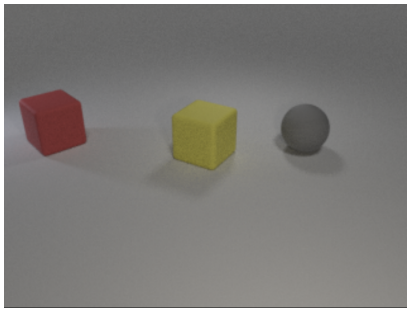
Let’s Play GANs

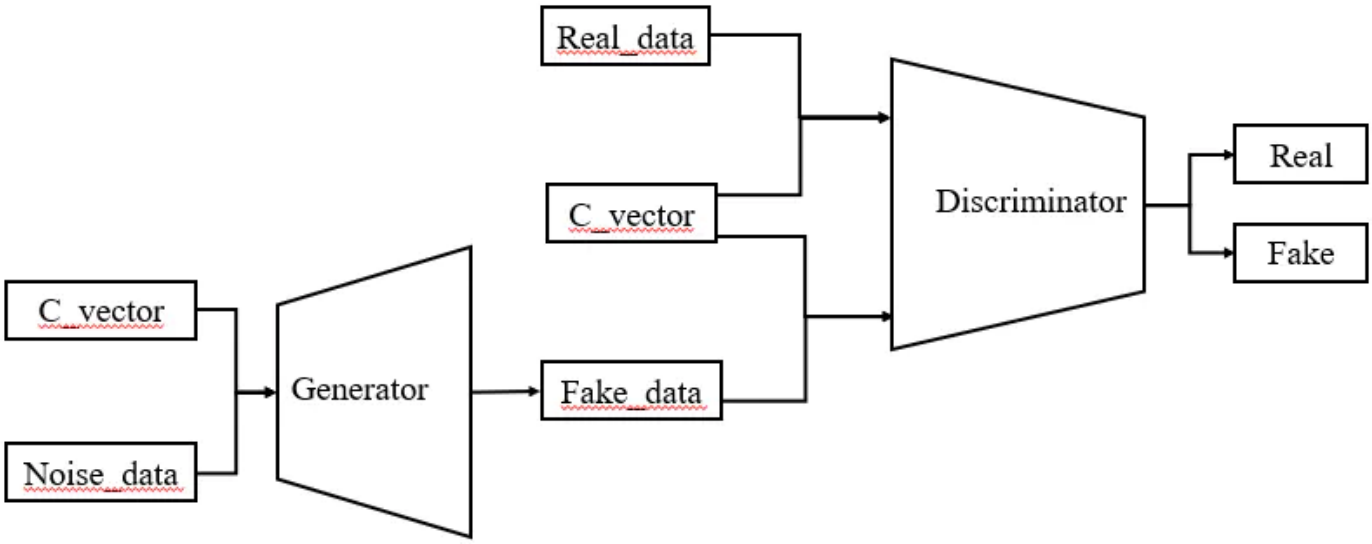
• Introduction (5%)

在這個實驗中，我們使用conditional GAN(CGAN)訓練一個可以根據不同的條件生成合成圖像的model。如，輸入"紅色立方體"和"藍色圓柱體"後，模型便可以生成包含"紅色立方體"和"藍色圓柱體"的高清合成圖像。

training data為包含24種不同物體(幾何形狀*顏色=24)的圖片，而condition則是為一個 24-dim 的vector。如: [0,0,0,1,0,0,1,.....0,0,0]。



▲training data



▲conditional GAN(CGAN)

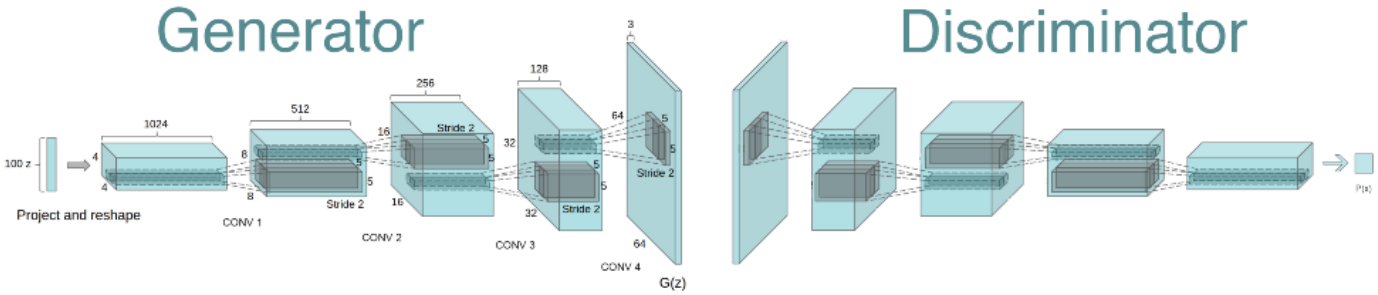
CGAN與GAN的主要區別為:

CGAN在生成器和判別器的輸入數據中都加入類別標籤向量（C_vector），也就是說GAN所生成的內容是隨機的，而CGAN則是可以實現根據輸入標籤生成指定類別的內容。

• Implementation details (15%)

– Describe how you implement your model, including your choice of cGAN, model architectures, and loss functions. (10%)

我採用的model為DCGAN



DCGAN是一個 GAN 的變形，DCGAN 就是結合 CNN、GAN 的一種模型。其主要是利用反卷積網路(Deconvolution network)反覆生成圖像，再將生成圖像放入 GAN 模型中執行，不斷訓練，最後產出真偽難辨的圖像。

Generator:
1.經過第一層反捲積後
2.把latent vector 和 condition vector concat起來
3.再經過4層反捲積
4.得到 64 x 64 x 3 的圖像。

```
class Generator(nn.Module):
    def __init__(self, d=128):
        super(Generator, self).__init__()
        self.deconv1_1 = nn.ConvTranspose2d(opt.latent_dim, d*4, 4, 1, 0) #(stride ,padding
,output_padding ,dilation(controls the spacing between the kernel points) ,groups )
        self.deconv1_1_bn = nn.BatchNorm2d(d*4) #對輸入的批數據進行歸一化，映射到均值為 0 ，方差為 1 的正態分
佈。

        self.deconv1_2 = nn.ConvTranspose2d(opt.n_classes, d*4, 4, 1, 0)
        self.deconv1_2_bn = nn.BatchNorm2d(d*4)

        self.deconv2 = nn.ConvTranspose2d(d*8, d*4, 4, 2, 1)
        self.deconv2_bn = nn.BatchNorm2d(d*4)

        self.deconv3 = nn.ConvTranspose2d(d*4, d*2, 4, 2, 1)
        self.deconv3_bn = nn.BatchNorm2d(d*2)

        self.deconv4 = nn.ConvTranspose2d(d*2, d, 4, 2, 1)
        self.deconv4_bn = nn.BatchNorm2d(d)

        self.deconv5 = nn.ConvTranspose2d(d, 3, 4, 2, 1)

    def weight_init(self, mean, std):
        for m in self._modules:
            normal_init(self._modules[m], mean, std)

    def forward(self, input, label):
        input = input.view(-1, opt.latent_dim, 1, 1) #參數中的-1就代表這個位置由其他位置的數字來推斷
        label = label.view(-1, opt.n_classes, 1, 1)
        x = F.leaky_relu(self.deconv1_1_bn(self.deconv1_1(input)), 0.2)
        y = F.leaky_relu(self.deconv1_2_bn(self.deconv1_2(label)), 0.2)

        x = torch.cat([x, y], 1)
        x = F.leaky_relu(self.deconv2_bn(self.deconv2(x)), 0.2)
```

```
        x = F.leaky_relu(self.deconv3_bn(self.deconv3(x)), 0.2)
        x = F.leaky_relu(self.deconv4_bn(self.deconv4(x)), 0.2)
        x = torch.tanh(self.deconv5(x))
        return x
```

Discriminator:
1.圖像向量(input)和條件向量(label)經過第一層捲積
2.把圖像向量(input)和條件向量(label)concatenated
3.再經過四層卷積
4.輸出一個scalar，指示圖像是否真實並匹配條件。

```
class Discriminator(nn.Module):

    def __init__(self, d=128):
        super(Discriminator, self).__init__()
        self.conv1_1 = nn.Conv2d(3, int(d/2), 4, 2, 1)
        self.conv1_2 = nn.Conv2d(opt.n_classes, int(d/2), 4, 2, 1)
        self.conv2 = nn.Conv2d(d, d*2, 4, 2, 1)
        self.conv2_bn = nn.BatchNorm2d(d*2)
        self.conv3 = nn.Conv2d(d*2, d*4, 4, 2, 1)
        self.conv3_bn = nn.BatchNorm2d(d*4)
        self.conv4 = nn.Conv2d(d*4, d*8, 4, 2, 1)
        self.conv4_bn = nn.BatchNorm2d(d*8)
        self.conv5 = nn.Conv2d(d*8, 1, 4, 1, 0)

    def weight_init(self, mean, std):
        for m in self._modules:
            normal_init(self._modules[m], mean, std)
    def forward(self, input, label):
        x = F.leaky_relu(self.conv1_1(input), 0.2)
        y = F.leaky_relu(self.conv1_2(label), 0.2)
        x = torch.cat([x, y], 1)
        x = F.leaky_relu(self.conv2_bn(self.conv2(x)), 0.2)
        x = F.leaky_relu(self.conv3_bn(self.conv3(x)), 0.2)
        x = F.leaky_relu(self.conv4_bn(self.conv4(x)), 0.2)
        x = torch.sigmoid(self.conv5(x))
        return x.view(-1, 1)
```

Loss Function:

基本上二分法都是用 BCELoss

```
# Loss functions
adversarial_loss = torch.nn.BCELoss()
```

discriminator loss:
在cGAN的discriminator 中應該考慮三部分損失：
1.清晰圖片+正確標籤的loss
2.清晰圖片+錯誤標籤的loss
3.生成noise做 generator input

discriminator判別器的loss = (清晰圖片+正確標籤的loss) + [(清晰圖片+錯誤標籤的loss)+(不清晰圖片的loss)]/2

```
#清晰圖片+正確標籤的loss
validity_real_correct = discriminator(real_imgs, real_fill)
d_real_loss_correct = adversarial_loss(validity_real_correct, valid)

#清晰圖片+錯誤標籤的loss
validity_real_mismatch = discriminator(real_imgs, mismatch_fill)
d_real_loss_mismatch = adversarial_loss(validity_real_mismatch, mismatch_fake)

# 生成noise做 generator input
z = FloatTensor(np.random.normal(0, 1, (batch_size, opt.latent_dim)))

#不清晰圖片的loss
gen_imgs = generator(z, labels)
validity_fake = discriminator(gen_imgs, real_fill)
d_fake_loss = adversarial_loss(validity_fake, fake)

# discriminator判別器的loss = (清晰圖片+正確標籤的loss) + [(清晰圖片+錯誤標籤的loss)+(不清晰圖片的loss)]/2
d_loss = d_real_loss_correct + (d_real_loss_mismatch + d_fake_loss)/2
```

generator loss:

```
 # generator loss: 衡量generator欺騙discriminator的能力
                validity = discriminator(gen_imgs, real_fill)
                g_loss = adversarial_loss(validity, valid)
```

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

training epochs = 100

batch size = 128

learning rate = 0.0002

```
parser = argparse.ArgumentParser()
parser.add_argument("--n_epochs", type=int, default=100, help="number of epochs of training")
parser.add_argument("--batch_size", type=int, default=128, help="size of the batches")
parser.add_argument("--lr", type=float, default=0.0002, help="adam: learning rate")
parser.add_argument("--b1", type=float, default=0.5, help="adam: decay of first order momentum of
gradient")
parser.add_argument("--b2", type=float, default=0.999, help="adam: decay of first order momentum of
gradient")
parser.add_argument("--n_cpu", type=int, default=4, help="number of cpu threads to use during batch
generation")
parser.add_argument("--latent_dim", type=int, default=100, help="dimensionality of the latent space")
parser.add_argument("--n_classes", type=int, default=24, help="number of classes for dataset")
opt = parser.parse_args()
```

• Results and discussion (30%)

– Show your results based on the testing data. (5%) (including images)

```
generated image accuracy: 0.652778
```

– Discuss the results of different models architectures. (25%) For example, what is the effect with or without some specific loss terms, or what kinds of condition design is more effective to help cGAN.
有試過Generator和Discriminator都用RELU的不過後來發現Generator一樣用RELU，而Discriminator改用LeakyRelu效果會比較好。