```
# IMPORTANT: RUN THIS CELL IN ORDER TO IMPORT YOUR KAGGLE DATA SOURCES
# TO THE CORRECT LOCATION (/kaggle/input) IN YOUR NOTEBOOK,
# THEN FEEL FREE TO DELETE THIS CELL.
# NOTE: THIS NOTEBOOK ENVIRONMENT DIFFERS FROM KAGGLE'S PYTHON
# ENVIRONMENT SO THERE MAY BE MISSING LIBRARIES USED BY YOUR
# NOTEBOOK.

import os
import sys
from tempfile import NamedTemporaryFile
from urllib.request import urlopen
from urllib.parse import unquote, urlparse
from urllib.error import HTTPError
from zipfile import ZipFile
import tarfile
import shutil

CHUNK_SIZE = 40960
DATA_SOURCE_MAPPING = 'fashionmnist:https%3A%2F%2Fstorage.googleapis.com%2Fkaggle-data-sets%2F2243%2F9243%2Fbundle%2Farchive.zip%3FX-Go

KAGGLE_INPUT_PATH='/kaggle/input'
KAGGLE_WORKING_PATH='/kaggle/working'
KAGGLE_SYMLINK='kaggle'

!umount /kaggle/input/ 2> /dev/null
shutil.rmtree('/kaggle/input', ignore_errors=True)
os.makedirs(KAGGLE_INPUT_PATH, 0o777, exist_ok=True)
os.makedirs(KAGGLE_WORKING_PATH, 0o777, exist_ok=True)

try:
  os.symlink(KAGGLE_INPUT_PATH, os.path.join("..", 'input'), target_is_directory=True)
except FileExistsError:
  pass
try:
  os.symlink(KAGGLE_WORKING_PATH, os.path.join("..", 'working'), target_is_directory=True)
except FileExistsError:
  pass

for data_source_mapping in DATA_SOURCE_MAPPING.split(','):
    directory, download_url_encoded = data_source_mapping.split(':')
    download_url = unquote(download_url_encoded)
    filename = urlparse(download_url).path
    destination_path = os.path.join(KAGGLE_INPUT_PATH, directory)
    try:
        with urlopen(download_url) as fileres, NamedTemporaryFile() as tfile:
            total_length = fileres.headers['content-length']
            print(f'Downloading {directory}, {total_length} bytes compressed')
            dl = 0
            data = fileres.read(CHUNK_SIZE)
            while len(data) > 0:
                dl += len(data)
                tfile.write(data)
                done = int(50 * dl / int(total_length))
                sys.stdout.write(f"\r[{'=' * done}{' ' * (50-done)}] {dl} bytes downloaded")
                sys.stdout.flush()
                data = fileres.read(CHUNK_SIZE)
            if filename.endswith('.zip'):
              with ZipFile(tfile) as zfile:
                zfile.extractall(destination_path)
            else:
              with tarfile.open(tfile.name) as tarfile:
                tarfile.extractall(destination_path)
            print(f'\nDownloaded and uncompressed: {directory}')
    except HTTPError as e:
        print(f'Failed to load (likely expired) {download_url} to path {destination_path}')
        continue
    except OSError as e:
        print(f'Failed to load {download_url} to path {destination_path}')
        continue
print('Data source import complete.')
```

```
Downloading fashionmnist, 72114846 bytes compressed
[==================================================] 72114846 bytes downloaded
Downloaded and uncompressed: fashionmnist
Downloading machine-learning-architecture-diagrams, 136733 bytes compressed
[==================================================] 136733 bytes downloaded
Downloaded and uncompressed: machine-learning-architecture-diagrams
Data source import complete.
```

```
from IPython.display import SVG, display
svg_file = '/kaggle/input/machine-learning-architecture-diagrams/CGAN.svg'
display(SVG(filename=svg_file))
```

```
from torch import optim
import os
import torchvision.utils as utils
import numpy as numpy
from torchvision import datasets
from torchvision import transforms
from torch.utils.data import Dataset, DataLoader
import torch
import torch.nn as nn
import torch.nn.functional as F
```

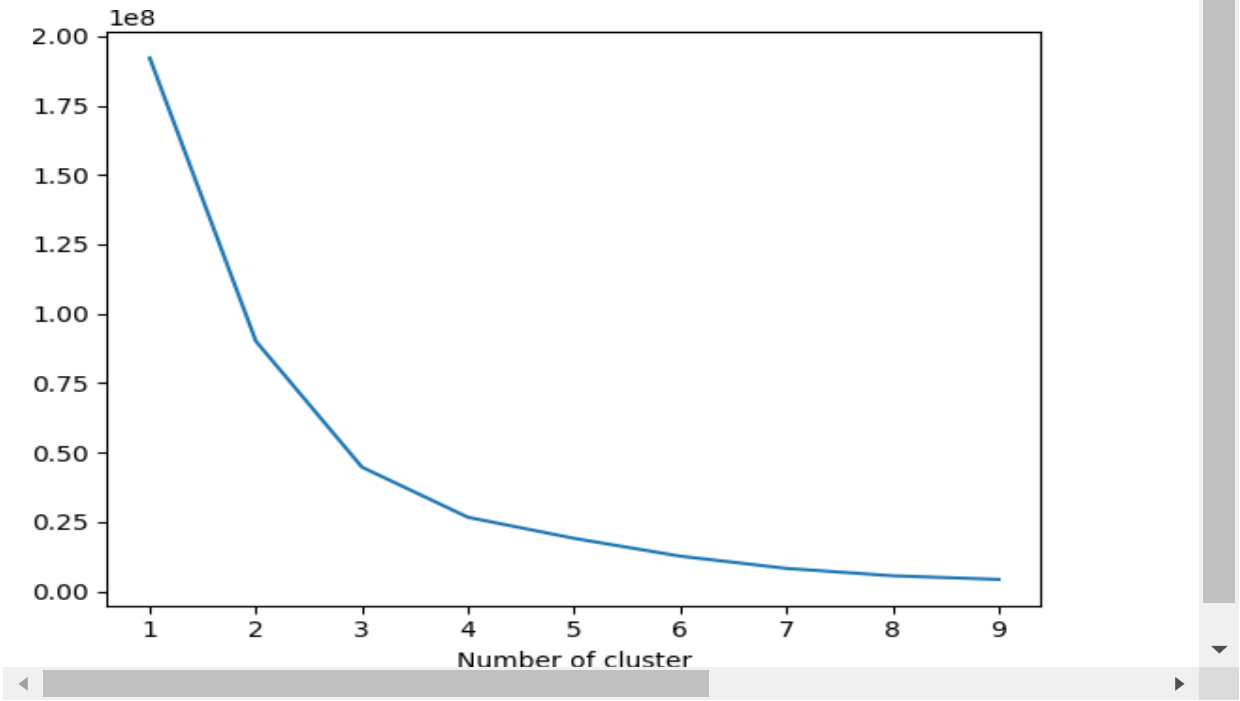
```
dataset_path = os.path.join('./data', 'FashionMNIST')
os.makedirs(dataset_path, exist_ok=True)
model_path = os.path.join('./model', 'FashionMNIST')
os.makedirs(model_path,exist_ok=True)
samples_path = os.path.join('./samples','FashionMNIST')
os.makedirs(samples_path,exist_ok=True)

transform = transforms.Compose([transforms.Resize([32,32]),
                                transforms.ToTensor(),
                                transforms.Normalize([0.5],[0.5])])

dataset = datasets.FashionMNIST(dataset_path, train=True, download=True, transform=transform)
```

0	12346.0	2011-01-18 10:17:00	325
1	12747.0	2011-12-07 14:34:00	1
2	12748.0	2011-12-09 12:20:00	0
3	12749.0	2011-12-06 09:56:00	3
4	12820.0	2011-12-06 15:12:00	2

```
train_loader = DataLoader(dataset=dataset, batch_size=256, shuffle=True, num_workers=4, drop_last=True)
```

```
for batch in train_loader:
    print(batch)
    break
```

```
[[[-1.0000, -1.0000, -1.0000,  ..., -1.0000, -1.0000, -1.0000],
  [-1.0000, -1.0000, -1.0000,  ..., -1.0000, -1.0000, -1.0000],
  [-1.0000, -1.0000, -1.0000,  ..., -1.0000, -1.0000, -1.0000],
  ...,
  [-1.0000, -1.0000, -1.0000,  ..., -1.0000, -1.0000, -1.0000],
  [-1.0000, -1.0000, -1.0000,  ..., -1.0000, -1.0000, -1.0000],
  [-1.0000, -1.0000, -1.0000,  ..., -1.0000, -1.0000, -1.0000]]],


[[[-1.0000, -1.0000, -1.0000,  ..., -1.0000, -1.0000, -1.0000],
  [-1.0000, -1.0000, -1.0000,  ..., -1.0000, -1.0000, -1.0000],
  [-1.0000, -1.0000, -1.0000,  ..., -1.0000, -1.0000, -1.0000],
  ...,
  [-1.0000, -1.0000, -1.0000,  ..., -1.0000, -1.0000, -1.0000],
  [-1.0000, -1.0000, -1.0000,  ..., -1.0000, -1.0000, -1.0000],
  [-1.0000, -1.0000, -1.0000,  ..., -1.0000, -1.0000, -1.0000]]],


...,


[[[-1.0000, -1.0000, -1.0000,  ..., -1.0000, -1.0000, -1.0000],
  [-1.0000, -1.0000, -1.0000,  ..., -1.0000, -1.0000, -1.0000],
  [-1.0000, -1.0000, -1.0000,  ..., -1.0000, -1.0000, -1.0000],
  ...,
  [-1.0000, -1.0000, -1.0000,  ..., -1.0000, -1.0000, -1.0000],
  [-1.0000, -1.0000, -1.0000,  ..., -1.0000, -1.0000, -1.0000],
  [-1.0000, -1.0000, -1.0000,  ..., -1.0000, -1.0000, -1.0000]]],


[[[-1.0000, -1.0000, -1.0000,  ..., -1.0000, -1.0000, -1.0000],
  [-1.0000, -1.0000, -1.0000,  ..., -1.0000, -1.0000, -1.0000],
  [-1.0000, -1.0000, -1.0000,  ..., -1.0000, -1.0000, -1.0000],
  ...,
  [-1.0000, -1.0000, -0.6078,  ..., -0.6000, -1.0000, -1.0000],
  [-1.0000, -1.0000, -0.5059,  ..., -0.4667, -1.0000, -1.0000],
  [-1.0000, -1.0000, -0.9686,  ..., -0.9843, -1.0000, -1.0000]]],


[[[-1.0000, -1.0000, -1.0000,  ..., -1.0000, -1.0000, -1.0000],
  [-1.0000, -1.0000, -1.0000,  ..., -1.0000, -1.0000, -1.0000],
  [-1.0000, -1.0000, -1.0000,  ..., -1.0000, -1.0000, -1.0000],
  ...,
  [-1.0000, -1.0000, -1.0000,  ..., -1.0000, -1.0000, -1.0000],
  [-1.0000, -1.0000, -1.0000,  ..., -1.0000, -1.0000, -1.0000],
  [-1.0000, -1.0000, -1.0000,  ..., -1.0000, -1.0000, -1.0000]]]]), tensor([9, 5, 4, 6, 3, 7, 7, 8, 3, 2, 5, 9, 2, 1, 6,
4, 4, 5, 9, 1, 2, 7, 0, 6, 5, 1, 4, 2, 8, 5, 8, 2, 7, 0, 4, 3, 8, 9, 6,
3, 4, 7, 4, 8, 7, 3, 7, 7, 1, 3, 9, 4, 0, 5, 9, 8, 7, 4, 9, 5, 6, 0, 5,
0, 3, 5, 2, 5, 7, 1, 6, 1, 9, 3, 2, 5, 4, 5, 0, 7, 5, 8, 2, 4, 7, 7, 9,
5, 8, 9, 6, 0, 7, 0, 4, 1, 0, 5, 4, 8, 0, 2, 0, 1, 0, 1, 4, 8, 7, 4, 3,
3, 6, 3, 2, 2, 2, 9, 0, 0, 5, 4, 7, 8, 0, 9, 8, 6, 7, 2, 8, 0, 9, 6, 1,
5, 0, 0, 3, 3, 7, 5, 7, 6, 9, 4, 4, 8, 7, 3, 3, 0, 2, 6, 6, 7, 4, 1, 8,
8, 9, 6, 9, 6, 6, 1, 8, 6, 2, 9, 9, 6, 2, 5, 4, 1, 8, 3, 0, 6, 7, 8, 4,
2, 3, 9, 4, 7, 9, 4, 6, 0, 1, 5, 0, 1, 3, 3, 5, 5, 8, 3, 7, 8, 7, 1, 4,
0, 4, 3, 0, 2, 8, 4, 3, 4, 6, 7, 5, 8, 7, 3, 4, 8, 4, 9, 8, 3, 1, 6, 5,
7  4  8  8  8  4  9  7  3  9  2  5  6  8  6  2])]
```




```
def convolution_block(in_channels,out_channels, kernel=4,stride=2, pad=1,bias=False, transpose=False):
    module= []
    if transpose:
        module.append(nn.ConvTranspose2d(in_channels,out_channels,kernel,stride, pad, bias=bias))
    else:
        module.append(nn.Conv2d(in_channels,out_channels,kernel,stride,pad,bias=bias))
    if bias == False:
        #use batch norm
        module.append(nn.BatchNorm2d(out_channels))

    return nn.Sequential(*module)
```

```
class Generator(nn.Module):
    def_init_(self,z_dim=10, num_classes=10, label_embed_size=5, channels=3, conv_dim=64):
        super().__init__()
        self.label_embedding = nn.Embedding(num_classes, label_embed_size)
        self.transpose_conv1 =convolution_block(z_dim+label_embed_size,conv_dim*4, pad=0, transpose=True)
        self.transpose_conv2 = convolution_block(conv_dim*4, conv_dim*2, transpose=True)
        self.transpose_conv3 = convolution_block(conv_dim*2, conv_dim, transpose=True)
        self.transpose_conv4 = convolution_block(conv_dim, channels, transpose=True,bias=True) #no batch norm

        for m in self.modules():
            #initialising  weights
            if isinstance(m,nn.Conv2d) or isinstance(m,nn.ConvTranspose2d):
                nn.init.normal_(m.weight, 0.0, 0.02)
            if isinstance(m,nn.BatchNorm2d):
                nn.init.constant_(m.weight,1)
                nn.init.constant_(m.bias,0)
    def forward(self,x,label):
        #reshaping x
        x = x.reshape([x.shape[0],-1,1,1])
        label_embed = self.label_embedding(label)
```

```
        label_embed = label_embed.reshape([label_embed.shape[0],-1,1,1])
        x = torch.cat((x,label_embed),dim=1)
        x = F.relu(self.transpose_conv1(x))
        x = F.relu(self.transpose_conv2(x))
        x = F.relu(self.transpose_conv3(x))
        x = torch.tanh(self.transpose_conv4(x))
        return x



class Discriminator(nn.Module):
    def_init_(self, num_classes=10, channels=3, conv_dim=64):
        super(Discriminator, self)._init__()
        self.image_size = 32
        self.label_embedding = nn.Embedding(num_classes, self.image_size*self.image_size)
        self.conv1 = convolution_block(channels + 1, conv_dim, bias=True)
        self.conv2 = convolution_block(conv_dim, conv_dim * 2)
        self.conv3 = convolution_block(conv_dim * 2, conv_dim * 4)
        self.conv4 = convolution_block(conv_dim * 4, 1, kernel=4, stride=1, pad=0, bias=True)

        for m in self.modules():
            if isinstance(m, nn.Conv2d):
                nn.init.normal_(m.weight, 0.0, 0.02)

            if isinstance(m, nn.BatchNorm2d):
                nn.init.constant_(m.weight, 1)
                nn.init.constant_(m.bias, 0)

    def forward(self, x, label):
        alpha = 0.2
        label_embed = self.label_embedding(label)
        label_embed = label_embed.reshape([label_embed.shape[0], 1, self.image_size, self.image_size])
        x = torch.cat((x, label_embed), dim=1)
        x = F.leaky_relu(self.conv1(x), alpha)
        x = F.leaky_relu(self.conv2(x), alpha)
        x = F.leaky_relu(self.conv3(x), alpha)
        x = torch.sigmoid(self.conv4(x))
        return x.squeeze()
```

0	17850.0	301	0	312	1	5288.63
1	14688.0	7	3	359	1	5107.38
2	13767.0	1	3	399	1	16945.71
3	15513.0	30	3	314	1	14520.08
4	14849.0	21	3	392	1	7904.28
…	...	...	...	...	...	...
3945	12748.0	0	3	4642	3	29072.10
3946	17841.0	1	3	7983	3	40340.78

```
Z_DIM=10
LABEL_EMBEDDING_SIZE=5
NUM_CLASSES=10
IMGS_TO_DISPLAY_PER_CLASS=10
LOAD_MODEL = False
CHANNELS=1
EPOCHS =100
BATCH_SIZE=256
gen = Generator(z_dim=Z_DIM, num_classes=NUM_CLASSES, label_embed_size=LABEL_EMBEDDING_SIZE, channels=CHANNELS)
dis = Discriminator(num_classes=NUM_CLASSES, channels=CHANNELS)



if LOAD_MODEL:
    gen.load_state_dict(torch.load(os.path.join(model_path,'gen.pth')))
    dis.load_state_dict(torch.load(os.path.join(model_path,'dis.pth')))


device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
gen = gen.to(device)
dis = dis.to(device)

loss_function = nn.BCELoss()


g_opt = optim.Adam(gen.parameters(), lr=0.0002, betas=(0.5, 0.999), weight_decay=2e-5)
d_opt = optim.Adam(dis.parameters(), lr=0.0002, betas=(0.5, 0.999), weight_decay=2e-5)



fixed_z = torch.randn(IMGS_TO_DISPLAY_PER_CLASS*NUM_CLASSES, Z_DIM)
fixed_label = torch.arange(0, NUM_CLASSES)
fixed_label = torch.repeat_interleave(fixed_label, IMGS_TO_DISPLAY_PER_CLASS)


fixed_label


      tensor([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2,
              2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4,
              4, 4, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 7, 7,
              7, 7, 7, 7, 7, 7, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 9, 9, 9, 9, 9, 9,
             9, 9, 9, 9])

real_label = torch.ones(BATCH_SIZE)
```