# Document Technique

Idriss Kourbanhoussen (05)

21/10/2022

# Table des matières

# Context

Une entreprise spécialisée dans la conception de jouets, et notamment de petits robots souhaite réaliser une POC (Proof Of Concept), sur une base web simple (navigateur + webcam), de détection et reconnaissance d'objet.

# Projet

Le projet permettra à partir d'une webcam d'identifier les signes (la gestuelle) issues de *ASL Alphabet* (American Sign Language) dans un objectif d'apprentissage pour les enfants. Le dispositif sera mis sur une peluche et vise un publique d'enfants qui ont dans leur famille ou dans leur entourange une personne sourd.

# Diagramme de Grant

J'ai choisi de représenter visuellement l'état d'avancement des différentes tâches qui constituent un projet sous la forme d'un diagramme de Grant.
La colonne de gauche du diagramme énumère toutes les tâches à effectuer, tandis que la ligne d'en-tête représente les unités de temps les plus adaptées au projet (jours, semaines, mois etc.). Chaque tâche est matérialisée par une barre horizontale, dont la position et la longueur représentent la date de début, la durée et la date de fin

Ce diagramme se trouve dans le répertoire *ressources/Gant.PNG*

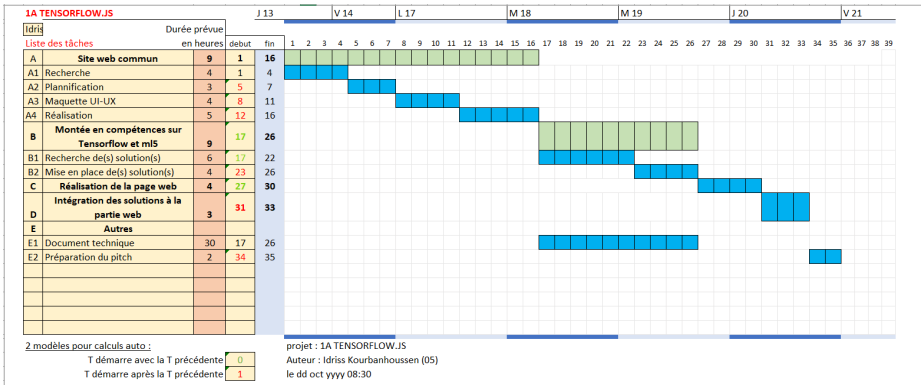


Figure 1: Diagramme de Gant

# Technologie employé

## p5*js

p5*js est une bibliothèque JavaScript. Cet considère la page de navigateur comme un croquis, y compris les objets HTML5 pour le texte, la saisie, la vidéo, la webcam et le son.

## ml5.js

ml5.js est l'apprentissage automatique pour le Web dans votre navigateur Web. Il est construit sur TensorFlow.js et utilise ces fonctionnalités au niveau du backend.

Ces bibliothèques sont importées en utilisant le CDN mais ont été téléchargé et stocké dans le répertoire *packages* pour une utilisation hors ligne.

## tensorflow.js

J'ai utilisé la commande ci-dessous me permettant d'obtenir à partir du model (stocké dans le répertoire *h5*) sauvegardé sous *keras*, le modèle exploitable en *ml5.js*. Il s'agit d'une commande executé sous *bash*.

```
tensorflowjs_converter --input_format keras \
                       model/model.h5 \
                       modelJS/
```

J'obtiens ainsi deux fichiers: - model.json (contient le graphe de flux de données et le manifeste en poids) - group1-shard1of1.bin (contient en binaire les poids)

La commande *tensorflowjs_converter* s'obtiens en installant *tensorflow.js* avec `pip install tensorflowjs` puis en indiquant lors de l'emploi de la commande le chemin du package *tensorflow*

Tous modèles exploitable par ml5.js doivent être sous la forme de ces deux fichiers.

# Architecture du projet

## Cas d'utilisation

L'application est hébergée sur GitHub et fait appel aux bibliothèques *ml5.js* et *p5.js* à partir du *CDN*. Le modèle est quant à lui appelé depuis teachable machine

## Répertoire

L'arbre du projet se trouve dans le répertoire *ressources/architecture/tree.txt*

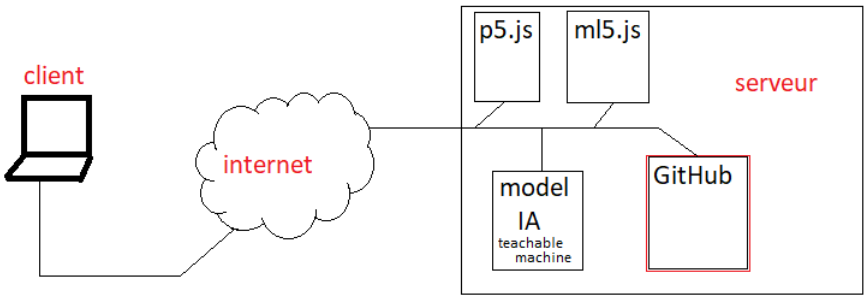


Figure 2: cas d'utilisation

#### sign_language.html

Le fichier *sign_language.html* est à la racine du projet et contient tous ce qui apparaît sur l'application web. Il fait appel à : - P5.js (à partir du CDN) - ml5.js (à partir du CDN) - css/sign_language.css - css/button.css - js/sign-language.js

#### CSS

Le dossier *css* contient la mise en forme de l'application Le fichier *button.css* met en forme uniquement les boutons

#### js

Ce dossier stocke les scrips permettant d'intéragir avec l'application web.

#### Modèles

Le répertoire modèle contient trois modèle de reconnaissance de signe dont seul le modèle *model0* est importé via un URL (ce modèle est stocké dans le répertoire model0 pour une futur utilisation hors ligne).

# Code

## sign-language.js

Le fichier *sign-language.js* sert à activer la webcam et à faire appel au modèle d'IA permettant la reconnaissance des signes

#### Variables globaux

- tauxR (int): Tau de rafraichissement.
- imageModelURL (str): Url du modèle hébergé sur teachablemachine.withgoogle.com.

- vidéo: Elément HTML5 qui contient le flux audio/vidéo d'une webcam.
- flippedVideo: Image retourné (pour mieux être traité par le modèle).
- classifier: Modèle.
- res: Résultats de la prédiction.
- label (str): Nom du signe prédit (issue de la prédiction).
- alphabet (list): Lettre de l'alphabet
- signe (str): Lettre (signe) a effectuer désigné par le programme 1 ou 2
- start (bool): Permet d'activer ou non la fonction permettant la vérification du signe (`SignIsCorrect()` dans la fonction `draw()`)
- index (int): Permet de parcourir l'alphabet
- fin (int): Indique à quel index doit s'arrêter le parcours;

- tagSign: point vers l'élément html qui permet d'afficher le signe a effectué.

**preload()**

La fonction preload() (l34) est utilisée pour gérer le chargement asynchrone de fichiers externes de manière bloquante. Si une fonction de préchargement est définie, *setup()* attendra que tous les appels de chargement soient terminés.

On charge le modèle de machine learning (l35) qui permettra d'analyser les images et d'en retirer une prédiction.

**setup()**

La fonction setup() (l39) est appelée une fois au démarrage du programme. Il est utilisé pour définir les propriétés initiales de l'environnement.

Je fixe la fréquence d'images (avec la fonction `frameRate`) à 0.8 (l40) pour ne pas surcharger le CPU. Je mets en place un cadre où sera mis en place la caméra (l40-48).
La fonction ml5.flipImage() (l49) pour retourner l'entrée vidéo horizontalement et renvoie l'image retournée. Ce qui permettra au modèle de mieux prédire le signe.

**draw()**

La fonction draw() (l55) est appelée directement après `setup()`, la fonction `draw()` exécute en continu les lignes de code contenues dans son bloc jusqu'à l'arrêt du programme

J'appelle le modèle avec la fonction `classifyVideo()` pour prédire le signe capter par la caméra (l66). Puis je teste si le signe est correcte ou non (l67-69) avec la fonction `SignIsCorrect()`

**classifyVideo()**

La fonction (l73) permet de prédire l'image capturé par la caméra.

**gotResult()**

Assigne le resultat obtenu par le modèle effectué par la fonction `classifyVideo()` aux variables globaux.

**say()**

Permet d'utiliser la synthèse vocale (l95)

**programme 1, 2()**

Lance une séquence de signe qui doivent être appris (l107-118).

**doSign()**

Affecte aux variables globaux: - signe: le signe qui doit être effectué - tagSign: affecte à la balise représentant l'élément signe de la page web le signe qui doit être fait. - start: true, pour lancer la fonction *SignIsCorrect()* - déclenche le synthétiseur vocal pour annoncer le signe qui doit être fait.

**SignIsCorrect()**

Teste si le signe capturé par la caméra est correcte et s'il a une performance au-delà de 0.75, s'il l'est, on passe au signe suivant. Si la variable *index* est supérieur à la variable *fin* alors cela signifie que l'utilisateur est arrivé à la fin du programme et à donc terminé.

**launchTP()**

Permet de lancer le programme correspondant aux boutons appuyés.

# Modèle

J'ai récupéré plusieurs modèles de reconnaissance de l'alphabet du langage des signes (stocké dans le répertoire *model*). Je n'ai qu'incorporé qu'un seul modèle et importé depuis une url. Le modèle est pré-entrainé à partir de l'application web *Teachable Machine* mais n'est pas très performant.
Les autres modèles sont aussi pré-entrainés mais ont été realisés à partir d'un data set de 87000 images donc plus performant . Le modèle contenu dans le répertoire *model/model2* n'a pas pu être utilisé dans l'immediat puisqu'il attendait un entrée d'un tableau de dimension bien spécifique.

Erreur généré par le modèle 2 (*model2*):

```
model 2: expected conv2d_3_input to have shape [null,64,64,3] but got array with shape [1,22
```

Les modèles sont stockés dans le répertoire *model* pour une futur utilisation hors ligne. Les modèles se présentent sous la forme: - d'un fichier *model.json*

contenant le graphe de flux de données et le manifeste en poids - et d'un ou plusieur fichiers *.bin* contenant en binaire les poids

## Réalisation et à l'avenir

L'application est fonctionnel et disponible depuis l'url https://devia05.github.io/demo/ mais le modèle pré-entrainé récupéré n'est pas très performant. C'est pourquoi j'ai prévu d'incorporé d'autre modèles en plus pour faire une moyenne des prédictions.

## Ressources

Le modèle model2 est obtenu à partir du travaille de Naman Manchanda.

Les données qui ont permis d'avoir ce modèle proviennent du projet ASL Alphabet d'Akash Nagaraj

Documentation de ML5.js

Documentation de P5.js

image issue de https://www.freepik.com/author/catalyststuff