Détection de la langue des signes avec Python

Bel hadj amor Noura

20 décembre 2024

Table des matières

1 Introduction **2**

2 Collecte des données **2**
2.1 Configuration de l'environnement 2
2.2 Configuration de la caméra 2
2.3 Collecte des données pour chaque classe 3
2.4 Processus de collecte des images 3
2.5 Libération des ressources 4

3 Traitement des données **5**
3.1 Initialisation de Mediapipe pour la détection des mains 5
3.2 Lecture et traitement des images 5
3.3 Sauvegarde des données traitées 7

4 Modèle de train **7**
4.1 Chargement et préparation des données 7
4.2 Séparation des données en ensembles d'entraînement et de test 8
4.3 Entraînement du modèle 8
4.4 Évaluation du modèle 9
4.5 Sauvegarde du modèle 9

5 Modèle de test **9**
5.1 Chargement du modèle entraîné 9
5.2 Capture vidéo et détection des mains 10
5.3 Préparation des données pour la prédiction 10
5.4 Prédiction du geste 10
5.5 Affichage des résultats 11
5.6 Boucle de test en temps réel 11
5.7 Libération des ressources 11

6 Conclusion **13**

1 Introduction

La détection de la langue des signes est un domaine fascinant qui combine la vision par ordinateur, l'apprentissage automatique et la reconnaissance des gestes pour permettre aux personnes malentendantes de communiquer plus facilement. Grâce à l'utilisation de bibliothèques Python comme **OpenCV**, **MediaPipe**, et **Scikit-Learn**, il est possible de créer des modèles de reconnaissance en temps réel des gestes de la main. Ce rapport décrit le processus de construction d'un modèle de détection de la langue des signes, qui inclut la collecte des données, le prétraitement, l'entraînement et l'évaluation du modèle.

2 Collecte des données

La collecte des données est l'une des étapes les plus importantes pour entraîner un modèle de reconnaissance des gestes de la langue des signes. Dans ce projet, nous avons utilisé la caméra de l'ordinateur pour capturer des vidéos des gestes de la main représentant des lettres de la langue des signes. Ces vidéos ont été ensuite transformées en images individuelles qui ont servi d'entrées pour notre modèle d'apprentissage automatique.

2.1 Configuration de l'environnement

Nous avons créé un répertoire pour stocker les images capturées à partir de la caméra. Le répertoire principal est appelé **./data**, et à l'intérieur de celui-ci, nous avons créé des sous-répertoires pour chaque classe (gestes de la langue des signes) que nous souhaitons détecter. Le code suivant permet de créer ces répertoires si nécessaire :

```
import os
import cv2

DATA_DIR = './data'
if not os.path.exists(DATA_DIR):
    os.makedirs(DATA_DIR)
```

2.2 Configuration de la caméra

Le code utilise OpenCV pour accéder à la caméra et capturer les vidéos. Le flux vidéo est récupéré en temps réel et les images sont sauvegardées une par une. Voici le code utilisé pour initialiser la caméra :

```
cap = cv2.VideoCapture(0)  # Index 0 pour la caméra par défaut
if not cap.isOpened():
    print("Error: Could not open camera.")
    exit()
```

Nous avons choisi d'utiliser l'index 0 pour la caméra par défaut. Si l'appareil n'est pas détecté, l'exécution du programme s'arrête avec un message d'erreur.

2.3 Collecte des données pour chaque classe

Nous avons collecté les données pour un nombre spécifique de classes, ici fixé à 3 (par exemple, A, B, L pour des lettres de l'alphabet en langue des signes). Le processus de collecte consiste à capturer des images qui montrent des gestes spécifiques, puis à les stocker dans un dossier correspondant à chaque classe.

Chaque classe a un répertoire dédié dans lequel les images capturées sont enregistrées avec un nom unique :

```
for j in range(number_of_classes):
    class_dir = os.path.join(DATA_DIR, str(j))
    if not os.path.exists(class_dir):
        os.makedirs(class_dir)
```

2.4 Processus de collecte des images

Le processus de collecte des données est réalisé en deux étapes :

- **Phase de préparation** : Afficher un message sur l'écran de la caméra pour informer l'utilisateur qu'il peut commencer à capturer les gestes. Une fois l'utilisateur prêt, la touche Q est utilisée pour passer à l'étape suivante.
- **Phase de capture des images** : Une fois l'utilisateur prêt, nous capturons un certain nombre d'images (défini par dataset-size, ici 100) pour chaque classe et les enregistrons dans le répertoire de la classe correspondante. Le code ci-dessous gère cette étape :

```
while counter < dataset_size:
    ret, frame = cap.read()
    if not ret:
        print("Error: Failed to capture frame")
        continue

    cv2.imwrite(os.path.join(class_dir, f'{counter}.jpg'), frame)
    counter += 1
```

Les images sont stockées dans les répertoires dédiés, avec des noms numérotés (par exemple, *0.jpg, 1.jpg*, etc.), afin de faciliter la gestion et l'organisation des données.

FIGURE 1 – classe 0

FIGURE 2 – classe 1

FIGURE 3 – classe 2

2.5 Libération des ressources

Une fois la collecte des données terminée pour toutes les classes, nous libérons la caméra et fermons les fenêtres ouvertes par **OpenCV** avec les commandes suivantes :


```
cap.release()
cv2.destroyAllWindows()
```

Les données collectées serviront à entraîner le modèle pour qu'il puisse reconnaître et classer les gestes de la main en temps réel. La qualité et la diversité des données sont cruciales pour la performance du modèle de détection des gestes de la langue des signes.

3 Traitement des données

Après avoir collecté les images représentant les gestes de la langue des signes, la prochaine étape consiste à prétraiter ces images afin d'extraire les caractéristiques (features) qui seront utilisées pour entraîner notre modèle de détection des gestes. Cette étape utilise **Mediapipe**, une bibliothèque de Google, pour détecter les repères des mains (landmarks) et extraire les coordonnées de ces points pour chaque image. Voici comment cette étape a été mise en œuvre.

3.1 Initialisation de Mediapipe pour la détection des mains

Pour détecter les repères des mains dans les images, nous avons utilisé **Mediapipe**, une bibliothèque puissante pour le traitement d'images en temps réel. Nous avons utilisé son modèle de détection de mains qui permet d'extraire les coordonnées des repères des mains, ce qui nous permet de capturer des informations significatives pour l'apprentissage.

L'initialisation du modèle Mediapipe se fait comme suit :

```
import mediapipe as mp

mp_hands = mp.solutions.hands
mp_drawing = mp.solutions.drawing_utils
mp_drawing_styles = mp.solutions.drawing_styles

hands = mp_hands.Hands(static_image_mode=True, min_detection_confidence=0.3)
```

Ici, nous avons défini une confiance minimale de détection à 0.3, ce qui permet de réduire les erreurs de détection dans les images où les mains sont moins visibles ou floues.

3.2 Lecture et traitement des images

Les images collectées lors de la phase précédente sont lues depuis les répertoires correspondants à chaque classe. Pour chaque image, nous appliquons plusieurs étapes de traitement pour extraire les coordonnées des repères des mains :

- **Chargement de l'image** : L'image est lue à l'aide d'OpenCV et convertie en format RGB, car Mediapipe fonctionne avec ce format.

- **Détection des repères des mains** : Mediapipe détecte les repères des mains dans chaque image. Si des mains sont détectées, les coordonnées des repères sont extraites.
- **Normalisation des coordonnées** : Les coordonnées des points de repère sont normalisées en soustrayant les valeurs minimales de chaque ensemble de coordonnées. Cela permet de rendre les coordonnées indépendantes de la taille de l'image et des différences de position de la main dans l'image.

Voici le code pour traiter les images et extraire les coordonnées des repères des mains :

```
import os
import cv2
import pickle

DATA_DIR = './data'
data = []
labels = []

# Parcours des répertoires contenant les images
for dir_ in os.listdir(DATA_DIR):
    dir_path = os.path.join(DATA_DIR, dir_)

    if os.path.isdir(dir_path):
        for img_path in os.listdir(dir_path):
            data_aux = []
            x_ = []
            y_ = []

            img = cv2.imread(os.path.join(dir_path, img_path))
            if img is None:
                continue

            img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
            results = hands.process(img_rgb)

            if results.multi_hand_landmarks:
                for hand_landmarks in results.multi_hand_landmarks:
                    for i in range(len(hand_landmarks.landmark)):
                        x = hand_landmarks.landmark[i].x
                        y = hand_landmarks.landmark[i].y
                        x_.append(x)
                        y_.append(y)

                    for i in range(len(hand_landmarks.landmark)):
                        x = hand_landmarks.landmark[i].x
```


```
                y = hand_landmarks.landmark[i].y
                data_aux.append(x - min(x_))
                data_aux.append(y - min(y_))

        data.append(data_aux)
        labels.append(dir_)
```

3.3 Sauvegarde des données traitées

Une fois que les données (les coordonnées des repères normalisées) sont extraites pour toutes les images, nous les enregistrons dans un fichier ***pickle***. Cela permet de les sauvegarder sous un format facilement réutilisable pour l'entraînement du modèle de machine learning.

Le code suivant permet de sauvegarder les données traitées :

```
with open('data.pickle', 'wb') as f:
    pickle.dump({'data': data, 'labels': labels}, f)
```

Cette étape de prétraitement est essentielle pour transformer les données brutes collectées (images) en un format exploitable par les algorithmes d'apprentissage automatique. La détection des repères des mains avec Mediapipe permet d'extraire des caractéristiques discrètes et robustes, qui seront utilisées pour la classification des gestes de la langue des signes.

4 Modèle de train

Une fois que les données ont été collectées et traitées, l'étape suivante consiste à entraîner un modèle de machine learning pour prédire les gestes de la langue des signes. Dans cette section, nous détaillons l'entraînement du modèle en utilisant l'algorithme **Random Forest Classifier** de *Scikit-Learn*.

4.1 Chargement et préparation des données

Les données traitées précédemment ont été enregistrées dans un fichier pickle. Ce fichier contient les **caractéristiques (features)** extraites des images ainsi que les **étiquettes (labels)** associées à chaque image. Nous avons commencé par charger ce fichier et organiser les données sous forme de tableaux compatibles avec le modèle d'apprentissage automatique.

```
import pickle
import numpy as np

with open('./data.pickle', 'rb') as f:
```


```
    data_dict = pickle.load(f)

data = np.asarray(data_dict['data'])
labels = np.asarray(data_dict['labels'])
```

Nous avons vérifié la cohérence des données en nous assurant que toutes les images avaient le même nombre de caractéristiques (features). Dans le cas contraire, nous avons effectué un **remplissage (padding)** ou une **coupe (trimming)** des vecteurs de caractéristiques pour uniformiser leur longueur.

```
# Vérification de la cohérence des données
max_len = max(len(sample) for sample in data)
data = np.array([np.pad(sample,
(0, max_len - len(sample)), mode='constant') for sample in data])
```

4.2 Séparation des données en ensembles d'entraînement et de test

Nous avons divisé l'ensemble des données en deux parties : un ensemble d'entraînement (80) et un ensemble de test (20). Cette division permet de former le modèle sur un sous-ensemble des données et de tester sa performance sur des données inconnues.

```
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test =
train_test_split(data, labels, test_size=0.2, shuffle=True, stratify=labels)
```

4.3 Entraînement du modèle

Nous avons choisi d'utiliser l'algorithme Random Forest pour la classification des gestes. Random Forest est un modèle robuste, adapté à des données de type tabulaire et offrant de bonnes performances dans de nombreux cas de classification.

Nous avons initialisé le modèle avec les paramètres suivants :

- **n-estimators=100** : nombre d'arbres dans la forêt,
- **max-depth=10** : profondeur maximale des arbres.

```
from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier(n_estimators=100, max_depth=10, random_state=42)
model.fit(x_train, y_train)
```

4.4 Évaluation du modèle

Une fois le modèle entraîné, nous avons effectué des prédictions sur l'ensemble de test. La performance du modèle a été évaluée à l'aide de la précision (accuracy), qui mesure la proportion de prédictions correctes sur l'ensemble des échantillons testés

```
from sklearn.metrics import accuracy_score

y_predict = model.predict(x_test)
score = accuracy_score(y_test, y_predict)
print(f'{score * 100:.2f}% des échantillons ont été correctement classés !')
```

Cette précision nous permet de savoir dans quelle mesure le modèle est capable de classer correctement les gestes de la langue des signes en fonction des données testées.

4.5 Sauvegarde du modèle

Une fois l'entraînement terminé, nous avons sauvegardé le modèle entraîné dans un fichier pickle afin de pouvoir l'utiliser ultérieurement pour effectuer des prédictions sans avoir à réentraîner le modèle à chaque fois.

```
with open('model.p', 'wb') as f:
    pickle.dump({'model': model}, f)
```

Cette étape a permis de transformer les données traitées en un modèle de machine learning capable de classer efficacement les gestes de la langue des signes. Le modèle Random Forest offre une solution robuste et performante pour ce type de problème de classification, et sa capacité à être facilement sauvegardé et réutilisé rend l'approche encore plus pratique.

5 Modèle de test

Une fois que le modèle a été entraîné et sauvegardé, il est essentiel de le tester sur de nouvelles données pour vérifier sa capacité à prédire les gestes de la langue des signes en temps réel. Dans cette section, nous allons expliquer comment utiliser le modèle entraîné pour effectuer des prédictions sur les gestes détectés via une caméra en direct.

5.1 Chargement du modèle entraîné

Le modèle précédemment entraîné a été sauvegardé dans un fichier pickle. Nous avons chargé ce modèle en utilisant *pickle*, ce qui nous permet de l'utiliser pour faire des prédictions sur des données nouvelles (issues de la caméra en temps réel).


```
import pickle

model_dict = pickle.load(open('./model.p', 'rb'))
model = model_dict['model']
```

5.2 Capture vidéo et détection des mains

L'étape suivante consiste à capturer des images en temps réel à partir de la caméra, puis à détecter les points de repère de la main dans chaque image. Pour ce faire, nous avons utilisé la bibliothèque **MediaPipe**, qui offre un détecteur de mains performant.

```
import cv2
import mediapipe as mp

cap = cv2.VideoCapture(0)  # Index de la caméra par défaut
```

Nous avons également initialisé le module **MediaPipe** Hands pour détecter et extraire les points de repère des mains à partir des images capturées.

```
mp_hands = mp.solutions.hands
hands = mp_hands.Hands(static_image_mode=True, min_detection_confidence=0.3)
```

5.3 Préparation des données pour la prédiction

Pour chaque image capturée, nous avons extrait les coordonnées des points de repère de la main, puis nous avons normalisé ces coordonnées pour garantir que les variations de taille de l'image ou de position de la main n'affectent pas la précision de la prédiction.

```
data_aux = []
x_ = []
y_ = []

# Traitement de l'image pour obtenir les points de repère
frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
results = hands.process(frame_rgb)
```

5.4 Prédiction du geste

Une fois les coordonnées des points de repère extraites et normalisées, nous avons utilisé le modèle préalablement chargé pour prédire l'étiquette du geste de la main. Cette étiquette correspond au geste de la langue des signes, que nous avons mappée à un caractère à l'aide d'un dictionnaire.


```
# Dictionnaire de mappage des labels prédits aux caractères
labels_dict = {0: 'A', 1: 'L', 2: '5'}

# Prédiction du geste
prediction = model.predict([np.asarray(data_aux)])
predicted_character = labels_dict[int(prediction[0])]
```

5.5 Affichage des résultats

Nous avons dessiné un **rectangle** autour de la main détectée et affiché le caractère prédit sur l'image. Ce rectangle permet de mettre en évidence la zone de la main où la prédiction a été effectuée.

```
# Dessiner le rectangle autour de la main
cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 0, 0), 4)
cv2.putText(frame, predicted_character,
(x1, y1 - 10), cv2.FONT_HERSHEY_SIMPLEX, 1.3, (0, 0, 0), 3, cv2.LINE_AA)
```

5.6 Boucle de test en temps réel

Nous avons continué à capturer des images en temps réel à partir de la caméra et avons effectué des prédictions sur chaque nouvelle image jusqu'à ce que l'utilisateur appuie sur la touche "Q" pour quitter le programme.

```
# Affichage de l'image avec la prédiction
cv2.imshow('frame', frame)

# Sortie lorsque l'utilisateur appuie sur "Q"
if cv2.waitKey(1) & 0xFF == ord('q'):
    break
```

5.7 Libération des ressources

Une fois le test terminé, nous avons libéré la caméra et fermé toutes les fenêtres OpenCV.

```
cap.release()
cv2.destroyAllWindows()
```

Cette étape a permis de valider le modèle en le testant sur des entrées en temps réel, garantissant ainsi son efficacité pour la reconnaissance des gestes de la langue des signes dans un contexte dynamique. La capacité à faire des prédictions en temps réel sur une vidéo en direct est essentielle pour l'application pratique du système de détection.

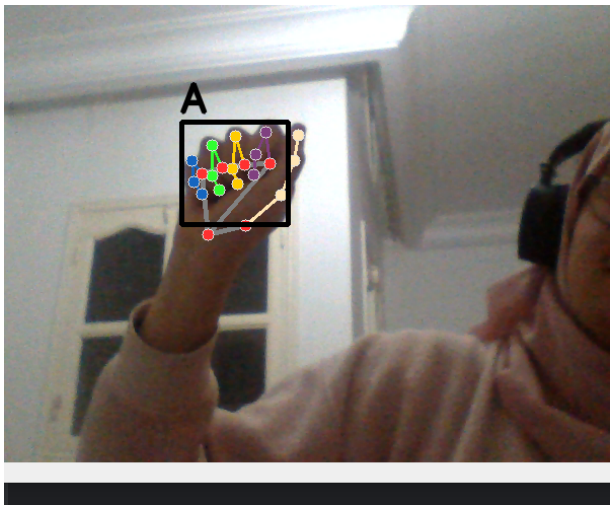

FIGURE 4 – classe 0 - A

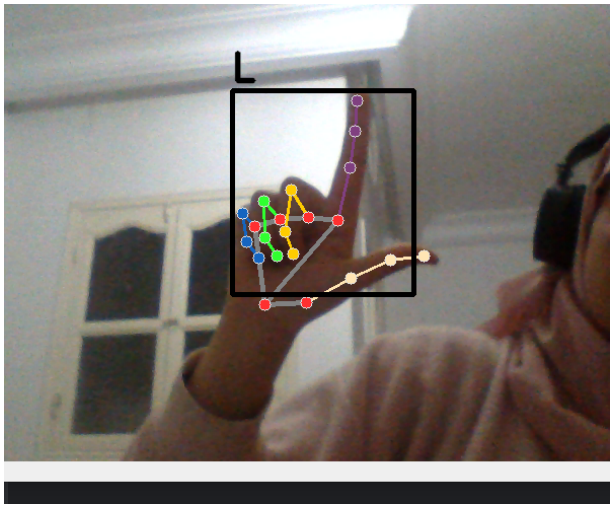

FIGURE 5 – classe 1 - L

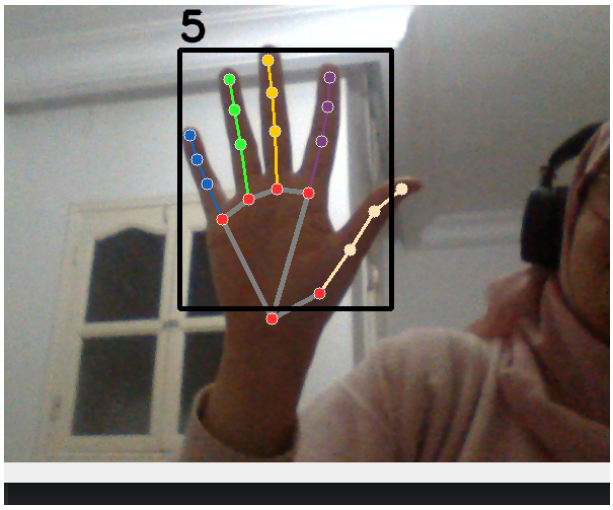

FIGURE 6 – classe 2 - 5

6 Conclusion

Ce projet a permis de construire un système de détection de la langue des signes basé sur la vision par ordinateur et l'apprentissage automatique. L'utilisation de **MediaPipe** pour l'extraction des points de repère des mains et de **Scikit-Learn** pour la construction d'un modèle de classification a montré que la détection des gestes de la main est faisable et efficace.

Quelques points clés du projet :

La détection de points de repère des mains est un préalable essentiel à la reconnaissance des gestes. Le **RandomForestClassifier** a bien performé pour la classification des lettres de la langue des signes. Le modèle peut être facilement étendu for the detection of other gestes or combinations of gestes.

gestes. De plus, l'intégration d'autres techniques de vision par ordinateur et de réseaux neuronaux profonds pourrait améliorer la précision et la robustesse du modèle, notamment pour des gestes plus complexes ou des conditions d'éclairage variables.

En conclusion, ce projet ouvre la voie à des applications pratiques dans le domaine de l'assistance aux personnes malentendantes, en offrant une interface plus fluide et plus accessible pour la communication en langue des signes.