Case Study Deep Learning

ATTALI, Hugo
hugo.attali@univ-lyon2.fr

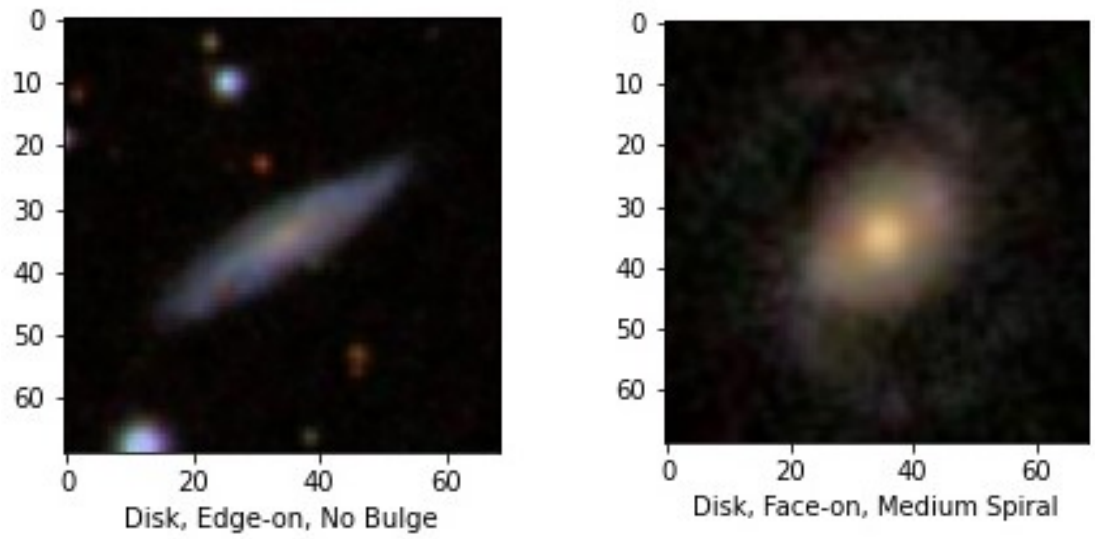
DJAALEB, Tom
tom.djaaleb@univ-lyon2.fr

13 novembre 2022

1 Choix du jeu de données

1.1 Nous avons choisi d'étudier un problème de reconnaissance de forme des galaxies. Le jeu de données que nous avons choisi est Galaxy 10 DECals Dataset proposé par la librairie `astroNN`. Celui-ci est composé de 21785 images de résolution 69×69 représentant des galaxies de divers formes. La variable cible est la forme de la galaxie et elle peut prendre une modalité parmi dix.

1.2 On remarque que la répartition des classes est très déséquilibrée. Pour faciliter notre problème, nous sélectionnons deux classes ayant une différence visuelle notable et ayant à peu près le même nombre d'observations. Les deux modalités choisies sont illustrées par la Figure 1. Notre sous ensemble de données contient 1495 images dont 906 sont des spirales (modalité 1).



(a) (b)

FIGURE 1 – Les deux formes de galaxies choisies pour notre étude.

1.3 Nous découpons nos données en un ensemble d'entraînement (1046 images) et un ensemble de test (449 images). Nous effectuons ce découpage de manière stratifié pour que les deux ensembles est les mêmes distributions selon les modalités.

2 Développement d'une architecture de classification

2.1 Pour la création de notre premier modèle, nous reprenons l'architecture présentée dans le TP sur la reconnaissance de fruits et regardons l'impact de différents hyperparamètres, en l'occurence, le

nombre de couches CNN, nombre de couches Dense, le taux de *dropout*, la valeur du *learning rate* de l'algorithme d'optimisation, la taille du *batch* et le nombre d'époques. Les paramètres de bases de notre modèle sont décrits dans la Table 1.

CNN	Dense	Dropout	Learning Rate	Batch	Epochs
4	2	0.2	0.01	256	20

TABLE 1 – Paramètres par défaut de notre architecture.

Pour cette partie et les suivantes, notre métrique sera l'*accuracy* sur l'ensemble de données test, mais nous verrons plus tard d'autres critère pour évaluer notre modèle. Pour chaque hyperparamètre, nous prenons le modèle par défaut et faisons uniquement varié l'hyperparamètre testé. Voici les résultats que nous obtenons pour chaque hyperparamètre :

CNN	Accuracy	Dense	Accuracy	Dropout	Accuracy
1	0.62	2	0.96	0	**0.97**
2	0.60	4	0.92	0.2	**0.97**
3	0.77	6	0.97	0.5	0.81
4	**0.96**	8	**0.98**		

TABLE 2 – Influence des hyperparamètres sur l'*accuracy*.

Learning Rate	Accuracy	Batch	Accuracy	Epochs	Accuracy
0.01	**0.97**	64	**0.98**	1	0.60
0.1	**0.97**	128	0.97	20	0.95
1	0.39	256	0.96	100	**0.98**
		512	0.61	200	**0.98**

TABLE 3 – Influence des hyperparamètres sur l'*accuracy*.

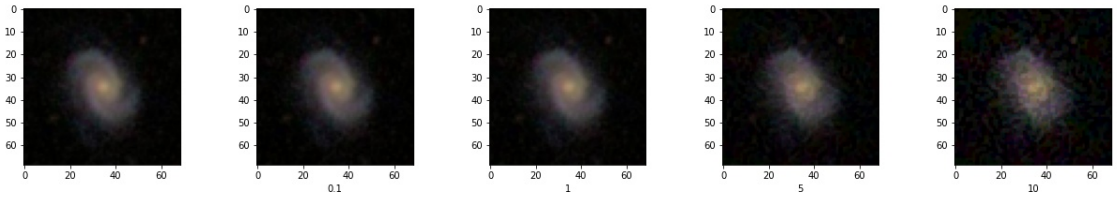
Nous remarquons que le paramètre le plus important est le nombre de couches CNN, en effet, avec trop peu de filtre, il est difficile d'obtenir des informations propres à une classe. Certains paramètres ne semble pas très influent, excepté lorsque ceux-ci prennent des valeurs extrêmes, comme le *learning rate* lorsqu'il dépasse 0.1 ou bien que la taille du *batch* est trop grande. Un détail important est que le nombre de couches denses à la sortie du réseau CNN n'est pas très important car les neurones CNN sont très efficaces pour extraire l'information. A noter que notre modèle est très sensible à l'initialisation ainsi qu'aux données d'entraînements.

2.2 Pour cette deuxième partie, nous décidons de tester deux modèles bien connus, VGG16 et ResNet50. Dans les deux cas nous gardons uniquement la partie convolutionnelle des modèles et ajoutons à la suite un neurone dense à 512 sorties avec activation *relu* puis un neurone dense à 2 sorties avec une activation *sigmoid* qui renverra l'*output* du modèle. Dans les deux cas nous obtenons de très bon résultats sur les données tests, 0.97 d'*accuracy* pour ResNet50 et 0.99 pour VGG16. Pour les deux modèles nous n'avons pas fait d'autres tests.

2.4 Dans cette partie, notre modèle sera le VGG16 adapté à notre problème. Nous reprenons le dataset Galaxy10 Decals et extrayons cette fois-ci les classes 3 et 9 (images proches de celles utilisées dans les parties précédentes). Le modèle obtient une *accuracy* de 0.82 sur ces données, ce qui n'est pas une performance exceptionnelle mais qui est tout de même correcte pour un modèle n'ayant pas eu d'images de ces classes en entraînement. Nous n'avons pas testé notre modèle sur d'autres jeux de données.

2.5 Après avoir testé la performance de notre modèle sur plusieurs jeux de données et avec différents paramètres, nous testons la robustesse de notre modèle. L'*Adversarial Machine Learning* à d'ailleurs

fait l'objet d'un stage cet été pour Tom au Laboratoire ERIC. L'attaque que nous avons décidé d'appliquer sur notre modèle est la *Fast Gradient Sign Method*, développé par Goodfellow et al. et publiée en 2015. Cette méthode consiste à ajouter une petite perturbation aux images en fonction du signe du gradient de la fonction de perte. La librairie `cleverhans` propose une fonction permettant de mettre en place la méthode FGSM. La Figure 2 illustre la modification des images selon le paramètre ϵ de la méthode FGSM et la Table 4 montre l'impact qu'ont les attaques sur les performances du modèle.



(a) Originale (b) $\epsilon = 0.1$ (c) $\epsilon = 1$ (d) $\epsilon = 5$ (e) $\epsilon = 10$

FIGURE 2 – Illustration de la perturbation par la méthode FGSM.

Epsilon	Accuracy
0	0.99
0.1	0.92
1	0.81
5	0.52
10	0.57

TABLE 4 – Performance du modèle en fonction de la "force" de l'attaque.