CAVEDU教育團隊部落格

[ 技術教學文-格式 ]

|  |  |  |  |
| --- | --- | --- | --- |
| **標題** | 利用Jetson Nano、Google Colab實作CycleGAN：將拍下來的照片、影片轉換成(偽)梵谷風格 – 基礎建構篇 | | |
|  | | | |
| **分類** |  | **標籤** |  |
| **撰寫/攝影** |  | | |
| **前情提要** |  | | |
| **時間** |  | **材料表** |  |
| **成本** |  |
| **難度** |  |
| **前言** | | | |
| 上次帶大家體驗了pix2pix並且將它應用在繪圖板上，畫出輪廓透過pix2pix去自動填上顏色，是不是很有趣呢？接下來我們再來玩一個風格轉換的經典作品Cycle GAN，並且將圖片轉換成梵谷風格圖片，那大家看到標題一定很好奇為什麼要叫做「偽」梵谷風格呢？因為看圖片就知道了，我總共只訓練了一百回合模型當然還沒訓練的很完整啦～不過這一百回合已經耗掉我一整個下午了！這次會教大家如何在Colab上實作、訓練，並且將自己訓練好的模型儲存到Jetson Nano上做應用。 | | | |
| **內文** | | | |
| |  |  | | --- | --- | | **作者** | **張嘉鈞** | | **難度** | 難 | | **材料表** | 1. Jetson Nano 2. PC or Notebook |  CycleGAN 在 pix2pix 的時候你可以發現數據都是成對的，所以在早期的風格轉換的案例中不僅僅是需要大量的數據，每一組數據都還必須是成雙成對！但現實中其實蒐集資料非常的費時以及困難，所以在pix2pix發表 (2016年11月) 過後沒多久便研發出不對稱的訓練方式，而這種訓練方式的鼻祖就是cycleGAN ( 2017年3月 )。  在CycleGAN圖片案例中，可以將真實圖片轉換成梵谷風格圖片，按照一般的GAN來思考，鑑別器只需要判斷生成是否真實、風格是否正確即可，但是仔細想想，不對稱的訓練會有一個問題，如果當我現在輸入任意一張真實圖片都變成了某一張特定的梵谷風格圖片，那神經網路分辨得出來嗎？  舉例來說，我輸入一張城堡的真實圖片，神經網路輸出給我一張梵谷的人像畫；按照神經網路的邏輯梵谷的人像畫，圖片風格也是梵谷風所以電腦覺得「我生成的很真實！！！」因為成像真實、風格也是梵谷風格的。  為了避免這種問題，CycleGAN提出了一個Reconstruction的概念：Reconstruction顧名思義就是重構的意思，我們將A風格圖片轉成B風格之後再將其轉換回A風格，檢查是否能成功轉回原始圖像。我們直接來看看概念圖，CycleGAN包含三個重點：    (a) 是基本的生成對抗架構，但CycleGAN總共有兩個鑑別器 ( , )、兩個生成器 ( )，代表有兩組GAN，其中 負責判斷 x 風格的圖片是否真實， 則是判斷 風格的圖片是否真實；G負責將 風格轉換成 風格，F則是將 轉成 。  (b) (c) 為重構的部分，分別稱為 forward cycle-consistency loss、backward cycle-consistency loss，將 轉換成 風格之後在將其轉換回原本風格 ( )，反之亦然。  **Google Colaboratory**  大致了解之後，先來簡單介紹一下Google Colaboratory (簡稱Colab)，它是Google出的一個深度學習平台，基礎的深度學習框架及套件都幫你安裝好，而且是類似於Jupyter Notebook的操作非常方便，更重要的是有提供每次12小時免費的GPU訓練 (這對於沒強力GPU算力的我是一大福音阿！)。這次因為我們的對手是CycleGAN，龐大又複雜的神經網路訓練就需要強大的GPU來支持拉！ 開始使用Colab安裝、開啟Colab 首先開啟你的Google雲端硬碟，在任意空白處右鍵 > 更多 > 連結更多應用程式    搜尋Colab並進行安裝即可    接著對空白處點右鍵就可以看到Google Colaboratory的選項了，而Jupyter Notebook的 ipnb檔也可以直接使用Colab開啟。    開啟後操作跟Jupyter幾乎是一樣的，所以就不多贅述了！主要的還是記得將環境改成GPU模式，在編輯 > 筆記本設定中可以調整，可以看到給的GPU還算是不錯了   綁定到雲端硬碟 執行的時候點擊網址，複製金鑰並且在輸入的方框中貼上   |  | | --- | | from google.colab import drive  drive.mount('/content/gdrive') |   成功掛接後會顯示現在的掛接的位置，接著就可以移動到檔案的位置   |  | | --- | | import os  os.chdir('/content/gdrive/My Drive/Cavedu/Article/cycleGAN')  %ls |   執行完成果如下：    這樣就已經到相同層級的目錄下了，接著就可以來打程式拉！透過程式碼去學習還是最快的呢～（好吧，對於理工宅宅來說比較適合） 處理數據 首先先來準備數據集，一樣到作者的數據庫中去撈取   |  | | --- | |  | | <https://people.eecs.berkeley.edu/~taesung_park/CycleGAN/> |   到datasets中找到 vangogh2photo下載下來並解壓縮到與程式碼同層    其中總共有四個資料夾，接著我們可以透過ImageFolder整理一個數據集並且透過DataLoader將其載入，這邊要注意的是ImageFolder會去吃下一層級資料夾的內容，所以如果你直接寫 ImageFolder(‘trainA’) 你會得到錯誤訊息，因為trainA裡面就是檔案了沒有資料夾，最值觀的做法就是再將所有檔案包一層資料夾，所以程式的部分我宣告了new\_trainA 並將 trainA放到他的底下，這樣ImageFolder就成功抓取到資料了：   |  | | --- | | ############ Prepare Data ############  transform = transforms.Compose(  [transforms.RandomHorizontalFlip(),  transforms.Resize((286, 286)),  transforms.RandomCrop((256, 256)),  transforms.ToTensor(),  transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])])  train\_path = r'.\vangogh2photo'  trainA\_path = os.path.join(train\_path, r'trainA')  targetA\_path = os.path.join(train\_path, r'new\_trainA')  trainB\_path = os.path.join(train\_path, r'trainB')  targetB\_path = os.path.join(train\_path, r'new\_trainB')  if os.path.exists(targetA\_path) == False:  os.makedirs(targetA\_path)  print('Create dir : ', targetA\_path)  shutil.move(trainA\_path, targetA\_path)  if os.path.exists(targetB\_path) == False:  os.makedirs(targetB\_path)  print('Create dir : ', targetB\_path)  shutil.move(trainB\_path, targetB\_path)  dataA\_loader = DataLoader(dsets.ImageFolder(targetA\_path, transform=transform), batch\_size=batch\_size, shuffle=True, num\_workers=4)  dataB\_loader = DataLoader(dsets.ImageFolder(targetB\_path, transform=transform), batch\_size=batch\_size, shuffle=True, num\_workers=4) |   這邊可以稍微做一下檢查，由於我們要同時匯入兩種風格的圖片，但他們又各自成一家 (各自有一個DataLoader)，所以我們必須透過 zip 這個函式把他們綁在一起，除了檢查圖片維度等等資訊之外，你們可以再呼叫show\_AB來看圖片是否正確：   |  | | --- | | import numpy as np  import cv2  def show\_AB():  img1 = data[0][0][0].numpy().transpose((1,2,0)) # vangogh  img2 = data[1][0][0].numpy().transpose((1,2,0)) # real pic  res = cv2.hconcat([img1, img2])  cv2.imshow('test' , res)  cv2.waitKey(0)  cv2.destroyAllWindows()  for idx, data in enumerate(zip(dataA\_loader, dataB\_loader)):  if idx > 0 :  break  else :  print(len(data)) # two data loader  print(data[0][0].shape) # trainA input\_x  print(data[0][1].shape) # trainA input\_y  print(data[0][0].shape[0]) # get batch size |  建立生成器 當數據資料已經導入，接下來就能來定義模型拉！首先先來建構一下生成器 (Generator) 跟鑑別器 (Discriminator)，以下簡稱G、D，這次我寫的CycleGAN當中，生成器用的技術不再是pix2pix所提到的U-Nets，而是改採用大名鼎鼎的ResNet核心技術- 殘缺塊 (Residual Block)    簡單講一下殘缺塊的概念，主要就是當神經網路越來越多層的時候，就可能會開始遇到梯度消失跟梯度爆炸等問題，而梯度消失你們可能會想到最直覺的就是LeakyReLU來約束 ( 不要等於0 )，而2015年的時候Kaiming He提出了更簡單的方式就是Skip Connection。假設說現在有四層權重在C-B的時候發生梯度消失，這時候梯度的訊息就沒辦法傳導到A，但是有了Skip Connection，D的梯度也能持續傳導到A不會受到影響，而這不僅僅能解決梯度消失的問題，也能改善梯度過小導致收斂過慢的問題。    上圖為該論文中提出的架構，其中Normalization的部份我們不是採用BatchNorm而是採用InstanceNorm，下圖為BN跟IN的示意圖 (HW為圖片像素或稱空間、C為通道、N為Batch的軸向)，可以看到BN會對 (H,W) 跟N都進行正規化，但是在風格轉換的案例中，結果通常會依賴在某幾張圖片的特徵，這時候BN會將該批次的都綜合平均掉，可能導致該張特徵降低、有偏差；所以為了不影響結果又能做正規化加快收斂速度，我們使用Instance Norm，針對每一個實例都進行正規化，而不是一個批次的實例一起正規化。    好， 生成器的細部技術終於帶完了，那我們直接來看程式碼吧！通常會先定義常用的層做一個區塊。在這次的實作當中　Conv-InstancNorm-ReLU 通常都會綁在一起，所以我先宣告副函示將其包裝在一起。   |  | | --- | | def conv\_norm\_relu(in\_dim, out\_dim, kernel\_size, stride = 1, padding=0):    layer = nn.Sequential(nn.Conv2d(in\_dim, out\_dim, kernel\_size, stride, padding),  nn.InstanceNorm2d(out\_dim),  nn.ReLU(True))  return layer  def dconv\_norm\_relu(in\_dim, out\_dim, kernel\_size, stride = 1, padding=0, output\_padding=0):    layer = nn.Sequential(nn.ConvTranspose2d(in\_dim, out\_dim, kernel\_size, stride, padding, output\_padding),  nn.InstanceNorm2d(out\_dim),  nn.ReLU(True))  return layer |   接著因為有殘缺塊所以會特別定義一個類別包裝殘缺塊：   |  | | --- | | class ResidualBlock(nn.Module):    def \_\_init\_\_(self, dim, use\_dropout):  super(ResidualBlock, self).\_\_init\_\_()  res\_block = [nn.ReflectionPad2d(1),  conv\_norm\_relu(dim, dim, kernel\_size=3)]    if use\_dropout:  res\_block += [nn.Dropout(0.5)]  res\_block += [nn.ReflectionPad2d(1),  nn.Conv2d(dim, dim, kernel\_size=3, padding=0),  nn.InstanceNorm2d(dim)]  self.res\_block = nn.Sequential(\*res\_block)  def forward(self, x):  return x + self.res\_block(x) |   看到這裡是不是覺得還有一點怪怪的，其實還有一個小技巧沒提到，這邊都沒使用Padding來讓圖片大小相同，反而去使用ReflectionPad，我自己的理解是要讓圖片顯得更自然，保留的細節更好。  在這裡我們定義生成器，總共使用了6層殘缺塊，通常會看到是６層或９層。   |  | | --- | | import torch  from torch import nn  from torchsummary import summary  class Generator(nn.Module):    def \_\_init\_\_(self, input\_nc=3, output\_nc=3, filters=64, use\_dropout=True, n\_blocks=2):  super(Generator, self).\_\_init\_\_()    # 向下採樣 ( shape + 2 \* padding - kernel + 1 ) / stride  # 256 + 3\*2 = 262  # 262 - 7 + 0 + 1 = 256  # ( 256 + 2 - 3 + 1 / 2 = 128  # 128 + 2 - 3 + 1 / 2 = 64  model = [nn.ReflectionPad2d(3),  conv\_norm\_relu(input\_nc , filters \* 1, 7),  conv\_norm\_relu(filters \* 1, filters \* 2, 3, 2, 1),  conv\_norm\_relu(filters \* 2, filters \* 4, 3, 2, 1)]  # 頸脖層  for i in range(n\_blocks):  model += [ResidualBlock(filters \* 4, use\_dropout)]  # 向上採樣 (input-1)\*stride + kernel - 2\*padding + output\_padding  # (64-1)\*2 + 3 -2 +1 = 128  # (128-1)\*2 + 3 -2 + 1 = 256  # 256 + 6 = 262  # 262 - 7 + 1 = 256  model += [dconv\_norm\_relu(filters \* 4, filters \* 2, 3, 2, 1, 1),  dconv\_norm\_relu(filters \* 2, filters \* 1, 3, 2, 1, 1),  nn.ReflectionPad2d(3),  nn.Conv2d(filters, output\_nc, 7),  nn.Tanh()]  self.model = nn.Sequential(\*model) # model 是 list 但是 sequential 需要將其透過 , 分割出來  def forward(self, x):  return self.model(x) |   最後我們透過之前介紹的torchsummary來顯示出來：   |  | | --- | | G = Generator()  summary(G, (3,256,256)) |    建立鑑別器 鑑別器相對就簡單很多了，我們一樣使用PatchGAN的技術，簡單複習一下，一般生成對抗的鑑別器會輸出一個數值介於 [0, 1] 之間，但PatchGAN的技術輸出的是一組tensor，代表圖像中各區域的真實程度；其他的部分就是一般DCGAN要注意的像是使用LeakyReLU以及輸出層用tanh等。   |  | | --- | | import torch  from torch import nn  def conv\_norm\_leakyrelu(in\_dim, out\_dim, kernel\_size, stride = 1, padding=0, output\_padding=0):  layer = nn.Sequential(nn.Conv2d(in\_dim, out\_dim, kernel\_size, stride, padding),  nn.InstanceNorm2d(out\_dim),  nn.LeakyReLU(0.2,True))  return layer  class Discriminator(nn.Module):    def \_\_init\_\_(self, input\_nc=3, filters=64, n\_layer = 3):  super(Discriminator, self).\_\_init\_\_()      # 第一層不做 batchNorm  # 256 -1 +1 = 256  model = [  nn.Conv2d(input\_nc, filters, kernel\_size=1, stride=1, padding=0),  nn.LeakyReLU(0.2, True)]    # 第二、三層相同  # 256 +2 -4 +1 / 2 =  for i in range(1, n\_layer):  n\_filters\_prev = 2\*\*(i-1)  n\_filters = 2\*\*i  model += [conv\_norm\_leakyrelu(filters \* n\_filters\_prev , filters \* n\_filters, kernel\_size=4,  stride=2, padding=1)]  # 第四層 stride 為 1  n\_filters\_prev = 2\*\*(n\_layer-1)  n\_filters = 2\*\*n\_layer  model += [conv\_norm\_leakyrelu(filters \* n\_filters\_prev , filters \* n\_filters, kernel\_size=4,  stride=1, padding=1)]  # 輸出層  model += [nn.Conv2d(filters \* n\_filters, 1, kernel\_size=4, stride=1, padding=1)]    self.model = nn.Sequential(\*model)  def forward(self, input):  return self.model(input)    D = Discriminator()  D = Discriminator()  summary(D, (3,256,256)) |     從這邊的輸出可以注意到我們輸出的是 (62, 62) 的tensor，再次驗證了某些論文看似很難執行起來卻很簡單的道理。 定義CycleGAN 看到這裡恭喜你已經將CycleGAN摸了一半了，接著我們在訓練CycleGAN之前要先將依些基本的參數給定義好：   |  | | --- | | ###### initial ######  def weights\_init\_normal(m):  classname = m.\_\_class\_\_.\_\_name\_\_  if classname.find('Conv') != -1:  init.normal\_(m.weight.data, 0.0, 0.02)  ###### basic parameters ######    device = 'cuda:0' if torch.cuda.is\_available() else 'cpu'  batch\_size = 4  epochs = 100  decay\_epoch = 10  lr = 2e-3  log\_freq = 100 |   CycleGAN總共需要兩組GAN，這邊也先定義好，並且套用初始化權重的副函式：  G\_A2B – 學習將A風格轉換成B風格  G\_A2B – 學習將B風格轉換成A風格  D\_A – 學習判斷是否為A風格的圖像  D\_B -學習判斷是否為B風格的圖像   |  | | --- | | ############ Define Model ############  G\_A2B = Generator().to(device)  G\_B2A = Generator().to(device)  D\_A = Discriminator().to(device)  D\_B = Discriminator().to(device)  G\_A2B.apply(weights\_init\_normal)  G\_B2A.apply(weights\_init\_normal)  D\_A.apply(weights\_init\_normal)  D\_B.apply(weights\_init\_normal) |   接著定義損失函式、優化器、學習率的更新工具，這邊要注意的是我們需要定義兩個損失函式 (MSE、L1)，下一篇訓練的時候會再提到；再來就是學習率的更新器了，在PyTorch當中更新學習率的方法有很多種，除了直接提取出來更改之外還有scheduler的方法，詳細就不多做介紹，在相關文章中我有貼一篇在講lr\_scheduler的文章可以參考看看，在這邊我LambdaLR的更新方式參考了這一篇github <https://github.com/Lornatang/CycleGAN-PyTorch> 的寫法：   |  | | --- | | ############ define Loss function ############  MSE = nn.MSELoss()  L1 = nn.L1Loss()  ############ define optimizer ############  class LambdaLR():  def \_\_init\_\_(self, epochs, offset, decay\_epoch):  self.epochs = epochs  self.offset = offset  self.decay\_epoch = decay\_epoch  def step(self, epoch):  return 1.0 - max(0, epoch + self.offset - self.decay\_epoch)/(self.epochs - self.decay\_epoch)  optim\_G = torch.optim.Adam(itertools.chain(G\_A2B.parameters(), G\_B2A.parameters()), lr=lr, betas=(0.5, 0.999))  optim\_D = torch.optim.Adam(itertools.chain(D\_A.parameters() , D\_B.parameters()), lr=lr, betas=(0.5, 0.999))    lr\_scheduler\_G = torch.optim.lr\_scheduler.LambdaLR(optim\_G,lr\_lambda=LambdaLR(epochs, 0, decay\_epoch).step)  lr\_scheduler\_D = torch.optim.lr\_scheduler.LambdaLR(optim\_D, lr\_lambda=LambdaLR(epochs, 0, decay\_epoch).step) |   在訓練之前我們還要定義最後一個東西，叫做ReplayBuffer，其實不使用他也是可以的，因為我最一開始是用tensorflow寫cycleGAN也沒提到相關技術一樣可以訓練，而稍微查了一下類似的技術在Q-learning中比較常見，在這邊使用的概念比較像是儲存50張生成的圖像並且從其中隨機抓取來做預測，目的可能是不要讓神經網路學習到順序的規則，如果理解有錯誤還請各位大神糾正。   |  | | --- | | # To store 50 generated image in a pool and sample from it when it is full  # Shrivastava et al’s strategy  class ReplayBuffer:  def \_\_init\_\_(self, max\_size=50):  assert (max\_size > 0), "Empty buffer or trying to create a black hole. Be careful."  self.max\_size = max\_size  self.data = []  def push\_and\_pop(self, data):  to\_return = []  for element in data.data:  element = torch.unsqueeze(element, 0)  if len(self.data) < self.max\_size:  self.data.append(element)  to\_return.append(element)  else:  if random.uniform(0, 1) > 0.5:  i = random.randint(0, self.max\_size - 1)  to\_return.append(self.data[i].clone())  self.data[i] = element  else:  to\_return.append(element)  return torch.cat(to\_return)    fake\_A\_sample = ReplayBuffer()  fake\_B\_sample = ReplayBuffer() |   到這邊我們已經將CycleGAN要準備的都搞定了，接下來就是訓練的重頭戲，我們將在下一篇帶大家實作如何訓練、預測以及應用！ | | | |
| **相關文章** | | | |
| CycleGAN 論文  <https://arxiv.org/pdf/1703.10593.pdf>  Pytorch中的学习率衰减方法  <https://www.jianshu.com/p/9643cba47655>  CycleGAN using PyTorch  <https://github.com/arnab39/cycleGAN-PyTorch>  CycleGAN-PyTorch  <https://github.com/Lornatang/CycleGAN-PyTorch> | | | |