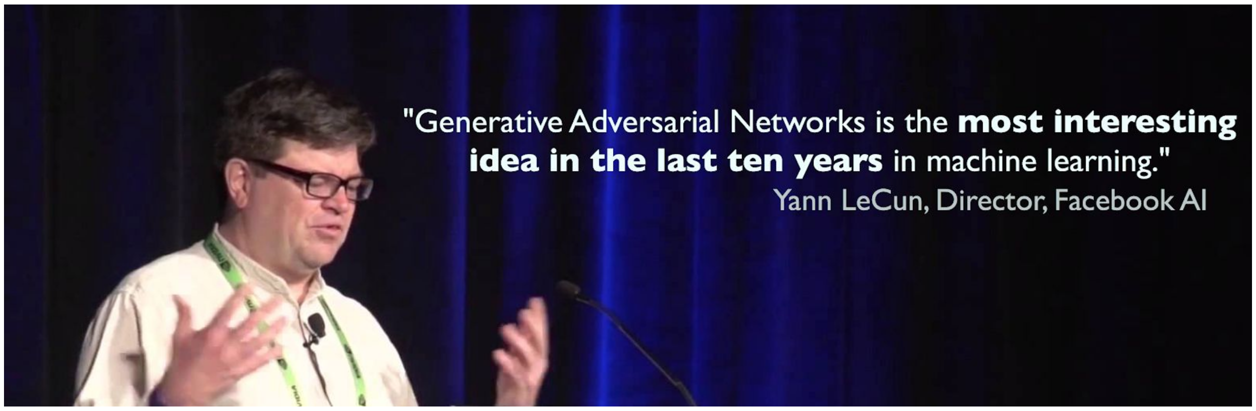
Generative Adversarial Networks
Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Advances in Neural Information Processing Systems, pages 2672–2680, 2014.

Comments
Yann LeCun (Facebook AI research director) described it as “the most interesting idea in the last 10 years in Machine Learning.”


"Generative Adversarial Networks is the **most interesting idea in the last ten years** in machine learning."
Yann LeCun, Director, Facebook AI

What are GANs used for?
The previous deep learning techniques we learned are used for recognition. GAN technique is going to let the machine generate stuff based on what it learned. To note, GAN does not simply memorize the given dataset.

For example, we first learned how to recognize digits like 0-9, then we tried to mimic the shape of digits and created digits in our styles. It is called generation.

Image generation.

Image-to-Image translation
Supervised / unsupervised

Text-to-Image

Super resolution

Photo inpainting

What is a GAN?

Example

How to be a master of producing fake dollars?
Criminor: Produced 1st version fake dollars and the FBI cannot figure them out.
FBI: Found the new fake dollars and successfully figured all 1st version fake dollars out.
Criminor: Produced 2nd version fake dollars and the FBI cannot figure them out.
FBI: Found the new fake dollars and successfully figured all 2nd version fake dollars out.
…
n-th version fake dollars
...

=> Criminor: An expert on making fake dollars
=> FBI: An expert on figuring fake dollars.

In the image generation field

The underlying essence of GAN is simulating the distribution of real data. Then we can sample any random points from this distribution.
If a model only learns the data points rather than the distribution, this model can only memorize the dataset.
A model can generate new data samples when it learns the whole distribution.

Objective for each model

- Discriminator is going to distinguish all fake images.
  Intuitively, the objective function is maximizing:
  $$J^{(D)} = high\_score(real\ images) + low\_score(fake\ images)$$
- We are familiar with such models, which are going to classify two types of images.
  Discriminator is like a binary classifier to distinguish real or fake:
  Input a given image, Discriminator function needs to output its probability of reality.
  $D(x) = p$. (by softmax or sigmoid)
- Loss function: Cross-entropy (preferred), MSE

[Cross-entropy loss]

Any loss consisting of a negative log-likelihood is a cross-entropy between the empirical distribution defined by the training set and the probability distribution defined by model. For example, mean squared error is the cross-entropy between the empirical distribution and a Gaussian model.

— Page 132, Deep Learning, 2016.

- Cross-entropy loss:
  Minimize the divergence between real distribution X_r in the training set and the probability distribution defined by D model.
  Maximize the divergence between generated distribution X_g and the probability distribution defined by D model.
  Then the loss function for D: θ_D

$$\begin{aligned}\min_{\theta_D} L^{(D)}\left(X_r, X_g\right) = &- \mathbb{E}_{x_r\sim X_r}\left[\log\left(D\left(x_r\right)\right)\right] \\ &- \mathbb{E}_{x_g\sim X_g}\left[\log\left(1-D\left(x_g\right)\right)\right]\end{aligned}$$

\begin{align*}
\underset{\theta_D}\min L^{\left(D\right)}\left(X_r,X_g\right)=&-\mathbb{E}_{x_r\sim X_r}\left[\log\left(D\left(x_r\right)\right)\right]\\
&-\mathbb{E}_{x_g\sim X_g}\left[\log\left(1-D\left(x_g\right)\right)\right]
\end{align*}

- Generator is going to generate real-like images to fool the discriminator.
  Intuitively, the objective function is maximizing:

$$J^{(G)} = high\_score(fake\ images)$$

  where high_score is criticized by D, while G can only control the parameters of generating fake images $x_g = G(z)$.
  Input a noise vector z to control the generated image $x_g = G(z)$.
  Generator function G minimize the divergence between generated distribution X_g and the probability distribution defined by D model.

Then the loss function for G: θ_G
Non-saturating: Min(divergence b/w generated distribution and distribution defined by D model)

$$\min_{\theta_G} L^{(G)}\left(X_g\right) = -\mathbb{E}_{x_g\sim X_g}\left[\log\left(D\left(x_g\right)\right)\right]$$

\underset{\theta_G}\min L^{\left(G\right)}\left(X_g\right)=-\mathbb{E}_{x_g\sim X_g}\left[\log\left(D\left(x_g\right)\right)\right]

where \textit{D} denotes the discriminator function, \textit{G} denotes the generator function, $\theta_G $ is the parameters of the generator, $\theta_D $ is the parameters of the discriminator; $x_r $ is sampled from the real distribution $X_r $, $x_g $ is sampled from the generated distribution $X_g $, where $x_g=G\left(z\right) $ and z~is a random noise vector sample from normal distribution$z\sim N\left(0,I_{dim(z)}\right) $.

...Beyond the introduction level. Saturating: Max(divergence b/w generated distribution and distribution defined by NOT D model)

$$\mathbb{E}_{x_g\sim X_g}\left[\log\left(1-D\left(x_g\right)\right)\right]$$

Combine these two loss functions together. Minimax problem in Game Theory. (Understand L_D and L_G is more important than this formula)

$$\begin{align*}\underset{\theta_G}{\min}\underset{\theta_D}{\max}L\left(X_r,X_g\right)&=\mathbb{E}_{x_r\sim X_r}\left[\log(D\left(x_r\right))\right]\\ &+\mathbb{E}_{x_g\sim X_g}\left[\log(1-D\left(x_g\right))\right]\end{align*}$$

\begin{align*}
\underset{\theta_G}{\min}\underset{\theta_D}{\max}L\left(X_r,X_g\right)&=\mathbb{E}_{x_r\sim X_r}\left[\log\left(D\left(x_r\right)\right)\right]\\
&+\mathbb{E}_{x_g\sim X_g}\left[\log\left(1-D\left(x_g\right)\right)\right]
\end{align*}

We cannot optimize this combined loss function by changing θ_G and θ_D simultaneously. In practice, we can only train these two models alternatively.

How to train a GAN?

Algorithm

Initialization: random $\theta^0{}_G$ and $\theta^0{}_D$

1st iteration

Train D (freeze G)

Input: real images x_r , noise vector $z \sim N(0,1)$

Calculate: $x^0{}_g = G^0(z)$ generated by 0-th version G^0

Output: $D^0(x)$

Optimization: minimize $-logD^0(x_r) - log(1 - D^0(x^0{}_g))$

Update: $\theta^1{}_D$

Train G (freeze D)

Input: noise vector $z \sim N(0,1)$

Calculate: $x^0{}_g = G^0(z)$ generated by 0-th version G^0

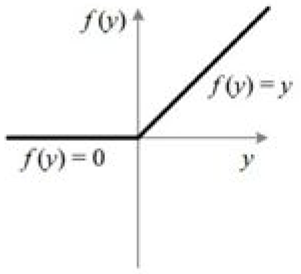
Output: $D^1(G^0(z))$

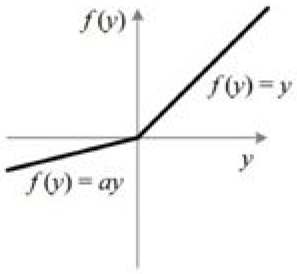
Optimization: minimize $-logD^1(G^0(z))$

Update: $\theta^1{}_G$

… n-th iteration

ReLU and LeakyReLU

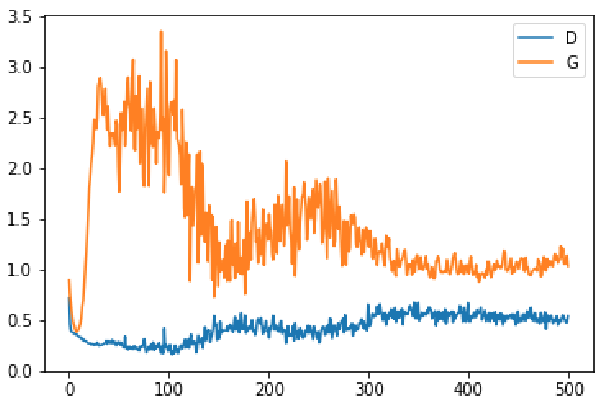




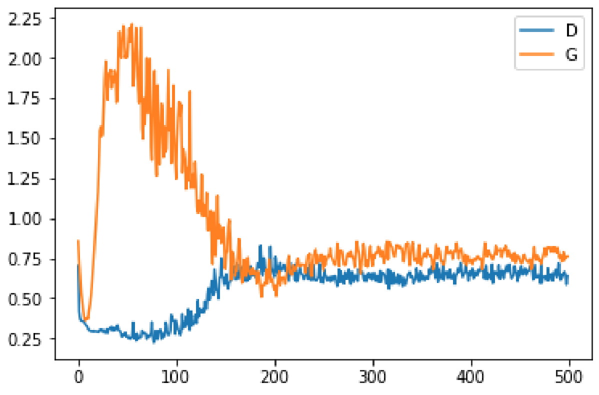
ReLU vs LeakyReLU

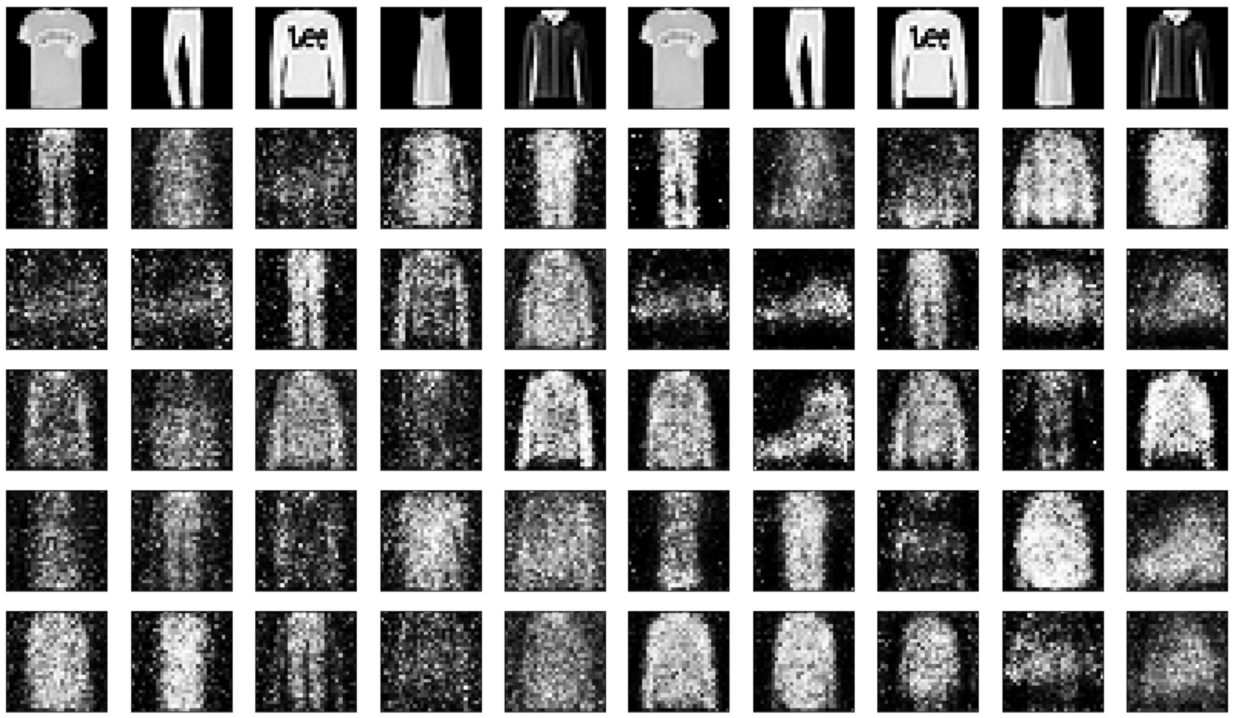
MLPGAN:

ReLU



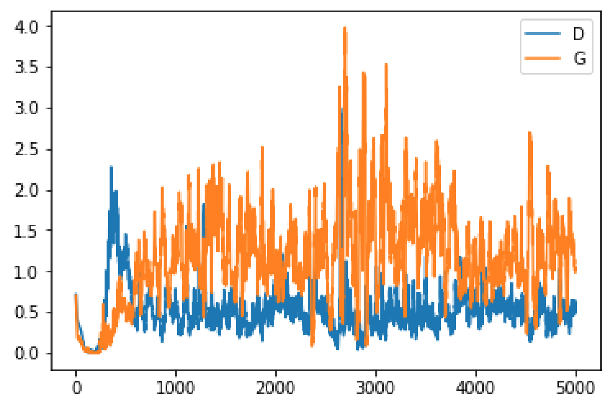
LeakyReLU



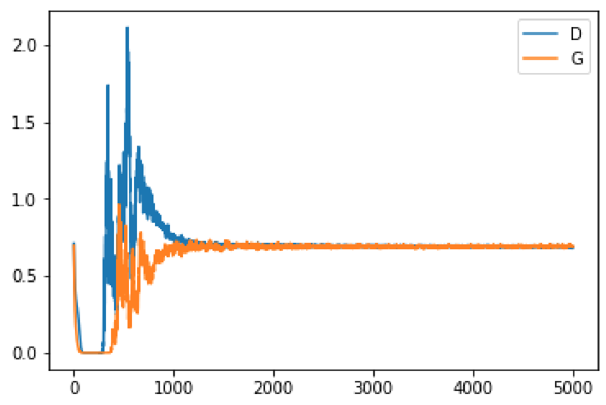


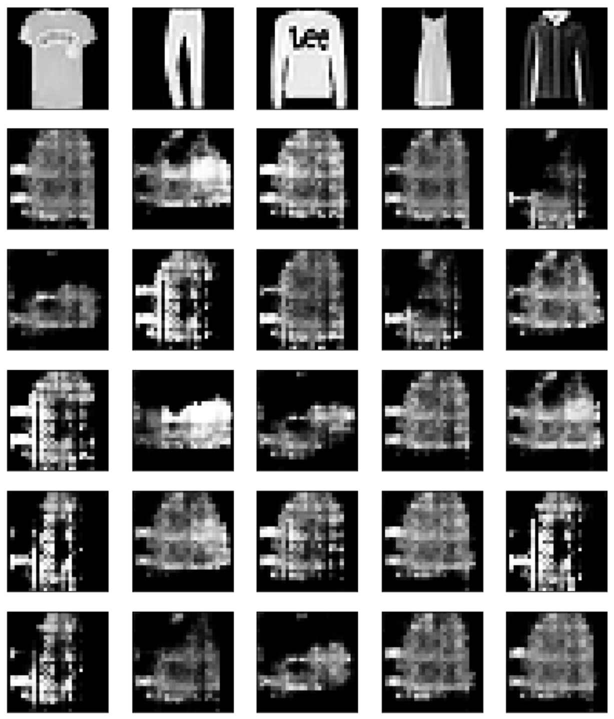
DCGAN:

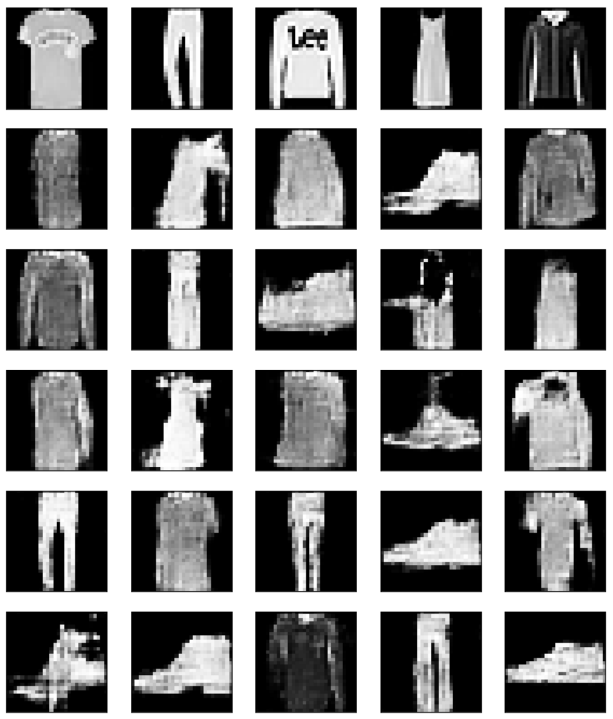
ReLU



LeakyReLU





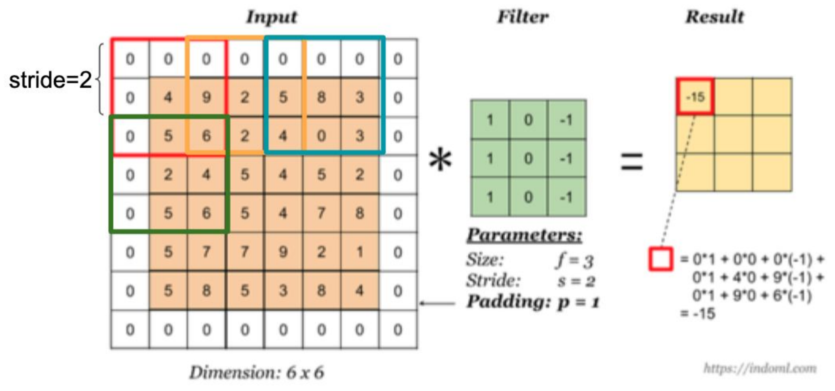


The architecture of Generator and Discriminator.
continuous and differentiable functions of D and G → more stable

Discriminator - Downsampling networks
Dense (not suggested)
Fully connected layer is not good at extracting features.

Maxpooling2D (not suggested)
The output is just selecting the maximum input value within the [height, width] window of input values.
There is no weight and no trainable parameters introduced by this operation.

Conv2D(stride=2)
The output is a linear combination of the input values times a weight for each cell in the [height, width] kernel.
These weights become trainable parameters in your model.



Stride is the moving step size.

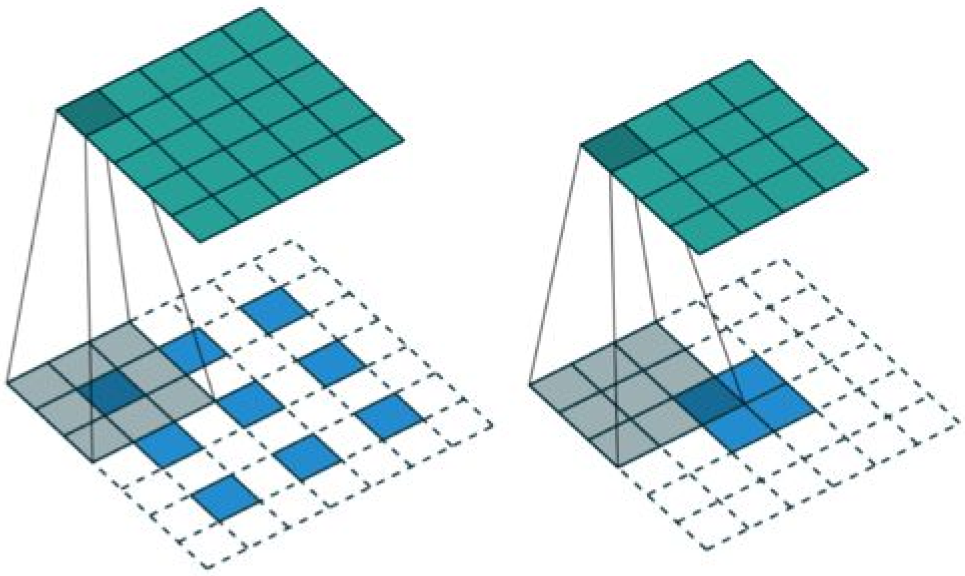
Generator - Upsampling networks
Dense

Upsampling2D

```
        [1, 2]
Input = [3, 4]

         [1, 1, 2, 2]
Output = [1, 1, 2, 2]
         [3, 3, 4, 4]
         [3, 3, 4, 4]
```

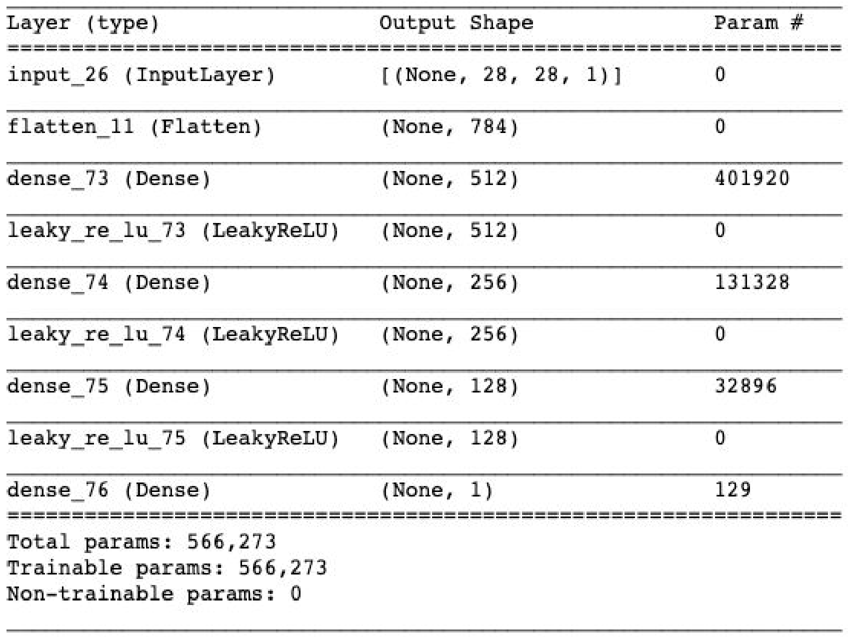
ConvTranspose / Deconvolution
Stride(2,2)   Stride(1,1), Padding=0



Stride here is the reciprocal of the moving step. E.g. stride=2 ⇒ moving step=½. How to move ½ step? Adding zero columns and rows to the original input.

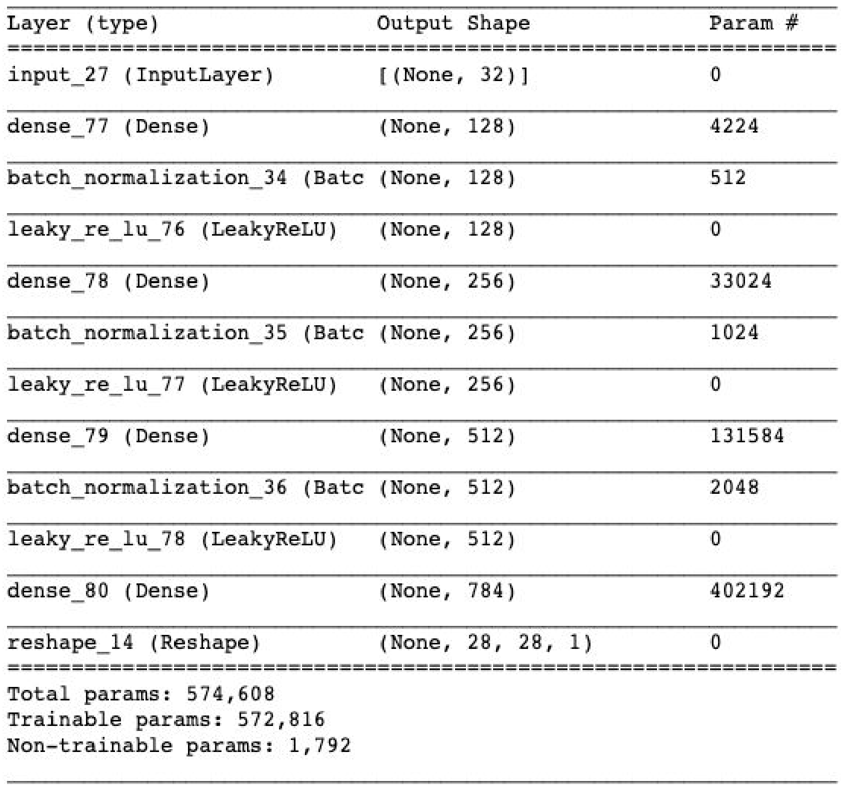
Try our first GAN model with simple MLP-GAN.

Model: ‘Discriminator’



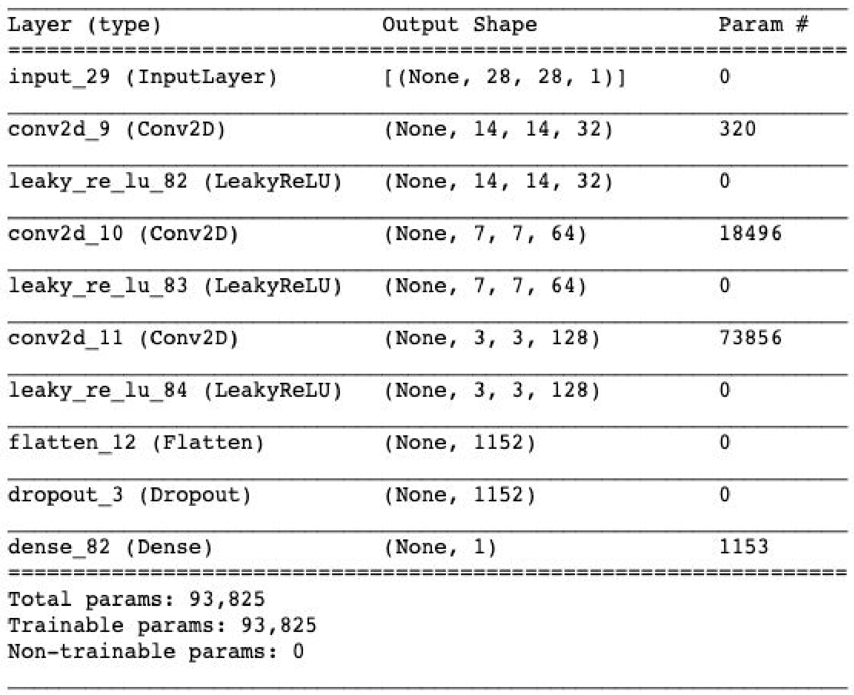
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_26 (InputLayer)        [(None, 28, 28, 1)]       0
_________________________________________________________________
flatten_11 (Flatten)         (None, 784)               0
_________________________________________________________________
dense_73 (Dense)             (None, 512)               401920
_________________________________________________________________
leaky_re_lu_73 (LeakyReLU)   (None, 512)               0
_________________________________________________________________
dense_74 (Dense)             (None, 256)               131328
_________________________________________________________________
leaky_re_lu_74 (LeakyReLU)   (None, 256)               0
_________________________________________________________________
dense_75 (Dense)             (None, 128)               32896
_________________________________________________________________
leaky_re_lu_75 (LeakyReLU)   (None, 128)               0
_________________________________________________________________
dense_76 (Dense)             (None, 1)                 129
=================================================================
Total params: 566,273
Trainable params: 566,273
Non-trainable params: 0
_________________________________________________________________
```

Model: ‘Generator’



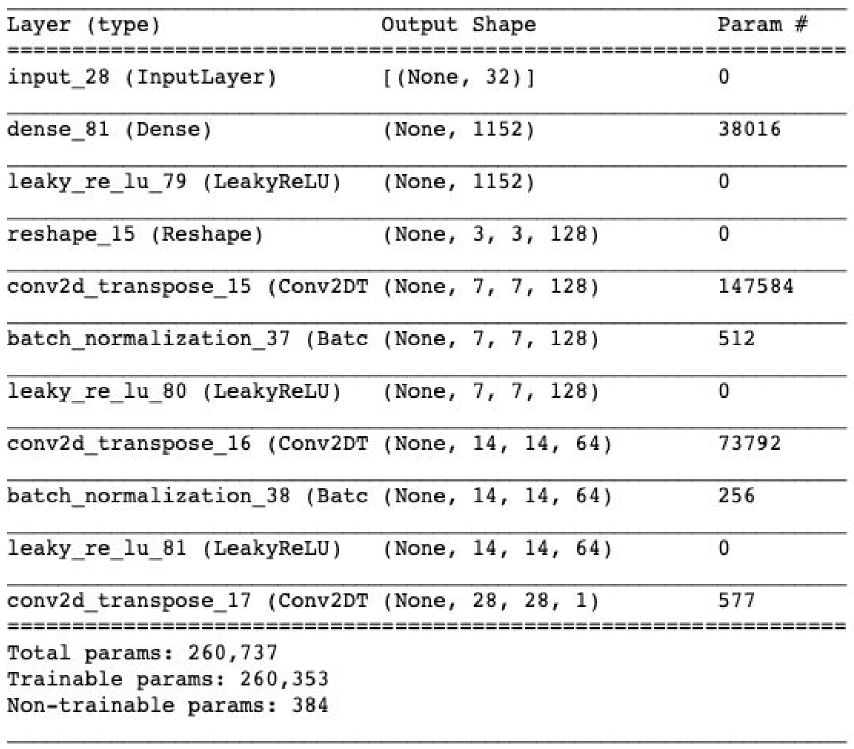
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_27 (InputLayer)        [(None, 32)]              0
_________________________________________________________________
dense_77 (Dense)             (None, 128)               4224
_________________________________________________________________
batch_normalization_34 (Batc (None, 128)               512
_________________________________________________________________
leaky_re_lu_76 (LeakyReLU)   (None, 128)               0
_________________________________________________________________
dense_78 (Dense)             (None, 256)               33024
_________________________________________________________________
batch_normalization_35 (Batc (None, 256)               1024
_________________________________________________________________
leaky_re_lu_77 (LeakyReLU)   (None, 256)               0
_________________________________________________________________
dense_79 (Dense)             (None, 512)               131584
_________________________________________________________________
batch_normalization_36 (Batc (None, 512)               2048
_________________________________________________________________
leaky_re_lu_78 (LeakyReLU)   (None, 512)               0
_________________________________________________________________
dense_80 (Dense)             (None, 784)               402192
_________________________________________________________________
reshape_14 (Reshape)         (None, 28, 28, 1)         0
=================================================================
Total params: 574,608
Trainable params: 572,816
Non-trainable params: 1,792
_________________________________________________________________
```

Discriminator with convolution layer



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_29 (InputLayer)        [(None, 28, 28, 1)]       0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 14, 14, 32)        320
_________________________________________________________________
leaky_re_lu_82 (LeakyReLU)   (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 7, 7, 64)          18496
_________________________________________________________________
leaky_re_lu_83 (LeakyReLU)   (None, 7, 7, 64)          0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 3, 3, 128)         73856
_________________________________________________________________
leaky_re_lu_84 (LeakyReLU)   (None, 3, 3, 128)         0
_________________________________________________________________
flatten_12 (Flatten)         (None, 1152)              0
_________________________________________________________________
dropout_3 (Dropout)          (None, 1152)              0
_________________________________________________________________
dense_82 (Dense)             (None, 1)                 1153
=================================================================
Total params: 93,825
Trainable params: 93,825
Non-trainable params: 0
_________________________________________________________________
```

Generator with convolution layer



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_28 (InputLayer)        [(None, 32)]              0
_________________________________________________________________
dense_81 (Dense)             (None, 1152)              38016
_________________________________________________________________
leaky_re_lu_79 (LeakyReLU)   (None, 1152)              0
_________________________________________________________________
reshape_15 (Reshape)         (None, 3, 3, 128)         0
_________________________________________________________________
conv2d_transpose_15 (Conv2DT (None, 7, 7, 128)         147584
_________________________________________________________________
batch_normalization_37 (Batc (None, 7, 7, 128)         512
_________________________________________________________________
leaky_re_lu_80 (LeakyReLU)   (None, 7, 7, 128)         0
_________________________________________________________________
conv2d_transpose_16 (Conv2DT (None, 14, 14, 64)        73792
_________________________________________________________________
batch_normalization_38 (Batc (None, 14, 14, 64)        256
_________________________________________________________________
leaky_re_lu_81 (LeakyReLU)   (None, 14, 14, 64)        0
_________________________________________________________________
conv2d_transpose_17 (Conv2DT (None, 28, 28, 1)         577
=================================================================
Total params: 260,737
Trainable params: 260,353
Non-trainable params: 384
_________________________________________________________________
```

Problems in tuning

How to measure the quality of generated images?