Amelioration IA

Loïc Curbière

December 2021

Contents

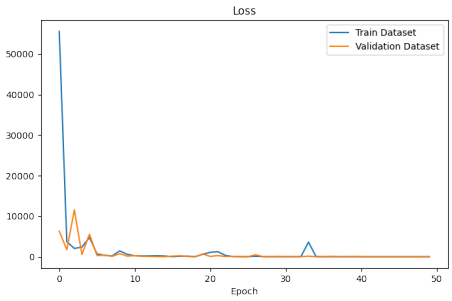
1 Introduction

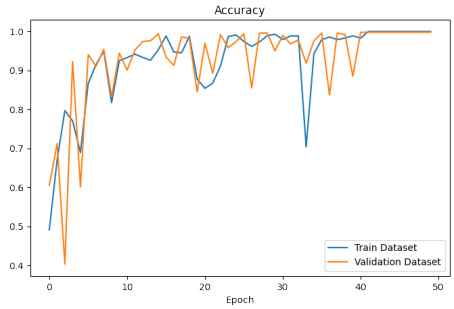
Notre client de shifumi Co possède déjà une application de shifumi basé sur de l'IA mais celle-ci est très loin d'être optimale Dans ces quelques pages, je parlerai tout d'abord de l'évolution de l'IA (grâce à de très légères modifications/améliorations) puis de passage à une application pour rendre encore meilleure l'expérience ludique du joueur.

2 Amélioration de l'IA

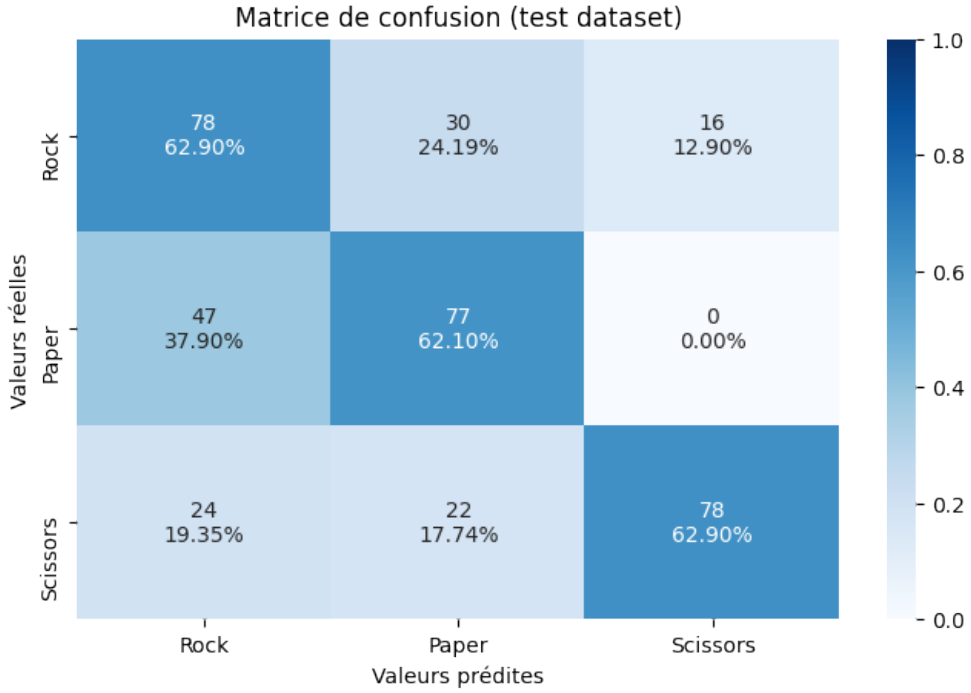
2.1 La version initiale

Dans la version initiale du projet, les performance de l'algorithme était des plus mauvaises Ci dessous les courbes d'apprentissage du modèle





En regardant de plus près la matrice de confusion, on note que la classification n'est pas bonne du tout



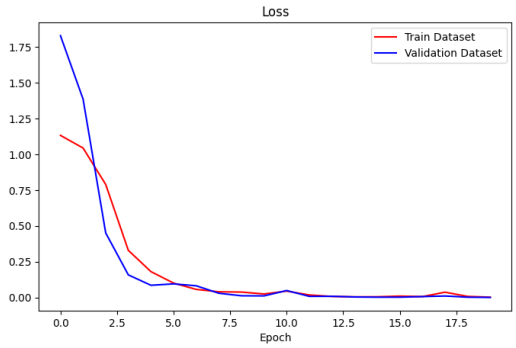
2.2 Une version plus aboutie

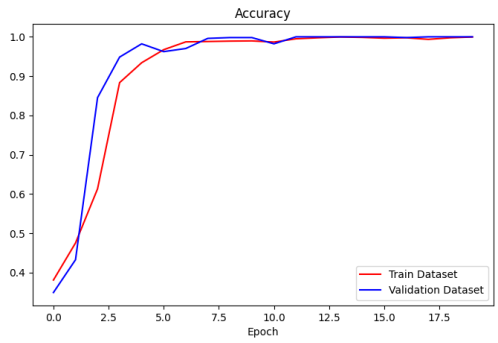
2.2.1 A quoi ressemble dataset

Les images du dataset (qui proviennent du site tensorflow)

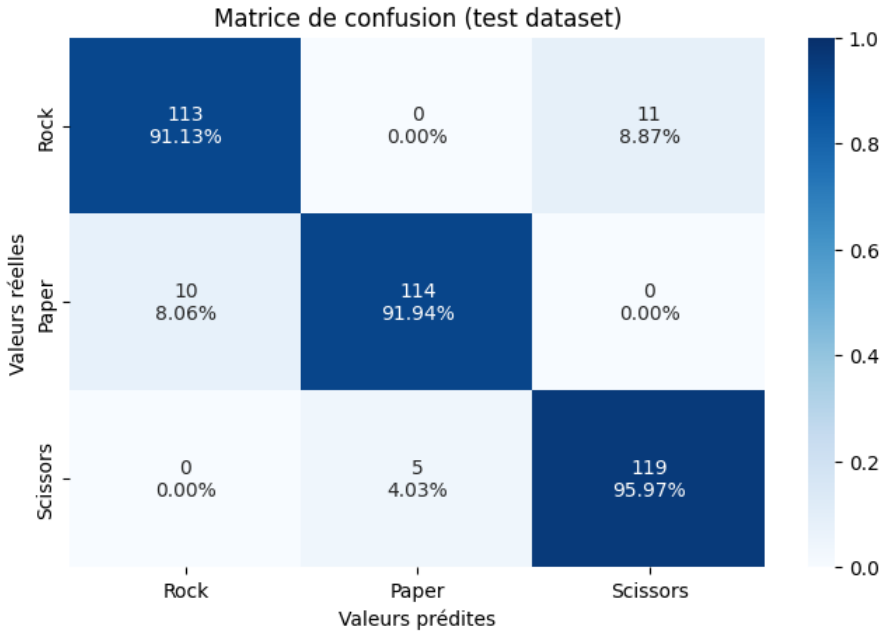
2.2.2 Les résultats tous beaux

Pour obtenir ces résultats j'ai utilisé un CNN et de la data-augmentation
Ci dessous les courbes d'apprentissage de mon modèle





Les résultats sont biens meilleurs (quoi que perfectibles)



2.3 Pour aller plus loin

Même si les résultats sont meilleurs, ils restent perfectibles.

2.3.1 Explicitation de mon modèle

Comme dans la version de base, j'ai utilisé des données fournit par tensorflow

```
X_train, y_train = tfds.as_numpy(tfds.load(
    'rock_paper_scissors',
    split='train',
    batch_size=-1,
    as_supervised=True))
```

J'ai aussi rescalé les images pour que toutes les informations se retrouvent entre 0 et 1

```
# Normalize pixel values to be between 0 and 1
train_ds=train_ds/255.0
test_ds=test_ds/255.0
```

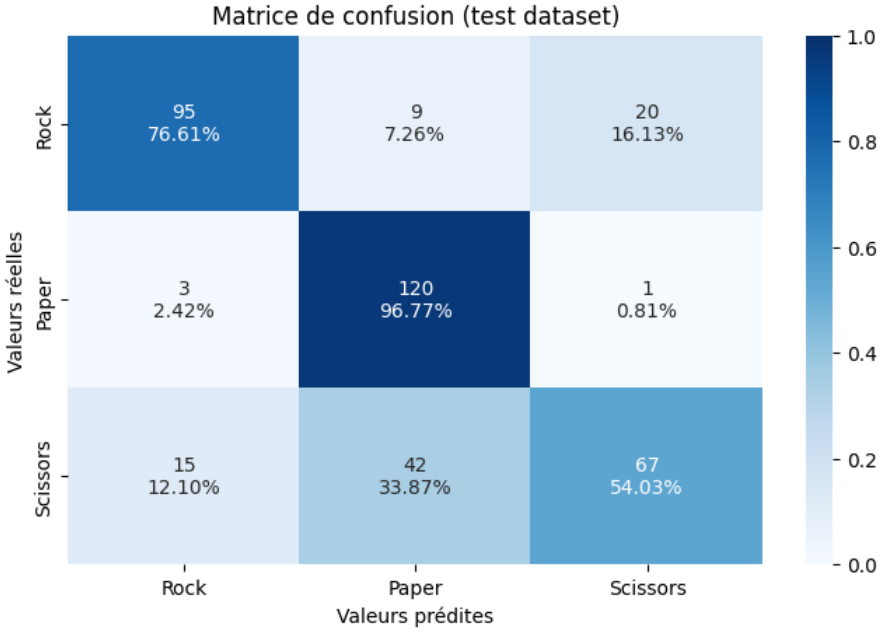
Cette étape permet de modifier la valeur des pixel pour les avoir dans un format plus adéquat pour leur analyse.

J'ai aussi fait de la data-augemntation, j'ai utilisé les propriétés de Keras, pour mettre en place cette data-augmentation directement DANS les couches de mon modèle.

```
model.add(tf.keras.layers.experimental.preprocessing.RandomFlip('horizontal'))
model.add(tf.keras.layers.experimental.preprocessing.RandomRotation(0.2))
```

Ces augmentations ne sont pas le fruit du hasard, j'ai choisi ce RandomFlip, car il simule bien le fait d'être droitier ou gaucher ; ce qui a une incidence seulement sur ciseaux. Lorsque je ne fais pas

de data-augmentation, il y a beaucoup de confusion entre ciseaux et feuille. L'autre augmentation (RandomRotation) est quant à elle plus pour pallier la confusion entre pierre et feuille. Voici ce à quoi ressemble la Matrice de confusion sans data augmentations



En résumé, je passe de 63 à 90% d'accuracy.

Maintenant que nous avons un modèle de plus performant

3 Un peu de convivialité

3.1 État des lieux

Pour l'instant, le joueur doit fournir une image à la machine (pierre, feuille ou ciseaux), cette interaction est peu naturelle et ne donne pas une expérience des plus conviviale

3.2 L'interface Originale

Pour l'instant l'utilisateur doit fournir image (avec soit des ciseaux, soit une feuille ou papier), cette interaction est peu (voire pas) conviviale. Donc le joueur fournit une image, qui désormais est désormais presque toujours reconnut(un petit plus que 9 fois sur 10. Il faut noté que les images fournies doivent représenter des mains faisant un des trois signes.

3.3 Ajout de la computer vision

Je pense développer cette option en utilisant la librairie openCV qui offre de nombreux avantages, comme par exemple la gratuité de l'open-source et la force de sa communauté de développeurs. Je pense donc développer une API avec un visuel proche de celui-ci, permettant donc de jouer sur un appareil portable :

4 Et ça marche Bien ?

4.0.1 Tests de non régression

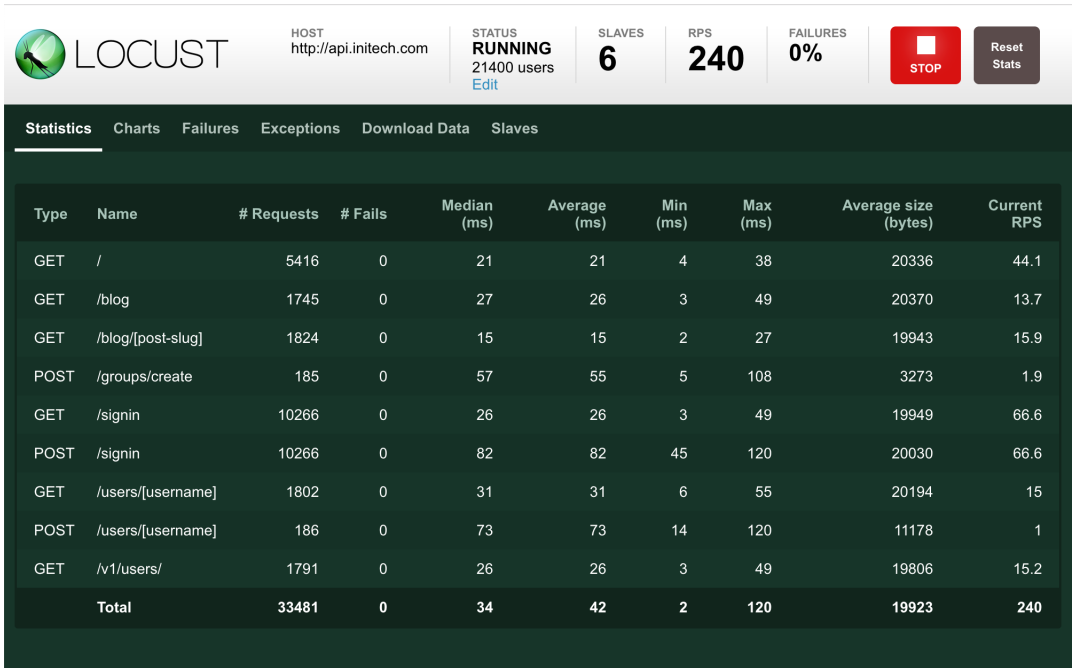
```
import time
...
start= time.time()
...
script du shifumi
...
end= time.time()

elapsed=end-start
print("Temps d'exécution: ",elapsed)
...
```

Cette fonction permet de voir si la nouvelle version de l'application est moins réactive que l'ancienne. Lors de la reconnaissance par l'ancienne version, le temps que l'image (300*300) soit chargée puis analysée, avoir un retour(pas toujours pertinent) prenait environ 1.5 secondes. Désormais le retour (avec beaucoup moins d'erreur) prend un peu moins de 2s. Malgré un imperceptible allongement du temps de réponse, l'évolution de l'API donne un nouveau regain d'intérêt pour le shifumi.

4.0.2 Tests fonctionnels

- Disponibilité du service Pour vérifier cela, j'utilise le Framework Locust ; je l'utilise car il a le gros avantage de déjà être utiliser par des services comme AWS, heroku, etc. Ce qui est l'assurance d'avoir un framework stable et s'améliore



Type	Name	# Requests	# Fails	Median (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS
GET	/	5416	0	21	21	4	38	20336	44.1
GET	/blog	1745	0	27	26	3	49	20370	13.7
GET	/blog/[post-slug]	1824	0	15	15	2	27	19943	15.9
POST	/groups/create	185	0	57	55	5	108	3273	1.9
GET	/signin	10266	0	26	26	3	49	19949	66.6
POST	/signin	10266	0	82	82	45	120	20030	66.6
GET	/users/[username]	1802	0	31	31	6	55	20194	15
POST	/users/[username]	186	0	73	73	14	120	11178	1
GET	/v1/users/	1791	0	26	26	3	49	19806	15.2
	Total	**33481**	**0**	**34**	**42**	**2**	**120**	**19923**	**240**

- Test de l'interface joueur J'ai effectué différents tests validant cette fonction, voici le détail des tests :

	Je peux me connecter	OK
	Le signe est reconnu	OK
	J'ai un résultat du set	OK
	J'ai le résultat du match	OK
	Je peux relancer une partie	OK

4.0.3 Estimation de la charge

L'estimation de la charge prend en compte la partie développement et la partie maintenance.

- Analyse de l'existant 2j/homme
- Développement de la nouvelle IA 4j/homme
- Test de la solution produite 3j/homme

Mise en production

- Maintenance et amélioration de l'IA 2j/homme pendant 3 semaines