

LocalBinaryPattern

Auteur : Hector van der Aa

Date : May 11, 2025

Documentation du Projet

Ce document fournit une vue d'ensemble complète de la base de code **LocalBinaryPattern**. Il inclut des descriptions de modules, de fonctions, des exemples d'utilisation et des instructions de compilation.

Table des matières

- Installation
 - Binaires pré-compilés
 - Compilation manuelle
- Utilisation
 - Options d'utilisation
 - Exemples d'utilisation
- Vue d'ensemble du code
 - Modules principaux
- Utilisation de la classe Image
 - Structure de données
 - Constructeurs
 - Destructeurs
 - Modification d'image
 - Entrée / Sortie
 - Affichage
 - LBP (Local et Global)
 - Calcul d'histogramme
 - Opérateurs
 - Fonctions utilitaires
- Exemples
- Licence

Installation

Il existe deux options pour l'installation :

- Binaires pré-compilés
- Compilation manuelle

Binaires pré-compilés

Des binaires pré-compilés sont disponibles pour les plateformes suivantes :

- Linux x86_64
- Linux aarch64

- macOS Universal
- Windows

Ils se trouvent dans le dossier *Binaries* ou sur la page Releases de GitHub.

Compilation manuelle

Si vous souhaitez compiler à partir des sources :

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

Utilisation

Une fois compilé, exécutez le binaire comme suit :

`LocalBinaryPattern [options]`

Exemple :

`LocalBinaryPattern --interactive`

Options d'utilisation

-h, --help

Affiche ce manuel d'aide puis quitte.

-i <fichier>

Spécifie le fichier d'entrée. Formats supportés : .IMAT, .TGA

-o <fichier>

Spécifie le fichier de sortie. Formats supportés : .IMAT, .TGA, .TIF, .CSV

-e <edgeType>

Spécifie la méthode de gestion des bords. Valeurs acceptées : CropEdge, BlackEdge, WhiteEdge, MirrorEdge

-L

Effectue le calcul du Local Binary Pattern (LBP).

-P <startPos>

Spécifie la position de départ pour le LBP. Valeurs acceptées : TL, TC, TR, CR, BR, BC, BL, CL

-D <direction>

Spécifie la direction de rotation pour le LBP. Valeurs acceptées : CW, CCW

-I

Active le mode invariant à la rotation. Ne peut pas être utilisé avec -P ou -D.

-H <histType>

Effectue le calcul de l'histogramme. Valeurs acceptées : **Raw**, **Normalized**. L'histogramme est écrit dans un fichier **.csv** ou **.hist** selon l'extension.

-D

Affiche l'image LBP calculée après traitement.

--interactive

Lance le programme en mode interactif.

Exemples d'utilisation

LocalBinaryPattern --help

Affiche le manuel d'aide.

LocalBinaryPattern -i demo.tga -L

Calcule le LBP pour **demo.tga**.

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

Calcule le LBP pour **demo.tga** avec gestion des bords miroir, écrit le résultat dans **lbp.tiff** et l'affiche.

LocalBinaryPattern -i demo.tga -H Raw

Calcule l'histogramme brut pour **demo.tga**.

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

Calcule l'histogramme brut pour **demo.tga** avec gestion des bords noirs et l'enregistre dans **hist.csv**.

LocalBinaryPattern --interactive

Lance le programme en mode interactif et vous guide pas à pas.

Vue d'ensemble du code

Modules principaux

main.cpp

Point d'entrée du programme. Initialise l'environnement et gère l'interface utilisateur principale.

Image.h

En-tête de la classe Image principale, contenant toutes les méthodes liées au traitement d'images.

Image.cpp

Contient tout le code des méthodes listées dans **Image.h**.

help_text.h

Contient le texte du manuel d'aide.

TypeDetect.h

Fichier d'en-tête pour l'analyse des arguments chaîne ; contient plusieurs fonctions convertissant les chaînes d'entrée utilisateur en valeurs entières utilisables par la classe Image.

TypeDetect.cpp

Implémente les fonctions listées dans TypeDetect.h.

InteractiveWizard.h

Fichier d'en-tête de l'assistant interactif.

InteractiveWizard.cpp

Implémente l'assistant interactif.

Utilisation de la classe Image

Vous pouvez utiliser la classe `Image` indépendamment dans d'autres projets. Copiez simplement `Image.h` et `Image.cpp` dans votre projet et incluez l'en-tête dans votre `main.cpp` :

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

Structure de données

La classe Image se compose de deux valeurs de dimension et d'un tableau 2D stockant les données d'image :

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

Le tableau 2D est en mode *row-major* (lignes avant colonnes). Pour accéder au pixel à la position (x, y), utilisez `m_p_data[y][x]`.

Constructeurs

`Image()`

Constructeur par défaut :

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

`Image(int width, int height)`

Constructeur avec dimensions :

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

`m_p_data` est alors un tableau 2D rempli de zéros.

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

Constructeur avec dimensions et remplissage aléatoire :

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

Si `randFill == true`, le tableau est rempli de valeurs aléatoires de 0 à 255 ; sinon le comportement est identique à `Image(int width, int height)`.

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

Constructeur initialisant l'image avec des dimensions données et en la remplissant d'une valeur constante :

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

et remplit le tableau 2D avec la valeur `valFill`.

`Image(const Image& image)`

Constructeur de copie par défaut.

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

Constructeur de copie ajoutant une bordure de largeur `borderWidth` et de valeur `borderValue` autour de l'image.

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

Constructeur de copie ajoutant une bordure miroir de largeur `mirrorBorderWidth` autour de l'image.

Destructeurs

`~Image()`

Destructeur par défaut.

Modification d'image

`void randFill() const`

Remplit l'image avec des valeurs aléatoires de 0 à 255.

`void valFill(int value) const`

Remplit toute l'image avec la valeur constante spécifiée.

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

Attribue la valeur `val` au pixel `(x, y)`.

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

Remplit le rectangle de `(startX, startY)` à `(endX, endY)` avec la valeur `val`.

Entrée / Sortie

`bool writeIMAT(const path& filename)`

Écrit l'image dans un fichier `.IMAT`. Retourne `true` en cas de succès.

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

Écrit l'image dans un fichier `.TGA`. `colorType` doit être `Grayscale` ou `RGB`.

bool writeTIF(const path& filename, int colorType)

Écrit l'image dans un fichier .TIF. colorType doit être Grayscale ou RGB.

void readIMAT(const path& filename)

Lit une image depuis un fichier .IMAT.

void readTGA(const path& filename)

Lit une image depuis un fichier .TGA.

Affichage

void displayImage()

Affiche l'image via le visualiseur par défaut.

LBP (Local et Global)

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

Renvoie un pointeur vers les 8 pixels voisins de (x, y) en commençant à startPos et en tournant selon rotation.

- x, y : coordonnées du pixel.
- startPos : voisin de départ (par ex. TL, TC, etc.).
- rotation : sens de rotation (CW, CCW).
- **Renvoie** : tableau uint8_t* de 8 valeurs voisines.

int startPosRLBP(int x, int y)

Renvoie l'index du voisin ayant la plus grande différence absolue avec le pixel central, utilisé pour le LBP invariant à la rotation.

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

Calcule le LBP local pour le pixel (x, y) selon ses voisins. Renvoie un pointeur vers 8 valeurs binaires.

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

Calcule l'image LBP globale selon le type de bord et l'ordre des voisins. Renvoie un nouvel objet Image.

Image computeRILBP(int edgeType)

Calcule l'image LBP globale invariant à la rotation. Renvoie un nouvel objet Image.

Calcul d'histogramme

uint32_t* computeRawHist()

Calcule et renvoie l'histogramme brut des valeurs de pixels (tableau de 256 cases uint32_t).

double* computeNormHist()

Calcule et renvoie l'histogramme normalisé des valeurs de pixels (tableau de 256 cases double).

Opérateurs

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

Surcharge de l'opérateur de flux de sortie pour imprimer les métadonnées de l'image.

Image& operator=(const Image& image)

Surcharge de l'opérateur d'affectation.

Fonctions utilitaires

uint8_t castToInt(const uint8_t* input)

Convertit un LBP binaire 8 bits en entier.

uint8_t castToInt(const uint8_t* input, bool rotationInvariant)

Convertit un LBP binaire en entier, en tenant compte de l'invariance à la rotation.

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

Écrit un histogramme brut au format `.hist`.

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

Écrit un histogramme brut au format `.csv`.

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

Écrit un histogramme normalisé au format `.hist`.

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

Écrit un histogramme normalisé au format `.csv`.

uint32_t* readRHIST(const path& filename)

Lit un histogramme brut depuis un fichier.

double* readNHIST(const path& filename)

Lit un histogramme normalisé depuis un fichier.

void clearCache()

Efface toute donnée d'image mise en cache si applicable.

void displayImage(const path& filename)

Affiche l'image située au chemin spécifié.

void displayTestImage()

Affiche une image de test par défaut.

Exemples

Calcul de LBP de base

```
#include "Image.h"

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

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

Calcul d'un LBP invariant à la rotation et affichage

```
#include "Image.h"

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

Génération et sauvegarde d'un histogramme normalisé

```
#include "Image.h"

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

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

Remplir une région avec une valeur constante

```
#include "Image.h"

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

Licence

Ce projet est distribué sous licence MIT.