Astruc Lélio
Del Rosso Luca
Edery Nathan

POLYTECH NICE-SOPHIA

Rapport de projet : compression d'une image par méthode DCT2

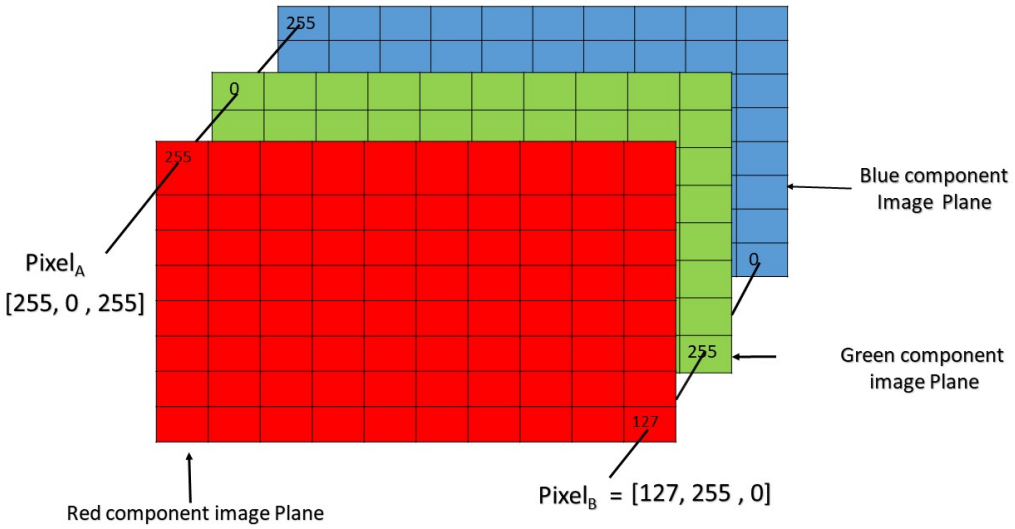
---

1 Présentation du projet

Dans le cadre de cette UE EIMA53, nous avons eu la possibilité de réaliser un programme permettant de compresser une image à l'aide des transformées de Fourier.
Commençons par comprendre ce qu'est une image.

Représentation d'une image couleur

Une image en couleur peut être vu comme plusieurs tableaux qui correspondent chacun aux trois canaux de couleur de l'image.



Pixel of an RGB image are formed from the corresponding pixel of the three component images

Explication du programme

Nous allons ici utiliser la transformée de cosinus discrète (DCT-2). Puisque les dimensions de l'image seront tronquées à des multiples de 8, puis l'intensité sera recentrée vers 0, on aura une période de 16 dans chaque direction. Il sera ainsi possible d'appliquer la transformée de Fourier pour obtenir des informations sur les hautes fréquences de l'image, puis appliquer la transformée de Fourier inverse. Cela permet de réduire la taille d'une image pour par exemple, la partager sur internet ou tout simplement, stocker sur son propre disque des images plus légère. Naturellement, il faut pouvoir garder un maximum. Enfin, le programme calculera le taux de compression et l'erreur

2 Implémentation de l'algorithme

Initialisation

La première étape de ce projet est de récuperer un tableau des données de l'image. Pour cela, on utilise matplotlib :

```
import matplotlib.pyplot as plt

image = plt.imread("photos/image.jpg")
```

On s'assure ensuite que l'image est en tableau d'entiers de 8 bits

```
image  =  (image * 255).astype(int)
```

On tronque ensuite l'image à des multiples de 8

```
nbLignes = int(nbLignes/8)*8
nbColonnes = int(nbColonnes/8)*8
```

On récupère chaque canal de couleur de l'image pour traiter chaque canal séparemment

```
canalRouge = decompositionMatrice(matriceInitiale[:,:,0],nbLignes,nbColonnes)
canalVert = decompositionMatrice(matriceInitiale[:,:,1],nbLignes,nbColonnes)
canalBleu = decompositionMatrice(matriceInitiale[:,:,2],nbLignes,nbColonnes)
```

On définit la matrice de passage P.
Le sujet donne la formule suivante :

$$D_{k,l} = \frac{1}{4} C_k C_l \sum_{i=0}^{7} \sum_{j=0}^{7} M_{i,j}\ cos\left(\frac{(2i+1)k\pi}{16}\right) cos\left(\frac{(2j+1)l\pi}{16}\right)$$

On a ainsi

$$\begin{aligned} d_{k,l} &= \frac{1}{4} C_k C_l \sum_{i=0}^{7} \sum_{j=0}^{7} m_{i,j} cos\left(\frac{(2i+1)k\pi}{16}\right) cos\left(\frac{(2j+1)l\pi}{16}\right) \\ &= \frac{1}{4} \sum_{i=0}^{7} \sum_{j=0}^{7} m_{i,j} C_k C_l cos\left(\frac{(2i+1)k\pi}{16}\right) cos\left(\frac{(2j+1)l\pi}{16}\right) \\ &= \frac{1}{2} \sum_{i=0}^{7} C_i cos\left(\frac{(2j+1)i\pi}{16}\right) \end{aligned}$$

Ce qui nous donne les coefficients de cette matrice

```
import math as maths

matricePassage  =  np.zeros((8, 8))
   for i in range(8):
       for j in range(8):
           matricePassage[i, j]  =  (1/2) * maths.cos((2*j+1)*i*maths.pi/16)
matricePassage[0, :] *=  1/maths.sqrt(2)
```

Compression et décompression

Pour compresser l'image, nous devons d'abord la séparer en plusieurs blocs de 8 par 8. Pour ce faire, nous avons écrit une fonction dans un fichier à part.

```
def decompositionMatrice(mat, numeroLigne, numeroColonne):
    sousMatrices = []
        for x in range(0, numeroLigne, 8):
            for y in range(0, numeroColonne, 8):
                sousMatrices.append(mat[x:x+8, y:y+8])
    return sousMatrices
```

Une que l'on a une liste des blocs de l'images, nous pouvons appliquer, à chaque bloc le changement de base $D = PMP^t$. Après avoir fait ce calcul, on peut appliquer la matrice de quantification $D./Q$
Nous avons pour ceci écrit une fonction compression.

```
import numpy as np

def compression(mat, matriceQuantification, matricePassage):
    matriceTransformee = np.zeros((8, 8))
    matriceIntermediaire = np.zeros((8, 8))

    # DCT 2 par changement de base : matriceTransformee = matricePassage * mat
    #*matricePassage^T
    matriceIntermediaire = np.matmul(mat, np.transpose(matricePassage))
    matriceTransformee = np.matmul(matricePassage, matriceIntermediaire)

    # On divise par la matrice de quantification
    matriceTransformee = np.divide(matriceTransformee, matriceQuantification)
    matriceTransformee = matriceTransformee.astype(int)

    return matriceTransformee
```

Pour appliquer les filtres des hautes fréquences, la formule du sujet indique qu'il faut tronquer à 0 les coefficients de $D_{i,j}$ si $i + j \geq seuil$. Pour ce faire nous avons écrit une fonction filtreHauteFrequences :

```
def filtreHauteFrequences(mat, seuil):
    for i in range(8):
        for j in range(8):
            if i + j >= seuil:
                mat[i, j] = 0
    return mat
```

Pour recomposer l'image avec ces blocs 8 par 8, nous avons écrit une fonction recompositionImage :

```
import numpy as np

def recompositionImage(canalCouleur,nombreLignes,nombreColonnes):
    matriceDecompressee=np.zeros((nombreLignes,nombreColonnes))

    compteurMatrice=0
    for ligne in range (int(nombreLignes/8)):
        compteurCols=0
        while compteurCols<int(nombreColonnes/8):
            matrice=canalCouleur[compteurMatrice]
            for i in range(8):
                for j in range(8):
```

```
                    matriceDecompressee[ligne*8+i,compteurCols*8+j]=matrice[i,j]
            compteurCols+=1
            compteurMatrice+=1
    return matriceDecompressee
```

On peut enfin calculer le taux de compression avec :

```
tauxDeCompression = (1-nbCoefNonNul/(nbLignes*nbColonnes*3))*100
```

On peut ensuite recomposer chaque canal :

```
canalRouge = recompositionImage(canalRouge,nbLignes,nbColonnes)
canalVert = recompositionImage(canalVert,nbLignes,nbColonnes)
canalBleu = recompositionImage(canalBleu,nbLignes,nbColonnes)
```

Déclarer un matrice décompressée de la taille de l'image et lui ajouter chaque canal décompressé :

```
matriceDecompressee = np.zeros((nbLignes,nbColonnes,3))

matriceDecompressee[:,:,0] = canalRouge
matriceDecompressee[:,:,1] = canalVert
matriceDecompressee[:,:,2] = canalBleu
```

Post-processing

Enfin, on calcule l'erreur avec :

```
erreur = calculErreur(matriceInitiale,matriceDecompressee)
```

où calculErreur est :

```
def calculErreur(matriceInitiale,matriceDecompressee):
    erreur=0
    norm=0
    for k in range (3):
        erreur+=np.linalg.norm(matriceInitiale[:,:,k]-matriceDecompressee[:,:,k])
        norm+=np.linalg.norm(matriceInitiale[:,:,k])
    norm/=3
    erreur=erreur/3
    erreur=erreur/norm*100
    return erreur
```

Et pour finir, on repasse les entiers du tableau entre 0 et 255 puis à des réels entre 0 et 1 :

```
matriceDecompressee+= 128
```

```
matriceDecompressee = matriceDecompressee/255
```

3 Répartition des tâches

Jour 1 : Lélio: Mise en place des librairies utiles pour pouvoir commencer le projet et début de création du squelette du projet

Nathan: Étude du projet et de la partie mathematique sur papier
Luca: Étude du projet et de la partie mathematique sur papier

Jour 2 : Lélio: Travail sur l'implementation de l'algorithme
Nathan: Travail sur l'implementation de l'algorithme
Luca: Travail sur l'implementation de l'algorithme

Jour 3 : Lélio: Programme presque finalisé donc correction du bug sur le compression d'image en format `png` et `jpg`
Nathan: Correction d'un bug lors de la compression d'image et son affichage
Luca: Correction du bug de compression suivant le format de l'image et début du compte rendu

4 Résultats

Voici le résultat du programme sur des images. Nous obtenons un taux de compression de 97.85% et une erreur de 14.5%

Figure 1: Image initiale

Figure 2: Image compressée avec Q de base et $seuil = 6$

On remarque que l'on perd également la transparence du `png` initial.