Explanation of Code and Instructions for Use

./Data_Prep

This is the folder in which I prepared all of my training datasets.

- saveasnii.m

  Based on part of your code, it takes a .mat file containing a 4DMR and separates it into magnitude and velocity vectoral components and saves each of them as a separate .nii file.

- nii2img.py

  Modified from a script found online. This separates out all slices in a .nii file and saves them as .png files into a new folder.

- crop_hr.py

  Takes all images in uncropped_dir, crops them from the centre according to the chosen scale, and saves the cropped images into cropped_dir.

- hist_match.py

  The core functions of this code were found online. It takes all images in unmatched_dir, histogram matches them with ref_img and saves them in matched_dir.

- downsample.py

  This script takes all HR images in hr_img_dir and for every image in the directory it applies whichever smoothing function is not commented out, then downsamples the image based on the desired downsampling factor. It then saves all the images in the corresponding folder within lr_img_dir.

- downsample_simple_bicubic.py

  Does the same as downsample.py but without the smoothing step.

- 9to3layers.py

  One of the datasets had 9 greyscale layers per .png file rather than 3, which is an invalid input to the SR programs. Rather than change downsample.py and recreate all the images again, this script simply took all the images and turned them into 3 layer .png files.

- serialise_images.py

  This simply takes all files in a folder, sorts them numerically and renames them to the desired numbering convention. This was used for SRCNN but was not needed for VDSR and SRGAN.

./Final_Report_Images

- The HTML files are figures for the final report. Since Powerpoint blurs LR images, I made them in HTML and CSS instead.

- statscalculator.py

Since SRGAN doesn't output PSNR and SSIM during training, this script takes a HR image and an SR image as inputs and calculates these values between them. The core code of this script was found online.

./Dataset Downsampling

- generate_histograms.py
  Generates and outputs the three image histograms used in the final report.

./SRGAN

If you want to run this program, I can send across the virtual environment with all of the correct python module versions loaded, but it is quite large so I haven't included it here.

- config.py
  Sets all the parameters for the network train, and specifies the locations of various folders. All variables concerning training are saved in Config.TRAIN and all variables concerning testing are saved in config.VALID.

- model.py
  This script contains two functions to set up the generator and discriminator models. This is all core SRGAN code and I didn't change much in here.

- train.py
  This is the file that is run during training and testing.

  python train.py train.
  python train.py --mode=evaluate test

  train() and get_train_data() contain code related to the training phase, and evaluate() contains the code for testing.

- h5inspector.py
  This is a standalone script that I wrote to look inside an .h5 file by printing out the names of each of its layers.

./samples
This is the folder where the results of a test are placed.

./models
This is where the program saves the weights files during training in the form of .h5 files.
g.h5 are the generator network weights.
d.h5 are the discriminator network weights.

I have included the weights files for 200 epochs for both datasets.

vgg19.npy is a deep CNN used by SRGAN that has been pretrained on over 1 million images to recognise feature representations on a wide variety of images.

./VDSR

- main.m

  This file sets the main parameters and sets the program running. ***isTrain*** defines if the program should run the train() or test() function.
  isGreyscale defines if the input images are greyscale or colour, I kept the colour functionality just in case any velocity data was not greyscale in the future.
  isStandardDownsampling defines which type of downsampling operation to perform during training and testing, as VDSR creates the LR images as it runs. Setting this to *True* will result in simple bicubic downsampling of the training set. *False* includes histogram matching and 8x8 kernel bicubic smoothing.

  The pre-loaded weights file is also specified for testing, and the testing images path is specified also. The training image file path is specified in train.m.

- test.m

  This runs a test. You select a desired scale factor and the LR reference image. The testing dataset was already defined in main.m. It applies whichever type of downsampling was chosen, reupsamples the image bicubically and with VDSR, calculates PSNR and SSIM scores and generates outputs. This script can only calculate PSNR and SSIM for individual images, but to calculate scores across an entire dataset the superResolutionMetrics() function can be used.

- superResolutionMetrics.m

  This contains a function to calculate the mean PSNR and VDSR scores across the entire testing dataset and for multiple upsampling factors at the same time. This was not required for the project, but is a useful function to verify statistical significance over a large testing dataset, so I left it in.

- train.m

  This starts the train by defining the network parameters and calling createVDSRTrainingSet and vdsrLayer functions to set up the dataset and layers. It then starts the train and saves the weights into a .mat file once training has completed.

- createVDSRTrainingSet.m

  downsamples the images according to the chosen downsampling method, then bicubically upsamples it back to the ground truth dimensions. It then calculates the residual image and saves both of these into imageDatastore, which temporarily stores them for the duration of the train.

- vdsrLayers.m

  defines all of the layers of the VDSR.

- matRead.m and nii_test.m

These were both just short scripts that I wrote at the beginning of the project to look inside .mat and .nii files and output an image slice.