COMPUTER VISION APPLIQUEE AU JEU D'ECHECS

Kulcsar Jeremy, Fayolle Lucas
Option OSY – Computer Vision
Ecole Centrale Paris - 2020

***Abstract* – Dans ce projet, nous allons réaliser une IA qui jouera aux échecs de manière autonome. Celle-ci prendra une photo de l'échiquier, reconnaîtra la position des pièces, et enverra les informations à une IA interne qui calculera le coup optimal à jouer afin de gagner la partie.**

I – INTRODUCTION

C'est en 1997 que l'on a pris conscience de la puissance de l'IA pour les jeux de plateaux quand Kasparov, ancien champion du monde d'échecs, a été battu par le superordinateur Deep Blue d'IBM. Depuis, les technologies n'ont cessé de s'améliorer, et l'étude de l'IA pour les jeux de plateau est devenu un domaine à part entière. Le niveau des IA est tel que celles d'aujourd'hui s'entrainent en jouant contre elles-mêmes, le niveau des humains ayant été largement dépassé.

Ces IA sont spécialisées dans le calcul du coup optimal pour vaincre l'adversaire. On leur envoie les informations du plateau, et celles-ci renvoient le coup à jouer en retour. Notre idée a été de coupler ce concept à ceux que l'on a vus en cours de Computer Vision : et si, plutôt que de devoir leur envoyer les informations, on permettait à l'IA de directement lire le plateau ?

II – PIPELINE

Nous avons construit un pipeline dont chaque étape aura un rôle précis pour notre IA, de la reconnaissance du plateau au calcul du coup optimal.

Les différentes parties détailleront le sens et le fonctionnement de chacune d'entre elles.

1. Reconnaissance visuelle

Dans ce premier bloc, l'IA prendra une photo de l'écran. Sur cette photo, il y aura évidemment l'échiquier, mais il y aura également l'environnement : il faut donc trouver une manière d'isoler l'échiquier du reste de l'image pour la suite.

Considérons l'image suivante [Fig1], qui correspond à un screenshot sur le site chess.com.

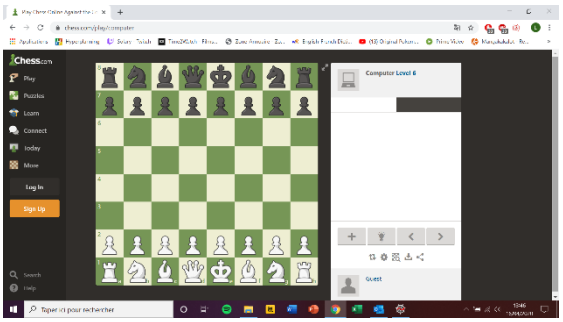

Fig1 : Screenshot sur chess.com

Nous lui appliquons trois transformations :

- Mise en noir et blanc
- Application d'un kernel pour aiguiser les lignes

On obtient alors l'image [Fig2].

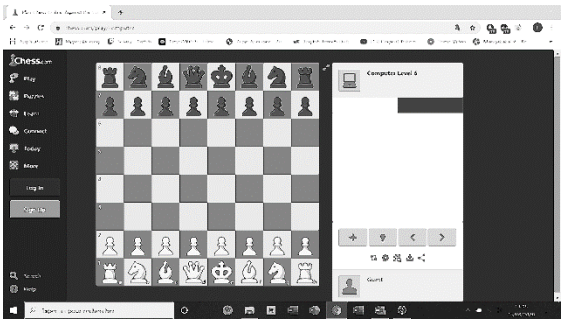

Fig2 : Screenshot après premières modification

Puis, on applique quatre transformations supplémentaires afin de mieux faire apparaître les lignes horizontales et verticales :

- Threshhold binaire
- Kernel rectangulaire
- Dilation de l'image
- Erosion de l'image

On obtient finalement l'image suivante en [Fig3].

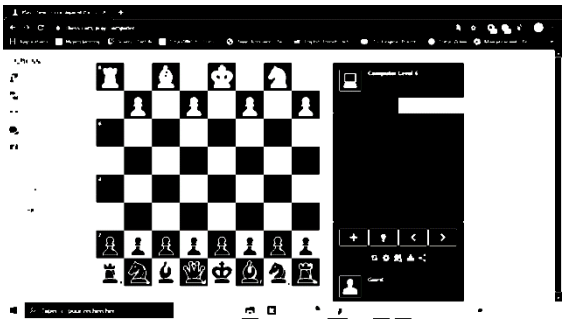

Fig3 : Screenshot après modifications ultérieures

L'image est pratiquement prête. Il suffit d'appliquer deux transformations de Sobel, selon x et y pour ne garder que les lignes horizontales et verticales. Les résultats sont montrés sur [Fig4] et [Fig5].

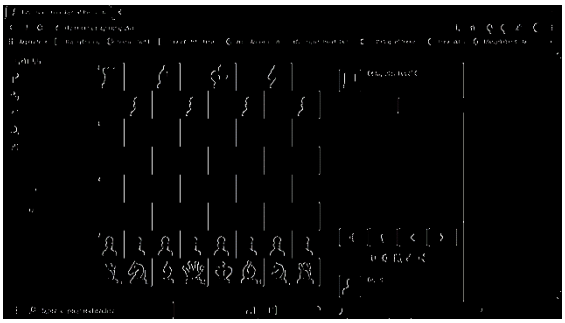

Fig4 : Screenshot après Sobel selon x

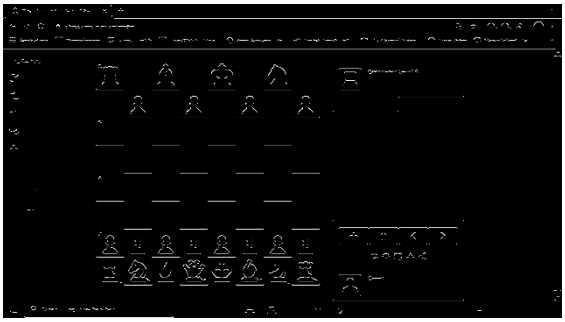

Fig5 : Screenshot après Sobel selon y

Nous avons désormais nos lignes clairement définies. Pour les identifier, nous appliquons une transformée de Hough. Il s'agit d'une transformée pensée pour détecter les lignes dans des images numériques.

« *La transformée de Hough consiste à représenter chaque point de contour détecté dans un espace de paramètres à deux dimensions. La transformée de Hough d'un point de l'image analysée est la courbe de l'espace des paramètres correspondant à l'ensemble des droites passant par ce point. Si des points sont colinéaires, alors toutes les courbes de l'espace de paramètres se coupent au point représentant la droite en question. » [Wikipédia : Transformée de Hough]*

Cette transformée peut être vue comme un algorithme en trois étapes :

- Pour chaque point de contour détecté, transformation du point vers la courbe correspondante de l'espace des paramètres

- Construction d'une matrice dite d'accumulation, qui réunira en fait toutes ces courbes

- Détection des pics dans cette matrice

Nous avons donc procédé à cette transformation à partir des figures 4 et 5. Les résultats sont visibles sur les graphes [Fig6].

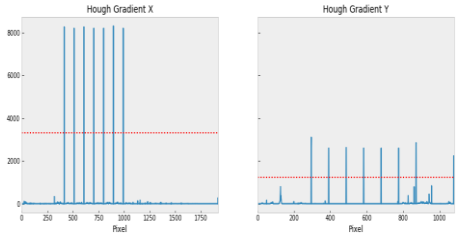

Fig6 : Graphes de Hough avec threshold

Il suffit désormais de trouver l'intersection entre les lignes pour quadriller l'échiquier, comme visible sur la [Fig7].

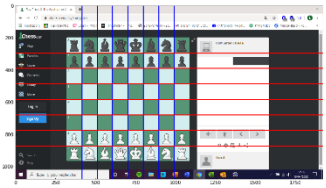

Fig7 : Screenshot avec quadrillage de l'échiquier

Une fois que les lignes de l'échiquier ont été trouvées sur le screenshot, le reste du code permet de découper le plateau. Nous pouvons alors nous concentrer dessus, et étudier chaque case séparément.

La suite du processus fait appel à deux CNN :

- Un premier CNN qui permet d'analyser si la case observée est vide ou non

- Un deuxième CNN qui permet, si la case n'est pas vide, de détecter la pièce présente sur la case.

Nous avons choisi des architectures relativement simple (3 couches convolutives et 1 couche dense) puisqu'il est relativement simple de distinguer les différentes pièces d'échecs.

Le code renvoie finalement l'information complète du plateau, à savoir où est chaque pièce, en format FEN.

2 – Calcul du coup optimal

Les informations lues lors de la première étape seront alors envoyées à une IA interne qui s'occupera de calculer le coup optimal à jouer. Celle-ci renverra ainsi le meilleur coup.

Nous avons choisi d'utiliser une IA d'échecs déjà préconçue pour jouer. Nous utilisons donc Arasan, une IA free of use.

Un exemple de réponse de l'IA est « e2e4 ». Cet exemple de coup optimal signifie que la pièce localisée en e2 devra être déplacée en e4.

3 – Reconnaissance de la case à jouer

Pour cette étape, notre code va de nouveau faire de la reconnaissance visuelle : il s'agit de déterminer la localisation des cases de départ et d'arrivée du coup sur l'image.

Pour cela, nous utilisons le même algorithme que lors de la première étape.

4 – Jouer le coup

Une fois que l'on a l'information du coup à jouer et la localisation des cases sur l'image il ne reste plus qu'à envoyer l'information même du jeu du coup à notre ordinateur.

Ensuite, il suffit de cliquer successivement sur les deux cases pour déplacer la pièce.

Pour rester sur notre exemple « e2e4 », notre code va regarder la case e2, cliquer dessus, et ensuite reconnaître la case e4 pour cliquer dessus et provoquer le déplacement.

III – ENTRAINEMENT ET RESULTATS

1 – Entrainement

Nous avons utilisé un set de 10 000 images de plateaux d'échecs pour entraîner la partie « reconnaissance visuelle » de notre IA. Celle-ci devant regarder chaque case individuelle de chaque échiquier, il y a en fait 640 000 images pour permettre l'entrainement.

Nous avons opté pour un split 90/10 afin de donner 9000 plateaux soit 576 000 cases pour permettre un apprentissage sur un nombre élevé de données.

Notre test set est alors constitué de 1000 images de plateau, soit 64 000 cases à tester.

Après avoir entraîné notre modèle, notre accuracy sur le test set est de 100% : C'est ici souhaité et normal car l'objet est de reconnaître exactement la pièce sur la case afin de passer à la suite du pipeline.

2 – Mise en situation

Notre IA est capable de jouer contre un humain ou une machine en ligne. Nous avons pu la tester en la faisant jouer des centaines de parties contre des adversaires réels en ligne sur un site internet.

Après quelques jours de parties, notre IA a atteint un score supérieur à 2600 sur ce site, ce qui correspond à un niveau très élevé. Bien entendu, le niveau de l'IA dépend du programme utilisé pour calculer les meilleurs coups. Néanmoins, notre pipeline est suffisamment rapide pour permettre à l'IA de jouer dans des cadences blitz (3min par partie par joueur) avec une très bonne performance.

IV – CONCLUSION

Notre IA marche très bien. Elle est capable de jouer aux échecs en ligne sans problème à une cadence relativement soutenue. L'échiquier est repérable dans la majorité des cas sur une image (il faut qu'il soit suffisamment grand et qu'aucun autre quadrillage ne soit présent sur la même image).

Pour aller plus loin, il serait intéressant d'être capable de détecter plusieurs échiquiers sur la même image ou bien détecter des échiquiers non alignés avec l'image.