Rapport - TP - Reconnaissance de formes

Introduction

L'objectif de ce TP était de mettre en place un système capable de réaliser de la reconnaissance de formes.
Pour ce projet, j'ai choisi de m'intéresser à une problématique pratique : la détection du nombre de doigts levés sur une main. Ce choix permettait d'explorer à la fois :

- la reconnaissance visuelle,
- l'utilisation de modèles entraînés,
- les traitements géométriques basés sur les points clés d'une main.

Au fil du TP, j'ai expérimenté plusieurs approches successives, chacune révélant ses avantages et ses limites.

Les fichiers à utiliser dans ce projet sont : `alogorithme.py` et `algorithme_model.py`

Première approche : contruction d'un modèle entrainé

Création et entrainenement du model

Dans un premier temps, j'ai choisi de développer un modèle d'apprentissage automatique chargé de reconnaître directement, à partir d'une image, le nombre de doigts levés.
Le modèle a été conçu en Python avec TensorFlow/Keras (fichiers `create_model.py` et `train.py`) et entraîné sur plusieurs datasets trouvés sur Kaggle.

Qualité insuffisante des datasets publics

Lors de l'entraînement, j'ai constaté de nombreuses limites dans les datasets existants :

- beaucoup d’images représentaient des mains découpées sur fond noir, très éloignées d’un usage réel ;
- la diversité des positions, des angles ou des personnes était souvent trop faible ;
- certains jeux de données contenaient trop peu d’images pour une bonne généralisation.

Cela entraînait un modèle qui semblait fonctionner pendant l’entraînement, mais se révélait très peu fiable lorsque je lui fournissais des images issues de ma propre caméra.

Deuxième approche : apprentissage supervisé en temps réel

Constat et décision

Face à la faiblesse des datasets existants, ma première idée a été de créer mon propre dataset, directement à partir de ma webcam.
Cependant, avec un flux d’environ 10 images/seconde, générer un dataset comparable aux standards (~54 000 images) aurait nécessité près de 2 heures d’enregistrement continu — sans compter les risques de biais (même main, même environnement).

Développement d'un outil de colecte supervisé

Pour faciliter ce processus, j’ai commencé à développer une interface permettant :

- d’afficher la caméra ;
- de capturer automatiquement les images ;
- d’associer à chaque image la valeur correcte à apprendre.

Pour que ce système fonctionne, il me fallait un algorithme externe capable de fournir une estimation fiable du nombre de doigts levés.
C’est ce rôle qu’occupait le fichier algorithme_model.py, destiné à alimenter le modèle en données étiquetées.

Limites rencontrées

Malgré plusieurs essais, cette approche s’est révélée très complexe :

- la génération du dataset était lente et chronophage ;
- la qualité des images n'était pas suffisante ;
- le modèle apprenait mais ne produisait pas de résultats cohérents en conditions réelles.

Ces limites m'ont poussé à explorer une approche plus déterministe.

Troisième approche : un algorithme géométrique

Après plusieurs essais infructueux avec les modèles entraînés et supérvisé, j'ai décidé de concevoir une solution basé uniquement sur un algorithme utilisant la géométrie de la main, en utilisant les coordonnées fournies par `MediaPipe`.

Principe

L'idée est de déterminer si un doigt est levé en analysant :

- la position relative de ses joints (MCP, PIP, DIP, TIP),
- les distances successives par rapport au poignet,
- les angles formés entre les différentes phalanges.

Cette approche permet de :

- ne plus dépendre d'un dataset ;
- obtenir un résultat précis et interprétable ;
- éviter les comportements imprévisibles des modèles mal entraînés.

Implémentation

L'algorithme final se trouve dans `algorithme.py`.
Il calcule :

- l'état (levé ou non) de chaque doigt,
- le nombre total de doigts levés,
- un dictionnaire debug contenant des valeurs utiles pour l'analyse (distances, angles, etc.).

Cette approche s'est révélée la plus stable et la plus efficace dans le cadre du TP.

Installation

Les fichier des ce projets utilise certaine version de librairie spécifique, il est donc recommandé d'utiliser les commandes suivantes pour créer un environnement python adapté.
Avant de taper la première commande, dans un terminal, placez vous à la racine du projet (`cd tp2`) :

```
python3 -m venv env
source env/bin/activate
cd src
```

Ensuite choisissez le fichier à éxécuter :

- La version avec le model (algorithme simple)

```
python3 algorithme_model.py
```

- La version avec l'algorithme plus complèxe :

```
python3 algorithme.py
```

Description des fichiers

- Les fichiers `algorithme.py` et `algorithme_model.py` sont les fichiers pemettant respectivement une reconnaissance précise et par IA d'une main sur un flux de caméra.
- Les fichiers `model.py` et `create_model.py` permette la création et l'usage d'un model.
- Le fichier `predict_one.py` permet de tester le model sur une image.
- Le fichier `test.py` m'a servis à visualiser le contenu d'une image passer au model. (Ne marche pas dans l'état => chager l'image à visualiser).
- Le fichier `train.py` m'a permis de créer et d'entrainer mes premiers models.
- Les fichers contenue dans le dossier `tools` m'ont permis de séparer et d'augementer la taille de certain datasets.

Conclusion

Ce TP m’a permis d’explorer trois méthodes différentes de reconnaissance de formes :

- Apprentissage sur datasets publics → résultats insuffisants à cause de la qualité et de la diversité des images.
- Apprentissage supervisé en temps réel → approche intéressante mais trop lourde, lente et difficile à fiabiliser avec du matériel simple.
- Algorithme géométrique basé sur MediaPipe → solution la plus précise, la plus stable et la plus adaptée au contexte du TP.

Cette évolution progressive m’a permis de mieux comprendre les limites des modèles d’apprentissage lorsqu’ils ne disposent pas d’un dataset adapté, et de découvrir la puissance d’un raisonnement géométrique appliqué à une structure articulée comme la main.