# LocalBinaryPattern

Author: Hector van der Aa

Date: May 15, 2025

## Project Documentation

This document provides a comprehensive overview of the **LocalBinaryPattern** codebase. It includes descriptions of modules, functions, usage examples, and build instructions.

## Table of Contents

## Installation

There are two options when it comes to installation:

- Prebuilt Binaries
- Manual Building

### Prebuilt Binaires

Prebuilt binaries are available for the following platforms:

- Linux x86_64
- Linux aarch64

- macOS Universal
- Windows

These can be found in the Binaries folder or on the Releases GitHub page.

### Manual Building

If you wish to build from source:

```
git clone https://github.com/H3ct0r55/LocalBinaryPattern.git
cd LocalBinaryPattern
mkdir build && cd build
cmake ..
make
```

## Usage

Once built, run the binary as follows:

```
LocalBinaryPattern [options]
```

Example:

```
LocalBinaryPattern --interactive
```

### Usage Options

**`-h, --help`**

Show this help manual and exit.

**`-i <filename>`**

Specify input file. Supported formats: `.IMAT`, `.TGA`

**`-o <filename>`**

Specify output file. Supported formats: `.IMAT`, `.TGA`, `.TIF`, `.CSV`

**`-e <edgeType>`**

Specify edge handling method. Accepted values: `CropEdge`, `BlackEdge`, `WhiteEdge`, `MirrorEdge`

**`-L`**

Perform Local Binary Pattern (LBP) computation.

**`-P <startPos>`**

Specify the starting position for LBP. Accepted values: `TL`, `TC`, `TR`, `CR`, `BR`, `BC`, `BL`, `CL`

**`-R <direction>`**

Specify rotation direction for LBP. Accepted values: `CW`, `CCW`

**`-I`**

Enable rotation-invariant mode. Cannot be used with `-P` or `-D`.

**`-H <histType>`**

Perform histogram computation. Accepted values: `Raw`, `Normalized`. The output histogram is written to a `.csv` or `.hist` file depending on the extension.

**`-D`**

Display the computed LBP image after processing.

**`--interactive`**

Launch the program in interactive mode.

### Usage Examples

**`LocalBinaryPattern --help`**

Dispalys the Help Manual

**`LocalBinaryPattern -i demo.tga -L`**

Compute the LBP for `demo.tga`

**`LocalBinaryPattern -i demo.tga -L -e MirrorEdge -o lbp.tiff -D`**

Compute the LBP for `demo.tga` with mirror edge case handling, output as `lbp.tiff` and display the result

**`LocalBinaryPattern -i demo.tga -H Raw`**

Computes the raw Histogram for `demo.tga`

**`LocalBinaryPattern -i demo.tga -H Raw -e BlackEdge -o hist.csv`**

Computes the raw Histogram for `demo.tga` with black edge case handling and output as `hist.csv`

**`LocalBinaryPattern --interactive`**

Launches the program in interactive mode, you will be guided through the usage

## Code Overview

### Main Modules

**`main.cpp`**

This is the program entry point. It initializes the environment and handles the main user interface.

**`Image.h`**

This is the main Image class header file, this contains all of the methods linked to image processing.

**`Image.cpp`**

This contains all of the code for the methods listed in `Image.h`

**`help_text.h`**

This contains the help manual text

`TypeDetect.h`

This is the header file for parsing string arguments and contains a number of functions to parse user input strings into int values that can be used with the Image class

`TypeDetect.cpp`

This contains all of the code for the functions listed in `TypeDetect.h`

`InteractiveWizard.h`

This is the header file for the Interactive Wizard

`InteractiveWizard.cpp`

This contains all of the code for the Interactive Wizard

# Using the Image Class

You can use the `Image` class independently in other projects. To do so, copy `Image.h` and `Image.cpp` into your project and include the header in your `main.cpp` as follows:

```
#include "Image.h"
```

## Data Structure

The image class is composed to two dimension values and a 2D array storing image data as follows:

```
int m_width;
int m_height;
uint8_t** m_p_data;
```

The 2D array is row-major, meaning rows (height) come before columns (width). To access a pixel at location `(x, y)`, use `m_p_data[y][x]`.

## Constructors

`Image()`

This is the default constructor, it sets the object to the following:

```
m_width = 0;
m_height = 0;
m_p_data = nullptr;
```

`Image(int width, int height)`

This is the constructor with dimensions, it sets the object to the following:

```
m_width = width;
m_height = height;
```

And `m_p_data` is constructed to a 2D array full of 0's

`Image(int width, int height, bool randFill)`

This is the constructor with dimensions and random fill, it sets the object to the following:

```
m_width = width;
m_height = height;
```

if `randFill == true` then the array is filled with random values from 0-255, otherwise if `randFill == false`, the constructor behaves the same as `Image(int width, int height)`.

**`Image(int width, int height, int valFill)`**

This constructor initializes the image with specified dimensions and fills it with a constant value.

```
m_width = width;
m_height = height;
```

and fills the 2D array with the value passed as `valFill`

**`Image(const Image& image)`**

This is the default copy constructor

**`Image(const Image& image, int borderWidth, int borderValue)`**

This is the copy constructor that adds a border of width `borderWidth` and of value `borderValue` around the image

**`Image(const Image& image, int mirrorBorderWidth)`**

This is the copy constructor that adds a mirrored border of width `mirrorBorderWidth` around the image

### Destructors

**`~Image()`**

This is the default destructor

### Image Modification

**`void randFill() const`**

Fills the image with random values from 0 to 255.

**`void valFill(int value) const`**

Fills the entire image with the specified constant value.

**`void setVal(int x, int y, uint8_t val)`**

Sets the value of the pixel at `(x, y)` to `val`.

**`void fillRange(int startX, int startY, int endX, int endY, uint8_t val)`**

Fills a rectangular region from `(startX, startY)` to `(endX, endY)` with the value `val`.

### Input / Output

**`bool writeIMAT(const path& filename)`**

Writes the image to a `.IMAT` file. Returns `true` on success.

**`bool writeTGA(const path& filename, int colorType)`**

Writes the image to a `.TGA` file. `colorType` must be `Grayscale` or `RGB`.

**bool writeTIF(const path& filename, int colorType)**

Writes the image to a `.TIF` file. `colorType` must be `Grayscale` or `RGB`.

**void readIMAT(const path& filename)**

Reads an image from a `.IMAT` file.

**void readTGA(const path& filename)**

Reads an image from a `.TGA` file.

## Display

**void displayImage()**

Displays the image using a default viewer.

## LBP (Local and Global)

**uint8_t* unwrapLocal(int x, int y, int startPos, int rotation)**

Returns a pointer to 8 neighboring pixels of `(x, y)` starting from `startPos` and rotating in `rotation` direction.

- `x`, `y`: Pixel coordinates.
- `startPos`: Starting neighbor (e.g., TL, TC, etc.).
- `rotation`: Rotation direction (`CW`, `CCW`).
- **Returns:** `uint8_t*` array of 8 neighbor values.

**int startPosRLBP(int x, int y)**

Returns the index of the neighboring pixel with the largest absolute difference to the center pixel, used for rotation-invariant LBP.

**uint8_t* localLBP(int x, int y, int startPos, int rotation)**

Computes the local binary pattern for pixel `(x, y)` based on its neighbors. Returns a pointer to 8 binary values.

**Image computeLBP(int edgeType, int startPos, int rotation)**

Computes the global LBP image using the given edge type and neighbor ordering. Returns a new `Image` object.

**Image computeRILBP(int edgeType)**

Computes the rotation-invariant global LBP image. Returns a new `Image` object.

## Histogram Computation

**uint32_t* computeRawHist()**

Computes and returns the raw histogram of pixel values (array of 256 `uint32_t` bins).

**double* computeNormHist()**

Computes and returns the normalized histogram of pixel values (array of 256 `double` bins).

## Operators

**`friend ostream& operator<<(ostream& os, const Image& image)`**

Overloads the output stream operator to print image metadata.

**`Image& operator=(const Image& image)`**

Overloads the assignment operator.

## Helper Functions

**`uint8_t castToInt(const uint8_t* input)`**

Casts an 8-bit LBP binary to an integer.

**`uint8_t castToInt(const uint8_t* input, bool rotationInvariant)`**

Casts an LBP binary to integer, accounting for rotation invariance.

**`bool writeRHIST(uint32_t* histogram, const path& filename)`**

Writes a raw histogram to `.hist` format.

**`bool writeRHISTCSV(uint32_t* histogram, const path& filename)`**

Writes a raw histogram to `.csv`.

**`bool writeNHIST(double* histogram, const path& filename)`**

Writes a normalized histogram to `.hist`.

**`bool writeNHISTCSV(double* histogram, const path& filename)`**

Writes a normalized histogram to `.csv`.

**`uint32_t* readRHIST(const path& filename)`**

Reads a raw histogram from file.

**`double* readNHIST(const path& filename)`**

Reads a normalized histogram from file.

**`void clearCache()`**

Clears any cached image data if applicable.

**`void displayImage(const path& filename)`**

Displays the image located at the specified path.

**`void displayTestImage()`**

Displays a default test image.

## Examples

### Basic LBP Computation

```
#include "Image.h"

int main() {
    Image img;
    img.readTGA("demo.tga");

    Image lbp = img.computeLBP(MirrorBorder, TL, CW);
    lbp.writeTIF("output.tif", Grayscale);
}
```

### Compute Rotation-Invariant LBP and Display

```
#include "Image.h"

int main() {
    Image img("demo.tga");
    Image rilbp = img.computeRILBP(BlackBorder);
    rilbp.displayImage();
}
```

### Generate and Save Normalized Histogram

```
#include "Image.h"

int main() {
    Image img;
    img.readTGA("demo.tga");

    double* normHist = img.computeNormHist();
    writeNHISTCSV(normHist, "histogram.csv");
    delete[] normHist;
}
```

### Fill Region with Constant Value

```
#include "Image.h"

int main() {
    Image img(512, 512);
    img.fillRange(100, 100, 400, 400, 255);
    img.writeIMAT("filled.imat");
}
```

## License