

Vision par ordinateur et détection d'objets

Rapport de projet

Colin BOUREZ, Gabriel GONTIER

7 avril 2025



Introduction

Ce projet consiste à nous permettre de jouer à **Space Invaders** par **détection de mouvements** face à une caméra. Pour cela, il va s'agir d'entraîner un **réseau de neurones** convolutionnel afin de réaliser une tâche de **classification** à partir d'échantillons de mouvements créés nous-mêmes.

Voici les commandes attendues :

- **Lever la main gauche** : déplacement vers la gauche
- **Lever la main droite** : déplacement vers la droite
- **Lever les deux mains** : démarrer le jeu
- **Lever un poing** : tirer

Nous avons donc commencé par capturer un total d'environ **5000 images** des différents mouvements que nous avons réalisés face à la caméra. Précisons que nous avons capturé des images à part pour former un échantillon de **test** bien distinct de celui d'**entraînement**.

Ensuite, nous avons créé un modèle simple de **CNN** puis un modèle similaire mais plus complexe. Nous les avons entraînés et testés à partir de nos données.

Enfin, nous avons modifié le script Python qui sert de **module de contrôle** afin d'adapter les commandes du jeu à la reconnaissance de mouvements face caméra à partir du CNN.

Table des matières

1	Capture d'images	3
2	Création d'un modèle	4
2.1	Mise en place	4
2.2	Chargement des données	4
2.3	Traitement et analyse des données	5
2.4	Augmentation des données	6
2.5	Construction du modèle	6
2.6	Entraînement du modèle	6
3	Module de contrôle	6
4	Jouer avec ses mains	6

1 Capture d'images

Afin d'effectuer une capture d'images de nos mouvements, nous avons écrit un **script Python** *Capture.ipynb* qui accède à la webcam de l'ordinateur sur lequel il est exécuté.

Ce script capture des images sur une fréquence de **20 par secondes** jusqu'à arrêt par l'utilisateur via la touche *q*.

Le programme permet ensuite d'**enregistrer les différentes images** ainsi récupérées dans le dossier où Jupyter Notebook est ouvert et avec le titre souhaité.

Enfin, le script affiche le nombre d'images qui ont été capturées et enregistrées afin de suivre la taille des différentes **classes** de l'échantillon, à savoir :

- **LEFT** : main gauche levée
- **RIGHT** : main droite levée
- **ENTER** : deux mains levées
- **FIRE** : un poing levé

Pour répartir les photographies dans leurs classes respectives, nous les avons manuellement placées dans un dossier portant le nom de la classe en question. Nous avons finalement obtenu un dossier *Train* et un dossier *Test* pour les échantillons d'**entraînement** et de **test** respectivement, chacun contenant un dossier pour chaque classe.

Voici quelques exemples d'images ainsi récupérées :



FIGURE 1 – 'Enter'



FIGURE 2 – 'Fire'

Les dossiers des échantillons sont hébergés aux liens suivants :

- **Entraînement** : <https://drive.google.com/drive/folders/1S5nBNDMvK647DkojJQlNKy6zglN6Pe0u?usp=sharing>
- **Test** : https://drive.google.com/drive/folders/1DuO82yhIdM51kMhKFZFUmBgFS_vFzXIX?usp=drive_link

2 Création d'un modèle

2.1 Mise en place

Dans le script **model.ipynb**, nous commençons par importer les modules nécessaires tels que **PyTorch** pour programmer un réseau de neurones, ou **MediaPipe** afin de détecter nos mains sur les jeux de données.

Nous fixons ensuite des seeds aléatoires pour permettre une reproductibilité de nos expériences. Enfin, nous définissons le périphérique qui sera utilisé pour les calculs : un **GPU** si disponible ou, à défaut, le **CPU** de l'ordinateur.

Précisons que pour faire fonctionner le script qui suit, il est nécessaire de télécharger les dossiers **Train** et **Test** aux liens ci-dessus dans le répertoire **space-invaders** du projet.

2.2 Chargement des données

Nous initialisons **MediaPipe** afin de récupérer 63 coordonnées de mains (ou landmarks) par image. Le nombre maximal de mains détectables est fixé à 2 en raison de la commande **ENTER**.

Une fonction **extract_landmarks** est définie pour cette extraction de landmarks et va boucler sur la totalité des dossiers d'images d'entraînement puis de test via une fonction **process_dataset** afin d'enregistrer les coordonnées dans deux fichiers **CSV** distincts : **train_landmarks.csv** et **test_landmarks.csv**.

Enfin, les données des deux fichiers CSV sont lues et chargées en tant que **DataFrames Pandas**. Nous en affichons à chaque fois les 5 premières lignes pour vérifier que ce chargement s'est bien déroulé. Nous observons alors que chaque ligne correspond à une image : son label se trouve en première colonne puis nous retrouvons ses 63 landmarks pour un total de 64 colonnes.

	label	x_0	y_0	z_0	x_1	y_1	z_1
0	ENTER	0.313832	0.764376	2.474214e-07	0.358308	0.734033	-0.026857
1	ENTER	0.650379	0.380406	2.110198e-07	0.603407	0.368063	-0.015487
2	ENTER	0.663164	0.614228	1.384285e-07	0.619237	0.591525	-0.012178
3	ENTER	0.338837	0.690563	3.175797e-07	0.293740	0.682436	-0.011694
4	ENTER	0.716060	0.874130	3.947711e-07	0.661055	0.861623	-0.025433

FIGURE 3 – **Set d'entraînement** : labels et 7 premiers landmarks des 5 premières images

2.3 Traitement et analyse des données

Il s'agit ici de préparer les données d'entraînement et de test des deux fichiers CSV en les normalisant à l'aide d'un scaler **MinMax**. Cela fait en sorte que tous les landmarks de chaque image soient compris **entre 0 et 1** : nous n'avons plus de valeurs négatives. Pour comparaison, voici le même aperçu du jeu d'entraînement que ci-dessus :

	label	x_0	y_0	z_0	x_1	y_1	z_1
0	ENTER	0.329598	0.679027	0.780843	0.373168	0.673895	0.312188
1	ENTER	0.662875	0.273039	0.747177	0.631281	0.284959	0.485181
2	ENTER	0.675536	0.520269	0.680040	0.647952	0.522444	0.535528
3	ENTER	0.354360	0.600981	0.845729	0.305172	0.619060	0.542895
4	ENTER	0.727917	0.795074	0.917120	0.691990	0.809492	0.333856

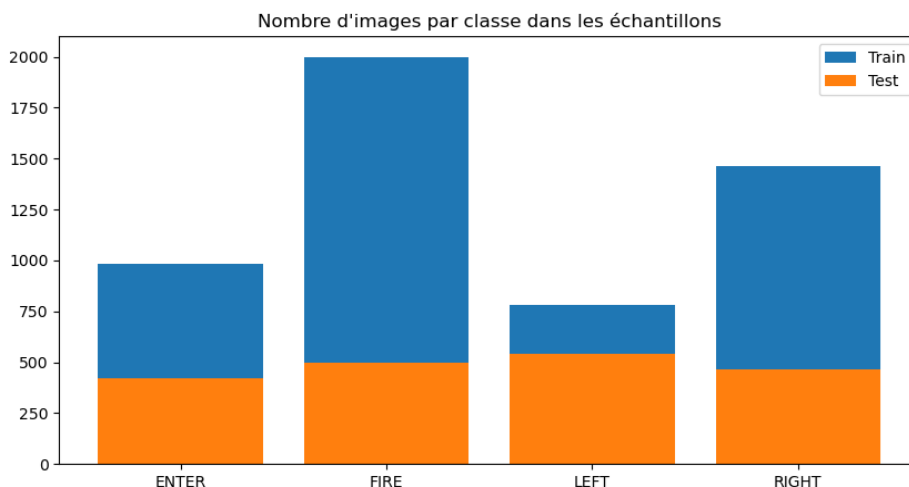
FIGURE 4 – **Set d'entraînement normalisé** : 8 premières colonnes et 5 premières lignes

Nous appliquons cette même transformation sur **l'échantillon de test**.

Nous séparons ensuite les features et les labels dans les deux jeux de données : la première colonne de chacun des deux échantillons correspond aux labels **y_train** et **y_test**, tandis que les 63 autres colonnes donnent les features **X_train** et **X_test** :

- **X_train** : 5158 lignes (images) avec 63 colonnes (landmarks) à chaque fois
- **y_train** : on a bien 5158 lignes (labels) aussi
- **X_test** : 1833 lignes (images) avec 63 colonnes (landmarks) à chaque fois
- **y_test** : on a bien 1833 lignes (labels) aussi

Pour conclure cette section, nous définissons une fonction **analyze_dataset** afin d'analyser les échantillons d'entraînement et de test. Nous l'appelons donc sur **X_train** et **X_test** et obtenons de nouvelles informations : le nombre d'images par classe et par échantillon. La représentation graphique de ces informations apparaît ci-dessous. Notez que nous avons à l'origine 500 images supplémentaires dans la classe **FIRE** mais nous les avons supprimées pour rééquilibrer légèrement les classes.



2.4 Augmentation des données

2.5 Construction du modèle

2.6 Entraînement du modèle

3 Module de contrôle

4 Jouer avec ses mains