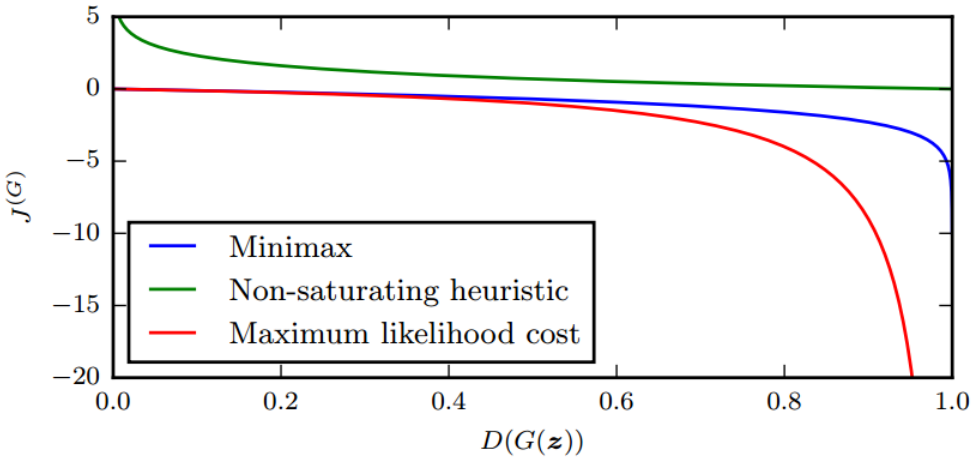
Report (50%)

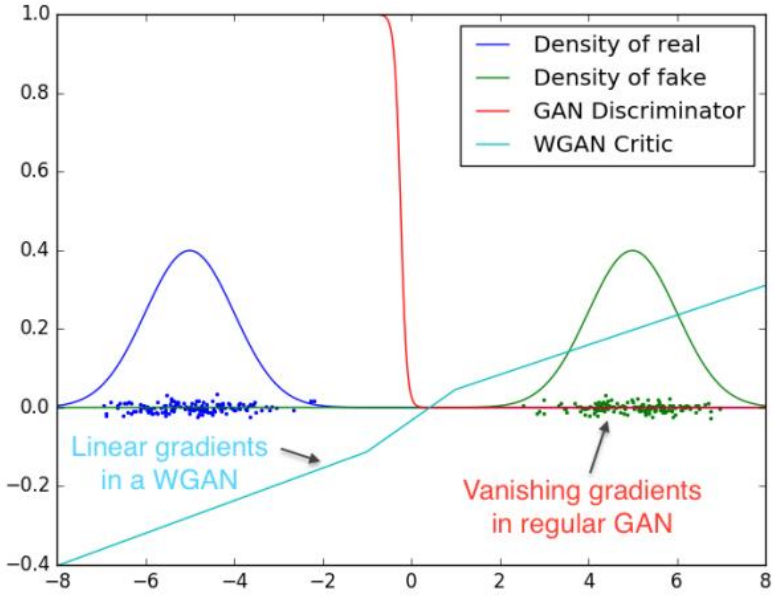
Introduction (5%)

此次作業主要是要用訓練出一個 condition GAN ，使 GAN 能根據 condition 來生成對應的圖片，condition 一共有 24 個 labels，也就是說一共有 24 種要生成的圖形，而當中會有多個 labels 同時輸入，也即是同時生成多個圖形的情況，最後會用一個 learned evaluation network 來判斷生成圖形的分數。

這次我所使用的是 WGAN，因為一般 GAN 的很容易出現 gradient vanishing 和 exploding 很難訓練的情況(如下圖)。



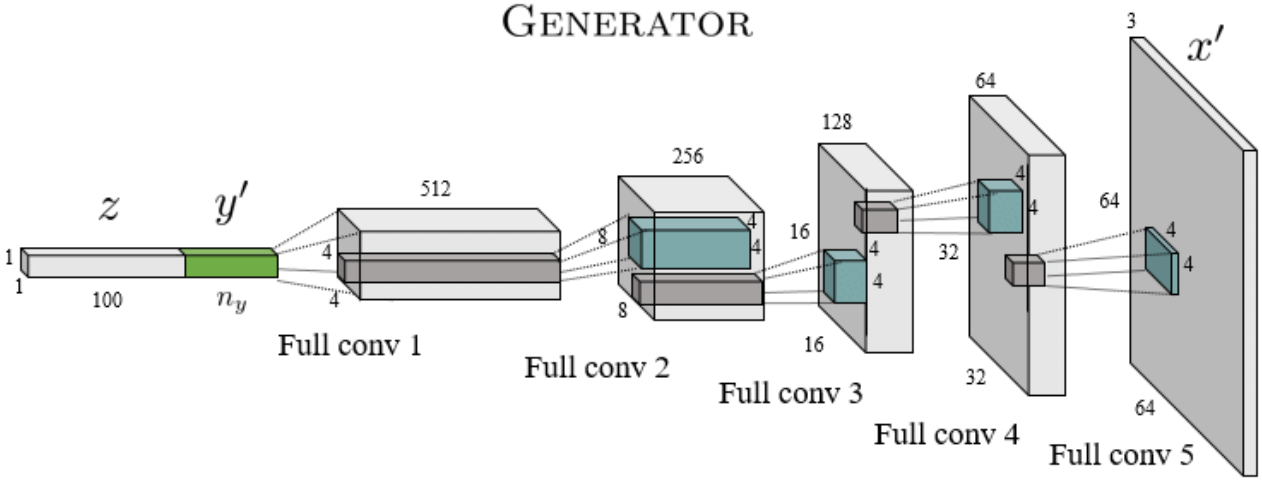
WGAN 則可以解決這個問題，相對容易訓練，我們可以看到(下圖)在 GAN gradient vanishing 時，WGAN 的 gradient 仍然不為零，可以繼續用 gradient descent 的方法來訓練。

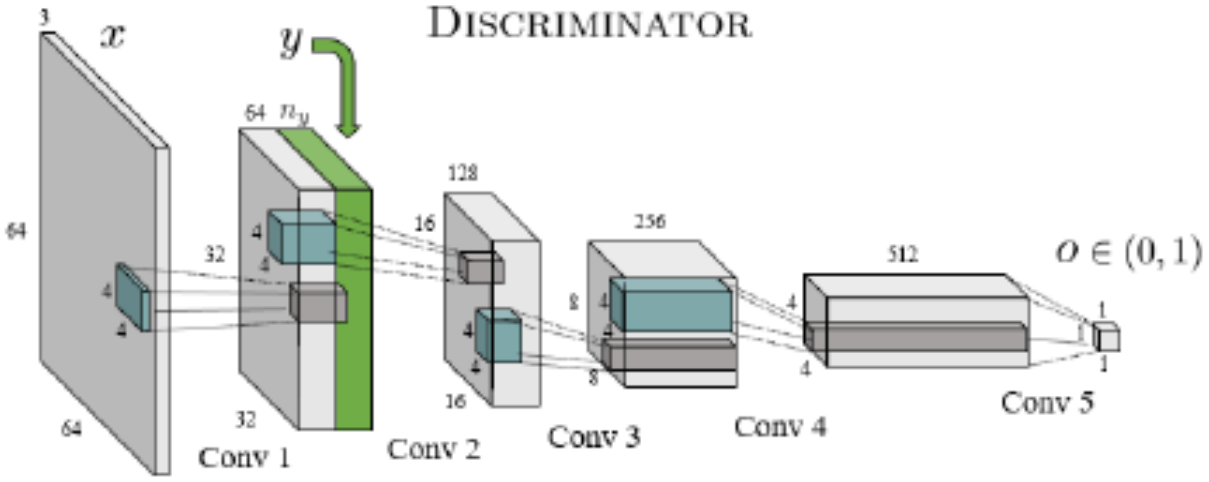


Implementation details (15%)

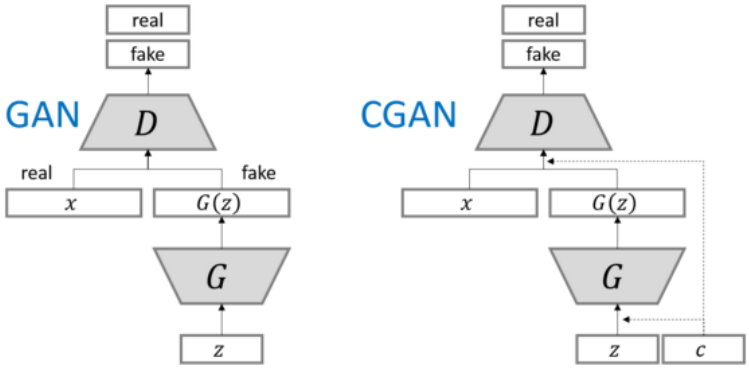
Describe how you implement your model, including your choice of cGAN, model architectures, and loss functions. (10%)

要生成圖形，首先要在 normal distribution sample 一個 vector 加上 condition 來作為輸入，讓 GAN 可以根據輸入來生成圖形。而由於 WGAN 在結構上和 DCGAN 十分相似，所以我以 DCGAN 作為參考再加上 WGAN 所需的改動作為此次作業的 WGAN，Generator 和 Discriminator 的結構如下。





由於此次作業是 cGAN，下圖為 GAN 和 cGAN 的區別，在結構上加多了一層 linear layer 作為 condition 輸入，讓 condition label 加入到 normal distribution vector 作為 WGAN 的輸入(如下圖)。



以下是實際使用時 WGAN 的 Generator 結構

```
WGAN_Generator(
  (fc): Sequential(
    (0): Linear(in_features=24, out_features=156, bias=True)
    (1): ReLU(inplace=True)
  )
  (layer1): Sequential(
    (0): ConvTranspose2d(256, 512, kernel_size=(4, 4), stride=(1, 1), bias=False)
    (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): ReLU(inplace=True)
  )
  (layer2): Sequential(
    (0): ConvTranspose2d(512, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1),
bias=False)
    (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): ReLU(inplace=True)
  )
  (layer3): Sequential(
    (0): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1),
bias=False)
    (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): ReLU(inplace=True)
  )
  (layer4): Sequential(
    (0): ConvTranspose2d(128, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1),
bias=False)
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): ReLU(inplace=True)
  )
  (layer5): Sequential(
    (0): ConvTranspose2d(64, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1),
bias=False)
    (1): BatchNorm2d(3, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): Tanh()
  )
)
```

Discriminator 和 DCGAN 用的 Discriminator 基本一樣，也同樣加了 linear layer 方便我們結合 condition label，而和其他 Discriminator 不一樣的是 Sigmoid 在使用 WGAN 上要移除，因為 WGAN 的 Discriminator 已經不是單單做真假二分類任務，而是做 Wasserstein 距離的擬合，屬於回歸任務，所以就不需要使用 sigmoid。

```
WGAN_Discriminator(
  (fc): Sequential(
    (0): Linear(in_features=24, out_features=4096, bias=True)
    (1): LeakyReLU(negative_slope=0.2, inplace=True)
  )
  (layer1): Sequential(
    (0): Conv2d(4, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): LeakyReLU(negative_slope=0.2, inplace=True)
  )
  (layer2): Sequential(
    (0): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1),
bias=False)
    (1): LeakyReLU(negative_slope=0.2, inplace=True)
  )
  (layer3): Sequential(
    (0): Conv2d(128, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1),
bias=False)
    (1): LeakyReLU(negative_slope=0.2, inplace=True)
  )
  (layer4): Sequential(
    (0): Conv2d(256, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1),
bias=False)
    (1): LeakyReLU(negative_slope=0.2)
  )
  (layer5): Sequential(
    (0): Conv2d(512, 1, kernel_size=(4, 4), stride=(1, 1), bias=False)
  )
)
```

相對傳統 GAN 使用 Adam，WGAN 使用 RMS 作為 optimizer，而不使用有關動量的優化，因為 Discriminator 使用 Adam 遇到 loss gradient 不穩定的情況下更新會和真實 gradient 的方向不一樣，這應該是因為動量的優化器的原因，而這亦不利於訓練 GAN。

optim 下面是是一些訓練參數的設置，然後根據 WGAN 的 loss 的推導結論，可以得出真實的 Discriminator 輸出 label 減去 Discriminator 對 Generator 生成的圖片輸出的 label 即為 Discriminator 的 loss，而 Generator 的 loss 則是由 Discriminator 判別出來的 label 和真實的 label 相減。

$$Loss_{Generator} = -E_{x\sim P_g}[f_w(x)],\ Loss_{Discriminator} = E_{x\sim P_g}[f_w(x)] - E_{x\sim P_r}[f_w(x)]$$

```
opt_g = torch.optim.RMSprop(model_g.parameters(), lr=lrg)  # WGAN2: use RMS optim
opt_d = torch.optim.RMSprop(model_d.parameters(), lr=lrd)
g_lr_step = torch.optim.lr_scheduler.StepLR(opt_g, step_size=gamma_step, gamma=gamma)
d_lr_step = torch.optim.lr_scheduler.StepLR(opt_d, step_size=gamma_step, gamma=gamma)
```

```
best_score, best_new_score = 0, 0
epoch_score, epoch_new_score, loss_real_total, loss_fake_total, loss_g_total,
acc_real_total, \
acc_fake_total, acc_g_total, img_total, img_new_total = [], [], [], [], [], [], [],
[], [], []
total_epochs = total_epochs - start_epochs
progress = tqdm(total=total_epochs)

for epoch in range(total_epochs):
    score, score_new = 0, 0
    loss_real_batch, loss_fake_batch, loss_g_batch, acc_real_batch, acc_fake_batch,
acc_g_batch, img_batch = \
        [], [], [], [], [], [], []
    for data in data_loader_train:
        img = data[0].to(device)
        cond = data[1].to(device, dtype=torch.float)
        size = img.size()[0]
        one= torch.ones(size).to(device)
        mone = -1 * real_label  # WGAN3: use -1 instead of 0
        z = torch.randn(size, 100, 1, 1).to(device)
        loss_real_sum, acc_real_sum, loss_fake_sum, acc_fake_sum, loss_g_sum,
acc_g_sum = 0, 0, 0, 0, 0, 0
        model_d.train()
        for iter in range(iter_d):
            model_d.zero_grad()
            pred_label = model_d(img, cond).view(-1)
            pred_label.backward(one)   # WGAN3: remove loss function and directly
backward pred_label with coef one and mone
            acc_real = pred_label.mean().item()
            # loss_real_sum += loss_real.item()
            acc_real_sum += acc_real
            pred_img = model_g(z, cond).detach()  # fix Gen, prevent bp
            pred_label_fake = model_d(pred_img, cond).view(-1)
            pred_label_fake.backward(mone)  # WGAN3: remove loss
            acc_fake = pred_label_fake.mean().item()  # edit
            acc_fake_sum += acc_fake
            opt_d.step()
        loss_size = size * iter_d
        loss_real_batch.append(loss_real_sum / loss_size)
        acc_real_batch.append(acc_real_sum / loss_size)
        loss_fake_batch.append(loss_fake_sum / loss_size)
        acc_fake_batch.append(acc_fake_sum / loss_size)
        for para in model_d.parameters():  # WGAN3: limit model_d
            para.data.clamp_(-args.clp, args.clp)

        model_g.train()
        for iter in range(iter_g):
            model_g.zero_grad()
            pred_img = model_g(z, cond)
            pred_label = model_d(pred_img, cond).view(-1)
            pred_label.backward(one)  # WGAN3
            opt_g.step()
            acc_g = pred_label.mean().item()
            acc_g_sum += acc_g
```

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

如上的 coding 和 DCGAN 一樣，我也引入了 iteration 的次數作為 hyperparameter，除此之外，還有 clip(在 WGAN 限制 Lipschitz 的常數上下限)、Generator 和 Discriminator 的 learning rate、learning rate step、batch size、epoch 等。

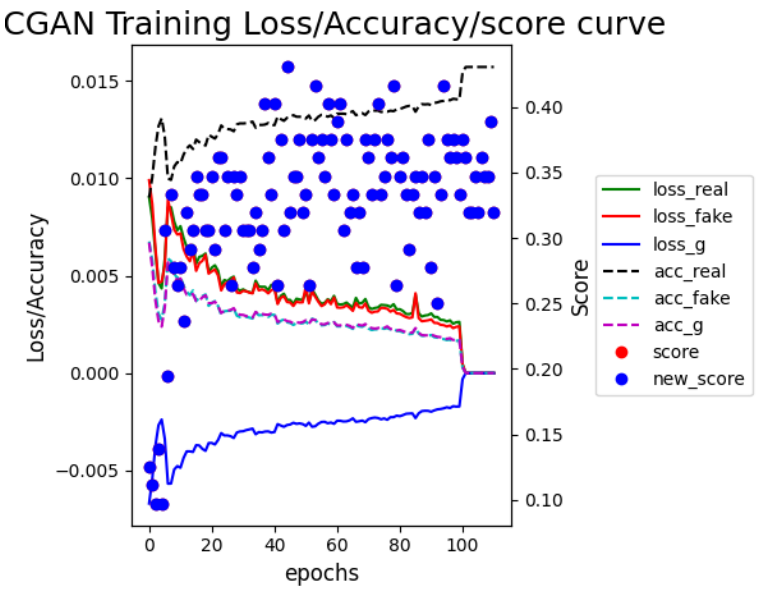
Results and discussion (30%)

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

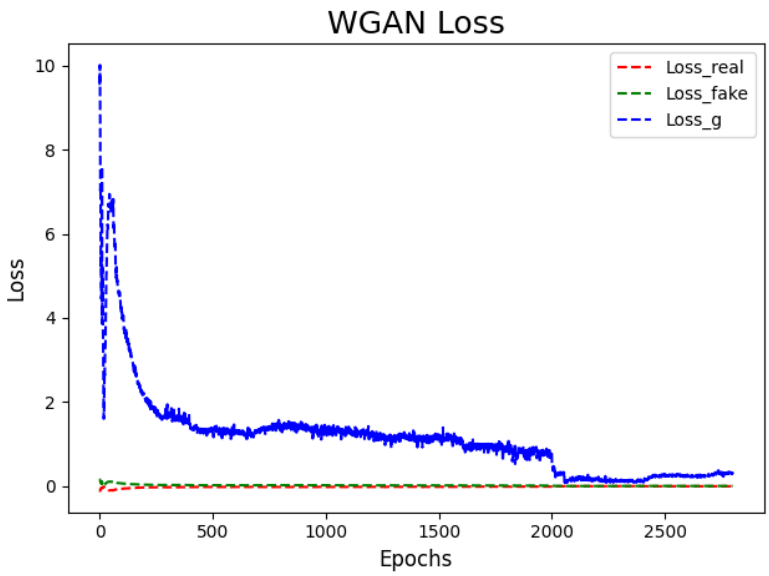
Best scores of test: 0.61111, Best scores of new test: 0.72619

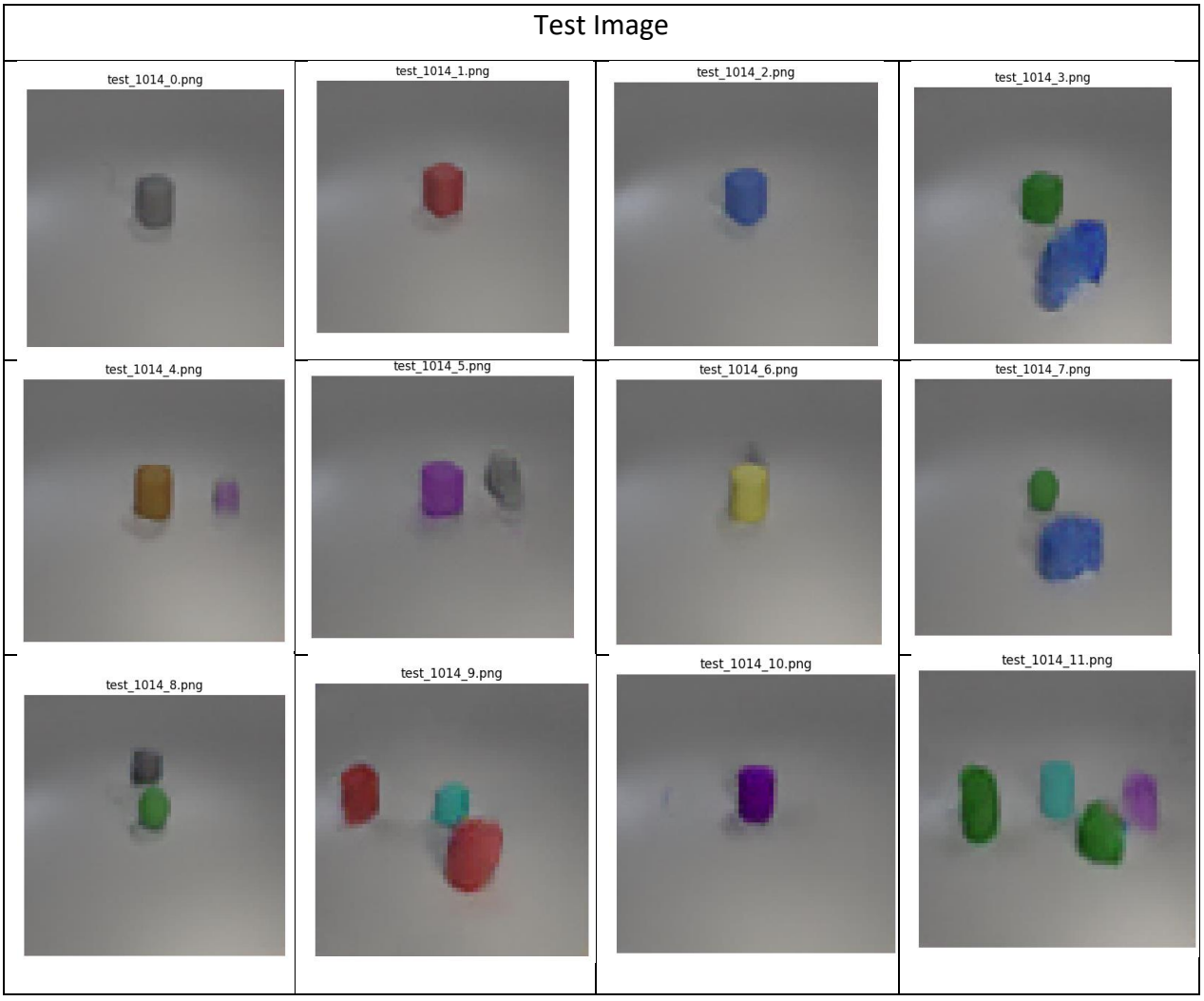
要用 GAN 來訓練一個生成圖形的模型其實也不太容易，首先要選好要用的 cGAN，再根據所選的 GAN 的特性來調整各個參數。舉個例來說，Discriminator 和 Generator 各自的訓練次數是一個很重要的參數，Discriminator 學太好，Generator 就很難學到東西，相反也是一樣。各自的 learning rate 也是一個相對比較重要的參數，相差不能太大，否則會導致其中一個學不到東西。

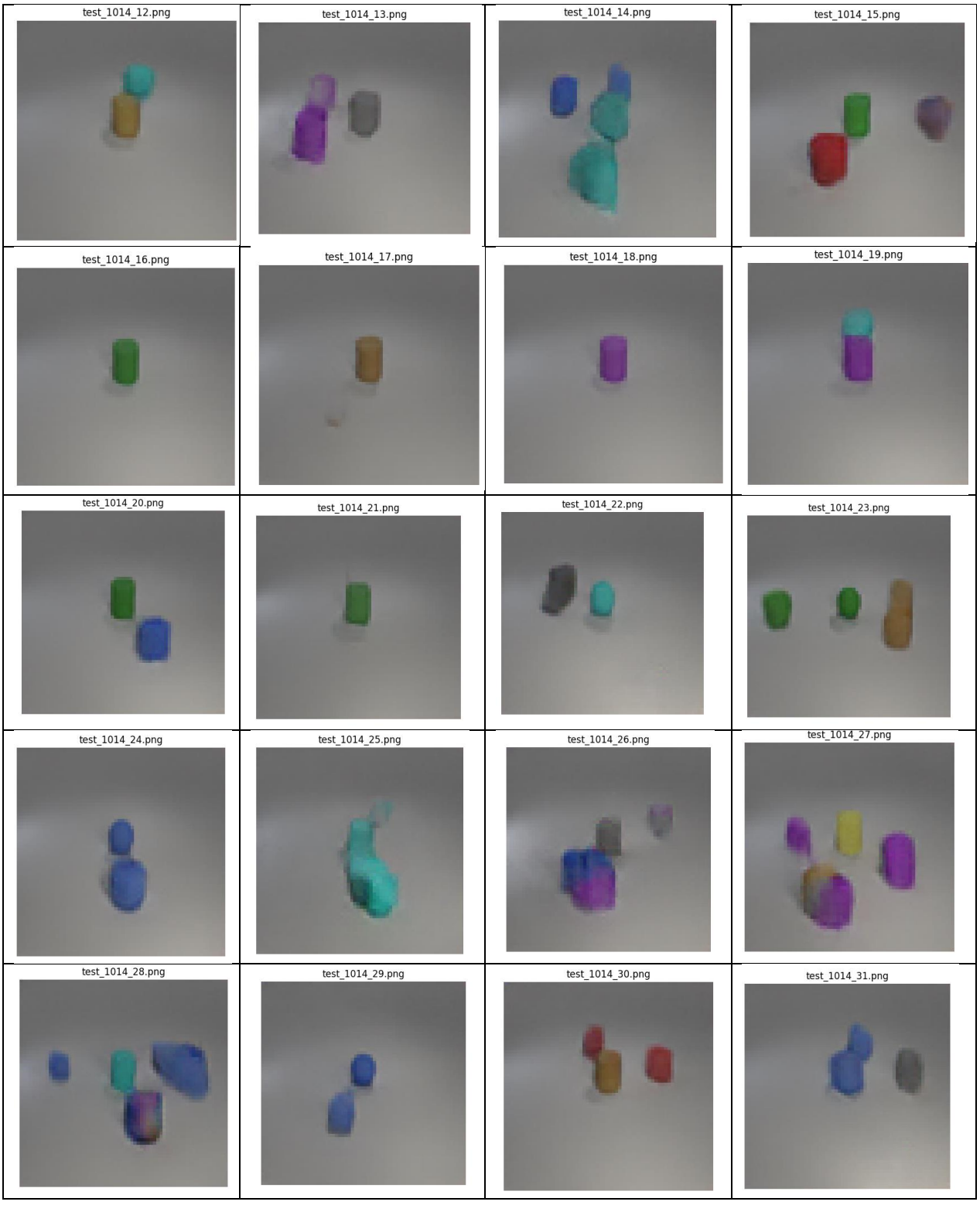
另外我也有嘗試過一般的 DCGAN，但因為很容易出現 gradient vanishing 導致調參更加困難，score 也一直上不去(如下圖)，所以最後選擇了更進一步使用 WGAN。



可以看到調整過後的 WGAN 很平穩的提升，score 也越來越高，偶而會生成出很好的圖形。


WGAN Loss
Loss_real
Loss_fake
Loss_g
Loss
Epochs
0
2
4
6
8
10
0
500
1000
1500
2000
2500


Test Image
test_1014_0.png
test_1014_1.png
test_1014_2.png
test_1014_3.png
test_1014_4.png
test_1014_5.png
test_1014_6.png
test_1014_7.png
test_1014_8.png
test_1014_9.png
test_1014_10.png
test_1014_11.png


test_1014_12.png
test_1014_13.png
test_1014_14.png
test_1014_15.png
test_1014_16.png
test_1014_17.png
test_1014_18.png
test_1014_19.png
test_1014_20.png
test_1014_21.png
test_1014_22.png
test_1014_23.png
test_1014_24.png
test_1014_25.png
test_1014_26.png
test_1014_27.png
test_1014_28.png
test_1014_29.png
test_1014_30.png
test_1014_31.png

New_test image

new_test_2612_0.png

new_test_2612_1.png

new_test_2612_2.png

new_test_2612_3.png

new_test_2612_4.png

new_test_2612_5.png

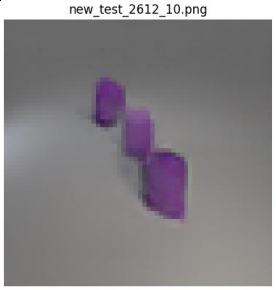
new_test_2612_6.png

new_test_2612_7.png

new_test_2612_8.png

new_test_2612_9.png

new_test_2612_10.png



new_test_2612_11.png

new_test_2612_12.png

new_test_2612_13.png

new_test_2612_14.png

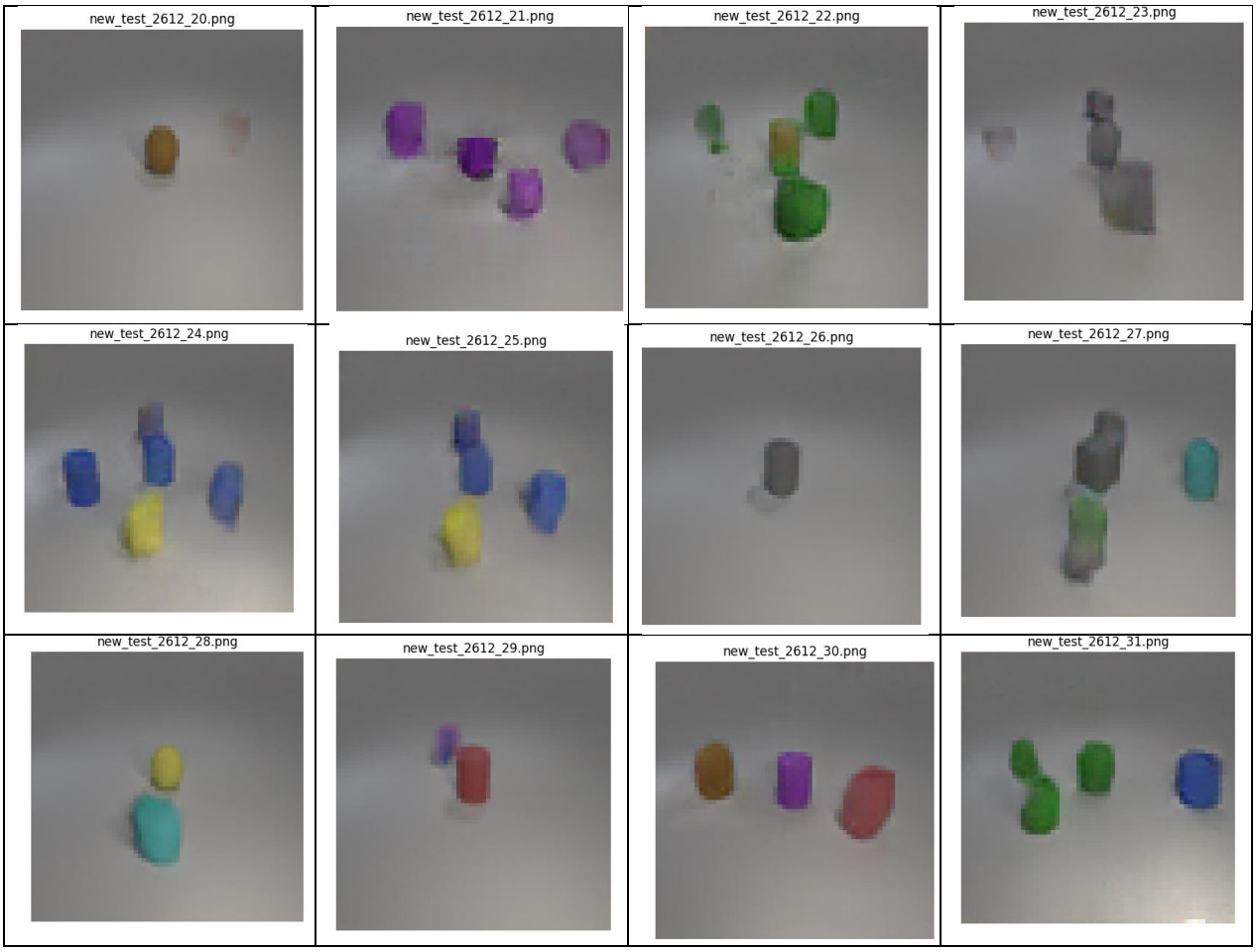
new_test_2612_15.png

new_test_2612_16.png

new_test_2612_17.png

new_test_2612_18.png

new_test_2612_19.png

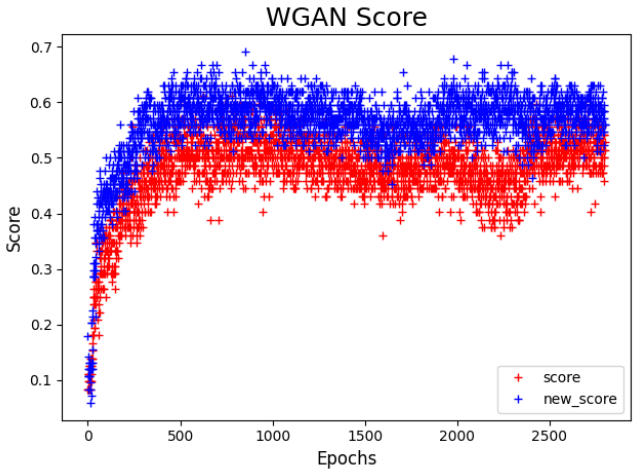


Discuss the results of different models architectures. (25%)

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.

在實作上，我加入了 linear layer 來作為 condition 的輸入，另外也嘗試了 learning rate scheduler 來改變 learning rate。

我覺得 learning rate scheduler 蠻有用的我是設計成 1.1 讓網絡不會卡在 local minimal 有機會借著 step 跳過 local minimal，比較過後沒有 step 的 GAN 會比較容易收斂，但不再會有提升，相對加入了 scheduler 會比較浮動，偶而會出現更好的 score。



Experimental results (50%) (based on results shown in your report)

Classification accuracy on test.json. (25%)

```
best_epoch = 2403 | best score = 0.65278
```

可以觀察出來，雖然 WGAN 可以生成圖形了，但是外觀上還是未如理想，所以在分類時得到的分數上不去，關於這點我也試著調參讓他更好的學習，但結果還是不太好。在 test 裡最高得分有 0.65278。

Classification accuracy on new_test.json. (25%)

```
best_new_epoch = 2184 | best new score = 0.76190
```

在 new_test 的表現相對比 test 好，可能是我的 GAN 比較傾向生成多個圖形，在生成單個圖形時精度不夠而導致這樣的結果，同樣最高的 score 是 0.76190。

最後附上最高分數的 scroe 分佈，但 marker 沒選好，看得不夠清楚。