“Neural Networks”

Shervin Halat

98131018

Homework 6

“FCGAN”

1.

There are multiple proposed loss functions for both Discriminator and Generator models in recent papers. These functions are shown in the table below:

GAN	Discriminator Loss	Generator Loss
MM GAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{GAN}} = -\mathbb{E}_{x\sim p_d}[\log(D(x))] - \mathbb{E}_{\hat{x}\sim p_g}[\log(1-D(\hat{x}))]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{GAN}} = \mathbb{E}_{\hat{x}\sim p_g}[\log(1-D(\hat{x}))]$
NS GAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{NSGAN}} = -\mathbb{E}_{x\sim p_d}[\log(D(x))] - \mathbb{E}_{\hat{x}\sim p_g}[\log(1-D(\hat{x}))]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{NSGAN}} = -\mathbb{E}_{\hat{x}\sim p_g}[\log(D(\hat{x}))]$
WGAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{WGAN}} = -\mathbb{E}_{x\sim p_d}[D(x)] + \mathbb{E}_{\hat{x}\sim p_g}[D(\hat{x})]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{WGAN}} = -\mathbb{E}_{\hat{x}\sim p_g}[D(\hat{x})]$
WGAN GP	$\mathcal{L}_{\mathrm{D}}^{\mathrm{WGANGP}} = \mathcal{L}_{\mathrm{D}}^{\mathrm{WGAN}} + \lambda\mathbb{E}_{\hat{x}\sim p_g}[(\|\nabla D(\alpha x + (1-\alpha\hat{x})\|_2 - 1)^2]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{WGANGP}} = -\mathbb{E}_{\hat{x}\sim p_g}[D(\hat{x})]$
LS GAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{LSGAN}} = -\mathbb{E}_{x\sim p_d}[(D(x)-1)^2] + \mathbb{E}_{\hat{x}\sim p_g}[D(\hat{x})^2]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{LSGAN}} = -\mathbb{E}_{\hat{x}\sim p_g}[(D(\hat{x}-1))^2]$
DRAGAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{DRAGAN}} = \mathcal{L}_{\mathrm{D}}^{\mathrm{GAN}} + \lambda\mathbb{E}_{\hat{x}\sim p_d+\mathcal{N}(0,c)}[(\|\nabla D(\hat{x})\|_2 - 1)^2]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{DRAGAN}} = \mathbb{E}_{\hat{x}\sim p_g}[\log(1-D(\hat{x}))]$
BEGAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{BEGAN}} = \mathbb{E}_{x\sim p_d}[\|\|x - \mathrm{AE}(x)\|\|_1] - k_t\mathbb{E}_{\hat{x}\sim p_g}[\|\|\hat{x} - \mathrm{AE}(\hat{x})\|\|_1]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{BEGAN}} = \mathbb{E}_{\hat{x}\sim p_g}[\|\|\hat{x} - \mathrm{AE}(\hat{x})\|\|_1]$

Although two of the newest losses proposed are ‘least squares’ and ‘Wasserstein’ loss functions, for this project loss functions of ‘binary cross-entropy’ was considered for three main reasons. First, the two new losses are mostly used in larger and more recent GAN models. Second, according to a recent paper published in 2018, it is discovered that all loss functions performed approximately the same when all other properties of the model were held constant. Lastly, since both generator and discriminator models’ outputs are binary with two classes of 0 (fake) and 1 (real), selection of ‘binary cross-entropy’ seems rational.

2.

‘Evaluation on number of layers’

For this evaluation, several experiments on different types of layers were conducted in order to find a plausible GAN model the results of which are as follows.

It should be noted that, initially a base model will be defined and all other modifications will be applied on that with the intention of a fairer comparison among different models.

The base model properties are as follows:

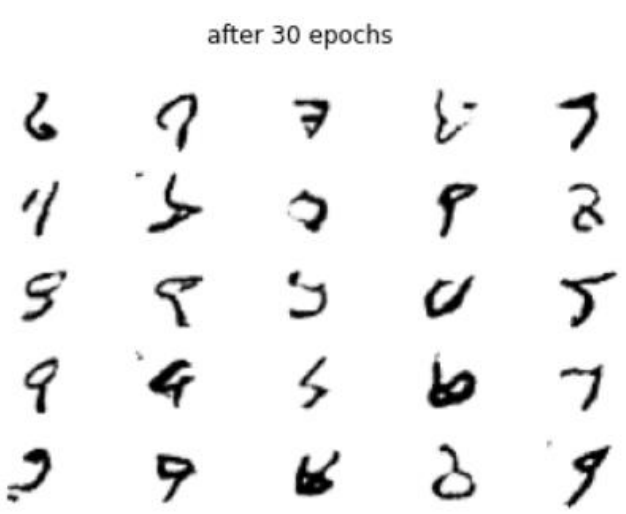
(It should be noted that these selected properties are based on the latest suggestions for GAN models according to recent papers)

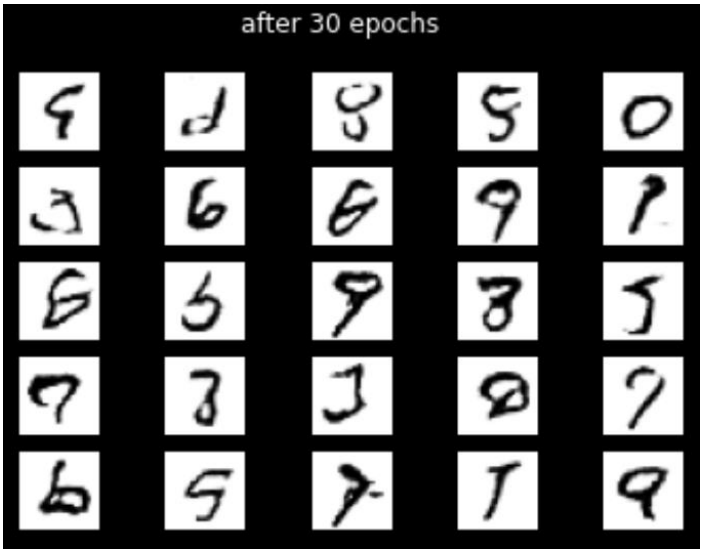
1. Contrary to the past, recently, LeakyReLU activation function with parameter of 0.2 is suggested instead of Relu or other activations.
2. Although it is suggested to use batch normalization layer, no batch normalization layer was considered for base model as it is evaluated in the upcoming experiments.
3. Gaussian weigh initialization was considered for all layers. (with zero mean and 0.02 standard deviation)
4. Adam optimizer with 0.0002 learning rate and 0.5 beta was considered.
5. All images were normalized to the range of [-1 , 1]; therefore, activation function of ‘tanh’ was considered for the output layer of generator.
6. Gaussian distribution was considered for latent space.
7. Discriminator model is trained on real and fake images separately.

8. Also, it should be noted that contrary to the suggestions of using smoothed labels, here, hard labels were preferred since it seemed that it resulted in more favorable outputs at least for 50 epochs. Results of this experiment are as follows:

   (left one was obtained with smooth labels and right one with hard labels which seems much more favorable.)

   (Images below are obtained from DCGAN!)





The only main properties which are not mentioned above are number of layers (such as Dense and batch normalization and dropout layers) which will be evaluated in the upcoming experiments.

(All models are obtained with batch size of 256 and 50 number of epochs.)

“base model”

Base model configuration:

Base models’ discriminator consists of Dense and LeakyReLU layers with one Flatten at the bottom.

Base models’ generator consists of Dense and LeakyReLU layers with one Reshap and Dense layers at the bottom.

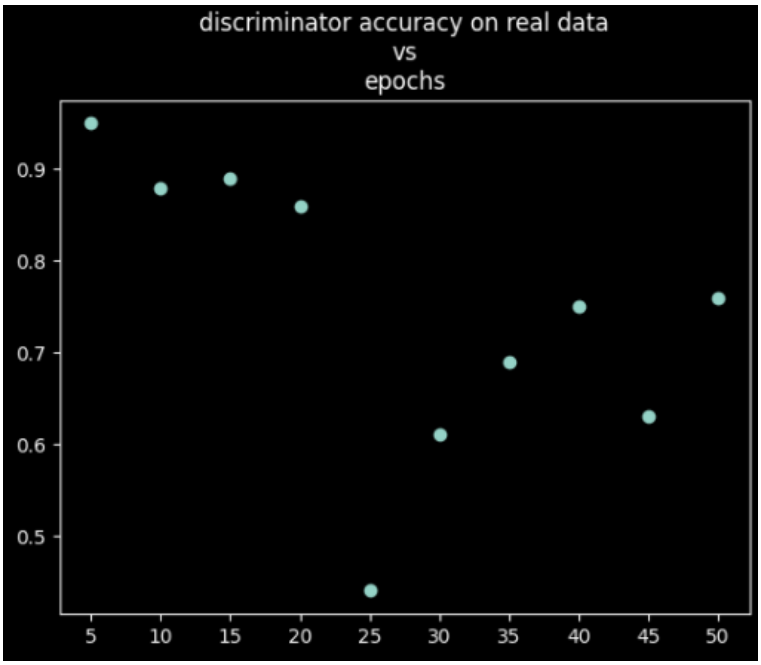
And lastly, the GAN model is defined by stacking discriminator and generator models with turning trainability of discriminator to false.

```python
def define_discriminator(in_shape=(28,28,1)):
  discriminator = Sequential()
  init = RandomNormal(mean=0.0,stddev=0.02)
  discriminator.add(Flatten(input_shape=in_shape))

  discriminator.add(Dense(1024,kernel_initializer=init))
  discriminator.add(LeakyReLU(0.2))

  discriminator.add(Dense(512,kernel_initializer=init))
  discriminator.add(LeakyReLU(0.2))

  discriminator.add(Dense(256,kernel_initializer=init))
  discriminator.add(LeakyReLU(0.2))

  discriminator.add(Dense(1, activation='sigmoid',kernel_initializer=init))
  opt = Adam(lr=0.0002, beta_1=0.5)
  discriminator.compile(loss='binary_crossentropy', optimizer=opt, metrics=['accuracy'])
  return discriminator

def define_generator(latent_dim):
  generator = Sequential()
  init = RandomNormal(mean=0.0, stddev=0.02)

  generator.add(Dense(256, input_dim=latent_dim,kernel_initializer=init))
  generator.add(LeakyReLU(0.2))

  generator.add(Dense(512,kernel_initializer=init))
  generator.add(LeakyReLU(0.2))

  generator.add(Dense(1024,kernel_initializer=init))
  generator.add(LeakyReLU(0.2))

  generator.add(Dense(28*28, activation='tanh',kernel_initializer=init))
  generator.add(Reshape((28, 28, 1)))
  return generator

def define_gan(g_model, d_model):
  d_model.trainable = False
  model = Sequential()
  model.add(g_model)
  model.add(d_model)
  opt = Adam(lr=0.0002, beta_1=0.5)
  model.compile(loss='binary_crossentropy', optimizer=opt)
  return model
```

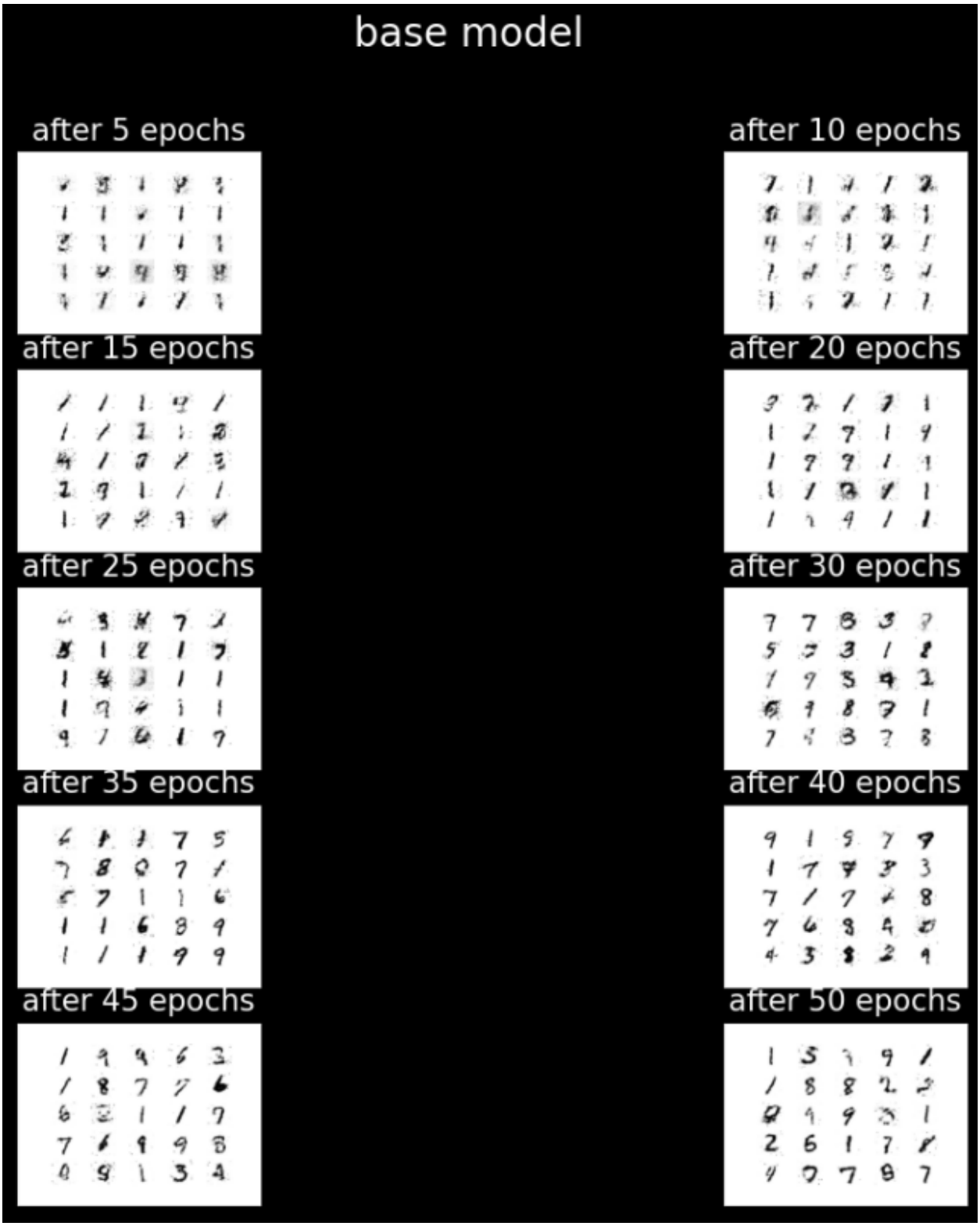
Base models' accuracies in details are as follows:

```
>5, 234/234, d_loss=0.229, g_loss=2.127
>Discriminator accuracy on real_data: 95%, fake_data: 86%
>10, 234/234, d_loss=0.234, g_loss=2.955
>Discriminator accuracy on real_data: 88%, fake_data: 100%
>15, 234/234, d_loss=0.349, g_loss=1.828
>Discriminator accuracy on real_data: 89%, fake_data: 91%
>20, 234/234, d_loss=0.290, g_loss=1.893
>Discriminator accuracy on real_data: 86%, fake_data: 97%
>25, 234/234, d_loss=0.418, g_loss=2.040
>Discriminator accuracy on real_data: 44%, fake_data: 100%
>30, 234/234, d_loss=0.461, g_loss=1.638
>Discriminator accuracy on real_data: 61%, fake_data: 94%
>35, 234/234, d_loss=0.474, g_loss=1.540
>Discriminator accuracy on real_data: 69%, fake_data: 93%
>40, 234/234, d_loss=0.460, g_loss=1.564
>Discriminator accuracy on real_data: 75%, fake_data: 85%
>45, 234/234, d_loss=0.387, g_loss=1.881
>Discriminator accuracy on real_data: 63%, fake_data: 88%
>50, 234/234, d_loss=0.444, g_loss=1.206
>Discriminator accuracy on real_data: 76%, fake_data: 77%
```

Discriminator accuracies during training are as follow:



Output of base model after 50 epochs:



As we can see in the figure above, model has the best output after 50 epochs which is rather consistent with the accuracies of discriminator shown in the previous figure.

“addition of batch normalization layer”

For this experiment, batch normalization layer was added just after dense layers in the generator model but all other properties were kept unchanged.

```
def define_generator(latent_dim):
  generator = Sequential()
  init = RandomNormal(mean=0.0, stddev=0.02)

  generator.add(Dense(256, input_dim=latent_dim,kernel_initializer=init))
  generator.add(BatchNormalization())
  generator.add(LeakyReLU(0.2))

  generator.add(Dense(512,kernel_initializer=init))
  generator.add(BatchNormalization())
  generator.add(LeakyReLU(0.2))

  generator.add(Dense(1024,kernel_initializer=init))
  generator.add(BatchNormalization())
  generator.add(LeakyReLU(0.2))

  generator.add(Dense(28*28, activation='tanh',kernel_initializer=init))
  generator.add(Reshape((28, 28, 1)))
  return generator
```

In total, three batch-normalization layers are added to the base model.

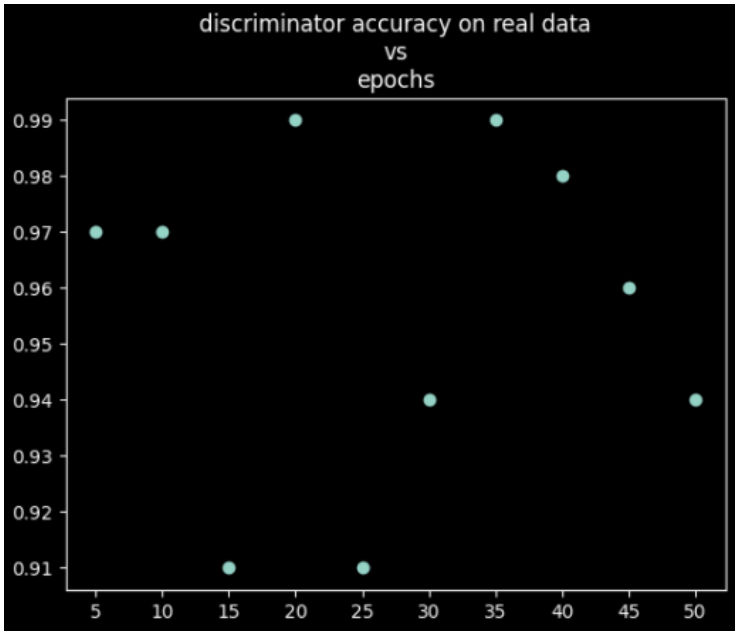
The results after this modification are as follow:

Accuracies and losses of all models during training are listed below:

```
>5, 234/234, d_loss=0.241, g_loss=0.396
>Discriminator accuracy on real_data: 97%, fake_data: 79%
>10, 234/234, d_loss=0.236, g_loss=0.405
>Discriminator accuracy on real_data: 97%, fake_data: 94%
>15, 234/234, d_loss=0.284, g_loss=1.049
>Discriminator accuracy on real_data: 91%, fake_data: 87%
>20, 234/234, d_loss=0.400, g_loss=0.441
>Discriminator accuracy on real_data: 99%, fake_data: 60%
>25, 234/234, d_loss=0.139, g_loss=1.487
>Discriminator accuracy on real_data: 91%, fake_data: 100%
>30, 234/234, d_loss=0.082, g_loss=2.385
>Discriminator accuracy on real_data: 94%, fake_data: 99%
>35, 234/234, d_loss=0.106, g_loss=1.580
>Discriminator accuracy on real_data: 99%, fake_data: 94%
>40, 234/234, d_loss=0.078, g_loss=2.152
>Discriminator accuracy on real_data: 98%, fake_data: 98%
>45, 234/234, d_loss=0.111, g_loss=1.871
>Discriminator accuracy on real_data: 96%, fake_data: 92%
>50, 234/234, d_loss=0.088, g_loss=1.977
>Discriminator accuracy on real_data: 94%, fake_data: 99%
```

According to the results above, batch-normalization results in more stable training of the model.

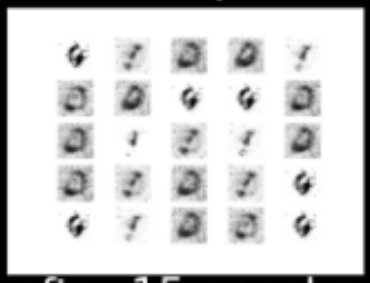
Discriminator accuracies during training are as follow:



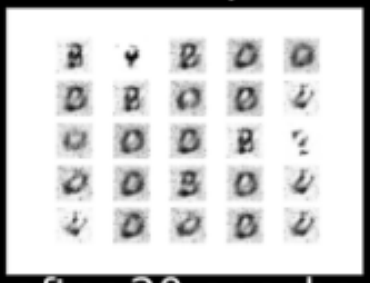
And the output of the model after 50 epochs are as follows:

batch-normalization addition

after 5 epochs



after 10 epochs



after 15 epochs

after 20 epochs

after 25 epochs

after 30 epochs

after 35 epochs

after 40 epochs

after 45 epochs

after 50 epochs

As it can be figured out from the figure above, after 50 epochs, the model has generated less favorable and interpretable outputs but more generalized and smoother outputs compared to the base model.

“addition of dropout layer”

For this experiment, dropout layers with rate of 0.2 was added just after activation layers in the generator model but all other properties were kept unchanged.

```
def define_generator(latent_dim):
  generator = Sequential()
  init = RandomNormal(mean=0.0, stddev=0.02)

  generator.add(Dense(256, input_dim=latent_dim,kernel_initializer=init))
  generator.add(LeakyReLU(0.2))
  generator.add(Dropout(0.2))

  generator.add(Dense(512,kernel_initializer=init))
  generator.add(LeakyReLU(0.2))
  generator.add(Dropout(0.2))

  generator.add(Dense(1024,kernel_initializer=init))
  generator.add(LeakyReLU(0.2))
  generator.add(Dropout(0.2))

  generator.add(Dense(28*28, activation='tanh',kernel_initializer=init))
  generator.add(Reshape((28, 28, 1)))
  return generator
```

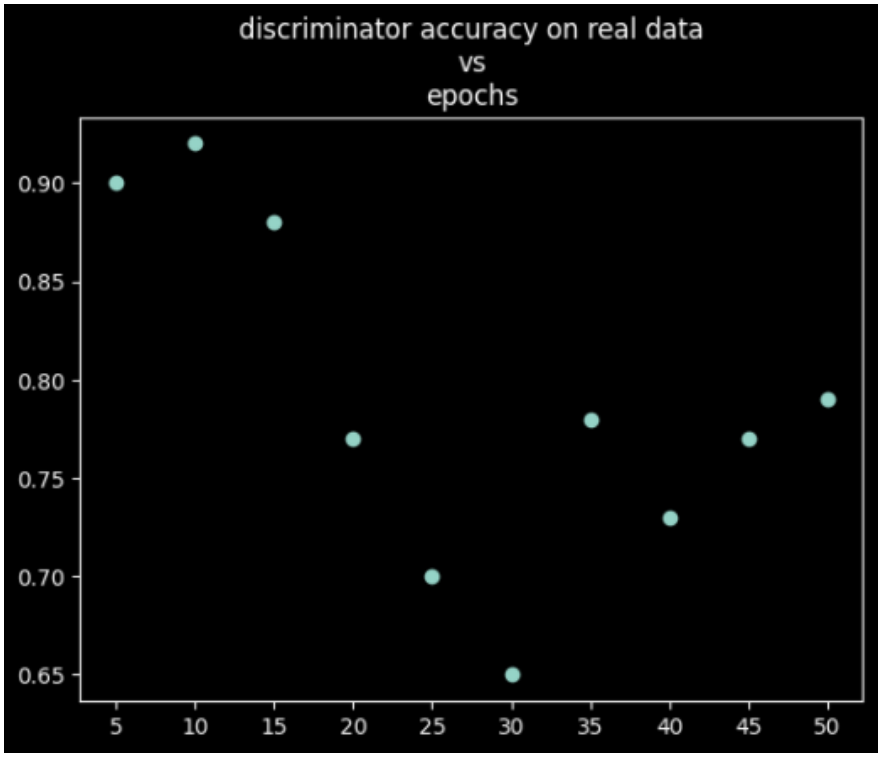
In total, three dropout layers were added to the base model.

The results after this modification are as follow:

Accuracies and losses of all models during training are listed below:

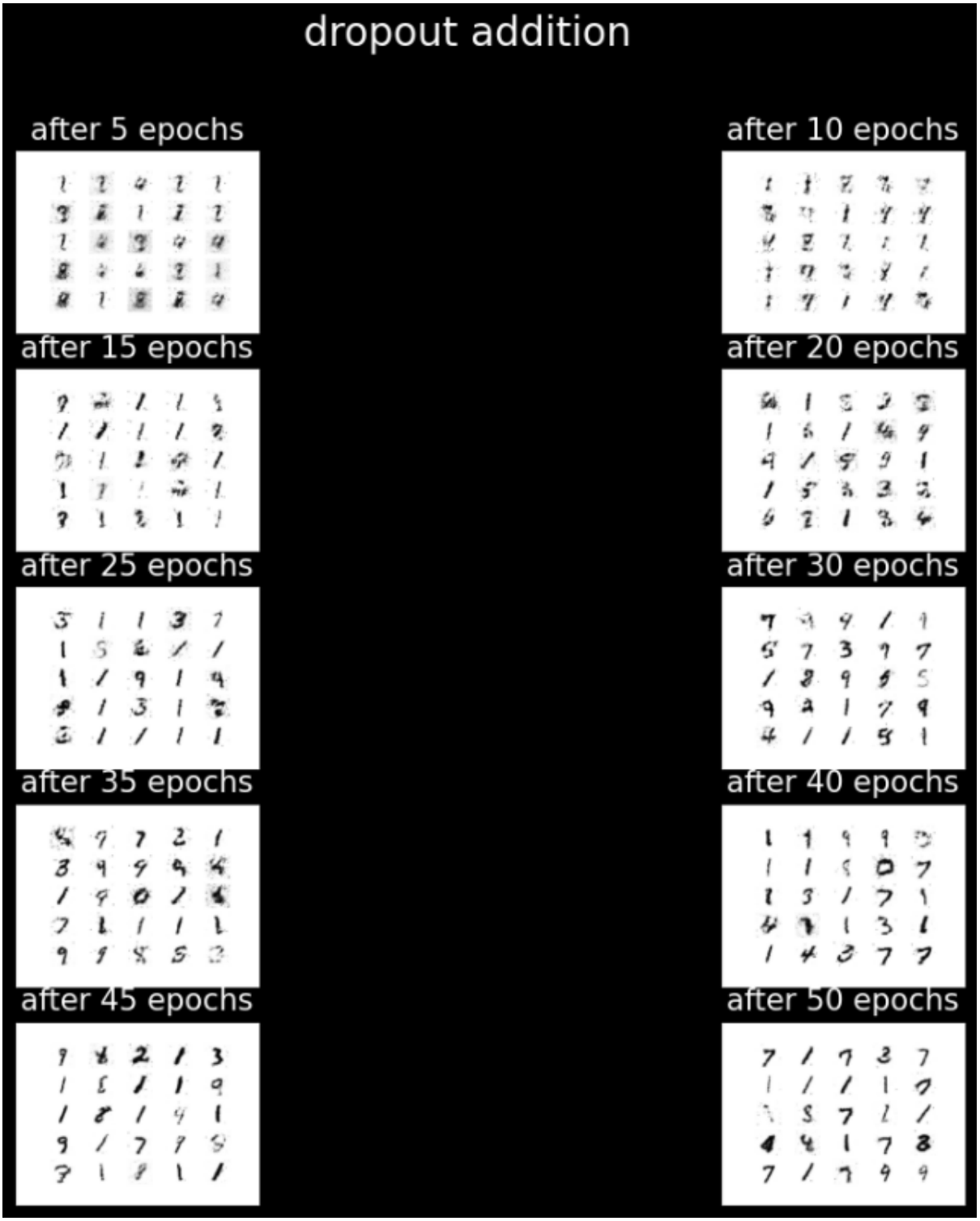
```
>5, 234/234, d_loss=0.150, g_loss=3.807
>Discriminator accuracy on real_data: 90%, fake_data: 100%
>10, 234/234, d_loss=0.181, g_loss=3.420
>Discriminator accuracy on real_data: 92%, fake_data: 100%
>15, 234/234, d_loss=0.365, g_loss=3.321
>Discriminator accuracy on real_data: 88%, fake_data: 99%
>20, 234/234, d_loss=0.301, g_loss=1.827
>Discriminator accuracy on real_data: 77%, fake_data: 97%
>25, 234/234, d_loss=0.403, g_loss=2.105
>Discriminator accuracy on real_data: 70%, fake_data: 98%
>30, 234/234, d_loss=0.426, g_loss=2.134
>Discriminator accuracy on real_data: 65%, fake_data: 98%
>35, 234/234, d_loss=0.333, g_loss=1.673
>Discriminator accuracy on real_data: 78%, fake_data: 95%
>40, 234/234, d_loss=0.430, g_loss=1.842
>Discriminator accuracy on real_data: 73%, fake_data: 85%
>45, 234/234, d_loss=0.434, g_loss=1.608
>Discriminator accuracy on real_data: 77%, fake_data: 90%
>50, 234/234, d_loss=0.409, g_loss=1.600
>Discriminator accuracy on real_data: 79%, fake_data: 89%
```

Discriminator accuracies during training are as follow:



As we can see in the plot above, training procedure is more stable than base model.

And the output of the model after 50 epochs are as follows:



As it can be obviously figured out from the figure above, after 50 epochs, the model has generated less generalized but more

favorable and smoother outputs compared to both previous models.

“addition of dropout + batch-normalization layer”

For this experiment, both dropout and batch-normalization layers were added to the base model with the following configurations:

```
def define_generator(latent_dim):
  generator = Sequential()
  init = RandomNormal(mean=0.0, stddev=0.02)

  generator.add(Dense(256, input_dim=latent_dim,kernel_initializer=init))
  generator.add(BatchNormalization())
  generator.add(LeakyReLU(0.2))
  generator.add(Dropout(0.2))

  generator.add(Dense(512,kernel_initializer=init))
  generator.add(BatchNormalization())
  generator.add(LeakyReLU(0.2))
  generator.add(Dropout(0.2))

  generator.add(Dense(1024,kernel_initializer=init))
  generator.add(BatchNormalization())
  generator.add(LeakyReLU(0.2))
  generator.add(Dropout(0.2))

  generator.add(Dense(28*28, activation='tanh',kernel_initializer=init))
  generator.add(Reshape((28, 28, 1)))
  return generator
```

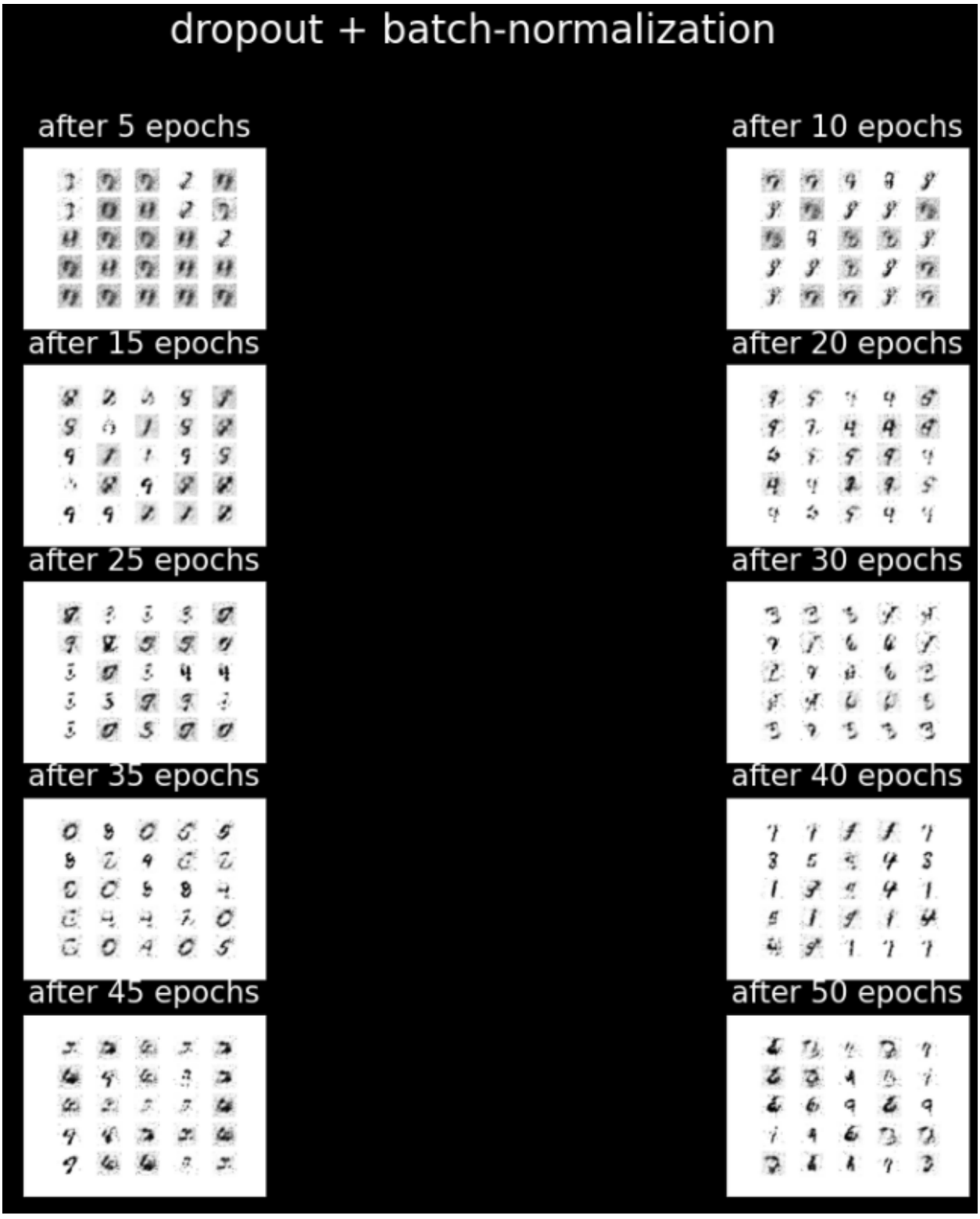
The results after this modification are as follow:

Accuracies and losses of all models during training are listed below:

```
>5, 234/234, d_loss=0.233, g_loss=0.245
>Discriminator accuracy on real_data: 100%, fake_data: 58%
>10, 234/234, d_loss=0.101, g_loss=0.818
>Discriminator accuracy on real_data: 94%, fake_data: 97%
>15, 234/234, d_loss=0.148, g_loss=1.242
>Discriminator accuracy on real_data: 93%, fake_data: 100%
>20, 234/234, d_loss=0.126, g_loss=1.040
>Discriminator accuracy on real_data: 88%, fake_data: 98%
>25, 234/234, d_loss=0.164, g_loss=1.120
>Discriminator accuracy on real_data: 97%, fake_data: 89%
>30, 234/234, d_loss=0.122, g_loss=2.394
>Discriminator accuracy on real_data: 89%, fake_data: 100%
>35, 234/234, d_loss=0.184, g_loss=2.670
>Discriminator accuracy on real_data: 94%, fake_data: 100%
>40, 234/234, d_loss=0.128, g_loss=2.867
>Discriminator accuracy on real_data: 97%, fake_data: 100%
>45, 234/234, d_loss=0.117, g_loss=2.734
>Discriminator accuracy on real_data: 91%, fake_data: 98%
>50, 234/234, d_loss=0.092, g_loss=2.916
>Discriminator accuracy on real_data: 92%, fake_data: 97%
```

As we can see in the plot above, training procedure is less stable than previous models.

And the output of the model after 50 epochs are as follows:


dropout + batch-normalization
after 5 epochs
after 10 epochs
after 15 epochs
after 20 epochs
after 25 epochs
after 30 epochs
after 35 epochs
after 40 epochs
after 45 epochs
after 50 epochs

It seems that after 50 epochs, the model has generated less favorable and interpretable outputs compared to all previous models. This may be because the model has faced generalization more than it is required.

“addition of Dense layers”

Now we are going to evaluate the model with addition of dense layers with addition of one more dense layer.

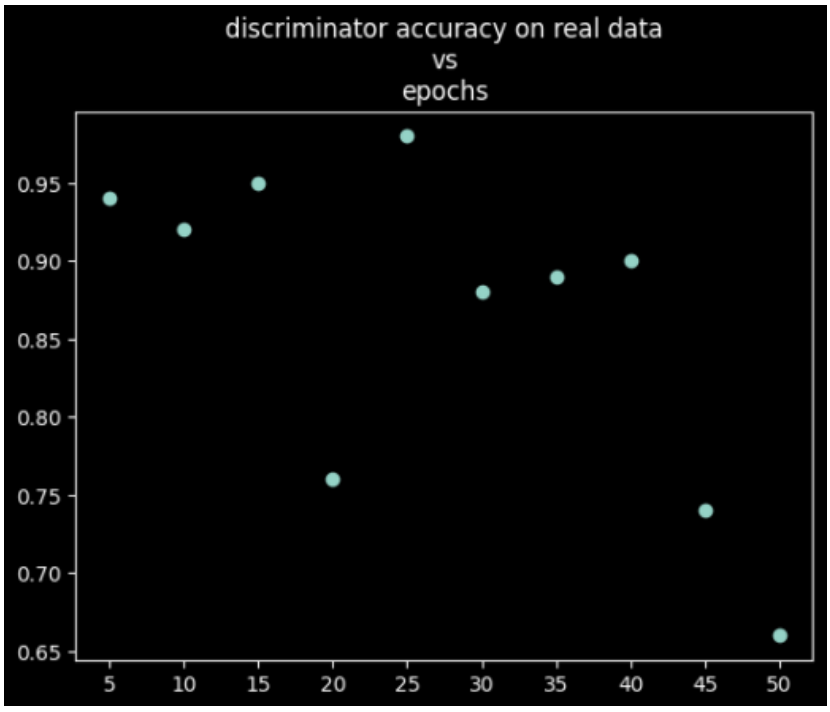
```
def define_generator(latent_dim):
  generator = Sequential()
  init = RandomNormal(mean=0.0, stddev=0.02)

  generator.add(Dense(128, input_dim=latent_dim,kernel_initializer=init))
  generator.add(LeakyReLU(0.2))

  generator.add(Dense(256,kernel_initializer=init))
  generator.add(LeakyReLU(0.2))

  generator.add(Dense(512,kernel_initializer=init))
  generator.add(LeakyReLU(0.2))

  generator.add(Dense(1024,kernel_initializer=init))
  generator.add(LeakyReLU(0.2))

  generator.add(Dense(28*28, activation='tanh',kernel_initializer=init))
  generator.add(Reshape((28, 28, 1)))
  return generator
```

The results after this modification are as follow:

Accuracies and losses of all models during training are listed below:

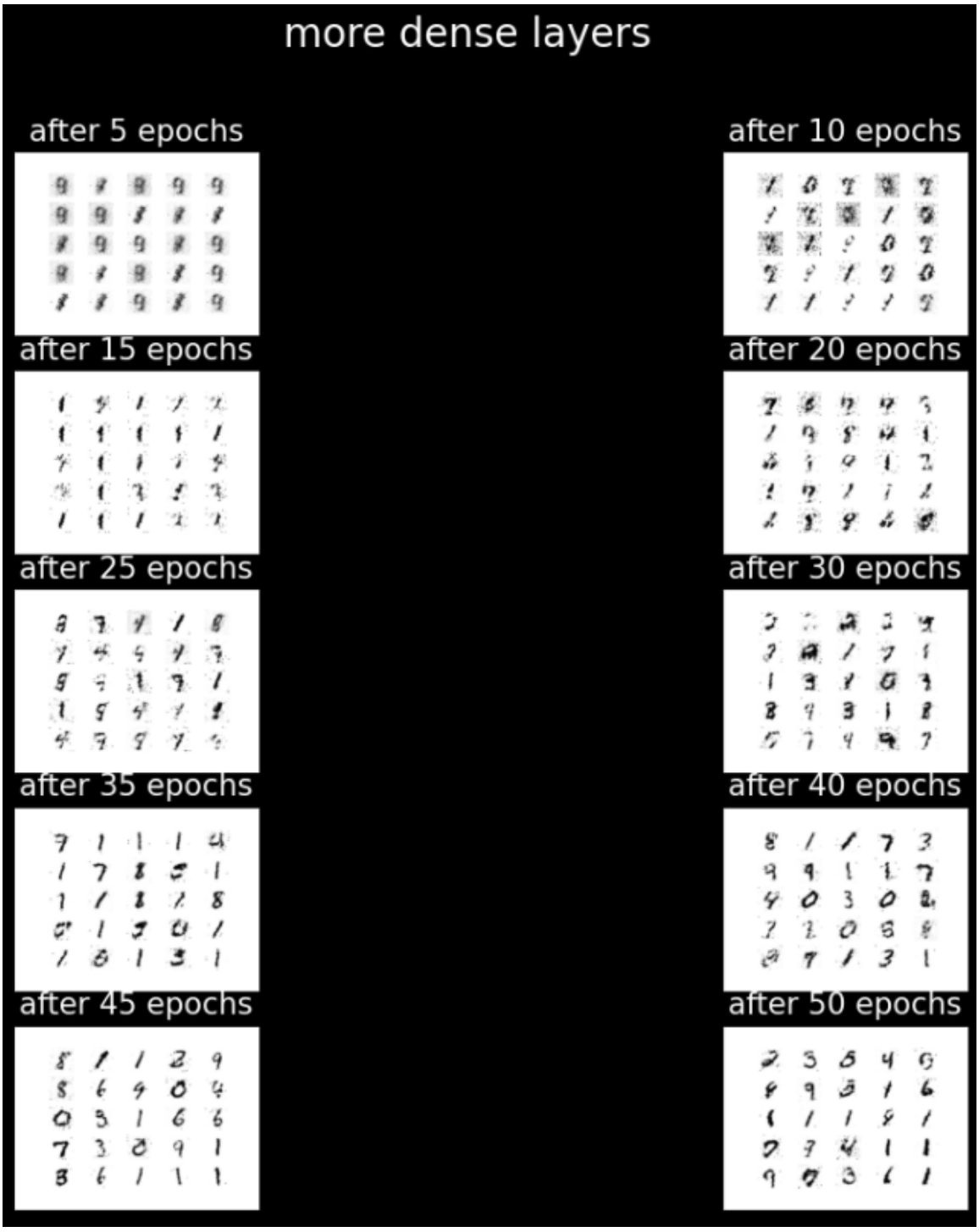
```
>5, 234/234, d_loss=0.194, g_loss=2.218
>Discriminator accuracy on real_data: 94%, fake_data: 99%
>10, 234/234, d_loss=0.750, g_loss=2.797
>Discriminator accuracy on real_data: 92%, fake_data: 98%
>15, 234/234, d_loss=0.108, g_loss=2.747
>Discriminator accuracy on real_data: 95%, fake_data: 99%
>20, 234/234, d_loss=0.636, g_loss=2.853
>Discriminator accuracy on real_data: 76%, fake_data: 95%
>25, 234/234, d_loss=0.282, g_loss=1.564
>Discriminator accuracy on real_data: 98%, fake_data: 92%
>30, 234/234, d_loss=0.419, g_loss=1.262
>Discriminator accuracy on real_data: 88%, fake_data: 73%
>35, 234/234, d_loss=0.419, g_loss=1.193
>Discriminator accuracy on real_data: 89%, fake_data: 72%
>40, 234/234, d_loss=0.442, g_loss=0.968
>Discriminator accuracy on real_data: 90%, fake_data: 60%
>45, 234/234, d_loss=0.427, g_loss=1.761
>Discriminator accuracy on real_data: 74%, fake_data: 82%
>50, 234/234, d_loss=0.370, g_loss=1.784
>Discriminator accuracy on real_data: 66%, fake_data: 97%
```

Discriminator accuracies during training are as follow:



As we can see in the plot above, training procedure is much more stable than all previous models.

And the output of the model after 50 epochs are as follows:



As it can be figured out from the figure above, after 50 epochs, the model has generated more generalized, realistic and favorable outputs compared to all previous models.

Considering all experiments above, it can be figured out that change in number of layers may improve or weaken the performance of the GAN model depending on type of the layer to be added. In the case of FCGAN, increase in dropout and dense layers may improve the performance of GAN model.

3.

For the experiments of this problem, all evaluations will be studied on the base model.

It should be noted that noises were added to images using built-in layer of keras named as ‘GaussianNoise’ with the default mean of 0 and specified standard deviation which in fact controls noise intensity. This layer is added at the bottom of the discriminator which is shown in the figure below:

```
def define_discriminator(in_shape=(28,28,1)):
  discriminator = Sequential()
  discriminator.add(GaussianNoise(0.1,input_shape=in_shape))
  init = RandomNormal(mean=0.0,stddev=0.02)
  discriminator.add(Flatten)

  discriminator.add(Dense(1024,kernel_initializer=init))
  discriminator.add(LeakyReLU(0.2))

  discriminator.add(Dense(512,kernel_initializer=init))
  discriminator.add(LeakyReLU(0.2))

  discriminator.add(Dense(256,kernel_initializer=init))
  discriminator.add(LeakyReLU(0.2))

  discriminator.add(Dense(1, activation='sigmoid',kernel_initializer=init))
  opt = Adam(lr=0.0002, beta_1=0.5)
  discriminator.compile(loss='binary_crossentropy', optimizer=opt, metrics=['accuracy'])
  return discriminator
```

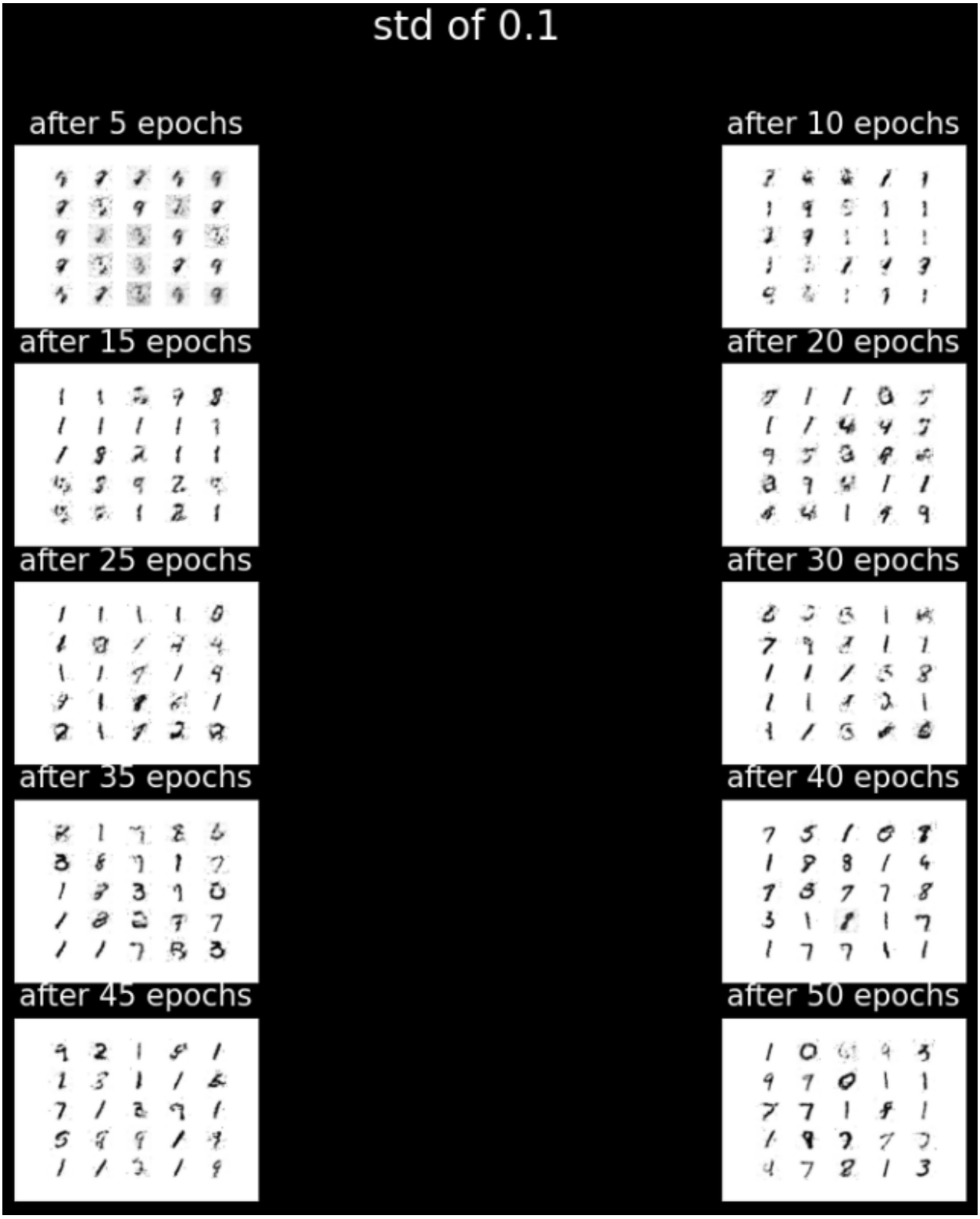
This evaluation will be performed using standard deviation values of 0.1, 0.5, and 2.

(For this part, only accuracies and losses during training along with generated images for each value will be shown and the conclusions will be brought at the end)

Stddev of 0.1:

```
>5, 234/234, d_loss=0.242, g_loss=3.421
>Discriminator accuracy on real_data: 82%, fake_data: 100%
>10, 234/234, d_loss=0.368, g_loss=2.404
>Discriminator accuracy on real_data: 91%, fake_data: 85%
>15, 234/234, d_loss=0.393, g_loss=1.360
>Discriminator accuracy on real_data: 93%, fake_data: 84%
>20, 234/234, d_loss=0.346, g_loss=1.250
>Discriminator accuracy on real_data: 90%, fake_data: 74%
>25, 234/234, d_loss=0.399, g_loss=2.389
>Discriminator accuracy on real_data: 69%, fake_data: 98%
>30, 234/234, d_loss=0.378, g_loss=1.644
>Discriminator accuracy on real_data: 81%, fake_data: 90%
>35, 234/234, d_loss=0.504, g_loss=2.629
>Discriminator accuracy on real_data: 50%, fake_data: 100%
>40, 234/234, d_loss=0.405, g_loss=1.680
>Discriminator accuracy on real_data: 76%, fake_data: 89%
>45, 234/234, d_loss=0.395, g_loss=1.708
>Discriminator accuracy on real_data: 81%, fake_data: 94%
>50, 234/234, d_loss=0.439, g_loss=1.677
>Discriminator accuracy on real_data: 83%, fake_data: 88%
```

According to the results shown above, the deviation between discriminator and generator losses has decreased; therefore, increase in noise has improved stability of the training and balance between discriminator and generator.

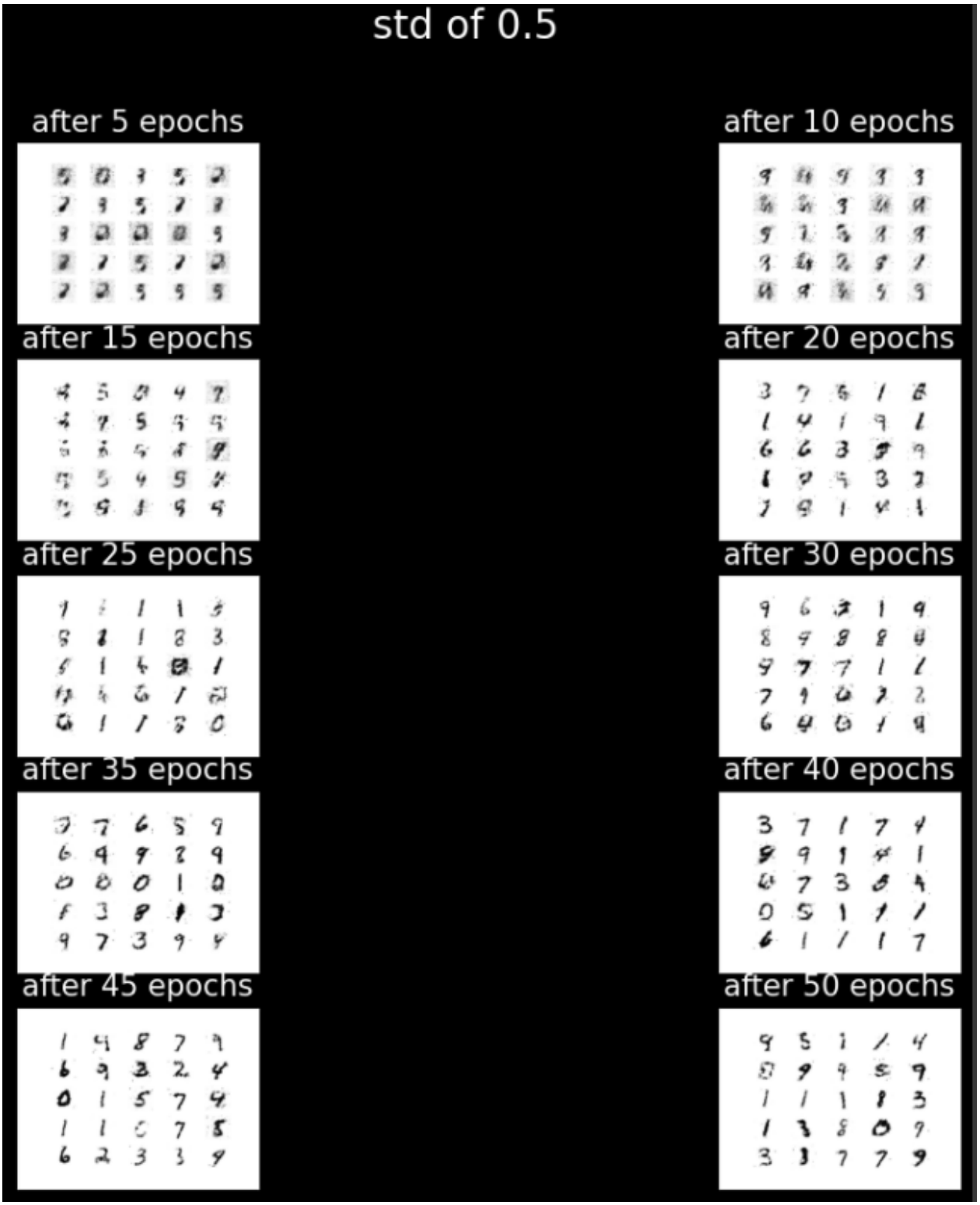


compared to the base model, it seems that the outputs have slightly improved in both quality and in the balance between

discriminator and generator losses according to the two figures above.

Stddev of 0.5:

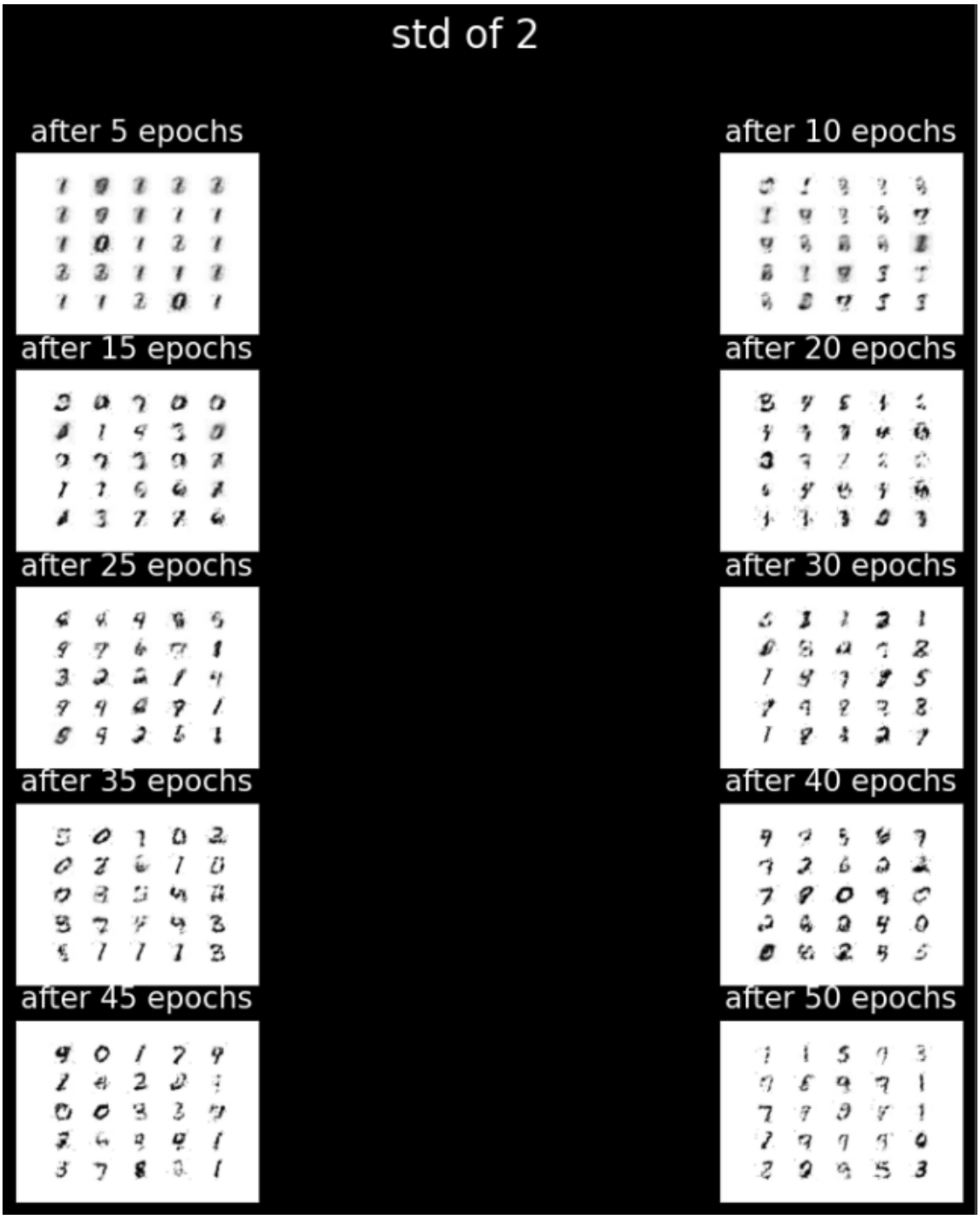
```
>5, 234/234, d_loss=0.339, g_loss=2.490
>Discriminator accuracy on real_data: 70%, fake_data: 98%
>10, 234/234, d_loss=0.495, g_loss=2.090
>Discriminator accuracy on real_data: 71%, fake_data: 100%
>15, 234/234, d_loss=0.308, g_loss=1.665
>Discriminator accuracy on real_data: 94%, fake_data: 92%
>20, 234/234, d_loss=0.459, g_loss=1.628
>Discriminator accuracy on real_data: 71%, fake_data: 94%
>25, 234/234, d_loss=0.515, g_loss=1.103
>Discriminator accuracy on real_data: 81%, fake_data: 71%
>30, 234/234, d_loss=0.558, g_loss=1.301
>Discriminator accuracy on real_data: 77%, fake_data: 77%
>35, 234/234, d_loss=0.563, g_loss=1.018
>Discriminator accuracy on real_data: 82%, fake_data: 52%
>40, 234/234, d_loss=0.619, g_loss=1.420
>Discriminator accuracy on real_data: 68%, fake_data: 87%
>45, 234/234, d_loss=0.563, g_loss=1.277
>Discriminator accuracy on real_data: 60%, fake_data: 75%
>50, 234/234, d_loss=0.528, g_loss=1.194
>Discriminator accuracy on real_data: 78%, fake_data: 77%
```



The results for this value is similar to that of the value of 0.1.

Stddev of 2:

```
>5, 234/234, d_loss=0.543, g_loss=1.203
>Discriminator accuracy on real_data: 62%, fake_data: 100%
>10, 234/234, d_loss=0.643, g_loss=0.893
>Discriminator accuracy on real_data: 71%, fake_data: 71%
>15, 234/234, d_loss=0.663, g_loss=0.986
>Discriminator accuracy on real_data: 62%, fake_data: 78%
>20, 234/234, d_loss=0.686, g_loss=0.840
>Discriminator accuracy on real_data: 74%, fake_data: 67%
>25, 234/234, d_loss=0.630, g_loss=0.762
>Discriminator accuracy on real_data: 67%, fake_data: 70%
>30, 234/234, d_loss=0.666, g_loss=0.839
>Discriminator accuracy on real_data: 55%, fake_data: 80%
>35, 234/234, d_loss=0.689, g_loss=0.726
>Discriminator accuracy on real_data: 77%, fake_data: 48%
>40, 234/234, d_loss=0.665, g_loss=0.743
>Discriminator accuracy on real_data: 72%, fake_data: 48%
>45, 234/234, d_loss=0.670, g_loss=0.784
>Discriminator accuracy on real_data: 66%, fake_data: 60%
>50, 234/234, d_loss=0.664, g_loss=0.750
>Discriminator accuracy on real_data: 45%, fake_data: 82%
```



Results of the value of 2 shows decrease in the performance of the model in both quality of the outputs and losses and accuracies of the models. From this we can interpret that increasing in noise intensity to a certain level improves the performance and then it starts to degrade.

4.

100 generated fake images using ‘Dense Model’:

END of FCGAN

5.

“DCGAN”

2.

‘Evaluation on number of layers’

For this evaluation, several experiments on different types of layers were conducted in order to find a plausible GAN model the results of which are as follows.

It should be noted that, initially a base model will be defined and all other modifications will be applied on that with the intention of a fairer comparison among different models.

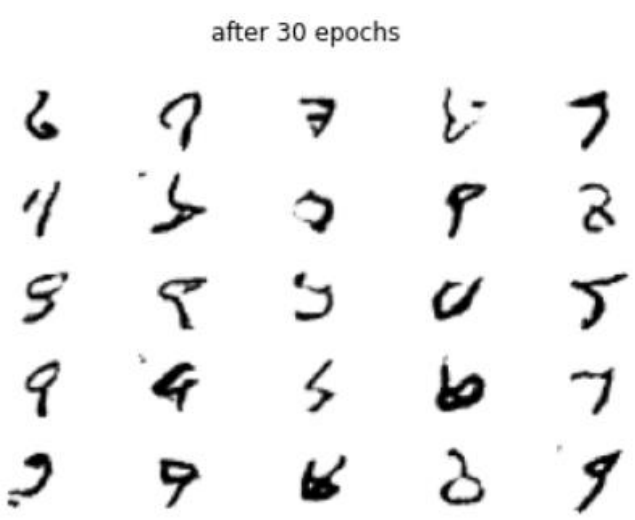
The base model properties are as follows:

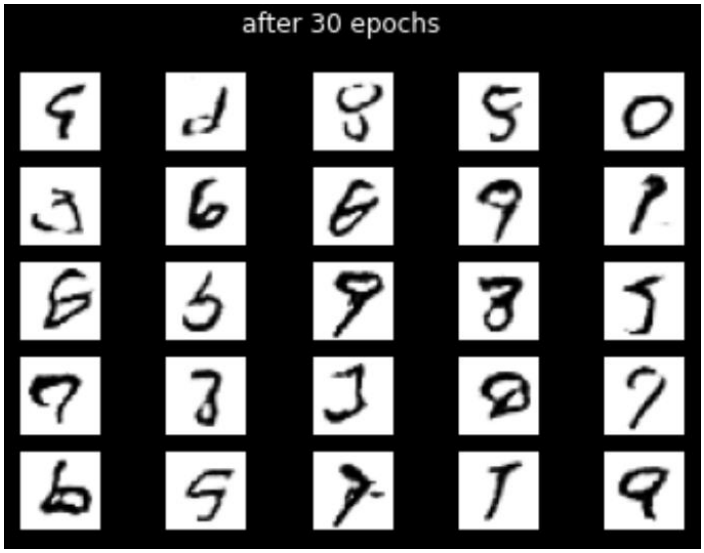
(It should be noted that these selected properties are based on the latest suggestions for DCGAN models according to recent papers)

9. No pooling layers were used. Instead, downsampling was implemented using strided convolutions of size (2,2)
10. Also, upsampling was implemented exactly the same as downsampling.
11. Contrary to the past, recently, LeakyReLU activation function with parameter of 0.2 is suggested instead of Relu or other activations.
12. No batch normalization layer was considered for base model as it is evaluated in the following experiments.
13. Gaussian weigh initialization was considered for all layers. (with zero mean and 0.02 standard deviation)

14. Adam optimizer with 0.0002 learning rate and 0.5 beta was considered.
15. All images were normalized to the range of [-1 , 1]; therefore, activation function of ‘tanh’ was considered for the output layer of generator.
16. Gaussian distribution was considered for latent space.
17. Discriminator model is trained on real and fake images separately.
18. Also, it should be noted that contrary to the suggestions of using smoothed labels, here, hard labels were preferred since it seemed that it resulted in more favorable outputs. Results of this experiment is as follows:

    (left one was obtained with smooth labels and right one with hard labels which seems much more favorable.)





The only main properties which are not mentioned are number of layers (such as conv and convtranspose and batch normalization and dropout layers) and number of filters of convolutional layers which will be evaluated in the following experiments.

(All models are obtained with batch size of 256 and selection of best model within 30 epochs of training.)

“base model”

Base model configuration:

Base models’ discriminator consists of convolution, LeakyReLU, and Drop out layers with one Flatten and Dense layers on the top.

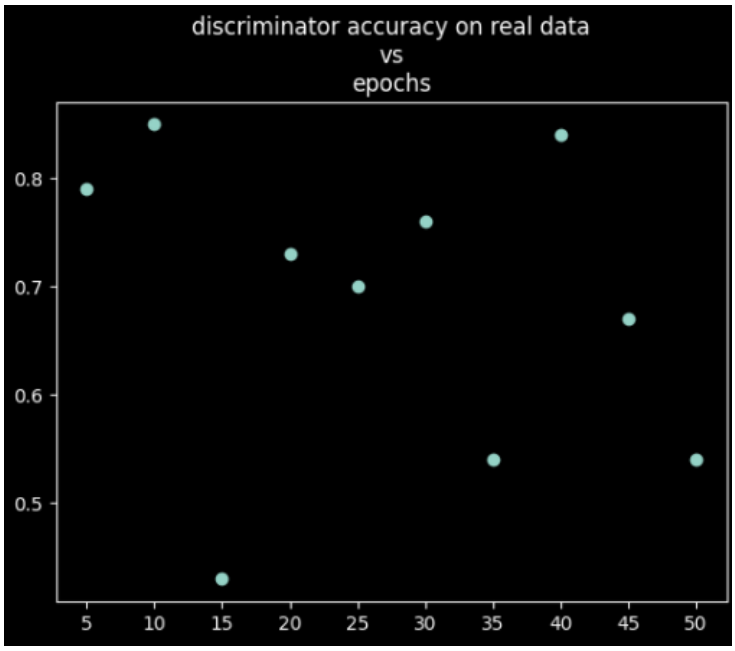
Base models’ generator consists of convolution, LeakyReLU, and layers with one Reshap and Dense layers at the bottom.

And lastly, the GAN model is defined by stacking discriminator and generator models with turning trainability of discriminator to false.

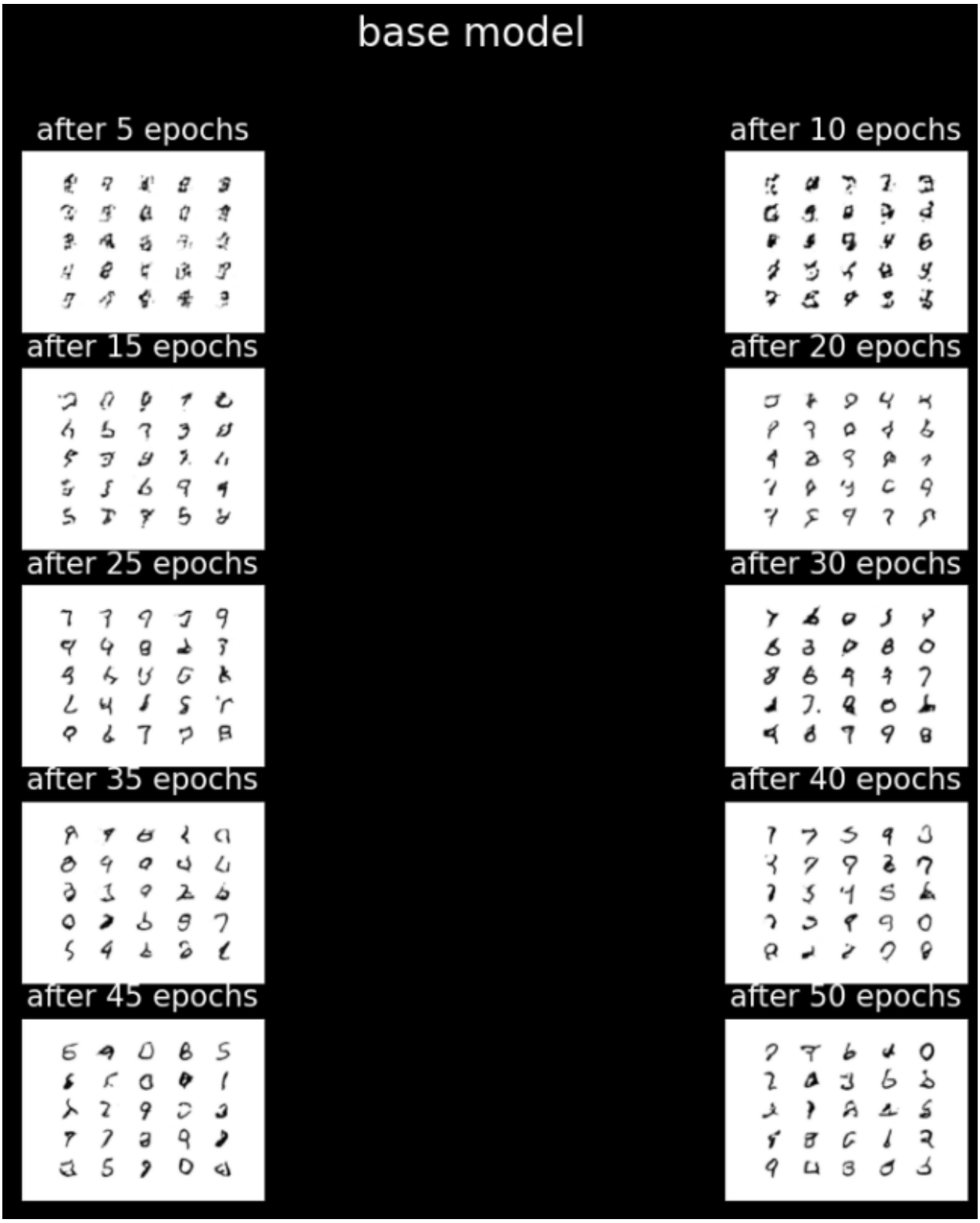
```python
def define_discriminator(in_shape=(28,28,1)):
  model = Sequential()
  init = RandomNormal(mean=0.0, stddev=0.02)
  model.add(Conv2D(64, (3,3), strides=(2, 2), padding='same',kernel_initializer=init, input_shape=in_shape))
  model.add(LeakyReLU(alpha=0.2))
  model.add(Dropout(0.2))
  model.add(Conv2D(64, (3,3), strides=(2, 2), padding='same',kernel_initializer=init))
  model.add(LeakyReLU(alpha=0.2))
  model.add(Dropout(0.2))
  model.add(Flatten())
  model.add(Dense(1, activation='sigmoid',kernel_initializer=init))
  opt = Adam(lr=0.0002, beta_1=0.5)
  model.compile(loss='binary_crossentropy', optimizer=opt, metrics=['accuracy'])
  return model

def generator(latent_dim):
  model = Sequential()
  init = RandomNormal(mean=0.0, stddev=0.02)
  n_nodes = 64 * 7 * 7
  model.add(Dense(n_nodes, input_dim=latent_dim))
  model.add(LeakyReLU(alpha=0.2))
  model.add(Reshape((7, 7, 64)))
  model.add(Conv2DTranspose(64, (4,4), strides=(2,2), padding='same',kernel_initializer=init))
  model.add(LeakyReLU(alpha=0.2))
  model.add(Conv2DTranspose(64, (4,4), strides=(2,2), padding='same',kernel_initializer=init))
  model.add(LeakyReLU(alpha=0.2))
  model.add(Conv2D(1, (3,3), activation='tanh', padding='same',kernel_initializer=init))
  return model

def define_gan(g_model, d_model):
  d_model.trainable = False
  model = Sequential()
  model.add(g_model)
  model.add(d_model)
  opt = Adam(lr=0.0002, beta_1=0.5)
  model.compile(loss='binary_crossentropy', optimizer=opt)
  return model
```

Discriminator accuracies during training are as follow:

```
>5, 234/234, d_loss=0.558, g_loss=0.869
>Discriminator accuracy on real_data: 79%, fake_data: 92%
>10, 234/234, d_loss=0.616, g_loss=0.809
>Discriminator accuracy on real_data: 85%, fake_data: 75%
>15, 234/234, d_loss=0.665, g_loss=0.825
>Discriminator accuracy on real_data: 43%, fake_data: 76%
>20, 234/234, d_loss=0.631, g_loss=0.807
>Discriminator accuracy on real_data: 73%, fake_data: 81%
>25, 234/234, d_loss=0.651, g_loss=0.786
>Discriminator accuracy on real_data: 70%, fake_data: 74%
>30, 234/234, d_loss=0.649, g_loss=0.781
>Discriminator accuracy on real_data: 76%, fake_data: 64%
>35, 234/234, d_loss=0.667, g_loss=0.810
>Discriminator accuracy on real_data: 54%, fake_data: 73%
>40, 234/234, d_loss=0.663, g_loss=0.746
>Discriminator accuracy on real_data: 84%, fake_data: 51%
>45, 234/234, d_loss=0.686, g_loss=0.742
>Discriminator accuracy on real_data: 67%, fake_data: 63%
>50, 234/234, d_loss=0.692, g_loss=0.759
>Discriminator accuracy on real_data: 54%, fake_data: 70%
```



Output of base model after 30 epochs:



As we can see in the figure above, model has the best output after 30 epochs which is rather consistent with the accuracy of discriminator after 30 epochs in the previous plot.

“addition of batch normalization layer”

For this experiment, batch normalization layer was added just after dense and convolutional layers in the generator model but all other properties were kept unchanged.

```
def generator(latent_dim):
  model = Sequential()
  init = RandomNormal(mean=0.0, stddev=0.02)
  n_nodes = 64 * 7 * 7
  model.add(Dense(n_nodes, input_dim=latent_dim))
  model.add(BatchNormalization())
  model.add(LeakyReLU(alpha=0.2))

  model.add(Reshape((7, 7, 64)))

  model.add(Conv2DTranspose(64, (4,4), strides=(2,2), padding='same',kernel_initializer=init))
  model.add(BatchNormalization())
  model.add(LeakyReLU(alpha=0.2))

  model.add(Conv2DTranspose(64, (4,4), strides=(2,2), padding='same',kernel_initializer=init))
  model.add(BatchNormalization())
  model.add(LeakyReLU(alpha=0.2))

  model.add(Conv2D(1, (3,3), activation='tanh', padding='same',kernel_initializer=init))
  return model
```

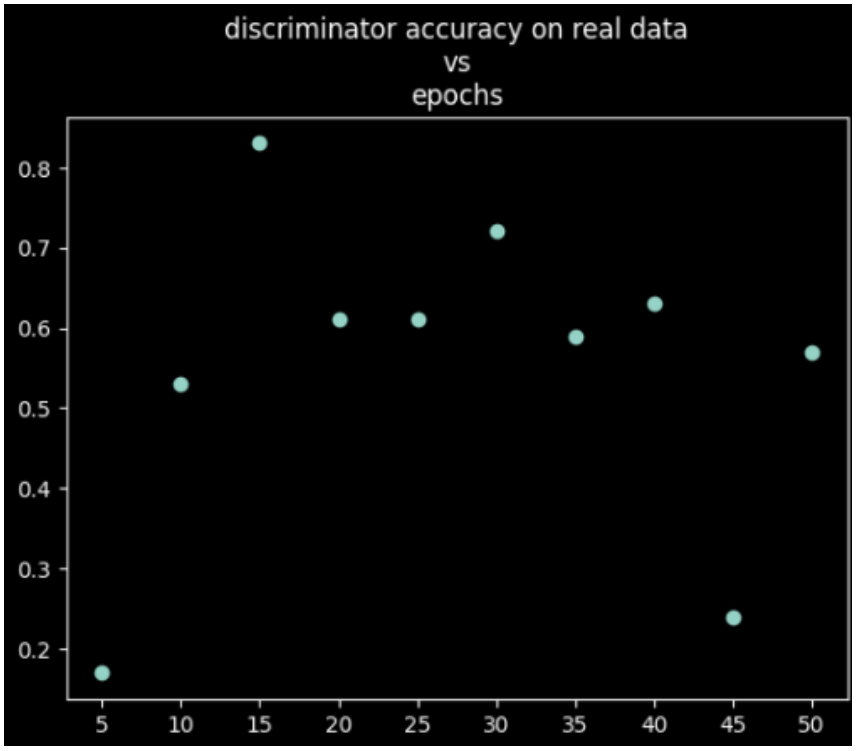
In total, three batch-normalization layers are added to the base model.

The results after this modification are as follow:

Accuracies and losses of all models during training are listed below:

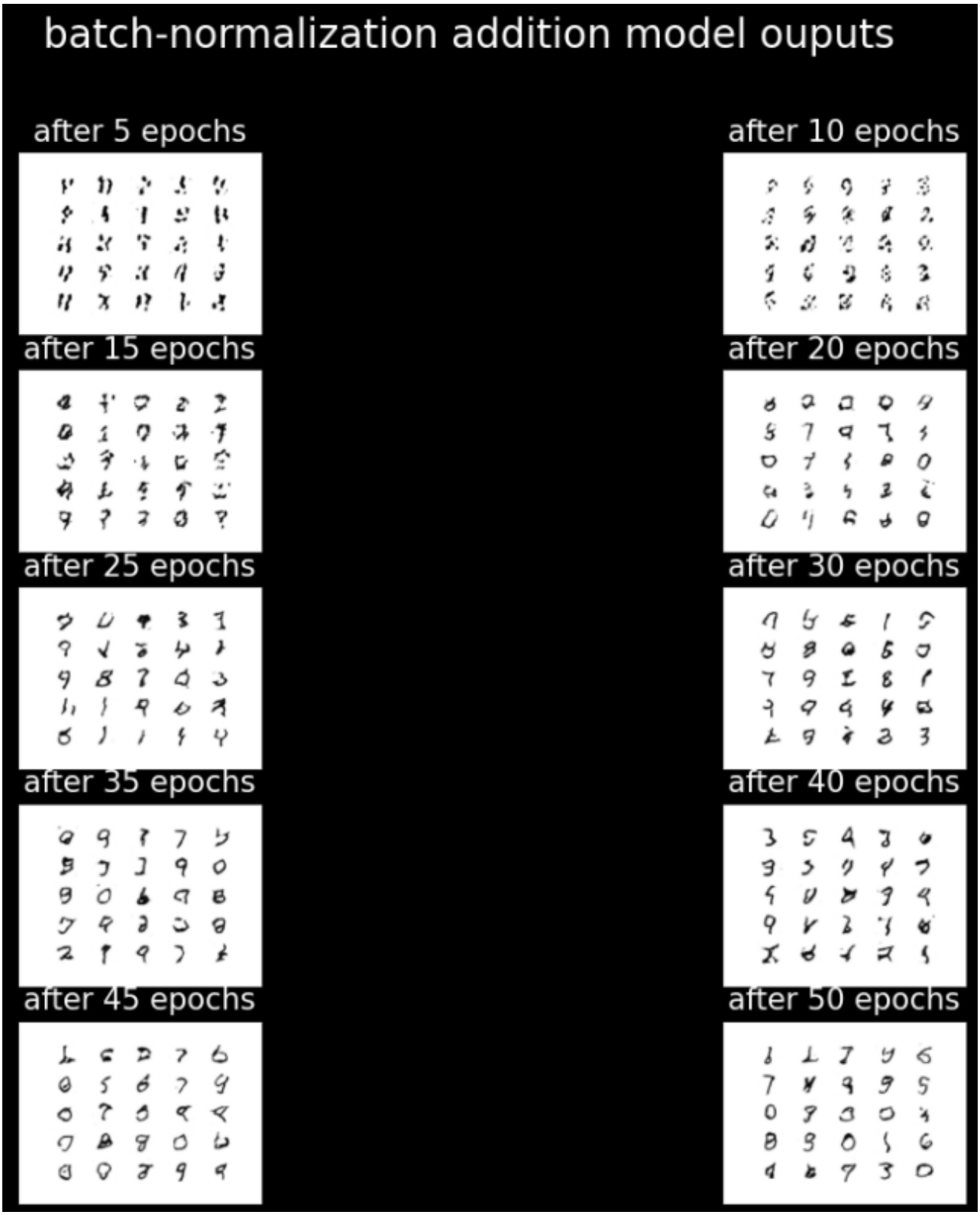
```
>5, 234/234, d_loss=0.731, g_loss=0.677
>Discriminator accuracy on real_data: 17%, fake_data: 26%
>10, 234/234, d_loss=0.681, g_loss=0.720
>Discriminator accuracy on real_data: 53%, fake_data: 84%
>15, 234/234, d_loss=0.703, g_loss=0.677
>Discriminator accuracy on real_data: 83%, fake_data: 19%
>20, 234/234, d_loss=0.688, g_loss=0.720
>Discriminator accuracy on real_data: 61%, fake_data: 52%
>25, 234/234, d_loss=0.684, g_loss=0.726
>Discriminator accuracy on real_data: 61%, fake_data: 62%
>30, 234/234, d_loss=0.687, g_loss=0.736
>Discriminator accuracy on real_data: 72%, fake_data: 40%
>35, 234/234, d_loss=0.670, g_loss=0.719
>Discriminator accuracy on real_data: 59%, fake_data: 44%
>40, 234/234, d_loss=0.672, g_loss=0.730
>Discriminator accuracy on real_data: 63%, fake_data: 55%
>45, 234/234, d_loss=0.691, g_loss=0.783
>Discriminator accuracy on real_data: 24%, fake_data: 99%
>50, 234/234, d_loss=0.688, g_loss=0.729
>Discriminator accuracy on real_data: 57%, fake_data: 56%
```

Discriminator accuracies during training are as follow:



As we can see in the plot above, the training procedure is rather more stable than the base model.

And the output of the model after 50 epochs are as follows:



As it can be obviously figured out from the figure above, after 50 epochs, the model has generated much more realistic and favorable outputs compared to the base model. The generated outputs at 50 epochs are much smoother and more interpretable.

“addition of dropout layer”

For this experiment, dropout layers with rate of 0.2 was added just after activation layers in the generator model but all other properties were kept unchanged.

```
1
2 def generator(latent_dim):
3   model = Sequential()
4   init = RandomNormal(mean=0.0, stddev=0.02)
5   n_nodes = 64 * 7 * 7
6   model.add(Dense(n_nodes, input_dim=latent_dim))
7   model.add(LeakyReLU(alpha=0.2))
8   model.add(Dropout(0.2))
9
10  model.add(Reshape((7, 7, 64)))
11
12  model.add(Conv2DTranspose(64, (4,4), strides=(2,2), padding='same',kernel_initializer=init))
13  model.add(LeakyReLU(alpha=0.2))
14  model.add(Dropout(0.2))
15
16  model.add(Conv2DTranspose(64, (4,4), strides=(2,2), padding='same',kernel_initializer=init))
17  model.add(LeakyReLU(alpha=0.2))
18  model.add(Dropout(0.2))
19
20  model.add(Conv2D(1, (3,3), activation='tanh', padding='same',kernel_initializer=init))
21  return model
22
```

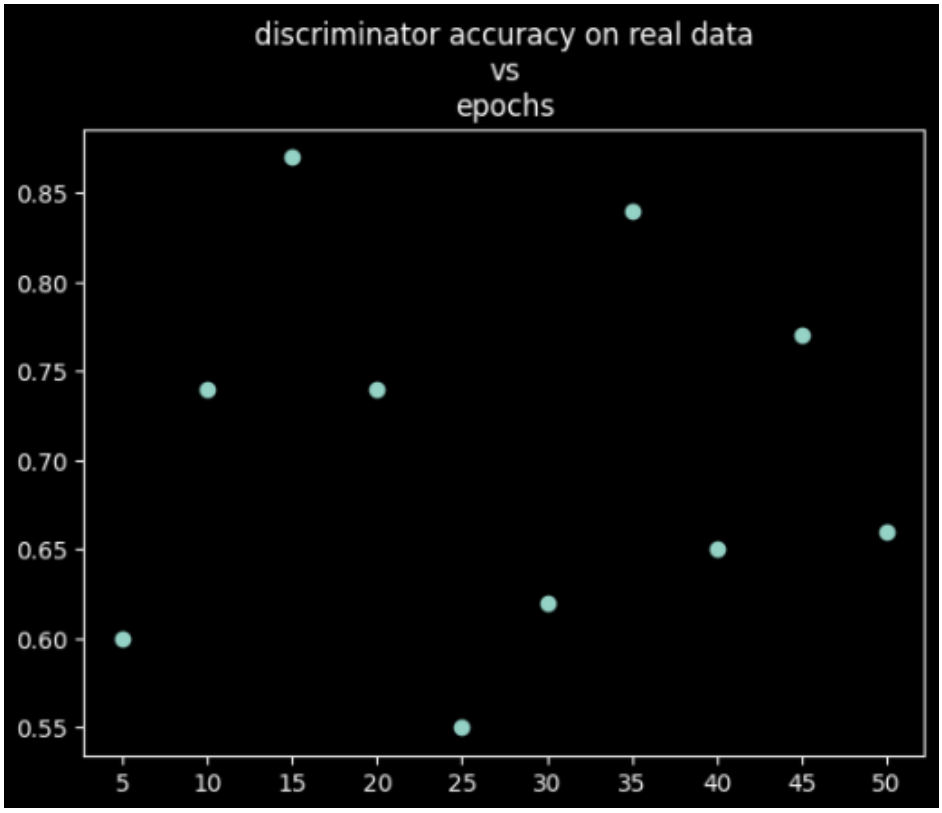
In total, three dropout layers were added to the base model.

The results after this modification are as follow:

Accuracies and losses of all models during training are listed below:

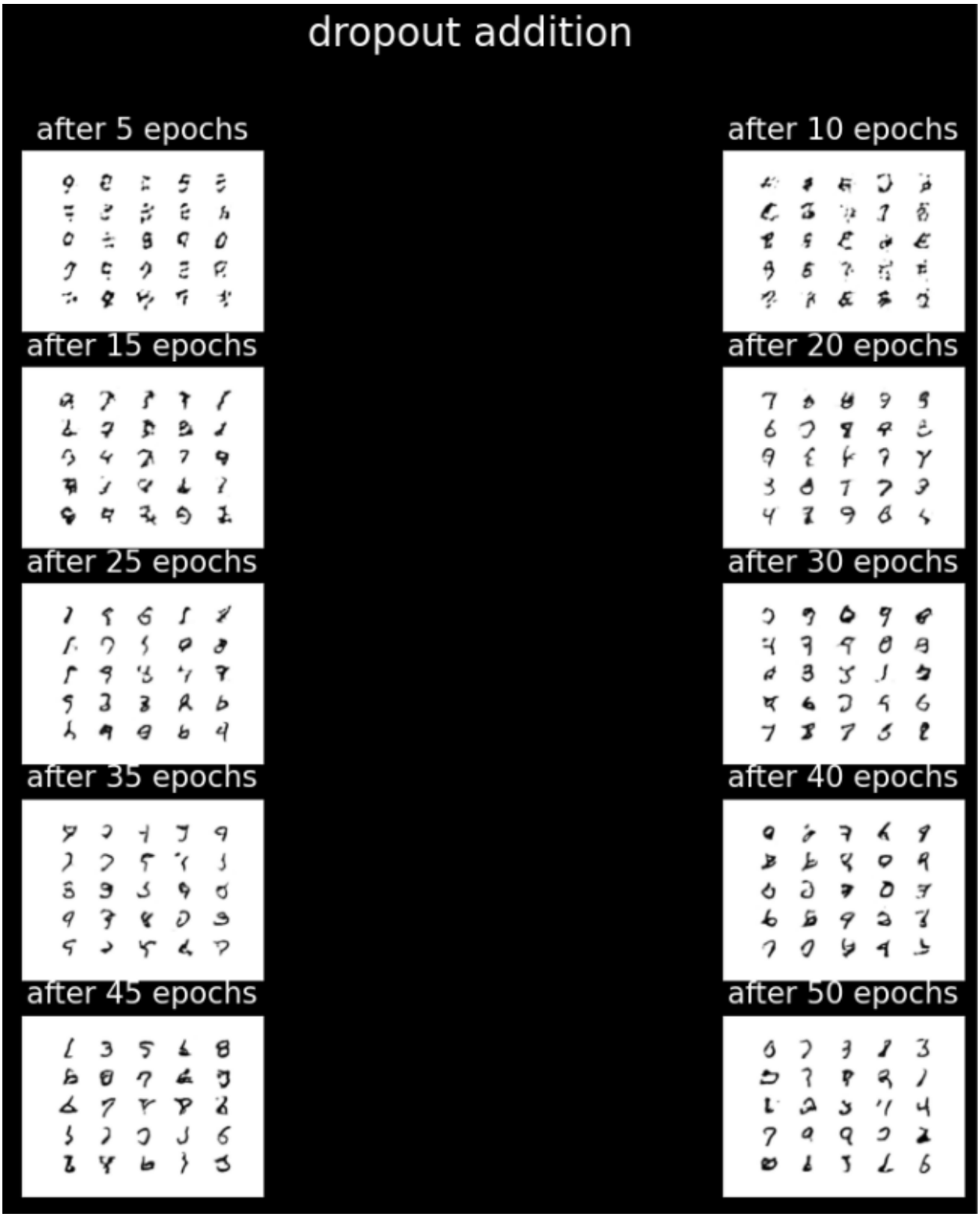
```
>5, 234/234, d_loss=0.665, g_loss=0.767
>Discriminator accuracy on real_data: 60%, fake_data: 55%
>10, 234/234, d_loss=0.672, g_loss=0.743
>Discriminator accuracy on real_data: 74%, fake_data: 41%
>15, 234/234, d_loss=0.647, g_loss=0.752
>Discriminator accuracy on real_data: 87%, fake_data: 69%
>20, 234/234, d_loss=0.622, g_loss=0.843
>Discriminator accuracy on real_data: 74%, fake_data: 75%
>25, 234/234, d_loss=0.628, g_loss=0.923
>Discriminator accuracy on real_data: 55%, fake_data: 89%
>30, 234/234, d_loss=0.646, g_loss=0.875
>Discriminator accuracy on real_data: 62%, fake_data: 66%
>35, 234/234, d_loss=0.636, g_loss=0.845
>Discriminator accuracy on real_data: 84%, fake_data: 67%
>40, 234/234, d_loss=0.652, g_loss=0.874
>Discriminator accuracy on real_data: 65%, fake_data: 89%
>45, 234/234, d_loss=0.626, g_loss=0.840
>Discriminator accuracy on real_data: 77%, fake_data: 70%
>50, 234/234, d_loss=0.669, g_loss=0.895
>Discriminator accuracy on real_data: 66%, fake_data: 77%
```

Discriminator accuracies during training are as follow:



As we can see in the plot above, training procedure is less stable than previous model.

And the output of the model after 50 epochs are as follows:



As it can be obviously figured out from the figure above, after 50 epochs, the model has generated more generalized, realistic and favorable outputs compared to the base model but less compared to previous model (batch-normalization). The generated outputs at 50 epochs are much smoother and more interpretable than base model but again less compared to previous model.

“addition of dropout and batch-normalization layers”

Now we are going to evaluate the model with addition of both dropout and batch-normalization layers simultaneously.

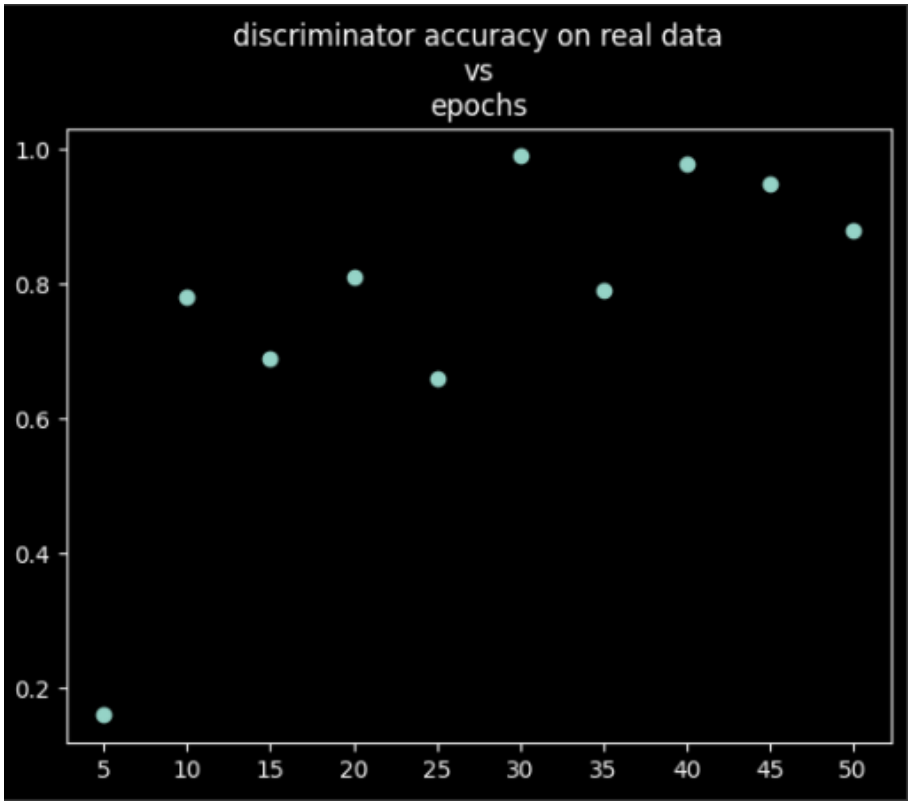
```
1
2 def generator(latent_dim):
3   model = Sequential()
4   init = RandomNormal(mean=0.0, stddev=0.02)
5   n_nodes = 64 * 7 * 7
6   model.add(Dense(n_nodes, input_dim=latent_dim))
7   model.add(BatchNormalization())
8   model.add(LeakyReLU(alpha=0.2))
9   model.add(Dropout(0.2))
10
11  model.add(Reshape((7, 7, 64)))
12
13  model.add(Conv2DTranspose(64, (4,4), strides=(2,2), padding='same',kernel_initializer=init))
14  model.add(BatchNormalization())
15  model.add(LeakyReLU(alpha=0.2))
16  model.add(Dropout(0.2))
17
18  model.add(Conv2DTranspose(64, (4,4), strides=(2,2), padding='same',kernel_initializer=init))
19  model.add(BatchNormalization())
20  model.add(LeakyReLU(alpha=0.2))
21  model.add(Dropout(0.2))
22
23  model.add(Conv2D(1, (3,3), activation='tanh', padding='same',kernel_initializer=init))
24  return model
25
```

The results after this modification are as follow:

Accuracies and losses of all models during training are listed below:

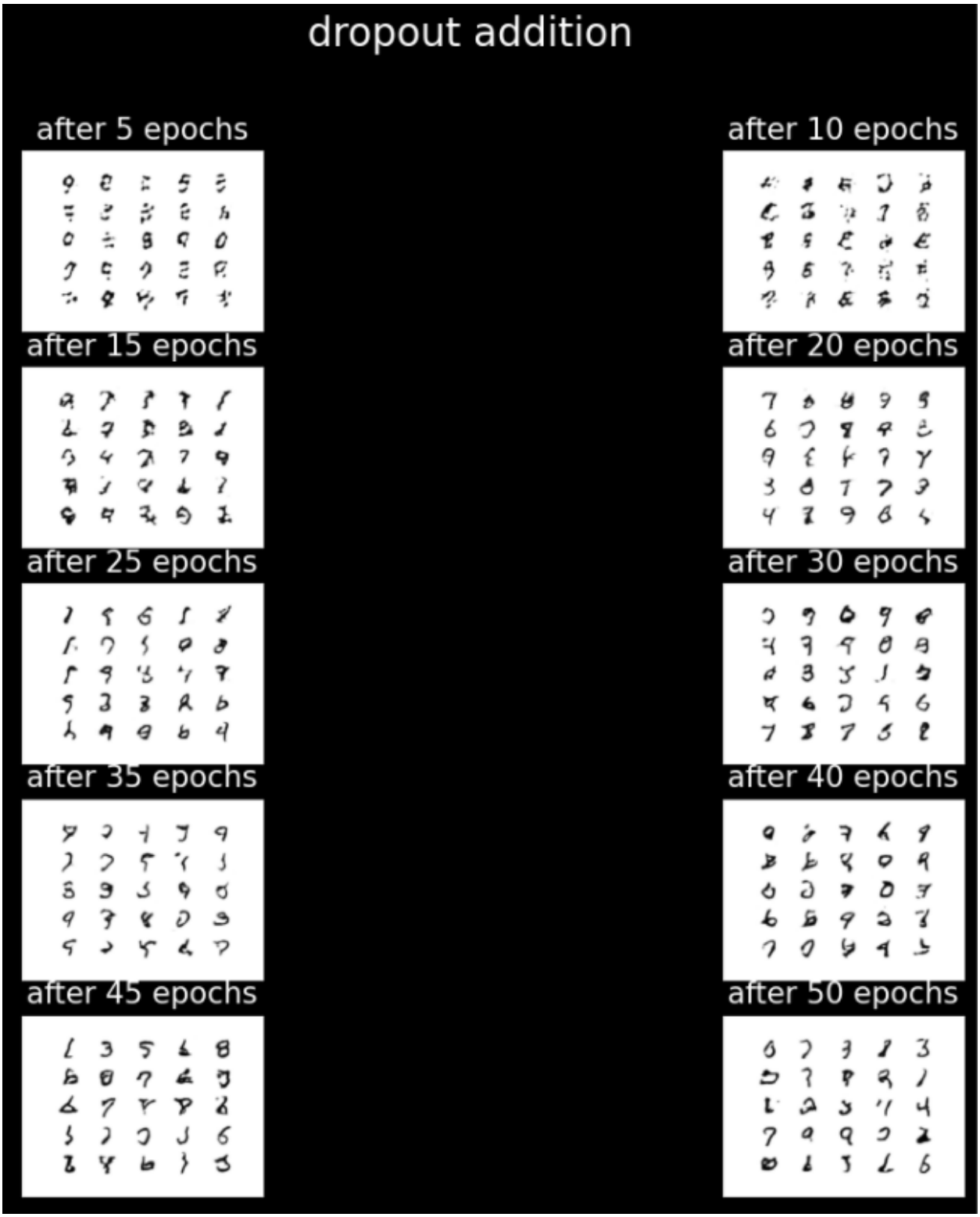
```
>5, 234/234, d_loss=0.713, g_loss=0.723
>Discriminator accuracy on real_data: 16%, fake_data: 73%
>10, 234/234, d_loss=0.684, g_loss=0.692
>Discriminator accuracy on real_data: 78%, fake_data: 44%
>15, 234/234, d_loss=0.683, g_loss=0.720
>Discriminator accuracy on real_data: 69%, fake_data: 59%
>20, 234/234, d_loss=0.686, g_loss=0.714
>Discriminator accuracy on real_data: 81%, fake_data: 33%
>25, 234/234, d_loss=0.695, g_loss=0.698
>Discriminator accuracy on real_data: 66%, fake_data: 26%
>30, 234/234, d_loss=0.681, g_loss=0.668
>Discriminator accuracy on real_data: 99%, fake_data: 19%
>35, 234/234, d_loss=0.661, g_loss=0.784
>Discriminator accuracy on real_data: 79%, fake_data: 65%
>40, 234/234, d_loss=0.663, g_loss=0.651
>Discriminator accuracy on real_data: 98%, fake_data: 10%
>45, 234/234, d_loss=0.652, g_loss=0.791
>Discriminator accuracy on real_data: 95%, fake_data: 56%
>50, 234/234, d_loss=0.648, g_loss=0.792
>Discriminator accuracy on real_data: 88%, fake_data: 50%
```

Discriminator accuracies during training are as follow:



As we can see in the plot above, training procedure is much more stable than all previous models.

And the output of the model after 50 epochs are as follows:



As it can be figured out from the figure above, after 50 epochs, the model has generated more generalized, realistic and favorable outputs compared to all previous models except batch-normalization. The generated outputs at 50 epochs are much smoother and more interpretable than both base model and dropout.

“addition of convolution layers”

Now we are going to evaluate the model with addition of two more convolutional layers.

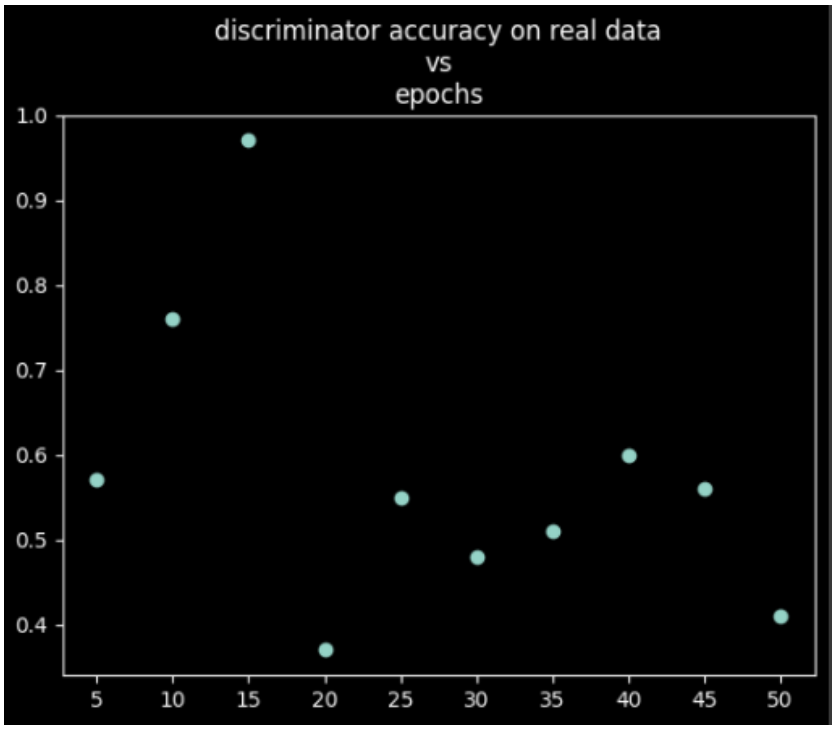
```
def generator(latent_dim):
  model = Sequential()
  init = RandomNormal(mean=0.0, stddev=0.02)
  n_nodes = 64 * 7 * 7

  model.add(Dense(n_nodes, input_dim=latent_dim))
  model.add(LeakyReLU(alpha=0.2))

  model.add(Reshape((7, 7, 64)))

  model.add(Conv2DTranspose(64, (4,4), strides=(2,2), padding='same',kernel_initializer=init))
  model.add(LeakyReLU(alpha=0.2))

  model.add(Conv2DTranspose(64, (4,4), strides=(1,1), padding='same',kernel_initializer=init))
  model.add(LeakyReLU(alpha=0.2))

  model.add(Conv2DTranspose(64, (4,4), strides=(2,2), padding='same',kernel_initializer=init))
  model.add(LeakyReLU(alpha=0.2))

  model.add(Conv2DTranspose(64, (4,4), strides=(1,1), padding='same',kernel_initializer=init))
  model.add(LeakyReLU(alpha=0.2))

  model.add(Conv2D(1, (3,3), activation='tanh', padding='same',kernel_initializer=init))
  return model
```

The results after this modification are as follow:

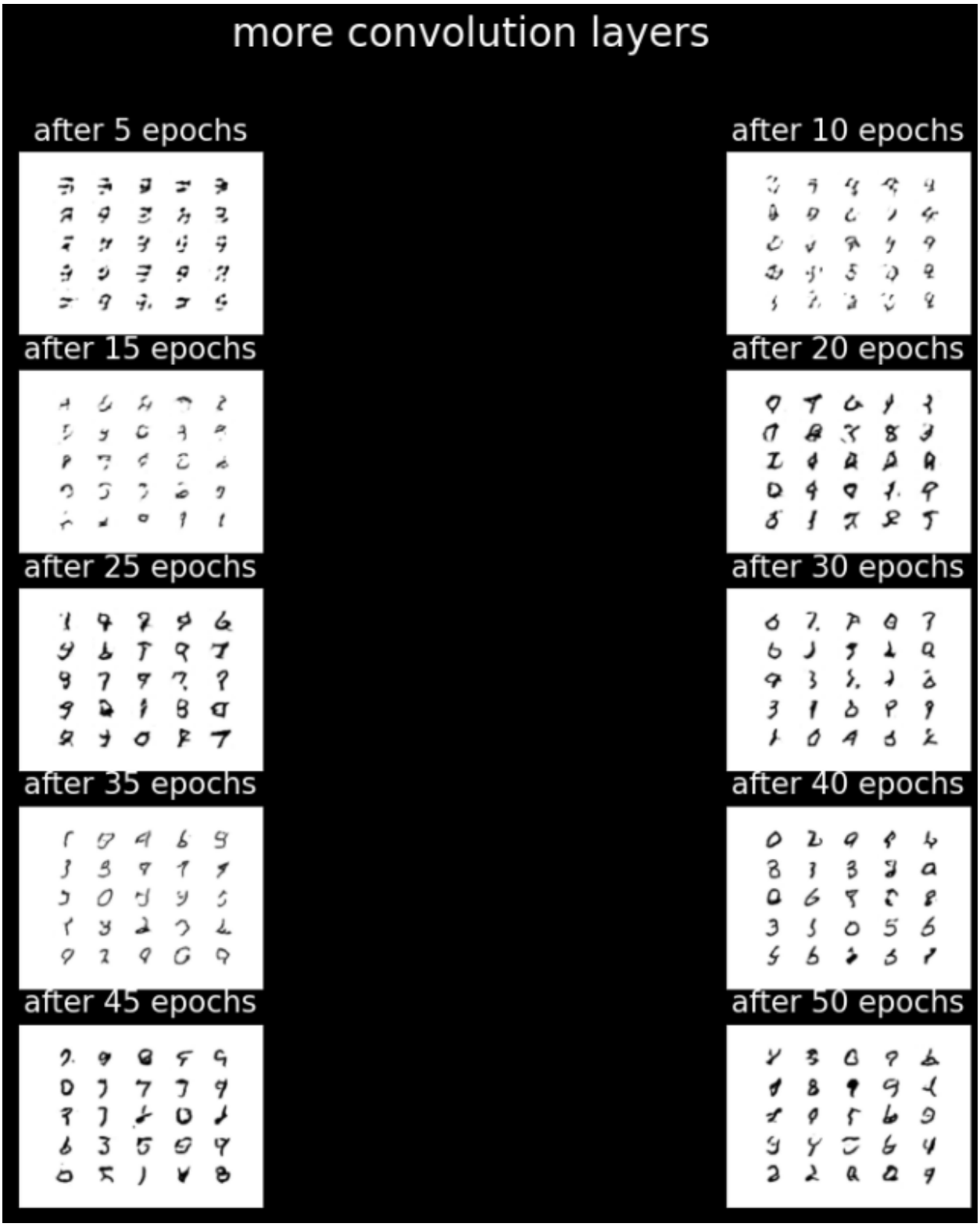
Accuracies and losses of all models during training are listed below:

```
>5, 234/234, d_loss=0.687, g_loss=0.769
>Discriminator accuracy on real_data: 57%, fake_data: 78%
>10, 234/234, d_loss=0.702, g_loss=0.644
>Discriminator accuracy on real_data: 76%, fake_data: 11%
>15, 234/234, d_loss=0.685, g_loss=0.652
>Discriminator accuracy on real_data: 97%, fake_data: 6%
>20, 234/234, d_loss=0.663, g_loss=0.805
>Discriminator accuracy on real_data: 37%, fake_data: 100%
>25, 234/234, d_loss=0.668, g_loss=0.770
>Discriminator accuracy on real_data: 55%, fake_data: 95%
>30, 234/234, d_loss=0.693, g_loss=0.699
>Discriminator accuracy on real_data: 48%, fake_data: 31%
>35, 234/234, d_loss=0.711, g_loss=0.687
>Discriminator accuracy on real_data: 51%, fake_data: 11%
>40, 234/234, d_loss=0.683, g_loss=0.694
>Discriminator accuracy on real_data: 60%, fake_data: 45%
>45, 234/234, d_loss=0.704, g_loss=0.688
>Discriminator accuracy on real_data: 56%, fake_data: 21%
>50, 234/234, d_loss=0.699, g_loss=0.705
>Discriminator accuracy on real_data: 41%, fake_data: 47%
```

Discriminator accuracies during training are as follow:



And the output of the model after 50 epochs are as follows:



As it can be figured out from the figure above, after 50 epochs, the model has generated more generalized, realistic and favorable

outputs compared to all previous models. The generated outputs at 40 epochs are much smoother and more interpretable than all previous models.

Considering all experiments above, it can be figured out that change in number of layers may improve or weaken the performance of the GAN model depending on type of the layer to be added. In the case of DCGAN, increase in batch-normalization and convolutional layers may improve the performance of GAN model.

3.

For the experiments of this problem, all evaluations will be studied on the base model.

It should be noted that noises were added to images using built-in layer of keras named as ‘GaussianNoise’ with the default mean of 0 and specified standard deviation which in fact controls noise intensity. This layer is added at the bottom of the discriminator which is shown in the figure below:

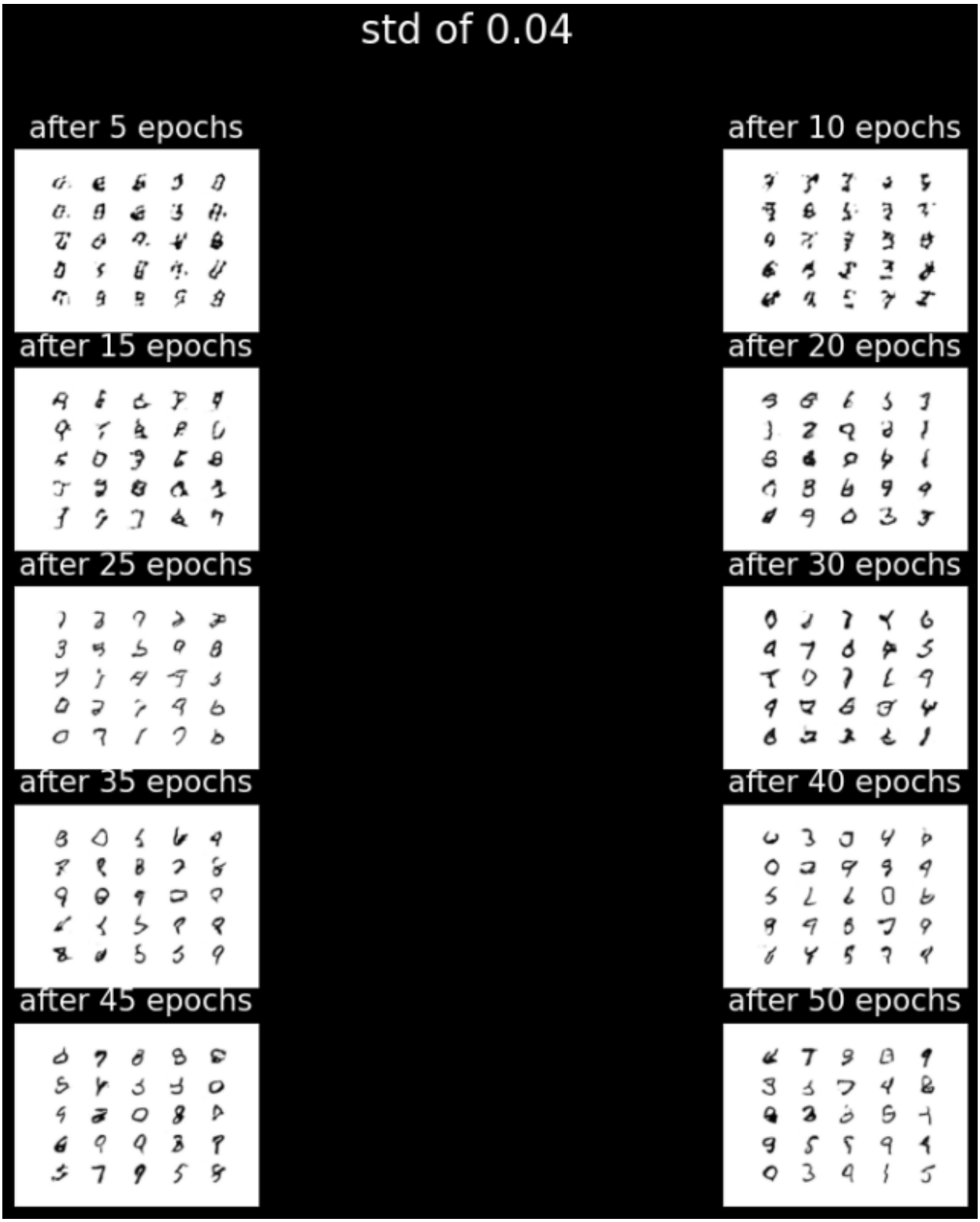
```python
def define_discriminator(in_shape=(28,28,1)):
  model = Sequential()
  init = RandomNormal(mean=0.0, stddev=0.02)
  model.add(GaussianNoise(0.1,input_shape=in_shape))
  model.add(Conv2D(64, (3,3), strides=(2, 2), padding='same',kernel_initializer=init))
  model.add(LeakyReLU(alpha=0.2))
  model.add(Dropout(0.2))
  model.add(Conv2D(64, (3,3), strides=(2, 2), padding='same',kernel_initializer=init))
  model.add(LeakyReLU(alpha=0.2))
  model.add(Dropout(0.2))
  model.add(Flatten())
  model.add(Dense(1, activation='sigmoid',kernel_initializer=init))
  opt = Adam(lr=0.0002, beta_1=0.5)
  model.compile(loss='binary_crossentropy', optimizer=opt, metrics=['accuracy'])
  return model
```

This evaluation will be performed using standard deviation values of 0.04, 0.4, 0.8, and 2.

(For this part, only accuracies and losses during training along with generated images for each value will be shown and the conclusions will be brought at the end)

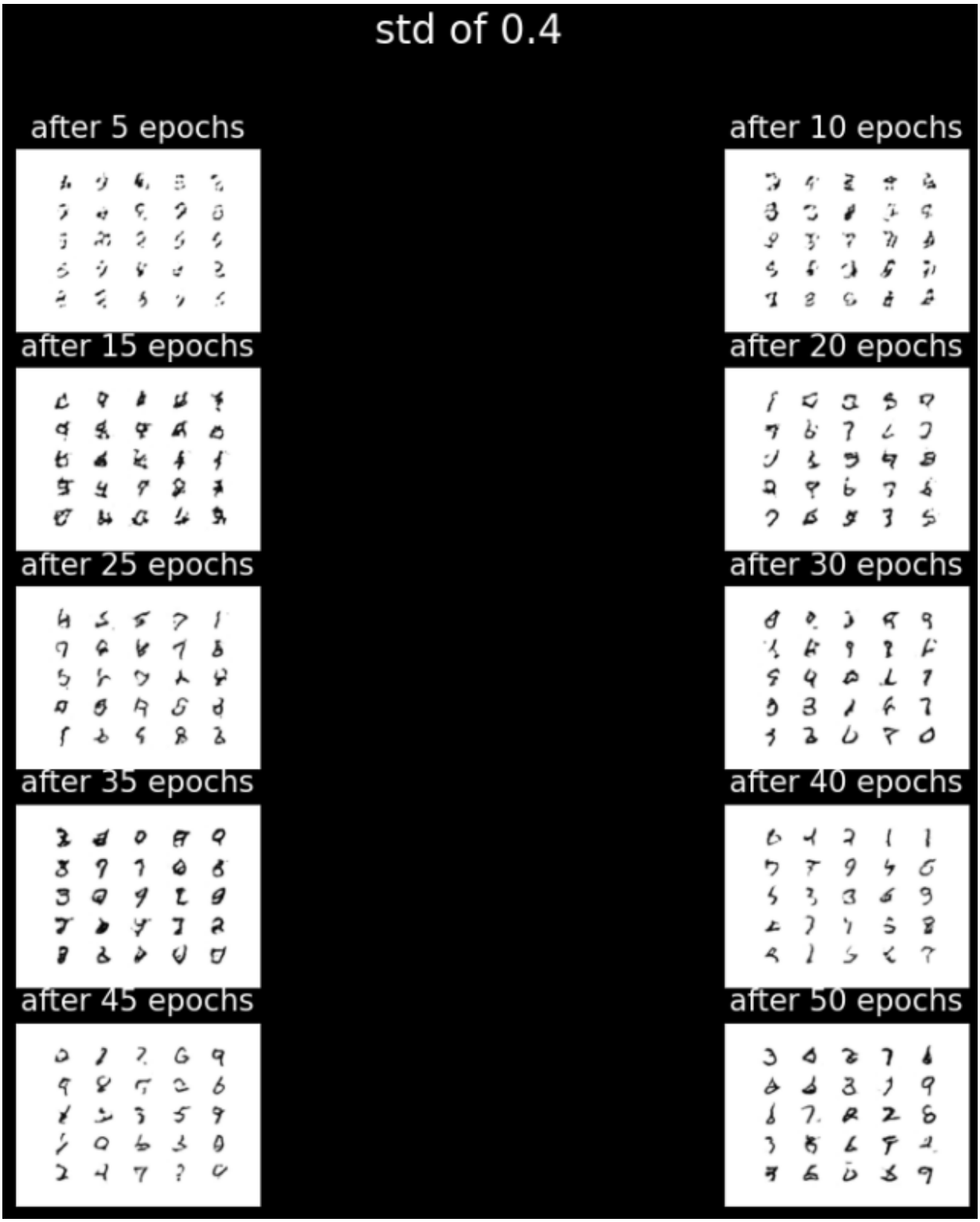
Stddev of 0.04:

```
>5, 234/234, d_loss=0.666, g_loss=0.730
>Discriminator accuracy on real_data: 67%, fake_data: 59%
>10, 234/234, d_loss=0.628, g_loss=0.772
>Discriminator accuracy on real_data: 85%, fake_data: 65%
>15, 234/234, d_loss=0.624, g_loss=0.789
>Discriminator accuracy on real_data: 68%, fake_data: 67%
>20, 234/234, d_loss=0.677, g_loss=0.736
>Discriminator accuracy on real_data: 70%, fake_data: 64%
>25, 234/234, d_loss=0.675, g_loss=0.817
>Discriminator accuracy on real_data: 53%, fake_data: 90%
>30, 234/234, d_loss=0.628, g_loss=0.792
>Discriminator accuracy on real_data: 79%, fake_data: 73%
>35, 234/234, d_loss=0.631, g_loss=0.835
>Discriminator accuracy on real_data: 60%, fake_data: 82%
>40, 234/234, d_loss=0.661, g_loss=0.764
>Discriminator accuracy on real_data: 61%, fake_data: 61%
>45, 234/234, d_loss=0.665, g_loss=0.751
>Discriminator accuracy on real_data: 53%, fake_data: 74%
>50, 234/234, d_loss=0.679, g_loss=0.753
>Discriminator accuracy on real_data: 39%, fake_data: 87%
```


std of 0.04
after 5 epochs
after 10 epochs
after 15 epochs
after 20 epochs
after 25 epochs
after 30 epochs
after 35 epochs
after 40 epochs
after 45 epochs
after 50 epochs

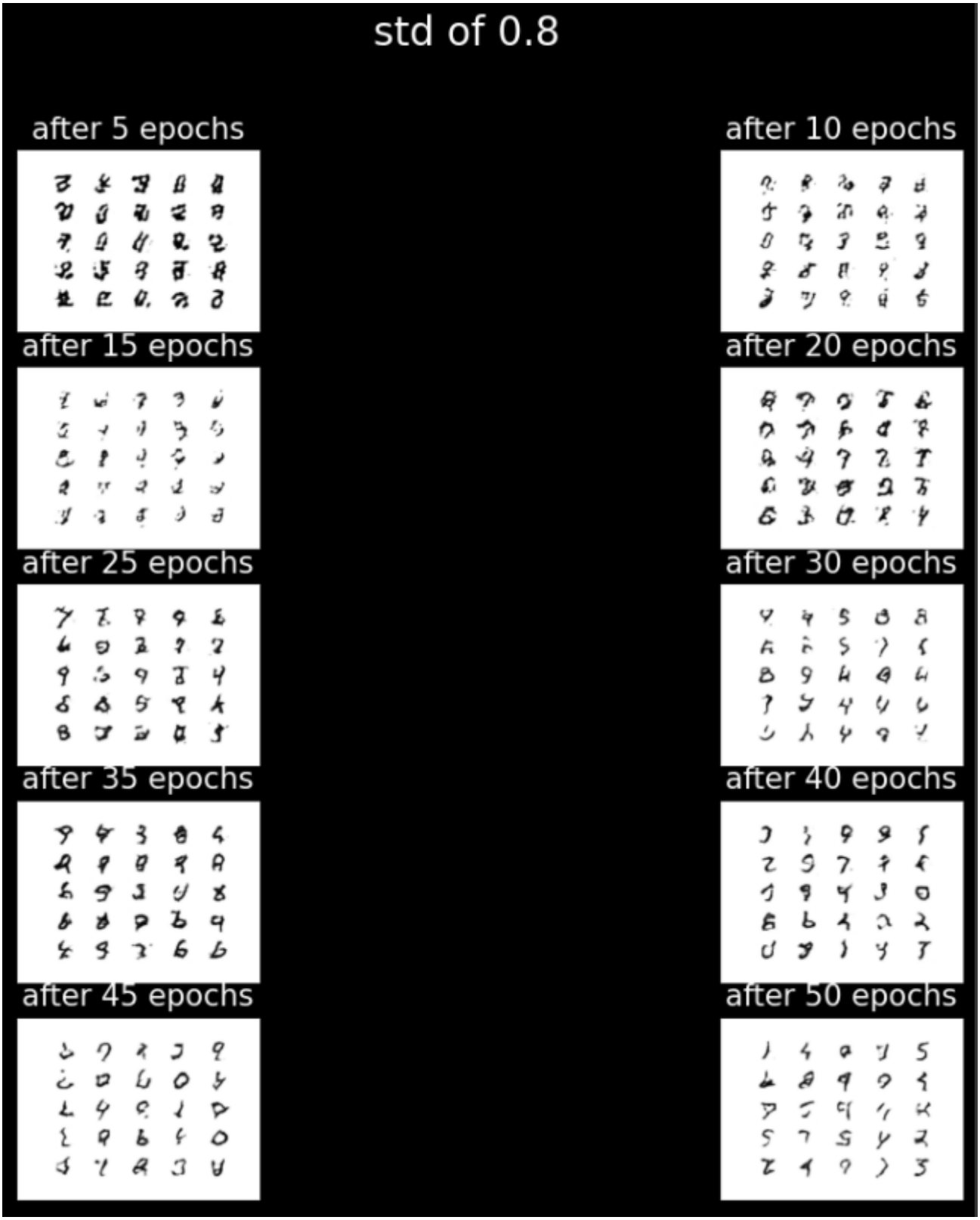
Stddev of 0.4:

```
>5, 234/234, d_loss=0.675, g_loss=0.766
>Discriminator accuracy on real_data: 63%, fake_data: 92%
>10, 234/234, d_loss=0.637, g_loss=0.766
>Discriminator accuracy on real_data: 86%, fake_data: 60%
>15, 234/234, d_loss=0.685, g_loss=0.731
>Discriminator accuracy on real_data: 93%, fake_data: 9%
>20, 234/234, d_loss=0.684, g_loss=0.721
>Discriminator accuracy on real_data: 94%, fake_data: 2%
>25, 234/234, d_loss=0.671, g_loss=0.741
>Discriminator accuracy on real_data: 98%, fake_data: 7%
>30, 234/234, d_loss=0.668, g_loss=0.739
>Discriminator accuracy on real_data: 95%, fake_data: 5%
>35, 234/234, d_loss=0.716, g_loss=0.635
>Discriminator accuracy on real_data: 100%, fake_data: 0%
>40, 234/234, d_loss=0.663, g_loss=0.786
>Discriminator accuracy on real_data: 97%, fake_data: 4%
>45, 234/234, d_loss=0.712, g_loss=0.752
>Discriminator accuracy on real_data: 91%, fake_data: 4%
>50, 234/234, d_loss=0.678, g_loss=0.723
>Discriminator accuracy on real_data: 94%, fake_data: 2%
```


std of 0.4
after 5 epochs
after 10 epochs
after 15 epochs
after 20 epochs
after 25 epochs
after 30 epochs
after 35 epochs
after 40 epochs
after 45 epochs
after 50 epochs

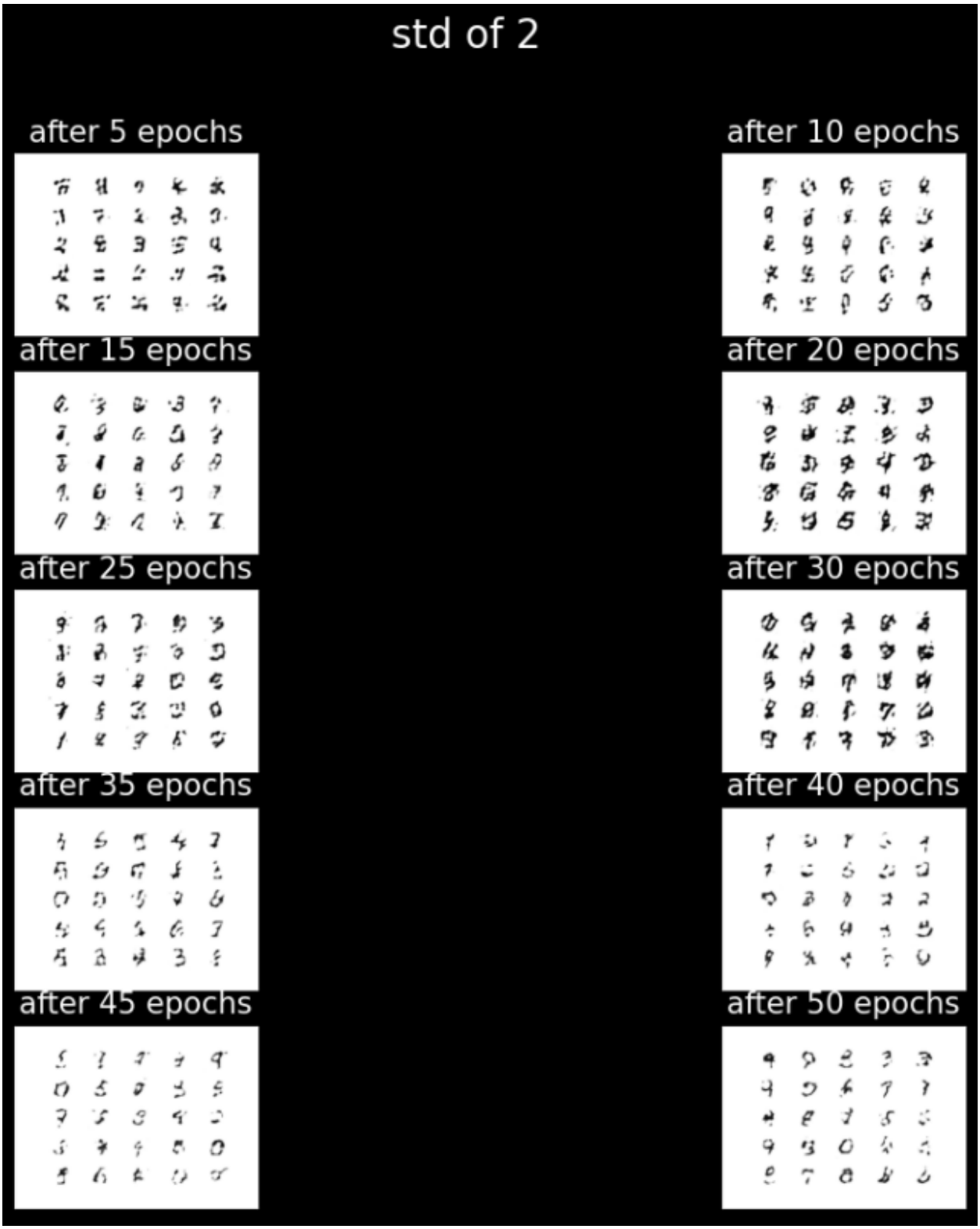
Stddev of 0.8:

```
>5, 234/234, d_loss=0.721, g_loss=0.679
>Discriminator accuracy on real_data: 39%, fake_data: 38%
>10, 234/234, d_loss=0.698, g_loss=0.709
>Discriminator accuracy on real_data: 58%, fake_data: 69%
>15, 234/234, d_loss=0.674, g_loss=0.723
>Discriminator accuracy on real_data: 94%, fake_data: 38%
>20, 234/234, d_loss=0.677, g_loss=0.764
>Discriminator accuracy on real_data: 94%, fake_data: 32%
>25, 234/234, d_loss=0.704, g_loss=0.688
>Discriminator accuracy on real_data: 100%, fake_data: 1%
>30, 234/234, d_loss=0.710, g_loss=0.689
>Discriminator accuracy on real_data: 97%, fake_data: 0%
>35, 234/234, d_loss=0.690, g_loss=0.705
>Discriminator accuracy on real_data: 100%, fake_data: 0%
>40, 234/234, d_loss=0.683, g_loss=0.703
>Discriminator accuracy on real_data: 99%, fake_data: 0%
>45, 234/234, d_loss=0.696, g_loss=0.694
>Discriminator accuracy on real_data: 99%, fake_data: 0%
>50, 234/234, d_loss=0.694, g_loss=0.705
>Discriminator accuracy on real_data: 100%, fake_data: 0%
```


std of 0.8
after 5 epochs
after 10 epochs
after 15 epochs
after 20 epochs
after 25 epochs
after 30 epochs
after 35 epochs
after 40 epochs
after 45 epochs
after 50 epochs

Stddev of 2:

```
>5, 234/234, d_loss=0.687, g_loss=0.690
>Discriminator accuracy on real_data: 49%, fake_data: 78%
>10, 234/234, d_loss=0.688, g_loss=0.713
>Discriminator accuracy on real_data: 0%, fake_data: 99%
>15, 234/234, d_loss=0.693, g_loss=0.709
>Discriminator accuracy on real_data: 70%, fake_data: 26%
>20, 234/234, d_loss=0.696, g_loss=0.704
>Discriminator accuracy on real_data: 66%, fake_data: 55%
>25, 234/234, d_loss=0.692, g_loss=0.716
>Discriminator accuracy on real_data: 95%, fake_data: 0%
>30, 234/234, d_loss=0.689, g_loss=0.716
>Discriminator accuracy on real_data: 87%, fake_data: 54%
>35, 234/234, d_loss=0.701, g_loss=0.686
>Discriminator accuracy on real_data: 100%, fake_data: 0%
>40, 234/234, d_loss=0.695, g_loss=0.692
>Discriminator accuracy on real_data: 100%, fake_data: 0%
>45, 234/234, d_loss=0.698, g_loss=0.696
>Discriminator accuracy on real_data: 100%, fake_data: 0%
>50, 234/234, d_loss=0.695, g_loss=0.697
>Discriminator accuracy on real_data: 100%, fake_data: 0%
```



Results of the value of 2 shows decrease in the performance of the model in both quality of the outputs and losses and accuracies

of the models. From this we can interpret that increasing in noise intensity to a certain level improves the performance and then it starts to degrade.

4.

100 generated fake images using ‘Conv Model’:

END of DCGAN