Code

Functions in my code:-

The code is made up of a number of functions that can handle different image processing jobs. The first section of the code reads a directory's worth of images and puts them in a list along with the labels that go with them. The different image processing functions described later in the code will take these photos as input.

The **compute histogram function** determines the histogram of an input picture. The corresponding histogram bin grows as the function loops through each pixel in the image. To represent the histogram, the computer first creates a 256-element array of zeros. The resulting histogram is returned as an array.

Example output:-

```
Histogram for image 1: [    0  1009     0    31     0    11     0     2     0     1     0     2
     0     0     0     4     0    10     0    21     0    29     0    41
     0    40     0    26     0    37     0    34     0    29     0    14
     0     5     0     2     0     3     0     1     0     2     0     9
     0     8     0    15     0    30     0    35     0    43     0    42
     0    39     0    24     0    17     0    11     0     3     0     3
     0     1     0     4     0     2     0     3     0    10     0    22
     0    37     1    47     1    39     6    30    14    34    40    22
    71     7   126    13   157     3   190     0   210     2   206     7
   227    21   284    30   332    66   356    87   370   158   459   184
   531   234   618   242   719   262   671   313   736   315   790   320
   759   353   721   454   717   546   648   702   616   927   547  1118
   493  1668   527  2376   478  3750   484  6220   477 10599   474 17884
   507 28406   486 42451   490 55613   368 63224   290 60803   163 50232
    73 34394    31 19395    10  8939    11  3544     4  1133     1   414
     0   199     0   132     0    68     0    24     0    11     0     6
     0     1     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0]
```

Average histogram of each class example:-

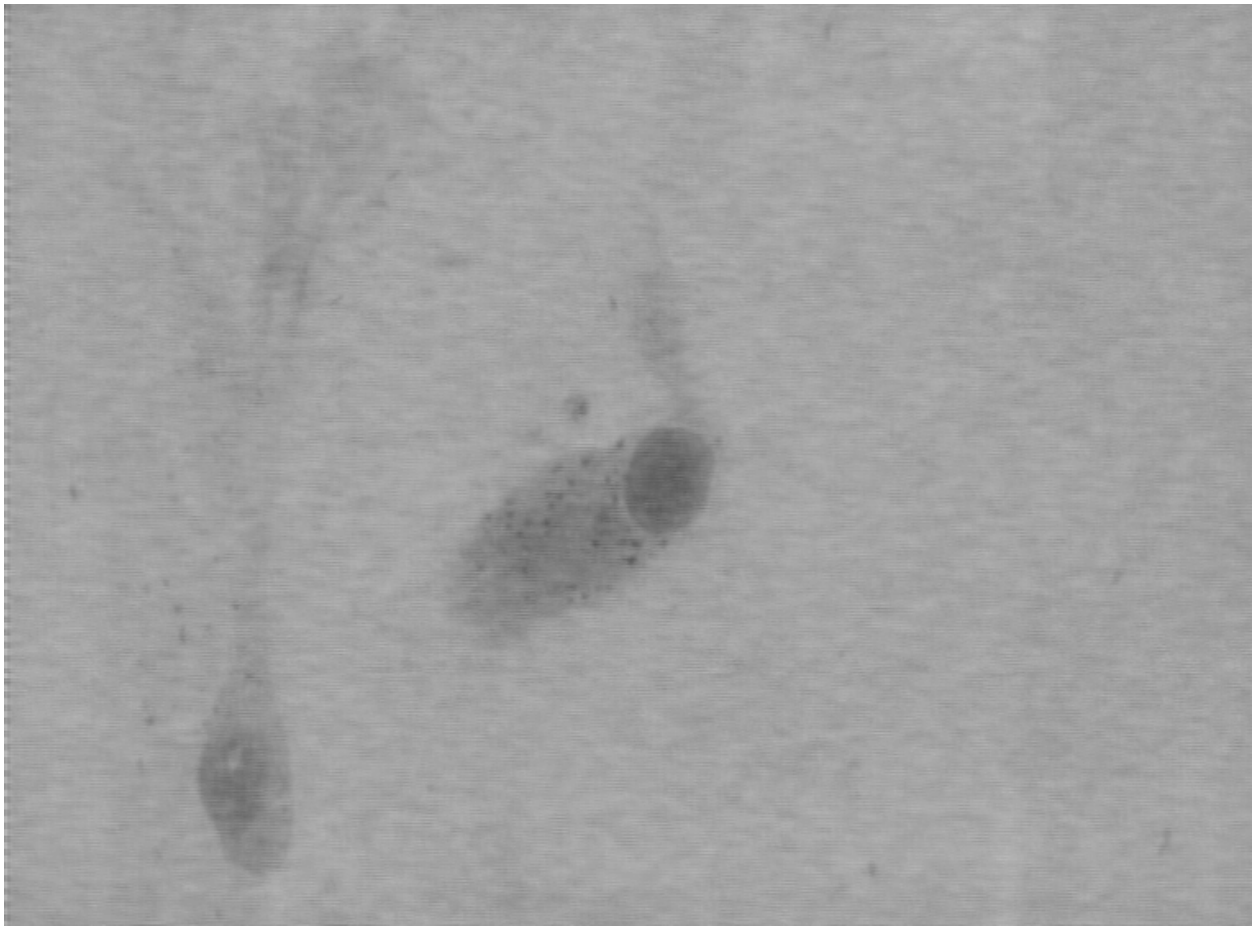
```
Averaged histogram for class cyl: [0.000000e+00 9.910200e+02 0.000000e+00
2.990000e+01 0.000000e+00
```

1.910000e+01 0.000000e+00 1.188000e+01 0.000000e+00 9.820000e+00
0.000000e+00 1.166000e+01 2.000000e-02 1.450000e+01 2.000000e-02
1.874000e+01 4.000000e-02 2.818000e+01 1.400000e-01 3.904000e+01
3.000000e-01 4.978000e+01 6.800000e-01 5.968000e+01 1.180000e+00
7.014000e+01 1.640000e+00 8.204000e+01 2.340000e+00 9.428000e+01
2.640000e+00 1.115800e+02 3.320000e+00 1.222400e+02 4.740000e+00
1.393400e+02 5.220000e+00 1.501000e+02 5.380000e+00 1.637600e+02
6.300000e+00 1.898600e+02 6.920000e+00 2.100000e+02 7.880000e+00
2.312600e+02 9.280000e+00 2.560800e+02 9.720000e+00 2.840800e+02
1.144000e+01 3.147400e+02 1.320000e+01 3.350000e+02 1.554000e+01
3.630600e+02 1.836000e+01 3.892600e+02 2.086000e+01 4.272000e+02
2.410000e+01 4.582000e+02 2.756000e+01 4.725800e+02 3.008000e+01
5.067400e+02 3.478000e+01 5.299200e+02 3.572000e+01 5.466200e+02
3.880000e+01 5.882600e+02 4.548000e+01 6.279600e+02 4.698000e+01
6.571600e+02 5.302000e+01 7.081400e+02 5.946000e+01 7.481200e+02
5.962000e+01 7.732200e+02 6.386000e+01 8.111400e+02 6.430000e+01
8.378600e+02 6.958000e+01 8.706800e+02 6.948000e+01 9.052200e+02
7.180000e+01 9.358400e+02 7.196000e+01 9.785200e+02 7.908000e+01
1.029560e+03 8.132000e+01 1.079960e+03 9.008000e+01 1.142160e+03
9.708000e+01 1.210620e+03 1.038200e+02 1.238760e+03 1.172400e+02
1.313740e+03 1.325600e+02 1.370180e+03 1.429600e+02 1.434280e+03
1.588800e+02 1.506540e+03 1.764200e+02 1.593140e+03 1.941600e+02
1.680480e+03 2.124600e+02 1.801360e+03 2.416600e+02 1.962120e+03
2.674000e+02 2.118920e+03 2.879000e+02 2.297640e+03 3.128400e+02
2.501420e+03 3.319800e+02 2.724920e+03 3.397800e+02 2.950520e+03
3.438600e+02 3.220880e+03 3.408200e+02 3.598020e+03 3.406200e+02
4.091680e+03 3.437600e+02 4.855780e+03 3.306800e+02 5.825420e+03
3.280200e+02 7.112420e+03 3.247400e+02 8.551680e+03 3.205200e+02
1.006458e+04 3.286800e+02 1.167788e+04 3.230400e+02 1.324784e+04
3.230400e+02 1.494030e+04 3.172200e+02 1.677306e+04 3.098400e+02
1.848282e+04 2.990400e+02 2.014304e+04 2.802200e+02 2.138380e+04
2.514200e+02 2.201766e+04 2.315600e+02 2.235316e+04 2.049600e+02
2.262430e+04 1.805400e+02 2.302134e+04 1.583200e+02 2.310700e+04
1.400600e+02 2.215696e+04 1.063600e+02 1.992588e+04 7.340000e+01
1.613142e+04 4.732000e+01 1.217856e+04 2.862000e+01 8.616480e+03
1.496000e+01 5.945680e+03 8.200000e+00 4.069660e+03 5.880000e+00
2.784320e+03 4.840000e+00 1.882060e+03 4.920000e+00 1.284160e+03
4.460000e+00 9.503200e+02 3.580000e+00 8.521600e+02 3.520000e+00
8.972000e+02 3.560000e+00 8.913600e+02 2.080000e+00 7.510000e+02
1.020000e+00 5.080800e+02 5.200000e-01 2.761400e+02 1.800000e-01

1.158000e+02 4.000000e-02 4.034000e+01 0.000000e+00 1.504000e+01
0.000000e+00 5.740000e+00 0.000000e+00 2.760000e+00 0.000000e+00
1.580000e+00 0.000000e+00 1.540000e+00 0.000000e+00 8.600000e-01
0.000000e+00 4.200000e-01 0.000000e+00 1.600000e-01 0.000000e+00
4.000000e-02 0.000000e+00 0.000000e+00 0.000000e+00 2.000000e-02
0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
0.000000e+00]

The **convert to single color function** produces a single-color grayscale image from a grayscale image and a color (red, green, or blue). The function turns the image into grayscale by deleting the desired color channel from the input image and returns the result.

Example **grayscale image:-**



With an input image, the **histogram equalization function** applies histogram equalization. The histogram of the input image and its cumulative distribution function are computed by the function. The function then applies the resulting mapping to the input image to carry out the equalization after normalizing the cumulative distribution function to the range.

Example result:-

```
Histogram for image 1: [[  0   0   2 ...  54  54 112]
 [  0   0   0 ...  37  26  20]
 [  0   0   0 ...  37  54  54]
 ...
 [  0   0   3 ...   9   9  12]
 [  0   0   1 ...   5   5   8]
 [  0   0   0 ...   0   0   0]]
```

With a predefined mask and weight array, the apply **linear filter function** applies a linear filter on an input picture. A blank output image with the same dimensions as the input image is initially created by the function. The function then extracts a neighborhood centered on the pixel that is mask-sized from the input image, computes the weighted sum of the pixels in the neighborhood, and sets the corresponding pixel in the output image to the result.
Example result:-

```
apply_linear_filter filter for image 1: [[ 11  45 119 ... 157 157 161]
 [  1  17  85 ... 155 153 151]
 [  1  17  83 ... 155 157 157]
 ...
 [  1  47 121 ... 141 141 145]
 [  5  53 105 ... 129 129 137]
 [  1   1   1 ...   1   1   1]]
```

The **median filter function** applies a specified mask size to the median filter of an input image. The function initially adds reflection of the boundary pixels to the input image as padding. The function then extracts a mask-sized neighborhood centered on each pixel from the input image, sorts the values of the neighborhood, and sets the corresponding pixel in the output image to the median of the sorted values.
Example result:-

```
median_filter filter for image 1: [[ 11  45 119 ... 157 157 161]
 [  1  17  85 ... 155 153 151]
 [  1  17  83 ... 155 157 157]
 ...
 [  1  47 121 ... 141 141 145]
 [  5  53 105 ... 129 129 137]
 [  1   1   1 ...   1   1   1]]
```

An image's input is sent to the salt and pepper function, which adds **salt-and-pepper noise**. An exact replica of the input image is initially made by the function. The remaining about strength/2 fraction of pixels are then set to black (0) and roughly strength/2 fraction to white (255).
Example result:-

```
salt_pepper noise for image 1: [[ 11  45 119 ... 157 157 161]
 [  1  17  85 ... 155 153 255]
 [  1  17  83 ... 155 157 157]
 ...
```

[1 47 121 ... 141 141 145]
[5 53 105 ... 129 129 137]
[1   1   1 ...   1 255   1]]

Gaussian_noise is added to an input image via the gaussian noise function. By selecting samples from a normal distribution with a defined mean and standard deviation, the function creates an array with the same form as the input image. The output is created by combining the generated noise array with the input image.
Example result:-
gaussian_noise noise for image 1: [[10 45 119 ... 157 157 161]
[0 16 85 ... 155 153 254]
[1 16 83 ... 155 157 157]
 ...
[1 47 121 ... 140 140 145]
[4 53 105 ... 128 129 136]
[1   0   1 ...   1 255   0]]