Photocheap CC

Rapport de projet BITMAP

Par Emeline EHLES et Robin DELL

12/05/2017

INTRODUCTION

Nous avons choisi le projet bitmap, il avait pour but de lire une image binaire au format bitmap, effectué des modifications dessus (contraste, saturation, niveau de gris…), puis de sauvegarder l'image en respectant les normes bitmap. Le traitement d'image joue un rôle important dans la société, on le retrouve partout sur beaucoup d'appareil.
Le véritable essor du traitement d'images n'a eu lieu que dans les années 1960 quand les ordinateurs commencent à être suffisamment puissants pour travailler sur des images. C'est la recherche médicale qui effectue de grosse demande pour les traitements d'image afin d'améliorer les diagnostics. Ce sont ensuite les publicitaires, puis le grand public qui se familiarisa avec la retouche d'image grâce à certain logiciel.
Notre projet c'est surtout tourné vers la colorisation d'image. Nous avons eu quelque difficulté a trouvé la meilleure façon pour coloriser les images, certaine façon était impossible car trop volumineuse, ou on pourrait utiliser une IA afin de faire une moyenne de plusieurs images. La colorisation d'image est un moyen de retrouver les couleurs sur des anciennes images en noir et blanc.

Une documentation est disponible sur Github : https://github.com/Rodd8/Photocheap

Contexte

Actuellement, les traitements d'images peuvent être effectués sur des logiciels tels que Photoshop mis en place par Adobe, paint.net, The Gimp, Google Photo… Le traitement d'image est presque indispensable dans notre société. En effet, beaucoup de professionnel utilise par exemple Photoshop pour les magazines de stars. Même les personnes de tous les jours en utilise pour améliorer leurs image et donné des effets afin de les partagés à ses amis.
Il existe aussi des algorithmes de détection de visage ou d'objet via la fonction de détection des contours par exemple. Cette méthode est utilisée par Facebook, afin de pouvoir identifier le visage de ces amis. Ils existent aussi d'autres méthodes, cette fois pour la détection d'objet, comme la mise en œuvre de techniques d'apprentissages, ces techniques utilisent un algorithme de recherche de critères pour différencier un objet d'un autre, en exploitant des bases de données d'objets définis.
Le traitement d'image est donc utilisé aussi à des fins scientifiques comme pour les IRM ou encore analyser le déplacement des personnes ou des souris.

METHODOLOGIE

L'image Bitmap

1) Structure

Une image bitmap est une image matricielle au format BITMAP, ce format de fichier requière une mise en forme particulière pour pouvoir être lue par un ordinateur. Le format bitmap se compose de 3 parties principales : le Header (ou en-tête), le DIB Header (ou InfoHeader) et enfin la liste des pixels dans l'ordre little-endian, c'est à dire en commençant par la fin.
Nous avons donc converti l'image bitmap d'entré en 1 « objet » informatique manipulable en langage C.

Cet objet se compose de 3 structures :

```
struct BMPHeader
{
    char        Type[2];
    int         Size;
    int         Reserved;
    int         Offset;
    InfoHeader  InfoHeader;
} ;
```

Figure 1 : Structure du Header

```
{
    int     Size;
    int     Width;
    int     Height;
    short   Planes;
    short   Bits;
    int     Compression;
    int     ImageSize;
    int     xResolution;
    int     yResolution;
    int     Colors;
    int     ImportantColors;
};
```

Figure 2 : Structure de l'InfoHeader

```
struct BMP
{
    int     width, height;
    Pixel*  data;
};
```

Figure 3 : Structure BMP

2) Manipulation

```
FILE* bmp_input = fopen(filename, "rb");
if(!bmp_input)
{
    printf("\nOuverture du fichier \"%s\" impossible.", filename);
    exit(1);
}
fread(&header, sizeof(Header), 1, bmp_input);

bmp     = newBMP(header.InfoHeader.Width, header.InfoHeader.Height);
padding = corr[(3*header.InfoHeader.Width)%4];

fseek(bmp_input, header.Offset, SEEK_SET);
```

Figure 1: Code permettant la lecture du Header

Le paramètre « rb » de la fonction fopen permet de spécifier que le fichier lu est au format binaire (rb = read binary).

La variable *padding* est une variable permettant de « remplir » les lignes, en effet, la norme bitmap exige que chaque ligne contienne un nombre de pixels multiple de 4, cette petite astuce permet de compléter une ligne si elle est trop courte, si par exemple l'image fait 400 pixels de large, *padding* vaudra 0, au contraire si l'image fait 401 pixels de large, *padding* vaudra cette fois 3 (401+2 = 404, multiple de 4).

Enfin, fseek() permet de déplacer le curseur de lecture au « header.Offset », soit le début de la liste de pixel.

Nous avons utilisé le pragma pack 1 pour « compresser » nos structures. En effet, selon la version du compilateur, le système d'exploitation, la version du langage utilisé, il est possible que les structures ne prennent pas autant de place en mémoire sur un ordinateur ou sur un autre.

Dans l'exemple ci-dessous, nous avons une structure qui fait techniquement 6 octets, or, il est possible qu'elle prenne 12 octets en mémoire selon la méthode de stockage utilisé par le compilateur. Dans notre cas, à cause de la construction stricte du format BMP, nous avons dû utiliser le pragma pack pour nous assurer que chaque attribut de l'image fait bien tel ou tel nombre d'octets, pas plus pas moins pour ne pas risquer de corrompre l'image.

```
struct Exemple
{
    char    A;  /* 1 octet                  */
    int     B;  /* 4 octets    -> 6 octets  */
    char    C;  /* 1 octet                  */
};

        Sans pragma pack
    |  1  |  2  |  3  |  4  |
    |-----|-----|-----|-----|
    | A(1)| ..............  |
    | B(1)| B(2)| B(3)| B(4)|
    | C(1)| ..............  |

Avec pragma pack
    |  1  |
    |-----|
    | A(1)|
    | B(1)|
    | B(2)|
    | B(3)|
    | B(4)|
    | C(1)|
```

Figure 2: Explication simplifier du fonctionnement du pragma pack(1)

Les Pixels

Le stockage des pixels en bitmap se fait au format RVB, c'est à dire une composante de Rouge, une composante de Vert, et une composante de Bleu.
Chaque valeur est comprise entre 0 et 255, 0 étant la teinte la plus sombre, et 255 la teinte la plus claire.
Mais il existe plusieurs variantes pour stocker des couleurs :
RVB, HSV, HSL, CMYK etc. Nous avons utilisé la méthode RVB et HSL

1) RVB

```
struct Pixel
{
    unsigned char Blue;
    unsigned char Green;
    unsigned char Red;
};
```

Figure 3: Structure des Pixels (Blue, Vert, Rouge)

Les attributs dans l'ordre BVR et non pas RVB comme l'habitude le voudrait car étant donné que le Bitmap utilise le système little-endian pour le codage des couleurs, il faudra plus tard pouvoir stocker les valeurs RVB à « l'envers », nous avons donc inversé l'ordre des éléments pour plus de simplicité.
Le type utilisé « unsigned char » est particulièrement pratique dans ce cas car il fait pile 2 octets, soit 256 valeurs possibles (de 0 à 255)

2) HSL

```
struct HSL
{
    float Hue;
    float Sat;
    float Light;
};
```

Figure 4: Structure des Pixels HSL (Hue, Saturation, Light)

Ce mode de représentation des couleurs est souvent utilisé dans les nuanciers sur de nombreux logiciel de traitement d'image car il représente la teinte d'une couleur en degrés, sur un cercle dont le rayon est la saturation. La lumière se règle généralement avec un curseur. Nous avons choisi de légèrement simplifier le cercle en le représentant en pourcentage, pour harmoniser le cet attribut avec les 2 autres, et surtout pour simplifier les calculs.

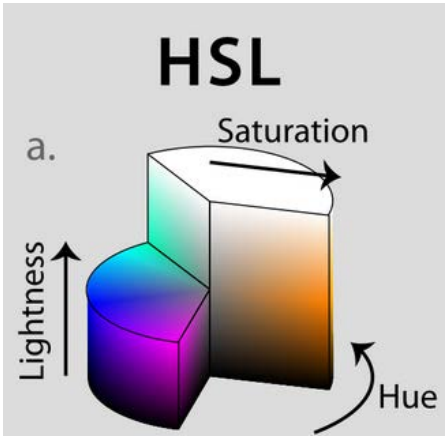


Figure 5: Représentation cylindrique du système HSL

3) RGB2HSL et HSL2RGB

Comme nous ne lisons en entrer que du RVB et que nous ne pouvons qu'écrire en RVB, il a fallu convertir les pixels en RVB en HSL, et vice versa.
Pour cela, nous avons trouvé des fonctions prévues à cet effet qu'il nous a fallu adapté à notre code et à notre simplification.

HSL to RGB conversion formula

When $0 \le H < 360$, $0 \le S \le 1$ and $0 \le L \le 1$:

$$C = (1 - |2L - 1|) \times S$$

$$X = C \times (1 - |(H / 60°) \bmod 2 - 1|)$$

$$m = L - C/2$$

$$(R', G', B') = \begin{cases} (C, X, 0) & , 0° \le H < 60° \\ (X, C, 0) & , 60° \le H < 120° \\ (0, C, X) & , 120° \le H < 180° \\ (0, X, C) & , 180° \le H < 240° \\ (X, 0, C) & , 240° \le H < 300° \\ (C, 0, X) & , 300° \le H < 360° \end{cases}$$

$$(R,G,B) = ((R'+m)\times 255, (G'+m)\times 255, (B'+m)\times 255)$$

Figure 6: Formule générale pour le passage de HSL a RVB

Les fonctions de modification

1) Niveau de gris

```
for(i=0; i<bmpTemp->width; i++)
{
    for(j=0; j<bmpTemp->height; j++)
    {
        p       = getPixel(bmpTemp, i, j);
        grey    = p.Red*0.2125 + p.Green*0.7154 + p.Blue*0.0721;
        p.Blue  = p.Red = p.Green = grey;
        setPixel(bmpTemp, i, j, p);
    }
}
```

Figure 7: Algorithme permettant le passage en niveau de gris - formule du CIE

La formule utilisé est celle fournie par le CIE (Comité International de l'Éclairage), contrairement à un calcul de moyenne basique ($G = \frac{R+V+B}{3}$), cette formule tend à faire ressortir plus certaines couleurs a d'autre car, par exemple, le vert est plus lumineux que le bleu.

2) Négatif

```
for(i=0; i<bmp->width; i++)
{
    for(j=0; j<bmp->height; j++)
    {
        p       = getPixel(bmpTemp, i, j);
        p.Red   = 255-p.Red;
        p.Green = 255-p.Green;
        p.Blue  = 255-p.Blue;
        setPixel(bmpTemp, i, j, p);
    }
}
```

Figure 8: Algorithme de mise en négatif

Inversion des valeurs couleurs, étant bornées a 255, il suffit pour chaque composante R V B d'un pixel de l'ôter a 255 et d'y assigner la valeur obtenue.

3) Détection des contour, Sobel et Pewitt

La détection de contour a été une fonction assez difficile à mettre en place, il a fallu comprendre le fonctionnement des masques (ou filtres) de Pewitt et de Sobel ainsi qu'adapter les algorithmes à notre projet. Pour faire simple, c'est deux algorithmes recherche les changements bruts de luminosité sur une image en nuance de gris. Les forts changements de luminosité sur l'image seront marqués par des pixels blancs, les zones sans changement brutal de luminosité seront quant à eux, noir. On aura donc au final une « carte » des contours des éléments présents sur l'image.

4) La colorisation

Pour re-coloriser une image issus d'un thème précis comme l'automne par exemple, nous avons essayé de trouver le triplet RVB moyen associé à chaque nuance de gris en analysant une centaine d'image portant sur le même thème.

```
 Gris     (R1,V1,B1); (R2,V2,B2); (R3,V3,B3)     RVB moyen

   9  <-  ( 0, 0,27); ( 9,11, 7); (12, 8, 7)  => [ 7, 6,14]
  10  <-  ( 8,12,10); ( 0, 5,15); (10,10,10)  => [ 6, 9,12]
  11  <-  (11, 1,21); (33, 0, 0); ( 6,14,13)  => [17, 5,11]
```

Figure 9: Explication du moyennage des couleurs pour une nuance de gris donnée

Cette méthode fonctionnait grossièrement, en effet, étant donné que chaque image étaient différentes au niveau des contrastes, de la saturation, de la résolution, de la luminosité, etc ; les données étaient faussées. Un tronc marron sur une photo apparaissait noir sur une autre, une forêt de chênes en automne n'a pas les même couleurs qu'une forêt de hêtres.
Afin d'affiner nos colorisations, nous avons deux méthodes le moyennage des couleurs RVB et le moyennages des couleurs selon le principe HSL. Ce test nous a démontré que le système HSL était plus performant pour les zones claires et de hautes lumières, alors que le

mode RVB était lui plus efficace pour les couleurs « sombres ».
Nous avons donc fusionné les deux méthodes, pour déterminer quand passer du HSL au RVB nous avons tout essayé, les 256 possibilités en déplaçant un curseur pas à pas.
Pour chaque image ainsi obtenu on calcule sa différence avec l'image de base, cette différence est reportée sur une troisième image temporaire. Plus cette image est sombre,

plus elle est proche de l'image de base. Pour déterminer ce paramètre, on calcule le triplet RVB moyen de toute l'image, qui devrait idéalement être proche de (0,0,0).

Figure 10: Image « delta » encore pleine de couleur

Figure 11: Image « delta » presque noire, donc proche de l'image originale

Une fois le triplet RVB optimal trouvé, on colorise donc l'image en plaquant le curseur a la coordonnée correspondante.

On peut, avec la même technique, coloriser une image avec les informations d'une autre image.

RESULTATS

Figure 12: Image de base

Nous utiliserons cette image comme support de travail dans la suite de cette partie, la figure 12 est l'image de base.

1) Nuance de gris

Figure 14: Via le calcul de la moyenne

Figure 13: Via la formule du CIE

On voit bien que les couleurs qui étaient clair de base (jaune, blanc, rose), sont restés dans un niveau de gris clair pour la version CIE alors que pour la moyenne, le jaune est resté dans la gamme des gris plutôt foncé, quasi identique au gris des ailes. On remarque, de plus, pour les couleurs foncés tel que le bleu, le rouge et le vert, des niveaux de gris assez sombre surtout le rouge qui est le plus sombre dans la version CIE. Tandis que pour la méthode de la moyenne, on a plus de mal a distingué le passage du rouge au vert puis au bleu au niveau des ailes quand on le passe en niveau de gris.

La méthode la plus correcte, après comparaison est donc celle du CIE.

2) Négatif

Figure 16: Image en négatif

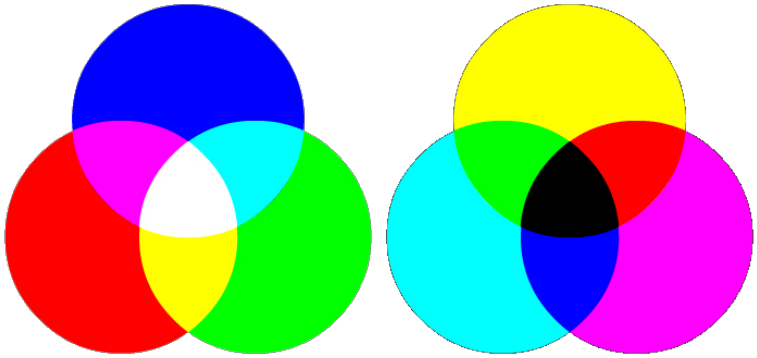


Figure 15: Cercle chromatique (gauche normal - droite négatif)

D'après les cercles chromatiques (figure 15), on observe bien le passage des couleurs à leurs opposés. En effet, le bleu clair passe au rouge et vice versa, le vert passe au violet, le jaune passe au bleu foncé.

3) Contrastes

Figure 17 : Augmentation des contrastes de +120

Figure 18 : Contraste négatif

Avec une augmentation des contrastes de +120 (sur une échelle allant jusqu'à 255) on observe bien que l'image est beaucoup plus vive ; les noirs sont plus sombres, les zones de lumière sont encore plus lumineuses.
Par contre nous avons remarqué que les contrastes dans des valeurs négatives ne fonctionnent pas comme l'on voit sur la figure, surement lié à une erreur dans le code. Nous pensons qu'il s'agit simplement d'une erreur liée au troncage des valeurs, en effet, si les valeurs contrastées sont inférieures à 0, ou supérieur à 255, le code va mal les interpréter.

4) Saturation

	
Saturation : -50%	Saturation : -100%

	
Saturation : +50%	Saturation : +100%

Avec une saturation de -50%, on observe des couleurs plus terne et pâle dans son ensemble. Si on continu, avec une saturation de -100%, soit une désaturation maximale des couleurs, les teintes deviennent tellement ternes, que l'image ne se compose plus que de nuance de gris.

Avec une saturation de +50%, on observe des couleurs beaucoup plus vives. On observe bien la différence avec celle de -50%. De même, plus on augmente la saturation, plus les couleurs semble vive, surtout pour le rouge et le vert.

5) Contours

Lorsque l'on compare Pewitt à Sobel, on remarque que Pewitt ne prend que les changements de luminosité les plus abrupt pour effectuer le contour alors que Sobel prend

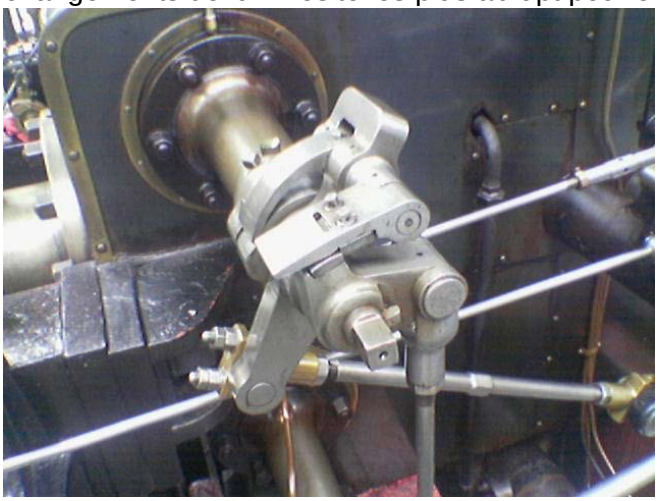
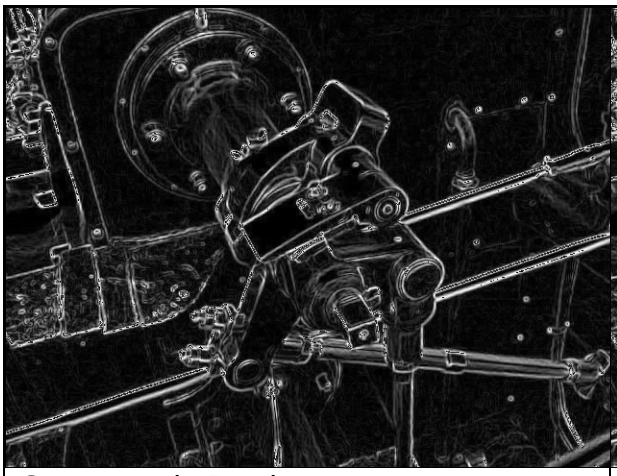
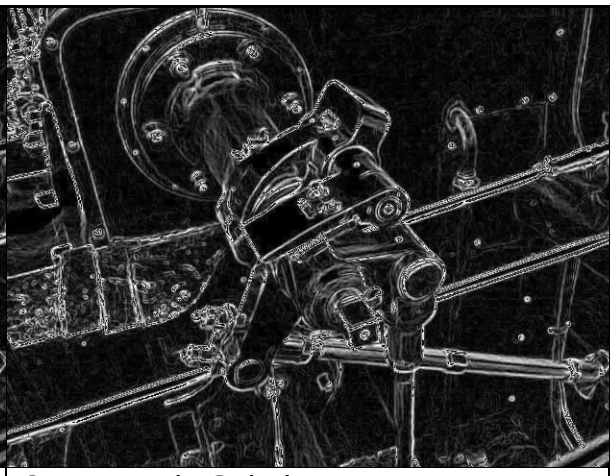


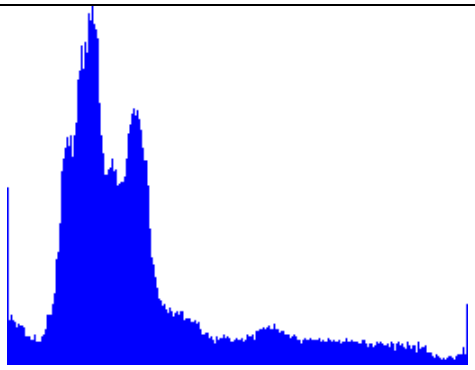
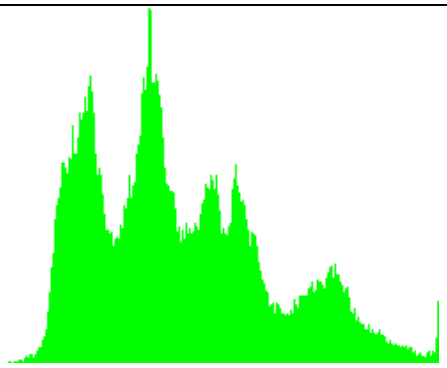
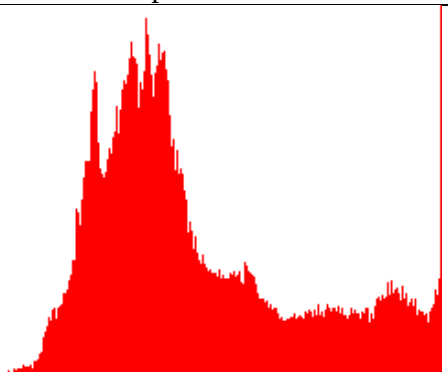
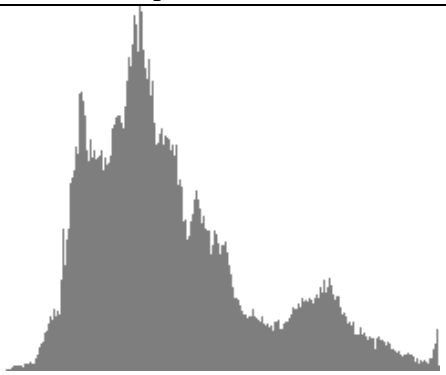
Figure 19 : Image servant à la détection de contours

de même les changements de luminosité mais de manière beaucoup plus de détail dans les contours. Sobel est donc beaucoup plus détaillé pour les contours, mais dans la robotique Sobel entrainerait des parasites, en effet on peut observer sur l'image plein de pixels blancs. Le plus pertinent est donc le contour par le filtre de Pewitt, beaucoup plus net.

	
Contours via Pewitt	Contours via Sobel

6) Histogrammes

Tableau 1: Histogrammes selon différentes composantes de la photo des perroquets

 Composante bleue	 Composante verte
 Composante rouge	 Histogramme de l'image en nuance de gris

Notre image de base est colorée mais assez terne, les histogrammes le montre assez bien. Toutes les couleurs sont tous assez bien regroupé vers des couleurs foncés. On voit de même que le vert est une couleur très présente ainsi que le rouge, puis en dernier le bleu un peu moins présent. En effet, la plupart des pics de couleurs sont la première moitié des histogrammes, soit la partie plus sombre des couleurs.

7) Colorisation

1ere méthode :

On passe l'image de base en noir et blanc, et à partir des données de l'image de base on va recoloriser celle en noir et blanc. On obtient une image quasi-identique, même si l'on remarque une petite perte de luminosité au niveau du cheval et un peu des arbres.

Figure 17: image d'origine

Figure 21 : Image recolorisée

Figure 22 : Image d''origine

Figure 23 : Image recolorisée

Dans le cas présent, on remarque que la couleur du tronc devient verte, la teinte orange du soleil est atténuée et le rose pâle des fleurs devient lui aussi dans les tons verts.

2eme méthode :

On a pris les données de l'image de base, puis on a pris une autre image qu'on a passée en noir et blanc et on l'a recolorées. On obtient une image se rapprochant a des teintes d'automne comme sur l'image de base mais par rapport à l'image réel, on a une grosse perte de donnée comme le vert par exemple.

Figure 24 : Image a recoloriser

Figure 25 : En nuance de gris

Figure 26 : Image dont nous extrayons les informations de couleurs

Figure 18: Image recolorisé

PERSPECTIVE

Pour faire évoluer notre projet nous devrions, en premier lieu, ajouter d'autre outils de modifications tel qu'une fonction de flou gaussien, une fonction de réduction (ou d'ajout) de bruit, etc.
Mais la chose la plus importante à faire serait d'optimiser la fonction de colorisation assez bancale pour le moment. Actuellement, il y a deux principaux moyens de recolorisation : le faire à la main, ou le faire automatiquement avec une intelligence artificielle.
La seconde méthode est basé sur le « Machine Learning », l'ordinateur apprend lui-même a coloriser une image à partir de ses connaissances. Pour faire simple, on fournit à une IA un lot d'image en couleurs et leur équivalent en nuance de gris ; il va analyser les couples d'image et apprendre quel couleur correspond le mieux à tel nuance de gris. Il fait cela en se basant sur les contours des objets présents sur l'image, les contrastes, la saturation des couleurs ainsi que d'autre facteurs.
Cela en mémoire, la machine sera capable de reconnaître un tronc d'une voiture, un vélo d'un paysage, etc. Il pourra donc, toujours d'après ses connaissances savoir qu'un tronc est généralement marron et non pas rose. Cette technique, si bien maîtrisée, est très performante.

Aussi, nous pourrions essayer de trouver un autre moyen de stocker les couleurs : étant donné qu'il n'y a que 256 nuances de gris possible dans le système RGB, nous ne pouvons au final que 256 couleurs. Une par valeur de gris. On perd donc énormément d'information, c'est pour cela qu'une image avec peu de couleurs de base sera mieux recolorisé avec notre programme qu'une image contenant une large palette de couleurs.

Nous sommes actuellement limités au format BMP, certes simple mais peu utilisé ni optimisé. De ce fait, il pourrait être plus utile de travailler sur des formats courants, comme le JPG ou le PNG. Dans la même logique, pour gagner du temps nous pourrions utiliser une librairie spécialisée dans le traitement d'image comme QuicknDirtyBMP ou bien Magick++, respectivement une librairie libre pour la lecture et l'écriture d'image BMP, et une librairie libre en C++ pour un support optimal de plus de 200 formats d'image différents, des fonctions de modifications optimisé etc.
Mais pour cela, il faudrait recommencer notre programme en C++. Ce langage est, comme l'a dit son auteur, un « C avec classe ». C'est un langage tout aussi puissant que le C, mais étant plus récent et permettant la programmation orienté objet (POO), il est actuellement plus populaire dans le domaine du développement. Et donc, il y a plus de ressources disponibles pour le traitement d'image.

BIBLIOGRAPHIE

http://linusfurious.blogspot.fr/2004/10/strange-genetic-wonders-over-centuries.html

https://en.wikipedia.org/wiki/Sobel_operator

http://www.resimbox.com/wp-content/uploads/2017/01/Doga_Manzara_Resimleri_Nature_View_Pictures_00000013_ilkbahar_Manzaralari_Spring_Views.jpg

https://funtastique.fr/wp-content/uploads/2014/02/un-cheval-dans-la-neige.jpg

http://1.bp.blogspot.com/-IzUkuroBmyE/TqsAiq_AurI/AAAAAAAAAJc/fJ2YJshqOx4/w1200-h630-p-k-no-nu/fall-foliage.jpg

https://pandagif.com/gifs/aibvF69KAg.jpg

http://www.rapidtables.com/convert/color/rgb-to-hsl.htm

http://www.dfstudios.co.uk/articles/programming/image-programming-algorithms/image-processing-algorithms-part-5-contrast-adjustment/

https://en.wikipedia.org/wiki/BMP_file_format

https://www.unige.ch/medecine/bioimaging/files/5714/2547/4555/ImageJ_Introduction.pdf

https://fr.wikipedia.org/wiki/Traitement_d%27images