Mona Mokart

COMPTE RENDU CHALLENGE IMA205

CLASSIFICATION D'IMAGES DE CELLULES

L'objectif de ce challenge est de trouver les meilleurs modèles possibles pour classer des images de cellules. Ce projet est composé de deux challenges : un challenge de classification binaire et un challenge de classification à classes multiples. Cela s'est fait via Kaggle. On avait à disposition des images de cellules ainsi que les masques des noyaux et des cytoplasmes. Mon travail détaillé sous forme de code se trouve dans le fichier jupyter notebook « ChallengeIMA205_MonaMokart.ipynb »

Sommaire :

I – Travail sur les features/data

Tout d'abord, j'ai travaillé sur des features crées artificiellement à partir des images. J'y ai extrait :

- l'aire du noyau, l'air du cytoplasme, le rapport noyau/cytoplasme
- la tension et la courbure du noyau et du cytoplasme
- pour chaque canal : moyenne, variance, et gradient du noyau et du cytoplasme

Cela fait 25 features. Evidemment il y avait des cellules dont le masque du noyau était entièrement noir, ce qui aurait pu poser un problème, dans ces cas-ci j'ai attribué une valeur de 0 pour les features correspondantes. Pour chaque canal j'ai effectué une égalisation d'histogramme pour éviter d'avoir des informations biaisées si deux images n'ont pas le même éclairage. Pour éviter de devoir effectuer le code de création des features à chaque fois que je redémarrais le kernel, j'ai enregistré les features sous un fichier csv « features_train.csv », « features_test.csv ». J'ai évidemment travaillé sur des features normalisées afin d'éviter des problèmes avec de nombreux classifiers qui sont influencés par les différences de proportions entre les features (SVM, KNN) et mélangé les images avant d'appliquer des modèles.

J'ai par la suite amélioré les features pour avoir de meilleurs résultats en en créant de nouvelles grâces à des fonctions de la bibliothèque skimage.measure. J'ai pu ajouter ces nouvelles features:

- Le décalage de la position des centroïdes des noyaux et des cytoplasmes
- L'excentricité des noyaux et des cytoplasmes
- L'axe majeur, mineur, le périmètre, l'orientation
- Et pour chaque canal j'y ai ajouté la valeur max et min

J'avais un total de 49 features que j'ai enregistré sous le nom de « features_train2.csv », et « features_test2.csv ».

Ensuite, j'ai utilisé les images normalisées et augmentées pour les algorithmes de deep learning.

Finalement j'ai testé d'implémenter des deep features comme ce qui est proposé dans l'articles1 (*https://ieeexplore.ieee.org/document/8451588*), c'est-à-dire extraire des vecteurs d'un réseau de neurones dont l'entrée est l'image. J'ai testé avec VGG19. J'ai également tenté l'algorithme de PCA sur les images redimensionnées en vecteurs pour y appliquer des modèles qui avaient fonctionné avec les premières et deuxièmes features.

II– Premiers modèles, premières stratégies

Ma stratégie première était d'appliquer les algorithmes de machine learning dans l'ordre vu en classe, et puis confirmer et sélectionner les meilleurs pour les améliorer.

1. LDA et QDA

Binary

	LDA	QDA
Train	0.798283	0.696799
Test	0.777810	0.653390

Multiclass

	LDA	QDA
Train	0.680676	0.727094
Test	0.59987	0.513790

Pour la binary classification, j'ai eu des bons premiers résultats mais pas suffisants, ces modèles devaient être trop simples car linéaires, on était donc en underfitting.

Pour la multiclass classification, j'avais de bons résultats sur LDA surtout car puisque le modèle était simple, je n'avais presque pas d'overfitting. Avec QDA j'avais un bon training score, cependant puisqu'on suppose que les 9 casses n'ont pas les mêmes covariances cela ajoute des paramètres et donc de la complexité, ce qui a créé de l'overfitting je pense.

2. Modèles avec cross validation : KNN et SVM

Binary

	KNN	Lin SVM	SVM (rbf)
Train	0.84344	0.838846	0.842674
Test	0.73830	0.756630	0.810230

Multiclass

	KNN
Train	0.72039
Test	0.51379

J'ai par la suite appliqué des modèles avec des hyperparamètres et appliqué de la cross validation pour trouver les meilleurs paramètres et éviter l'overfitting.

J'ai appliqué KNN aux deux problèmes de classifications. En binaire les résultats n'étaient pas meilleurs. Il est possible que ce modèle ne convienne pas à la classification en deux classes avec ces données et que certaines données de la classe 0 se trouvent eu milieu de la classe 1 (spatialement parlant avec les features comme coordonnées).

Pour la classification multiple il semble là aussi que KNN fasse de l'overfitting et qu'il faille rester sur des modèles simples.

Linear SVM : le linear SVM est un classifier très simple et efficace qui sépare les données par un hyperplan, cependant il n'est pas efficace quand les données ne sont pas linéairement séparables. Ici on peut voir que c'est le cas puisque les résultats du linear SVM sont moins bon que le SVM à noyau gaussien qui lui, permet de séparer non-linéairement les données. Le SVM gaussien est donc le meilleur ici car il semble que les données se séparent de façon complexe parmi les deux classes.

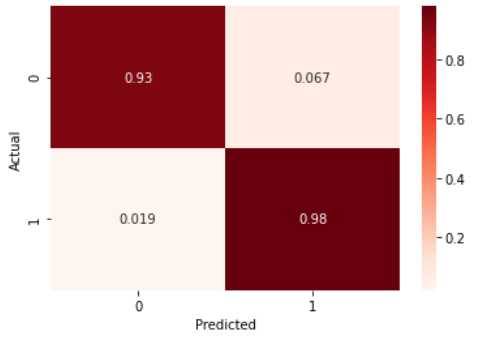
III– Amélioration des modèles et méthodes d'ensemble

Après avoir un peu ciblé les modèles intéressants, j'ai voulu être plus minutieuse en créant de nouvelles features (features_train2.csv, features_test2.csv) et en séparant la data en mini training set et mini testing set. J'ai choisi les meilleurs paramètres en faisant de la cross validation sur le mini training set, puis j'ai confirmé sur le mini testing set avant d'appliquer le modèle sur le training set initial (pour avoir plus de données possibles)

1. Nouvelles features

J'ai testé ces nouvelles features avec SVM pour la binary classification et avec LDA pour la classification multiple, c'est-à-dire les modèles avec les meilleurs résultats précédemment.

Binary : SVM



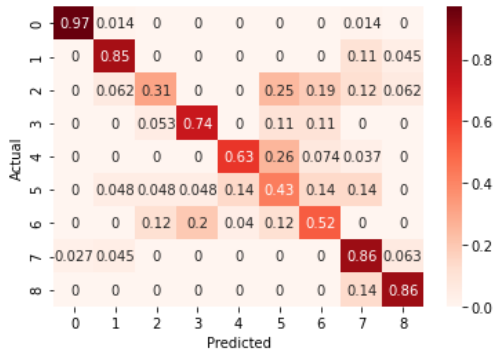
Training score : 0.965375

Testing score : 0.808840

On peut voir qu'ici le problème est la variance et non le biais, on est donc dans le cas d'overfitting. Les résultats ne sont pas meilleurs sur le testing set, ainsi les nouvelles features n'était pas utiles à la classification binaire par svm

Multiclass : LDA

Training score : 0.723998

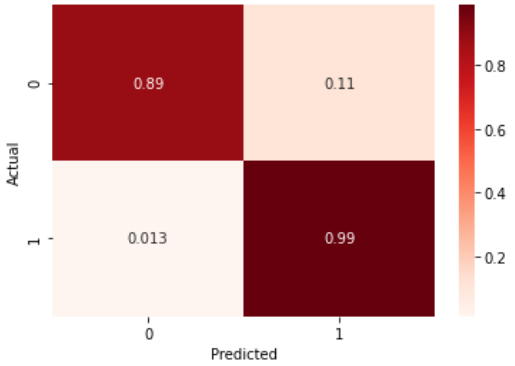


Testing score : 0.706840

Ici les Nouvelles features ont amélioré le score, peut-être car LDA reste un modèle simple et y ajouter des données le complexifie un peu plus. Cependant on peut voir qu'au niveau des classes du milieu, il y a des confusions, cela ne ressemble pas à de la variance mais plus à un biais. Peut-être que c'est dû à la simplicité du modèle qui n'arrive pas ç bien distinguer ces classes-ci.

2. Méthodes d'ensembles

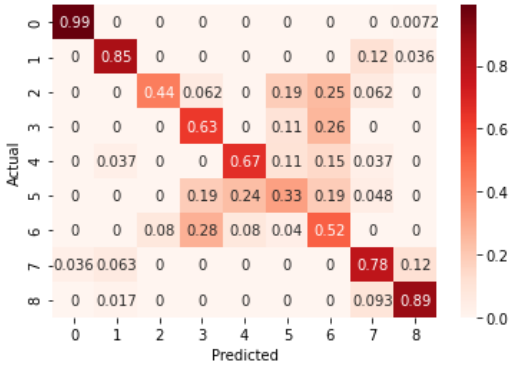
Random Forest est censé éviter la variance des prédictions, voici les résultats :



Training score : 0.993077

Testing score : 0.801210

Cela n' a pas un résultat meilleur qu'avec SVM. Il y a beaucoup d'overfitting car les paramètres sélectionnés n'aident pas : 2 comme nombre minimum dans une feuille est trop petit, les arbres seront trop grands et s'adaptent trop au training set et perd de la puissance de généralisation

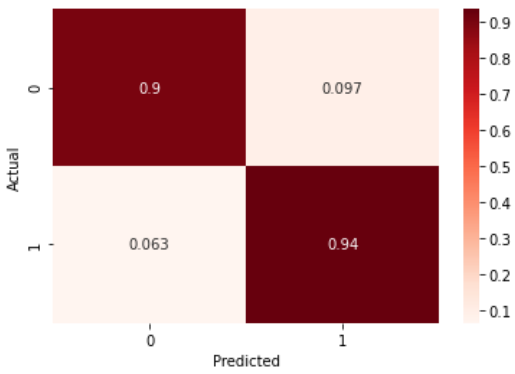


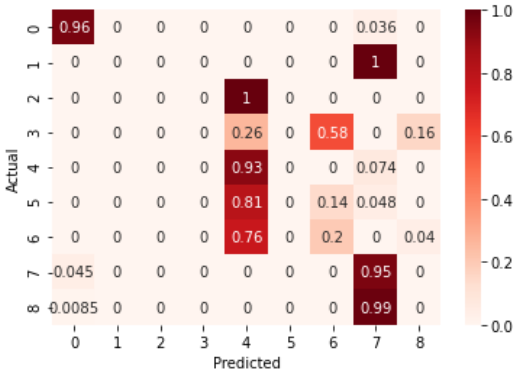
Training score : 0.997679

Testing score : 0.671710

Cela reste correct mais il y a beaucoup d'overfitting

J'ai également tenté le boosting avec AdaBoost mais les résultats n'étaient pas concluants du tout, j'en ai déduit que le décision tree initial était très mauvais où que le learning rate n'était pas bien choisi :





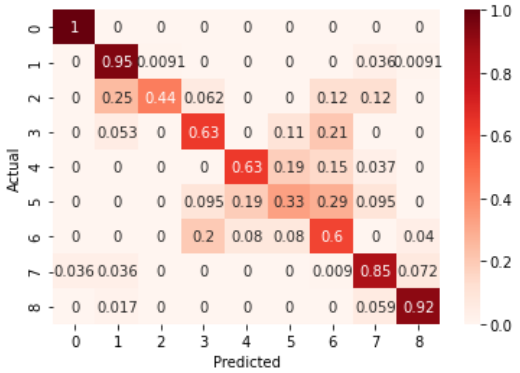
Training score : 0.939798

Testing score : 0.786340

La méthode de bagging comme RF semble être meilleure que le boosting dans ce cas là

3. SVM one vs Rest

Puisque SVM fonctionnait bien pour la classification binaire, j'ai tenté avec la classification multiple :



Training score : 0.83768

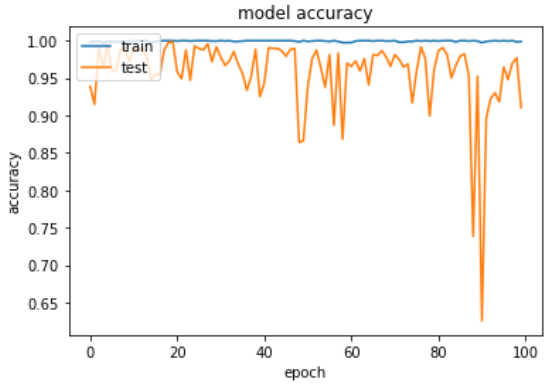
Testing score : 0.69823

Il y a beaucoup d'overfitting mais le testing score reste l'un des meilleurs. Le SVM a pu être un peu plus complexe ce qui a créé de l'overfitting en ayant quand même un bon résultat final.

IV– Deep Learning

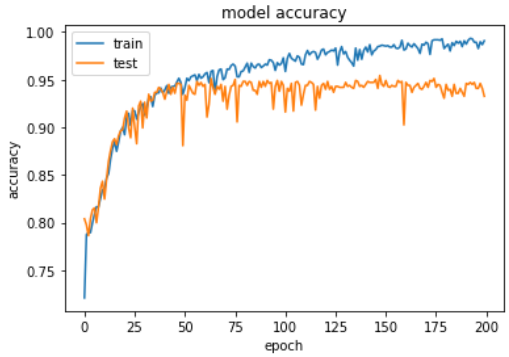
1. MLP

J'ai décidé d'appliquer un modèle MLP sur les features 2 et non sur les images pour gagner du temps de calcul. J'ai fait un premier modèle sans optimiser les paramètres. Cela m'a donné des bons scores mais c'était de la chance. En effet le learning rate, le batch size étaient trop grand et l'accuracy oscillait beaucoup trop :



J'ai eu de la chance avec la classification binaire avec un testing score de 0.846770 mais pas pour la classification multiple avec un score de 0.668700.

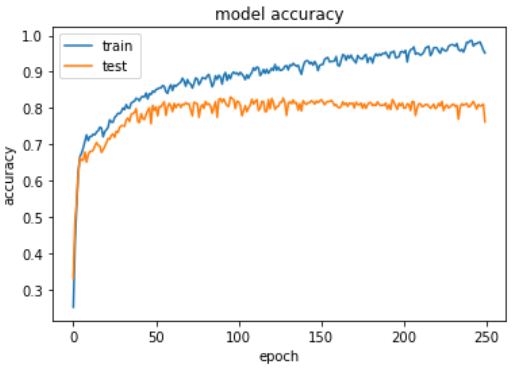
J'ai donc créé un modèle plus sûr :



Binary

Training score : 0.955669

Testing score : 0.846350



Multiclass

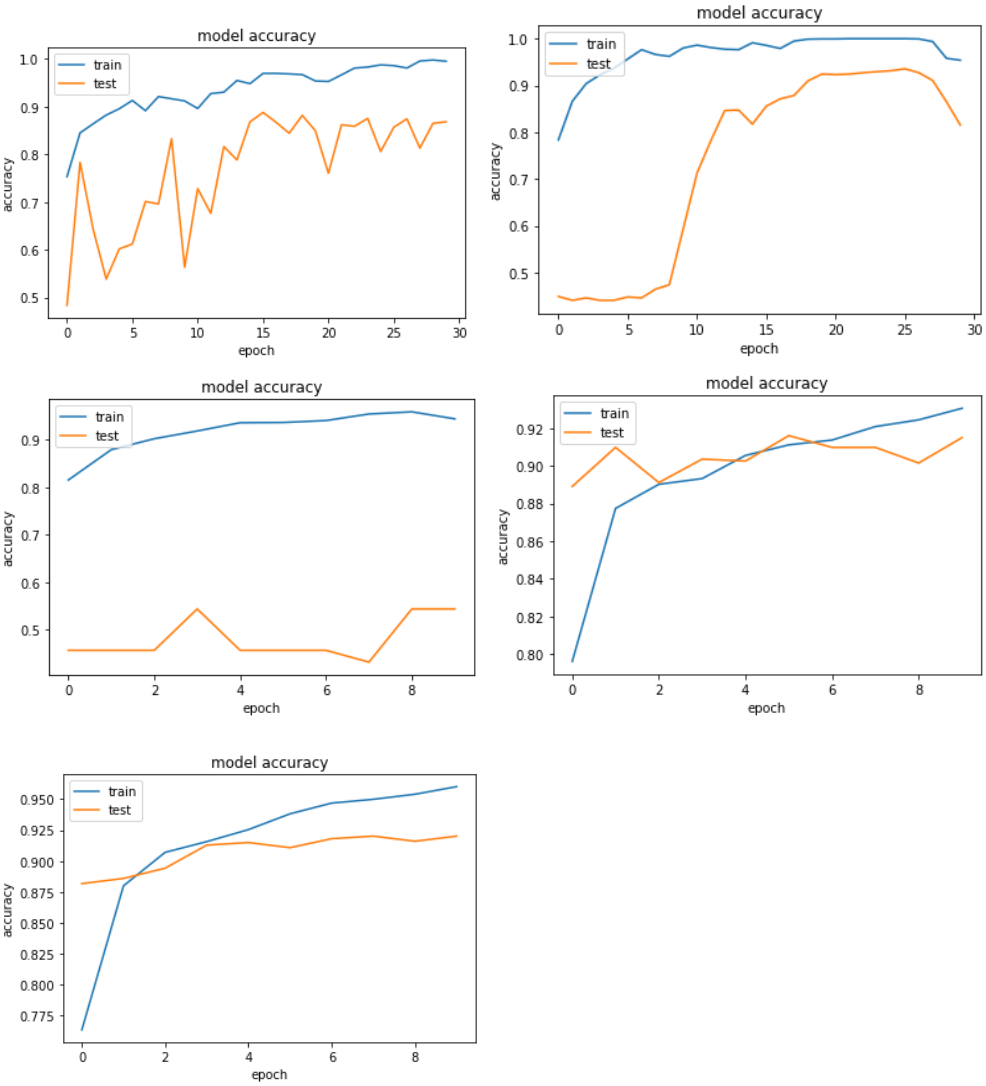
Training score : 1.0000

Testing score : 0.6687

Ces modèles n'ont pas de meilleurs résultats que celui d'avant mais au moins on est sur qu'il converge. Pour la classification binaire MLP augmente le score mais pour la classification multiple le modèle semble encore être trop compliqué et faire de l'overfitting.

2. CNN

Finalement je pensais avoir faire un travail approfondi avec les features faites artificiellement. J'ai donc décidé de passer directement à des images, resized en 80x80x3. On a donc des données plus complexes et il faut ainsi appliquer des modèles plus complexes. J'ai alors tenté plusieurs modèles et j'en ai gardé 3 finalement. Cependant j'ai quand même tracé la courbe d'accuracy (sur les images initiales, sans data augmentation) pour chacun d'entre eux :



J'ai gardé les modèles 2 4 et 5.

Le modèle 1 était trop simple, le modèle 3 était presque la même architecture que VGG19 mais sans les poids pré-entrainés donc cela ne convergeait pas.

Le modèle 2 est fait à la main, le 4 est fait par du fine tuning avec VGG19 et le 5 avec MobileNet. J'ai choisi ces deux réseaux pré-entrainés car je me suis rendue compte que c'étaient ceux avec le moins de couches et que les modèles trop lourds ne faisaient pas converger et étaient trop long.

Après avoir fait la sélection des ces trois modèles j'ai fait de la data augmentation : j'ai multipliéx2 la taille du training set par transformation géométrique des images initiales : rotation, flip vertical et horizontal.

Voici les résultats :

Binary

	Modèle 2	Modèle 4	Modèle 5
Train	0.895223	0.876026	1.00000
Test	0.778230	0.883360	0.84665

Ici le modèle 2 semble être trop simple, il n'avait pas beaucoup de couche. En revanche le modèle 4 marche très bien ! Le modèle 5 a l'air trop compliqué et cela a créé de l'overfitting.

Multiclass

	Modèle 1	Modèle 4
Train	0.92940	0.83250
Test	0.70592	0.64358

Ici on voit que le modèle 2 qui est plus simple fonctionne mieux. Finalement on peut dire que la classification multiple demande des modèles pas trop complexes.

V- Autres méthodes

1. PCA

J'ai tenté de faire une réduction de features sur les images sous forme de vecteur pour ensuite appliquer des algorithmes simples comme SVM. Cependant même sur le training set j'avais un mauvais score (0.68) donc je n'ai pas poursuivi

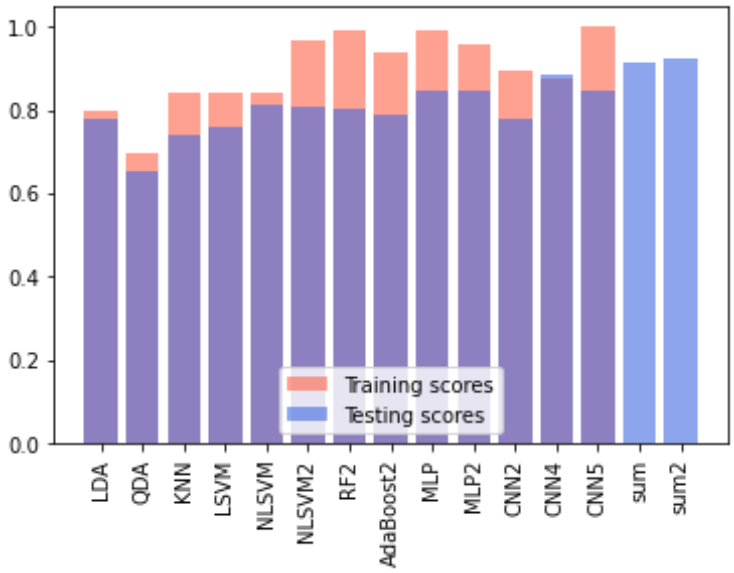
2. Somme des prédictions

Puisque la méthode d'ensemble qui fonctionnait le mieux était le bagging, j'ai tenté de faire une chose similaire, c'est-a-dire de sommer les meilleures prédictions et de faire un vote.

Binary : pour 5 prédictions j'ai eu **0.91221**, pour 10 prédictions j'ai eu **0.92350** qui sont les meilleurs scores !

Multiclass : pour 5 prédiction : **0.67876**, cela n'a pas amélioré

VI- Conclusion



Binary

Multiclass

Finalement pour la classification binaire, le mieux est la méthode de rassemblement des prédictions, cela a réduit la variance et a nettement amélioré les résultats. Sinon le fine tuning avec VGG19 semble le meilleur.

Pour la multiclass classification, cela semble mieux de prendre des modèles simples comme LDA ou un CNN avec peu de couche. D'ailleurs les résultats pour ces deux méthodes sont similaire donc autant garder la simplicité avec LDA.

Si j'avais eu plus de temps, j'aurais peut-être appliquer du bagging avec comme base SVM pour la binary classification et faire un modèle MLP pour les images direct, cela aurait été un modèle plus simple de deep learning pour la multiclass classification peut-être.