Jan 19th, 2025

Analysis on Data preprocessing and algorithm description

Main Directory Structure

```
├── BRAF and Raf Data - Summer 2024/
│   ├── BRAF images/
│   └── raf images/
└── CTNNB1 and arm/
    ├── CTNNB1/
    └── ARM/
```

The eyes information is embedded within the BRAF and Raf folders.
A folder contains types, a type 3 vials. Each vial contains .czi and .ims files.

Information domain and data characterization

The dimensionality of an image taken from a .czi file is variable in the number of stack images, here it is presented as 16. The rest of the information was consistent

```
(1, 1, 1, 1, 16, 1024, 1024, 1)
n = 8
```

The libraries required to process .czi files are czifile, Pillow and numpy. I'm using the pytorch-cuda environment in my machine. The program produced by Claude to process .czi files reduces the dimensionality to

```
(16, 1024, 1024)
```

I noticed many of the folders and files are named with spaces in them. A script *batch_czi_mappting.py* clones the directory structure while correcting for spaces, by changing " " for "_"

While the first version of the algorithm for *map_czi.py* gave reasonable results, there was a lot of information being discarded, producing a suboptimal representation on the information contained on each file, keeping 1 image and discarding 15.

Moving forward with the assumption that each of the 16 layers represent different levels of depth in a single sample, a second version of the algorithm applies a normal distribution with standard deviation values of 2 - 4

(where higher values of sigma produce more equally balanced contribution of each layer to the final composite image).

PCA and image decomposition

Problem statement

I am working in a new algorithm that has the goal of taking a dataset of images, decomposing them into a latent space representation of the features that characterize the distribution of images, and use it to provide a continuous qualifier for phenotypic expression in the ommatidia and wings of Drosophila. Based on supervisor's suggestion, the developed code starts from a tutorial in scikit-learn called Faces Dataset Decomposition:

```
https://scikit-
learn.org/stable/auto_examples/decomposition/plot_faces_decomposition.html
```

Algorithm

Data preprocessing: as stated in Information domain and characterization. Dataset structure as follows:

```
├── Dataset_eyes
    ├── BRAF_images/
    |   └ Vial_#
    |       └ Line_<A-Z>
    |           └ .png files
    └── raf_images/
    |   └ Vial_#
    |       └ Line_<A-Z>
    |           └ .png files
```

The EYES dataset is composed of

```
1011 images
```

Algorithm:

- Apply PCA to extract the features in the dataset
- Apply decomposition and generate a latent space representation of the dataset
- Use two images, located in opposite margins of phenotypic expression, to obtain a specific vector from the latent space

The visualization of the dataset revealed the need for normalization. Values were divided by 255, and 0.5 was taken from the value to ensure a mean of 0.

Decomposition

The tutorial "Faces dataset decomposition" includes many subtypes of similar methodologies:

- PCA using randomized SVD
- Non-negative components - NMF
- Independent components - FastICA
- Sparse components - MinibatchSparsePCA
- Dictionary Learning
- Cluster centers - MiniBatchKMeans
- Factor Analysis components - FA

After consultation with the LLM Claude Sonnet 3.5 on Jan 19th, I got feedback on the use of NMF as a first approximation to solve the problem because:

The additive nature makes it easier to interpret components
It often naturally separates facial features
The non-negativity constraint matches the nature of image data
It's generally more stable than ICA while being more interpretable than PCA

Jan 20th, 2025

I erased one of the subfolders from the Dataset_eyes for testing purposes, as testing with the whole dataset of 1011 images took too long to run (ca. 8 minutes).

In response to this performace issue, the dimensions of each image was reduced to 128x128 matrices, and the algorithm for merging the multistack was also updated by adding an *enhance_slice* parameter to the function *weighted_stack_combine*, which calls upon a helper function *enhance_slice*. The helper function incorporates the following methods:

Adaptive Histogram Equalization (AHE):

The function applies exposure.equalize_adapthist from the skimage (scikit-image) library to enhance the local contrast of the image. This method adapts the contrast enhancement to different regions of the image by dividing it into small tiles and applying histogram equalization separately in each region. The parameters used: kernel_size=None: Lets the function automatically determine the optimal tile size. clip_limit=0.01: Prevents excessive contrast enhancement, which could amplify noise.

Intensity Rescaling:

The function calculates the 2nd and 98th percentiles of the pixel intensity values to identify robust intensity limits (excluding extreme values). exposure.rescale_intensity is used to rescale the image's pixel values within the computed percentile range, effectively stretching contrast and improving visibility.

ChatGPT 4o. Jan 20th 2025

Turns out that each multistack has a different number of slices, ranging from ~14 to ~20. The proram automatically handles the calculation of weights based on the number of slices of each independent image. However, I increased the sigma parameter in the helper function *create_gaussian_weights* to 32 to ensure a uniform contribution across images (i.e. marginal slices add as much to the final pixel value as the middle ones)