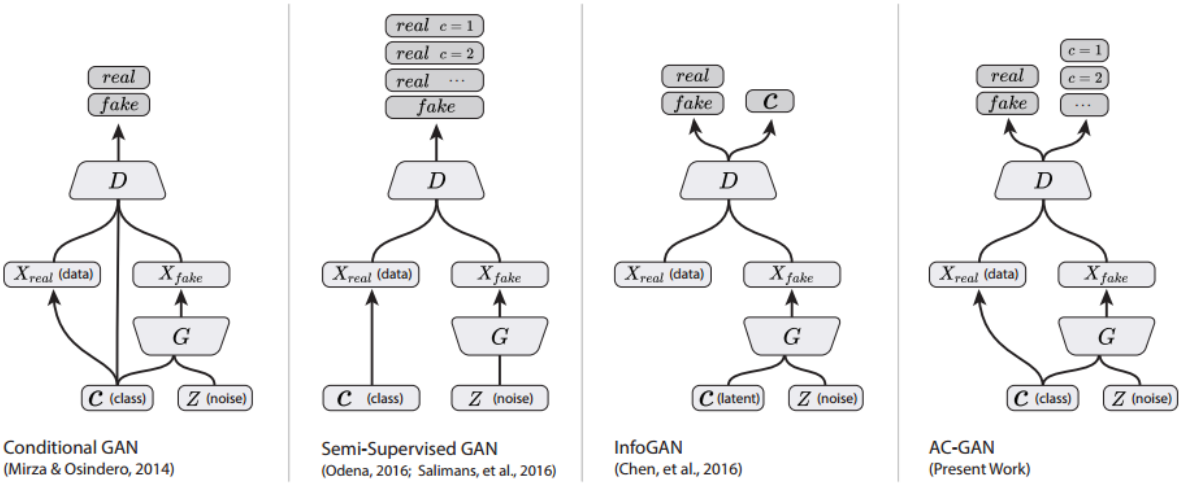
HW7 report

Introduction

ACGAN

ACGAN的原理與GAN（CGAN）相似。對於CGAN和ACGAN，Generater輸入均為latent vector及其label，輸出是屬於輸入label的仿造圖片。對於CGAN，Discriminator的輸入是圖片（包含真的和仿造的）和他的label，輸出是圖片屬於真實圖片的機率。對於ACGAN，判別器的輸入是一幅圖片，而輸出是該圖片屬於真實圖片的機率和其分類機率。



本質上，在CGAN中，向model提供了label。在ACGAN中，使用輔助decoder model重建輔助訊息。ACGAN理論上認為，強制model執行其他任務可以提高原始任務的性能。在這樣情況下，輔助任務是圖片分類，原始任務是生成仿造圖片。

Discriminator的目標函數：

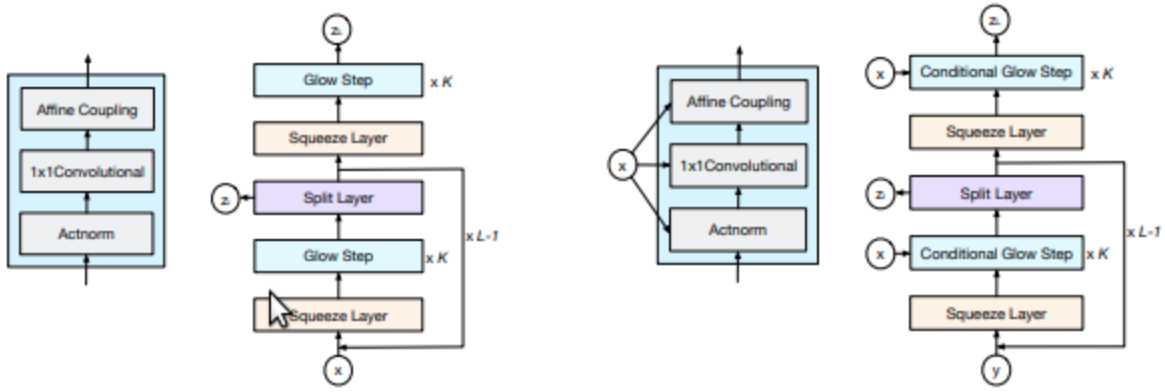
$$\mathcal{L}^{(D)} = -\mathbb{E}_{x\sim p_{data}} logD(x) - \mathbb{E}_z log[1-D(G(z|y))] - \mathbb{E}_{x\sim p_{data}} p(c|x) - \mathbb{E}_z logp(c|g(z|y))$$

Generater的目標函數：

$$\mathcal{L}^{(G)} = -\mathbb{E}_z log D(g(z|y)) - \mathbb{E}_z log p(c|g(z|y))$$

c-GLOW

傳統的結構化預測模型嘗試學習conditional likelihood，即 p(y|x)，以捕捉結構化輸出 y 和輸入特徵 x 之間的關係。 對於許多模型，計算似然性是棘手的。 因此，這些模型很難訓練，需要使用替代目標或變分推理來近似似然。 c-Glow是一種用於結構化輸出學習的 conditional Glow。 c-Glow 受益於基於flow的模型準確有效地計算 p(y|x) 的能力。 使用 c-Glow 學習不需要替代目標或在訓練期間進行推理。 一旦經過訓練，就可以直接有效地生成條件樣本。



(a) Glow architecture　　(b) c-Glow architecture

Implementation details

ACGAN

clvrai/ACGAN-PyTorch

As part of the implementation series of Joseph Lim's group at USC, our motivation is to accelerate (or sometimes delay) research in the AI community by promoting open-source

https://github.com/clvrai/ACGAN-PyTorch

clvrai/**ACGAN-PyTorch**

3 Contributors　13 Issues　171 Stars　45 Forks

參考這個repo的架構，然後做一些修改來處理我們的dataset，努力把tensor size出錯的地方解決就可以了。

在訓練的過程中，Discriminator的輸入是，真實資料或Generater吃noise（疊上condition）產出的仿造資料。Generater的輸入是latent vector疊上condition。

訓練時會先用真實資料訓練Discriminator，再用輸出結果計算分類loss和判斷真假的loss，相加得到總loss並更新model。再用仿造資料訓練Discriminator並重複上述步驟。最後才是訓練Generater。

分類loss因為是多元分類問題故選用nn.BCELoss()，判斷真假的loss則選用nn.NLLLoss()。

optimizer都是Adam，lr=0.0002, betas=(0.5, 0.999)。

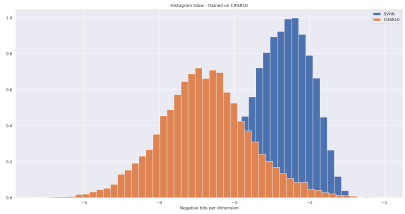
其他超參數設定，batchSize = 32，nz = 36，epochs = 1000。

c-Glow

y0ast/Glow-PyTorch

This repository implements the Glow model using PyTorch on the CIFAR-10 and SVHN dataset. We use the trained Glow to reproduce some of the results of the paper "Do Deep

https://github.com/y0ast/Glow-PyTorch



主要是參考這個repo。基本上修改num_classes和其他dataloader相關的地方就可以運作。然後再針對任務需求寫inference。

batch_size=16，epochs=10，K=6，L=3。其他超參數都是原始repo提供的預設值。

特別的是，這個repo的原作者，在flow_coupling的部份寫了additive和affine兩種版本可供選擇。他是說affine的在還原能力上比較好，additive的在條件生成的表現比較好。

我在Task 1試過affine和additive，Task 2用affine。

```
def normal_flow(self, input, logdet):
    assert input.size(1) % 2 == 0
    # 1. actnorm
    z, logdet = self.actnorm(input, logdet=logdet, reverse=False)
    # 2. permute
```

```
    z, logdet = self.flow_permutation(z, logdet, False)
    # 3. coupling
    z1, z2 = split_feature(z, "split")
    if self.flow_coupling == "additive":
        z2 = z2 + self.block(z1)
    elif self.flow_coupling == "affine":
        h = self.block(z1)
        shift, scale = split_feature(h, "cross")
        scale = torch.sigmoid(scale + 2.0)
        z2 = z2 + shift
        z2 = z2 * scale
        logdet = torch.sum(torch.log(scale), dim=[1, 2, 3]) + logdet
    z = torch.cat((z1, z2), dim=1)

    return z, logdet
```

Task2的第一題，是根據給予不同condition生成具有不同特徵的臉。讀進condition後呼叫reverse的model即可。

```
attribute_list = [20, 31, 26, 16] # Male, Smiling, Pale_Skin, Goatee
generate_x_list = torch.Tensor([]).cuda()
for yes in [1,0]:
  for i,attribute in enumerate(attribute_list):
    y = torch.zeros(40).unsqueeze(dim=1)
    y[attribute] = yes
    predict_x = model(y_onehot=y.cuda(), temperature=1, reverse=True)
    generate_x_list = torch.cat((generate_x_list,predict_x), 0)
save_image(generate_x_list, 'images/task2_Conditional_face.png',normalize=True)
```

Task 2的第二題，是從兩張臉中內插出多張臉。這題不需要用到condition，但是我的model都是用同一個，所以還是需要給予y的輸入。隨機選擇三對臉，將個照片x和對應的y餵給forward的model，得到個別的z。然後對這對z做線性內插，算出中間過程的多個z。最後將這些z餵給reverse的model，即生成漸進變化的各張臉。

內插演算法：

```
def interpolations(z1, z2, n):
    z_list = []
    for j in range(n):
        list_n = []
        for i in range(len(z1)):
            top = z1[i]
            down = z2[i]
```

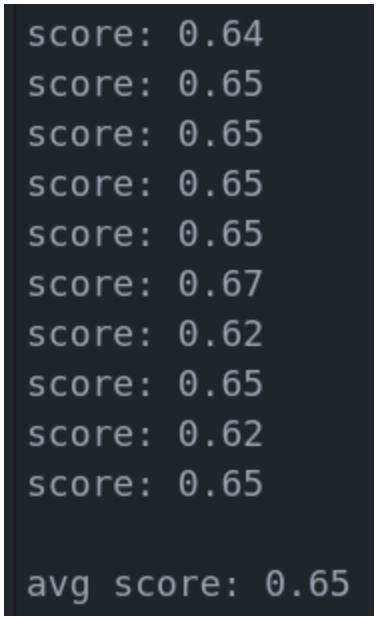
```
            value = down + 1.0 * j*(top-down)/n
            list_n.append(value)
        z_list.append(list_n)
    return np.array(z_list)
```

Task 2的第三題，要調整臉的某項特徵變化。我的作法是，選一張照片，將x,y餵給forward model得到z。檢查我選用的這張臉是否具備該項特徵。假設是"有"，接著開始讀進整個資料集，遇到"沒有"該特徵的就把x,y餵給forward model得到z。然後將所有"沒有"該特徵的z取個平均。然後從我現在所選的，"有"該特徵的z，內插到平均的z。但是這樣做的結果是除了該項特徵外，其他特徵也越來越趨近平均值。

Results

Task1(ACGAN)

test:



new_test:

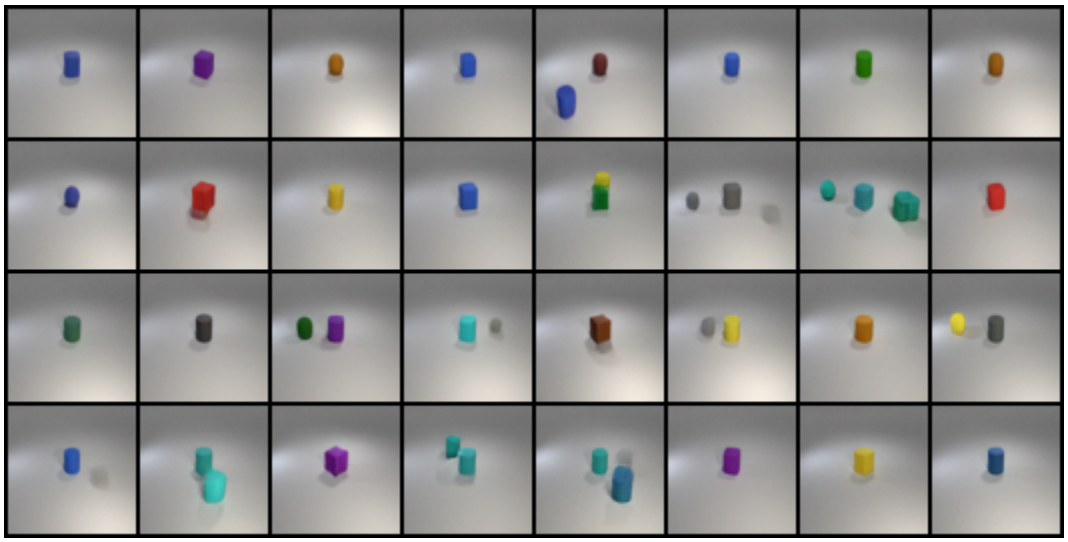
```
score: 0.67
score: 0.63
score: 0.65
score: 0.64
score: 0.64
score: 0.62
score: 0.65
score: 0.64
score: 0.65
score: 0.64

avg score: 0.65
```

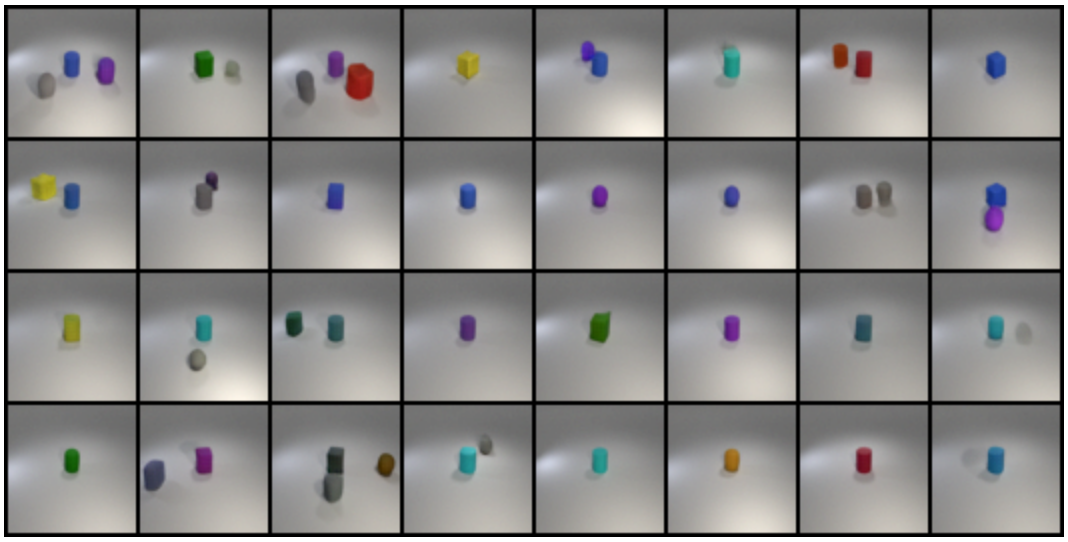
Task 1(c-Glow)

condition徹底失敗，已試過各種參數組合，生成能力很好，圖片很漂亮但是分數都<0.2。

test:



new_test:



Task 2 Conditional face generation

左至右特徵分別是：男性/女性。笑/不笑。皮膚蒼白/不蒼白。沒有鬍子/鬍子。

Task 2 Linear interpolation

Task 2 Attribute manipulation

第一列：男性。第二列：笑。第三列：蒼白皮膚。

Discuss

以往的cGAN只有分類真假的功能，而ACGAN還能作到圖片的多元分類。以往的cGAN都會將condition輸入至Discriminator，而ACGAN不這樣做，他做的事情反而是讓Discriminator有一個分類器當作輔助(Auxiliary Classifier)。

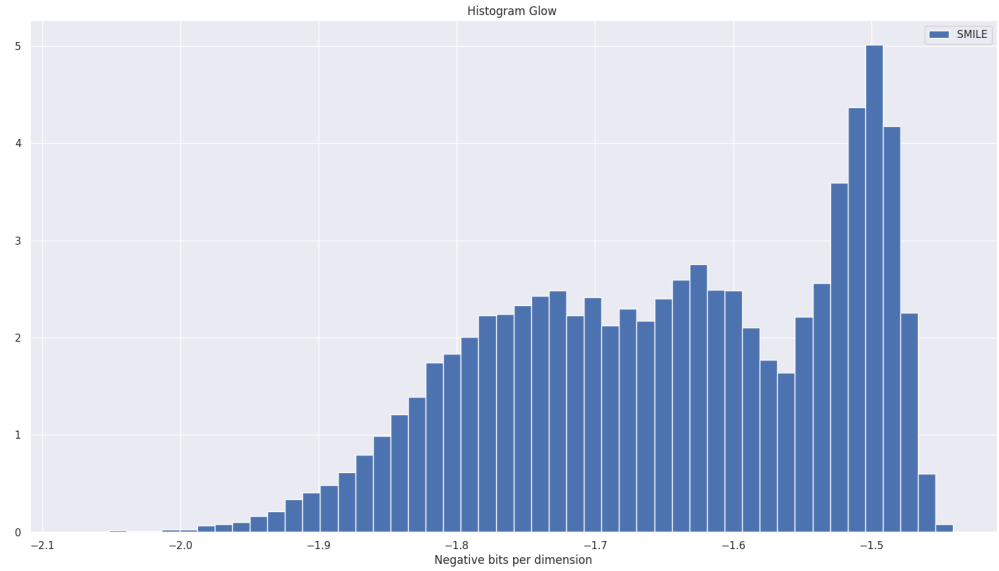
因為類別太多，ACGAN訓練過程不太收斂，Generator和Discriminator的loss都會很震盪。最後生成圖片的能力沒有很好，圖片解析度不高，看起來也蠻醜的。參考一些網路上的ACGAN實做，都很少用到有那麼多類別的資料集，也只處理單一類別的分類問題。

我的c-Glow套用在Task 1似乎完全沒有學到condition。loss收斂很快，可能已經overfit。生成的圖片非常清晰但是condition上一直沒有進展。

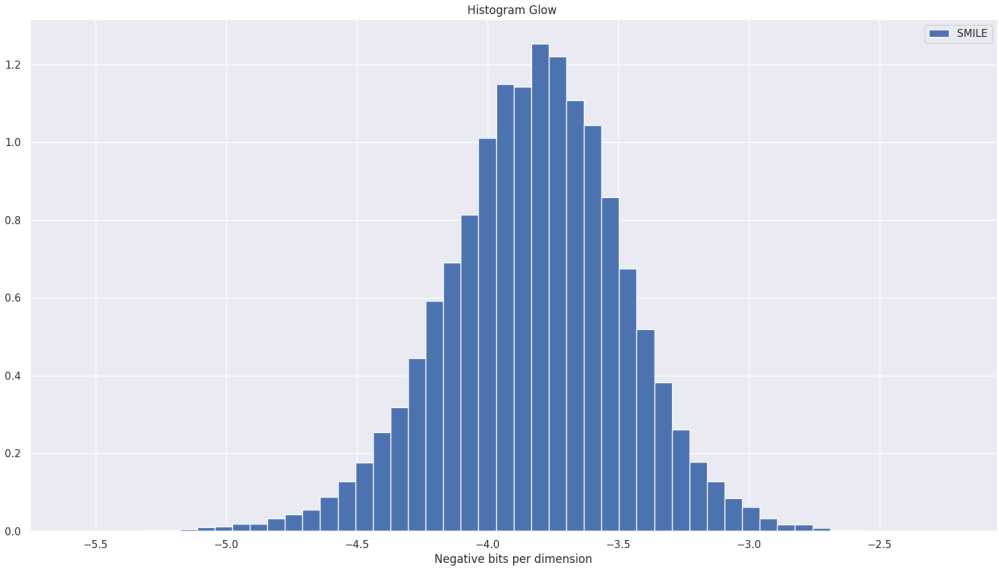
我參考的原始repo，實做的原始任務是重現Do Deep Generative models know what they don't know這篇paper中的實驗和結果。因此，他提供了一個衡量generative model的程式，能畫出bits per dimension的直方圖分佈。paper中指出log-likelihood越高越好，bits-per-dimension越低越好。

於是我也畫了一下我用Task 1 和Task 2兩個資料集訓練出來的model。

如下圖，Task 1 的BPD明顯偏高。c-Glow沒能好好的minimize BPD。



而Task 2 看起來似乎蠻不錯的:



在model的推導過程中，GAN沒有明確學習真實數據的機率密度函數 p(x)，$p(x)=\int p(x|z)p(z)dz$ 很難計算，因為遍歷latent z 的所有可能值是很困難的。

基於flow的深度生成模型借助NF（一種用於密度估計的強大統計工具）克服了這個難題。 對 p(x) 的良好估計可以有效地完成許多下游任務：採樣未觀察到但真實的新數據點（數據生成）、預測未來事件的稀有性（密度估計）、推斷潛在變量、填充不完整的數據樣本 ， 等等。

Flow和其他generative model很不一樣的地方就是該模型明確地學習數據分佈 p(x)。它的優勢和其他特性可以參考Normalizing Flows for Probabilistic Modeling and Inference這篇paper。主要優勢有，可逆映射，可以計算映射後的分佈體積，容易模擬等。但是當前流行的generative model還是以GAN為大宗，因為較容易用隨機sample noise的方式生出未知的新東西。