به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

درس شبکه‌های عصبی و یادگیری عمیق

تمرین ششم

نام و نام خانوادگی	محمد ناصری – مریم عباس‌زاده
شماره دانشجویی	810100406 - 810100486
تاریخ ارسال گزارش	۱۴۰۱.۱۰.۳۰

فهرست

پاسخ 1. شبکه‌های عصبی مول تخاصمی کانولوشنال عمیق 4

۱-۱. 4

۱-۲ پایدارسازی شبکه 10

پاسخ ۲ – شبکه متخاصم مولد طبقه بند کمکی و شبکه Wassertein 13

۲-۱. شبکه متخاصم مولد طبقه بند کمکی 13

۲-۲ شبکه Wassertein 17

شکل‌ها

شکل 1 - مجموعه داده 6
شکل 2 - معماری کلی dcgan 6
شکل 4 - معماری مولد در noraml dcgan 7
شکل 3 - ساخت معماری مولد normal dcgan 7
شکل 5 - معماری تفکیک کننده در normal dcgan 8
شکل 6 - ساخت معماری تفکیک کننده در normal dcgan 8
شکل 7 - خروجی‌های آموزش normal dcgan 9
شکل 8 - نمودار Loss برای normal dcgan 9
شکل 9 - کد هموارسازی برچسب 10
شکل 10 - کد اضافه کردن نویز به برچسب ها 11
شکل 11 - خروجی‌های آموزش stable dcgan 12
شکل 12 - نمودار loss برای stable dcgan 12
شکل 13 - معماری مولد acgan 14
شکل 14 - معماری تفکیک کننده acgan 15
شکل 15 - نمونه خروجی acgan 16
شکل 16 - نمودار loss برای acgan 16
شکل 17 - معماری مولد wgan 19
شکل 18 - معماری تفکیک کننده wgan 20
شکل 19 - خروجی wgan 21
شکل 20 - نمودار loss برای wgan 21

جدول‌ها

No table of figures entries found.

پاسخ ۱. شبکه‌های عصبی مول تخاصمی کانولوشنال عمیق

۱-۱. پیاده سازی مولد تصویر با استفاده از شبکه های مولد تخاصمی کانولوشنال عمیق

GAN چیست؟

GANیک شبکه متخاصم مولد (GAN) نوعی معماری شبکه عصبی است که برای تولید داده های جدید مشابه یک مجموعه داده معین استفاده می شود. از دو بخش اصلی تشکیل شده است: یک مولد و یک تشخیص دهنده. مولد برای ایجاد نمونه های داده جدید آموزش دیده است، در حالی که متمایز کننده برای تشخیص نمونه های تولید شده از نمونه های واقعی در مجموعه داده داده شده آموزش دیده است.

مولد و تمایز به طور همزمان به شیوه ای خصمانه آموزش می بینند، به طوری که مولد سعی می کند نمونه هایی را تولید کند که بتواند متمایزکننده را فریب دهد و تمایزکننده سعی می کند به درستی تشخیص دهد که کدام نمونه واقعی است و کدام نمونه تولید شده است. این روند تا زمانی ادامه می یابد که ژنراتور نمونه هایی را تولید کند که از نمونه های واقعی قابل تشخیص نیستند و متمایز کننده دیگر نمی تواند آنها را از هم تشخیص دهد.

GAN ها برای طیف گسترده ای از وظایف، مانند ترکیب تصویر و ویدئو، تبدیل متن به گفتار، و ترجمه زبان استفاده شده اند. آنها برای تولید تصاویر، فیلم ها، صدا و متن جدید استفاده شده اند. داده های تولید شده می توانند کیفیت بالایی داشته باشند و می توانند در بسیاری از برنامه ها مانند ایجاد شبیه سازی های واقعی، بهبود وضوح تصویر و تشخیص تقلب استفاده شوند.

DCGAN یا Deep Convolutional Generative Adversarial Network، نوعی از GAN است که برای تولید تصاویر جدید استفاده می شود. "DC" در نام به استفاده از شبکه های عصبی کانولوشن عمیق اشاره دارد که شبکه های عصبی هستند که به ویژه برای داده های تصویری مناسب هستند. در یک DCGAN، شبکه مولد تصاویر جدید ایجاد می کند، در حالی که شبکه تشخیص دهنده تلاش می کند تصاویر تولید

شده را از تصاویر واقعی متمایز کند. این دو شبکه با هم به شیوه ای متخاصم آموزش داده می شوند، با هدف تولید کننده تصاویری که از تصاویر واقعی قابل تشخیص نیستند.

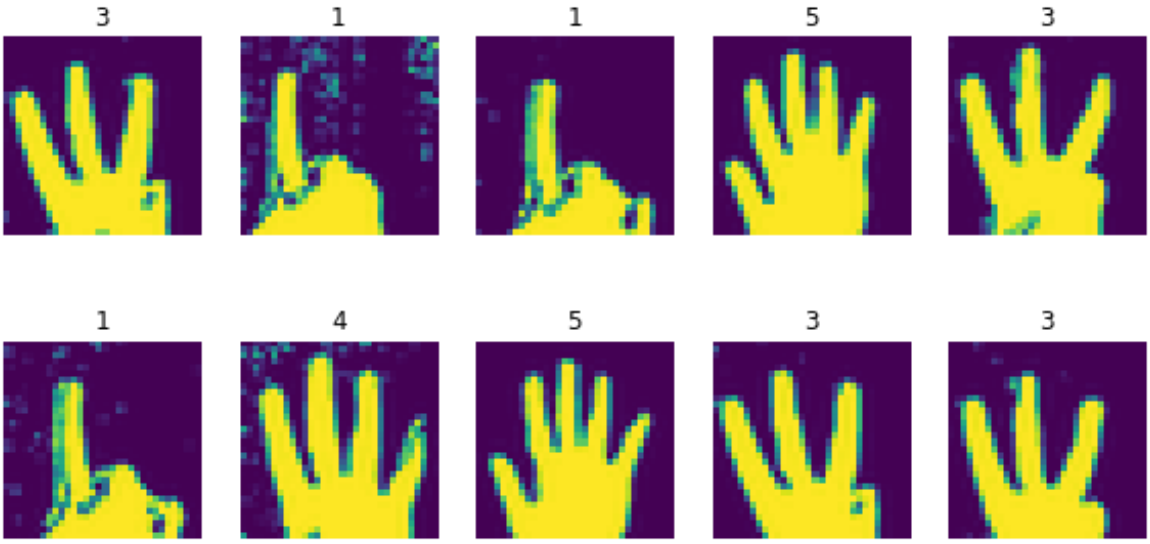
هنگامی که DCGAN بر روی تصاویر 32*32 آموزش داده می شود، در حال یادگیری تولید تصاویر جدید با همان اندازه و ابعاد است. شبکه مولد یاد می گیرد که با نمونه برداری از تصاویر با وضوح پایین، تصاویر جدید ایجاد کند، در حالی که شبکه تشخیص دهنده برای تشخیص الگوها در تصاویر 32*32 آموزش دیده است. این روند تا زمانی ادامه می یابد که ژنراتور تصاویری با کیفیت بالا تولید کند که به سختی می توان آنها را از تصاویر واقعی متمایز کرد.

مراحل آموزش یک شبکه متخاصم مولد عمیق (DCGAN) به شرح زیر است:

- مجموعه داده ای از تصاویر را جمع آوری و پیش پردازش کنید. اندازه تصاویر باید تغییر داده شود و به اندازه ورودی مورد نظر برای DCGAN (در این مورد، 32*32) نرمال شود.
- شبکه های مولد و تفکیک کننده را تعریف کنید. شبکه ژنراتور معمولاً از چندین لایه از لایه های کانولوشنال جابجا شده تشکیل شده است که برای نمونه برداری از یک بردار نویز ورودی به اندازه خروجی مورد نظر استفاده می شود. شبکه تفکیک کننده معمولاً از چندین لایه از لایه های کانولوشن تشکیل شده است که برای طبقه بندی تصاویر به عنوان واقعی یا جعلی استفاده می شود.
- شبکه های مولد و تفکیک کننده را با هم آموزش دهید. مولد تصاویر جدیدی تولید می کند، در حالی که متمایز کننده سعی می کند آنها را به عنوان واقعی یا جعلی طبقه بندی کند. مولد برای تولید تصاویری آموزش دیده است که می تواند متمایز کننده را فریب دهد، در حالی که تشخیص دهنده آموزش دیده است که تصاویر تولید شده را به درستی شناسایی کند.
- مرحله 3 را برای چند دوره تکرار کنید. مولد و متمایز کننده همچنان به پیشرفت خود ادامه می دهند زیرا یاد می گیرند که با یکدیگر همکاری بهتری داشته باشند. این روند تا زمانی ادامه می یابد که ژنراتور تصاویری با کیفیت بالا تولید کند که به سختی می توان آنها را از تصاویر واقعی متمایز کرد.

- از مولد برای تولید تصاویر جدید استفاده کنید. پس از اتمام آموزش، می توان از مولد برای ایجاد تصاویر جدید با ارائه یک بردار نویز به عنوان ورودی استفاده کرد.

در این پیاده‌سازی برای مدل مجموعه داده از تصاویر انگشتان دست نشان‌دهنده اعداد استفاده شده است.



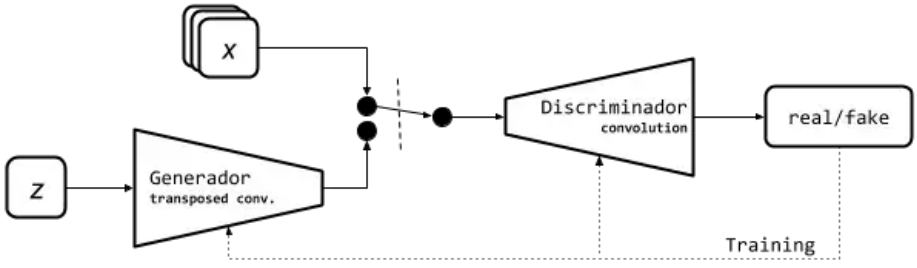
شکل 1 - مجموعه داده

این مجموعه شامل ۱۰۰۰ تصویر ۳۲*۳۲ است که هر تصویر در ۳ کانال رنگی تعریف شده ولی از آنجا که نهایتا مقادیر ۳ کانال برابر هستند و به عبارتی grayscale هستند برای کاهش محاسبات به جای ۳ کانال از یک کانال برای تصاویر استفاده شده است.

از اقدامات انجام شده روی دیتاست می‌توان به نرمال‌سازی (بین ۱ و -۱) مقادیر و همچنین expand کردن ابعاد برای آماده‌سازی شبکه اشاره کرد.

```
# Rescale -1 to 1
X_train = X_train / 127.5 - 1.
X_train = np.expand_dims(X_train, axis=3)
```

معماری کلی dcgan به صورت زیر است:



شکل 2 - معماری کلی **dcgan**

معماری استفاده شده برای مدل مولد به شرح زیر میباشد:

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_1 (Dense)             (None, 16384)             1638400

 batch_normalization (BatchN  (None, 16384)            65536
 ormalization)

 leaky_re_lu_2 (LeakyReLU)   (None, 16384)             0

 reshape (Reshape)           (None, 8, 8, 256)         0

 conv2d_transpose (Conv2DTra  (None, 8, 8, 128)        819200
 nspose)

 batch_normalization_1 (Batc  (None, 8, 8, 128)        512
 hNormalization)

 leaky_re_lu_3 (LeakyReLU)   (None, 8, 8, 128)         0

 conv2d_transpose_1 (Conv2DT  (None, 16, 16, 64)       204800
 ranspose)

 batch_normalization_2 (Batc  (None, 16, 16, 64)       256
 hNormalization)

 leaky_re_lu_4 (LeakyReLU)   (None, 16, 16, 64)        0

 conv2d_transpose_2 (Conv2DT  (None, 32, 32, 1)        1600
 ranspose)

=================================================================
Total params: 2,730,304
Trainable params: 2,697,152
Non-trainable params: 33,152
_________________________________________________________________
```

شکل 4 - معماری مولد در **noraml dcgan**

```python
model = tf.keras.Sequential()
model.add(layers.Dense(8*8*256, use_bias=False, input_shape=(100,)))
model.add(layers.BatchNormalization())
model.add(layers.LeakyReLU())

model.add(layers.Reshape((8, 8, 256)))
assert model.output_shape == (None, 8, 8, 256)  # Note: None is the batch size

model.add(layers.Conv2DTranspose(128, (5, 5), strides=(1, 1), padding='same', use_bias=False))
assert model.output_shape == (None, 8, 8, 128)
model.add(layers.BatchNormalization())
model.add(layers.LeakyReLU())

model.add(layers.Conv2DTranspose(64, (5, 5), strides=(2, 2), padding='same', use_bias=False))
assert model.output_shape == (None, 16, 16, 64)
model.add(layers.BatchNormalization())
model.add(layers.LeakyReLU())

model.add(layers.Conv2DTranspose(1, (5, 5), strides=(2, 2), padding='same', use_bias=False, activation='tanh'))
assert model.output_shape == (None, 32, 32, 1)
model.summary()
```

شکل 3 - ساخت معماری مولد **normal dcgan**

معماری استفاده شده برای discriminatorبه شرح زیر است:

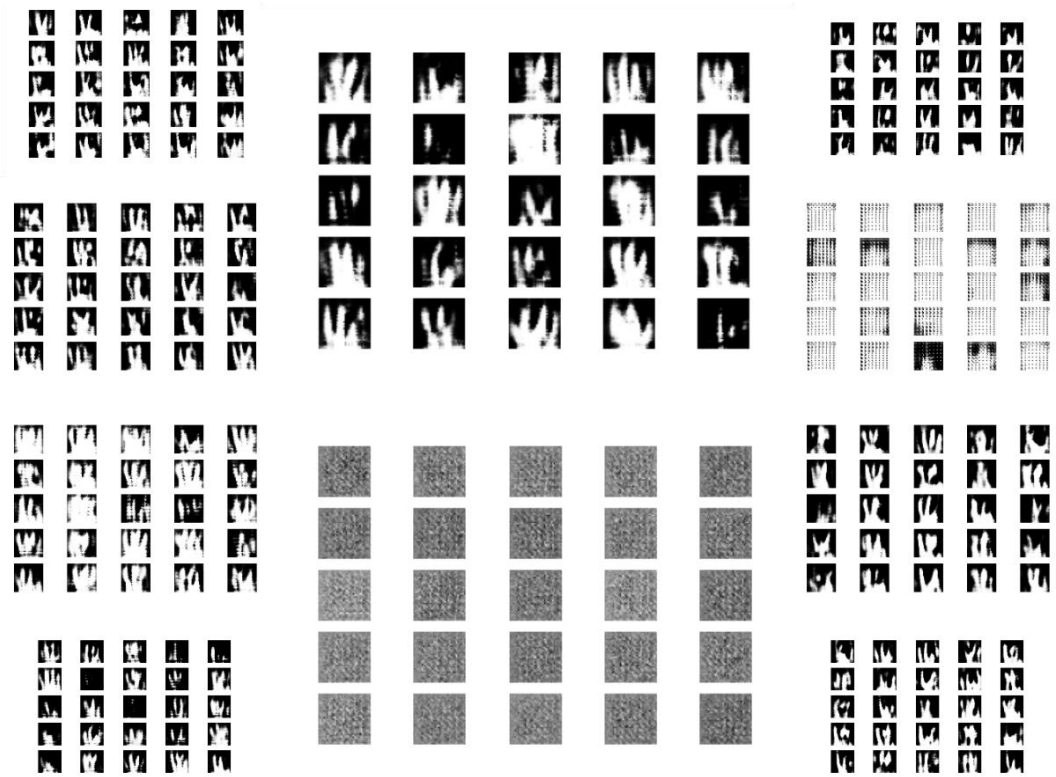
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 16, 16, 64)        1664

 leaky_re_lu (LeakyReLU)     (None, 16, 16, 64)        0

 dropout (Dropout)           (None, 16, 16, 64)        0

 conv2d_1 (Conv2D)           (None, 8, 8, 128)         204928

 leaky_re_lu_1 (LeakyReLU)   (None, 8, 8, 128)         0

 dropout_1 (Dropout)         (None, 8, 8, 128)         0

 flatten (Flatten)           (None, 8192)              0

 dense (Dense)               (None, 1)                 8193

=================================================================
Total params: 214,785
Trainable params: 214,785
Non-trainable params: 0
_________________________________________________________________
```

شکل 5 - معماری تفکیک کننده در **normal dcgan**

```
model = tf.keras.Sequential()
model.add(layers.Conv2D(64, (5, 5), strides=(2, 2), padding='same',
                        input_shape=[32, 32, 1]))
model.add(layers.LeakyReLU())
model.add(layers.Dropout(0.3))

model.add(layers.Conv2D(128, (5, 5), strides=(2, 2), padding='same'))
model.add(layers.LeakyReLU())
model.add(layers.Dropout(0.3))

model.add(layers.Flatten())
model.add(layers.Dense(1))
model.summary()
```

شکل 6 - ساخت معماری تفکیک کننده در **normal dcgan**

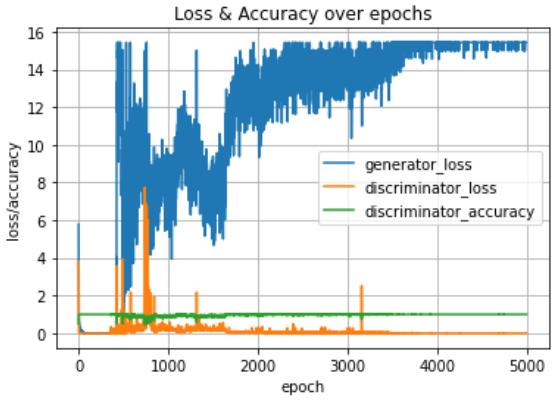
برای تابع lossدر این پیاده‌سازی از binary cross entropy استفاده کردیم. پس از آموزش روی دادگان موجود خروجی های زیر حاصل شدند.



شکل 7 - خروجی‌های آموزش **normal dcgan**

دو تصویر میانی مربوط به مرحله اول و آخر آموزش هستند. همانطور که مشاهده میشود شبکه به درک قابل توجهی از ساختار تصاویر رسیده و قادر بوده تصاویر تقریبا مشابهی ایجاد نماید.

در نهایت نمودار loss مدل مربوطه بعد از ۵۰۰۰ ایپاک به صورت زیر خواهد بود:



شکل 8 - نمودار **Loss** برای **normal dcgan**

۱-۲ پایدارسازی شبکه

در مرحله بعدی برای پایدارسازی بیشتر شبکه از تکنیک‌های label smoothing و Noisy labels استفاده شده است.

مدل‌های GAN در مقایسه با سایر شبکه‌های عمیق در حوزه‌های زیر آسیب‌های بدی خواهند دید.

- عدم همگرایی: مدل ها همگرا نمی شوند و بدتر از آن ناپایدار می شوند.
- فروپاشی حالت: ژنراتور حالت های محدودی تولید می کند و
- تمرین آهسته: گرادیان برای آموزش مولد ناپدید شد.

به عنوان بخشی از سری GAN، این مقاله به روش هایی برای بهبود GAN می پردازد. به خصوص،

- تابع هزینه را برای یک هدف بهینه سازی بهتر تغییر میدهد.
- برای اعمال محدودیت ها، جریمه های اضافی را به تابع هزینه اضافه میکند.
- روش های بهتر برای بهینه سازی مدل
- برچسب های نویز را اضافه میکند.

شبکه های عمیق ممکن است از اعتماد بیش از حد رنج ببرند. به عنوان مثال، از ویژگی های بسیار کمی برای طبقه بندی یک شی استفاده می کند. برای کاهش مشکل، یادگیری عمیق از مقررات و ترک تحصیل برای جلوگیری از اعتماد بیش از حد استفاده می کند.

در GAN، اگر تمایزگر به مجموعه کوچکی از ویژگی‌ها برای تشخیص تصاویر واقعی وابسته باشد، مولد ممکن است این ویژگی‌ها را فقط برای سوء استفاده از تمایزکننده تولید کند. بهینه سازی ممکن است بیش از حد حریص باشد و هیچ منفعتی در دراز مدت نداشته باشد. در GAN، اعتماد به نفس زیاد به شدت آسیب می زند. برای جلوگیری از مشکل، زمانی که پیش‌بینی هر تصویر واقعی از 0.9 فراتر رود، متمایزکننده را جریمه می‌کنیم (D(تصویر واقعی)> 0.9). این کار با تنظیم مقدار برچسب هدف ما به جای 1.0 روی 0.9 انجام می شود. کد:

```
# example of smoothing class=0 to [0.0, 0.3]
def smooth_negative_labels(y):
 return y + random(y.shape) * 0.3

 # example of smoothing class=1 to [0.7, 1.2]
def smooth_positive_labels(y):
 return y - 0.3 + (random(y.shape) * 0.5)
```

شکل 9 - کد هموارسازی برچسب

از طرفی برچسب‌هایی که هنگام آموزش مدل تمایز استفاده می‌شوند همیشه صحیح هستند.

این بدان معناست که تصاویر جعلی همیشه با کلاس 0 و تصاویر واقعی همیشه با کلاس 1 برچسب گذاری می شوند.

توصیه می‌شود در این برچسب‌ها خطاهایی وارد شود که برخی از تصاویر جعلی واقعی و برخی از تصاویر واقعی جعلی هستند.

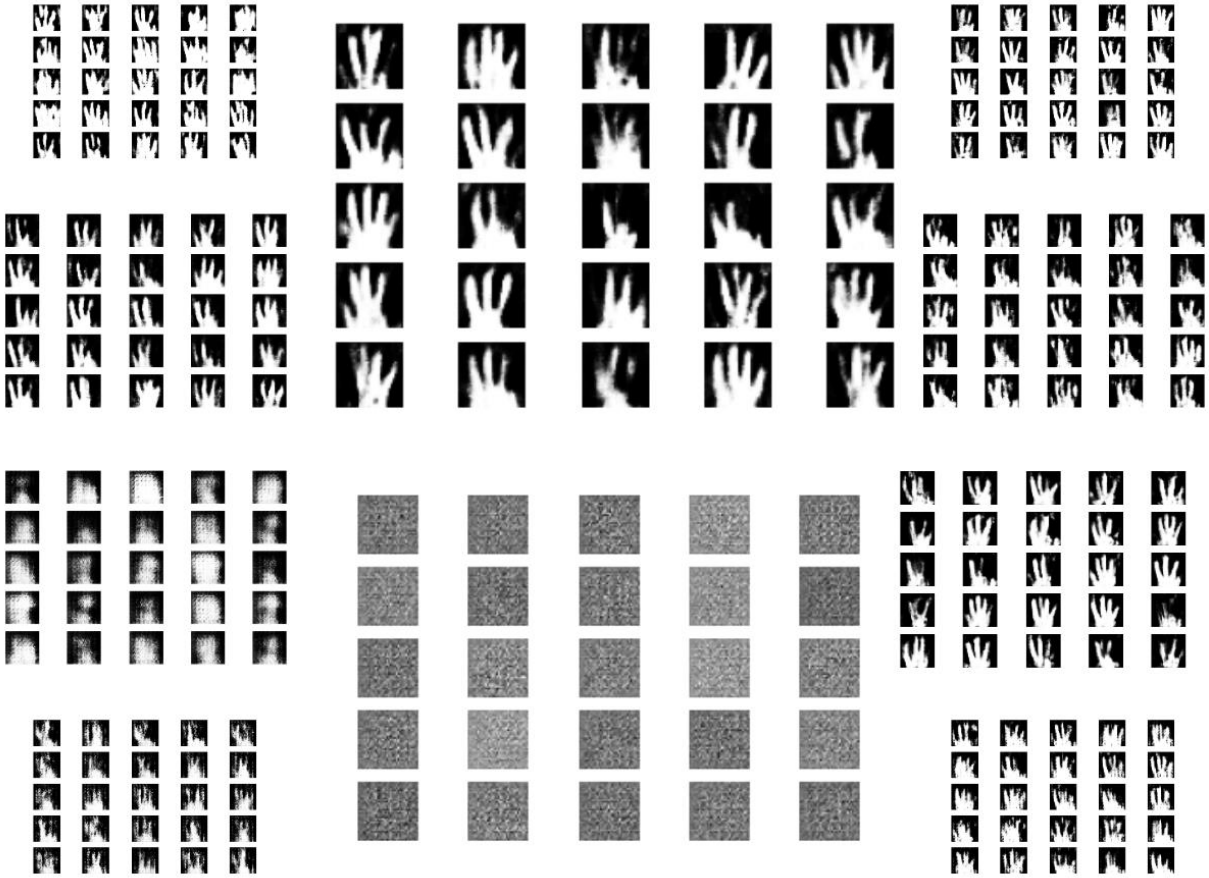
اگر از دسته‌های جداگانه برای به‌روزرسانی تشخیص‌دهنده تصاویر واقعی و جعلی استفاده می‌کنید، این ممکن است به معنای افزودن تصادفی برخی از تصاویر جعلی به دسته‌ای از تصاویر واقعی یا افزودن تصادفی برخی از تصاویر واقعی به دسته‌ای از تصاویر جعلی باشد.

اگر در حال به روز رسانی متمایز کننده با مجموعه ای ترکیبی از تصاویر واقعی و جعلی هستید، ممکن است این کار شامل برگرداندن تصادفی برچسب روی برخی از تصاویر باشد.

```
# randomly flip some labels
def noisy_labels(y, p_flip):
 # determine the number of labels to flip
 n_select = int(p_flip * y.shape[0])
 # choose labels to flip
 flip_ix = choice([i for i in range(y.shape[0])], size=n_select)
 # invert the labels in place
 y[flip_ix] = 1 - y[flip_ix]
 return y
```

شکل 10 - کد اضافه کردن نویز به برچسب ها

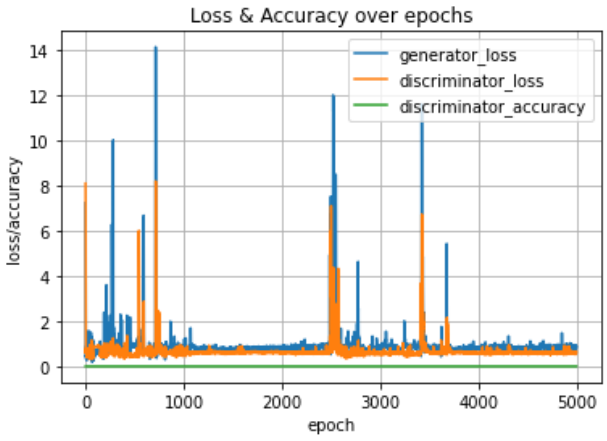
پس از آموزش شبکه با تغییرات جدید نتایج زیر حاصل شد:



شکل 11 - خروجی‌های آموزش **stable dcgan**

در این نتایج به وضوح میتوان افزایش کیفیت پیش‌بینی‌ها را مشاهده کرد و تصاویر نهایی با دقت بسیار بیشتری نسبت به مولد قبلی تولید میشوند و شباهت بیشتری به تصاویر مرجه دارند.

همچنین نمودار lossمربوط به این شبکه نیز به شرح زیر است:



شکل 12 - نمودار **loss** برای **stable dcgan**

پاسخ ۲ – شبکه متخاصم مولد طبقه بند کمکی و شبکه Wassertein

۲-۱. شبکه متخاصم مولد طبقه بند کمکی

یک شبکه متخاصم مولد طبقه‌بندی کننده کمکی (ACGAN) گونه‌ای از معماری GAN است که برای تولید داده‌های جدید طراحی شده است که هم شبیه به یک مجموعه داده معین است و هم به یک کلاس یا برچسب خاص تعلق دارد. ACGAN از دو بخش اصلی تشکیل شده است: یک مولد و یک تشخیص دهنده. مولد برای ایجاد نمونه های داده جدید آموزش دیده است، در حالی که متمایز کننده برای طبقه بندی نمونه های واقعی و تولید شده در کلاس های مربوطه آموزش دیده است.

مراحل آموزش ACGAN مشابه مراحل GAN استاندارد است، اما با چند تفاوت کلیدی:

- شبکه های مولد و تفکیک کننده را تعریف کنید. شبکه ژنراتور معمولاً از چندین لایه از لایه های کانولوشنال جابجا شده تشکیل شده است که برای نمونه برداری از یک بردار نویز ورودی به اندازه خروجی مورد نظر استفاده می شود. شبکه تشخیص دهنده معمولاً از چندین لایه از لایه های کانولوشن تشکیل شده است که برای طبقه بندی تصاویر به عنوان واقعی یا جعلی و همچنین پیش بینی کلاس تصویر استفاده می شود.
- شبکه های مولد و تفکیک کننده را با هم آموزش دهید. مولد تصاویر جدیدی تولید می کند، در حالی که متمایز کننده سعی می کند آنها را به عنوان واقعی یا جعلی طبقه بندی کند و همچنین کلاس آنها را پیش بینی کند. مولد برای تولید تصاویری آموزش دیده است که می تواند متمایز کننده را فریب دهد و همچنین تصاویر یک کلاس خاص را تولید می کند، در حالی که تشخیص دهنده آموزش می بیند که تصاویر تولید شده را به درستی شناسایی کند و همچنین کلاس آنها را پیش بینی کند.
- مرحله 3 را برای چند دوره تکرار کنید. مولد و متمایز کننده همچنان به پیشرفت خود ادامه می دهند زیرا یاد می گیرند که با یکدیگر همکاری بهتری داشته باشند. این روند تا زمانی ادامه می یابد که ژنراتور تصاویری با کیفیت بالا تولید کند که به سختی از تصاویر واقعی متمایز می شوند و همچنین متعلق به کلاس خاصی هستند.

- از مولد برای تولید تصاویر جدید از یک کلاس خاص استفاده کنید. پس از اتمام آموزش، می توان از ژنراتور برای ایجاد تصاویر جدید از یک کلاس خاص با ارائه یک بردار نویز و برچسب کلاس به عنوان ورودی استفاده کرد.

ACGAN ها برای تولید تصاویر، فیلم ها، صدا و متن جدید و بهبود وظایف طبقه بندی تصویر و همچنین وظایف تولید تصویر استفاده شده اند..

معماری های استفاده شده در این شبکه به شرح زیر هستند:

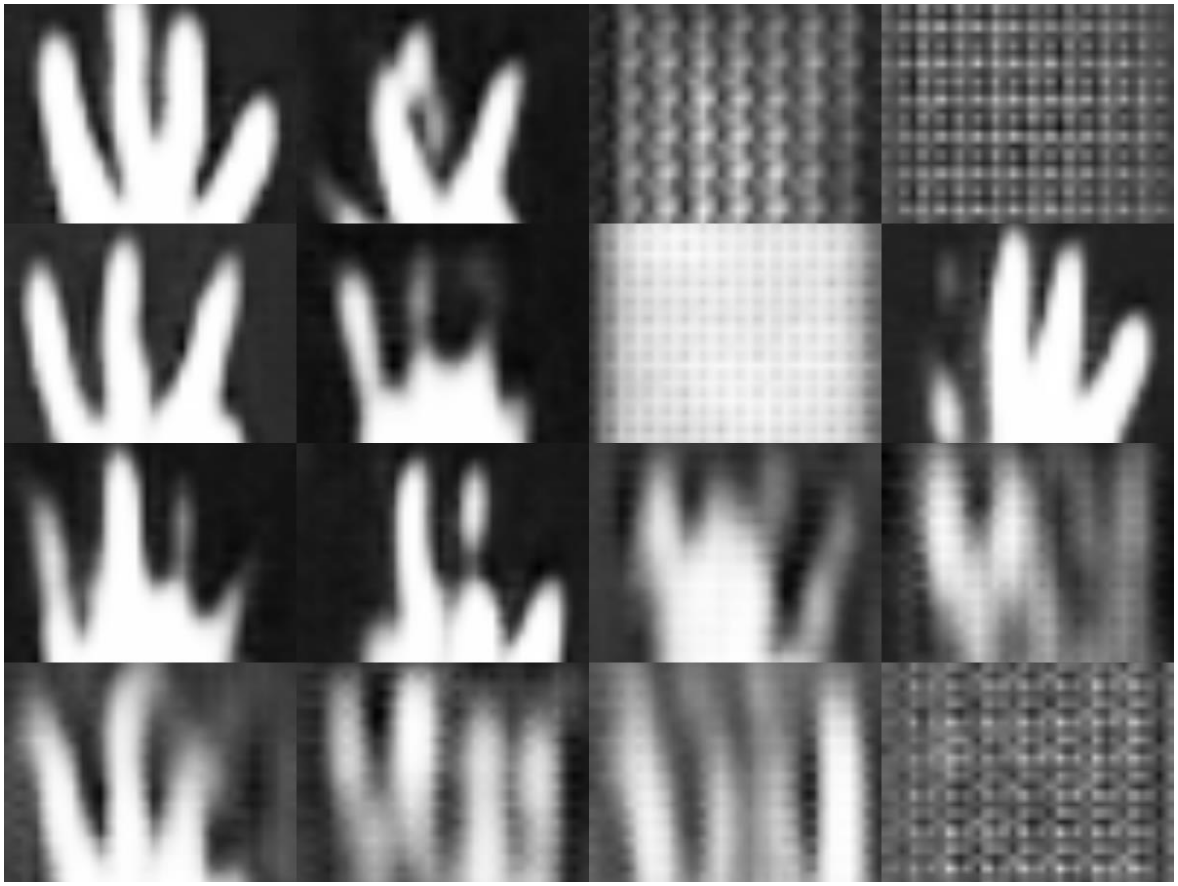
```
Generator model:
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 8192)              1056768

 leaky_re_lu (LeakyReLU)     (None, 8192)              0

 reshape (Reshape)           (None, 8, 8, 128)         0

 conv2d_transpose (Conv2DTra  (None, 16, 16, 128)      262272
 nspose)

 leaky_re_lu_1 (LeakyReLU)   (None, 16, 16, 128)       0

 batch_normalization (BatchN  (None, 16, 16, 128)      512
 ormalization)

 conv2d_transpose_1 (Conv2DT  (None, 32, 32, 128)      262272
 ranspose)

 leaky_re_lu_2 (LeakyReLU)   (None, 32, 32, 128)       0

 batch_normalization_1 (Batc  (None, 32, 32, 128)      512
 hNormalization)

 conv2d (Conv2D)             (None, 32, 32, 1)         6273

=================================================================
Total params: 1,588,609
Trainable params: 1,588,097
Non-trainable params: 512
_________________________________________________________________
```

شکل 13 - معماری مولد **acgan**

```
Discriminator model:
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_1 (Conv2D)           (None, 16, 16, 32)        320

 leaky_re_lu_3 (LeakyReLU)   (None, 16, 16, 32)        0

 dropout (Dropout)           (None, 16, 16, 32)        0

 conv2d_2 (Conv2D)           (None, 16, 16, 64)        18496

 leaky_re_lu_4 (LeakyReLU)   (None, 16, 16, 64)        0

 dropout_1 (Dropout)         (None, 16, 16, 64)        0

 conv2d_3 (Conv2D)           (None, 8, 8, 128)         73856

 leaky_re_lu_5 (LeakyReLU)   (None, 8, 8, 128)         0

 dropout_2 (Dropout)         (None, 8, 8, 128)         0

 conv2d_4 (Conv2D)           (None, 8, 8, 256)         295168

 leaky_re_lu_6 (LeakyReLU)   (None, 8, 8, 256)         0

 dropout_3 (Dropout)         (None, 8, 8, 256)         0

 global_max_pooling2d (Globa  (None, 256)              0
 lMaxPooling2D)

 dense_1 (Dense)             (None, 1)                 257

=================================================================
Total params: 388,097
Trainable params: 388,097
Non-trainable params: 0
_________________________________________________________________
```

شکل 14 - معماری تفکیک کننده **acgan**

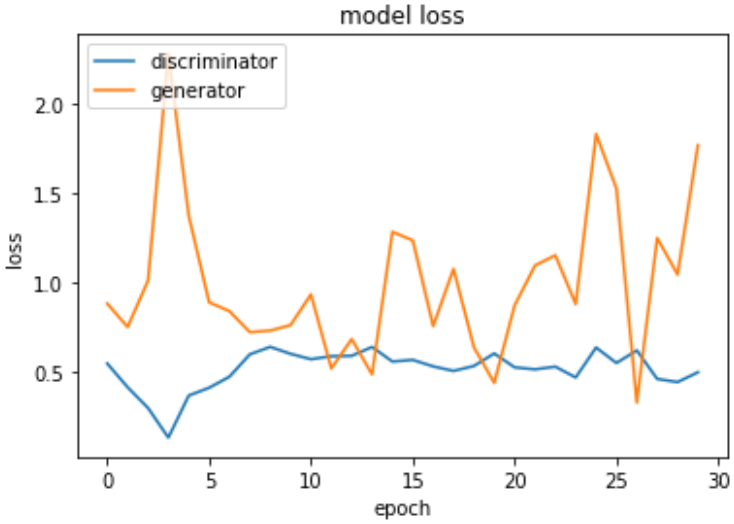
پس از آموزش شبکه و گرفتن برخی نتایج برای مراحل مختلف خروجی‌های نمونه زیر را خواهیم داشت:



شکل 15 - نمونه خروجی **acgan**

خروجی های مشاهده شده در بالا برای کلاس‌های ۲و ۳ و ۴ برای ایپاک‌های مختلف تولید شده و بعنوان نمونه آورده شده است.

همچنین نمودار loss برای این شبکه به صورت زیر است:



شکل 16 - نمودار **loss** برای **acgan**

۲-۲ شبکه Wassertein

مشکل فروپاشی حالت یک مسئله رایج است که در شبکه‌های متخاصم مولد (GAN) رخ می‌دهد که در آن ژنراتور به جای تولید مجموعه‌ای متنوع از نمونه‌ها که کل توزیع هدف را پوشش می‌دهد، نمونه‌ها را تنها از یک زیرمجموعه کوچک از توزیع هدف تولید می‌کند. این می تواند منجر به کیفیت پایین نمونه های تولید شده و تنوع کمتر شود.

مشکل فروپاشی حالت ممکن است به دلیل تعدادی از عوامل مانند قوی بودن بیش از حد ژنراتور، ضعیف بودن بیش از حد تشخیص دهنده، یا بهینه نبودن GAN به درستی رخ دهد. هنگامی که ژنراتور بیش از حد قدرتمند می شود، به راحتی می تواند متمایز کننده را فریب دهد، که می تواند باعث شود ژنراتور به یک راه حل غیربهینه همگرا شود که در آن فقط نمونه هایی از زیر مجموعه کوچکی از توزیع هدف تولید می کند. این می تواند منجر به تولید نمونه هایی از ژنراتور شود که متنوع نیستند و تنوع داده های واقعی ندارند.

چندین تکنیک برای جلوگیری از فروپاشی حالت وجود دارد، مانند استفاده از معماری های مختلف، توابع مختلف از دست دادن یا روش های مختلف آموزشی. برخی از انواع GAN مانند WGAN، WGAN-GP و SNGAN برای رفع این مشکل با استفاده از توابع اتلاف مختلف، یا با افزودن یک عبارت جریمه گرادیان به تابع ضرر، یا با استفاده از نرمال سازی طیفی در مولد و تفکیک کننده پیشنهاد شده اند.

شایان ذکر است که فروپاشی حالت یک مشکل دشوار برای غلبه بر آن است و هنوز یک منطقه فعال تحقیقاتی در زمینه GAN است.

شبکه متخاصم مولد Wasserstein (WGAN) گونه‌ای از معماری GAN است که هدف آن بهبود پایداری و کیفیت نمونه‌های تولید شده با استفاده از یک تابع تلفات متفاوت به نام ضرر Wasserstein است.

مراحل آموزش WGAN مشابه مراحل GAN استاندارد است، اما با چند تفاوت کلیدی:

- مجموعه داده ای از تصاویر را جمع آوری و پیش پردازش کنید. اندازه تصاویر باید تغییر داده شود و به اندازه ورودی مورد نظر برای WGAN نرمال شود.
- شبکه های مولد و تفکیک کننده را تعریف کنید. شبکه ژنراتور معمولاً از چندین لایه از لایه های کانولوشنال جابجا شده تشکیل شده است که برای نمونه برداری از یک بردار نویز ورودی به اندازه خروجی مورد نظر استفاده می شود. شبکه تفکیک کننده معمولاً از چندین لایه از

لایه های کانولوشن تشکیل شده است که برای طبقه بندی تصاویر به عنوان واقعی یا جعلی و تخمین فاصله Wasserstein بین توزیع داده های واقعی و تولید شده استفاده می شود.

- شبکه های مولد و تفکیک کننده را با هم آموزش دهید. ژنراتور تصاویر جدیدی تولید می کند، در حالی که متمایز کننده سعی می کند آنها را به عنوان واقعی یا جعلی طبقه بندی کند و همچنین فاصله Wasserstein را تخمین بزند. ژنراتور برای تولید تصاویری که می تواند فاصله Wasserstein را افزایش دهد آموزش دیده است، در حالی که تشخیص دهنده برای تخمین صحیح فاصله Wasserstein آموزش دیده است.
- مرحله 3 را برای چند دوره تکرار کنید. مولد و متمایز کننده همچنان به پیشرفت خود ادامه می دهند زیرا یاد می گیرند که با یکدیگر همکاری بهتری داشته باشند. این روند تا زمانی ادامه می یابد که ژنراتور تصاویری با کیفیت بالا تولید کند که به سختی می توان آنها را از تصاویر واقعی متمایز کرد.
- از مولد برای تولید تصاویر جدید استفاده کنید. پس از اتمام آموزش، می توان از ژنراتور برای ایجاد تصاویر جدید با ارائه یک بردار نویز به عنوان ورودی استفاده کرد.

منظور از Wassertein برای loss قطعه کد زیر است:

```
# Using Wasserstein Loss as it shows promising results.
    # https://arxiv.org/pdf/1701.07875.pdf
def wasserstein_loss(self, y_true, y_pred):
    #y_true and y_pred are tensors and thus tensor computations are needed.
    # K.mean (keras.backend) takes mean of tensor.
    return K.mean(y_true * y_pred)
```

معماری مورد استفاده برای این شبکه به شرح زیر است:

```
Model: "sequential_9"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_6 (Dense)             (None, 3136)              316736

 reshape_3 (Reshape)         (None, 7, 7, 64)          0

 up_sampling2d_9 (UpSampling  (None, 14, 14, 64)       0
 2D)

 conv2d_27 (Conv2D)          (None, 10, 10, 256)       409856

 batch_normalization_21 (Bat  (None, 10, 10, 256)      1024
 chNormalization)

 activation_9 (Activation)   (None, 10, 10, 256)       0

 up_sampling2d_10 (UpSamplin  (None, 20, 20, 256)      0
 g2D)

 conv2d_28 (Conv2D)          (None, 16, 16, 128)       819328

 batch_normalization_22 (Bat  (None, 16, 16, 128)      512
 chNormalization)

 activation_10 (Activation)  (None, 16, 16, 128)       0

 up_sampling2d_11 (UpSamplin  (None, 32, 32, 128)      0
 g2D)

 conv2d_29 (Conv2D)          (None, 29, 29, 64)        131136

 batch_normalization_23 (Bat  (None, 29, 29, 64)       256
 chNormalization)

 activation_11 (Activation)  (None, 29, 29, 64)        0

 conv2d_30 (Conv2D)          (None, 28, 28, 1)         257

_________________________________________________________________
```

شکل 17 - معماری مولد **wgan**

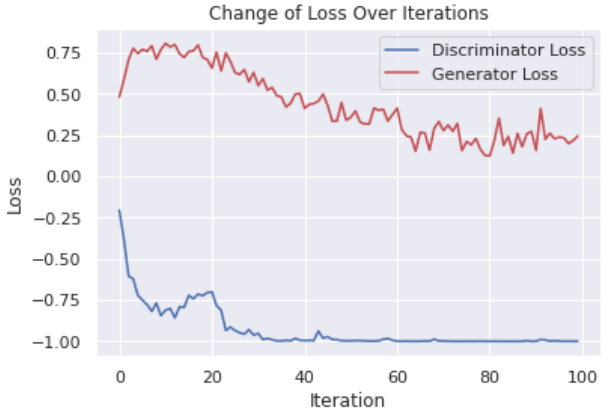
conv2d_31 (Conv2D)	(None, 13, 13, 16)	160
leaky_re_lu_15 (LeakyReLU)	(None, 13, 13, 16)	0
dropout_15 (Dropout)	(None, 13, 13, 16)	0
conv2d_32 (Conv2D)	(None, 7, 7, 32)	4640
batch_normalization_24 (BatchNormalization)	(None, 7, 7, 32)	128
leaky_re_lu_16 (LeakyReLU)	(None, 7, 7, 32)	0
dropout_16 (Dropout)	(None, 7, 7, 32)	0
conv2d_33 (Conv2D)	(None, 4, 4, 64)	18496
batch_normalization_25 (BatchNormalization)	(None, 4, 4, 64)	256
leaky_re_lu_17 (LeakyReLU)	(None, 4, 4, 64)	0
dropout_17 (Dropout)	(None, 4, 4, 64)	0
conv2d_34 (Conv2D)	(None, 4, 4, 128)	73856
batch_normalization_26 (BatchNormalization)	(None, 4, 4, 128)	512
leaky_re_lu_18 (LeakyReLU)	(None, 4, 4, 128)	0
dropout_18 (Dropout)	(None, 4, 4, 128)	0
conv2d_35 (Conv2D)	(None, 4, 4, 256)	131328
batch_normalization_27 (BatchNormalization)	(None, 4, 4, 256)	1024
leaky_re_lu_19 (LeakyReLU)	(None, 4, 4, 256)	0
dropout_19 (Dropout)	(None, 4, 4, 256)	0
flatten_3 (Flatten)	(None, 4096)	0

شکل 18 - معماری تفکیک کننده **wgan**

در نهایت پس از آموزش نتایج زیر حاصل میشود:

شکل 19 - خروجی **wgan**

همانطور که مشاهده میشود دقت dcgan با درنظر گرفتن cross entropy به عنوان loss نتایج بهتری را به نظر به همراه داشته است. همچنین نمودار lossبه شرح زیر است:



شکل 20 - نمودار **loss** برای **wgan**