REPORT - ASSIGNMENT 3

PRINCY GAUTAM
B20BB051

Dataset upload

- This code is implementing a deep learning model for image manipulation using the CelebA dataset. The code begins by importing the necessary libraries such as PyTorch, Numpy, and Matplotlib. It also imports some classes from the torchvision package.
- Next, it mounts the Google Drive to the Colab instance by importing the drive module from Google Colab and using the drive.mount() function to mount the Google Drive at the specified directory.
- After that, it unzips the img_align_celeba.zip[1] file to the /content/gdrive/My Drive/data/ directory using the !unzip command.
- The code then defines two transformation functions using the transforms module from the torchvision package, one for the input and one for the output.

Data preprocessing

- Next, it defines a load_image function that takes in four parameters, the root directory, directory name, file name, and extension, and returns the image loaded using the Image.open() method from the PIL library.
- Then, it defines a CelebADataset class that inherits from the torch.utils.data.Dataset class. It takes in four parameters, the directory, the input transformation function, the output transformation function, and the selected attributes. The class reads the attributes file and creates a dictionary mapping the filename to the selected attributes. It also creates a list of input files to be used by the dataset. The __getitem__() method returns the input image and its target attributes at the specified index. It uses the load_image() function to load the input image, applies the input transformation function to it, and returns the target attributes as a tensor. The __len__() method returns the length of the list of inputs.
- After that, it sets the batch size to 16 and creates a CelebADataset object with the specified parameters. It then creates a DataLoader object with the dataset and the batch size, and sets shuffle=True.
- The code checks if a GPU is available and sets the device accordingly. It prints the length of the data loader and the device used for training.

Selection of 8 attributes

- To get the top 8 attributes with maximum variability we use PCAfrom sklearn library:
    - This code is performing Principal Component Analysis (PCA) on the CelebA dataset, which contains facial images of celebrities and their associated attribute labels.
    - First, the code unzips the CelebA dataset, which contains a CSV file with the attribute labels for each image. The CSV file is then read into a pandas dataframe, where the

'image_id' column is set as the index. Any values of '-1' in the dataframe are replaced with '0' to indicate the absence of the attribute in the image.
    - Next, PCA[3] is applied to the data to reduce its dimensionality to 8 principal components. The loadings of the principal components are then calculated, and the top 8 attributes with the highest loadings for each principal component are printed out. The attribute names are based on the original column names of the CelebA dataset.

Model and training

- This is a PyTorch script for training an attribute prediction model using a VGG16 backbone on a dataset of face images. The model takes an input image and outputs a binary vector of size 8, indicating the presence or absence of certain facial attributes (such as "male", "big nose", "bags under eyes", etc.). Here's a step-by-step breakdown of the code:
- The script defines a list of 8 attributes that the model will predict.
- It imports the necessary PyTorch modules: torch.nn.functional for defining the loss function, torchvision.models.vgg16 for the pre-trained VGG16 backbone, and torch.optim for defining the optimizer.
- It defines a custom PyTorch module called AttributePredictionModel that inherits from nn.Module. This module has three parts:
    - self.features: the pre-trained VGG16 backbone[2]
    - self.avgpool: an adaptive average pooling layer that converts the output of the backbone to a fixed-size tensor
    - self.classifier: a three-layer fully connected network that takes the output of the pooling layer and produces an output vector of size 8 (corresponding to the 8 attributes to predict)
- It defines the loss function as nn.BCEWithLogitsLoss(), which combines a sigmoid activation function with a binary cross-entropy loss.
- It creates an instance of AttributePredictionModel and moves it to the GPU (if available). It also freezes the weights of the pre-trained VGG16 backbone so that they don't get updated during training.
- It defines an optimizer using optim.Adam() and sets the learning rate to 0.0002.
- It defines the number of training epochs and enters a training loop that iterates over the training dataset (which is assumed to be loaded into train_dataloader). For each batch of images and labels:
    - It moves the data to the GPU (if available).
    - It zeroes the gradients of the optimizer.
    - It computes the forward pass of the model and calculates the loss.
    - It computes the backward pass and performs a parameter update step.
    - It computes some statistics for tracking progress (loss, accuracy, etc.).
- After each epoch, it prints out the epoch number, loss, and accuracy for each of the 8 attributes.
- Finally, it prints out the overall accuracy of the model on the training set.

Analysis

a. Mention your choice of attributes, backbone, and other parameters and the reason behind your choice.

Choice of attributes
The 8 attribute selection is based on the task executed by Principal Component Analysis (PCA) which return the the 8 attributes which have highest loading for each principal components are printed out. Thus we select the following 8 attributes:

Top_8_attributes = ['Male', 'Big_Nose', 'Bags_Under_Eyes', '5_o_Clock_Shadow', 'Wearing_Necktie', 'Goatee', 'Sideburns', 'Bushy_Eyebrows']

Backbone
The VGG16 is a deeper and more complex model than ResNet18, with 16 convolutional layers compared to 18 layers in ResNet18. This makes VGG16 better suited for handling large and complex datasets with high inter-class variations, such as CelebA, which contains over 200K celebrity face images with 40 different attribute annotations.

Parameters
Adam optimizer - based on accuracy and sets the learning rate to 0.0002.
Loss function - 'nn.BCEWithLogitsLoss()', which combines a sigmoid activation function with a binary cross-entropy loss.
Batch size - we use a smaller batch size (here, 16) since this is a large dataset using batch size greater than 32 gives gpu out of memory error.

Training time - 6.36 it/s for every epoch

b. Report task-wise accuracy.

```
Epoch 10/10 - Loss: 0.4301 - Acc: 88.2316
      Male: 83.1719
      Big_Nose: 85.3926
      Bags_Under_Eyes: 89.2603
      5_o_Clock_Shadow: 90.0010
      Wearing_Necktie: 94.0932
      Goatee: 87.5392
      Sideburns: 80.4123
      Bushy_Eyebrows: 91.6702
```

c. Report the overall accuracy.

```
Overall accuracy: 92.32
```

References

[1] Liu, Ziwei, Ping Luo, Xiaogang Wang, and Xiaoou Tang. "Large-scale celebfaces attributes (celeba) dataset." Retrieved August 15, no. 2018 (2018): 11.
[2] https://medium.com/@mygreatlearning/everything-you-need-to-know-about-vgg16-7315defb5918

[3] https://vitalflux.com/feature-extraction-pca-python-example/