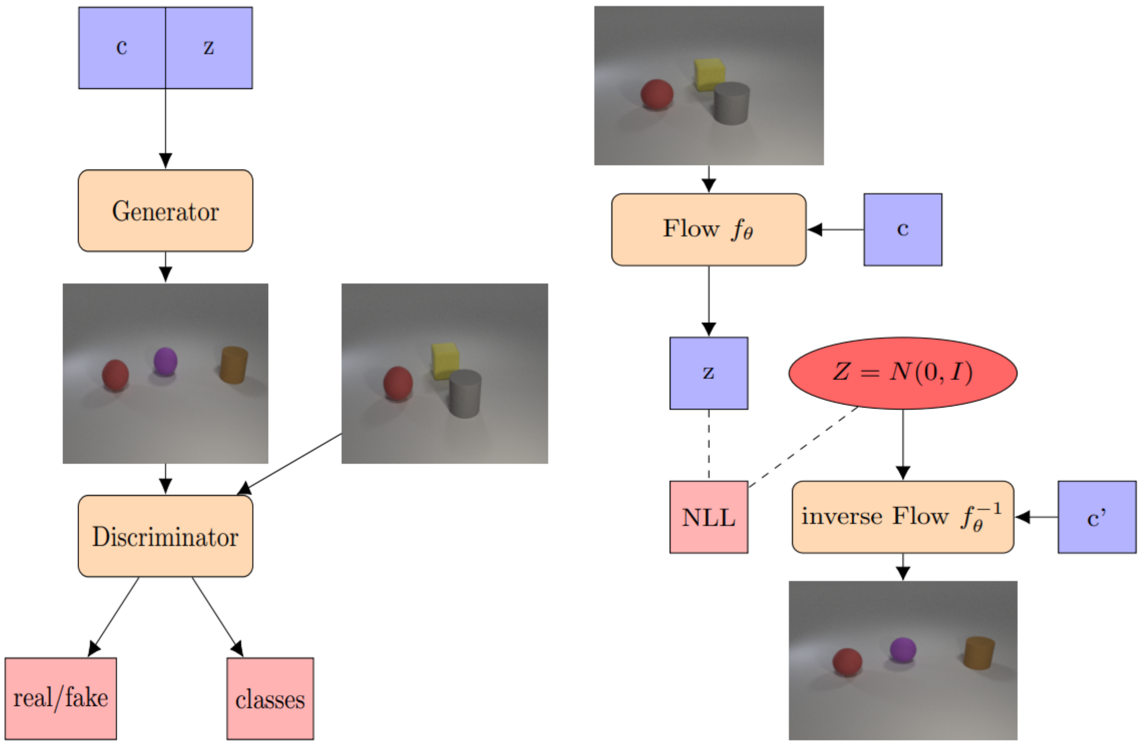
Deep Learning and Practice

Lab 7　　　　　309505018 郭俊廷

1. Introduction (10%)

訓練兩種generator, 一種是Conditional GAN, 另一種是Normalizing Flow models, 並生成出有特定條件的圖片, training data為ICLEVR的幾何物體圖片, 總共有24種不同的幾何物體, 因此condition為一個24 dimention vector, 向量裡的每個數值都是0~1.



(a) Illustration of training procedure using cGAN.　(b) Illustration of training procedure using cNF.

Figure 1: Illustration of training procedure for object images generation

2. Implementation details (15%)

1) Describe how you implement your model, including your choices in Section 3.1 & 3.2. (10%)

- GAN:
  我使用 Conditional Deep Convolutional GAN (cDCGAN) 作為model architecture.

  Generator的部分會讓 24-dim 的conditional vector經過fully connected layer擴充成200-dim ,然後把 200-dim conditional vector與100-dim雜訊z concatenate起來, 成為一個300-dim的vector, 然後連續做5次的 transposed convolution 變成fake image

```python
class Generator(nn.Module):
    def __init__(self,z_dim,c_dim):
        super(Generator,self).__init__()
        self.z_dim=z_dim
        self.c_dim=c_dim
        self.conditionExpand=nn.Sequential(
            nn.Linear(24,c_dim),
            nn.ReLU()
        )
        kernel_size=(4,4)
        channels=[z_dim+c_dim,512,256,128,64]
        paddings=[(0,0),(1,1),(1,1),(1,1)]
        for i in range(1,len(channels)):
            setattr(self,'convT'+str(i),nn.Sequential(
                nn.ConvTranspose2d(channels[i-1],channels[i],kernel_size,stride=(2,2),
                                                                    padding=paddings[i-1]),
                nn.BatchNorm2d(channels[i]),
                nn.ReLU()
            ))
        self.convT5=nn.ConvTranspose2d(64,3,kernel_size,stride=(2,2),padding=(1,1))
        self.tanh=nn.Tanh()

    def forward(self,z,c):
        """
        :param z: (batch_size,100) tensor
        :param c: (batch_size,24) tensor
        :return: (batch_size,3,64,64) tensor
        """
        z=z.view(-1,self.z_dim,1,1)
        c=self.conditionExpand(c).view(-1,self.c_dim,1,1)
        out=torch.cat((z,c),dim=1)  # become(N,z_dim+c_dim,1,1)
        out=self.convT1(out)  # become(N,512,4,4)
        out=self.convT2(out)  # become(N,256,8,8)
        out=self.convT3(out)  # become(N,128,16,16)
        out=self.convT4(out)  # become(N,64,32,32)
        out=self.convT5(out)  # become(N,3,64,64)
        out=self.tanh(out)    # output value between [-1,+1]
        return out
```

Discriminator的部分會讓24-dim的condition vector 經由 fully connected layer擴充資訊, 然後再reshape變成(1,64,64)的圖片, 之後再與training data或generator生成出來的圖片concatenate起來變成(4,64,64)的圖片, 再連續做5次convolution+BatchNormalized+Leaky ReLU就可以得到一個scalar做output
由於output為一個代表是否為真的照片的scalar, 所以 loss function使用binary cross entropy

```
class Discriminator(nn.Module):
    def __init__(self,img_shape,c_dim):
        super(Discriminator, self).__init__()
        self.H,self.W,self.C=img_shape
        self.conditionExpand=nn.Sequential(
            nn.Linear(24,self.H*self.W*1),
            nn.LeakyReLU()
        )
        kernel_size=(4,4)
        channels=[4,64,128,256,512]
        for i in range(1,len(channels)):
            setattr(self,'conv'+str(i),nn.Sequential(
                nn.Conv2d(channels[i-1],channels[i],kernel_size,stride=(2,2),padding=(1,1)),
                nn.BatchNorm2d(channels[i]),
                nn.LeakyReLU()
            ))
        self.conv5=nn.Conv2d(512,1,kernel_size,stride=(1,1))
        self.sigmoid=nn.Sigmoid()

    def forward(self,X,c):
        """
        :param X: (batch_size,3,64,64) tensor
        :param c: (batch_size,24) tensor
        :return: (batch_size) tensor
        """
        c=self.conditionExpand(c).view(-1,1,self.H,self.W)
        out=torch.cat((X,c),dim=1)  # become(N,4,64,64)
        out=self.conv1(out)  # become(N,64,32,32)
        out=self.conv2(out)  # become(N,128,16,16)
        out=self.conv3(out)  # become(N,256,8,8)
        out=self.conv4(out)  # become(N,512,4,4)
        out=self.conv5(out)  # become(N,1,1,1)
        out=self.sigmoid(out)  # output value between [0,1]
        out=out.view(-1)
        return out
```

- Normalizing Flow Model:
  我使用 SRFlow (Super-Resolution Space with Normalizing Flow) 作為model architecture.

  Normalizing Flow models 的架構是input一張圖片加上condition, 經過function後得到z, z再經過 inverse function得到新的一張圖片, 跟GAN相比的優點在於生成的圖片會跟原始圖片比較接近,
  我的fordward pass可以分為normal_flow跟reverse_flow

```python
def forward(self, gt=None, lr=None, z=None, eps_std=None, reverse=False, epses=None, reverse_with_grad=False,
    lr_enc=None,add_gt_noise=False, step=None, y_label=None):

    if not reverse:
        return self.normal_flow(gt, lr, epses=epses, lr_enc=lr_enc, add_gt_noise=add_gt_noise, step=step,
                                y_onehot=y_label)
    else:
        # assert lr.shape[0] == 1
        assert lr.shape[1] == 3
        # assert lr.shape[2] == 20
        # assert lr.shape[3] == 20
        # assert z.shape[0] == 1
        # assert z.shape[1] == 3 * 8 * 8
        # assert z.shape[2] == 20
        # assert z.shape[3] == 20
        if reverse_with_grad:
            return self.reverse_flow(lr, z, y_onehot=y_label, eps_std=eps_std, epses=epses,
                lr_enc=lr_enc,add_gt_noise=add_gt_noise)
        else:
            with torch.no_grad():
                return self.reverse_flow(lr, z, y_onehot=y_label, eps_std=eps_std, epses=epses,
                    lr_enc=lr_enc,add_gt_noise=add_gt_noise)
def normal_flow(self, gt, lr, y_onehot=None, epses=None, lr_enc=None, add_gt_noise=True, step=None):
    if lr_enc is None:
        lr_enc = self.rrdbPreprocessing(lr)

    logdet = torch.zeros_like(gt[:, 0, 0, 0])
    pixels = thops.pixels(gt)
    z = gt

    if add_gt_noise:
        # Setup
        noiseQuant = opt_get(self.opt, ['network_G', 'flow', 'augmentation', 'noiseQuant'], True)
        if noiseQuant:
            z = z + ((torch.rand(z.shape, device=z.device) - 0.5) / self.quant)
        logdet = logdet + float(-np.log(self.quant) * pixels)

    # Encode
    epses, logdet = self.flowUpsamplerNet(rrdbResults=lr_enc, gt=z, logdet=logdet, reverse=False, epses=epses,
                                          y_onehot=y_onehot)
    objective = logdet.clone()

    if isinstance(epses, (list, tuple)):
        z = epses[-1]
    else:
        z = epses

    objective = objective + flow.GaussianDiag.logp(None, None, z)
    nll = (-objective) / float(np.log(2.) * pixels)

    if isinstance(epses, list):
        return epses, nll, logdet
    return z, nll, logdet
def reverse_flow(self, lr, z, y_onehot, eps_std, epses=None, lr_enc=None, add_gt_noise=True):
    logdet = torch.zeros_like(lr[:, 0, 0, 0])
    pixels = thops.pixels(lr) * self.opt['scale'] ** 2

    if add_gt_noise:
        logdet = logdet - float(-np.log(self.quant) * pixels)

    if lr_enc is None:
        lr_enc = self.rrdbPreprocessing(lr)

    x, logdet = self.flowUpsamplerNet(rrdbResults=lr_enc, z=z, eps_std=eps_std, reverse=True, epses=epses,
                                      logdet=logdet)

    return x, logdet
```

其中normal flow的generator是用三個ResidualDenseBlock的做fordward, 每一個layer是用五個convolutional加LeakyReLU

```python
class ResidualDenseBlock_5C(nn.Module):
    def __init__(self, nf=64, gc=32, bias=True):
        super(ResidualDenseBlock_5C, self).__init__()
        # gc: growth channel, i.e. intermediate channels
        self.conv1 = nn.Conv2d(nf, gc, 3, 1, 1, bias=bias)
        self.conv2 = nn.Conv2d(nf + gc, gc, 3, 1, 1, bias=bias)
        self.conv3 = nn.Conv2d(nf + 2 * gc, gc, 3, 1, 1, bias=bias)
        self.conv4 = nn.Conv2d(nf + 3 * gc, gc, 3, 1, 1, bias=bias)
        self.conv5 = nn.Conv2d(nf + 4 * gc, nf, 3, 1, 1, bias=bias)
        self.lrelu = nn.LeakyReLU(negative_slope=0.2, inplace=True)

        # initialization
        mutil.initialize_weights([self.conv1, self.conv2, self.conv3, self.conv4, self.conv5], 0.1)

    def forward(self, x):
        x1 = self.lrelu(self.conv1(x))
        x2 = self.lrelu(self.conv2(torch.cat((x, x1), 1)))
        x3 = self.lrelu(self.conv3(torch.cat((x, x1, x2), 1)))
        x4 = self.lrelu(self.conv4(torch.cat((x, x1, x2, x3), 1)))
        x5 = self.conv5(torch.cat((x, x1, x2, x3, x4), 1))
        return x5 * 0.2 + x


class RRDB(nn.Module):
    '''Residual in Residual Dense Block'''

    def __init__(self, nf, gc=32):
        super(RRDB, self).__init__()
        self.RDB1 = ResidualDenseBlock_5C(nf, gc)
        self.RDB2 = ResidualDenseBlock_5C(nf, gc)
        self.RDB3 = ResidualDenseBlock_5C(nf, gc)

    def forward(self, x):
        out = self.RDB1(x)
        out = self.RDB2(out)
        out = self.RDB3(out)
        return out * 0.2 + x
```

2) Specify the hyperparameters (learning rate, epochs, etc.). (5%)

- GAN:
  z=100-dim, conditional vector= 200-dim, image_shape=(64,64,3)
  epochs=500, learning rate=0.0002, batch_size=64

- Normalizing Flow Model:
  z=100-dim, conditional vector= 200-dim, image_shape=(64,64,3)
  epochs=300, learning rate=0.001, batch_size=64

3. Task 1 (45%)

1) Result (generated images) (5%)

avg score: 0.71

2) Classification accuracy on test.json (5% for each model)

- GAN: avg score: 0.71

```
(dlp) jackkuo@lab708-Default-string:~/lab7$ python3 evaluate\ model.py
score:  0.72
score:  0.68
score:  0.68
score:  0.74
score:  0.69
score:  0.72
score:  0.71
score:  0.71
score:  0.72
score:  0.71

avg score:  0.71
```

- Normalizing Flow Model: avg score: 0.70

```
(dlp) jackkuo@lab708-Default-string:~/lab7$ python3 evaluate\ model.py
score:  0.72
score:  0.67
score:  0.69
score:  0.71
score:  0.67
score:  0.71
score:  0.74
score:  0.71
score:  0.68
score:  0.68

avg score:  0.70
(dlp) jackkuo@lab708-Default-string:~/lab7$
```

3) Classification accuracy on new_test.json (10% for each model)

- GAN: avg score: 0.65

```
(dlp) jackkuo@lab708-Default-string:~/lab7$ python3 evaluate\ model.py
score: 0.64
score: 0.64
score: 0.65
score: 0.63
score: 0.65
score: 0.67
score: 0.64
score: 0.64
score: 0.64
score: 0.67

avg score: 0.65
```

- Normalizing Flow Model: avg score: 0.64

```
(dlp) jackkuo@lab708-Default-string:~/lab7$ python3 evaluate\ model.py
score: 0.61
score: 0.65
score: 0.63
score: 0.65
score: 0.64
score: 0.65
score: 0.64
score: 0.67
score: 0.65
score: 0.63

avg score: 0.64
(dlp) jackkuo@lab708-Default-string:~/lab7$
```

4) Discuss the results of different models architectures (10%)

1. Normalizing Flow models所生成的 output 會跟 input 比較接近, 原因是他生成方式是通過input通過函式f 再通過 inverse f, 所以生成圖片會比GAN更接近原始圖片

2. Normalizing Flow只有一個 Loss, 而GAN有兩個Loss, Generator跟discriminator各一個Loss, 所以Normalizing Flow Loss會比較不穩定, 而GAN在前期會希望兩個loss越接近越好, 所以我的Generator跟discriminator訓練次數設成4:1

3. GAN加入Batch Normalize會使表現比較好

4. Activate function, Generator要用ReLU, Discriminator要用Leaky ReLU

5.感覺Normalizing Flow models應該要表現得比GAN好, 可能我有那裡寫錯了所以score怪怪的

4. Task 2 (30%)

1) **Conditional face generation: at least 4 images with at least 3 conditions(10%)**

2) **Linear interpolation: 3 pairs of images with at least 5-image interpolation(10%)**

3) **Attribute manipulation: At least 2 attributes of same image(10%)**