ZOIDBERG 2.0

T-DEV-810_MSC2023

Problématique du projet:

À partir d'un dataset d'images de radiographie des poumons, nous devions produire un algorithme capable de prédire en analysant une image si celle-ci peut révéler des symptômes de pneumonies.
Notre dataset est composé de 5216 images d'entrainement, dont 1341 ne présentant pas de symptômes de maladies et 3875 détectés comme étant représentatives de pneumonie, 624 images de tests composés de 234 sans symptômes et 390 détectés comme pneumonie, et 16 images dans le dataset val.

Déroulement du projet:

Dans un premier temps, nous nous sommes renseigné sur comment procéder pour pouvoir répondre à notre problématique.

Nous avons suivi de nombreux cours et vidéos en ligne, dont cette playlist pour comprendre le concept des réseaux convolutifs:
https://www.youtube.com/playlist?list=PLII0-qAzf2SbIELzNxxIV2Sku_nCpL2Qv

Pour le premier modèle, nous avons développé un model avec tensorflow contenant 3 couches de convolution 2D et une couche dense.
Nos images démarraient avec une résolution de 180px par 180px pour finir avec une taille de 22px par 22px.

```
model = Sequential([
    layers.Rescaling(1. / 255, input_shape=(img_height, img_width, 1)),
    layers.Conv2D(16, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(32, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(64, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(num_classes)
])
```

Nous avons choisi de définir le paramètre padding à same pour garder les contours de notre image. Cela peut nous être utile afin d'avoir une taille d'image qui permet une couche de convolution supplémentaire avant que celle-ci ne soit trop petite pour être utilisé.

En voulant tester notre modèle, nous nous sommes rendu compte qu'avec la quantité d'images à traiter, notre fonction d'entrainement prenait beaucoup de temps à être exécuté.
Nous avons donc mis en place un scaling de nos données. Nous avons définit ce scale à 0.1 pour avoir seulement 10% de notre dataset qui soit traités.

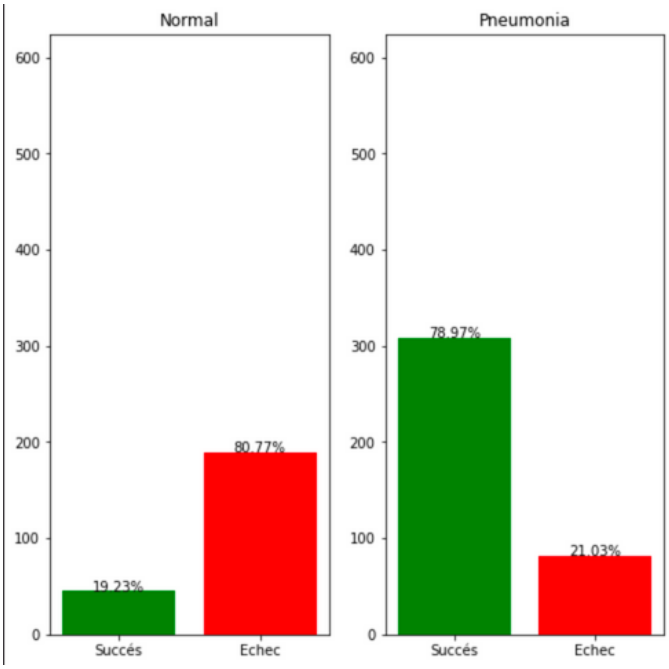
```
Epoch 1/10
163/163 [======
Epoch 2/10
163/163 [======
Epoch 3/10
163/163 [======
Epoch 4/10
163/163 [======
Epoch 5/10
163/163 [======
Epoch 6/10
163/163 [======
Epoch 7/10
163/163 [======
Epoch 8/10
163/163 [======
Epoch 9/10
163/163 [======
Epoch 10/10
163/163 [======
```

```
Epoch 1/10
17/17 [==========
Epoch 2/10
17/17 [==========
Epoch 3/10
17/17 [==========
Epoch 4/10
17/17 [==========
Epoch 5/10
17/17 [==========
Epoch 6/10
17/17 [==========
Epoch 7/10
17/17 [==========
Epoch 8/10
17/17 [==========
Epoch 9/10
17/17 [==========
Epoch 10/10
17/17 [==========
```

Ici, nos epochs sont donc exécutées beaucoup plus rapidement, ce qui, dans le cadre de la vérification du code, nous ai plus utile.

Nous pourrons revenir a un scale de 1 plus tard pour entrainer notre modèle sur la totalité de nos images.

Après test avec un scale de 1, notre premier notebook n'a pas eu de très bons résultats, notament avec un taux de faux-négatifs de 80%, ce qui est très grave. Nous ne pouvons pas dire à une personne qu'elle n'est pas malade alors qu'elle l'est en réalité.



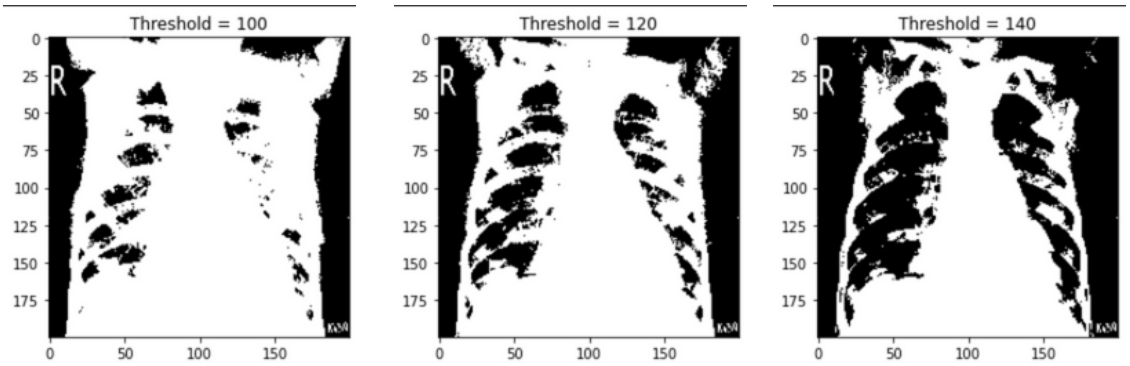
Nous avons donc cherchés des moyens d'augmenter la précision de notre algorythme en ajoutant d'autres modèles et en mettant en place de l'augmentation de données.

Nous avons donc récupéré et édité 2 nouveaux modèles:

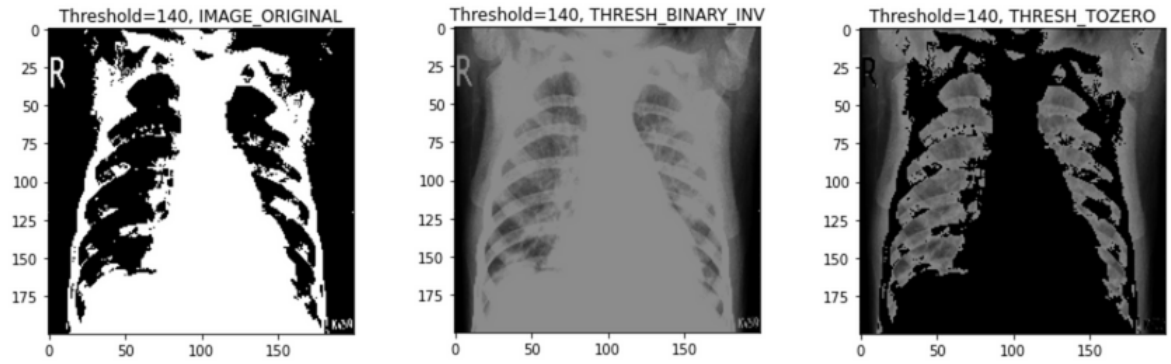
```python
def model_v1(img_size):
    model_v1 = Sequential()

    model_v1.add(Conv2D(32, (3,3), strides=1, padding='same', activation='relu', input_shape=img_size))
    model_v1.add(BatchNormalization())
    model_v1.add(MaxPool2D((2,2), strides=2, padding='same'))

    model_v1.add(Conv2D(64, (3,3), strides=1, padding='same', activation='relu'))
    model_v1.add(Dropout(0.2))

    model_v1.add(BatchNormalization())
    model_v1.add(MaxPool2D((2,2), strides=2, padding='same'))
    model_v1.add(Conv2D(64, (3,3), strides=1, padding='same', activation='relu'))

    model_v1.add(BatchNormalization())
    model_v1.add(MaxPool2D((2,2), strides=2, padding='same'))
    model_v1.add(Conv2D(128, (3,3), strides=1, padding='same', activation='relu'))
    model_v1.add(Dropout(0.2))

    model_v1.add(BatchNormalization())
    model_v1.add(MaxPool2D((2,2), strides = 2, padding='same'))

    model_v1.add(Flatten())
    model_v1.add(Dense(units=128, activation='relu'))
    model_v1.add(Dropout(0.2))
    model_v1.add(Dense(2, activation='softmax'))

    return model_v1
```

```python
def model_v2(img_size):
    model_v2 = Sequential()
    model_v2.add(Conv2D(32, (3, 3), activation='relu', input_shape=img_size))
    model_v2.add(MaxPool2D((2, 2)))
    model_v2.add(Dropout(0.2))

    model_v2.add(Conv2D(64, (3, 3), activation='relu'))
    model_v2.add(MaxPool2D((2, 2)))
    model_v2.add(Dropout(0.2))

    model_v2.add(Conv2D(64, (3, 3), activation='relu'))
    model_v2.add(MaxPool2D((2, 2)))
    model_v2.add(Dropout(0.2))

    model_v2.add(Conv2D(128, (3, 3), activation='relu'))
    model_v2.add(MaxPool2D((2, 2)))

    model_v2.add(Flatten())
    model_v2.add(Dense(120))
    model_v2.add(Dropout(0.4))
    model_v2.add(Dense(2, activation='softmax'))

    return model_v2
```

Et pour augmenter notre quantité de données, nous avons testés la mise en place d'un threshold qui est une technique permettant de sépare l'arrière plan du premier plan, soit en modifiant la granularité du threshold:



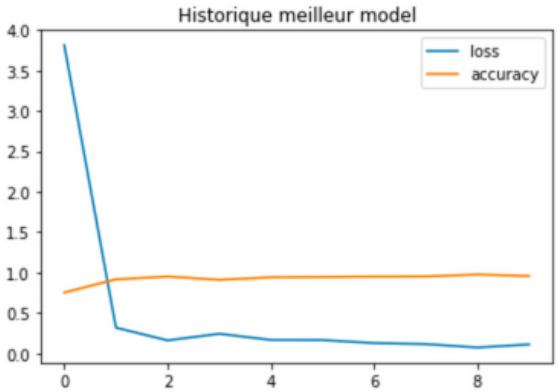
Soit en modifiant les techniques de thresholding:



Autre technique que nous avons tenté est d'utiliser la fonction *ImageDataGenerator* de tensorflow afin d'appliquer des transformations choisies de façon aléatoire à nos images et de remplacer notre dataset avec des images transformées:

```python
def data_argumentation(datagen=None):
    if datagen is None:
        return ImageDataGenerator(
                    featurewise_center=False,
                    samplewise_center=False,
                    featurewise_std_normalization=False,
                    samplewise_std_normalization=False,
                    zca_whitening=False,
                    rotation_range = 30)
    return datagen
```

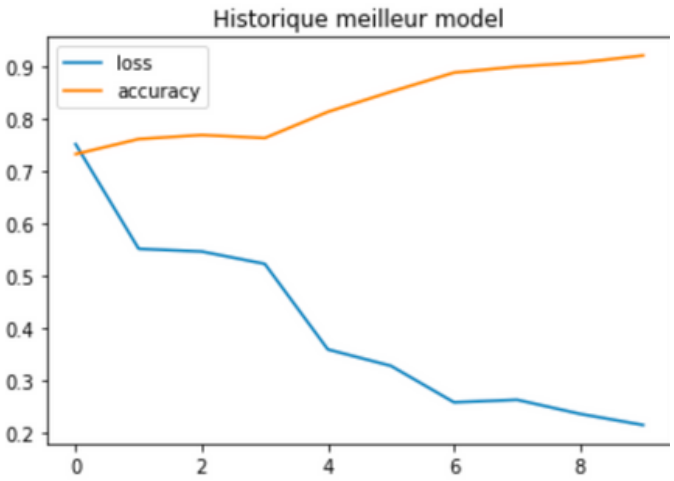
Ainsi, on peut voir qu'avec notre premier modèle nous arrivons très rapidement au plateau de notre model à partir de la 2e epoch mais que notre précision n'est pas très bonne.



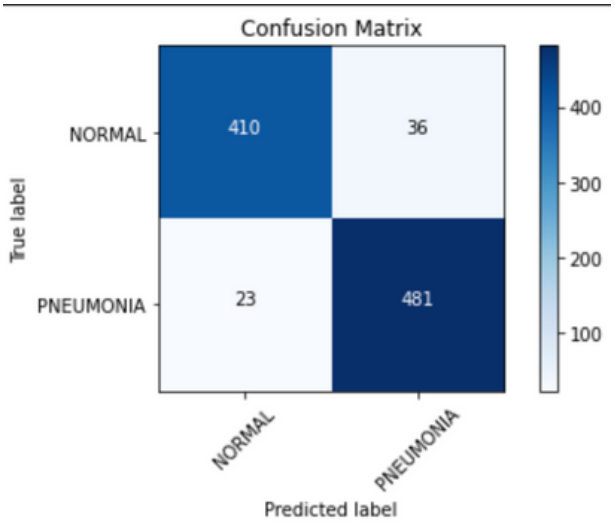
	precision	recall	f1-score	support
PNEUMONIA	0.65	1.00	0.78	40
NORMAL	0.00	0.00	0.00	22
accuracy			0.65	62
macro avg	0.32	0.50	0.39	62
weighted avg	0.42	0.65	0.51	62

Mais qu'avec notre second model qui nécessite plus de temps et d'epoch d'exécution, nous arrivons a des résultats beaucoup plus précis.

	precision	recall	f1-score	support
PNEUMONIA	0.77	0.93	0.84	40
NORMAL	0.79	0.50	0.61	22
accuracy			0.77	62
macro avg	0.78	0.71	0.73	62
weighted avg	0.78	0.77	0.76	62



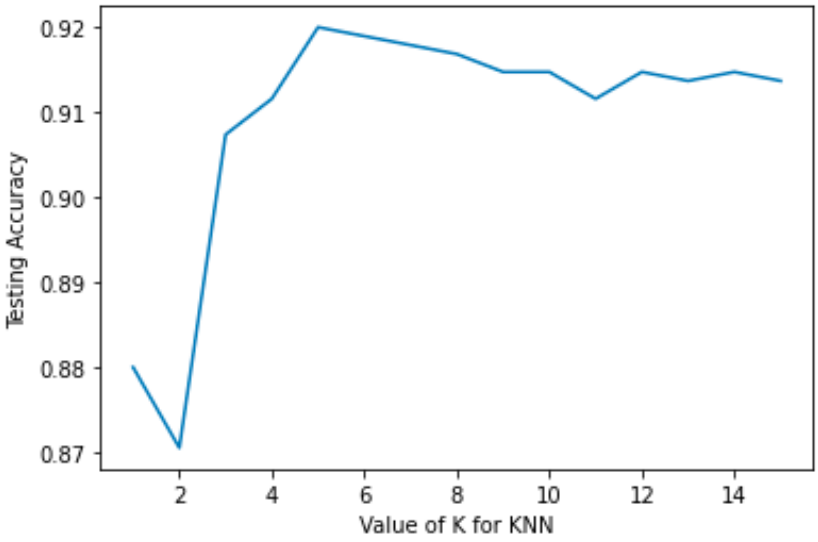
Cependant, les résultats ne sont pas très parlant de la sorte. Nous avons donc ajouter des matrices de confusion et en même temps retravailler le model. Ainsi avec ce nouveau model, nous obtenons des résultats plus concluant et plus parlant visuellement:



	precision	recall	f1-score	support
NORMAL	0.92	0.95	0.93	433
PNEUMONIA	0.95	0.93	0.94	517
accuracy			0.94	950
macro avg	0.94	0.94	0.94	950
weighted avg	0.94	0.94	0.94	950

Pour avoir une vue d'ensemble et d'autres résultats comparatifs, nous avons ensuite testé d'autres algorithmes.
Pour ca, nous sommes parti sur du KNN (k-nearest neighbors).
Nous avons utilisé la librairie sklearn pour ca.
A partir d'une boucle, nous avons put déterminer un nombre de voisin dont la précision de l'algorithme knn était la plus optimisée:

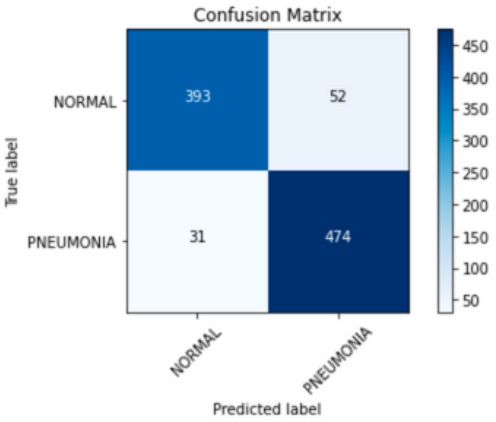
```
for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(x_train, label_train)
    y_pred = knn.predict(x_test)
    scores.append(metrics.accuracy_score(label_test, y_pred))
```



Pour l'équité de l'entrainement de nos algorithmes, nous avons équilibré le dataset (50% de pneumonie, 50% de normal).

De plus, pour avoir un autre levier de comparaison, nous avons utilisé l'ACP (PCA: Principal Composent Analysis), un autre principe de traitement qui permet de réduire notre jeu de données en gardant un taux de variance supérieur à 95%, permettant de gagner du temps tout en gardant la précision.

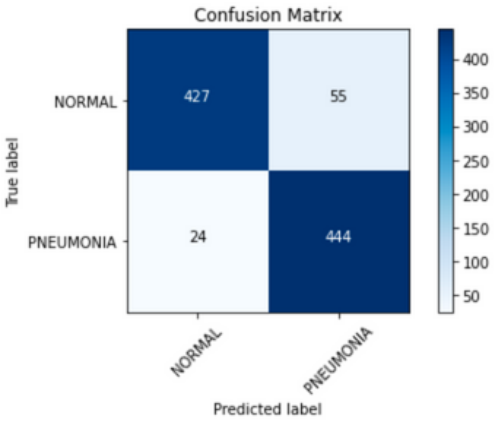
Avec ACP:



	precision	recall	f1-score	support
NORMAL	0.88	0.93	0.90	424
PNEUMONIA	0.94	0.90	0.92	526
accuracy			0.91	950
macro avg	0.91	0.91	0.91	950
weighted avg	0.91	0.91	0.91	950

0:01:40.715280

Sans ACP:



	precision	recall	f1-score	support
NORMAL	0.89	0.95	0.92	451
PNEUMONIA	0.95	0.89	0.92	499
accuracy			0.92	950
macro avg	0.92	0.92	0.92	950
weighted avg	0.92	0.92	0.92	950

0:01:39.658456

Nous pouvons donc voir que pour notre exemple, les temps d'exécution ainsi que les résultats de nos matrices de confusion sont relativement similaires. Mettre en place de l'ACP pour notre jeu de données ne serait pas pertinent.

Conclusion:

Pour conclure, nous avons put, avec ce premier projet, découvrir différents modèle d'algorithmes ainsi que des notions essentiels pour la compréhension du fonctionnement du deep/machine Learning.
Ainsi, dans le cadre de ce projet, l'utilisation de l'algorithme KNN est rapide en exécution mais moins précis dans ces résultats, alors que algorithme CNN est plus lourd et long dans son execution mais a une précision bien plus
importante.