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、影片轉換成梵谷風格 – 訓練、預測以及應用篇 | | |
|  | | | |
| **分類** |  | **標籤** |  |
| **撰寫/攝影** |  | | |
| **前情提要** |  | | |
| **時間** |  | **材料表** |  |
| **成本** |  |
| **難度** |  |
| **前言** | | | |
| 延續上一篇，我們即將開始訓練跟測試！此外還想做一些應用，就如標題所說拍照進行風格轉換甚至是影片轉換，看到這邊就準備大量的時間跟我一起來訓練神經網路吧！ | | | |
| **內文** | | | |
| 訓練CycleGAN 首先先取得訓練資料：   |  | | --- | | from tqdm import tqdm  import torchvision.utils as vutils  total\_len = len(dataA\_loader) + len(dataB\_loader)  for epoch in range(epochs):  progress\_bar = tqdm(enumerate(zip(dataA\_loader, dataB\_loader)), total = total\_len)  for idx, data in progress\_bar:  ############ define training data & label ############  real\_A = data[0][0].to(device) # vangogh image  real\_B = data[1][0].to(device) # real picture |   我們要先訓練G，總共有三個標準要來衡量生成器：   1. 是否能騙過鑑別器 (Adversial Loss )：對於G\_B2A來說，將A轉換成B之後給予1的標籤，並且計算跟real\_B 之間的距離。  |  | | --- | | ############ Train G ############  optim\_G.zero\_grad()    ############ Train G - Adversial Loss ############  fake\_A = G\_B2A(real\_B)  fake\_out\_A = D\_A(fake\_A)  fake\_B = G\_A2B(real\_A)  fake\_out\_B = D\_B(fake\_B)  real\_label = torch.ones( (fake\_out\_A.size()) , dtype=torch.float32).to(device)  fake\_label = torch.zeros( (fake\_out\_A.size()) , dtype=torch.float32).to(device)  adversial\_loss\_B2A = MSE(fake\_out\_A, real\_label)  adversial\_loss\_A2B = MSE(fake\_out\_B, real\_label)  adv\_loss = adversial\_loss\_B2A + adversial\_loss\_A2B |  1. 是否能重新建構 (Consistency Loss)：舉例 G\_B2A(real\_B) 產生風格A的圖像 (fake\_A) 後，再丟進 G\_A2B(fake\_A) 重新建構成B風格的圖像 (rec\_B)，並且計算 real\_B 跟 rec\_B之間的差距。  |  | | --- | | ############ G - Consistency Loss (Reconstruction) ############  rec\_A = G\_B2A(fake\_B)  rec\_B = G\_A2B(fake\_A)  consistency\_loss\_B2A = L1(rec\_A, real\_A)  consistency\_loss\_A2B = L1(rec\_B, real\_B)  rec\_loss = consistency\_loss\_B2A + consistency\_loss\_A2B |  1. 是否能保持一致 (Identity Loss)：以G\_A2B來說，是否在丟入 real\_B的圖片後，確實能輸出 B風格的圖片，是否能保持原樣？  |  | | --- | | ############ G - Identity Loss ############  idt\_A = G\_B2A(real\_A)  idt\_B = G\_A2B(real\_B)  identity\_loss\_A = L1(idt\_A, real\_A)  identity\_loss\_B = L1(idt\_B, real\_B)  idt\_loss = identity\_loss\_A + identity\_loss\_B |   最後將其所有損失都計算梯度，並且更新參數，這邊可以注意到重構的loss乘上10，而一致性的部分乘上5，代表在CycleGAN當中能不能重構佔了相當大的比例。   |  | | --- | | ############ G - Total Loss ############  lambda\_rec = 10  lambda \_idt = 5  loss\_G = adv\_loss + ( rec\_loss \* lambda \_rec ) + ( idt\_loss \* lambda \_idt )    ############ G - Backward & Update ############  loss\_G.backward()  optim\_G.step() |   接著訓練D，它只要將自己的本份顧好就好了，也就是「能否分辨得出該風格的成像是否真實」。   |  | | --- | | ############ Train D ############  optim\_D.zero\_grad()    ############ D - Adversial D\_A Loss ############  real\_out\_A = D\_A(real\_A)  real\_out\_A\_loss = MSE(real\_out\_A, real\_label)  fake\_out\_A = D\_A(fake\_A\_sample.push\_and\_pop(fake\_A))  fake\_out\_A\_loss = MSE(real\_out\_A, fake\_label)    loss\_DA = real\_out\_A\_loss + fake\_out\_A\_loss    ############ D - Adversial D\_B Loss ############    real\_out\_B = D\_B(real\_B)  real\_out\_B\_loss = MSE(real\_out\_B, real\_label)  fake\_out\_B = D\_B(fake\_B\_sample.push\_and\_pop(fake\_B))  fake\_out\_B\_loss = MSE(fake\_out\_B, fake\_label)    loss\_DB = ( real\_out\_B\_loss + fake\_out\_B\_loss )    ############ D - Total Loss ############    loss\_D = ( loss\_DA + loss\_DB ) \* 0.5    ############ Backward & Update ############    loss\_D.backward()  optim\_D.step() |   最後我們可以將一些資訊透過tqdm印出來   |  | | --- | | ############ progress info ############  progress\_bar.set\_description(  f"[{epoch}/{epochs - 1}][{idx}/{len(dataloader) - 1}] "  f"Loss\_D: {(loss\_DA + loss\_DB).item():.4f} "  f"Loss\_G: {loss\_G.item():.4f} "  f"Loss\_G\_identity: {(idt\_loss).item():.4f} "  f"loss\_G\_GAN: {(adv\_loss).item():.4f} "  f"loss\_G\_cycle: {(rec\_loss).item():.4f}") |   接著訓練GAN非常重要的環節就是要記得儲存權重，因為說不定訓練第100回合的效果比200回合的還要好，所以都會傾向一定的回合數就儲存一次。儲存的方法很簡單大家可以上PyTorch的官網查看，大致上總共有兩種儲存方式：   1. 儲存模型結構以及權重  |  | | --- | | torch.save( model ) |  1. 只儲存權重  |  | | --- | | torch.save( model.static\_dict() ) |   而我採用的方式是只儲存權重，這也是官方建議的方案：   |  | | --- | | if i % log\_freq == 0:    vutils.save\_image(real\_A, f"{output\_path}/real\_A\_{epoch}.jpg", normalize=True)  vutils.save\_image(real\_B, f"{output\_path}/real\_B\_{epoch}.jpg", normalize=True)    fake\_A = ( G\_B2A( real\_B ).data + 1.0 ) \* 0.5  fake\_B = ( G\_A2B( real\_A ).data + 1.0 ) \* 0.5    vutils.save\_image(fake\_A, f"{output\_path}/fake\_A\_{epoch}.jpg", normalize=True)  vutils.save\_image(fake\_B, f"{output\_path}/fake\_A\_{epoch}.jpg", normalize=True)      torch.save(G\_A2B.state\_dict(), f"weights/netG\_A2B\_epoch\_{epoch}.pth")  torch.save(G\_B2A.state\_dict(), f"weights/netG\_B2A\_epoch\_{epoch}.pth")  torch.save(D\_A.state\_dict(), f"weights/netD\_A\_epoch\_{epoch}.pth")  torch.save(D\_B.state\_dict(), f"weights/netD\_B\_epoch\_{epoch}.pth")  ############ Update learning rates ############  lr\_scheduler\_G.step()  lr\_scheduler\_D.step()  ############ save last check pointing ############  torch.save(netG\_A2B.state\_dict(), f"weights/netG\_A2B.pth")  torch.save(netG\_B2A.state\_dict(), f"weights/netG\_B2A.pth")  torch.save(netD\_A.state\_dict(), f"weights/netD\_A.pth")  torch.save(netD\_B.state\_dict(), f"weights/netD\_B.pth") |  測試 其實測試非常的簡單，跟著以下的步驟就可以完成：   1. 導入函式庫  |  | | --- | | import os  import torch  import torchvision.datasets as dsets  from torch.utils.data import DataLoader  import torchvision.transforms as transforms  from tqdm import tqdm  import torchvision.utils as vutils |  1. 將測試資料建一個數據集並透過DataLoader載入：這邊我創了一個Custom資料夾存放我自己的數據，並且新建了一個output資料夾方便察看結果。  |  | | --- | | batch\_size = 12  device = 'cuda:0' if torch.cuda.is\_available() else 'cpu'  transform = transforms.Compose( [transforms.Resize((256,256)),  transforms.ToTensor(),  transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])])  root = r'vangogh2photo'  targetC\_path = os.path.join(root, 'custom')  output\_path = os.path.join('./', r'output')  if os.path.exists(output\_path) == False:  os.mkdir(output\_path)  print('Create dir : ', output\_path)  dataC\_loader = DataLoader(dsets.ImageFolder(targetC\_path, transform=transform), batch\_size=batch\_size, shuffle=True, num\_workers=4) |  1. 實例化生成器、載入權重 (load\_static\_dict)、選擇模式 ( train or eval )，如果選擇 eval，PyTorch會將Drop給自動關掉；因為我只要真實照片轉成梵谷所以只宣告了G\_B2A：  |  | | --- | | # get generator  G\_B2A = Generator().to(device)  # Load state dicts  G\_B2A.load\_state\_dict(torch.load(os.path.join("weights", "netG\_B2A.pth")))  # Set model mode  G\_B2A.eval() |  1. 開始進行預測：取得資料＞丟進模型取得輸出＞儲存圖片  |  | | --- | | progress\_bar = tqdm(enumerate(dataC\_loader), total=len(dataC\_loader))  for i, data in progress\_bar:  　　　# get data  real\_images\_B = data[0].to(device)  # Generate output  fake\_image\_A = 0.5 \* (G\_B2A(real\_images\_B).data + 1.0)  # Save image files  vutils.save\_image(fake\_image\_A.detach(), f"{output\_path}/FakeA\_{i + 1:04d}.jpg", normalize=True)  progress\_bar.set\_description(f"Process images {i + 1} of {len(dataC\_loader)}") |  1. 去output察看結果！可能是因為我只有訓練100回合，梵谷風格的細節線條還沒學起來，大家可以嘗試再訓練久一點，理論上200回合就會有不錯的成果了！  |  | | --- | | ORIGINAL | |  | | TRANSFORM | |  |   好的，那現在已經會建構、訓練以及預測了，接下來我們來想個辦法應用它！講到Style Transfer的應用，第一個就想到微軟大大提供的Style Transfer Azure Website。 Azure 的Style Transfer  |  | | --- | |  | | <https://styletransfers.azurewebsites.net/> |   這種拍一張照片就可以直接做轉換的感覺真的很棒！所以我們理論上也可以透過簡單的opencv程式來完成這件事情，再實作之前先去體驗看看Style Transfer 。    按下Create就能進來這個頁面，透過點擊Capture就可以拍照進行轉換也可以點擊Upload a picture上傳照片，總共有4種風格可以選擇：   |  |  |  |  | | --- | --- | --- | --- | | Candy | Feather | Mosaic | Robert | |  |  |  |  |   感覺真的超級酷的！所以我們也來試著實作類似的功能： 在JetsonNano中進行風格轉換  1. 首先要將權重放到Jetson Nano中，我新增了一個weights資料夾並且將netG\_B2A.pth放入其中，此外還在同一層級新增了jupyter book的程式：      1. **重建生成器並導入權重值** 這邊可能會有版本問題，像我就必須升級成Torch 1.6版本，而安裝PyTorch的方法我會放在文章結尾補述，回歸正題，還記得剛剛我儲存的時候只有儲存權重對吧，所以我們必須建一個跟當初訓練一模一樣的模型才能匯入哦！所以來複製一下之前寫的生成器吧！  |  | | --- | | import torch  from torch import nn  from torchsummary import summary  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  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)  class Generator(nn.Module):    def \_\_init\_\_(self, input\_nc=3, output\_nc=3, filters=64, use\_dropout=True, n\_blocks=6):  super(Generator, self).\_\_init\_\_()    # 向下採樣  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  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) |   接下來要做實例化模型並導入權重：   |  | | --- | | def init\_model():    device = 'cuda:0' if torch.cuda.is\_available() else 'cpu'  G\_B2A = Generator().to(device)  G\_B2A.load\_state\_dict(torch.load(os.path.join("weights", "netG\_B2A.pth"), map\_location=device ))  G\_B2A.eval()    return G\_B2A |  1. **在Colab中拍照**   我先寫了一個副函式來進行模型的預測，丟進去的圖片記得也要做transform，將大小縮放到256、轉換成tensor以及正規化，這部分squeeze目的是要模擬成有batch\_size的格式：   |  | | --- | | def test(G, img):  device = 'cuda:0' if torch.cuda.is\_available() else 'cpu'  transform = transforms.Compose([transforms.Resize((256,256)),  transforms.ToTensor(),  transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])])  data = transform(img).to(device)  data = data.unsqueeze(0)  out = (0.5 \* (G(data).data + 1.0)).squeeze(0) return out |   我們接著使用OpenCV來完成拍照，按下q離開，按下s進行儲存，那我們可以在按下s的時候進行風格轉換，存下兩種風格的圖片，這邊要注意的是PyTorch吃的是PIL的圖檔格式，所以還必須將OpenCV的nparray格式轉換成PIL.Image格式：   |  | | --- | | if \_\_name\_\_=='\_\_main\_\_':    G = init\_model()    trans\_path = 'test\_transform.jpg'  org\_path = 'test\_original.jpg'    cap = cv2.VideoCapture(0)  while(True):  ret, frame = cap.read()    cv2.imshow('webcam', frame)  key = cv2.waitKey(1)  if key==ord('q'):  cap.release()  cv2.destroyAllWindows()  break  elif key==ord('s'):    output = test(G, Image.fromarray(frame))  style\_img = np.array(output.cpu()).transpose([1,2,0])  org\_img = cv2.resize(frame, (256, 256))    cv2.imwrite(trans\_path, style\_img\*255)  cv2.imwrite(org\_path, org\_img)  break    cap.release()  cv2.destroyWindow('webcam') |   執行的畫面如下：    最後再將兩種風格照片合併顯示出來：   |  | | --- | | res = np.concatenate((style\_img, org\_img/255), axis=1)  cv2.imshow('res',res )  cv2.waitKey(0)  cv2.destroyAllWindows() |    在Jetson Nano中做即時影像轉換 概念跟拍照轉換雷同，這邊我們直接在取得到攝影機的圖像之後就做風格轉換，我額外寫了一個判斷，按下t可以進行風格轉換，並且用cv2.putText將現在風格的標籤顯示在左上角。   |  | | --- | | if \_\_name\_\_=='\_\_main\_\_':    G = init\_model()  cap = cv2.VideoCapture(0)  change\_style = False  save\_img\_name = 'test.jpg'  cv2text = ''  while(True):  ret, frame = cap.read()  # Do Something Cool  ############################  if change\_style:  style\_img = test(G, Image.fromarray(frame))  out = np.array(style\_img.cpu()).transpose([1,2,0])  cv2text = 'Style Transfer'  else:  out = frame  cv2text = 'Original'    out = cv2.resize(out, (512, 512))  out = cv2.putText(out, f'{cv2text}', (20, 40), cv2.FONT\_HERSHEY\_SIMPLEX ,  1, (255, 255, 255), 2, cv2.LINE\_AA)  ###########################  cv2.imshow('webcam', out)  key = cv2.waitKey(1)  if key==ord('q'):  break  elif key==ord('s'):  if change\_style==True:  cv2.imwrite(save\_img\_name,out\*255)  else:  cv2.imwrite(save\_img\_name,out)  elif key==ord('t'):  change\_style = False if change\_style else True  cap.release()  cv2.destroyAllWindows() |  即時影像風格轉換成果  結語 這次GAN影像風格轉換的部分就告一段落了，利用Colab來訓練風格轉換的範例真的還是偏硬了一點，雖然我們只有訓練100回合但也跑了半天多一點了，但是！GAN就是個需要耐心的模型，不跑個三天兩夜他是不會給你多好的成效的。  至於在Inference的部分，Jetson Nano還是擔當起重要的角色，稍微有一些延遲不過還算是不錯的了，或許可以考慮透過ONNX轉換成TensorRT再去跑，應該又會加快許多了，下一次又會有什麼GAN的範例大家可以期待一下，或者留言跟我說。 補充 – Nano 安裝Torch 1.6的方法 首先，JetPack版本要升級到4.4哦！不然CUDA核心不同這部分官網就有升級教學所以就不多贅述了。  將PyTorch等相依套件更新至1.6版本：   |  | | --- | | $ wget https://nvidia.box.com/shared/static/yr6sjswn25z7oankw8zy1roow9cy5ur1.whl -O torch-1.6.0rc2-cp36-cp36m-linux\_aarch64.whl  $ sudo apt-get install python3-pip libopenblas-base libopenmpi-dev  $ pip3 install Cython  $ pip3 install torch-1.6.0rc2-cp36-cp36m-linux\_aarch64.whl |   將TorchVision更新至對應版本：   |  | | --- | | $ sudo apt-get install libjpeg-dev zlib1g-dev  $ git clone --branch v0.7.0 https://github.com/pytorch/vision torchvision  $ cd torchvision  $ export BUILD\_VERSION=0.7.0 # where 0.x.0 is the torchvision version  $ sudo python3 setup.py install # use python3 if installing for Python 3.6  $ cd ../ # attempting to load torchvision from build dir will result in import error | | | | |
| **相關文章** | | | |
|  | | | |