Assignment 3 - Generative Adversarial Network Programming (Full points: 100)

This assignment is more like a tutorial.

About Grading

(10 points) Essentially, you need to go through all the coding details we provide to you, and experiment with it. We will grade you by checking the history to see if you have run the code. (60 points) You need to complete certain code segments. If you don't completely go through the tutorial, you might fail this part. (30 points) Finally, you should adjust the hyperparameters to achieve a reasonable result (a lower printed loss) in order to receive a high grade.

Content

In this assignment (with tutorial), you're going to create your first generative adversarial network (GAN). Specifically, you need to build and train a GAN that can generate hand-written images of digits (0 to 9). You will be using PyTorch in this specialization, so if you're not familiar with this framework, you may find the PyTorch documentation useful. The hints will also often include links to relevant documentation.

Warning

If you encounter code errors (bugs), even in the tutorial, please attempt to resolve them independently, search for solutions using search engines such as Google, or seek advice from fellow classmates (plagiarism is not permitted). Failing to rectify bugs within the assignment or merely writing code without producing executable results will result in a significant deduction of points.

Objectives

1. Build the generator and discriminator components of a GAN from scratch.
2. Create generator and discriminator loss functions.
3. Train your GAN and visualize the generated images.

Key Concepts & Review

MNIST Dataset

The training images your discriminator will be using is from a dataset called MNIST. It contains 60,000 images of handwritten digits, from 0 to 9. (Google it if you want to see the details.)

You may notice that the images are quite pixelated -- this is because they are all only 28 x 28! The small size of its images makes MNIST ideal for simple training. Additionally, these images are also in black-and-white so only one dimension, or "color channel").

Tensor

You will represent the data using tensors. Tensors are a generalization of matrices: for example, a stack of three matrices with the amounts of red, green, and blue at different locations in a 64 x 64 pixel image is a tensor with the shape 3 x 64 x 64.

Tensors are easy to manipulate and supported by PyTorch, the library you will be using. Feel free to explore them more, but you can imagine these as multi-dimensional matrices or vectors!

Batches

While you could train your model after generating one image, it is extremely inefficient and leads to less stable training. In GANs, and in deep learning in general, you will process multiple images per training step. These are called batches.

This means that your generator will generate an entire batch of images and receive the discriminator's feedback on each before updating the model. The same goes for the discriminator, it will calculate its loss on the entire batch of generated images as well as on the reals before the model is updated.

```
import torch

print(torch.__version__)

if torch.cuda.is_available():
    print("GPU is available.")
else:
    print("GPU is not available.")
```

```
2.1.0+cu118
GPU is available.
```

Tutorial (10 points)

You will begin by importing some useful packages and the dataset you will use to build and train your GAN.

```
import torch
from torch import nn
from tqdm.auto import tqdm
from torchvision import transforms
from torchvision.datasets import MNIST # Training dataset
from torchvision.utils import make_grid
from torch.utils.data import DataLoader
import matplotlib.pyplot as plt
torch.manual_seed(0) # Set for testing purposes, please do not change!

def show_tensor_images(image_tensor, num_images=25, size=(1, 28, 28)):
    '''
```

```
    Function for visualizing images: Given a tensor of images, number
of images, and
    size per image, plots and prints the images in a uniform grid.
    '''
    image_unflat = image_tensor.view(-1, *size).detach()#.cpu()
    image_grid = make_grid(image_unflat[:num_images], nrow=5)
    plt.imshow(image_grid.permute(1, 2, 0).squeeze())
    plt.show()
```

Generator

The first step is to build the generator component.

You will start by creating a function to make a single layer/block for the generator's neural network. Each block should include a linear transformation to map to another shape, a batch normalization for stabilization, and finally a non-linear activation function (you use a ReLU here) so the output can be transformed in complex ways.

```
def get_generator_block(input_dim, output_dim):
    '''
    Function for returning a block of the generator's neural network
    given input and output dimensions.
    Parameters:
        input_dim: the dimension of the input vector, a scalar
        output_dim: the dimension of the output vector, a scalar
    Returns:
        a generator neural network layer, with a linear transformation
          followed by a batch normalization and then a relu activation
    '''
    return nn.Sequential(
        nn.Linear(input_dim, output_dim),
        nn.BatchNorm1d(output_dim),
        nn.ReLU(inplace=True),
    )

# Verify the generator block function
def test_gen_block(in_features, out_features, num_test=1000):
    block = get_generator_block(in_features, out_features)

    # Check the three parts
    assert len(block) == 3
    assert type(block[0]) == nn.Linear
    assert type(block[1]) == nn.BatchNorm1d
    assert type(block[2]) == nn.ReLU

    # Check the output shape
    test_input = torch.randn(num_test, in_features)
    test_output = block(test_input)
    assert tuple(test_output.shape) == (num_test, out_features)
```

```
    assert test_output.std() > 0.55
    assert test_output.std() < 0.65

test_gen_block(25, 12)
test_gen_block(15, 28)
print("Success!")
```

```
Success!
```

Now you can build the generator class. It will take 3 values:

- The noise vector dimension
- The image dimension
- The initial hidden dimension

Using these values, the generator will build a neural network with 5 layers/blocks. Beginning with the noise vector, the generator will apply non-linear transformations via the block function until the tensor is mapped to the size of the image to be outputted (the same size as the real images from MNIST). You will need to fill in the code for final layer since it is different than the others. The final layer does not need a normalization or activation function, but does need to be scaled with a sigmoid function.

Finally, you are given a forward pass function that takes in a noise vector and generates an image of the output dimension using your neural network.

```
class Generator(nn.Module):
    '''
    Generator Class
    Values:
        z_dim: the dimension of the noise vector, a scalar
        im_dim: the dimension of the images, fitted for the dataset
used, a scalar
          (MNIST images are 28 x 28 = 784 so that is your default)
        hidden_dim: the inner dimension, a scalar
    '''
    def __init__(self, z_dim=10, im_dim=784, hidden_dim=128):
        super(Generator, self).__init__()
        # Build the neural network
        self.gen = nn.Sequential(
            get_generator_block(z_dim, hidden_dim),
            get_generator_block(hidden_dim, hidden_dim * 2),
            get_generator_block(hidden_dim * 2, hidden_dim * 4),
            get_generator_block(hidden_dim * 4, hidden_dim * 8),

            nn.Linear(hidden_dim * 8, im_dim),
            nn.Sigmoid()
        )
    def forward(self, noise):
        '''
        Function for completing a forward pass of the generator: Given
```

```
a noise tensor,
        returns generated images.
        Parameters:
            noise: a noise tensor with dimensions (n_samples, z_dim)
        '''
        return self.gen(noise)

    def get_gen(self):
        '''
        Returns:
            the sequential model
        '''
        return self.gen

# Verify the generator class
def test_generator(z_dim, im_dim, hidden_dim, num_test=10000):
    gen = Generator(z_dim, im_dim, hidden_dim).get_gen()

    # Check there are six modules in the sequential part
    assert len(gen) == 6
    test_input = torch.randn(num_test, z_dim)
    test_output = gen(test_input)

    # Check that the output shape is correct
    assert tuple(test_output.shape) == (num_test, im_dim)
    assert test_output.max() < 1, "Make sure to use a sigmoid"
    assert test_output.min() > 0, "Make sure to use a sigmoid"
    assert test_output.std() > 0.05, "Don't use batchnorm here"
    assert test_output.std() < 0.15, "Don't use batchnorm here"

test_generator(5, 10, 20)
test_generator(20, 8, 24)
print("Success!")
```

```
Success!
```

Noise

To be able to use your generator, you will need to be able to create noise vectors. The noise vector z has the important role of making sure the images generated from the same class don't all look the same -- think of it as a random seed. You will generate it randomly using PyTorch by sampling random numbers from the normal distribution. Since multiple images will be processed per pass, you will generate all the noise vectors at once.

Note that whenever you create a new tensor using torch.ones, torch.zeros, or torch.randn, you either need to create it on the target device, e.g. `torch.ones(3, 3, device=device)`, or move it onto the target device using `torch.ones(3, 3).to(device)`. You do not need to do this if you're creating a tensor by manipulating another tensor or by using a variation that defaults the device to the input, such as `torch.ones_like`. In general, use

`torch.ones_like` and `torch.zeros_like` instead of `torch.ones` or `torch.zeros` where possible.

```
def get_noise(n_samples, z_dim, device='cpu'):
    '''
    Function for creating noise vectors: Given the dimensions
(n_samples, z_dim),
    creates a tensor of that shape filled with random numbers from the
normal distribution.
    Parameters:
        n_samples: the number of samples to generate, a scalar
        z_dim: the dimension of the noise vector, a scalar
        device: the device type
    '''
    # NOTE: To use this on GPU with device='cuda', make sure to pass
the device
    # argument to the function you use to generate the noise.
    return torch.randn(n_samples, z_dim, device=device)

# Verify the noise vector function
def test_get_noise(n_samples, z_dim, device='cpu'):
    noise = get_noise(n_samples, z_dim, device)

    # Make sure a normal distribution was used
    assert tuple(noise.shape) == (n_samples, z_dim)
    assert torch.abs(noise.std() - torch.tensor(1.0)) < 0.01
    assert str(noise.device).startswith(device)

test_get_noise(1000, 100, 'cpu')
if torch.cuda.is_available():
    test_get_noise(1000, 32, 'cuda')
    print("Success on gpu!")
else:
    print("Success!")
```

```
Success on gpu!
```

Discriminator

The second component that you need to construct is the discriminator. As with the generator component, you will start by creating a function that builds a neural network block for the discriminator.

Note: You use leaky ReLUs to prevent the "dying ReLU" problem, which refers to the phenomenon where the parameters stop changing due to consistently negative values passed to a ReLU, which result in a zero gradient. You will learn more about this in the following lectures!

```
def get_discriminator_block(input_dim, output_dim):
    '''
    Discriminator Block
```

```
    Function for returning a neural network of the discriminator given
input and output dimensions.
    Parameters:
        input_dim: the dimension of the input vector, a scalar
        output_dim: the dimension of the output vector, a scalar
    Returns:
        a discriminator neural network layer, with a linear
transformation
          followed by an nn.LeakyReLU activation with negative slope
of 0.2

(https://pytorch.org/docs/master/generated/torch.nn.LeakyReLU.html)
    '''
    return nn.Sequential(
        nn.Linear(input_dim, output_dim),
        nn.LeakyReLU(0.2, inplace=True)
    )

# Verify the discriminator block function
def test_disc_block(in_features, out_features, num_test=10000):
    block = get_discriminator_block(in_features, out_features)

    # Check there are two parts
    assert len(block) == 2
    test_input = torch.randn(num_test, in_features)
    test_output = block(test_input)

    # Check that the shape is right
    assert tuple(test_output.shape) == (num_test, out_features)

    # Check that the LeakyReLU slope is about 0.2
    assert -test_output.min() / test_output.max() > 0.1
    assert -test_output.min() / test_output.max() < 0.3
    assert test_output.std() > 0.3
    assert test_output.std() < 0.5

test_disc_block(25, 12)
test_disc_block(15, 28)
print("Success!")
```

```
Success!
```

Now you can use these blocks to make a discriminator! The discriminator class holds 2 values:

- The image dimension
- The hidden dimension

The discriminator will build a neural network with 4 layers. It will start with the image tensor and transform it until it returns a single number (1-dimension tensor) output. This output classifies whether an image is fake or real. Note that you do not need a sigmoid after the output layer

since it is included in the loss function. Finally, to use your discrimator's neural network you are given a forward pass function that takes in an image tensor to be classified.

```
class Discriminator(nn.Module):
    '''
    Discriminator Class
    Values:
        im_dim: the dimension of the images, fitted for the dataset
used, a scalar
            (MNIST images are 28x28 = 784 so that is your default)
        hidden_dim: the inner dimension, a scalar
    '''
    def __init__(self, im_dim=784, hidden_dim=128):
        super(Discriminator, self).__init__()
        self.disc = nn.Sequential(
            get_discriminator_block(im_dim, hidden_dim * 4),
            get_discriminator_block(hidden_dim * 4, hidden_dim * 2),
            get_discriminator_block(hidden_dim * 2, hidden_dim),
            # Hint: You want to transform the final output into a
single value,
            #       so add one more linear map.
            nn.Linear(hidden_dim, 1)
        )

    def forward(self, image):
        '''
        Function for completing a forward pass of the discriminator:
Given an image tensor,
        returns a 1-dimension tensor representing fake/real.
        Parameters:
            image: a flattened image tensor with dimension (im_dim)
        '''
        return self.disc(image)

    # Needed for grading
    def get_disc(self):
        '''
        Returns:
            the sequential model
        '''
        return self.disc

# Verify the discriminator class
def test_discriminator(z_dim, hidden_dim, num_test=100):

    disc = Discriminator(z_dim, hidden_dim).get_disc()

    # Check there are three parts
    assert len(disc) == 4
```

```
    # Check the linear layer is correct
    test_input = torch.randn(num_test, z_dim)
    test_output = disc(test_input)
    assert tuple(test_output.shape) == (num_test, 1)

    # Make sure there's no sigmoid
    assert test_input.max() > 1
    assert test_input.min() < -1

test_discriminator(5, 10)
test_discriminator(20, 8)
print("Success!")
```

```
Success!
```

(60 points) Complete the missing code

(30 points) Turning hyperparameters to get a great performance (with lower printed loss).

Now you can put it all together! First, you will set your parameters:

- criterion: the loss function
- n_epochs: the number of times you iterate through the entire dataset when training
- z_dim: the dimension of the noise vector
- display_step: how often to display/visualize the images
- batch_size: the number of images per forward/backward pass
- lr: the learning rate
- device: the device type, here using a GPU (which runs CUDA), not CPU

Next, you will load the MNIST dataset as tensors using a dataloader.

```
# Set your parameters
criterion = nn.BCEWithLogitsLoss()
n_epochs = 200
z_dim = 64
display_step = 500
batch_size = 128
lr = 0.00001

# Load MNIST dataset as tensors
dataloader = DataLoader(
    MNIST('.', download=True, transform=transforms.ToTensor()),
    batch_size=batch_size,
```

```
    shuffle=True)

device = 'cuda' # or 'cpu'
```

```
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-
ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-
ubyte.gz to ./MNIST/raw/train-images-idx3-ubyte.gz

100%|██████████| 9912422/9912422 [00:00<00:00, 82543899.39it/s]

Extracting ./MNIST/raw/train-images-idx3-ubyte.gz to ./MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-
ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-
ubyte.gz to ./MNIST/raw/train-labels-idx1-ubyte.gz

100%|██████████| 28881/28881 [00:00<00:00, 111955354.74it/s]

Extracting ./MNIST/raw/train-labels-idx1-ubyte.gz to ./MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
to ./MNIST/raw/t10k-images-idx3-ubyte.gz

100%|██████████| 1648877/1648877 [00:00<00:00, 29918071.81it/s]

Extracting ./MNIST/raw/t10k-images-idx3-ubyte.gz to ./MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz
to ./MNIST/raw/t10k-labels-idx1-ubyte.gz

100%|██████████| 4542/4542 [00:00<00:00, 21190799.52it/s]

Extracting ./MNIST/raw/t10k-labels-idx1-ubyte.gz to ./MNIST/raw
```

Now, you can initialize your generator, discriminator, and optimizers. Note that each optimizer only takes the parameters of one particular model, since we want each optimizer to optimize only one of the models.

```
gen = Generator(z_dim).to(device)
gen_opt = torch.optim.Adam(gen.parameters(), lr=lr)
disc = Discriminator().to(device)
disc_opt = torch.optim.Adam(disc.parameters(), lr=lr)
```

Before you train your GAN, you will need to create functions to calculate the discriminator's loss and the generator's loss. This is how the discriminator and generator will know how they are doing and improve themselves. Since the generator is needed when calculating the

discriminator's loss, you will need to call .detach() on the generator result to ensure that only the discriminator is updated!

Remember that you have already defined a loss function earlier (`criterion`) and you are encouraged to use `torch.ones_like` and `torch.zeros_like` instead of `torch.ones` or `torch.zeros` to set the labels. If you use `torch.ones` or `torch.zeros`, you'll need to pass `device=device` to them.

```
def get_disc_loss(gen, disc, criterion, real, num_images, z_dim,
device):
    '''
    Return the loss of the discriminator given inputs.
    Parameters:
        gen: the generator model, which returns an image given z-
dimensional noise
        disc: the discriminator model, which returns a single-
dimensional prediction of real/fake
        criterion: the loss function, which should be used to compare
               the discriminator's predictions to the ground truth
reality of the images
               (e.g. fake = 0, real = 1)
        real: a batch of real images
        num_images: the number of images the generator should produce,
                which is also the length of the real images
        z_dim: the dimension of the noise vector, a scalar
        device: the device type
    Returns:
        disc_loss: a torch scalar loss value for the current batch
    '''
    #     These are the steps you will need to complete:
    #       1) Create noise vectors and generate a batch (num_images)
of fake images.
    #            Make sure to pass the device argument to the noise.
    #       2) Get the discriminator's prediction of the fake image
    #            and calculate the loss. Don't forget to detach the
generator!
    #            (Remember the loss function you set earlier --
criterion. You need a
    #            'ground truth' tensor in order to calculate the loss.
    #            For example, a ground truth tensor for a fake image
is all zeros.)
    #       3) Get the discriminator's prediction of the real image
and calculate the loss.
    #       4) Calculate the discriminator's loss by averaging the
real and fake loss
    #            and set it to disc_loss.
    #     Note: Please do not use concatenation in your solution. The
tests are being updated to
    #           support this, but for now, average the two losses as
described in step (4).
```

```
    #     *Important*: You should NOT write your own loss function
here - use criterion(pred, true)!
    #### START CODE HERE ####
    # 1) Create noise vectors and generate a batch (num_images) of
fake images.
    noise = torch.randn(num_images, z_dim, device=device)
    fake_image_gen = gen(noise)

    # 2) Get the discriminator's prediction of the fake image and
calculate the loss.
    fake_image_pred = disc(fake_image_gen.detach())  # Detach the
generator output
    fake_image_loss = criterion(fake_image_pred,
torch.zeros_like(fake_image_pred))  # Ground truth for fake images is
all zeros

    # 3) Get the discriminator's prediction of the real image and
calculate the loss.
    real_image_pred = disc(real)
    real_image_loss = criterion(real_image_pred,
torch.ones_like(real_image_pred))  # Ground truth for real images is
all ones

    # 4) Calculate the discriminator's loss by averaging the real and
fake loss.
    disc_loss = (fake_image_loss + real_image_loss) / 2
    #### END CODE HERE ####
    return disc_loss

def get_gen_loss(gen, disc, criterion, num_images, z_dim, device):
    '''
    Return the loss of the generator given inputs.
    Parameters:
        gen: the generator model, which returns an image given z-
dimensional noise
        disc: the discriminator model, which returns a single-
dimensional prediction of real/fake
        criterion: the loss function, which should be used to compare
               the discriminator's predictions to the ground truth
reality of the images
               (e.g. fake = 0, real = 1)
        num_images: the number of images the generator should produce,
                which is also the length of the real images
        z_dim: the dimension of the noise vector, a scalar
        device: the device type
    Returns:
        gen_loss: a torch scalar loss value for the current batch
    '''
    #     These are the steps you will need to complete:
    #       1) Create noise vectors and generate a batch of fake
```

```
images.
    #           Remember to pass the device argument to the get_noise
function.
    #       2) Get the discriminator's prediction of the fake image.
    #       3) Calculate the generator's loss. Remember the generator
wants
    #          the discriminator to think that its fake images are
real
    #     *Important*: You should NOT write your own loss function
here - use criterion(pred, true)!
    #### START CODE HERE ####
    # 1) Create noise vectors and generate a batch of fake images.
    noise = get_noise(num_images, z_dim, device=device)
    fake_image_gen = gen(noise)

    # 2) Get the discriminator's prediction of the fake image.
    fake_image_pred = disc(fake_image_gen)

    # 3) Calculate the generator's loss. The generator wants the
discriminator to think that its fake images are real.
    gen_loss = criterion(fake_image_pred,
torch.ones_like(fake_image_pred))
    #### END CODE HERE ####
    return gen_loss
```

Finally, you can put everything together! For each epoch, you will process the entire dataset in batches. For every batch, you will need to update the discriminator and generator using their loss. Note that you may see a loss to be greater than 1, this is okay since binary cross entropy loss can be any positive number for a sufficiently confident wrong guess.

It's also often the case that the discriminator will outperform the generator, especially at the start, because its job is easier. It's important that neither one gets too good (that is, near-perfect accuracy), which would cause the entire model to stop learning. Balancing the two models is actually remarkably hard to do in a standard GAN.

In addition, be warned that this runs very slowly on a CPU. One way to run this more quickly is to use Google Colab:

1. Download the .ipynb
2. Upload it to Google Drive and open it with Google Colab
3. Make the runtime type GPU (under "Runtime" -> "Change runtime type" -> Select "GPU" from the dropdown)
4. Replace `device = "cpu"` with `device = "cuda"`
5. Make sure your `get_noise` function uses the right device -->

```
genLoss=[]
disLoss=[]

# Initialize variables
cur_step = 0
```

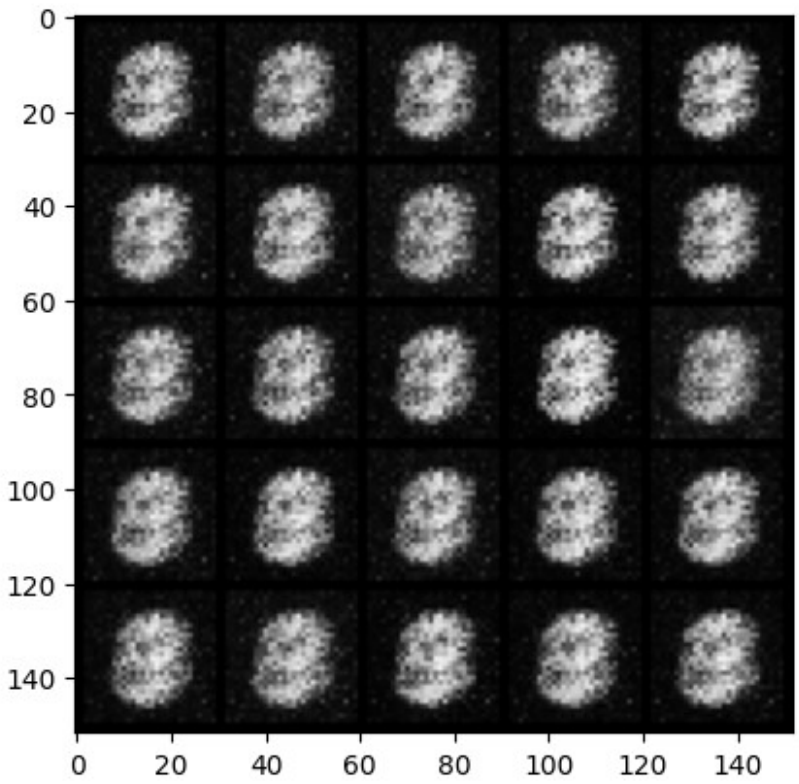
```
mean_generator_loss = 0
mean_discriminator_loss = 0
gen_loss = None
error = None

# Training loop
for epoch in range(n_epochs):
    # Dataloader returns the batches
    for real, _ in tqdm(dataloader):
        cur_batch_size = len(real)

        # Flatten the batch of real images from the dataset
        real = real.view(cur_batch_size, -1).to(device)

        ### Update discriminator ###
        # Zero out the gradients before backpropagation
        disc_opt.zero_grad()

        # Calculate discriminator loss
        disc_loss = get_disc_loss(gen, disc, criterion, real,
cur_batch_size, z_dim, device)
        disc_loss.backward()
        disc_opt.step()

        ### Update generator ###
        #### START CODE HERE ####
        #1) Zero out the gradients.
        gen_opt.zero_grad()
        #2) Calculate the generator loss, assigning it to gen_loss.
        gen_loss = get_gen_loss(gen, disc, criterion, cur_batch_size,
z_dim, device)
        #3) Backprop through the generator: update the gradients and
optimizer.
        gen_loss.backward()
        gen_opt.step()
        #### END CODE HERE ####

        # Keep track of the average discriminator loss
        mean_discriminator_loss += disc_loss.item() / display_step

        # Keep track of the average generator loss
        mean_generator_loss += gen_loss.item() / display_step

        ### Visualization code ###
        if cur_step % display_step == 0 and cur_step > 0:
            print(f"Epoch {epoch}, step {cur_step}: Generator loss:
{mean_generator_loss}, discriminator loss: {mean_discriminator_loss}")
            genLoss.append(mean_generator_loss)
            disLoss.append(mean_discriminator_loss)
            fake_noise = get_noise(cur_batch_size, z_dim,
```

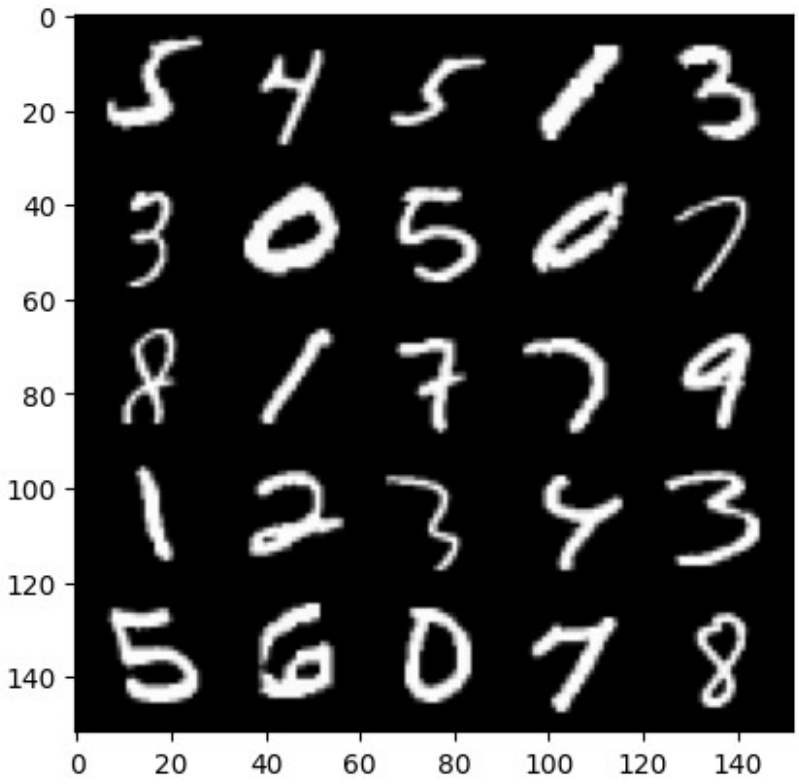
```
device=device)
            fake = gen(fake_noise)

            real_cpu = real.cpu()  # Move the real images to the CPU
            fake_cpu = fake.cpu()  # Move the fake images to the CPU
            show_tensor_images(fake_cpu) #show fake images
            show_tensor_images(real_cpu) #show real images
            mean_generator_loss = 0
            mean_discriminator_loss = 0

        cur_step += 1
```

{"model_id":"2a6700f4a7074b6ca2b0175b23efe3b0","version_major":2,"version_minor":0}

{"model_id":"2b8eaaf475234fe18e7cdce6a99a1ccf","version_major":2,"version_minor":0}

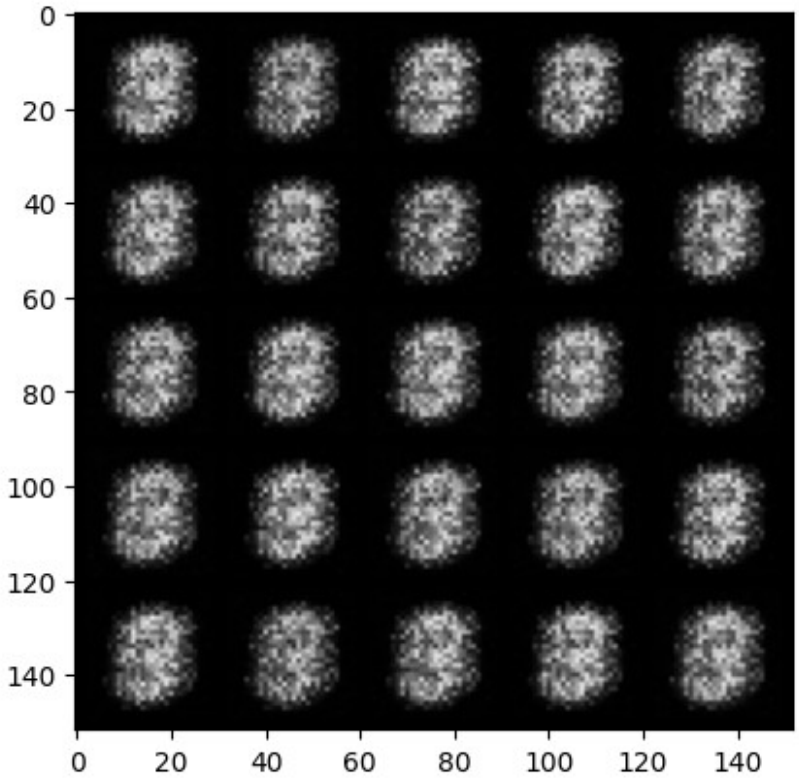
```
Epoch 1, step 500: Generator loss: 1.3948533631563191, discriminator
loss: 0.4179060722589492
```

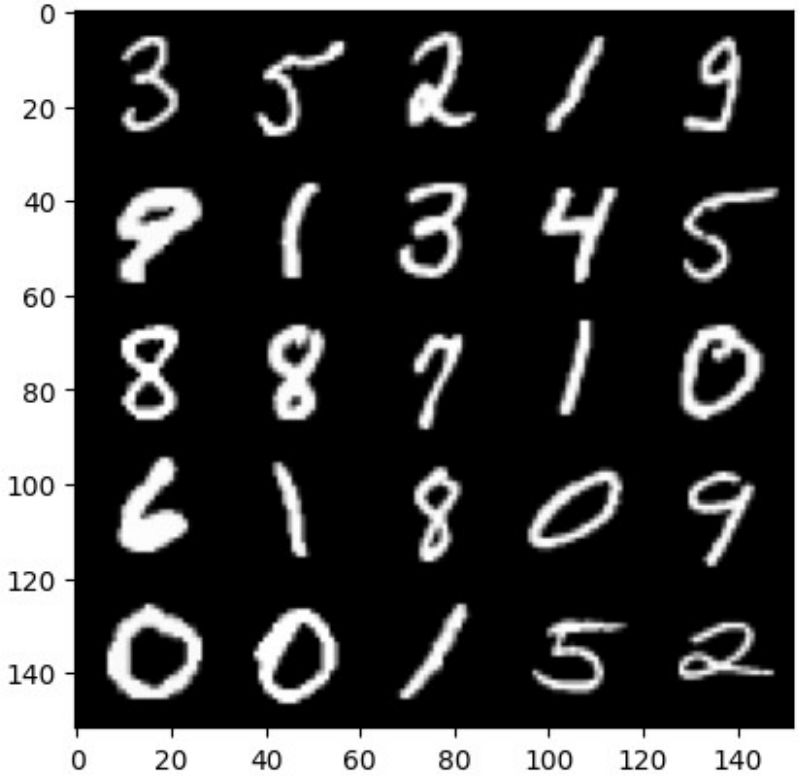




{"model_id":"1ff324655177465d85750556fcc51911","version_major":2,"version_minor":0}

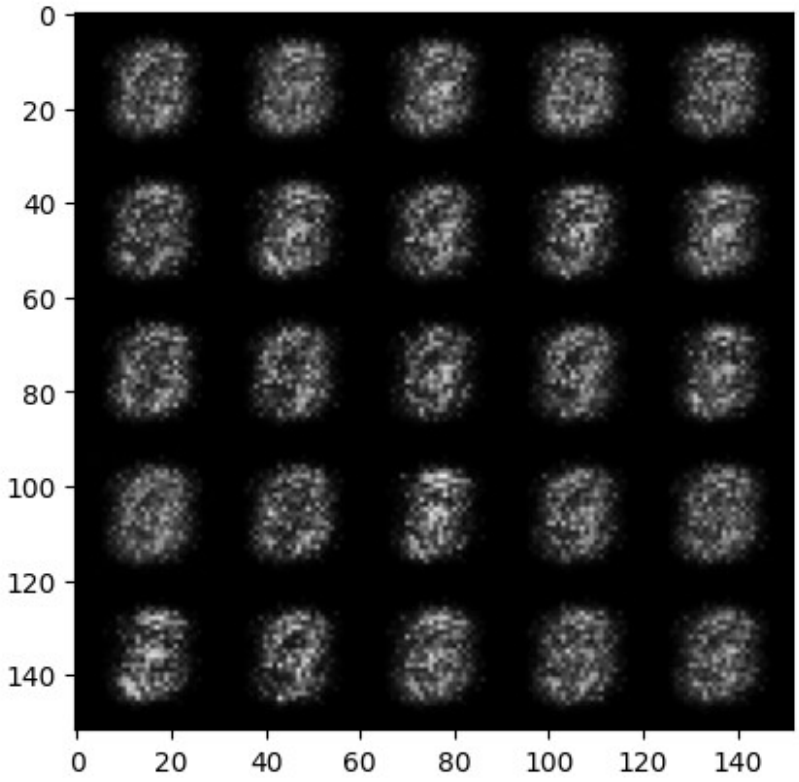
```
Epoch 2, step 1000: Generator loss: 1.7499893734455108, discriminator loss: 0.27954769375920285
```

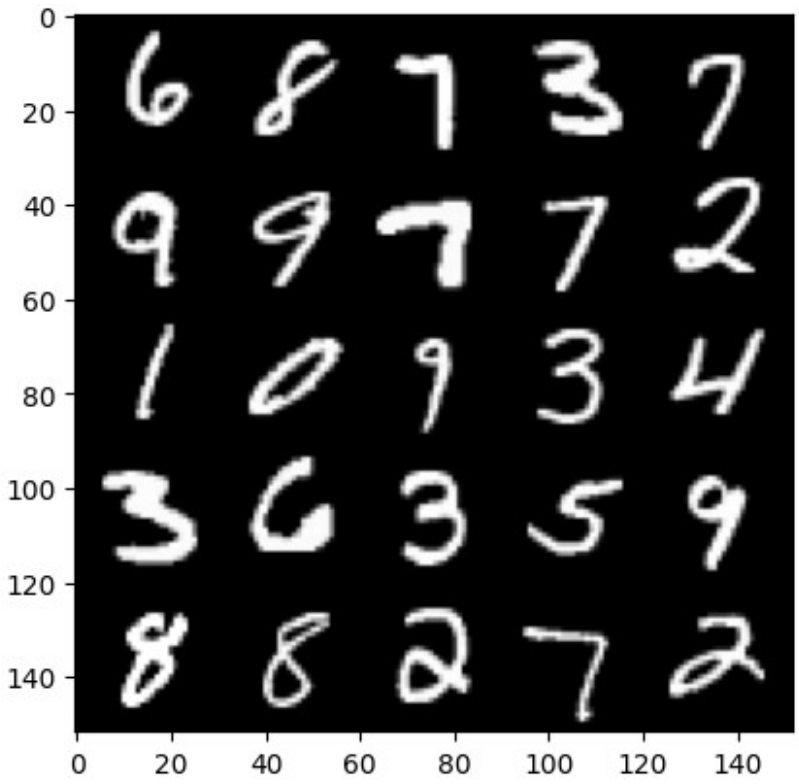

0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140


0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

{"model_id":"a4b058a7249f4bcda6ee516defee8dd0","version_major":2,"version_minor":0}

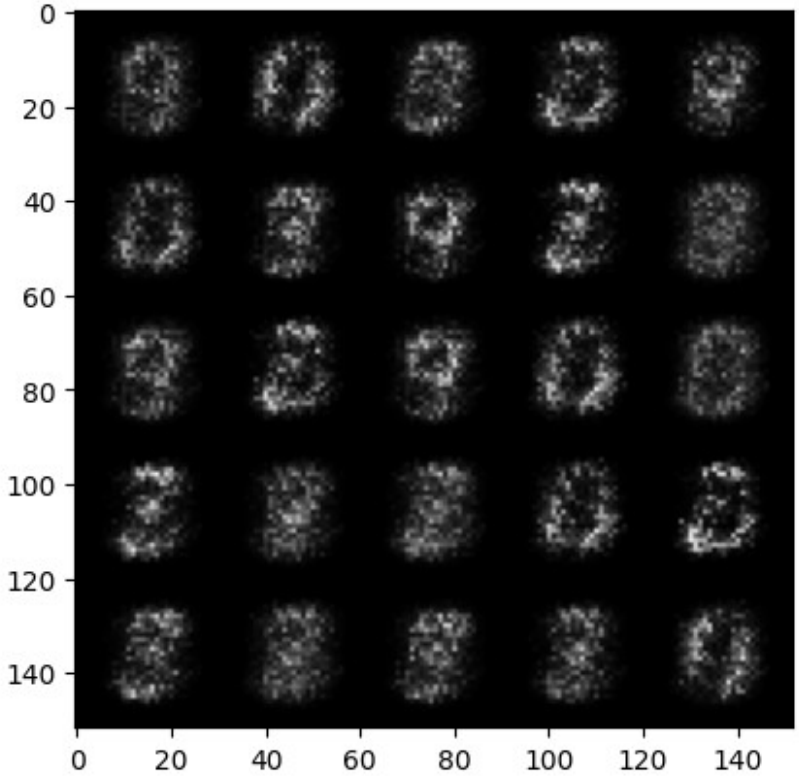
```
Epoch 3, step 1500: Generator loss: 2.0534333646297447, discriminator loss: 0.15915410882234562
```

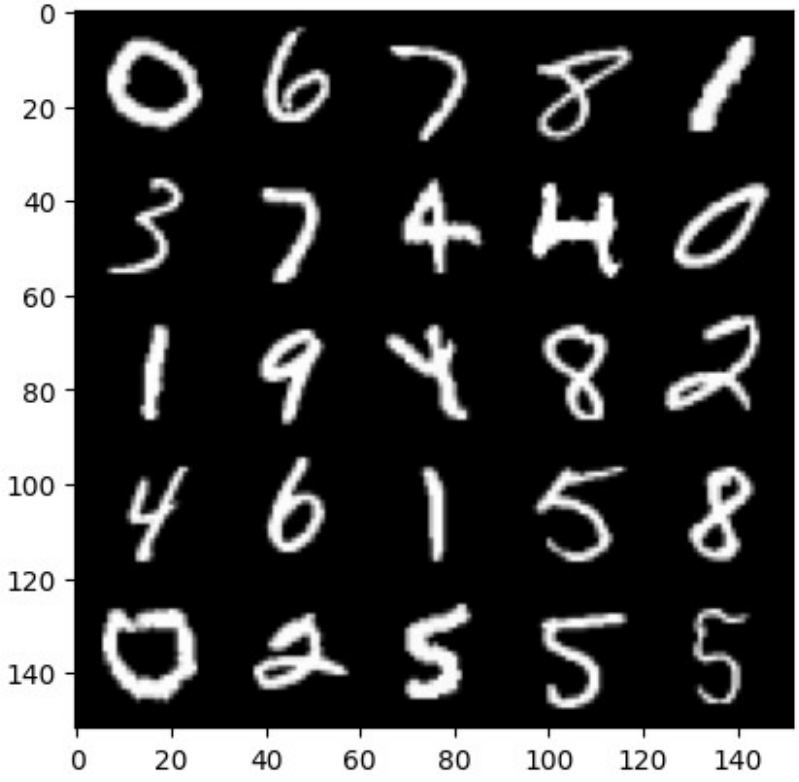




{"model_id":"2f3f0a44f8ed46ce828d3c7f9b403f05","version_major":2,"version_minor":0}

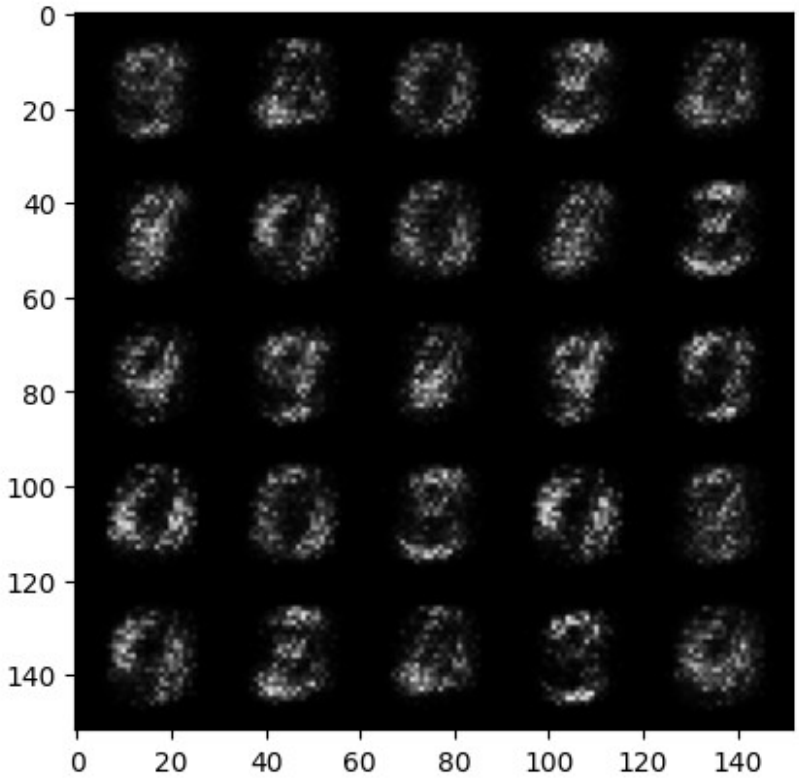
```
Epoch 4, step 2000: Generator loss: 1.660265215873717, discriminator loss: 0.22786851745843872
```

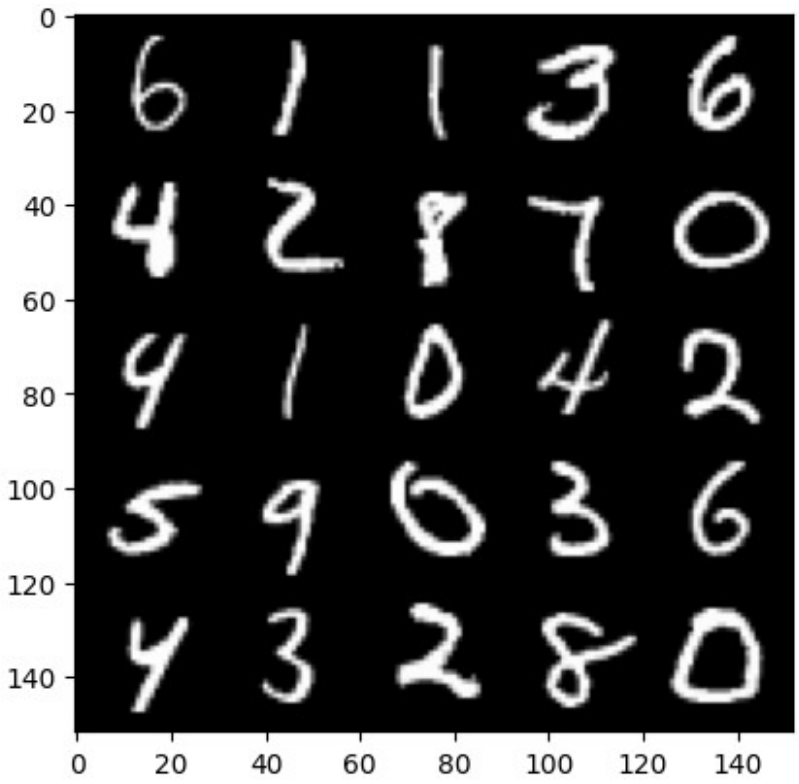

0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140


0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

{"model_id":"ca5196a4b0f14f709ffca166de29ba79","version_major":2,"version_minor":0}

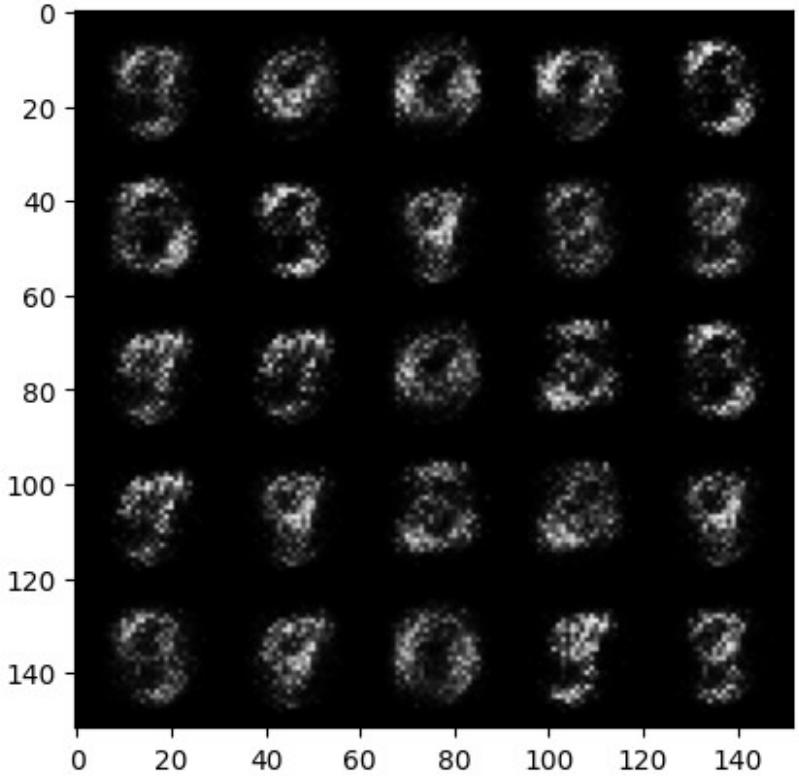
```
Epoch 5, step 2500: Generator loss: 1.6384419167041786, discriminator loss: 0.2122755697667598
```

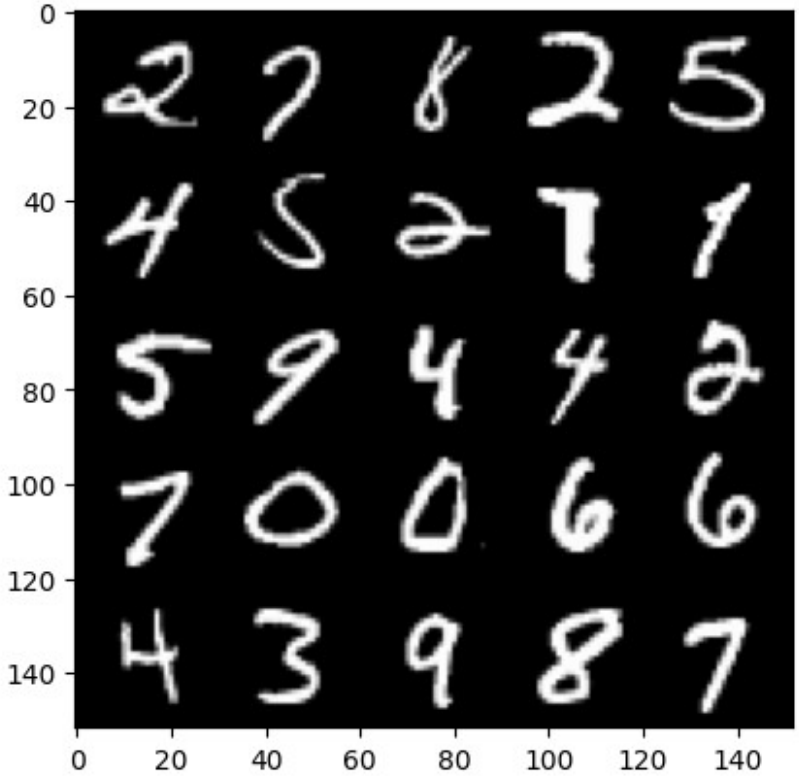




{"model_id":"e4845a4071684326b1c1d80e6f378d50","version_major":2,"version_minor":0}

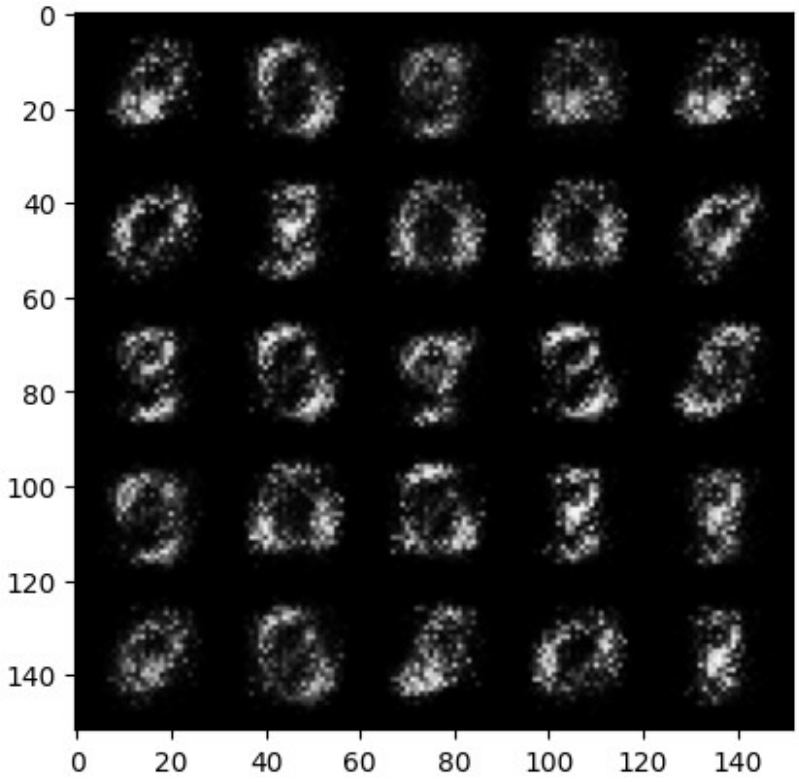
```
Epoch 6, step 3000: Generator loss: 1.89134983062744, discriminator loss: 0.17519412446022006
```

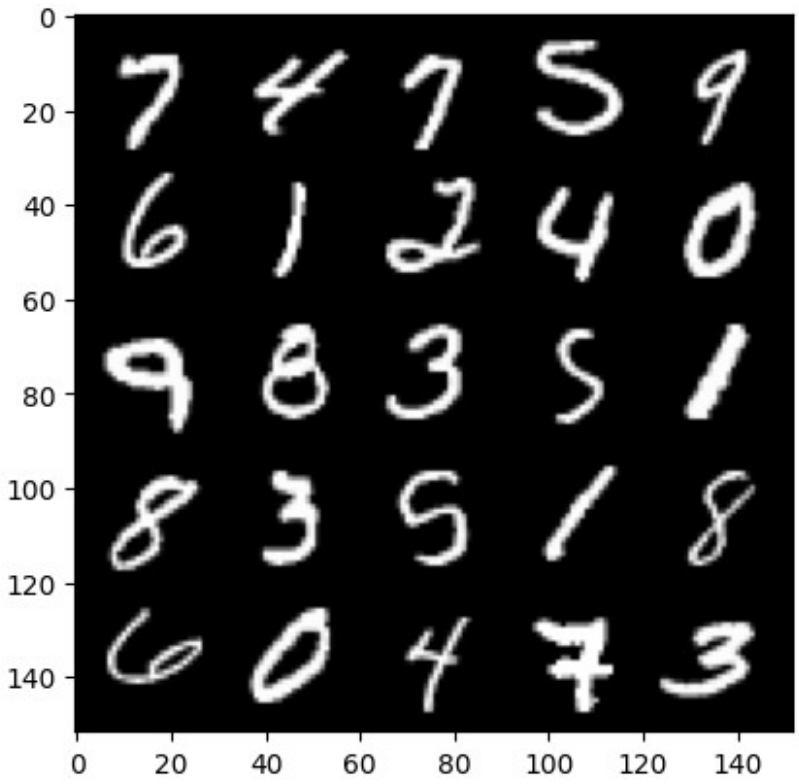

0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140


0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

{"model_id":"00bc4f0e965548258bff0ee2aee3eff2","version_major":2,"version_minor":0}

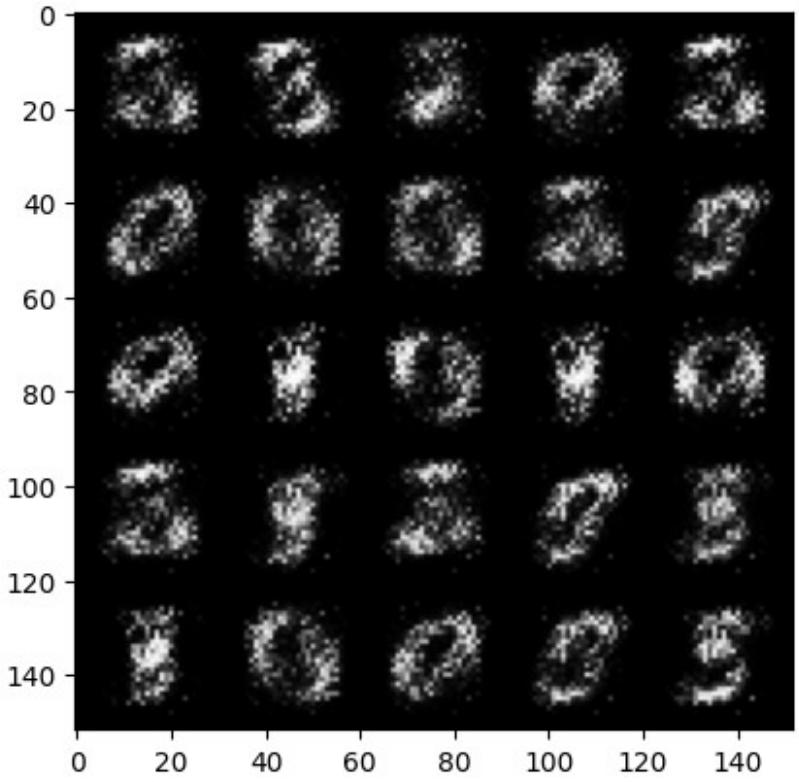
```
Epoch 7, step 3500: Generator loss: 2.279083190917967, discriminator loss: 0.13824221533536912
```

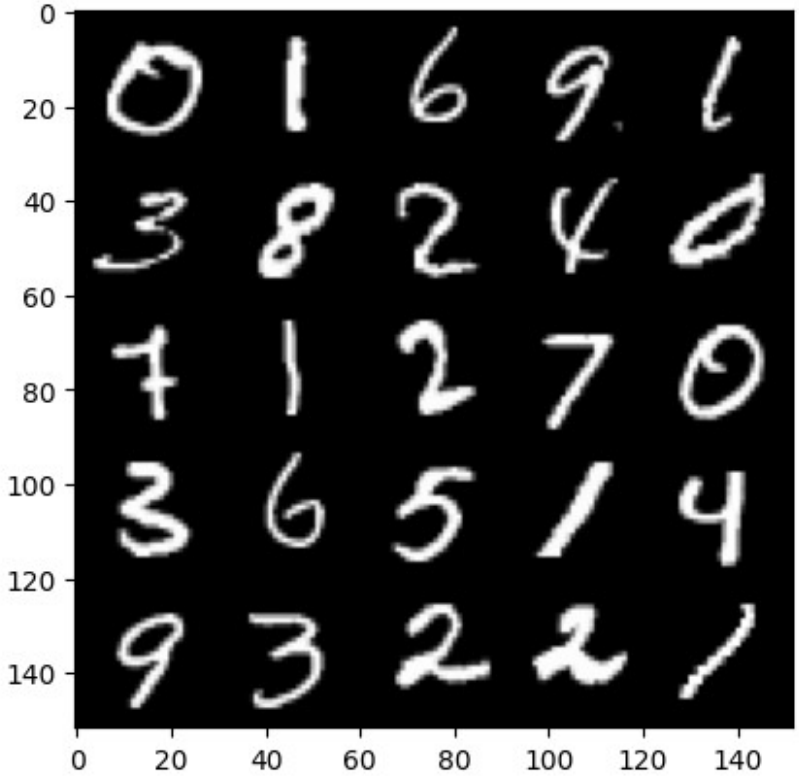




{"model_id":"228ab13ccb0d4990944c2e915397968c","version_major":2,"version_minor":0}

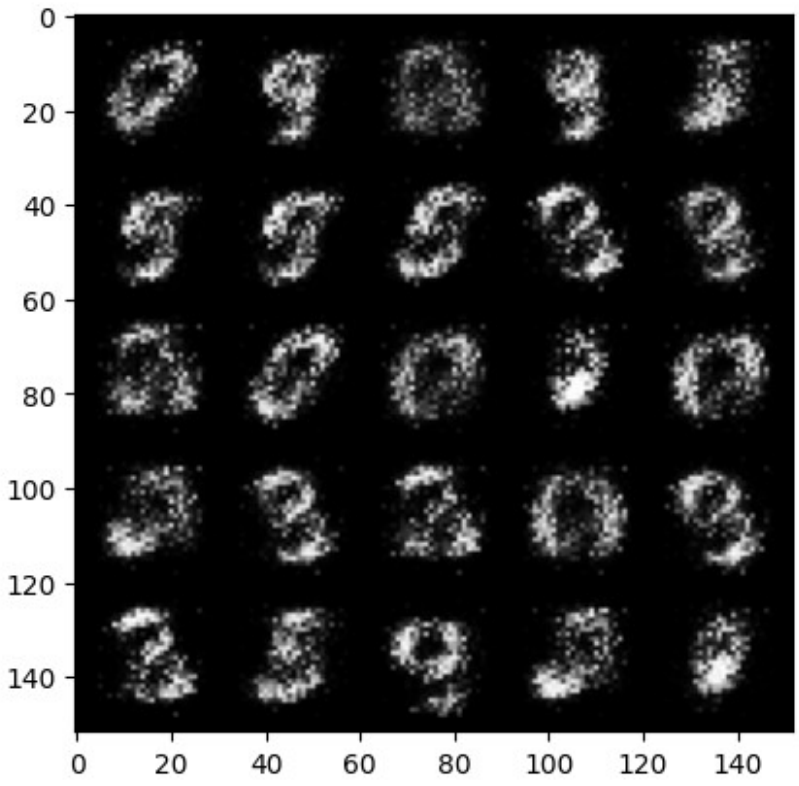
```
Epoch 8, step 4000: Generator loss: 2.7006059756278975, discriminator loss: 0.1107034523040056
```

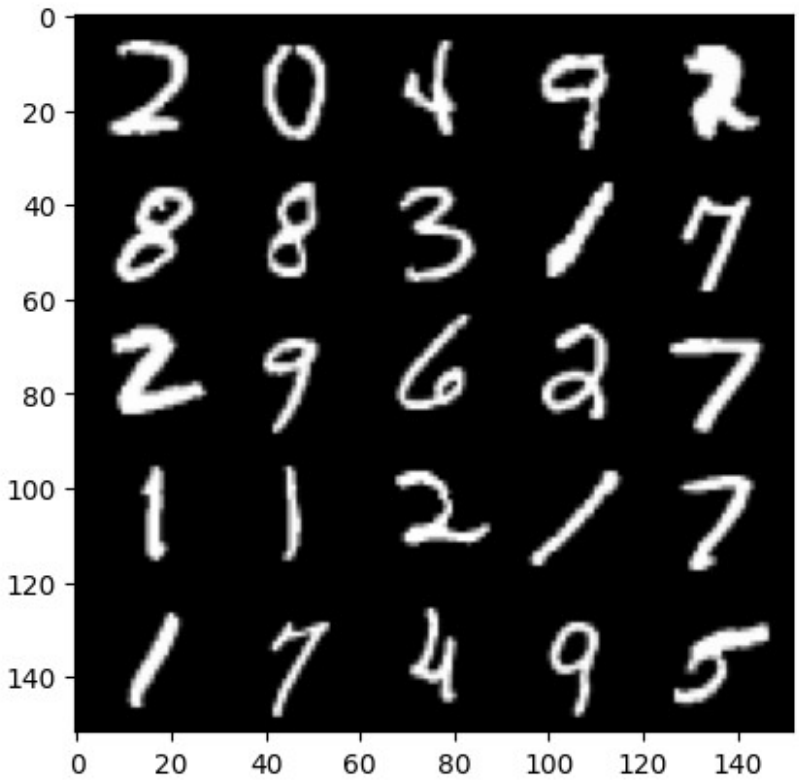

0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140


0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

{"model_id":"63c6d7e0103e43eeab2418e2c00c41bb","version_major":2,"version_minor":0}

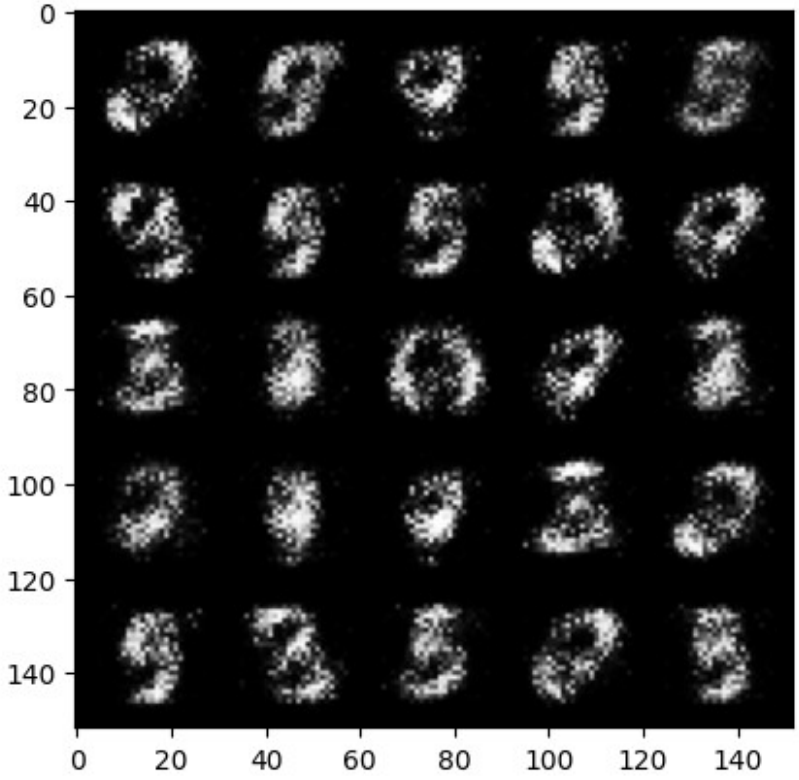
```
Epoch 9, step 4500: Generator loss: 3.1411079783439653, discriminator loss: 0.08360796292871243
```

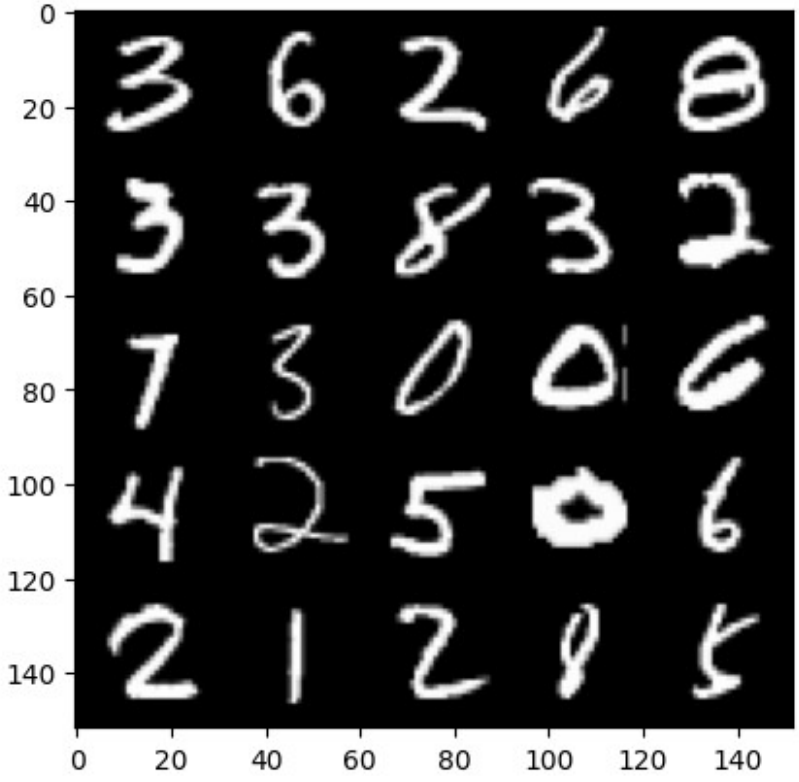




{"model_id":"6a660e7f5e34450796b333a5e9bb5ebe","version_major":2,"version_minor":0}

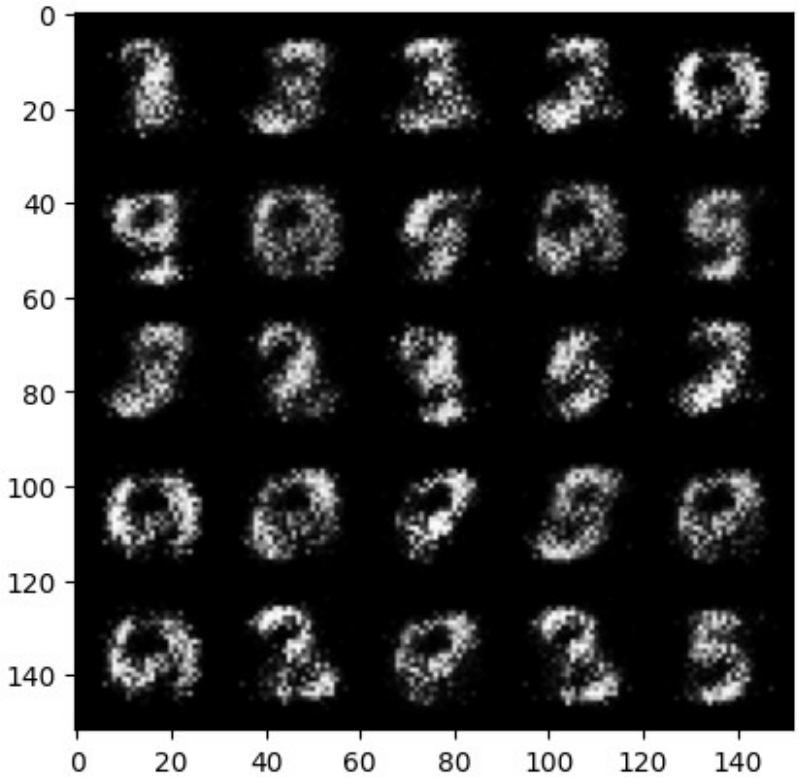
```
Epoch 10, step 5000: Generator loss: 3.4574751863479602, discriminator loss: 0.06851891132444146
```

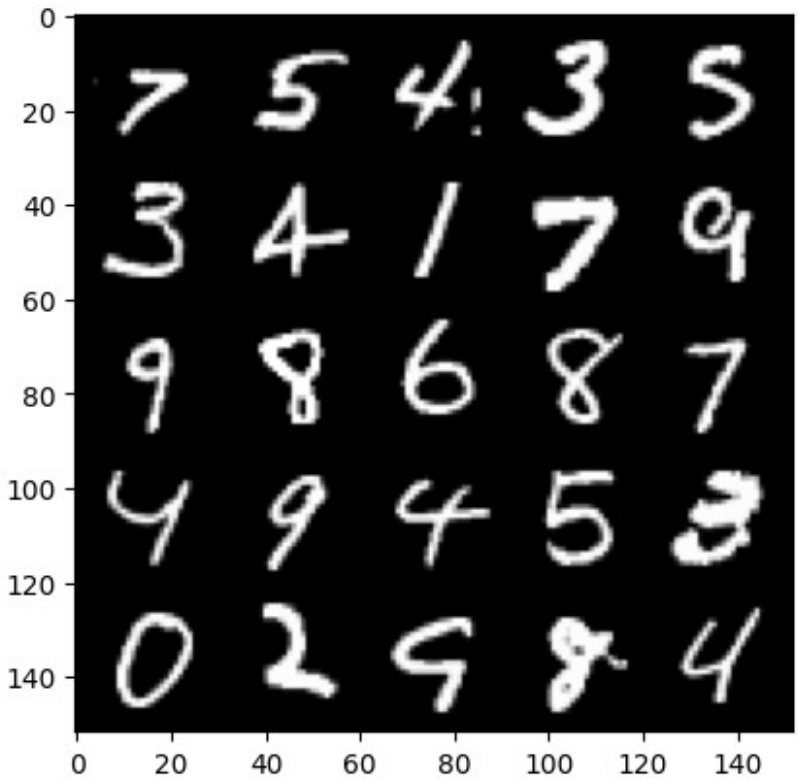

0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140


0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

{"model_id":"af8352936aa948fc82786bebf55d6585","version_major":2,"version_minor":0}

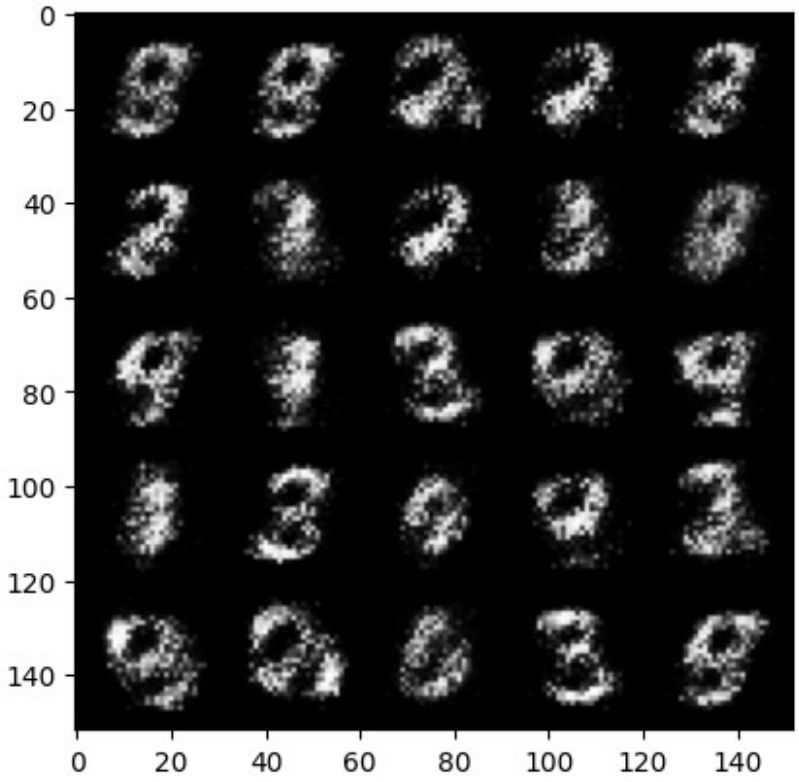
```
Epoch 11, step 5500: Generator loss: 3.886338778018952, discriminator loss: 0.05593983914703129
```

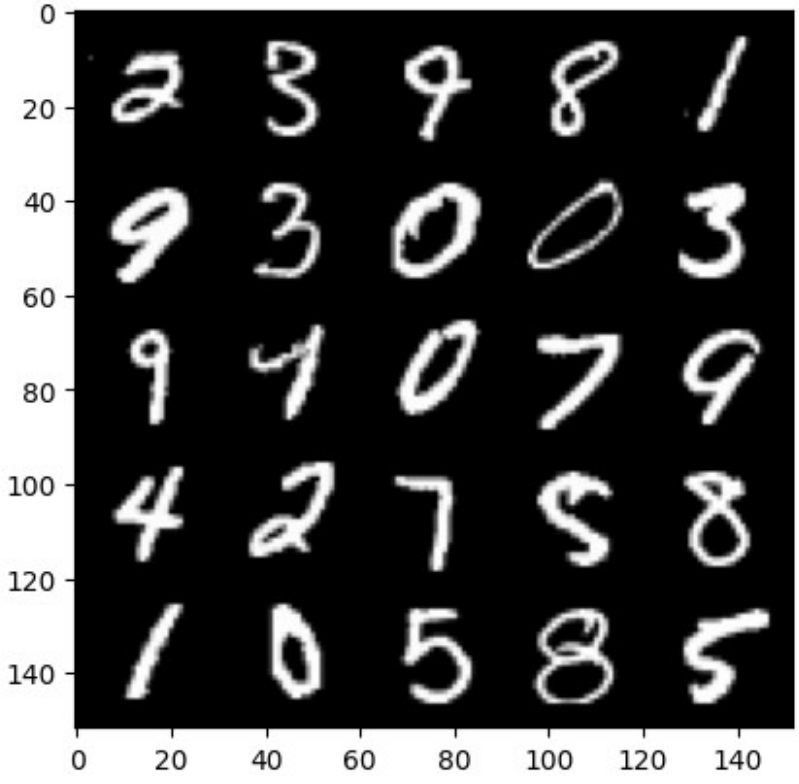




{"model_id":"365331a90d9340329a550b510da33a15","version_major":2,"version_minor":0}

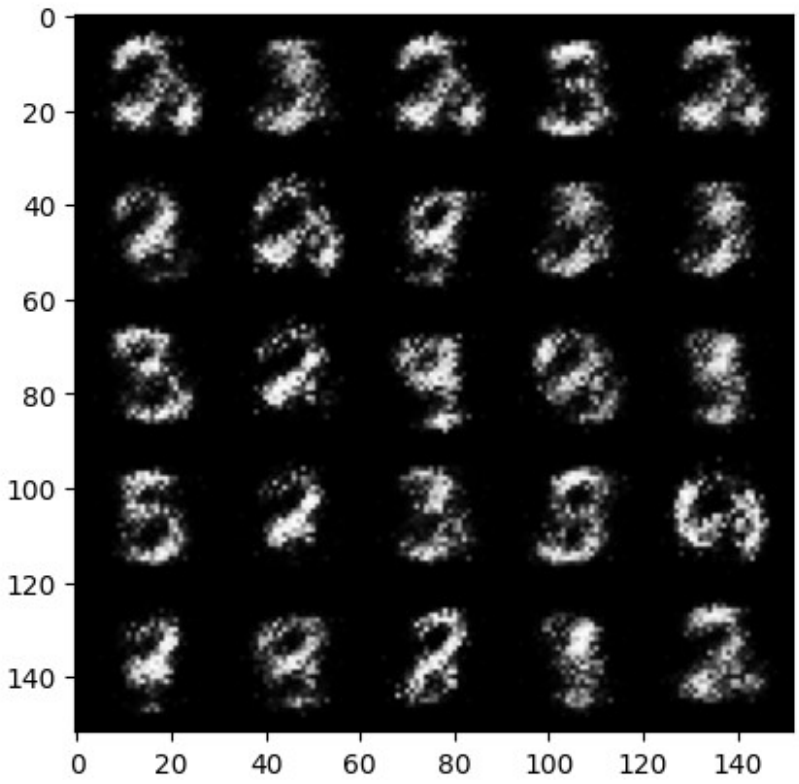
```
Epoch 12, step 6000: Generator loss: 3.9295068826675412, discriminator loss: 0.051135006938129654
```

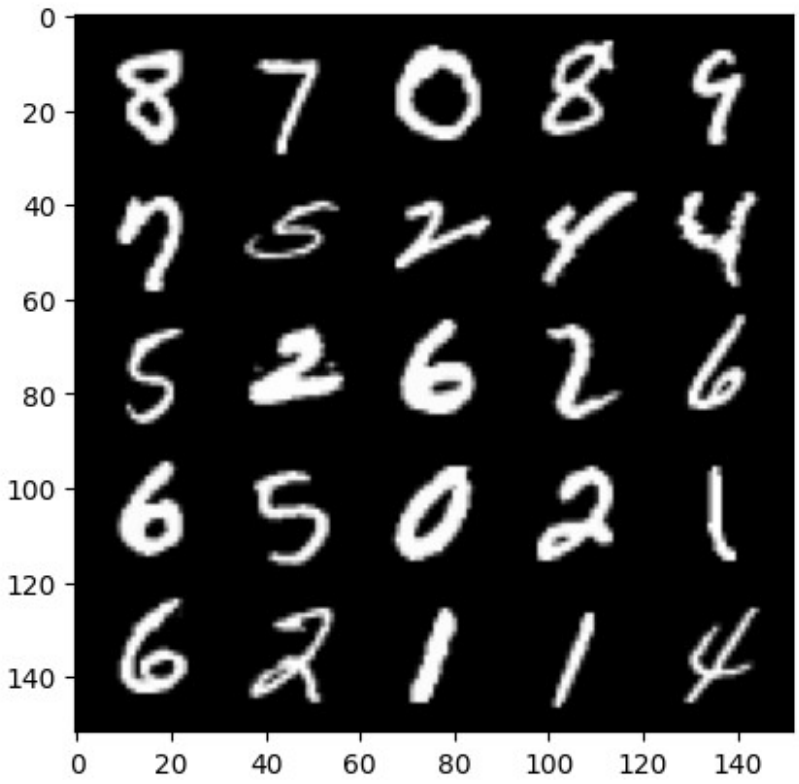

0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140


0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

{"model_id":"810c390e370243b2a87fb6e491f60513","version_major":2,"version_minor":0}

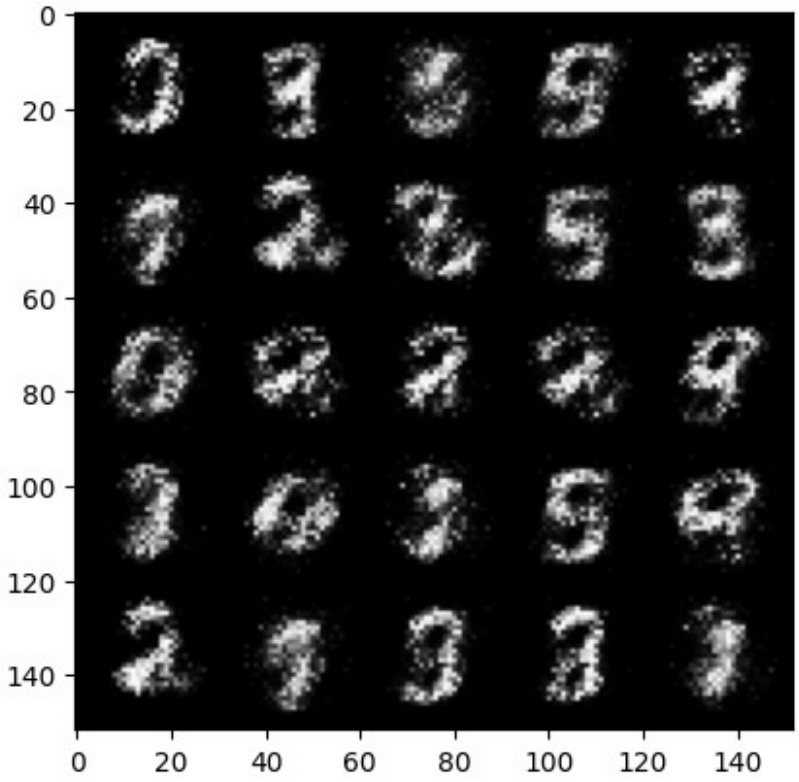
```
Epoch 13, step 6500: Generator loss: 4.1483565402030935, discriminator loss: 0.04690669849142433
```

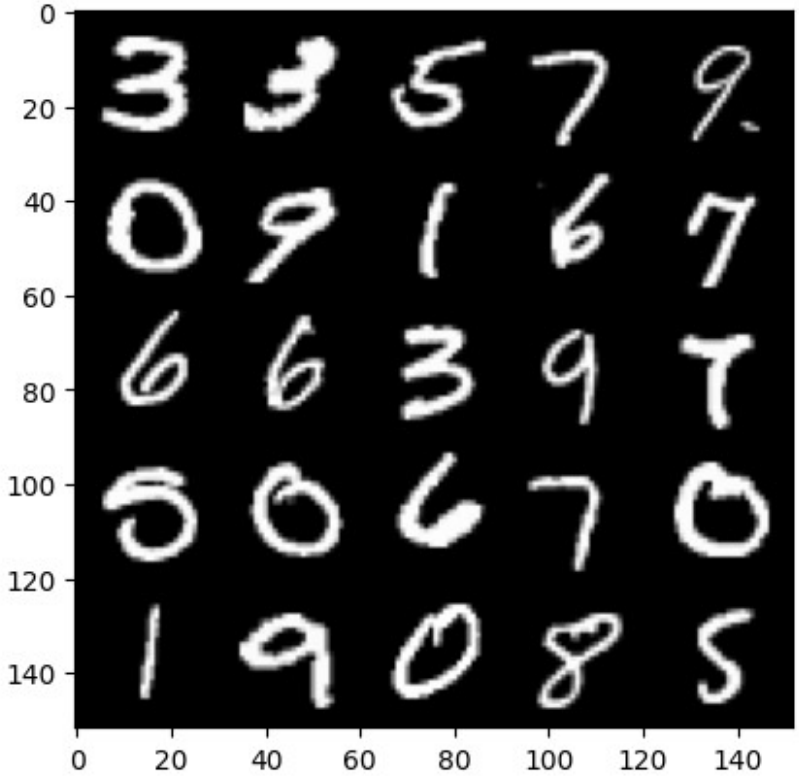




{"model_id":"6640638d673249908fa80b6f6f0fa91c","version_major":2,"version_minor":0}

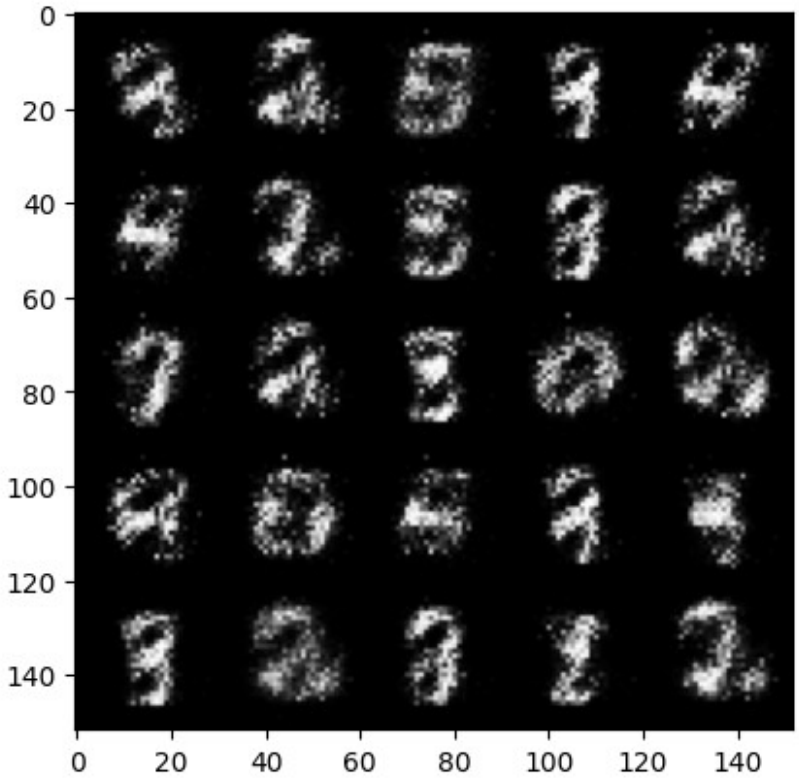
```
Epoch 14, step 7000: Generator loss: 4.1344814257621785, discriminator loss: 0.04442652777954932
```

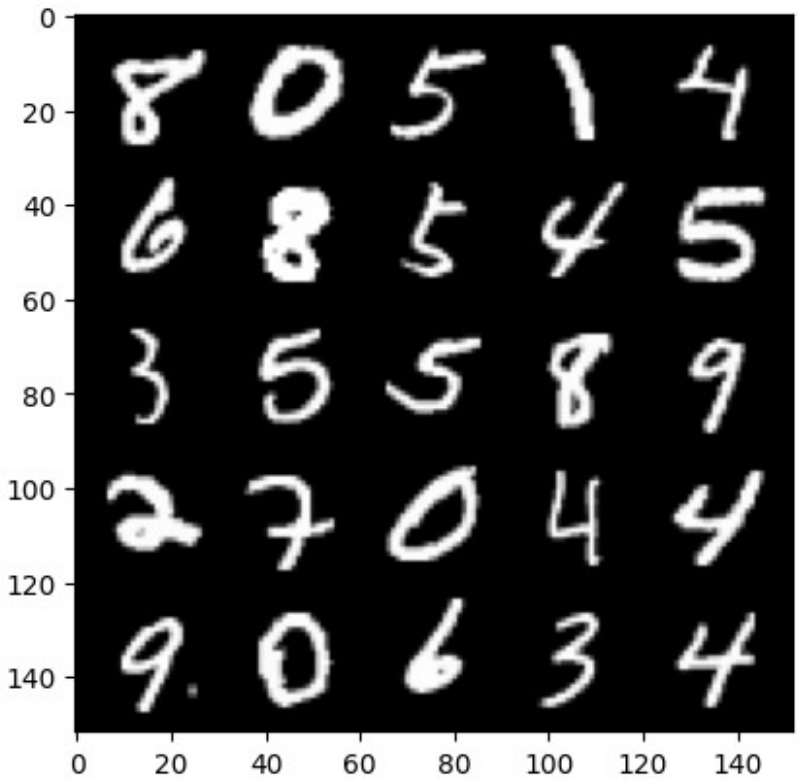

0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140


0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

{"model_id":"78210364a172418b966879a22f0c2df3","version_major":2,"version_minor":0}

```
Epoch 15, step 7500: Generator loss: 4.530333877563476, discriminator loss: 0.042298933070153005
```

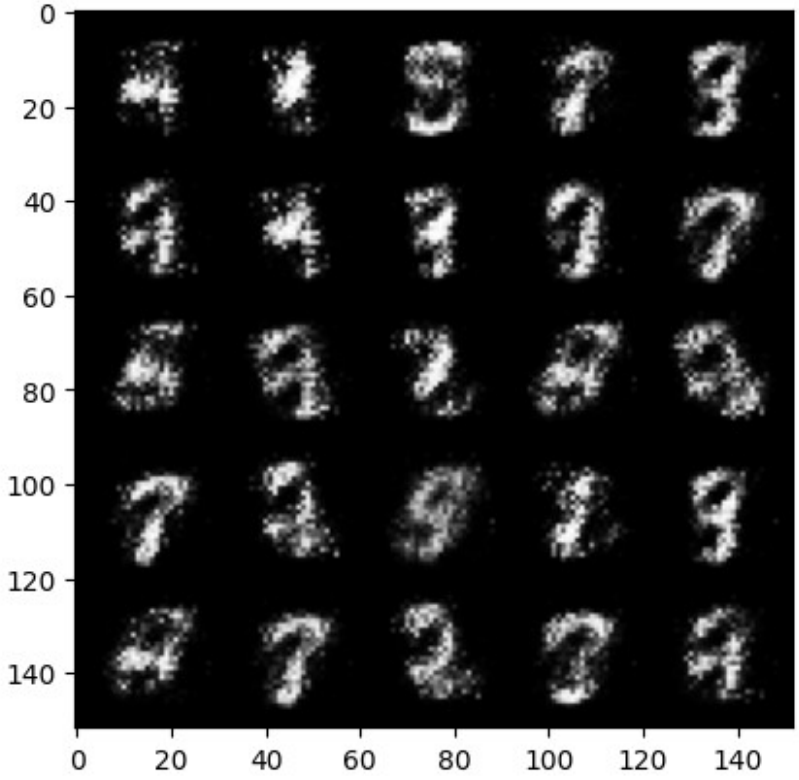


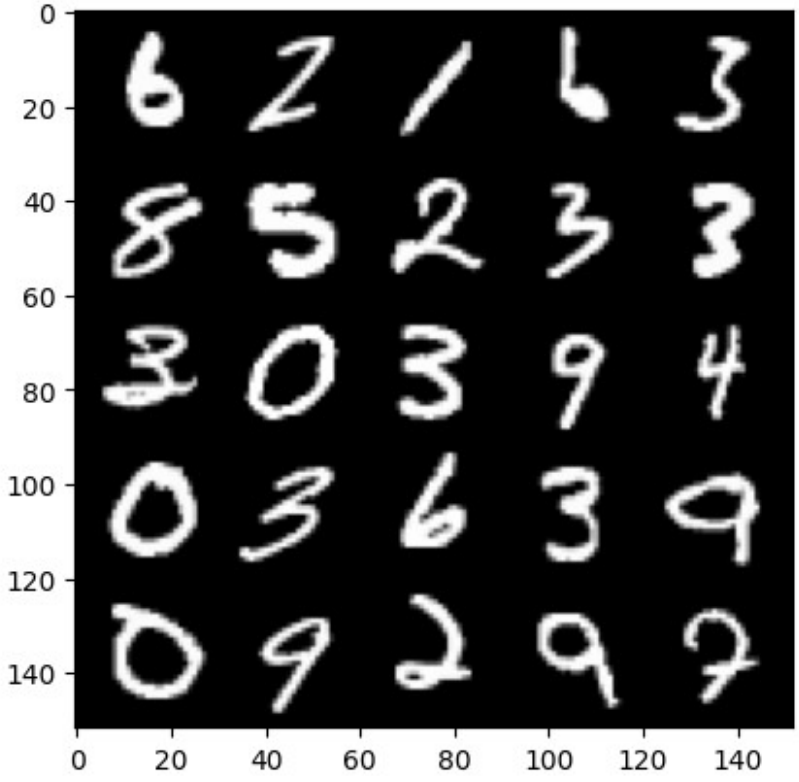


{"model_id":"46f9930b68d8414ca05fb4eeb4a73501","version_major":2,"version_minor":0}

{"model_id":"46c68ce1f4d9445e9e6eea6d1cddd155","version_major":2,"version_minor":0}

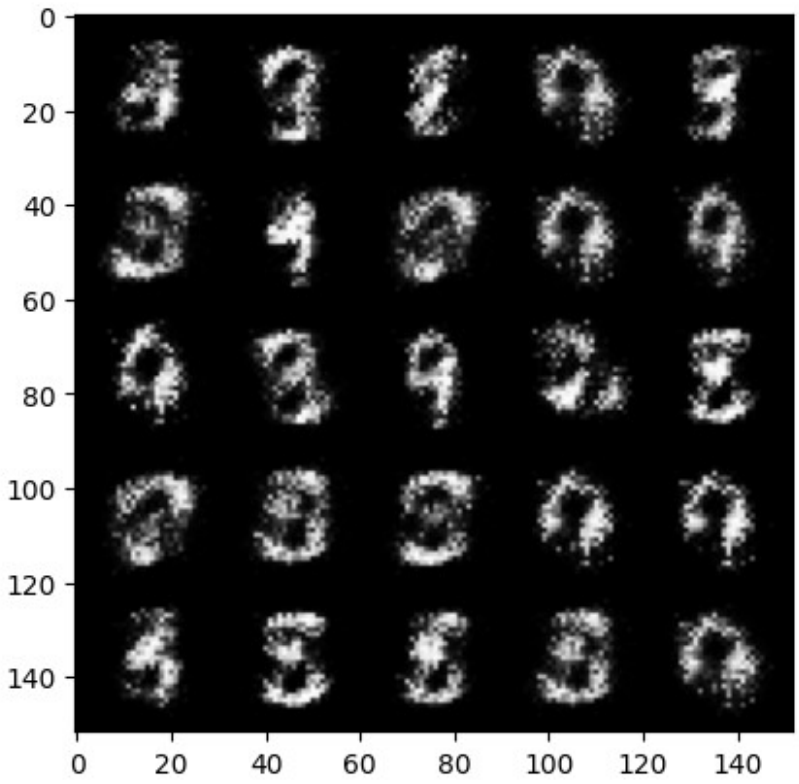
```
Epoch 17, step 8000: Generator loss: 4.455792129993438, discriminator loss: 0.05031867891550061
```

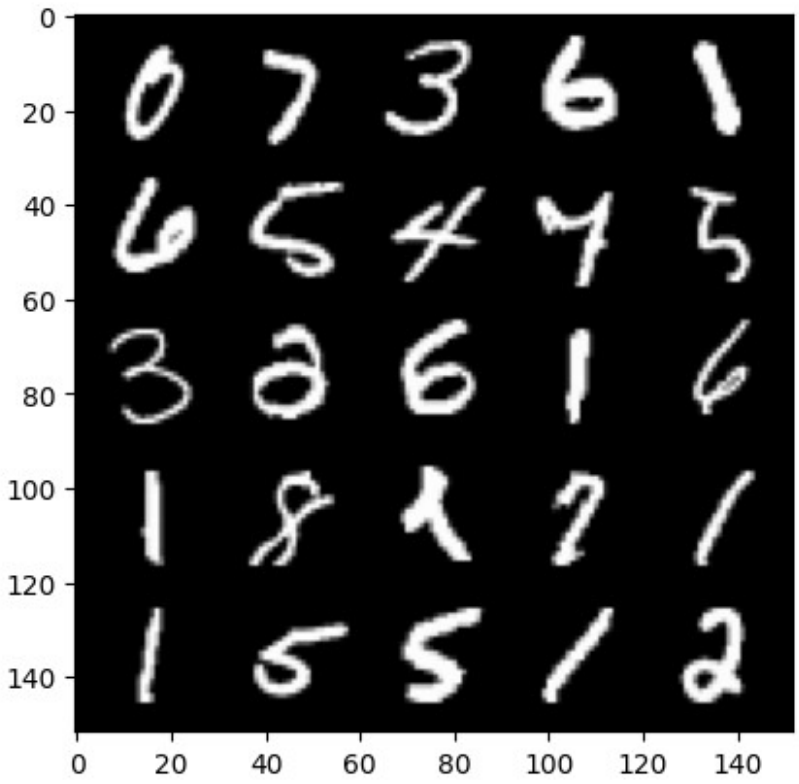

0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140


0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

{"model_id":"0b69ac2afd4c48ccb55cbc466f9d3428","version_major":2,"version_minor":0}

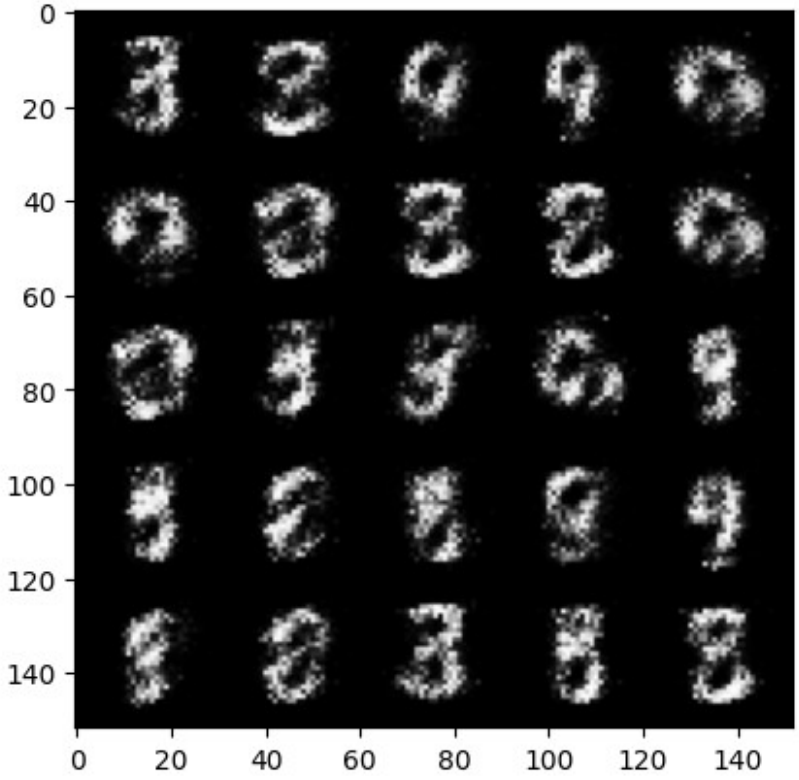
```
Epoch 18, step 8500: Generator loss: 4.445554067134859, discriminator loss: 0.050166048232465996
```

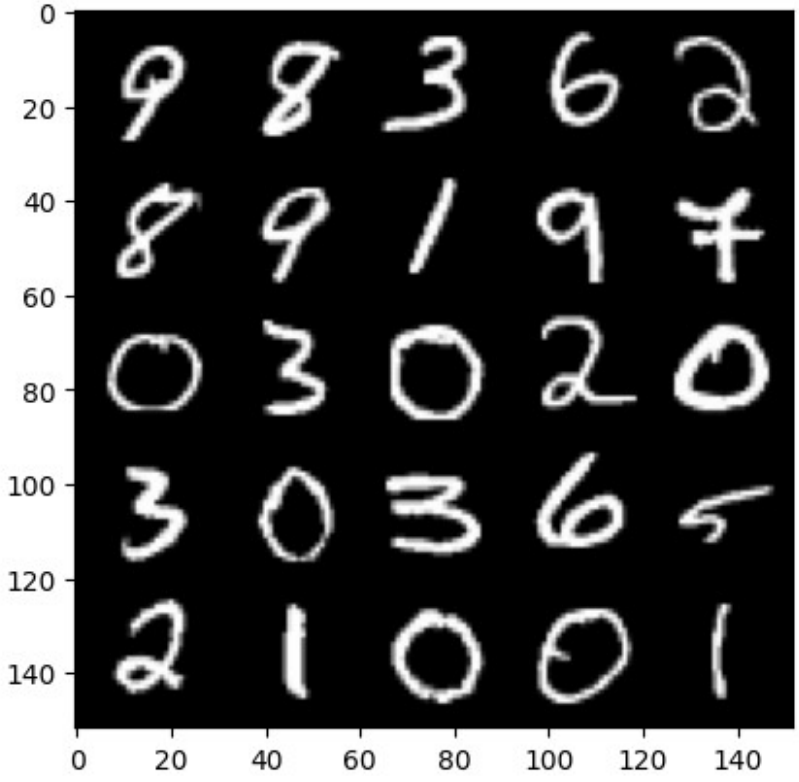




{"model_id":"8703f3265e36471599957d9ae5ff3fb0","version_major":2,"version_minor":0}

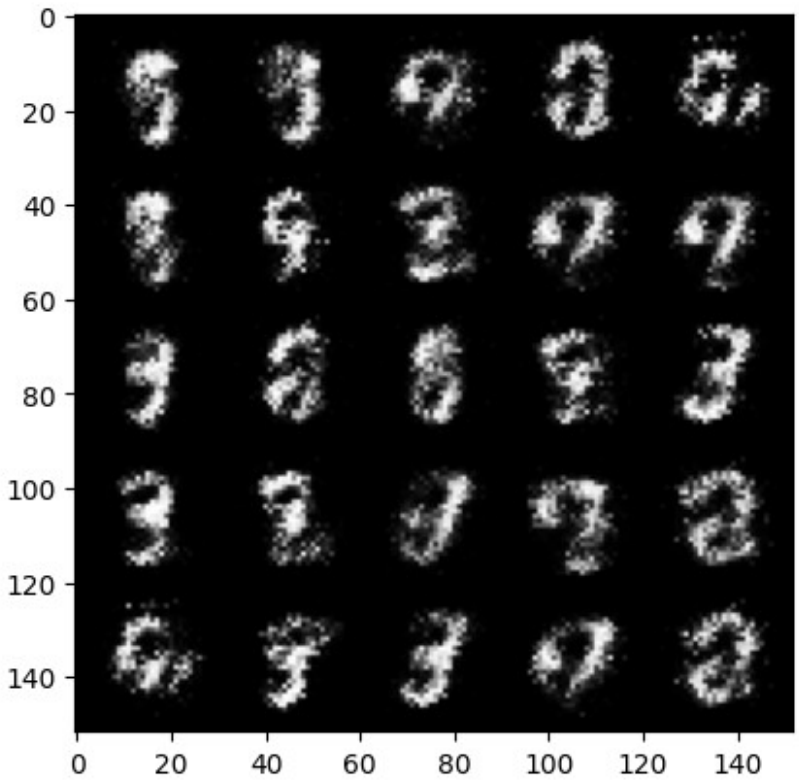
```
Epoch 19, step 9000: Generator loss: 4.408470261096955, discriminator loss: 0.054042059533297986
```

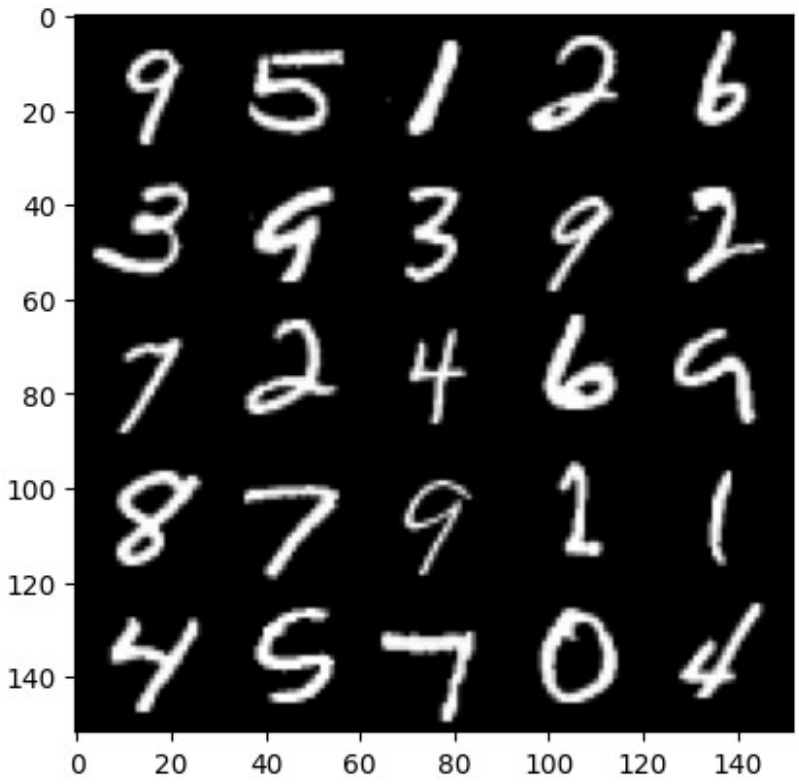

0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140


0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

{"model_id":"ee25861cc1d444418b2ac76895d488ff","version_major":2,"version_minor":0}

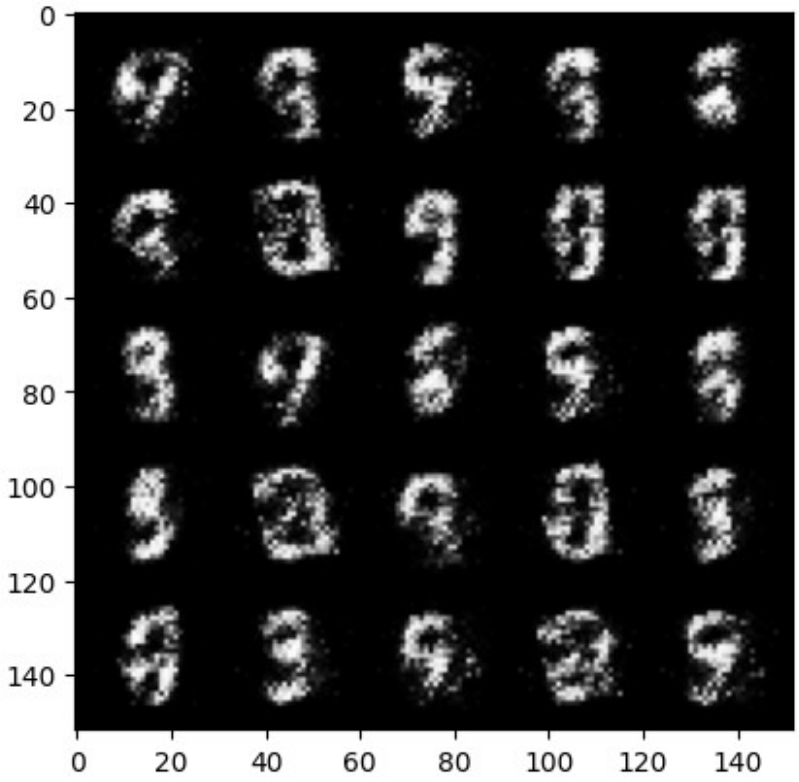
```
Epoch 20, step 9500: Generator loss: 4.502455300331117, discriminator loss: 0.049759022217243924
```

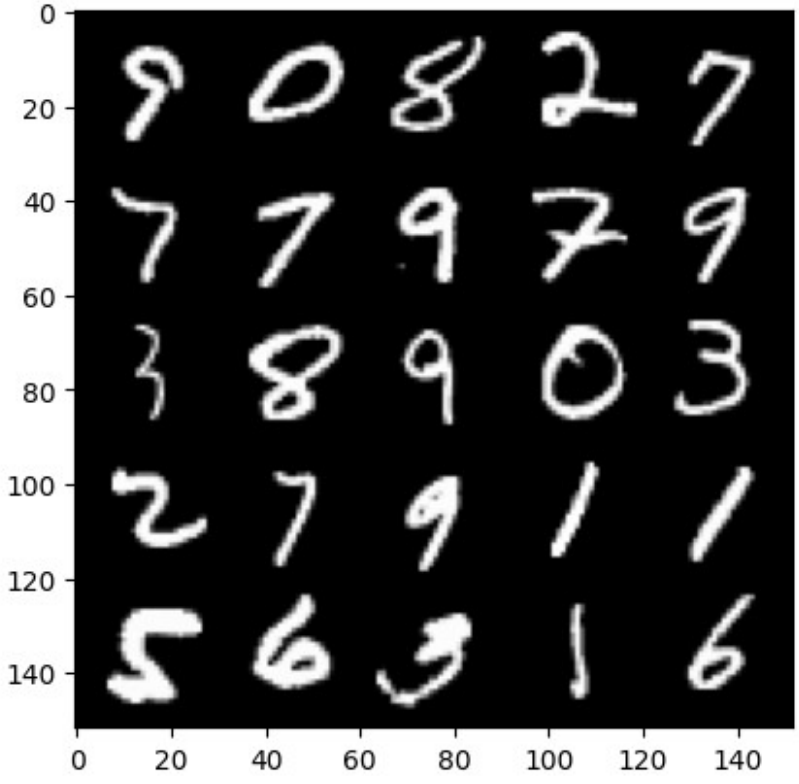




{"model_id":"fca8b4ea5c974d4ea89dca00b36806f4","version_major":2,"version_minor":0}

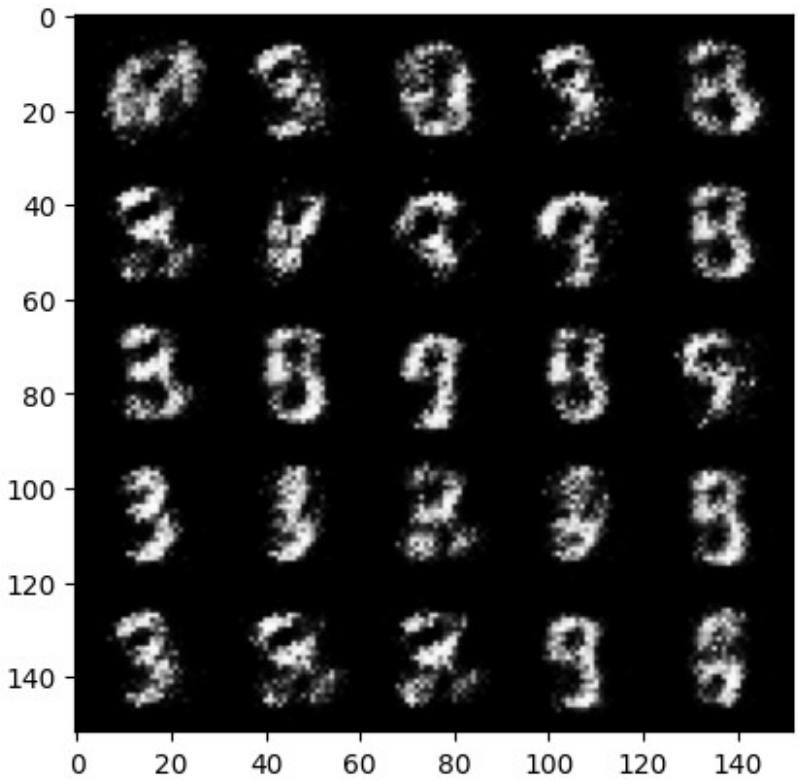
```
Epoch 21, step 10000: Generator loss: 4.439548857688901, discriminator loss: 0.049698601976037066
```

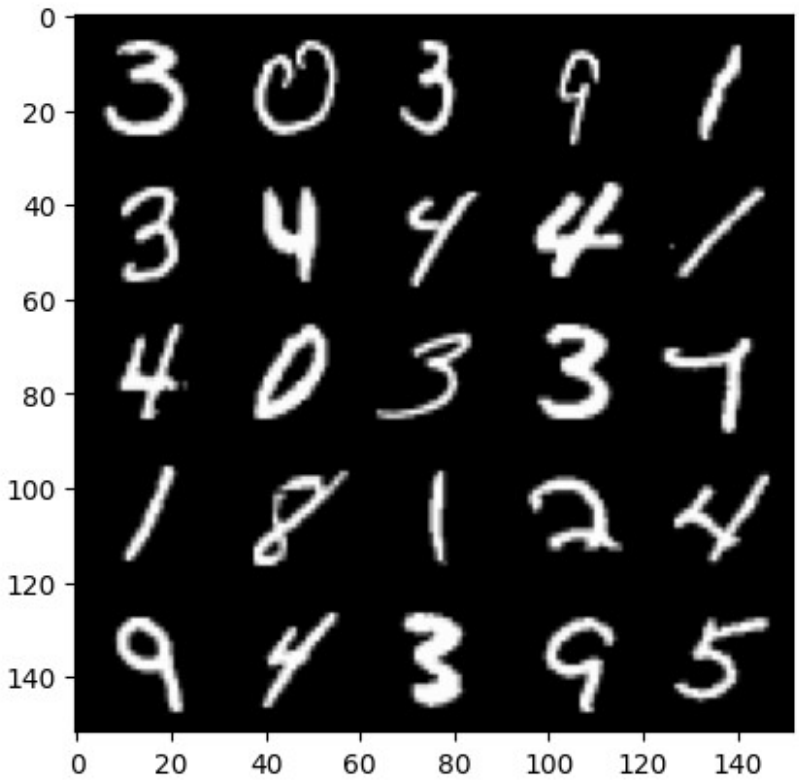

0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140


0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

{"model_id":"ff446c0819b2407f8d28f35981f81c08","version_major":2,"version_minor":0}

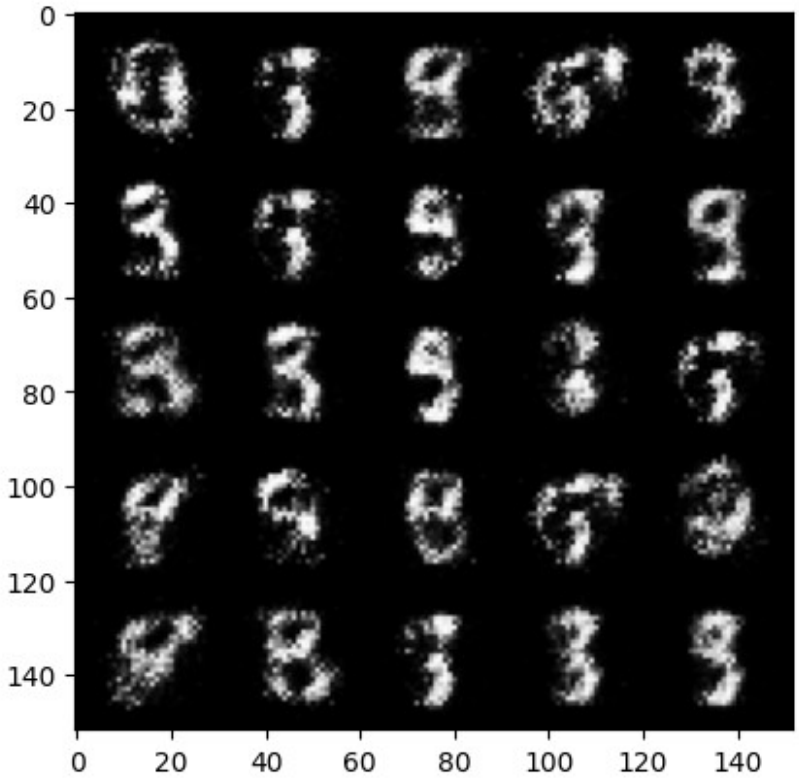
```
Epoch 22, step 10500: Generator loss: 4.5501135087013225,
discriminator loss: 0.054854219201952215
```

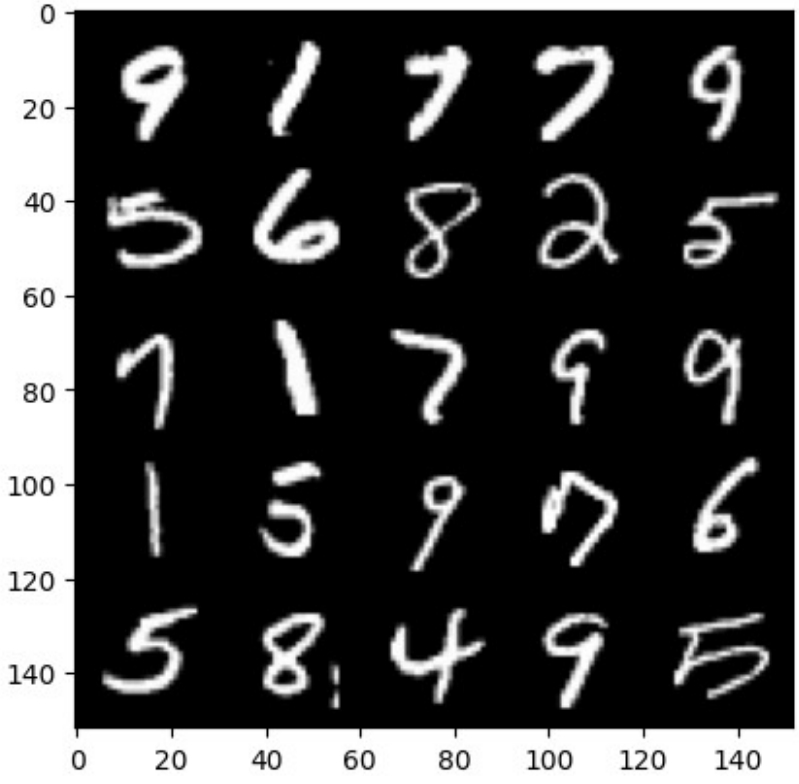




{"model_id":"558a5970e32b43ca8c1be7097de3067a","version_major":2,"version_minor":0}

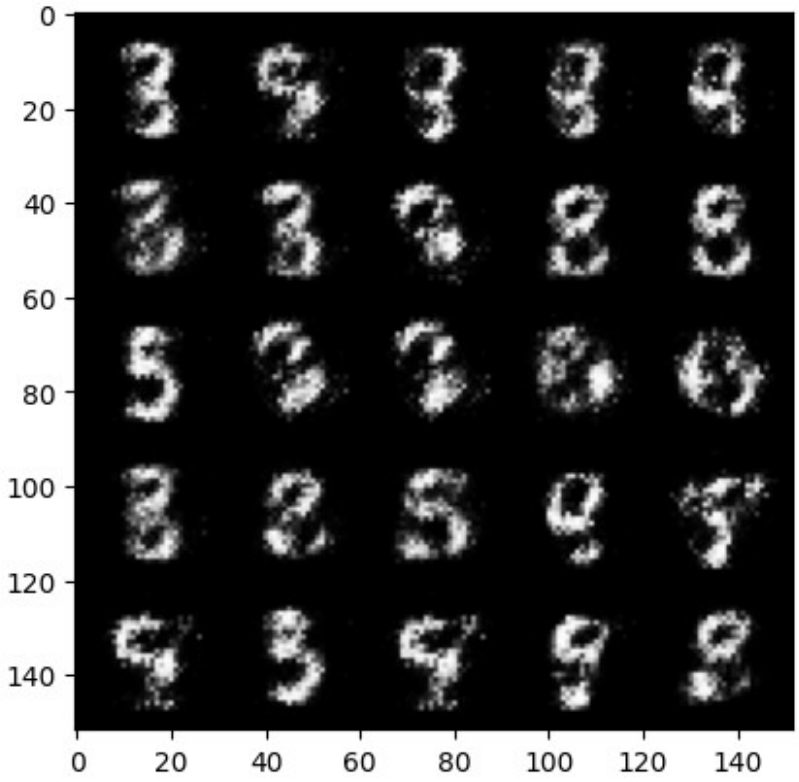
```
Epoch 23, step 11000: Generator loss: 4.629221978187568, discriminator loss: 0.05210746140033009
```

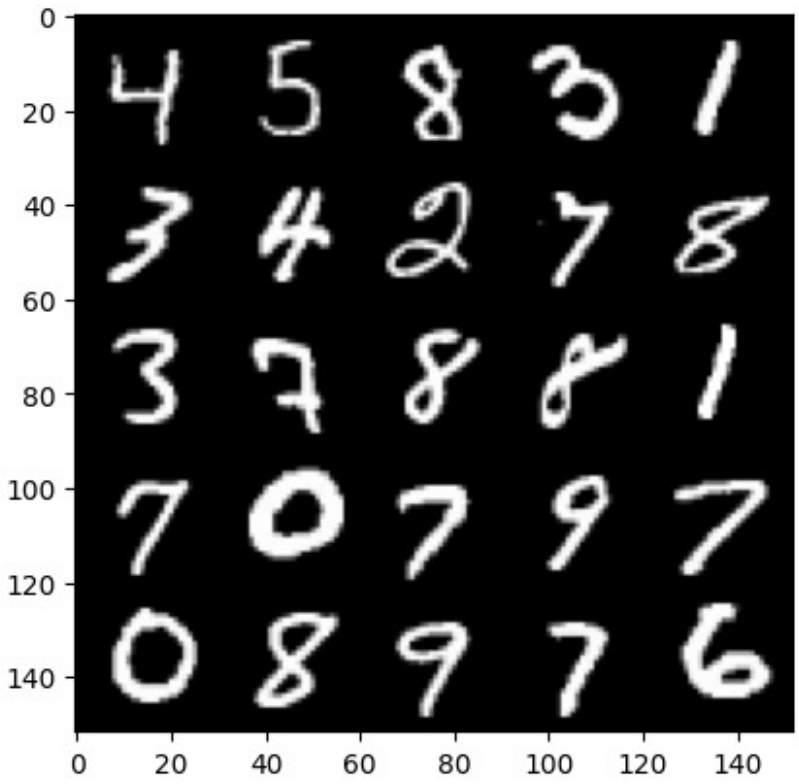

0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140


0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

{"model_id":"f757b25d717248bc89897b00e4cc8b79","version_major":2,"version_minor":0}

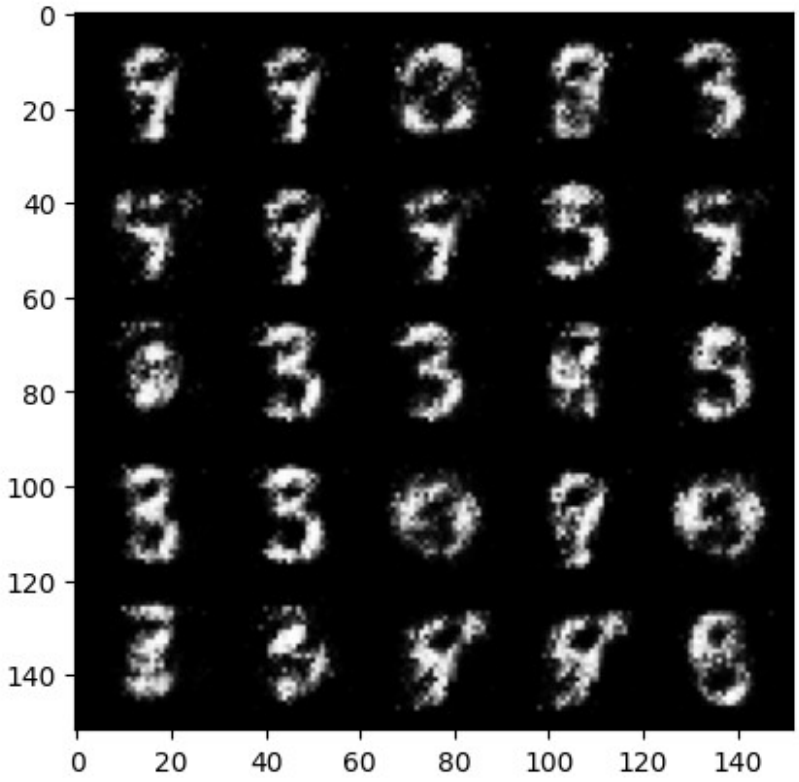
```
Epoch 24, step 11500: Generator loss: 4.454098214149476, discriminator loss: 0.0663463540636003
```

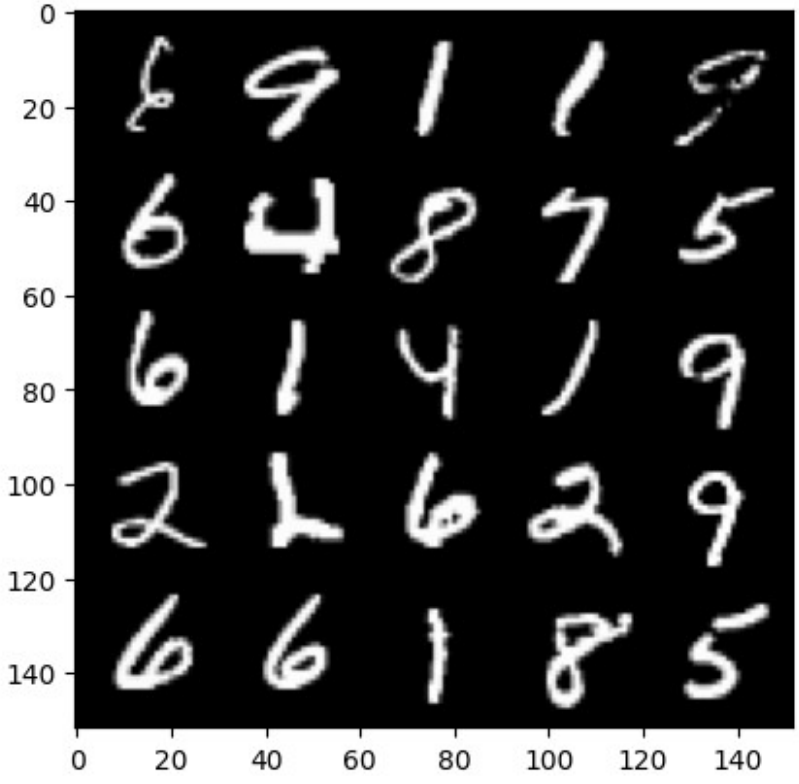




{"model_id":"4ece8d2a8e574cec9e392fdeb538904d","version_major":2,"version_minor":0}

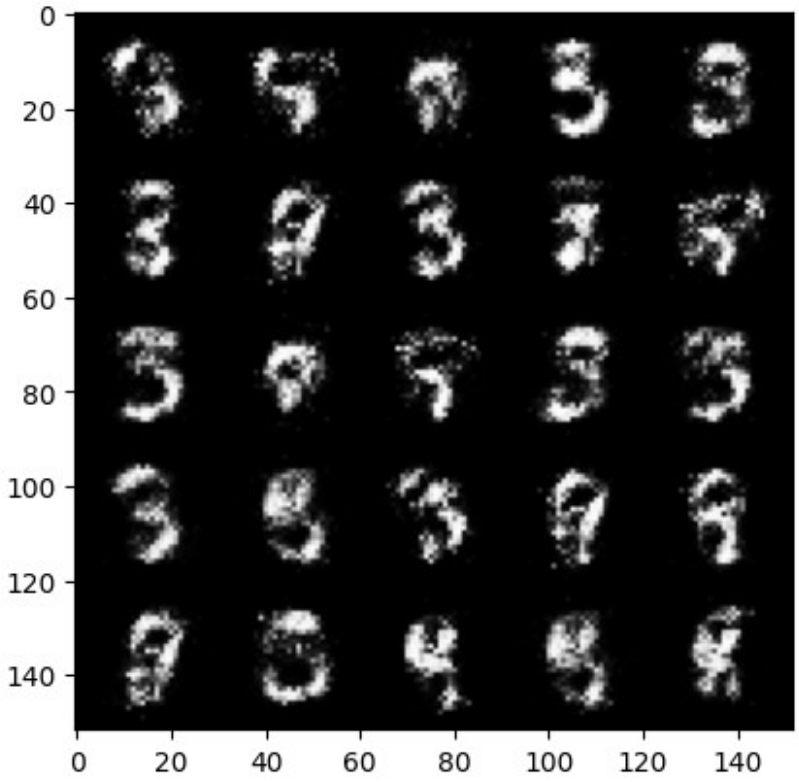
```
Epoch 25, step 12000: Generator loss: 4.33309952449798, discriminator loss: 0.062295961260795576
```

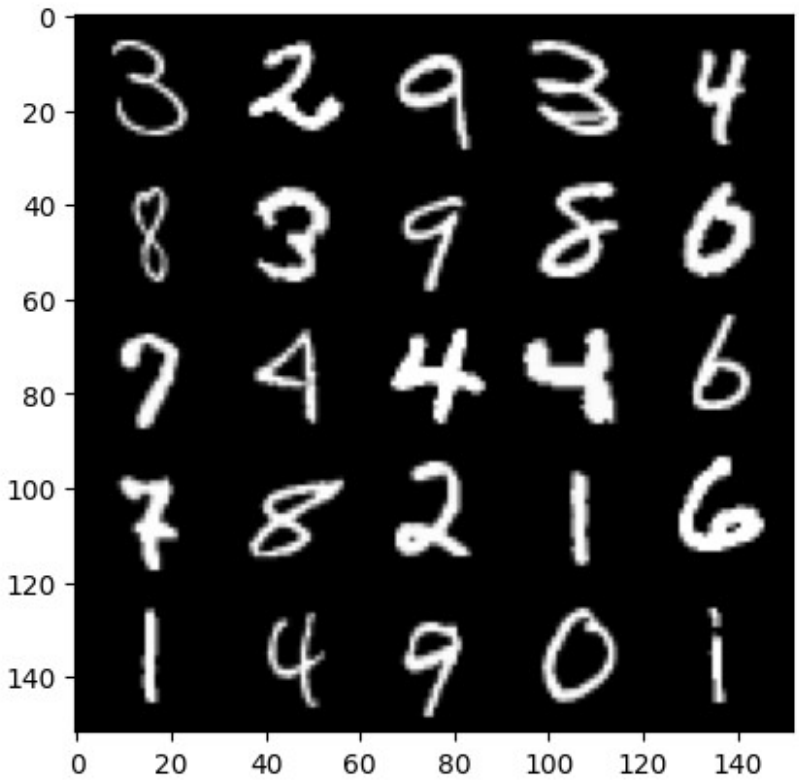

0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140


0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

{"model_id":"568eb58dccc44ea6a647cf723930bf53","version_major":2,"version_minor":0}

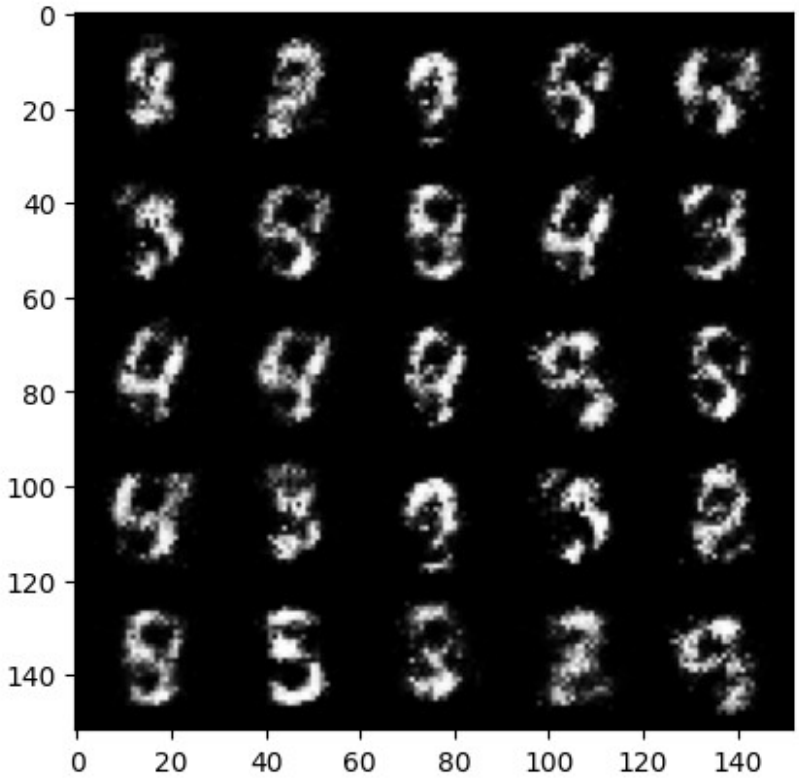
```
Epoch 26, step 12500: Generator loss: 4.399031805992123, discriminator loss: 0.06753970164805652
```

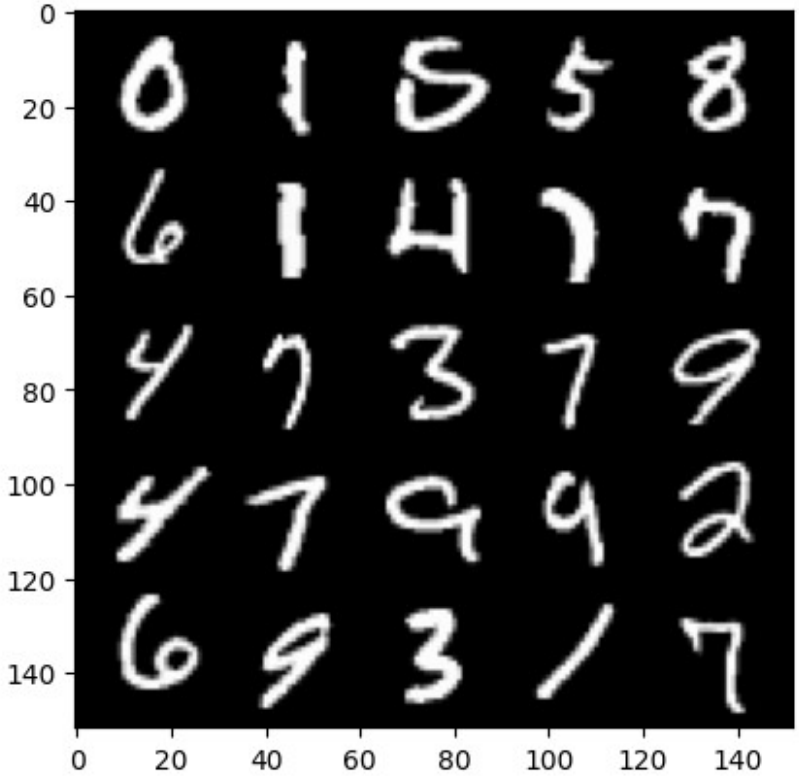




{"model_id":"ce23b9298f5d4b6a9263f3925a68948b","version_major":2,"version_minor":0}

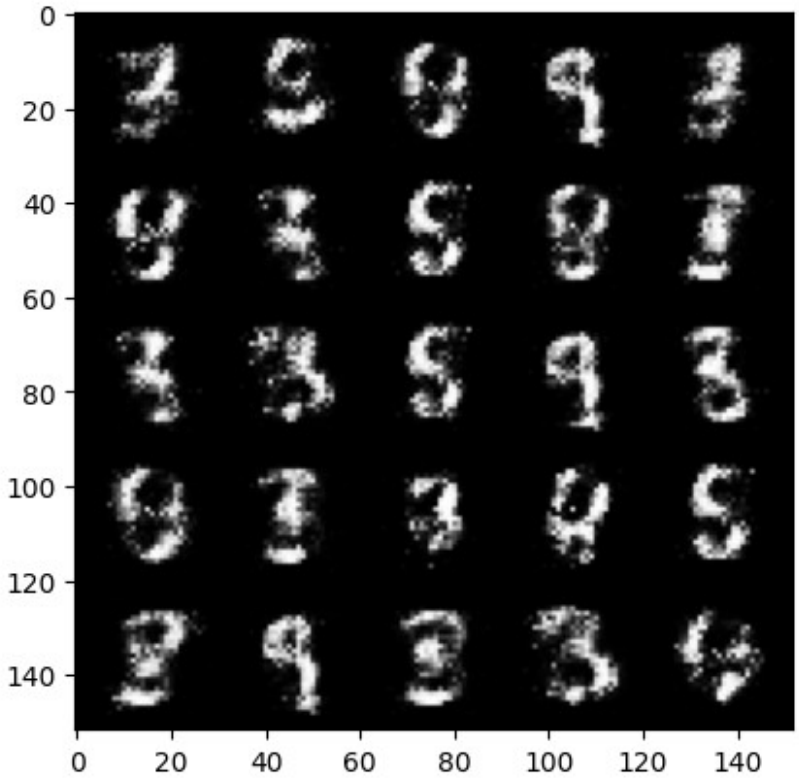
```
Epoch 27, step 13000: Generator loss: 4.351641575813293, discriminator loss: 0.07828917152807117
```

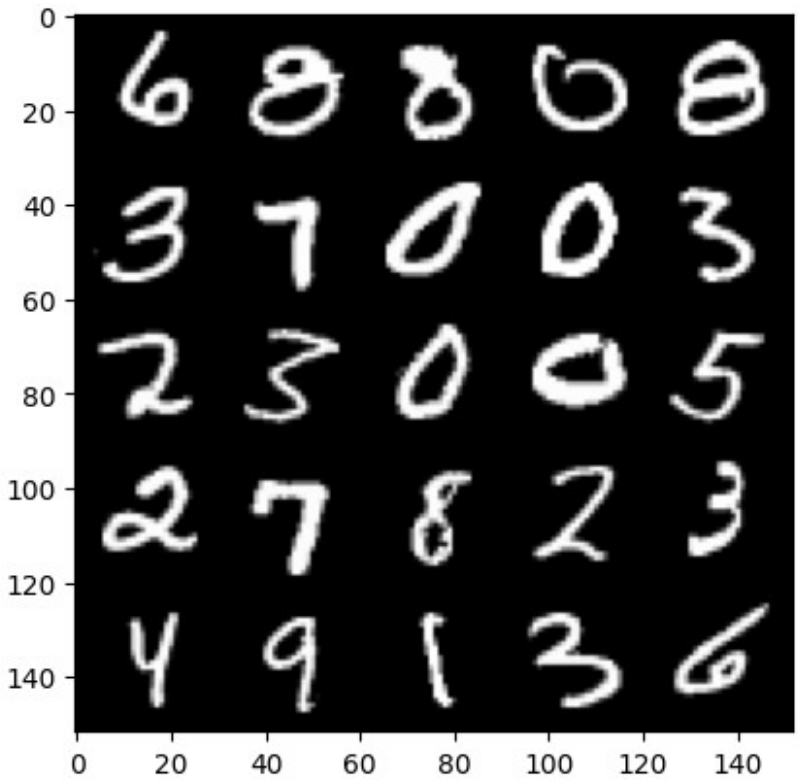

0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140


0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

{"model_id":"b2fbcfe7a7794af5a6dc94d01d2f1756","version_major":2,"version_minor":0}

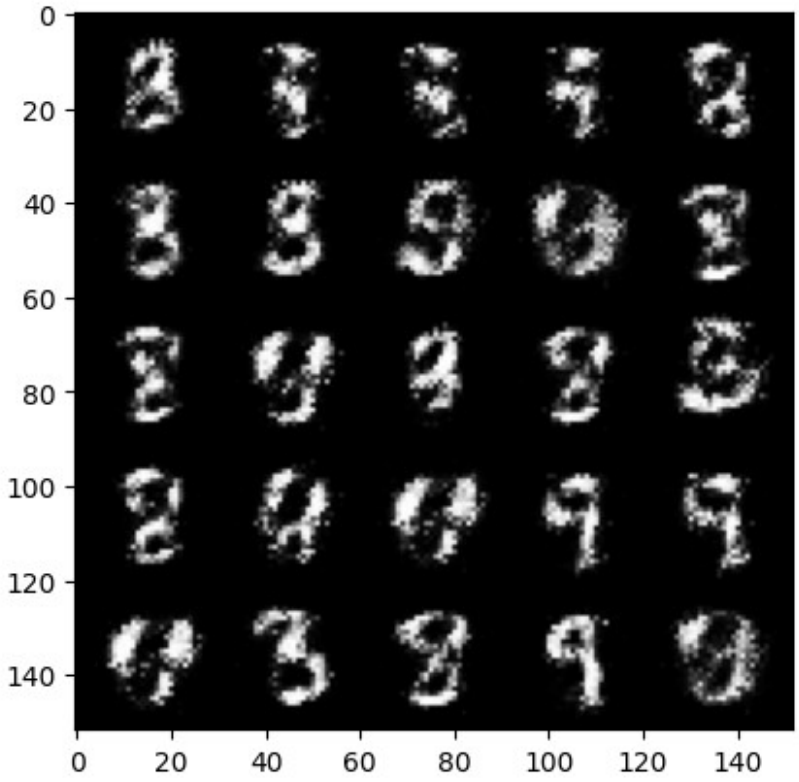
```
Epoch 28, step 13500: Generator loss: 3.978682627677914, discriminator loss: 0.09179804474860428
```

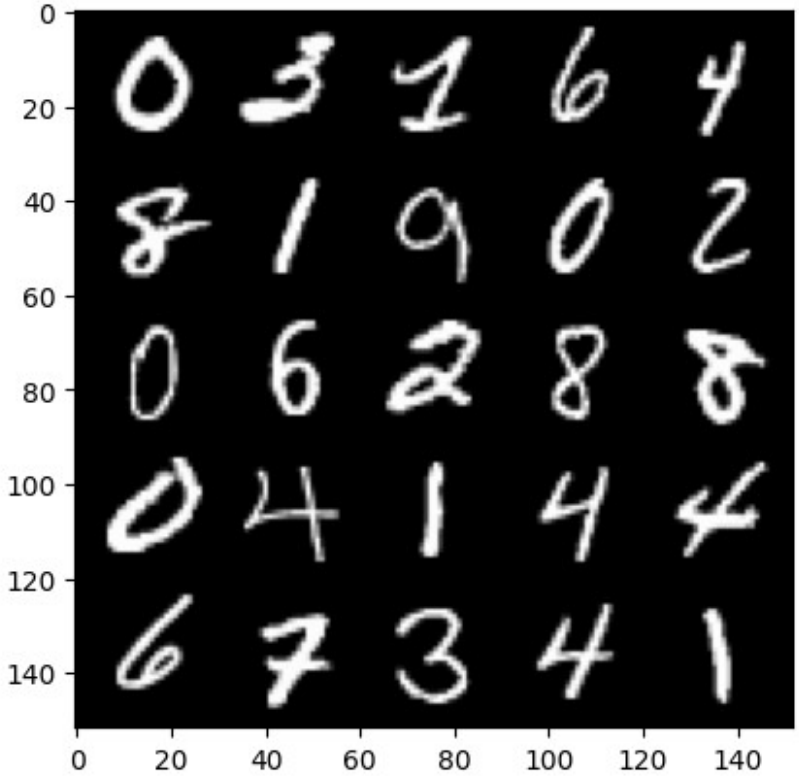




{"model_id":"7e26c4dc2c1c41a2abce4a7617795adf","version_major":2,"version_minor":0}

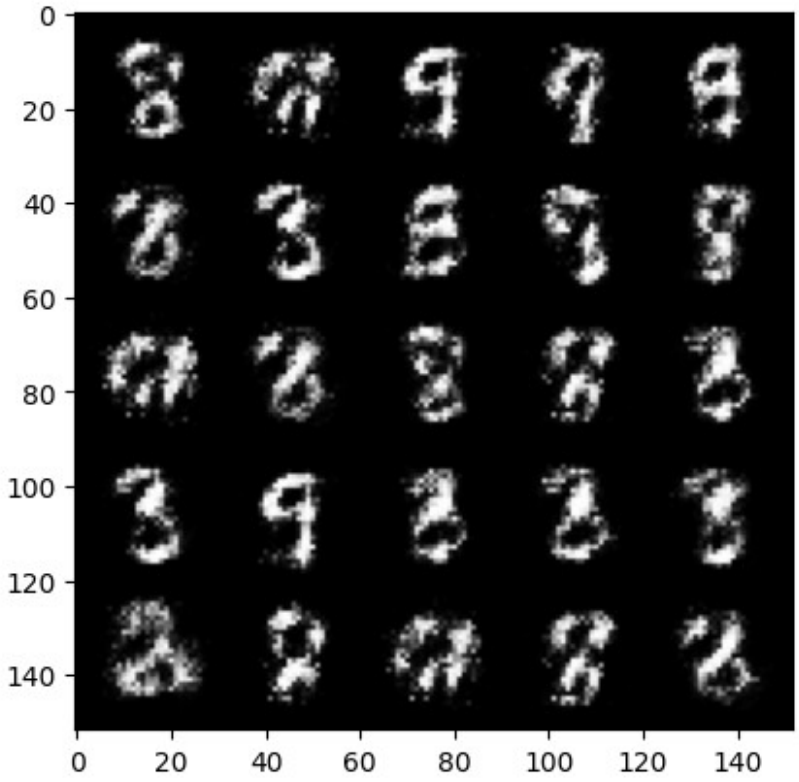
```
Epoch 29, step 14000: Generator loss: 4.076661603450776, discriminator loss: 0.0844755531549454
```

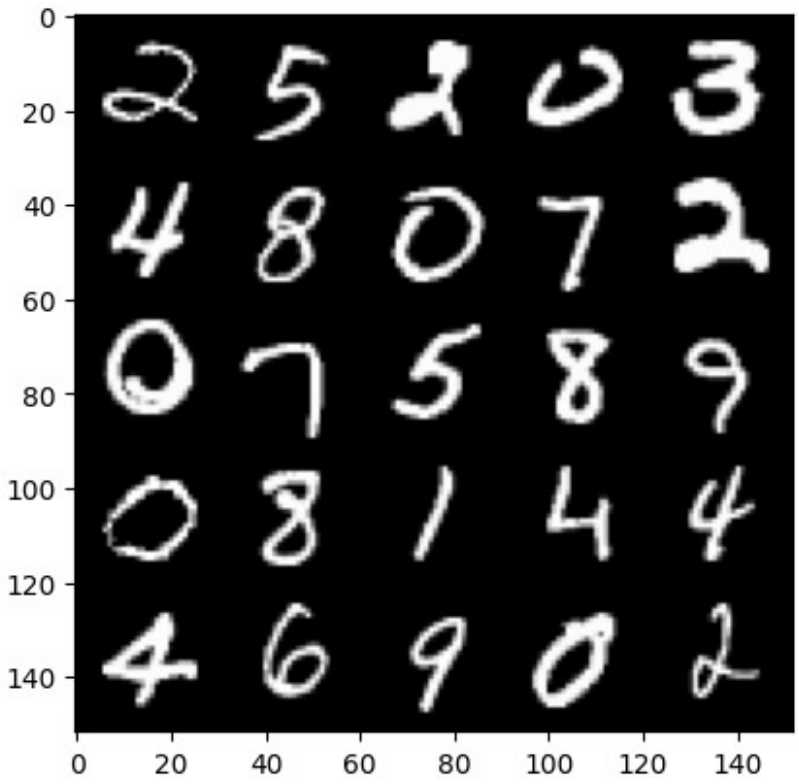

0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140


0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

{"model_id":"c68a196e18704668b0ccbb5eaa1887b0","version_major":2,"version_minor":0}

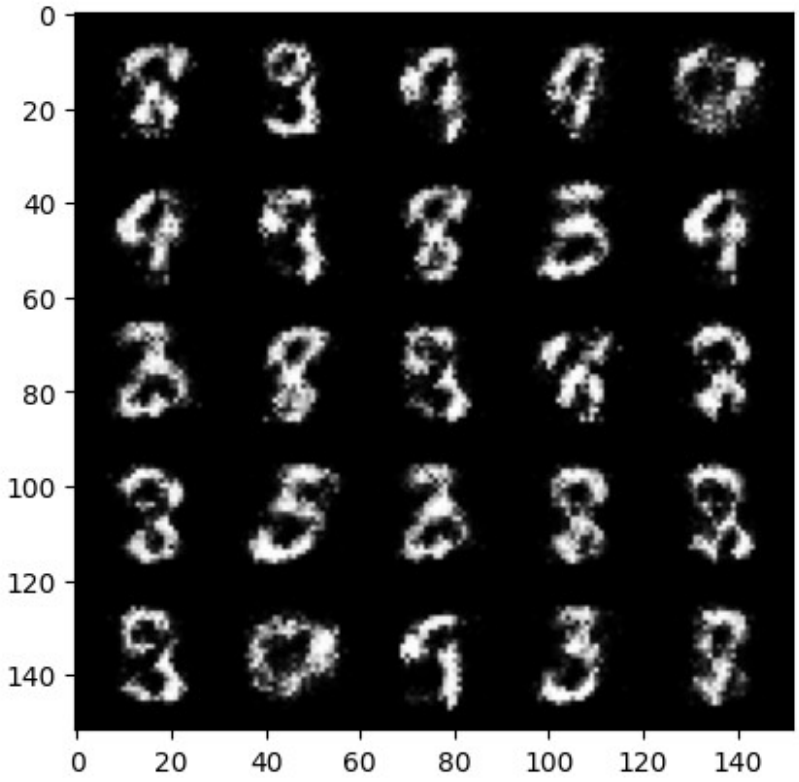
```
Epoch 30, step 14500: Generator loss: 4.221042400836944, discriminator loss: 0.08687657794728874
```

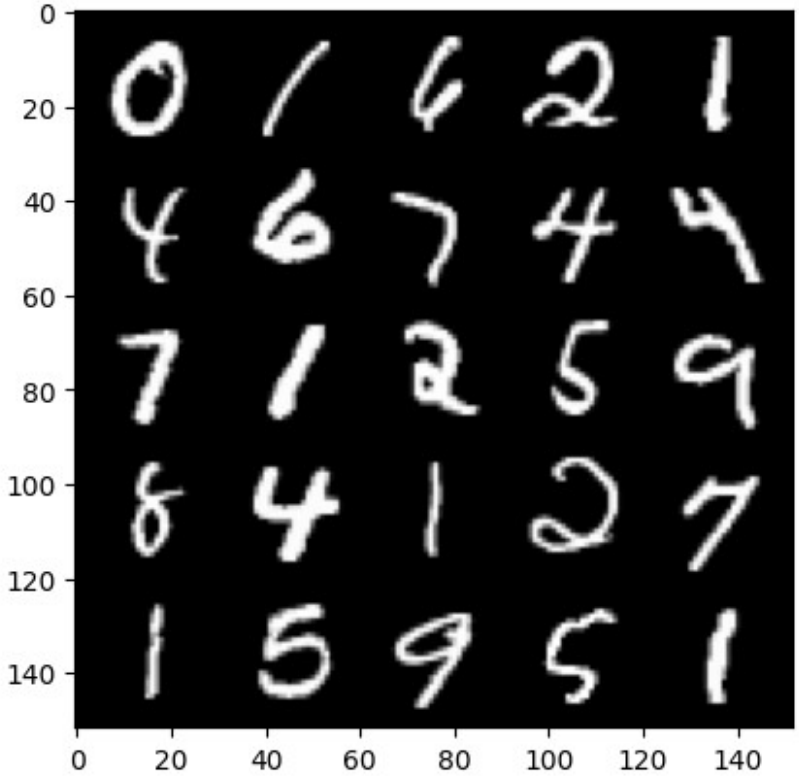




{"model_id":"7ab39c490f4c46e690947d0a706dcd94","version_major":2,"version_minor":0}

```
Epoch 31, step 15000: Generator loss: 4.1769024400711094,
discriminator loss: 0.08275702934712167
```

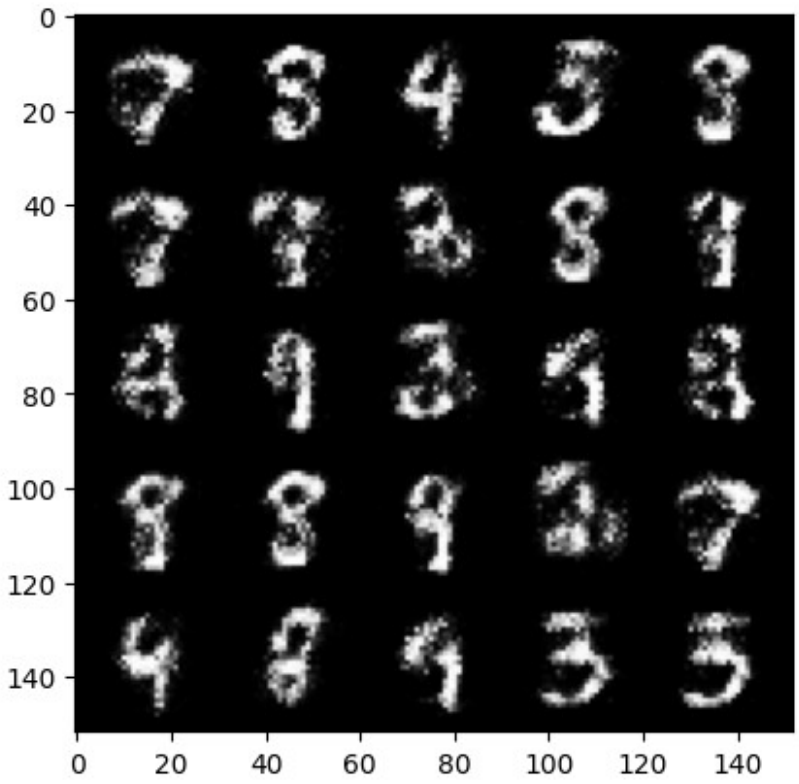

0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

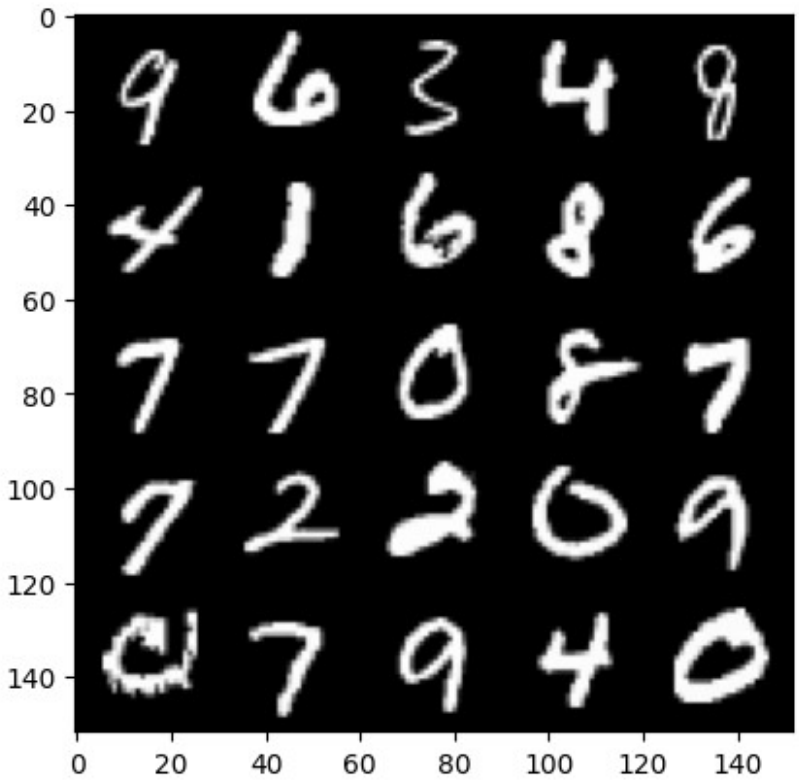

0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

{"model_id":"afad067603f94a9cbdb30c8c6e5ff3e0","version_major":2,"version_minor":0}

{"model_id":"c32d24e46b714f629ffd06097e3e2dd5","version_major":2,"version_minor":0}

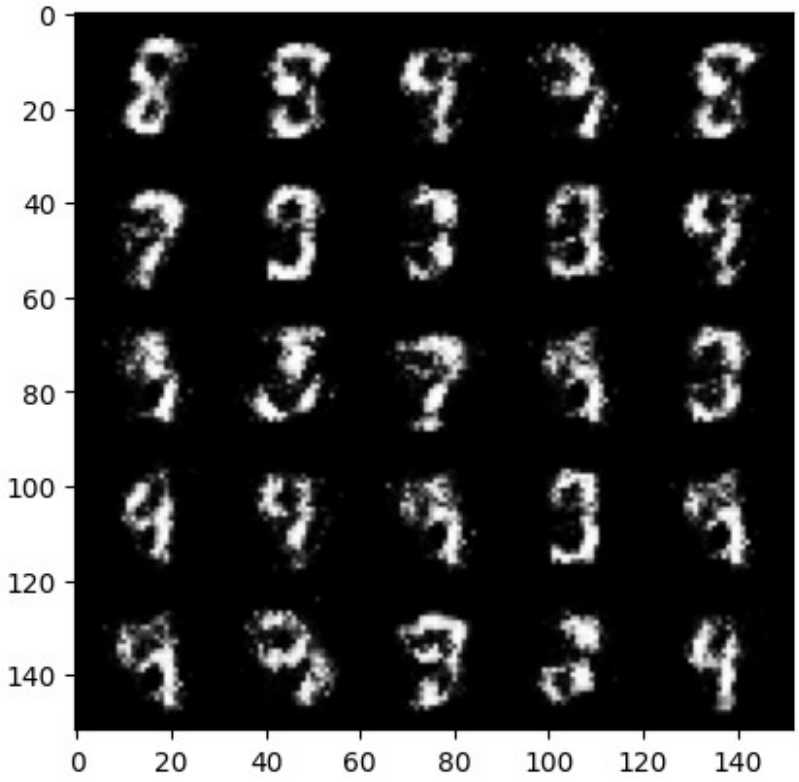
```
Epoch 33, step 15500: Generator loss: 3.914483067989351, discriminator loss: 0.09955570389330387
```

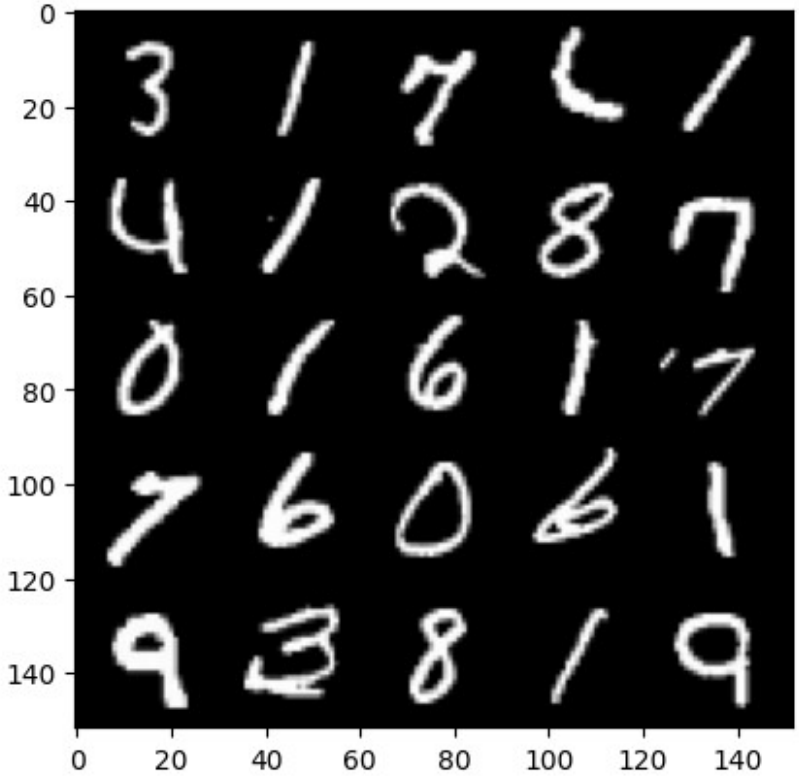




{"model_id":"705eba427e454e7799e47f9d5d32311a","version_major":2,"version_minor":0}

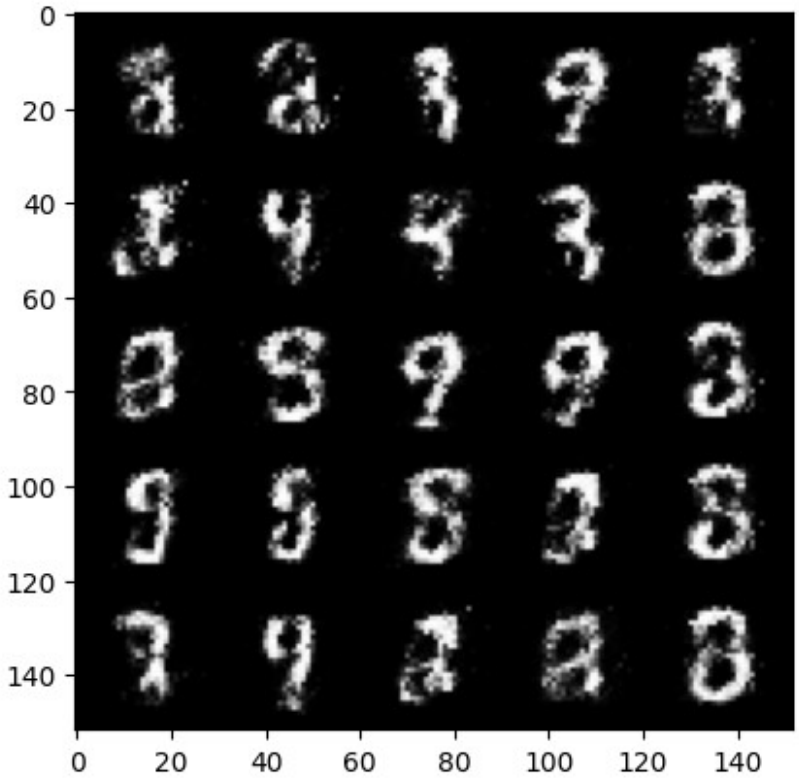
```
Epoch 34, step 16000: Generator loss: 3.859961364269258, discriminator loss: 0.11282271665334709
```

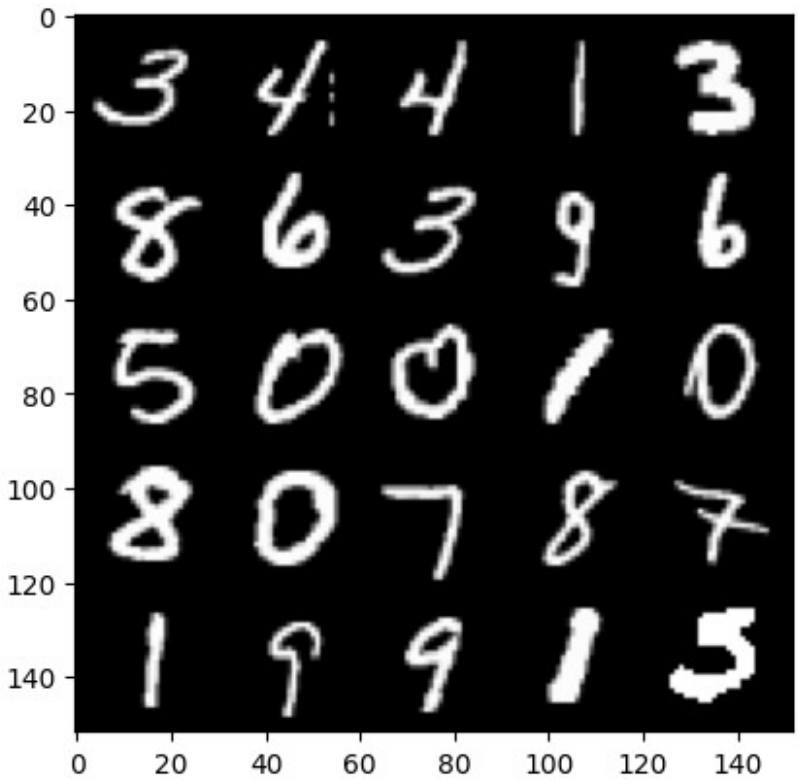

0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140


0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

{"model_id":"465981674f6a4ec6b90f0198ed01c684","version_major":2,"version_minor":0}

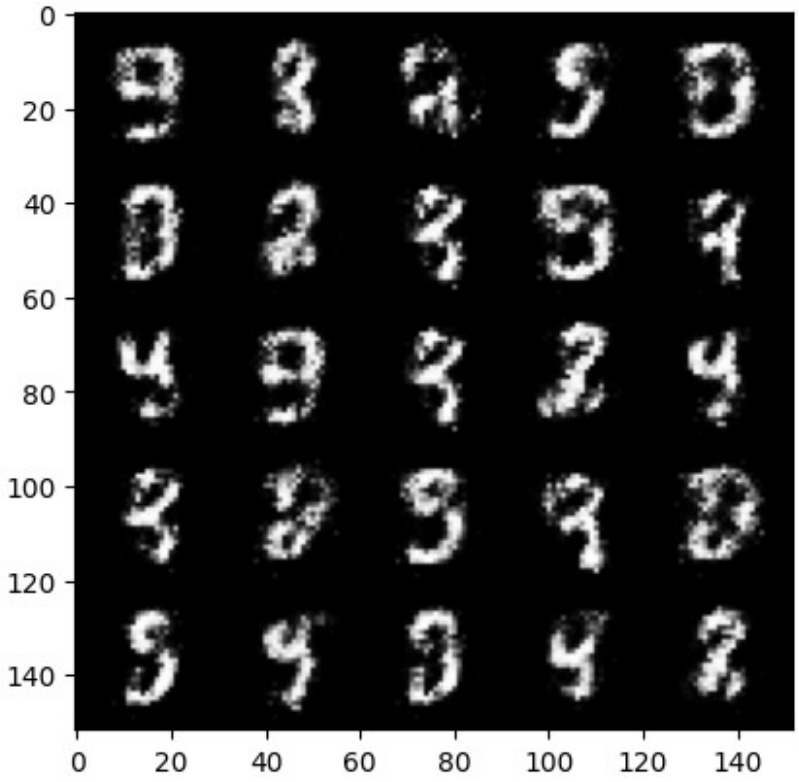
```
Epoch 35, step 16500: Generator loss: 3.7203861169815062,
discriminator loss: 0.1163589611873032
```

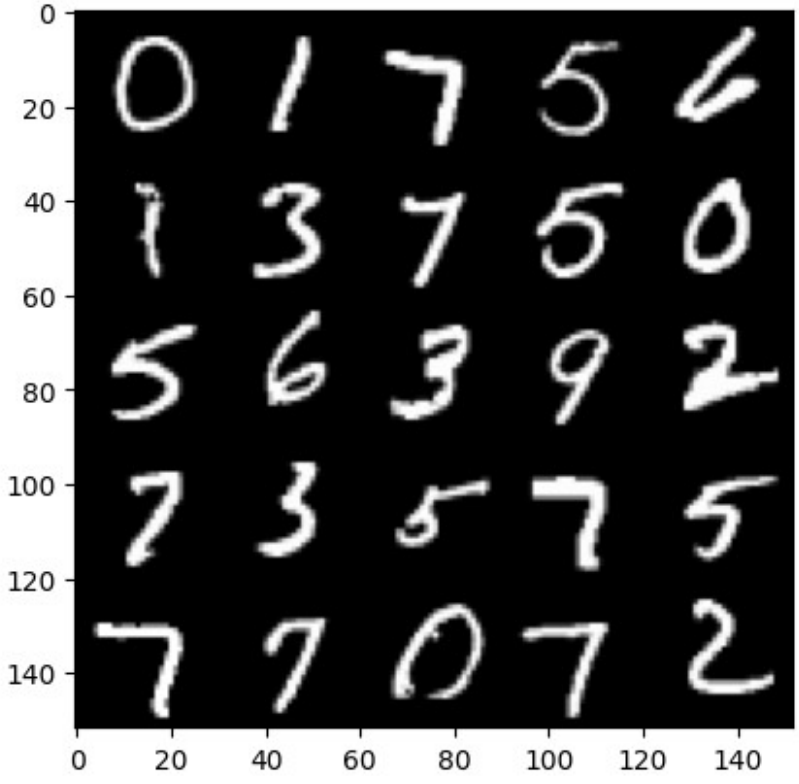




{"model_id":"81d5c1041a3840eb907a1f0f3df9eb0a","version_major":2,"version_minor":0}

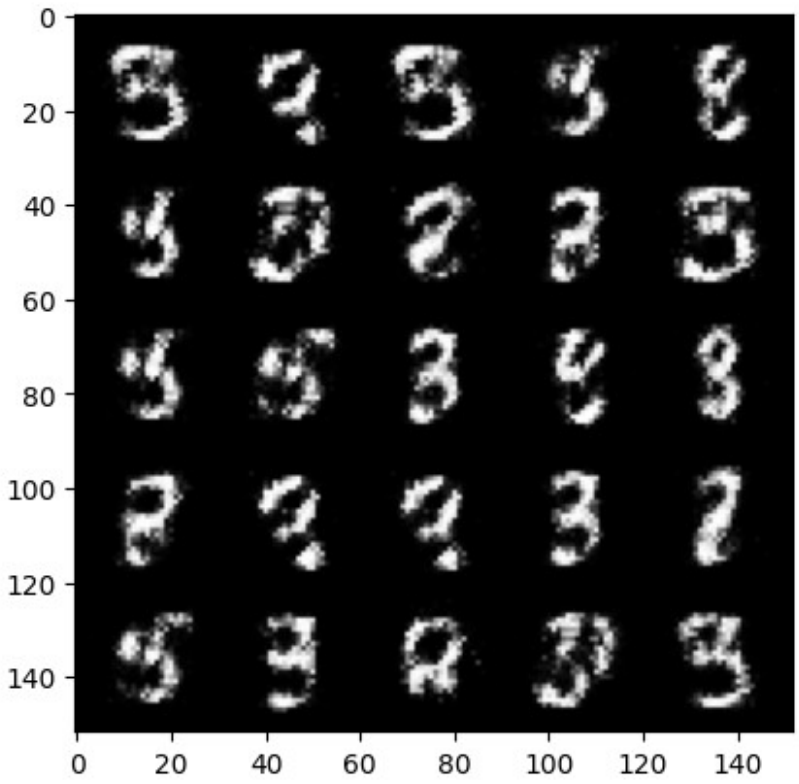
```
Epoch 36, step 17000: Generator loss: 3.766864835739132, discriminator loss: 0.11550457379966975
```

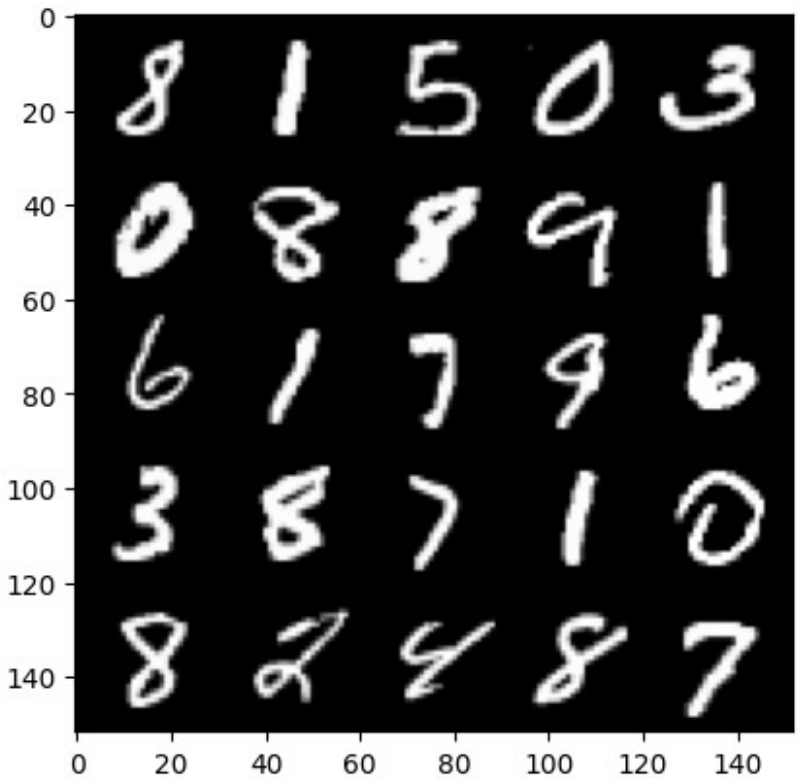

0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140


0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

{"model_id":"f4bb399a41d543eba76200c8592011d8","version_major":2,"version_minor":0}

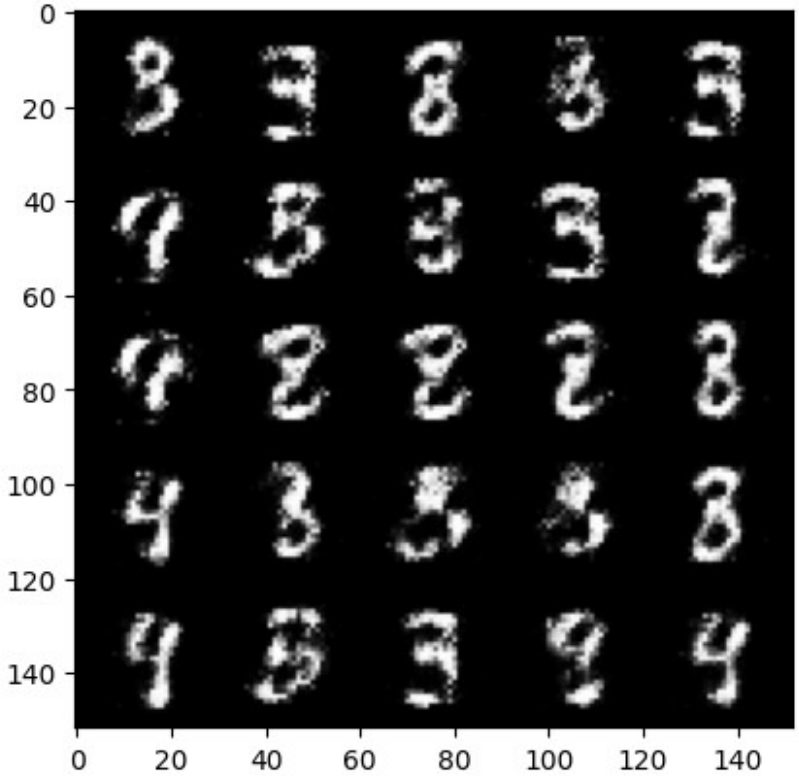
```
Epoch 37, step 17500: Generator loss: 3.4412817115783665,
discriminator loss: 0.1494532505124807
```

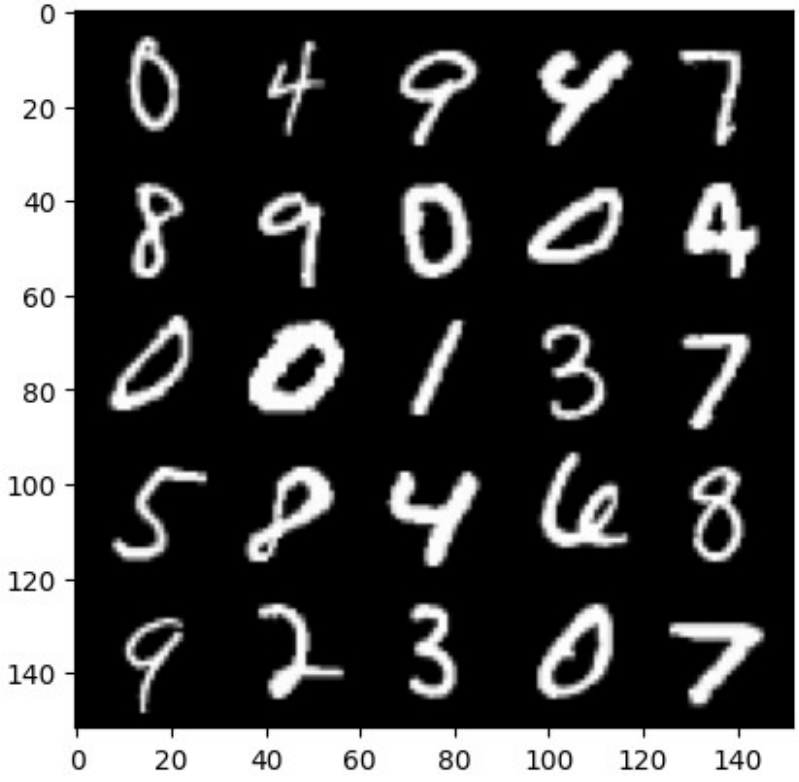




{"model_id":"b3ca7a2f88274cfeb7f50b6a624953d6","version_major":2,"version_minor":0}

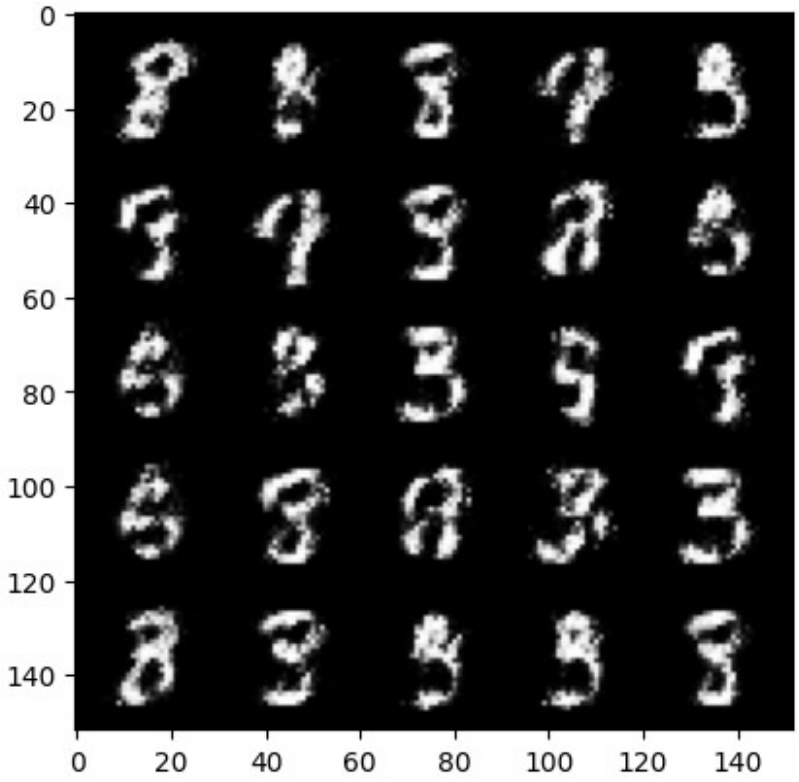
```
Epoch 38, step 18000: Generator loss: 3.3694844994544977,
discriminator loss: 0.14208598984777923
```

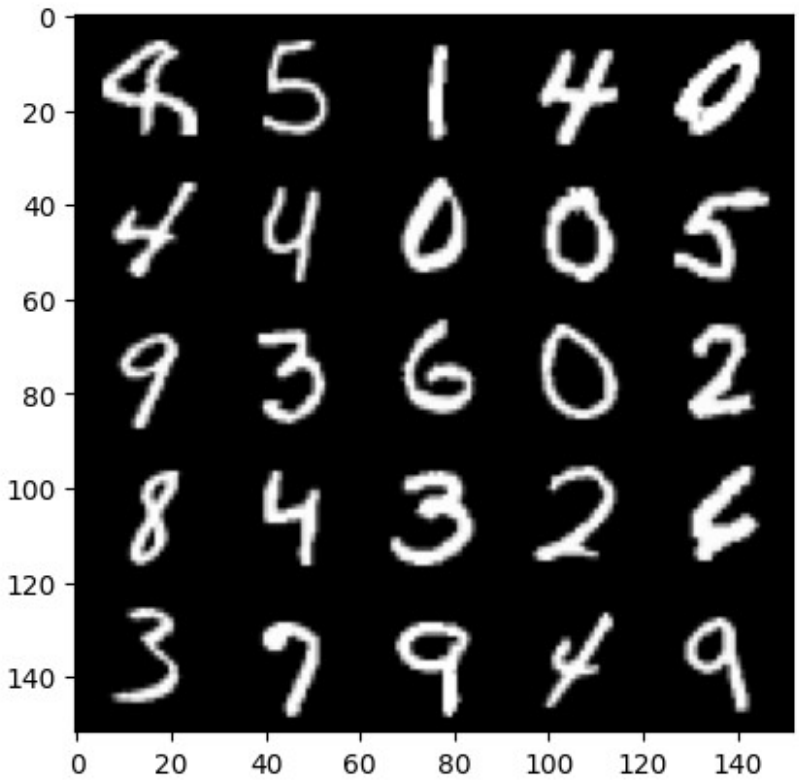

0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140


0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

{"model_id":"971948b2cc7041d0a7504b5d2634e487","version_major":2,"version_minor":0}

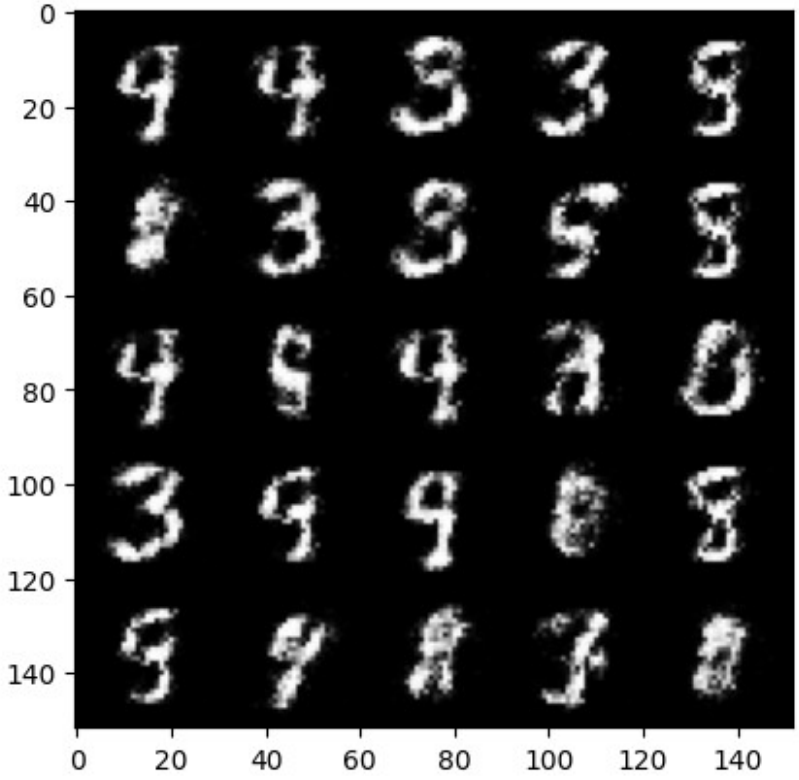
```
Epoch 39, step 18500: Generator loss: 3.4047308573722885,
discriminator loss: 0.13821293988078856
```

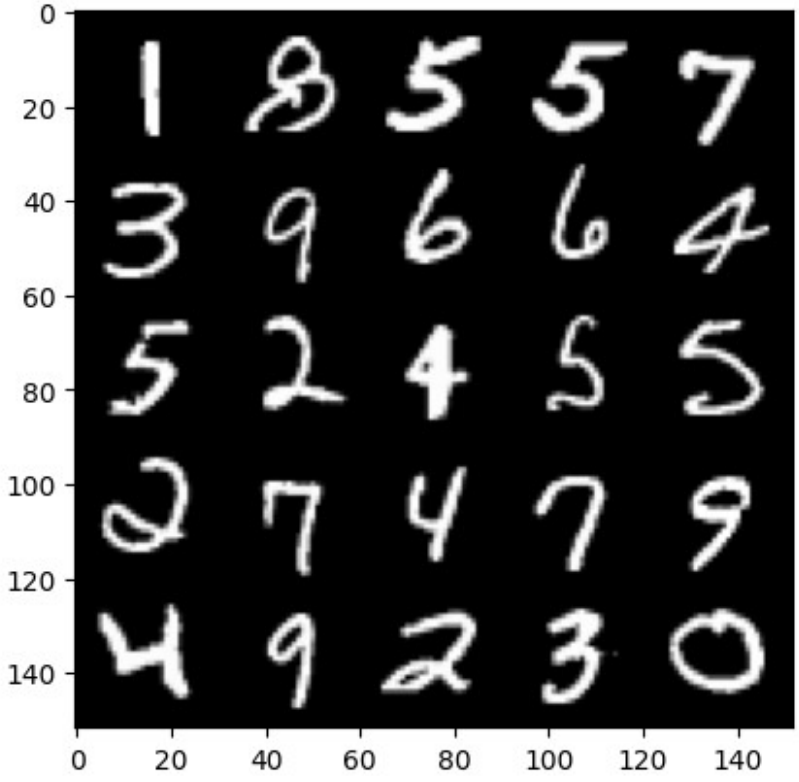




{"model_id":"330ab8dc920e4e5aad7cc5c431c33cbe","version_major":2,"version_minor":0}

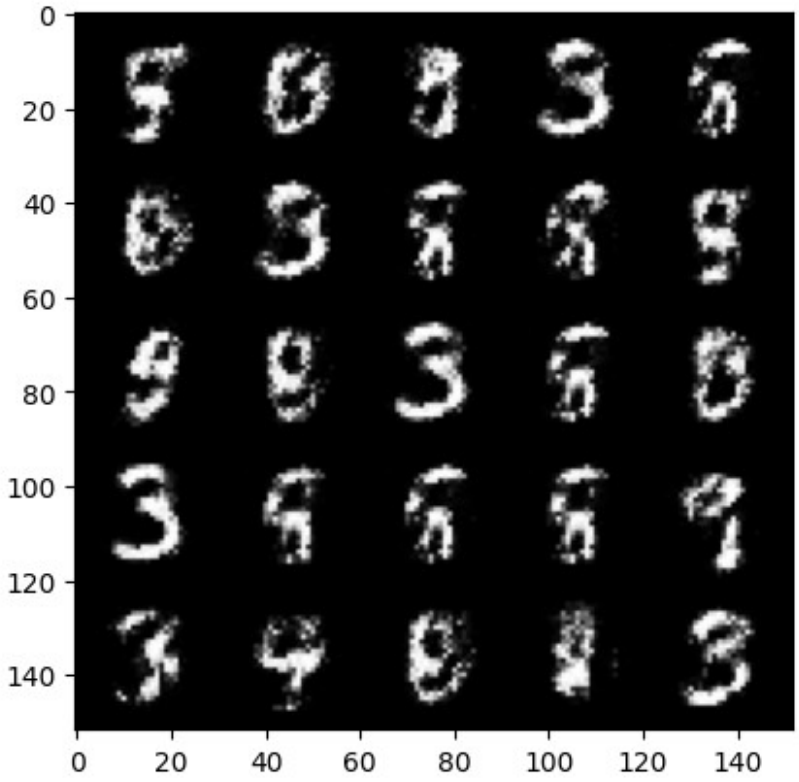
```
Epoch 40, step 19000: Generator loss: 3.2325598292350795,
discriminator loss: 0.1622313260734081
```

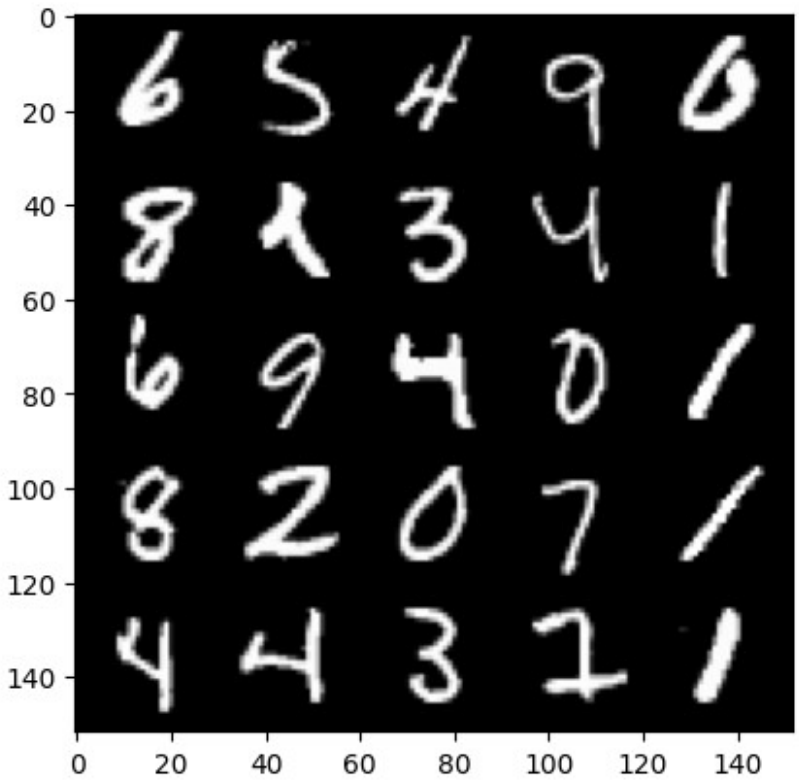

0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140


0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

{"model_id":"9ca4462b39fe4d2e80d09147e2625b71","version_major":2,"version_minor":0}

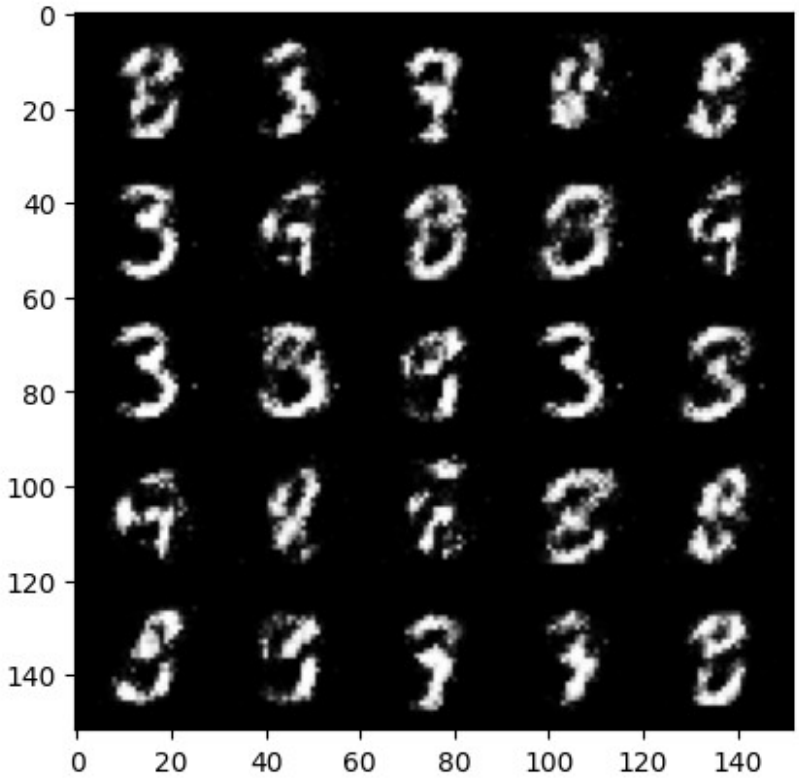
```
Epoch 41, step 19500: Generator loss: 3.1353144655227663,
discriminator loss: 0.14937264829874042
```

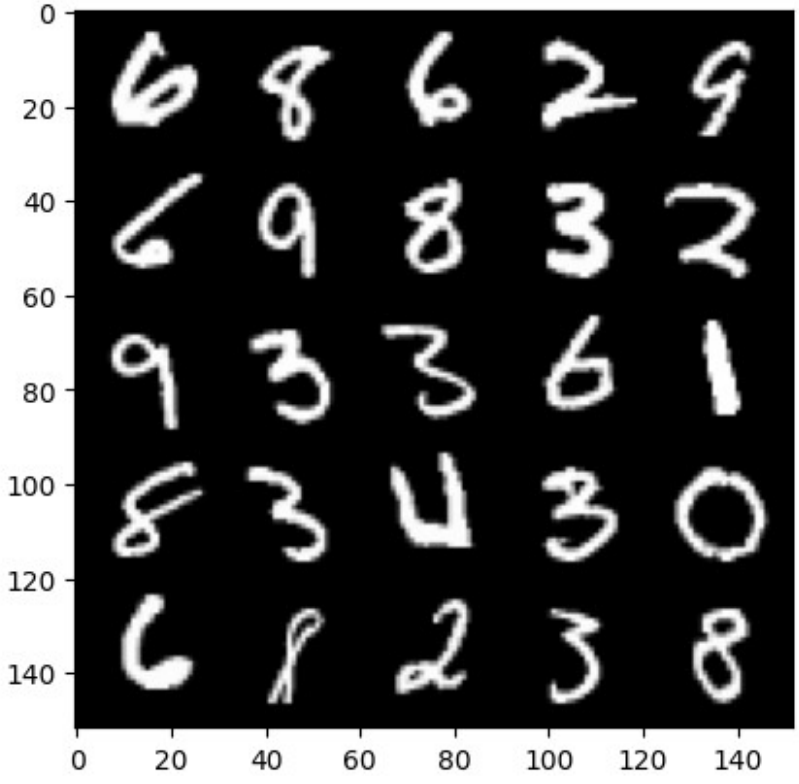




{"model_id":"fd2bb75e5c56490e85ae0d0105b109b1","version_major":2,"version_minor":0}

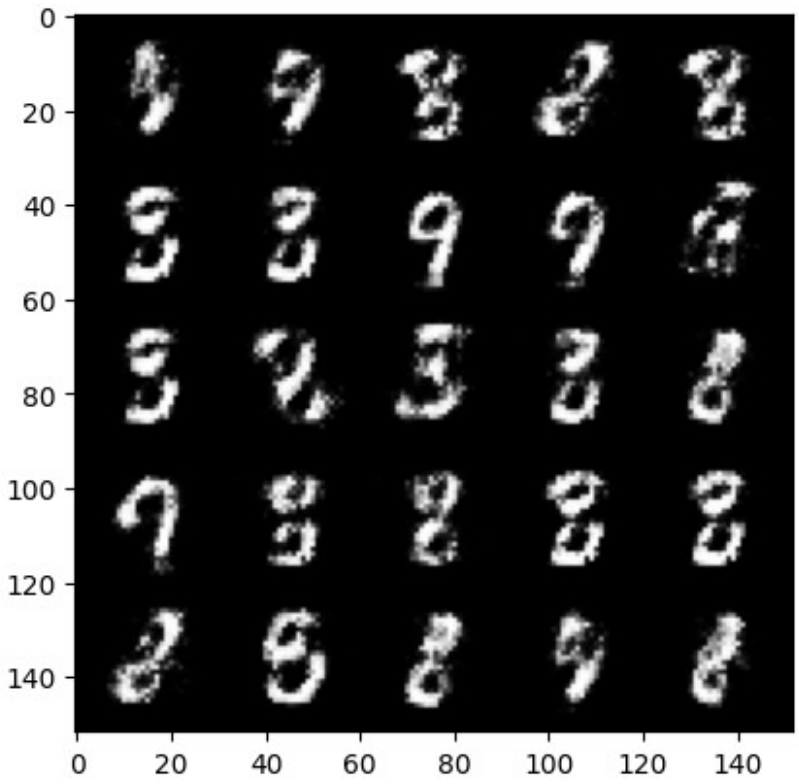
```
Epoch 42, step 20000: Generator loss: 3.375720155715939, discriminator loss: 0.13819575936347236
```

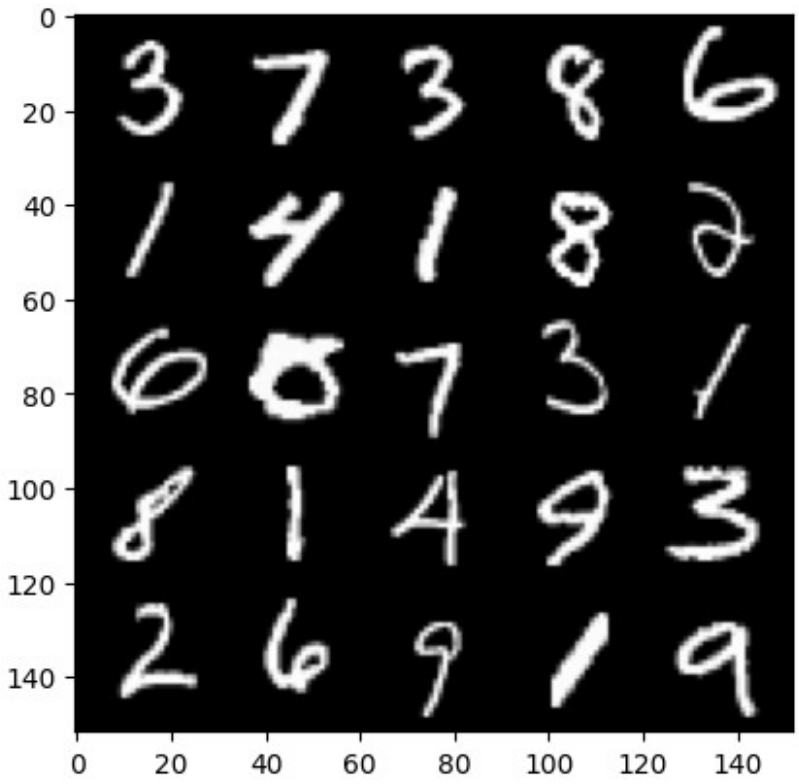

0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140


0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

{"model_id":"d3b262313b8641cc9d8ab1ac62cfca04","version_major":2,"version_minor":0}

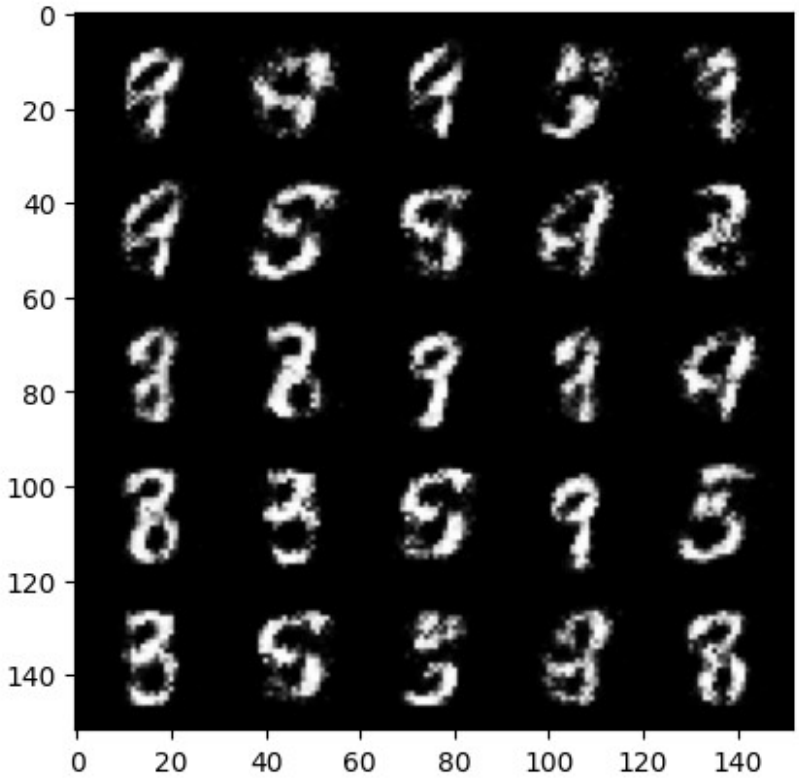
```
Epoch 43, step 20500: Generator loss: 3.349744007110595, discriminator loss: 0.13976071672141546
```

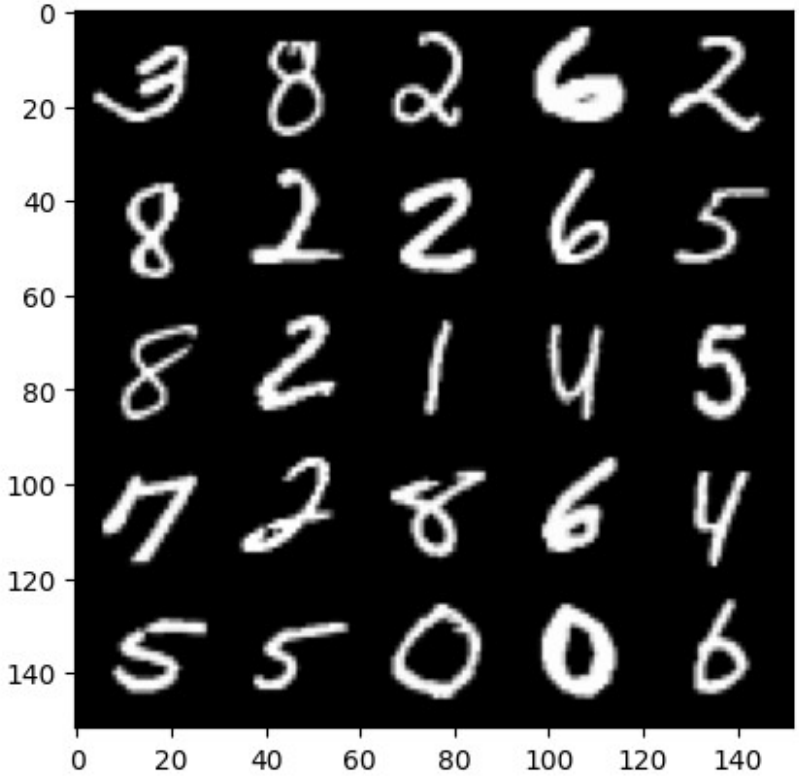




{"model_id":"232e751ee70d4738a434a13f640d4a47","version_major":2,"version_minor":0}

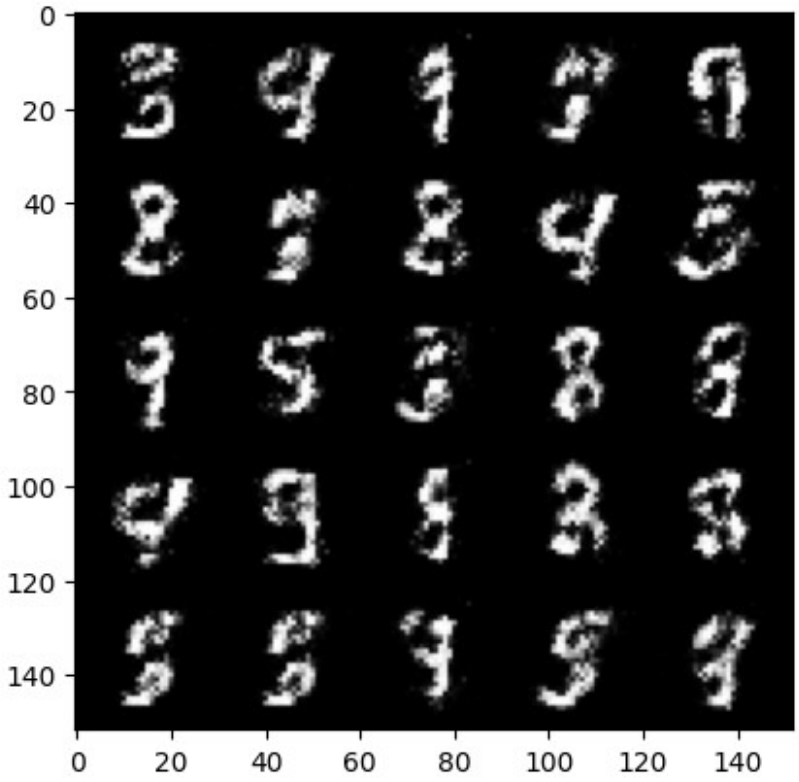
```
Epoch 44, step 21000: Generator loss: 3.2593429808616636,
discriminator loss: 0.14106425206363196
```

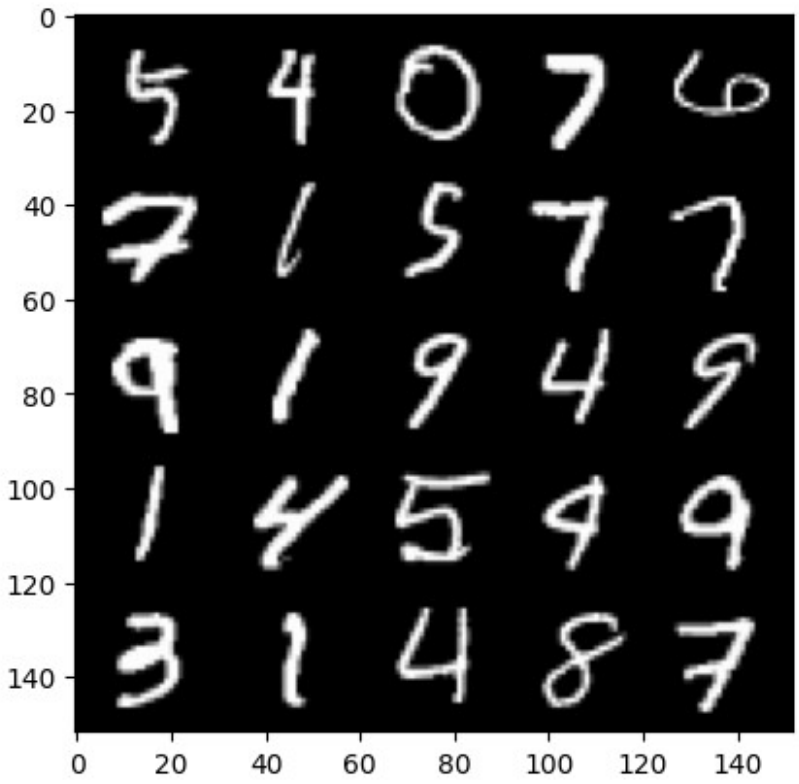

0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140


0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

{"model_id":"82ef75d3c1e04fd4b6a76ecc03dd8938","version_major":2,"version_minor":0}

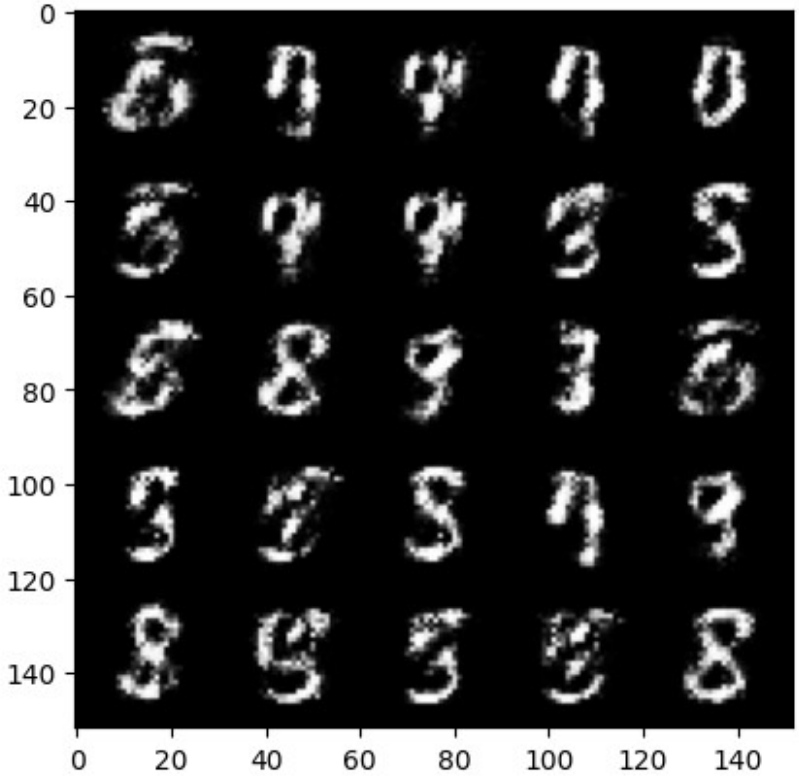
```
Epoch 45, step 21500: Generator loss: 3.1602979645729055,
discriminator loss: 0.15399476133286943
```

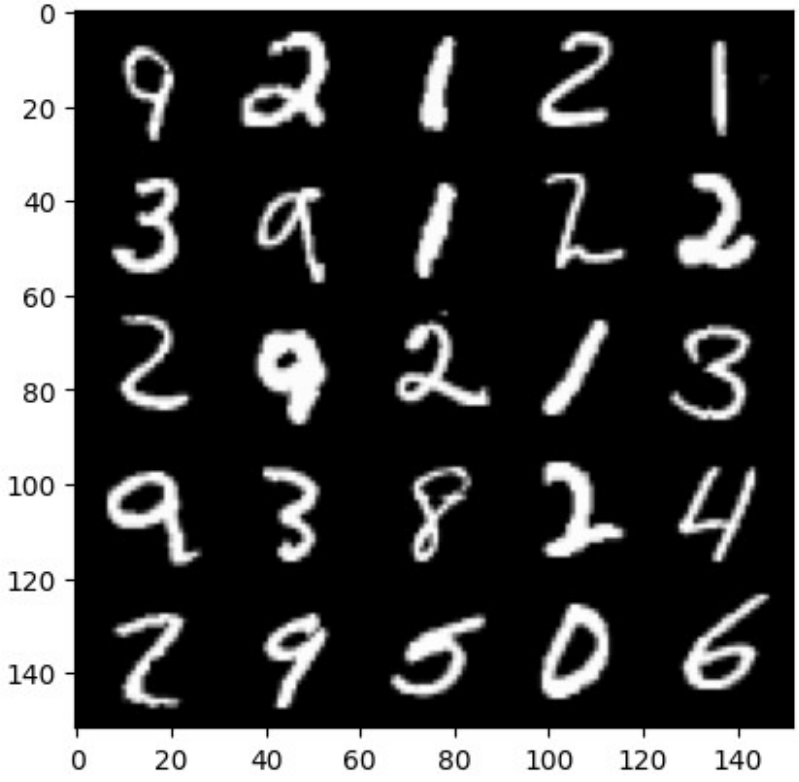




{"model_id":"e4f465c25f3843c5bd66439f8b827ebe","version_major":2,"version_minor":0}

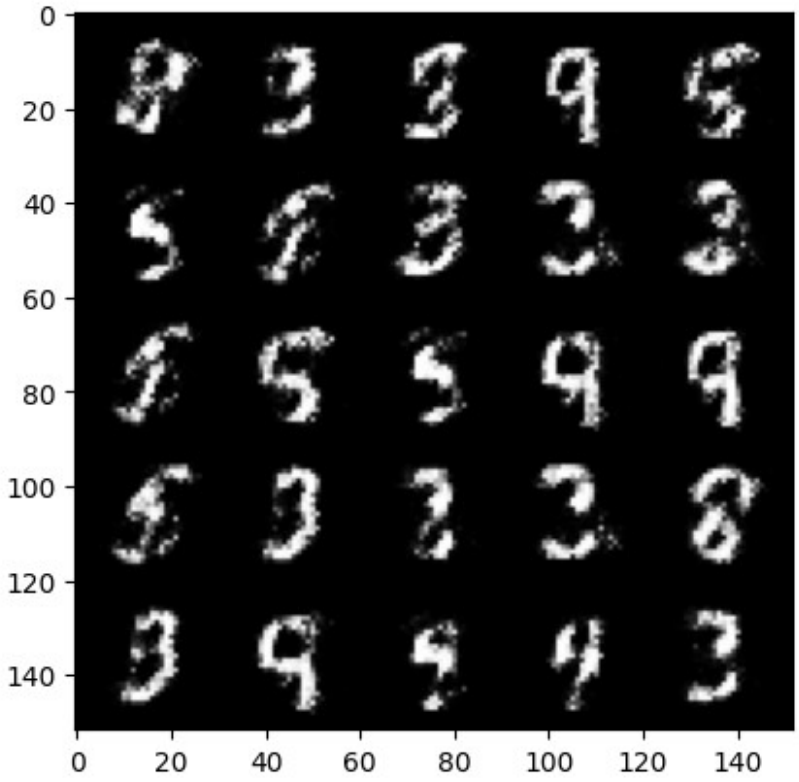
```
Epoch 46, step 22000: Generator loss: 3.1768172087669417,
discriminator loss: 0.15833034729957587
```

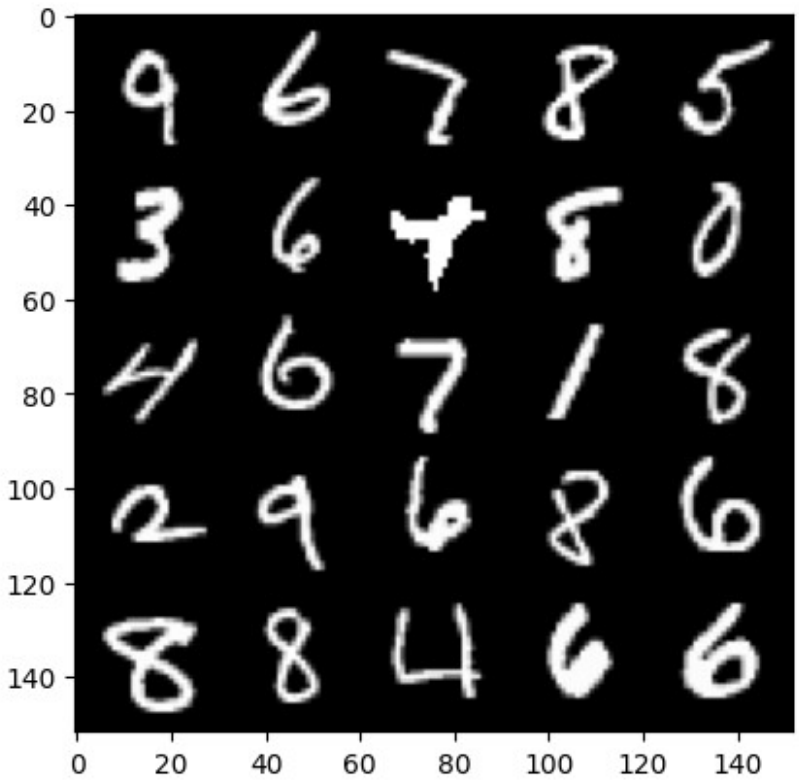

0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140


0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

{"model_id":"214455e33db34c8ea01d832152079866","version_major":2,"version_minor":0}

```
Epoch 47, step 22500: Generator loss: 3.0808194017410266,
discriminator loss: 0.1595121924728156
```

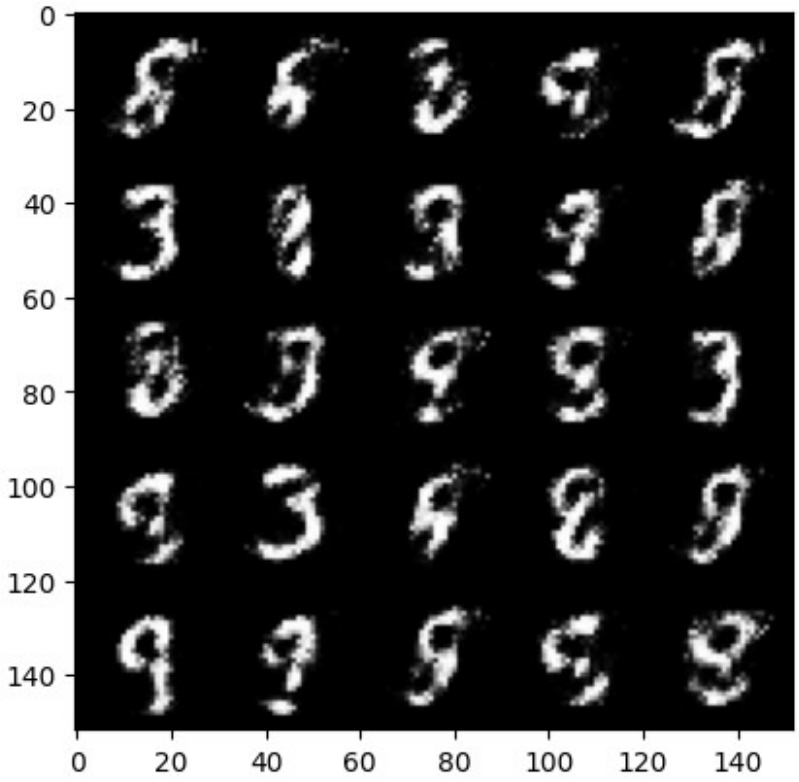


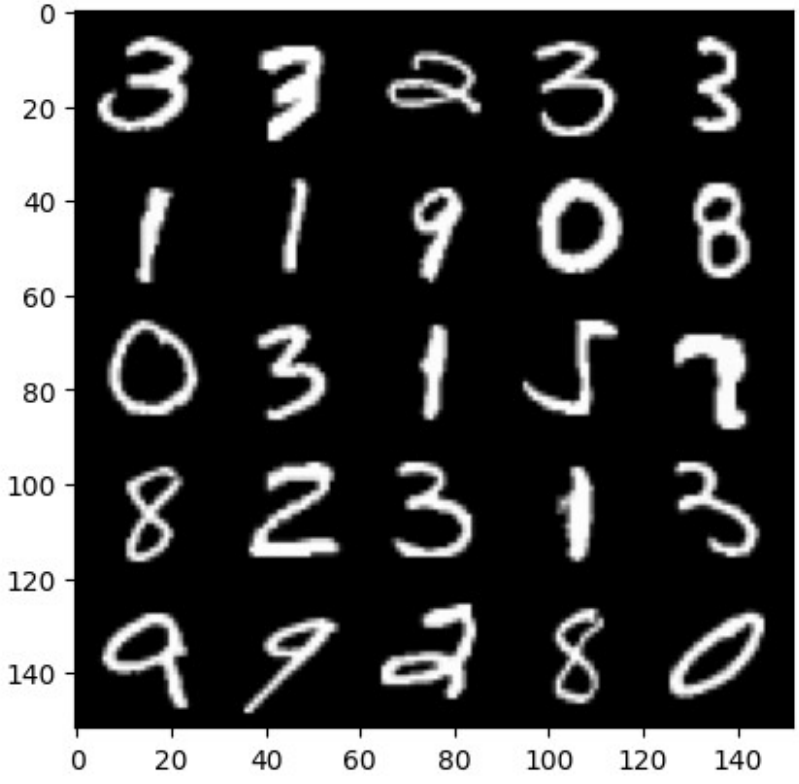


{"model_id":"0e652024b8b1466997d0f88a3abbc679","version_major":2,"version_minor":0}

{"model_id":"10a98b52127b457a957eb604ecd5a4df","version_major":2,"version_minor":0}

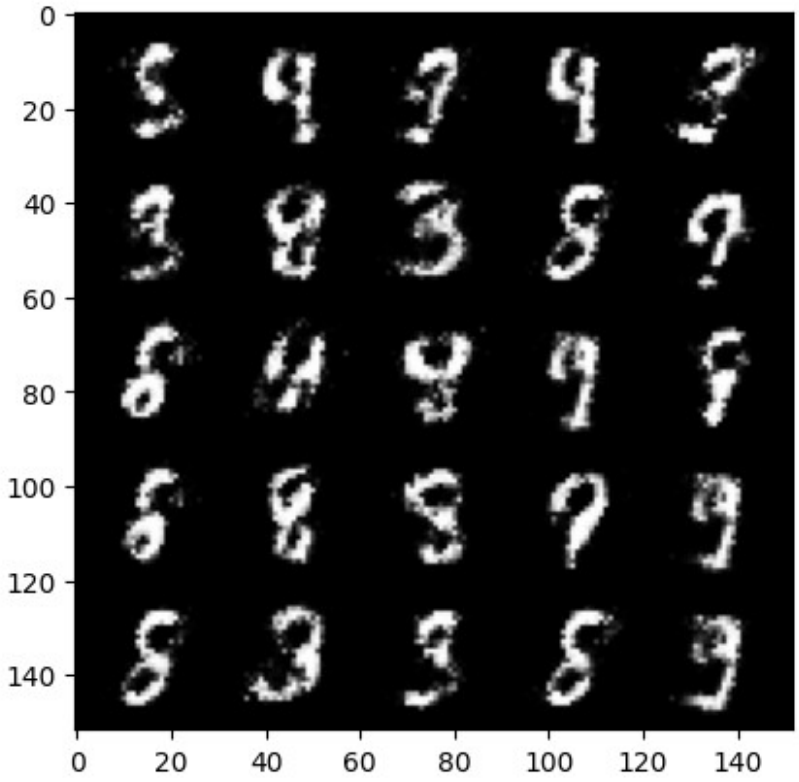
```
Epoch 49, step 23000: Generator loss: 3.0994081845283508,
discriminator loss: 0.1566800025850535
```

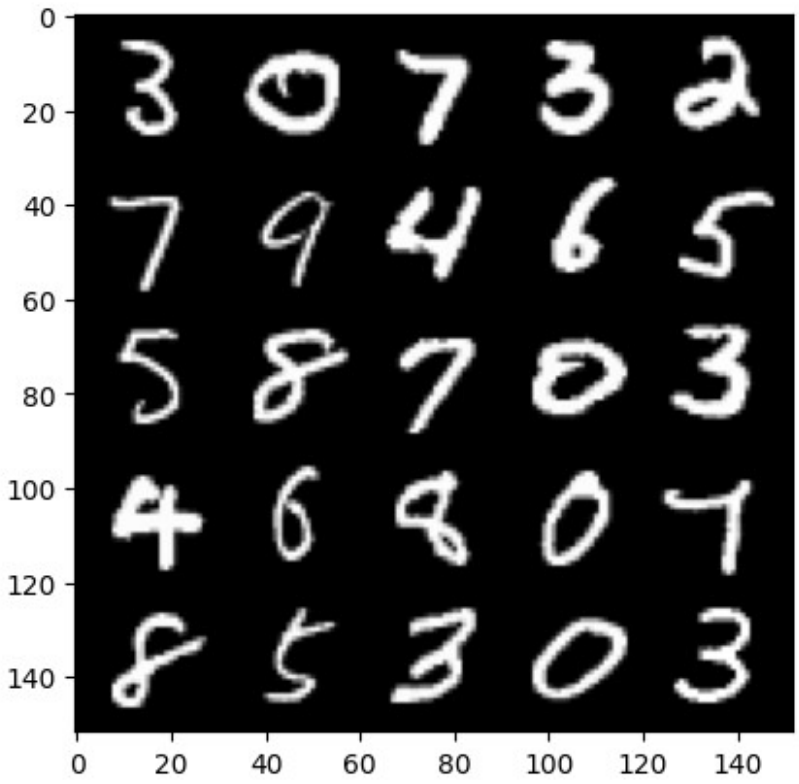

0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140


0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

{"model_id":"c729dcf2fb0a42ef8a8090d1911d4c75","version_major":2,"version_minor":0}

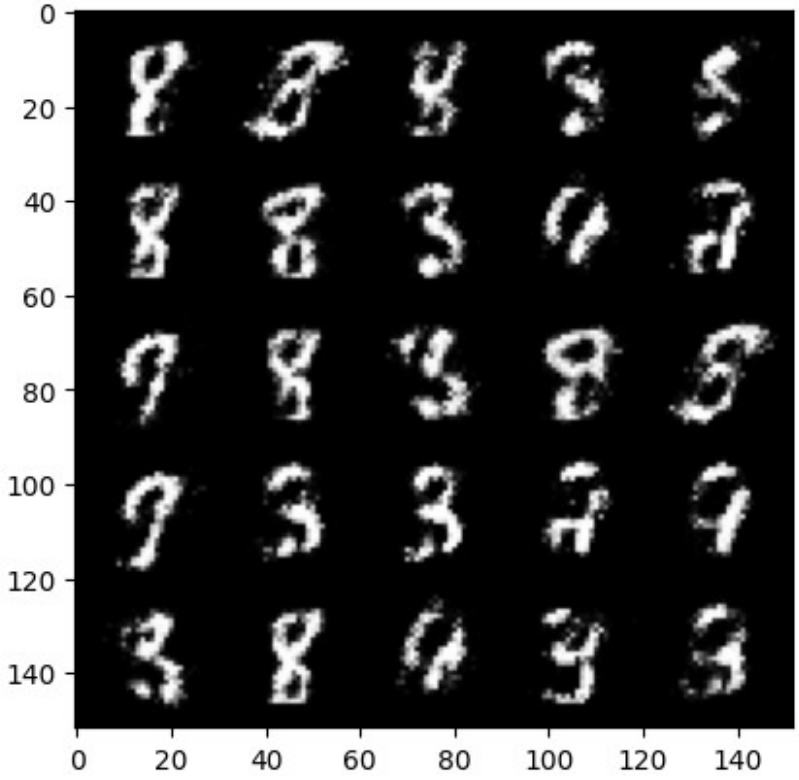
```
Epoch 50, step 23500: Generator loss: 3.2093095731735226,
discriminator loss: 0.14292749600112448
```

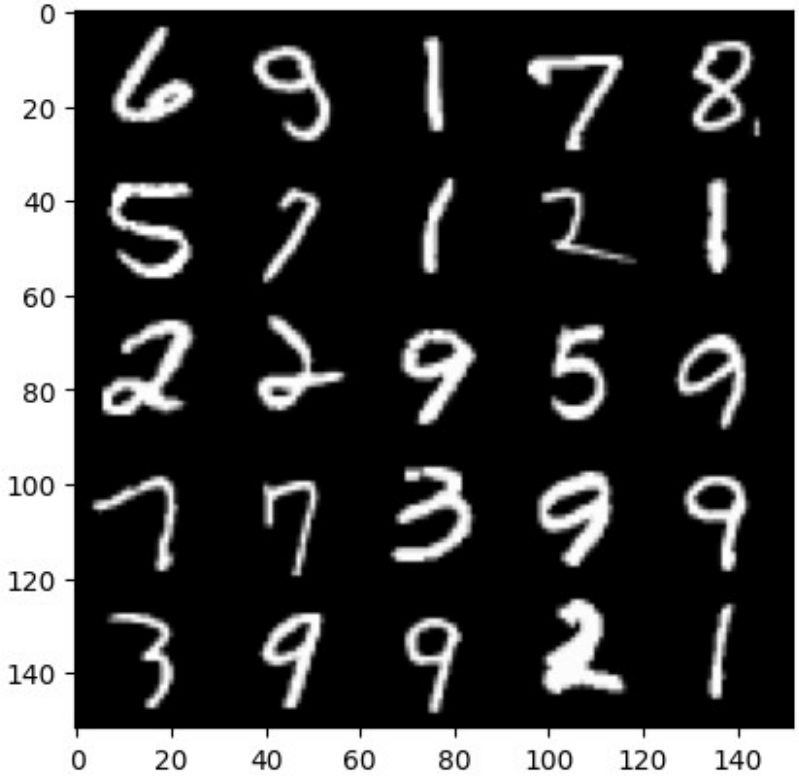




{"model_id":"a3e03a82aaf04e55a6e6d1f990999ca8","version_major":2,"version_minor":0}

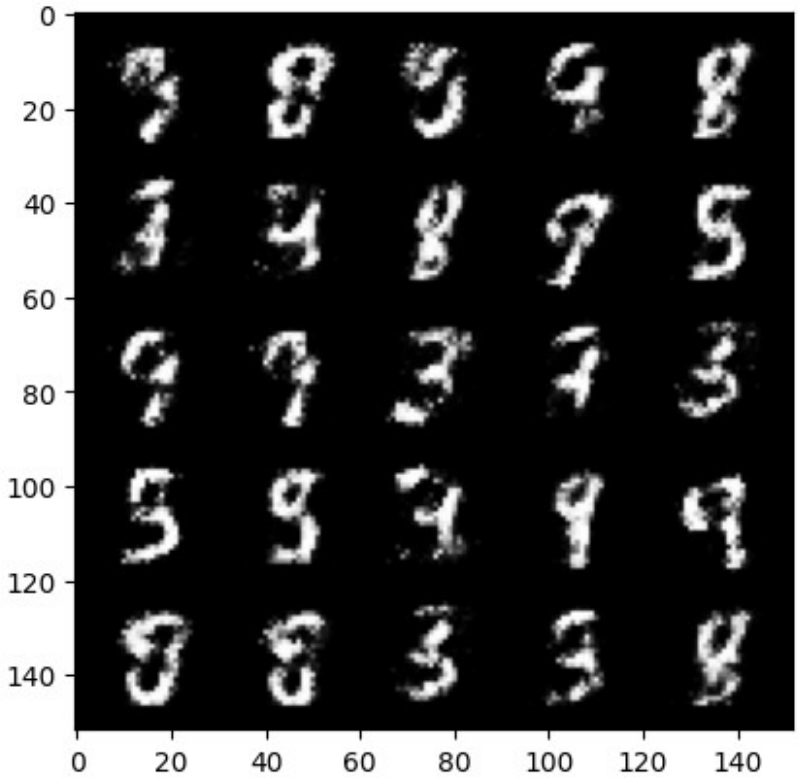
```
Epoch 51, step 24000: Generator loss: 3.359498355388643, discriminator loss: 0.14219286490976807
```

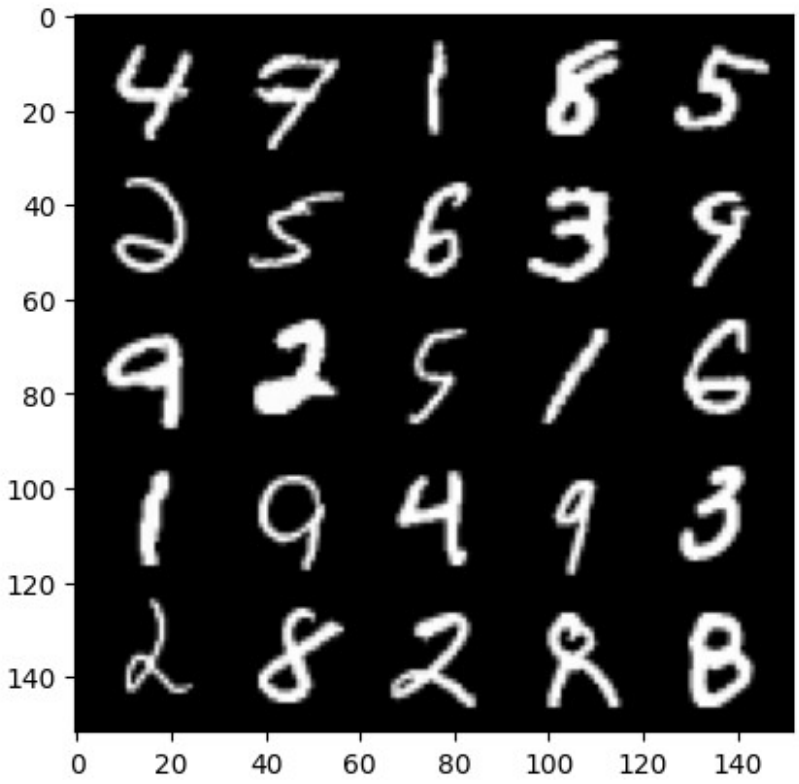

0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140


0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

{"model_id":"65827d8fd20f416e929a9770b79e3f36","version_major":2,"version_minor":0}

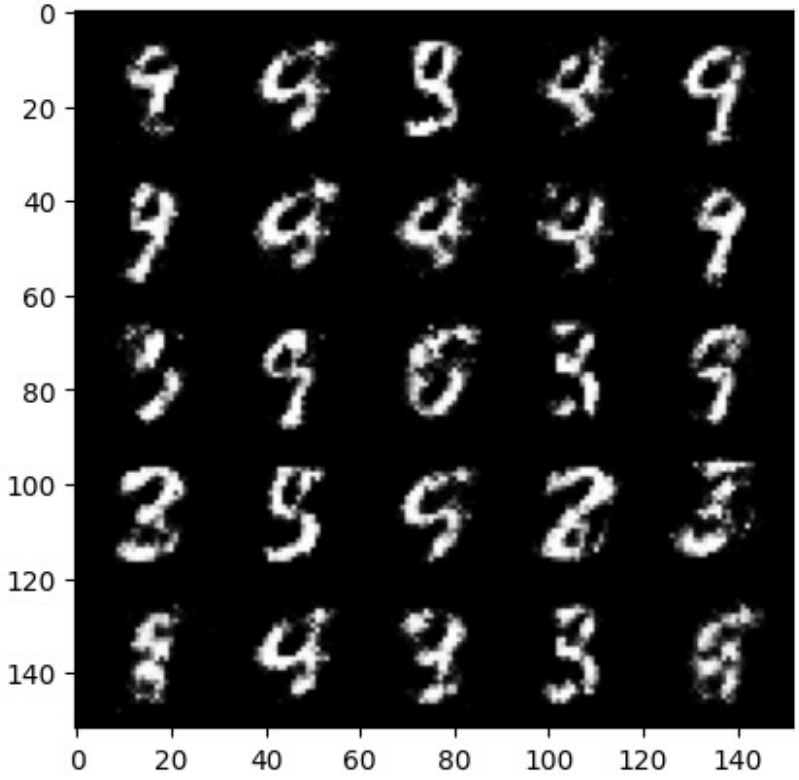
```
Epoch 52, step 24500: Generator loss: 3.0203952655792246,
discriminator loss: 0.17642457640171041
```

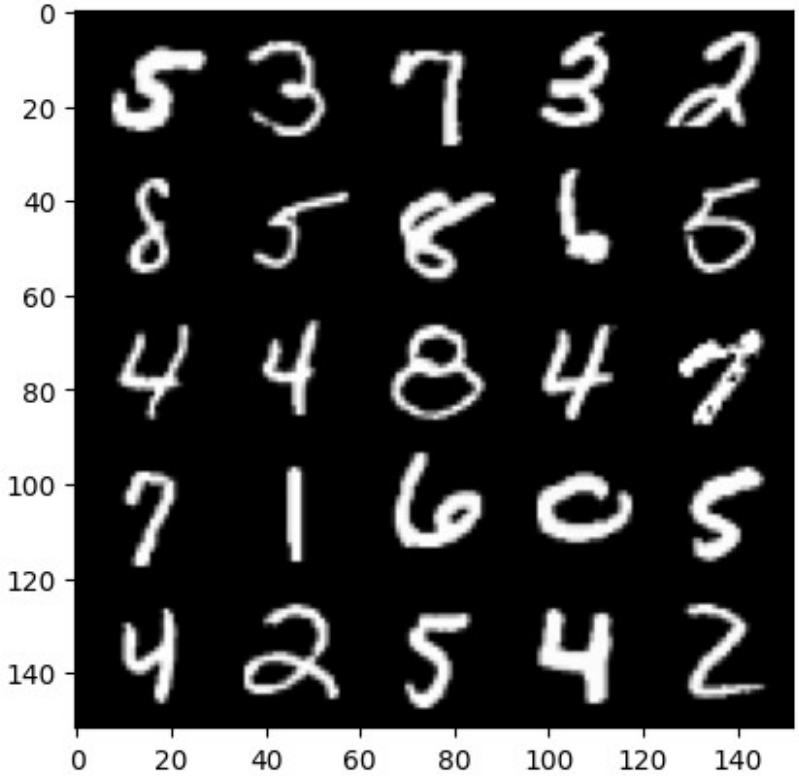




{"model_id":"8aad0ac02b9d4da68d39f58dcb1af44e","version_major":2,"version_minor":0}

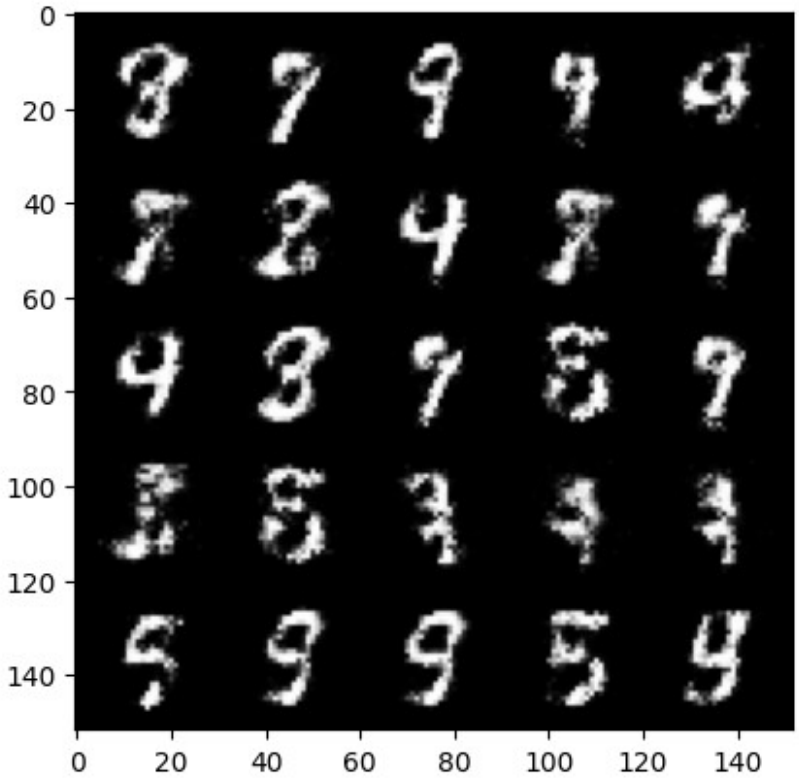
```
Epoch 53, step 25000: Generator loss: 2.9543503336906434,
discriminator loss: 0.17945732223987576
```

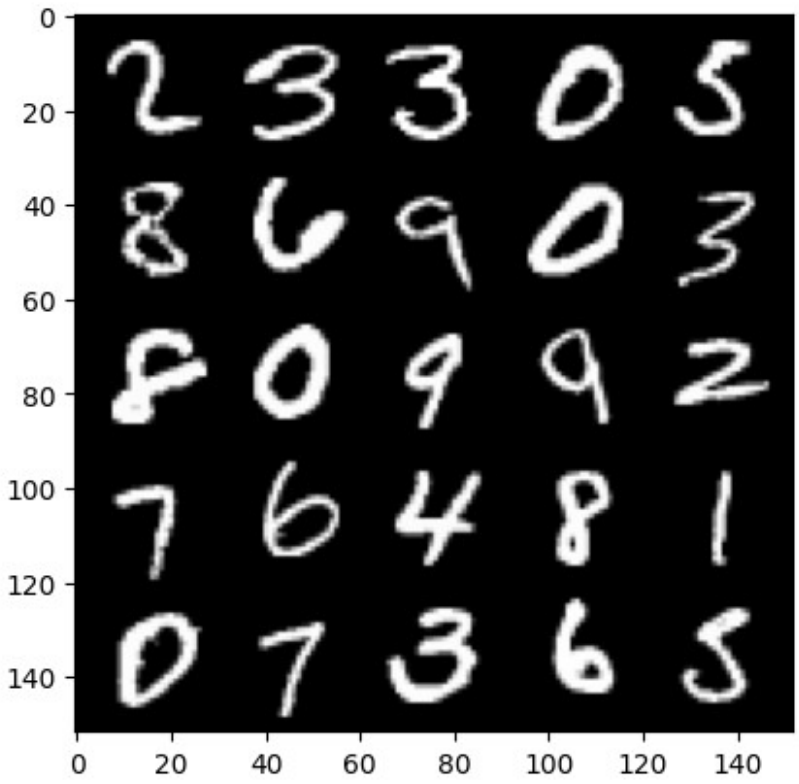

0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140


0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

{"model_id":"3ba74dcb44494057aafa1d67e871d524","version_major":2,"version_minor":0}

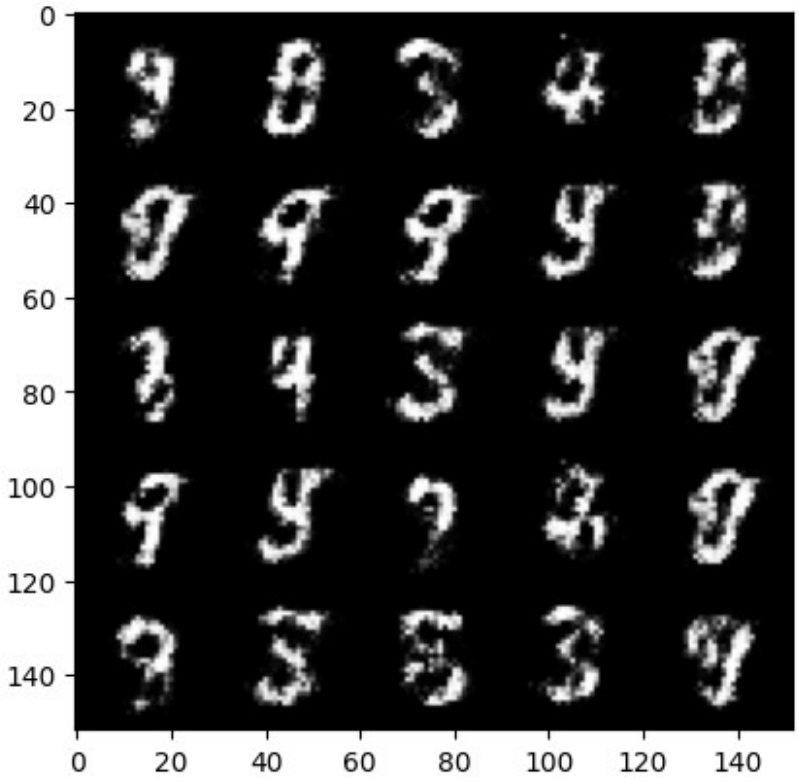
```
Epoch 54, step 25500: Generator loss: 2.8760844783782993,
discriminator loss: 0.18041508468985576
```

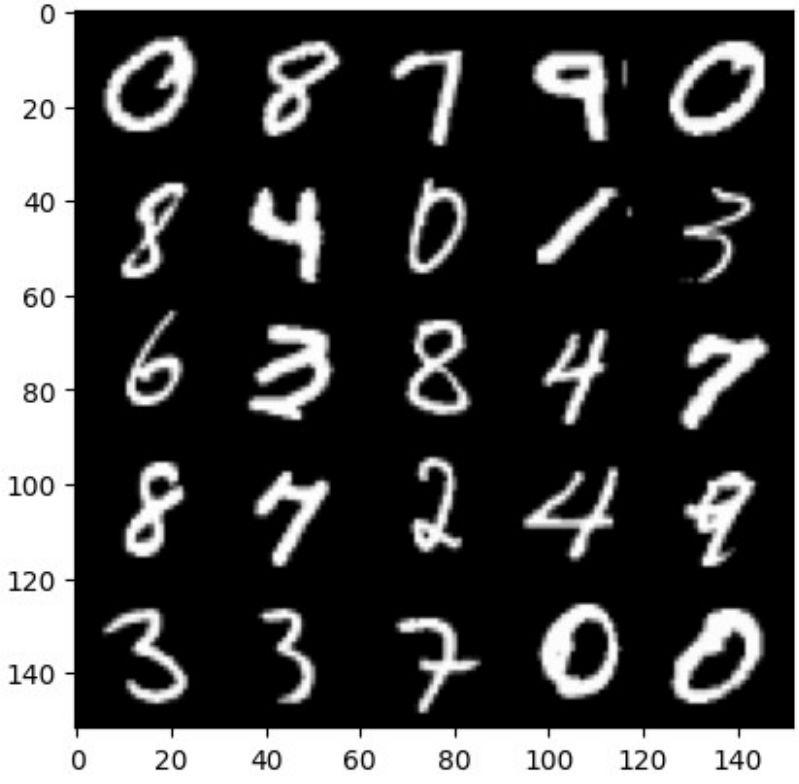




{"model_id":"49f3ac122ec24a149903594b6d9f862f","version_major":2,"version_minor":0}

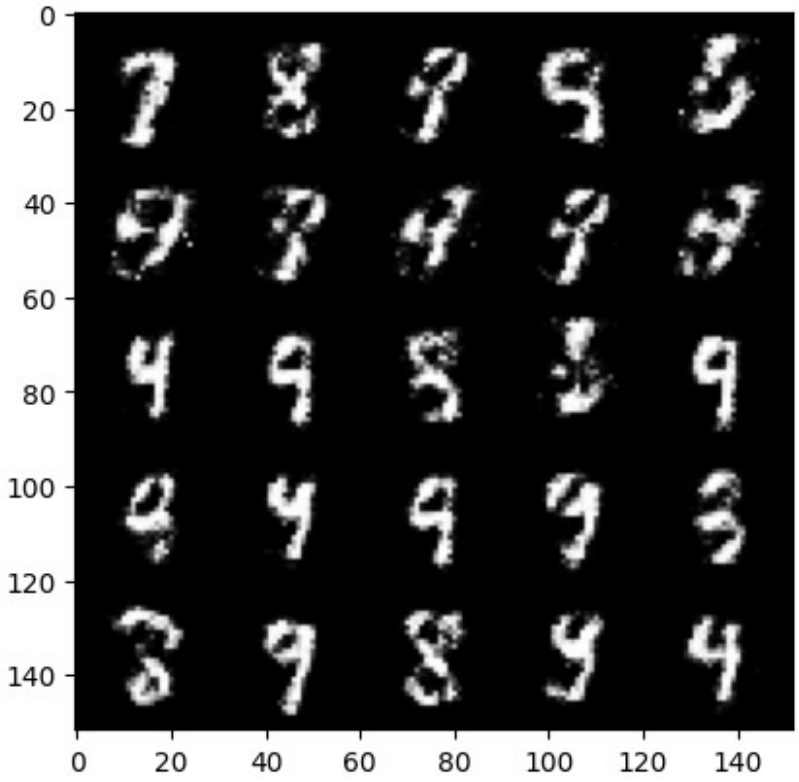
```
Epoch 55, step 26000: Generator loss: 2.748655465602874, discriminator loss: 0.20576251102983936
```

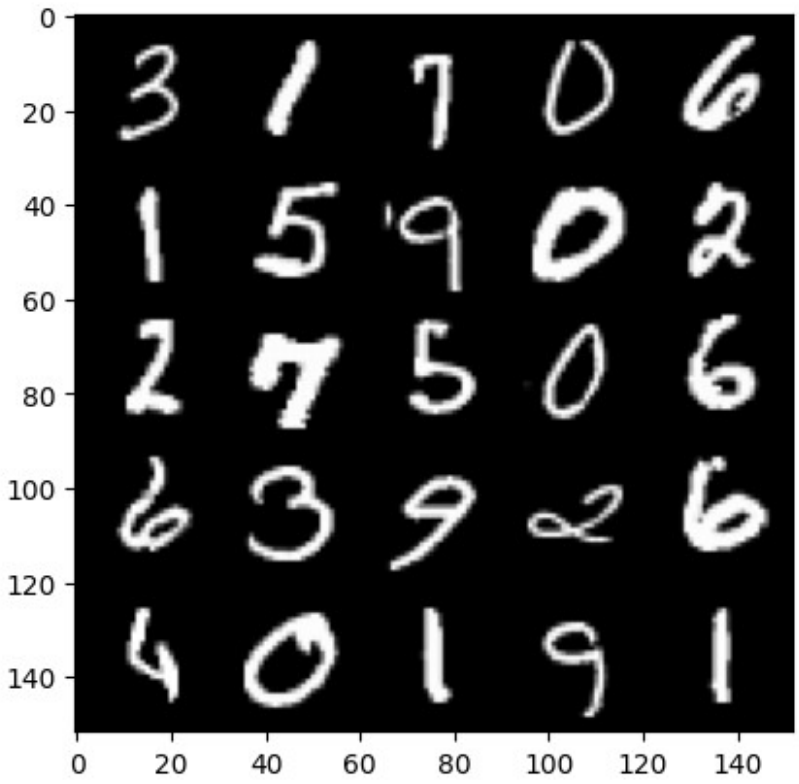

0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140


0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

{"model_id":"f8db595540f64cc5b9a1ad05e4ceee55","version_major":2,"version_minor":0}

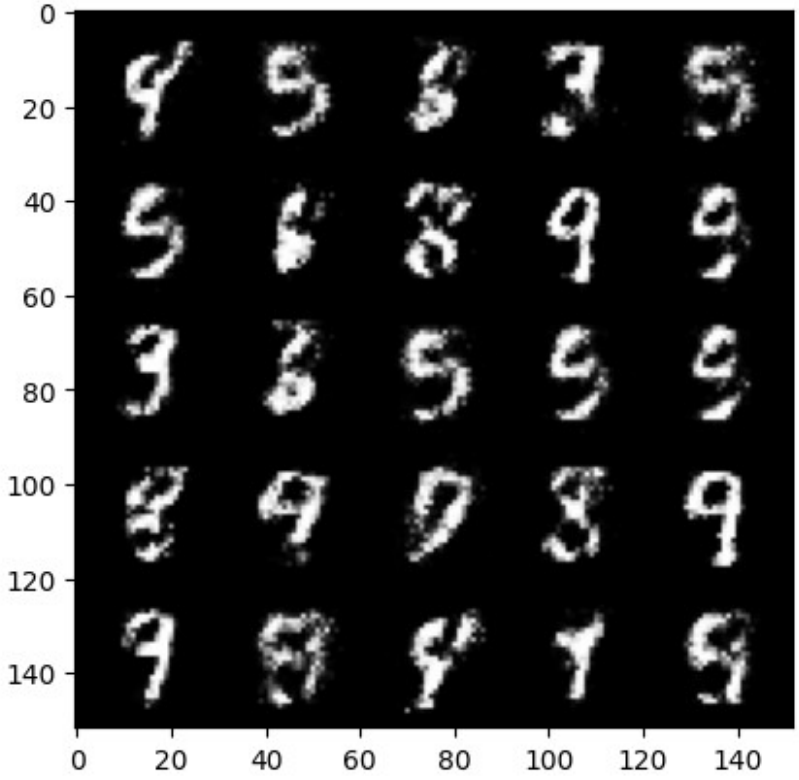
```
Epoch 56, step 26500: Generator loss: 2.7928773589134206,
discriminator loss: 0.19257682706415655
```

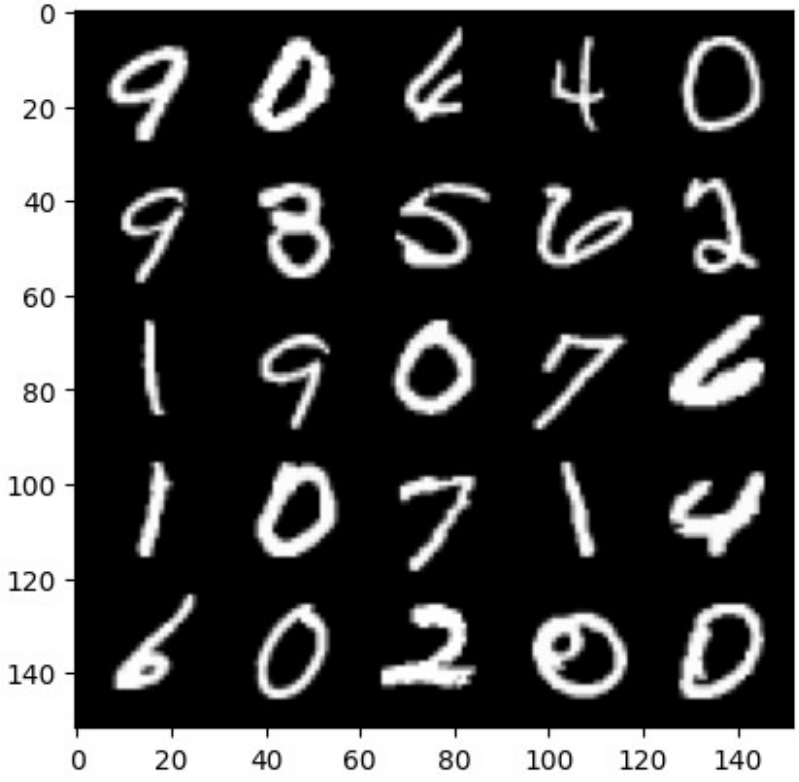




{"model_id":"93b31bfe5e6448808fe282b8aaa50b72","version_major":2,"version_minor":0}

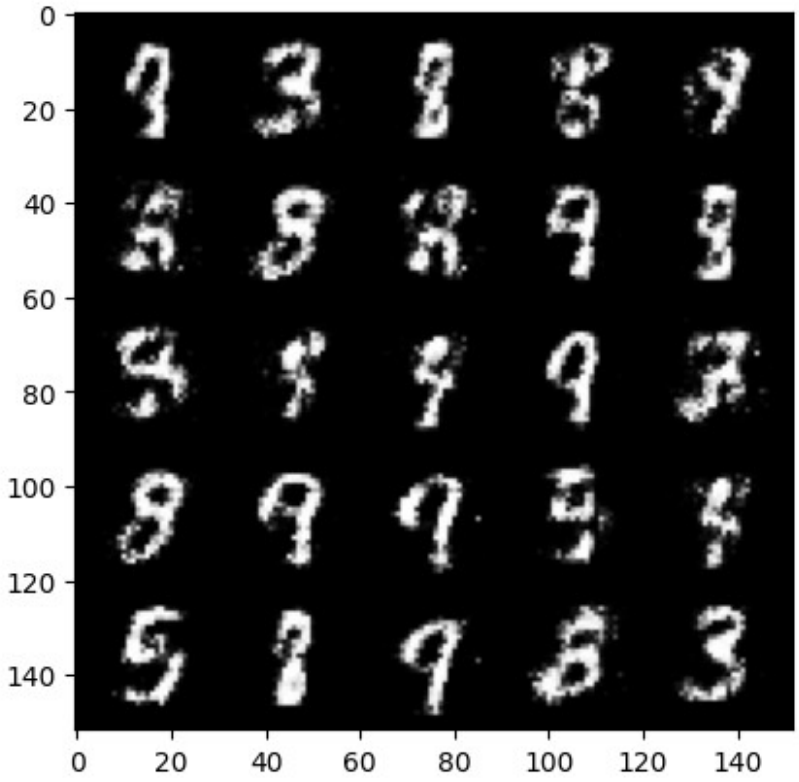
```
Epoch 57, step 27000: Generator loss: 2.837762551784514, discriminator loss: 0.18406006267666825
```

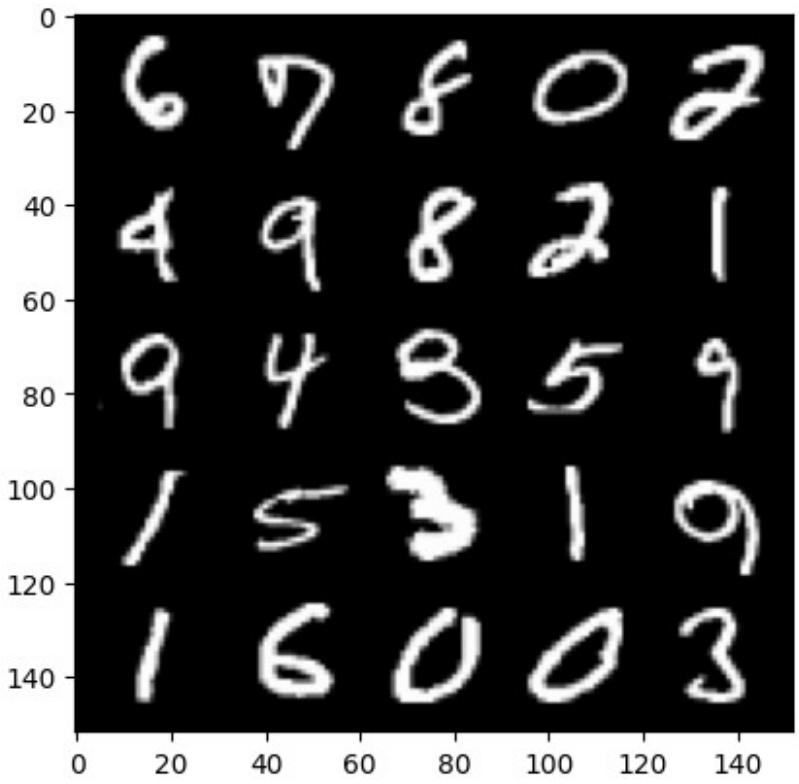

0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140


0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

{"model_id":"cbdcfb1d9f94408ba850be39ba9154de","version_major":2,"version_minor":0}

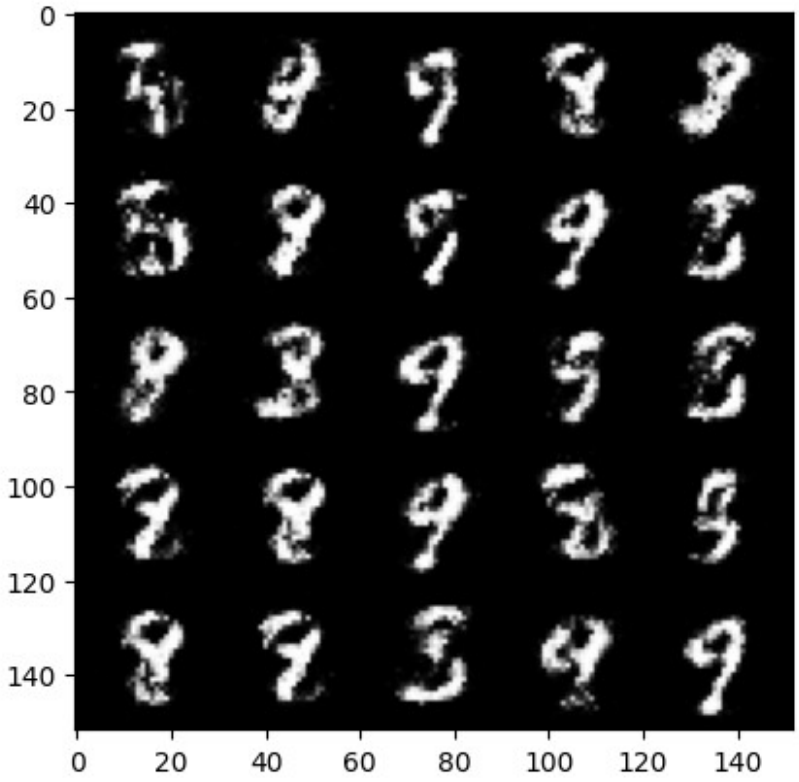
```
Epoch 58, step 27500: Generator loss: 2.7172904186248767,
discriminator loss: 0.19584448160231108
```

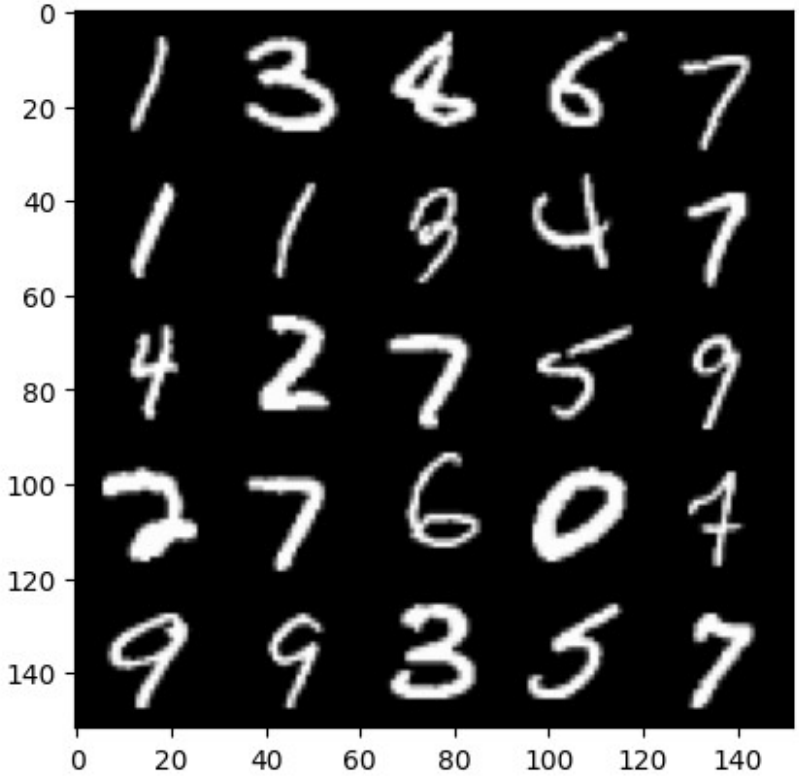




{"model_id":"97601021609547b9911c7268849226c5","version_major":2,"version_minor":0}

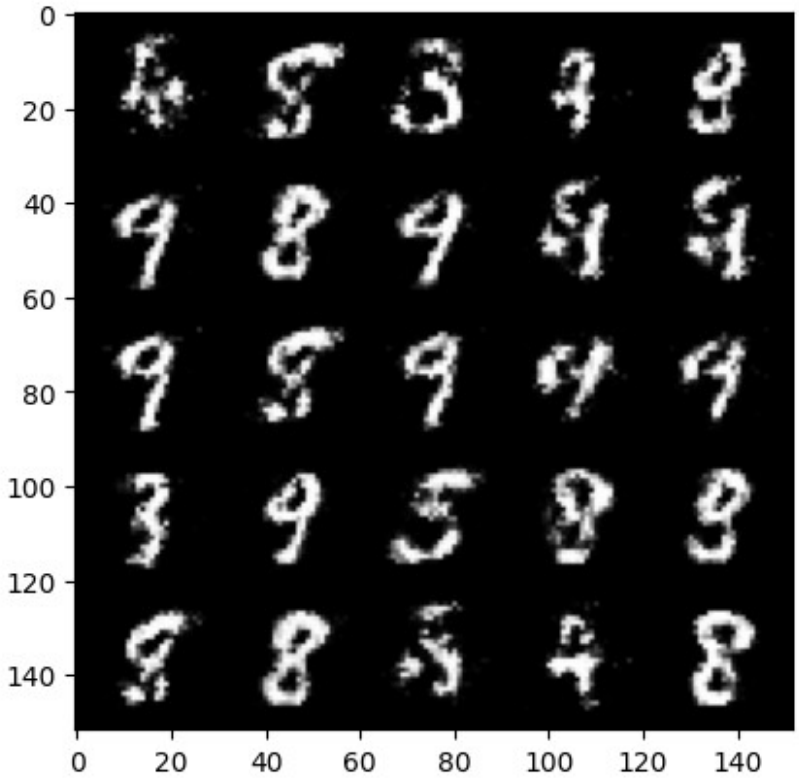
```
Epoch 59, step 28000: Generator loss: 2.8133621578216546,
discriminator loss: 0.18905928996205337
```

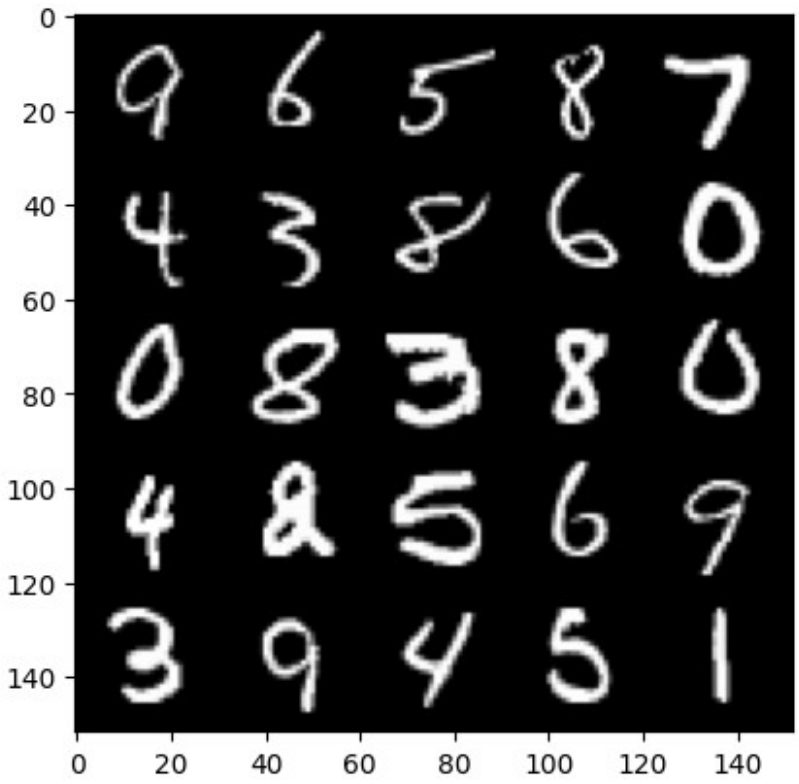

0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140


0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

{"model_id":"0ecac1fe915c4b378a52b3b07709a1ce","version_major":2,"version_minor":0}

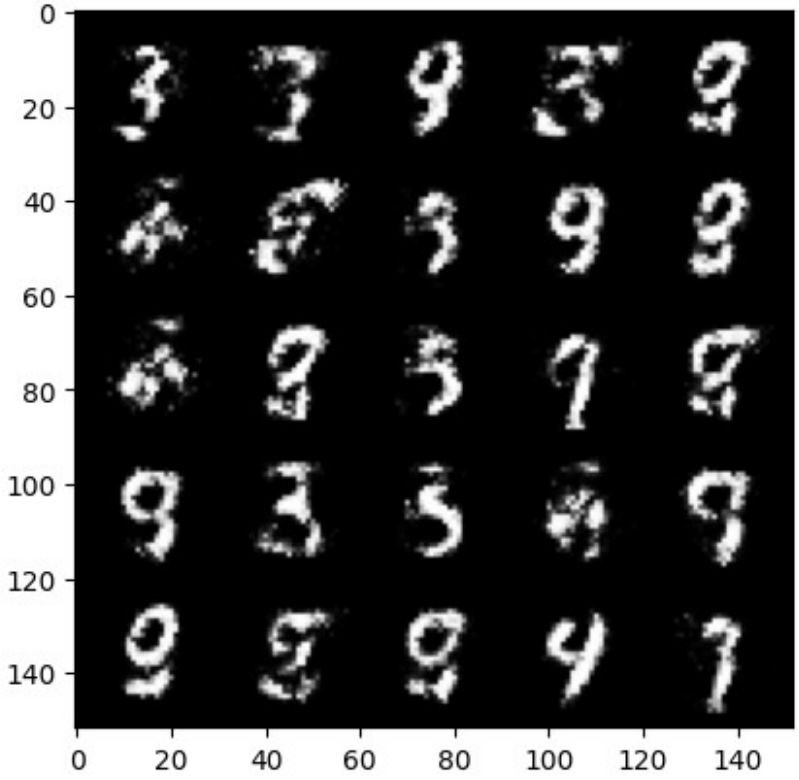
```
Epoch 60, step 28500: Generator loss: 2.748109043121338, discriminator loss: 0.20175519746541964
```

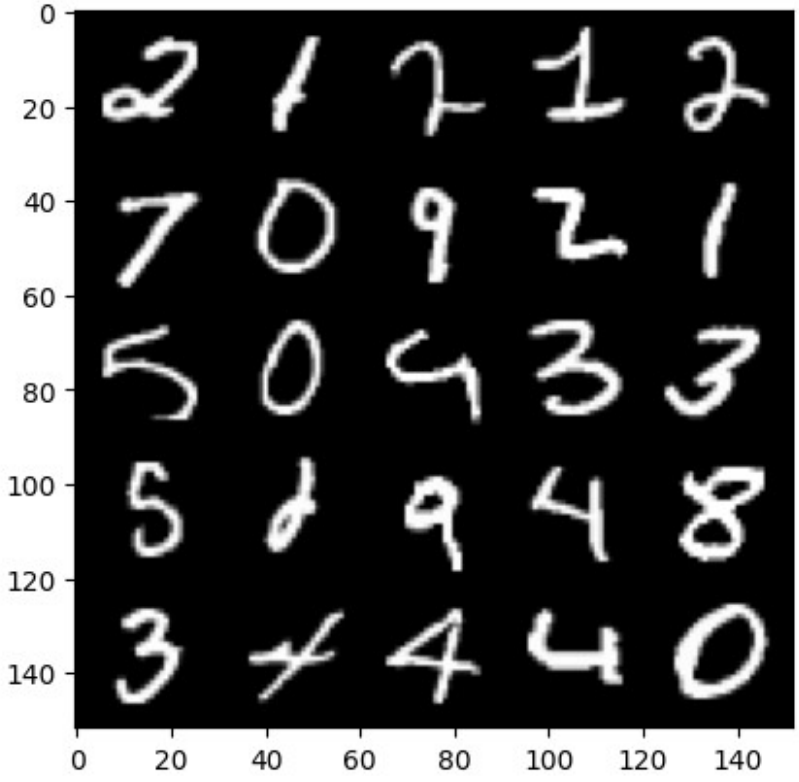




{"model_id":"c10406419e4f4ce8ac6e1f93d7ea76e9","version_major":2,"version_minor":0}

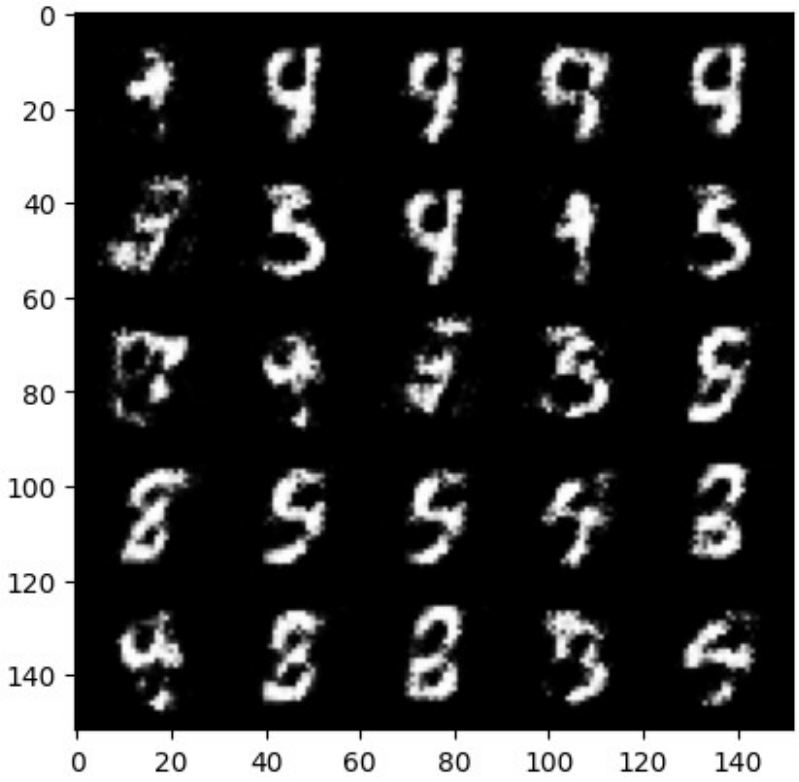
```
Epoch 61, step 29000: Generator loss: 2.700974921226501, discriminator loss: 0.20326546831428985
```

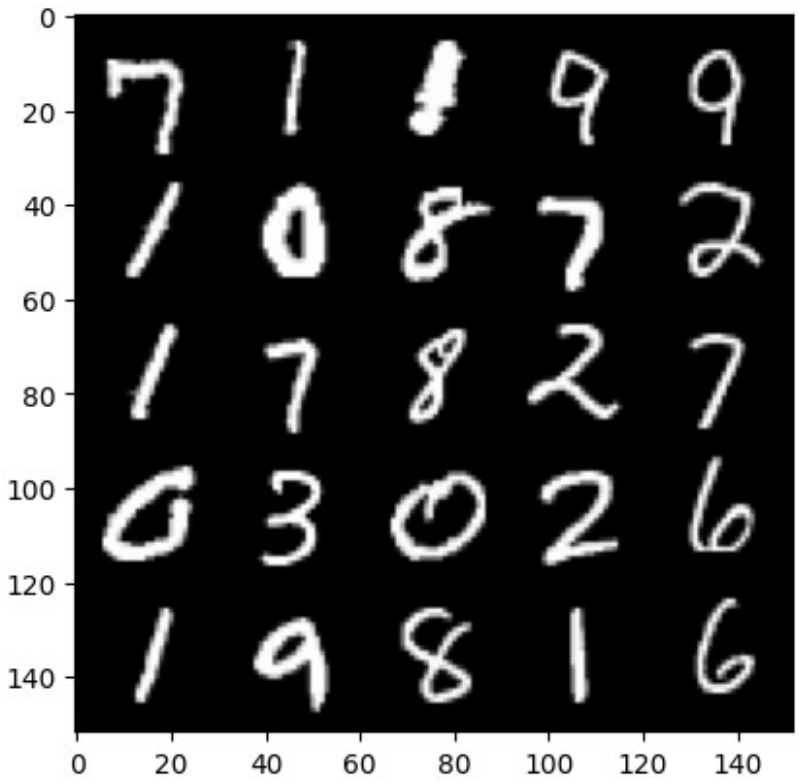

0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140


0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

{"model_id":"9e00acb21cde4d239278fef39481113c","version_major":2,"version_minor":0}

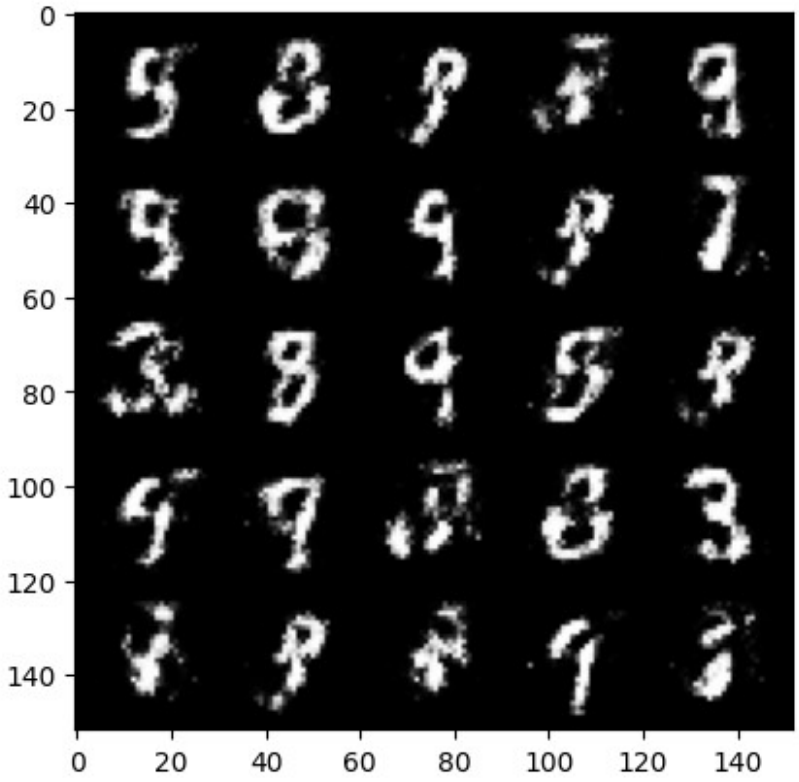
```
Epoch 62, step 29500: Generator loss: 2.6254148361682885,
discriminator loss: 0.21656457133591187
```

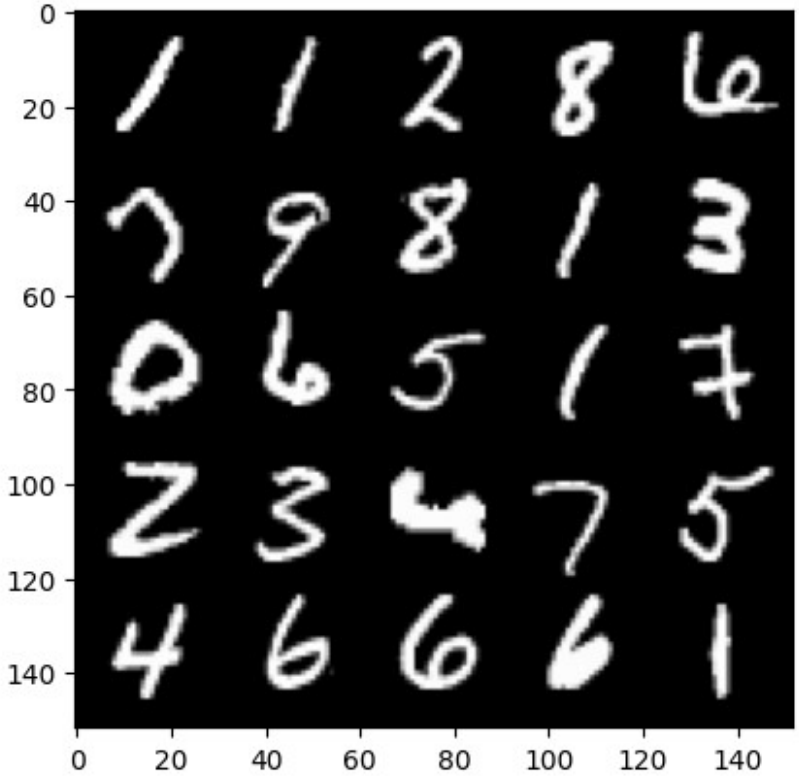




{"model_id":"6477271db1a445fd97af79137a505faf","version_major":2,"version_minor":0}

```
Epoch 63, step 30000: Generator loss: 2.60786535024643, discriminator loss: 0.21809917029738435
```

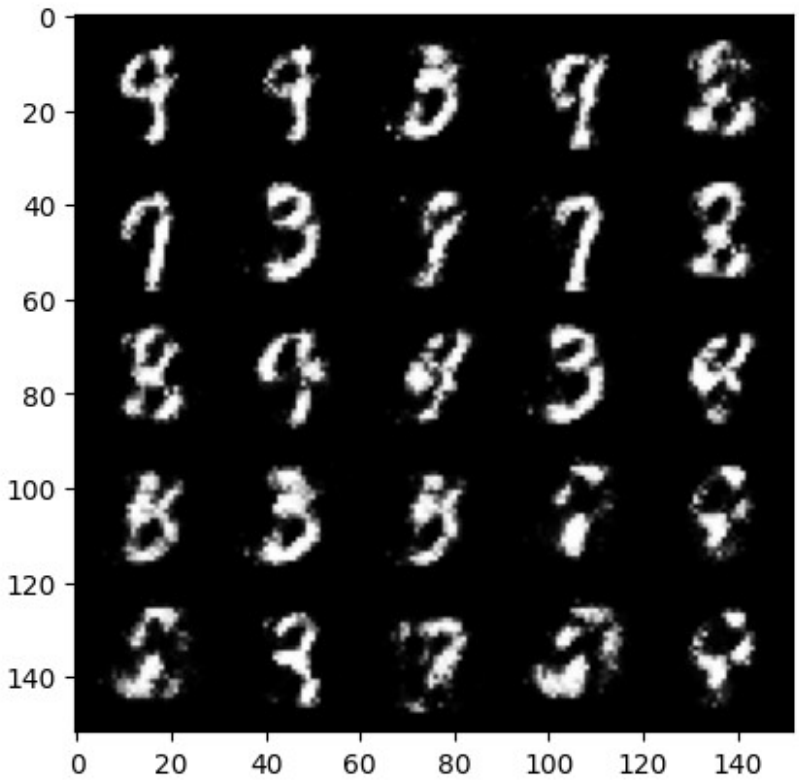

0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

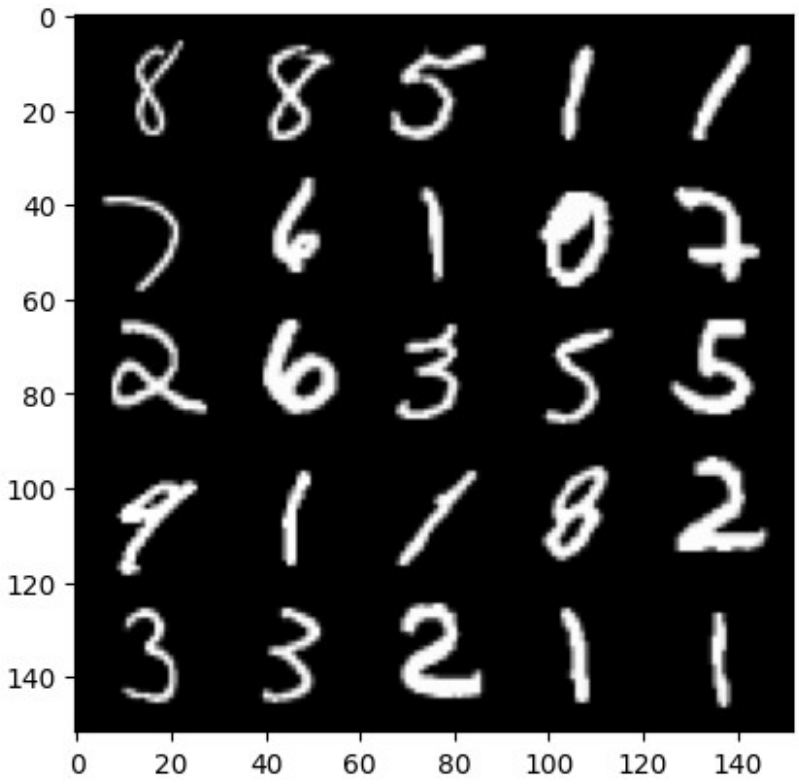

0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

{"model_id":"f3aafc8e4a6c4323992a7f34a669d642","version_major":2,"version_minor":0}

{"model_id":"1a3ca98c303b44068c8bb4d182a8bd15","version_major":2,"version_minor":0}

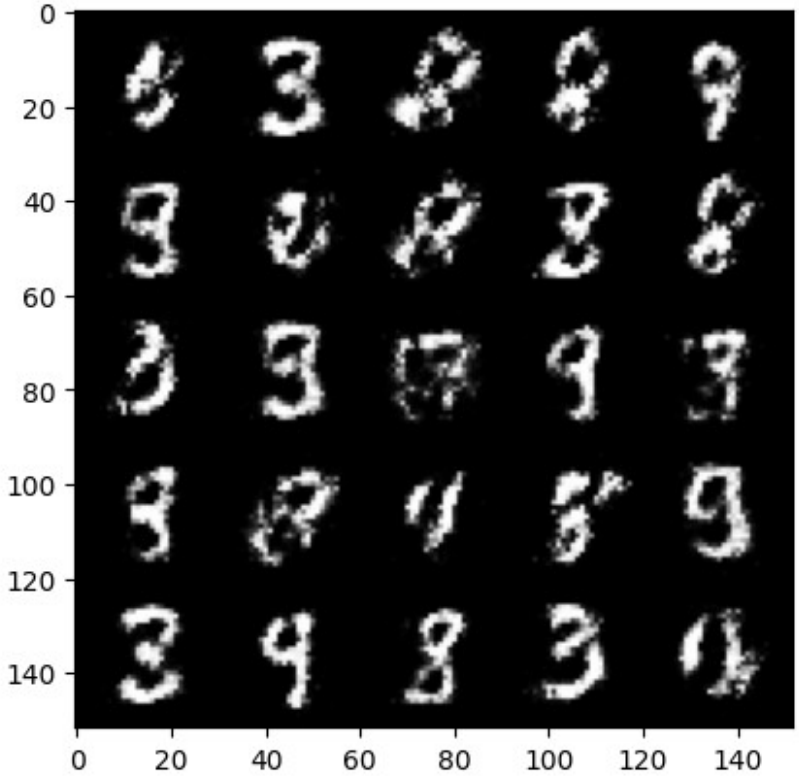
```
Epoch 65, step 30500: Generator loss: 2.550341434001921, discriminator loss: 0.2319159057140349
```

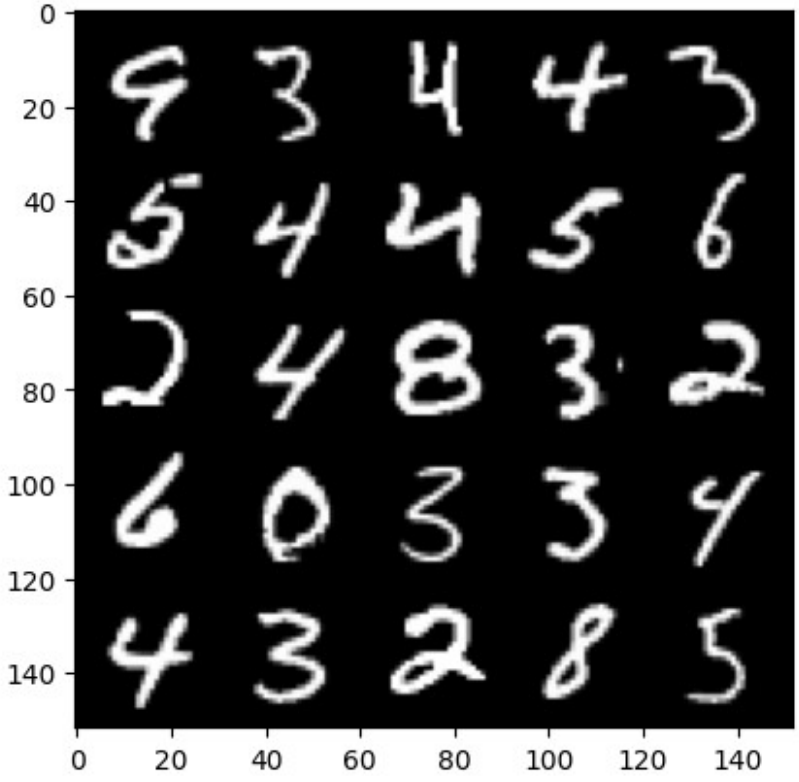




{"model_id":"82cce15b3d4c40bbafe490da612536f8","version_major":2,"version_minor":0}

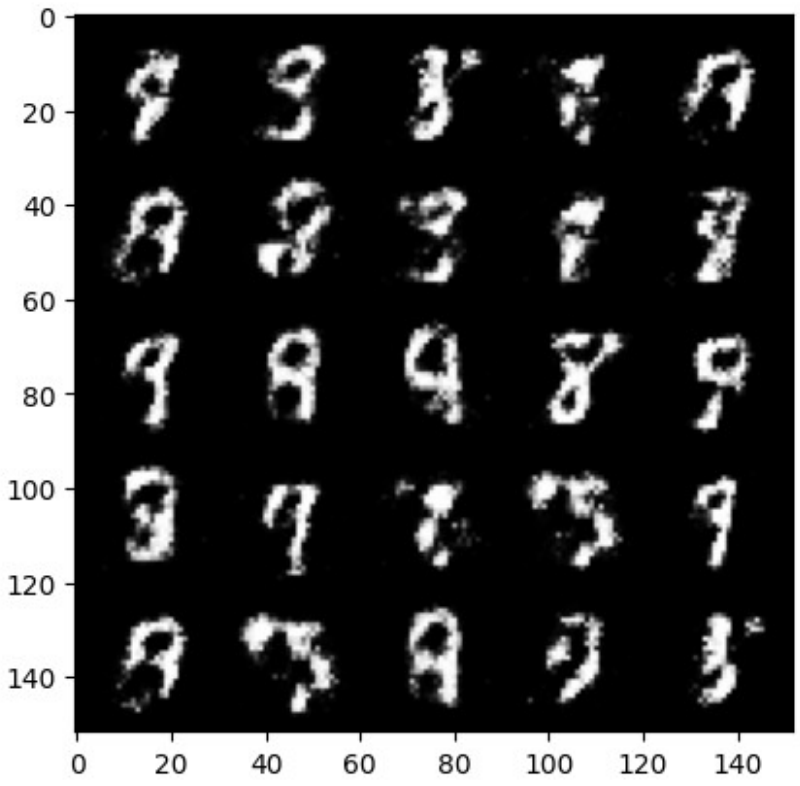
```
Epoch 66, step 31000: Generator loss: 2.52556666254997, discriminator loss: 0.2328094471395013
```

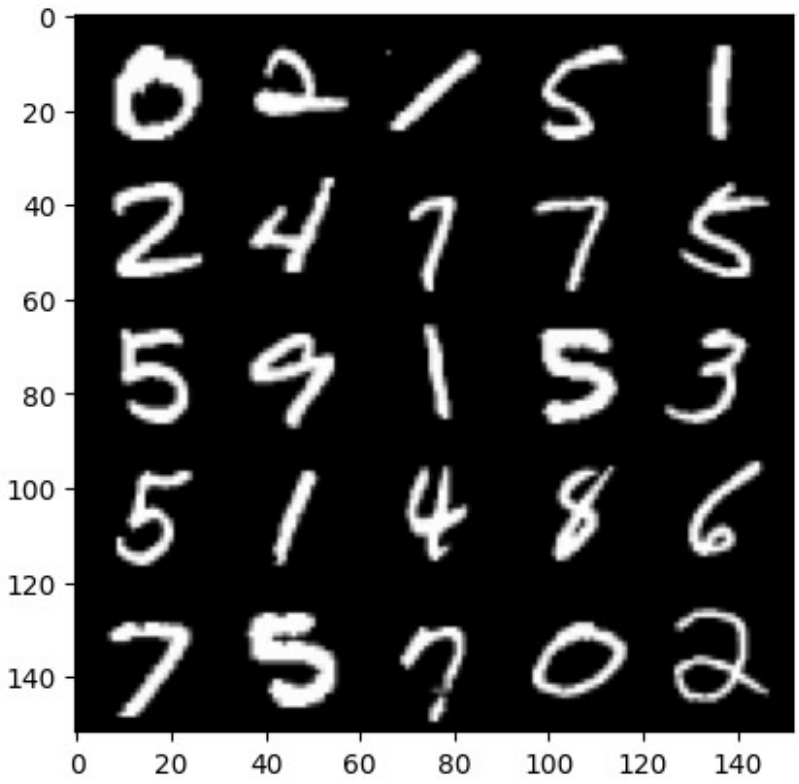

0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140


0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

{"model_id":"96f211003a294dd78ef448f9928af5d9","version_major":2,"version_minor":0}

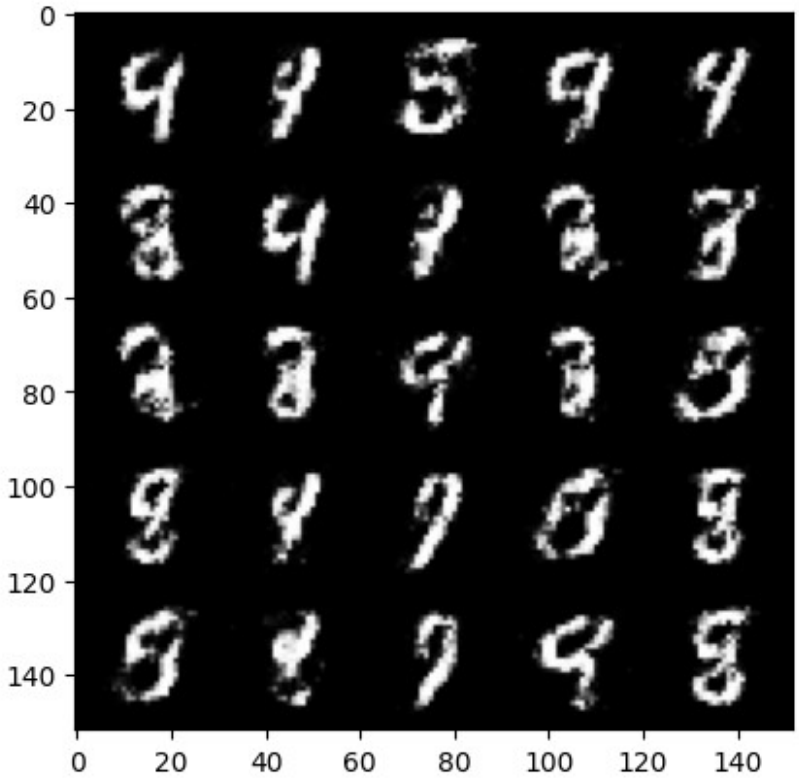
```
Epoch 67, step 31500: Generator loss: 2.4900746090412116,
discriminator loss: 0.23844771465659126
```

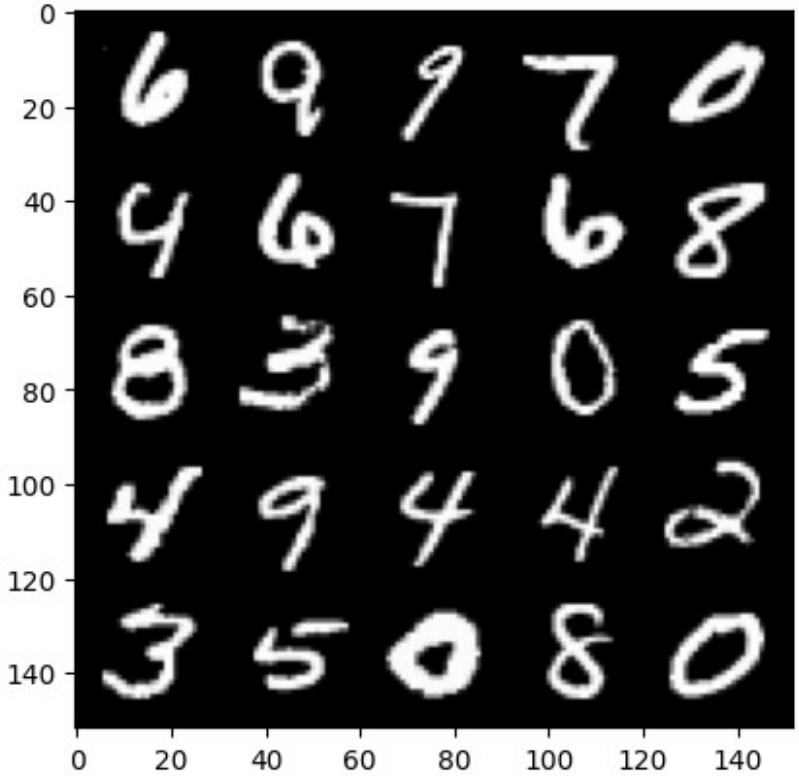




{"model_id":"55b88558c24b4fe68b317229f2060423","version_major":2,"version_minor":0}

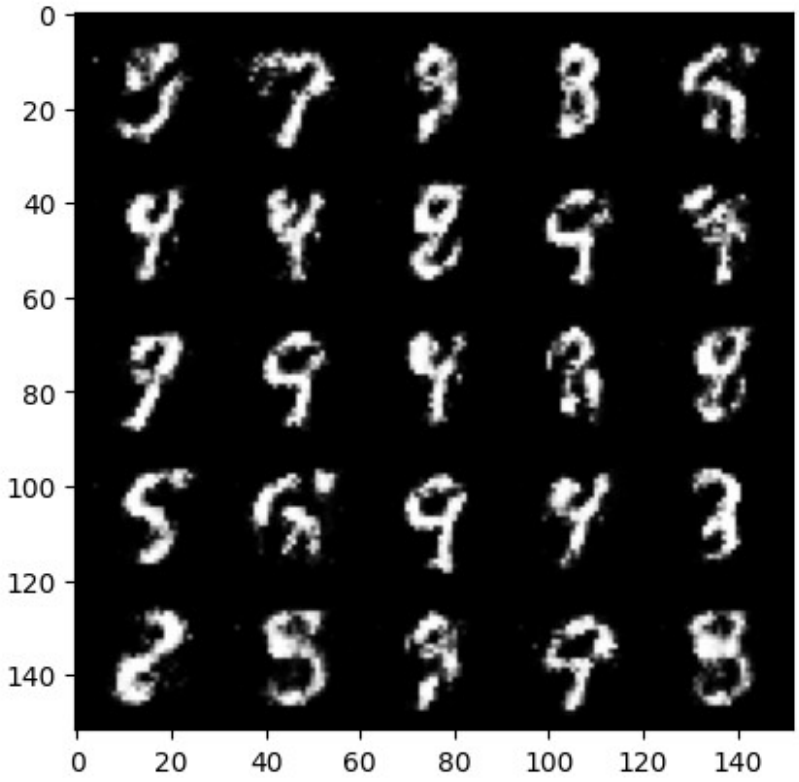
```
Epoch 68, step 32000: Generator loss: 2.5725721511840813,
discriminator loss: 0.22436232122778882
```

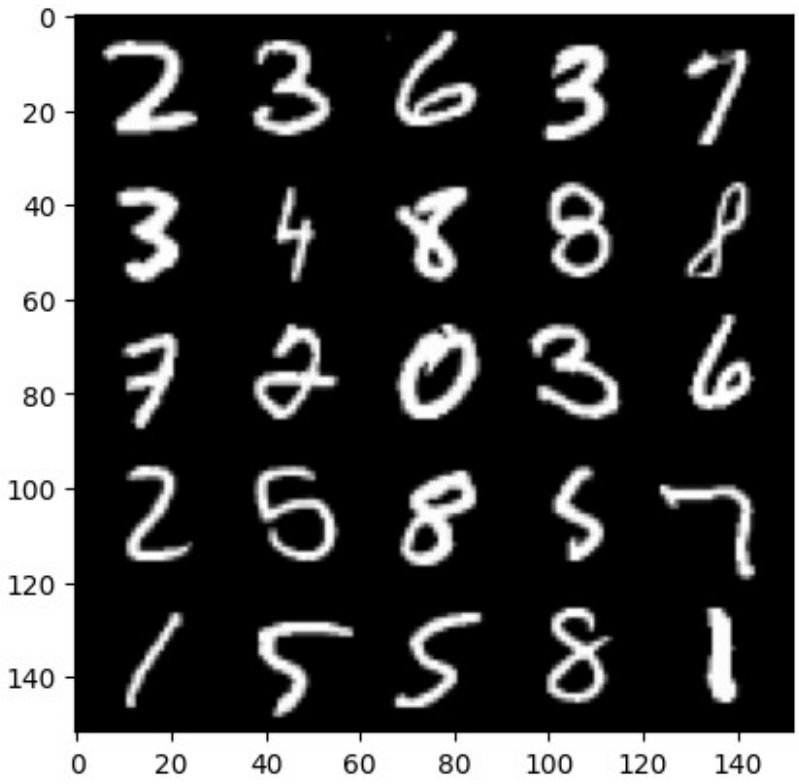

0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140


0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

{"model_id":"dee7fd2a3b1842ce8c3a948a5e134690","version_major":2,"version_minor":0}

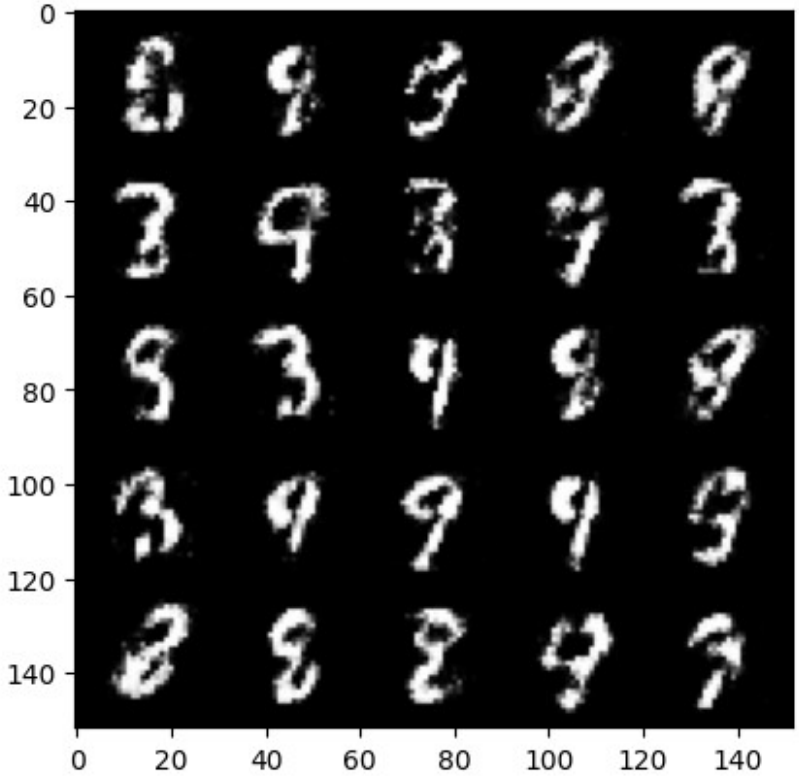
```
Epoch 69, step 32500: Generator loss: 2.4786407928466825,
discriminator loss: 0.235916384994984
```

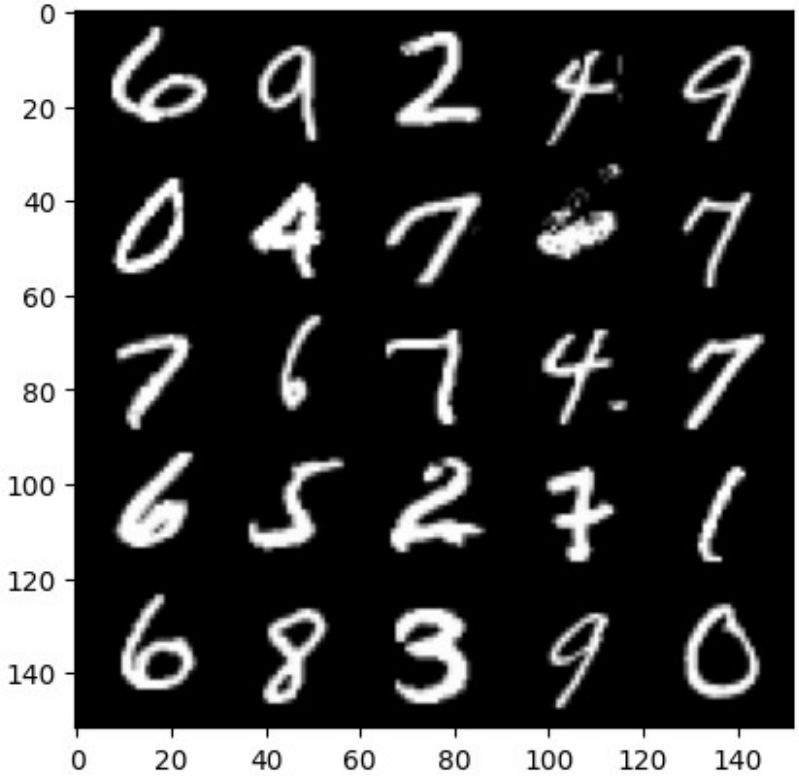




{"model_id":"9a76a57e4176487d8a756013112302a8","version_major":2,"version_minor":0}

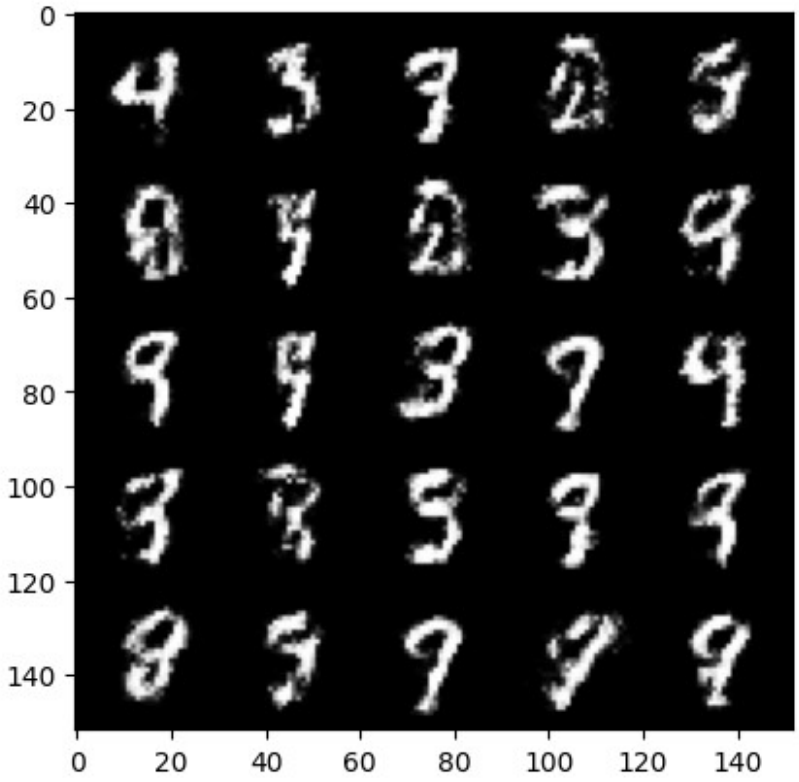
```
Epoch 70, step 33000: Generator loss: 2.455299036502838, discriminator loss: 0.24134469684958454
```

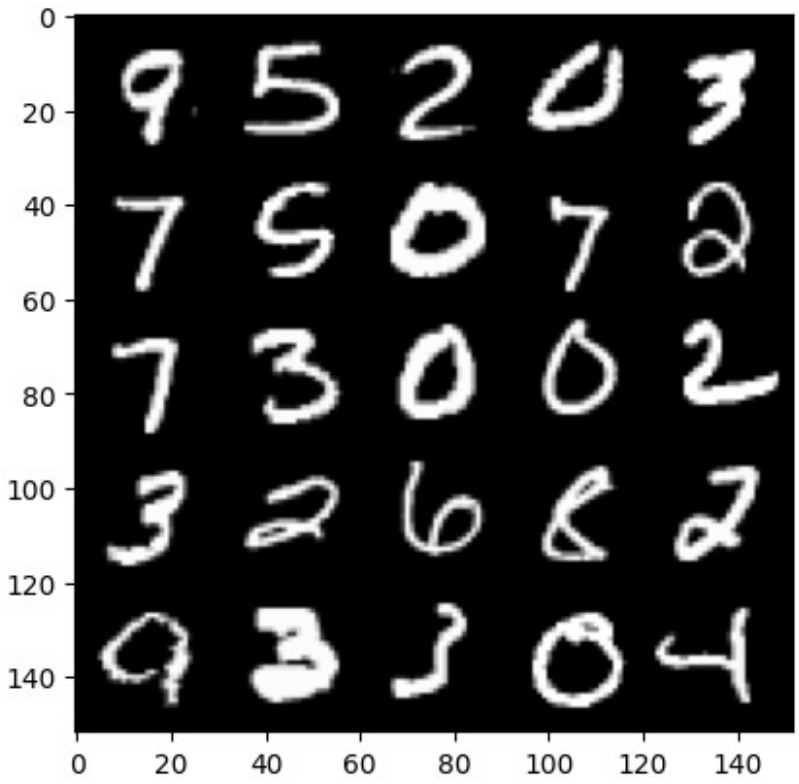

0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140


0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

{"model_id":"319d6c2f9f1844ca81967825c67e0900","version_major":2,"version_minor":0}

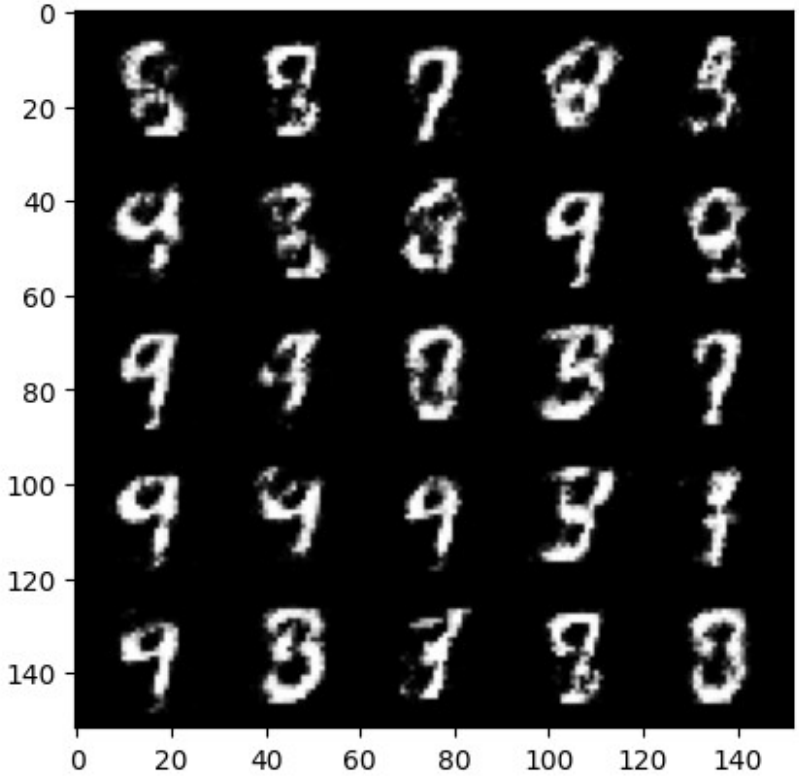
```
Epoch 71, step 33500: Generator loss: 2.3097813191413885,
discriminator loss: 0.25805420044064514
```

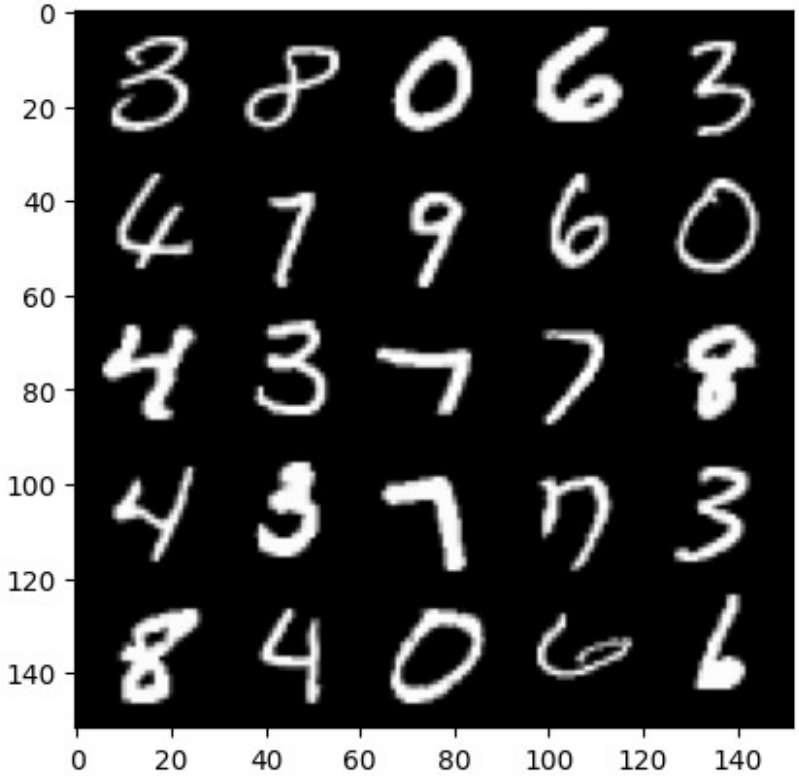




{"model_id":"8bdbb24e724d4096bddeb390a849f33e","version_major":2,"version_minor":0}

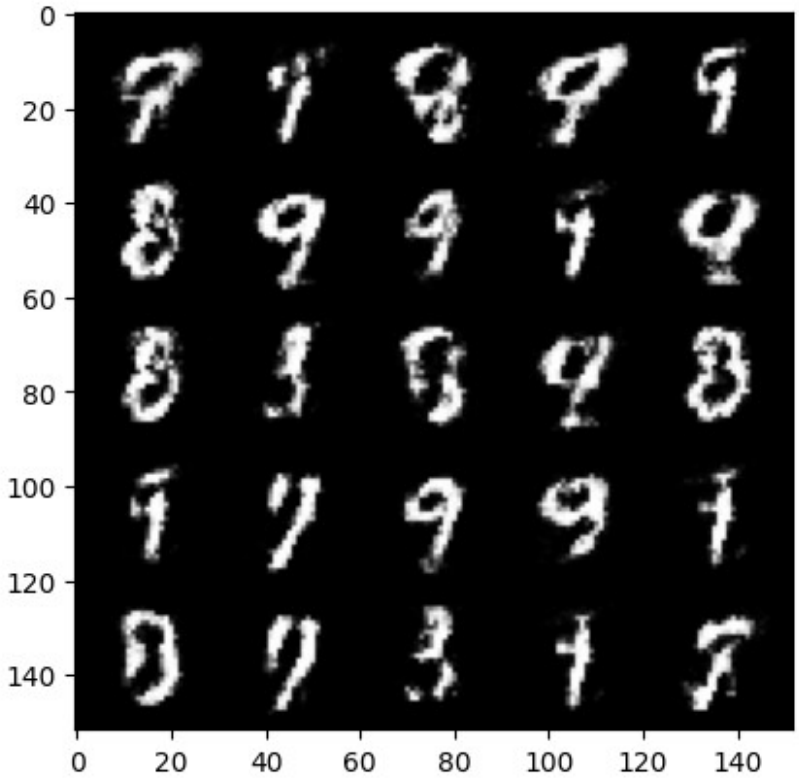
```
Epoch 72, step 34000: Generator loss: 2.290262448549271, discriminator loss: 0.2583626323342322
```

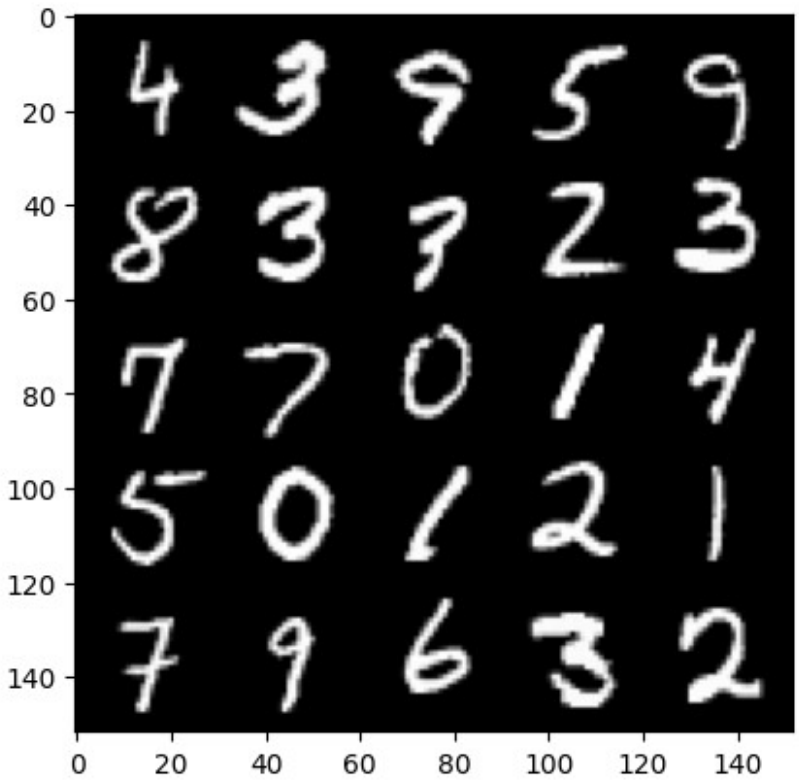

0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140


0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

{"model_id":"c50193be30f842faa1519809ba9c6d33","version_major":2,"version_minor":0}

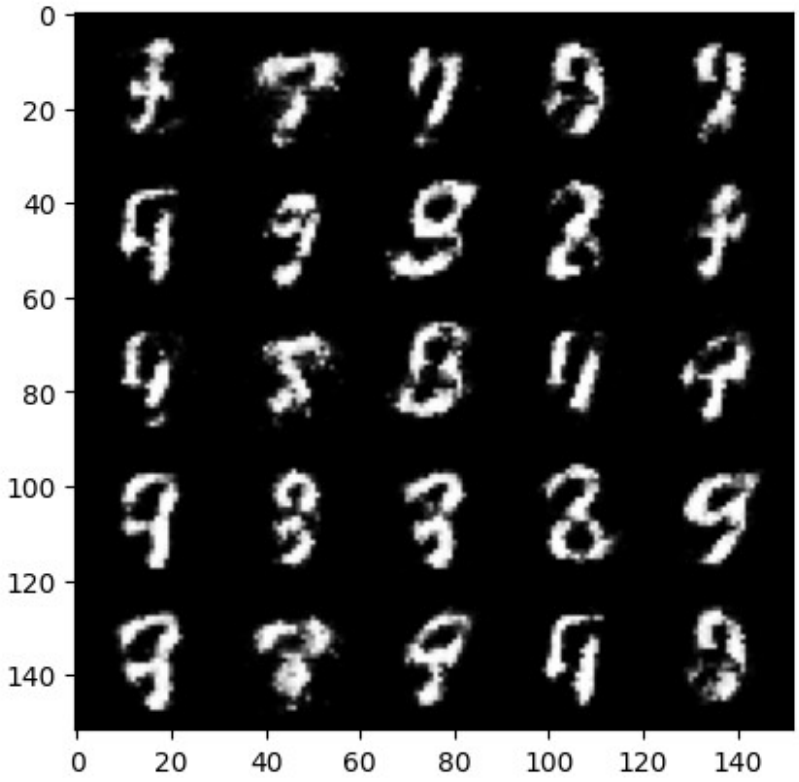
```
Epoch 73, step 34500: Generator loss: 2.1726385078430175,
discriminator loss: 0.2919870389699936
```

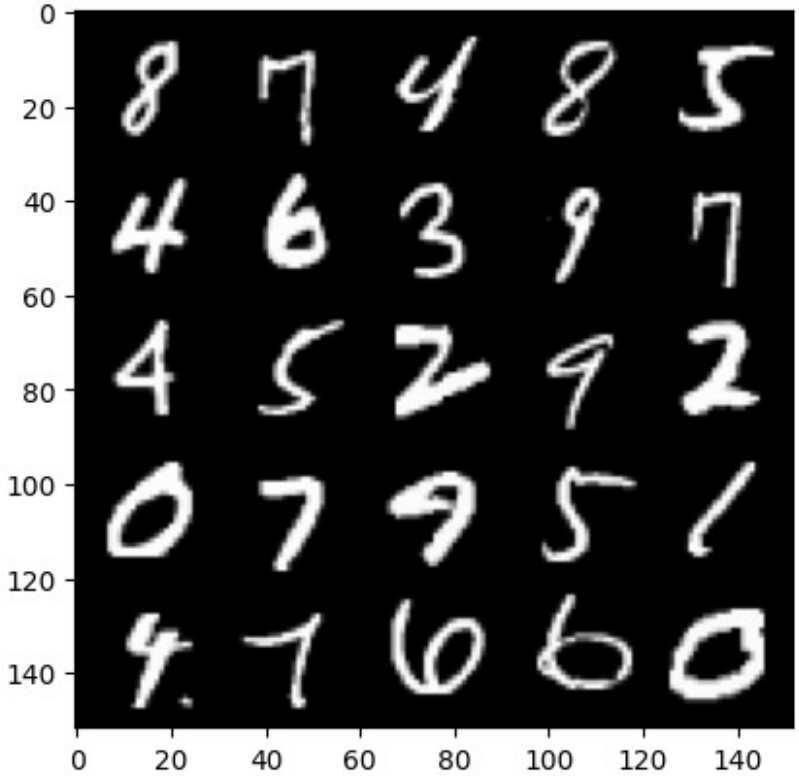




{"model_id":"50163c6be5924db4901f8cb5429d6e77","version_major":2,"version_minor":0}

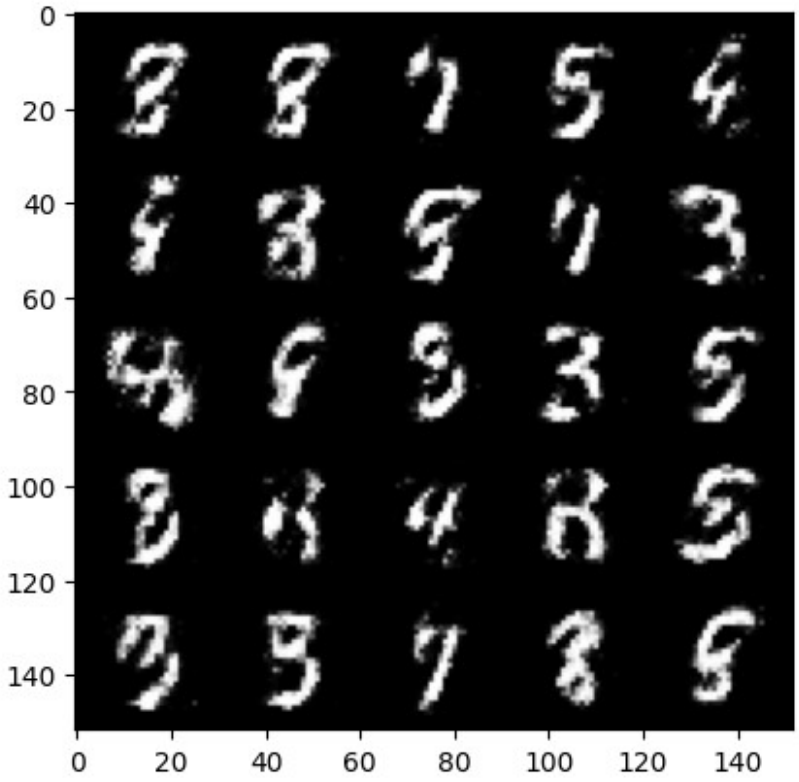
```
Epoch 74, step 35000: Generator loss: 2.1528872311115275,
discriminator loss: 0.2849010809063912
```

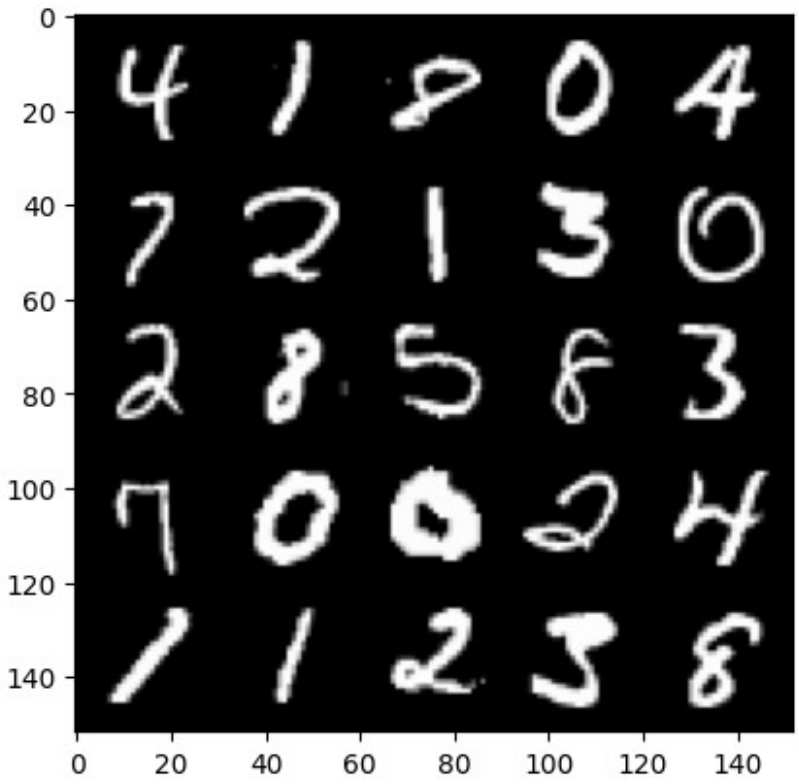

0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140


0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

{"model_id":"a0e05dae20dc4fde929b6db87945770c","version_major":2,"version_minor":0}

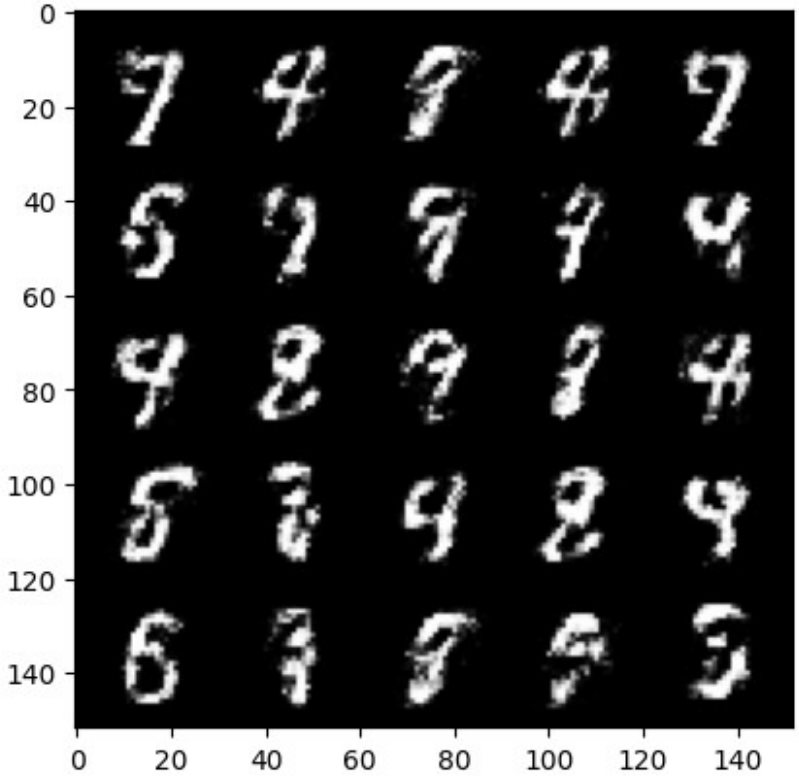
```
Epoch 75, step 35500: Generator loss: 2.178538301467895, discriminator loss: 0.26386456972360617
```

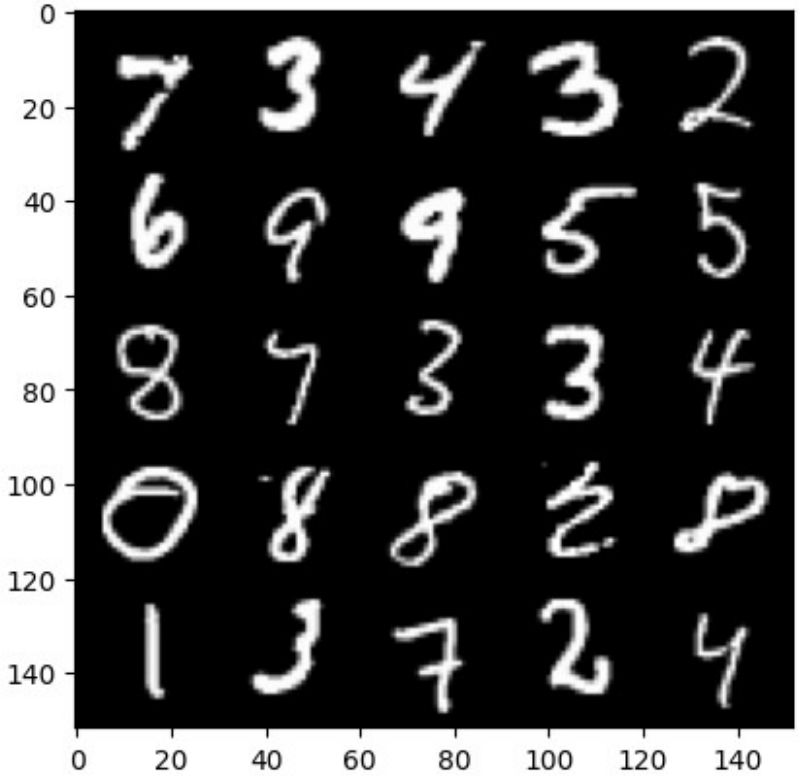




{"model_id":"8475d95625e34eeb8a353d28ed264e76","version_major":2,"version_minor":0}

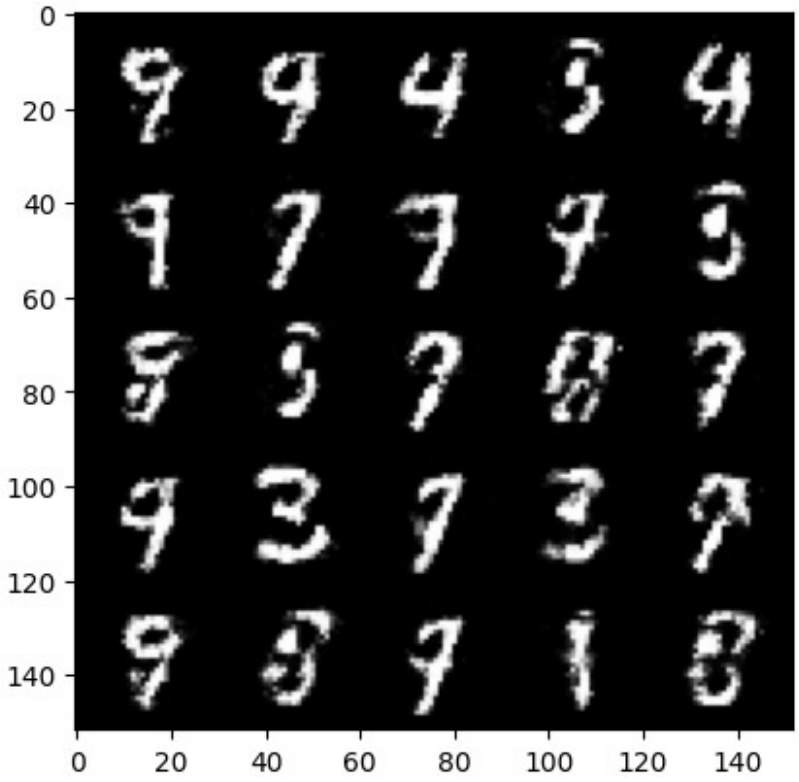
```
Epoch 76, step 36000: Generator loss: 2.2543047492504122,
discriminator loss: 0.25314573326706896
```

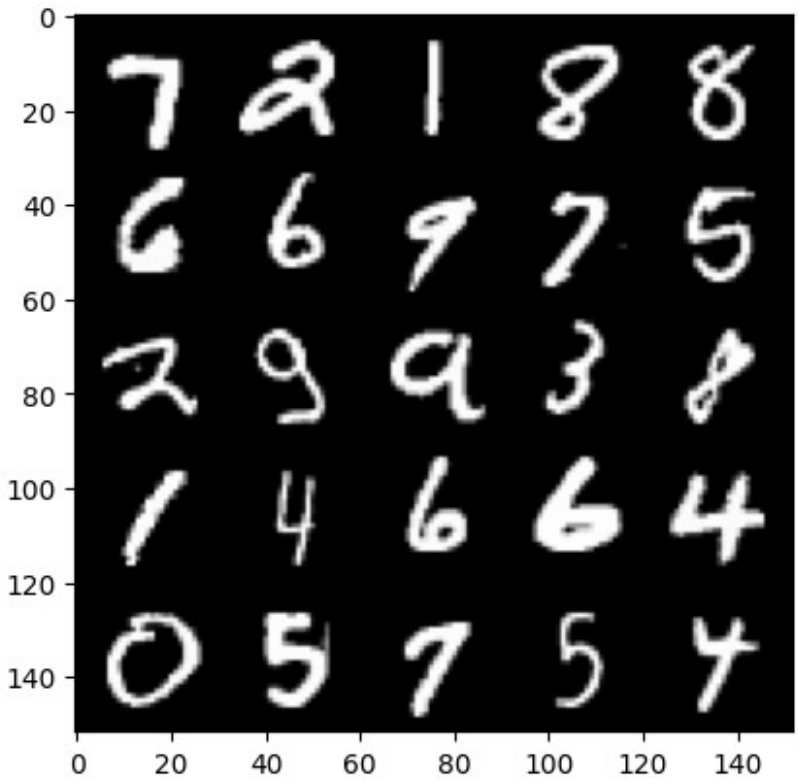

0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140


0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

{"model_id":"1adb1533f9764784a700fef7e80ce623","version_major":2,"version_minor":0}

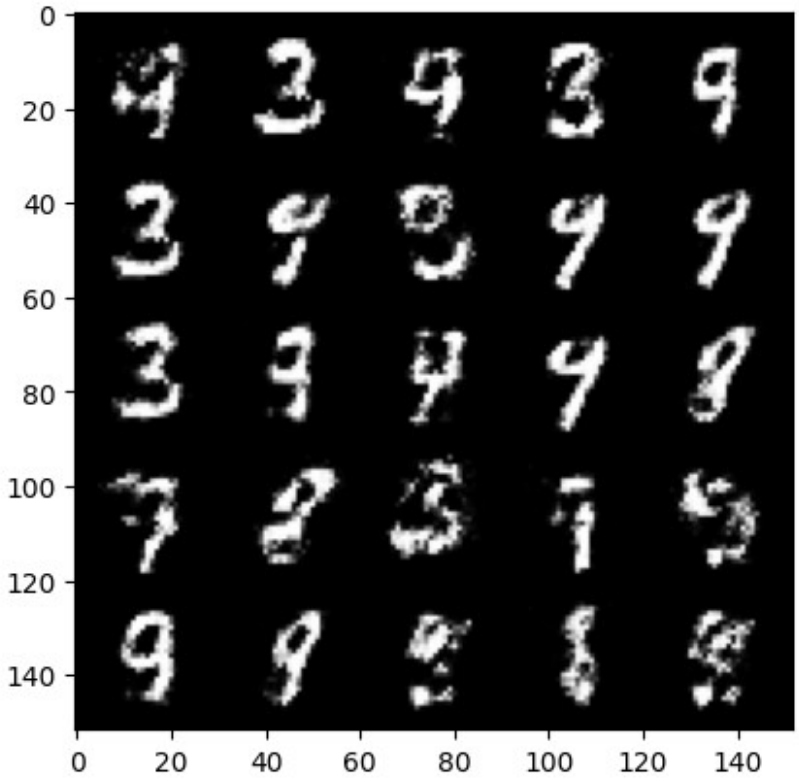
```
Epoch 77, step 36500: Generator loss: 2.3165097384452826,
discriminator loss: 0.25293912997841833
```

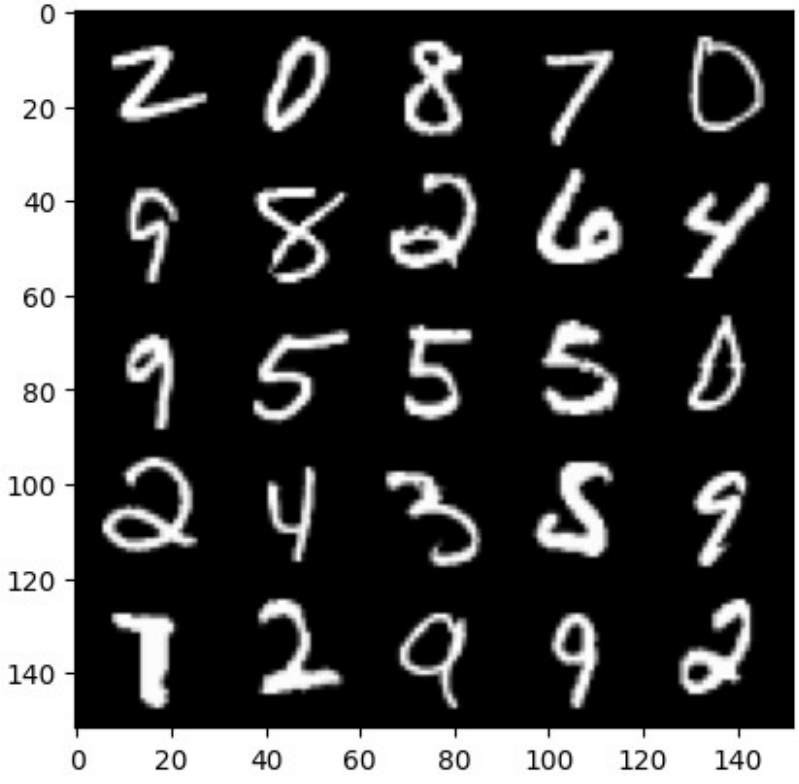




{"model_id":"5238634807774e5593d7ec53a012ce4b","version_major":2,"version_minor":0}

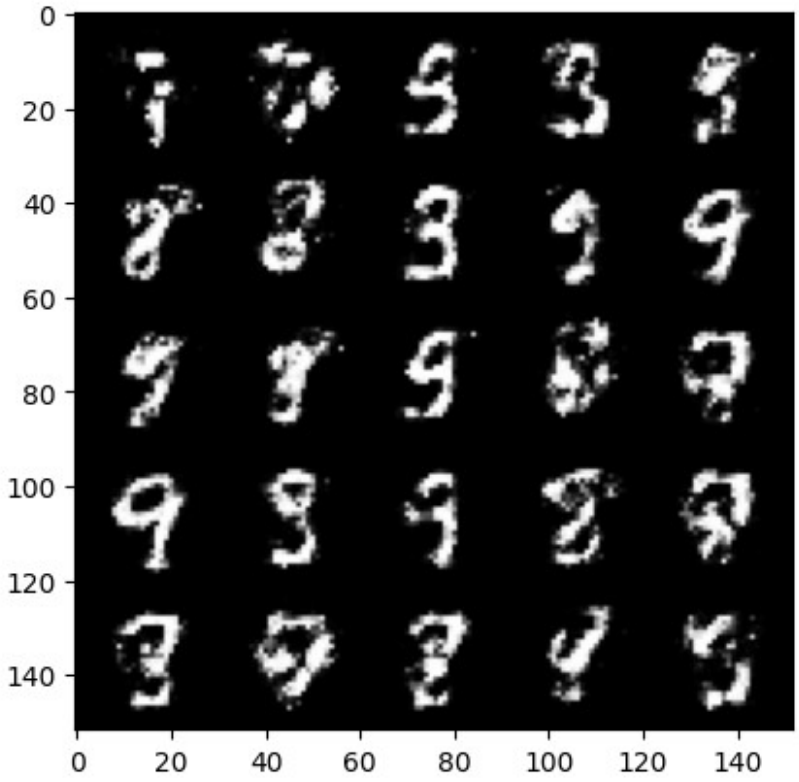
```
Epoch 78, step 37000: Generator loss: 2.312712960243226, discriminator loss: 0.25047641199827186
```

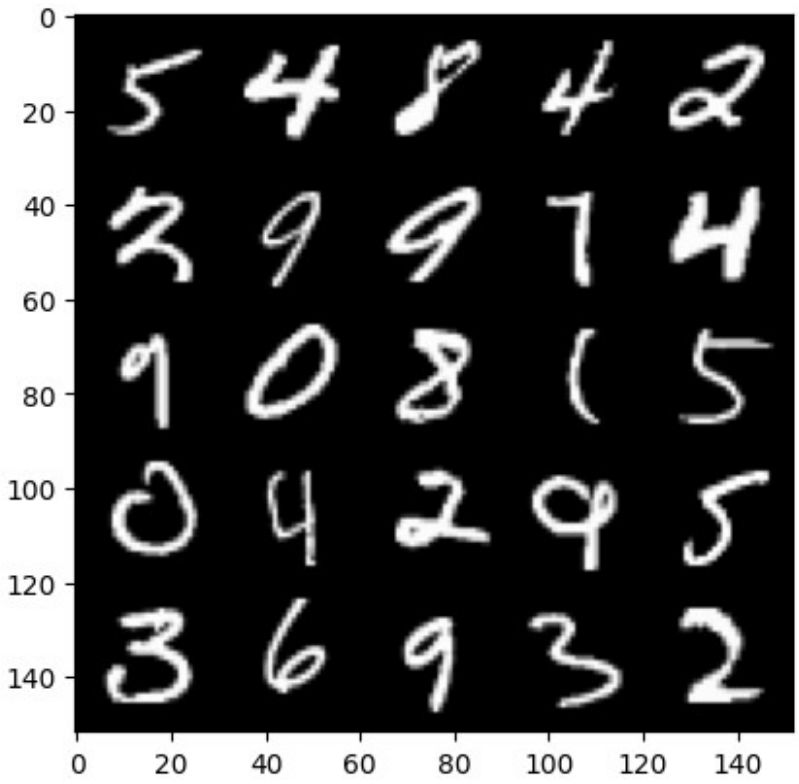

0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140


0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

{"model_id":"93e030b63ad149fbb203016c0d24ee7d","version_major":2,"version_minor":0}

```
Epoch 79, step 37500: Generator loss: 2.3390510284900694,
discriminator loss: 0.24893252319097522
```

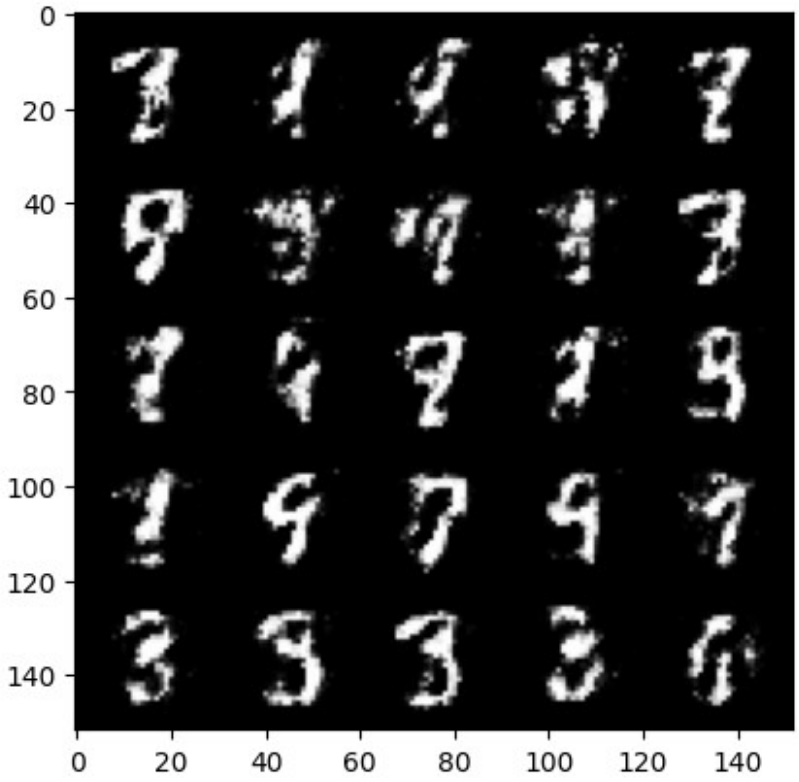


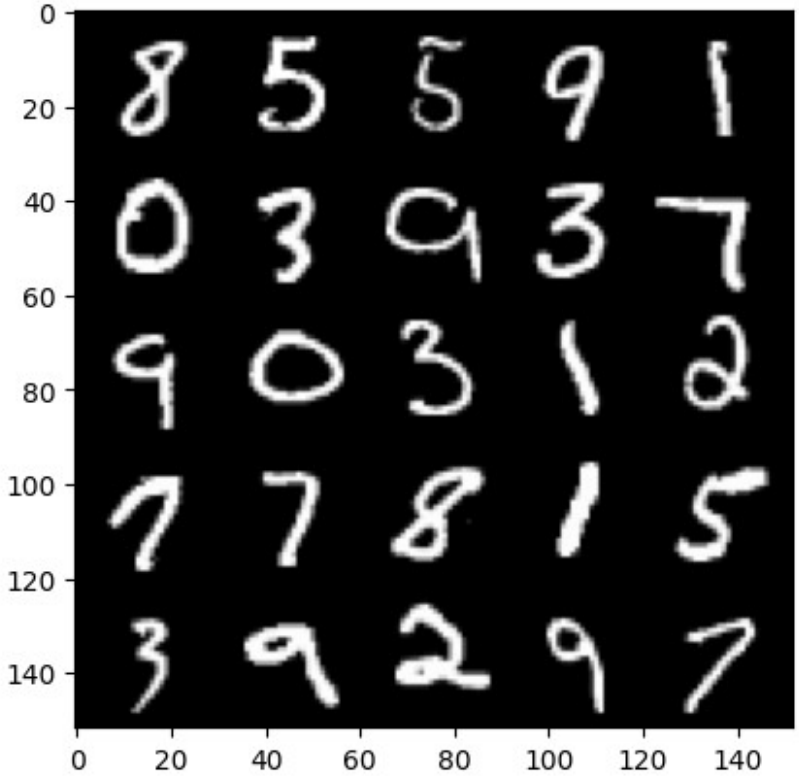


{"model_id":"ad986ba8482143839414c173ec4dcd5f","version_major":2,"version_minor":0}

{"model_id":"b7311eaf7bb74fb48c246744b2c1f869","version_major":2,"version_minor":0}

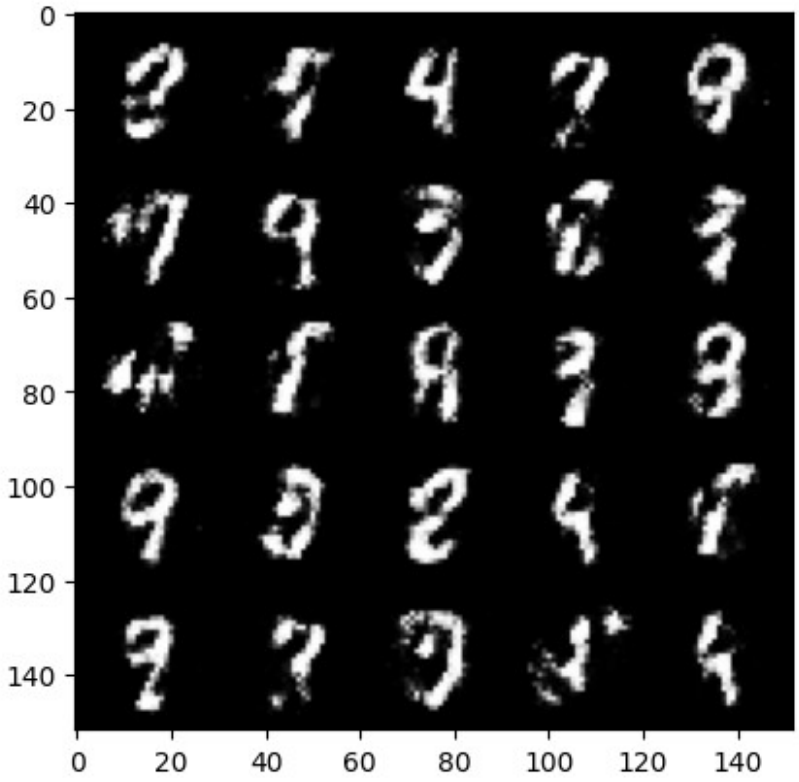
```
Epoch 81, step 38000: Generator loss: 2.3535137720108015,
discriminator loss: 0.25592084661126135
```

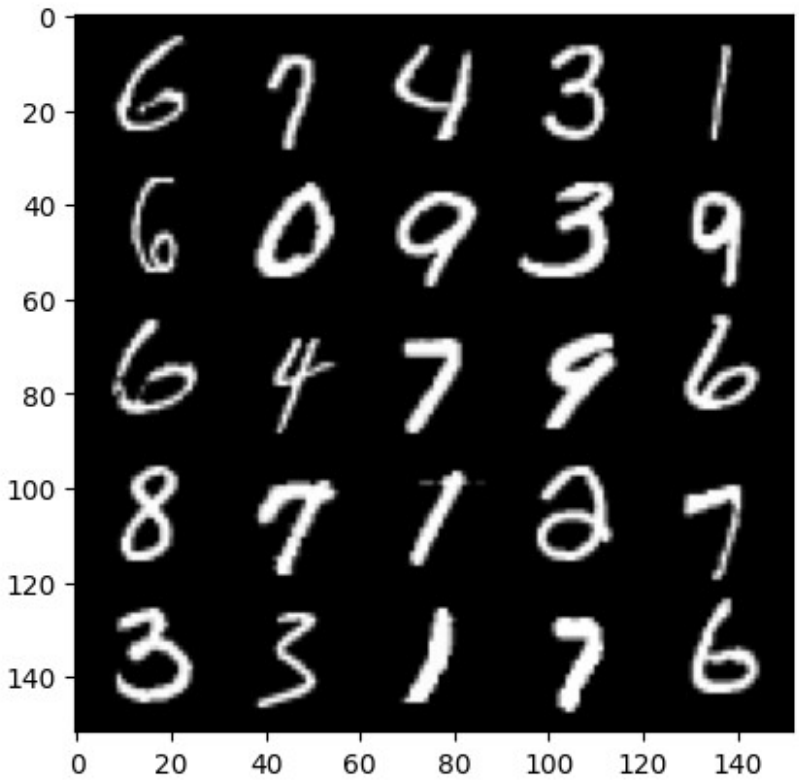

0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140


0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

{"model_id":"26b74af669734696b2520a4280e440af","version_major":2,"version_minor":0}

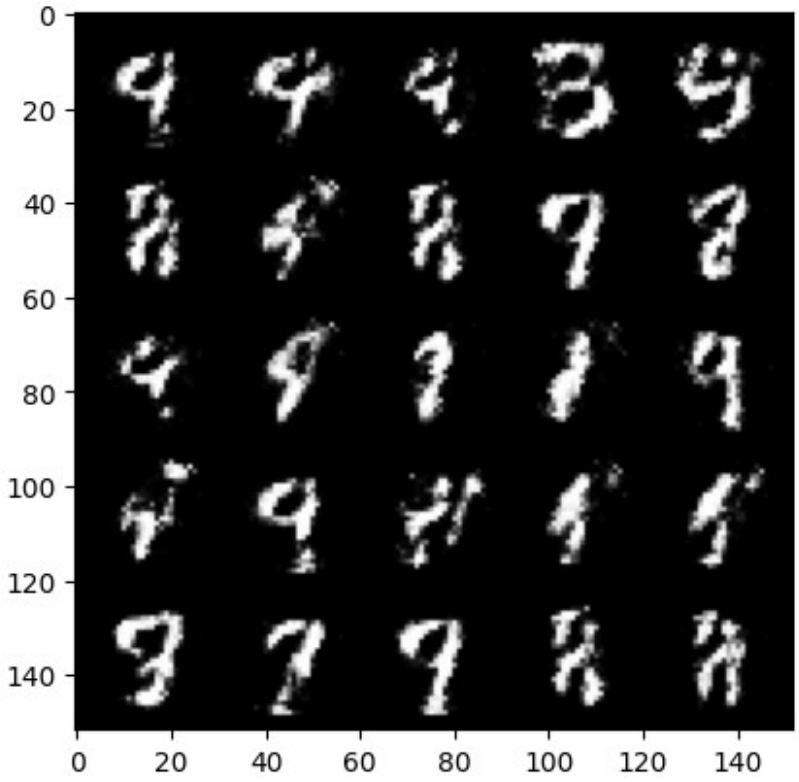
```
Epoch 82, step 38500: Generator loss: 2.2600471658706662,
discriminator loss: 0.2691488147079947
```

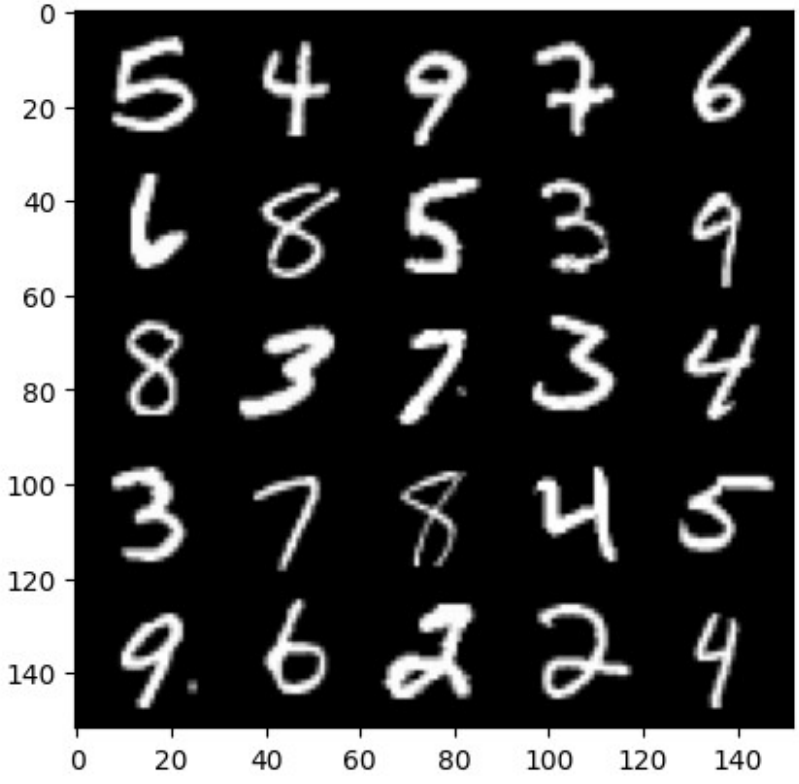




{"model_id":"7de9754be3d9420ca531d2bf0a948223","version_major":2,"version_minor":0}

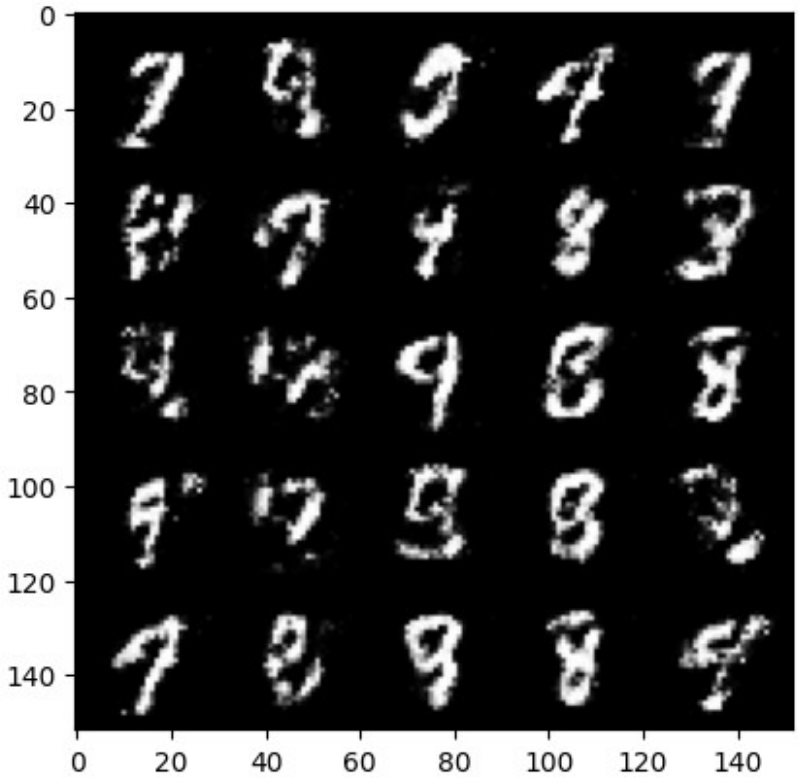
```
Epoch 83, step 39000: Generator loss: 2.1694052565097817,
discriminator loss: 0.28218802756071104
```

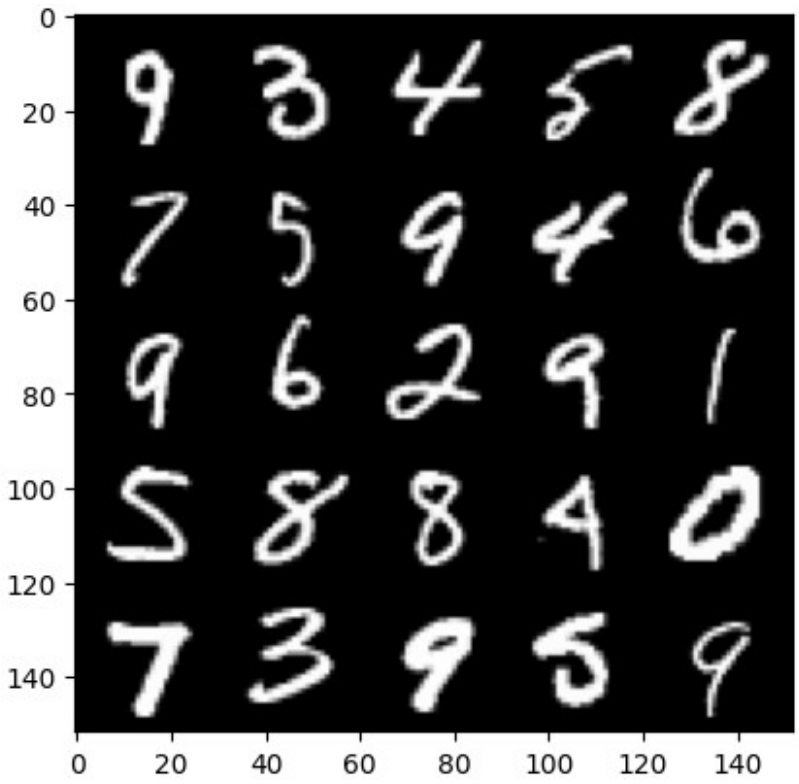

0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140


0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

{"model_id":"c299dd3db1d44ecc88974c7f6a5d20f8","version_major":2,"version_minor":0}

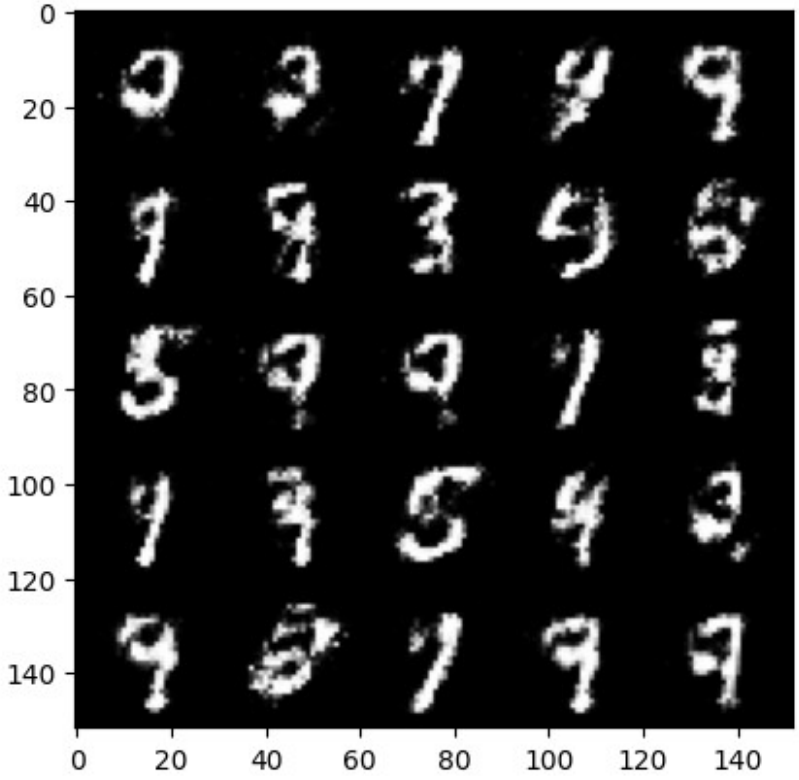
```
Epoch 84, step 39500: Generator loss: 2.10688072323799, discriminator loss: 0.30659259110689197
```

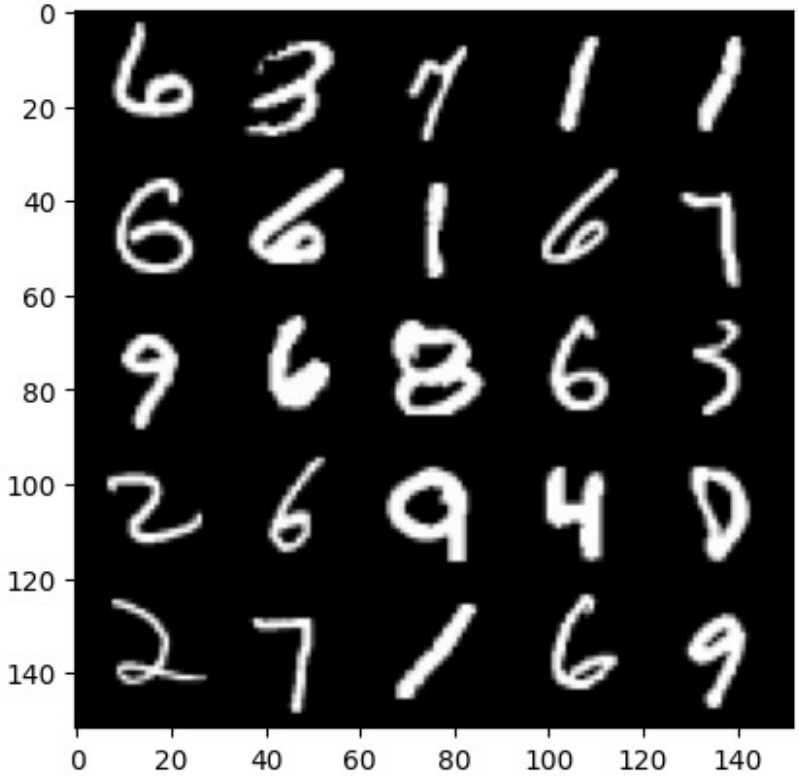




{"model_id":"901d249b3f05405a981062896c9b0856","version_major":2,"version_minor":0}

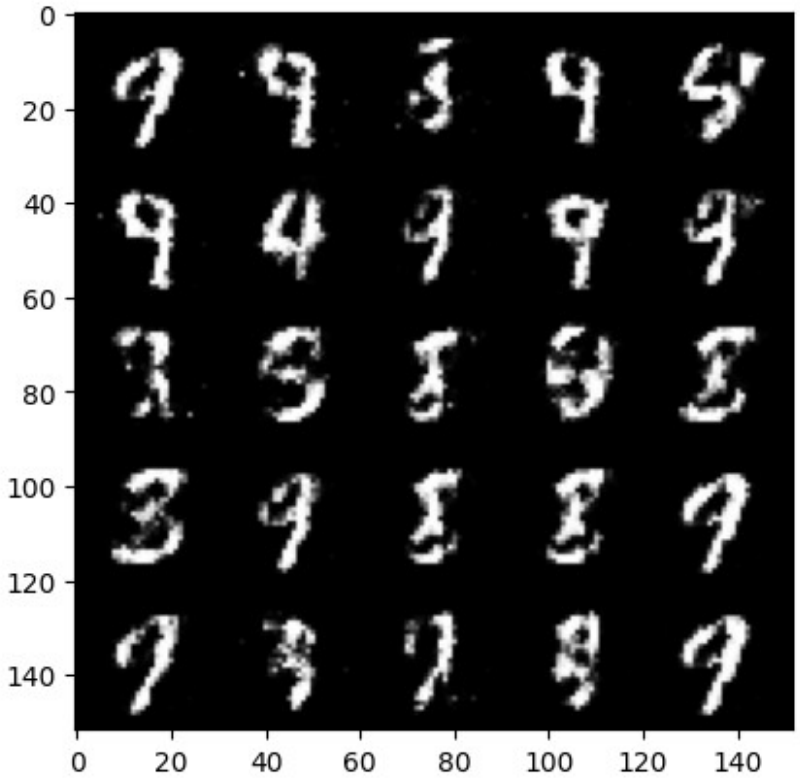
```
Epoch 85, step 40000: Generator loss: 2.08184289431572, discriminator loss: 0.2979447114169599
```

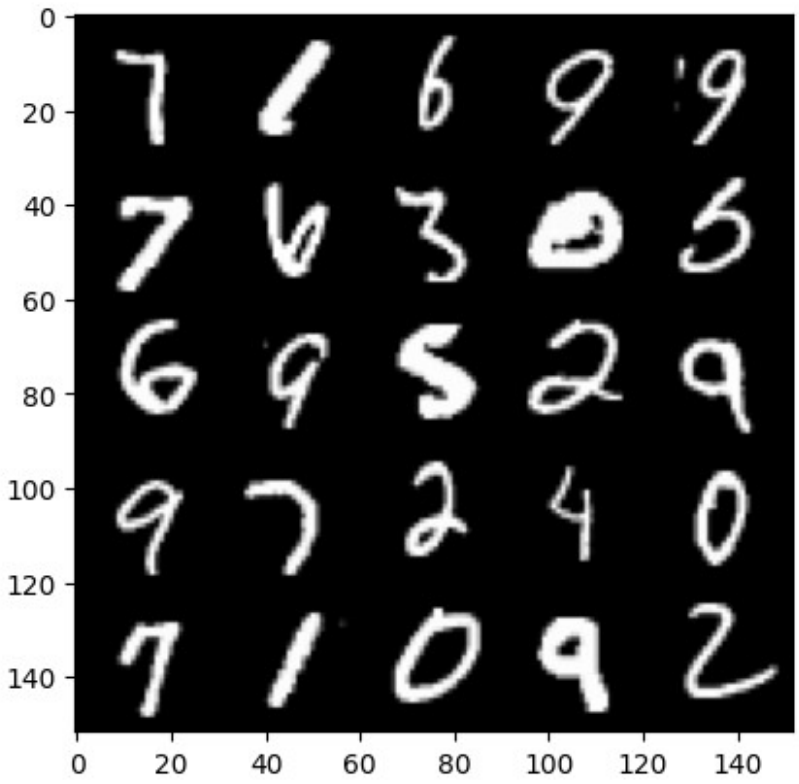

0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140


0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

{"model_id":"41d46cbfb4aa4b92986cb4e03d8ebb0e","version_major":2,"version_minor":0}

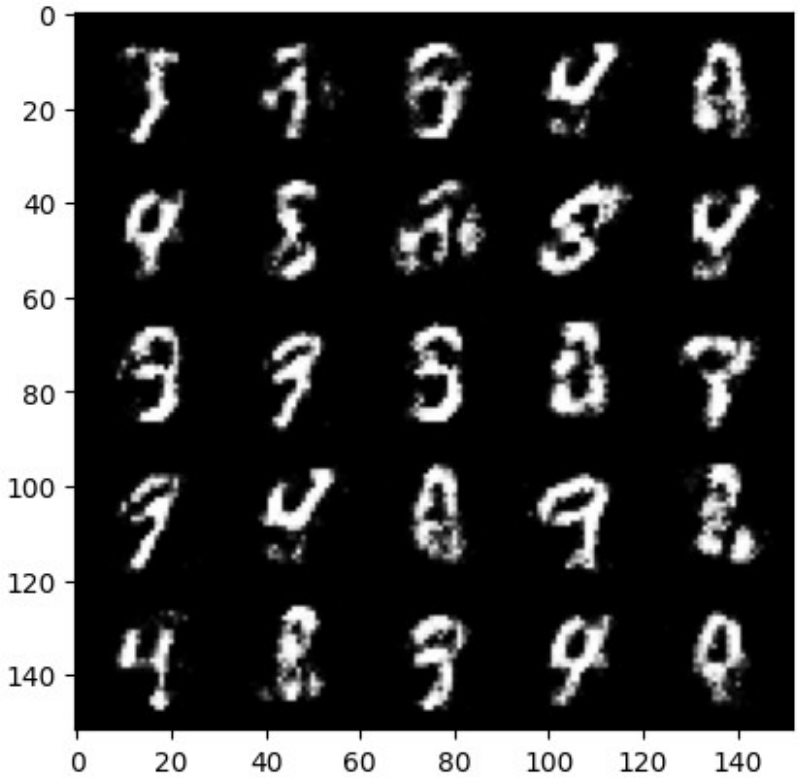
```
Epoch 86, step 40500: Generator loss: 2.105682126998902, discriminator loss: 0.28599080571532237
```

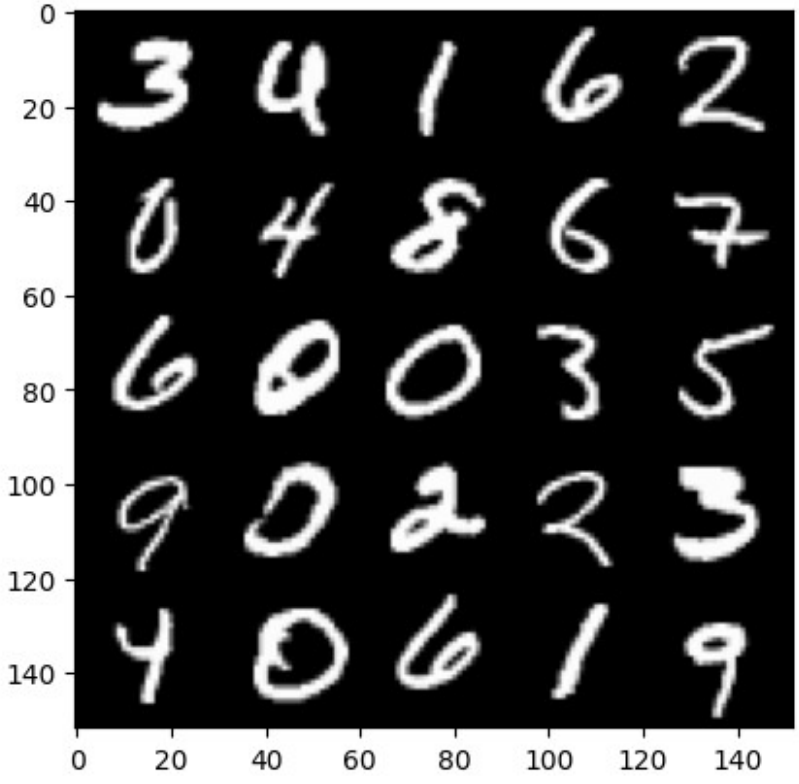




{"model_id":"f51754a572a84a0b9604379952a7fc98","version_major":2,"version_minor":0}

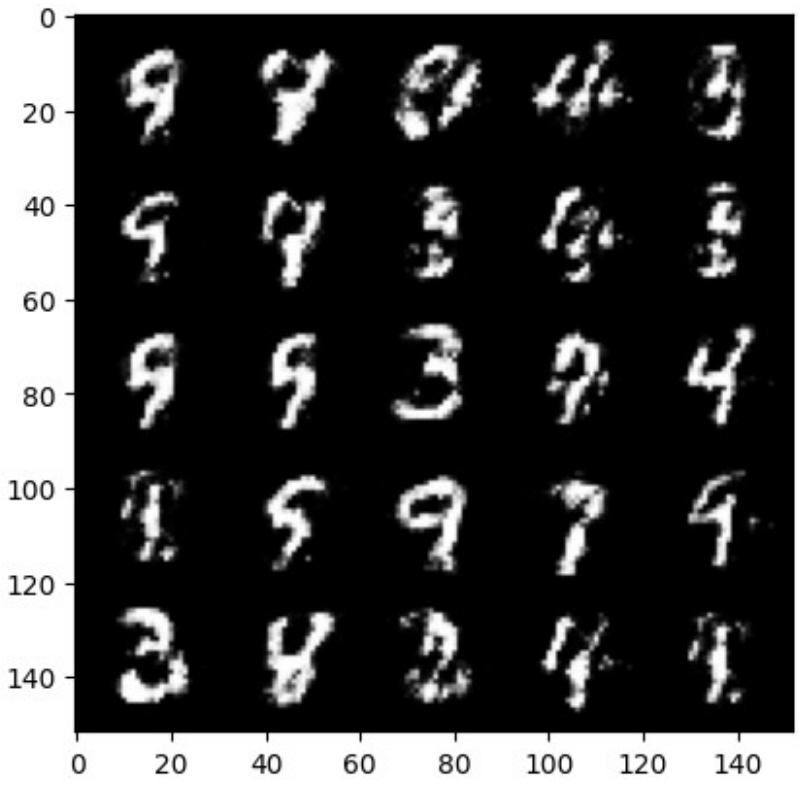
```
Epoch 87, step 41000: Generator loss: 2.1950925602912914,
discriminator loss: 0.28677351999282824
```

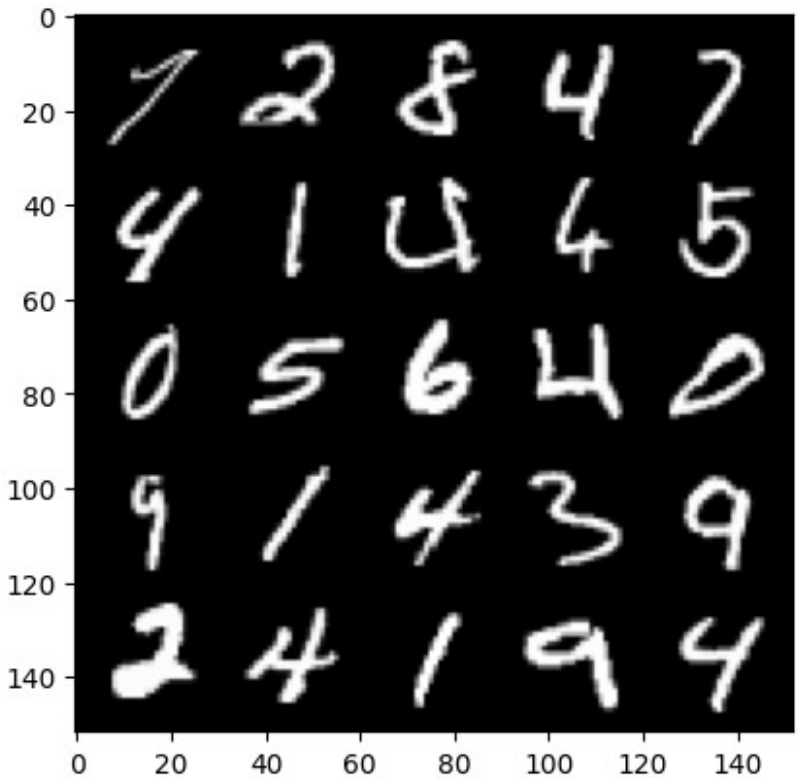

0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140


0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

{"model_id":"fc1226ac616a482ba854d7255a9f3950","version_major":2,"version_minor":0}

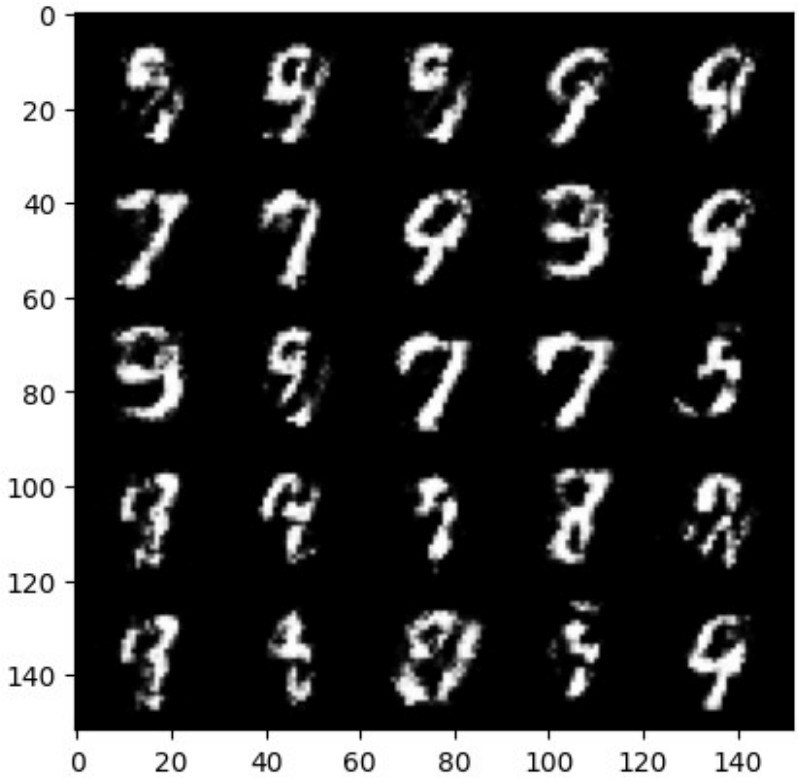
```
Epoch 88, step 41500: Generator loss: 2.0015112655162817,
discriminator loss: 0.31834088853001646
```

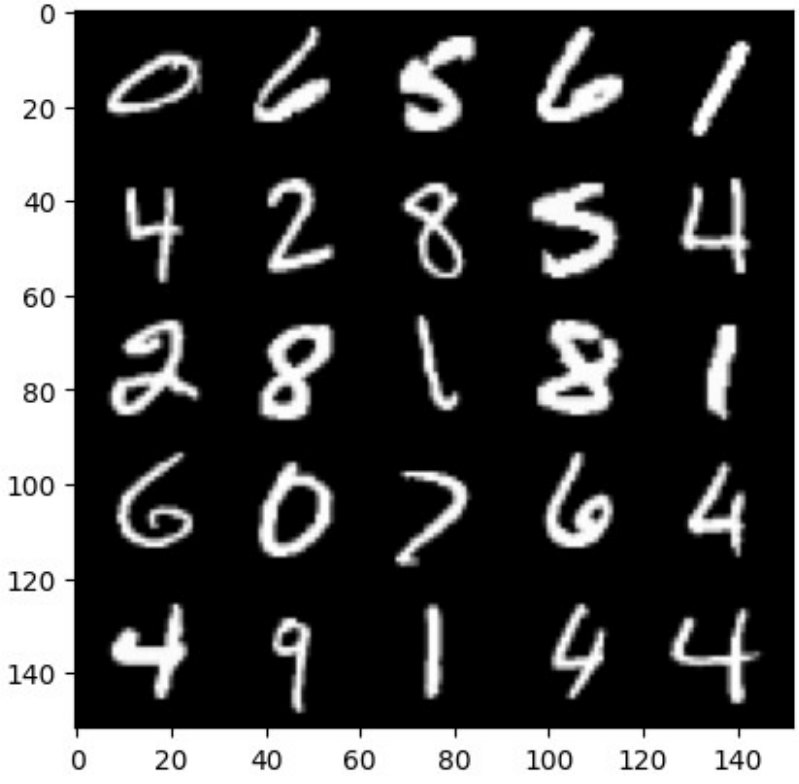




{"model_id":"a6fd76c87427460abf87df14ca606285","version_major":2,"version_minor":0}

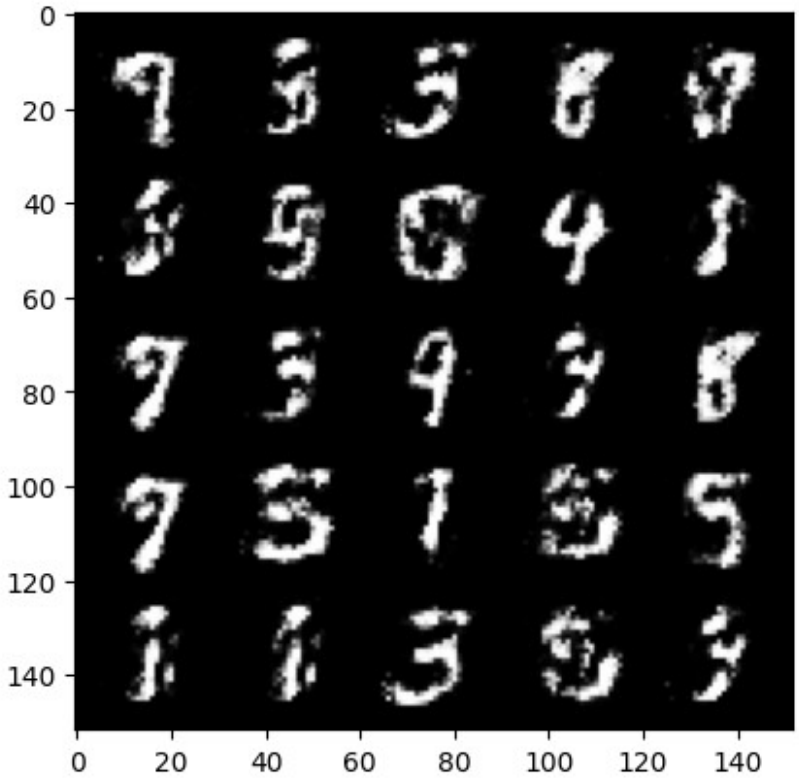
```
Epoch 89, step 42000: Generator loss: 2.024844153642654, discriminator loss: 0.3087469011247157
```

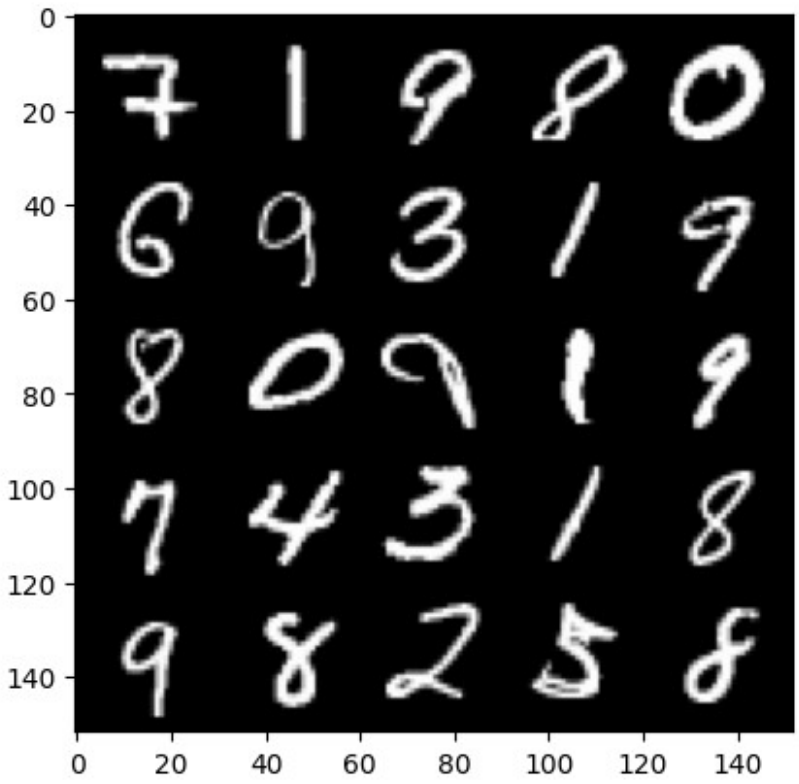

0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140


0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

{"model_id":"af89e0cb7f464acfaa8815df890e262e","version_major":2,"version_minor":0}

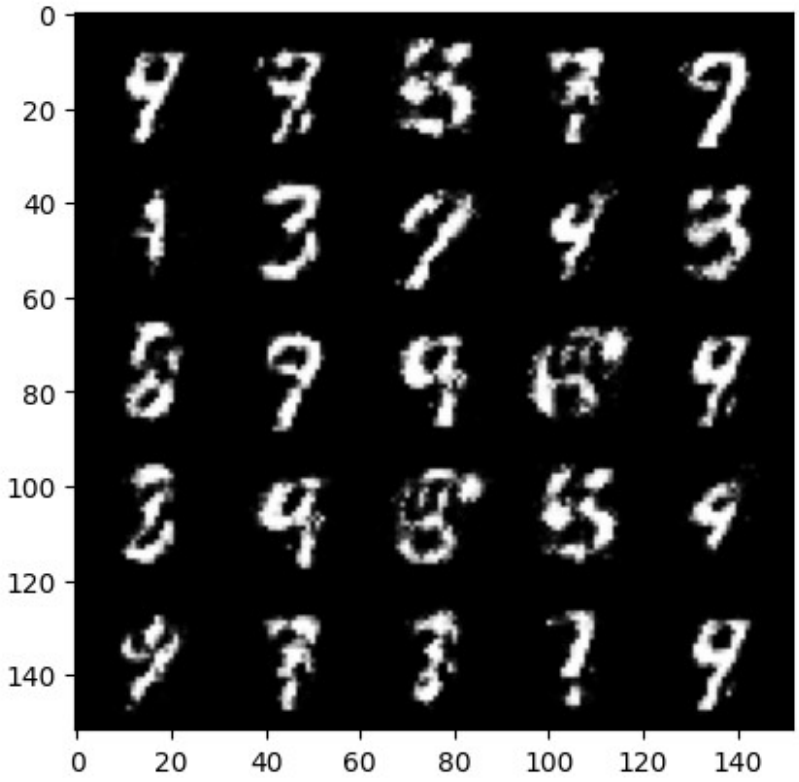
```
Epoch 90, step 42500: Generator loss: 1.9826328754425053,
discriminator loss: 0.317435572385788
```

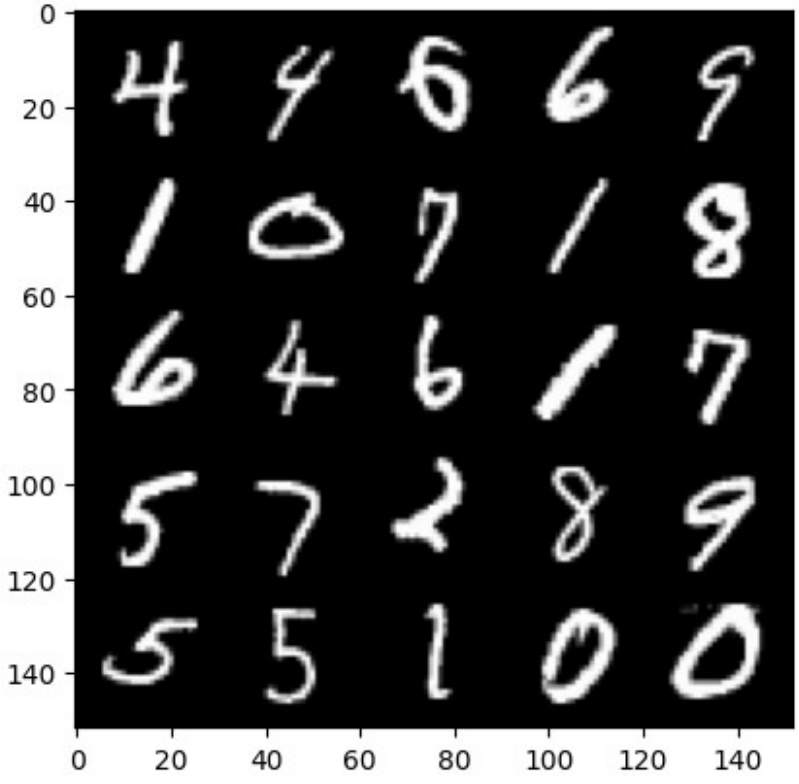




{"model_id":"ec22737b20b54fe49e7ec4b8b01cda05","version_major":2,"version_minor":0}

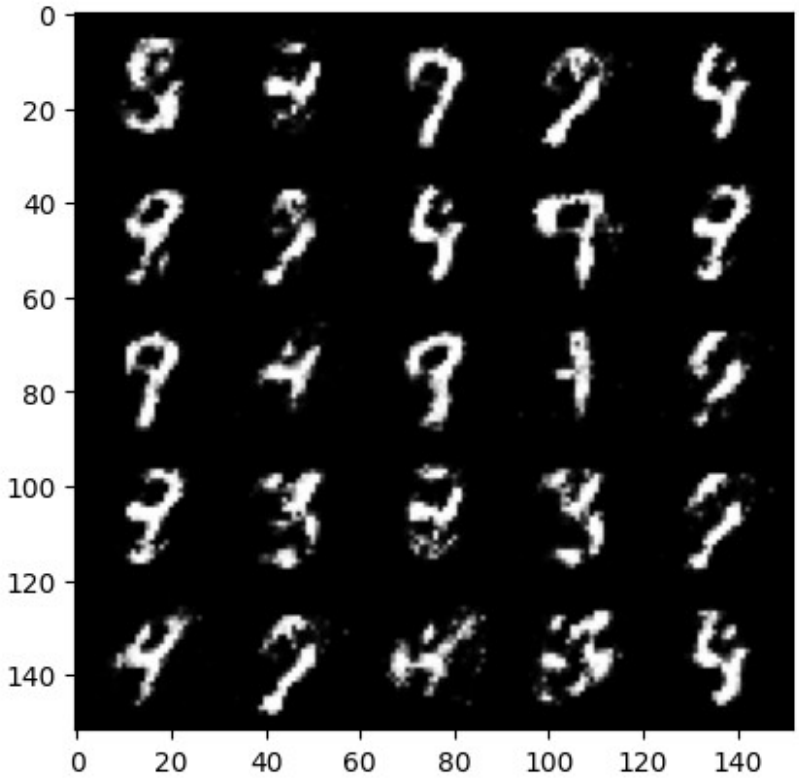
```
Epoch 91, step 43000: Generator loss: 2.031661021232608, discriminator loss: 0.3020039656460284
```

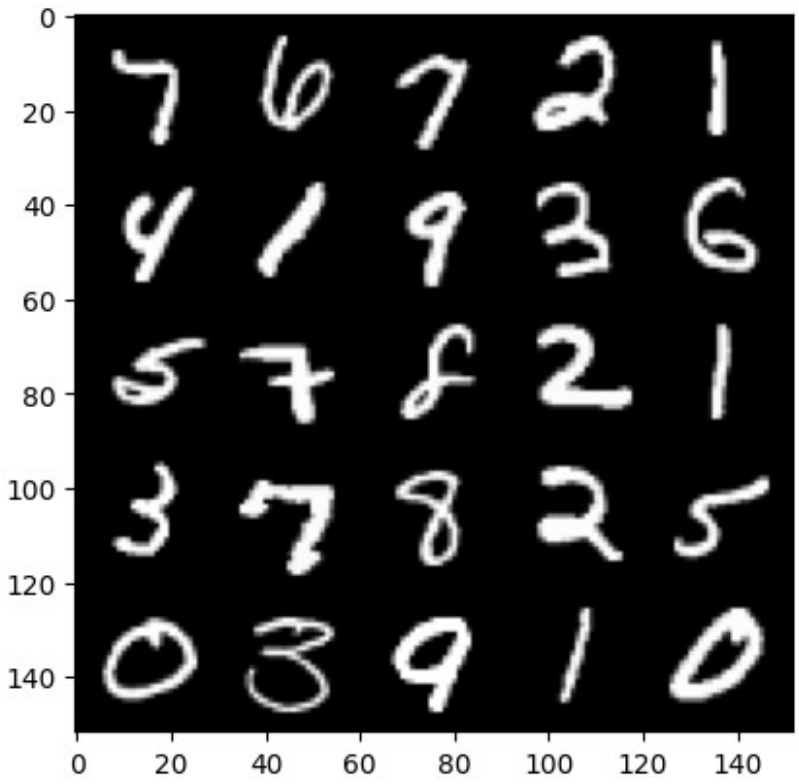

0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140


0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

{"model_id":"01b0970239a544b795085df4bd61c669","version_major":2,"version_minor":0}

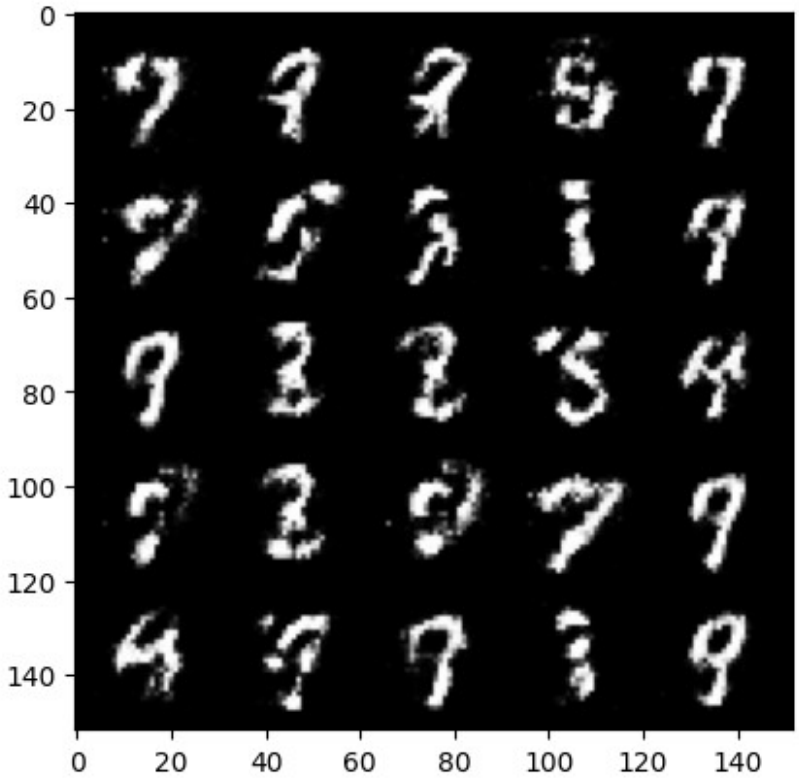
```
Epoch 92, step 43500: Generator loss: 2.0427751924991604,
discriminator loss: 0.29737438735365884
```

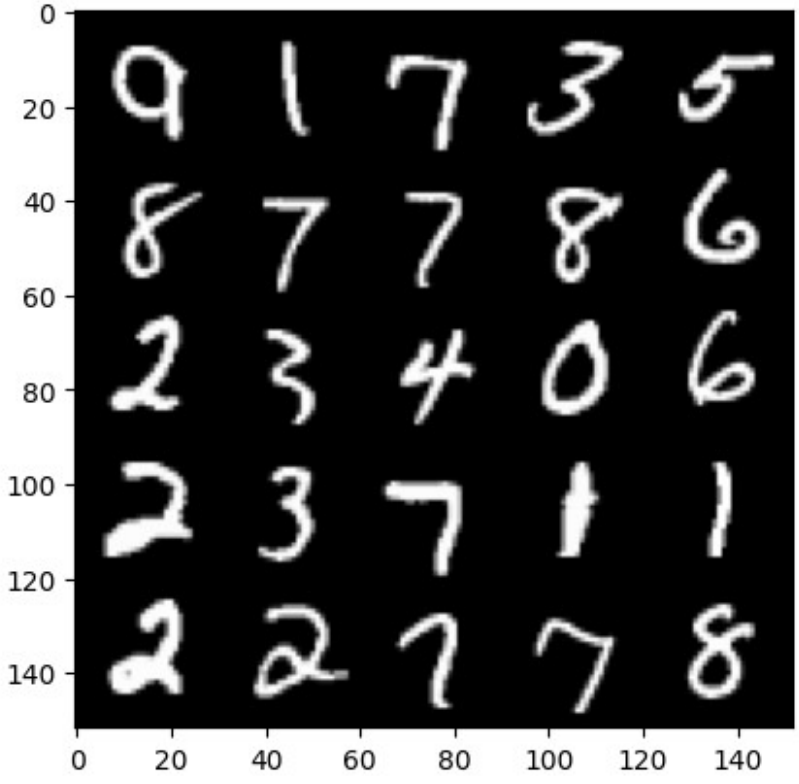




{"model_id":"ad8ea3b999954b93acbc73fee16b48f7","version_major":2,"version_minor":0}

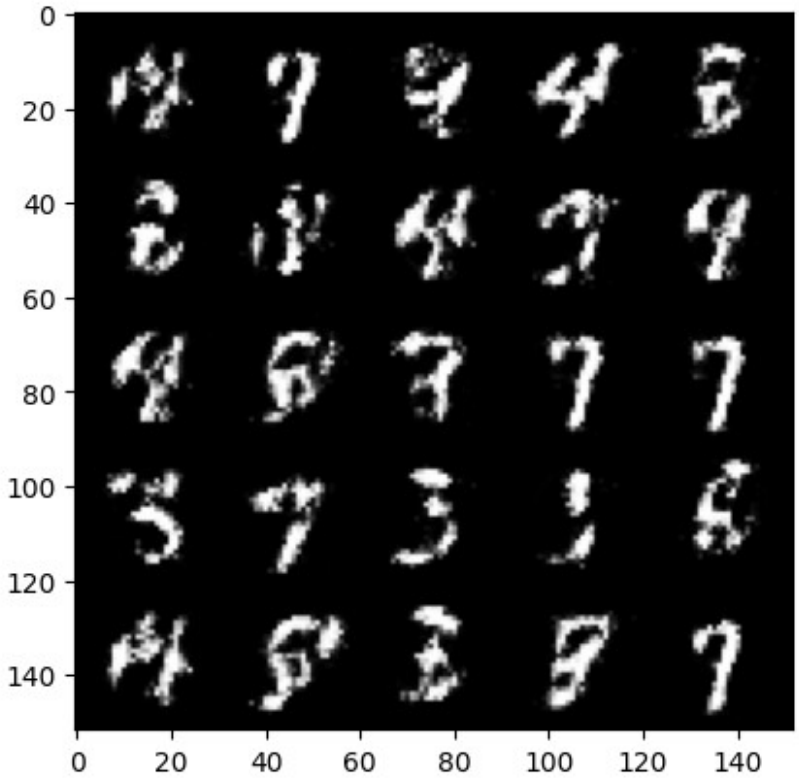
```
Epoch 93, step 44000: Generator loss: 2.042820212602614, discriminator loss: 0.31296597802639003
```

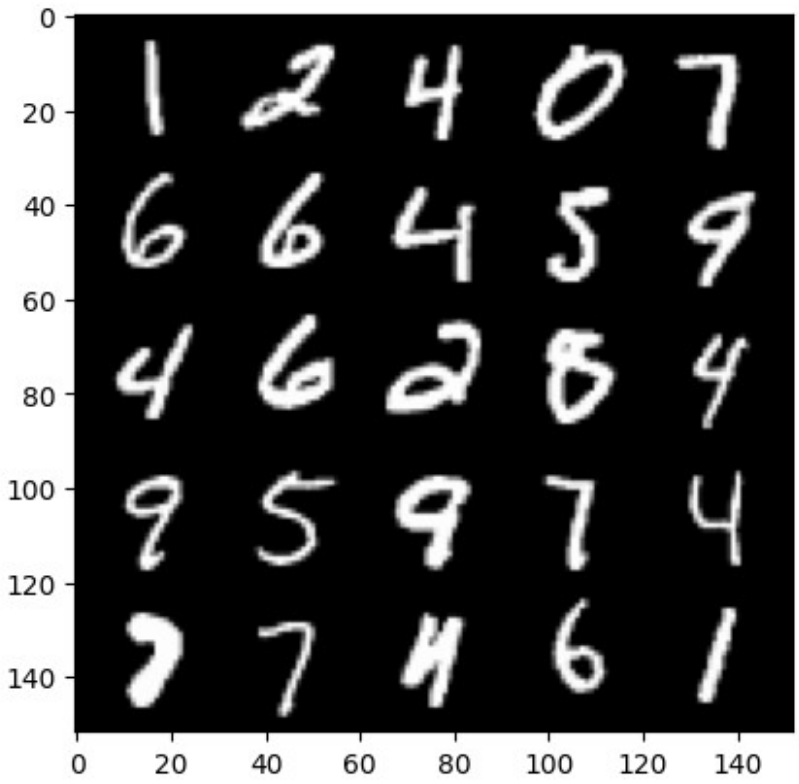

0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140


0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

{"model_id":"952ec7761cf14a04b097fb7bc5c3ab32","version_major":2,"version_minor":0}

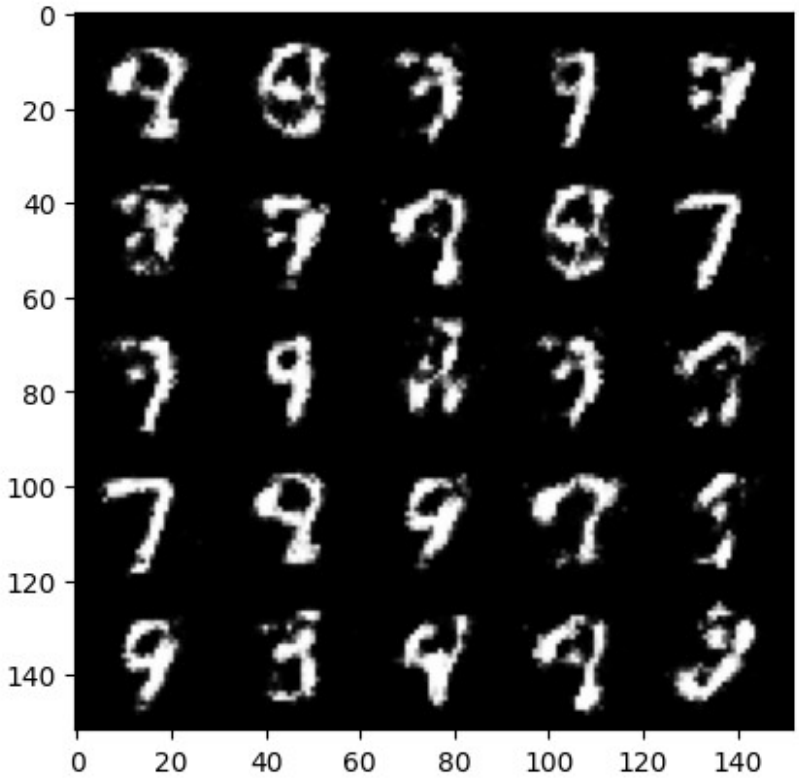
```
Epoch 94, step 44500: Generator loss: 1.9399408123493191,
discriminator loss: 0.31346294179558754
```

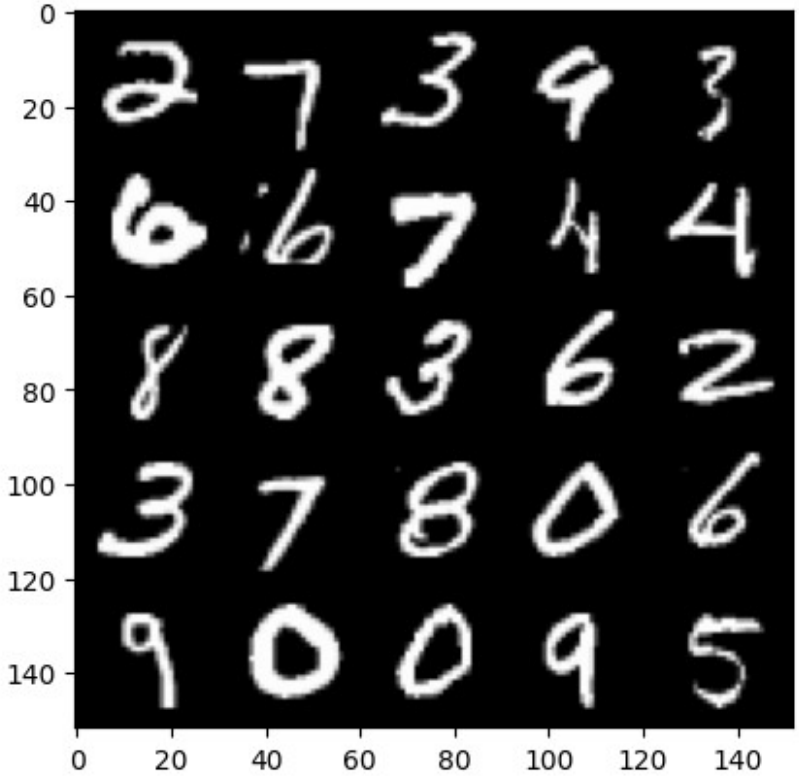




{"model_id":"d1f6dbe8362a4f28b2b27b7975a3a819","version_major":2,"version_minor":0}

```
Epoch 95, step 45000: Generator loss: 2.0677083611488327,
discriminator loss: 0.2985959801375865
```

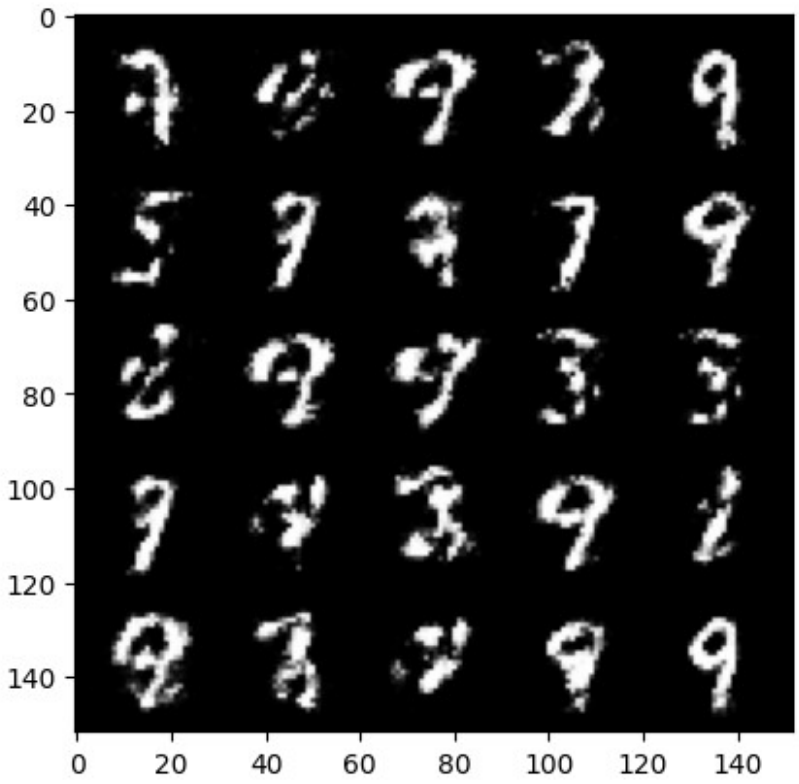

0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

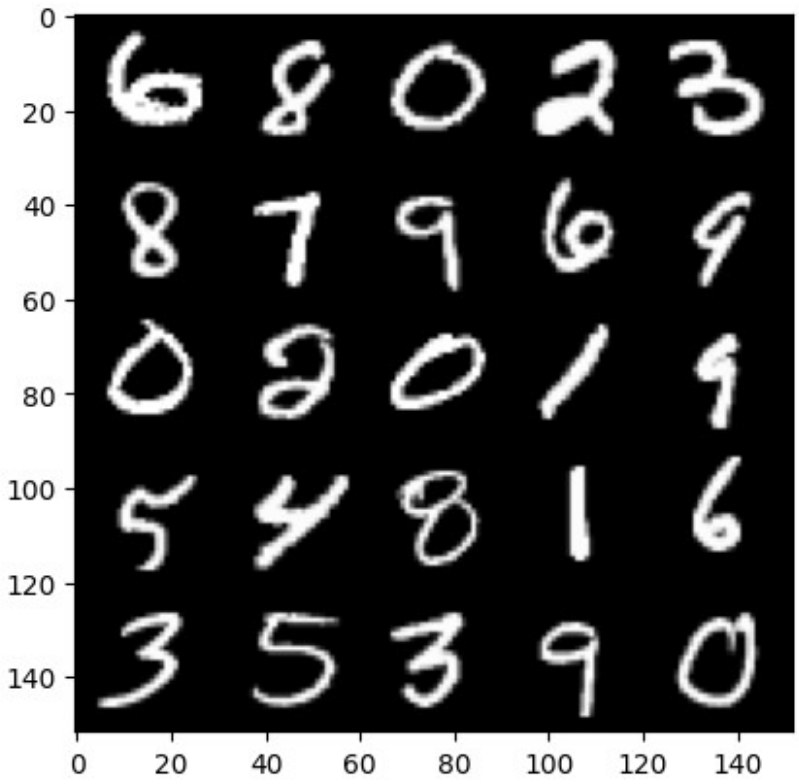

0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

{"model_id":"ec405ab33f25437dbeb08e03de9fcd76","version_major":2,"version_minor":0}

{"model_id":"e75244bf35ec4f00a3b47b01421916e0","version_major":2,"version_minor":0}

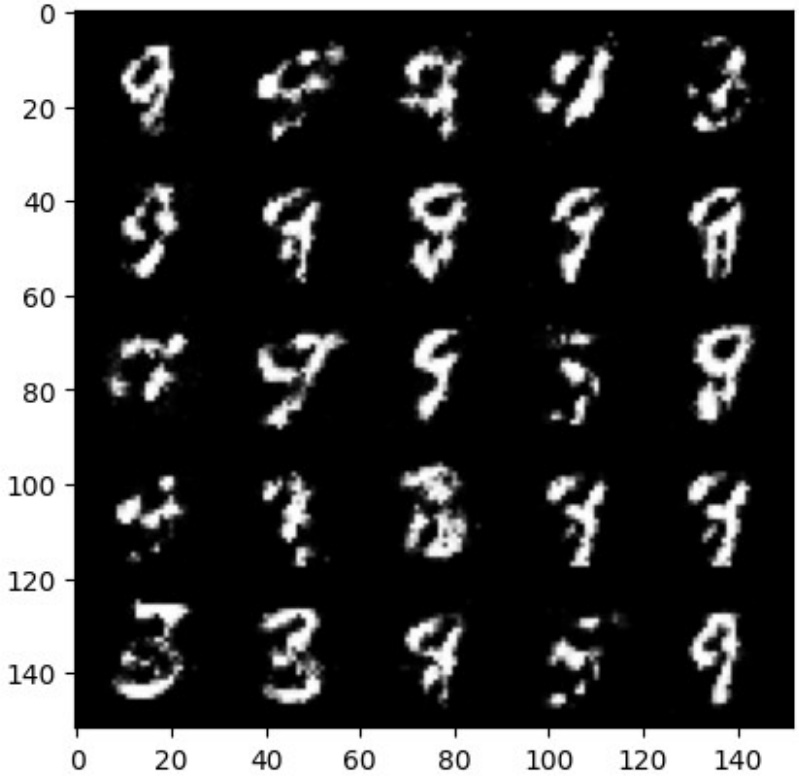
```
Epoch 97, step 45500: Generator loss: 2.050627742767333, discriminator loss: 0.287604117900133
```

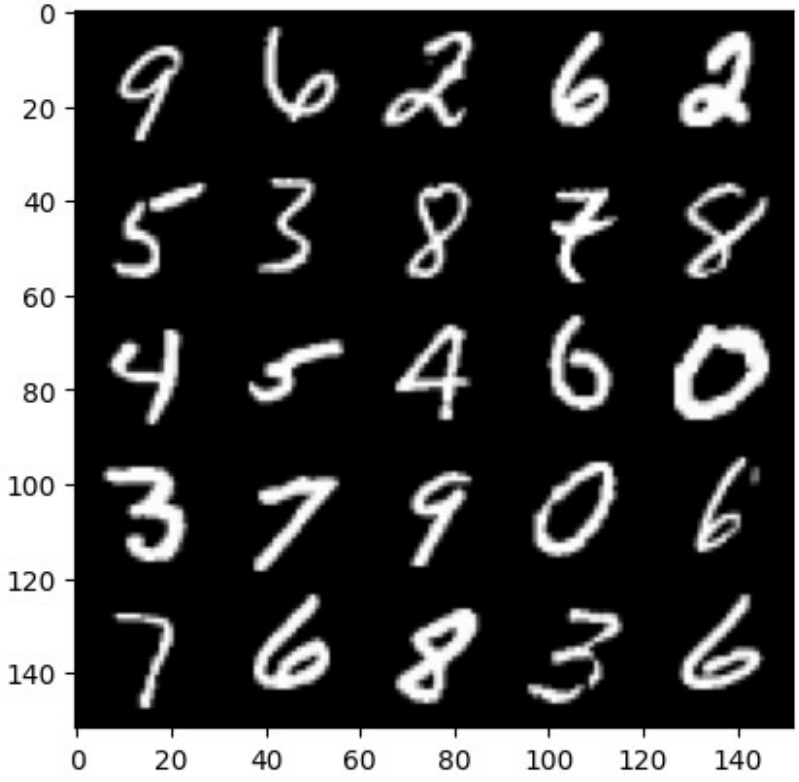




{"model_id":"c6063a734d2b42dfaeb2e73ddeca670b","version_major":2,"version_minor":0}

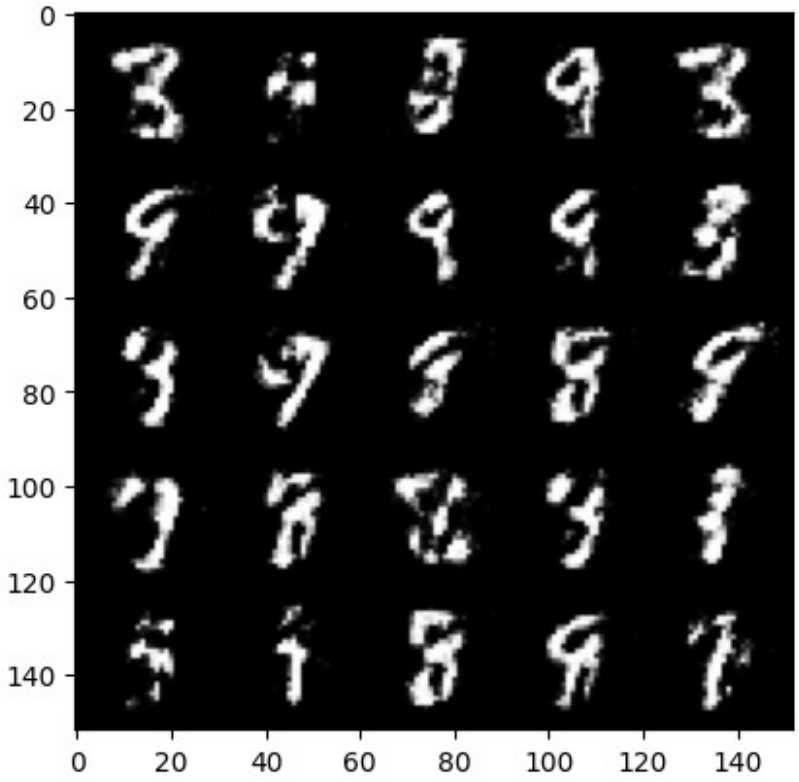
```
Epoch 98, step 46000: Generator loss: 2.049791711568831, discriminator loss: 0.2844227881431584
```

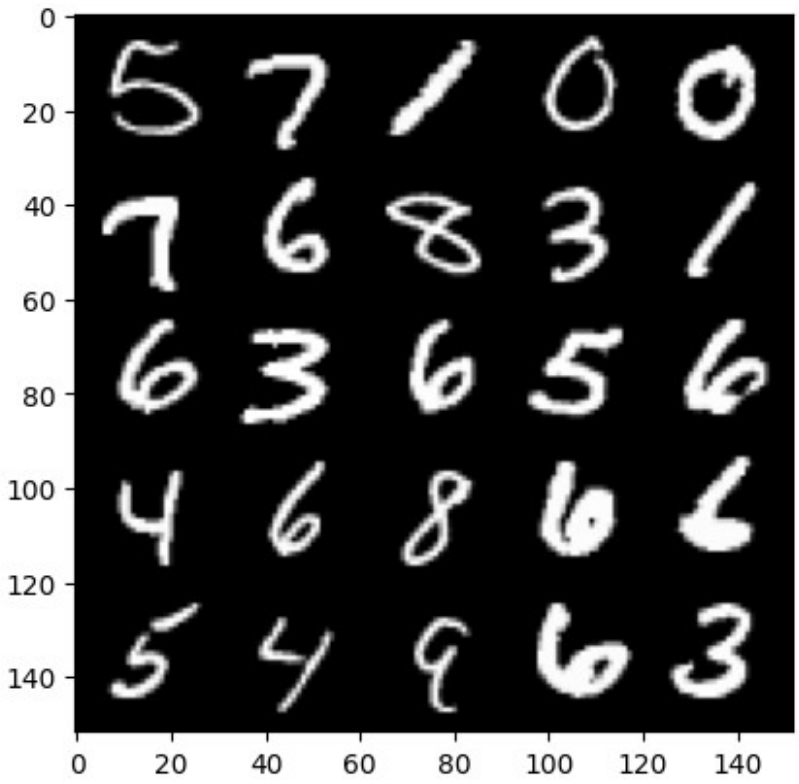

0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140


0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

{"model_id":"55e9bc35515f487c8e328e7accb6247f","version_major":2,"version_minor":0}

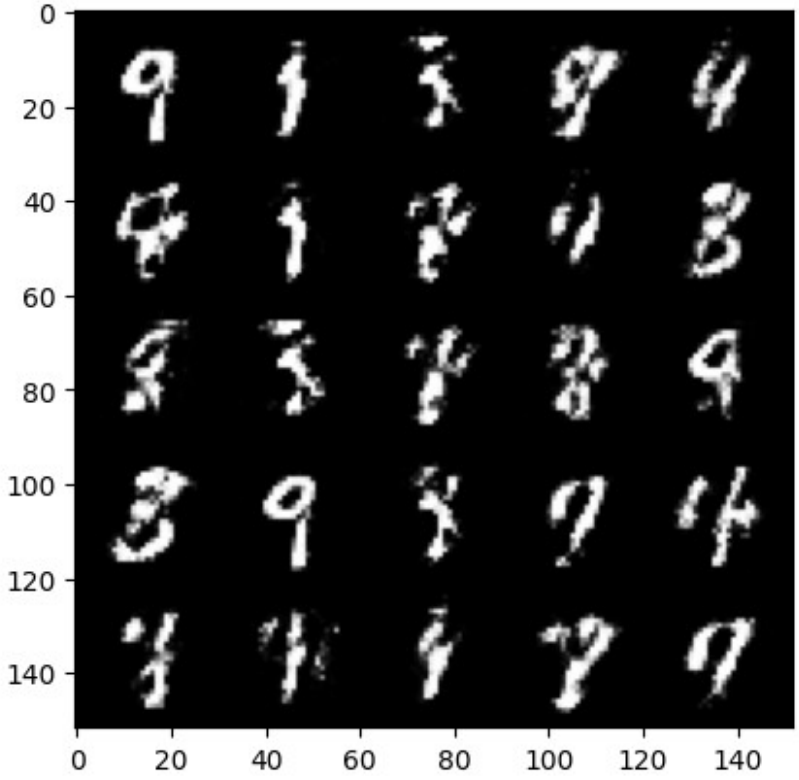
```
Epoch 99, step 46500: Generator loss: 2.0088699650764443,
discriminator loss: 0.31008650705218294
```

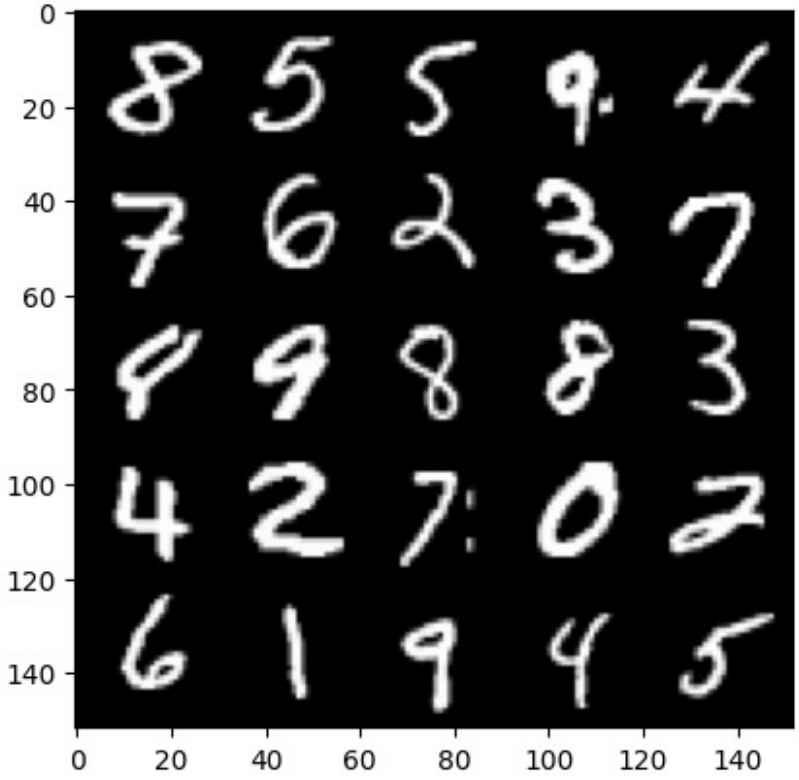




{"model_id":"0cc1ca7b443347e1b277e820597c98ac","version_major":2,"version_minor":0}

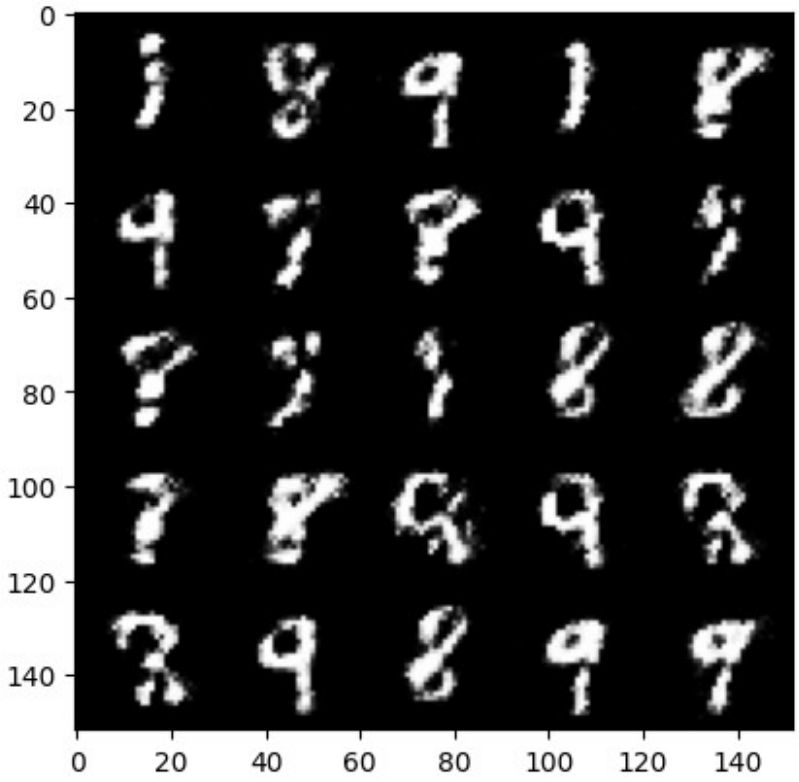
```
Epoch 100, step 47000: Generator loss: 1.9460034976005556,
discriminator loss: 0.33631784853339197
```

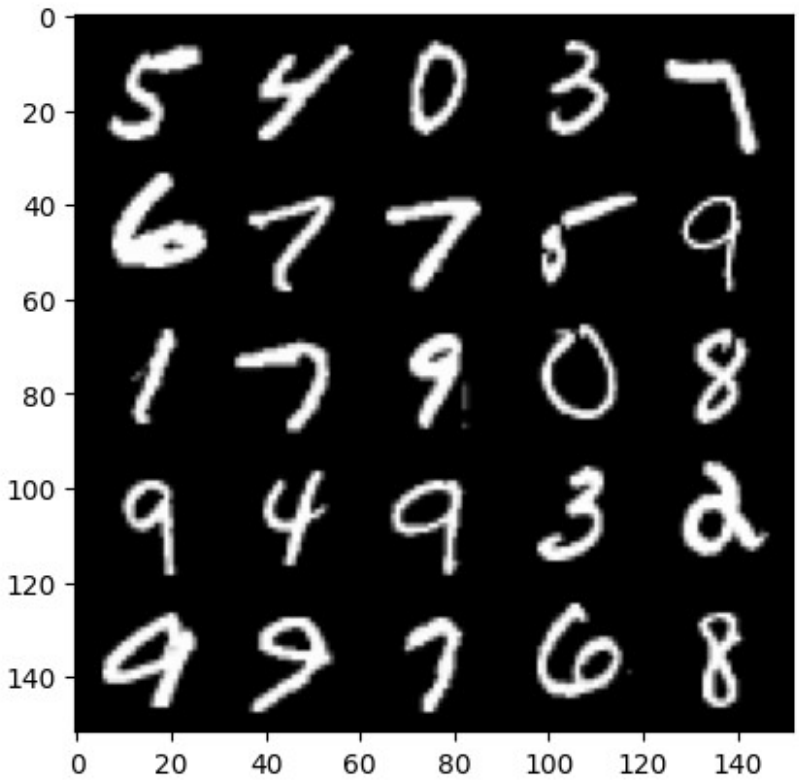

0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140


0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

{"model_id":"23d3f0aa5c974b8c9f881e5b2f200666","version_major":2,"version_minor":0}

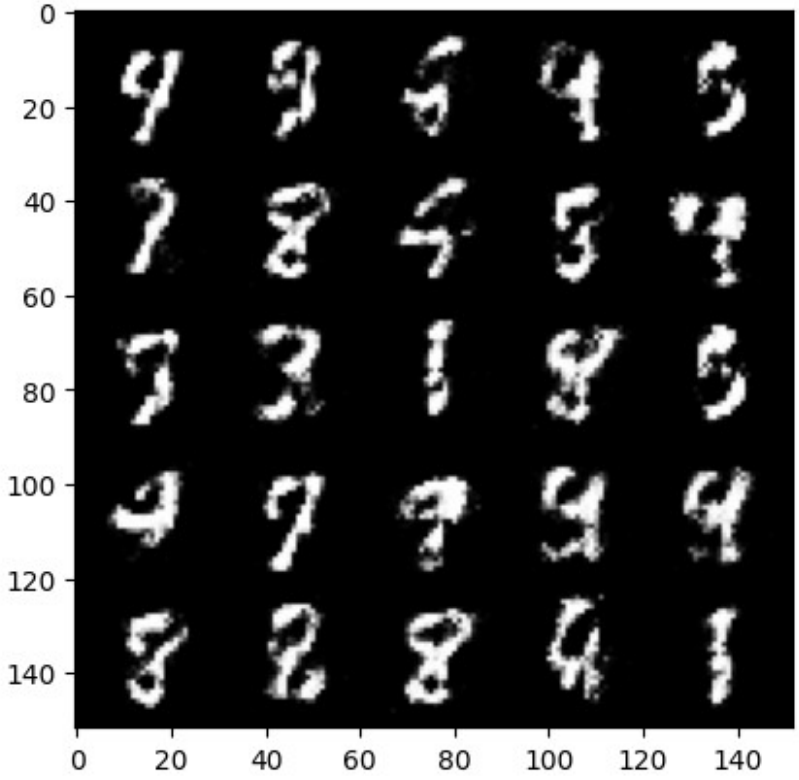
```
Epoch 101, step 47500: Generator loss: 1.8999926116466541,
discriminator loss: 0.3249207303225993
```

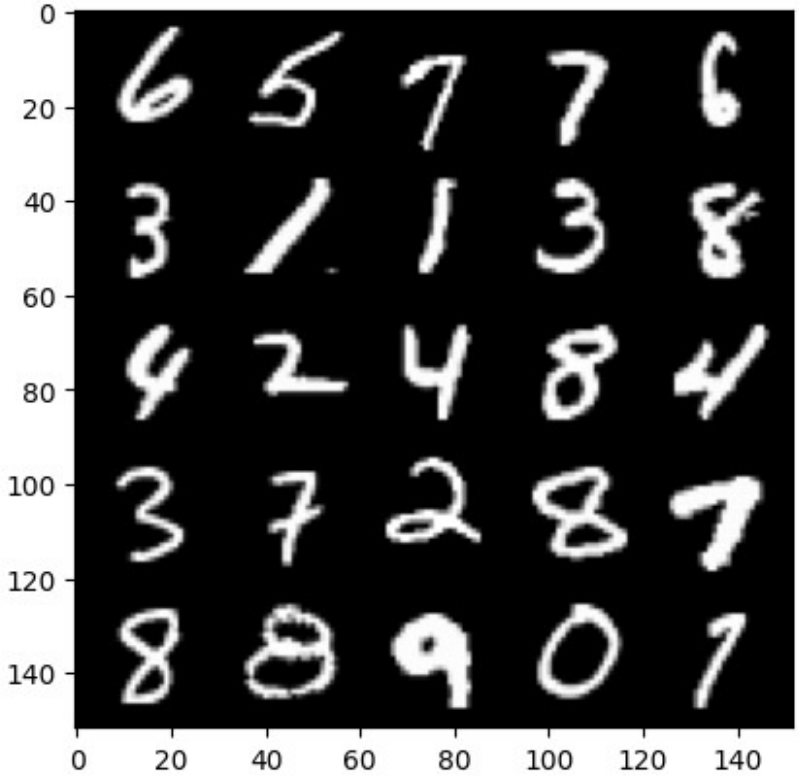




{"model_id":"18cf1fb7a53b48e0b338aa6c7b4903dd","version_major":2,"version_minor":0}

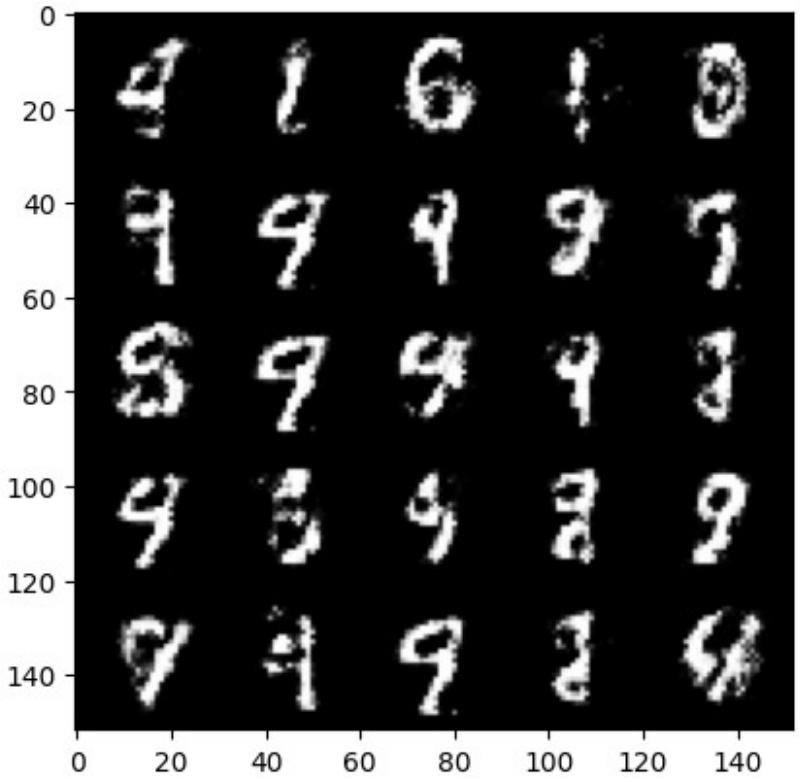
```
Epoch 102, step 48000: Generator loss: 2.000184968709947,
discriminator loss: 0.30743020722270037
```

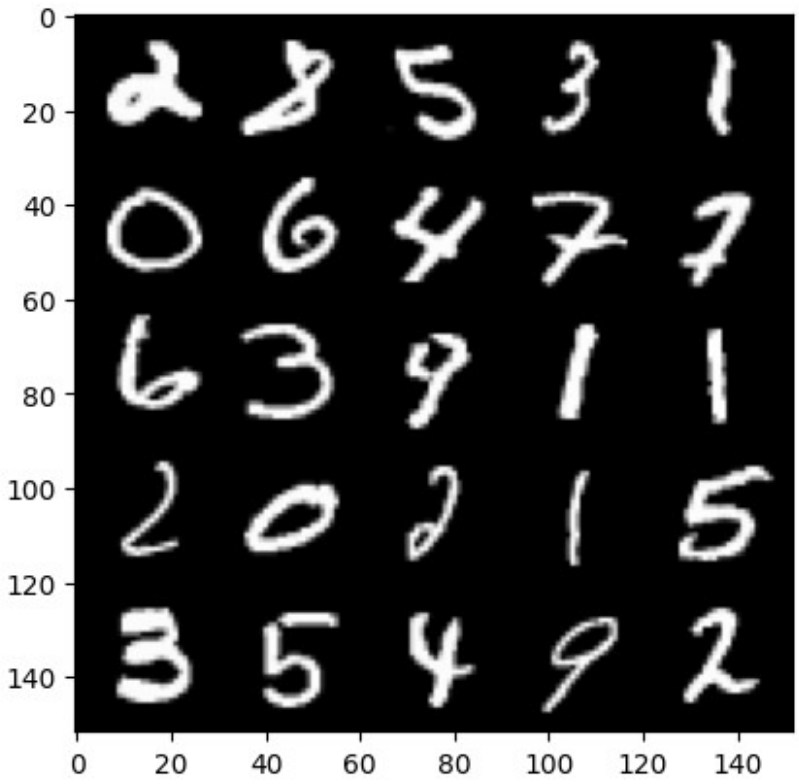

0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140


0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

{"model_id":"206948cb01854013a70815ba36429d8d","version_major":2,"version_minor":0}

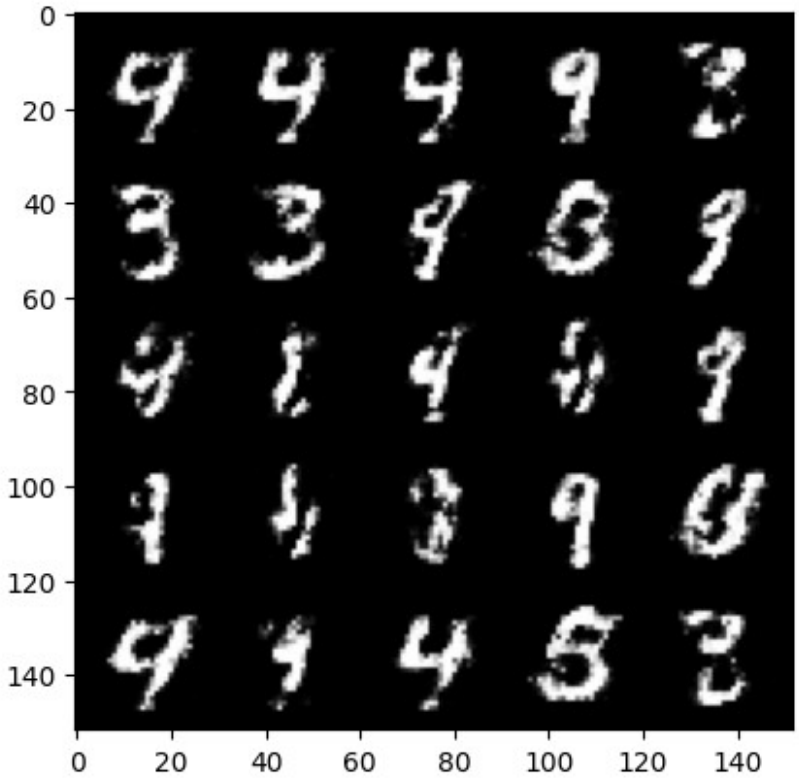
```
Epoch 103, step 48500: Generator loss: 1.848583833456039,
discriminator loss: 0.347163343727589
```

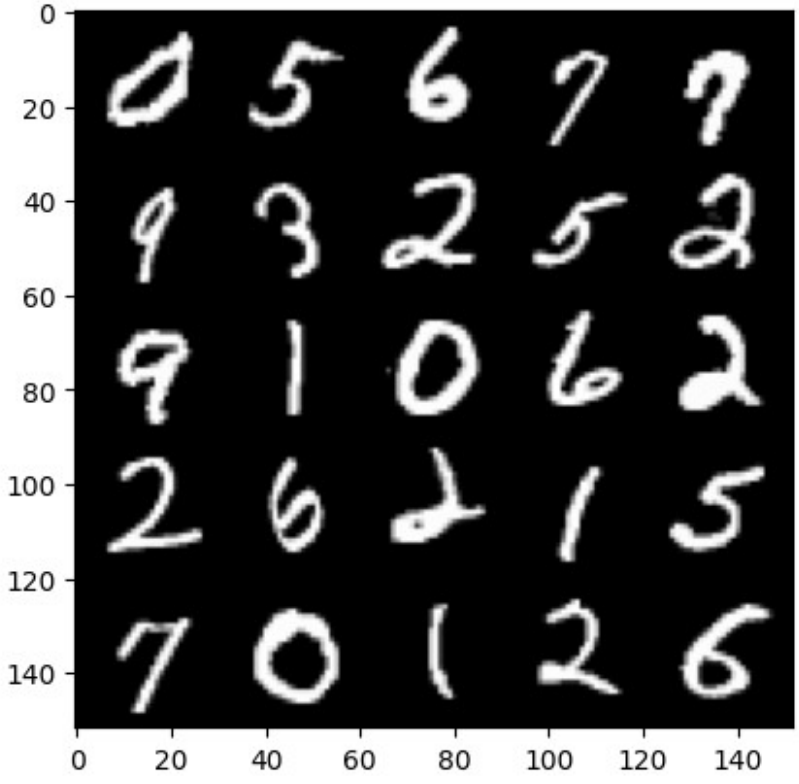




{"model_id":"df977322aca74368ac61406803ca7d84","version_major":2,"version_minor":0}

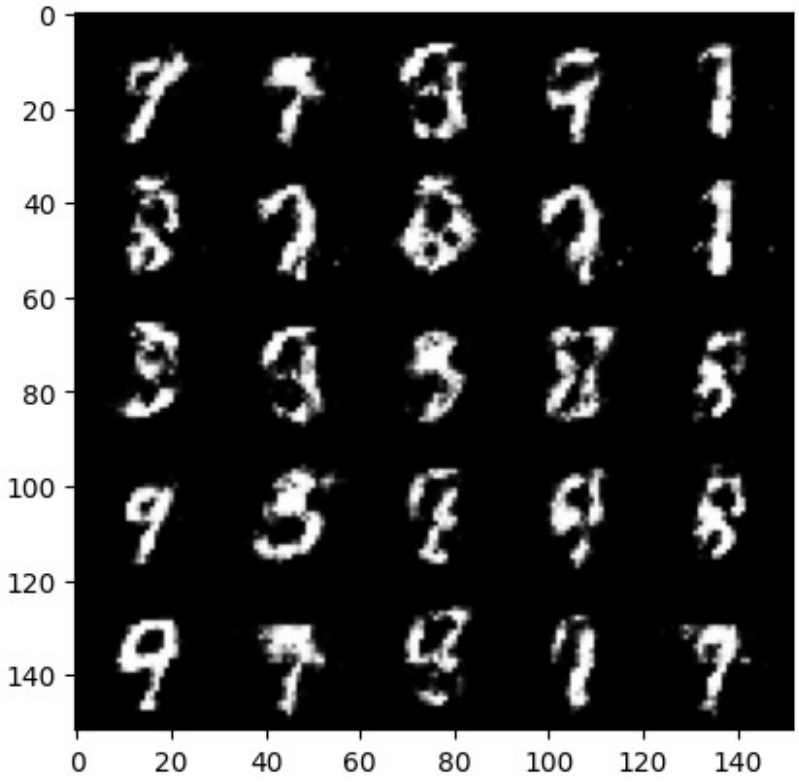
```
Epoch 104, step 49000: Generator loss: 1.8797879698276514,
discriminator loss: 0.32699353572726264
```

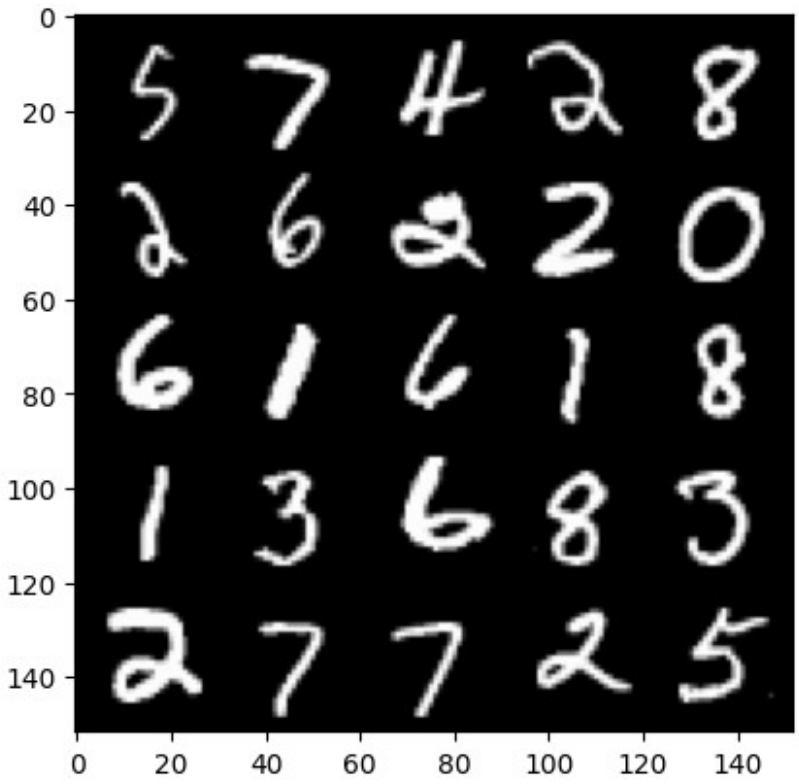

0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140


0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

{"model_id":"ec66951f93f84aa0b0a0a758e4628f67","version_major":2,"version_minor":0}

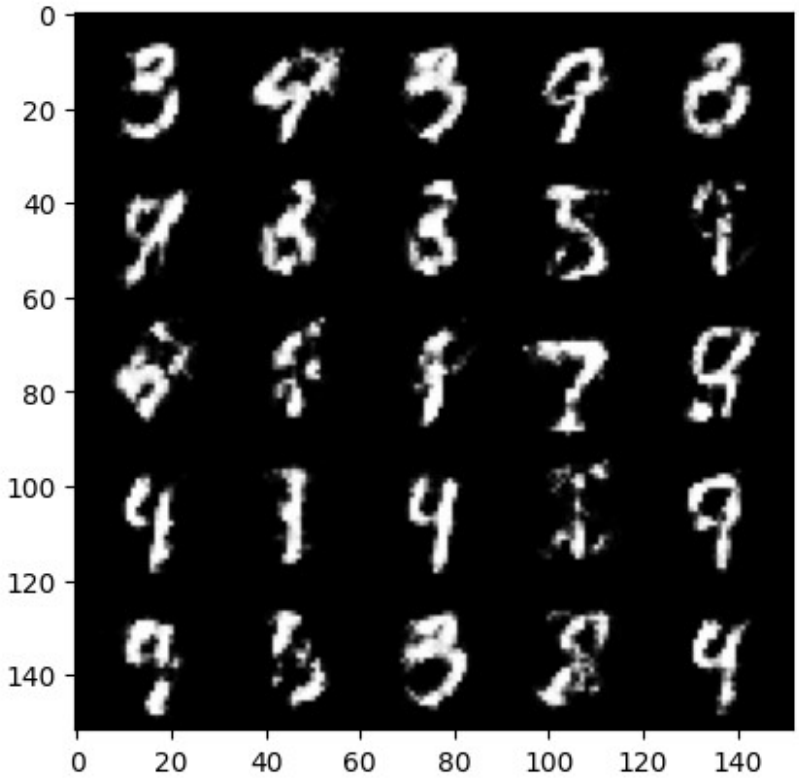
```
Epoch 105, step 49500: Generator loss: 1.8578396122455598,
discriminator loss: 0.3376135498583316
```

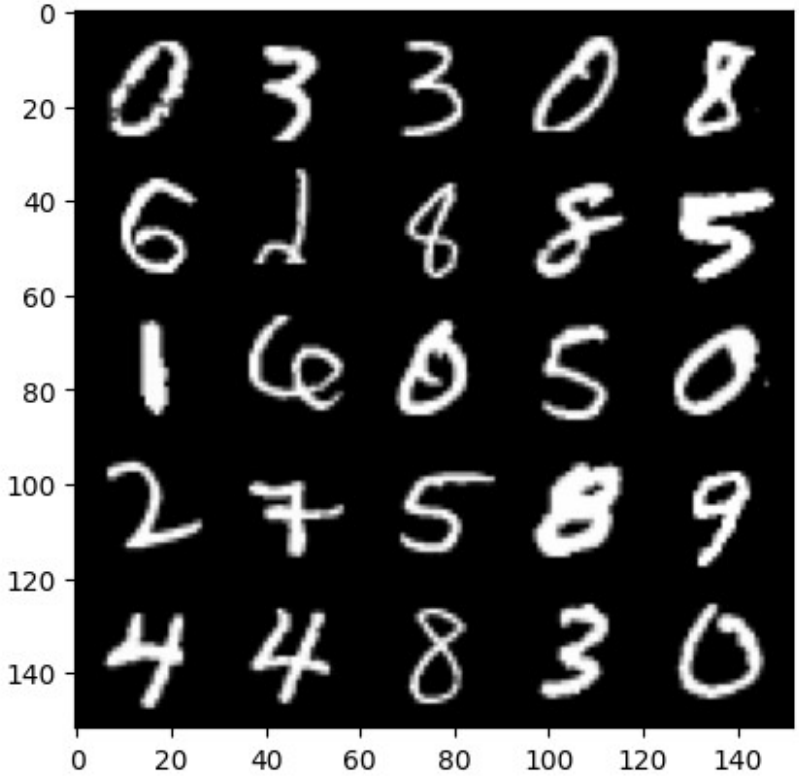




{"model_id":"2d9a3a376e924df798ecdb156cd7ca4e","version_major":2,"version_minor":0}

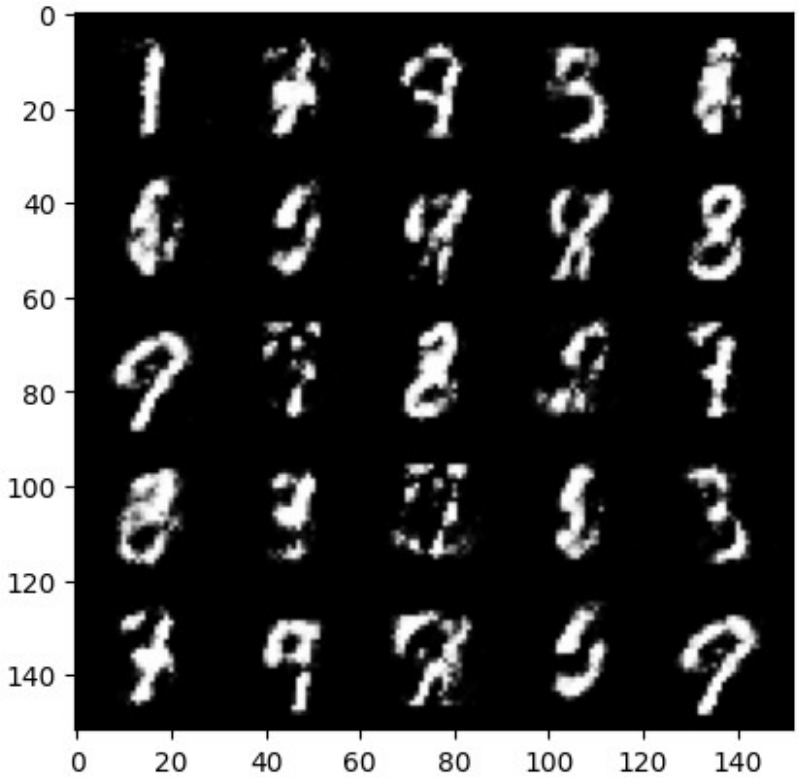
```
Epoch 106, step 50000: Generator loss: 1.9281547787189497,
discriminator loss: 0.32511053767800335
```

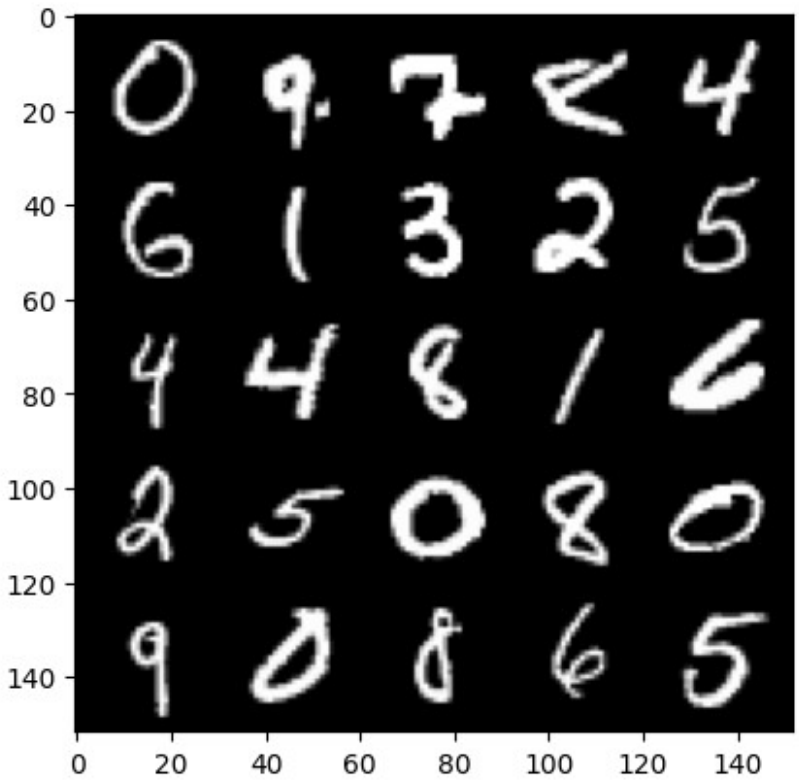

0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140


0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

{"model_id":"fd2fb7df3c844da49e5647b7d1d8c3b3","version_major":2,"version_minor":0}

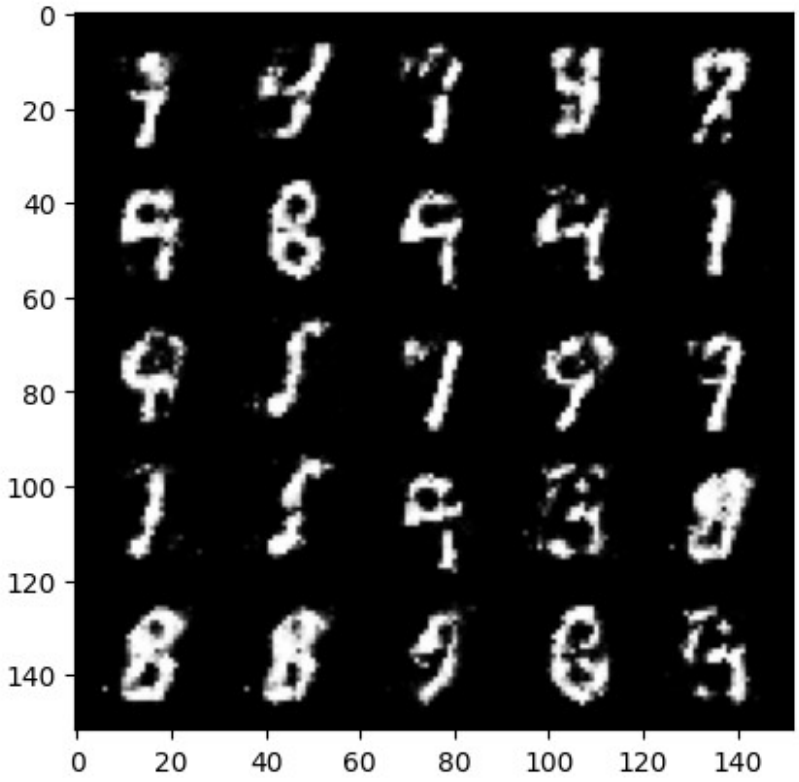
```
Epoch 107, step 50500: Generator loss: 1.7888150286674516,
discriminator loss: 0.35139302867650973
```

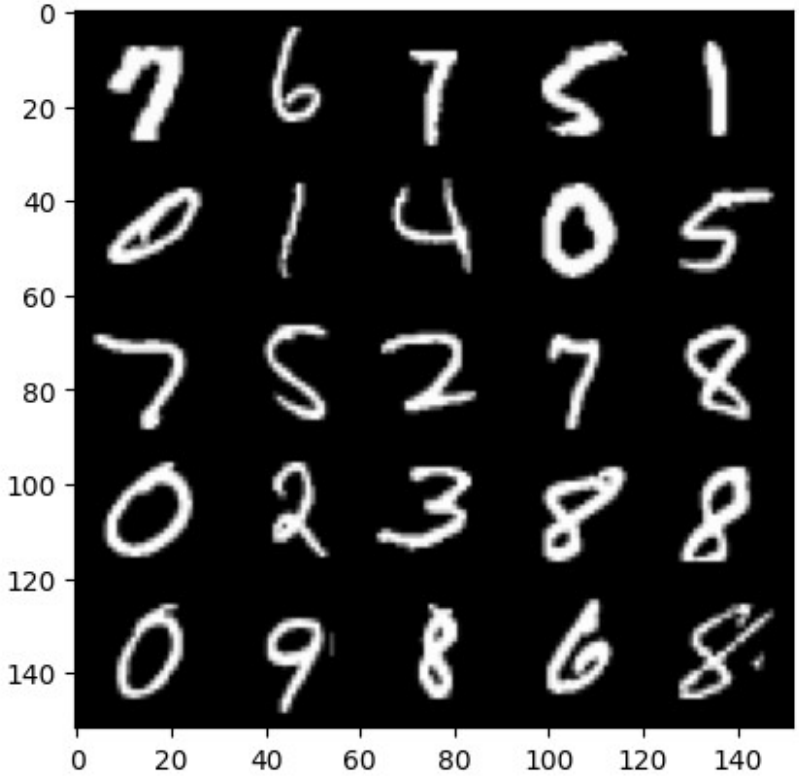




{"model_id":"3d490e64e83349069ce91c7895ba542e","version_major":2,"version_minor":0}

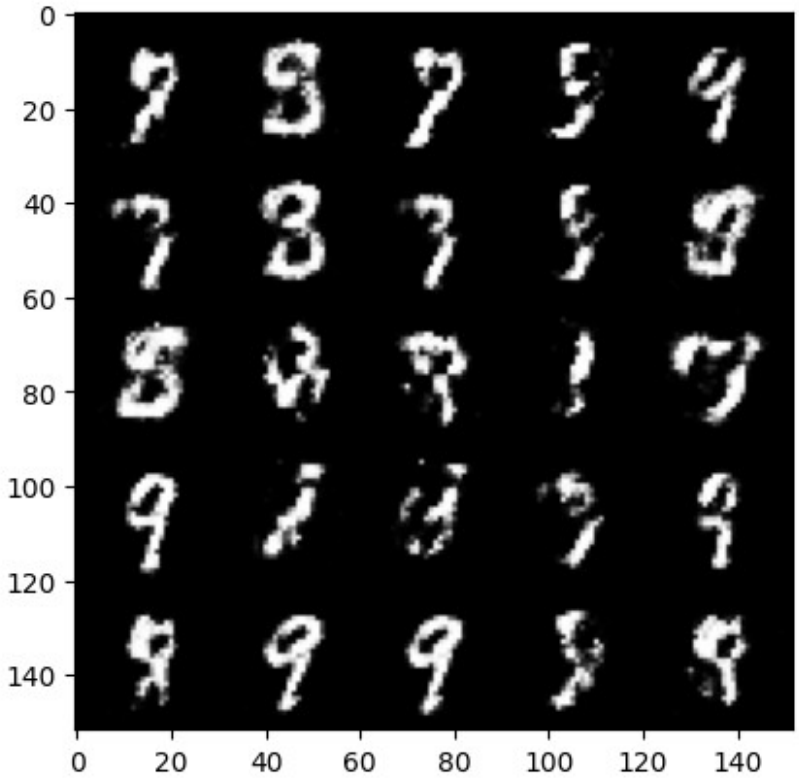
```
Epoch 108, step 51000: Generator loss: 1.733864026546481,
discriminator loss: 0.3531230059862139
```

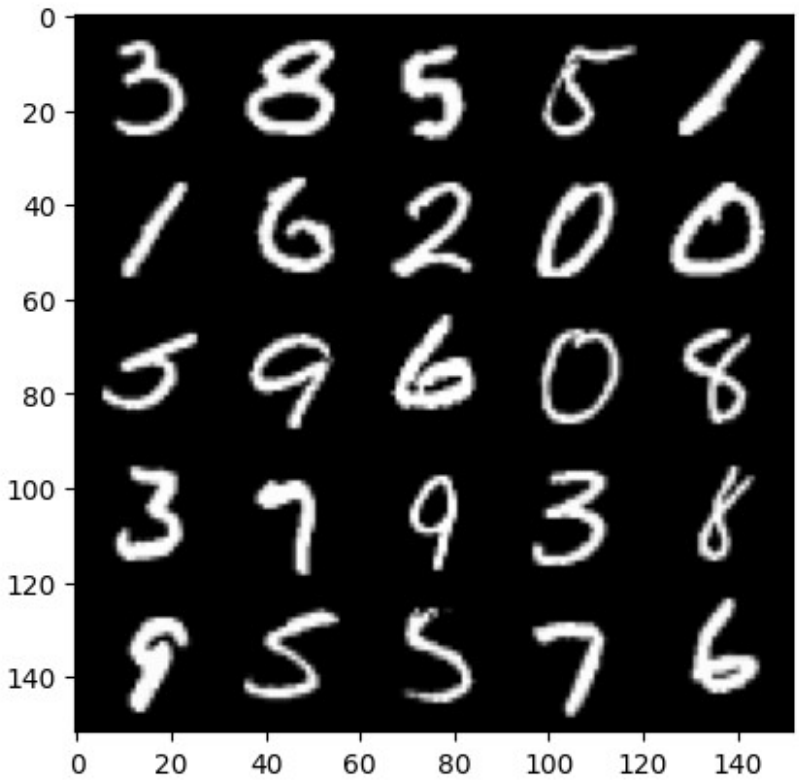

0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140


0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

{"model_id":"90005a73ce6348388d4ce30b0a2fd18e","version_major":2,"version_minor":0}

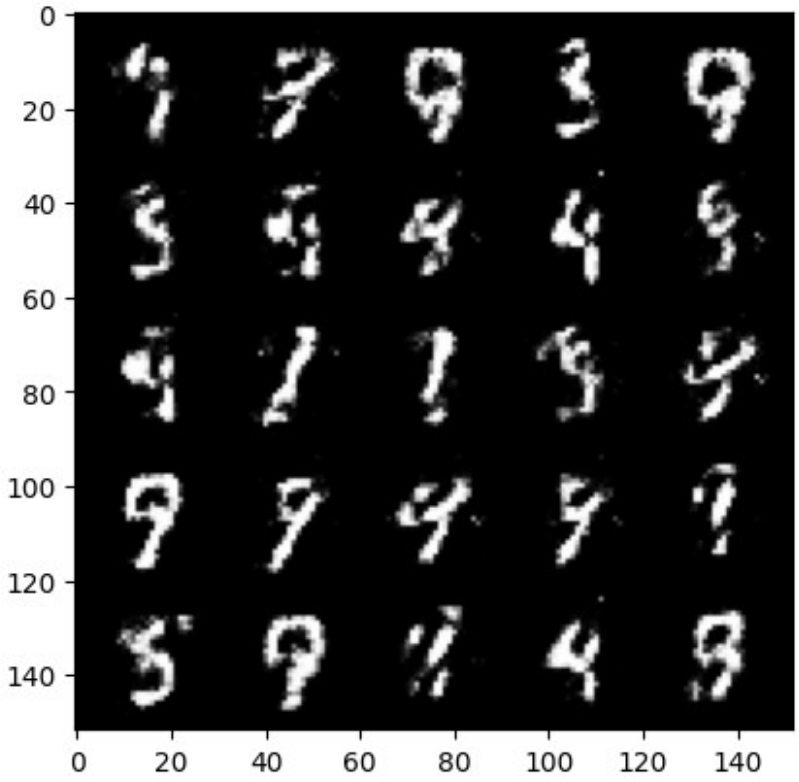
```
Epoch 109, step 51500: Generator loss: 1.7599264080524453,
discriminator loss: 0.3476374045908452
```

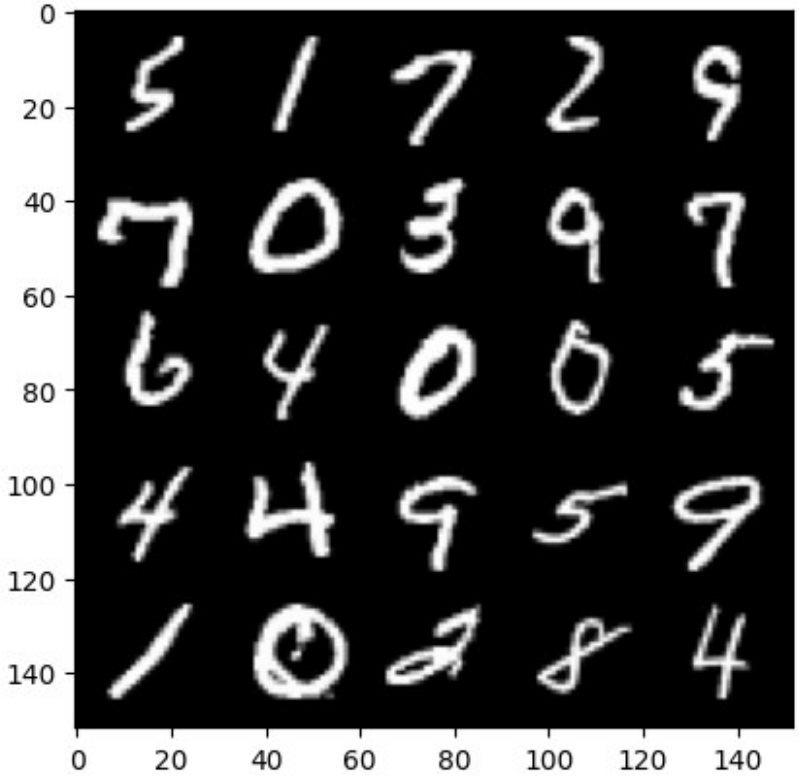




{"model_id":"21c9dc4536424a9e9efc477ec4e25683","version_major":2,"version_minor":0}

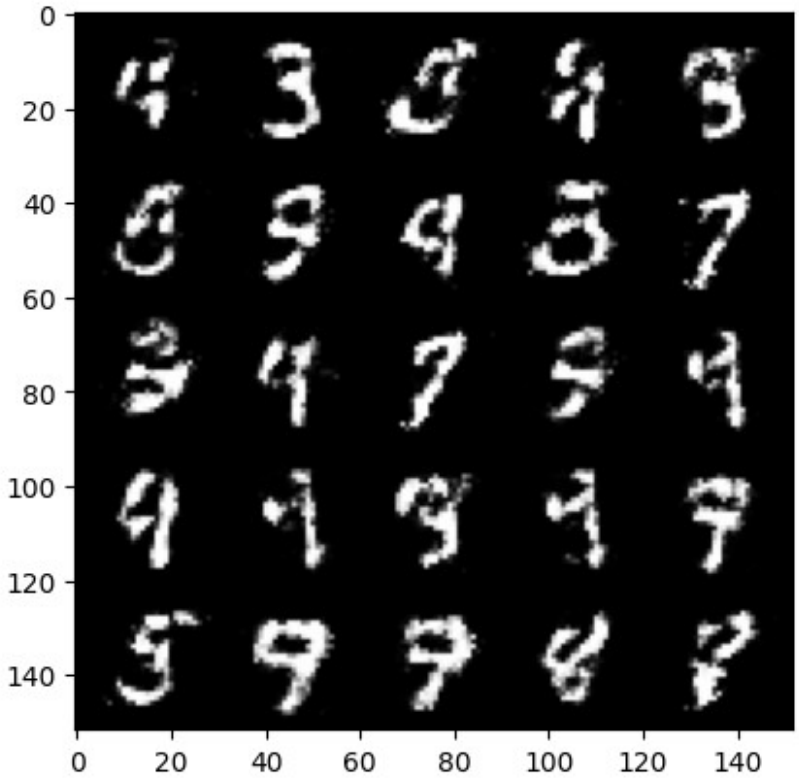
```
Epoch 110, step 52000: Generator loss: 1.7542405421733855,
discriminator loss: 0.35239261773228653
```

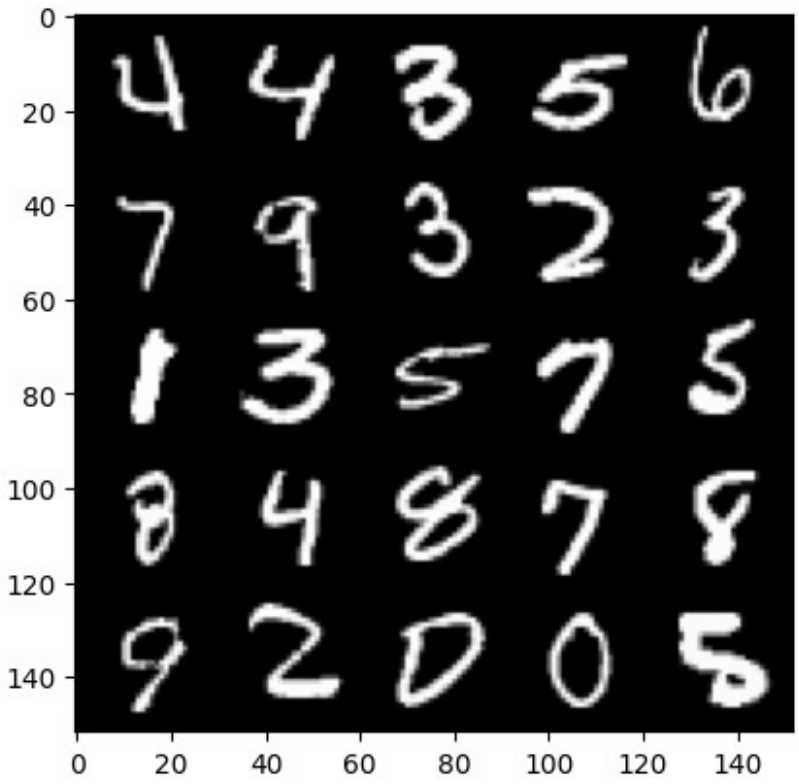

0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140


0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

{"model_id":"5aa6e75060704ababcf5b33226d1c1b7","version_major":2,"version_minor":0}

```
Epoch 111, step 52500: Generator loss: 1.7397026507854463,
discriminator loss: 0.36253120237588854
```

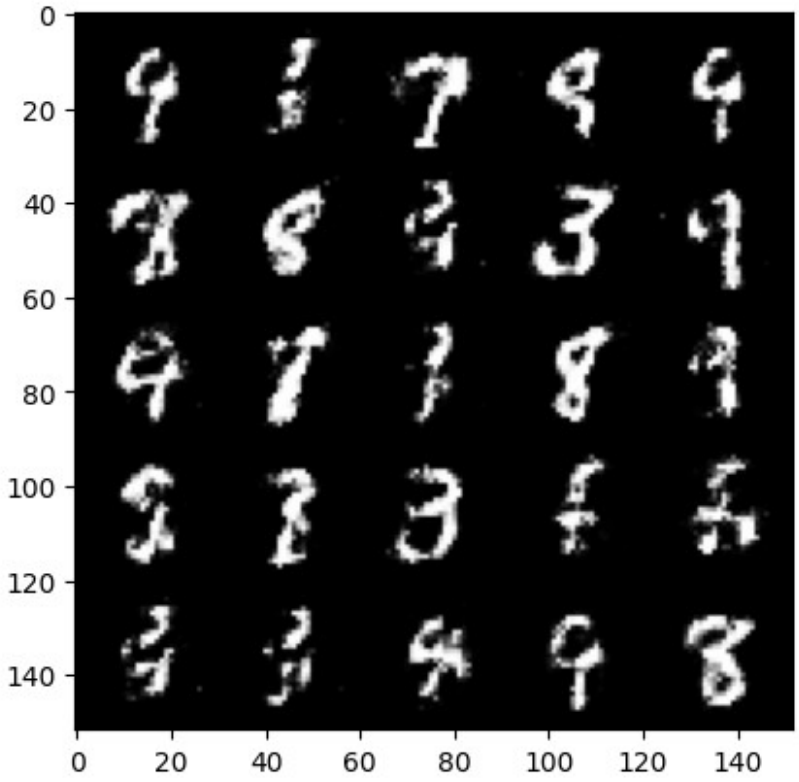


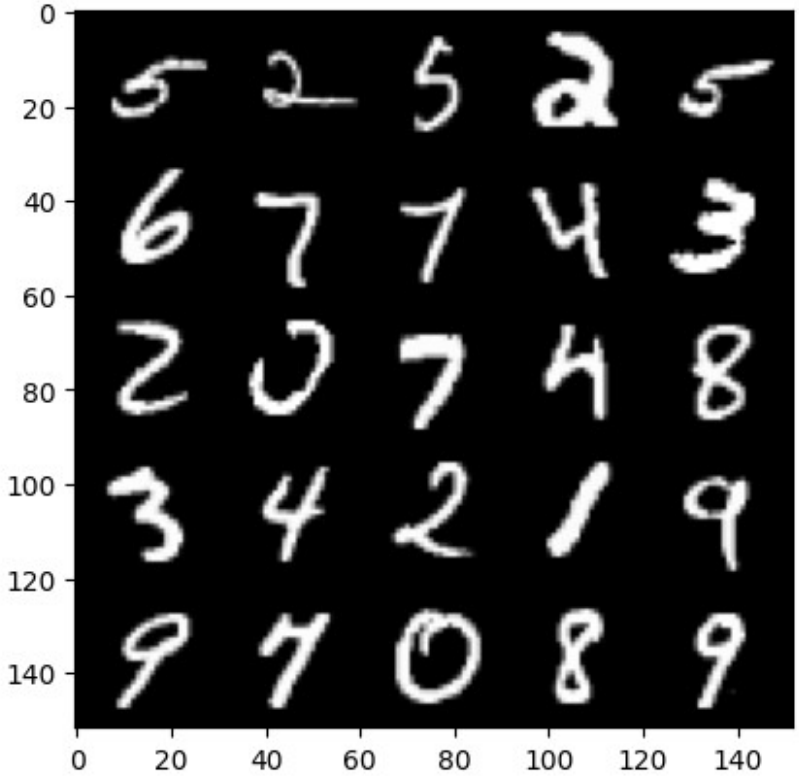


{"model_id":"8562eeff7fee456293c771aa5f82f142","version_major":2,"version_minor":0}

{"model_id":"6a7c54bf5c114d90a563ed97ebfa3381","version_major":2,"version_minor":0}

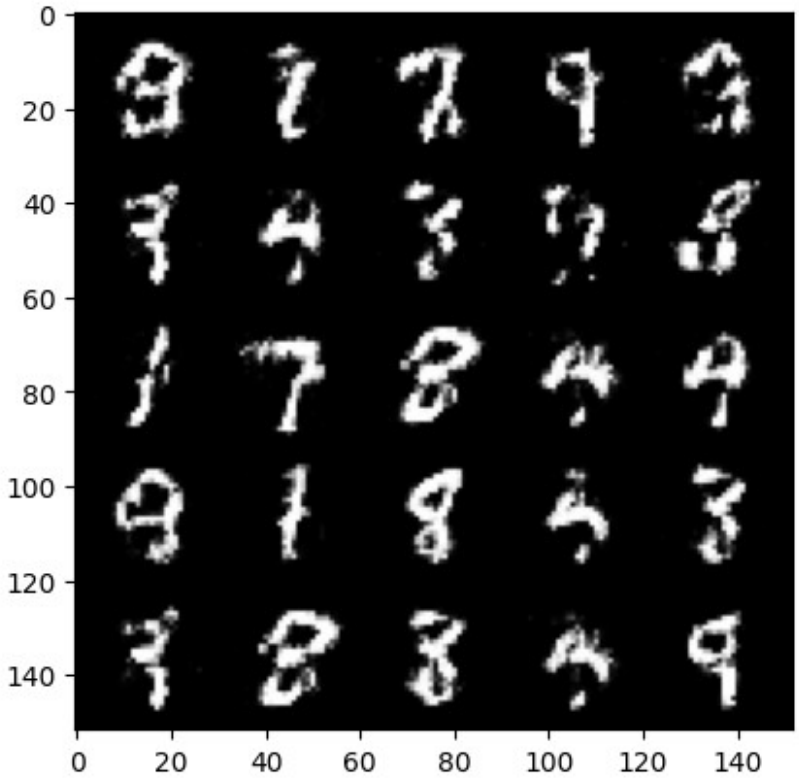
```
Epoch 113, step 53000: Generator loss: 1.7682886781692502,
discriminator loss: 0.35847614642977665
```

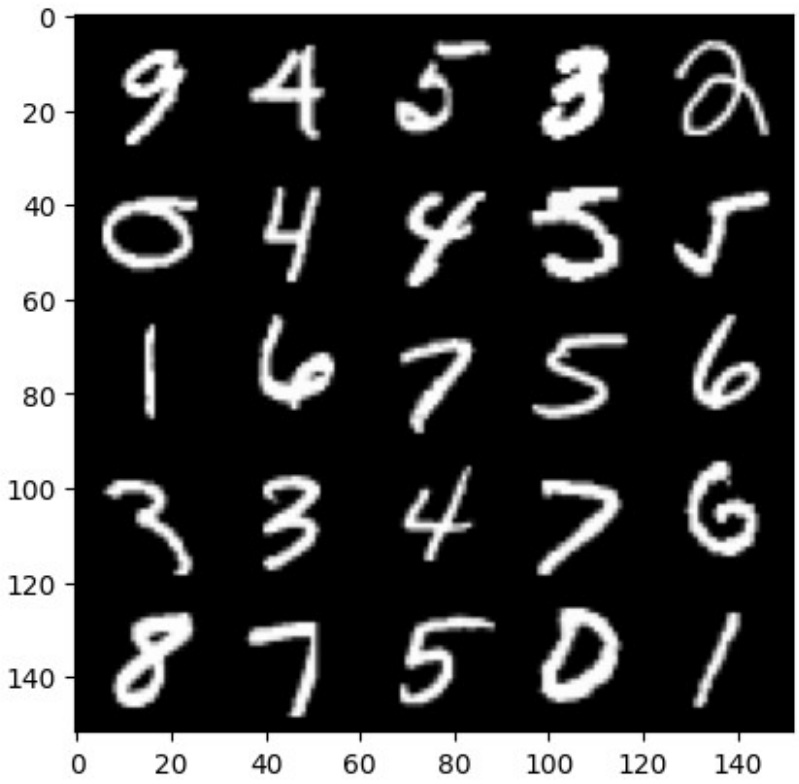

0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140


0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

{"model_id":"90ea2760f15b4c1ca3665ed8ea6b044f","version_major":2,"version_minor":0}

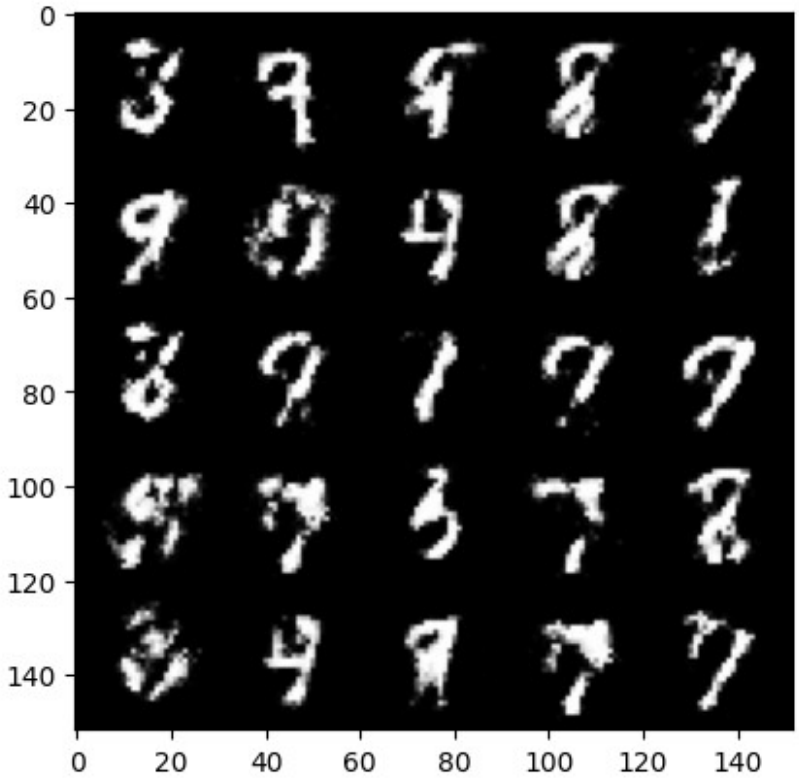
```
Epoch 114, step 53500: Generator loss: 1.6380481088161478,
discriminator loss: 0.3800544746518133
```

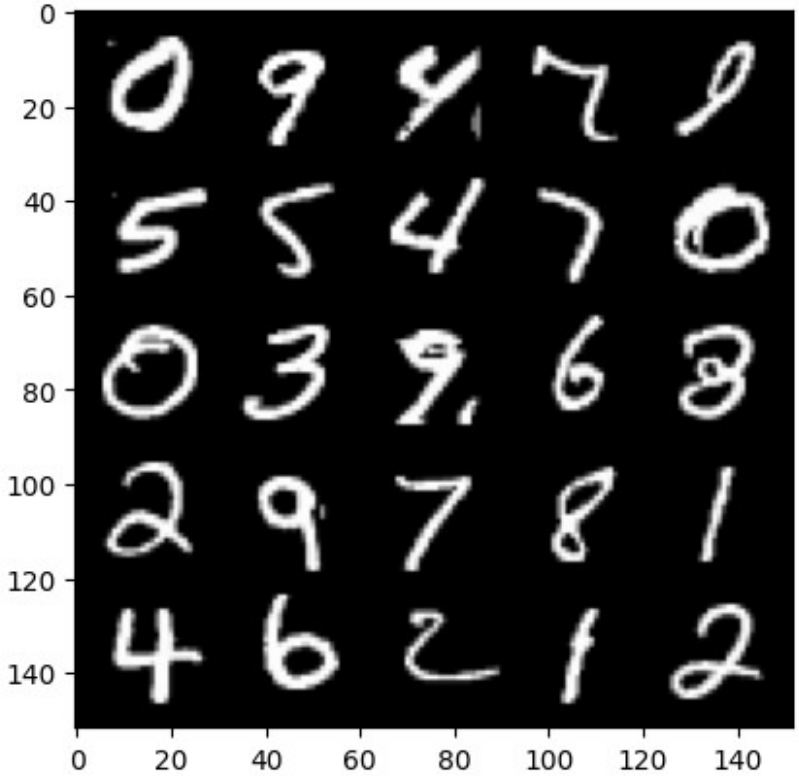




{"model_id":"a057d7417ef547c49506034d1bcc087f","version_major":2,"version_minor":0}

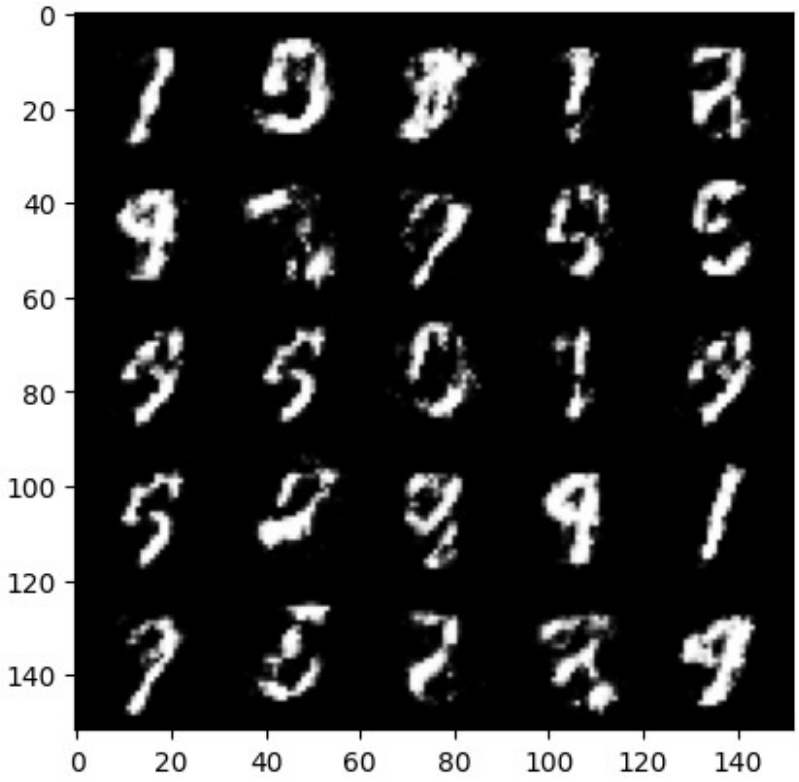
```
Epoch 115, step 54000: Generator loss: 1.748678301095962,
discriminator loss: 0.35282795006036743
```

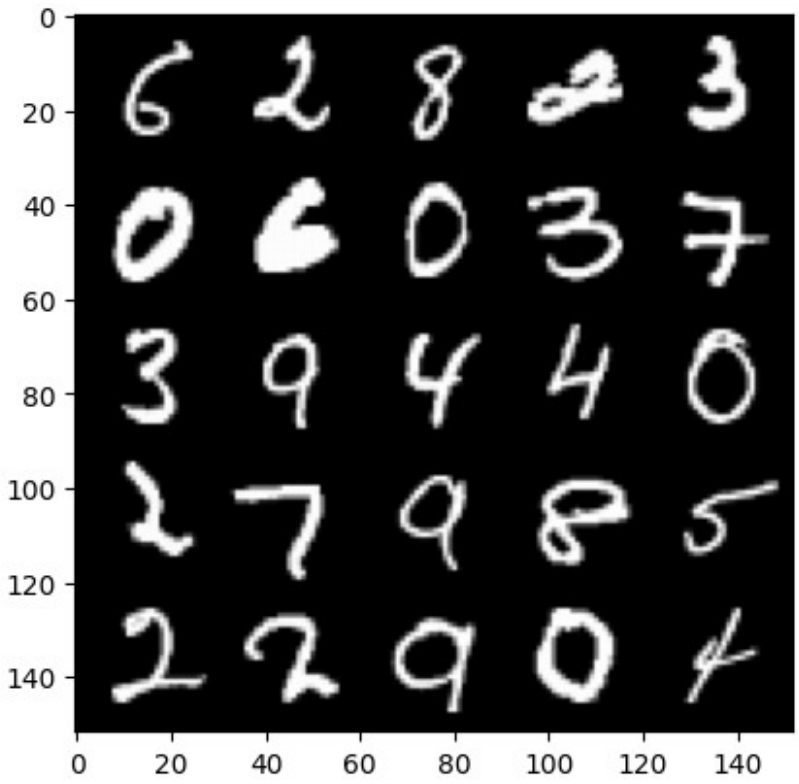

0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140


0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

{"model_id":"6471c21c389a4bc7ae89e597aab47d9d","version_major":2,"version_minor":0}

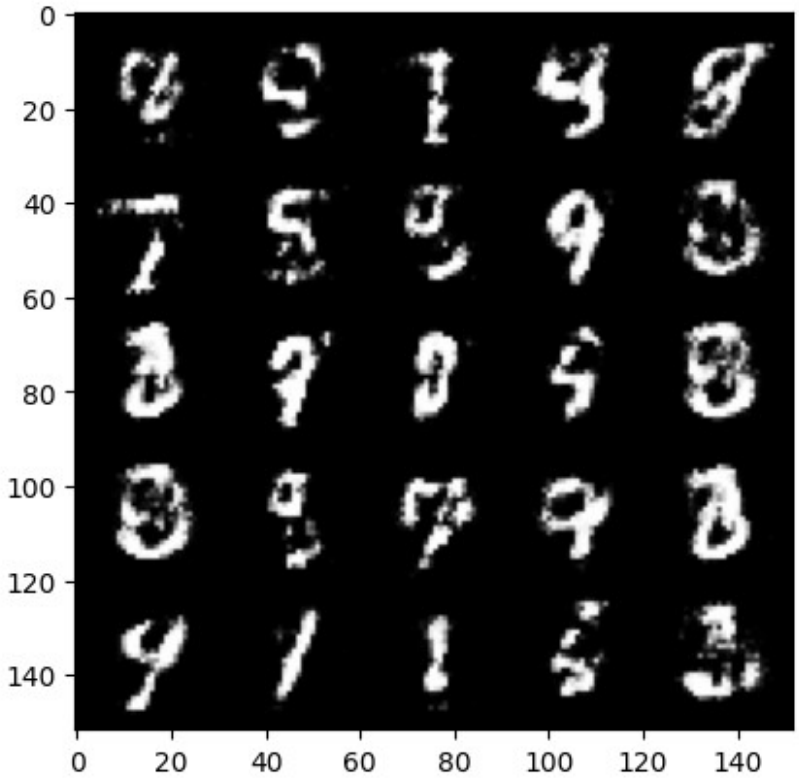
```
Epoch 116, step 54500: Generator loss: 1.7121074163913712,
discriminator loss: 0.366961000621319
```

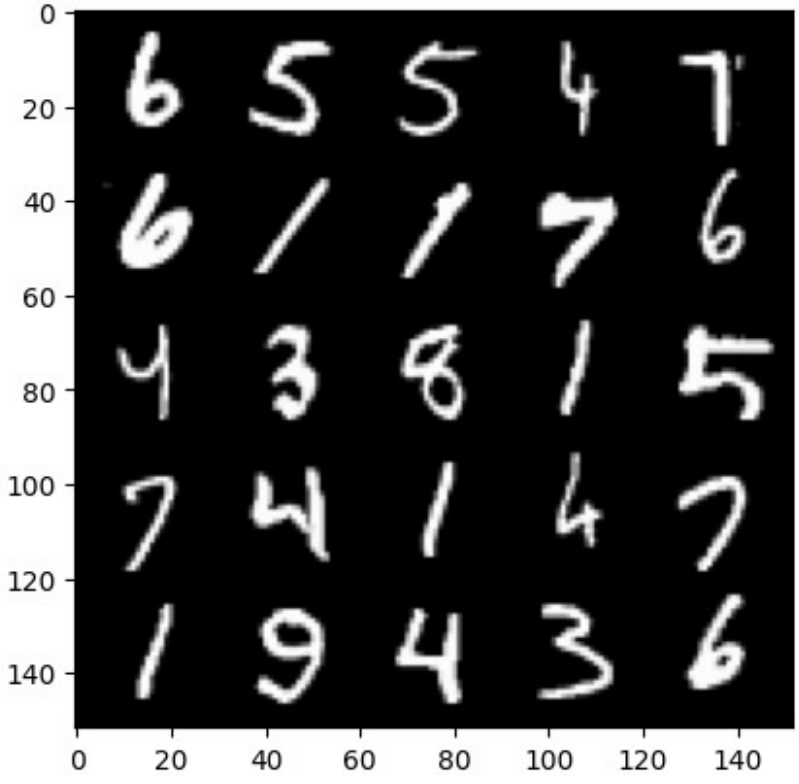




{"model_id":"97848aa6457747c99b7b51eb0d8203f0","version_major":2,"version_minor":0}

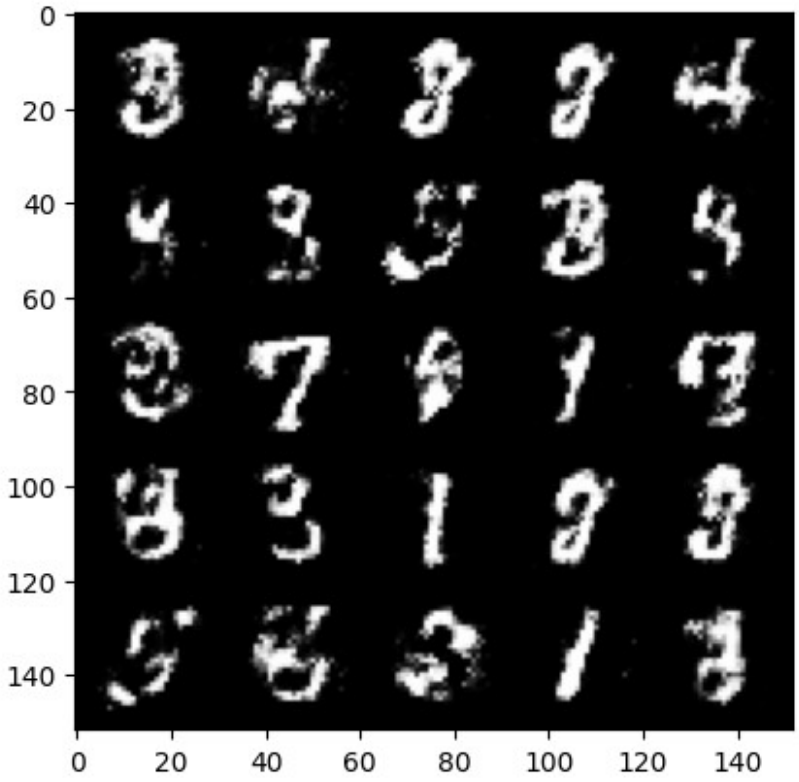
```
Epoch 117, step 55000: Generator loss: 1.5479956648349773,
discriminator loss: 0.40184807026386266
```

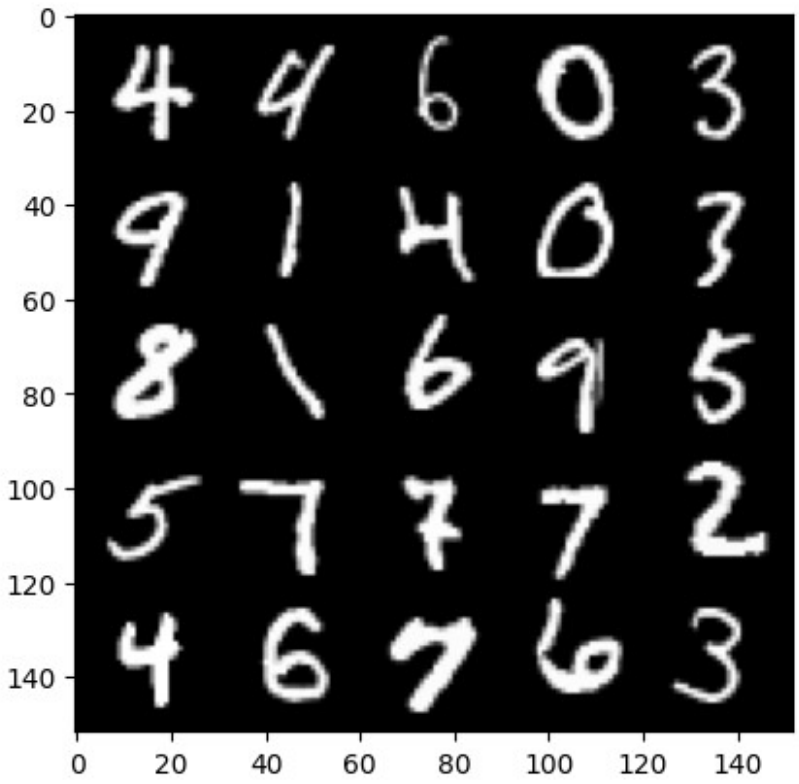

0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140


0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

{"model_id":"dc03205e7cf340ca99b8ce2d4c980272","version_major":2,"version_minor":0}

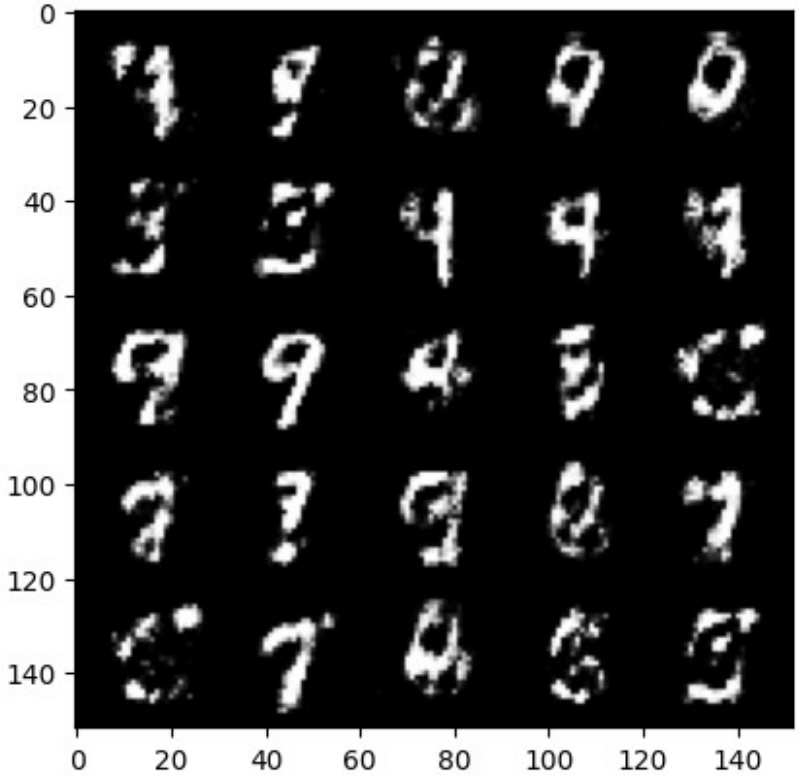
```
Epoch 118, step 55500: Generator loss: 1.6492255830764768,
discriminator loss: 0.3779219278693201
```

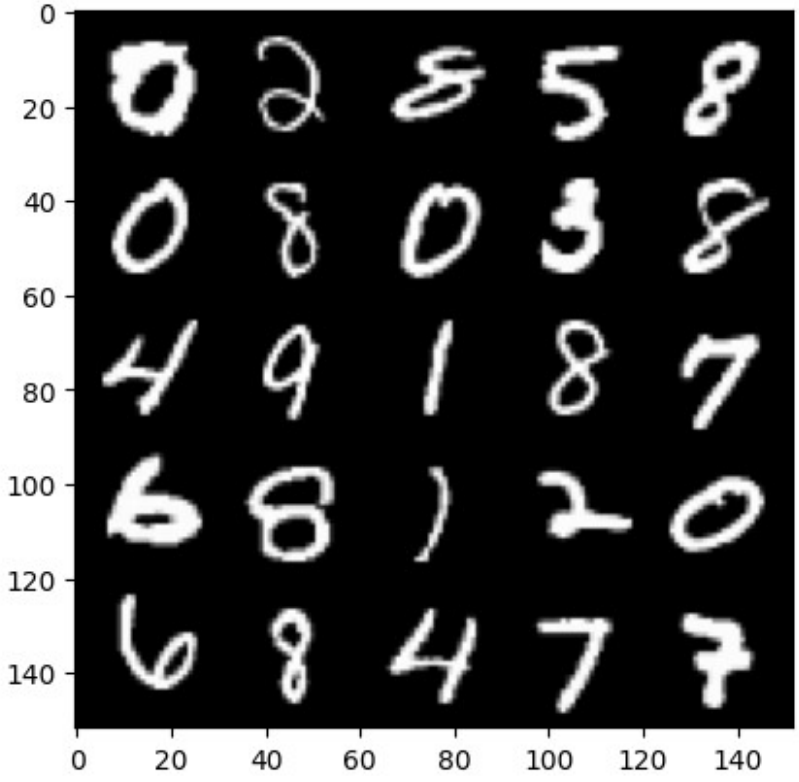




{"model_id":"dd262cac4c22482d8901ee21fc80b145","version_major":2,"version_minor":0}

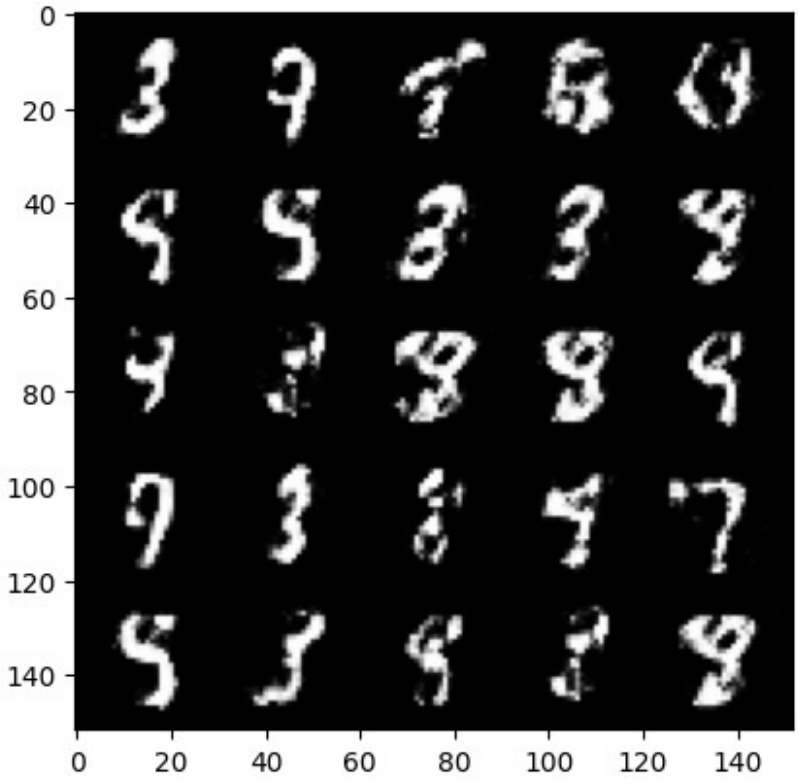
```
Epoch 119, step 56000: Generator loss: 1.7062084758281706,
discriminator loss: 0.3620736272931098
```

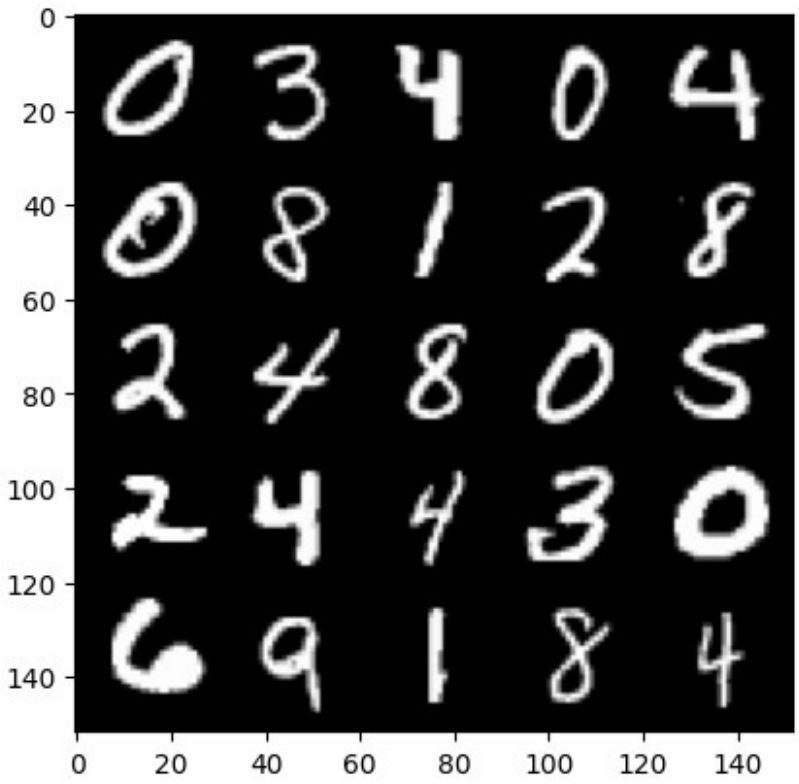

0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140


0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

{"model_id":"45b6b072c8d74ff9a5f90797558caed0","version_major":2,"version_minor":0}

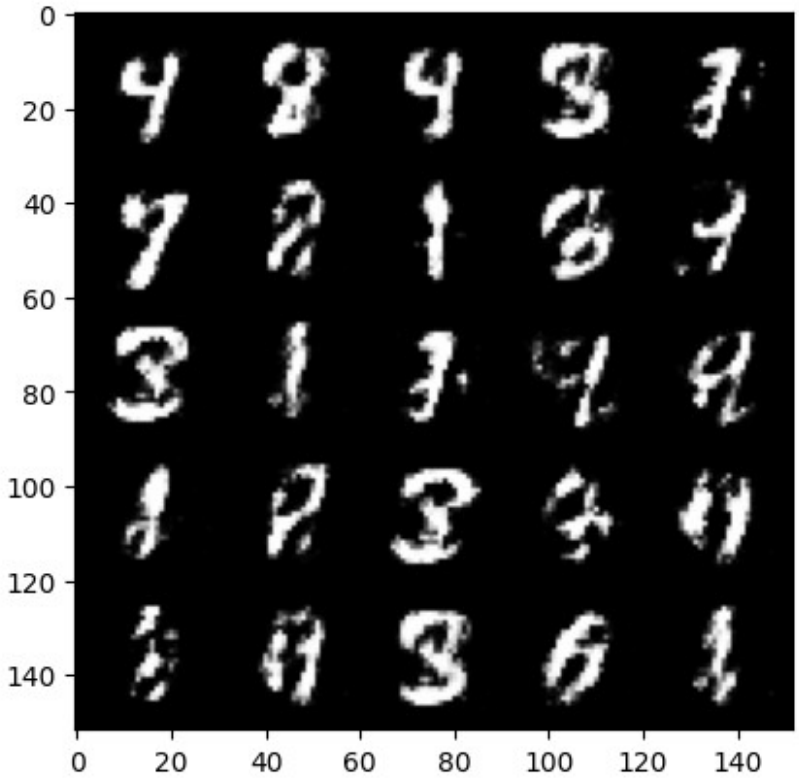
```
Epoch 120, step 56500: Generator loss: 1.64695772957802, discriminator loss: 0.3807286164164543
```

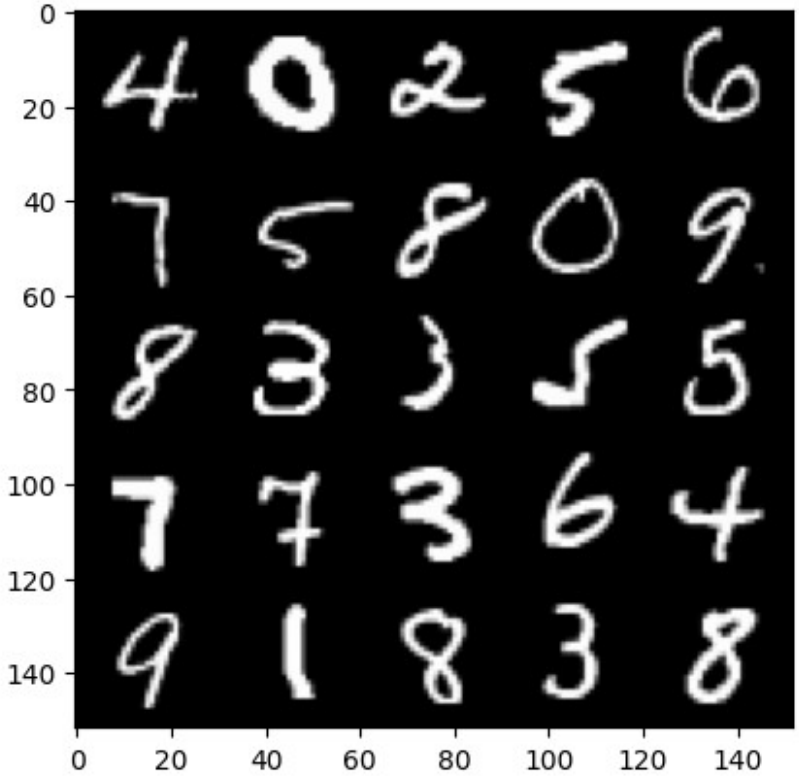




{"model_id":"1cf23b83b18947beaf1ac868b3232b91","version_major":2,"version_minor":0}

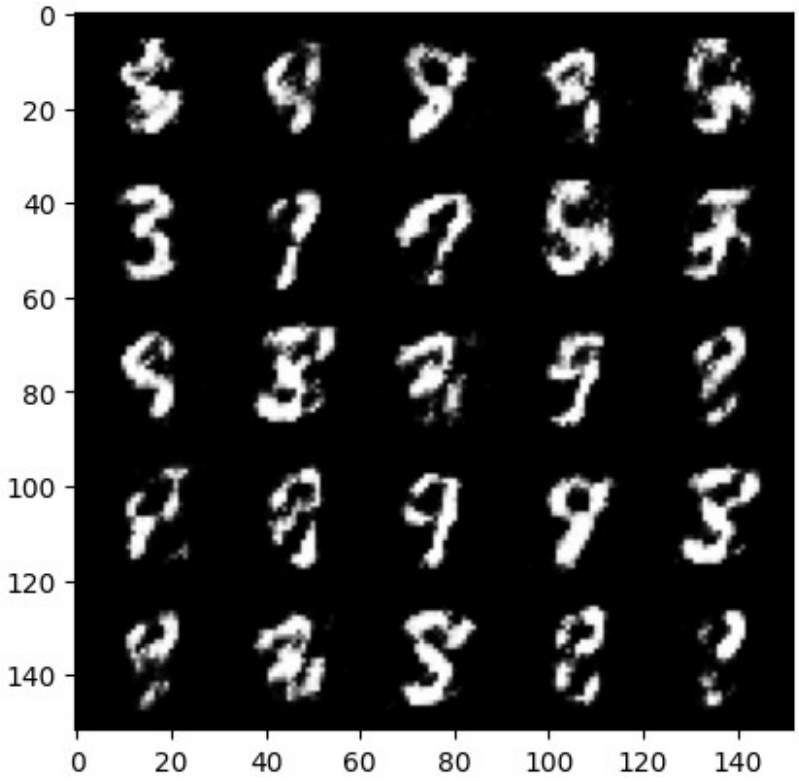
```
Epoch 121, step 57000: Generator loss: 1.7241800017356854,
discriminator loss: 0.37255384153127696
```

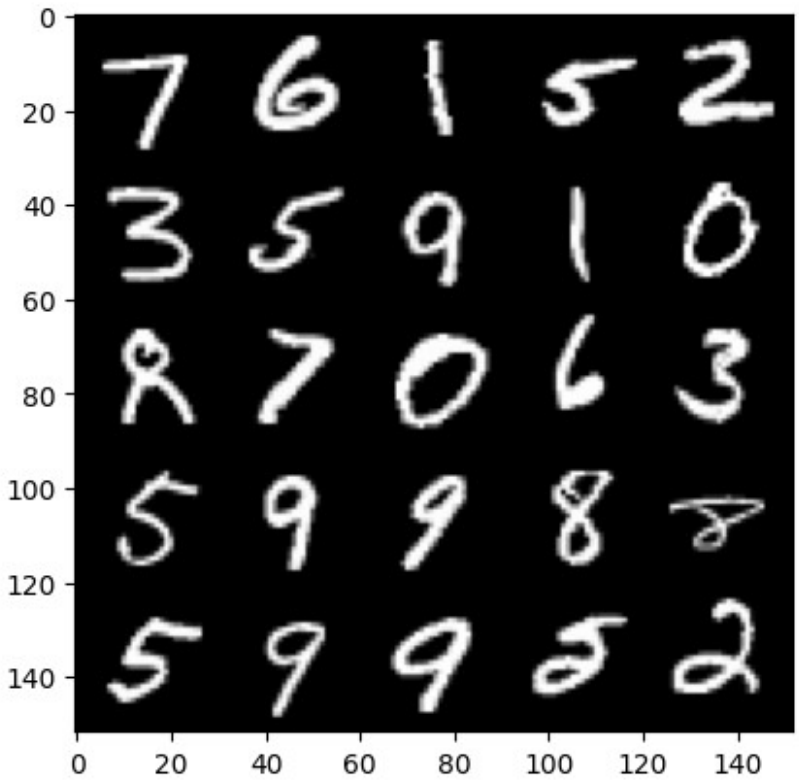

0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140


0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

{"model_id":"c87dfd149aad4945b9a8eae7c03e0b14","version_major":2,"version_minor":0}

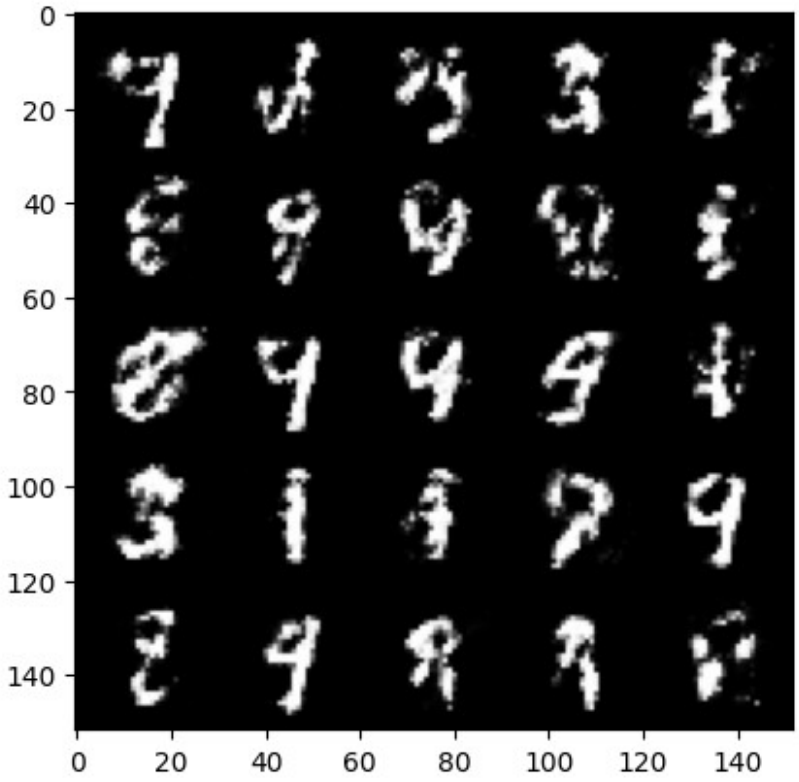
```
Epoch 122, step 57500: Generator loss: 1.6639376630783091,
discriminator loss: 0.3893752224445343
```

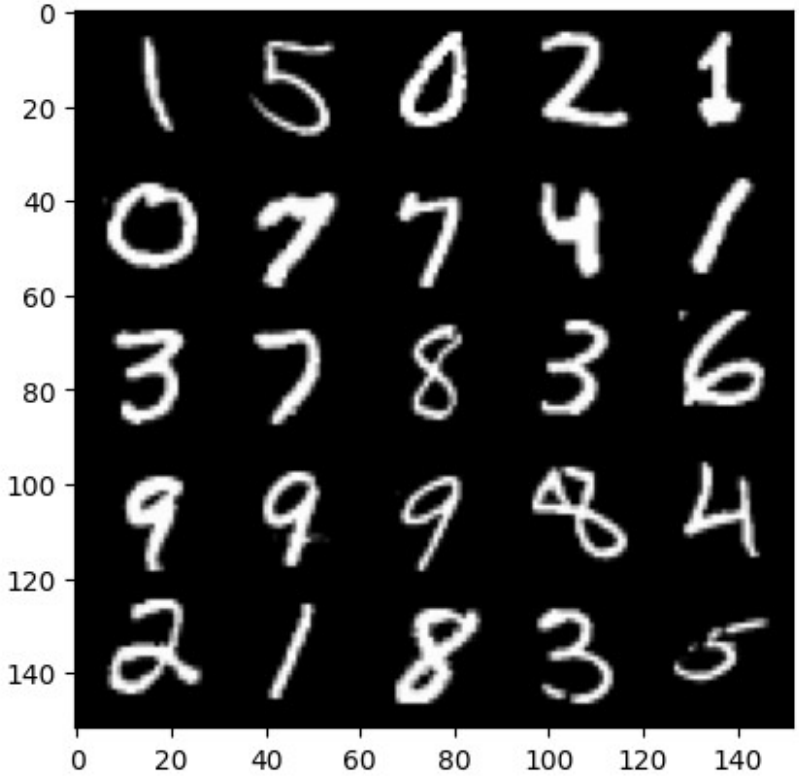




{"model_id":"abff7b80577f4d1db7a9ae16670895f0","version_major":2,"version_minor":0}

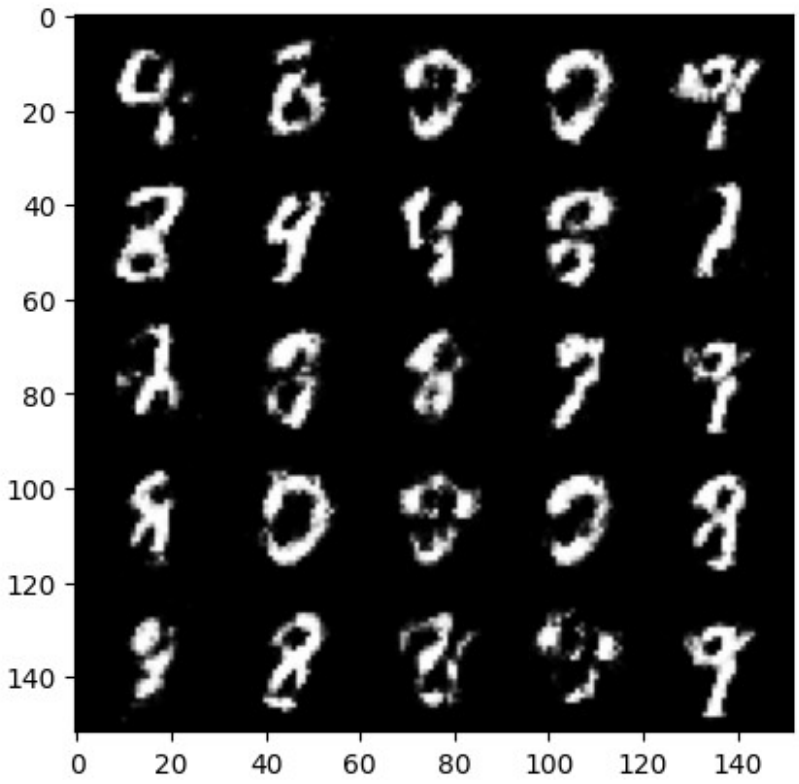
```
Epoch 123, step 58000: Generator loss: 1.5408071420192724,
discriminator loss: 0.40762056845426553
```

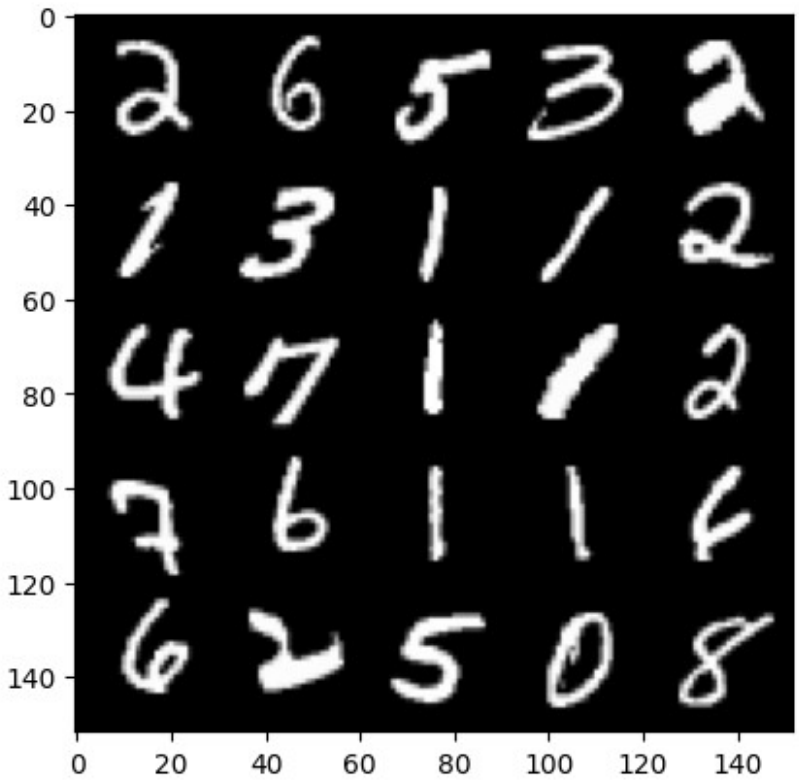

0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140


0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

{"model_id":"f71d4f37c885448291f187188be95810","version_major":2,"version_minor":0}

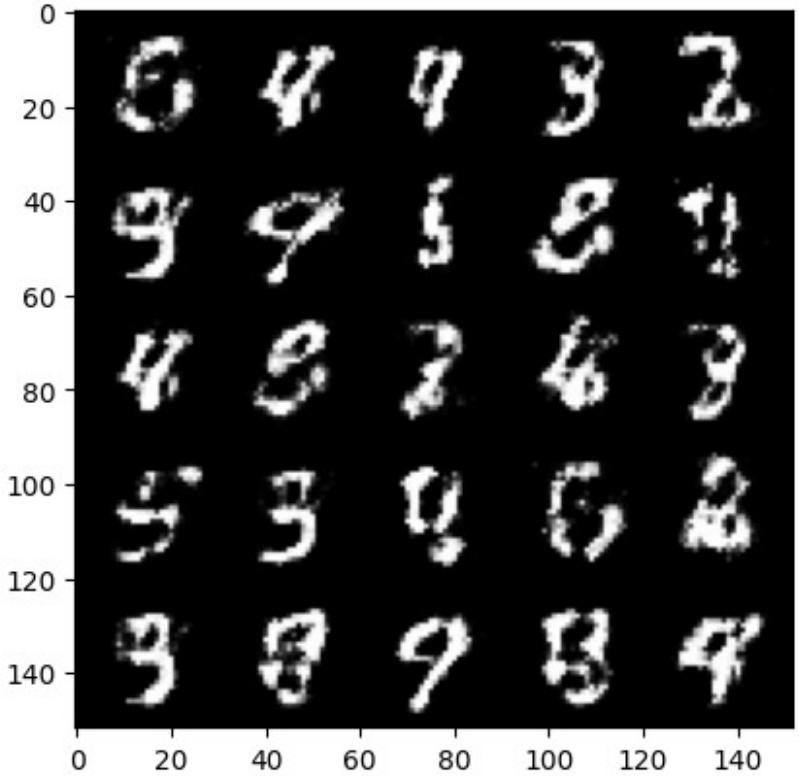
```
Epoch 124, step 58500: Generator loss: 1.6096824288368228,
discriminator loss: 0.38704015368223216
```

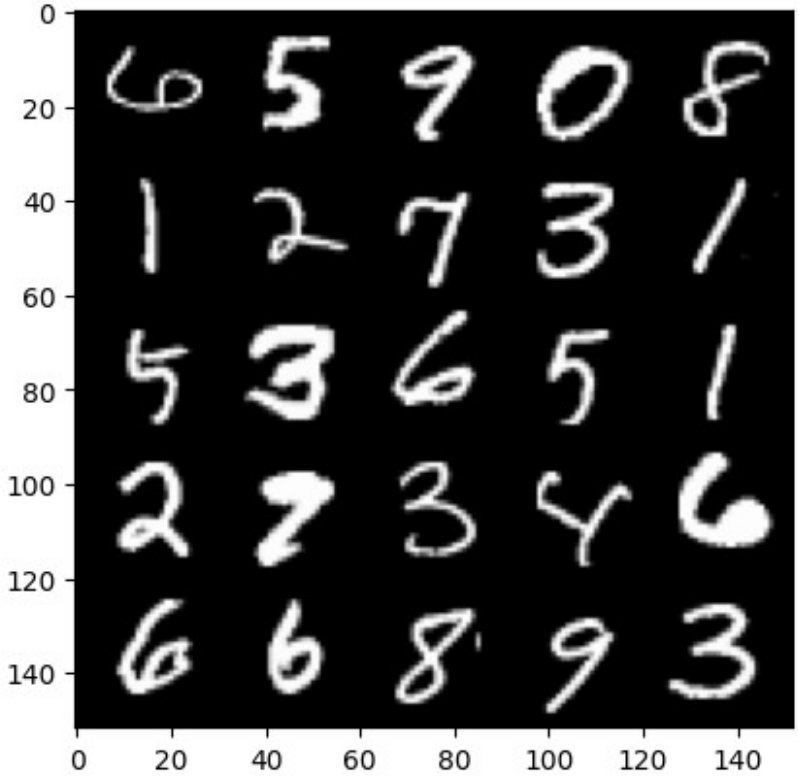




{"model_id":"771654411e834cd5b99c7c6d29e3464a","version_major":2,"version_minor":0}

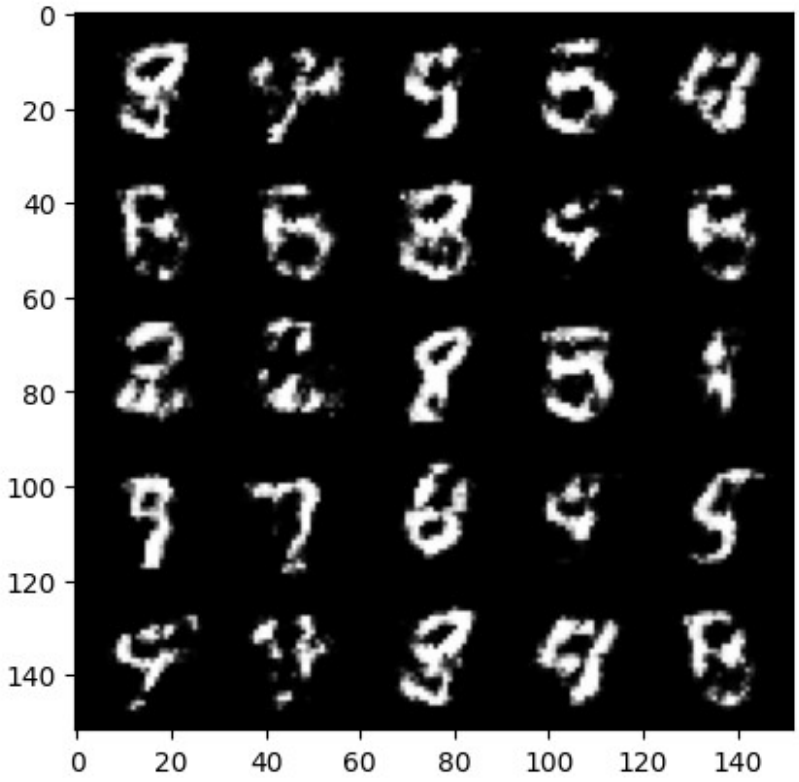
```
Epoch 125, step 59000: Generator loss: 1.5919881157875058,
discriminator loss: 0.3907961006164553
```

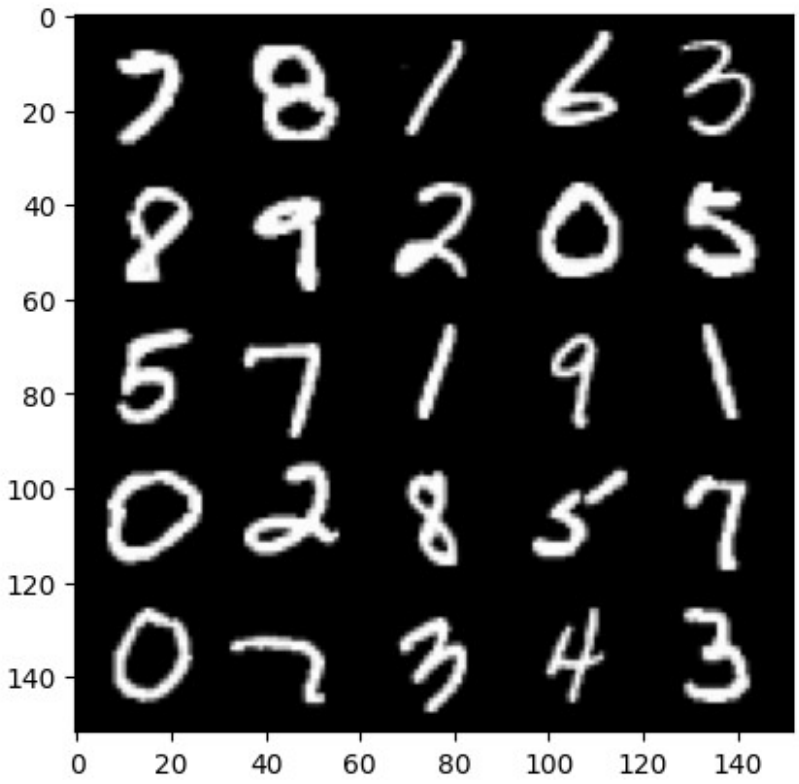

0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140


0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

{"model_id":"2d8ec3aec49a417486e7351d7f9be57c","version_major":2,"version_minor":0}

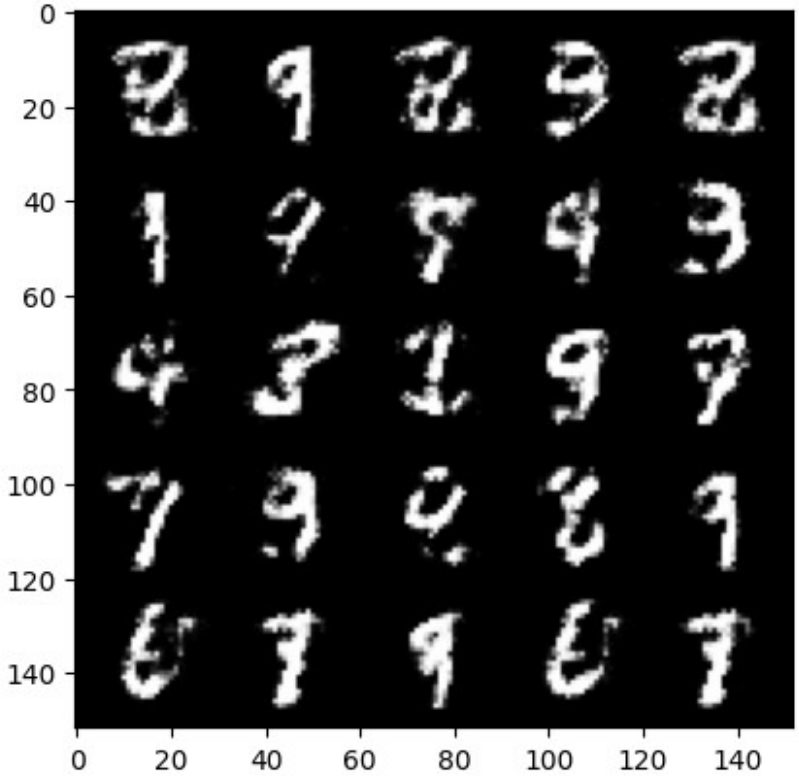
```
Epoch 126, step 59500: Generator loss: 1.6106592631340027,
discriminator loss: 0.38297334587573956
```

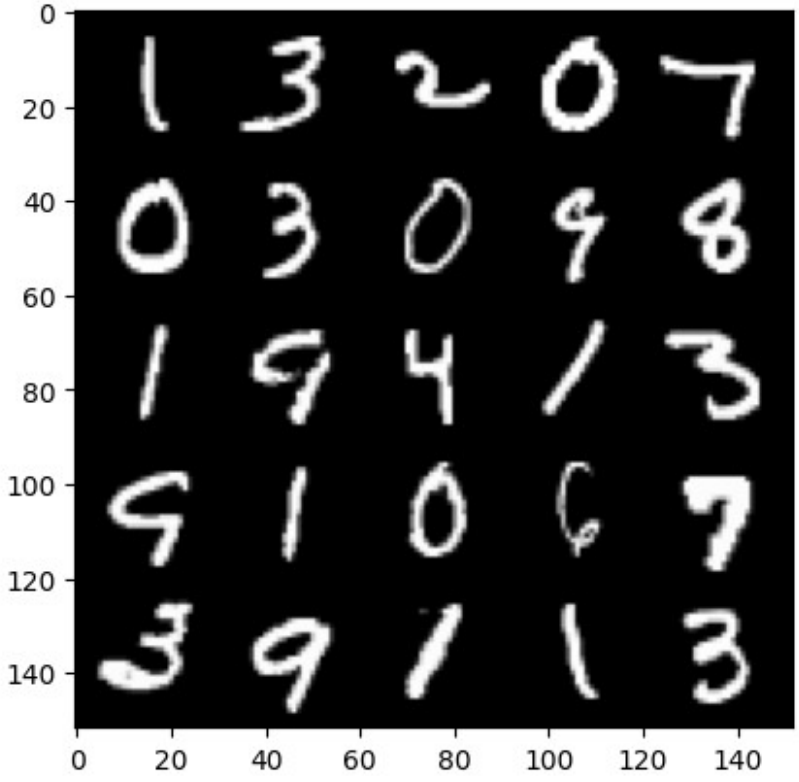




{"model_id":"532d0e55bfd74565ab3e780d3b69001d","version_major":2,"version_minor":0}

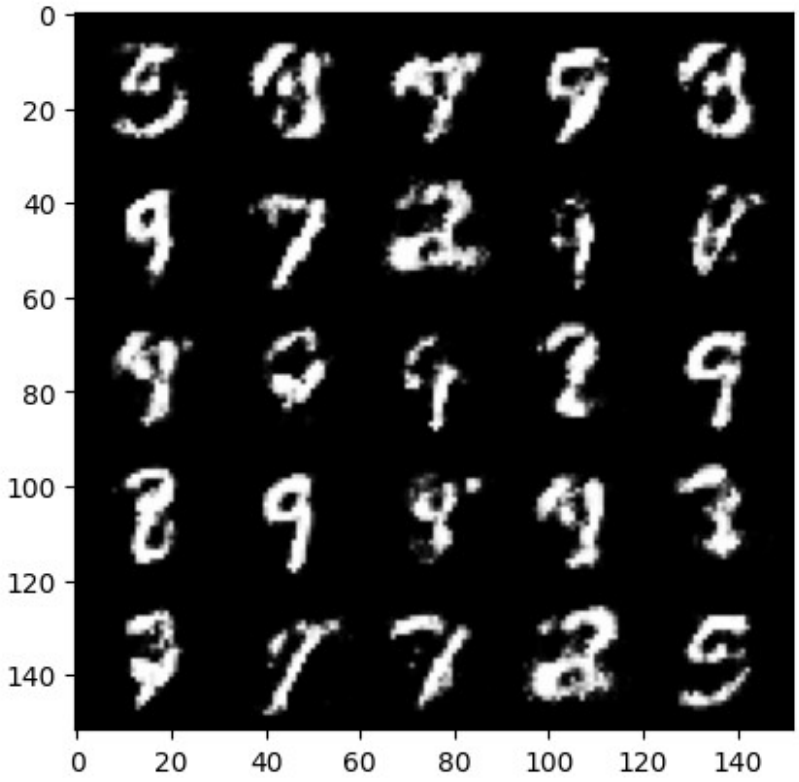
```
Epoch 127, step 60000: Generator loss: 1.5947945420742038,
discriminator loss: 0.39465465718507775
```

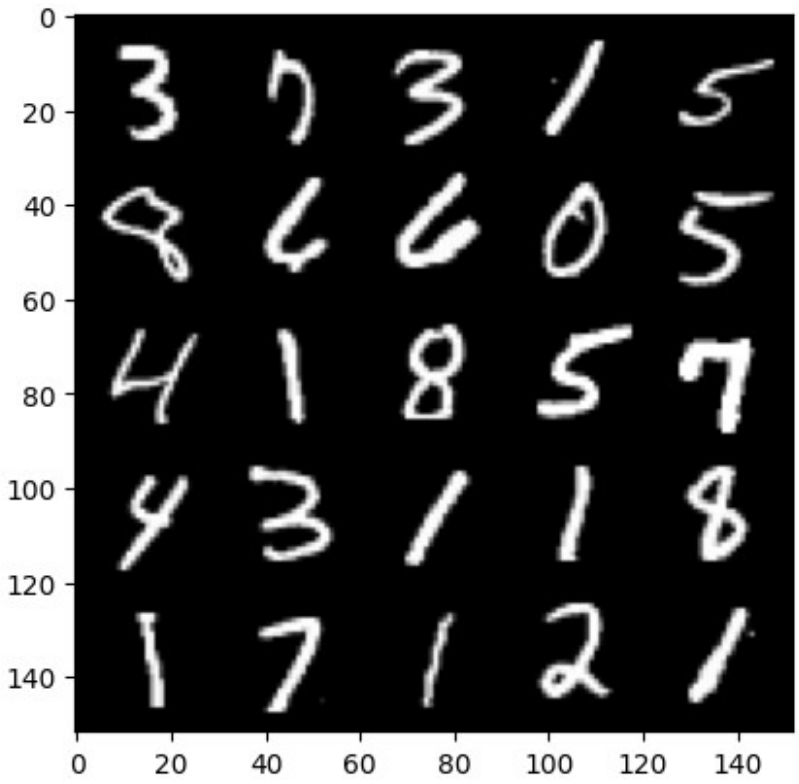

0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140


0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

{"model_id":"faf2cece93ce407db57f476641607ee6","version_major":2,"version_minor":0}

```
Epoch 128, step 60500: Generator loss: 1.5369649121761317,
discriminator loss: 0.41301939207315447
```

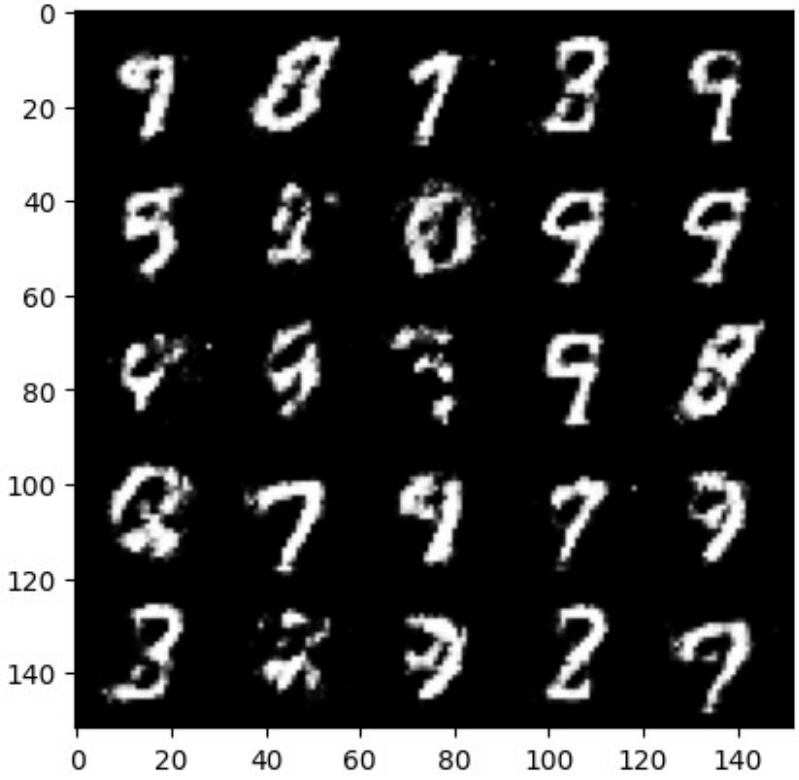


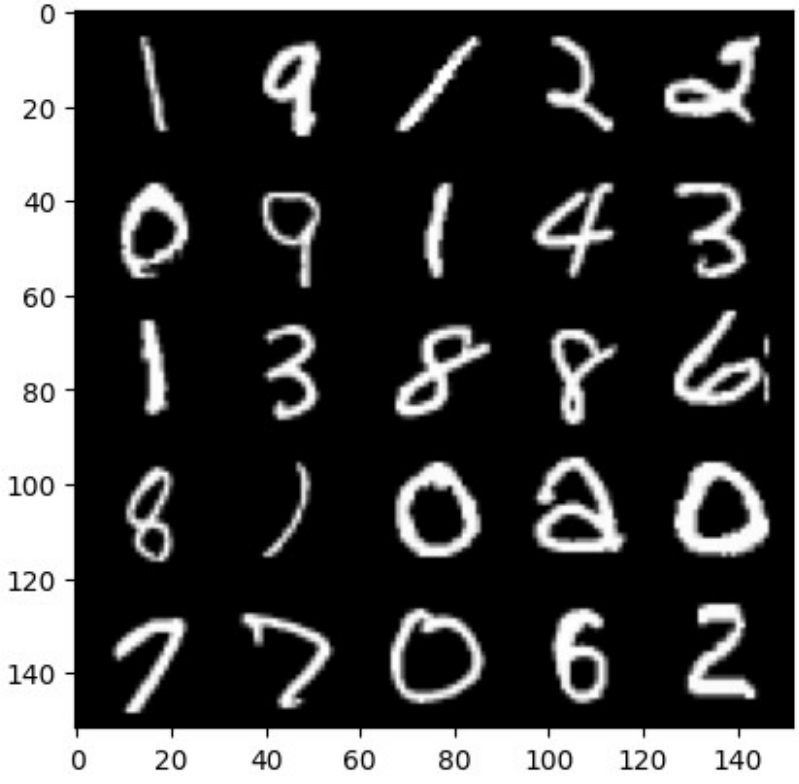


{"model_id":"ef74374fec97408faf786a7940cd3fde","version_major":2,"version_minor":0}

{"model_id":"a331542baba04269add13313c9e1a444","version_major":2,"version_minor":0}

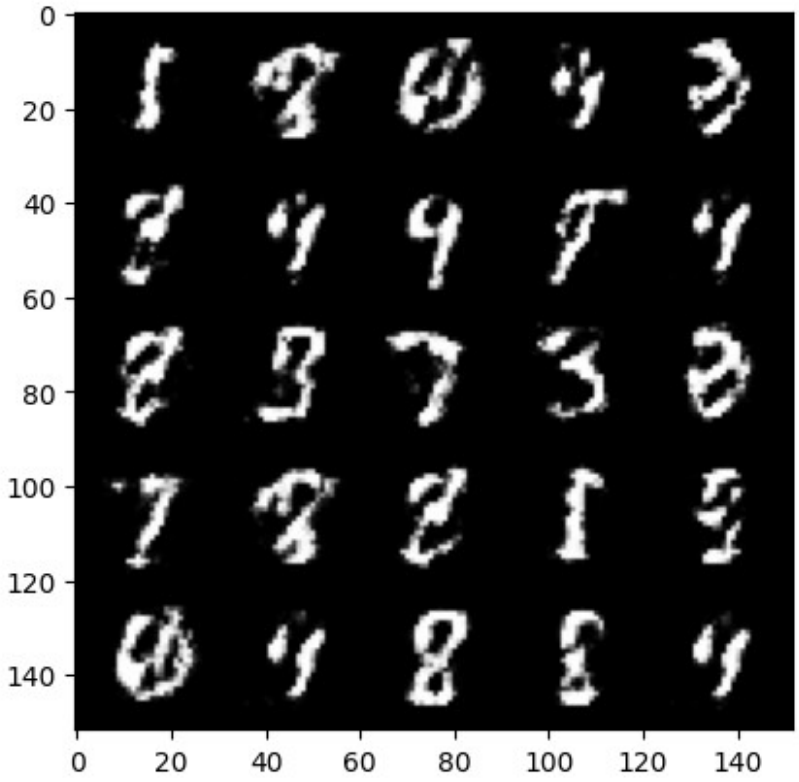
```
Epoch 130, step 61000: Generator loss: 1.5114179522991191,
discriminator loss: 0.3986522882580759
```

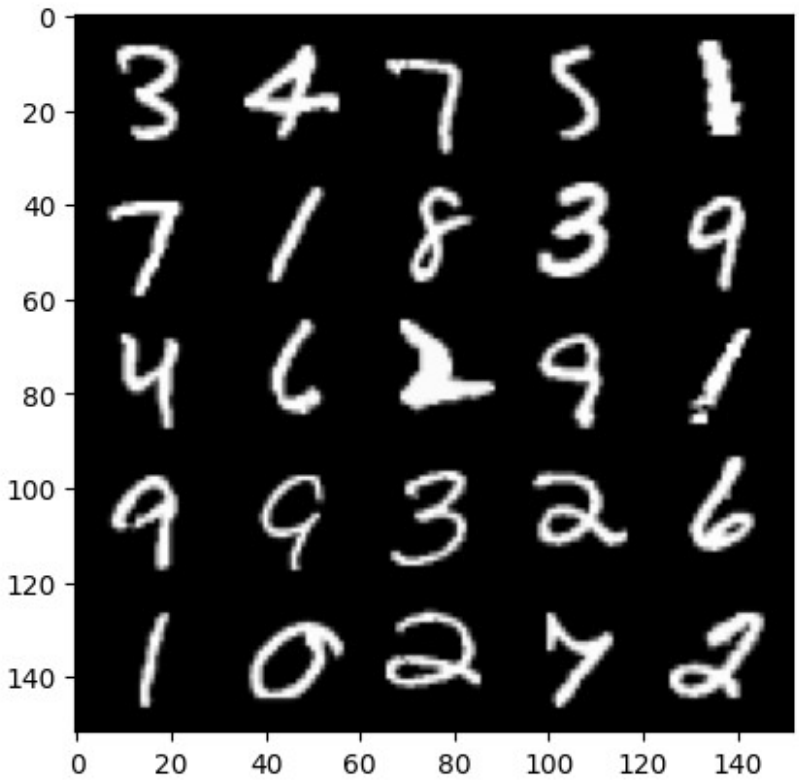

0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140


0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

{"model_id":"593eecfa72094d5b9a8898d3d2924dc3","version_major":2,"version_minor":0}

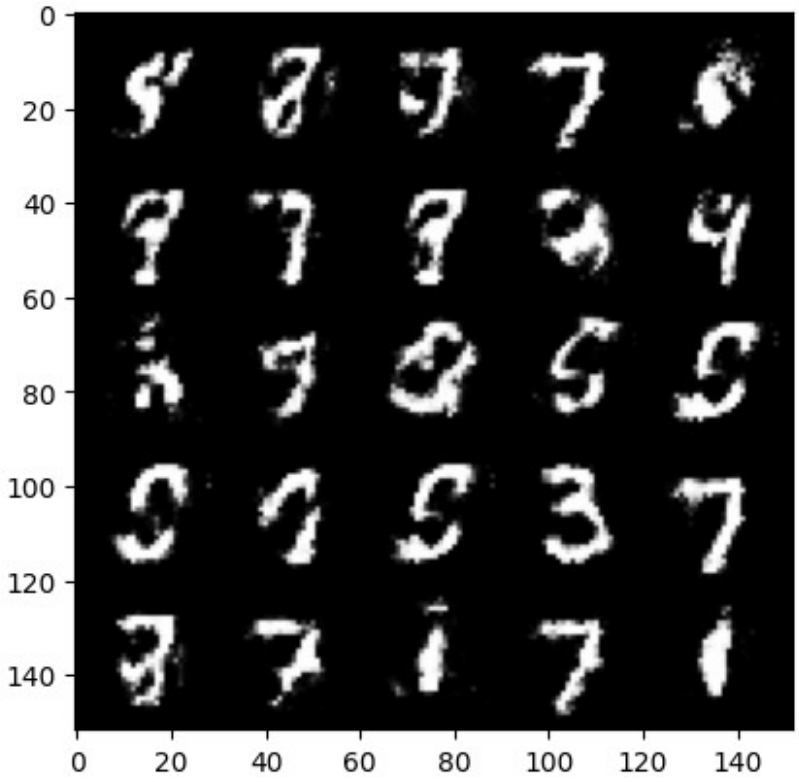
```
Epoch 131, step 61500: Generator loss: 1.5340157196521766,
discriminator loss: 0.3956468819379808
```

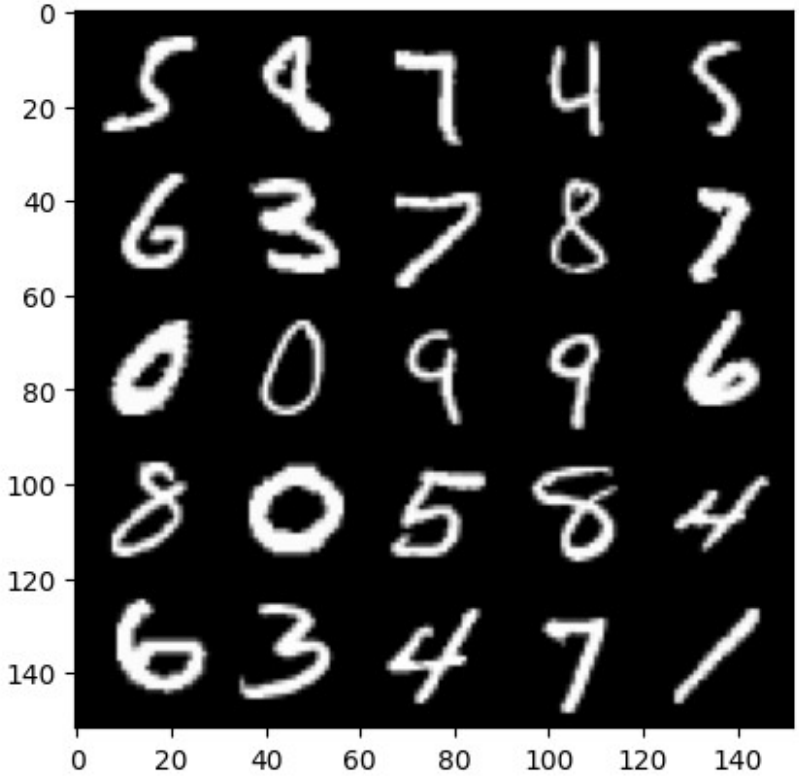




{"model_id":"e015e928cb4c4ee79c9024bc8f464d02","version_major":2,"version_minor":0}

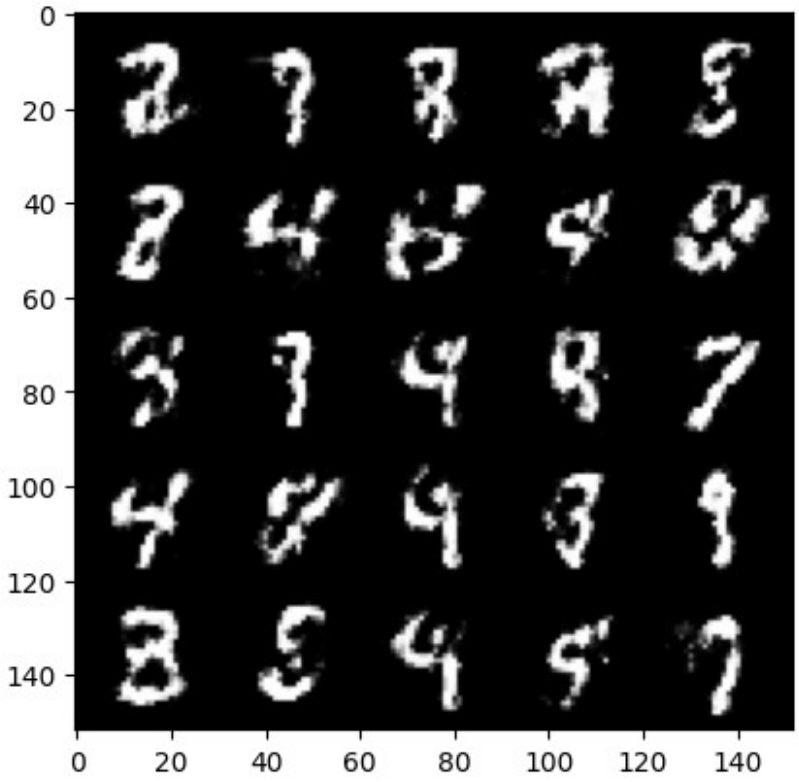
```
Epoch 132, step 62000: Generator loss: 1.5678553402423856,
discriminator loss: 0.39843031382560673
```

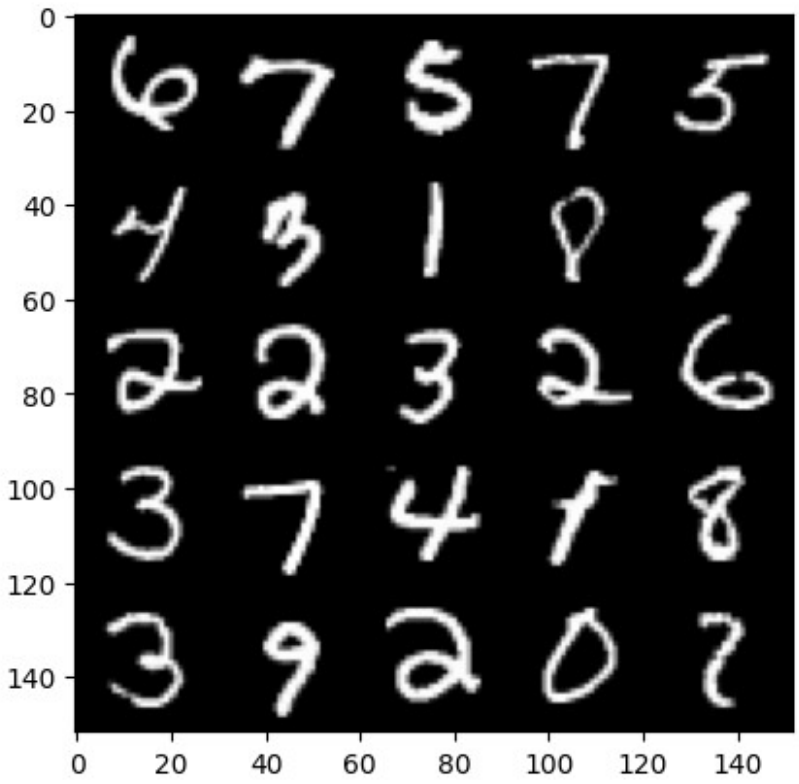

0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140


0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

{"model_id":"ed10331f857f4be7bc8a5fb894aaf981","version_major":2,"version_minor":0}

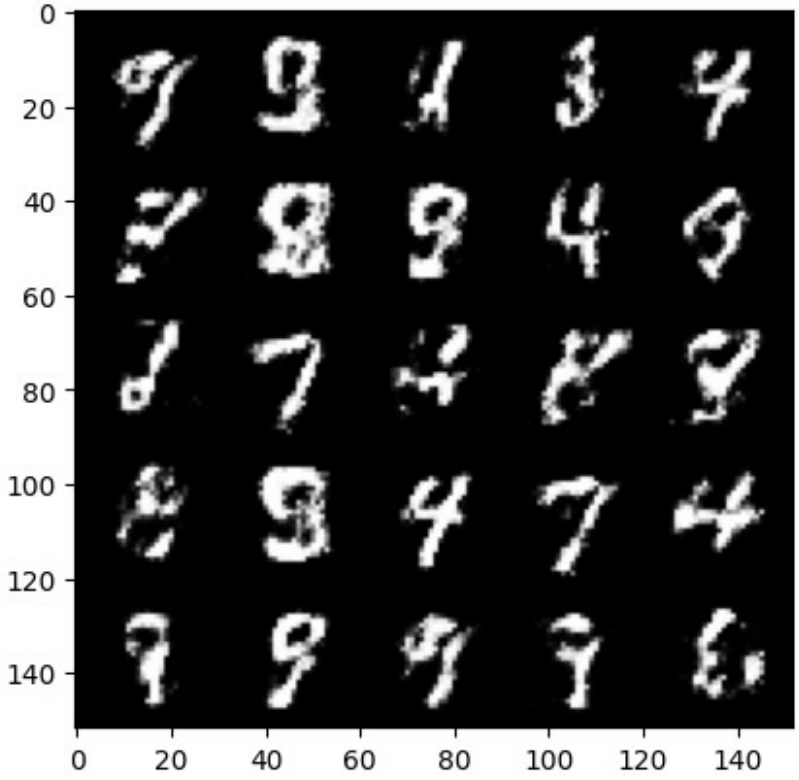
```
Epoch 133, step 62500: Generator loss: 1.5380628845691686,
discriminator loss: 0.39969476771354673
```

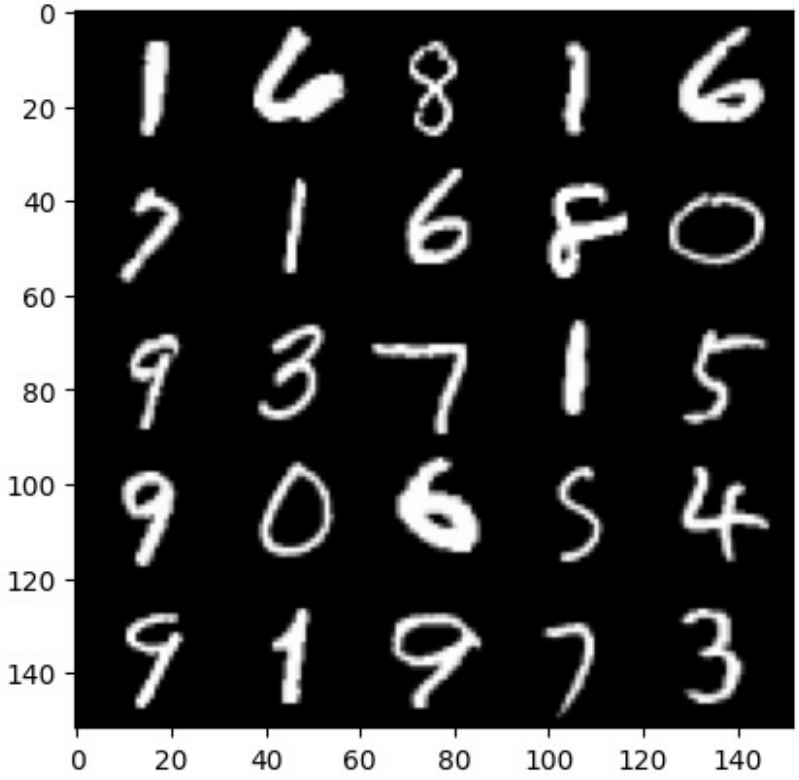




{"model_id":"6a057ba407b64766bf57969f09e3dd8f","version_major":2,"version_minor":0}

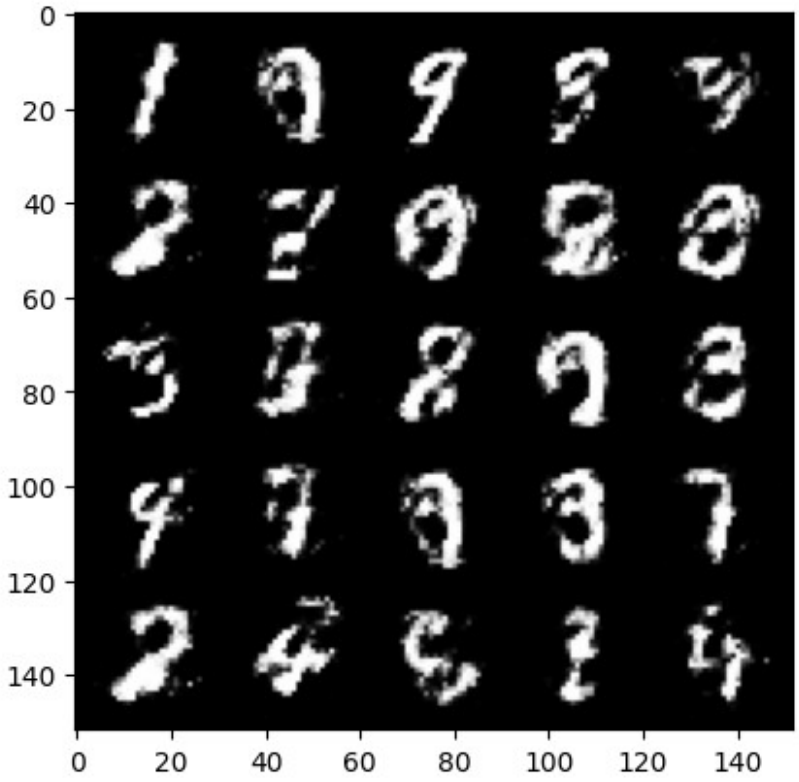
```
Epoch 134, step 63000: Generator loss: 1.5263159272670754,
discriminator loss: 0.41410890334844597
```

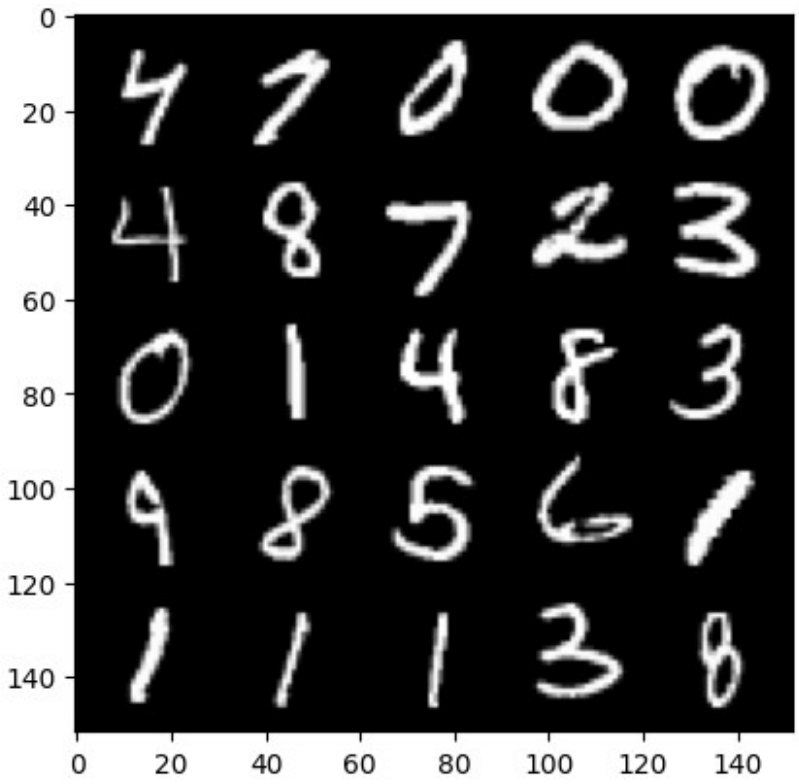

0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140


0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

{"model_id":"0ff88d08a6f84405829a85cdf51495f7","version_major":2,"version_minor":0}

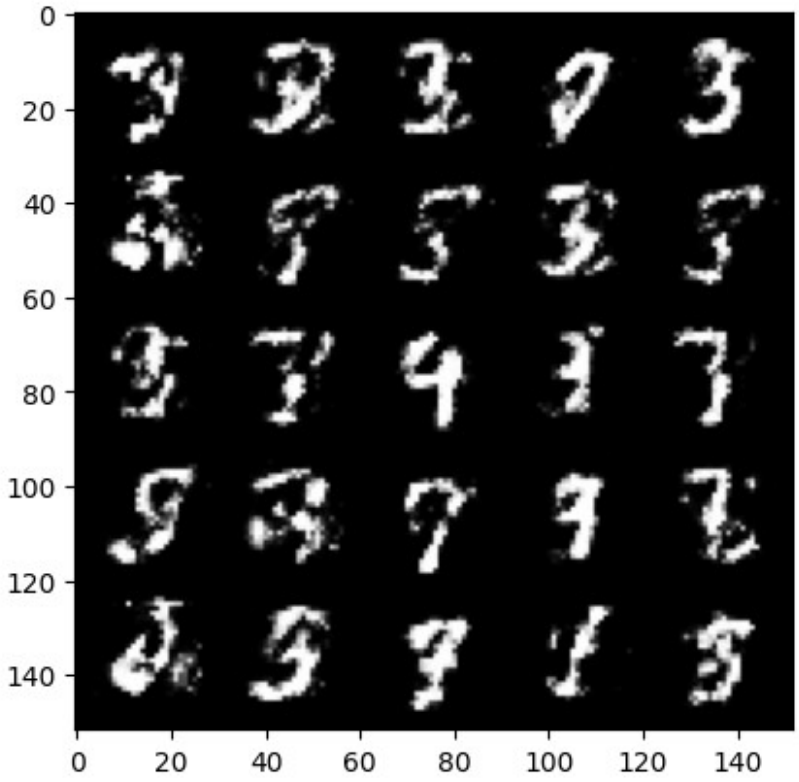
```
Epoch 135, step 63500: Generator loss: 1.4644474236965168,
discriminator loss: 0.42788133114576343
```

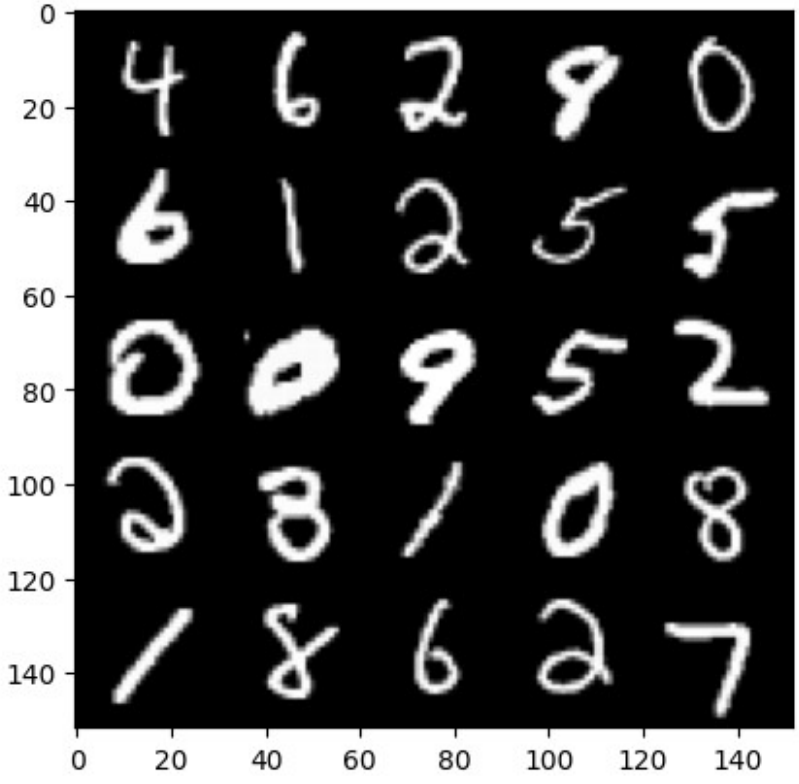




{"model_id":"f14763bb25f044eba1a67d0865dfaa5f","version_major":2,"version_minor":0}

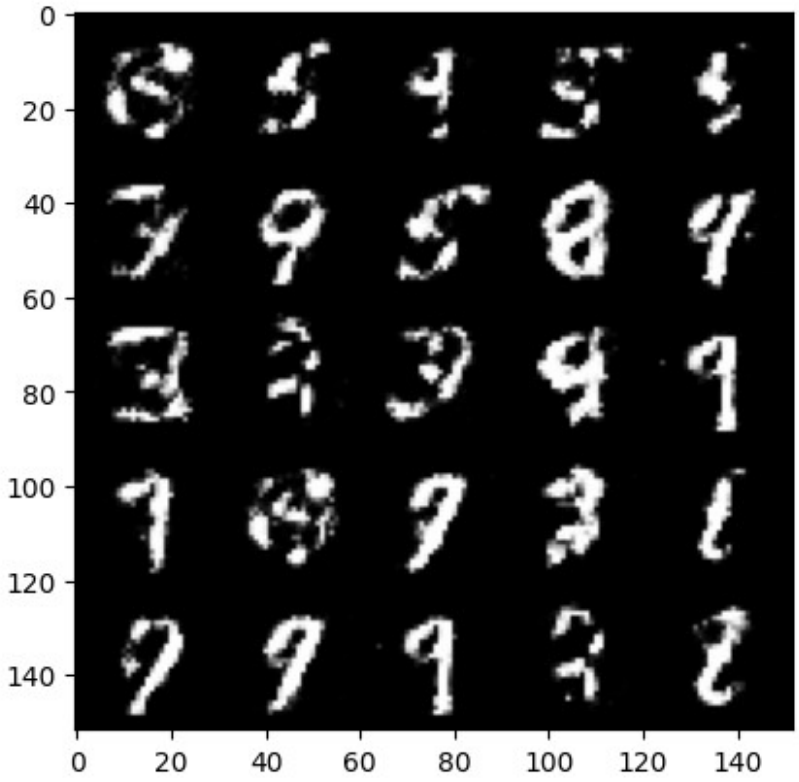
```
Epoch 136, step 64000: Generator loss: 1.5582034389972692,
discriminator loss: 0.38456566309928897
```

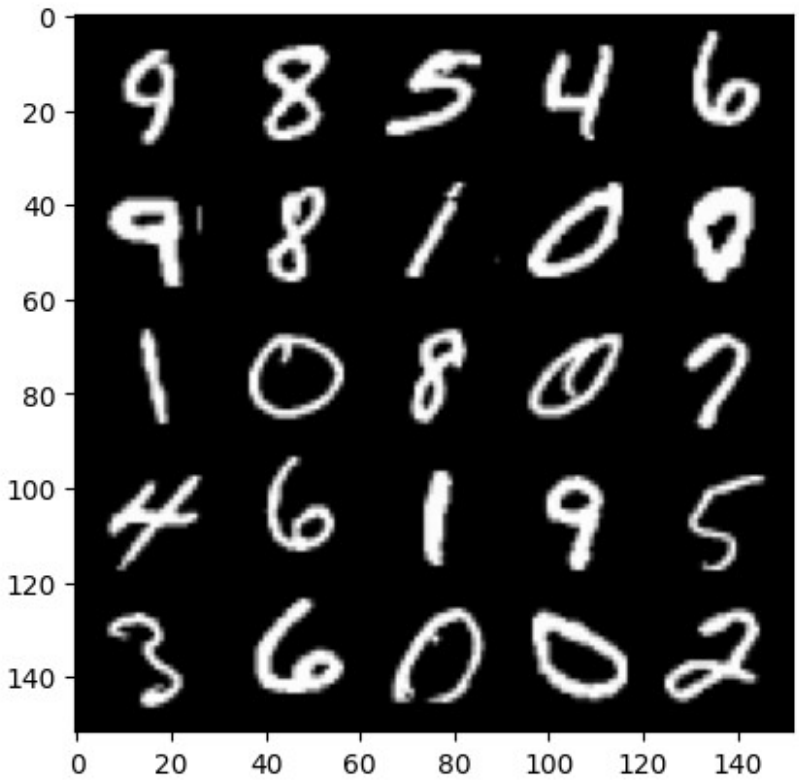

0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140


0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

{"model_id":"19e179c7a6814f49af9b10088e97085f","version_major":2,"version_minor":0}

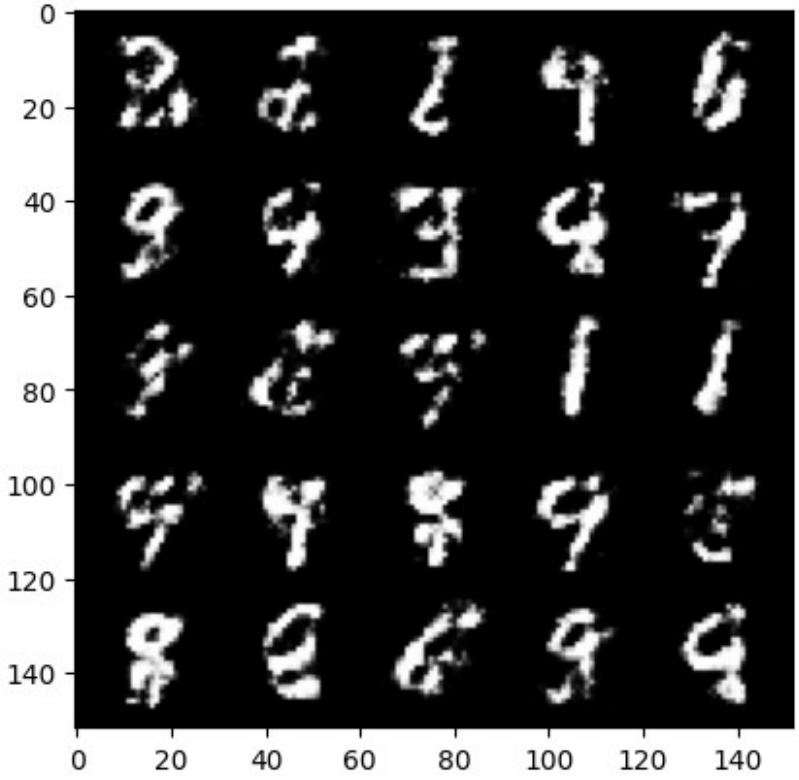
```
Epoch 137, step 64500: Generator loss: 1.545066702604293,
discriminator loss: 0.40264114433527026
```

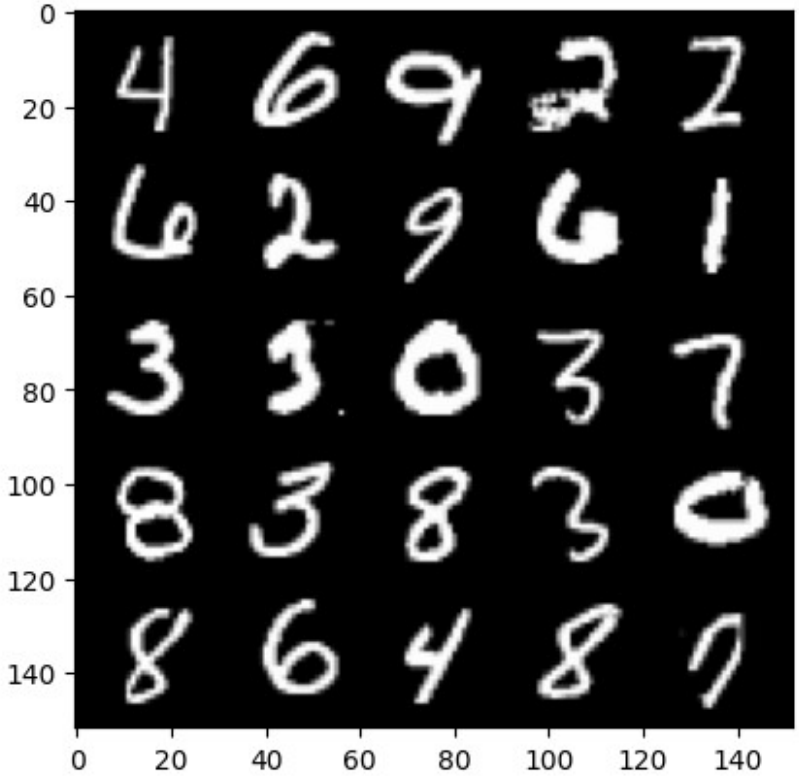




{"model_id":"43e06fa9d1d04d07b72e31d4ede74065","version_major":2,"version_minor":0}

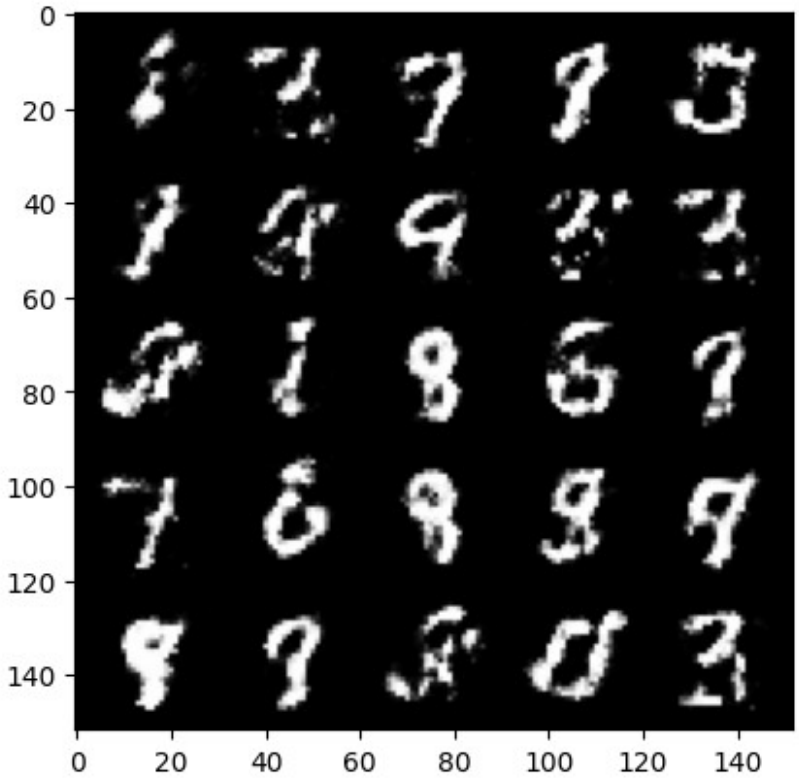
```
Epoch 138, step 65000: Generator loss: 1.4974509639739995,
discriminator loss: 0.41399756920337694
```

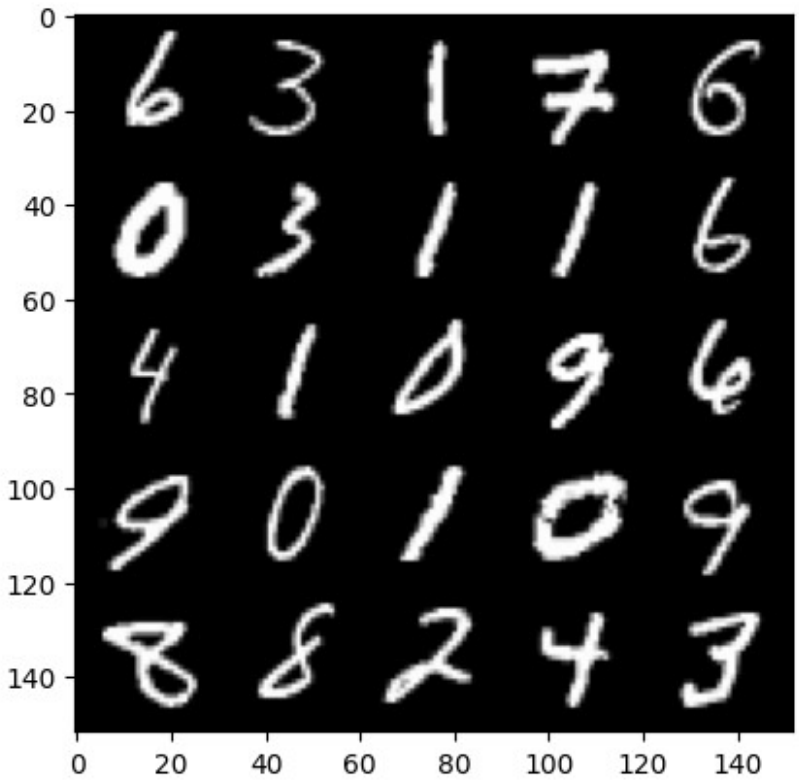

0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140


0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

{"model_id":"c9f9cd34bc6c420982a4d38ff3a26c27","version_major":2,"version_minor":0}

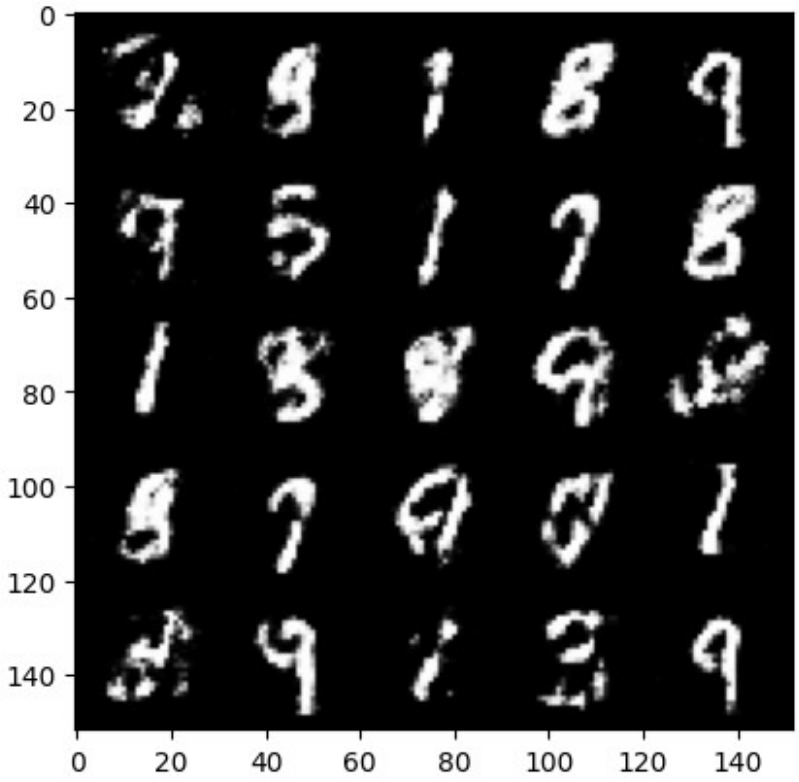
```
Epoch 139, step 65500: Generator loss: 1.5106878771781922,
discriminator loss: 0.40617393660545376
```

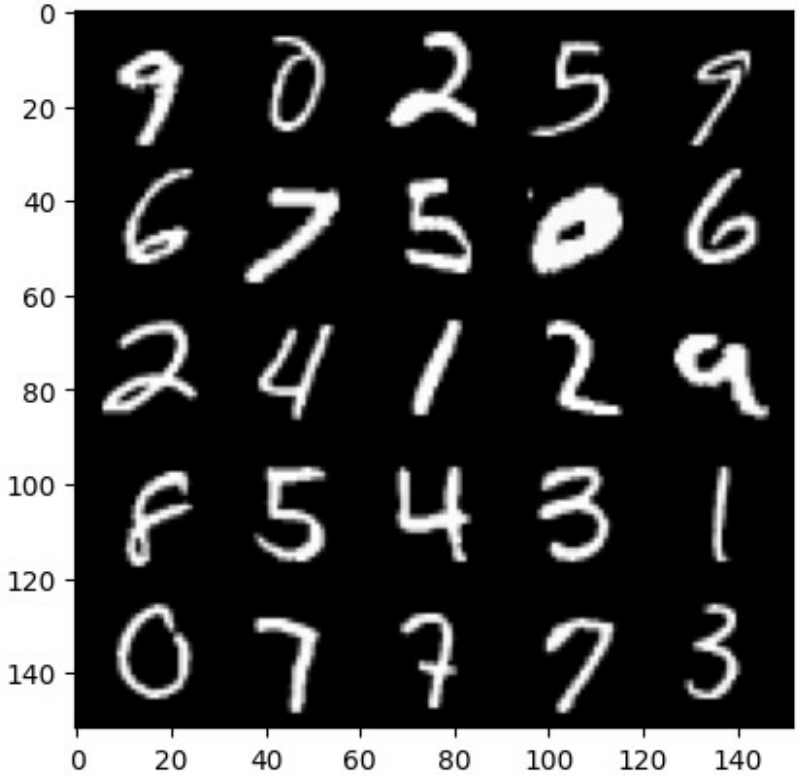




{"model_id":"8788cbece6f1484d8ae3d2662ea1f3d3","version_major":2,"version_minor":0}

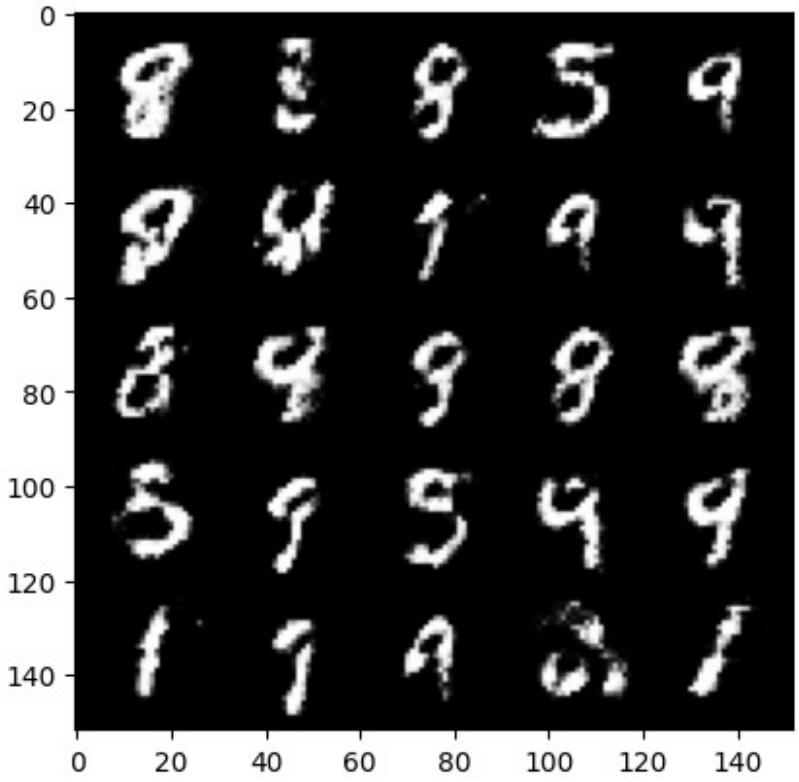
```
Epoch 140, step 66000: Generator loss: 1.537253501176833,
discriminator loss: 0.39090537780523293
```

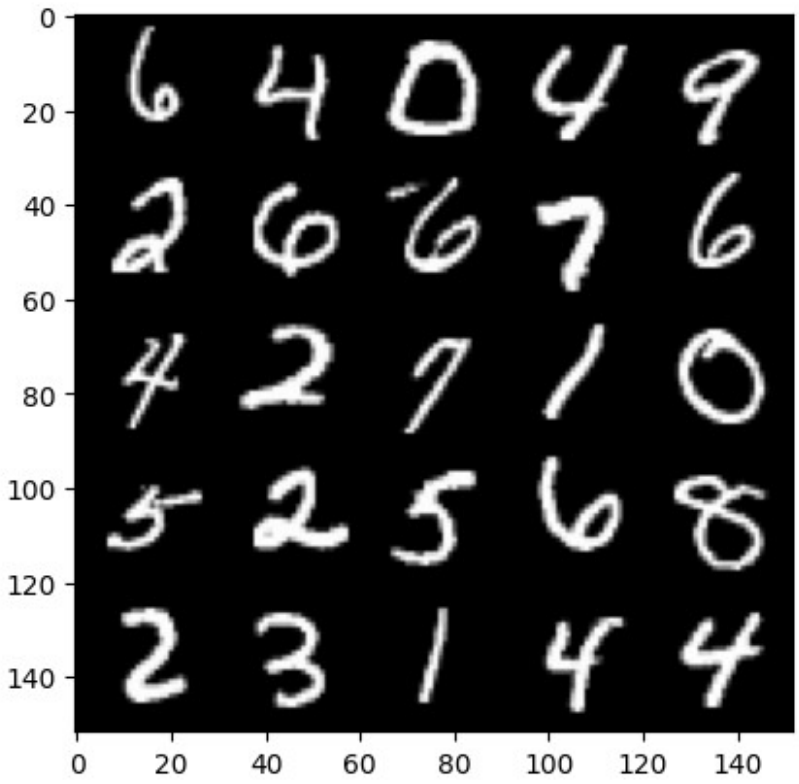

0
20
40
60
80
100
120
140
0 20 40 60 80 100 120 140


0
20
40
60
80
100
120
140
0 20 40 60 80 100 120 140

{"model_id":"55ba556c76834f51bea3cf6f7d19d56e","version_major":2,"version_minor":0}

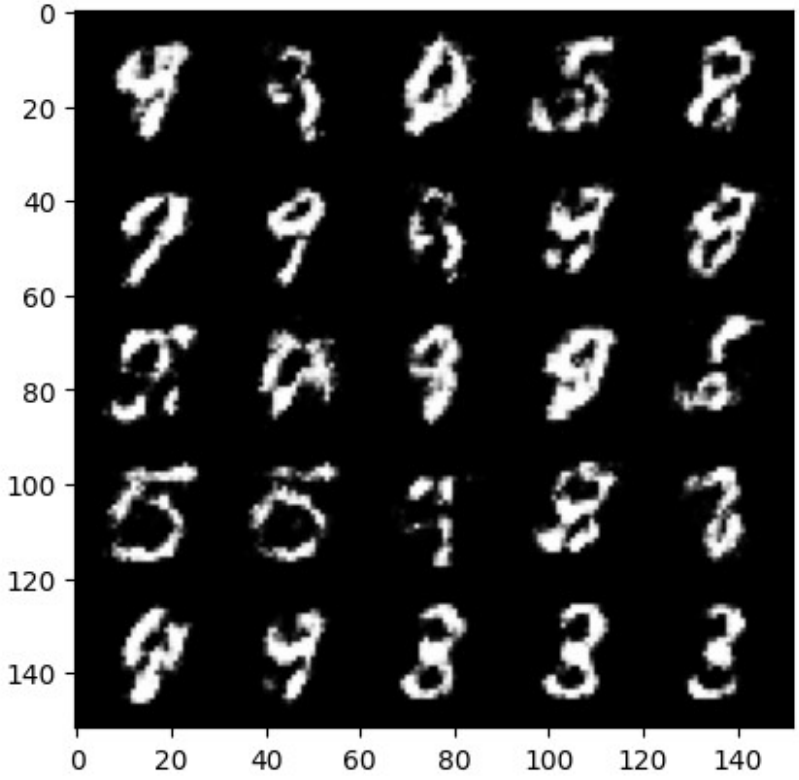
```
Epoch 141, step 66500: Generator loss: 1.4455390324592585,
discriminator loss: 0.43037462764978385
```

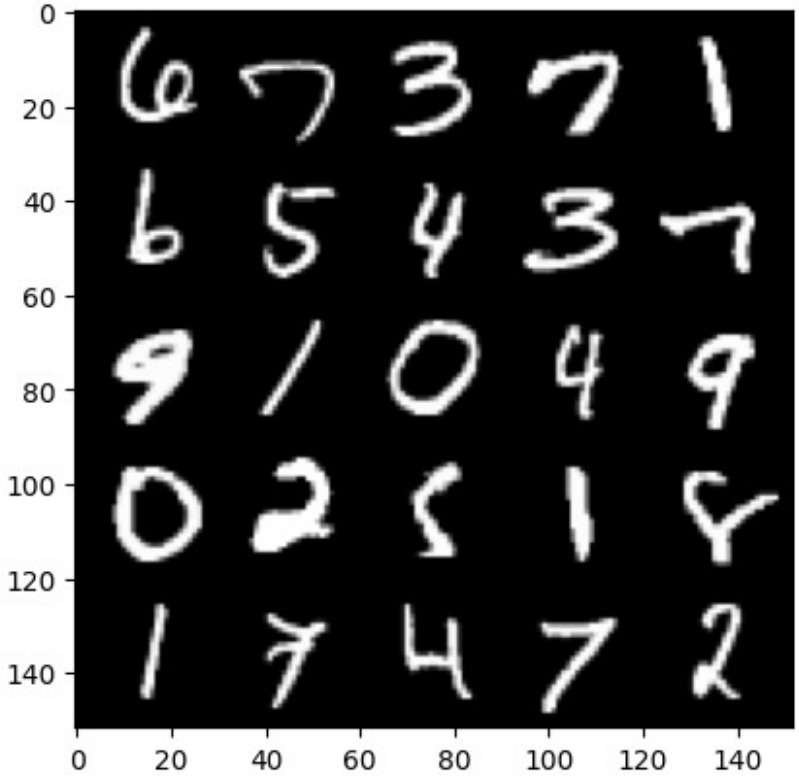




{"model_id":"57a23fe3b23a4a90b2f6716bd4dfd994","version_major":2,"version_minor":0}

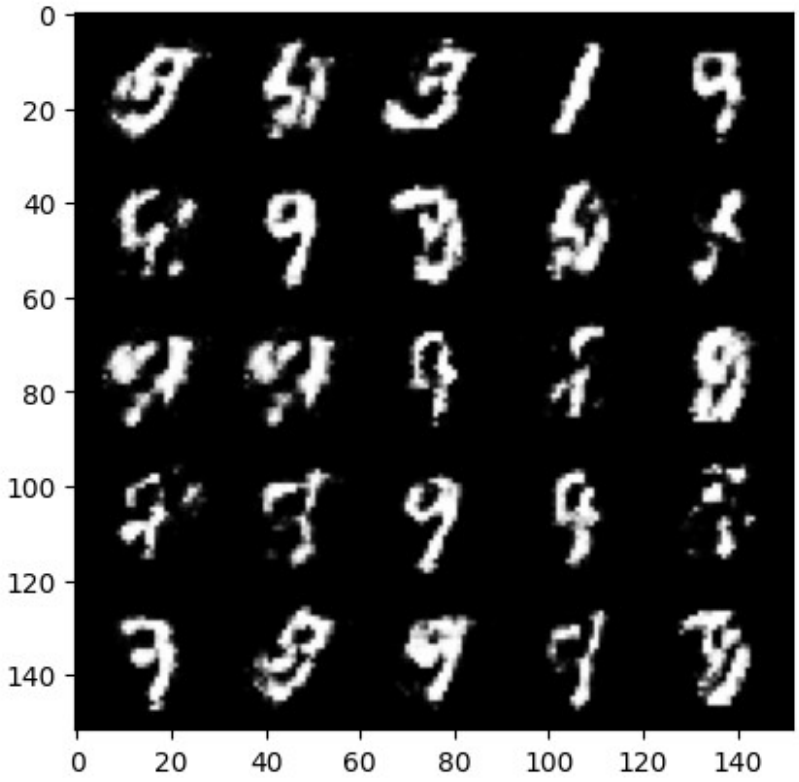
```
Epoch 142, step 67000: Generator loss: 1.4671534469127678,
discriminator loss: 0.41224610769748693
```

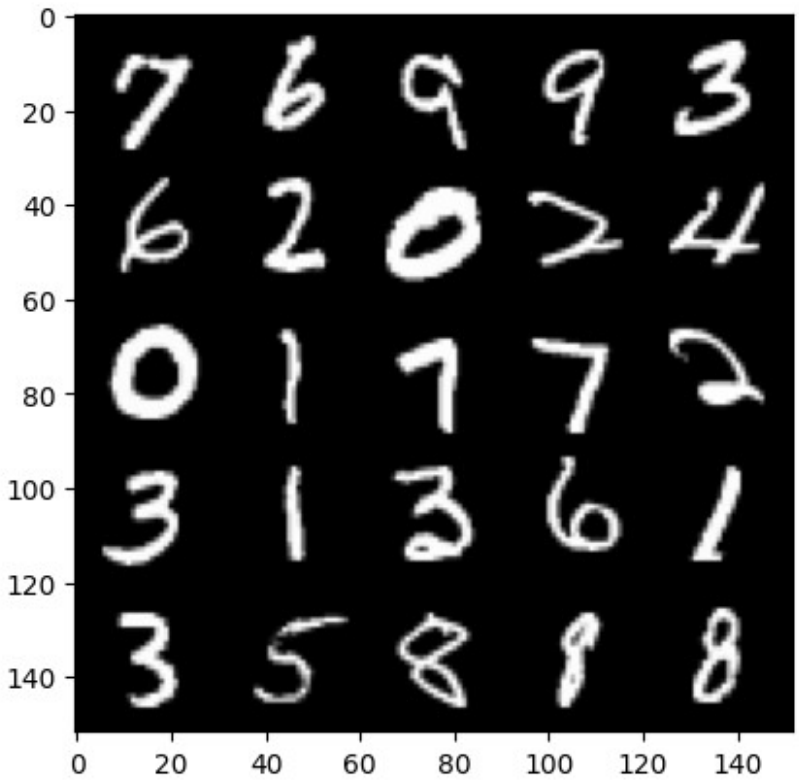

0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140


0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

{"model_id":"4d323239159a46edbb6912959d83f4f3","version_major":2,"version_minor":0}

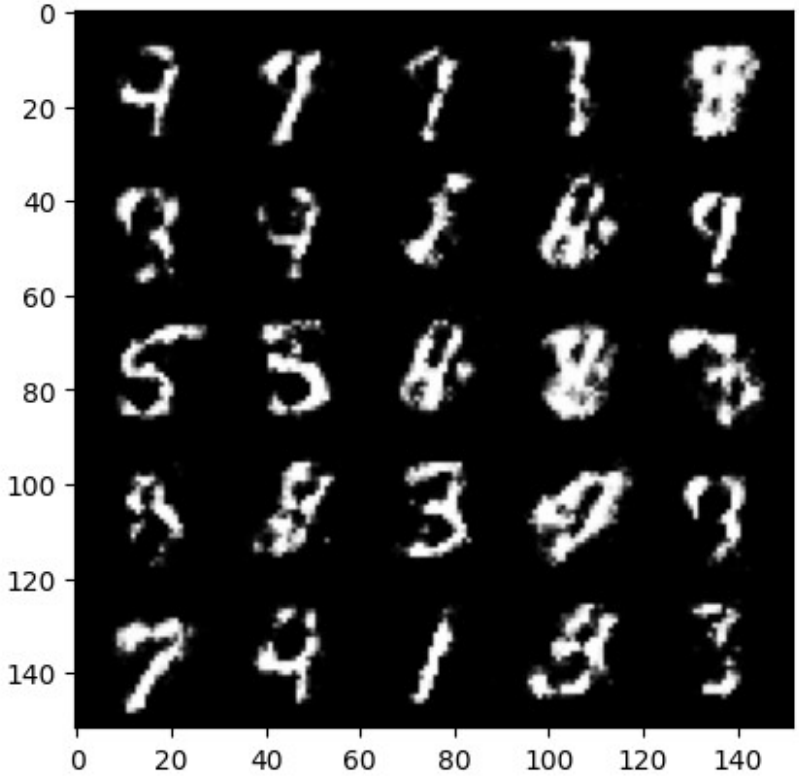
```
Epoch 143, step 67500: Generator loss: 1.3636382255554207,
discriminator loss: 0.4454119796752928
```

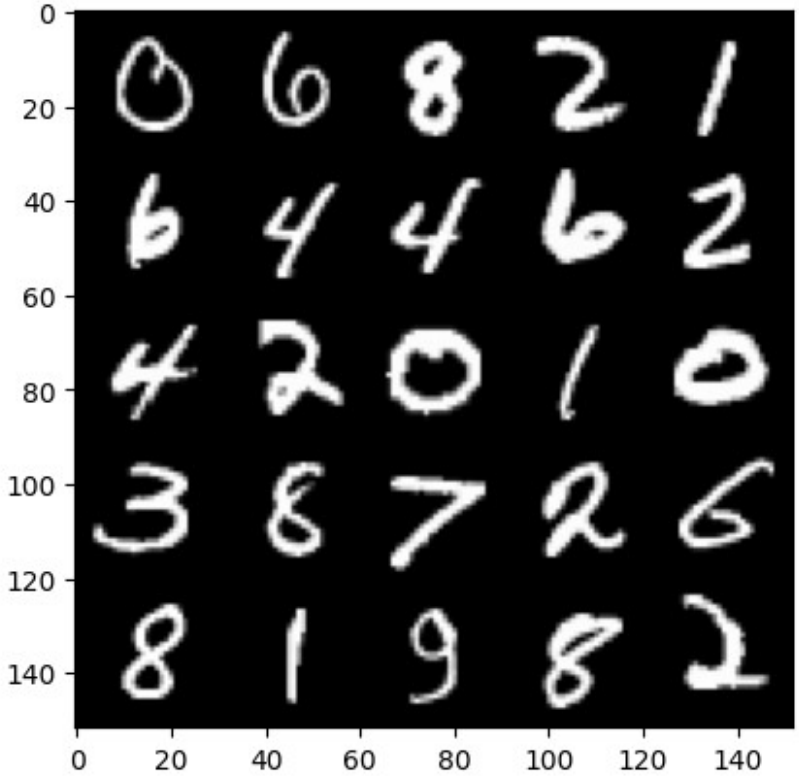




{"model_id":"f586b46121e043d3b4da67c27627bd4b","version_major":2,"version_minor":0}

```
Epoch 144, step 68000: Generator loss: 1.4057199401855474,
discriminator loss: 0.43692314267158505
```

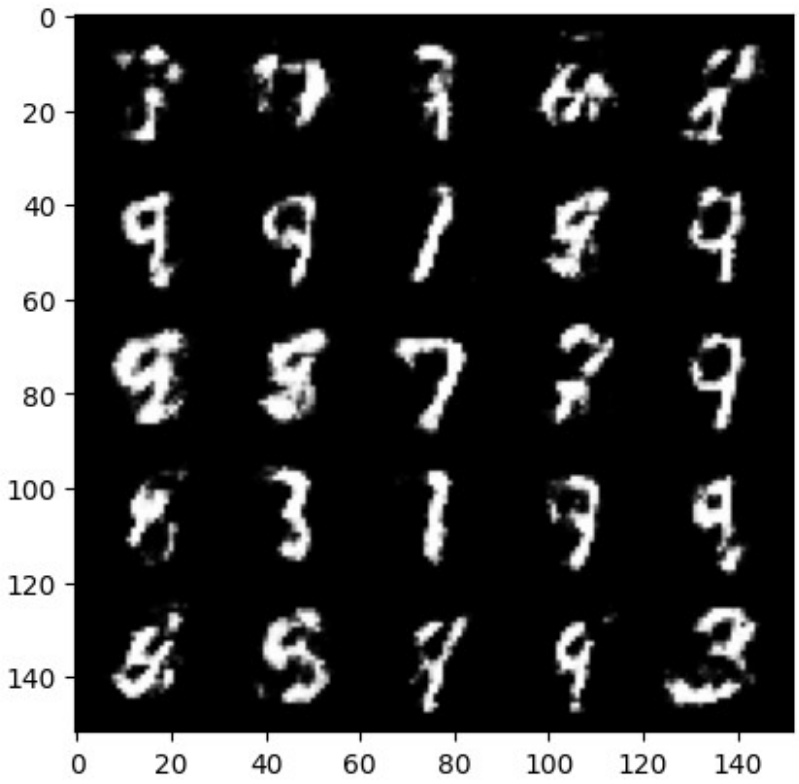

0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

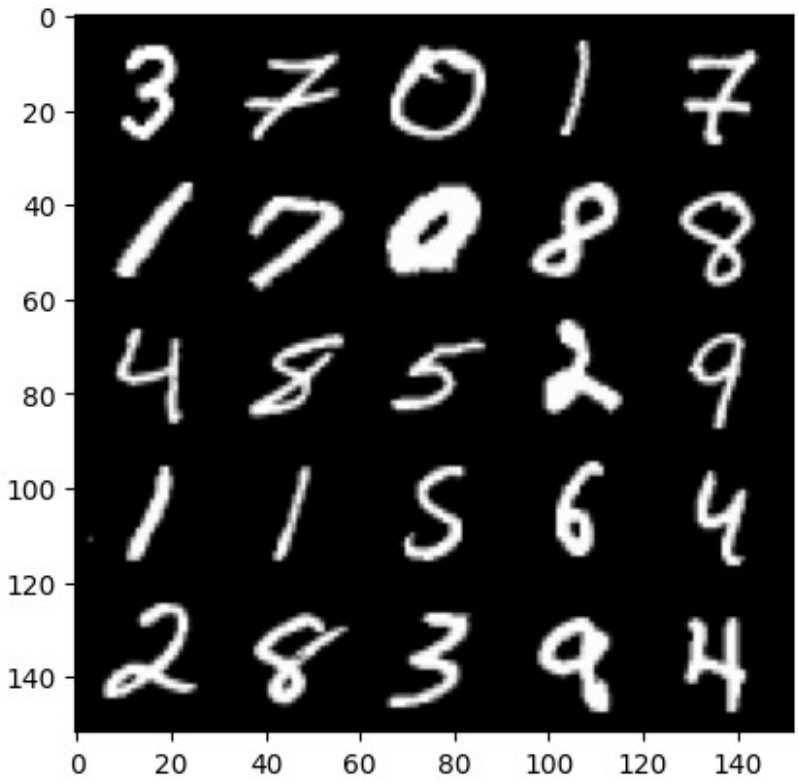

0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

{"model_id":"238e71a7f5a7439c9667af93285c592e","version_major":2,"version_minor":0}

{"model_id":"048300e1d31f4d6ab6148396056bd3e6","version_major":2,"version_minor":0}

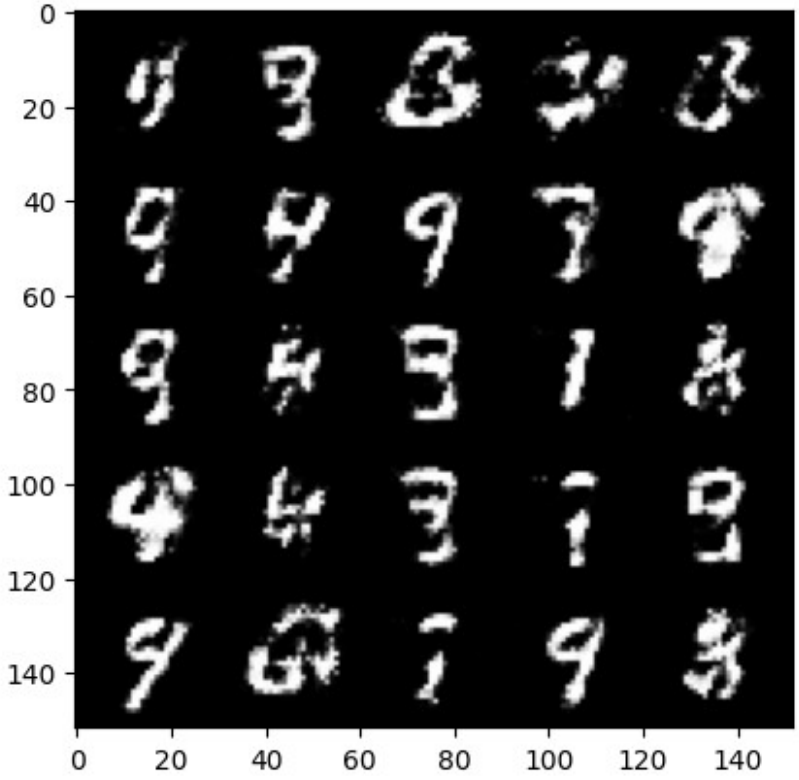
```
Epoch 146, step 68500: Generator loss: 1.4357072925567624,
discriminator loss: 0.42270983409881596
```

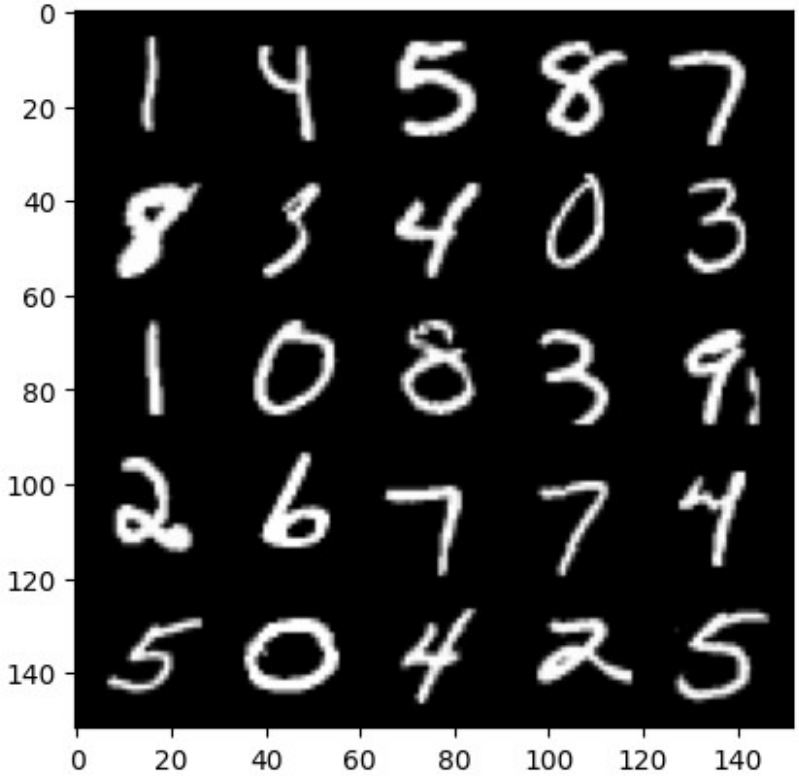




{"model_id":"15b385b1a6224b9b903c740dbea7fc7e","version_major":2,"version_minor":0}

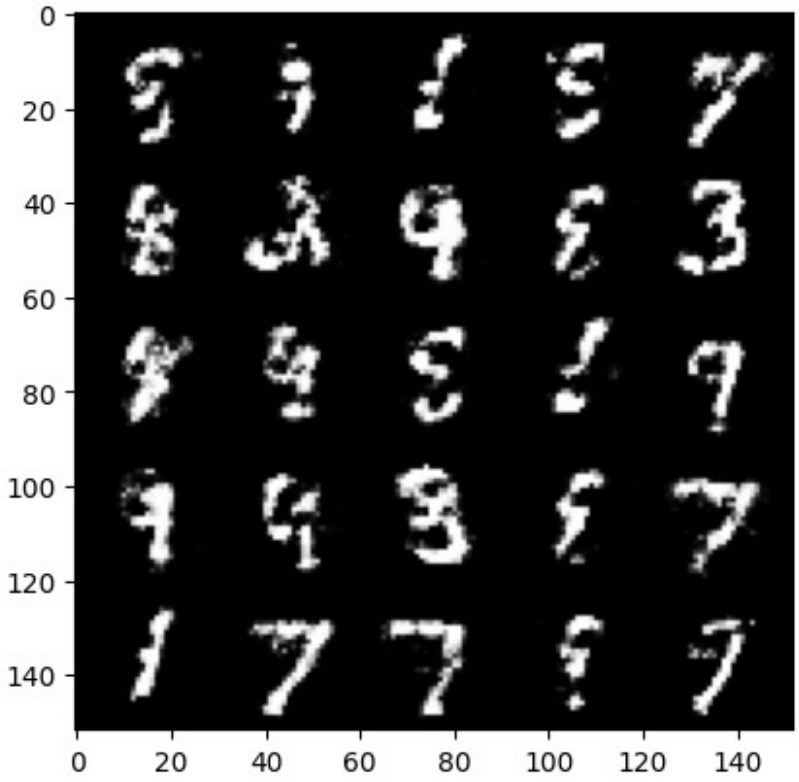
```
Epoch 147, step 69000: Generator loss: 1.4382275807857519,
discriminator loss: 0.42939499151706706
```

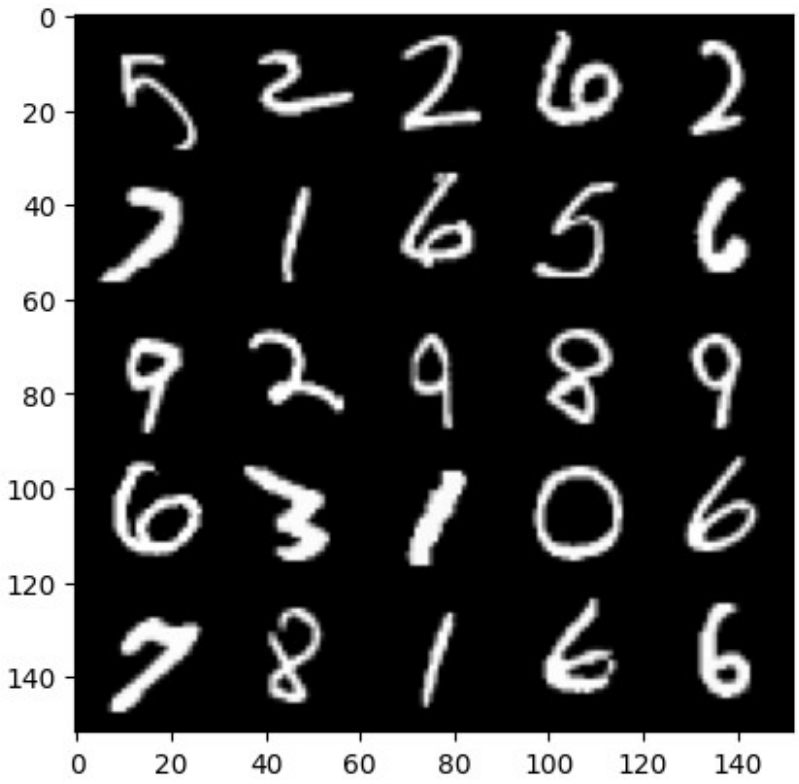

0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140


0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

{"model_id":"fb32ec03ad124bd4b9baf5c29c7c85e2","version_major":2,"version_minor":0}

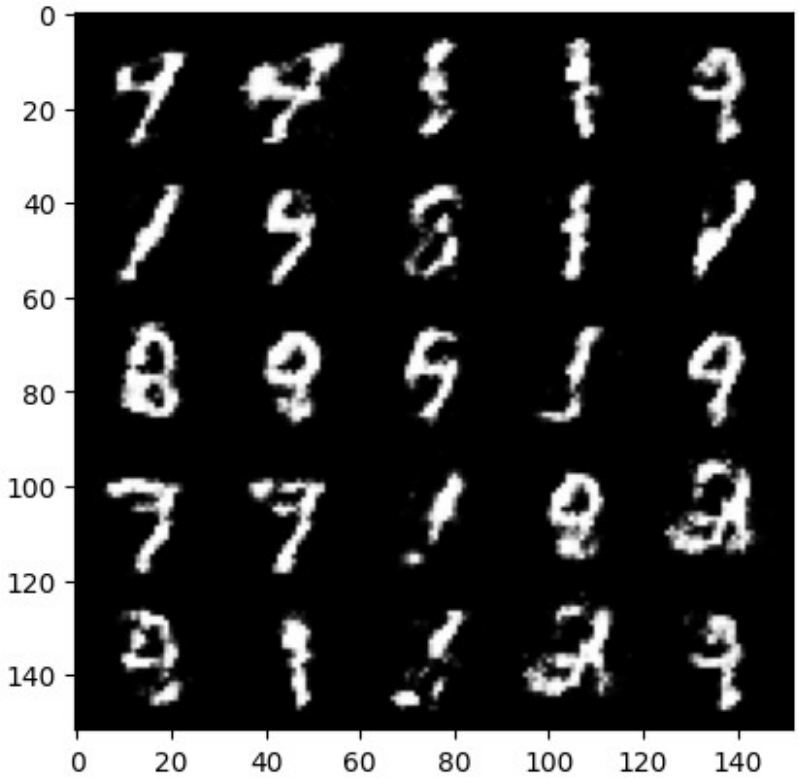
```
Epoch 148, step 69500: Generator loss: 1.3831023888587957,
discriminator loss: 0.4393476263284674
```

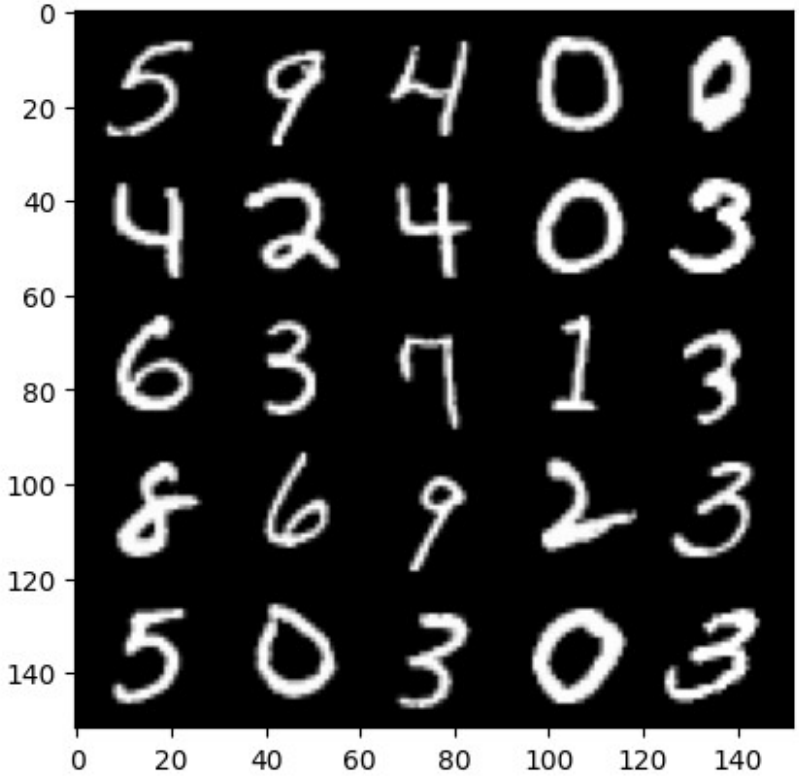




{"model_id":"0c84f4c71a9f456887708ddf0ae4c0c9","version_major":2,"version_minor":0}

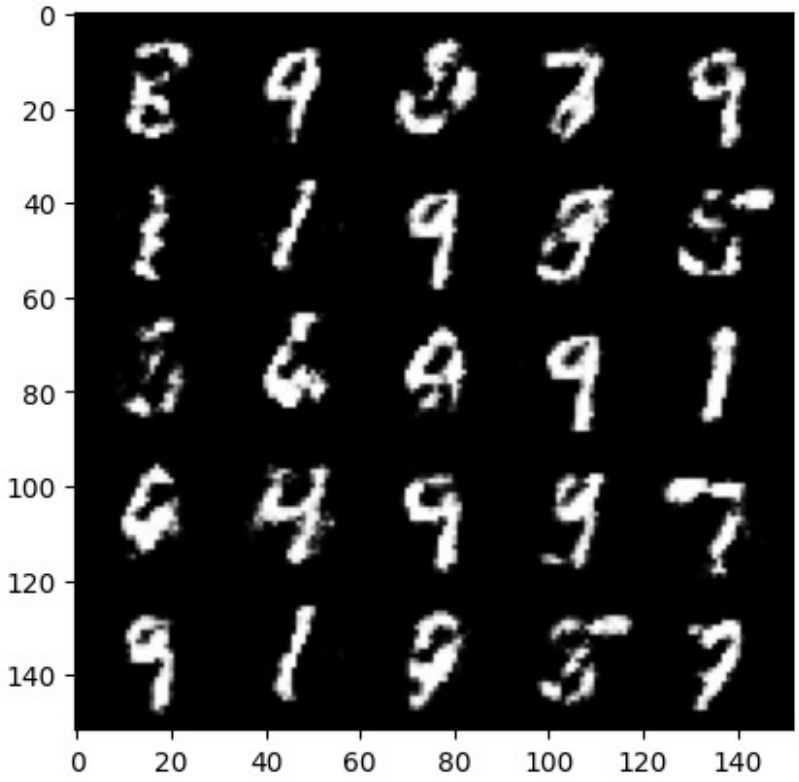
```
Epoch 149, step 70000: Generator loss: 1.4047783074378963,
discriminator loss: 0.432302458882332
```

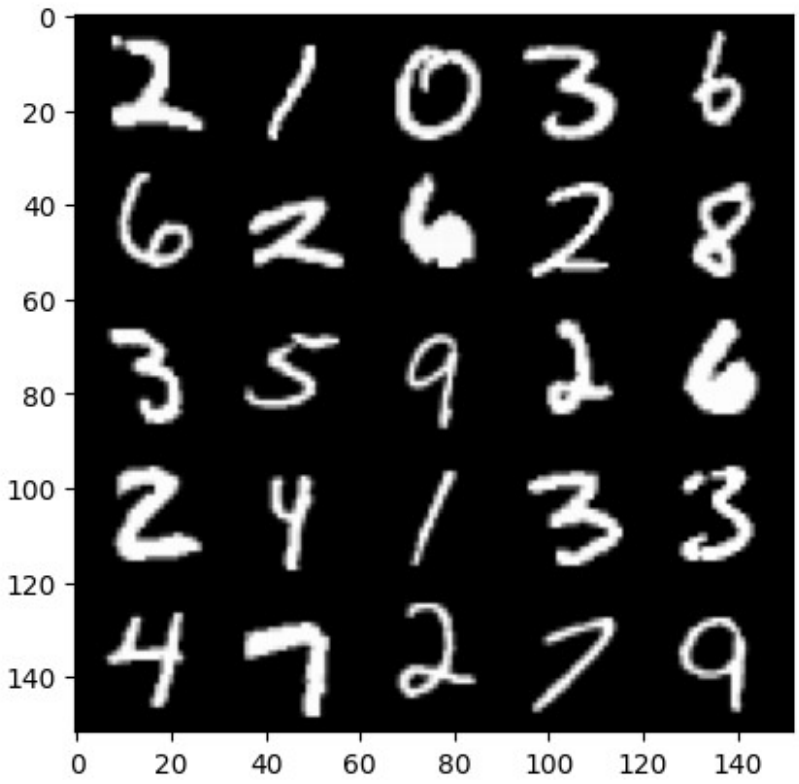

0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140


0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

{"model_id":"f074dc4653db4ee495516e1102a9ee3c","version_major":2,"version_minor":0}

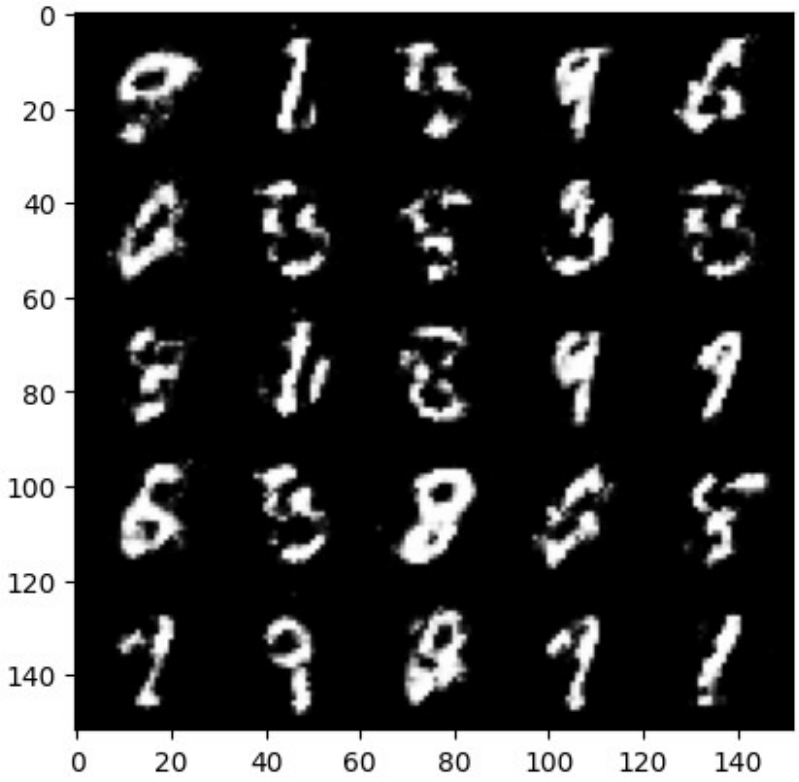
```
Epoch 150, step 70500: Generator loss: 1.4116410303115856,
discriminator loss: 0.4420718860626218
```

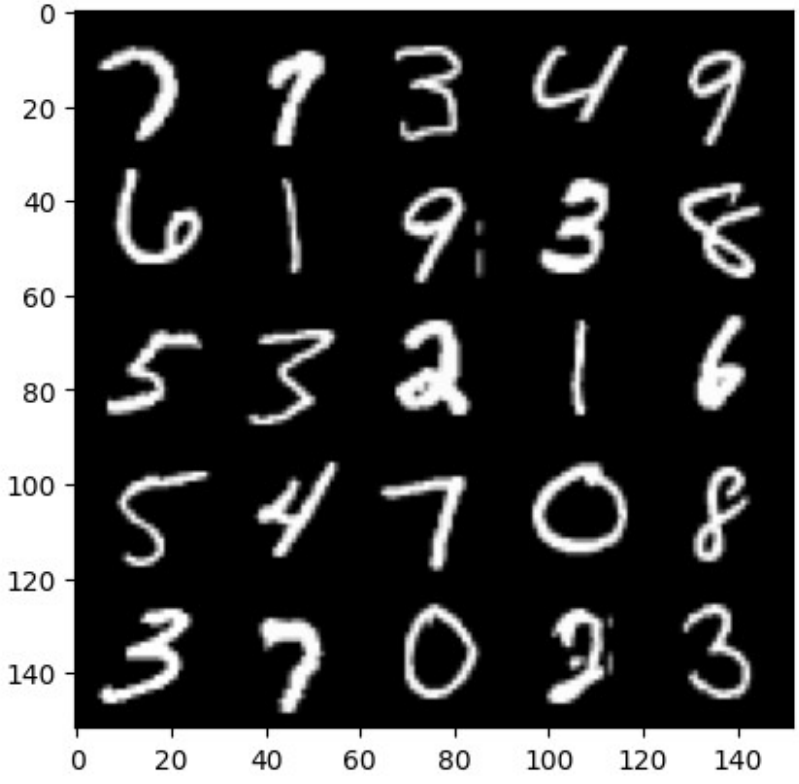




{"model_id":"2b9105c8fb7e45a1bc162a7252281e0e","version_major":2,"version_minor":0}

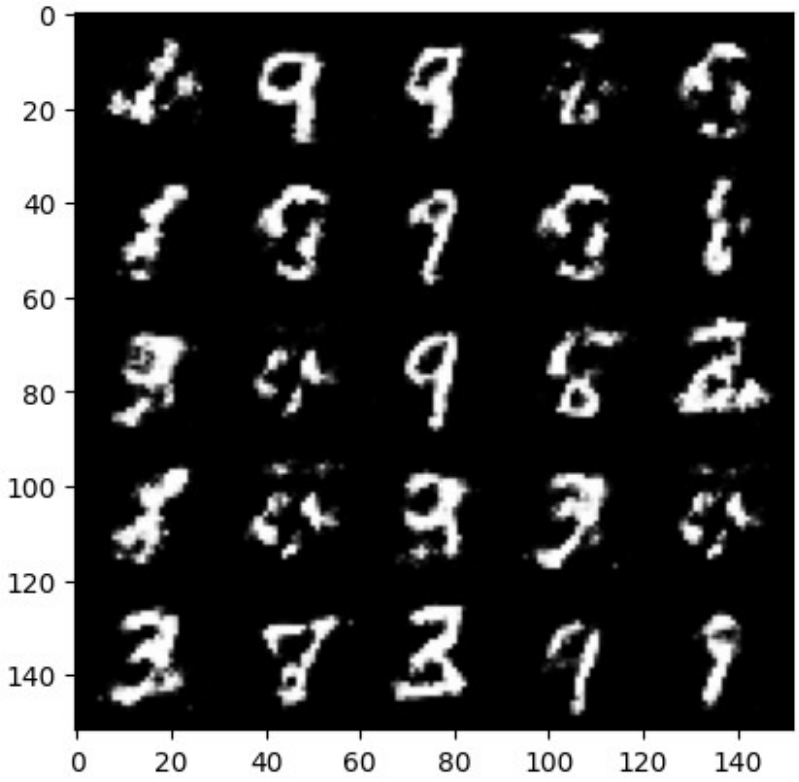
```
Epoch 151, step 71000: Generator loss: 1.3657322702407835,
discriminator loss: 0.44407184660434695
```

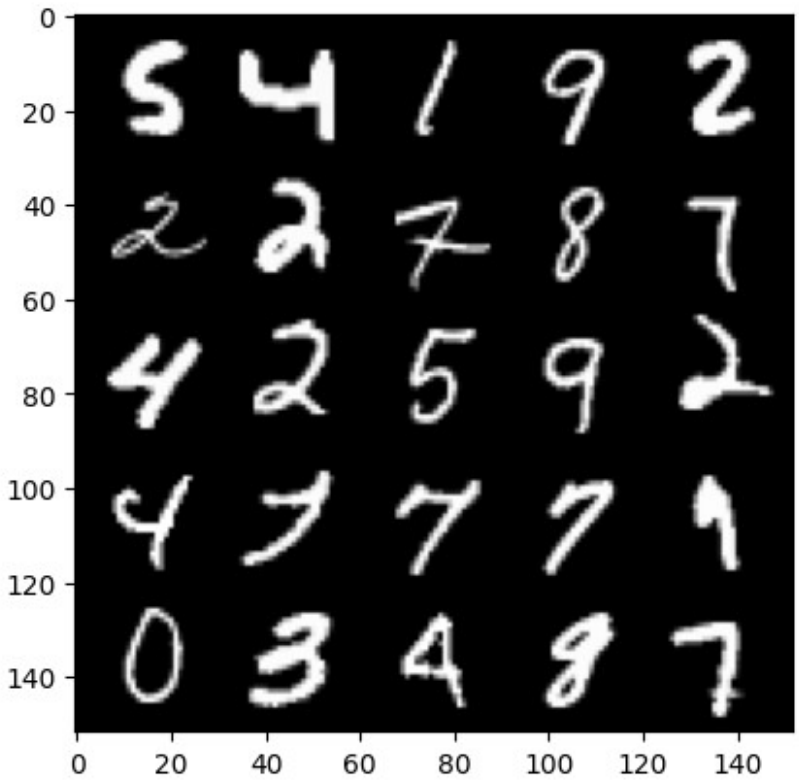

0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140


0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

{"model_id":"39a1199d9c134c2c817a7a2d04b7b0f8","version_major":2,"version_minor":0}

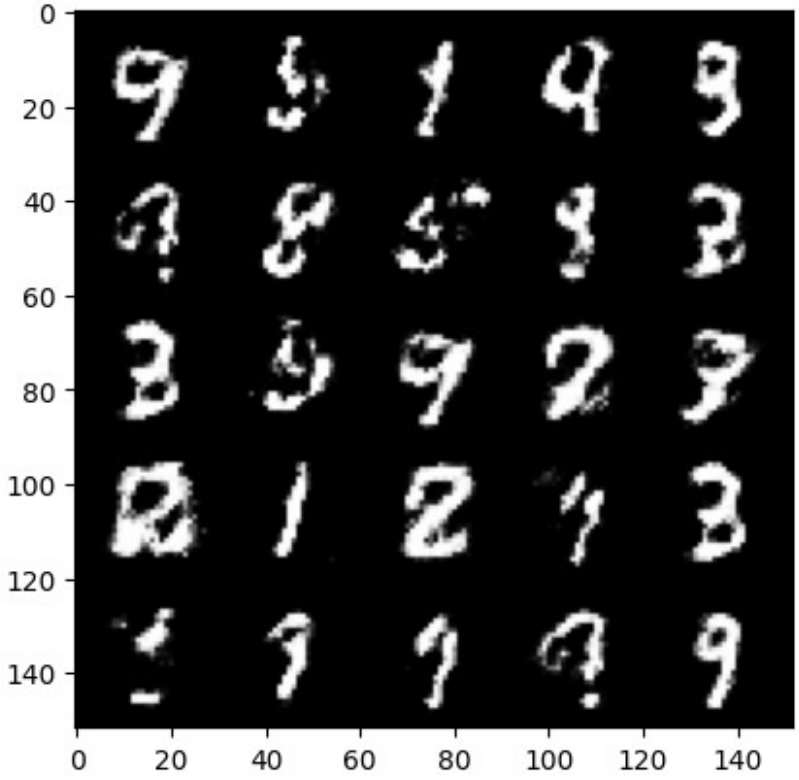
```
Epoch 152, step 71500: Generator loss: 1.3853626236915593,
discriminator loss: 0.43572284263372424
```

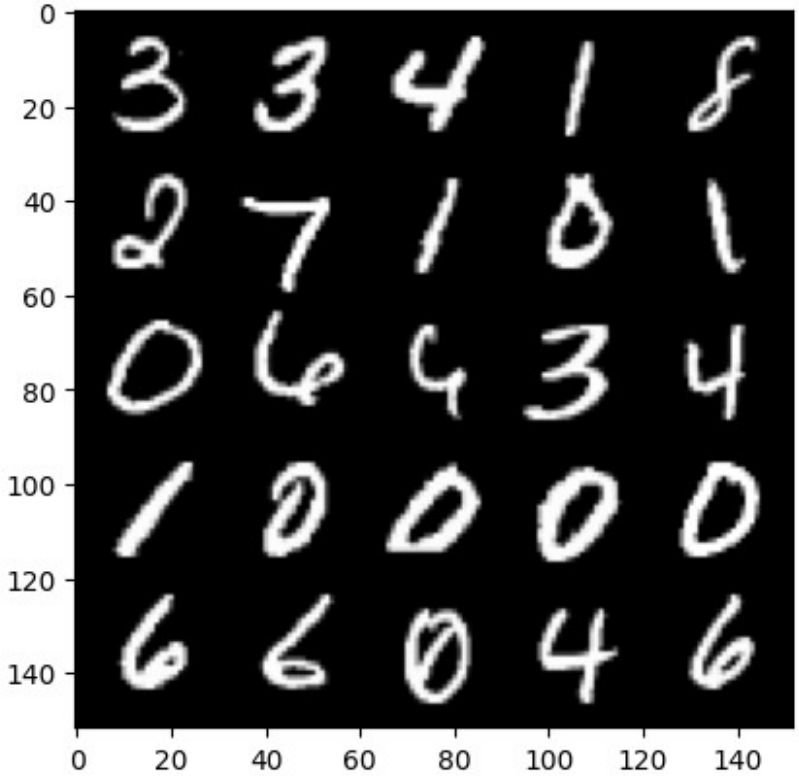




{"model_id":"b59afc44ede64e70b3dff13ae7fb70c0","version_major":2,"version_minor":0}

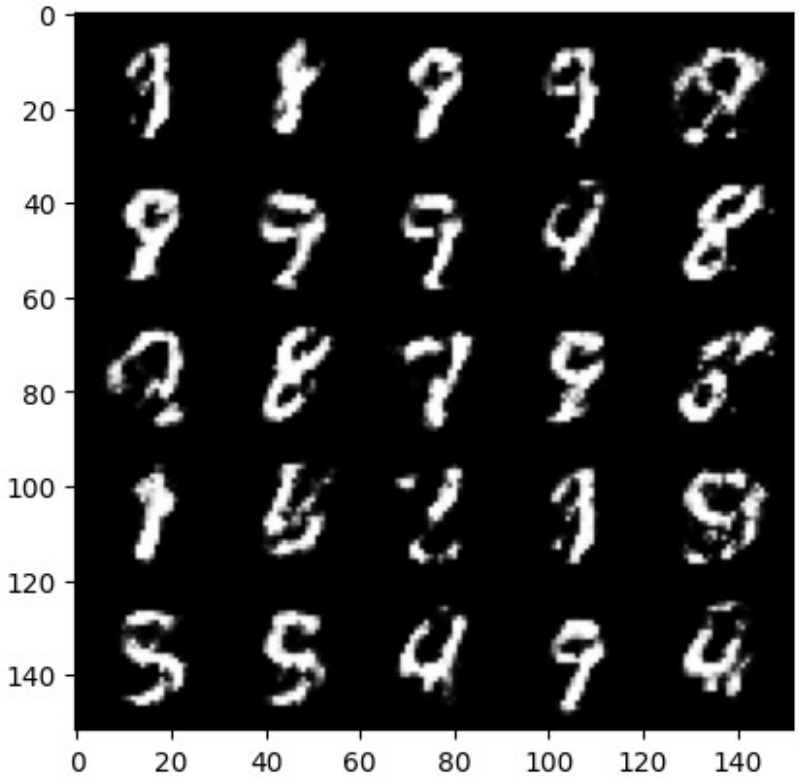
```
Epoch 153, step 72000: Generator loss: 1.4304361550807965,
discriminator loss: 0.42983599877357465
```

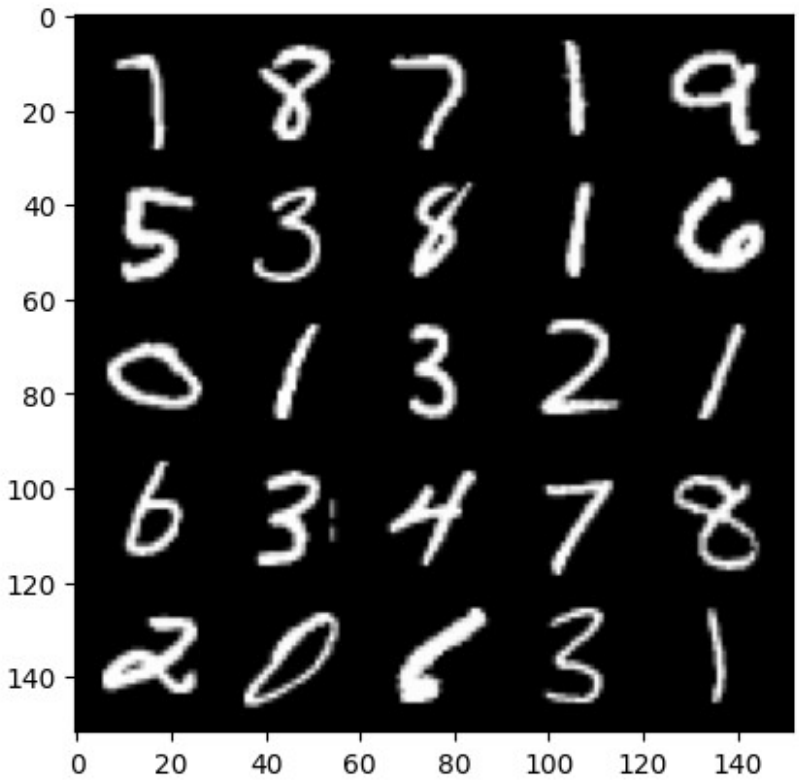

0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140


0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

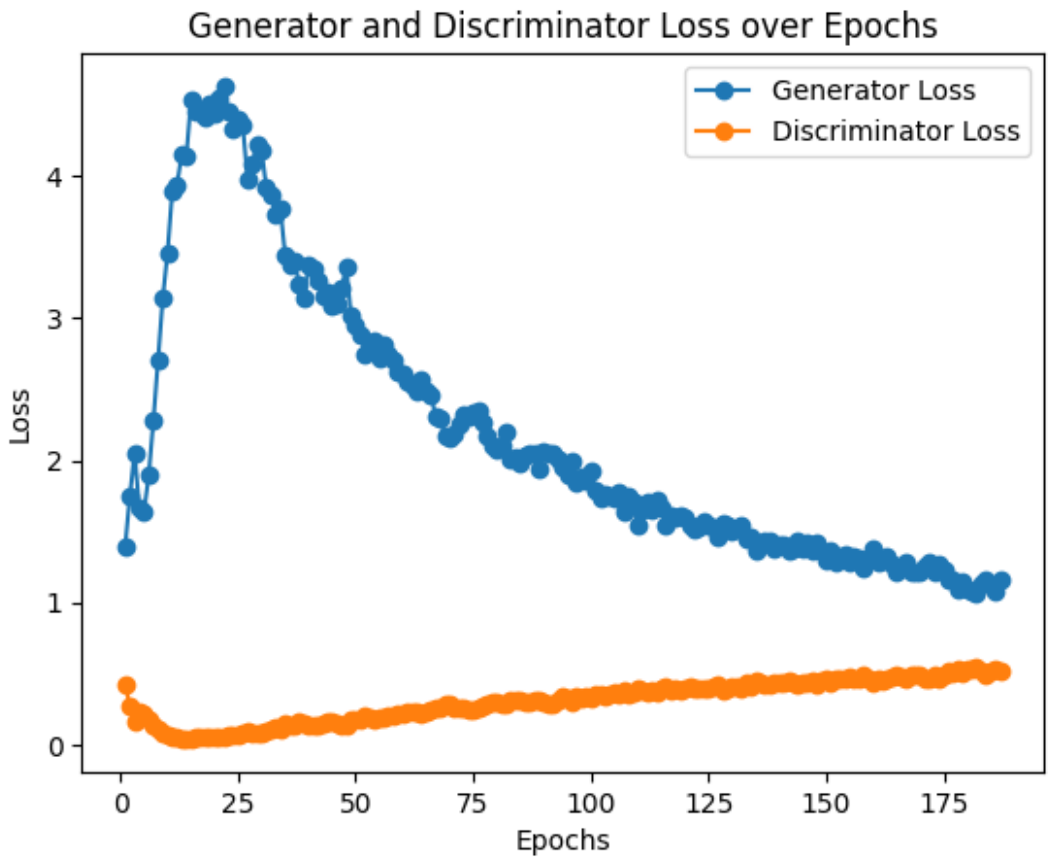
{"model_id":"6ea1d87231444a9ba7b408332ef6e34b","version_major":2,"version_minor":0}

```
Epoch 154, step 72500: Generator loss: 1.3809303138256075,
discriminator loss: 0.43183136492967616
```






```
import matplotlib.pyplot as plt

# Generate x-axis values (epochs, for example)
epochs = range(1, len(genLoss) + 1)

# Plotting
plt.plot(epochs, genLoss, label='Generator Loss', marker='o')
plt.plot(epochs, disLoss, label='Discriminator Loss', marker='o')

# Adding labels and title
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Generator and Discriminator Loss over Epochs')

# Adding legend
plt.legend()

# Display the plot
plt.show()
```



Observation from the above plot, as we see the generator and discriminator model plays a zero sum game. At starting, we see that the generators loss is very high, moving on with epochs, the generator try to reporoduce the images with less noise, generator's loss and discrimanator's loss became almost same.