

INPAINTING D'IMAGES : MÉTHODE PAR PATCHES

BAHOUS Youness - SRIR Mohammed Ali

Encadrés par : Alasdair James Newson

1 Introduction

Ce projet est une implémentation et une examination de la méthode proposé par l'article [1] , l'ensemble de codes PYTHON de ce projet sont disponibles sur le dépôt Git suivant : lien

Il s'agit d'un algorithme qui sert à remplir des régions inconnues d'une image en se basant sur son contenu en choisissant un ordre de remplissage et des patchs permettant de bien propager les structures de l'image et de prioriser les pixels pour lesquels on a plus d'information sûre.

C'est une méthode particulièrement efficace pour la suppression d'objets en les remplaçant par des fonds convainquant et qui sert aussi à la complétion de textures.

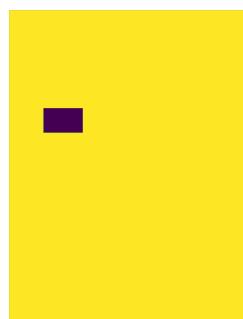
2 Code

2.1 Manipulation de masques

On a choisi d'utiliser dans un premier temps de ne travailler qu'avec des masques rectangulaires car les notions de morphologie n'ont pas été abordées en classe . mais pour la version finale de l'algorithme on a utilisé pour calculer (et mettre à jour) les bords la différence entre le masque et son érosion . cette méthodes est bien plus efficace pour deux raisons :

- Elle est compatible avec n'importe quelle forme de masque
- Elle est compatible avec des masques en plusieurs zones

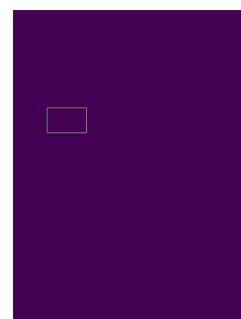
```
1  """def init_bord_m(mask):
2      n,m=mask.shape
3      strell=morpho.disk(1)
4      bords= mask-morpho.erosion(mask,strell)
5      L=[]
6      for i in range(n):
7          for j in range(m):
8              if(bords[i,j]==1):
9                  L.append((i,j))
10     return L
```



(a) Masque



(b) Image Masquée.



(c) Frontière.

2.2 Manipulations de Patches

2.2.1 Extraction de Patches

```
1  """def get_patch(image,p,patch_size=16):
2      """
3          Returns a patch centered on p
4      """
5      r = patch_size//2
6      clip = np.array(image[p[0]-r:p[0]+r+1,p[1]-r:p[1]+r+1])
7      return clip
```

2.2.2 Mesure de similarité de Patches

Lors du remplissage . on a besoin d'avoir une quantité qui nous permettra de comparer le similarité d'un patch de frontière et un autre patch placé ailleurs sur l'image . On a utilisé la norme 2 . en tenant en compte la partie masquée . sur le code on prends la somme des distances euclidiennes sur les 3 canaux

```
1  """def similarity(patch1 , patch2 , maskpatch):
2      """
3          Measures distance between Patches
4      """
5      d=0
6      for i in range(3):
7          d+= np.sum(maskpatch*(patch1[:, :, i]-patch2[:, :, i])**2)
8      return d
```

2.3 Calcul du terme de confiance C

Une fois la frontière déterminée on cherche à définir un ordre de remplissage , on veut bien que cet ordre prends en compte l'information qu'on possède au préalable . si on note I notre image et Ω le masque et ψ_p patch centré sur le point p , alors C est initialisé par 1 pour les pixels p dans $I - \Omega$ dont tout le patch est sur $I - \Omega$ et 0 pour ceux dont le patch est complètement dans Ω . pour les autres pixels on le définit ainsi :

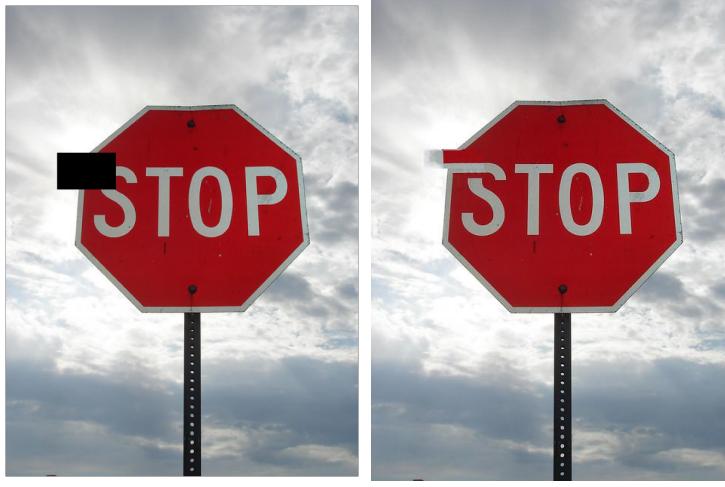
$$C_p = \frac{\sum_{q \in \psi_p \cap (I - \Omega)} C_q}{|\psi_p|}$$

```

1  """
2  def c_matrix(mask):
3      n,m= mask.shape
4      c=np.zeros(mask.shape)
5      for k in range(n):
6          for l in range(m):
7              patch = get_patch(mask,(k,l))
8              c[k,l]=np.sum(patch)/(patch.shape[0]*patch.shape[1])
9      return c

```

Mais ce terme n'est pas suffisant ! en effet on veut que notre algorithme de remplissage prends aussi compte des gradients autour du masque , pour ne pas couper des lignes et mettre des zones homogènes là où on en veut pas , la figure suivante donne un exemple d'un tel cas :



(a) Image multiplié par le masque

(b) Image reconstruite

Figure 2: Reconstruction ne tenant compte que du terme C , on remarque bien que le rouge sort du cadre après le remplissage

2.4 Calcul du terme du terme D

Comme le montre la figure 2 on besoin d'un autre terme qui va prioriser la propagation de structure , et qui est constitué d'un produit scalaire entre le gradient de l'image en points de frontière du masque , avec la normale à cette frontière

$$D = |\Delta I_p^T \cdot n_p|$$

2.4.1 Calcul du gradient I

Pour le calcul du gradient on a choisi de ne le calculer qu'au points de la frontière . on prends à chaque fois un patch centré su p , on calcule le gradient des trois canaux RGB . et c'est le max sur la norme du gradient sur les trois canaux qui est choisi.

```

1  """
2  def gradient_I(image ,mask , bordure):
3

```

```

4     We compute I(p) for all p in delta(\Omega)
5     h, w = image.shape[:2]
6     c=image.copy()
7     c_r= img_as_float(c[:, :, 0])
8     c_g= img_as_float(c[:, :, 1])
9     c_b= img_as_float(c[:, :, 2])
10    # We fill the outside of the mask with nones
11    c_r[mask == 0] = np.NaN
12    c_g[mask == 0] = np.NaN
13    c_b[mask == 0] = np.NaN
14    # We compute the gradient
15    fgradx,fgrady = np.array([np.zeros([h, w]),np.zeros([h, w])])
16    for point in bordure:
17        patch_r=get_patch(c_r,point)
18        patch_g=get_patch(c_g,point)
19        patch_b=get_patch(c_b,point)
20
21        gradientR = np.nan_to_num(np.gradient(patch_r))
22        gradientG = np.nan_to_num(np.gradient(patch_g))
23        gradientB = np.nan_to_num(np.gradient(patch_b))
24
25        normeR= np.sqrt(gradientR[0]**2 + gradientR[1]**2)
26        normeG= np.sqrt(gradientG[0]**2 + gradientG[1]**2)
27        normeB= np.sqrt(gradientB[0]**2 + gradientB[1]**2)
28
29        norme = np.maximum(normeR,normeG,normeB)
30
31        max_patch = np.unravel_index(
32            norme.argmax(),
33            norme.shape
34        )
35        fgradx[point[0], point[1]] = gradientR[0][max_patch]
36        fgrady[point[0], point[1]] = gradientR[1][max_patch]
37
38    return [fgrady,fgradx]

```

En inversant les coordonées x et y on obtient le vecteur normal.

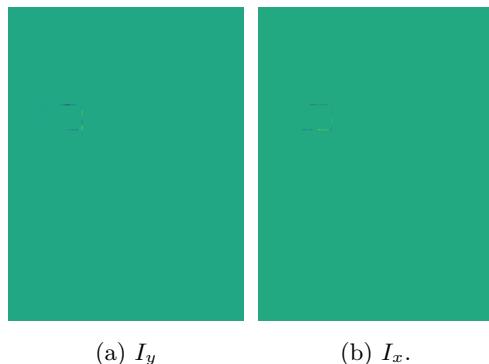


Figure 3: Le vecteur Gradient tracé pour l'exemple de figure 2

2.4.2 Calcul du vecteur normal \vec{n}

```

1  """
2  def normal_vect(image,mask,bord) : #n(p)
3      h, w = mask.shape[:2]
4      coordx , coordy= np.zeros((h, w)),np.zeros((h, w))
5      for p in bord:
6          i,j=p

```

```

7     patch = get_patch(mask,(i,j))
8     grad = len(patch)*len(patch[0])*np.nan_to_num(np.array(np.gradient(patch)
9   )))
10    gradX = grad[0]
11    gradY = grad[1]
12    centerX, centerY = patch.shape[0]//2 ,patch.shape[1]//2
13    coordx[i][j] =gradX[centerX][centerY]
14    coordy[i][j] =gradY[centerX][centerY]
15
16    return coordx,coordy

```

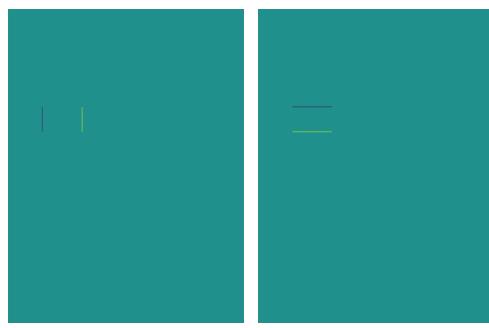


Figure 4: Le vecteur Normal tracé pour l'exemple de figure 2.

2.5 Calcul du terme de priorité

le terme de priorité combine le terme de confiance et celui de data pour nous donner grandeur à prendre en compte pour savoir quels points de la frontière prioriser. il est donné par

$$P_p = C_p D_p$$

```

1  """def P(image,mask,bordure):
2
3      "Computes P for points of the bordure"
4
5      h, w = mask.shape[:2]
6      P=np.zeros((h,w))
7      C=c_matrix(mask)
8
9      #test without C
10     #C=np.ones((h,w))
11     I=gradient_I(image,mask, bordure)
12
13     #Test without I
14     #I=np.array([np.ones((h,w)),np.ones((h,w))])
15
16     N=normal_vect(image,mask,bordure)
17
18     #Test without N
19     #N=np.array([np.ones((h,w)),np.ones((h,w))])
20
21     for (i,j) in bordure:
22         P[i][j]=np.abs(I[0][i][j]*N[0][i][j]+
23                         I[1][i][j]*N[1][i][j])/255 * C[i][j]
24
25     return P

```

2.5.1 Choix d'élément ayant priorité maximale

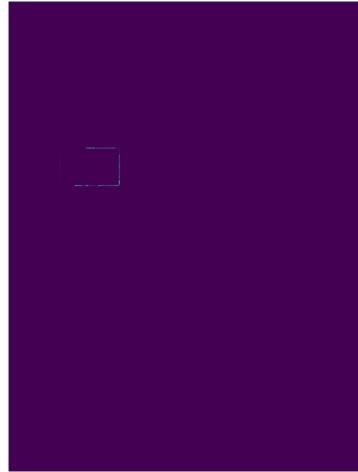


Figure 5: La matrice de priorité

```

1  """def maxP(image,mask,bordure):
2
3      "Finds point with max value of P"
4
5      p=P(image,mask,bordure)
6      maximum=p[bordure[0]]
7      argmax=bordure[0]
8      for point in bordure:
9          i,j=point
10         if(p[i][j]>=maximum):
11             maximum = p[i][j]
12             argmax=point
13     return argmax

```

3 Itération de l'algorithme

```

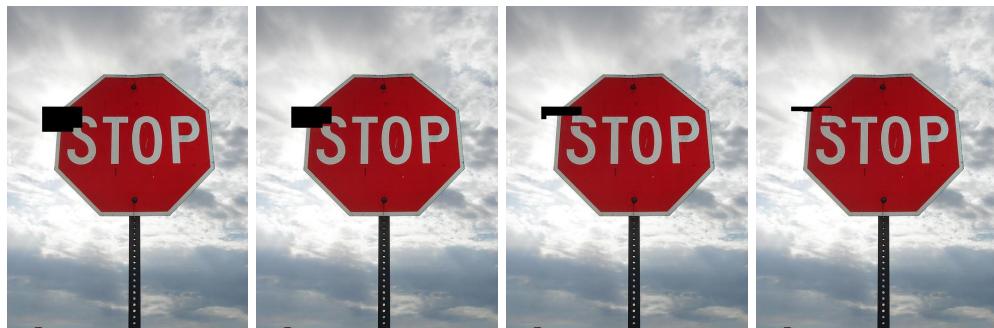
1  """def iterate(image , firstmask , patch_size = 16):
2      "We Iterate the algorithm to fill the masked region"
3      mask=np.copy(firstmask)
4      frt = init_bord_m(mask)
5      new_image= delete_zone(image,mask)
6      view(new_image)
7      K=0
8      ps=patch_size//2
9      while (len(frt)>0):
10          # Look for point with max Priority
11          p_point=maxP(new_image,mask,frt)
12          p_patch=get_patch(new_image,p_point)
13          new_patch = get_patch(new_image, (ps,ps))
14          d = similarity(new_patch,p_patch,get_patch(mask,(ps,ps)))
15          chosenX,chosenY = p_point
16          maskpatch = get_patch(mask,(chosenX,chosenY))
17          # Looking for a patch that is the closest to the content of uncomplete
18          patch
19          for x in range(ps, new_image.shape[0]-ps):
20              for y in range(ps,new_image.shape[1]-ps):
21                  potential = True
22                  fpatch=get_patch(mask,(x,y))
23                  for i in range(patch_size):
24                      for j in range(patch_size):

```

```

24             if (fpatch[i,j]==0):
25                 potential=False
26             if (potential):
27                 testPatch = get_patch(new_image,(x,y))
28                 dtest = similarity(p_patch,testPatch,maskpatch)
29                 if dtest < d :
30                     d = dtest
31                     new_patch = np.copy(testPatch)
32             # Filling and updating the mask
33             for i in range(-ps,ps+1) :
34                 for j in range(-ps,ps+1):
35                     if(mask[chosenX+i,chosenY+j]==0):
36                         new_image[chosenX+i,chosenY+j]= new_patch[ps+i,ps+j]
37                         mask[chosenX+i,chosenY+j]= 1
38             # Updating the border
39             frt=init_bord_m(mask)
40             # Viewing and saving results
41             view(new_image)
42             iio.imwrite("output/"+str(K)+".jpg", new_image)
43             K+=1

```



(a) First iteration.

(b) 10th iteration.

(c) 32th iteration.

(d) 46th iteration.

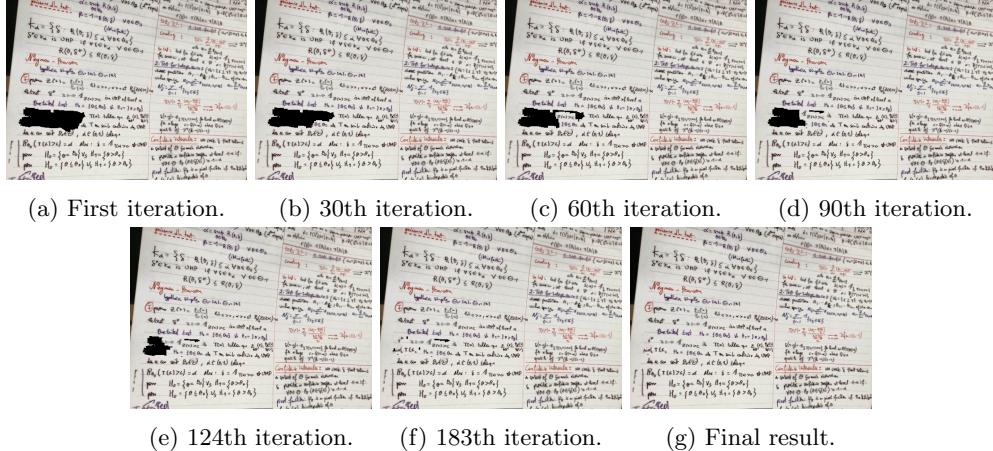


(e) Résultat Final.

4 Exemples et manipulations

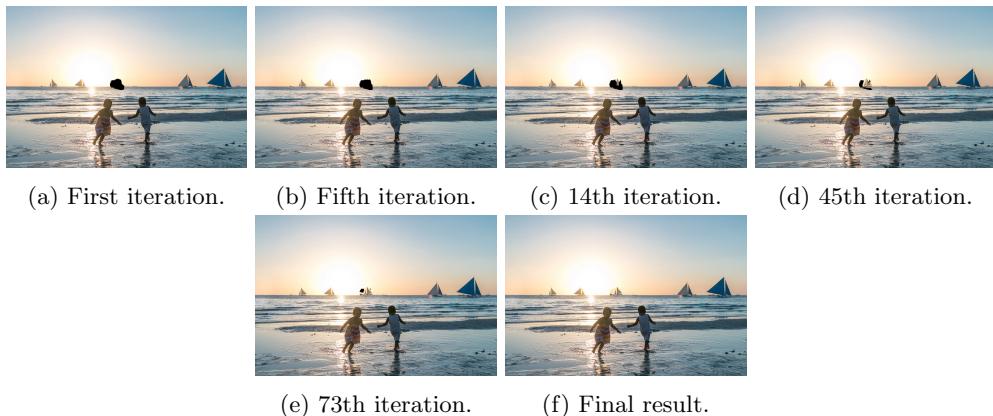
4.1 Complétion de texte manuscrit

Dans cet exemple on voit bien l'effet du terme D , si on se retrouve sur un point de la frontière ou on est susceptibles d'avoir du texte . on commence par cette région là et le remplissage des zones homogènes vient après la complétiōn du texte.



4.2 Sunset Image

Dans cette image on utilise des patches de taille 16 , on remarque l'algorithme a recopié le bateau situé à gauche . mais le remplissage de la zone claire (Soleil) est sombre.



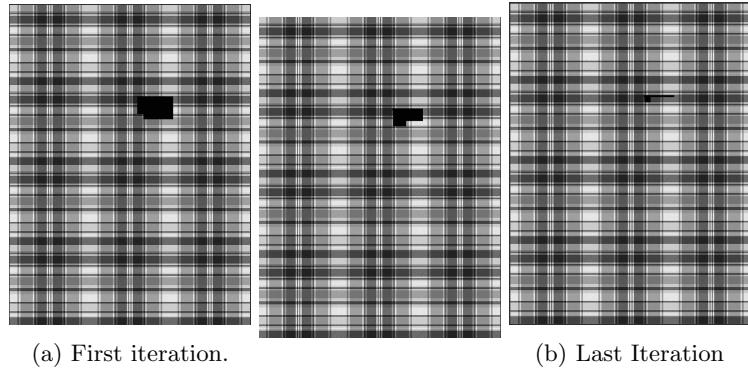
4.3 Complétion de texture

La complétiōn de textures est très efficace avec cette méthode car les textures ont une certaine homogénéité. (Fig. 9)

4.4 limitations

4.4.1 le Tripoint

Dans cet exemple particulier la méthode de recherche de patch ne donne pas un résultat satisfaisant . car tout simplement on n'arrive pas à trouver un patch convainquant qui donne un résultat naturel .



(a) First iteration.

(b) Last Iteration

Figure 9: Remplissage de motif

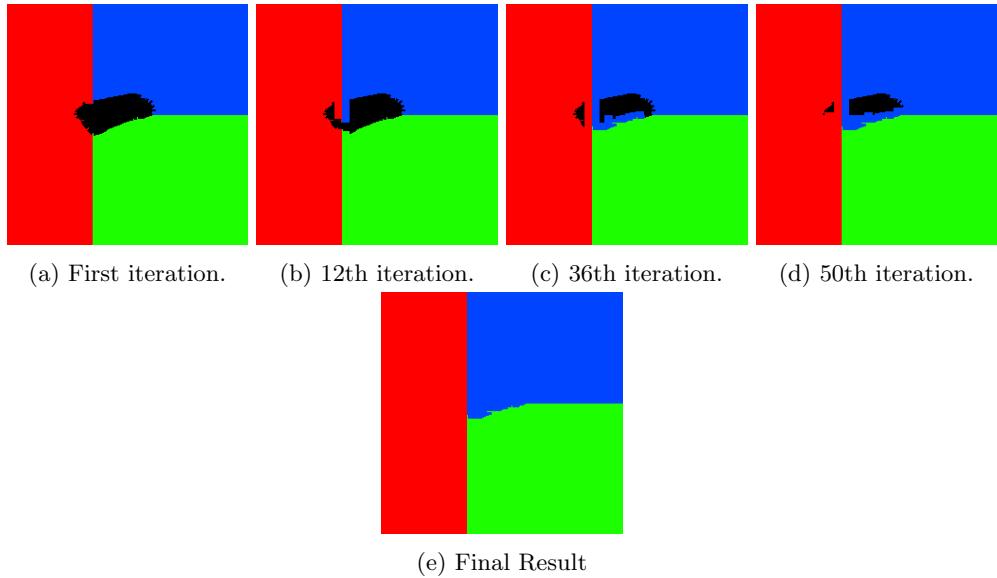


Figure 10: Illustration on the Tripoint problem



Figure 11: Suite d'itération qui nous a donné l'idée de faire des transformations géométriques sur des patches

5 Améliorations

On voit dans l'exemple ci-dessus, que il est impossible de bien remplir le ciel, car la méthode crée des copies et il n'y a aucun patch avec la bonne orientation (un copieur ne peut pas réussir ses examens si ses amis sont absents). Il serait intéressant de pouvoir comparer des patchs avec certaines transformations (rotation, symétrie ...). Pour diminuer la complexité, on pourrait appliquer ses modifications directement à l'image initial et parcourir toutes les copies à chaque fois au lieu d'appliquer les transformations à chaque patch . (Fig.11)

References

- [1] A. Criminisi, P. Perez, and K. Toyama. Object removal by exemplar-based inpainting. In *2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003. Proceedings.*, volume 2, pages II–II, 2003.