RECONNAISSANCE DE KYSTES BENINS OU GRAVES SUR DES IMAGES D'ECHOGRAPHIES MAMMAIRES

Par Arthur LASHERMES et Jaden COUCHOT

INTRODUCTION

Ce projet consiste en la réalisation d'une intelligence artificielle permettant la reconnaissance de kystes et ganglions bénins, graves ou normaux à partir d'images d'échographies mammaires et de masques binaires leur correspondant. Nous avons choisi ce sujet car nous voulions avoir un sujet à la fois concret et utile pour la santé, car cet outil pourrait être utilisé en complément de diagnostic médical.

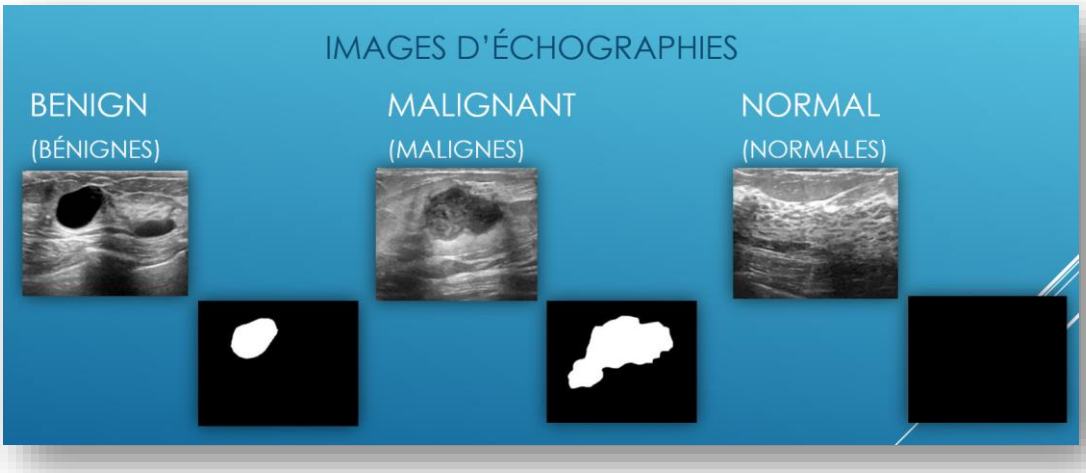
PROBLEMATIQUE

Le but est donc d'obtenir un système permettant la classification d'une échographie ou son masque binaire dans l'une des trois catégories.

SOMMAIRE

I. Base de données

La base de données utilisée est constituée de trois répertoires contenant des images d'échographies mammaires ainsi que de leur masque binaire correspondant.



Le dataset était donc constitué de 891 images d'échographies mammaires bénignes, 421 malignes et 266 normales.

Chargement de données

Les données ont été récupérées dans une liste « data » en parcourant le répertoire cible.

```
data = []
i=0;

nbb = 0
nbm = 0
nbn = 0

# La boucle parcourt le répertoire 'Dataset_BUSI_with_GT/' puis tout ces sous-répertoires afin de placer
# les images dans la liste data
for directory in sorted(listdir('Dataset_BUSI_with_GT/')):
    print(directory)
    for img in sorted(listdir('Dataset_BUSI_with_GT/' + directory + '/')):
        #Récupération des images non-masques et incrémentation du nombre d'images contenant dans les trois
        if ('mask' not in img) :
            if directory == 'benign' : nbb = nbb + 1
            if directory == 'malignant' : nbm = nbm + 1
            if directory == 'normal' : nbn = nbn + 1
            data.append(cv2.imread('Dataset_BUSI_with_GT/' + directory + '/' + img))
```

Nous avons dans un premier temps récupéré uniquement les images qui n'étaient pas des masques pour effectuer une première classification. Nous ne savions pas encore jusqu'où nous irions à ce stade.

```
print('Nombre benin :', nbb, ' --------- rang [ 0 ] à [', nbb-1,']')
print('Nombre malin :', nbm, ' --------- rang [', nbb,'] à [', nbb + nbm-1,']')
print('Nombre normal :', nbn, ' -------- rang [', nbb + nbm,'] à [', nbb + nbm + nbn - 1,']')
```

```
Nombre benin : 437  --------- rang [ 0 ] à [ 436 ]
Nombre malin : 210  --------- rang [ 437 ] à [ 646 ]
Nombre normal : 133  -------- rang [ 647 ] à [ 779 ]
```

Une fois le chargement des données effectué, il a alors été nécessaire de définir les index jalons des différentes images (benign, malignant ou normal) afin de créer une liste « target » donc chaque valeur au rang i correspond à l'image de la liste « data » au même rang.

```
target = []

for i in range (780):
    if (i < 437) : target.append('benign')
    if (i > 436) and (i<647) : target.append('malignant')
    if (i > 646): target.append('normal')

```

Cette liste, nous servira par la suite lors de l'apprentissage de la classification.

Uniformisation des données

Les images étant toutes différentes, il a alors été nécessaire de les uniformiser en les convertissant d'une part en niveau de gris

```
# Cette fonction permet la conversion
def rgb2gray(rgb):
    return np.dot(rgb[...,:3], [0.2125, 0.7154, 0.0721])

datagray = []
for i in range (780):
    datagray.append(rgb2gray(data[i]))
```

Vérification de la conversion

```
print(len(datagray))
print(datagray[0].shape)
plt.imshow(datagray[0], cmap='gray')
```

```
780
(471, 562)
```

Cette conversion a été possible grâce à la fonction rgb2gray ci-dessus.

Dans un second temps, il fallait ensuite redimensionner les images (nous avons alors choisi une définition de 64 x 64) grâce à la bibliothèque cv2, puis de basculer ces images sous formes de tableau à une dimension afin de préparer les données pour l'apprentissage.

```
dataresized = []

def resizeimage(img) :
    return cv2.resize(img,(64, 64), interpolation = cv2.INTER_AREA)

for i in range (780):
    dataresized.append(resizeimage(datagray[i]))
```

La liste datatrain ci-dessous contient les données sous forme d'un tableau à une dimension grâce à la fonction ravel de numpy

```
datatrain = []

for i in range (780):
    datatrain.append(np.ravel(dataresized[i]))
```

II. Classification par les images d'échographies

Une fois les données chargées et uniformisées, nous avons préparé les données à l'apprentissage puis formé les bases d'apprentissage avec train_test_split :

```
# Préparation des données d'apprentissage
X = np.array(datatrain)

# Préparation des données cibles
Y = np.array(target)

print(X.shape)
print(Y.shape)

# Formation des bases d'apprentissage et de test
Xtrain, Xtest, ytrain, ytest = train_test_split(X, Y,test_size=0.25, random_state=0)
```

Test de la meilleure méthode de classification

Afin d'obtenir le meilleur taux de précision possible de classification, nous avons utilisés les 3 modèles vus en cours à savoir l'Arbre de décision, la classification avec le plus proche voisin et la classification avec SVM.

Pour l'arbre de décision, nous avons obtenu un pourcentage de précision environ égal à 62,5%. Pour le KNeighborsClassifier, environ 59,5% et enfin avec SVM, environ 72%. Ces résultats ont été trouvés en modifiant les paramètres des différentes méthodes de classification « au hasard ». C'est pourquoi nous avons décidé d'utiliser les méthodes de GridSearchCV afin d'effectuer une recherche des meilleurs paramètres pour la classification ayant le meilleur taux de précision : SVM.

```
# Définition de la grillle de paramètres à tester
param_grid = {'C': [0.1, 1, 10, 100, 1000],
              'gamma': [1, 0.1, 0.01, 0.001, 0.0001, 0.00001, 0.0000003],
              'kernel': ['rbf']}

grid = GridSearchCV(svm.SVC(), param_grid, refit = True, verbose = 3)

# Raccordement du modèle pour la grid search
grid.fit(Xtrain, ytrain)
```

```
Fitting 5 folds for each of 35 candidates, totalling 175 fits
[CV 1/5] END ........C=0.1, gamma=1, kernel=rbf;, score=0.547 total time=   0.4s
[CV 2/5] END ........C=0.1, gamma=1, kernel=rbf;, score=0.547 total time=   0.4s
[CV 3/5] END ........C=0.1, gamma=1, kernel=rbf;, score=0.556 total time=   0.4s
[CV 4/5] END ........C=0.1, gamma=1, kernel=rbf;, score=0.556 total time=   0.4s
[CV 5/5] END ........C=0.1, gamma=1, kernel=rbf;, score=0.556 total time=   0.4s
```

Nous avons ainsi trouvé les meilleurs paramètres, permettant ainsi d'attendre un taux de précision environ égal à 79% :

```
# Meilleures paramètres obtenus
print(grid.best_params_)

# Paramètres à modifier sur la méthode SVC
print(grid.best_estimator_)
```

```
{'C': 10, 'gamma': 3e-07, 'kernel': 'rbf'}
SVC(C=10, gamma=3e-07)
```

```
grid_predictions = grid.predict(Xtest)

# Affichage des pourcentages de réussites obtenus après la recherche
print(classification_report(ytest, grid_predictions))
```

```
              precision    recall  f1-score   support

      benign       0.77      0.95      0.85       114
   malignant       0.86      0.65      0.74        49
      normal       0.88      0.47      0.61        32

    accuracy                           0.79       195
   macro avg       0.84      0.69      0.73       195
weighted avg       0.81      0.79      0.78       195
```

Classification avec SVM

```
clf = svm.SVC(C=10, gamma=0.0000003)
clf.fit(Xtrain,ytrain)
```

```
SVC(C=10, gamma=3e-07)
```

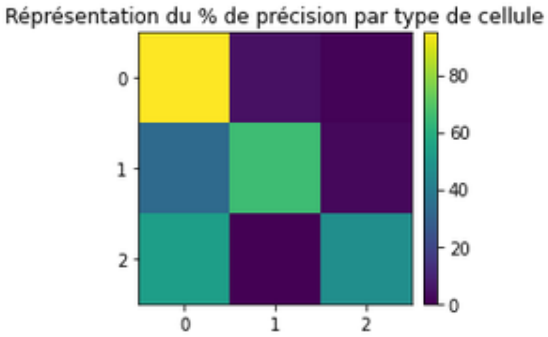
```
ypredit = clf.predict(Xtest)
print('Pourcentage de précision : ', accuracy_score(ytest, ypredit), '%')
```

```
Pourcentage de précision :  0.7948717948717948 %
```

Le pourcentage obtenu ci-dessus est le plus haut pourcentage obtenu en modifiant les paramètres sans vraie méthode

```
plot_confusion_matrix(metrics.confusion_matrix(ytest, ypredit))
```

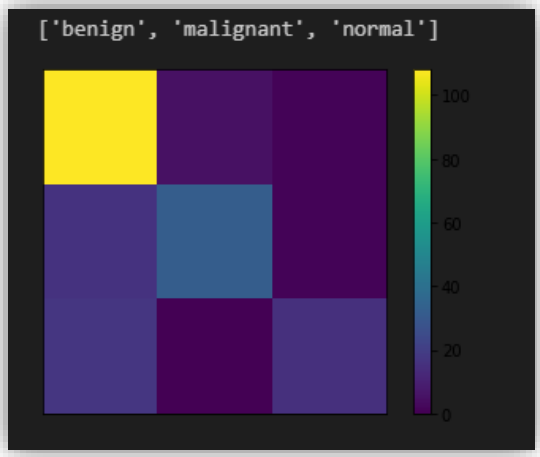
```
[[108   5   1]
 [ 16  32   1]
 [ 17   0  15]]
```



Afin d'avoir un aperçu du résultat plus visuel, nous utilisons la fonction « plot_confusion_matrix » permettant ainsi de voir une matrice de confusion des résultats en pourcentage.

```python
def plot_confusion_matrix(cmx, vmax1=None):
    cmx_norm = 100*cmx / cmx.sum(axis=1, keepdims=True)
    cmx_zero_diag = cmx_norm.copy()

    np.fill_diagonal(cmx_zero_diag, 0)

    fig, ax = plt.subplots(ncols=1)
    fig.set_size_inches(12, 3)
    ax.set_xticks(range(len(cmx)+1))
    ax.set_yticks(range(len(cmx)+1))

    print(cmx)

    ax.set_title('Réprésentation du % de précision par type de cellule')
    im1 = ax.imshow(cmx_norm, vmax=vmax1)

    divider = make_axes_locatable(ax)
    cax1 = divider.append_axes("right", size="5%", pad=0.1)

    fig.colorbar(im1, cax=cax1)
    fig.tight_layout()
```

Affichage :



III. Classification par les masques binaires

Afin de compléter notre projet, nous avons voulu par la suite réaliser une classification en utilisant les masques afin de pouvoir confronter les résultats. Les étapes de préparations pour la classification étaient similaires à celles des images d'échographies c'est-à-dire le chargement des données, la conversion des images en niveau de gris, la création de la liste cible ainsi que le redimensionnement.

Une fois les 3 modèles testés, nous avons là encore gardé la méthode SVM en utilisant les meilleurs paramètres trouvés via GridSearchCV.

Les résultats obtenus furent les suivant :
Arbre de décision : 60 % de précision,
Plus proche voisin : 70,5 % de précision,
Et avec SVM 83.5 % de précision.

Classification avec SVM

```
clfm = svm.SVC(gamma=0.0000001)
clfm.fit(Xtrain,ytrain)
```

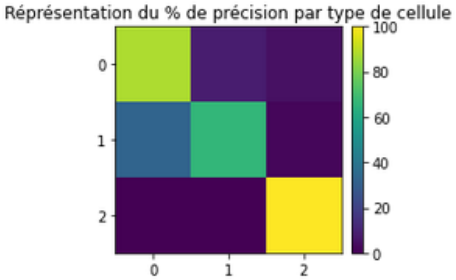
```
SVC(gamma=1e-07)
```

```
ypredit = clfm.predict(Xtest)
print('Pourcentage de précision : ', accuracy_score(ytest, ypredit), '%')
```

```
Pourcentage de précision :  0.835 %
```

```
plot_confusion_matrix(metrics.confusion_matrix(ytest, ypredit))
```

```
[[98  9  5]
 [18 37  1]
 [ 0  0 32]]
```



IV. Test de classification de 6 images : échographies mammaire, bénigne, maligne et normale

Afin de tester nos deux modèles sur des photos de manière plus concrète, nous avons chargé des images de test trouvées sur Google qui ne faisaient pas partie du dataset. Nous avons alors appliqué les méthodes de préparation des données pour ensuite effectuer la prédiction.

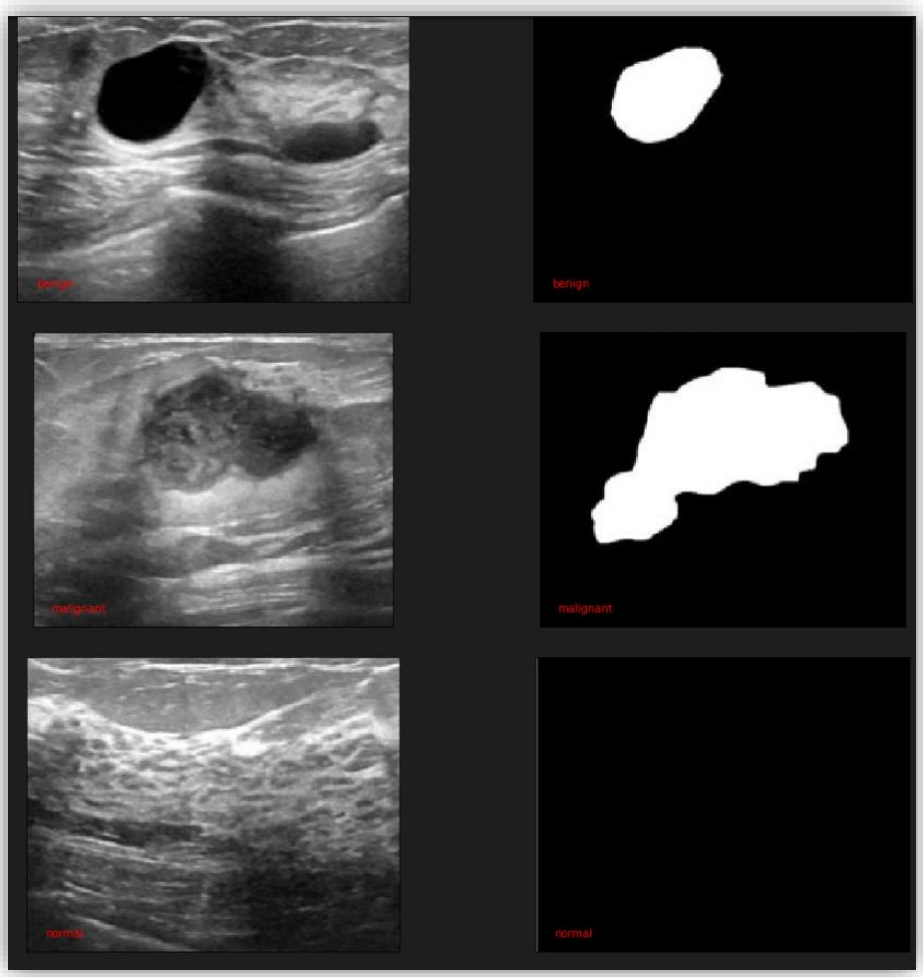
```
image = []
imagemasks = []
allimages = []

# Tri des images dans des listes différentes afin d'utiliser le modèle correspondant
for img in sorted(listdir('ImageTest/')):
        if ('mask' not in img) :
            image.append(cv2.imread('ImageTest/' + img))
        else :
            imagemasks.append(cv2.imread('ImageTest/' + img))

# Liste contenant successivement les images et leur masque
for i in range (3) :
    allimages.append(image[i])
    allimages.append(imagemasks[i])
```

```
imagegrey = []
imageresized = []
prediction = []

for i in range (3) :
    imagegrey = rgb2gray(image[i])
    imageresized = resizeimage(imagegrey)
    prediction.append(clf.predict(imageresized.reshape(1,-1))[0])
    imagegrey = rgb2gray(imagemasks[i])
    imageresized = resizeimage(imagegrey)
    prediction.append(clfm.predict(imageresized.reshape(1,-1))[0])
```

```
print(prediction)
```

```
['benign', 'benign', 'malignant', 'malignant', 'normal', 'normal']
```

Nos résultats sont concluants car on a pu obtenir exactement pour les 6 images (3 images d'échographies et 3 avec le masque correspondant) le type de pathologie correspondant.



On peut alors affirmer que les prédictions sont justes, avec 79,5% de certitude pour les échographies et 83,5% pour les masques binaires

V. Conclusion

Pour conclure, nous sommes globalement satisfais car nous avons su répondre à la problématique que nous nous étions fixé : nous avons créé une intelligence artificielle permettant de donner le type de pathologie d'un kyste avec une certitude très satisfaisante à partir d'images d'échographies. L'utilisation de masques comme données pour la classification a montré une meilleure efficacité que les images de l'échographie.