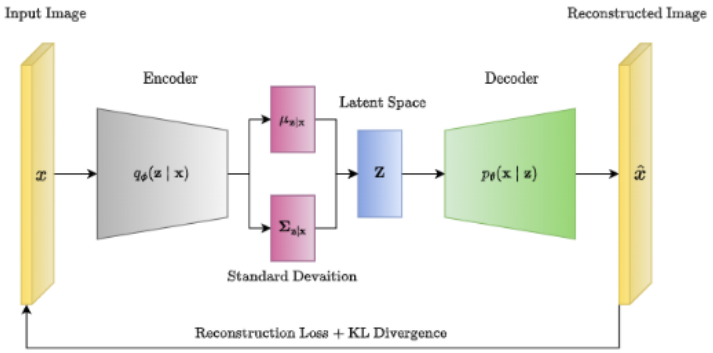
VAE MNIST



Load data 並測試 input image 跟 label 是否異常

```
def plot_images(images, labels, num_images=5):
    _, axes = plt.subplots(1, num_images, figsize=(12, 3))
    for i in range(num_images):
        axes[i].imshow(images[i], cmap='gray')
        axes[i].set_title(labels[i])
        axes[i].axis('off')
    plt.show()

plot_images(xtrain, ytrain)
```

把資料二值化來降低圖像複雜度，一開始訓練的時候沒有這部分會導致產生的數字糾結再一起。

```
xtrain = np.where(xtrain > 128, 1, 0)
x_val = np.where(x_val > 128, 1, 0)
xtrain = xtrain.astype(np.float32)
x_val = x_val.astype(np.float32)
```

Reconstruction error + KL divergence 作為 loss 計算方式。

```
def loss_function(y, x, mu, std):
    ERR = F.binary_cross_entropy(y, x.view(-1, 784), reduction='sum')
    KLD = -0.5 * torch.sum(1 + torch.log(std**2) - mu**2 - std**2)
    return ERR + KLD
```

VAE 網路架構:
VAE 透過 encoder 全連階層得到兩個 return value 用在 sampling 的地方。而 sampling 是在做 reparameterization 來保證 backward 進行。而這個 return 在這邊稱為 z，透過 decoder 可以重建我們要的數據。

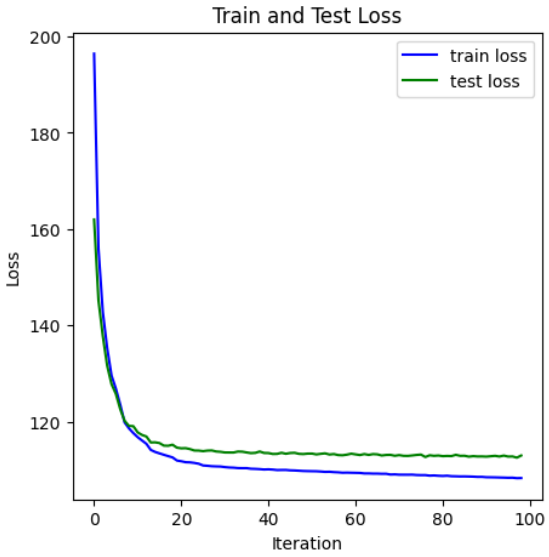
```python
class VAE(nn.Module):
    def __init__(self):
        super(VAE, self).__init__()

        self.fc1 = nn.Linear(784, 128)  # Encoder
        self.fc2 = nn.Linear(128, 16)   # Encoder
        self.fc21 = nn.Linear(16, 8)    # mu
        self.fc22 = nn.Linear(16, 8)    # sigma

        self.fc3 = nn.Linear(8, 16)    # Decoder
        self.fc4 = nn.Linear(16, 128)  # Decoder
        self.fc5 = nn.Linear(128, 784)

    def encoder(self, x):
        h = nn.ReLU()(self.fc2(nn.ReLU()(self.fc1(x))))
        return self.fc21(h), self.fc22(h)  # mu, std

    def sampling(self, mu, std):  # Reparameterization trick
        eps1 = torch.randn_like(std)
        eps2 = torch.randn_like(std)
        return 0.5 * ((eps1 * std + mu) + (eps2 * std + mu))

    def decoder(self, z):
        h = nn.ReLU()(self.fc4(nn.ReLU()(self.fc3(z))))
        return torch.sigmoid(self.fc5(h))

    def forward(self, x):
        mu, std = self.encoder(x.view(-1, 784))
        z = self.sampling(mu, std)
        return self.decoder(z), mu, std
```

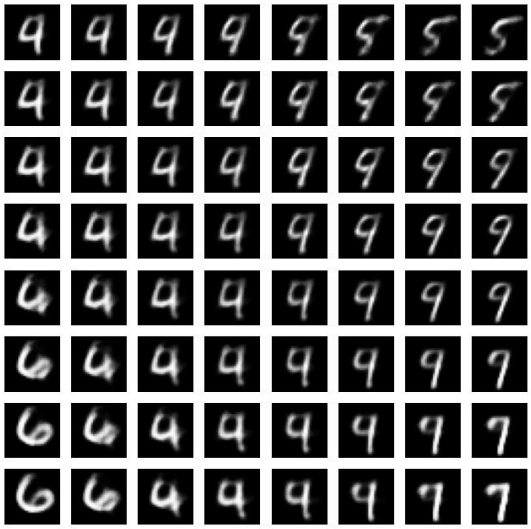
最後在圖片生成方面用 generate_num 隨機取得 4 個數字來得到 train 裡面的 image，然後使用 encoder 得到兩個 return value 在使用 sampling 取得不同的 z，我在後面稱為 generate。接者，透過迴圈讓他在每個位置有不同的縱軸以及橫軸的變化量。

```python
def generate_num():
    num = random.randint(1,1000)
    print(testset[num][1])
    input_pic = testset[num][0].to(device).view(-1)


    mu, std = model.encoder(input_pic)
    z = model.sampling(mu, std)
    return z
```

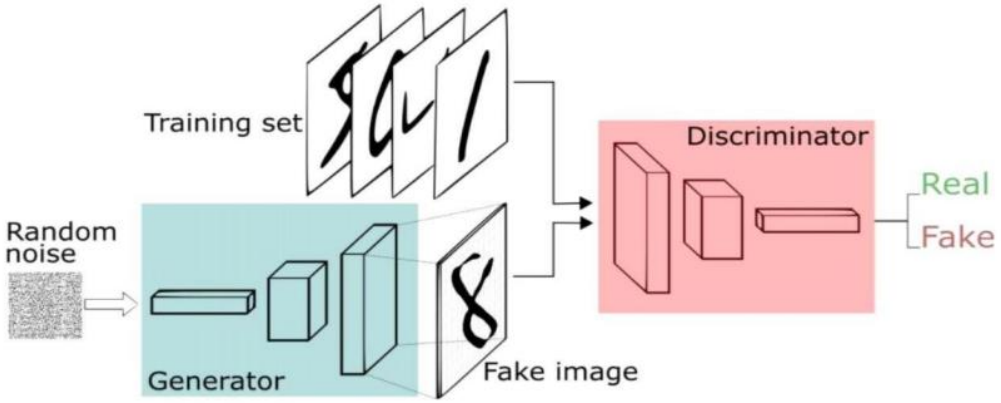
```python
fig, axs = plt.subplots(8, 8, figsize=(8, 8))
for i in range(8):
    for j in range(8):
        # Calculate the interpolation between generate1 and generate2
        gap1 = (generate2 - generate1) / 7
        next1 = generate1 + gap1 * i

        # Calculate the interpolation between generate3 and generate4
        gap2 = (generate4 - generate3) / 7
        next2 = generate3 + gap2 * i

        # Calculate the final interpolation between next1 and next2
        final_gap = (next2 - next1) / 7
        final_result = next1 + final_gap * j

        # Generate image using the decoder
        produce = model.decoder(final_result)
        result = produce.reshape((28, 28)).detach().cpu()

        # Plot the image
        axs[i, j].imshow(result, cmap='gray')
        axs[i, j].axis('off')

plt.show()
```

Train 100 epochs 的 loss 圖以及成果(4,6,5,7 是 generate_num 產生的數字)





GAN CIFAR10



GAN 架構以及保存圖像。Generator 透過輸入 noise 和一個 condition label 經過網路生成圖像，最後一層 embedding 代表對應的 10 個 label。在 discriminator 的部分經過 sigmoid 得到一個 0 到 1 的值，然後透過 softmax 得到 11 個類別的機率，其中一個類別是 fake。

```python
class Generator(nn.Module):

    def __init__(self):
        super(Generator,self).__init__()

        #input 100*1*1
        self.layer1 = nn.Sequential(nn.ConvTranspose2d(100,512,4,1,0,bias = False),nn.ReLU(True)) #input output kernel stride padding
        #input 512*4*4
        self.layer2 = nn.Sequential(nn.ConvTranspose2d(512,256,4,2,1,bias = False),nn.BatchNorm2d(256),nn.ReLU(True))
        #input 256*8*8
        self.layer3 = nn.Sequential(nn.ConvTranspose2d(256,128,4,2,1,bias = False),nn.BatchNorm2d(128),nn.ReLU(True))
        #input 128*16*16
        self.layer4 = nn.Sequential(nn.ConvTranspose2d(128,64,4,2,1,bias = False),nn.BatchNorm2d(64),nn.ReLU(True))
        #input 64*32*32
        self.layer5 = nn.Sequential(nn.ConvTranspose2d(64,3,4,2,1,bias = False),nn.Tanh())
        #output 3*64*64

        self.embedding = nn.Embedding(10,100)  #10 classes


    def forward(self,noise,label):

        label_embedding = self.embedding(label)
        x = torch.mul(noise,label_embedding)
        x = x.view(-1,100,1,1)

        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        x = self.layer4(x)
        x = self.layer5(x)
        return x
```

```python
class Discriminator(nn.Module):

    def __init__(self):
        super(Discriminator,self).__init__()

        #input 3*64*64
        self.layer1 = nn.Sequential(nn.Conv2d(3,64,4,2,1,bias = False),nn.BatchNorm2d(64),nn.LeakyReLU(0.2,True),nn.Dropout2d(0.5))
        #input 64*32*32
        self.layer2 = nn.Sequential(nn.Conv2d(64,128,4,2,1,bias = False),nn.BatchNorm2d(128),nn.LeakyReLU(0.2,True),nn.Dropout2d(0.5))
        #input 128*16*16
        self.layer3 = nn.Sequential(nn.Conv2d(128,256,4,2,1,bias = False),nn.BatchNorm2d(256),nn.LeakyReLU(0.2,True),nn.Dropout2d(0.5))
        #input 256*8*8
        self.layer4 = nn.Sequential(nn.Conv2d(256,512,4,2,1,bias = False),nn.BatchNorm2d(512),nn.LeakyReLU(0.2,True))
        #input 512*4*4
        self.validity_layer = nn.Sequential(nn.Conv2d(512,1,4,1,0,bias = False),nn.Sigmoid())

        self.label_layer = nn.Sequential(nn.Conv2d(512,11,4,1,0,bias = False),nn.LogSoftmax(dim = 1))  #one is 'fake' so 11 classes

    def forward(self,x):

        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        x = self.layer4(x)
        validity = self.validity_layer(x)
        plabel = self.label_layer(x)

        validity = validity.view(-1)
        plabel = plabel.view(-1,11)

        return validity,plabel
```

```python
def save_img(epoch):
    noise = torch.randn(64, 100, device=device)
    sample_labels = torch.randint(0, 10, (64,), device=device, dtype=torch.long)

    # 使用generator生成假圖
    fake_images = gen(noise, sample_labels)
    fake_images_cpu = fake_images.cpu()

    showImage(make_grid(fake_images_cpu),epoch)
```

下面 code 的部分是為了平滑化模型的訓練，real 為 1，fake 為 0，所以藉由調整不同 label 代表的數值來避免結果不良

```
real_labels = 0.7 + 0.5 * torch.rand(10, device = device)
fake_labels = 0.3 * torch.rand(10, device = device)
```

```
if idx % 25 == 0:
    real_label, fake_label = fake_label, real_label
```

Train loss for 500 epochs 跟結果