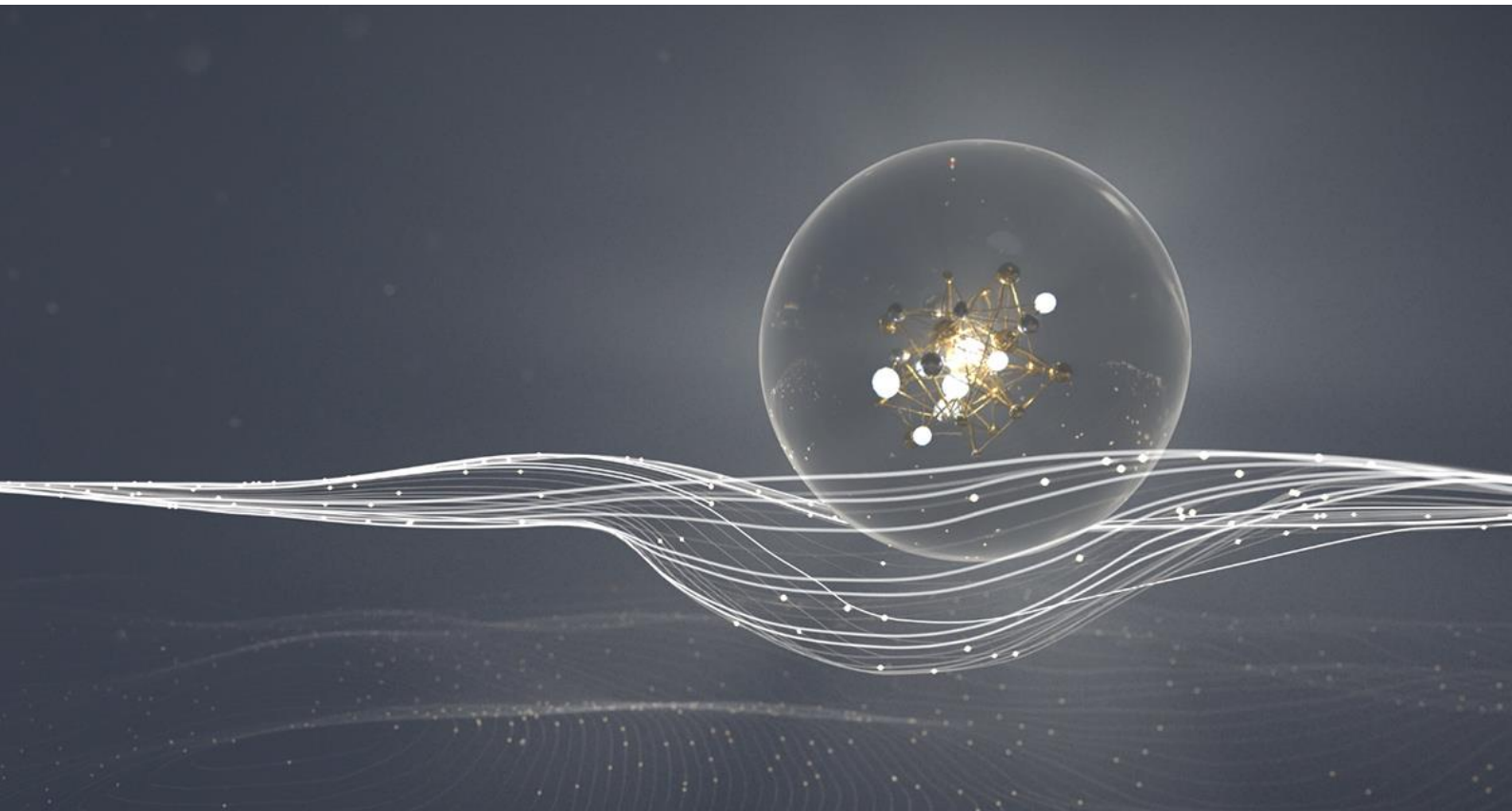




DEEP  
LEARNING  
INSTITUTE



MEDICAL IMAGE *CLASSIFICATION* USING

CONTEXTE.....	3
AMELIORATION IA EXISTANTE .....	4
ESTIMATION CHARGE DE TRAVAIL .....	4
MODELE ACTUEL .....	4
MODELE ACTUEL AMELIORE .....	4
NOUVEAU MODELE : SUPPORT VECTOR MACHINE .....	6
CONCLUSION.....	9
AJOUT D'UNE FONCTIONNALITE A L'APPLICATION EXISTANTE.....	10
ESTIMATION CHARGE DE TRAVAIL .....	10
SELECTION MULTIPLE .....	10
CHARGEMENT ET ENREGISTREMENT DES IMAGES .....	11
UTILISATION DU MODELE ET ENREGSITREMENT DES IMAGES CLASSIFIEES .....	11
AFFICHAGE DES RESULTATS.....	12
NON REGRESSION .....	13
REPO GIT .....	13

## CONTEXTE

Vous venez de rejoindre une start-up MedTech en tant que développeur.se IA.

L'entreprise a remporté un appel d'offre du CHRU de Nancy pour la réalisation d'un POC (Proof Of Concept) d'une solution IA capable de classer les images radios en six catégories.

L'objectif étant de prouver les compétences techniques de votre start-up à mener à bien ce projet et faire adhérer le corps médical au projet Health Data Hub.

Le Health Data Hub étant en cours de mise en œuvre, l'entreprise utilise le dataset MedNIST.

Une première version a été développée en Pytorch (MedNet by NVIDIA) et Flask. Il s'agit d'une interface simple pour sélectionner une image en local et qui lance le modèle pour prédire sa classe.

Vous avez comme mission d'améliorer le modèle MedNet (en modifiant ses paramètres ou en utilisant une autre architecture) et d'ajouter une fonctionnalité qui permet de sélectionner un ensemble d'images et de les classer dans le dossier approprié en utilisant le modèle amélioré.

## AMELIORATION IA EXISTANTE

### ESTIMATION CHARGE DE TRAVAIL

Afin d'améliorer l'IA existante de deep learning et créer une IA de comparaison en machine learning, la charge de travail estimée est d'environ d'une journée.

### MODELE ACTUEL

Le Framework utilisé pour l'entraînement de ce modèle de Deep Learning est PyTorch. C'est une bibliothèque logicielle Python open source d'apprentissage machine qui s'appuie sur Torch développée par Facebook<sup>9</sup>.

Les résultats obtenus par ce modèle sont très bons, on observe une accuracy de plus de 99 % et très bonne précision.

**Prédictions correctes: 5709 sur 5740, soit seulement 31 erreurs.**

Matrice de confusion :

```
[[966  0  5  0  0  1]
 [ 1997  0  4  0  1]
 [ 8  0977  0  0  0]
 [ 0  3 1954  0  1]
 [ 0  0  0 4947  0]
 [ 2  0  0  0 0868]]
```

### MODELE ACTUEL AMELIORE

Afin d'améliorer ce modèle, je vais tenter plusieurs approches :

- Augmentation du ratio t2v (Ratio maximum autorisé de validation à la perte de formation)

⇒ Un entraînement de plus en plus long peut entraîner une amélioration continue de la perte d'entraînement, mais la précision du modèle sur l'ensemble de données de test se stabilise peu de temps après que la validation et la perte d'entraînement commencent à diverger.

## - Modification des hyper paramètres.

⇒ Après plusieurs tentatives, je suis arrivé à améliorer légèrement le modèle avec ces nouveaux paramètres :

### Code Block 10

```
learnRate = 0.015 # Define a learning rate.
maxEpochs = 50 # Maximum training epochs
t2vRatio = 2 # Maximum allowed ratio of validation to training loss
t2vEpochs = 3 # Number of consecutive epochs before halting if validation loss exceeds above limit
batchSize = 50 # Batch size. Going too large will cause an out-of-memory error.
trainBats = nTrain // batchSize # Number of training batches per epoch. Round down to simplify last batch
validBats = nValid // batchSize # Validation batches. Round down
testBats = -(nTest // batchSize) # Testing batches. Round up to include all
CEweights = torch.zeros(numClass) # This takes into account the imbalanced dataset.
for i in trainY.tolist(): # By making rarer images count more to the loss,
    CEweights[i].add_(1) # we prevent the model from ignoring them.
CEweights = 1. / CEweights.clamp_(min=1.) # Weights should be inversely related to count
CEweights = (CEweights * numClass / CEweights.sum()).to(dev) # The weights average to 1
opti = optim.SGD(model.parameters(), lr = learnRate) # Initialize an optimizer
```

Les paramètres qui ont été modifiés sont :

- **Le Learning Rate** (détermine la taille du pas à chaque itération tout en se déplaçant vers un minimum d'une fonction de perte).
- **Le nombre maximum d'époques** (contrôle le nombre de passages complets à travers l'ensemble de données d'apprentissage).
- **Le t2vRatio** (Ratio maximum autorisé de validation à la perte de formation).
- **Le batch size** (contrôle le nombre d'échantillons d'apprentissage à traiter avant la mise à jour des paramètres internes du modèle).

- Modification du nombre et de la taille de convolutions, ainsi que du nombre de couches entièrement connectées.

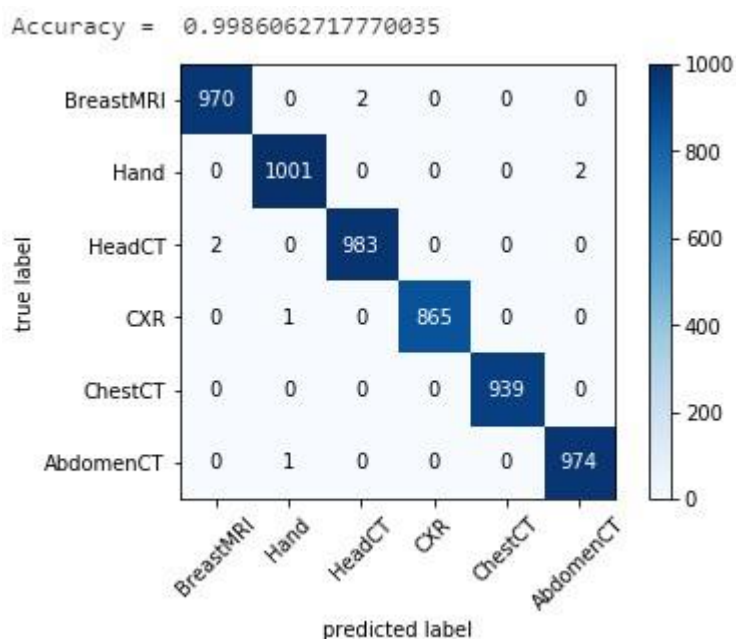
⇒ Après plusieurs tentatives, je ne suis pas arrivé à améliorer significativement le modèle. Exemple de paramètres :

```
numConvs1 = 5
convSize1 = 7
numConvs2 = 10
convSize2 = 7
numNodesToFC = numConvs2*(xDim-(convSize1-1)-(convSize2-1))*(yDim-(convSize1-1)-(convSize2-1))
```

- Résultat du modèle amélioré.

⇒ **Prédictions correctes: 5732 sur 5740, soit seulement 8 erreurs.**

Matrice de confusion :



#### NOUVEAU MODELE : SUPPORT VECTOR MACHINE

Afin d'améliorer la classification du dataset MedNIST, j'ai décidé de tenter ma chance avec un algorithme de machine Learning très utilisé dans la classification d'images : le Support Vector Machine.

Les SVM sont des modèles d'apprentissages supervisés avec des algorithmes d'apprentissage associés qui analysent les données utilisées pour la classification et l'analyse de régression.

## - Preprocessing.

⇒ *Redimensionnement des images*

⇒ *PCA* : L'analyse en composantes est une méthode de la famille de l'analyse des données et plus généralement de la statistique multivariée, qui consiste à transformer des variables liées entre elles (dites « corrélées » en statistique) en nouvelles variables décorrélées les unes des autres. Ces nouvelles variables sont nommées « composantes principales », ou axes principaux. Elle permet au praticien de réduire le nombre de variables et de rendre l'information moins redondante.

⇒ *StandardScaler* : Recalibrage des données pour des répartitions normales.

## - Entraînement du modèle.

⇒ *GridSearchCV* : La fonction GridSearchCV automatise la recherche d'un optimum parmi les hyper paramètres, elle utilise notamment la validation croisée.

```
▶ ▶ M4

param_grid = [
    {'C': [1, 10, 100, 1000], 'kernel': ['linear']},
    {'C': [1, 10, 100, 1000], 'gamma': [0.001, 0.0001], 'kernel': ['rbf']},
]
svc = svm.SVC(probability=True)

model = GridSearchCV(svc, param_grid, verbose=1, n_jobs=-1)

model.fit(X_train, y_train)

model.best_params_

Fitting 5 folds for each of 12 candidates, totalling 60 fits
{'C': 10, 'gamma': 0.001, 'kernel': 'rbf'}
```

- Résultats du modèle.

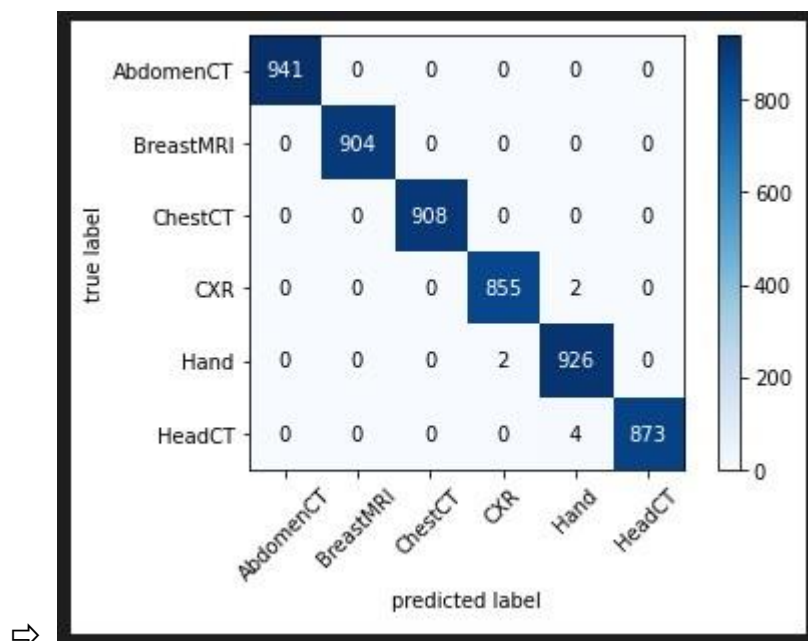
⇒ Accuracy :

0.9985226223453371

⇒ Précision :

	precision	recall	f1-score	support
0	1.00	1.00	1.00	941
1	1.00	1.00	1.00	904
2	1.00	1.00	1.00	908
3	1.00	1.00	1.00	857
4	1.00	0.99	1.00	932
5	1.00	1.00	1.00	873
accuracy			1.00	5415
macro avg	1.00	1.00	1.00	5415
weighted avg	1.00	1.00	1.00	5415

⇒ Matrice de confusion :





Au vue des performances des différents modèles, mon choix se portera sur le SVM qui au niveau de la précision, est très légèrement supérieur au modèle de Deep Learning.

Cependant, à l'instar des SVM qui sont bien adaptés pour les ensembles de données relativement petits avec moins de valeurs aberrantes, les algorithmes de d'apprentissage en profondeur nécessitent des ensembles de données "relativement" volumineux pour bien fonctionner, et vous avez également besoin de l'infrastructure pour les former dans un délai raisonnable.

De plus, les algorithmes d'apprentissage en profondeur nécessitent beaucoup plus d'expérience : la configuration d'un réseau de neurones à l'aide d'algorithmes d'apprentissage en profondeur est beaucoup plus fastidieuse que l'utilisation de classificateurs standard tels les SVM.

D'un autre côté, l'apprentissage en profondeur brille vraiment lorsqu'il s'agit de problèmes complexes tels que la classification d'images complexes, le traitement du langage naturel et la reconnaissance vocale.

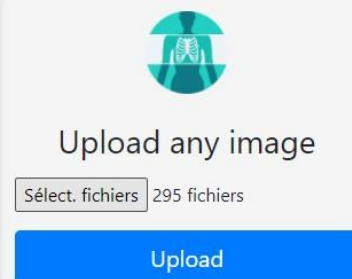
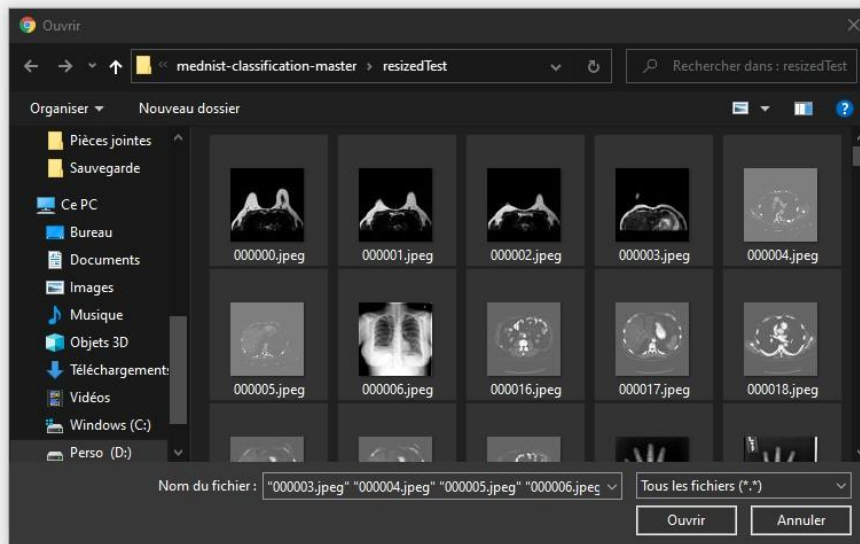
## AJOUT D'UNE FONCTIONNALITE A L'APPLICATION EXISTANTE

### ESTIMATION CHARGE DE TRAVAIL

L'estimation de la charge de travail pour l'ajout de cette nouvelle fonctionnalité est d'environ une journée.

### SELECTION MULTIPLE

```
<body class="text-center">
  <form class="form-signin" method=post enctype=multipart/form-data>
    
    <h1 class="h3 mb-3 font-weight-normal">Upload any image</h1>
    <input type="file" name="file" class="form-control-file" id="inputfile" multiple>
    <button class="mt-3 btn btn-lg btn-primary btn-block" type="submit" onclick="loading();">Upload</button>
    <br/>
    <p class="mb-3 text-muted">Built using Pytorch, Flask</p>
    <div class="col-lg-1" id="loading"></div>
  </form>
```



Built using Pytorch, Flask

```

# liste des images
data = request.files.getlist('file')

# suppression des anciennes images à afficher
for filename in os.listdir(os.path.join(app.config['UPLOAD_FOLDER'])) :
    os.remove(os.path.join(app.config['UPLOAD_FOLDER']) + "/" + filename)

# Sauvegarde des nouvelles images
for f in data :
    f.save(os.path.join(app.config['UPLOAD_FOLDER'], f.filename))
    f.seek(0)

```

```

#Application du modele et enregistrement des images classifiées
list = []
for f in data :
    img_bytes = f.read()
    f.seek(0)
    class_name ,class_id = get_prediction(image_bytes=img_bytes)
    values = Data(f.filename, class_id, class_name)
    list.append(values)

    try:
        os.mkdir(os.path.join(app.config['ImagesClassified']) + "/" + class_name)
    except:
        print('Dossier existant')

    f.save(os.path.join(app.config['ImagesClassified'] + "/" + class_name, f.filename))
    f.seek(0)

```

```









<h1 class="h3 mb-3 font-weight-normal">Prédictions</h1>
<a>Répertoire : ./ImagesClassified/</a>
</div>
</div>
<br>
<br>
<div class="row">
<div class="col-lg-12">
<table class="table table-dark table-striped text-center fixed">
<thead>
<tr>
<th>Images</th>
<th>Images Names</th>
<th>XRy Class ID</th>
<th>Detected XRy Class</th>
</tr>
</thead>
<tbody>
{%for i in range(0, len)%}
<tr>
<td></td>
<td>{{ list[i].image }}</td>
<td>{{ list[i].class_id }}</td>
<td>{{ list[i].class_name }}</td>
</tr>
{%endfor%}
</tbody>
</table>

```




## Prédictions

Répertoire : ./ImagesClassified/

Images	Images Names	XRy Class ID	Detected XRy Class
	000000.jpeg	1	BreastMRI
	000001.jpeg	1	BreastMRI
	000002.jpeg	1	BreastMRI
	000003.jpeg	1	BreastMRI
	000004.jpeg	2	ChestCT
	000005.jpeg	2	ChestCT
	000006.jpeg	3	CXR

## NON REGRESSION

L'ajout de la nouvelle fonctionnalité n'a en rien occasionné de régression à l'application d'origine. Il est toujours possible de faire une prédiction pour une seule image.




Upload any image

Sélect. fichiers 000006.jpeg


Upload

Built using Pytorch, Flask



Prédictions

Répertoire : ./ImagesClassified/

Images	Images Names	XRy Class ID	Detected XRy Class
	000006.jpeg	3	CXR

## REPO GIT

<https://github.com/Damosm/mednist-classification-master>