Image Analysis - Part1

Overview

- Take a batch of images as input.
- Run the operations on each image
  - Grey Scaling Salt and Pepper Noise, Gaussian Noise, Linear Filtering, Median Filtering, Histogram Calculations, Histogram Equalization, Image Quantization, and Mean Squared Error).
- All of these operations output a new image for each image in the batch.
- A TOML file for configuration of attributes like noise and the weights for filters.

Usage

```
git clone https://github.com/praveenarallabandi/ImageAnalysis.git
cd ImageAnalysis
pip3 install --user pipenv
python ImageAnalysisPart1.py
```

Implementation

The project implementation is done using Python. Using Python, we can rapidly develop and integrate each operation. Python's NumPy library, which allows for array operations.

Certain image array operations are time-consuming, and those scenarios were addressed with optimizing NumPy arrays (using NumPy methods as much as possible) and with numba. Numba is an open-source JIT compiler that translates a subset of Python and NumPy code into fast machine code. Numba has a python function decorator for just-in-time compiling functions to machine code before executing. Using this decorator on functions that use heavy math and looping (i.e., filters and noise) provides significant speed increases with speeds similar to using lower-level compiled languages like C/C++ or Rust. For plotting histograms, Python's `matplotlib,` the relatively standard and robust plotting library, outputs plots to a file with the rest of the exported output images.

Dependencies

- numpy - For Array operations
- matplotlib - Plot
- numba - JIT for speed exectuion
- toml - Configuration settings
- PIL (Image) - Used only for importing and exporting images

Functions

```
def calculateHistogram(image: np.array) -> np.array:
```

`calculate_histogram` Generates the histogram, equalized histogram, and quantized image based on the equalized histogram

```
def image_quantization_mse(image: np.array, imageQuant: np.array,
imageName: str) -> float:
```

`image_quantization_mse` Calculates mean square error for two input images

```
def convertToSingleColorSpectrum(image: np.array, colorSpectrum: str) ->
np.array:
```

`convertToSingleColorSpectrum` Generates the NumPy array for a single color spectrum.

```
def corruptImageGaussian(image: np.array, strength: int) ->  np.array:
```

`corruptImageGaussian` Generates image with gaussian noise applied

```
def corruptImageSaltAndPepper(image: np.array, strength: int) -> np.array:
```

`corruptImageSaltAndPepper` Generates image with salt and pepper noise applied

```
def linearFilter(image, maskSize=9, weights = List[List[int]]) ->
np.array:
```

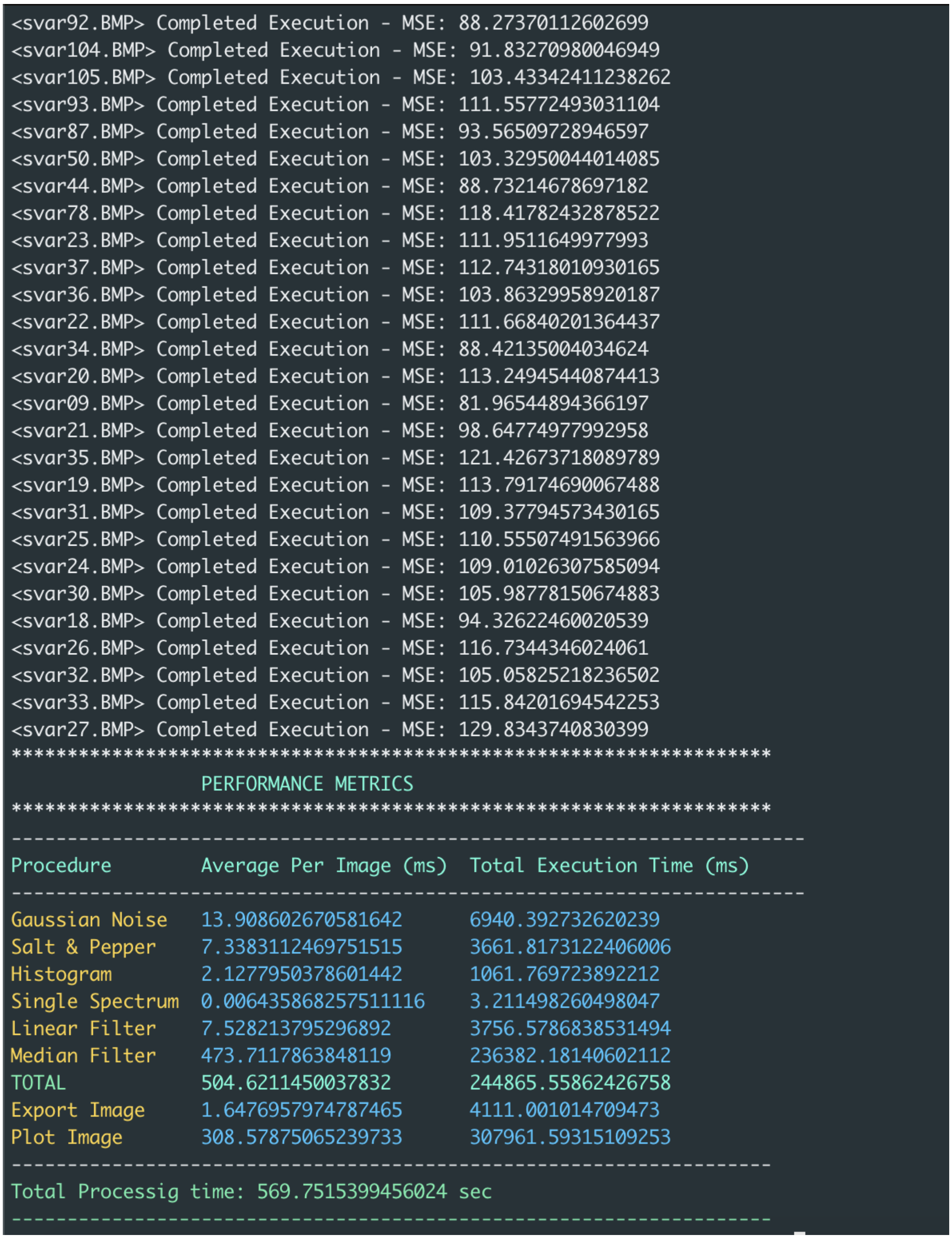
`linearFilter` Receives a kernel or matrix of weights as a two-dimensional input list and applies that kernel to a copy of an image. The filter is then applied in loops through each pixel in the image and multiples the neighboring pixels' values by the kernel weights. The larger the kernel, the larger the pixel's neighborhood that affects the pixel.

```
def medianFilter(image, maskSize=9, weights = List[List[int]]):
```

`medianFilter` The median filter is applied to the input image, and each pixel is replaced with the median value of its neighbors. The current pixel value as well is included in the median calculation.

Results

The output images are stored in output directory, mean square errors for each image is printed on stdout alsong with performance metrics Below is th snapshop of output



```
<svar92.BMP> Completed Execution - MSE: 88.27370112602699
<svar104.BMP> Completed Execution - MSE: 91.83270980046949
<svar105.BMP> Completed Execution - MSE: 103.43342411238262
<svar93.BMP> Completed Execution - MSE: 111.55772493031104
<svar87.BMP> Completed Execution - MSE: 93.56509728946597
<svar50.BMP> Completed Execution - MSE: 103.32950044014085
<svar44.BMP> Completed Execution - MSE: 88.73214678697182
<svar78.BMP> Completed Execution - MSE: 118.41782432878522
<svar23.BMP> Completed Execution - MSE: 111.9511649977993
<svar37.BMP> Completed Execution - MSE: 112.74318010930165
<svar36.BMP> Completed Execution - MSE: 103.86329958920187
<svar22.BMP> Completed Execution - MSE: 111.66840201364437
<svar34.BMP> Completed Execution - MSE: 88.42135004034624
<svar20.BMP> Completed Execution - MSE: 113.24945440874413
<svar09.BMP> Completed Execution - MSE: 81.96544894366197
<svar21.BMP> Completed Execution - MSE: 98.64774977992958
<svar35.BMP> Completed Execution - MSE: 121.42673718089789
<svar19.BMP> Completed Execution - MSE: 113.79174690067488
<svar31.BMP> Completed Execution - MSE: 109.37794573430165
<svar25.BMP> Completed Execution - MSE: 110.55507491563966
<svar24.BMP> Completed Execution - MSE: 109.01026307585094
<svar30.BMP> Completed Execution - MSE: 105.98778150674883
<svar18.BMP> Completed Execution - MSE: 94.32622460020539
<svar26.BMP> Completed Execution - MSE: 116.7344346024061
<svar32.BMP> Completed Execution - MSE: 105.05825218236502
<svar33.BMP> Completed Execution - MSE: 115.84201694542253
<svar27.BMP> Completed Execution - MSE: 129.8343740830399
********************************************************************
                 PERFORMANCE METRICS
********************************************************************
-----------------------------------------------------------------------
Procedure        Average Per Image (ms)  Total Execution Time (ms)
-----------------------------------------------------------------------
Gaussian Noise   13.908602670581642      6940.392732620239
Salt & Pepper    7.3383112469751515      3661.8173122406006
Histogram        2.1277950378601442      1061.769723892212
Single Spectrum  0.006435868257511116    3.211498260498047
Linear Filter    7.528213795296892       3756.5786838531494
Median Filter    473.7117863848119       236382.18140602112
TOTAL            504.6211450037832       244865.55862426758
Export Image     1.6476957974787465      4111.001014709473
Plot Image       308.57875065239733      307961.59315109253
--------------------------------------------------------------------
Total Processig time: 569.7515399456024 sec
--------------------------------------------------------------------
```