Université Gustave Eiffel

M2 Mathématiques et Applications
Parcours Probabilités et Statistiques

Projet vision par ordinateur : "Space Invaders"

Riyane HAMAMTI, Mohamed ELHABIB, Ayoub BENHEDDI, Abdoul Wahide MAMA

Enseignant : **Florian VALADE**

2024-2025

Introduction

Ce projet a pour objectif de proposer une nouvelle manière d'interagir avec le jeu Space Invaders. Il consiste à remplacer les commandes classiques au clavier par une interface basée sur des gestes de la main, détectés en temps réel à l'aide d'une caméra.

Dans ce rapport, nous détaillerons les points suivants :

— la collecte des données nécessaires à l'apprentissage des gestes,
— l'entraînement d'un modèle de classification à partir des points clés de la main,
— Les défis rencontrés et les solutions apportées.

Architecture du projet

L'objectif est de créer un jeu de données avec plusieurs gestes de la main. Ces gestes serviront à entraîner un modèle qui pourra ensuite les reconnaître automatiquement.

1. Collecte des données

Les gestes à reconnaître

On a choisi trois gestes simples :

- LEFT : la main va vers la gauche,
- RIGHT : la main va vers la droite,
- FIRE : la main est ouverte face à la caméra (comme un stop).

L'outil utilisé : MediaPipe

Pour détecter les gestes de la main, nous avons utilisé MediaPipe Hands, une bibliothèque développée par Google. Cette bibliothèque permet d'identifier une ou deux mains dans une image et d'en extraire des points clés, appelés landmarks. Chaque main est représentée par 21 points, allant du poignet jusqu'au bout des doigts, avec des coordonnées x, y et parfois z (la profondeur par rapport à la caméra). Les coordonnées sont normalisées, c'est-à-dire qu'elles sont comprises entre 0 et 1 en fonction de la taille de l'image.
Grâce à ces points, il est possible de visualiser la main à l'écran, d'observer sa position et, surtout, de reconnaître des gestes (comme si la main se déplace à gauche, à droite ou si un doigt est replié).

C'est avec ces données que nous avons construit un jeu de données pour entraîner un modèle capable de reconnaître trois gestes, ce que nous détaillerons dans la prochaine section.

Fonctionnement du modèle MediaPipe Hands et lien avec les CNN

La détection des gestes dans ce projet s'appuie sur le pipeline *MediaPipe Hands*, qui combine deux étapes complémentaires : la détection de la main et la régression des points clés (landmarks). Ces

deux modules sont fondés sur des réseaux de neurones convolutifs (CNN), comme ceux étudiés en cours de vision par ordinateur.

Pipeline du modèle

Le pipeline du modèle se compose de deux étapes principales :

— **Détecteur de main (Hand Detector)** : L'entrée du modèle est une image de dimensions $192 \times 192 \times 3$. Le modèle utilisé est un réseau de neurones convolutifs (CNN) de type SSD, qui est une architecture de détection en une étape. Son objectif est de localiser la main dans l'image en générant des boîtes englobantes autour de la main à partir d'ancres (anchors). Après la détection, un post-traitement est appliqué sous forme de Non-Maximum Suppression (NMS), permettant de filtrer les doublons et pout garder que la meilleure détection pour chaque main.

— **Trackeur de landmarks (Hand Landmark Model)** : L'entrée du modèle est la région rognée autour de la main, de dimensions $224 \times 224 \times 3$, extraite du détecteur de main. Ce modèle utilise un CNN de régression pour prédire plusieurs sorties. Il fournit d'abord la probabilité de présence d'une main dans la région donnée. Ensuite, il génère les coordonnées normalisées (en 2D ou 3D) des 21 **landmarks anatomiques** de la main, qui représentent les positions clés des doigts et des articulations. Enfin, le modèle prédit l'orientation de la main, en indiquant si elle est de type **gauche** ou **droite**.

Ainsi, ce pipeline combine efficacement la détection de la main et le suivi précis des points clés de la main. Ceci nous offre une solution complète pour l'analyse des mouvements et des gestes de la main.

Corrélation avec les notions du cours

Le modèle *MediaPipe Hands* s'appuie sur plusieurs éléments plusieurs notions vues en cours que l'on peut relier directement aux notions étudiées :
D'abord, la détection de la main repose sur un réseau de type *Single Shot Detector (SSD)*, un modèle CNN utilisé pour la détection d'objets, suivi d'un post-traitement appelé *Non-Maximum Suppression (NMS)* pour éviter les doublons. Une fois la main détectée, une image recadrée autour de celle-ci est envoyée dans un second réseau, cette fois pour prédire les coordonnées des points clés de la main (21 au total, en 3D). Cette étape correspond à une tâche de régression avec un CNN, où les sorties sont continues.

Le fonctionnement général du modèle suit un schéma classique : image en entrée, extraction de caractéristiques via des *feature maps*, puis prédictions. Cela illustre bien le principe de propagation avant dans les réseaux convolutifs. Par ailleurs, deux autres éléments sont également intéressants : la normalisation des coordonnées par la taille de la paume, qui permet de varuer l'échelle, et le fait de travailler sur une image recadrée autour de la main, ce qui améliore la précision grâce à une meilleure focalisation contextuelle. Enfin, le modèle est entraîné à partir d'un mélange de données réelles et

synthétiques (issues du modèle 3D GHUM), ce qui montre l'intérêt de compléter les bases de données avec d'autres données quand c'est possible.

Ce modèle reste particulièrement adapté à notre projet, car il fonctionne en temps réel et fournit une description complète de la main. Il constitue un excellent extracteur de caractéristiques afin de faciliter l'entraînement d'un modèle de classification supervisée comme le Random Forest pour la reconnaissance de gestes.

Récupération des données

Dans cette étape, l'objectif est de capturer les images de la main via la caméra et de récupérer les coordonnées des points clés de la main. Pour cela, nous utilisons les bibliothèques OpenCV et *MediaPipe*.

On commence par ouvrir la caméra avec OpenCV grâce à la fonction `cv2.VideoCapture(0)`. Cela permet de prendre des photos en temps réel depuis la webcam. Dans une boucle, chaque image est lue à l'aide de `cap.read()`, afin d'obtenir l'image dans une variable appelée `frame`. Chaque image est ensuite envoyée à MediaPipe, pour l'analyser. Pour chaque main détectée, les points sont extraits et dessinés sur l'image.

Une fois les données capturées et les points de la main détectés, il faut enregistrer ces informations sous forme de fichiers .csv pour pouvoir les utiliser pour l'entraînement du modèle. Pour chaque geste (LEFT, RIGHT, FIRE), on va collecter un certain nombre d'exemples. Chaque exemple est une ligne contient les coordonnées des 21 points de la main (soit 63 valeurs : 3 valeurs x, y, z pour chaque point).

2. Entraînement du modèle

Dans cette section, nous allons aborder l'étape de l'entraînement du modèle et la reconnaissance des gestes de la main. Pour ce faire, nous avons utilisé un algorithme de classification supervisée, plus précisément un Random Forest. Ce choix s'explique par sa capacité à gérer des données complexes, son efficacité sur des jeux de données modérés, et sa tolérance au bruit.

Les données utilisées pour entraîner notre modèle proviennent des coordonnées des landmarks de la main détaillées précédemment. Chaque geste, qu'il s'agisse de "gauche" (LEFT), "droite" (RIGHT) ou "tir" (FIRE), est associé à un ensemble de coordonnées 3D représentant les points clés de la main. Une fois les données ont été prétraitées et organisées, on les fusionne en un seul ensemble. Ce dernier est découpé en ensembles d'entraînement (80%) et de test (20%).

Défis rencontrés et solutions apportées

Ce projet a soulevé plusieurs défis techniques, principalement liés à la communication en temps réel entre les différents modules (détection gestuelle, serveur WebSocket, et jeu HTML5). Voici les principales difficultés rencontrées ainsi que les solutions apportées pour les surmonter.

1. Spam de commandes gestuelles

L'un des premiers problèmes rencontrés a été le *spam de commandes*, en particulier pour le geste `FIRE`. En l'absence de mécanisme de régulation, le modèle de reconnaissance envoyait la commande associée à chaque image capturée (environ 30 fois par seconde), ce qui provoquait un comportement instable dans le jeu, notamment des tirs continus incontrôlables.

Pour résoudre cela, un court délai (*cooldown*)(*cooldown*) a été introduit entre deux commandes identiques afin d'assurer ainsi une fréquence raisonnable d'activation par geste (environ une commande par demi-seconde). Cela a permis de simuler plus fidèlement un appui naturel sur une touche clavier.

2. Synchronisation WebSocket - Front-end

Un autre défi important concernait la **synchronisation entre le serveur WebSocket Node.js et le jeu dans le navigateur**. Initialement, les commandes envoyées par le script Python étaient bien reçues côté serveur, mais n'étaient pas interprétées par le jeu.
L'analyse du comportement du script de contrôle clavier a permis de comprendre que le serveur devait transformer chaque commande textuelle (par exemple `"LEFT"`) en un `event JSON` de type `keydown` suivi d'un `keyup`, avec les bons `keyCode`. Une fois cette logique ajoutée au serveur, et le jeu configuré pour écouter ces événements synthétiques, la synchronisation a pu être correctement établie. Cette étape reste importante pour assurer une communication fluide et naturelle entre les gestes détectés par la webcam et les actions visibles dans le jeu.