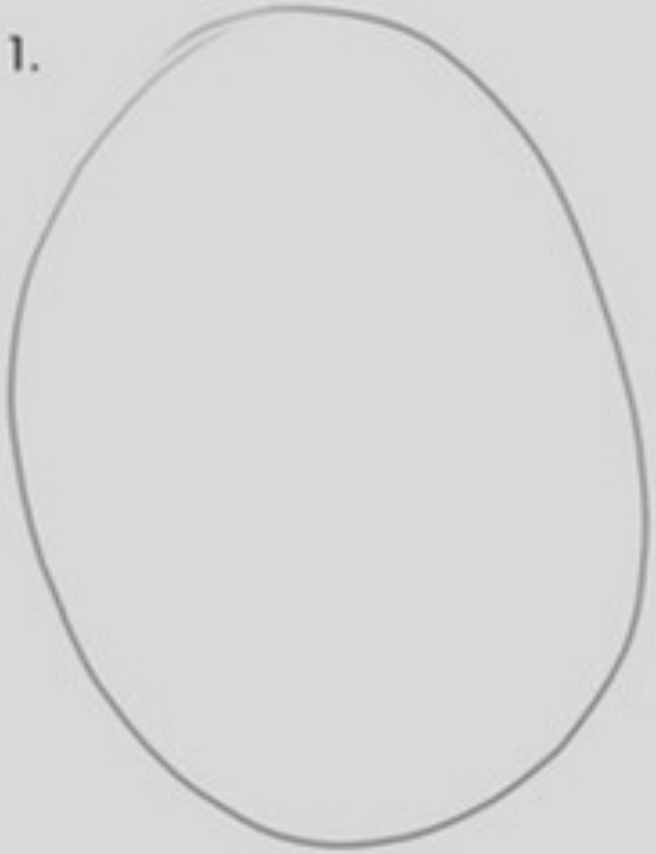
Coding papers in PYTORCH (or any other framework)

Soumith Chintala

How to draw a head

1.



1. Draw a circle

2.

2. Draw the rest of the head

Let’s discuss

- Unsupervised representation learning with deep convolutional generative adversarial networks
  Radford et. al.

Start with the experiments section

4 DETAILS OF ADVERSARIAL TRAINING

We trained DCGANs on three datasets, Large-scale Scene Understanding (LSUN) (Yu et al., 2015), Imagenet-1k and a newly assembled Faces dataset. Details on the usage of each of these datasets are given below.

No pre-processing was applied to training images besides scaling to the range of the tanh activation function [-1, 1]. All models were trained with mini-batch stochastic gradient descent (SGD) with a mini-batch size of 128. All weights were initialized from a zero-centered Normal distribution with standard deviation 0.02. In the LeakyReLU, the slope of the leak was set to 0.2 in all models. While previous GAN work has used momentum to accelerate training, we used the Adam optimizer (Kingma & Ba, 2014) with tuned hyperparameters. We found the suggested learning rate of 0.001, to be too high, using 0.0002 instead. Additionally, we found leaving the momentum term β_1 at the

Start with the experiments section

- Helps make a list of tasks

Start with the experiments section

- Which datasets were used?

Start with the experiments section

- Which datasets were used?
- architecture of the neural network (or model)

Start with the experiments section

- Which datasets were used?
- architecture of the neural network (or model)
- What is the loss function?

Start with the experiments section

- Which datasets were used?
- architecture of the neural network (or model)
- What is the loss function?

  If it is a standard function, only one or two words are said about it. For example: ″Cross-entropy loss is used″

Find Algorithm 1

- Some papers give easier to understand pseudo-code

Algorithm 1 WGAN with gradient penalty. We use default values of $\lambda = 10$, $n_{\text{critic}} = 5$, $\alpha = 0.0001$, $\beta_1 = 0$, $\beta_2 = 0.9$.

Require: The gradient penalty coefficient λ, the number of critic iterations per generator iteration n_{critic}, the batch size m, Adam hyperparameters α, β_1, β_2.
Require: initial critic parameters w_0, initial generator parameters θ_0.

while θ has not converged **do**
  for $t = 1, ..., n_{\text{critic}}$ **do**
    for $i = 1, ..., m$ **do**
      Sample real data $\boldsymbol{x} \sim \mathbb{P}_r$, latent variable $\boldsymbol{z} \sim p(\boldsymbol{z})$, a random number $\epsilon \sim U[0, 1]$.
      $\tilde{\boldsymbol{x}} \leftarrow G_\theta(\boldsymbol{z})$
      $\hat{\boldsymbol{x}} \leftarrow \epsilon \boldsymbol{x} + (1 - \epsilon)\tilde{\boldsymbol{x}}$
      $L^{(i)} \leftarrow D_w(\tilde{\boldsymbol{x}}) - D_w(\boldsymbol{x}) + \lambda(\|\nabla_{\hat{\boldsymbol{x}}} D_w(\hat{\boldsymbol{x}})\|_2 - 1)^2$
    end for
    $w \leftarrow \text{Adam}(\nabla_w \frac{1}{m} \sum_{i=1}^{m} L^{(i)}, w, \alpha, \beta_1, \beta_2)$
  end for
  Sample a batch of latent variables $\{\boldsymbol{z}^{(i)}\}_{i=1}^{m} \sim p(\boldsymbol{z})$.
  $\theta \leftarrow \text{Adam}(\nabla_\theta \frac{1}{m} \sum_{i=1}^{m} -D_w(G_\theta(\boldsymbol{z})), \theta, \alpha, \beta_1, \beta_2)$
end while

Write and verify dataset loader

- Before starting paper implementation!!!
-

Write and verify dataset loader

- Before starting paper implementation!!!
- Verify visually and with simple unit tests

Write and verify dataset loader

- Before starting paper implementation!!!
- Verify visually and with simple unit tests

Write and verify dataset loader

- Before starting paper implementation!!!
- Verify visually and with simple unit tests
- if __name__ == ″__main__″ is useful

Write and verify dataset loader

- Before starting paper implementation!!!
- Verify visually and with simple unit tests
- if __name__ == ″__main__″ is useful

```
126  if __name__ == '__main__':
127      #lsun = LSUNClassDataset(db_path='/home/soumith/local/lsun/train/bedroom_train_lmdb')
128      #a = lsun[0]
129      lsun = LSUNDataset(db_path='/home/soumith/local/lsun/train',
130                         classes=['bedroom_train', 'church_outdoor_train'])
131      print(lsun.classes)
132      print(lsun.dbs)
133      a, t = lsun[len(lsun)-1]
134      print(a)
135      print(t)
```

Starting with the paper

- Aim for the smallest dataset first

Starting with the paper

- Aim for the smallest dataset first
  quick iteration, faster initial development

Starting with the paper

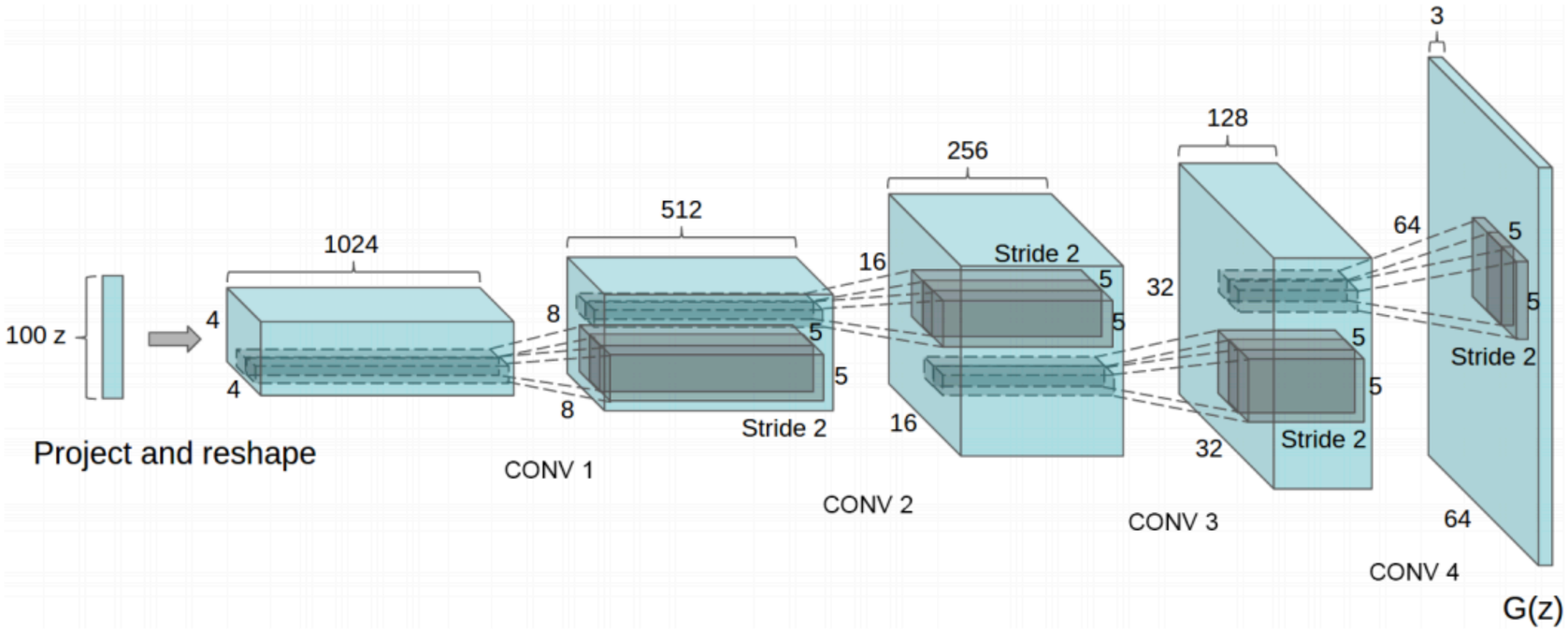
- Aim for the smallest dataset first
  quick iteration, faster initial development
  CIFAR10 is a good sanity check. MNIST is not!

Starting with the paper

- Aim for the smallest dataset first
- Implement the model

Model definition

- Aim for the smallest dataset first
- Implement the model



Model definition

- Aim for the smallest dataset first
- Implement the model

  Clear model details not available in most papers (sadly!)

Model definition

- Aim for the smallest dataset first
- Implement the model

  Clear model details not available in most papers (sadly!)

  See appendix

Model definition

- Aim for the smallest dataset first
- Implement the model

  Clear model details not available in most papers (sadly!)

  See appendix

  Sometimes, model definition is not easy:

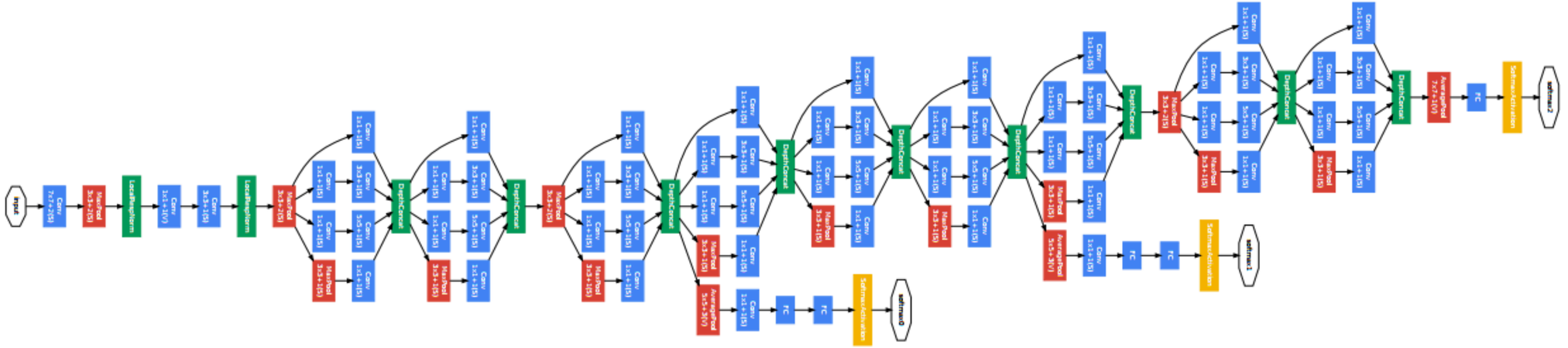
Model definition

- Aim for the smallest dataset first
- Implement the model

  Clear model details not available in most papers (sadly!)

  See appendix

  Son



Model definition

- Aim for the smallest dataset first
- Implement the model

  Clear model details not available in most papers (sadly!)

  See appendix

  Sometimes, model definition is not easy:

Model def

- Aim for the sr
- Implement th
  Clear model c (sadly!)
  See appendix
  Sometimes, m

```
class _netG(nn.Module):
    def __init__(self, ngpu):
        super(_netG, self).__init__()
        self.ngpu = ngpu
        self.main = nn.Sequential(
            # input is Z, going into a convolution
            nn.ConvTranspose2d(     nz, ngf * 8, 4, 1, 0, bias=False),
            nn.BatchNorm2d(ngf * 8),
            nn.ReLU(True),
            # state size. (ngf*8) x 4 x 4
            nn.ConvTranspose2d(ngf * 8, ngf * 4, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ngf * 4),
            nn.ReLU(True),
            # state size. (ngf*4) x 8 x 8
            nn.ConvTranspose2d(ngf * 4, ngf * 2, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ngf * 2),
            nn.ReLU(True),
            # state size. (ngf*2) x 16 x 16
            nn.ConvTranspose2d(ngf * 2,     ngf, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ngf),
            nn.ReLU(True),
            # state size. (ngf) x 32 x 32
            nn.ConvTranspose2d(    ngf,      nc, 4, 2, 1, bias=False),
            nn.Tanh()
            # state size. (nc) x 64 x 64
        )

    def forward(self, input):
        if isinstance(input.data, torch.cuda.FloatTensor) and self.ngpu > 1:
            output = nn.parallel.data_parallel(self.main, input, range(self.ngpu))
        else:
            output = self.main(input)
        return output
```

Model def

- Aim for the sr
- Implement th
  Clear model c
  See appendix
  Sometimes, n

```
class _netD(nn.Module):
    def __init__(self, ngpu):
        super(_netD, self).__init__()
        self.ngpu = ngpu
        self.main = nn.Sequential(
            # input is (nc) x 64 x 64
            nn.Conv2d(nc, ndf, 4, 2, 1, bias=False),
            nn.LeakyReLU(0.2, inplace=True),
            # state size. (ndf) x 32 x 32
            nn.Conv2d(ndf, ndf * 2, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ndf * 2),
            nn.LeakyReLU(0.2, inplace=True),
            # state size. (ndf*2) x 16 x 16
            nn.Conv2d(ndf * 2, ndf * 4, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ndf * 4),
            nn.LeakyReLU(0.2, inplace=True),
            # state size. (ndf*4) x 8 x 8
            nn.Conv2d(ndf * 4, ndf * 8, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ndf * 8),
            nn.LeakyReLU(0.2, inplace=True),
            # state size. (ndf*8) x 4 x 4
            nn.Conv2d(ndf * 8, 1, 4, 1, 0, bias=False),
            nn.Sigmoid()
        )

    def forward(self, input):
        if isinstance(input.data, torch.cuda.FloatTensor) and self.ngpu > 1:
            output = nn.parallel.data_parallel(self.main, input, range(self.ngpu))
        else:
            output = self.main(input)

        return output.view(-1, 1).squeeze(1)
```

; (sadly!)

Model de

- Aim for the s
- Implement th

  Clear model c ... (sadly!)

  See appendix

  Sometimes, n

```
class _netD(nn.Module):
    def __init__(self, ngpu):
        super(_netD, self).__init__()
        self.ngpu = ngpu
        self.main = nn.Sequential(
            # input is (nc) x 64 x 64
            nn.Conv2d(nc, ndf, 4, 2, 1, bias=False),
            nn.LeakyReLU(0.2, inplace=True),
            # state size. (ndf) x 32 x 32
            nn.Conv2d(ndf, ndf * 2, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ndf * 2),
            nn.LeakyReLU(0.2, inplace=True),
            # state size. (ndf*2) x 16 x 16
            nn.Conv2d(ndf * 2, ndf * 4, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ndf * 4)
```

WEIGHT INITIALIZATION IS IMPORTANT

```
            nn.Conv2d(ndf * 4, ndf * 8, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ndf * 8),
            nn.LeakyReLU(0.2, inplace=True),
            # state size. (ndf*8) x 4 x 4
            nn.Conv2d(ndf * 8, 1, 4, 1, 0, bias=False),
            nn.Sigmoid()
        )

    def forward(self, input):
        if isinstance(input.data, torch.cuda.FloatTensor) and self.ngpu > 1:
            output = nn.parallel.data_parallel(self.main, input, range(self.ngpu))
        else:
            output = self.main(input)

        return output.view(-1, 1).squeeze(1)
```

Training loop and optimizer

Find the correct hyper parameters from paper
- Hidden / hard-to-find in most papers

Training loop and optimizer

Find the correct hyper parameters from paper
- Hidden / hard-to-find in most papers

No pre-processing was applied to training images besides scaling to the range of the tanh activation function [-1, 1]. All models were trained with mini-batch stochastic gradient descent (SGD) with a mini-batch size of 128. All weights were initialized from a zero-centered Normal distribution with standard deviation 0.02. In the LeakyReLU, the slope of the leak was set to 0.2 in all models. While previous GAN work has used momentum to accelerate training, we used the Adam optimizer (Kingma & Ba, 2014) with tuned hyperparameters. We found the suggested learning rate of 0.001, to be too high, using 0.0002 instead. Additionally, we found leaving the momentum term β_1 at the suggested value of 0.9 resulted in training oscillation and instability while reducing it to 0.5 helped stabilize training.

```
# setup optimizer
optimizerD = optim.Adam(netD.parameters(), lr=opt.lr, betas=(opt.beta1, 0.999))
optimizerG = optim.Adam(netG.parameters(), lr=opt.lr, betas=(opt.beta1, 0.999))
```

Training loop and optimizer

```
for epoch in range(opt.niter):
    for i, data in enumerate(dataloader, 0):
```

Training loop and optimizer

```
for epoch in range(opt.niter):
    for i, data in enumerate(dataloader, 0):
        ############################
        # (1) Update D network: maximize log(D(x)) + log(1 - D(G(z)))
        ###########################
        # train with real
        netD.zero_grad()
        real_cpu, _ = data
        batch_size = real_cpu.size(0)
        if opt.cuda:
            real_cpu = real_cpu.cuda()
        input.resize_as_(real_cpu).copy_(real_cpu)
        label.resize_(batch_size).fill_(real_label)
        inputv = Variable(input)
        labelv = Variable(label)

        output = netD(inputv)
        errD_real = criterion(output, labelv)
        errD_real.backward()
        D_x = output.data.mean()
```

```
# train with fake
noise.resize_(batch_size, nz, 1, 1).normal_(0, 1)
noisev = Variable(noise)
fake = netG(noisev)
labelv = Variable(label.fill_(fake_label))
output = netD(fake.detach())
errD_fake = criterion(output, labelv)
errD_fake.backward()
D_G_z1 = output.data.mean()
errD = errD_real + errD_fake
optimizerD.step()
```

Training loop and optimizer

- Print statistics

Training loop and optimizer

- Print statistics
- Save checkpoints

Training loop and optimizer

- Print statistics
- Save checkpoints
- profile!

Finally!

Finally!

```
1  from __future__ import print_function
2  import argparse
3  import os
4  import random
5  import torch
6  import torch.nn as nn
7  import torch.nn.parallel
8  import torch.backends.cudnn as cudnn
9  import torch.optim as optim
10 import torch.utils.data
11 import torchvision.datasets as dset
12 import torchvision.transforms as transforms
13 import torchvision.utils as vutils
14 from torch.autograd import Variable
15
```

Finally!

```
17  parser = argparse.ArgumentParser()
18  parser.add_argument('--dataset', required=True, help='cifar10 | lsun | imagenet | folder | lfw | fake')
19  parser.add_argument('--dataroot', required=True, help='path to dataset')
20  parser.add_argument('--workers', type=int, help='number of data loading workers', default=2)
21  parser.add_argument('--batchSize', type=int, default=64, help='input batch size')
22  parser.add_argument('--imageSize', type=int, default=64, help='the height / width of the input image to network')
23  parser.add_argument('--nz', type=int, default=100, help='size of the latent z vector')
24  parser.add_argument('--ngf', type=int, default=64)
25  parser.add_argument('--ndf', type=int, default=64)
26  parser.add_argument('--niter', type=int, default=25, help='number of epochs to train for')
27  parser.add_argument('--lr', type=float, default=0.0002, help='learning rate, default=0.0002')
28  parser.add_argument('--beta1', type=float, default=0.5, help='beta1 for adam. default=0.5')
29  parser.add_argument('--cuda', action='store_true', help='enables cuda')
30  parser.add_argument('--ngpu', type=int, default=1, help='number of GPUs to use')
31  parser.add_argument('--netG', default='', help="path to netG (to continue training)")
32  parser.add_argument('--netD', default='', help="path to netD (to continue training)")
33  parser.add_argument('--outf', default='.', help='folder to output images and model checkpoints')
34  parser.add_argument('--manualSeed', type=int, help='manual seed')
35
```

Finally!

```
36  opt = parser.parse_args()
37  print(opt)
38
39  try:
40      os.makedirs(opt.outf)
41  except OSError:
42      pass
43
44  if opt.manualSeed is None:
45      opt.manualSeed = random.randint(1, 10000)
46  print("Random Seed: ", opt.manualSeed)
47  random.seed(opt.manualSeed)
48  torch.manual_seed(opt.manualSeed)
49  if opt.cuda:
50      torch.cuda.manual_seed_all(opt.manualSeed)
51
52  cudnn.benchmark = True
53
54  if torch.cuda.is_available() and not opt.cuda:
55      print("WARNING: You have a CUDA device, so you should probably run with --cuda")
56
```

Finally

```
if opt.dataset in ['imagenet', 'folder', 'lfw']:
    # folder dataset
    dataset = dset.ImageFolder(root=opt.dataroot,
                               transform=transforms.Compose([
                                   transforms.Resize(opt.imageSize),
                                   transforms.CenterCrop(opt.imageSize),
                                   transforms.ToTensor(),
                                   transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
                               ]))
elif opt.dataset == 'lsun':
    dataset = dset.LSUN(db_path=opt.dataroot, classes=['bedroom_train'],
                        transform=transforms.Compose([
                            transforms.Resize(opt.imageSize),
                            transforms.CenterCrop(opt.imageSize),
                            transforms.ToTensor(),
                            transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
                        ]))
elif opt.dataset == 'cifar10':
    dataset = dset.CIFAR10(root=opt.dataroot, download=True,
                           transform=transforms.Compose([
                               transforms.Resize(opt.imageSize),
                               transforms.ToTensor(),
                               transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
                           ]))
elif opt.dataset == 'fake':
    dataset = dset.FakeData(image_size=(3, opt.imageSize, opt.imageSize),
                            transform=transforms.ToTensor())
assert dataset
dataloader = torch.utils.data.DataLoader(dataset, batch_size=opt.batchSize,
                                         shuffle=True, num_workers=int(opt.workers))
```

Finally!

```
88  ngpu = int(opt.ngpu)
89  nz = int(opt.nz)
90  ngf = int(opt.ngf)
91  ndf = int(opt.ndf)
92  nc = 3
93
94
95  # custom weights initialization called on netG and netD
96  def weights_init(m):
97      classname = m.__class__.__name__
98      if classname.find('Conv') != -1:
99          m.weight.data.normal_(0.0, 0.02)
100     elif classname.find('BatchNorm') != -1:
101         m.weight.data.normal_(1.0, 0.02)
102         m.bias.data.fill_(0)
103
```

Finally!

```
105  class _netG(nn.Module):
106      def __init__(self, ngpu):
107          super(_netG, self).__init__()
108          self.ngpu = ngpu
109          self.main = nn.Sequential(
110              # input is Z, going into a convolution
111              nn.ConvTranspose2d(     nz, ngf * 8, 4, 1, 0, bias=False),
112              nn.BatchNorm2d(ngf * 8),
113              nn.ReLU(True),
114              # state size. (ngf*8) x 4 x 4
115              nn.ConvTranspose2d(ngf * 8, ngf * 4, 4, 2, 1, bias=False),
116              nn.BatchNorm2d(ngf * 4),
117              nn.ReLU(True),
118              # state size. (ngf*4) x 8 x 8
119              nn.ConvTranspose2d(ngf * 4, ngf * 2, 4, 2, 1, bias=False),
120              nn.BatchNorm2d(ngf * 2),
121              nn.ReLU(True),
122              # state size. (ngf*2) x 16 x 16
123              nn.ConvTranspose2d(ngf * 2,     ngf, 4, 2, 1, bias=False),
124              nn.BatchNorm2d(ngf),
125              nn.ReLU(True),
126              # state size. (ngf) x 32 x 32
127              nn.ConvTranspose2d(    ngf,      nc, 4, 2, 1, bias=False),
128              nn.Tanh()
129              # state size. (nc) x 64 x 64
130          )
131
132      def forward(self, input):
133          if isinstance(input.data, torch.cuda.FloatTensor) and self.ngpu > 1:
134              output = nn.parallel.data_parallel(self.main, input, range(self.ngpu))
135          else:
136              output = self.main(input)
137          return output
```

Finally!

```
140  netG = _netG(ngpu)
141  netG.apply(weights_init)
142  if opt.netG != '':
143      netG.load_state_dict(torch.load(opt.netG))
144  print(netG)
145
```

Finally!

```
147  class _netD(nn.Module):
148      def __init__(self, ngpu):
149          super(_netD, self).__init__()
150          self.ngpu = ngpu
151          self.main = nn.Sequential(
152              # input is (nc) x 64 x 64
153              nn.Conv2d(nc, ndf, 4, 2, 1, bias=False),
154              nn.LeakyReLU(0.2, inplace=True),
155              # state size. (ndf) x 32 x 32
156              nn.Conv2d(ndf, ndf * 2, 4, 2, 1, bias=False),
157              nn.BatchNorm2d(ndf * 2),
158              nn.LeakyReLU(0.2, inplace=True),
159              # state size. (ndf*2) x 16 x 16
160              nn.Conv2d(ndf * 2, ndf * 4, 4, 2, 1, bias=False),
161              nn.BatchNorm2d(ndf * 4),
162              nn.LeakyReLU(0.2, inplace=True),
163              # state size. (ndf*4) x 8 x 8
164              nn.Conv2d(ndf * 4, ndf * 8, 4, 2, 1, bias=False),
165              nn.BatchNorm2d(ndf * 8),
166              nn.LeakyReLU(0.2, inplace=True),
167              # state size. (ndf*8) x 4 x 4
168              nn.Conv2d(ndf * 8, 1, 4, 1, 0, bias=False),
169              nn.Sigmoid()
170          )
171
172      def forward(self, input):
173          if isinstance(input.data, torch.cuda.FloatTensor) and self.ngpu > 1:
174              output = nn.parallel.data_parallel(self.main, input, range(self.ngpu))
175          else:
176              output = self.main(input)
177
178          return output.view(-1, 1).squeeze(1)
```

Finally!

```
181   netD = _netD(ngpu)
182   netD.apply(weights_init)
183   if opt.netD != '':
184       netD.load_state_dict(torch.load(opt.netD))
185   print(netD)
186
```

Finally!

```
187    criterion = nn.BCELoss()
```

Finally!

```
189  input = torch.FloatTensor(opt.batchSize, 3, opt.imageSize, opt.imageSize)
190  noise = torch.FloatTensor(opt.batchSize, nz, 1, 1)
191  fixed_noise = torch.FloatTensor(opt.batchSize, nz, 1, 1).normal_(0, 1)
192  label = torch.FloatTensor(opt.batchSize)
193  real_label = 1
194  fake_label = 0
195
196  if opt.cuda:
197      netD.cuda()
198      netG.cuda()
199      criterion.cuda()
200      input, label = input.cuda(), label.cuda()
201      noise, fixed_noise = noise.cuda(), fixed_noise.cuda()
202
203  fixed_noise = Variable(fixed_noise)
```

Finally!

```
205   # setup optimizer
206   optimizerD = optim.Adam(netD.parameters(), lr=opt.lr, betas=(opt.beta1, 0.999))
207   optimizerG = optim.Adam(netG.parameters(), lr=opt.lr, betas=(opt.beta1, 0.999))
208
```

Finally!

```
for epoch in range(opt.niter):
    for i, data in enumerate(dataloader, 0):
        ############################
        # (1) Update D network: maximize log(D(x)) + log(1 - D(G(z)))
        ###########################
        # train with real
        netD.zero_grad()
        real_cpu, _ = data
        batch_size = real_cpu.size(0)
        if opt.cuda:
            real_cpu = real_cpu.cuda()
        input.resize_as_(real_cpu).copy_(real_cpu)
        label.resize_(batch_size).fill_(real_label)
        inputv = Variable(input)
        labelv = Variable(label)

        output = netD(inputv)
        errD_real = criterion(output, labelv)
        errD_real.backward()
        D_x = output.data.mean()

        # train with fake
        noise.resize_(batch_size, nz, 1, 1).normal_(0, 1)
        noisev = Variable(noise)
        fake = netG(noisev)
        labelv = Variable(label.fill_(fake_label))
        output = netD(fake.detach())
        errD_fake = criterion(output, labelv)
        errD_fake.backward()
        D_G_z1 = output.data.mean()
        errD = errD_real + errD_fake
        optimizerD.step()
```

Final

```
242         ############################
243         # (2) Update G network: maximize log(D(G(z)))
244         ###########################
245         netG.zero_grad()
246         labelv = Variable(label.fill_(real_label))  # fake labels are real for generator cost
247         output = netD(fake)
248         errG = criterion(output, labelv)
249         errG.backward()
250         D_G_z2 = output.data.mean()
251         optimizerG.step()
252
253         print('[%d/%d][%d/%d] Loss_D: %.4f Loss_G: %.4f D(x): %.4f D(G(z)): %.4f / %.4f'
254               % (epoch, opt.niter, i, len(dataloader),
255                  errD.data[0], errG.data[0], D_x, D_G_z1, D_G_z2))
256         if i % 100 == 0:
257             vutils.save_image(real_cpu,
258                     '%s/real_samples.png' % opt.outf,
259                     normalize=True)
260             fake = netG(fixed_noise)
261             vutils.save_image(fake.data,
262                     '%s/fake_samples_epoch_%03d.png' % (opt.outf, epoch),
263                     normalize=True)
264
265     # do checkpointing
266     torch.save(netG.state_dict(), '%s/netG_epoch_%d.pth' % (opt.outf, epoch))
267     torch.save(netD.state_dict(), '%s/netD_epoch_%d.pth' % (opt.outf, epoch))
```