CSE 5370: BIO-INFORMATICS

HOMEWORK – 4

1)

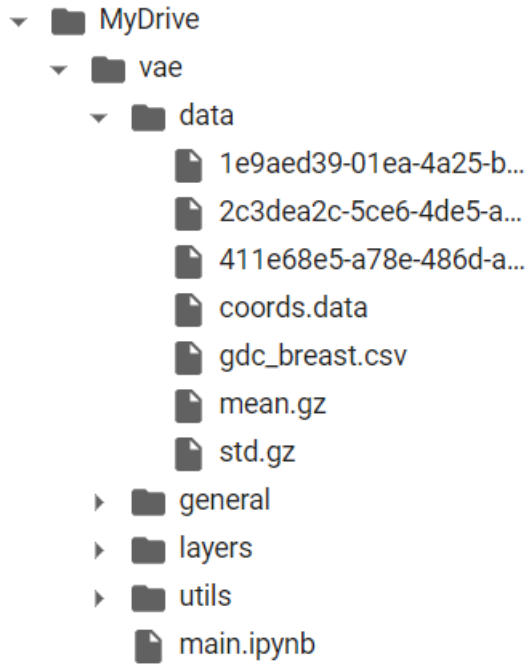
VAE Implementation

2)

Pytorch Implementation

Implemented the assignment on Google Colab as per the instructions

2.1) Setting up the Framework



2.2) Downloading the Dataset

```
[5]  from utils.download_tools import download_gdc_data

[6]  DATA_PATH = join(PROJ_PATH, "data")
     GDC_CSV_PATH = join(DATA_PATH, "gdc_breast.csv")

[7]  download_gdc_data(STUDENT_ID, num_samples=3, csv_path=GDC_CSV_PATH, save_dir=DATA_PATH)

     drive/MyDrive/vae/data/2c3dea2c-5ce6-4de5-ad15-b3f9416b0ab4.svs already exists! Download skipped...
     You can view the image at https://portal.gdc.cancer.gov/files/2c3dea2c-5ce6-4de5-ad15-b3f9416b0ab4...
     drive/MyDrive/vae/data/1e9aed39-01ea-4a25-b1a1-14b7ab37b982.svs already exists! Download skipped...
     You can view the image at https://portal.gdc.cancer.gov/files/1e9aed39-01ea-4a25-b1a1-14b7ab37b982...
     drive/MyDrive/vae/data/411e68e5-a78e-486d-aba6-fdef038f9082.svs already exists! Download skipped...
     You can view the image at https://portal.gdc.cancer.gov/files/411e68e5-a78e-486d-aba6-fdef038f9082...
```

2.3) Dataset.py Placeholders

The dataset.py code defines a PyTorch dataset for patching images of breast cancer biopsies. The dataset class takes as input several parameters such as the data directory, patch size, number of patches per image, whitespace threshold, and others, which are used to generate the patches.

The _make_coords() method is responsible for generating and storing the coordinates of the patches. It uses multiprocessing to speed up the process, and if the coordinates have already been generated before, it reads them from the coords.data file instead of calculating them again. The _fetch_coords() method is used by the multiprocessing function to extract the coordinates of patches from each SVS image. _load_file() method loads a single SVS image using the pyvips library, _patching() method extracts random patches from the image, _image_to_tensor() method converts the extracted patch into a PyTorch tensor, _filter_whitespace() method checks if the patch contains enough information, and finally _get_intersections() method checks if the patch overlaps with previously extracted patches or not.

The BreastDataset class uses the Dataset module from the PyTorch library and overrides the ___init___() method to initialize the class with the input parameters. It also overrides the ___len___() method to return the total number of patches, and the ___getitem___() method to return a single patch at a time.

Overall, this code generates patches of breast biopsy images to be used for training deep learning models to detect cancer in the images.

2.4) DataModule.py Placeholders

The datamodule.py code defines a DataModule for a PyTorch project. A DataModule in PyTorch Lightning is a way of organizing the code to handle loading, preprocessing and splitting of the data. The DataModule class is a subclass of pl.LightningDataModule.

The DataModule defined here loads the breast cancer histology image dataset, which consists of small image patches of size 32 x 32. The dataset is split into train, validation and test sets. During training, patches are randomly sampled from the train set, transformed and then passed to the model. The dataset normalization statistics (mean and standard deviation) are calculated from the train set and then used to normalize the image data in all datasets.

___init___: initializes the instance of the class with hyperparameters and other class attributes.

add_dataset_specific_args: adds dataset specific command-line arguments to the argument parser.

prepare_data: downloads the dataset and prepares it. This method is called only once and on a single GPU.

setup: This method is called on every GPU and it splits the dataset into train, validation and test sets.

train_dataloader: Returns the dataloader for training set.

val_dataloader: Returns the dataloader for validation set.

test_dataloader: Returns the dataloader for test set.

This DataModule class also inherits some other useful methods and attributes from the pl.LightningDataModule class such as to, cpu, cuda, num_workers, batch_size, etc.

2.5) Task.py Place holder

This is a PyTorch Lightning module for a Variational Autoencoder (VAE). It is a type of generative model that learns a latent representation of the input data and generates new data samples from the learned distribution.

The VAE class inherits from the LightningModule class and has the following methods:

add_model_specific_args: a static method that defines the model-specific arguments for the argparse module.

___init___: initializes the VAE model with the specified hyperparameters and the encoder and decoder architectures.

encode: a method that takes an input image and encodes it into a latent representation.

decode: a method that takes a latent representation and decodes it into an image.

forward: a method that combines encode and decode to perform the full VAE forward pass.

loss_function: a method that calculates the VAE loss function, which consists of a reconstruction loss and a Kullback-Leibler divergence term.

training_step: a method that runs a single training step for the VAE model.

validation_step: a method that runs a single validation step for the VAE model.

generate_images: a method that generates new images from random samples of the latent space.

from_pretrained: a class method that loads a pretrained VAE model from a checkpoint file. The checkpoints are stored on AWS S3 and can be accessed using the pretrained_urls dictionary.

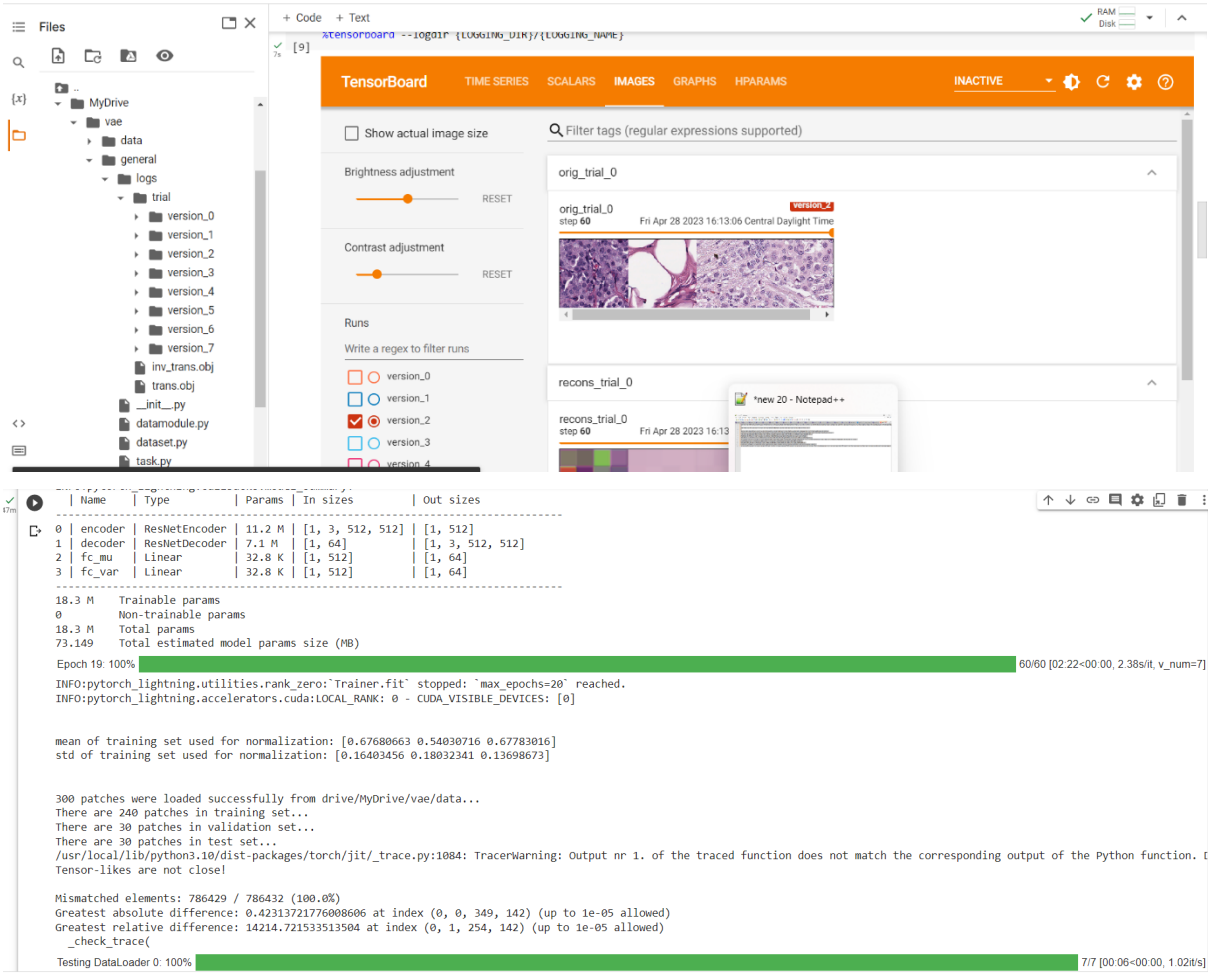
2.6) Train.py Script

Haven't changed script as per instructions, but used this function for training purposes.

2.7) main.ipynb

Implemented as per instructions, and used to call train.py and other files.

Included the screenshot, logs for reference.



3) Extra Credit

I ve tried to implement method on CNN. The implementation of a convolutional neural network (CNN) in Python using the Keras API of TensorFlow 2. The class provides methods to add layers such as convolutional layers, dense layers, and pooling layers. It also provides methods to set the loss function, metric, and optimizer, and to compile and train the model. Additionally, it provides methods to save and load the model, set and get the weights and biases of the layers, and remove the last layer.

The class implementation can be used to create and train a CNN model for various computer vision tasks such as image classification, object detection, and segmentation. The implementation provides a modular approach to add or

remove layers, set different hyperparameters, and save or load the model.

4) Difficulty Adjustment

In total assignment took more than 17 hours to complete, initially to understand the concept it took some time.

The implementation of datamodule.py section took time and implementation of extra credit took some more time.