Introduction

Materials

Input data

Input data for bionoi consists of three protein files: a .mol2 file, a .profile file, and .out file from POPSlegacy. The bionoi program currently utilizes the Tripos Atom .mol2 file type, which contains information about the atom’s location, charge, and residue type and number. Information extracted from the .profile file allows bionoi to calculate the information entropy of the given residue. The final input file, .out file, comes from POPSlegacy, located at <https://github.com/Fraternalilab/POPSlegacy>. POPSlegacy requires the user to input a .pdb, which must match the .mol2 and .profile files for bionoi to function. While generating the .out files using POPSlegacy, the user must change the --atomsOut option to ‘on.’ The POPSlegacy output file is then used to calculate the accessible surface area for each atom.

Programs Used

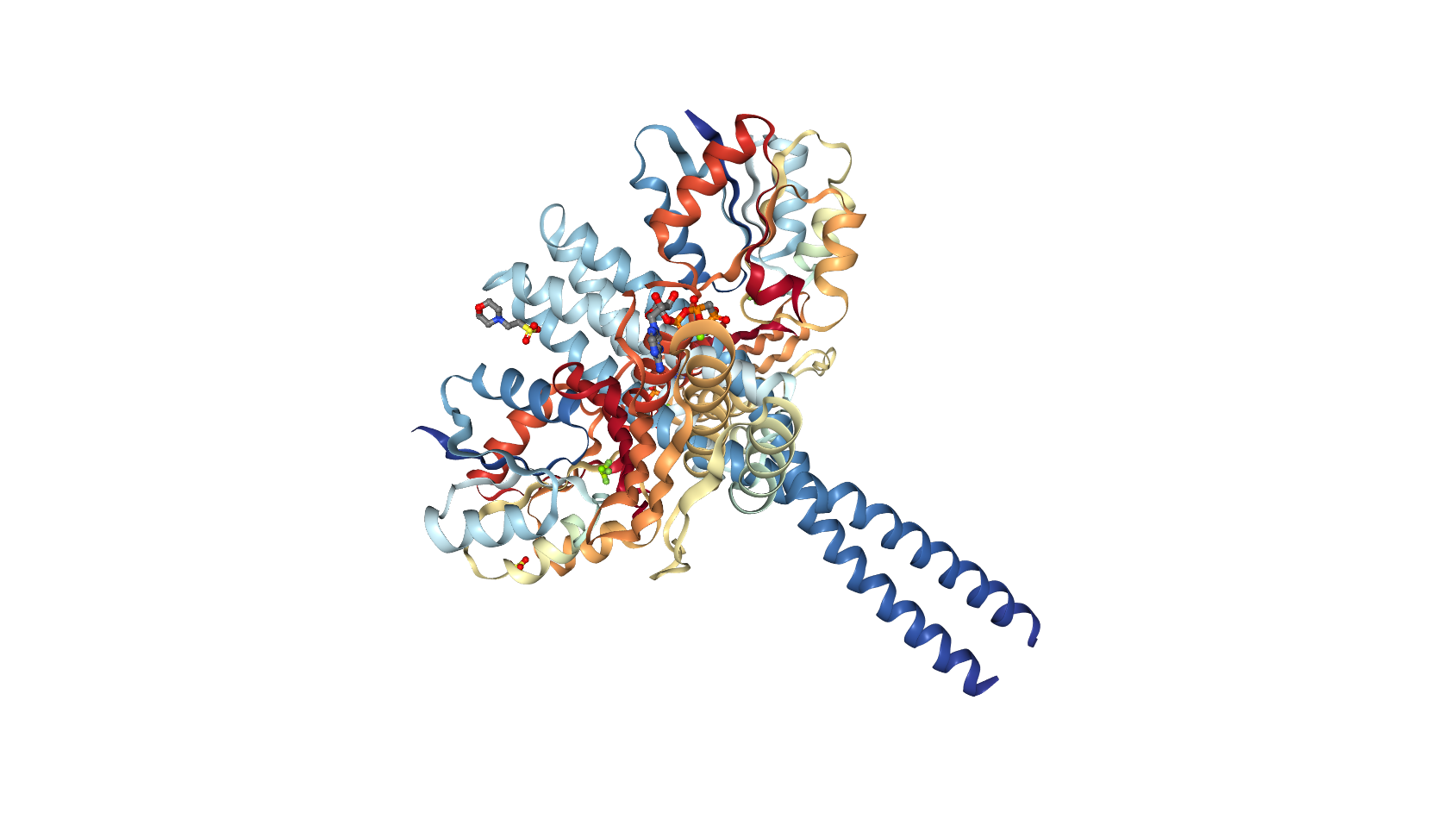
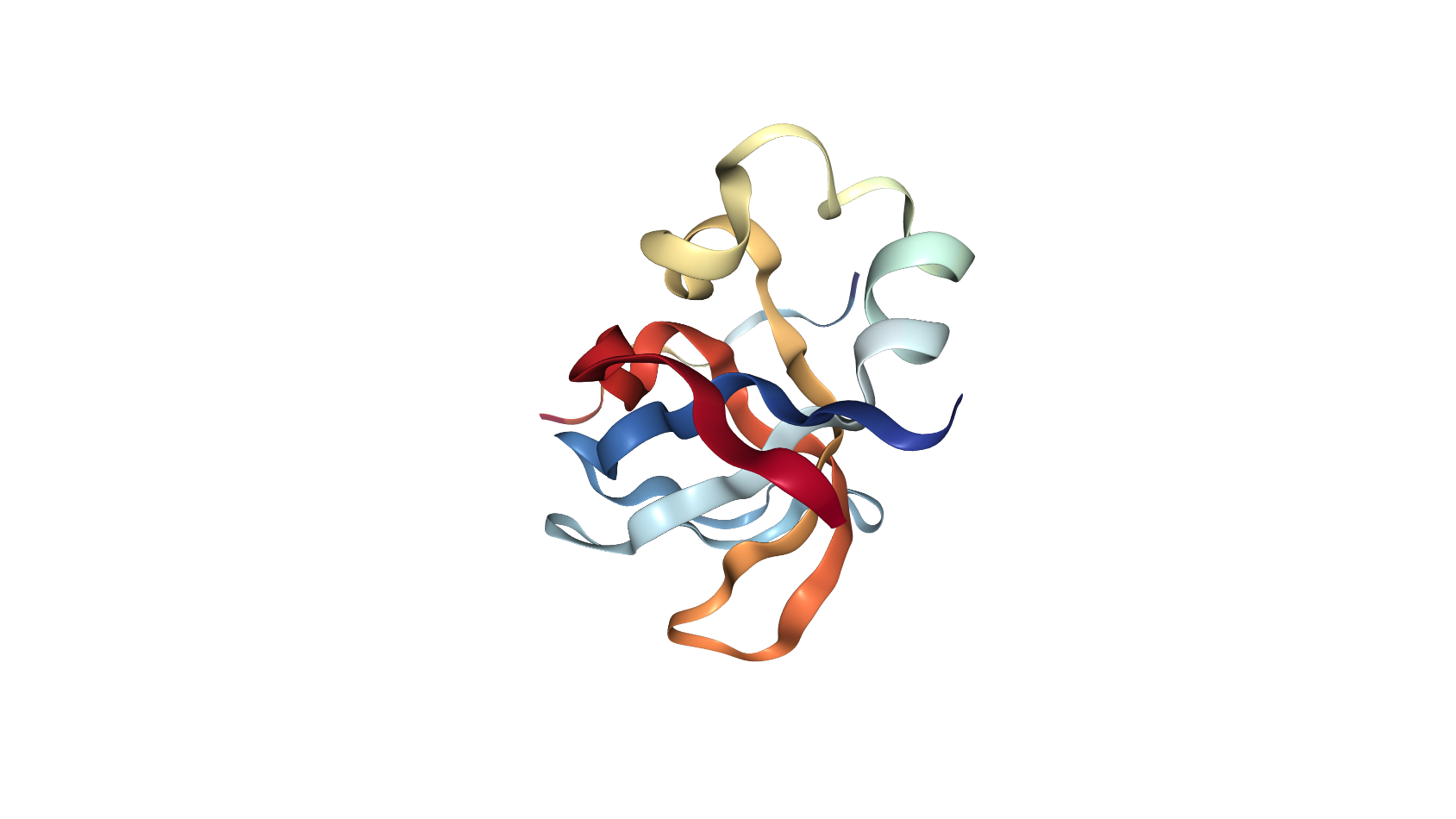
bionoi is written in Python 3 and requires the following packages: Numpy 1.14+ ([www.numpy.org](http://www.numpy.org)), scipy 0.18+ ([www.scipy.org](http://www.scipy.org)), Pandas 0.19+ ([pandas.pydata.org](http://pandas.pydata.org)), scikit-spatial 0.12+ ([pypi.org/project/scikit-spatial/](http://pypi.org/project/scikit-spatial/)), matplotlib 2.0.2+ ([matplotlib.org](http://matplotlib.org)), biopandas ([rasbt.github.io/biopandas/](http://rasbt.github.io/biopandas/)), and OpenCV ([opencv.org](http://opencv.org)).

Input Options

* mol - .mol2 file
* pop - .out file obtained from POPSlegacy
* profile - .profile file
* out - directory for output images
* dpi - output image quality in dots per inch
* size - output image size in pixels
* alpha - alpha for the color of the cells
* colorby - what property to color the voronoi cells by.
  + Options include:
    - atom\_type - color cells according to the atom
    - residue\_type - color cells according to the residue
    - charge - color cells according to the atom’s relative charge
    - binding\_prob - color cells according to the residue’s binding probability
    - hydrophobicity - color cells according to the residue’s hydrophobicity
    - center\_dist - color the cells based on the atom’s distance to the center of the pocket
    - sasa - color the cells based on the solvent accessible surface area
    - seq\_entropy - color the cells based on the residue’s sequence entropy
    - properties - color the cells using a blend of charge, binding\_prob, hydrophobicity, center\_dist, sasa, and seq\_entropy
* imageType - the format to save the output images as
* direction - the direction of the 2D projection
* rot\_angle - the angle of rotation
* flip - the type of flipping
* save\_fig - whether to save the original figure or not

Case Study

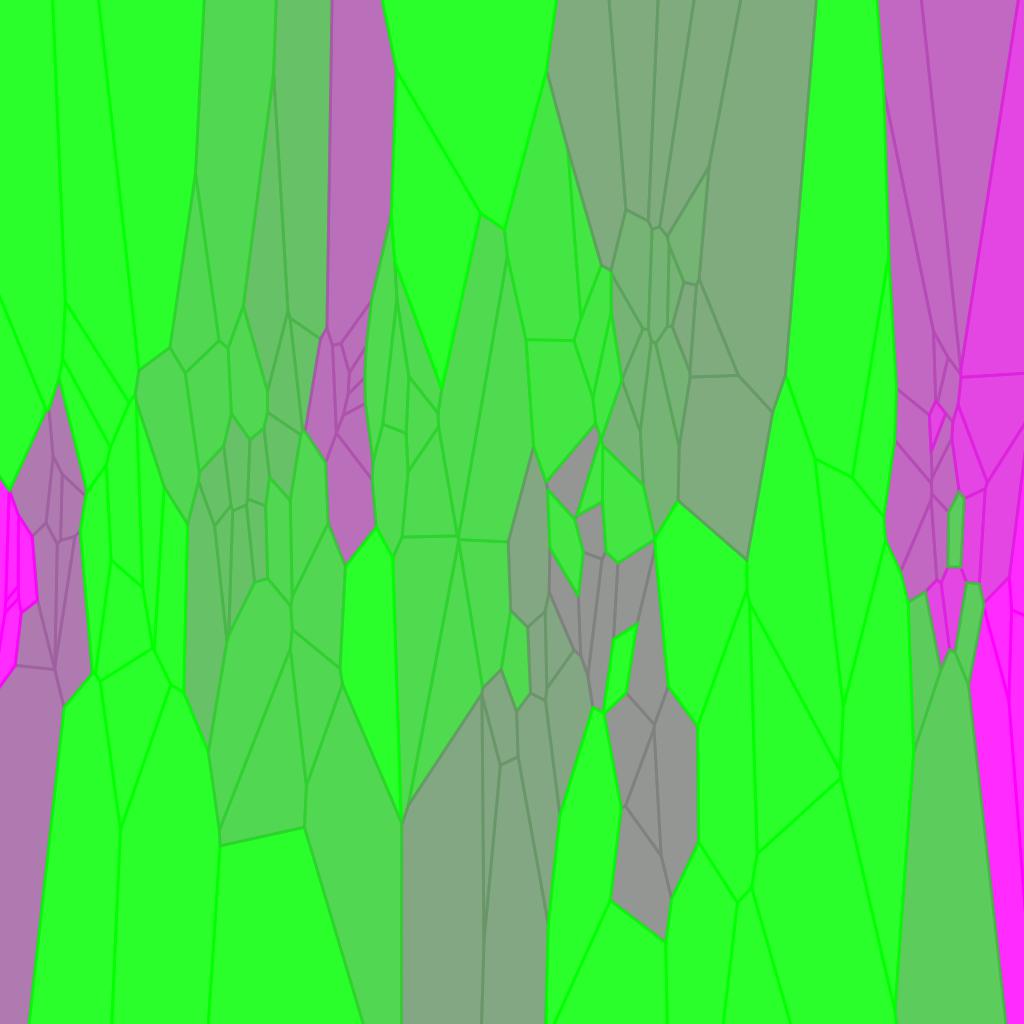
We randomly selected two pockets to demonstrate bionoi’s modeling capabilities: phosphatidylinositol 3-kinase regulatory subunit alpha (PDB-ID: 3i5rA) and

sensor histidine kinase DesK chain E (PDB-ID: 5iunE). 

The command that we will use to create a two dimensional model of 3i5rA is

“python main.py -mol ./mol2/3i5rA00.mol2 -profile ./profile/3i5rA.profile -pop ./pop/3i5rA.out -out ./images/ -dpi 512 -size 512 -alpha 0.8 -colorby seq\_entropy -direction 1 -rot\_angle 2 -flip 2”.

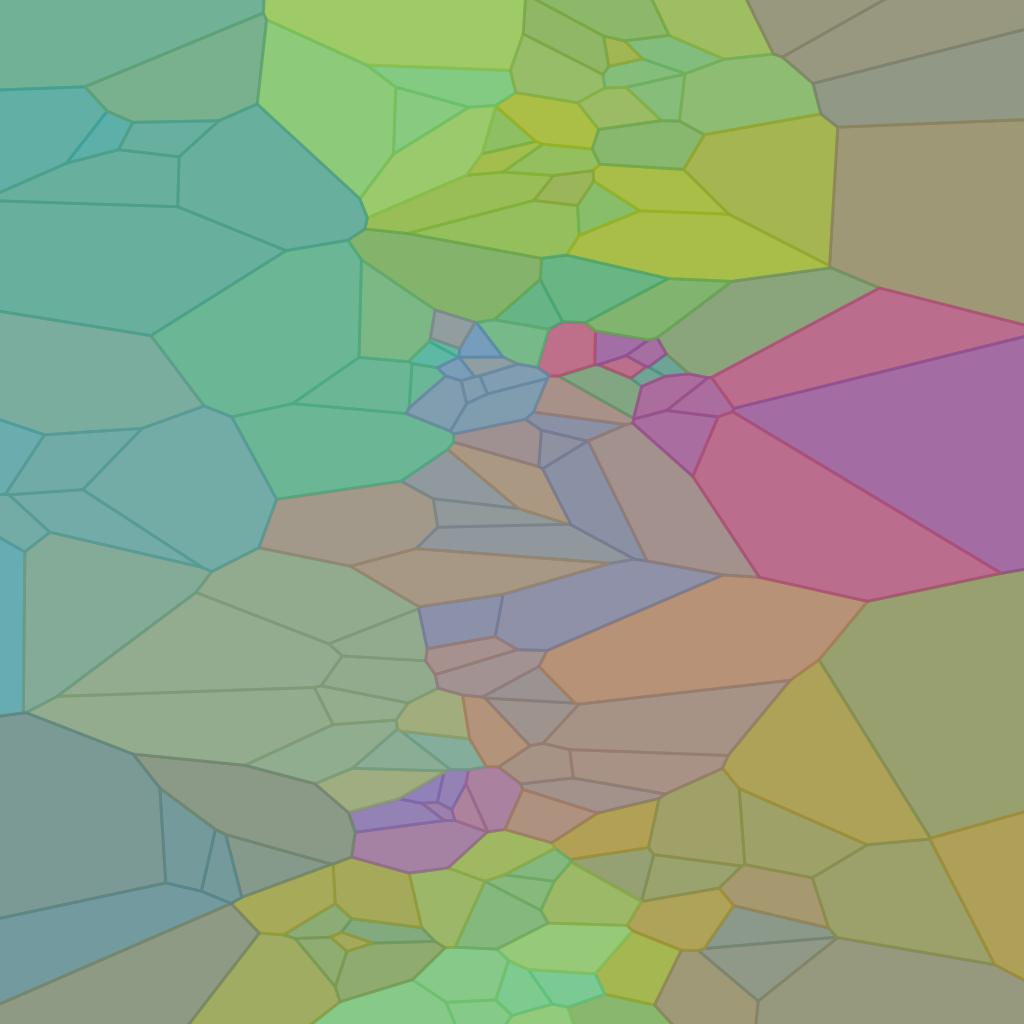
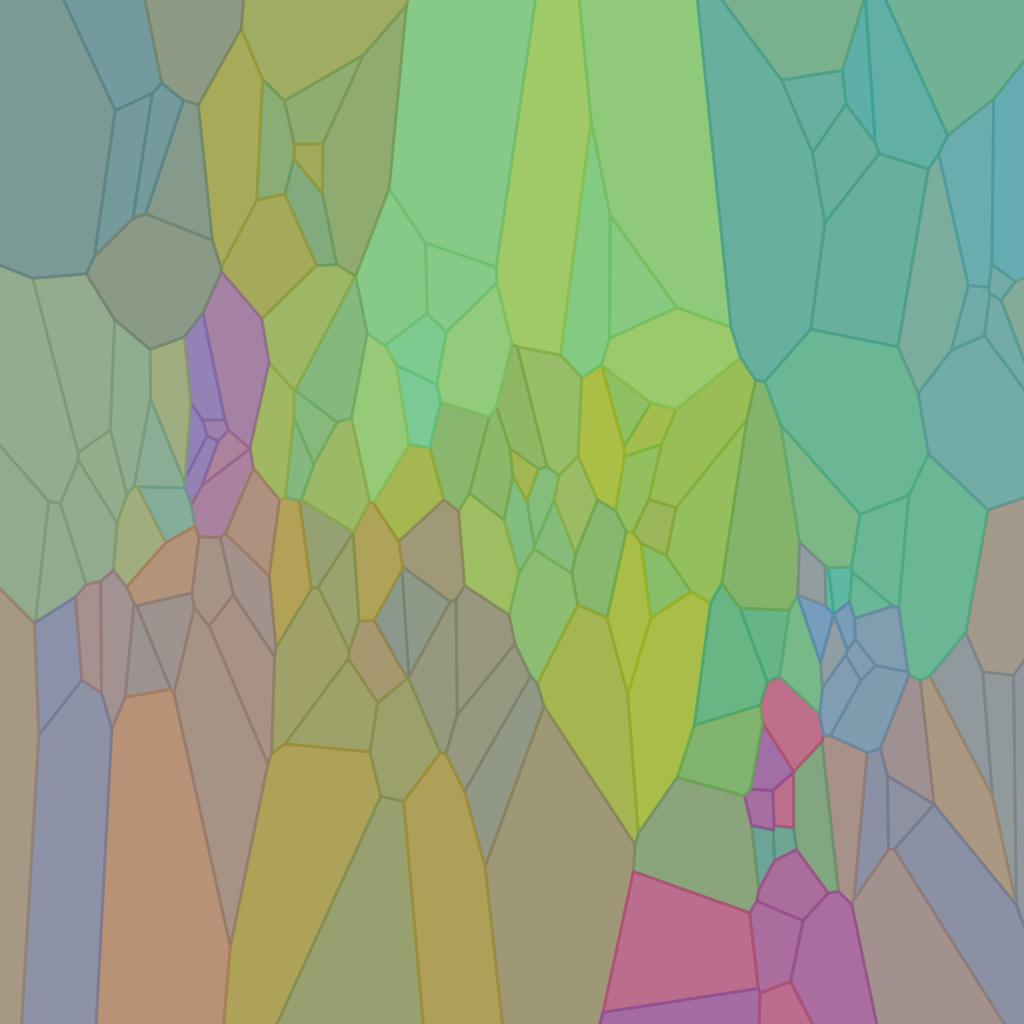
This command will create a Voronoi map of 3i5rA with dpi and size 512. The map will be colored using a linear color scale according to the normalized values of the residues’ sequence entropies. The final three commands deal with the various transformations to perform on the original image. -direction 1 will project the three dimensional model of the protein onto the positive xy plane. -rot\_angle 2 will rotate the image 90° clockwise. -flip 2 will flip the image along its x-axis.



To create a voronoi map of 5iunE that blends six different properties into one image, the command is

“python main.py -mol ./mol2/5iunE00.mol2 -profile ./profile/5iunE.profile -pop ./pop/5iunE.out -out ./images/ -dpi 512 -size 512 -alpha 0.8 -colorby properties -direction 0 -rot\_angle 0 -flip 0”.

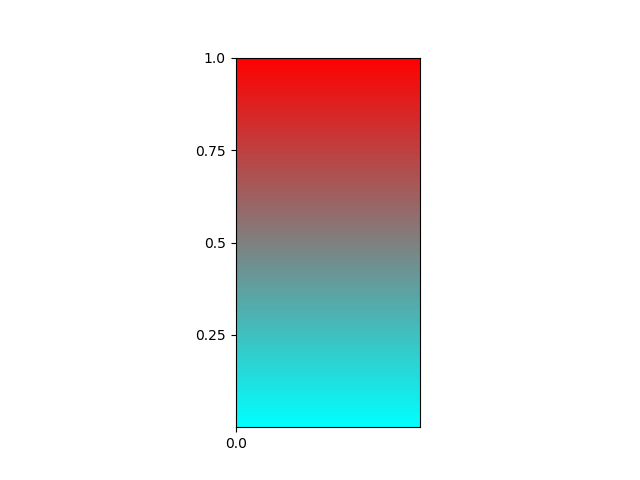
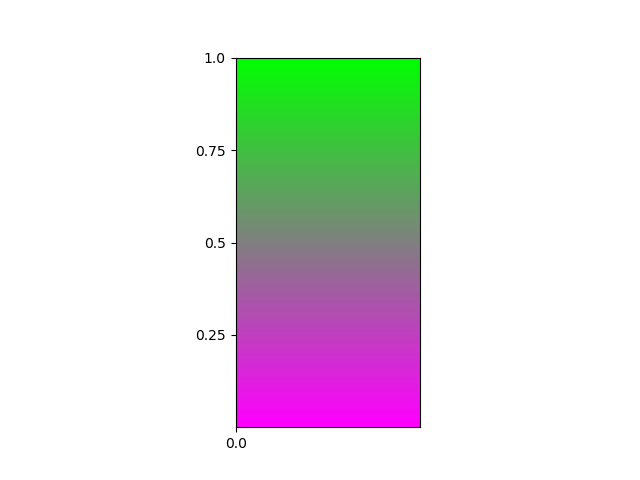
This command will create 72 different images of the 5iunE pocket. Because -direction is 0, bionoi will create six different images that each use a different plane to project the 3D model onto. -rot\_angle 0 will take each of the projection images and proceed to rotate each one 0°, 90°, 180°, and 270°. Finally, -flip 0 flips every one of the rotated images over the x-axis and the y-axis while saving the original. In total, 72 images are created of the same pocket. The -colorby option “properties” tells bionoi to create a set of images using charge, binding probability, distance to the center of the pocket, solvent accessible surface area, hydrophobicity, and binding probability. bionoi then blends the corresponding images from each set to create a unique image.

The two above images are of the same 5iunE pocket, but have had different transformations applied to them. The image on the right was originally projected onto the negative xy plane, rotated 0°, and flipped along its y-axis. The image on the left was projected onto the negative zx plane, rotated 270°, and was not flipped.

When bionoi is given a command, such as the ones above, bionoi first determines what dataset it needs to use to map the colors and the color scale that will be used. If colorby is specified as charge, seq\_entropy, center\_dist, or sasa, bionoi must extract and calculate the data. Charge is just extracted from the .mol2 file and then normalized. Solvent accessible surface area is extracted from the .out file generated by POPSlegacy. The distance to the center is calculated using an atom’s coordinates after it has been aligned to the principal axes in order to make the calculations easier. Sequence entropy data is extracted from the .profile file and computed by taking the negative logarithm of the probability mass function for each variable.

bionoi creates a linear color map using opposite colors on the RGB color wheel in order to best display the data. In the figure 1, for example, the sequence entropy data is mapped to the green magenta color map. The higher the normalized values appear more green while the lower normalized values appear more magenta.



Once the dataset has been mapped to a color map, bionoi constructs a single voronoi diagram. The location information taken from the .mol2 file is first used to align the pocket to the principal axes. The miller projection is then computed, flattening the coordinates along the specified plane. Finally, the atoms and their corresponding colors are matched to the polygons in the voronoi diagram, coloring the diagrams.

Autoencoder

Input data

Input data for the bionoi autoencoder consists of a directory of training images. The images may be in subdirectories up to two layers deep. If using the reconstruction script, input data consists of a directory of training images and a trained model.

Programs Used

The bionoi autoencoder is written in Python 3 and requires the following packages: Pytorch with Cuda 1.1.0+ ([pytorch.org](http://pytorch.org)), Numpy 1.14+ ([www.numpy.org](http://www.numpy.org)), and matplotlib 2.0.2+ ([matplotlib.org](http://matplotlib.org)).

Input Options

* Autoencoder
  + epoch - the number of epochs to train
  + feature\_size - size of output feature of the autoencoder
  + data\_dir - directory of training images
  + model\_file - file to save the trained model
  + batch\_size - the batch size
  + normalize - whether to normalize the dataset
  + style - the style of autoencoder to use
    - Options include:
      * conv - Convolutional style autoencoder. Upsampling is implemented in the ConvTranspose2d layers.
      * dense - A vanilla-style autoencoder, which takes 2D images as input, then the flattened vector is sent to the autoencoder, which is composed of multiple dense layers.
      * conv\_dense\_out - Convolutional style autoencoder. Upsampling is implemented in the ConvTranspose2d layers. A dense layer followed by relu is added to the end of encoder and beginning of the decoder to force the output features to be compact and sparse.
      * conv\_1x1 - Convolutional style autoencoder. Upsampling is implemented in the ConvTranspose2d layers. 1x1 convolution layers are used in last layer of encoder and first layer of decoder.
      * conv\_deeper - Convolutional style autoencoder. Upsampling is implemented in the ConvTranspose2d layers.
* Reconstruction
  + data\_dir - directory of training images
  + style - style of autoencoder used to create the trained model. Options are the same as above
  + feature\_size - size of output feature of autoencoder
  + normalize - whether to normalize the dataset
  + model - the trained model
  + gpu\_to\_cpu - if the model was trained using a gpu, reconstruct.py will convert the model to be usable on a cpu only machine
  + img\_count - how many images to reconstruct
* Feature Vector Generation
  + data\_dir - directory of training images
  + style - style of autoencoder used to create the trained model. Options are the same as above
  + feature\_dir - size of output feature of autoencoder
  + normalize - whether to normalize the dataset
  + model - the trained model
  + gpu\_to\_cpu - if the model was trained using a gpu, reconstruct.py will convert the model to be usable on a cpu only machine

Case Study

Autoencoder

python autoencoder\_general.py -data\_dir /var/scratch/jfeins1/bae-images-4M/ -normalize True -model\_file ./log/conv1x1-4M-batch512.pt -style conv\_1x1 -batch\_size 512

The above command will train an autoencoder on a directory of bionoi images. The autoencoder will automatically determine if the system has GPUs available and will parallelize the task if multiple GPUs are present. The user must remember whether or not training used GPUs because it will be important during reconstruction and feature vector generation. The images in the data directory may be up to three layers deep. Because autoencoders learn in unsupervised manners, the images do not have to be organized according to their classes as they would with a classification task. The autoencoder is only learning the most salient information of the entire dataset, not the specific properties of each class.

There are multiple models that the user can choose to use. When training on a bionoi dataset, we found that the convolutional style autoencoder with 1x1 convolutions in the last layer of the encoder and first layer of the decoder was able to learn best a bionoi dataset’s features.

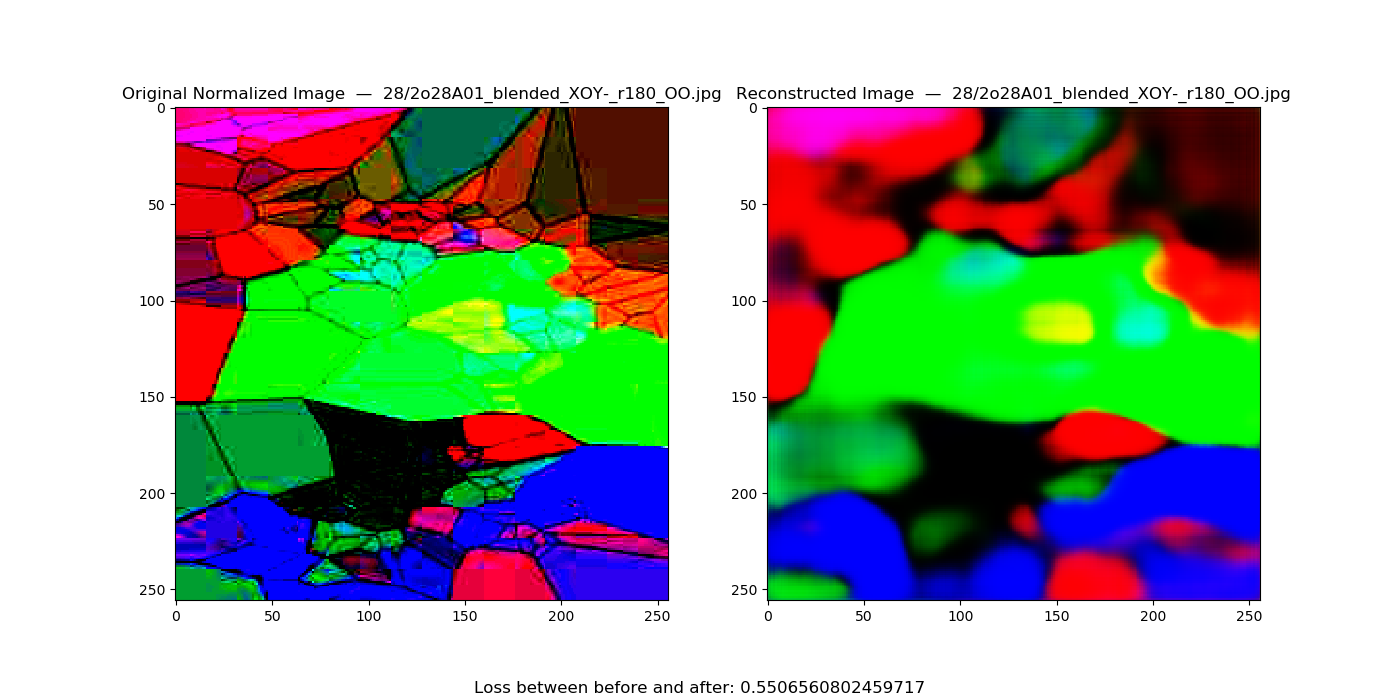
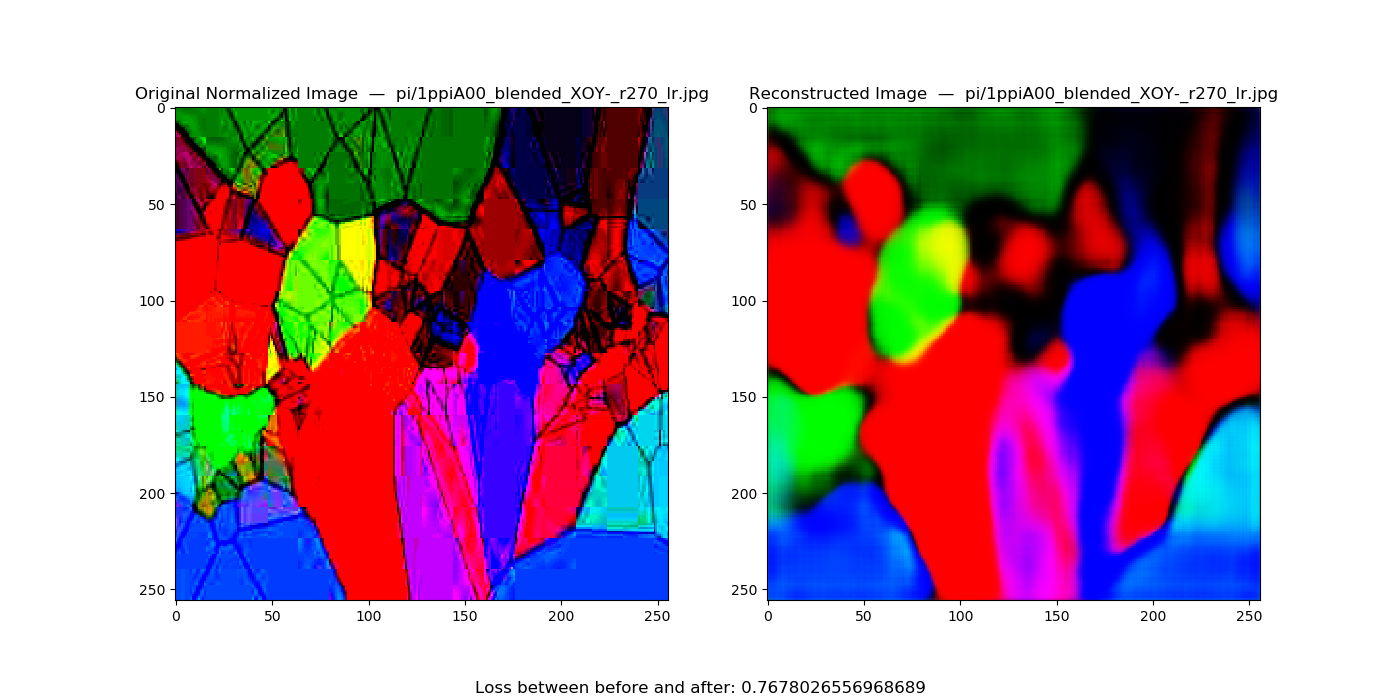
Note that if the user decides to normalize during training, they must then normalize in every subsequent step when the model is used. If the model was trained on normalized images, it may have trouble identifying unnormalized images. The loss will be smaller without normalization because the variance of the dataset will be lower than the normalized dataset. The absolute values of the loss are not as important as the loss’s relative decrease during training. Additionally, normalizing input data usually allows the model to generate better gradients and therefore converge faster and at better locations.

The -model\_file parameter tells the autoencoder where to save and what to name the trained model. -epoch determines how many epochs to train the autoencoder. The default is set at 30, but the user may want to experiment with this number to determine how long the model can train for before additional training becomes impractical. -batch\_size informs the model about the size of each training batch. It is recommended that the batch size be as large as possible because training speed increases linearly as the batch size increases. If the user specifies a batch size that requires more memory than the system has, an error will be returned saying that Cuda:0 ran out of memory. Once training is complete, the trained model and a graph of the loss function will be saved in the -model\_file location.

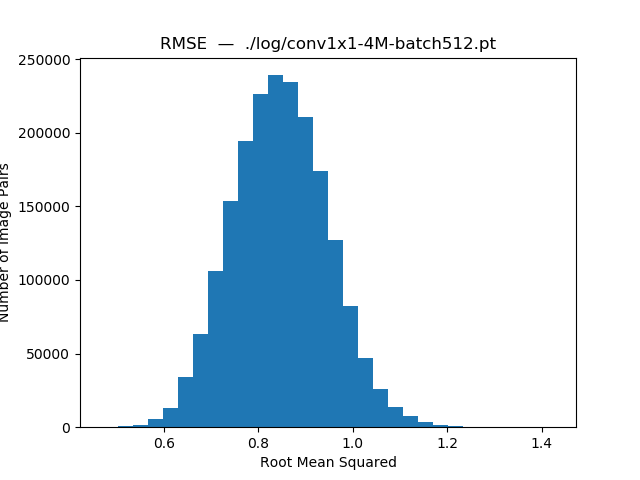
Reconstruction

python reconstrut.py -data\_dir /var/scratch/jfeins1/bionoi\_reconstruct\_images/ -normalize True -model ./log/conv1x1-4M-batch512.pt -style conv\_1x1 -gpu\_to\_cpu False -img\_count 15

reconstruct.py takes a trained model and reconstructs a specified number of images and then compares them to the original images, allowing the user to visually confirm the model’s accuracy. The above command will reconstruct 15 images from the directory /var/scratch/jfeins1/bionoi\_reconstruct\_images/ and will create a histogram of the root mean square error (RMSE) between all of the reconstructed images and original images in the dataset.

Just as with the autoencoder, -data\_dir may contain images up to three layers deep. If the user trained their model with normalized values, then they should normalize the dataset once again. -model should be the model file that the user trained earlier. The user should choose the same -style as done during training. If the user is reconstructing the images on a system that uses the ‘cpu’ but trained their model on a system that had GPU support, -gpu\_to\_cpu should be True. When True, gpu\_to\_cpu remaps the model to the current device and removes certain elements from the state\_dict that are incompatible with the CPU. If the device does not change between training and reconstruction, -gpu\_to\_cpu should be False. -img\_count determines how many images to reconstruct and plot. The output of the above command is shown below.

With -img\_count set to 15, reconstruct.py will create 15 of the above plots. The program randomly chooses images from the dataset to reconstruct and then plots them. The loss printed at the bottom of the image is the root mean square error between the original and reconstructed image.



The program will also create a histogram that plots the RMSE between all of the images in the dataset and their corresponding reconstructed images. The histogram allows the user to see the loss distribution. If the reconstructed images appear similar and the histogram shows a fairly small distribution of losses, the model learned the most important features successfully.

Feature Vector Generation

python feature\_gen.py -data\_dir /var/scratch/jfeins1/bionoi\_reconstruct\_images/ -feature\_dir /work/jfeins1/features -normalize True -model ./log/conv1x1-4M-batch512.pt -style conv\_1x1 -gpu\_to\_cpu False

feature\_gen.py will output a 512x1 feature vector for each of the images in the dataset as a .pickle file. The parameters for feature\_gen.py are the same as reconstruct.py with the exception of -img\_count and -feature\_dir. -feature\_dir is the directory to save all of the feature vectors in.

[[ 5.80632091e-01 4.71326023e-01 2.74976641e-01 -1.93283692e-01

-2.56383955e-01 -1.53253362e-01 -9.89855081e-02 2.31658027e-01

3.89869362e-01 2.80525327e-01 -2.81491548e-01 -5.13162136e-01

…

-1.36695683e-01 -3.34159657e-02 -9.25423279e-02 -1.28462255e-01

1.28456475e-02 6.49406761e-02 1.32325813e-01 1.14365943e-01

2.44563203e-02 2.38637184e-03 -9.04247444e-03 3.67040820e-02]]

Above is an example of a feature vector. The full vector consists of 512 features.