GARNERO Camille

MESSAGER Hugo

GOURET Samuel

Github : https://github.com/CamilleGAR/Trackmania-Autoguided

Vidéo de presentation : https://www.youtube.com/watch?v=ehIYeGms2Mk

Rapport PROJ831

Présentation du sujet :

Dans ce projet, on va créer une intelligence artificielle qui joue à Trackmania, en respectant certaines conditions :

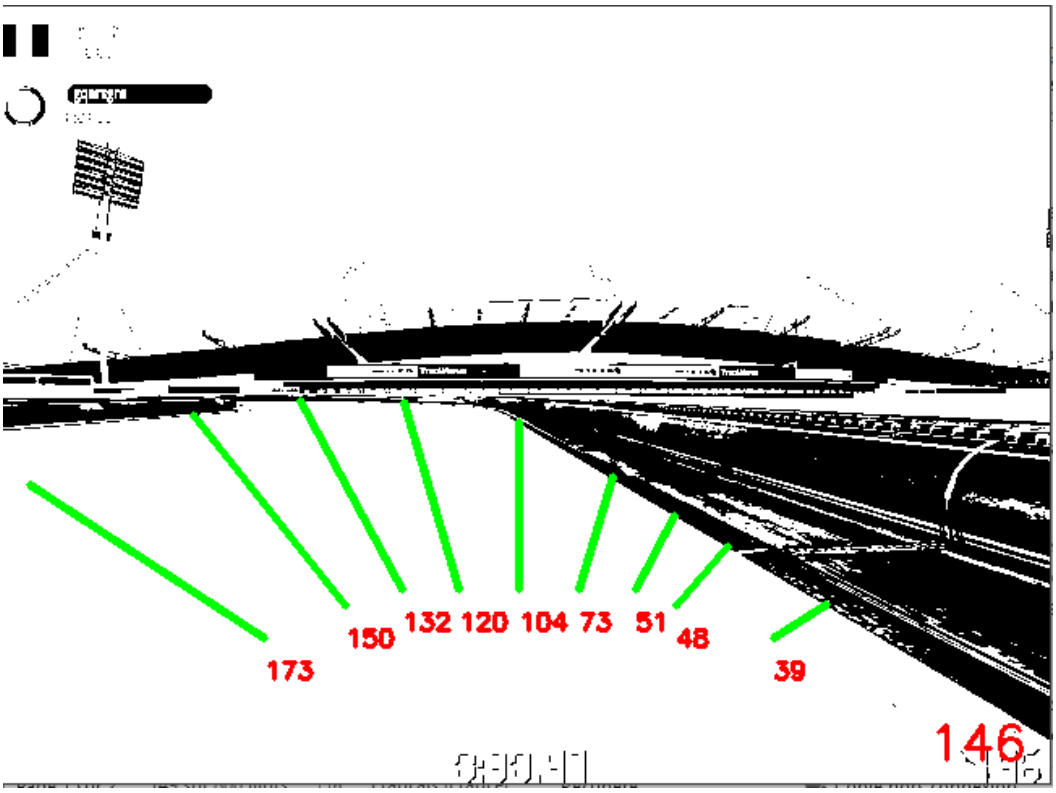
•Notre IA n'est pas destinée à finir un circuit le plus vite possible. On veut à tout prix éviter qu'elle apprenne par cœur un circuit et qu'elle anticipe toutes ses trajectoires. En fait, elle doit être capable de conduire de manière autonome sur un circuit qu'elle n'a jamais vue.

•L'IA fonctionnera avec un réseau de neurones, en Machine Learning.

•On utilisera un apprentissage supervisé. On se filmera en train de jouer, en enregistrant à la fois notre écran et nos entrées clavier. A la fin, notre IA analysera tous nos enregistrements. A partir de là, juste en nous ayant vu jouer, elle devra savoir conduire sur n'importe quel circuit. Elle n'apprendra pas de ses erreurs, elle apprendra à conduire uniquement à partir de notre dataset.

Présentation du réseau de neurones :

On met en place un système de capteurs qui calcule la distance entre la voiture et les murs. On utilise aussi un système de reconnaissance d'image pour lire la vitesse sur le compteur. Les capteurs et la vitesse seront utilisées comme inputs dans notre réseau de neurones.



Les entrées clavier associées à chaque image seront utilisées comme outputs dans notre réseau de neurones.

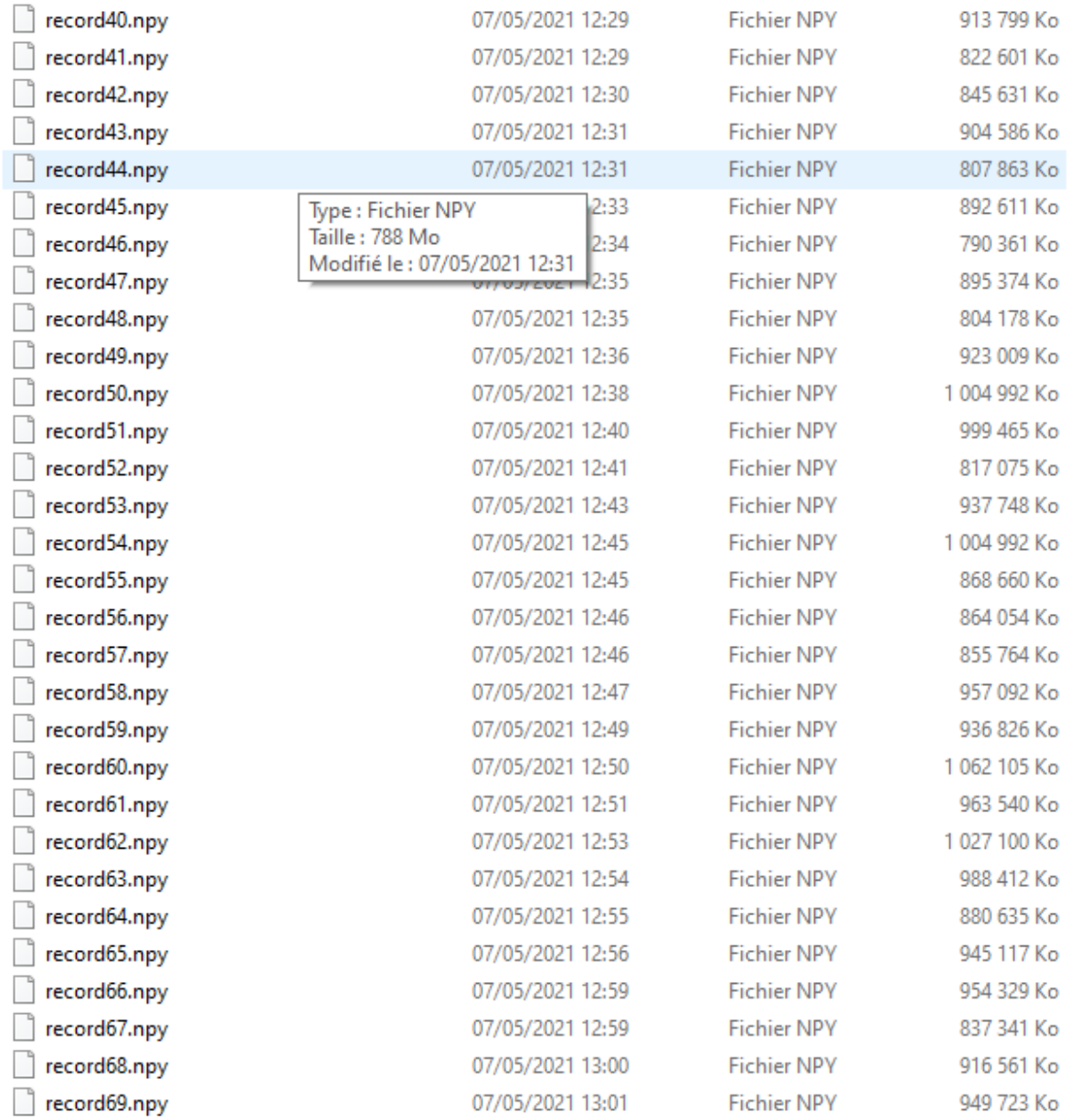
```
['haut', 'droite']
['haut', 'droite']
['haut', 'droite']
['haut', 'droite']
['haut', 'droite']
['haut', 'droite']
['haut', 'droite']
['haut']
['haut']
['haut', 'droite']
['haut', 'droite']
['haut', 'droite']
['haut', 'droite']
['haut', 'droite']
['haut', 'droite']
['haut', 'droite']
['haut', 'droite']
['haut', 'droite']
['haut', 'droite']
['haut', 'droite']
['haut', 'droite']
['haut', 'droite']
['haut', 'droite']
['haut']
['haut']
['haut']
['haut', 'droite']
['haut', 'droite']
['haut', 'droite']
['haut', 'droite']
['haut', 'droite']
['haut']
['haut']
['haut']
['haut']
['haut']
```

Acquisition de la data :

On va enregistrer 70 vidéos sur 7 circuits différents. On créer aussi un 8ème circuit qui ne sera utilisé que pour vérifier le bon fonctionnement de notre IA. On ne l'utilisera pas pour l'entrainer !

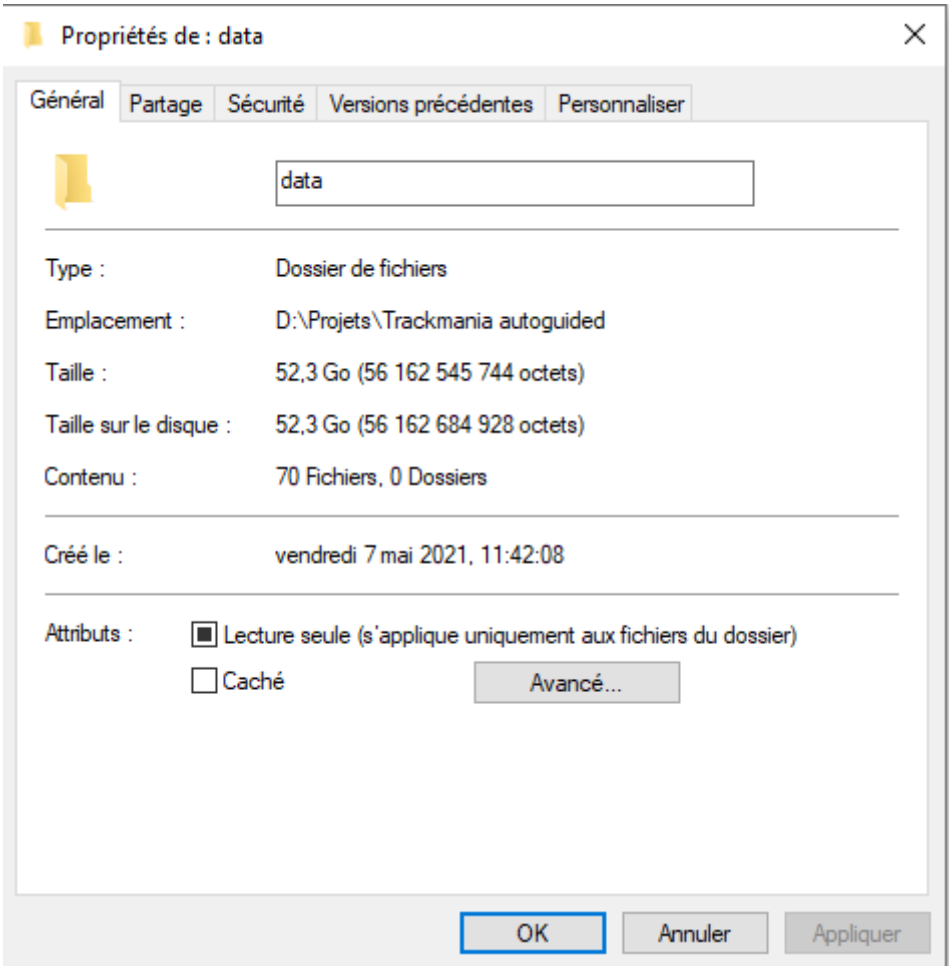
Il est important de noter que lors de nos enregistrements, on enregistre TOUTE l'image, et non pas seulement la valeur des capteurs. Ça peut sembler contreproductif, mais si on décide de changer notre traitement d'image, on n'est pas obligé refaire TOUS les enregistrements. Et puis, niveau temps de calcul ça ne change rien, puisque qu'il faudra faire le traitement d'image, tôt ou tard.

Les enregistrements sont assez simples à faire. A chaque fois qu'on finit une course, elle s'enregistre automatiquement. Si on fait une erreur et qu'on décide de recommencer notre course, l'enregistrement s'annule, et un nouveau commence.



En environ une heure, on arrive à enregistrer près de 70 vidéos (sous format numpy, qui contient à la fois l'image et les entrées clavier).

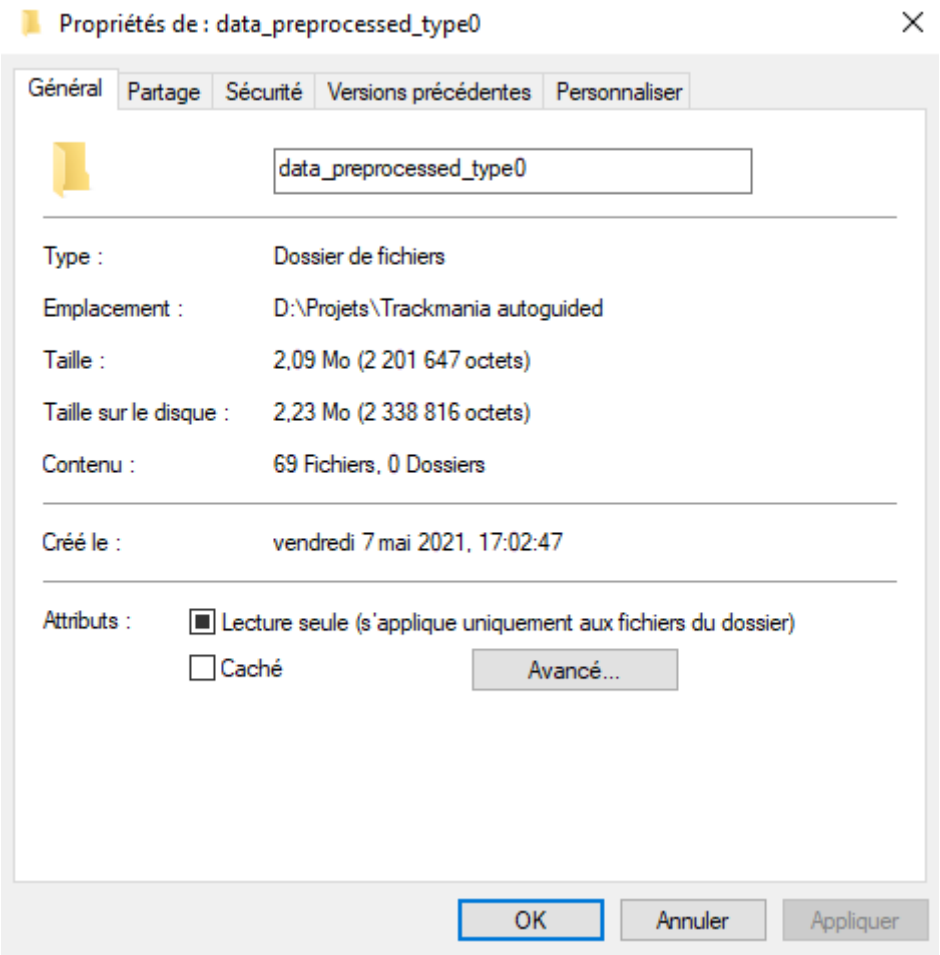
Notre data forme à présent un fichier de plus de 50 Go.



Traitement de la data :

Maintenant qu'on a fait notre acquisition de données, on doit la traiter. Pour chaque image, on ne récupère que la valeur de nos capteurs, la vitesse, et les sorties clavier.

On enregistre le tout, on obtient maintenant un fichier de 2 Mo.



C'est ce fichier data qui va être utilisé pour entrainer notre modèle.

Préprocessing :

On a maintenant une data qui semble exploitable. Cependant, même si son format est bon, il est déconseillé de l'utiliser tel quel.

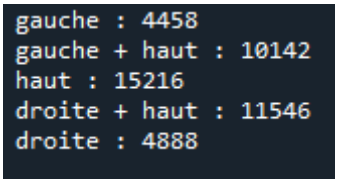
On va lister ci-dessous quelques-uns des points importants sur lesquels il faut travailler :

•Augmentation de la quantité de data :

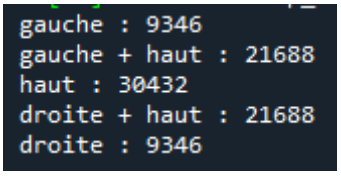
On peut aisément doubler notre quantité de données en « inversant » les images. Il suffit de d'inverser la liste des capteurs, ainsi que la liste des entrées clavier. ([1,0,0] -> [0,0,1])

Une image qui montre un virage a droite va désormais montrer un virage à gauche.

On peut ainsi passer de cette répartition entrées clavier :



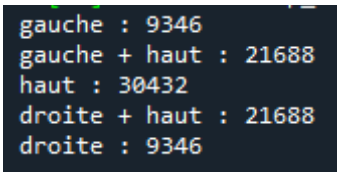
A celle-ci :



• Rééquilibrage de la quantité de données :

Quand on s'enregistre, les données ne sont pas équilibrées.

Comme vous pouvez le constater juste au dessus, on a beaucoup plus d'échantillons d'images qui corresponde à un input « aller tout droit » que d'échantillons « tourner à droite ».



Cela peut altérer notre modèle et il peut vite apprendre à toujours aller tout droit pour augmenter son accuracy.

On va donc réduire la quantité de données de chaque entrée clavier pour qu'elle corresponde à la quantité la plus faible (ici 9346 pour les directions).

```
gauche : 9346
gauche + haut : 9346
haut : 9346
droite + haut : 9346
droite : 9346
```

Attention cependant à ne pas oublier de shuffle notre dataset avant de l'équilibrer ! Ca évite que les même jeux de données soient récupérés à chaque fois.

On shuffle une deuxième fois après l'équilibrage, car lors de l'équilibrage, nos données sont triées.

- **Normalisation de données :**

Nos données ont des valeurs comprises entre 0 et presque 200. On veut utiliser des données globalement entre 0 et 1 pour notre réseau de neurones. Pour chaque entrée du réseau, on détermine le min et le max de notre dataset de training.

On utilise la formule suivant pour normaliser notre data : (input – min) / (max – min).

Il ne faut pas oublier de conserver ces valeurs min et max. Quand on va travailler sur notre dataset de validation, de test, ou faire des prédictions en direct, on utilisera les valeurs min et max qu'on a déterminé sur le dataset de training.

Training :

Pour entrainer notre modèle, pas grand-chose à dire. Après avoir travaillé sur notre modèle, celui-ci est celui qu'on a décidé de garder :

```
model.add(Dense(100,
                activation='relu',
                kernel_initializer='he_normal',
                input_shape=(nb_features,)))
model.add(Dropout(0.15))
model.add(Dense(100, activation='relu', kernel_initializer='he_normal'))
model.add(Dropout(0.15))
model.add(Dense(100, activation='relu', kernel_initializer='he_normal'))
model.add(Dropout(0.15))
model.add(Dense(100, activation='relu', kernel_initializer='he_normal'))
model.add(Dropout(0.15))
model.add(Dense(nb_outputs, activation='softmax'))
model.compile(optimizer = Adam(learning_rate = 0.0015),
              loss ='categorical_crossentropy',
              metrics = ['accuracy'])
history = model.fit(X_train, y_train, epochs=100, batch_size=128, verbose=1, validation_split=0.3)
```

Résultats :

Comme on peut le voir ci-dessous, durant notre training, on atteint un maximum de 70% d'accuracy.

```
205/205 [==============================] - 1s 3ms/step - loss: 0.7065 - accuracy: 0.6774 - val_loss:
0.6751 - val_accuracy: 0.6978
Epoch 98/100
205/205 [==============================] - 1s 3ms/step - loss: 0.6922 - accuracy: 0.6902 - val_loss:
0.6732 - val_accuracy: 0.6992
Epoch 99/100
205/205 [==============================] - 1s 3ms/step - loss: 0.6956 - accuracy: 0.6797 - val_loss:
0.6781 - val_accuracy: 0.6931
Epoch 100/100
205/205 [==============================] - 1s 3ms/step - loss: 0.6966 - accuracy: 0.6860 - val_loss:
0.6716 - val_accuracy: 0.6982
```

Ça peut paraitre faible, mais en fait, on se pouvait s'attendra à mieux. En effet, on s'est enregistré en train de jouer, pour ensuite demander à notre IA de mimer ce qu'elle a vue. Le problème est que nos propres data peuvent être incohérentes. Quand on s'est enregistré en train de jouer, dans une même situation, on ne faisait pas forcément la même action. Des fois on tourne un peu plus tôt, des fois un peu plus tard. L'important est que les 30% d'accuracy manquants soient dues à des erreurs qui n'impactent pas vraiment la conduite. Par exemple, l'IA ne tourne pas à gauche si elle doit aller à droite. Par contre, elle peut se « tromper », et aller tout droit, pour tourner à droite 10 millisecondes plus tard.

Au final, notre IA marche très bien et arrive à finir un circuit qu'elle n'a jamais vue sans toucher de mur. Elle mettra en moyenne 20% plus de temps que nous à finir un circuit (à condition qu'on conduise de la même manière que dans nos enregistrements, c'est-à-dire de manière très douce, et sans anticiper les trajectoires). Nos résultats sont plutôt satisfaisants.

Problèmes :

Les seuls problèmes que l'on peut rencontrer sont dues aux performances de l'ordinateur. Notre IA prend des Screenshot de l'écran en boucle, et traite l'image comme elle l'a appris. Malheureusement, un ralentissement de notre ordinateur entraine une fréquence d'image plus faible, et il peut arriver que notre IA se rende compte de la présence d'un virage quelques centièmes de secondes trop tard pour pouvoir éviter l'accident. C'est ce qui arrive notamment lorsqu'on enregistre notre écran. Nous avons eu du mal à tourner notre vidéo de présentation, car une fois l'écran enregistré, l'IA devient beaucoup moins performante.

Pour obtenir plus de détails techniques, vous pourrez trouver notre code commenté sur notre Github : https://github.com/CamilleGAR/Trackmania-Autoguided

Vous trouverez également notre vidéo de présentation ici :
https://www.youtube.com/watch?v=ehIYeGms2Mk