- Introduction

本次作業是要寫個CGAN。也就是將label和noise輸入進generator產生fake picture，true/fake picture和label輸入進discrimminator產生true/fake。透過上述的過程，來學習如何訓練GAN，用GAN產生圖片和學習如何寫GAN，並了解實作細節。

- Implementation details

- Describe how you implement your model, including your choice of cGAN, model architectures, and loss functions.

我的模型就用最簡單的ACGAN。

ACGAN的model architecture如下：

Generator:

Generator就是先將25*1*1的noise和24*1*1的label concat起來，然後再經過好幾層的ConvTranspose2d/Batchnorm2d/ReLU，最後再經過Tanh讓output範圍在1~-1，然後輸出1個batch的照片。

```
class NETG(nn.Module):
    def __init__(self,nz,nclass,nc):
        super(NETG, self).__init__()
        self.net = nn.Sequential(
            nn.ConvTranspose2d(nz + nclass, 512, 4, 1, 0, bias=False),
            nn.BatchNorm2d(512),
            nn.ReLU(True),
            nn.ConvTranspose2d(512, 256, 4, 2, 1, bias=False),
            nn.BatchNorm2d(256),
            nn.ReLU(True),
            nn.ConvTranspose2d(256, 128, 4, 2, 1, bias=False),
            nn.BatchNorm2d(128),
            nn.ReLU(True),
            nn.ConvTranspose2d(128, 64, 4, 2, 1, bias=False),
            nn.BatchNorm2d(64),
            nn.ReLU(True),
            nn.ConvTranspose2d(64, nc, 4, 2, 1, bias=False),
            nn.Tanh(),
        )
    def forward(self, x, label,nclass):
        label = label.view(-1, nclass, 1, 1)
        x = torch.cat([x, label], 1)
        return self.net(x)
```

Discrimminator:

Discrimminator就是先將24的label轉成64*64的embedding，再和image concat，然後再經過好幾層的Conv2d/Batchnorm2d/LeakyReLU，得到true/false。

```python
class NETD(nn.Module):
    def __init__(self,nc,nclass):
        super(NETD, self).__init__()
        self.feature_input = nn.Linear(nclass, 64 * 64)
        self.net = nn.Sequential(
            nn.Conv2d(nc + 1, 64, 4, 2, 1, bias=False),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Conv2d(64, 128, 4, 2, 1, bias=False),
            nn.BatchNorm2d(128),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Conv2d(128, 256, 4, 2, 1, bias=False),
            nn.BatchNorm2d(256),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Conv2d(256, 512, 4, 2, 1, bias=False),
            nn.BatchNorm2d(512),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Conv2d(512, 1, 4, 1, 0, bias=False),
        )
    def forward(self, x, label):
        label = self.feature_input(label).view(-1, 1, 64, 64)
        x = torch.cat([x, label], 1)
        return self.net(x).view(-1, 1)
```

至於loss的計算，我是使用MSELoss，就是照著講義的公式。

D_loss的目標就是盡量讓真照片都預測正確，假照片都預測錯誤。

G_loss的目標則是盡量讓假照片都預測正確。

$$\boldsymbol{\theta}^{(G)*} = \arg\min_{\boldsymbol{\theta}^{(G)}} \max_{\boldsymbol{\theta}^{(D)}} V(\boldsymbol{\theta}^{(D)}, \boldsymbol{\theta}^{(G)})$$

```python
netD_loss = Loss(prediction_real, tensor['label_real']) + Loss(prediction_fake, tensor['label_fake'])
netD_loss.backward()
optimizerD.step()

optimizerG.zero_grad()
prediction_fake = netD(fake_data, label)
netG_loss = Loss(prediction_fake, tensor['label_real'])
```

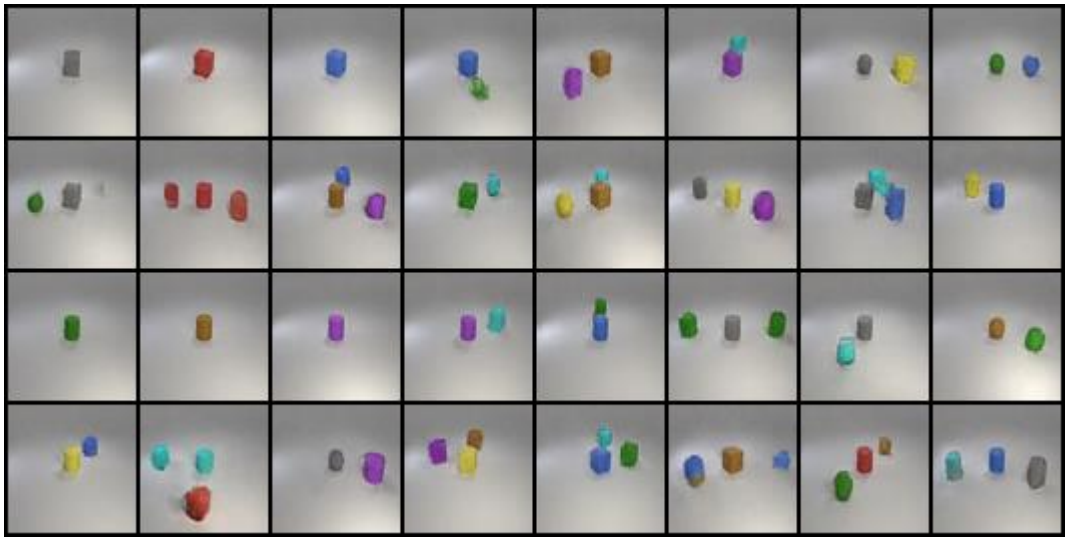
- Specify the hyperparameters (learning rate, epochs, etc.)

```
hyperparameter['lrG'] = 5e-4
hyperparameter['lrD'] = 3e-5
hyperparameter['batch_size'] = 32
hyperparameter['nz'] = 25
hyperparameter['nclass'] = 24
hyperparameter['nc'] = 3
hyperparameter['beta'] = (0.5, 0.999)
hyperparameter['epochs'] = 1000
```

- Results and discussion
  - Show your results based on the testing data.

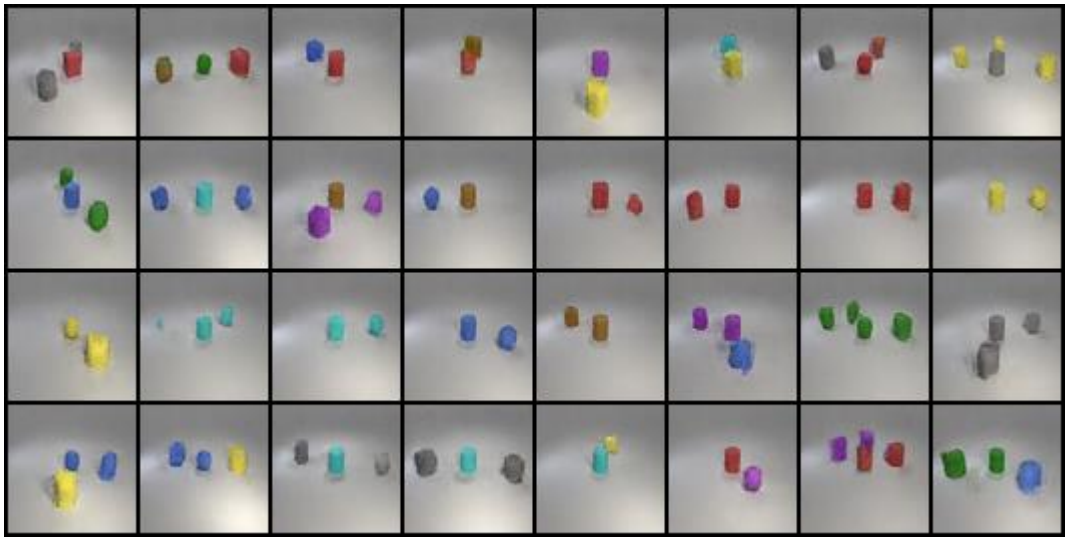
test.json:

```
Result:
Best test.json score :  0.8333333333333334
```



new_test.json:

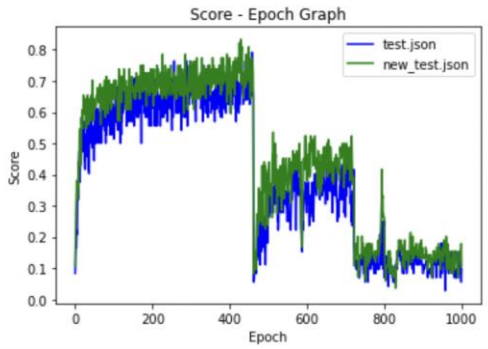
```
New test score: 0.8452380952380952
```



- Discuss the results of different models architectures.

1.首先我有強制拉長epoch，看能不能硬練上去，結果發現當lr較大，epoch過長時，generator很容易練壞。解決方法就是把lr弄小，或是把epoch弄小。
像底下這組就是練壞的例子。
hyperparameter如下：

```
hyperparameter['lrG'] = 2e-3
hyperparameter['lrD'] = 1e-4
hyperparameter['batch_size'] = 32
hyperparameter['nz'] = 25
hyperparameter['nclass'] = 24
hyperparameter['nc'] = 3
hyperparameter['beta'] = (0.5, 0.999)
hyperparameter['epochs'] = 1000
```



2.我也有直接調lr，發現lrG=5e-4，lrD=3e-5時效果最好，也就是前面的result。
這裡順便放其他組做參考：
(1)
hyperparameter['lrG'] = 2e-3
hyperparameter['lrD'] = 1e-4
Best test.json score： 0.7083333333333334
Best new_test.json score： 0.8095238095238095
(2)

hyperparameter['lrG'] = 2e-4
hyperparameter['lrD'] = 1e-5
Best test.json score : 0.7083333333333334
Best new_test.json score : 0.8095238095238095
(3)
hyperparameter['lrG'] = 1e-3
hyperparameter['lrD'] = 5e-5
Best test.json score : 0.75
Best new_test.json score : 0.8690476190476191
這組特別的是 他的new_test.json有練得比我最前面展示的result好，但是test.json卻沒出0.8。

3.另外我也有把MSELoss換成Sigmoid+BCELoss，從圖中看出，更換loss的影響不大。

Comparison: