



Rapport projet Traitement d'image

Analyse d'image ~ Saïda Bouakaz

2024/2025

Chouati Linda & Faussurier Marc

<b>I. Introduction.....</b>	<b>2</b>
<b>II. Filtrage médian.....</b>	<b>2</b>
<b>III. Convolution.....</b>	<b>3</b>
A. Convolution générique.....	3
B. Filtrage moyenneur par convolution.....	4
C. Généralisation de la convolution.....	4
1. Gauss.....	4
2. Laplacien.....	5
3. Sobel.....	5
4. Passe haut.....	6
<b>IV. Transformation d'histogramme et LUT.....</b>	<b>7</b>
<b>VII. Transformation géométrique.....</b>	<b>9</b>
<b>VIII. Interface graphique.....</b>	<b>10</b>

# I. Introduction

L'analyse d'image joue aujourd'hui un rôle essentiel dans de nombreux secteurs, allant de la médecine à la sécurité, en passant par les transports et la vision par ordinateur. Les avancées technologiques récentes permettent non seulement de traiter les images, mais aussi d'identifier automatiquement des objets et de décrire les interactions présentes sur une image, comme le font les modèles d'intelligence artificielle modernes.

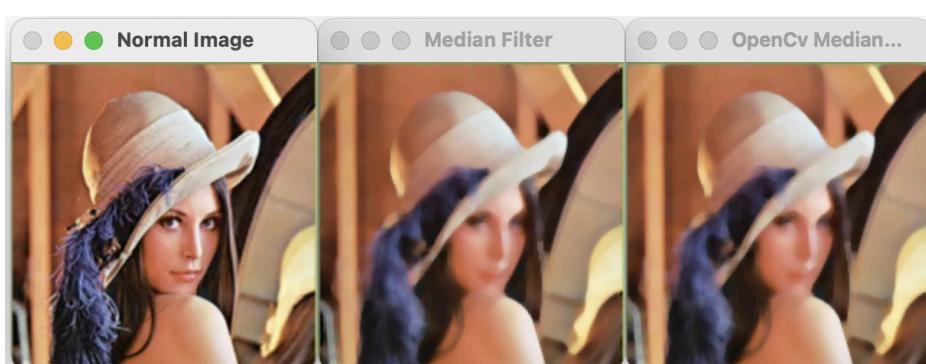
Ce rapport présente ainsi les travaux réalisés dans le cadre du premier projet de traitement d'image, dont l'objectif est de mettre en pratique les concepts fondamentaux du traitement d'images en C++. À travers ce projet, nous avons implémenté plusieurs algorithmes pour le lissage d'image, la détection des contours et l'amélioration d'images.

Pour évaluer nos implémentations, nous avons comparé les résultats obtenus avec ceux fournis par OpenCV, une bibliothèque spécialisée dans le traitement d'images.

## II. Filtrage médian

Le filtrage médian est une méthode non linéaire largement utilisée pour atténuer le bruit impulsif dans les images, souvent désigné comme bruit poivre et sel. Son fonctionnement repose sur un principe simple : la valeur d'un pixel est remplacée par la médiane des intensités des pixels voisins situés dans une fenêtre carrée de taille définie.

Dans notre approche, nous avons d'abord appliqué ce filtre aux images en niveaux de gris. Pour chaque pixel, les valeurs de ses voisins sont extraites, triées, puis la médiane est calculée afin de remplacer l'intensité initiale du pixel central. Pour les images en couleur, nous avons procédé en traitant séparément les trois canaux de l'image (Rouge, Vert et Bleu) étant donné qu'on ne savait pas à quel point on pouvait utiliser openCV. Après avoir appliqué le filtre médian sur chaque canal, on a combiné les résultats en un seul canal pour reconstituer l'image finale. Afin de valider notre implémentation, nous avons comparé les résultats obtenus avec ceux de la fonction "medianBlur" fournie par OpenCV.



Ainsi, au centre, nous avons à une image appliquée notre filtre médian avec un noyau de taille 5\*5. On observe une réduction du bruit avec notamment un lissage des zones unies, tels

que la peau et le chapeau mais tout en conservant les contours importants de l'image.

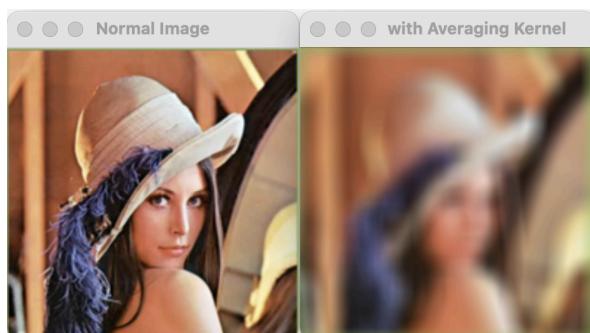
À droite, on retrouve le résultat de l'image avec le filtre médian d'openCV, et celui-ci semble similaire à notre implémentation.

### III. Convolution

#### A. Convolution générique

Nous avons implémenté une fonction générique de convolution capable de traiter à la fois les images en niveaux de gris et les images en couleur, comme précédemment. Le principe de la convolution repose sur l'application d'un noyau à chaque pixel de l'image où les valeurs des pixels voisins sont multipliées par les coefficients du noyau, puis additionnées afin de calculer la nouvelle valeur du pixel.

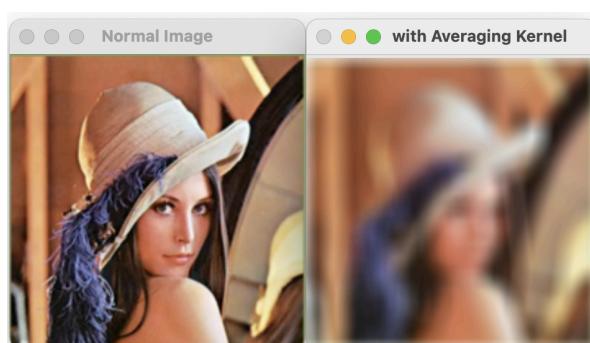
Un des défis de la réalisation de cette fonction générique de convolution a été de penser à comment gérer le bord des images pendant la convolution. Ainsi, on a testé plusieurs méthodes dont la duplication des pixels avec l'extension des valeurs des bords, le padding noir qui consiste de mettre les pixels à 0 contrairement au padding blanc où la valeur sont mises à 255 et sinon à tous simplement ignorer les bords.



1ere méthode : `duplicate the pixel`



2eme méthode : `black padding (pixel=0)`



3eme méthode : `white padding (pixel=255)`



4eme méthode : `ignore the edges`

On remarque bien que la méthode de duplication de pixels montre un meilleur résultat. C'est bien d'ailleurs cette méthode qui est très souvent le plus adaptée pour conserver une transition naturelle aux bords tout en assurant un filtrage uniforme. Les autres méthodes (padding noir/blanc ou ignorer les bords) introduisent des artefacts visibles (noir ou blanc

pour les bords). Ainsi, lors de la suite de ce projet, nous avons gardé cette même technique aussi.

## B. Filtrage moyenneur par convolution

Le filtrage moyenneur est une technique de lissage d'image visant à réduire le bruit en remplaçant la valeur d'un pixel par la moyenne des valeurs de ses voisins dans une fenêtre carrée (noyau). Dans notre implémentation, le noyau est paramétrable et chaque élément reçoit une valeur identique afin de réaliser une moyenne uniforme des pixels voisins.



Au centre, on retrouve donc une image appliquée avec un filtre moyenneur de taille 25\*25. L'image est nettement lissée, avec une réduction du bruit mais également une perte de netteté sur les contours et les détails fins. En effet, on

a remarqué que plus on augmentait la taille du noyau, plus le lissage est marqué mais cela entraîne une perte importante de détails de l'image, la rendant même floue. Par rapport, au résultat obtenu avec openCV, on obtient des résultats similaires.

## C. Généralisation de la convolution

### 1. Gauss

Le filtre gaussien est lui aussi un filtre linéaire souvent utilisé pour lisser une image et réduire le bruit. Mais contrairement au filtre moyenneur, il applique une moyenne pondérée où les pixels proches du centre du noyau ont une influence plus forte que ceux situés aux bords. Cette pondération est définie par une fonction gaussienne qui dépend de la taille du noyau et de l'écart-type (sigma). On a donc implémenté une fonction qui génère un noyau symétrique, où chaque coefficient est calculé selon la formule :  $\exp(-(i^*i + j^*j) / (2 * \text{sigma} * \text{sigma}))$ .



Ainsi, au centre on a appliqué notre filtre gaussien avec un noyau de 25\*25 et un alpha de 4, cela rend l'image plus lisse avec une réduction du bruit tout en conservant une transition douce entre les zones. Ici aussi, l'implémentation du

filtre de gauss avec openCv montre des résultats similaires avec notre propre implémentation du filtre gaussien. Bien plus, on a testé avec plusieurs tailles du noyau et de

la valeur du sigma et on remarque que ces deux paramètres influencent directement l'effet du flou. En effet, plus sigma est petit plus le lissage est léger, à l'inverse le lissage est plus marqué avec une perte de détails importante.

## 2. Laplacien

Le filtre Laplacien est un filtre différentiel utilisé pour détecter les contours dans une image en calculant la dérivée seconde des intensités des pixels, mettant ainsi en évidence les variations rapides d'intensité. Le noyau est construit avec un centre de valeur -4, des voisins immédiats égaux à 1, et 0 ailleurs. En ajustant le paramètre de la taille du noyau, on a observé que l'image devient progressivement plus claire jusqu'à saturation. Cela s'explique

par le renforcement excessif des zones uniformes, où les valeurs calculées dépassent les niveaux de gris standards, entraînant un rendu blanc.

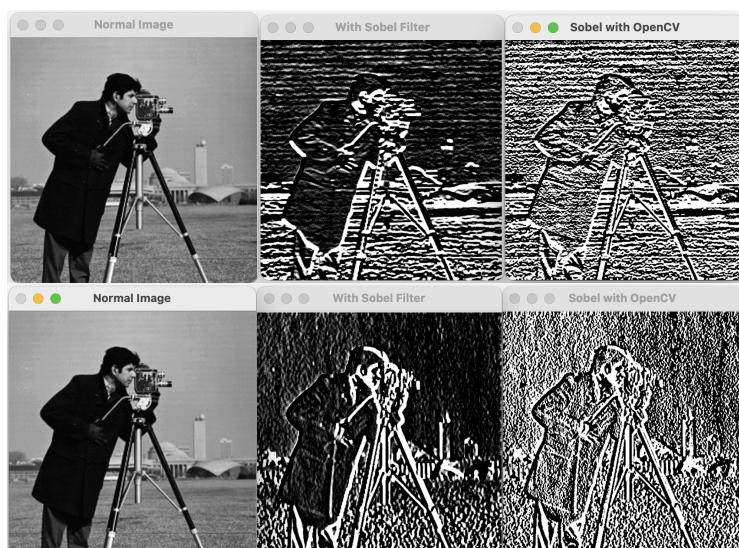
Par exemple, ici, avec un noyau de taille  $5 \times 5$ , les contours restent visibles, mais on remarque déjà un éclaircissement des zones

uniformes et une taille plus grande amplifie cet effet, rendant même l'image toute blanche et donc pas exploitable pour la détection des contours.

Ainsi, un petit noyau permet une détection précise des contours avec moins de saturation.

## 3. Sobel

Le filtre Sobel est un filtre différentiel utilisé pour détecter les contours dans une image en mettant en évidence les gradients d'intensité dans des directions spécifiques : horizontale et verticale. Il est basé sur la convolution de l'image avec un noyau orienté selon l'axe choisi où pour la détection horizontale, les valeurs augmentent avec la position des colonnes ( $j$ ), tandis que pour la détection verticale, elles varient avec la position des lignes ( $i$ ).



Ici, la détection des contours a été réalisée en utilisant le filtre Sobel avec un noyau de taille  $7 \times 7$ . Dans la première image, les contours verticaux sont clairement mis en évidence, ce qui permet de visualiser les transitions dans la direction verticale. Dans la seconde image, les contours horizontaux sont accentués, révélant ainsi les variations horizontales présentes dans l'image. Cette distinction entre les deux directions

démontre l'efficacité du filtre Sobel pour détecter les gradients d'intensité selon des axes spécifiques.

#### 4. Passe haut

Le filtre passe-haut est utilisé pour mettre en évidence les contours et les détails fins d'une image. Pour réaliser ce filtre, nous avons généré un noyau où toutes les valeurs sont initialisées à -1, sauf le centre, qui prend une valeur positive calculée en fonction de la taille du noyau. Cela permet de conserver les contours tout en éliminant les zones uniformes.



L'image obtenu avec notre implémentation du filtre montre des contours fortement accentués, tandis que les zones homogènes apparaissent en noir ou blanc en fonction des variations

d'intensité. Bien qu'avec openCV une similitude existe, il se trouve une meilleure gestion de niveaux de gris réduisant l'effet de saturation.

On remarque ainsi que le filtre passe-haut est efficace pour faire ressortir les contours des zones uniformes.

Pour conclure cette partie, bien que plusieurs filtres semblent produire des effets similaires, certains filtres s'avèrent plus adaptés que d'autres en fonction de la tâche spécifique à accomplir :

FILTRE	OBJECTIF	EFFET	PARAMÈTRE
Médian	Réduction du bruit impulsif	Bruit "poivre et sel" supprimé, contours préservés	Taille du noyau
Moyenneur	Réduction du bruit (lissage)	Image lissée avec perte de détails sur les contours	Taille du noyau
Gaussien	Lissage progressif et doux	Réduction du bruit avec conservation de transitions douces	Taille du noyau et sigma
Laplacien	Détection des variations rapides	Contours nets, saturation progressive pour noyau large	Taille du noyau
Sobel	Détection des gradients (bords)	Contours détectés selon les directions	Taille du noyau et direction
Passe-haut	Accentuation des contours	Contours accentués, zones uniformes éliminées	Taille du noyau

Ainsi, nous avons étudié des filtres de lissage, comme le moyenieur et le gaussien, qui réduisent efficacement le bruit, mais au prix d'une perte de détails. Le filtre médian, quant à

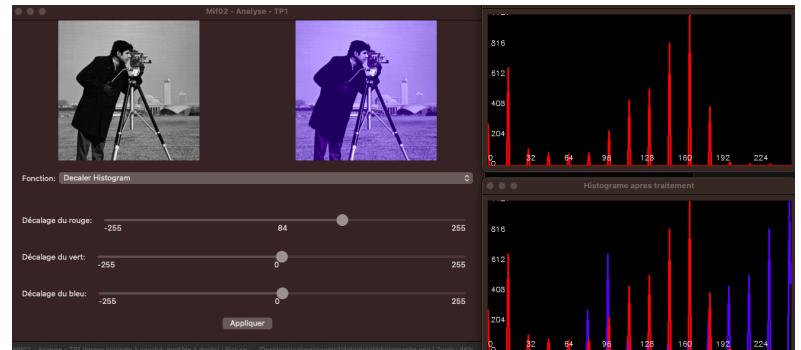
lui, est particulièrement efficace pour éliminer les bruits impulsifs tout en préservant les contours. Enfin, les filtres différentiels, tels que le Laplacien et le Sobel, permettent de détecter les contours en mettant en évidence les variations d'intensité dans l'image.

## IV. Transformation d'histogramme et LUT

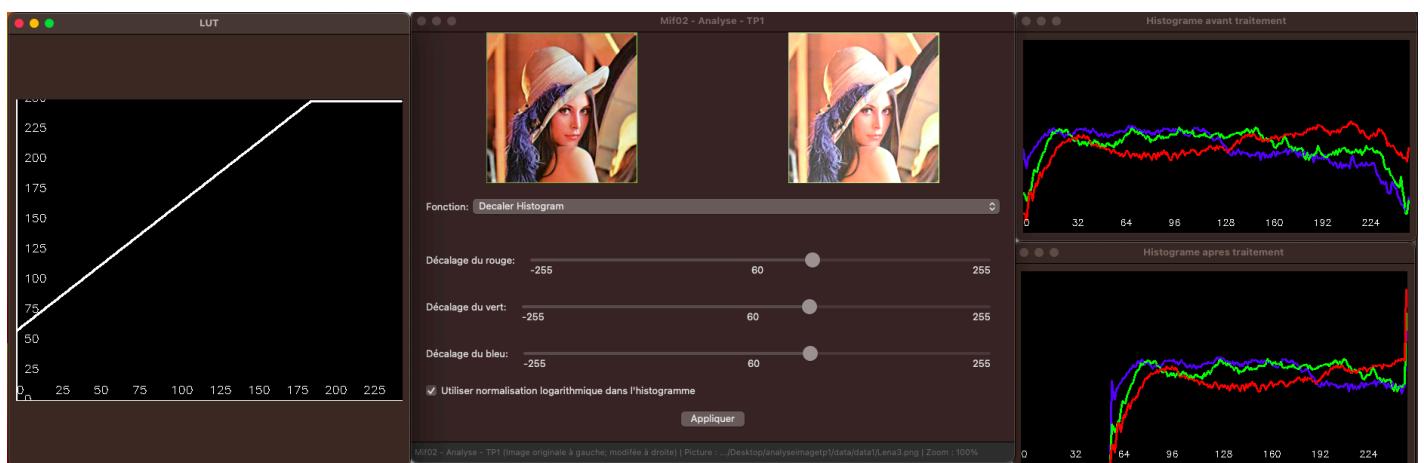
Pour chacune des opérations ci-dessous, l'histogramme avant traitement; après traitement; et le graphique LUT seront affichés au clique sur le bouton appliquer. Nous avons implémenté une checkbox pour visualiser l'histogramme avec une normalisation logarithmique; la normalisation sera linéaire si la checkbox n'est pas cochée.

### 1. Décalage des histogrammes

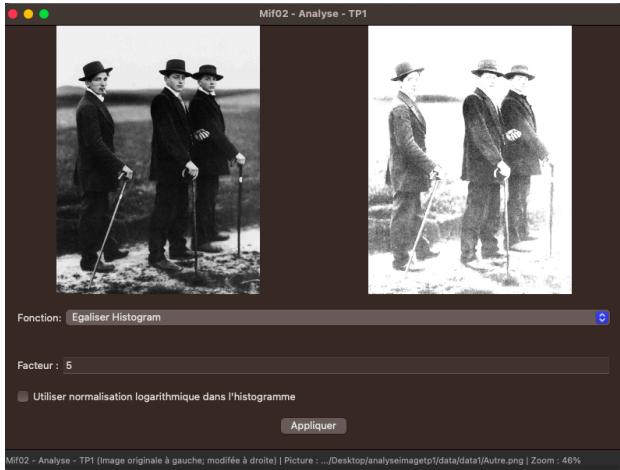
Nous avons implémenté un système pour décaler les histogrammes RGB. Les réduire en même temps va réduire la luminosité. Augmenter toutes les valeurs en même temps fera au contraire monter la luminosité. Aussi l'exemple à droite; augmenter le bleu apporte donc une teinte bleutée à l'image.



*Représentation des histogrammes et LUT d'une augmentation de la luminosité :*



## 2. Égalisation des histogrammes



Notre fonction “equalizeHist” implémente une égalisation d’histogramme sur une image couleur pour améliorer son contraste en réajustant la distribution des intensités des pixels dans chaque canal de couleur (BGR). Pour chaque canal, l’algorithme calcule l’histogramme de l’image, puis génère la fonction de distribution cumulative (CDF). Cette fonction est ensuite normalisée sur la plage [0, 255] pour obtenir une nouvelle distribution d’intensité. Chaque pixel de l’image est mis à jour en fonction de la valeur obtenue à partir de la CDF, avec un facteur de contraste appliqué pour moduler l’effet.

## V. Étirer l’histogramme



Notre fonction “stretchHist” implémente une opération d’étirement d’histogramme sur une image en couleur, afin d’augmenter son contraste en redistribuant les intensités des pixels dans chaque canal de couleur (BGR). Pour chaque canal, l’algorithme calcule d’abord l’histogramme de l’image, identifiant les valeurs minimales et maximales non nulles dans l’histogramme. Ces valeurs servent à normaliser les intensités des pixels de manière à les étirer sur toute la plage [0, 255], puis un facteur d’étirement est appliqué pour moduler l’intensité du contraste. Les nouvelles valeurs des pixels sont ensuite mises à jour pour chaque canal de couleur.

## VI. Compression

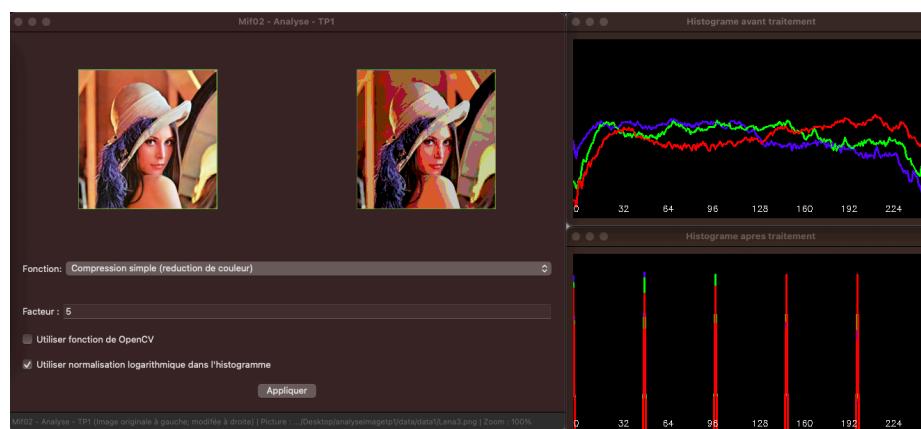
*Graphique LUT entre avant et après compression*



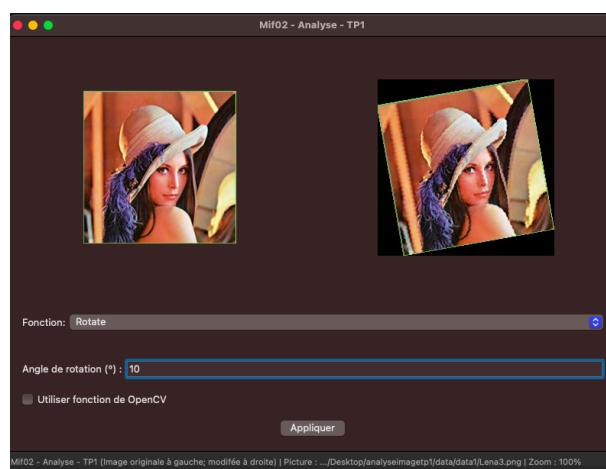
Notre implementation crée d'abord une table de correspondance (LUT) pour chaque canal (rouge, vert, bleu) qui réduit les niveaux de couleur à un nombre spécifié de niveaux et applique ces LUTs à chaque pixel de l'image en modifiant les valeurs de chaque canal selon la LUT correspondante, ce qui permet de réduire la précision des couleurs et de compresser l'image.

Le bouton bouton “utiliser fonction de openCV” va appeler un kmean afin de reduire le nombre de couleur

*Interface graphique et histogrammes avant / après compression*



## VII. Transformation géométrique



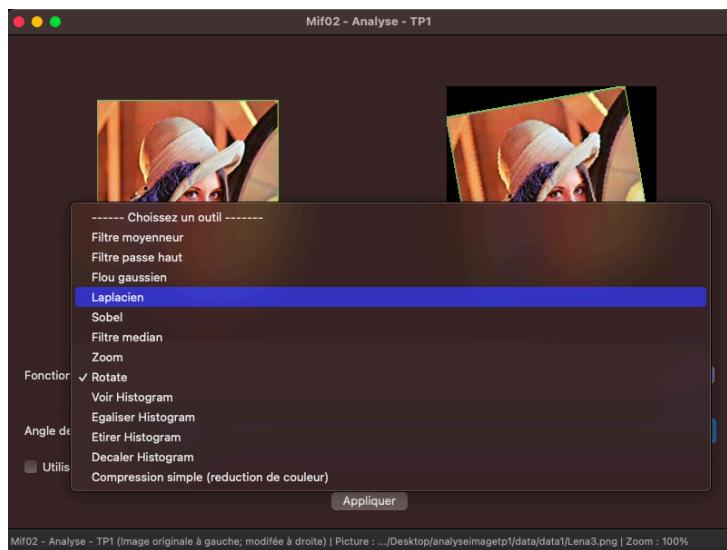
Dans notre projet, on a implémenté deux transformations géométriques dont le redimensionnement et la rotation. Ainsi, on a une fonction “applyResize” qui réalise un redimensionnement en calculant les pixels selon un facteur défini par l'utilisateur. La fonction “applyRotation” quant à elle applique une rotation en utilisant des formules de transformation de coordonnées. On a la possibilité de comparer ces fonctionnalités avec celle proposée par openCV dans notre interface graphique.

## VIII. Interface graphique

Afin d'améliorer l'expérience utilisateur, nous avons décidé de développer une petite interface graphique. Pour cela, nous avons utilisé wxWidgets, une bibliothèque graphique libre et multiplateforme. L'interface permet ainsi de charger des images directement et de sélectionner facilement les fonctionnalités à appliquer, telles que l'application de filtres (médian, moyenieur, gaussien, etc.), la visualisation et manipulation des histogrammes (calcul, égalisation, étirement, etc.), ainsi que la réalisation de transformations géométriques comme le zoom.

Nous avons implémenté un petit système de plugin afin de faciliter la découpe du code en plusieurs fichiers indépendants. Nous avons également créé quelques composants réutilisables tel que le slider pour la taille de kernel pour nos filtres. Enfin nous avons clairement séparé le code de l'interface (dossier gui) du code des traitements sur image (dossier src).

*Choix de la fonction:*



*Ouverture d'une image :*

