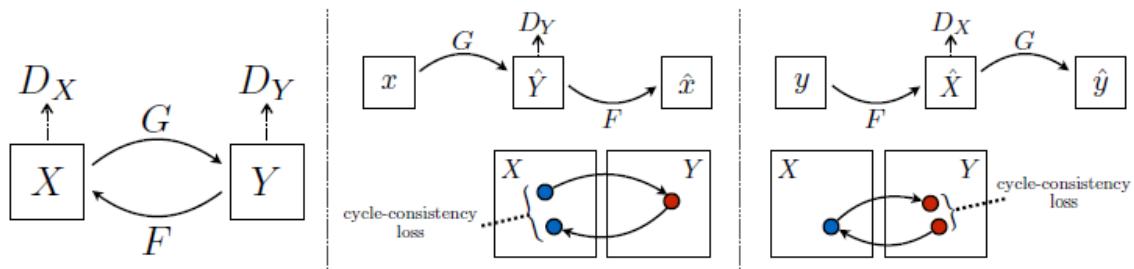
با توجه به دو شکل 3-24 و 3-25 و نوسانات نمودارها و اینکه در شکل 3-24 discriminator کمتر است و برعکس در برخی ایپاکها بیشتر از generator است نشان دهنده تنابع بین generator و discriminator در طی آموزش است.

سوال 4 – شبکه های Cycle GAN

(الف)

ترجمه تصویر به تصویر یکی از مسائل گرافیکی است که هدف آموزش Mapping بین تصویر ورودی و تصویر خروجی است.

طبق مقاله مدلی که در این سوال پیاده سازی خواهیم کرد شامل دو نوع mapping است. یک mapping به صورت $F: Y \rightarrow X$ و دیگری به صورت $G: X \rightarrow Y$ می باشد که discriminator های آنها به ترتیب D_Y و D_X هستند. G را طوری تشویق می کند که X را به تصاویری تبدیل کند که مشابه دامنه Y باشد و برعکس. ساختار کلی این شبکه در شکل 4-1 قابل مشاهده است:



شکل 4-1: ساختار کلی شبکه Cycle GAN

لازم به ذکر است همانطور که در شکل 4-1 مشاهده می شود از ترکیب دوتابع خطای Cycle Consistency و Adversarial Loss استفاده می کنیم که در ادامه توضیح می دهیم.

که در GAN های معمولی نیز استفاده می شود یک mapping میان G و F یاد می گیرد که خروجی هایی با توزیع یکسان در دامنه های X و Y تولید می کند. اما خوب با این کار شبکه ممکن است تصاویر ورودی را به هر جایگشت رندومی از تصاویر در target domain مپ کند به طوری که توزیع خروجی با توزیع ورودی برابر باشد. بنابرین این تابع خطای نمی تواند به تنها ی تضمین کند که ورودی هایمان را به خروجی مطلوب مپ می کند. برای رفع این مشکل باید mapping function های learn شده مطابق شکل وسط 4-1 باشند. به عبارتی به ازای هر تصویر ورودی x از دامنه X باید بتوان $G(x)$ را در حلقه image translation به ورودی x بازگرداند:

$$x \rightarrow G(x) \rightarrow F(G(x)) \approx x$$

که forward cycle consistency نامیده می شود. به طور مشابه که در شکل سمت راست 4-1 نشان داده شده است، برای تصویر y از دامنه Y ، F و G باید در backward cycle consistency صدق کنند:

$$y \rightarrow F(y) \rightarrow G(F(y)) \approx y$$

این تابع خطای طبق مقاله به صورت زیر تعریف می شود:

$$\begin{aligned}\mathcal{L}_{\text{cyc}}(G, F) = & \mathbb{E}_{x \sim p_{\text{data}}(x)} [\|F(G(x)) - x\|_1] \\ & + \mathbb{E}_{y \sim p_{\text{data}}(y)} [\|G(F(y)) - y\|_1].\end{aligned}$$

همچنین خطای Adversial GAN های معمولی نیز استفاده می شود به صورت زیر است:

$$\begin{aligned}\mathcal{L}_{\text{GAN}}(G, D_Y, X, Y) = & \mathbb{E}_{y \sim p_{\text{data}}(y)} [\log D_Y(y)] \\ & + \mathbb{E}_{x \sim p_{\text{data}}(x)} [\log(1 - D_Y(G(x)))]\end{aligned}$$

بنابرین تابع خطای کلی را می توان به صورت زیر در نظر گرفت:

$$\begin{aligned}\mathcal{L}(G, F, D_X, D_Y) = & \mathcal{L}_{\text{GAN}}(G, D_Y, X, Y) \\ & + \mathcal{L}_{\text{GAN}}(F, D_X, Y, X) \\ & + \lambda \mathcal{L}_{\text{cyc}}(G, F),\end{aligned}$$

که در آن λ ضریبی است که مهم بودن تابع خطای cycle consistency را کنترل می کند. هدف کلی این شبکه این است که مسئله زیر را حل کند:

$$G^*, F^* = \arg \min_{G, F} \max_{D_X, D_Y} \mathcal{L}(G, F, D_X, D_Y).$$

توجه داشته باشید این شبکه را می توان به صورت دو اتوانکدر $G \circ F: Y \rightarrow Y$ و $F \circ G: X \rightarrow X$ مدل کرد.

(ب)

این شبکه را به وسیله Transfer learning که جزئیات آن در فایل نوت بوک توضیح داده شده است پیاده سازی می کنیم. در نهایت شبکه را در 40 ایپاک آموزش می دهیم و در طی آموزش تصاویر تولیدی را برای چند ایپاک برای مشاهده روند بهبود شبکه رسم می کنیم که در شکل های زیر قابل مشاهده هستند:



شکل ۴-۲: ایپاک ۱



شکل ۴-۳: ایپاک ۸



شکل 4-4: ایپاک 16



شکل 4-5: ایپاک 24



شکل ۶-۴: ایپاک 32



شکل ۶-۷: ایپاک 40 (ایپاک آخر)

پس از آموزش شبکه می توانیم از generator برای تبدیل تصاویر استفاده کنیم. به طور مثال در شکل های زیر تعداد 5 تصویر را تبدیل کرده ایم که قابل مشاهده هستند:

