



# RAPPORT IA

## Introduction

Dans le cadre de notre projet «modèles IA et apprentissage», nous avons décidé d'opter pour un sujet portant sur l'apprentissage. Nous allons dans un premier temps, expliquer la problématique étudiée. Ensuite, nous discuterons des méthodes utilisées pour la résolution du problème et nous comparerons leurs résultats. Enfin, nous ferons une liste des compétences acquises durant le projet et nous concluerons.

Le projet est réalisé en Python sur Kaggle. Le notebook avec le code est également fourni en plus de ce rapport.

## Sommaire

<b>Introduction</b>	<b>1</b>
<b>Problématique</b>	<b>2</b>
Dataset pour la classification de terrains	3
La récolte d'image sur internet	3
Prendre en photo les images soi-même	5
La génération procédurale d'images	5
Une dernière méthode	6
Dataset pour la classification de paysages	6
<b>Les modèles de ML étudiées</b>	<b>7</b>
Réseau de neurones convolutif	8
SVM (Support Vector Machine)	8
KNN (k-Nearest Neighbors)	9
Random Forest Classifier	9
<b>Comparaison des résultats</b>	<b>10</b>
Test plus approfondi du réseau de neurones	11
Avec la limitation de 200 images et 15 itérations	11
Sans limitation d'images et 30 itérations	11
<b>Concepts acquis</b>	<b>12</b>
<b>Conclusion</b>	<b>12</b>
<b>Références</b>	<b>12</b>

## Problématique

L'utilisation des robots terrestres est déjà présente dans divers secteurs et va probablement se démocratiser davantage dans le futur. Concernant les robots terrestres autonomes, ou semi-autonomes, on se heurte à divers problèmes tel que le pathfinding par exemple. La problématique que nous avons décidée d'étudier ici est la reconnaissance de terrain, plus précisément la reconnaissance de terrain à l'aide d'images. Ce projet est inspiré de celui de deux étudiants de l'Université de Stanford, Hojung Choi et Rachel Thomasson, qui ont quant à eux étudié ce même problème mais avec un robot équipé de capteurs capacitifs tactiles sur les pattes [1].

Un robot terrestre avec des roues par exemple, ne peut pas adopter la même allure sur tous les types de terrains. Il est nécessaire d'adopter l'allure du robot en fonction de la friction (le frottement) avec le sol. Le robot devrait aller à une vitesse basse sur du sable pour ne pas faire du sur-place, et à une vitesse élevée sur du goudron par exemple. Notre but ici est de comparer plusieurs méthodes d'apprentissage pour classifier ces terrains.



*Un robot terrestre*

Malheureusement, nous n'avons pas trouvé de dataset gratuit comportant des images de terrains pour étudier notre problème. Nous avons donc décidé d'étudier un problème similaire: la classification des paysages. Nous considérons qu'il est plus difficile pour une machine de classifier des paysages plutôt que des terrains, nous estimons donc que le projet peut être qualifié de réussi si nous obtenons des résultats satisfaisants pour ce problème. Nous avons cependant recherché quelques méthodes pour obtenir un dataset satisfaisant avec des images de terrains, nous listons ces méthodes plus bas.

## Dataset pour la classification de terrains

Comme précisé plus haut, nous avons étudié les moyens d'obtenir un dataset satisfaisant avec des images de terrains. Nous n'avons pas pu utiliser ces méthodes par manque de temps. Nous proposons donc 3 méthodes:

- La récolte d'image sur internet
- Prendre en photo les images soi-même
- La génération procédurale d'images.

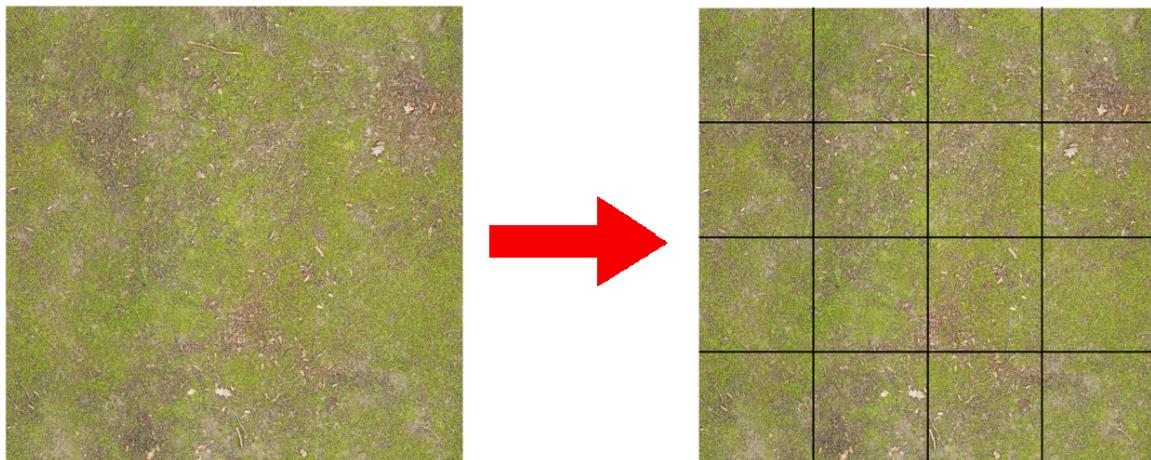
### La récolte d'image sur internet

Il est possible de trouver des images de terrains directement sur internet, sur Google Image par exemple, mais nous trouvons que les sites <https://ambientcg.com/> et <https://www.sharetextures.com/> correspondent particulièrement bien à notre besoin. Ces sites proposent des images de texture sous licence du domaine public. Les images sont catégorisées par texture, comprenant plusieurs textures de terrains. Ces images sont de différentes qualités, leurs résolutions allant jusqu'à 16K.



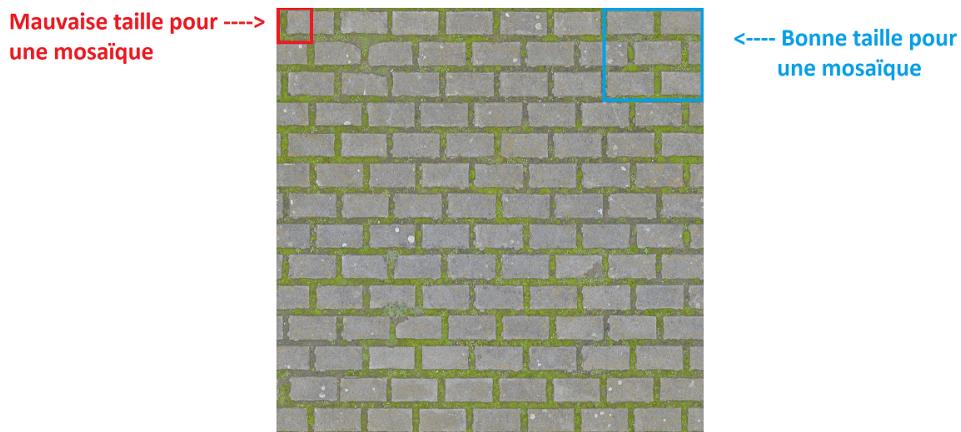
*Exemple d'image de terrain que nous pouvons trouver sur ces sites*

Le problème de cette méthode est que le nombre d'images de terrains sur ces sites est limité, ce qui peut poser problème pour l'apprentissage de nos modèles. Une solution possible à ce problème est de diviser ces images en plusieurs sous-images, pour obtenir une mosaïque d'images. On peut largement augmenter nos jeux de données de manière considérable à l'aide de cette méthode. Une image en résolution 8K (8192x8192 pixels), peut par exemple générer 16 images avec 512x512 pixels.



*Image séparée en une mosaïque de 16 images*

Cette méthode fonctionne bien avec des images de terrains comme le sable par exemple mais peut poser problème lorsqu'on essaye de diviser des images de pavages. Il faut faire attention à bien choisir la taille des images engendrées, car il y a un risque de perdre le pattern visible sur les pavages. Comme les images sont correctement catégorisées sur ces sites, nous n'aurons aucune difficulté à labelliser les images avec cette méthode.



*Illustration des bonnes et mauvaises tailles pour une mosaïque*

### Prendre en photo les images soi-même

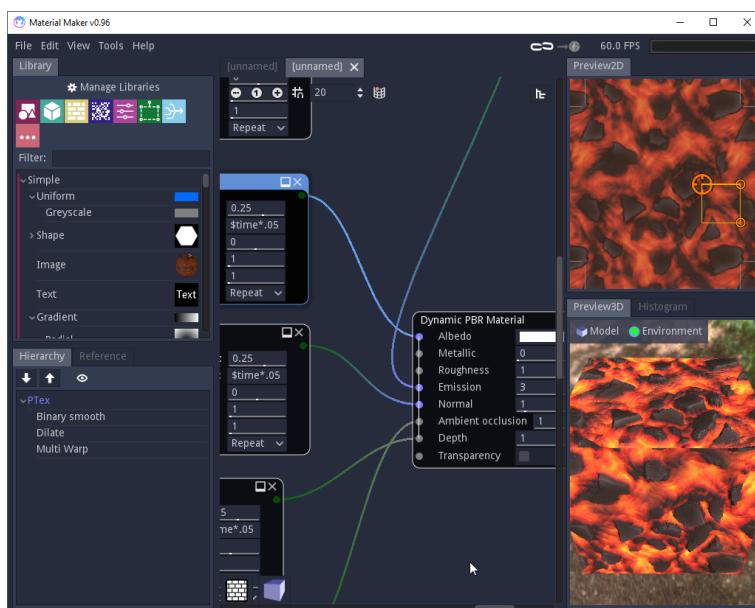
Une méthode qui nous permet d'obtenir un très grand nombre d'images adaptées à notre problème est de prendre en photo soi-même les images à l'aide d'un appareil photo, ou bien de prendre une vidéo et utiliser les images de la vidéo frame par frame. On peut bien entendu multiplier les images en appliquant la méthode des mosaïques décrites plus tôt dans la méthode précédente.

La contrainte de cette méthode est l'effort physique engendré et la limitation de terrains possibles dans la région. Nous devons par ailleurs labelliser les images nous même, ce qui demanderait énormément de temps.

## La génération procédurale d'images

La troisième méthode que nous proposons est d'utiliser des générateurs procéduraux d'images comme le logiciel Material Maker. L'avantage de cette méthode est d'avoir des jeux de données avec autant d'images que souhaitées, et correctement labellisés qui plus est.

L'inconvénient est bien évidemment le fait que ces images ne soient pas des images réelles. Les performances de nos modèles pourraient être réduites lorsque nous les utiliserons avec de vraies images.



Screen du logiciel Material Maker

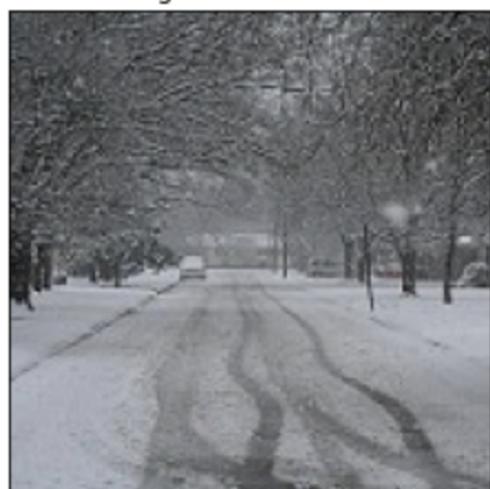
## Une dernière méthode

Une dernière méthode que nous proposons est de fusionner les 3 méthodes précédentes pour avoir un jeu de données varié. Nous pouvons également utiliser les images des deux premières méthodes pour générer procéduralement de nouvelles images.

## Dataset pour la classification de paysages

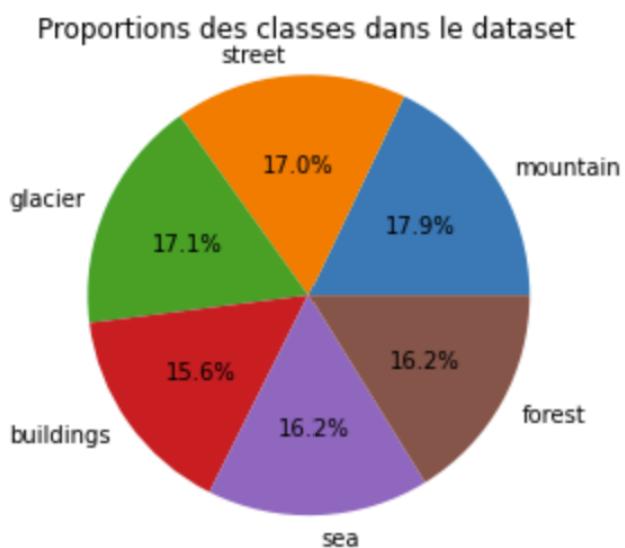
Nous utiliserons le dataset **Intel Image Classification** fourni sur Kaggle. Ce dataset contient ~25 000 images de taille 150x150 labellisées sous 6 catégories de paysages: bâtiments, forêt, glacier, montagne, mer, et rue.

Image #4304 : street



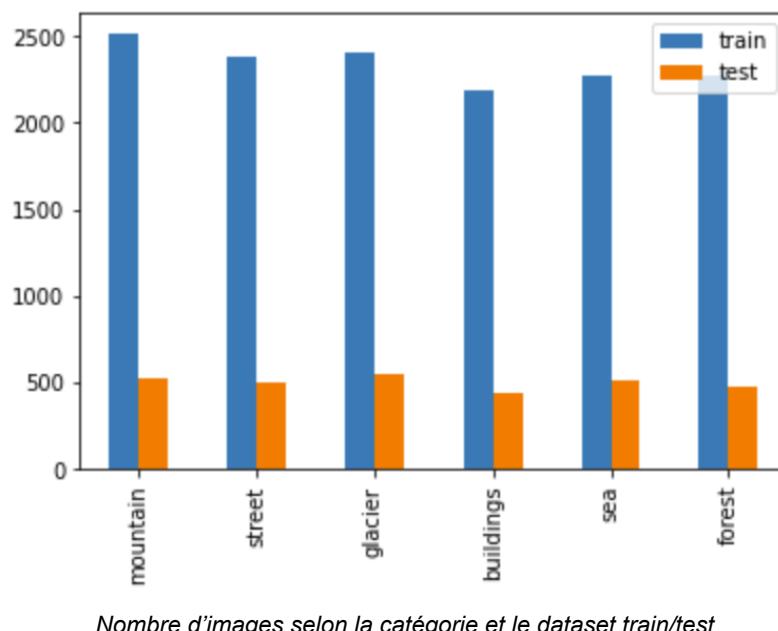
*Exemple d'image*

Le nombre d'images pour chaque catégorie est plus ou moins la même.



*Répartition des images selon la classe*

La séparation des images de test et d'apprentissage est déjà faite.



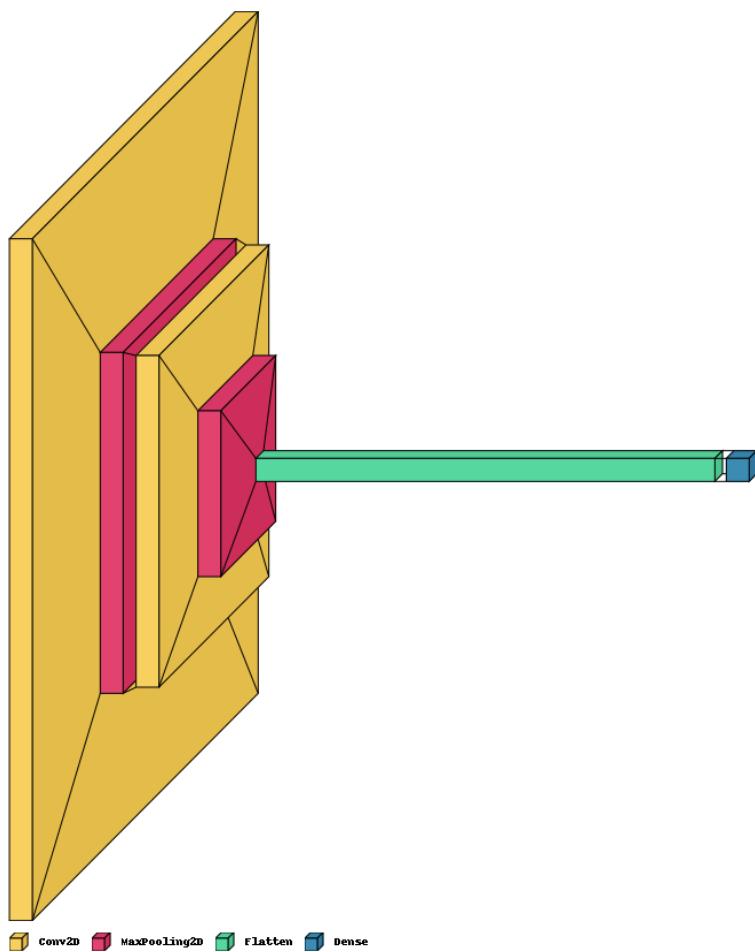
## Les modèles de ML étudiés

Nous avons décidé de comparer 4 modèles différents pour résoudre le problème de classification d'images: un réseau de neurones, SVM, KNN, et une forêt aléatoire.

### Réseau de neurones convolutif

Il existe différents types de réseaux de neurones. Le type de réseau de neurones le plus utilisé dans les problèmes liés aux images est le réseau de neurones convolutif. C'est ce type de réseau que nous avons implémenté et testé. Il existe différentes librairies disponibles en Python pour implémenter des réseaux de neurones. Nous avons décidé d'utiliser Keras, car c'est une librairie de haut niveau, facile à prendre en main, contrairement à d'autres librairies comme PyTorch qui demandent plus d'efforts et de temps que nous n'avons pas.

Le modèle implanté est composé de deux couches convolutives, dont nous réduisons les tailles par deux, une couche de réduction de la dimension qui est réduite à 1, et une dernière couche de 6 neurones qui donne la probabilité d'appartenir à une classe.



Visualisation du réseau implémenté

## SVM (Support Vector Machine)

Le modèle SVM est un modèle très utilisé dans la classification. Le principe du modèle est simple: il essaie de tracer une ligne entre les classes, de manière à laisser le plus d'espace entre cette ligne et les classes.

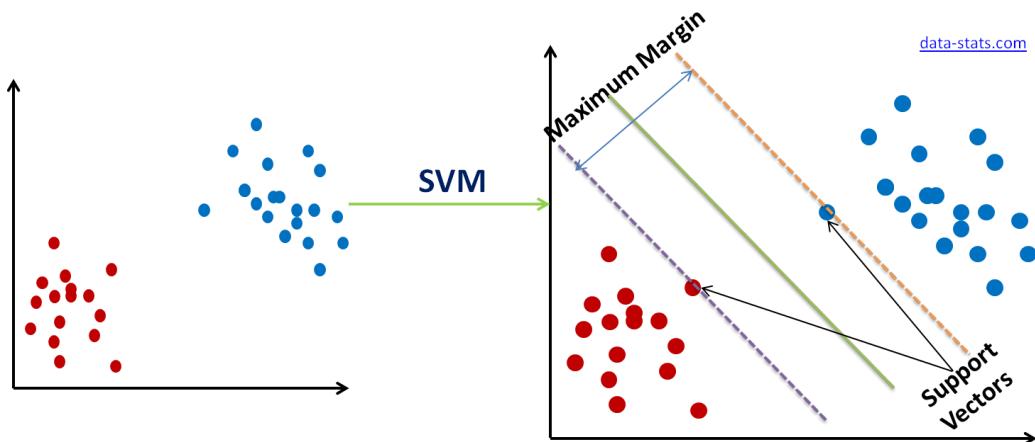
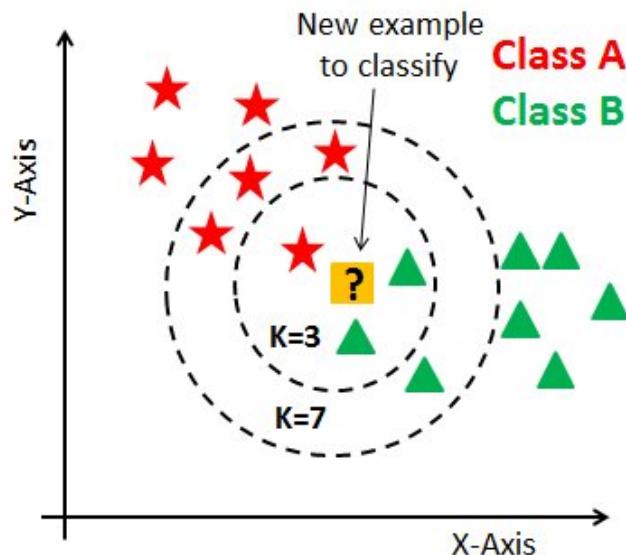


Illustration du principe du modèle SVM

## KNN (k-Nearest Neighbors)

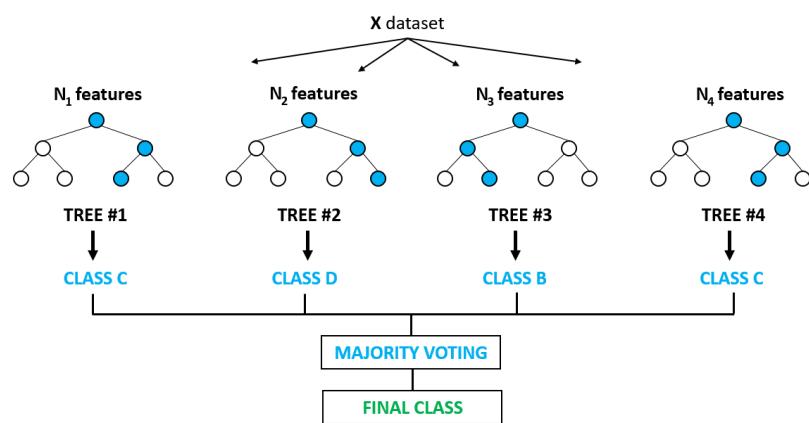
Le modèle KNN a pour principe de rechercher les k-voisins déjà labellisés les plus proches d'un élément pour le classifier. Nous avons utilisé  $k = 13$ .



*Illustration du principe du modèle KNN*

## Random Forest Classifier

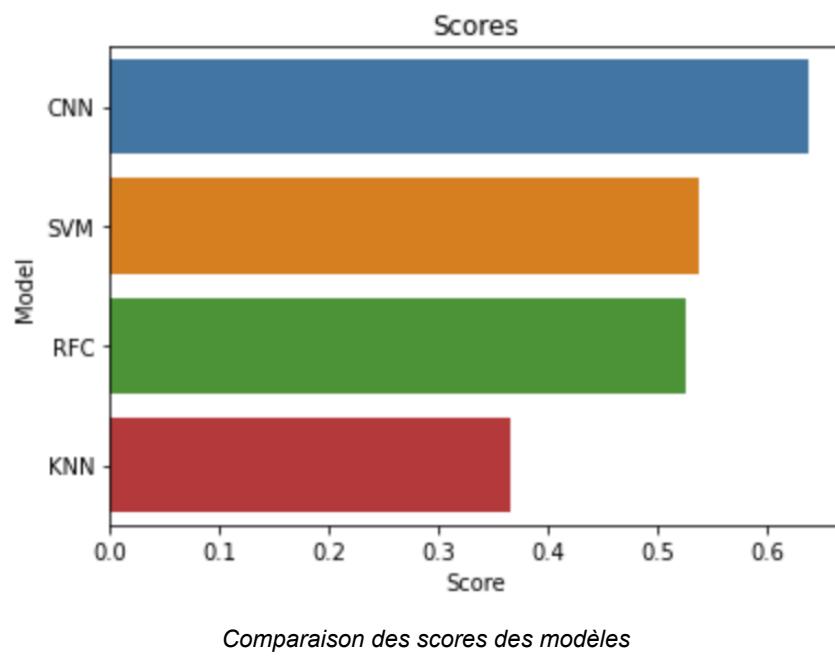
La forêt aléatoire, le dernier modèle que nous avons implémenté, a pour principe de générer plusieurs arbres de décisions, et de garder la décision majoritaire.



*Illustration du principe du Random Forest Classifier*

## Comparaison des résultats

Nos machines sont limitées en puissance, l'exécution des modèles pose donc problème à partir d'un certain nombre d'images. Le réseau de neurones ne pose pas de problèmes, mais les 3 autres modèles prennent trop de temps à être exécutées (pas dans un temps raisonnable) à partir d'un certain nombre. C'est pourquoi nous avons limité le nombre d'images par classe à 200 (donc  $200 \times 6 = 1200$  images) pour l'apprentissage.



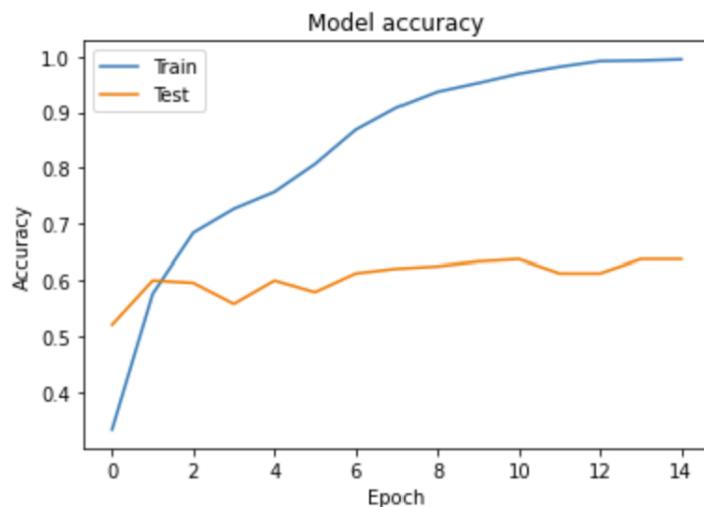
Comme il y a 6 classes possibles, un modèle aléatoire donnerait un score de  $\frac{1}{6} \approx 0,167$ , nos modèles ont donc l'air d'avoir de bonnes performances.

On remarque que le réseau convolutif a de meilleurs résultats que les autres modèles. Le réseau convolutif offre également ces meilleurs résultats avec un temps d'apprentissage beaucoup moins élevé que les autres. Les modèles SVM et RFC offrent de bonnes performances, KNN quant à lui offre les pires performances.

## Test plus approfondi du réseau de neurones

Nous avions limité le nombre d'itérations pour l'apprentissage à 15, nous allons cette fois-ci l'augmenter à 30. Nous allons également tester l'apprentissage avec le jeu de données complet.

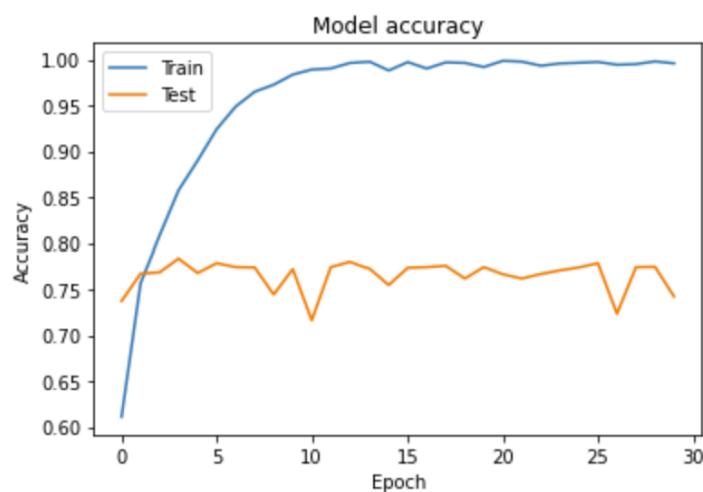
### Avec la limitation de 200 images et 15 itérations



*Illustration des scores sur l'ensemble d'apprentissage et de test*

Les résultats stagnent autour de ~0,6 à partir de 8 itérations.

### Sans limitation d'images et 30 itérations



*Illustration des scores sur l'ensemble d'apprentissage et de test*



Les résultats stagnent très vite autour de ~0,75. On peut donc en conclure que la taille du dataset a une grande importance pour ce problème. Le nombre d'itérations pendant l'apprentissage quant à lui influe peu sur les résultats.

## Concepts acquis

Nous avons étudié et acquis plusieurs concepts durant ce projet tels que:

- la classification d'images
- la constitution de dataset
- les librairies de machine learning en Python tels que Keras et sci-kit learn
- les librairies de visualisation graphique en Python tels que Seaborn, Matplotlib, tqdm

## Conclusion

Le réseau de neurones semble être le modèle le plus adapté pour traiter la problématique de classification d'images de paysages, et donc probablement pour la classification d'images de terrains. Il est possible d'obtenir de meilleures performances/résultats en traitant au préalable les images de différentes manières (en transformant leurs couleurs en nuances de gris par exemple, pour baisser le nombre d'entrées) ou en testant plusieurs paramètres pour les modèles.

## Références

- [1] Terrain Classification for Small-Legged Robots using Deep Learning on Tactile Data,  
Hojung Choi and Rachel Thomasson {hjchoi92, rthom}@Stanford.edu  
Department of Mechanical Engineering, Stanford University  
[Choi\\_Thomasson.pdf \(stanford.edu\)](http://www.stanford.edu/~rthom/TerrainClassification.pdf)