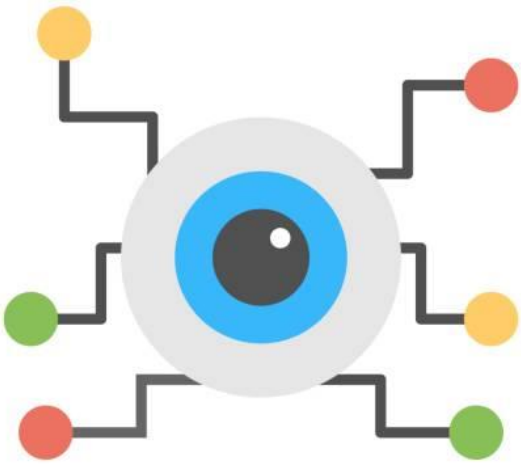
كلية العلوم و التقنيات فاس
ⵜⴰⵖⵙⵓⵍⵜ ⵏ ⵜⵇⴰⵙⵙⵉⵡⵉⵏ ⴷ ⵜⵓⵙⵙⵙⵉ
Faculté des Sciences et Techniques de Fès

جامعة سيدي محمد بن عبد الله
ⵜⴰⵙⴷⴰⵡⵉⵜ ⵙⵉⴷⵉ ⵎⵓⵃⴻⵎⵎⴷ ⴱⵏ ⵄⴱⴷⵍⵍⴰⵀ
Université Sidi Mohamed Ben Abdellah



Projet de Vision Par Ordinateur

REALISE PAR :

- EL MOKADEM MOHAMMED
- BOUDRA AYOUB

ENCADRÉ PAR :

- PR. YOUNESS DHASSI
- PR. ARSALANE ZARGHILI

I. Introduction :

L'objectif de ce mini-projet est de réaliser une application dans laquelle on implémente des fonctions de traitement d'image tel que :

Binarisation, égalisation et étirement d'image, filtrage d'image avec les filtres { Gaussien, moyenneur, médian **}, extraction des contours dans l'image en utilisant plusieurs algorithmes {** Gradient, Sobel, Robert, Laplacien **}, application de la morphologie mathématique {** Erosion, Dilatation, Ouverture, Fermeture, Filtrage morphologique **}, segmentation d'image en utilisant les algorithmes {** KMeans, Croissance de régions D, Partition de régions D **}, Détection des points d'intérêt et la compression d'image par les algorithmes suivants {** Huffman, LZW, Ondelette **}**

II. Les technologies utilisées :

Pour ce qui concerne les technologies qu'on a utilisé pour réaliser ce mini-projet :

Le langage qu'on a utilisé est **Python**, pourquoi on a choisi ce langage ?

Python est un langage de programmation populaire pour le traitement d'images pour plusieurs raisons :

Grande variété de bibliothèques : Python dispose d'un large éventail de bibliothèques pour le traitement d'images, telles qu'**OpenCV**, **Pillow** (PIL), **scikit-image**, **mahotas**, **SimpleCV**, etc. Ces bibliothèques fournissent des outils pour effectuer une grande variété de tâches de traitement d'images, telles que la lecture et l'écriture d'images, le filtrage, la transformation, la segmentation, la détection d'objets, la reconnaissance d'objets, la correspondance de caractéristiques, etc.

Syntaxe simple et facile à apprendre : Python a une syntaxe simple, lisible et facile à apprendre qui permet aux débutants d'apprendre rapidement le traitement d'images. Les bibliothèques de traitement d'images Python sont également bien documentées avec de nombreux exemples de code et de tutoriels disponibles en ligne.

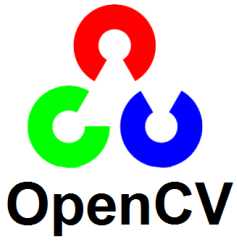
Interactivité : Python est un langage de programmation interprété qui permet d'exécuter du code en temps réel et de visualiser les résultats immédiatement. Cela permet aux utilisateurs de tester et de développer rapidement des solutions de traitement d'images en temps réel.

Pour l'interface :

On a utilisé **PYQT5** pour créer l'interface de notre application. PyQt5 est une bibliothèque Python qui permet de créer des interfaces graphiques (GUI) pour des applications de bureau. PyQt5 est construit sur la base de la bibliothèque **Qt**, qui est une bibliothèque de développement d'applications multiplateformes. PyQt5 permet de créer des applications pour les systèmes d'exploitation Windows, Mac OS X et Linux.

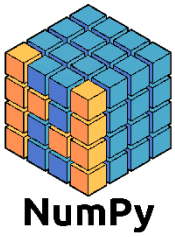
Pour les bibliothèques, qu'on a utilisé pour programmer les fonctions de traitement d'images :

On a utilisé plusieurs bibliothèques différentes pour que notre code soit divers :



OpenCV (Open Source Computer Vision Library) est une bibliothèque open source de vision par ordinateur et de traitement d'images. Elle fournit des fonctions pour le traitement d'images, le traitement vidéo et la vision par ordinateur en temps réel. **OpenCV** est écrit en C++, mais dispose d'une interface Python pour permettre l'utilisation de la bibliothèque dans des applications Python.

Pillow est une bibliothèque populaire pour le traitement d'images car elle est facile à utiliser et offre une grande variété de fonctions pour les tâches courantes de traitement d'images, telles que le redimensionnement, la recadrage, la rotation, la conversion de couleurs, l'ajout de texte, la modification de la qualité de l'image, etc. **Pillow** est également compatible avec la bibliothèque Python **NumPy**, ce qui permet d'utiliser des tableaux **NumPy** pour le traitement d'images.



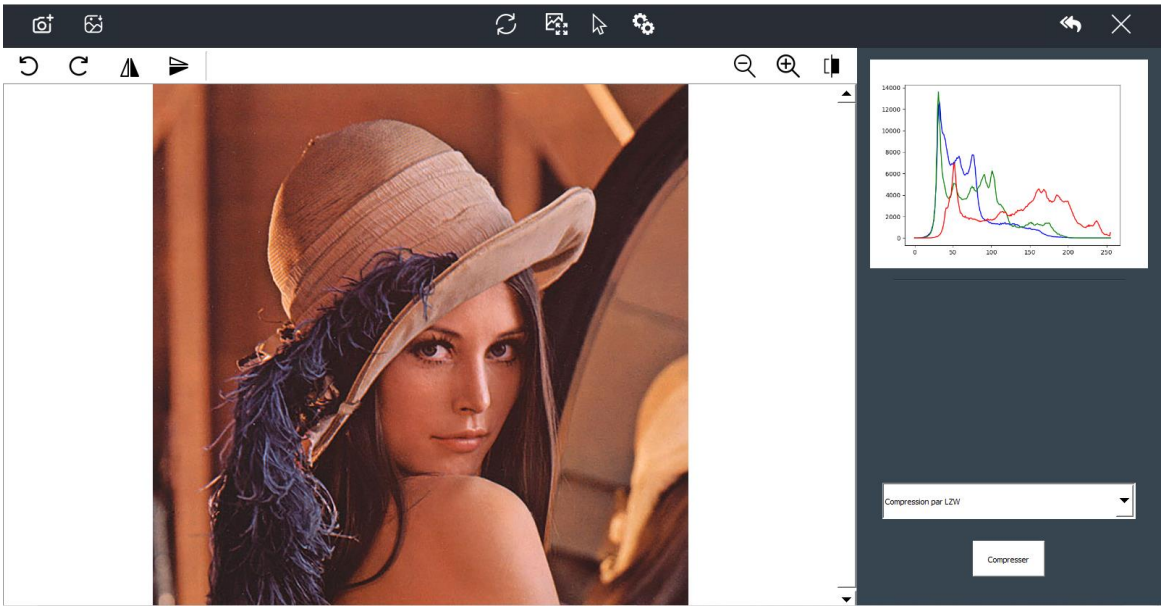
NumPy est largement utilisé dans le traitement d'images car il fournit des tableaux multidimensionnels (ndarray) optimisés pour le traitement d'images. Les images numériques peuvent être représentées sous forme de tableaux **NumPy** où chaque élément de tableau représente un pixel de l'image. Les opérations de traitement d'images, telles que le filtrage, la segmentation, la transformation et la reconstruction d'images peuvent être effectuées en utilisant des fonctions mathématiques de **NumPy** sur les tableaux d'images.

SciPy est une bibliothèque open source de calculs scientifiques pour le langage de programmation **Python**. Elle fournit des fonctions et des outils pour résoudre une variété de problèmes mathématiques, scientifiques et d'ingénierie, tels que l'optimisation, l'algèbre linéaire, l'analyse de signal, la statistique, la simulation, l'analyse de données, etc.

SciPy est construite sur **NumPy** et utilise des tableaux multidimensionnels **NumPy** pour effectuer des calculs. Elle fournit également des fonctions pour effectuer des opérations plus avancées sur les tableaux, telles que l'interpolation, la convolution, la transformée de Fourier, etc.

Scikit-image, également connu sous le nom skimage, est une bibliothèque open source de traitement d'images pour Python. Elle fournit des outils pour le prétraitement, le traitement et l'analyse d'images numériques. Scikit-image est construite sur NumPy, SciPy et matplotlib.

III. Application :



Fonctions :

Pour l'ajout de nouveau image et sauvegarde d'une image :

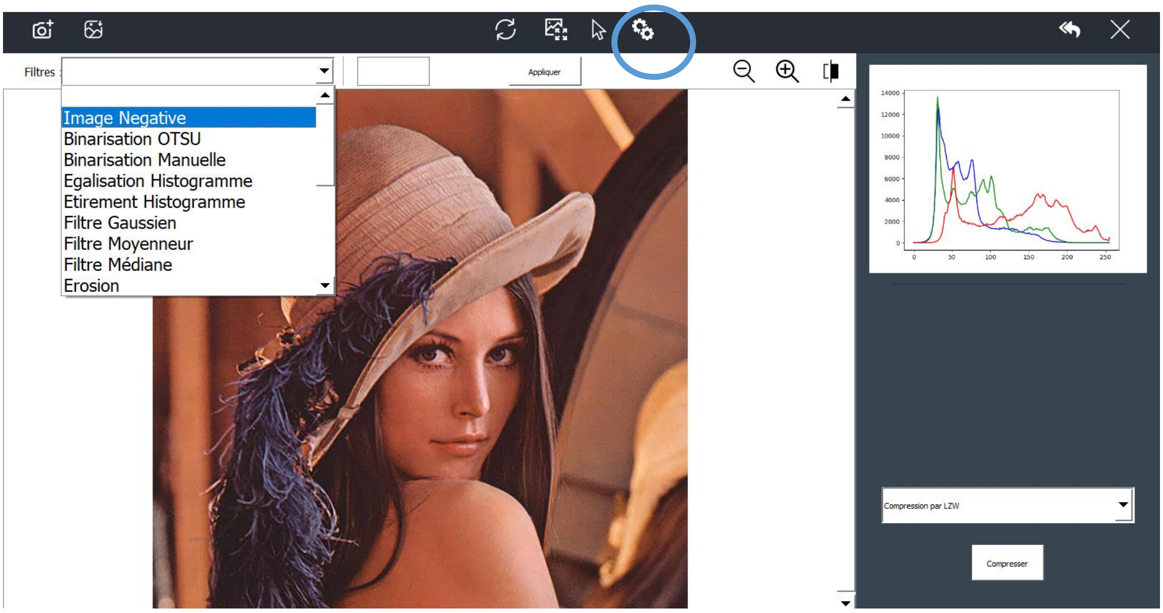
L'ajout :

```
def openFileDialog(self):
        file_types = "Images (*.png *.jpg *.jpeg *.tif *.tiff *.gif)"
        fileDialog = QFileDialog()
        self.fileName = fileDialog.getOpenFileName(None, "Open File", "", file_types)
        if(self.fileName[0] != ''):
                self.zoom_factor = 1
                self.image = QImage(self.fileName[0])
                self.original_image = self.image.copy()
                w = self.image.width()
                h = self.image.height()
                if w > 1080 or h > 900:
                        self.imageLabel.setPixmap(QPixmap().fromImage(self.image).scaled(900,500,aspectRatioMode=Qt.KeepAspectRatio)
                else:
                        self.imageLabel.setPixmap(QPixmap().fromImage(self.image))
                self.imageLabel.resize(self.imageLabel.pixmap().size())
                self.histogramImage()
        else:pass
```

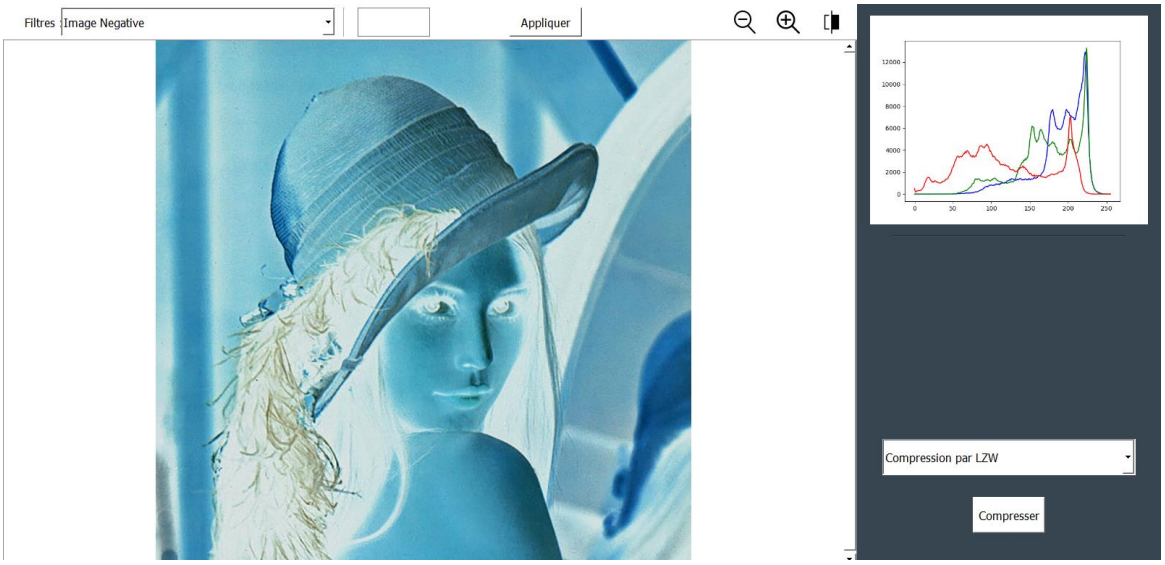
La sauvegarde :

```
def saveImage(self):
        """Save the image displayed in the label."""
        #TODO: Add different functionality for the way in which the user can save their image.
        if self.image.isNull() == False:
                image_file, _ = QFileDialog.getSaveFileName(self.imageLabel, "Save Image",
                        "", "PNG Files (*.png);;JPG Files (*.jpeg *.jpg );;Bitmap Files (*.bmp);;\
                        GIF Files (*.gif)")

                if image_file and self.image.isNull() == False:
                        self.image.save(image_file)
                else:
                        QMessageBox.information(self.imageLabel, "Error",
                        "Unable to save image.", QMessageBox.Ok)
        else:
                QMessageBox.information(self, "Empty Image",
                "There is no image to save.", QMessageBox.Ok)
```

L'image négative :

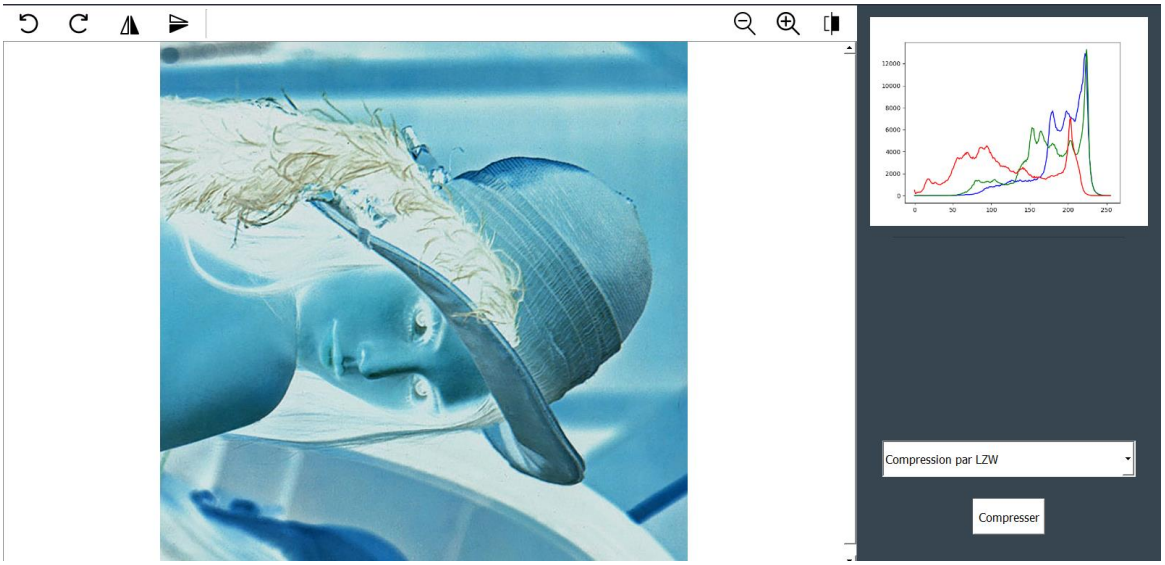


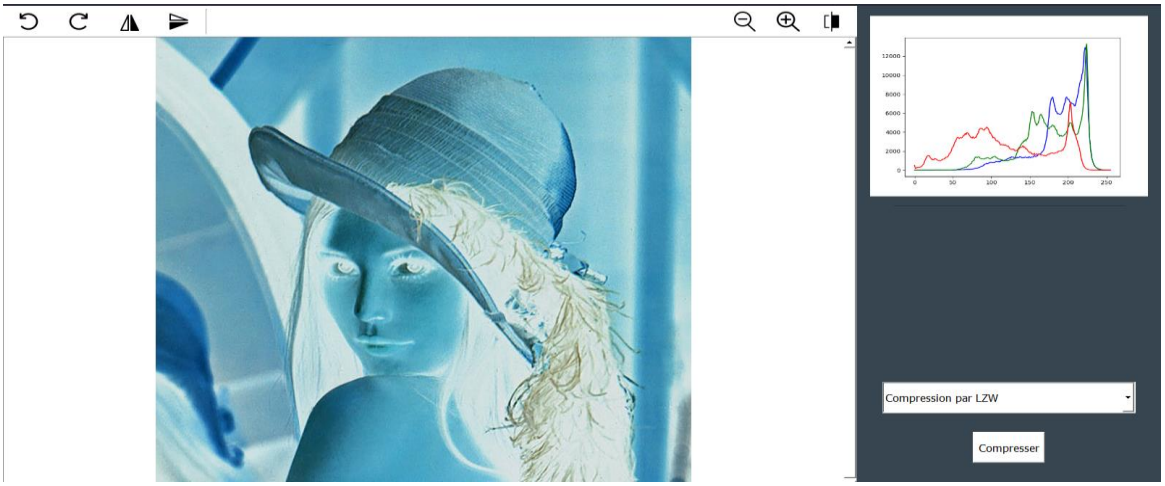

```python
def negative(self):
        picVal = self.imageLabel.pixmap()
        pic = picVal.toImage()
        image=self.convertQImageToMat(pic)
        image=cv2.cvtColor(image,cv2.COLOR_BGR2RGB)

        x,y,c = image.shape
        if self.is_grey_scale(image):
                bytes_per_line = 1 * x
                image = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
                for i in range(x):
                        for j in range(y):
                                image[i][j]=255-image[i][j]
                self.image = image
                q_image = QImage(image.data, y, x, bytes_per_line, QImage.Format.Format_Grayscale8)
                self.image = q_image
                self.imageLabel.setPixmap(QPixmap.fromImage(q_image))
        else:
                bytes_per_line = 3 * y
                for i in range(x):
                        for j in range(y):
                                for k in range(3):
                                        image[i][j][k]=255-image[i][j][k]
                cv2.imwrite('figure/negative.jpg',image) # Save the negative image
                imageNegative = cv2.imread('figure/negative.jpg') # Retrieve it so we can display it
                try:
                        os.remove('figure/negative.jpg') # Delete the negative image from folder
                except:pass
                q_image = QImage(imageNegative.data, y, x, bytes_per_line, QImage.Format.Format_RGB888)
                self.image = q_image
                self.imageLabel.setPixmap(QPixmap.fromImage(q_image))
        self.histogramImage()
```



Rotation d'image :

Les 2 premiers fassent une rotation -90°, +90° respectivement, et pour les 2 derniers ils retournent l'image verticalement et horizontalement respectivement.





Rotation en 90° et -90° :

```python
def rotateImage90(self, direction):
        """Rotate image 90º clockwise or counterclockwise."""
        if self.image.isNull() == False:
                if direction == "cw":
                        transform90 = QTransform().rotate(90)
                elif direction == "ccw":
                        transform90 = QTransform().rotate(-90)

                pixmap = QPixmap(self.image)

                #TODO: Try flipping the height/width when flipping the image

                rotated = pixmap.transformed(transform90, mode=Qt.SmoothTransformation)
                self.imageLabel.resize(self.image.height(), self.image.width())
                #rotated = pixmap.trueMatrix(transform90, pixmap.width, pixmap.height)

                #self.image_label.setPixmap(rotated.scaled(self.image_label.size(),
                #    Qt.KeepAspectRatio, Qt.SmoothTransformation))
                self.image = QImage(rotated)
                #self.setPixmap(rotated)
                self.imageLabel.setPixmap(rotated.scaled(self.imageLabel.size(), Qt.KeepAspectRatioByExpanding, Qt.SmoothTransformation))
                self.imageLabel.repaint() # repaint the child widget
        else:
                # No image to rotate
                pass
```

Flip l’image horizontalement et verticalement :

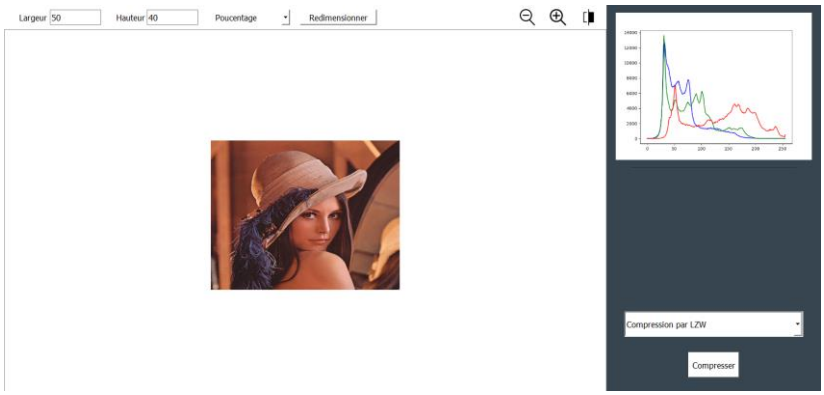
```python
                pass
    def flipImage(self, axis):
            if self.image.isNull() == False:
                    if axis == "horizontal":
                            flip_h = QTransform().scale(-1, 1)
                            pixmap = QPixmap(self.image)
                            flipped = pixmap.transformed(flip_h)
                    elif axis == "vertical":
                            flip_v = QTransform().scale(1, -1)
                            pixmap = QPixmap(self.image)
                            flipped = pixmap.transformed(flip_v)
                    self.image = QImage(flipped)
                    self.imageLabel.setPixmap(flipped)
                    self.imageLabel.repaint()
            else:
                    # No image to flip
                    pass
```

Redimensionnement :

Dans le select box on peut choisir soit pixel ou bien pourcentage et on insère les valeurs pour largeur et hauteur :

```
                pass
def resizeImage(self):
        """Resize image."""
        #TODO: Resize image by specified size
        if self.image.isNull() == False:
                img_x,img_y = self.image.width(),self.image.height()
                print(img_x,img_y)

                if self.heightTE.text() != "" and self.widthTE.text() != "" :
                        x=int(self.widthTE.text())
                        y=int(self.heightTE.text())
                        if self.comboBox.currentText() == "Pixels":
                                x,y = x/img_x,y/img_y
                        else:
                                x,y = x/100,y/100
                else:
                        x,y = 0.5,0.5
                print(x,y)
                resize = QTransform().scale(x, y)


                pixmap = QPixmap(self.image)


                resized_image = pixmap.transformed(resize, mode=Qt.SmoothTransformation)
                self.image = QImage(resized_image)
                self.imageLabel.setPixmap(resized_image)
                self.imageLabel.resize(self.imageLabel.pixmap().size())

                self.imageLabel.setScaledContents(True)
                self.imageLabel.repaint()
        else:
        # No image to rotate
                pass
```

Résultat :

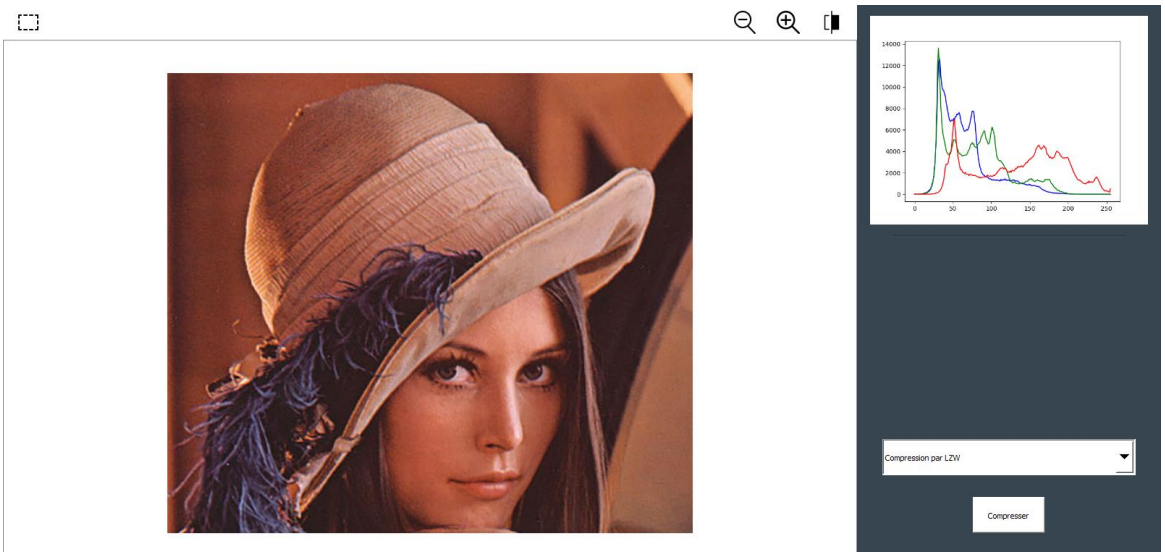


Sélectionner manuellement une zone dans une image :

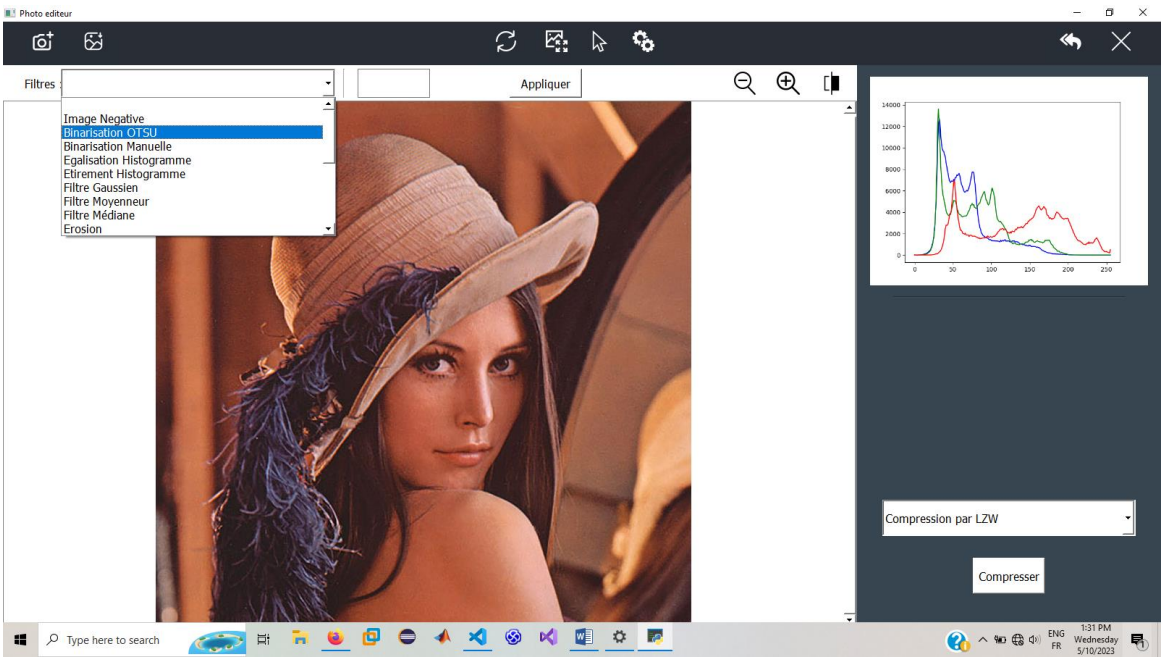
On sélectionne la zone qu'on veut dans l'image et on clique sur Entrer pour confirmer ou bien on sélectionne une autre zone.

```
def crop(self):
        picVal = self.imageLabel.pixmap()
        pic = picVal.toImage()
        imageS=self.convertQImageToMat(pic)
        imageS = cv2.cvtColor(imageS,cv2.COLOR_BGR2RGB)
        image=imageS
        image = cv2.cvtColor(image,cv2.COLOR_BGR2RGB)
        x, y, w, h = cv2.selectROI('image', image, False)

        # Draw the rectangle on the image
        cv2.rectangle(image, (x, y), (x + w, y + h), (255, 255, 255), 2)

        # Display the image with the rectangle
        cv2.imshow('image', image)
        cv2.waitKey(0)
        cv2.destroyAllWindows()
        imageS=cv2.cvtColor(imageS,cv2.COLOR_BGR2RGB)
        imageFin = imageS[y:y+h, x:x+w]
        imageFin = cv2.cvtColor(imageFin,cv2.COLOR_BGR2RGB)
        height, width, channel = imageFin.shape
        bytes_per_line = 3 * width
        q_image = QImage(imageFin.data, width, height, bytes_per_line, QImage.Format.Format_RGB888)
        self.image = q_image
        self.imageLabel.setPixmap(QPixmap.fromImage(q_image))
```

Résultat :



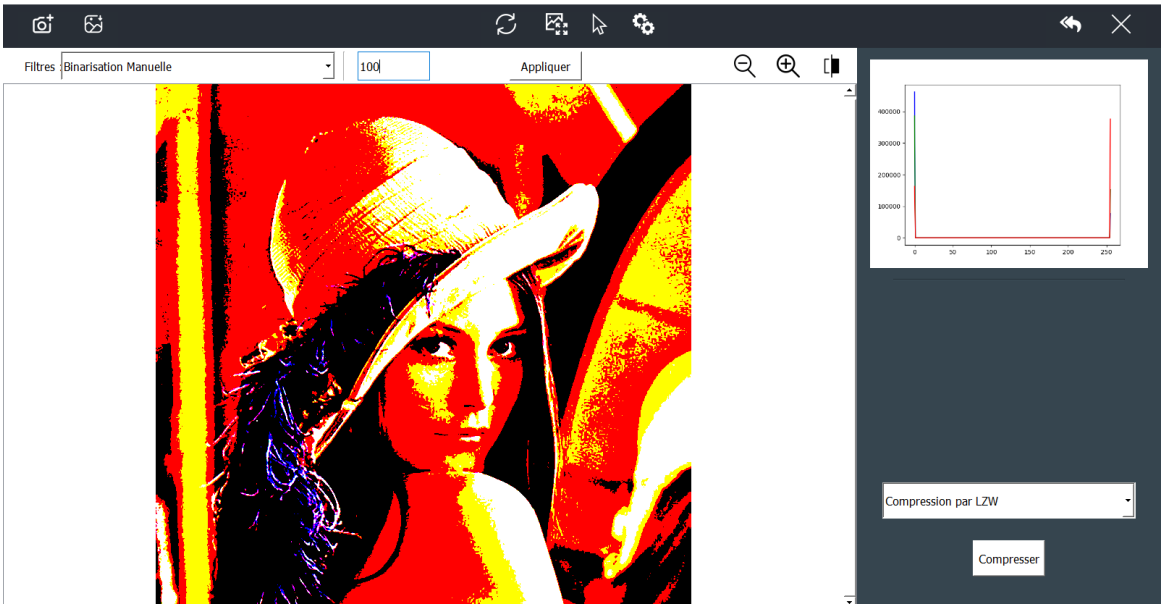
Binariser une image :



Binarisation manuel :

```
def binarisationManuel(self):
        thres = self.parametre.text()
        if(thres == ""):
                thres = 128
        else:
                thres=int(thres)
        picVal = self.imageLabel.pixmap()
        pic = picVal.toImage()
        image=self.convertQImageToMat(pic)
        image=cv2.cvtColor(image,cv2.COLOR_BGR2RGB)
        if self.is_grey_scale(image):
                image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
                th, image = cv2.threshold(image, thres, 255, cv2.THRESH_BINARY)
                image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
                height, width, channel = image.shape
                bytes_per_line = 3 * width
                print(th)
                q_image = QImage(image.data, width, height, bytes_per_line, QImage.Format.Format_RGB888)
                self.image = q_image
                self.imageLabel.setPixmap(QPixmap.fromImage(q_image))
        else:
                image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
                R=image[:,:,0]
                G=image[:,:,1]
                B=image[:,:,2]
                _, R_image = cv2.threshold(R, thres, 255, cv2.THRESH_BINARY)
                _, G_image = cv2.threshold(G, thres, 255, cv2.THRESH_BINARY)
                _, B_image = cv2.threshold(B, thres, 255, cv2.THRESH_BINARY)
                image = np.dstack((R_image,G_image,B_image))
                image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
                height, width, channel = image.shape
                bytes_per_line = 3 * width
                q_image = QImage(image.data, width, height, bytes_per_line, QImage.Format.Format_RGB888)
                self.image = q_image
                self.imageLabel.setPixmap(QPixmap.fromImage(q_image))
        self.histogramImage()
```

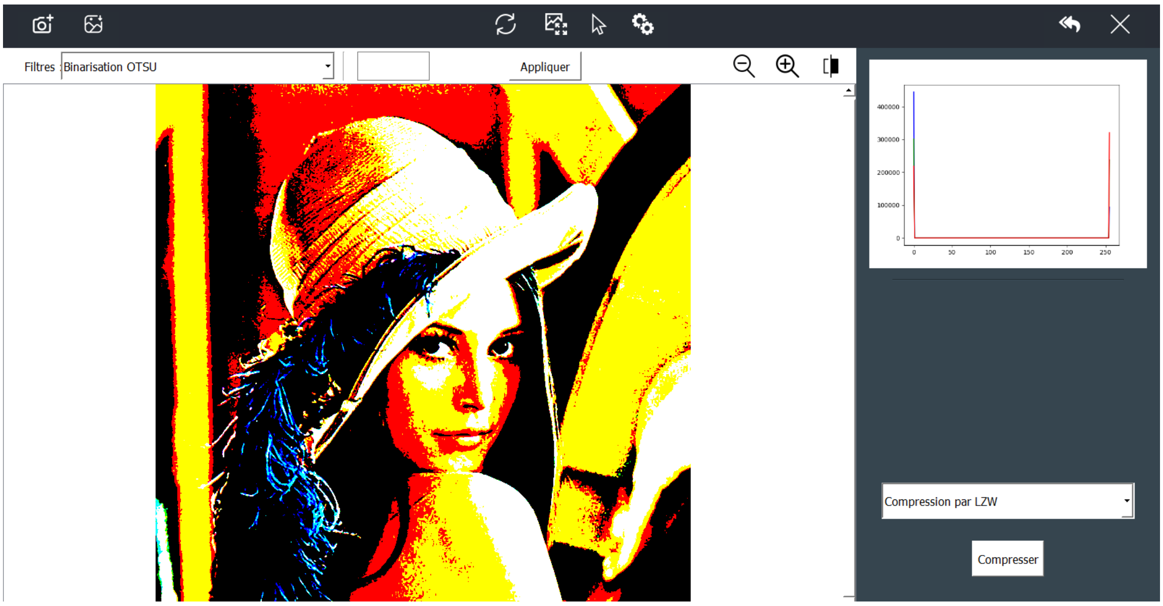
Résultat :



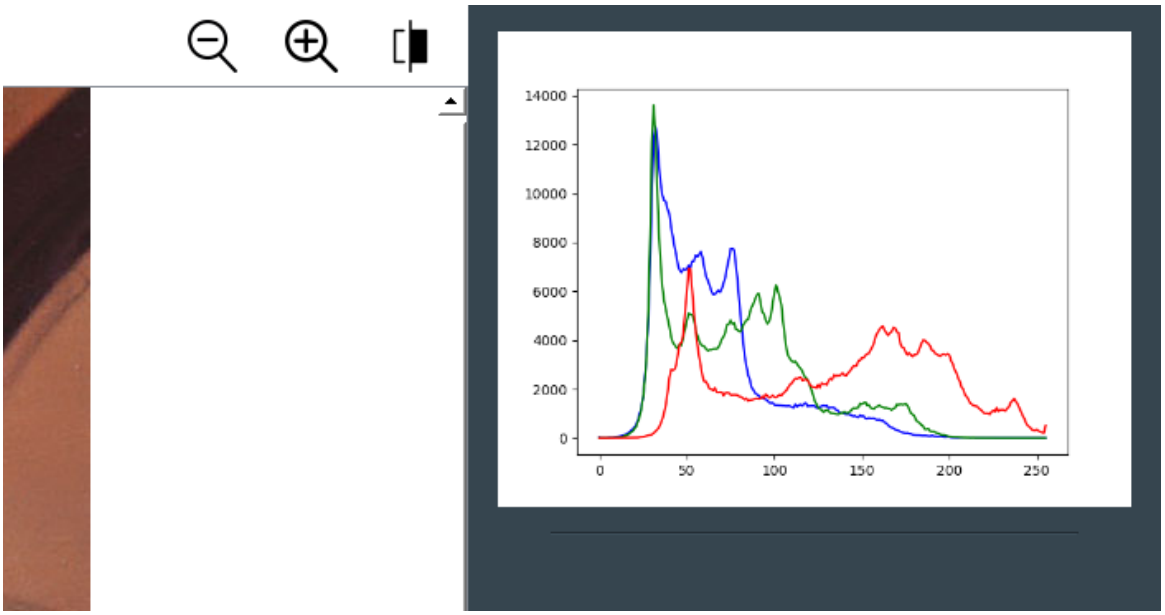
Binarisation OTSU :

```python
def binarisationOSTU(self):
        picVal = self.imageLabel.pixmap()
        pic = picVal.toImage()
        image=self.convertQImageToMat(pic)
        image=cv2.cvtColor(image,cv2.COLOR_BGR2RGB)
        if self.is_grey_scale(image):
                image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
                th, image = cv2.threshold(image, 0, 255, cv2.THRESH_OTSU)
                image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
                height, width, channel = image.shape
                bytes_per_line = 3 * width
                q_image = QImage(image.data, width, height, bytes_per_line, QImage.Format.Format_RGB888)
                self.image = q_image
                self.imageLabel.setPixmap(QPixmap.fromImage(q_image))
        else:
                image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
                R=image[:,:,0]
                G=image[:,:,1]
                B=image[:,:,2]
                _, R_image = cv2.threshold(R, 0, 255, cv2.THRESH_OTSU)
                _, G_image = cv2.threshold(G, 0, 255, cv2.THRESH_OTSU)
                _, B_image = cv2.threshold(B, 0, 255, cv2.THRESH_OTSU)
                image = np.dstack((R_image,G_image,B_image))
                image = cv2.cvtColor(image,cv2.COLOR_BGR2RGB)
                height, width, channel = image.shape
                bytes_per_line = 3 * width
                q_image = QImage(image.data, width, height, bytes_per_line, QImage.Format.Format_RGB888)
                self.image = q_image
                self.imageLabel.setPixmap(QPixmap.fromImage(q_image))
        self.histogramImage()
```

Résultat :



Histogramme d'image :



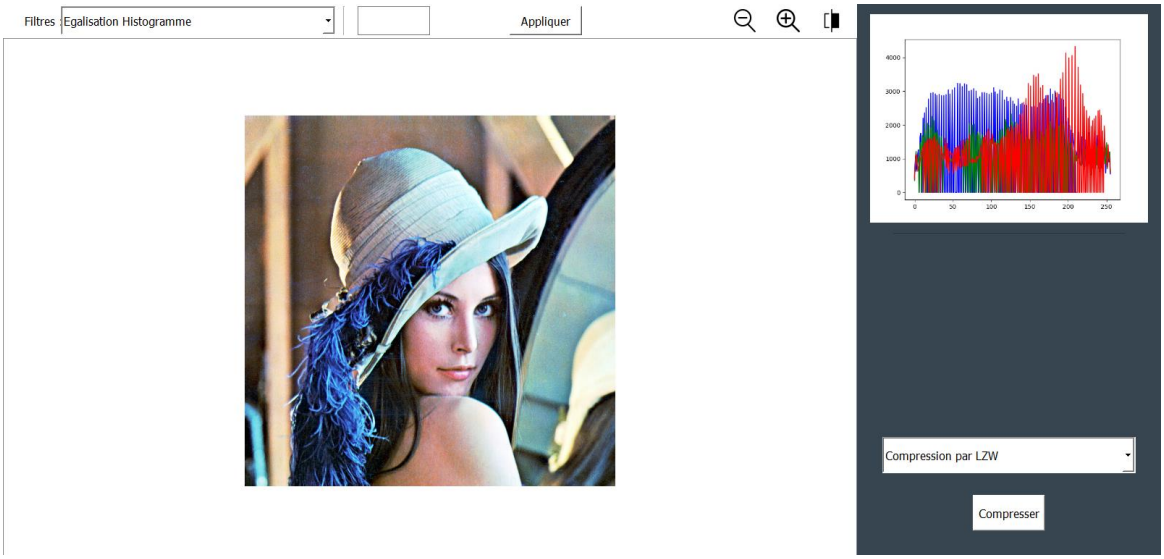
Ici on affiche l'histogramme de l'image qui est affichée, donc si on applique par exemple un filtre l'histogramme change.

```
def histogramImage(self):
        picVal = self.imageLabel.pixmap()
        image = picVal.toImage()
        image=self.convertQImageToMat(image)
        try:
                self.histogrammeLabel.clear()
                os.remove('figure\\fig.png')
        except:pass

        if self.is_grey_scale(image):
                editedImage = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
                hist = cv2.calcHist([editedImage],[0],None,[256],[0,256])
                plt.clf()
                plt.plot(hist)
                print('Lets edite')
                plt.savefig('figure\\fig.png')
                fig=cv2.imread('figure\\fig.png')
        else:
                B=image[:,:,0]
                G=image[:,:,1]
                R=image[:,:,2]
                histB=cv2.calcHist([B],[0],None,[256],[0,256])
                histG=cv2.calcHist([G],[0],None,[256],[0,256])
                histR=cv2.calcHist([R],[0],None,[256],[0,256])
                plt.clf()
                plt.plot(histB,color='blue')
                plt.plot(histG,color='green')
                plt.plot(histR,color='red')
                plt.savefig('figure\\fig.png')
                fig=cv2.imread('figure\\fig.png')

        finalImage = cv2.resize(fig,(0,0),fx=0.6,fy=0.6)

        binary_image = cv2.cvtColor(finalImage, cv2.COLOR_BGR2RGB)
        height, width, channel = binary_image.shape
        bytes_per_line = 3 * width
        q_image = QImage(binary_image.data, width, height, bytes_per_line, QImage.Format.Format_RGB888)
        # img = QPixmap(fileName[0])
        self.histogrammeLabel.setPixmap(QPixmap.fromImage(q_image))
```

Egalisation d'image :

```python
def equalizeHistogramme(self):
        picVal = self.imageLabel.pixmap()
        pic = picVal.toImage()
        image=self.convertQImageToMat(pic)
        if self.is_grey_scale(image):
                image = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
                image= cv2.equalizeHist(image)
                image = cv2.cvtColor(image,cv2.COLOR_BGR2RGB)
                height, width, channel = image.shape
                bytes_per_line = 3 * width
                q_image = QImage(image.data, width, height, bytes_per_line, QImage.Format.Format_RGB888)
                self.image = q_image
                self.imageLabel.setPixmap(QPixmap.fromImage(q_image))
                self.histogramImage()
        else:
                image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
                R=image[:,:,0]
                G=image[:,:,1]
                B=image[:,:,2]
                R_image = cv2.equalizeHist(R)
                G_image = cv2.equalizeHist(G)
                B_image = cv2.equalizeHist(B)
                image = np.dstack((R_image,G_image,B_image))
                image = cv2.cvtColor(image,cv2.COLOR_BGR2RGB)
                height, width, channel = image.shape
                bytes_per_line = 3 * width
                q_image = QImage(image.data, width, height, bytes_per_line, QImage.Format.Format_RGB888)
                self.image = q_image
                self.imageLabel.setPixmap(QPixmap.fromImage(q_image))
                self.histogramImage()
```

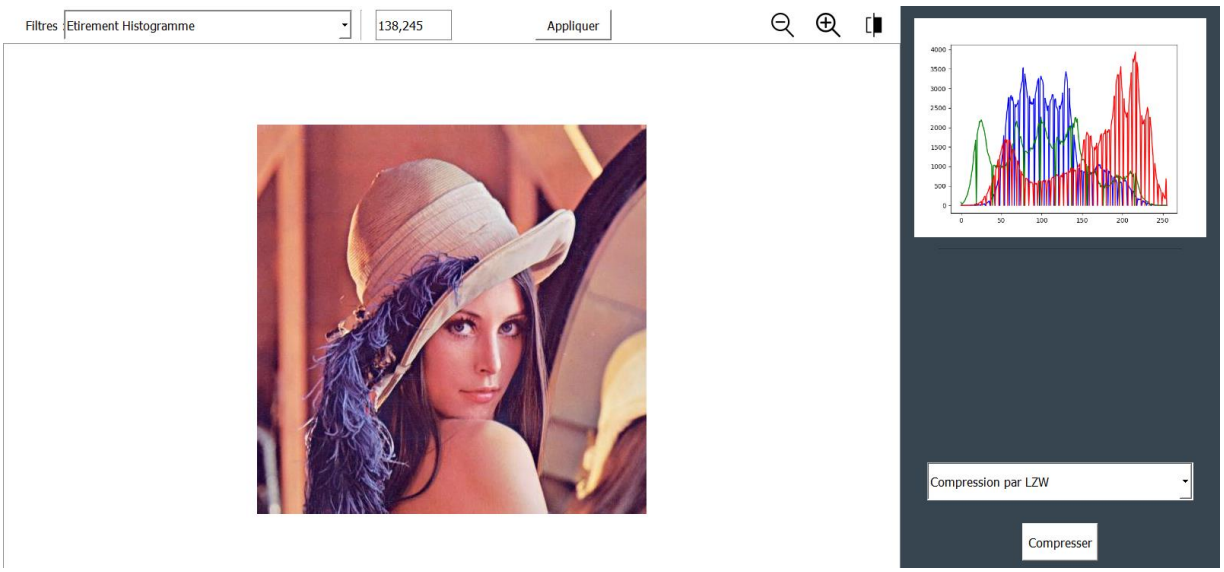
Résultat :



Etirement d'image :

```
def EtirerHistogramme(self):
        picVal = self.imageLabel.pixmap()
        pic = picVal.toImage()
        image=self.convertQImageToMat(pic)
        if self.is_grey_scale(image):
                image = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
                maximum = image.max()
                minimum = image.min()
                x=image.shape[1]
                y=image.shape[0]
                for i in range(x):
                        for j in range(y):
                                image[i][j] = (255 /(maximum-minimum))*(image[i][j]-minimum)

                image = cv2.cvtColor(image,cv2.COLOR_BGR2RGB)
                bytes_per_line = 3 * x
                q_image = QImage(image.data, x, y, bytes_per_line, QImage.Format.Format_RGB888)
                self.image = q_image
                self.imageLabel.setPixmap(QPixmap.fromImage(q_image))

        else:
                image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
                R=image[:,:,0]
                G=image[:,:,1]
                B=image[:,:,2]

                maximumR = R.max()
                minimumR = R.min()
                x=R.shape[1]
                y=R.shape[0]

                maximumG = G.max()
                minimumG = G.min()

                maximumB = B.max()
                minimumB = B.min()
                for i in range(y):
                        for j in range(x):
                                R[i][j] = (255 /(maximumR-minimumR))*(R[i][j]-minimumR)
                                G[i][j] = (255 /(maximumG-minimumG))*(G[i][j]-minimumG)
                                B[i][j] = (255 /(maximumB-minimumB))*(B[i][j]-minimumB)

                image= np.dstack((R,G,B))
                bytes_per_line = 3 * x
                q_image = QImage(image.data, x, y, bytes_per_line, QImage.Format.Format_RGB888)
                self.image = q_image
                self.imageLabel.setPixmap(QPixmap.fromImage(q_image))
        self.histogramImage()
```

Résultat :

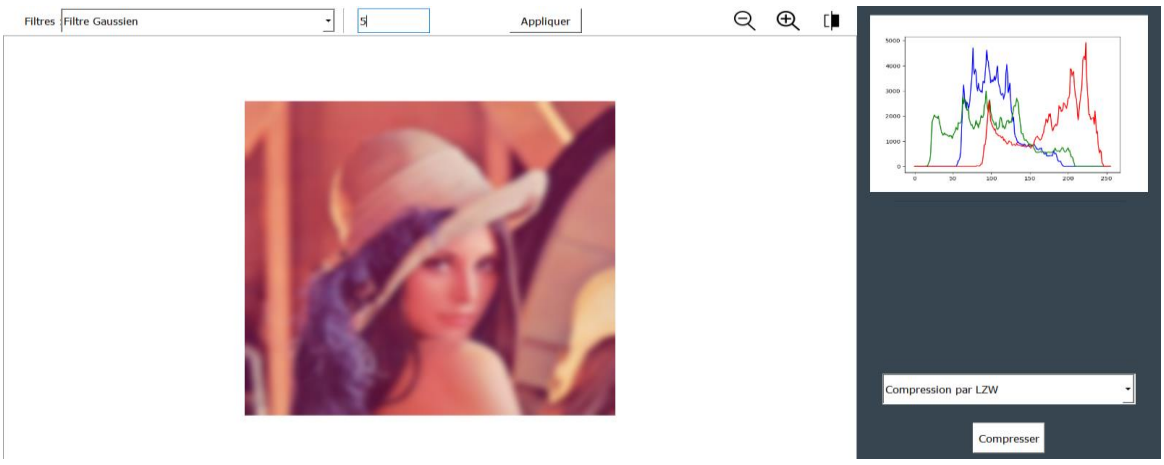


Filtrage d'image :

Gaussien :

```
def filtreGaussien(self):
        picVal = self.imageLabel.pixmap()
        pic = picVal.toImage()
        image=self.convertQImageToMat(pic)
        val = self.parametre.text()
        if val == "":
                sigma = 1.5
        else:
                sigma = float(val)

        if self.is_grey_scale(image):
                image = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
                image_filtered = gaussian_filter(image, sigma)
                image_filtered = image_filtered.astype(np.uint8)
                imageRes = cv2.cvtColor(image_filtered,cv2.COLOR_BGR2RGB)
                height, width, channel = imageRes.shape
                bytes_per_line = 3 * width
                q_image = QImage(imageRes.data, width, height, bytes_per_line, QImage.Format.Format_RGB888)
                self.image = q_image
                self.imageLabel.setPixmap(QPixmap.fromImage(q_image))
                self.histogramImage()
        else:
                images = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
                R=images[:,:,0]
                G=images[:,:,1]
                B=images[:,:,2]
                R_image = gaussian_filter(R, sigma)
                G_image = gaussian_filter(G, sigma)
                B_image = gaussian_filter(B, sigma)
                image = np.dstack((R_image,G_image,B_image))
                imageResultat = image.astype(np.uint8)
                height, width, channel = imageResultat.shape
                bytes_per_line = 3 * width
                q_image = QImage(imageResultat.data, width, height, bytes_per_line, QImage.Format.Format_RGB888)
                self.image = q_image
                self.imageLabel.setPixmap(QPixmap.fromImage(q_image))
                self.histogramImage()
```

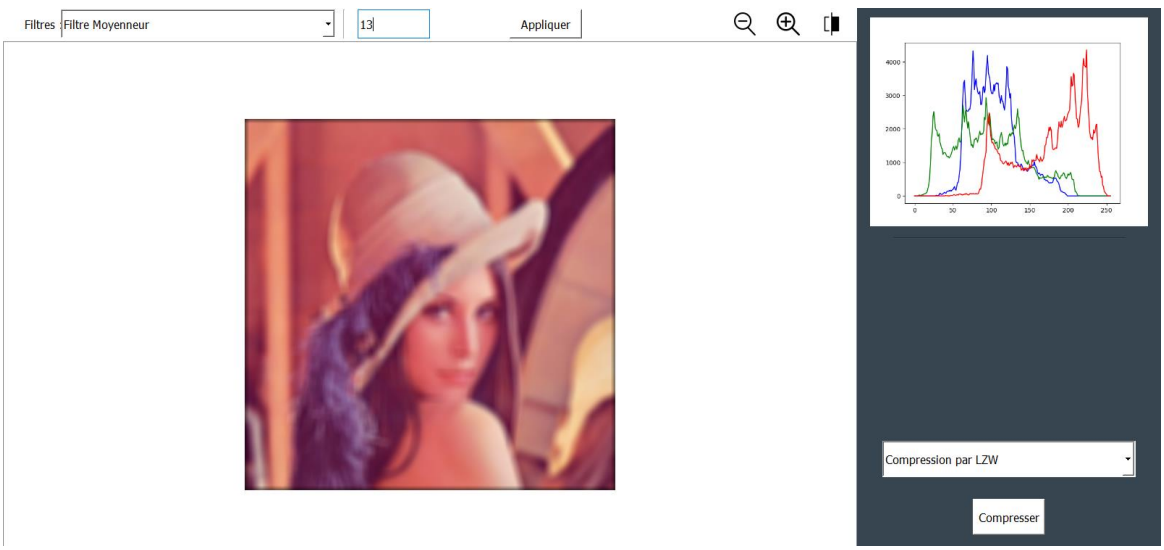
Résultat :



Moyenneur :

```
def filtre_moyenneur(self):
        picVal = self.imageLabel.pixmap()
        pic = picVal.toImage()
        image=self.convertQImageToMat(pic)
        val = self.parametre.text()
        if val == "":
                taille = 3
        else:
                taille = int(val)
        if self.is_grey_scale(image):
                images = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
                imageRes = self.filt_moy(images,taille)
                imageRes = imageRes.astype(np.uint8)
                imageRes = cv2.cvtColor(imageRes,cv2.COLOR_BGR2RGB)
                height, width, channel = imageRes.shape
                bytes_per_line = 3 * width
                q_image = QImage(imageRes.data, width, height, bytes_per_line, QImage.Format.Format_RGB888)
                self.image = q_image
                self.imageLabel.setPixmap(QPixmap.fromImage(q_image))
                self.histogramImage()

        else:
                images = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
                R=images[:,:,0]
                G=images[:,:,1]
                B=images[:,:,2]
                R_image = self.filt_moy(R,taille)
                G_image = self.filt_moy(G,taille)
                B_image = self.filt_moy(B,taille)
                image = np.dstack((R_image,G_image,B_image))
                imageResultat = image.astype(np.uint8)
                height, width, channel = imageResultat.shape
                bytes_per_line = 3 * width
                q_image = QImage(imageResultat.data, width, height, bytes_per_line, QImage.Format.Format_RGB888)
                self.image = q_image
                self.imageLabel.setPixmap(QPixmap.fromImage(q_image))
                self.histogramImage()
```

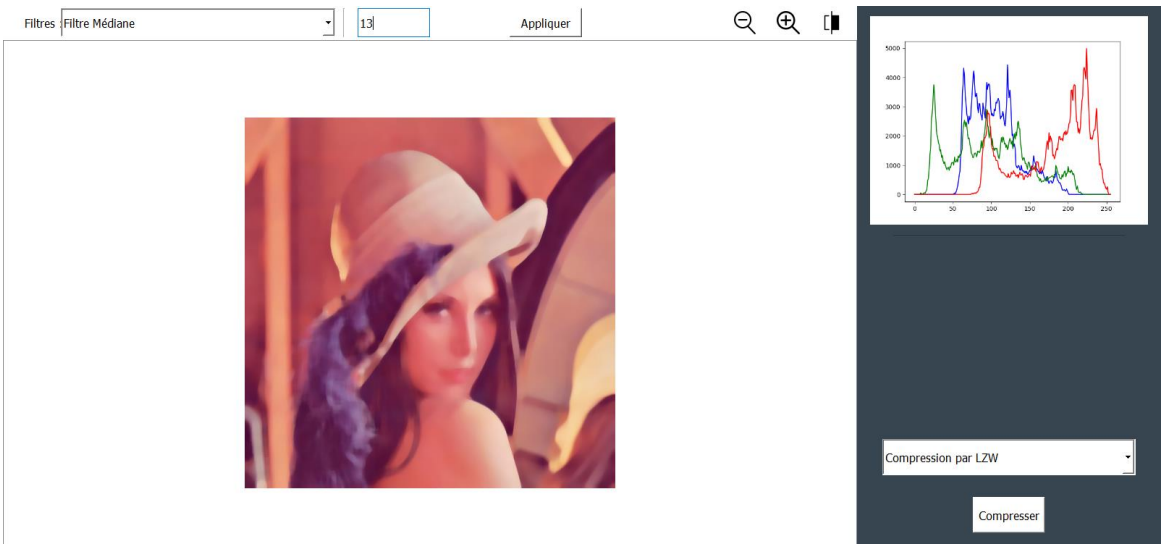
Résultat :



Médian :

```
def filtreMediane(self):
        picVal = self.imageLabel.pixmap()
        pic = picVal.toImage()
        image=self.convertQImageToMat(pic)
        val = self.parametre.text()
        if val == "":
                size = 3
        else:
                size = int(val)

        if self.is_grey_scale(image):
                image = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
                image_filtered = median_filter(image, size)
                image_filtered = image_filtered.astype(np.uint8)
                imageRes = cv2.cvtColor(image_filtered,cv2.COLOR_BGR2RGB)
                height, width, channel = imageRes.shape
                bytes_per_line = 3 * width
                q_image = QImage(imageRes.data, width, height, bytes_per_line, QImage.Format.Format_RGB888)
                self.image = q_image
                self.imageLabel.setPixmap(QPixmap.fromImage(q_image))
                self.histogramImage()
        else:
                images = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
                R=images[:,:,0]
                G=images[:,:,1]
                B=images[:,:,2]
                R_image = median_filter(R, size)
                G_image = median_filter(G, size)
                B_image = median_filter(B, size)
                image = np.dstack((R_image,G_image,B_image))
                imageResultat = image.astype(np.uint8)
                height, width, channel = imageResultat.shape
                bytes_per_line = 3 * width
                q_image = QImage(imageResultat.data, width, height, bytes_per_line, QImage.Format.Format_RGB888)
                self.image = q_image
                self.imageLabel.setPixmap(QPixmap.fromImage(q_image))
                self.histogramImage()
```

Résultat :



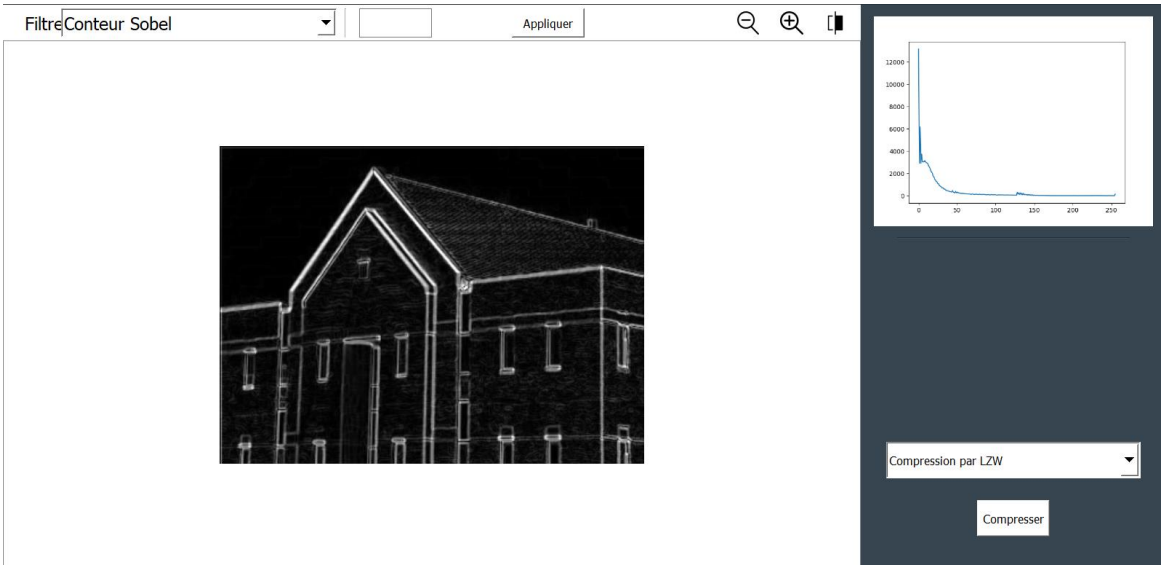
Extraction des contours :

```python
def conteur(self):
        picVal = self.imageLabel.pixmap()
        pic = picVal.toImage()
        image=self.convertQImageToMat(pic)
        image = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
        conteur = self.listFonction.currentText()

        if conteur == "Conteur Sobel":
                imagex = cv2.Sobel(image, cv2.CV_64F, 1, 0, ksize = 3)
                imgx = cv2.convertScaleAbs(imagex)
                imagey = cv2.Sobel(image, cv2.CV_64F, 0, 1, ksize = 3)
                imgy = cv2.convertScaleAbs(imagey)
                imageResultant = cv2.addWeighted(imgx,0.5,imgy,0.5,0)
        elif conteur == "Conteur Robert":
                robert_x = cv2.filter2D(image,-1, np.array([[-1, 0], [0, 1]], dtype=np.float32))
                robert_y = cv2.filter2D(image,-1, np.array([[0, -1], [1, 0]], dtype=np.float32))
                imgx = cv2.convertScaleAbs(robert_x)
                imgy = cv2.convertScaleAbs(robert_y)
                imageResultant = cv2.addWeighted(imgx,0.5,imgy,0.5,0)
        elif conteur == "Conteur Laplacien":
                imageResultant = cv2.Laplacian(image, cv2.CV_8U)
        elif conteur =="Conteur Gradient":
                kernelSize=3
                threshold= 100
                sobelx = cv2.Sobel(image, cv2.CV_64F, 1, 0, kernelSize)
                sobely = cv2.Sobel(image, cv2.CV_64F, 0, 1, kernelSize)
                mag, angle = cv2.cartToPolar(sobelx, sobely, angleInDegrees=True)

                thresholded = cv2.threshold(mag, 126, 255, cv2.THRESH_BINARY)[1]
                imageResultant = cv2.cvtColor(cv2.convertScaleAbs(thresholded),cv2.COLOR_BGR2RGB)

        imageRes = cv2.cvtColor(imageResultant,cv2.COLOR_BGR2RGB)
        height, width, channel = imageRes.shape
        bytes_per_line = 3 * width
        q_image = QImage(imageRes.data, width, height, bytes_per_line, QImage.Format.Format_RGB888)
        self.image = q_image
        self.imageLabel.setPixmap(QPixmap.fromImage(q_image))
        self.histogramImage()
```

Photo de test :

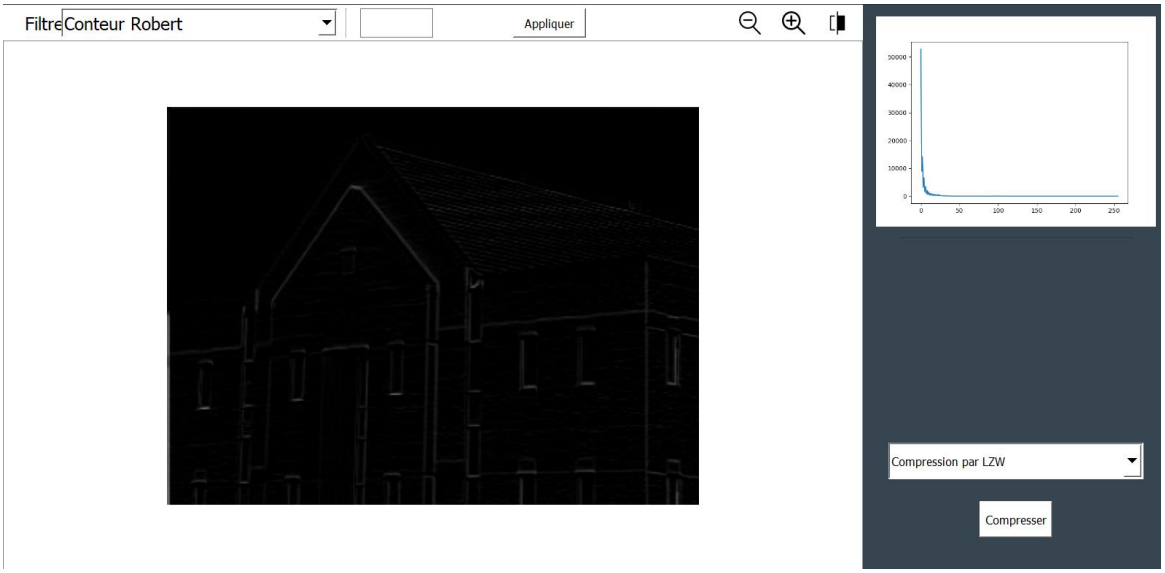
Sobel :

Résultat :



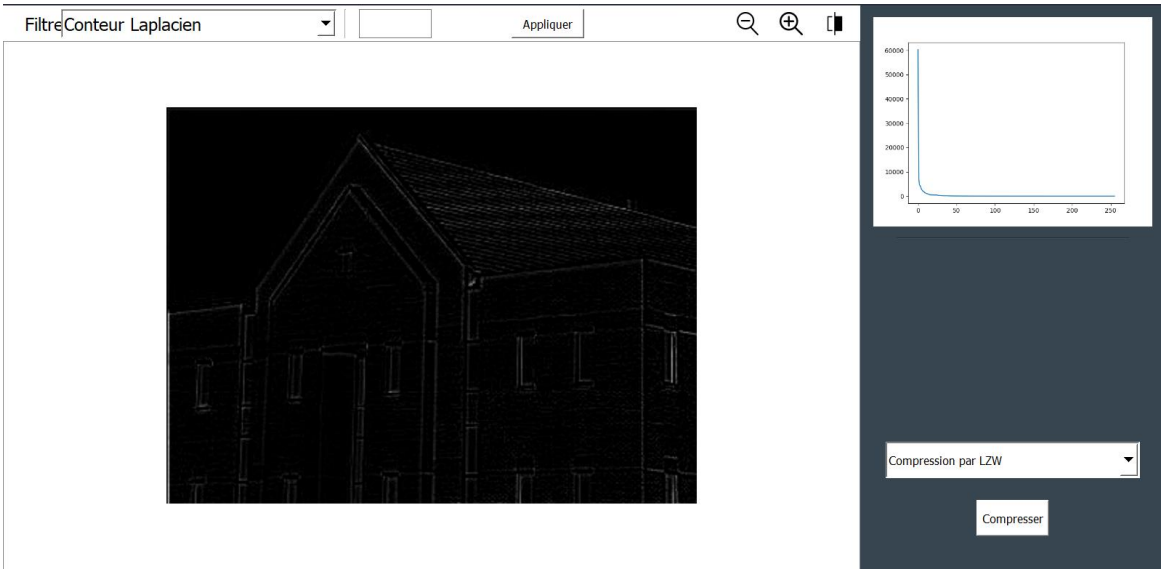
Robert :

Résultat :



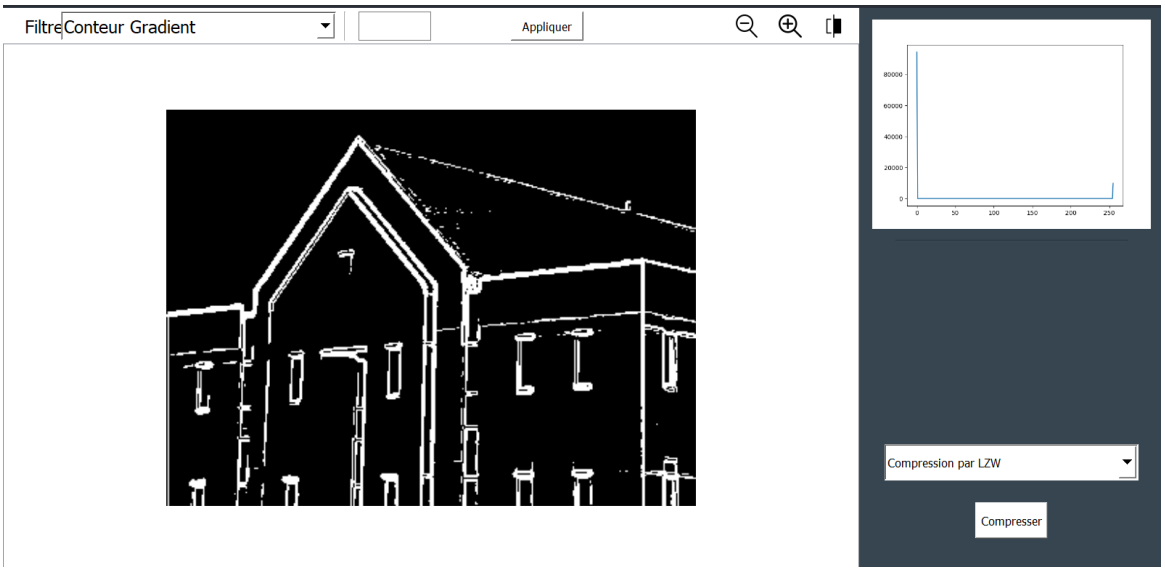
Laplacien :

Résultat :



Gradient :

Résultat :



Morphologie mathématique :

Image de test :

```
def Morphologie(self):
        size = self.parametre.text()
        if size != "":
                size = size.split(",")
                x = int(size[0])
                y = int(size[1])
        else:
                x = y = 3
        kernelType = self.morpho.currentText()
        if kernelType == "Rectangle":
                kernel = cv2.getStructuringElement(cv2.MORPH_RECT,ksize=(x,y))
        else:
                kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,ksize=(x,y))

        morphologie = self.listFonction.currentText()

        picVal = self.imageLabel.pixmap()
        pic = picVal.toImage()
        image=self.convertQImageToMat(pic)
        image = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)

        if morphologie == "Erosion":
                imageResultant =cv2.erode(image,kernel)
        elif morphologie =="Dilatation":
                imageResultant =cv2.dilate(image,kernel)
        elif morphologie == "Ouverture":
                imageResultant = cv2.morphologyEx(image, cv2.MORPH_OPEN, kernel)
        elif morphologie == "Fermeture":
                imageResultant = cv2.morphologyEx(image, cv2.MORPH_CLOSE, kernel)
        elif morphologie == "Filtrage Morphologique":
                imageResultant = cv2.morphologyEx(image, cv2.MORPH_OPEN, kernel)
                imageResultant = cv2.morphologyEx(imageResultant, cv2.MORPH_CLOSE, kernel)
```

Erosion :

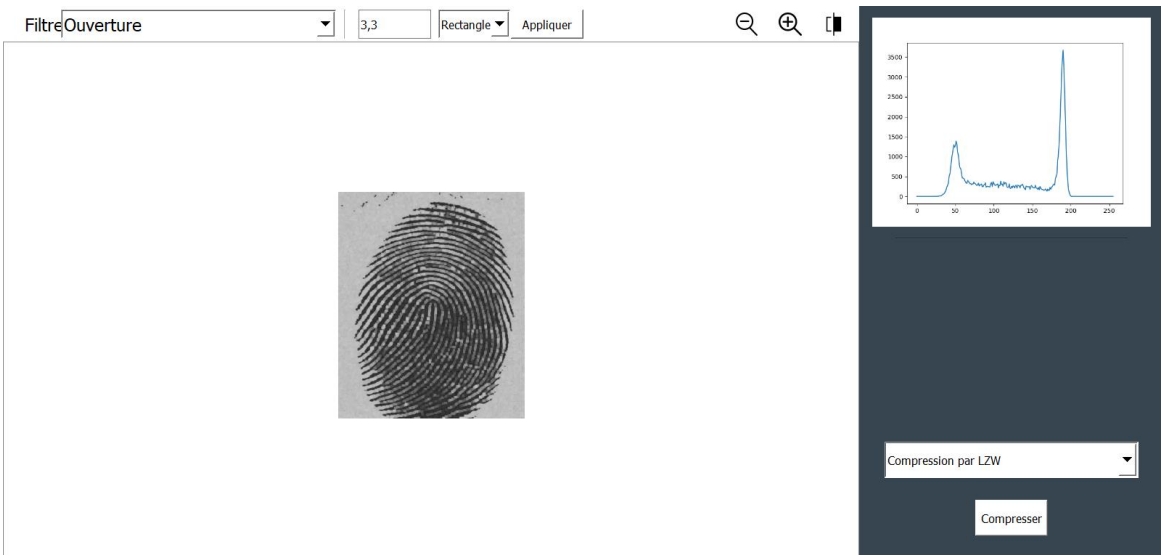
Résultat :

Dilatation :

Résultat :

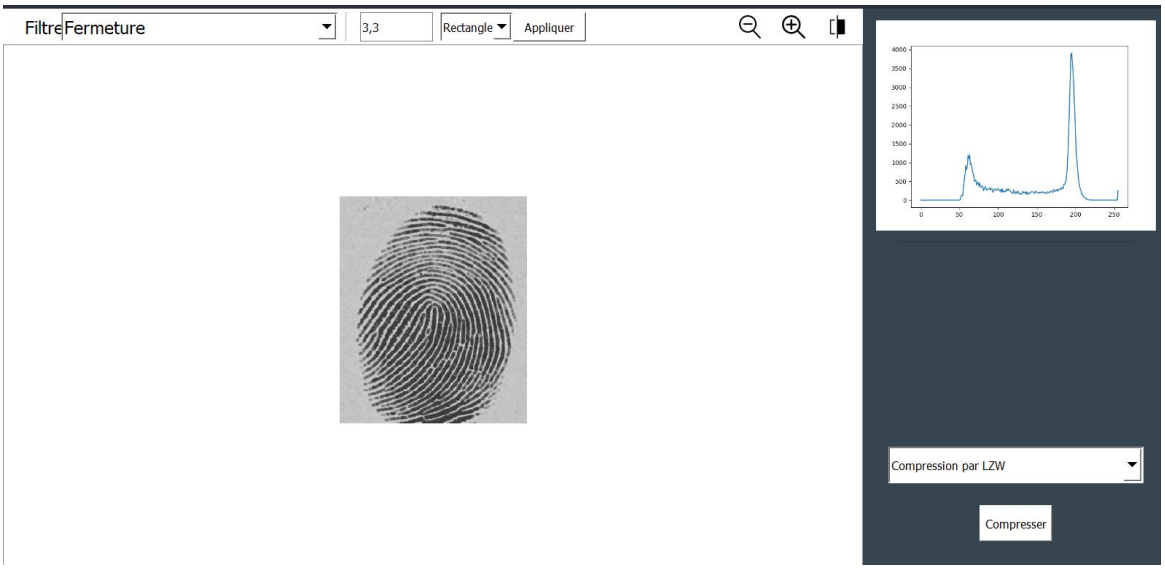
Ouverture :

Résultat :



Fermeture :

Résultat :



Filtrage morphologique :

Résultat :

Dans chaque exemple on a manuellement saisit la taille d'élément structurant et sa forme (rectangle ou ellipse).

Segmentation d'image :

Image de test :

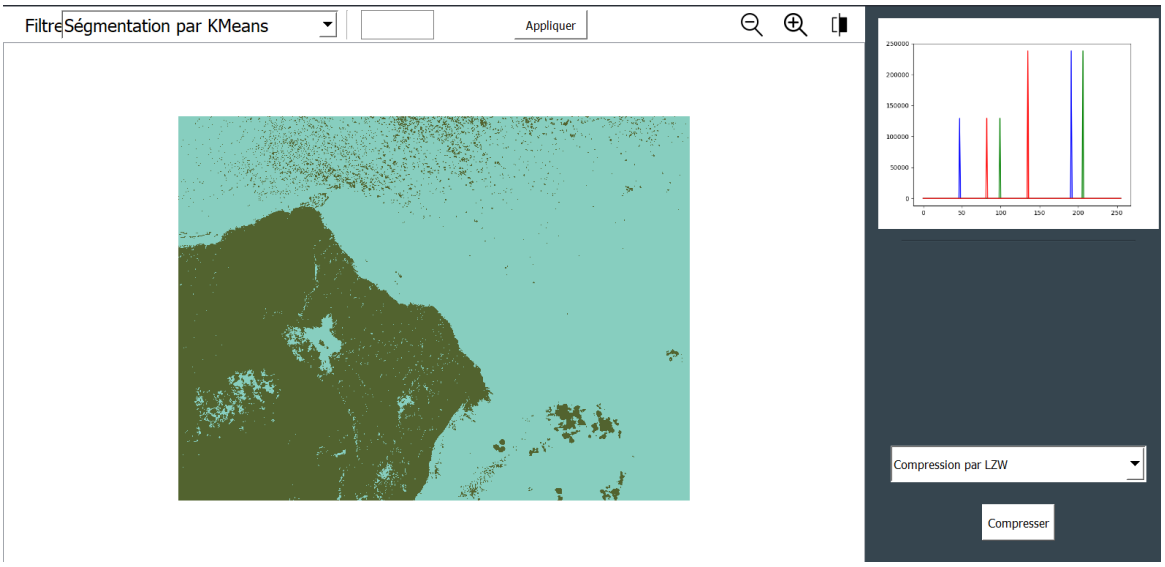
Méthode des k-means :

```
def segmentations(self):
        picVal = self.imageLabel.pixmap()
        pic = picVal.toImage()
        image=self.convertQImageToMat(pic)
        image = cv2.cvtColor(image,cv2.COLOR_BGR2RGB)

        pixel_values = image.reshape(-1,3)

        kmeans = KMeans(n_clusters=3)
        kmeans.fit(pixel_values)

        labels = kmeans.labels_
        centers = kmeans.cluster_centers_

        segmented_image = centers[labels].reshape(image.shape)

        segmented_image = segmented_image.astype(np.uint8)
        height, width, channel = segmented_image.shape
        bytes_per_line = 3 * width
        q_image = QImage(segmented_image.data, width, height, bytes_per_line, QImage.Format.Format_RGB888)
        self.image = q_image
        self.imageLabel.setPixmap(QPixmap.fromImage(q_image))
        self.histogramImage()
```

Résultat :

K=3 :

K=2 :



Croissance de régions D :

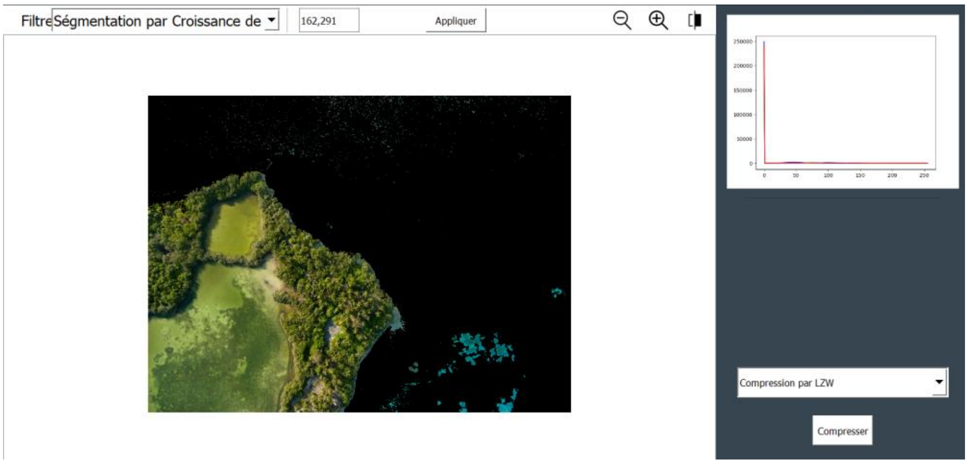
```
threshold = 100
# Ajout du point de départ à la file
fil.append(seed)

# Boucle principale de croissance de région
while len(fil) > 0:
        # Obtention du point suivant dans la file
        current_point = fil.pop(0)

        # Obtention des coordonnées du point
        x, y = current_point

        # Vérification de la validité des coordonnées
        if x < 0 or y < 0 or x >= height or y >= width:
                continue

        # Vérification si le pixel a déjà été visité
        if out_img[x][y] > 0:
                continue

        # Vérification de la différence de valeur de pixel
        if abs(int(image[x][y]) - int(image[seed])) > threshold:
                continue

        # Ajout du pixel à la région
        out_img[x][y] = 255

        fil.append((x - 1, y))
        fil.append((x + 1, y))
        fil.append((x, y - 1))
        fil.append((x, y + 1))
inverted = cv2.bitwise_not(out_img)
result = cv2.bitwise_and(img,img, mask= inverted)
out_img = cv2.cvtColor(result,cv2.COLOR_BGR2RGB)
```

Pour le seed on clique sur la région qu'on veut de l'image pour le choisir.

```
elif value == "Ségmentation par Croissance de régions D":
        self.imageLabel.mousePressEvent = self.getPixel
```

```
def getPixel(self, event):
        picVal = self.imageLabel.pixmap()
        pic = picVal.toImage()
        image=self.convertQImageToMat(pic)
        x = event.pos().x()
        y = event.pos().y()
        x = int(x * image.shape[1] / self.imageLabel.width())
        y = int(y * image.shape[0] / self.imageLabel.height())

        self.parametre.setText(str(x)+","+str(y))
```

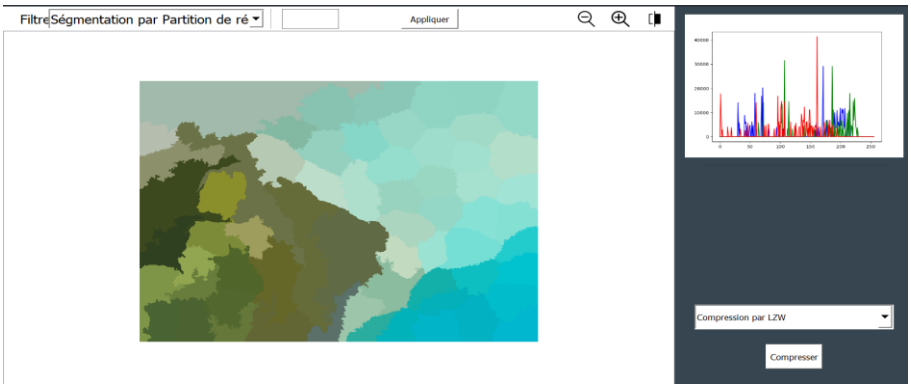
Résultat :



Partition des régions D :

```
def segmentationsParPartition(self):
        picVal = self.imageLabel.pixmap()
        pic = picVal.toImage()
        image=self.convertQImageToMat(pic)
        image = cv2.cvtColor(image,cv2.COLOR_BGR2RGB)
        # Segmenter l'image en utilisant l'algorithme SLIC
        segments = slic(image, n_segments=100, compactness=10)

        # Colorier les régions segmentées
        segmented_image = label2rgb(segments, image, kind='avg')

        height, width, channel = segmented_image.shape
        bytes_per_line = 3 * width
        q_image = QImage(segmented_image.data, width, height, bytes_per_line, QImage.Format.Format_RGB888)
        self.image = q_image
        self.imageLabel.setPixmap(QPixmap.fromImage(q_image))
        self.histogramImage()
```

Résultat :



Détection des points d'intérêts :

```
if pointsInteretAlgo == "Coins de Harris":
        dst = cv2.cornerHarris(gray, 2, 3, 0.04)
        dst = cv2.dilate(dst, None)
        image[dst > 0.01 * dst.max()] = [0, 0, 255]
elif pointsInteretAlgo == "Coins de Shi-Tomasi":
        # Détecter les coins de Shi-Tomasi
        corners = cv2.goodFeaturesToTrack(gray, 25, 0.01, 10)
        corners = np.int0(corners)
        for i in corners:
                x, y = i.ravel()
                cv2.circle(image, (x, y), 3, (0, 255, 0), -1)
elif pointsInteretAlgo == "SIFT":
        sift = cv2.SIFT_create()

        keypoints = sift.detect(image, None)

        image = cv2.drawKeypoints(image, keypoints, None, flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
else:
        edges = cv2.Canny(gray, 50, 150, apertureSize=3)

        lines = cv2.HoughLines(edges, 1, np.pi/180, 200)

        for line in lines:
        rho, theta = line[0]
        a = np.cos(theta)
        b = np.sin(theta)
        x0 = a*rho
        y0 = b*rho
        x1 = int(x0 + 1000*(-b))
        y1 = int(y0 + 1000*(a))
        x2 = int(x0 - 1000*(-b))
        y2 = int(y0 - 1000*(a))
        cv2.line(image, (x1, y1), (x2, y2), (0, 0, 255), 2)

imageRes = cv2.cvtColor(image,cv2.COLOR_BGR2RGB)
```

Image de test :

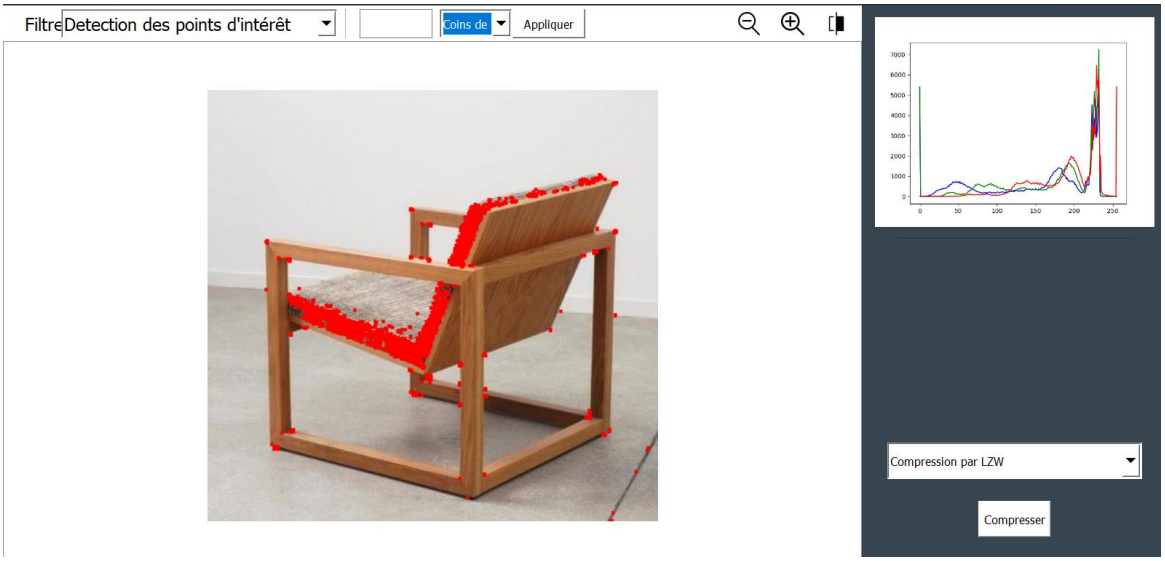
SIFT :

Résultat :

Les cercles sont utilisés pour représenter les points clés détectés dans une image. Chaque cercle correspond à un point clé, et sa taille et sa position représentent l'échelle et la position du point clé dans l'image. Le centre du cercle est situé aux coordonnées (x, y) du point clé, et son rayon est proportionnel à l'échelle du point clé.

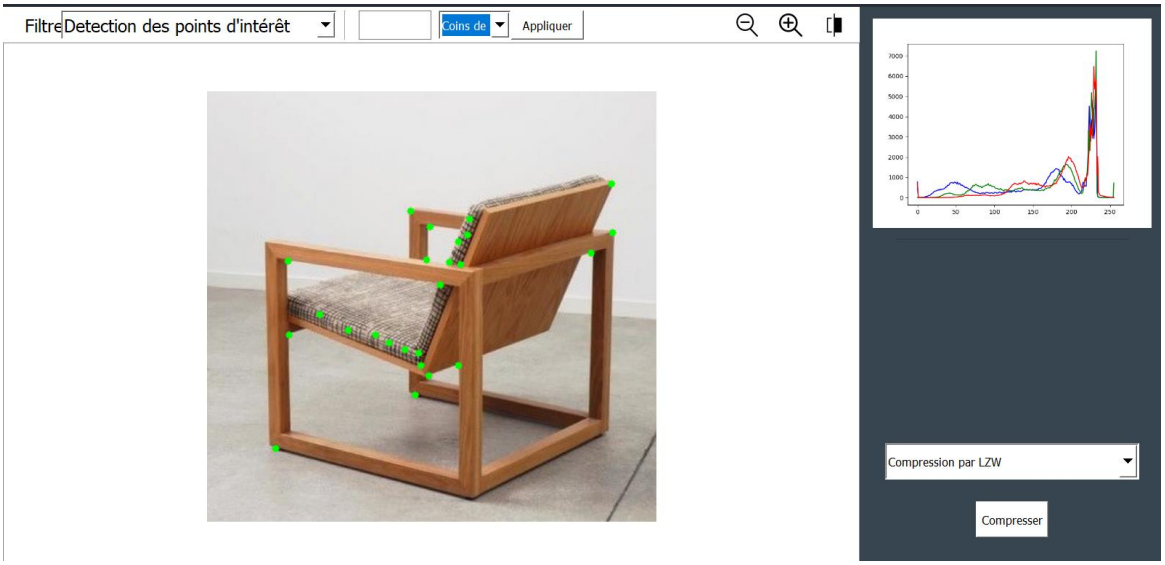
Coins de Harris :

Résultat :



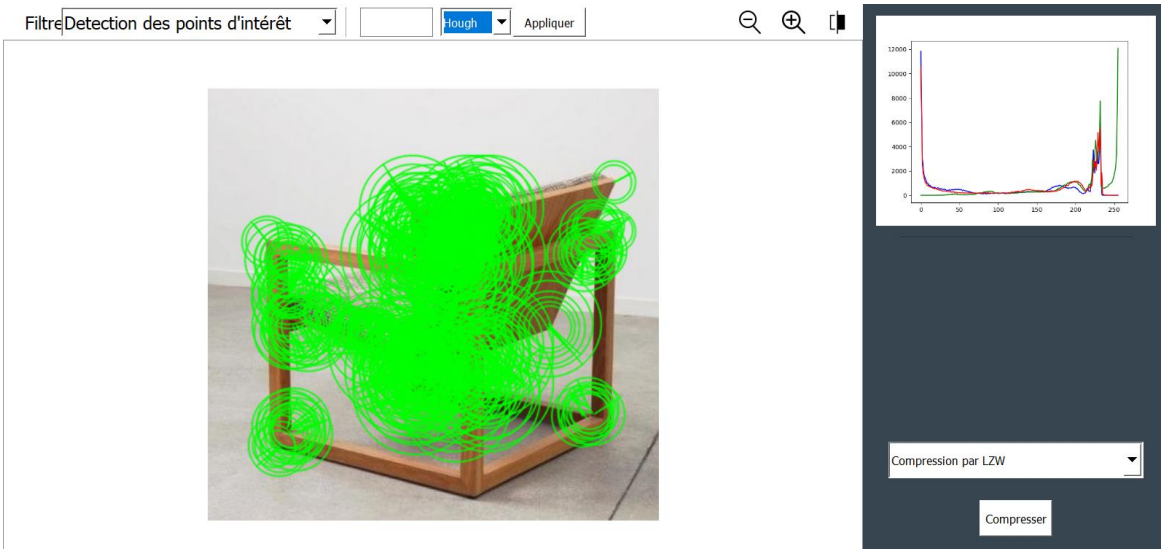
Coins de Shi-Tomasi :

Résultat :



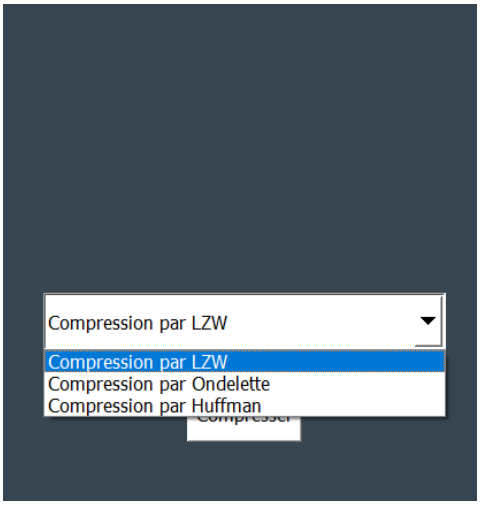
Hough :

Résultat :



Compression :

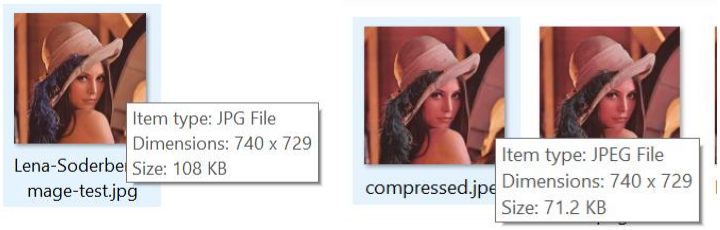
```
        pic = picVal.toImage()
        image=self.convertQImageToMat(pic)
        compression = self.compressBox.currentText()
        if compression == "Compression par Ondelette":
                coeffs = pywt.dwt2(image, 'haar')
                LL, (LH, HL, HH) = coeffs
                threshold = 30
                LH[np.abs(LH) < threshold] = 0
                HL[np.abs(HL) < threshold] = 0
                HH[np.abs(HH) < threshold] = 0
                reconstructed = pywt.idwt2((LL, (LH, HL, HH)), 'haar')
                reconstructed = np.uint8(reconstructed)
                imageRes = cv2.cvtColor(reconstructed,cv2.COLOR_BGR2RGB)

        elif compression == "Compression par LZW":
                compressor = LZW(image)
                compressor.compress()
                decompress = LZW(os.path.join("CompressedFiles",image+"Compressed.lzw"))
                decompress.decompress()
                imageCompressed = cv2.imread(os.path.join("DecompressedFiles",image+"Decompressed.tif"))
                imageRes = cv2.cvtColor(imageCompressed,cv2.COLOR_BGR2RGB)
        else:
                if len(image.shape) == 3:
                        b,g,r = cv2.split(image)
                        encoded_b, code = self.encodeHuffman(b)
                        decoded_b = self.decodeHuffman(encoded_b, code, b.shape)
                        encoded_g, code = self.encodeHuffman(g)
                        decoded_g = self.decodeHuffman(encoded_g, code, g.shape)
                        encoded_r, code = self.encodeHuffman(r)
                        decoded_r = self.decodeHuffman(encoded_r, code, r.shape)

                        decoded_image = cv2.merge((decoded_b, decoded_g, decoded_r))
                else:
                        encoded_image, code = self.encodeHuffman(b)
                        decoded_image = self.decodeHuffman(encoded_image, code, img.shape)

                imageRes = cv2.cvtColor(decoded_image,cv2.COLOR_BGR2RGB)
```



Huffman :

```python
def encodeHuffman(self, image):
        freq = defaultdict(int)
        for i in range(image.shape[0]):
                for j in range(image.shape[1]):
                        freq[image[i,j]] += 1
        heap = [[wt, [sym, ""]] for sym, wt in freq.items()]
        heapq.heapify(heap)
        while len(heap) > 1:
                lo = heapq.heappop(heap)
                hi = heapq.heappop(heap)
                for pair in lo[1:]:
                        pair[1] = '0' + pair[1]
                for pair in hi[1:]:
                        pair[1] = '1' + pair[1]
                heapq.heappush(heap, [lo[0] + hi[0]] + lo[1:] + hi[1:])
        code = dict(heapq.heappop(heap)[1:])
        encoded_image = []
        for i in range(image.shape[0]):
                for j in range(image.shape[1]):
                        encoded_image.append(code[image[i,j]])
        return encoded_image, code

def decodeHuffman(self, encoded_image, code, shape):
        inv_code = {v: k for k, v in code.items()}
        decoded_image = []
        current_code = ''
        for code in encoded_image:
                current_code += code
                if current_code in inv_code:
                        decoded_image.append(inv_code[current_code])
                        current_code = ''
        decoded_image = np.array(decoded_image).reshape(shape)
        return decoded_image
```

Résultat :



Ondelette :

Résultat :