In [ ]:
```
"""
Deep Learning Assignment 3
Conditional GAN Skeleton Code.
Adopted from public sources, customized and improved for this assignment.
"""

#import necessary modules
import torch
import torch.nn as nn
from torchvision import transforms, datasets
from torch import optim as optim
# for visualization
from matplotlib import pyplot as plt
import math
import numpy as np
```

In [ ]:
```
# tells PyTorch to use an NVIDIA GPU, if one is available.
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# loading the dataset
training_parameters = {
    "img_size": 28,
    "n_epochs": 24, #24
    "batch_size": 64,
    "learning_rate_generator": 0.0002,
    "learning_rate_discriminator": 0.0002,
}
# define a transform to 1) scale the images and 2) convert them into tensors
transform = transforms.Compose([
    transforms.Resize(training_parameters['img_size']), # scales the smaller
    transforms.ToTensor(),
])

# load the dataset
train_loader = torch.utils.data.DataLoader(
    datasets.FashionMNIST(
        './data', # specifies the directory to download the datafiles to, re
        train = True,
        download = True,
        transform = transform),
    batch_size = training_parameters["batch_size"],
    shuffle=True
    )

# Fashion MNIST has 10 classes, just like MNIST. Here's what they correspond
label_descriptions = {
      0: 'T-shirt/top',
      1 : 'Trouser',
      2 : 'Pullover',
      3 : 'Dress',
      4 : 'Coat',
      5 : 'Sandal',
      6 : 'Shirt',
      7 : 'Sneaker',
      8 : 'Bag',
      9 : 'Ankle boot'
}
```

In [ ]:
```
# Create the Generator model class, which will be used to initialize the gen
class Generator(nn.Module):
  def __init__(self, input_dim, output_dim, num_labels=10): # to initialize
```

```
    super(Generator,self).__init__() # initialize the parent class
    # TODO (5.4) Turn this Generator into a Conditional Generator by
    # 1. Adjusting the input dimension of the first hidden layer.
    # 2. Modifying the input to the first hidden layer in the forward class.

    self.label_embedding = nn.Embedding(10, 10) # This function will be usef
    self.hidden_layer1 = nn.Sequential(
        nn.Linear(input_dim+10, 256),
        nn.LeakyReLU(0.2)
    )
    self.hidden_layer2 = nn.Sequential(
        nn.Linear(256, 512),
        nn.LeakyReLU(0.2)
    )
    self.hidden_layer3 = nn.Sequential(
        nn.Linear(512, 1024),
        nn.LeakyReLU(0.2)
    )
    self.hidden_layer4 = nn.Sequential(
        nn.Linear(1024, output_dim),
        nn.Tanh()
    )
  def forward(self, x, labels):
      c = self.label_embedding(labels)
      x = torch.cat([x, c], 1)

      output = self.hidden_layer1(x)
      output = self.hidden_layer2(output)
      output = self.hidden_layer3(output)
      output = self.hidden_layer4(output)
      return output.to(device)
```

In [ ]:

```
class Discriminator(nn.Module):
    def __init__(self, input_dim, output_dim=1, num_labels=None):
        super(Discriminator, self).__init__()

        self.label_embedding = nn.Embedding(10, 10)
        # TODO (5.4) Modify this discriminator to function as a conditional
        self.hidden_layer1 = nn.Sequential(
            nn.Linear(input_dim+10, 1024),
            nn.LeakyReLU(0.2),
            nn.Dropout(0.3)
        )

        self.hidden_layer2 = nn.Sequential(
            nn.Linear(1024, 512),
            nn.LeakyReLU(0.2),
            nn.Dropout(0.3)
        )

        self.hidden_layer3 = nn.Sequential(
            nn.Linear(512, 256),
            nn.LeakyReLU(0.2),
            nn.Dropout(0.3)
        )

        self.hidden_layer4 = nn.Sequential(
            nn.Linear(256, output_dim),
            nn.Sigmoid()
        )

    def forward(self, x, labels=None): # labels to be used in 5.4.
        c = self.label_embedding(labels)
        x = torch.cat([x, c], 1)
```

```
        output = self.hidden_layer1(x)
        output = self.hidden_layer2(output)
        output = self.hidden_layer3(output)
        output = self.hidden_layer4(output)
        return output.to(device)
```

In [ ]:
```
discriminator = Discriminator(784,1).to(device) # initialize both models, an
generator = Generator(100,784).to(device)

discriminator_optimizer = optim.Adam(discriminator.parameters(), lr=training
generator_optimizer = optim.Adam(generator.parameters(), lr=training_paramet
```

In [ ]:
```
# Establish convention for real and fake labels during training
real_label = 1.
fake_label = 0.
```

In [ ]:
```
#Loss_D - discriminator loss calculated as the sum of losses for the all rea

loss_func = nn.BCELoss() # Binary Cross Entropy Loss
def train_generator(batch_size):
    """
    Performs a training step on the generator by
        1. Generating fake images from random noise.
        2. Running the discriminator on the fake images.
        3. Computing loss on the result.
    :arg batch_size: the number of training examples in the current batch
    Returns the average generator loss over the batch.
    """

    # Start by zeroing the gradients of the optimizer
    generator_optimizer.zero_grad()
    # 1. Create a new batch of fake images (since the discriminator has just
    noise = torch.randn(batch_size,100).to(device) # whenever you create new
    generated_labels = torch.randint(0, 10, (batch_size,)).to(device)
    generator_output = generator(noise, labels = generated_labels)
    # 2. Run the discriminator on the fake images
    discriminator_output = discriminator(generator_output, labels = generate
    ###----copied----
    real_label_vector = torch.full((batch_size,), real_label, dtype=torch.fl
    real_label_vector = real_label_vector.view(-1, 1)
    #-------
    # 3. Compute the loss
    loss = loss_func(discriminator_output, real_label_vector)
    loss.backward()
    generator_optimizer.step()

    loss = loss.mean().item()
    return loss

def train_discriminator(batch_size, images, labels=None): # labels to be use
    """
    Performs a training step on the discriminator by
        1. Generating fake images from random noise.
        2. Running the discriminator on the fake images.
        3. Running the discriminator on the real images
        3. Computing loss on the results.
    :arg batch_size: the number of training examples in the current batch
    :arg images: the current batch of images, a tensor of size BATCH x 1 x 6
    :arg labels: the labels corresponding to images, a tensor of size BATCH
    Returns the average loss over the batch.
    """
```

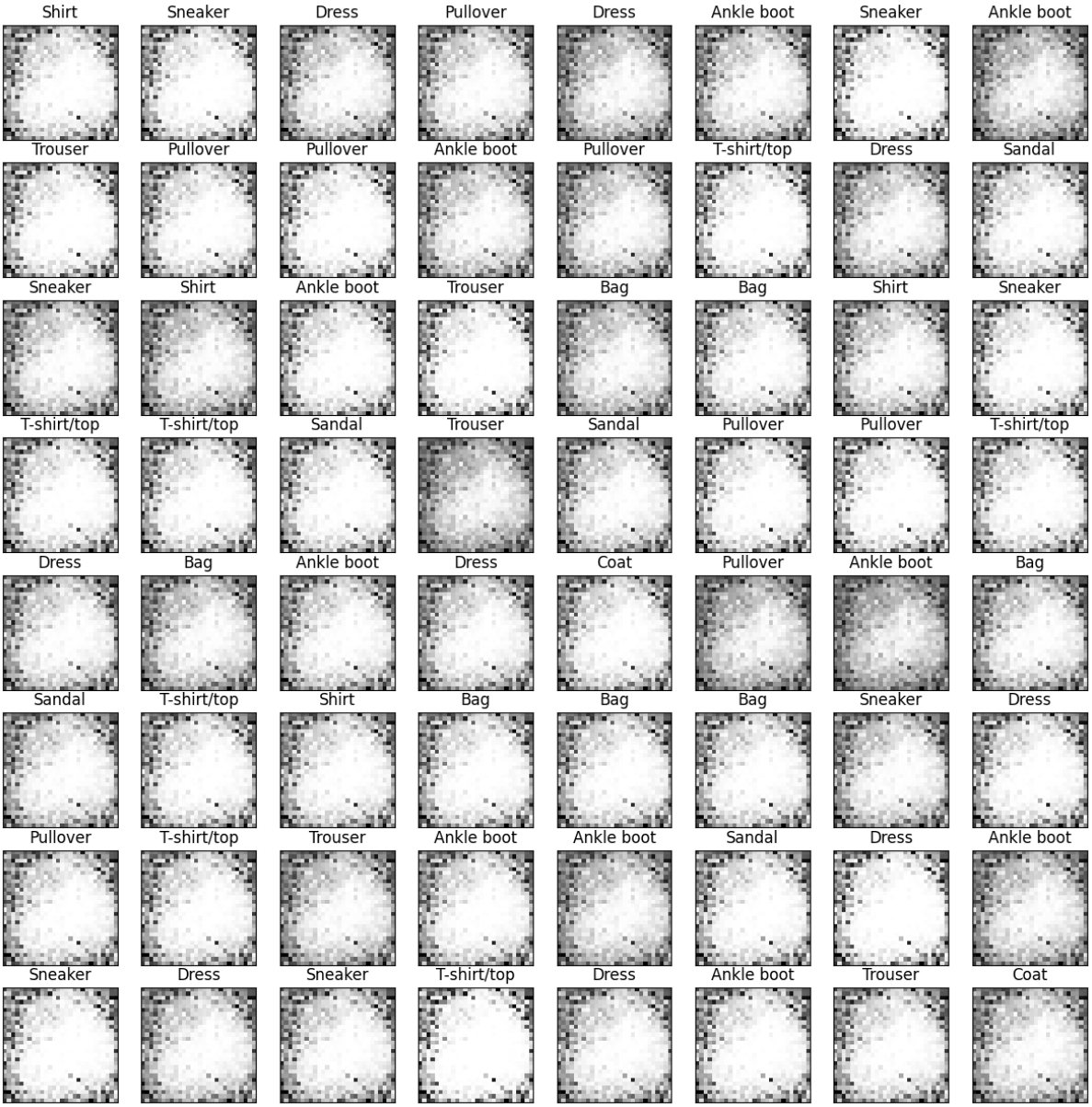
```
    discriminator_optimizer.zero_grad()
    ###----fake images----###
    # 1. Create a new batch of fake images (since the discriminator has just
    noise = torch.randn(batch_size,100).to(device) # whenever you create new
    generated_labels = torch.randint(0, 10, (batch_size,)).to(device)
    generator_output = generator(noise, labels = generated_labels)
    # 2. Run the discriminator on the fake images
    discriminator_output = discriminator(generator_output, labels = generate
    # 3. Compute the loss
    fake_label_vector = torch.full((batch_size,), fake_label, dtype=torch.fl
    fake_label_vector = fake_label_vector.view(-1, 1)
    loss_fake = loss_func(discriminator_output, fake_label_vector)

    ###----real images----###
    # 1. Run the discriminator on the real images
    images = torch.flatten(images, start_dim=1)
    discriminator_output = discriminator(images, labels = labels)
    # 2. Compute the loss
    real_label_vector = torch.full((batch_size,), real_label, dtype=torch.fl
    real_label_vector = real_label_vector.view(-1, 1)
    loss_real = loss_func(discriminator_output, real_label_vector)

    #combine losses
    loss = loss_real + loss_fake
    loss.backward()
    discriminator_optimizer.step()

    loss = loss.mean().item()
    return loss
```

In [ ]:

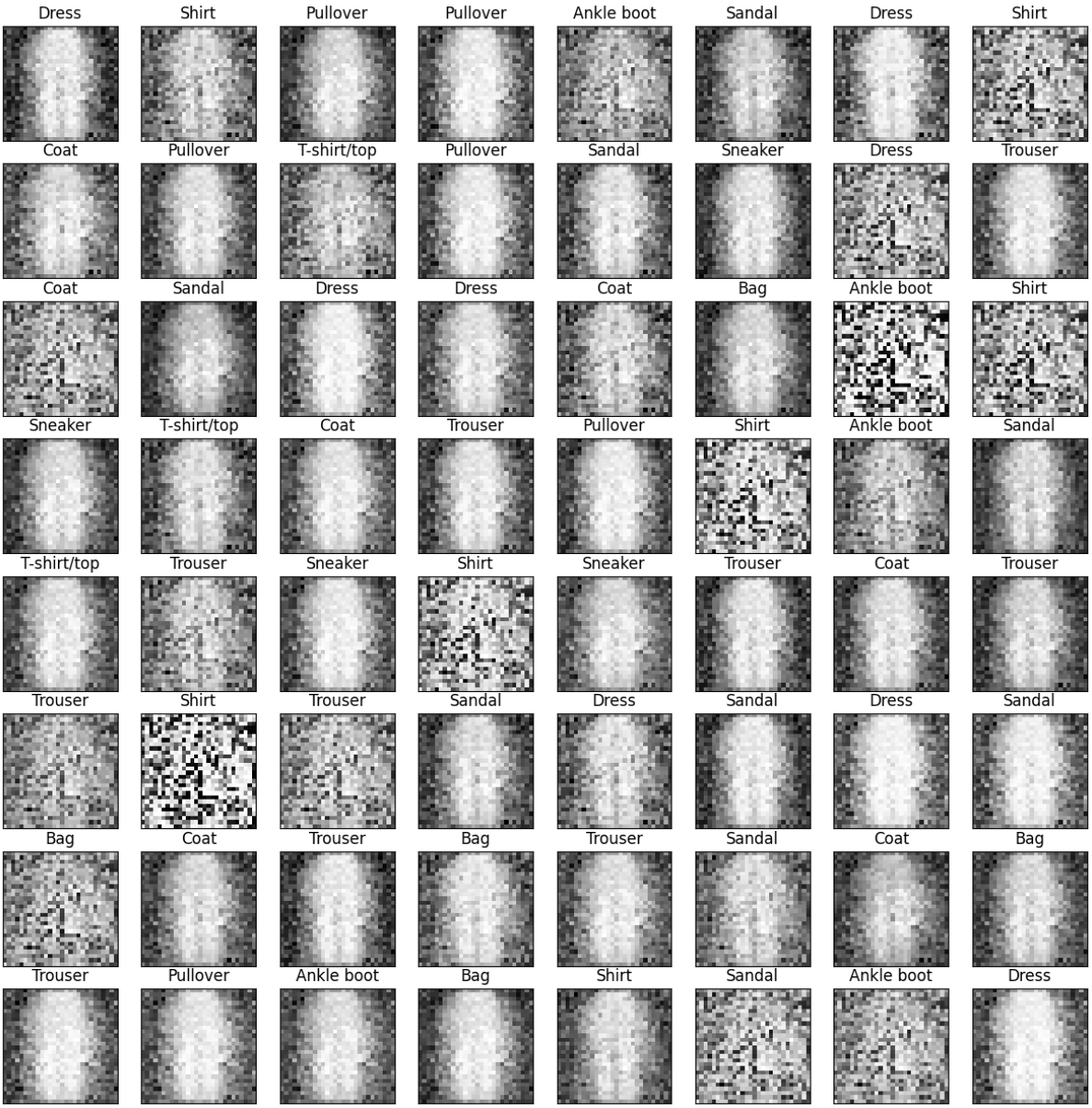
```
for epoch in range(training_parameters['n_epochs']):
    G_loss = []  # for plotting the losses over time
    D_loss = []
    for batch, (imgs, labels) in enumerate(train_loader):
        batch_size = labels.shape[0]  # if the batch size doesn't evenly div
        #generator first training
        lossG = train_generator(batch_size)
        G_loss.append(lossG)
        #single discriminator training
        lossD = train_discriminator(batch_size, imgs, labels)
        D_loss.append(lossD)
        #generator second training
        lossG = train_generator(batch_size)
        G_loss.append(lossG)

        if ((batch + 1) % 500 == 0 and (epoch + 1) % 1 == 0):
            # Display a batch of generated images and print the loss
            print("Training Steps Completed: ", batch)
            with torch.no_grad():  # disables gradient computation to speed
                noise = torch.randn(batch_size, 100).to(device)
                fake_labels = torch.randint(0, 10, (batch_size,)).to(device)
                generated_data = generator(noise, fake_labels).cpu().view(ba

                # display generated images
                batch_sqrt = int(training_parameters['batch_size'] ** 0.5)
                fig, ax = plt.subplots(batch_sqrt, batch_sqrt, figsize=(15,
                for i, x in enumerate(generated_data):
                    ax[math.floor(i / batch_sqrt)][i % batch_sqrt].set_title
                    ax[math.floor(i / batch_sqrt)][i % batch_sqrt].imshow(x.
                    ax[math.floor(i / batch_sqrt)][i % batch_sqrt].get_xaxis
                    ax[math.floor(i / batch_sqrt)][i % batch_sqrt].get_yaxis
                plt.show()
                #fig.savefig(f"./results/CGAN_Generations_Epoch_{epoch}")
                #fig.savefig(f"pset/pset3/results/CGAN_Generations_Epoch_{ep
```

```
                fig.savefig(f"CGAN_Generations_Epoch_{epoch}")
                print(
                    f"Epoch {epoch}: loss_d: {torch.mean(torch.FloatTensor(D
```

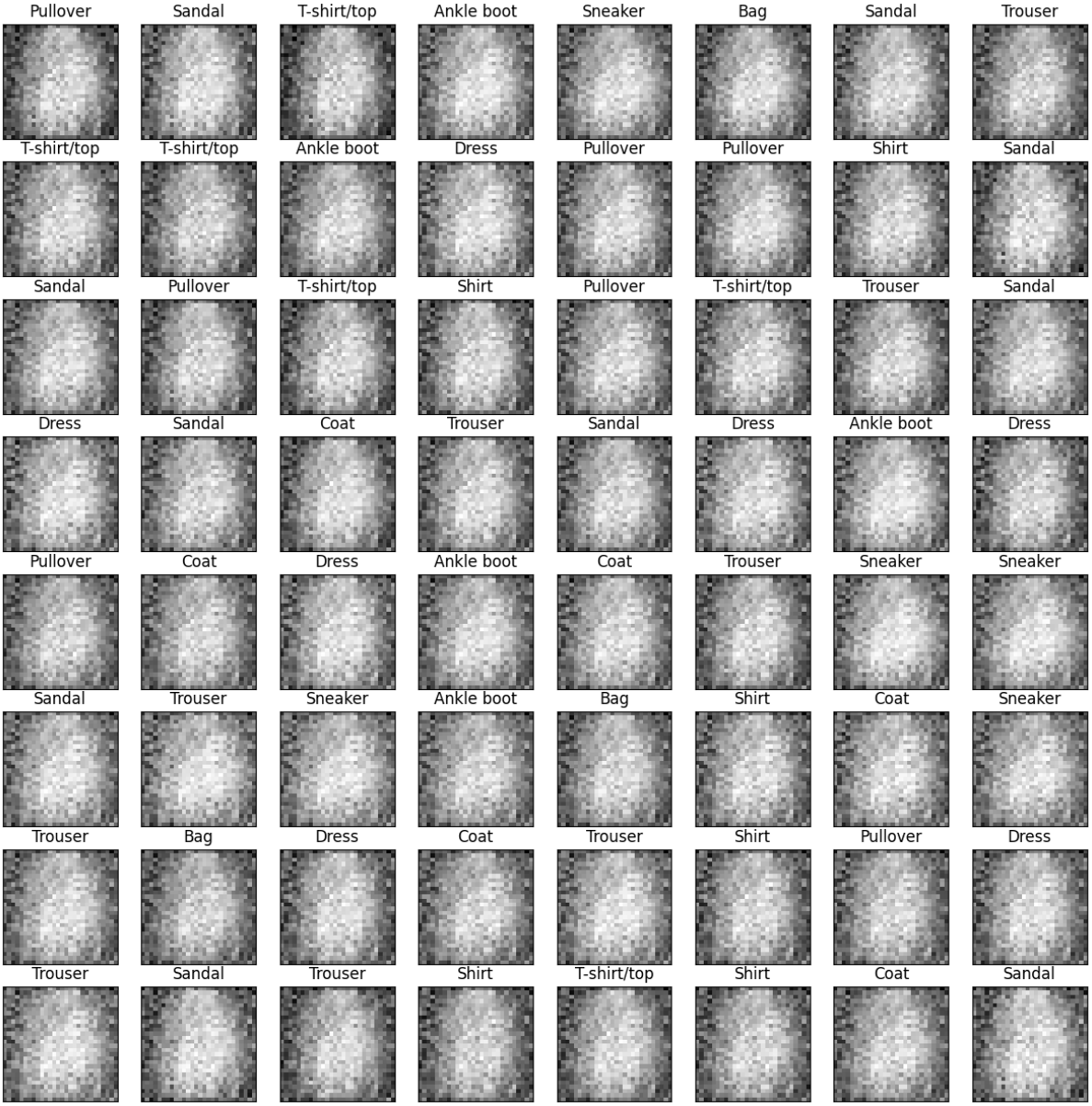
```
Training Steps Completed:  499
```



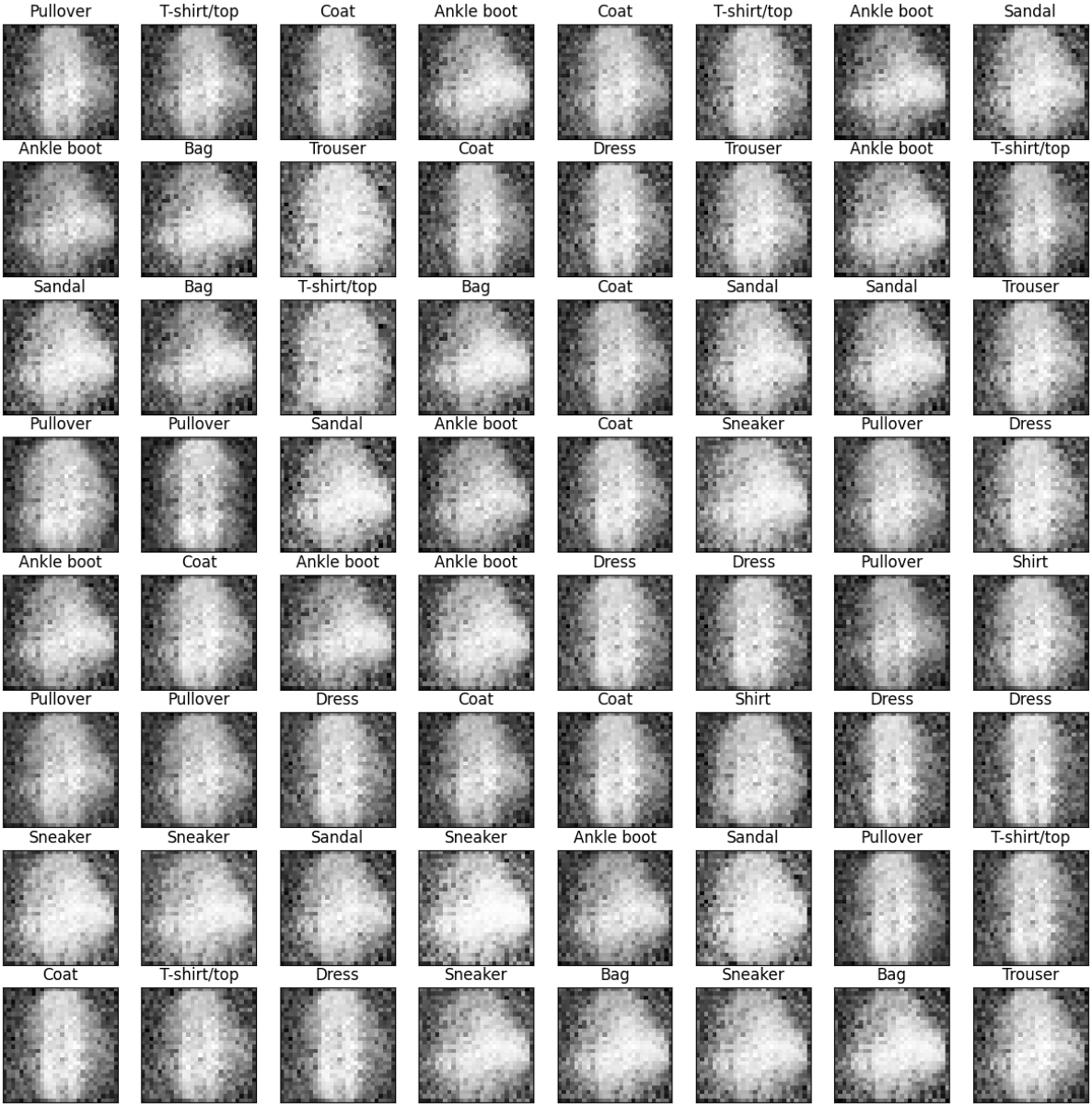

```
Epoch 0: loss_d: 1.4205117225646973, loss_g: 0.8135586977005005
Training Steps Completed:  499
```



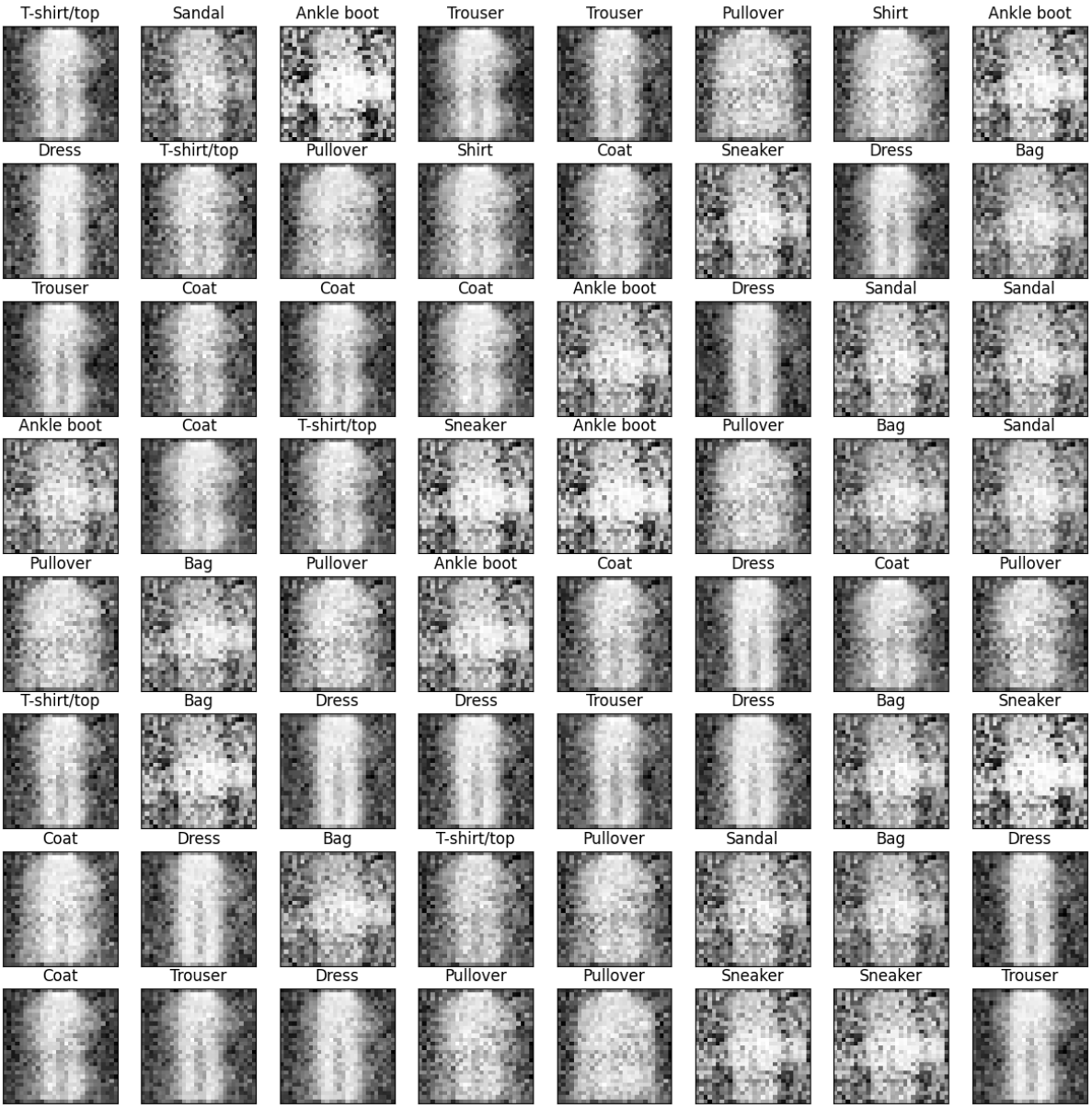

```
Epoch 1: loss_d: 1.4017736911773682, loss_g: 0.7450965046882629
Training Steps Completed:  499
```



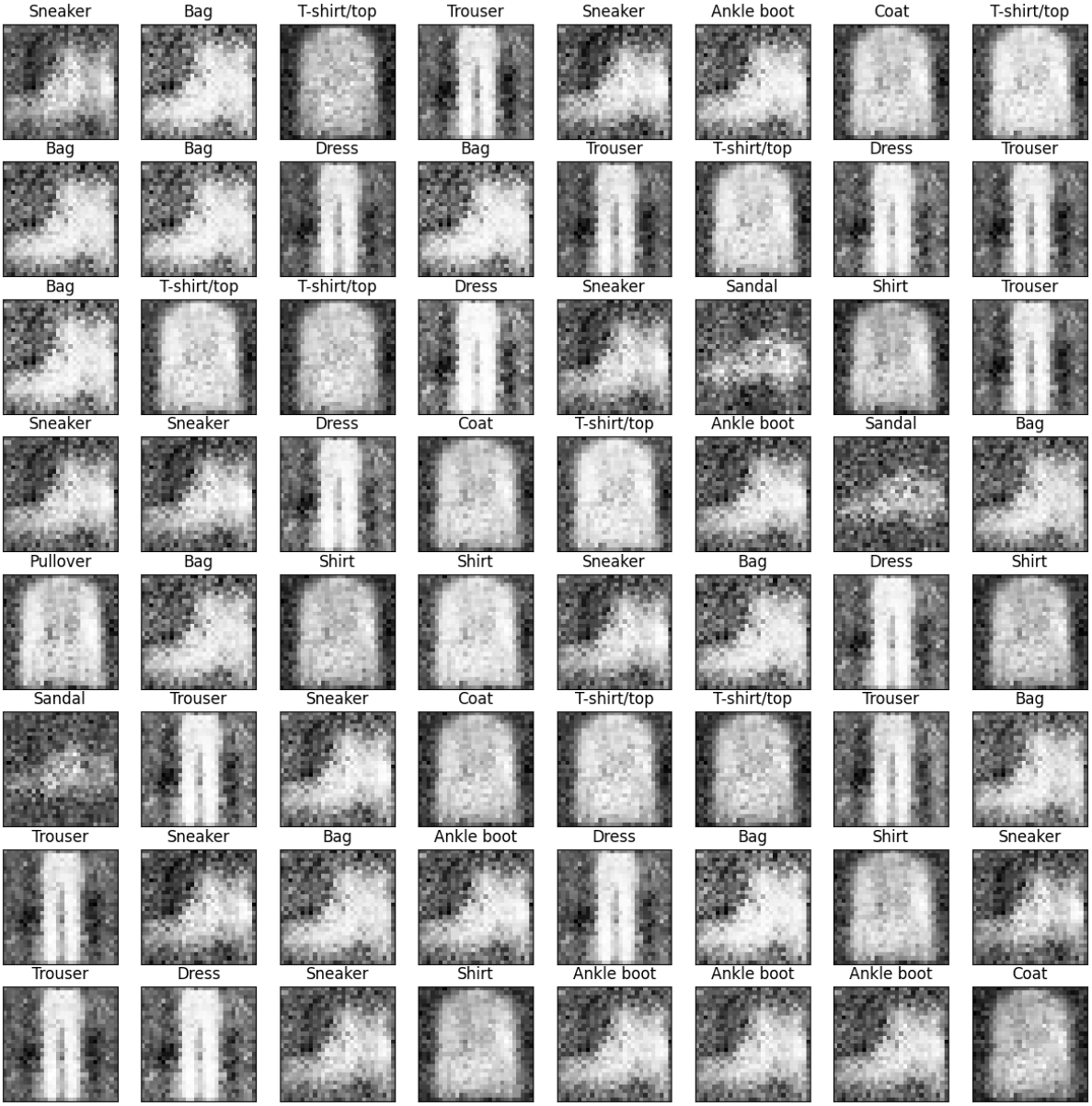

```
Epoch 2: loss_d: 1.4091084003448486, loss_g: 0.7506999969482422
Training Steps Completed:  499
```



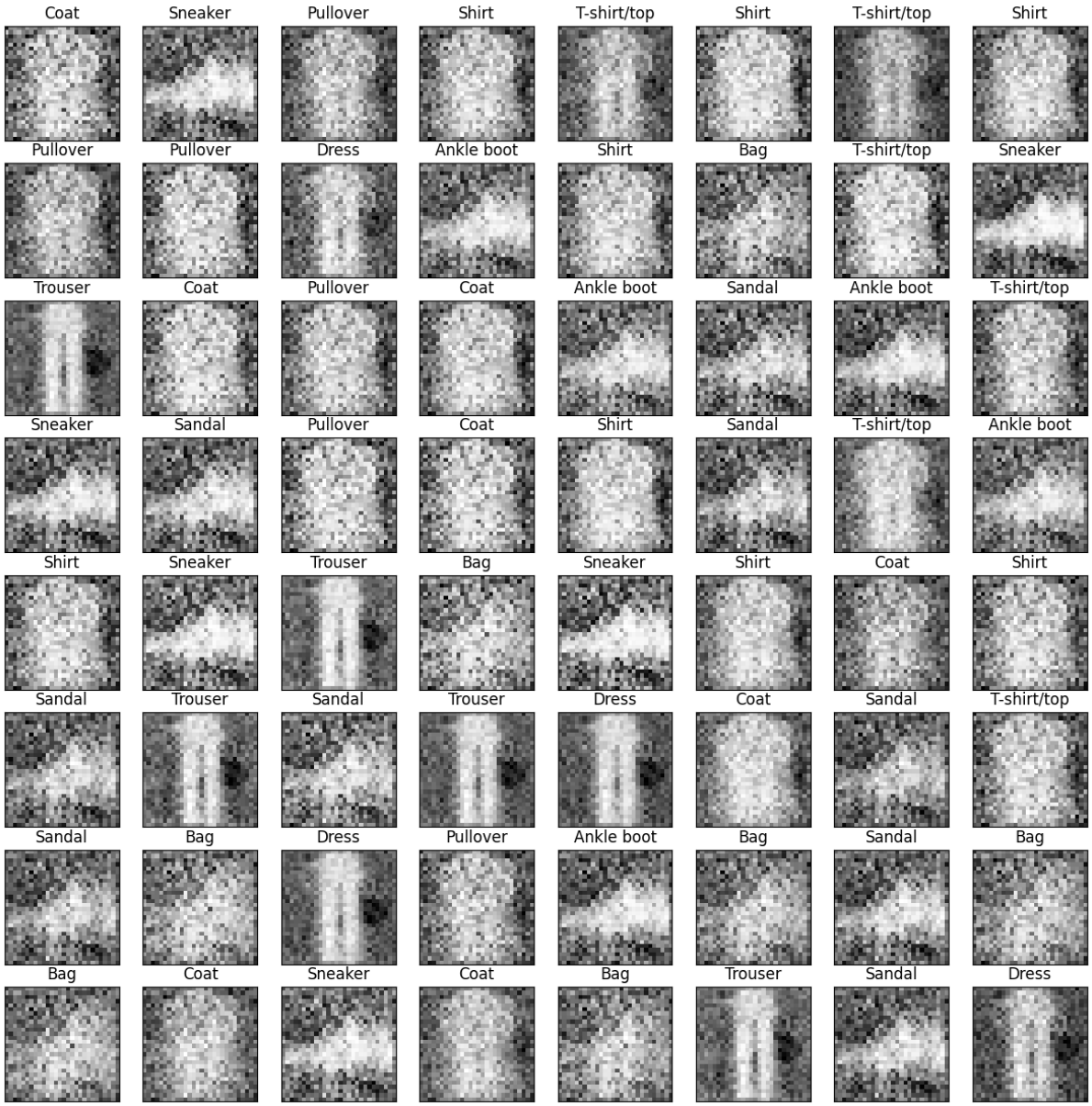

```
Epoch 3: loss_d: 1.3866671323776245, loss_g: 0.7329487204551697
Training Steps Completed:  499
```




```
Epoch 4: loss_d: 1.3881326913833618, loss_g: 0.7022793292999268
Training Steps Completed:  499
```



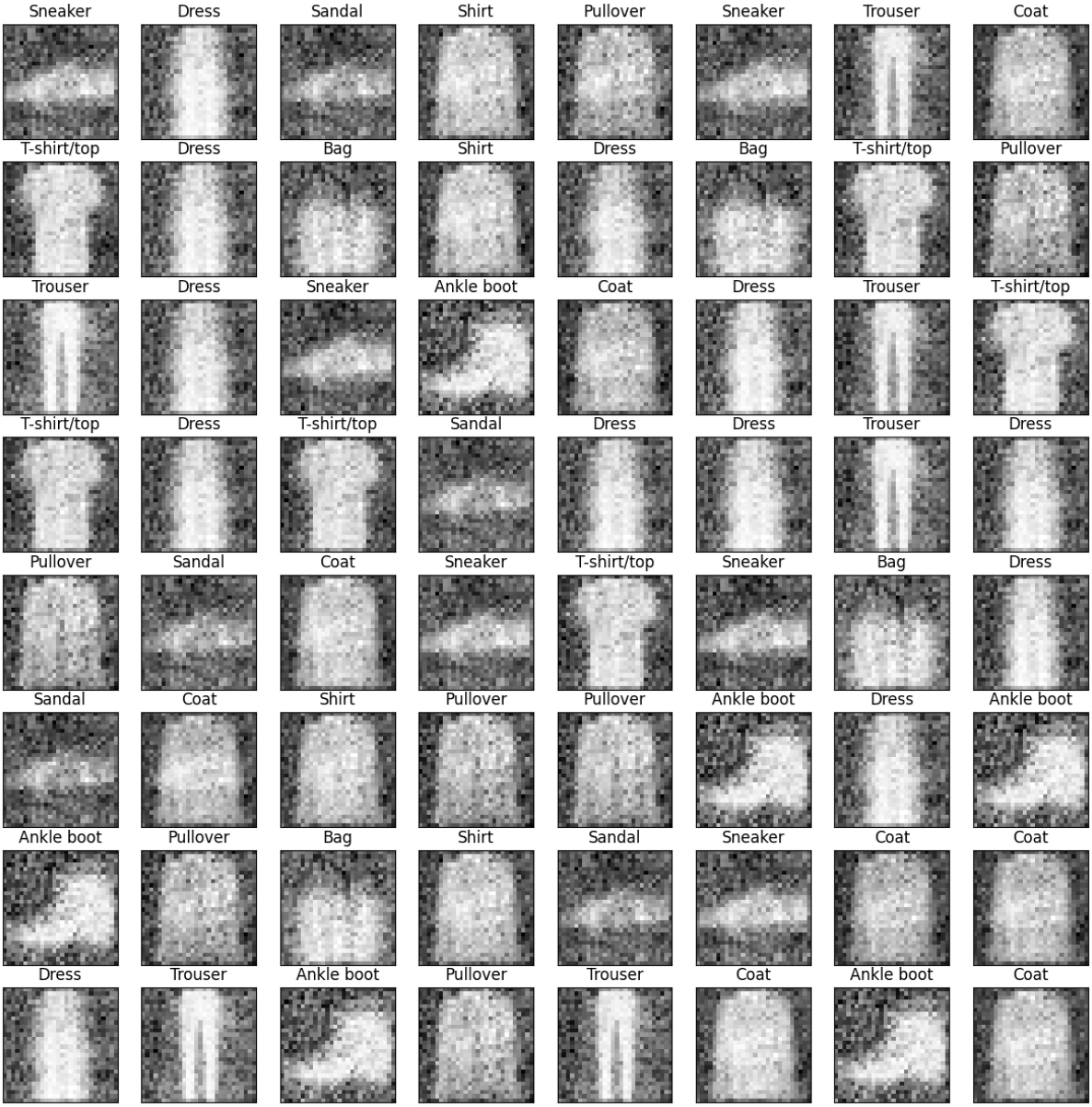

```
Epoch 5: loss_d: 1.3888123035430908, loss_g: 0.6981258392333984
Training Steps Completed:  499
```




```
Epoch 6: loss_d: 1.3961634635925293, loss_g: 0.7504462599754333
Training Steps Completed:  499
```

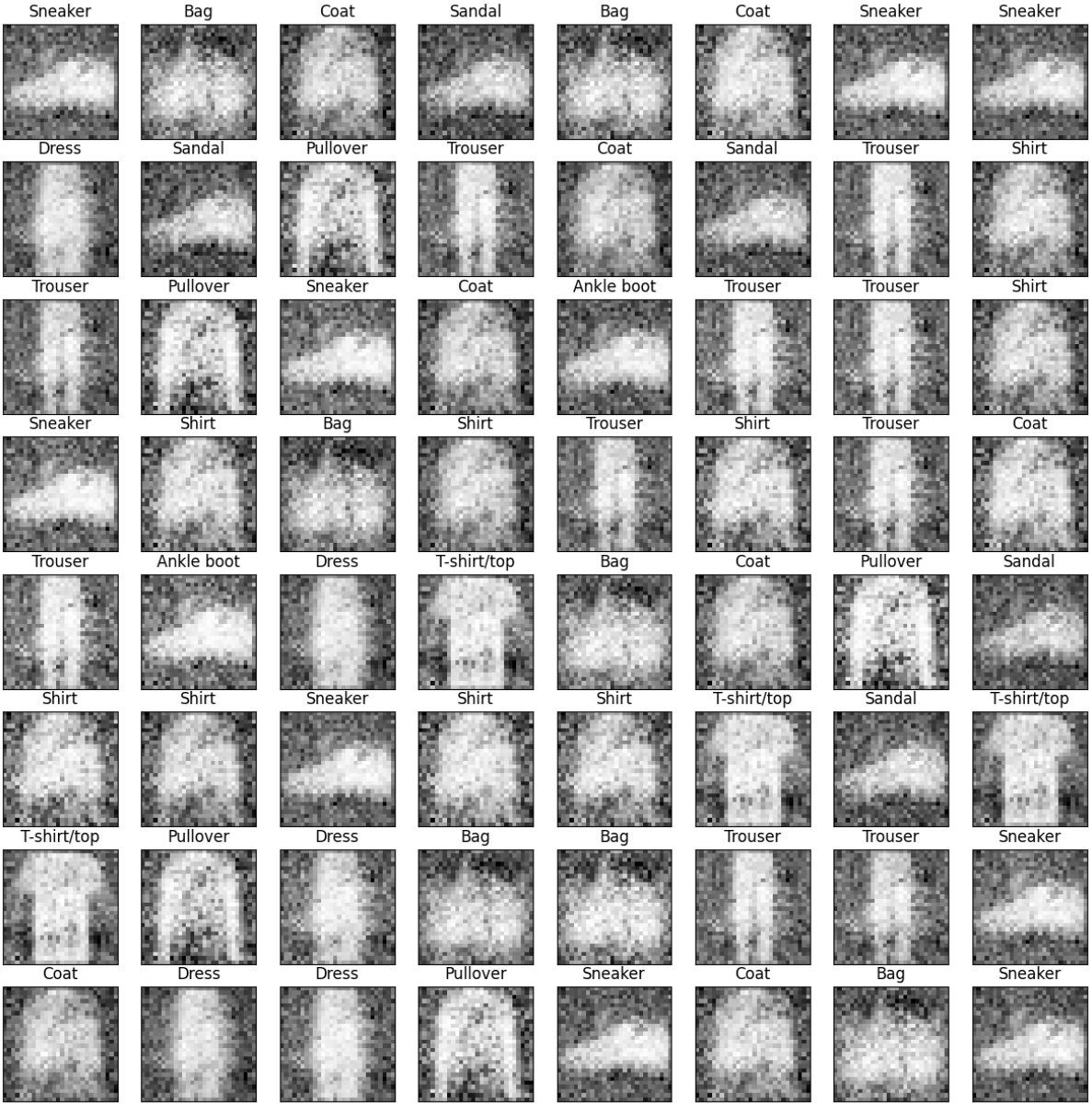

```
Epoch 7: loss_d: 1.3926175832748413, loss_g: 0.7233808636665344
Training Steps Completed:  499
```


```
Epoch 8: loss_d: 1.4169260263442993, loss_g: 0.7454743385314941
Training Steps Completed:  499
```

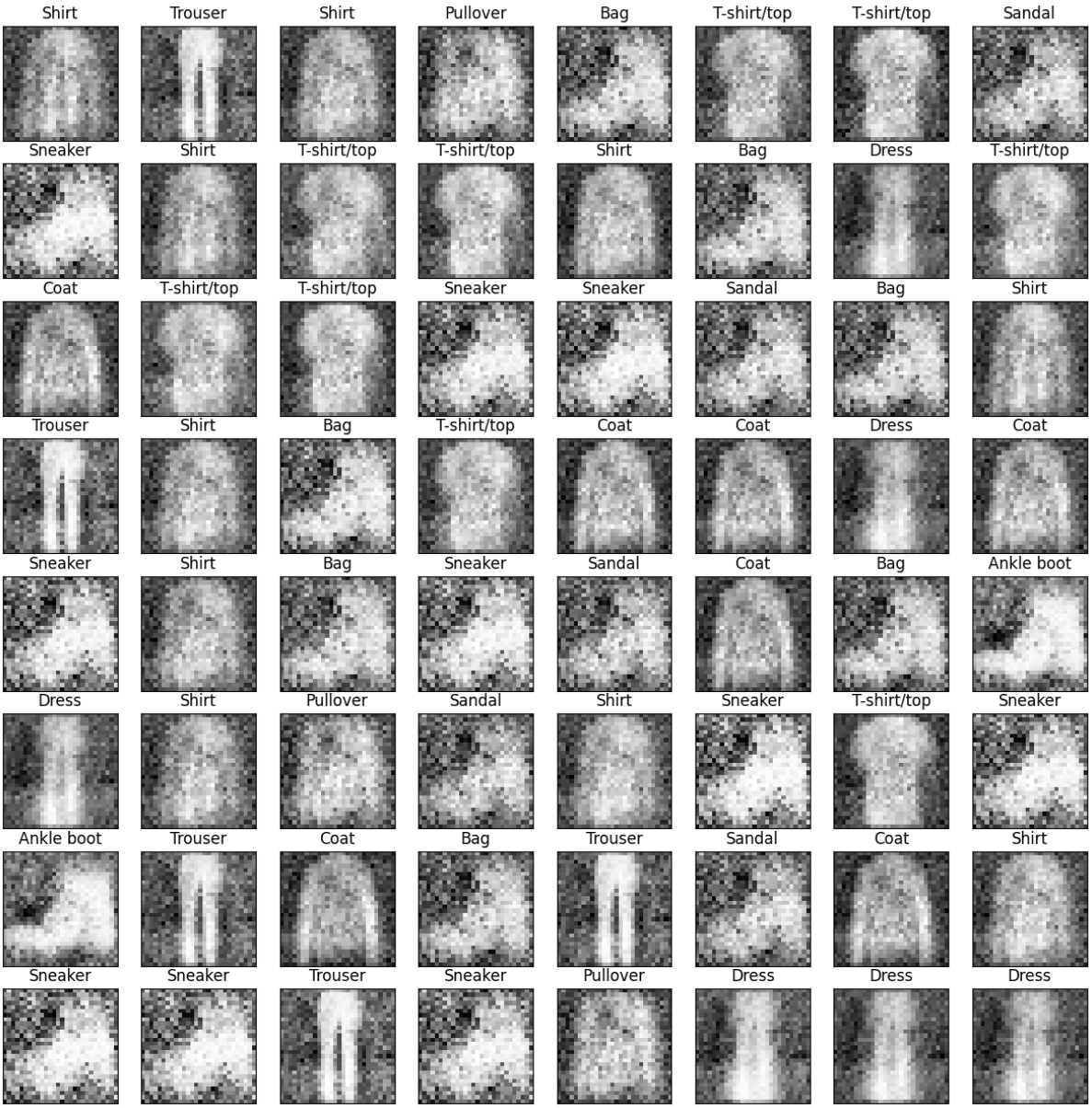



```
Epoch 9: loss_d: 1.3897982835769653, loss_g: 0.7014853358268738
Training Steps Completed:  499
```

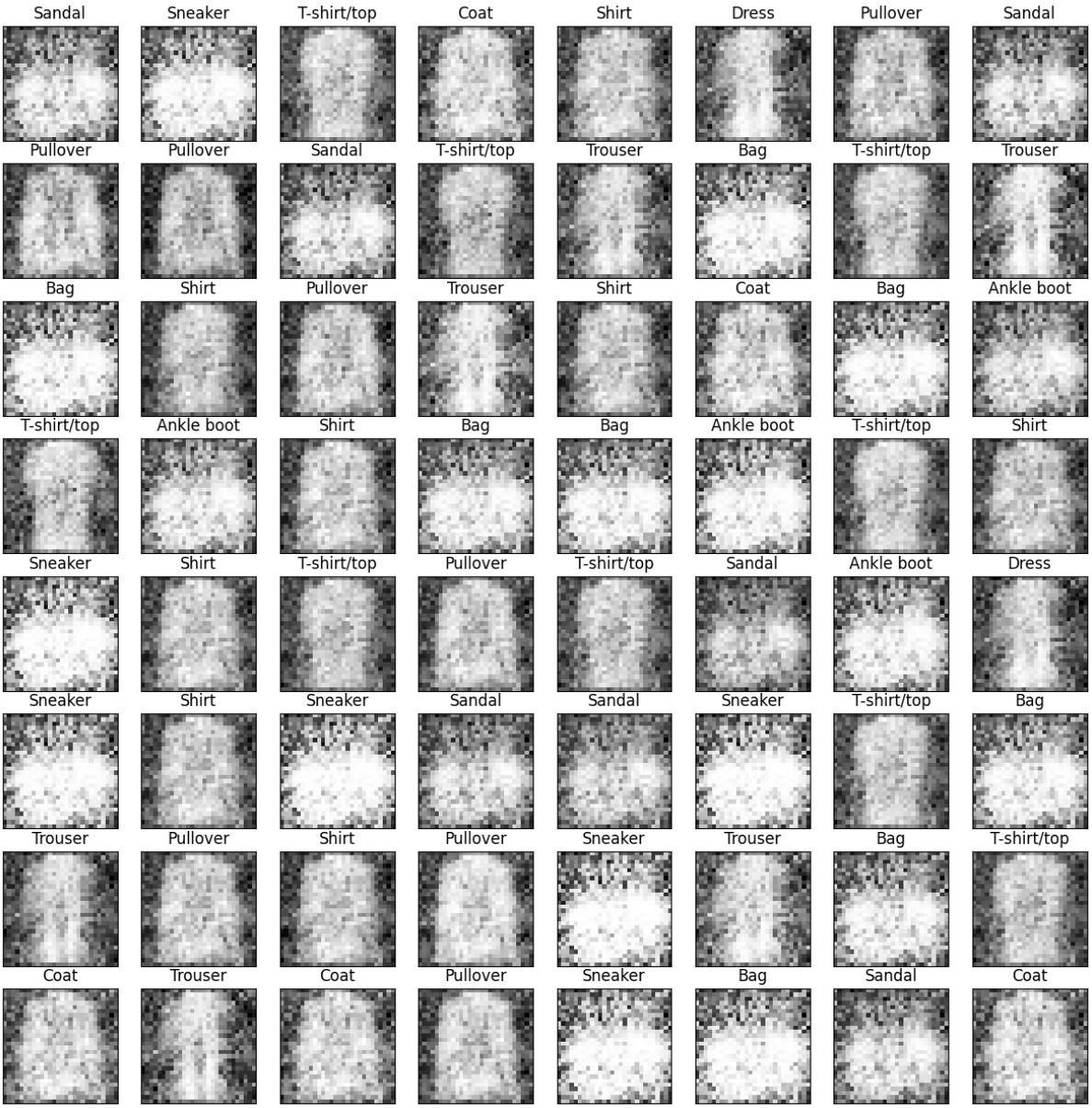

```
Epoch 10: loss_d: 1.398500919342041, loss_g: 0.8133668303489685
Training Steps Completed:  499
```



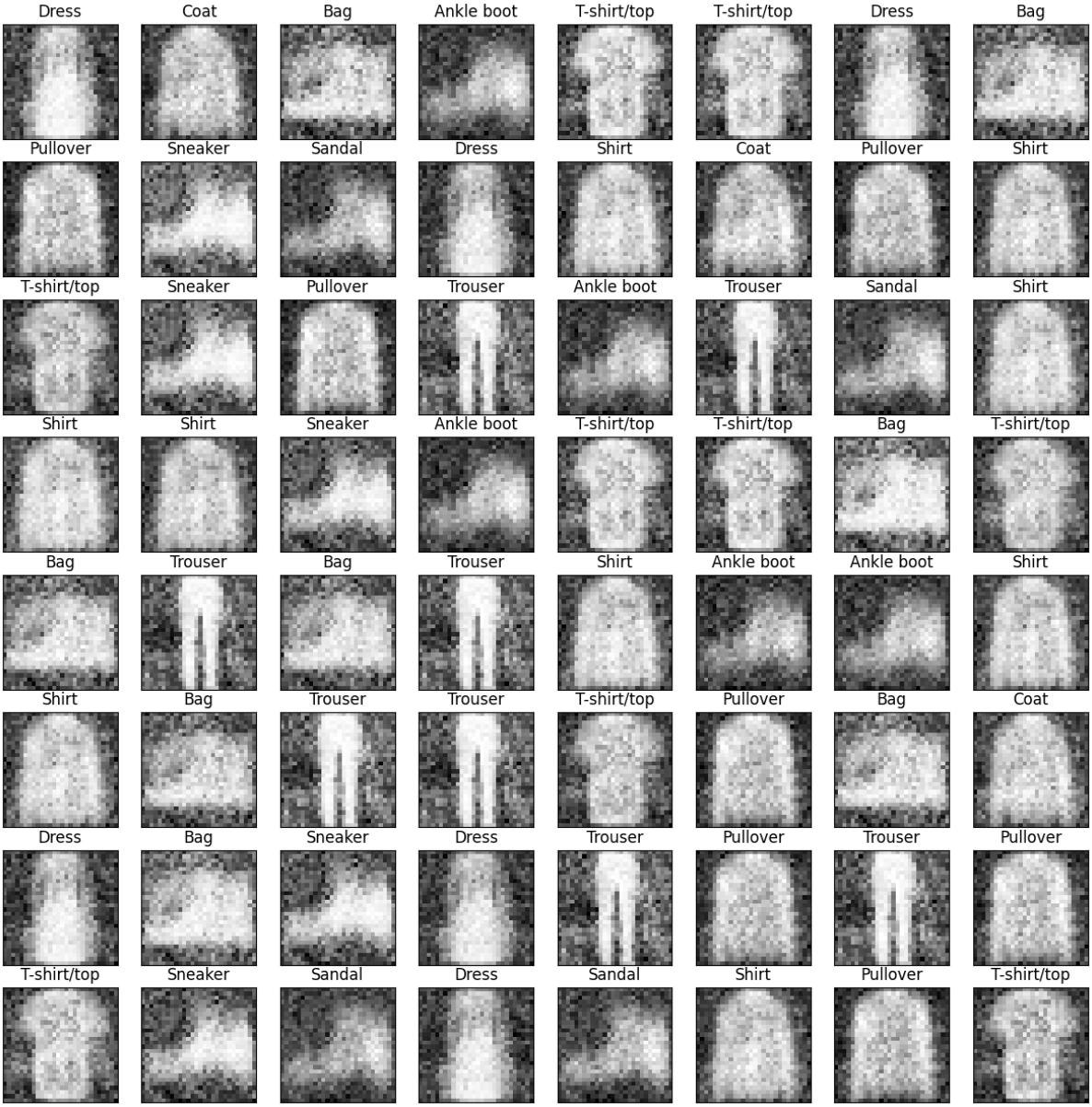

```
Epoch 11: loss_d: 1.395297646522522, loss_g: 0.6796686053276062
Training Steps Completed:  499
```



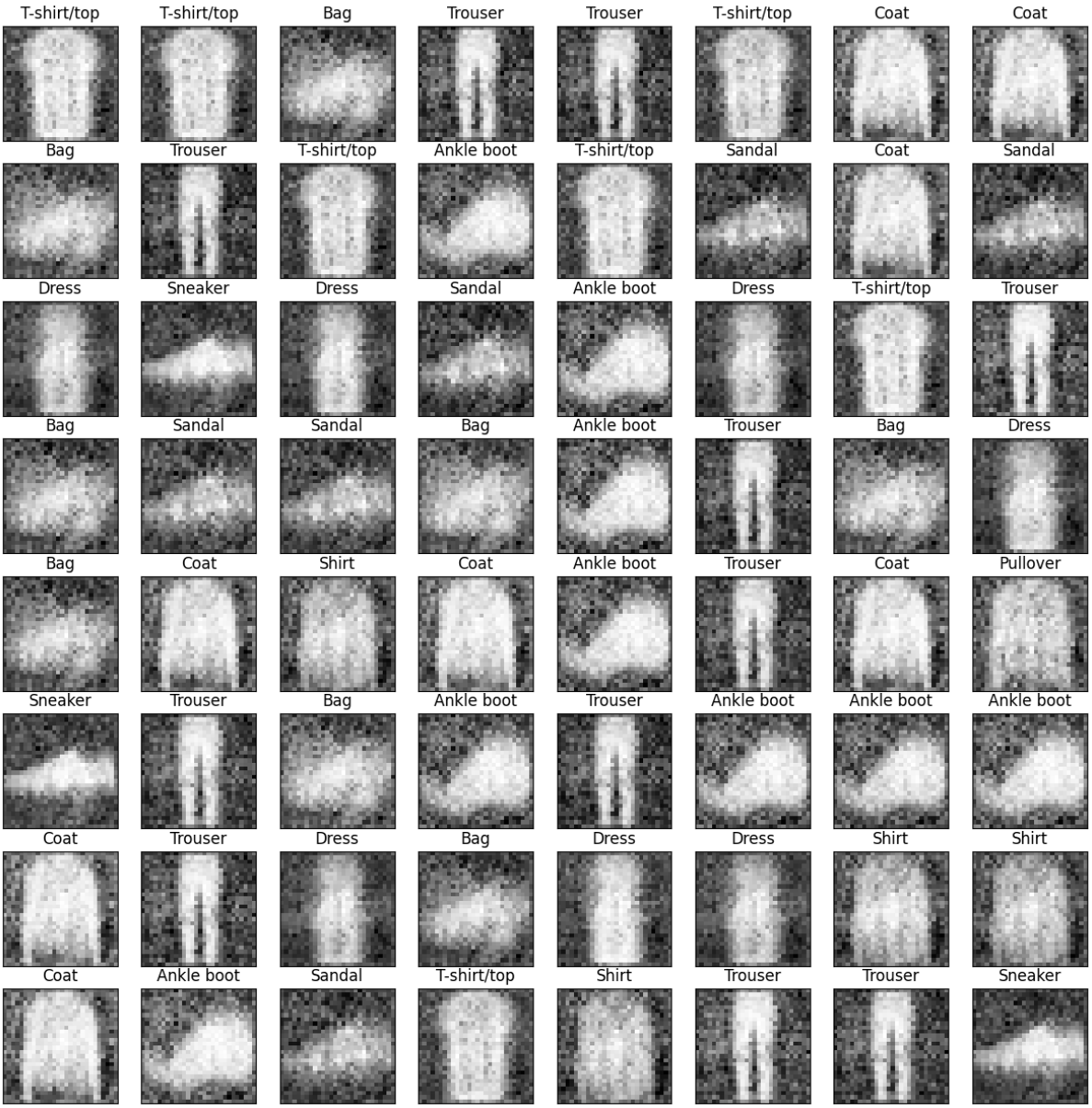

```
Epoch 12: loss_d: 1.390921711921692, loss_g: 0.6977830529212952
Training Steps Completed:  499
```



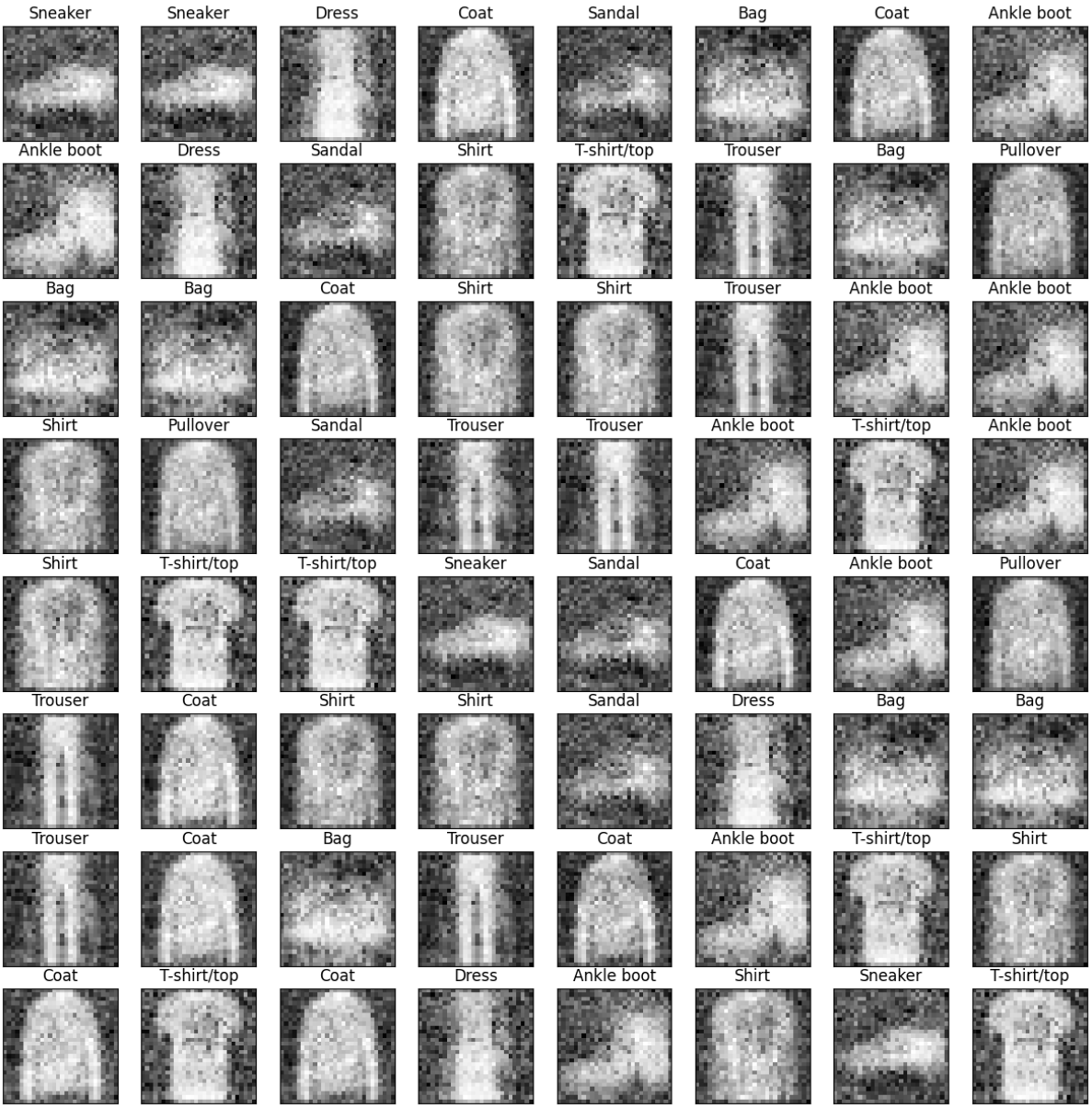

```
Epoch 13: loss_d: 1.4131816625595093, loss_g: 0.7748185992240906
Training Steps Completed:  499
```




```
Epoch 14: loss_d: 1.391539216041565, loss_g: 0.6985052227973938
Training Steps Completed:  499
```

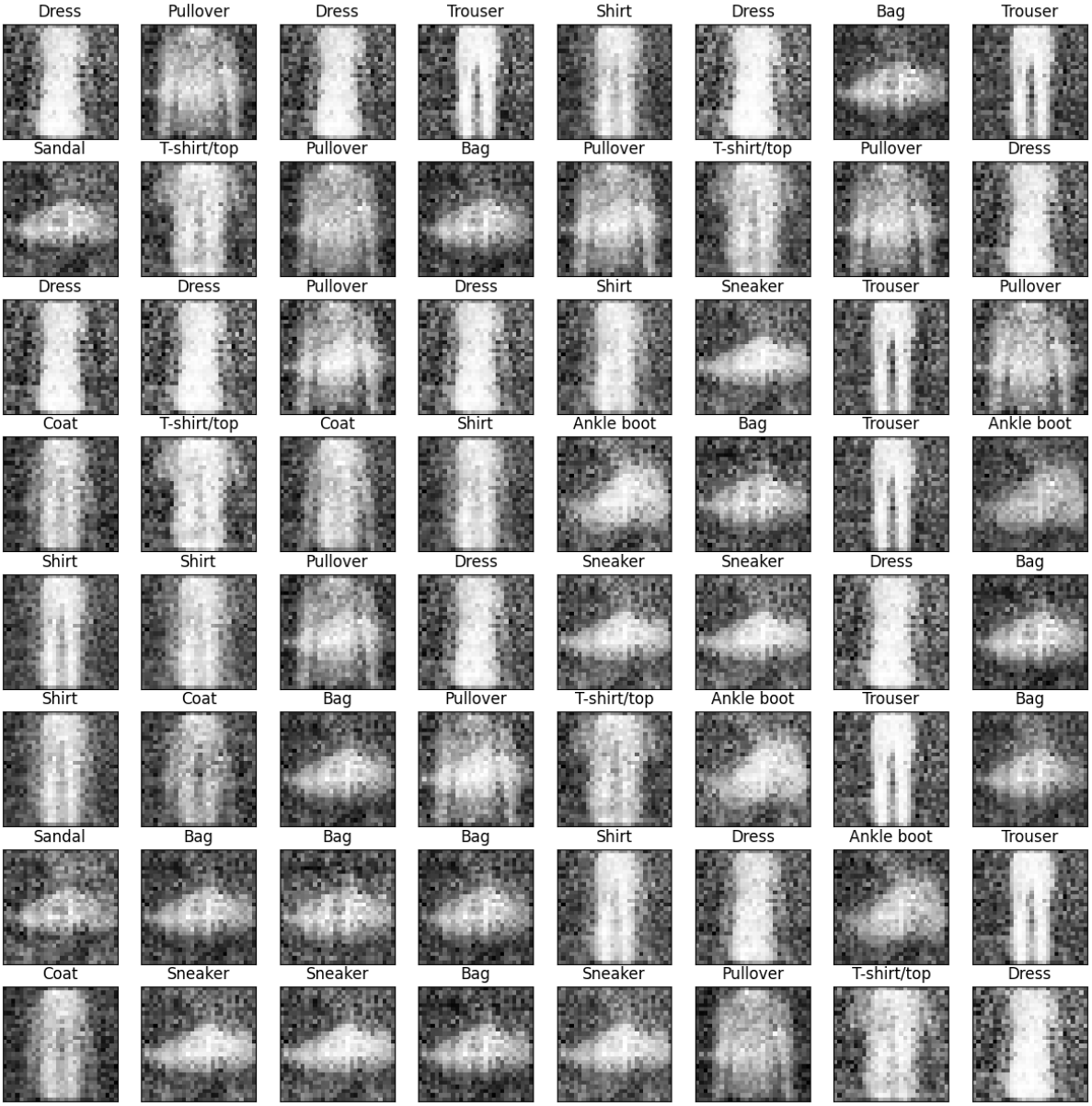



```
Epoch 15: loss_d: 1.3887841701507568, loss_g: 0.6978585124015808
Training Steps Completed:  499
```

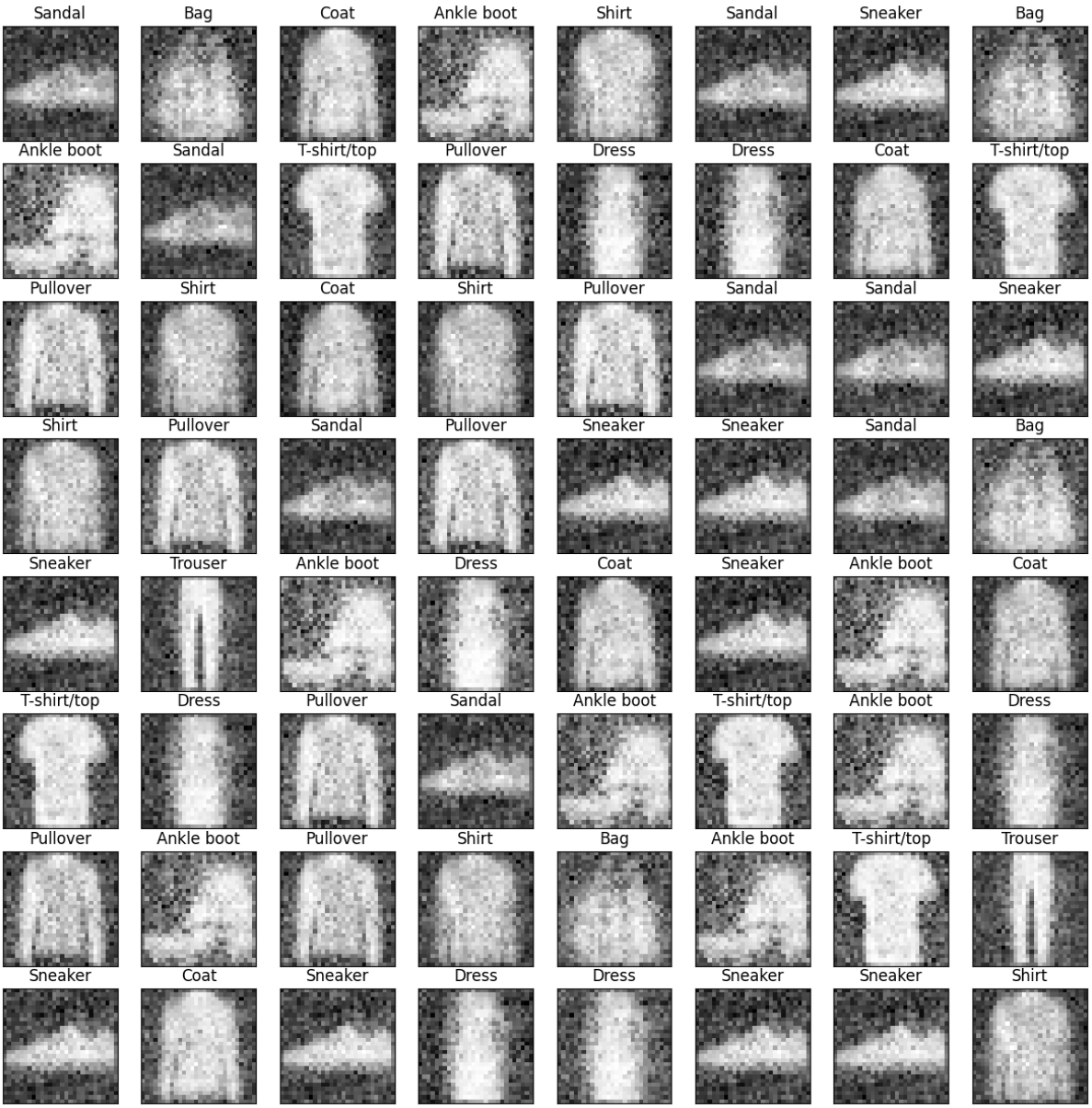



```
Epoch 16: loss_d: 1.393417239189148, loss_g: 0.6952545046806335
Training Steps Completed:  499
```

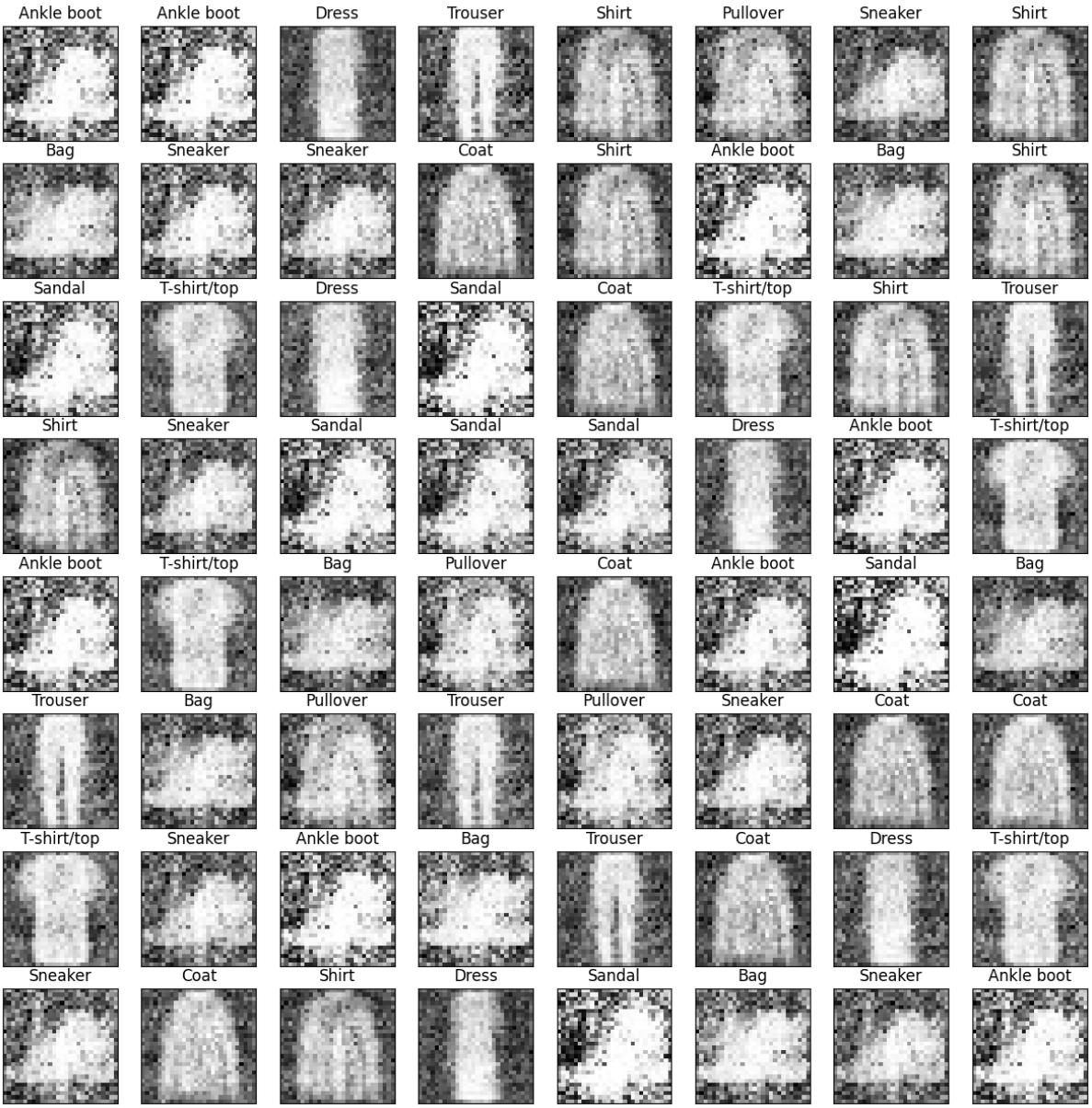

```
Epoch 17: loss_d: 1.3918980360031128, loss_g: 0.7024227976799011
Training Steps Completed:  499
```



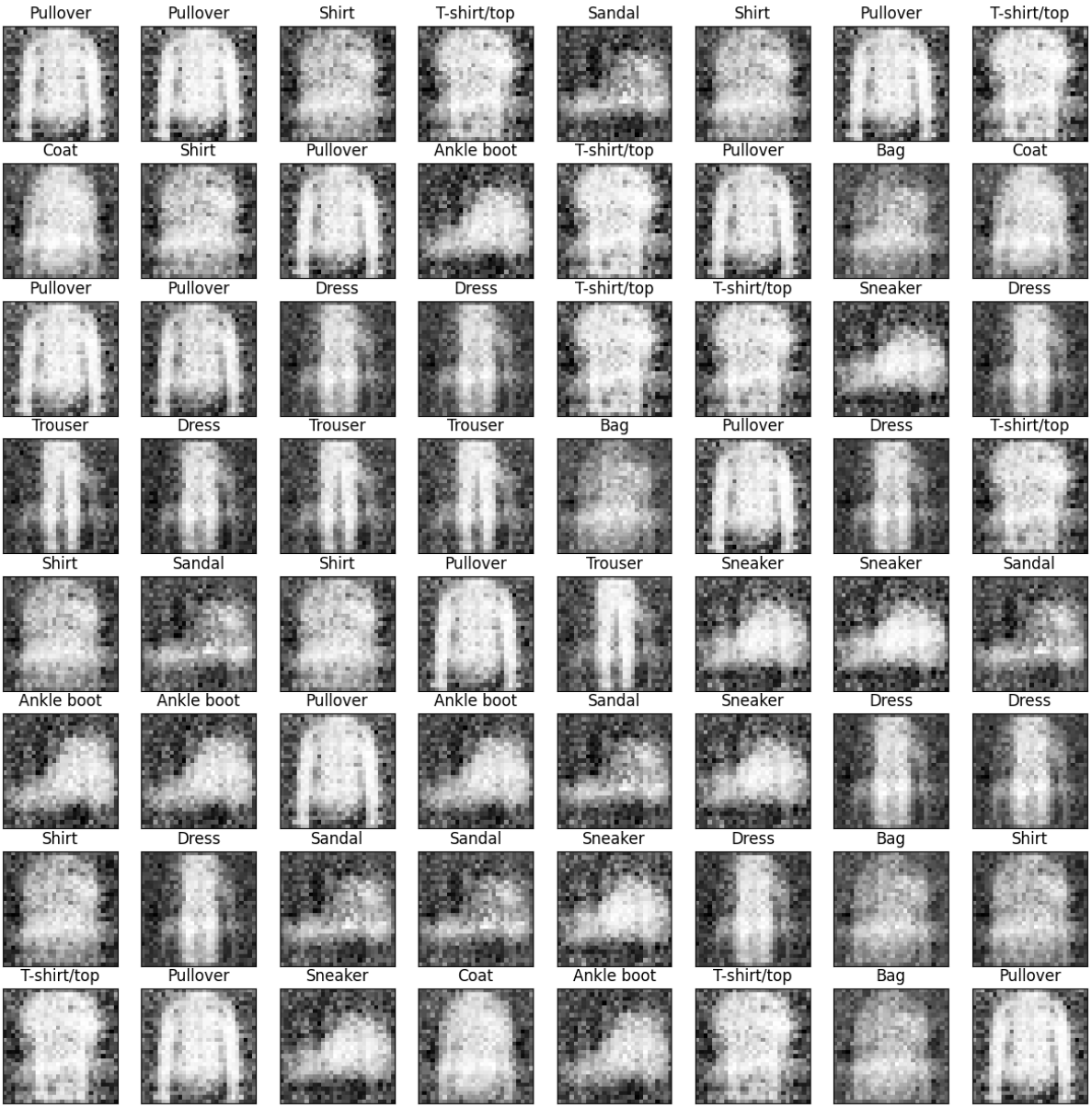

```
Epoch 18: loss_d: 1.3913910388946533, loss_g: 0.7016594409942627
Training Steps Completed:  499
```



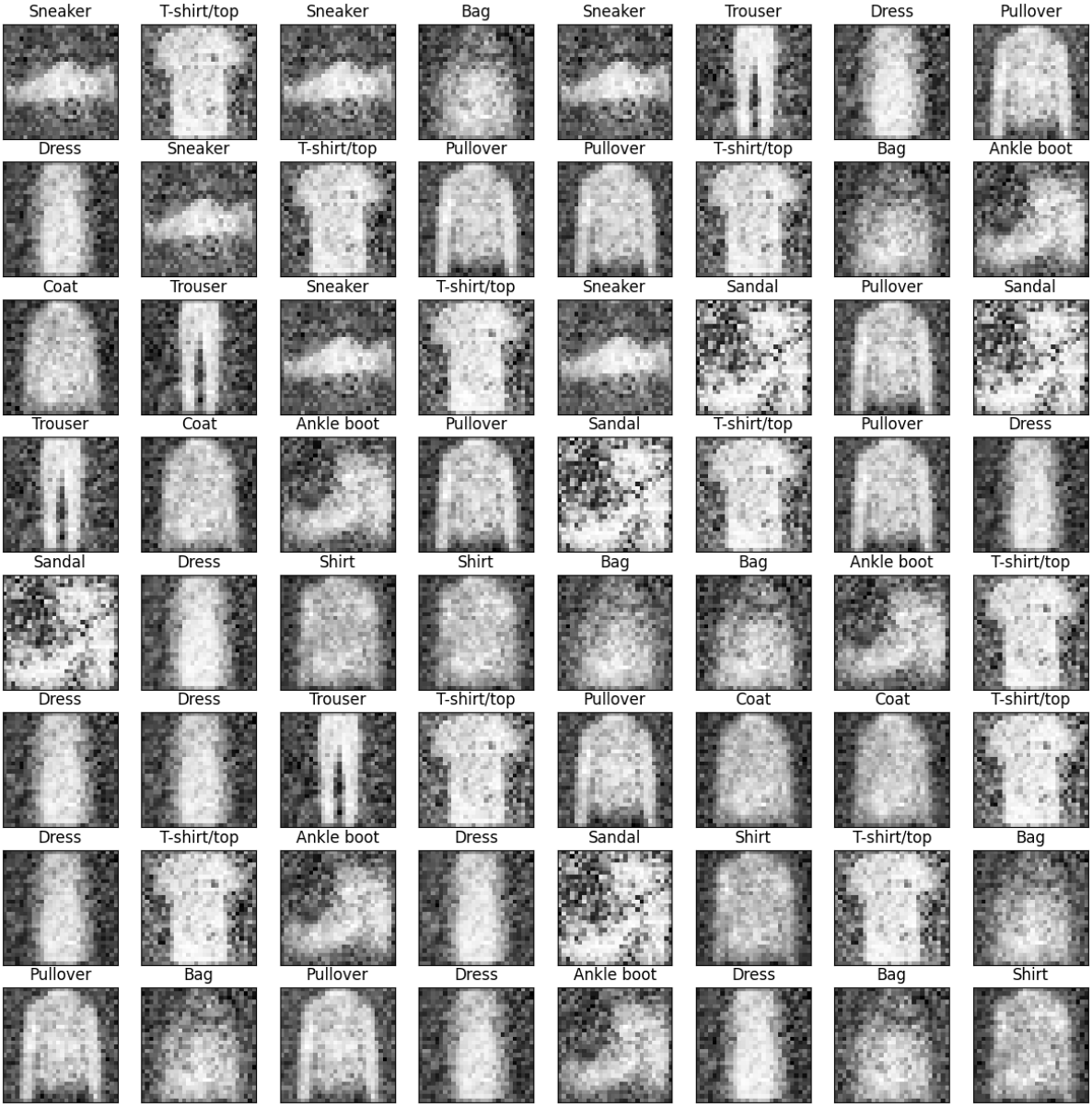

```
Epoch 19: loss_d: 1.3902037143707275, loss_g: 0.6938185095787048
Training Steps Completed:  499
```



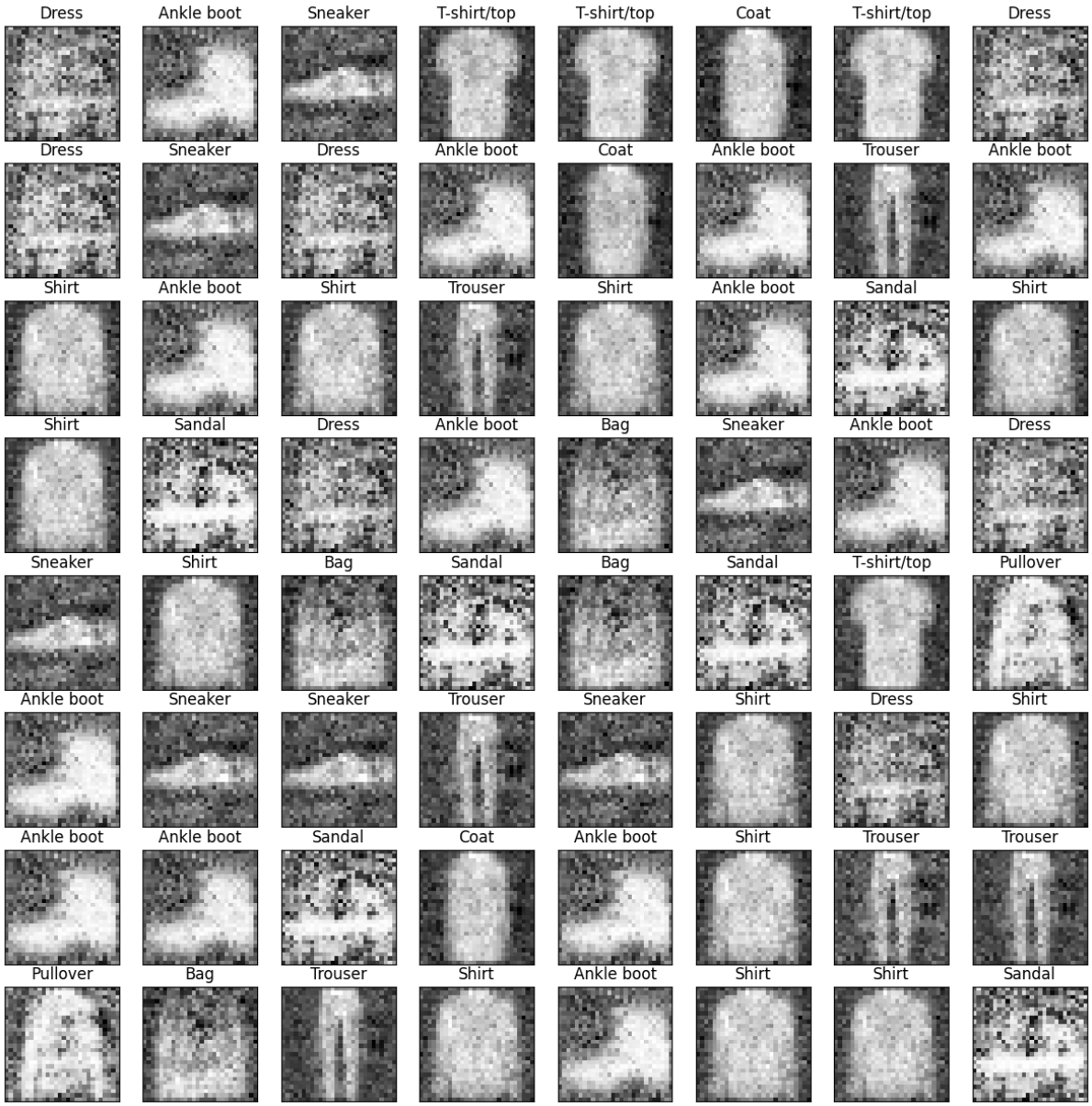

```
Epoch 20: loss_d: 1.3925540447235107, loss_g: 0.7033031582832336
Training Steps Completed:  499
```




```
Epoch 21: loss_d: 1.3885912895202637, loss_g: 0.6999940872192383
Training Steps Completed:  499
```




```
Epoch 22: loss_d: 1.3965940475463867, loss_g: 0.7255759239196777
Training Steps Completed:  499
```




```
Epoch 23: loss_d: 1.3893269300460815, loss_g: 0.7030029296875
```

In [ ]:
```
#save the model
torch.save(generator, 'generator_cond_double.pth')
torch.save(discriminator, 'discriminator_cond_double.pth')
```