
# iClicker

PROJET NUMÉRO 73

---

Étudiant : Sergiu Iacob
Encadrant : Marius Bilasco

mars, 2020

# Table des matières

# CHAPITRE 1

# INTRODUCTION

## 1.1 LE CONTEXTE

Au moment de travailler sur ce projet, j'ai passé 5 semestres comme étudiant à la Faculté d'Informatique, Iași, et le dernier (en cours) comme étudiant Erasmus, à l'Université de Lille. Comme le destin l'a voulu, l'épidémie de COVID-19 nous force actuellement à nous isoler pendant au moins deux semaines, alors quelle meilleure occasion de passer du temps de qualité à travailler sur ce projet ?

Il s'agit d'un projet à double objectif. En tant qu'étudiant Erasmus, j'aurai développé ce projet pour le cours "Projet Individuel", ayant un soutenance finale à la fin du mois de Mai, 2020. Après avoir terminé le programme Erasmus, le projet aura servi de thèse pour l'obtention d'une Licence, à la Faculté d'Informatique, Iași.

## 1.2 L'IDÉE

L'idée de base est d'avoir une logiciel qui suivra le visage de l'utilisateur, avec le but de déplacer la position du curseur en temps réel. Pour cela, nous utiliserons une webcam pour obtenir des images. Avec ceux, nous essaierons de calculer l'endroit où l'utilisateur regarde et de déplacer le curseur de la souris en conséquence.

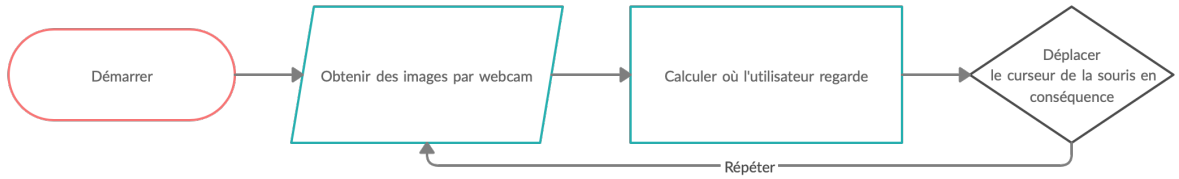


**FIGURE 1.1**
Procédure générale

## 1.3 MOTIVATION

Pour décider le projet sur lequel je voulais travailler, j'ai pris en compte deux facteurs clés : ma future carrière professionnelle et la facilité d'utilisation du projet. Je voulais travailler sur un projet qui alimenterait mon intérêt pour l'Intelligence Artificielle et qui me donnerait la possibilité d'appliquer les recherches que je ferai dans ce domaine. Aussi, je voulais avoir une approche pratique de ce projet et faire quelque chose qui soit utile aux gens.

Si l'on regarde en arrière, on constate que les gens ont toujours travaillé à rendre la technologie accessible au plus grand nombre. Qu'il s'agisse du système braille, de l'aide aux aveugles pour lire et écrire, des lecteurs d'écran ou des assistants intelligents (Siri, Bixby, etc.), l'accessibilité est un facteur clé pour faire en sorte que les gens bénéficient des avantages de la technologie. Par conséquent, je serais rassuré de savoir que j'ai moi aussi travaillé sur un projet qui pourrait potentiellement aider les personnes handicapées – et pas seulement elles – à utiliser l'ordinateur sans l'aide des mains.

Quant à l'Intelligence Artificielle, il est inutile de souligner son importance contemporaine. De l'applicabilité médicale, de la conduite et du pilotage autonomes et l'agriculture intelligente aux réfrigérateurs intelligents qui vous disent quand vous êtes à court de lait, l'Intelligence Artificielle est très répandue et sa croissance ne va pas s'arrêter de sitôt. Pour moi, c'est une raison supplémentaire de l'étudier et de mieux la comprendre, car elle est intégrée dans notre vie quotidienne.

## 1.4 LE BUT

### 1.4.1 OBJECTIFS ESSENTIELS

L'un des premiers objectifs que j'essaierai d'atteindre est de savoir approximativement où l'utilisateur regarde en utilisant uniquement ses yeux. Par exemple, un bon début serait de savoir sur quelle partie de l'écran l'utilisateur se concentre. En se basant sur la grille ci-dessous, si l'utilisateur regarde le carré numéro $i$, alors nous déplacerons le curseur de la souris au centre de ce carré.

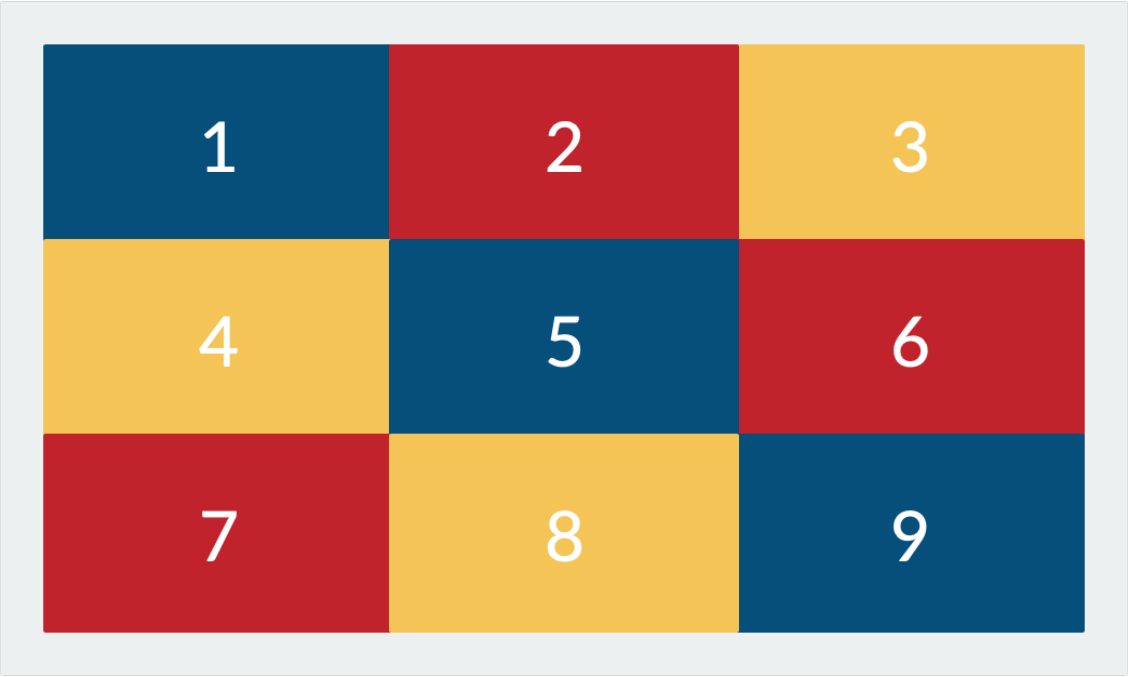


**FIGURE 1.2**
3x3 grille

Deuxièmement, je vais essayer de simuler les fonctions de clic comme suit : si l'utilisateur ferme son œil gauche pendant un certain temps, nous interpréterons cela comme un clic gauche. La même chose sera faite pour le clic droit, en utilisant l'œil droit.

A titre d'orientation générale, je vais essayer de développer l'application afin qu'elle soit facilement maintenable. Je me concentrerai également sur les concepts que j'ai appris tout au long de mes études

universitaires : “clean code”, principes OOP, principes SOLID, cohésion, etc. Une bonne condition serait également de rendre l’application multiplateforme.

### 1.4.2 OBJECTIFS PRÉFÉRABLES

Pour aller un peu plus loin, si la prédiction des carrés se passe bien, on peut essayer de prédire l’emplacement exact du curseur. Cela signifie que nous déplacerons le curseur de la souris exactement là où nous pensons que l’utilisateur regardera.

Avoir plus d’informations sur l’utilisateur pourrait nous aider dans nos calculs. Pour cela, je vais aussi essayer de voir comment je peux utiliser d’autres parties du visage pour améliorer les calculs : par exemple, la position du nez. L’image ci-dessous représente quelques points de repère du visage que nous utiliserons pour nos calculs : les points $[37, 48]$ sont pour les yeux et $[28, 36]$ pour le nez.

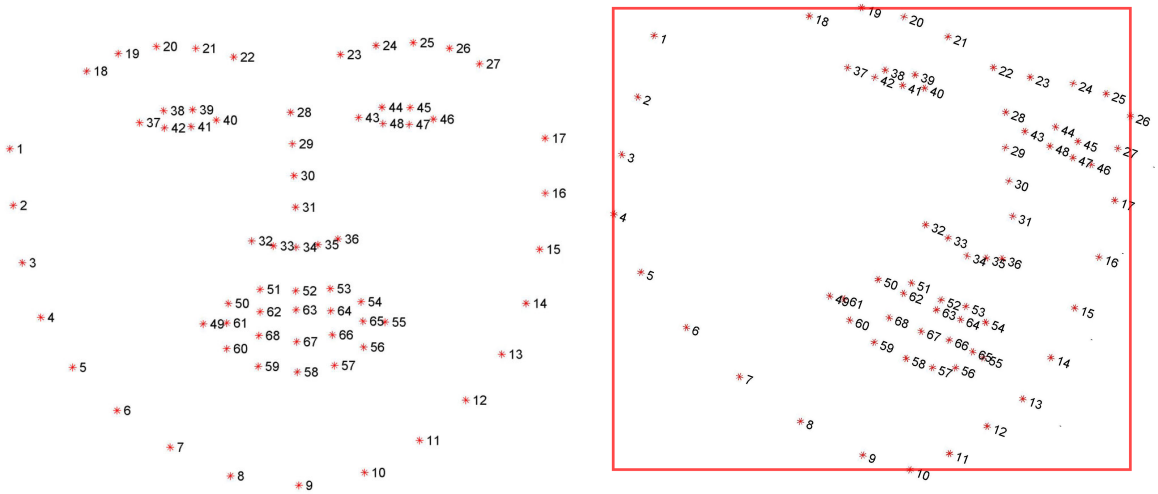

**FIGURE 1.3**
Repères faciaux
Source : ibug

# CHAPITRE 2

# PLAN DE DÉVELOPPEMENT

## 2.1 COMMENT RÉSOUDRE LE PROBLÈME

### 2.1.1 DÉFINIR LE PROBLÈME

Le problème que nous essayons de résoudre est de suivre précisément les yeux pour pouvoir déplacer le curseur de la souris. Cela fait partie d'une gamme plus large de problèmes de *Computer Vision*, et plus précisément de *eye tracking*. Selon WIKIPEDIA 2020, le suivi des yeux est "le processus de mesure du point de vue (où l'on regarde) ou du mouvement d'un œil par rapport à la tête".

Sachant cela, nous allons nous attaquer à ce problème en utilisant *l'Intelligence Artificielle*, car c'est devenu aujourd'hui un moyen viable de résoudre ces problèmes. Nous allons donc traiter un problème *d'Apprentissage Automatique*, plus précisément un problème *d'Apprentissage profond*.

Les tâches d'Apprentissage Profond peuvent être supervisées, semi-supervisées ou non supervisées. Nous nous concentrerons sur les tâches *supervisées* et nous traiterons notre problème comme l'une d'entre elles. Cela signifie que nous allons d'abord acquérir quelques données d'apprentissage, qui consisteront en des images de webcam marquées par la position du curseur de la souris. De cette façon, notre algorithme apprendra où le curseur de la souris doit se trouver, en fonction de l'endroit où l'utilisateur regardait lorsque l'image a été prise.

### 2.1.2 LE NEURONE ARTIFICIEL

Avant d'entrer dans des architectures d'Apprentissage Profond plus sophistiquées, nous devons d'abord examiner le neurone artificiel. En bref, il s'agit d'un modèle formel et simplifié de neurone biologique. Il peut nous aider à faire la *classification binaire* à l'aide de la formule suivante :

$$y = \varphi(w * x + b)$$

Dans la formule ci-dessus, $x$ est un vecteur de valeurs réelles, représentant notre entrée, $w$ est un vecteur de valeurs réelles, appelé *poids* et $b$ est le *biais*. Le produit $w * x$ représente le produit scalaire et est égal à $w * x = \sum_{i=1}^{n} w_i x_i$, $n$ étant la longueur de notre entrée.

Le résultat de la classification est donné par $\varphi$, qui est appelé une *fonction d'activation*. Si cette fonction sert de seuil, elle effectuera une classification binaire, donnant soit 0 soit 1. Nous pourrions même utiliser une fonction d'activation telle que $\varphi(x) = x$ qui ne nous donnera pas une classification binaire, mais pourrait nous aider à résoudre des problèmes de régression linéaire.

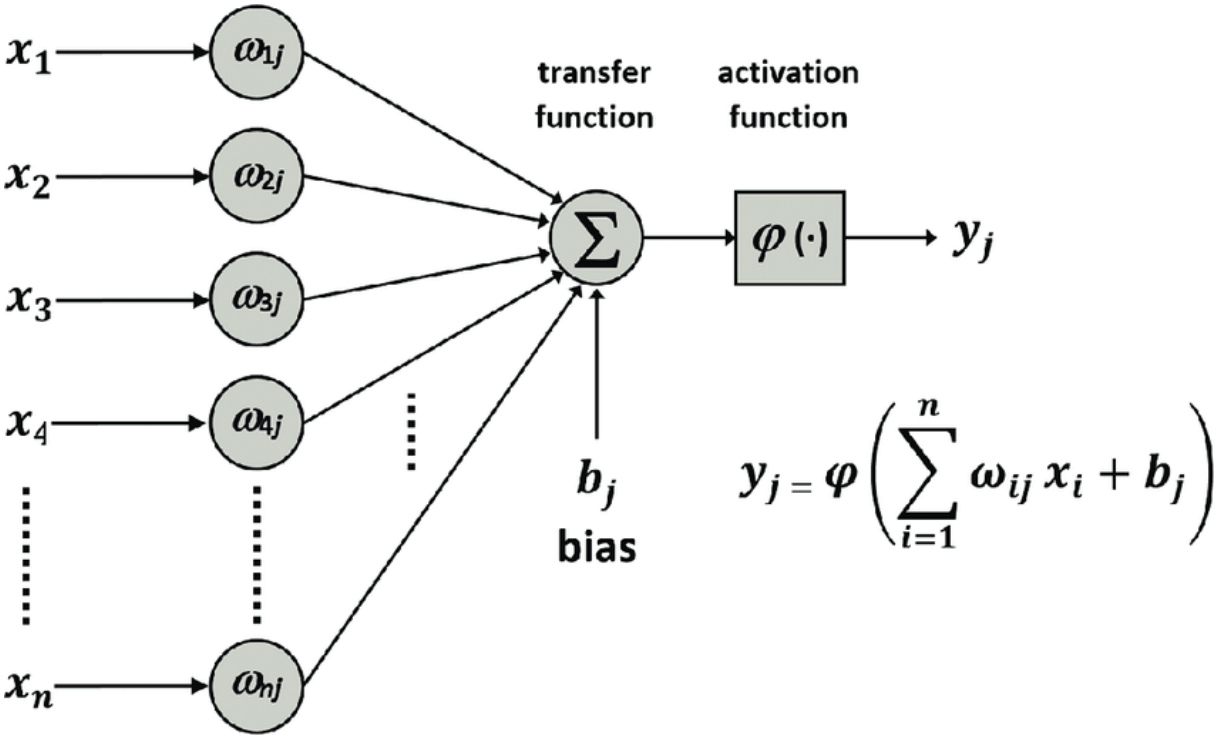


**FIGURE 2.1**
Schéma d'un neurone artificiel
Source : ResearchGate

### 2.1.3 LES RÉSEAUX DE NEURONES

Le "cheval de bataille" des problèmes de vision par ordinateur est le réseau de neurones artificiels. Celui-ci est, comme son nom l'indique, composé de plusieurs neurones artificiels, répartis sur plusieurs couches.

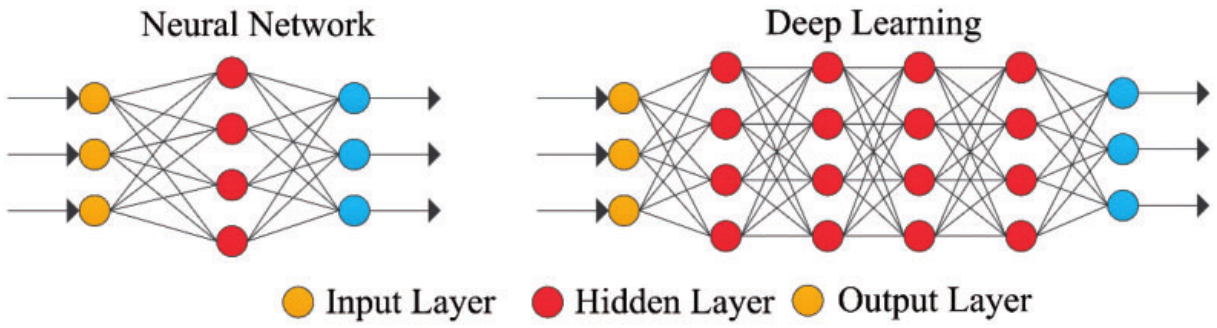


**FIGURE 2.2**
Schéma d'un neurone artificiel
Source : ResearchGate

## 2.2 STRATEGIE

### 2.2.1 OBTENIR DES DONNÉES

Il est bien connu qu'un algorithme d'Apprentissage Automatique n'est bon que si les données que nous lui fournissons le sont aussi. C'est incroyablement important, c'est pourquoi je vais m'efforcer de rassembler autant de *données brutes* que possible, ainsi que d'autres informations que je pourrais trouver importantes.

Un autre point essentiel est *traitement des données* et *analyse des données*. Le premier concerne le nettoyage de l'ensemble des données des instances inutiles et la mise en forme des données recueillies dans un format utilisable pour nos réseaux neuronaux. Nous examinerons également comment nous pouvons obtenir des informations supplémentaires qui pourraient être utiles à partir de nos images.

### 2.2.2 PERCEPTRONS MULTICOUCHES

Un type particulier de réseaux neuronaux est le réseau multicouche Perceptron. Ce sera ma première expérience et le point de départ pour tenter de résoudre le problème.

En utilisant ceci, je vais essayer de résoudre le premier objectif 1.4.1, et c'est approximativement de prédire où l'utilisateur regarde. Je vais d'abord essayer avec une grille plus petite de 2x2, puis je passerai à une grille de 3x3.

### 2.2.3 RÉSEAU NEURONES CONVOLUTIFS

Une étape supplémentaire consistera à utiliser les réseaux neuronaux convolutifs, une architecture de pointe pour travailler avec des images. Cela pourrait permettre d'atteindre un objectif plus préférable, à savoir prévoir exactement l'emplacement du curseur de la souris en fonction de l'endroit où l'utilisateur regarde.

### 2.2.4 RECHERCHE D'AUTRES MÉTHODES

Après avoir expérimenté les méthodes ci-dessus, je vais rechercher de meilleures façons de résoudre ce problème. Cela pourrait signifier un meilleur traitement des données, un meilleur réglage des réseaux de neurones, etc.

# CHAPITRE 3

# INFORMATIONS PRÉLIMINAIRES

## 3.1 MANIÈRE DE TRAVAILLER

Pour développer le projet, j'utiliserai tout d'abord GitHub pour structurer et sauvegarder le code. Vous pouvez trouver le dépôt du projet ici : https://github.com/sergiuiacob1/iClicker/

## 3.2 CONFIGURATION UTILISÉE

Pour chacun des résultats présentés ici, il est important de savoir que les expériences ont été réalisées sur la machine suivante : "MacBook Pro A1502, début 2015". CPU : Core i5 (I5-5287U), 2.9GHz. RAM : 16 GB, LPDDR3 à 1866 MHz. Ce portable n'a pas de GPU, donc les modèles d'entraînement prendront plus de temps.

La caméra de l'ordinateur portable n'est pas géniale, mais elle est capable de prendre des images HD, avec une résolution allant jusqu'à 1280x720. Cependant, elle ne fonctionne pas bien dans de mauvaises conditions d'éclairage, je vais donc essayer d'obtenir les données dans de bonnes conditions d'éclairage.

## 3.3 INFORMATIONS TECHNIQUES

L'un des langages de programmation les plus populaires pour les problèmes d'Apprentissage Profond est Python. J'ai donc choisi de développer ce projet en Python, car le soutien de la communauté et les ressources trouvées en ligne sont vastes. Voici une liste des technologies que j'ai utilisées :

| Technology name | Version | Useful links |
|---|---|---|
| Python3 | 3.7 | `https://www.python.org` |
| Conda | 4.8.0 | `https://conda.io/` |

Je dois également noter quelques bibliothèques Python importantes qui ont des fonctionnalités cruciales pour ce projet. L'une d'entre elles est OpenCV, que j'utiliserai pour la détection des visages et la capture d'images. PyQt5 a été utilisé pour construire une interface graphique pour l'application, afin qu'elle soit plus conviviale.

| Library name | Version | Useful links |
|---|---|---|
| Keras | 2.2.4 | `https://keras.io` |
| PyTorch | 1.4 | `https://pytorch.org` |
| OpenCV | 4.1.2 | `https://opencv.org` |
| PyQt5 | 5.14 | `https://pypi.org/project/PyQt5/` |
| dlib | 19.19.0 | `https://pypi.org/project/dlib/` |
| imutils | 0.5.3 | `https://pypi.org/project/imutils/` |

## 3.4 LIMITES

Au moment où nous écrivons ces lignes, les limites actuelles sont les suivantes : l'application ne fonctionne qu'avec un moniteur et une caméra. En outre, je n'ai pas encore étudié comment cela devrait fonctionner si l'utilisateur porte des lunettes, je suppose donc qu'il n'en porte pas. Les résultats présentés sont basés sur les données que j'ai recueillies, c'est-à-dire des images de moi-même regardant le moniteur. De plus, il n'a été testé que sur MacOS, mais il devrait fonctionner sans problème sous Linux également.

# CHAPITRE 4

# LA COLLECTE DE DONNÉES

## 4.1 DONNÉES NÉCESSAIRES

Comme il a été mentionné précédemment, les données sont extrêmement importantes. Nous avons besoin qu'elles soient variées, dans des scénarios différents, c'est-à-dire des arrière-plans différents, des poses de tête différentes, etc.

Nous travaillerons avec des images de la webcam et nous nous concentrerons uniquement sur le visage de l'utilisateur. Nous en extrairons les yeux et nous travaillerons avec ceux de la première phase. Ensuite, nous étudierons comment utiliser davantage d'informations, comme la pose de la tête, etc. Pour chaque expérience, nous mentionnerons les données que nous utiliserons et la manière dont elles ont été traitées.

## 4.2 ACQUISITION DE DONNÉES

### 4.2.1 MOYENS D'ACQUÉRIR DES DONNÉES

Pour la collecte des données, j'ai mis en place deux méthodes : une méthode "active" et une méthode "de fond". Pour la méthode active, l'utilisateur doit suivre des yeux le curseur lorsqu'il se déplace sur l'écran. Chaque fois que le curseur se déplace, une image est capturée de l'utilisateur qui regarde le curseur.

La méthode "de fond' signifie que l'utilisateur peut utiliser l'ordinateur comme il le souhaite, mais les données seront recueillies pendant ce temps. Chaque fois qu'il cliquera quelque part, une image sera capturée et enregistrée avec la position de la souris.

```
def start_collecting(self, collection_type="background"):
    print(f'Start collecting data in {collection_type} mode')
    WebcamCapturer.start_capturing()
    self.gui.start()
    print('DataCollectorGUI started')
    if collection_type == "background":
        self.mouse_listener.start_listening()
        print('Mouse listener started')
    else:
        threading.Thread(target=self.start_active_collection).start()
```

**LISTING 4.1**
Collecte de données

### 4.2.2 SAUVEGARDE DES DONNÉES

Lorsque la collecte des données est terminée, je les enregistre sous forme de "session". Chaque session est définie par le nombre d'images qui ont été capturées, la résolution de l'écran et la résolution de la

webcam. Vous pouvez voir ci-dessous la structure des dossiers et vous pouvez constater que les images ont été sauvegardées sans les modifier.

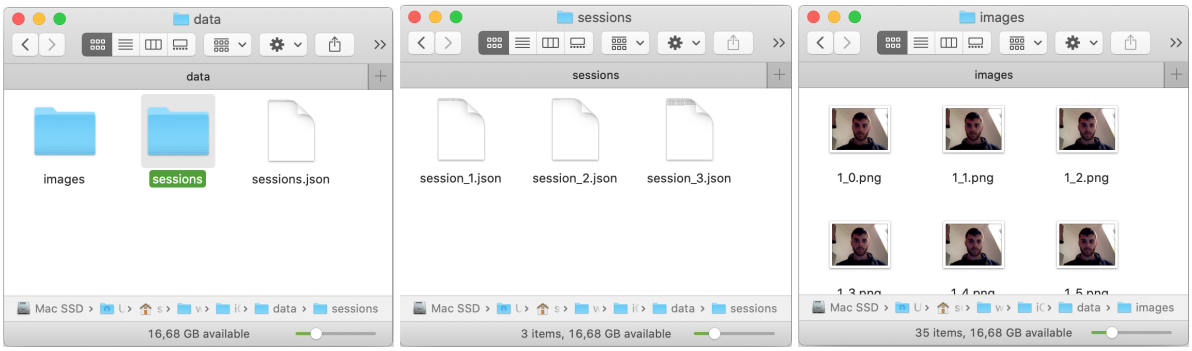


**FIGURE 4.1**
How data is structured

```python
def save_collected_data(self):
    if len(self.collected_data) == 0:
        return
    self.collect_data_lock.acquire()
    session_no = self.get_session_number()
    print(f"Saving data for session_{session_no}")
    self.save_session_info(session_no)
    self.save_images_info(session_no)
    self.save_images(session_no)
    self.collect_data_lock.release()
    print('Saving data done')
```

**LISTING 4.2**
Sauvegarde des données

# CHAPITRE 5

# TRAITEMENT DE L'INFORMATION

## 5.1 AVANTAGES DU TRAITEMENT DES DONNÉES

Le traitement des données est important pour de multiples raisons. Premièrement, il est important d'éliminer le bruit ou tout type d'information non essentielle des données. Deuxièmement, nous pouvons examiner comment nous pouvons tirer davantage d'informations de nos "données brutes".

## 5.2 TRAVAILLER UNIQUEMENT AVEC LES YEUX

Dans certaines des expériences suivantes, je me suis concentré sur l'extraction des yeux d'une image. Pour cela, j'ai utilisé deux bibliothèques : dlib et imutils. Sur la base des repères faciaux présentés dans l'introduction 1.3, j'ai extrait uniquement la partie de l'image décrite par les repères faciaux pour les yeux.

```
class FaceDetector(metaclass=Singleton):
    def __init__(self):
        self._face_detector = dlib.get_frontal_face_detector()
        self._face_predictor = dlib.shape_predictor(
            Config.face_landmarks_path)

    def get_eye_contours(self, cv2_image):
        """Returns a list of eye contours from a cv2_image. First contour is for the
     left eye"""
        contours = []
        gray_image = Utils.convert_to_gray_image(cv2_image)
        rects = self._face_detector(gray_image, 0)
        if len(rects) > 0:
            # only for the first face found
            shape = self._face_predictor(gray_image, rects[0])
            shape = face_utils.shape_to_np(shape)
            (left_eye_start,
             left_eye_end) = face_utils.FACIAL_LANDMARKS_IDXS["left_eye"]
            (right_eye_start,
             right_eye_end) = face_utils.FACIAL_LANDMARKS_IDXS["right_eye"]
            contours.append(shape[left_eye_start:left_eye_end])
            contours.append(shape[right_eye_start:right_eye_end])
        return contours
```

**LISTING 5.1**
Extraire le contour des yeux

Comme nous ne nous intéressons qu'au contraste entre l'iris et la pupille, j'ai ensuite appliqué un *seuil binaire* sur l'image grise de l'œil pour souligner la façon dont la pupille est située par rapport à l'ensemble de l'œil.

```
def get_binary_thresholded_image(cv2_image):
    img = convert_to_gray_image(cv2_image)
    img = cv2.medianBlur(img, 5)
    img = cv2.adaptiveThreshold(
        img, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 11, 2)
    return img
```

**LISTING 5.2**
Application d'un seuil binaire

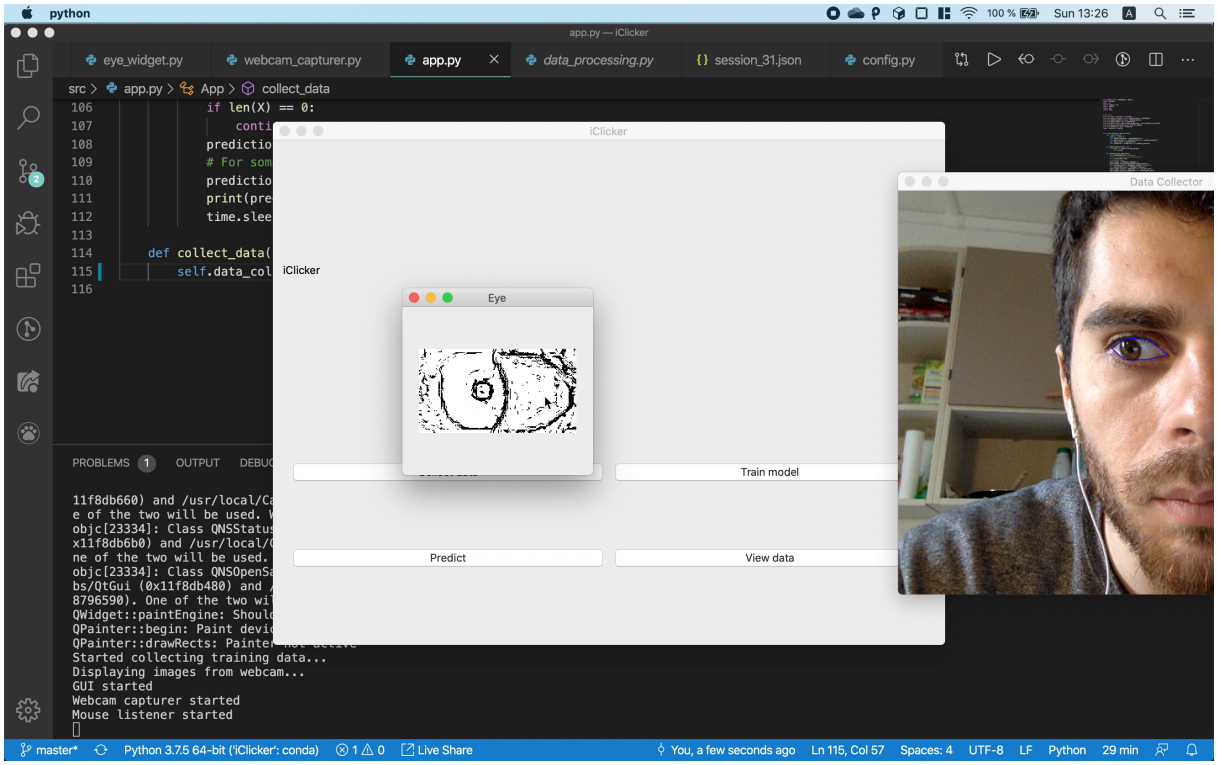


**FIGURE 5.1**
Obtenir des données sur les yeux

## 5.3 EN UTILISANT SEULEMENT LE VISAGE

Un travail en cours porte sur la façon dont je peux utiliser le visage entier comme intrant pour un Réseau Neuronal Convolutif et sur la question de savoir si je dois y appliquer un traitement quelconque. Je reviendrai lorsque j'aurai des résultats à ce sujet, car il s'agit d'un travail en cours.

# CHAPITRE 6

# EXPÉRIMENTATIONS

## 6.1 PERCEPTRONS MULTICOUCHES

### 6.1.1 DONNÉES UTILISÉES

Pour cette expérience, j'ai utilisé les données résultant du traitement des images originales et de l'extraction des seuls yeux 5.2. Le nombre de cas était $\approx 500$, les résultats pourraient donc ne pas être très précis. Je vais relancer cette expérience avec plus de données.

### 6.1.2 PARAMÈTRES D'ENTRAÎNEMENT

J'ai entraîné ce modèle en utilisant la configuration ci-dessous, pour environ 1000 époques. J'ai essayé plusieurs optimiseurs, tels que Adam, Adagrad et SGD, mais RMSprop semblait faire les choses très bien.

```
print('Loading train data...')
X, y = get_data()
X = np.array(X)
y = np.array(y)

print('Training neural network...')
model = Sequential([
    Dense(100, input_shape=(len(X[0]),),
            kernel_initializer='glorot_uniform'),
    Dropout(0.5),
    ReLU(),
    Dense(16, kernel_initializer='glorot_uniform'),
    ReLU(),
    Dense(4, activation='softmax')
])
rmsprop = RMSprop(lr=0.001)
model.compile(optimizer='adagrad',
                loss='categorical_crossentropy', metrics=['accuracy'])
start_time = time.time()
fit_history = model.fit(X, y, epochs=train_parameters["epochs"], verbose=1)
end_time = time.time()
print('Training done')
```

**LISTING 6.1**
Formation du MLP

### 6.1.3 RÉSULTATS

Les tests effectués sur les mêmes données de formation ont donné une précision de plus de $95\%$. Je vais réexaminer le score des mesures sur des données de test distinctes, après en avoir acquis d'autres. Cependant, j'ai essayé de prédire où l'utilisateur regarde sur une grille de 2x2 et cela a donné de bons résultats dans l'ensemble. Bien que ce ne soit pas parfait, il a fait le travail.

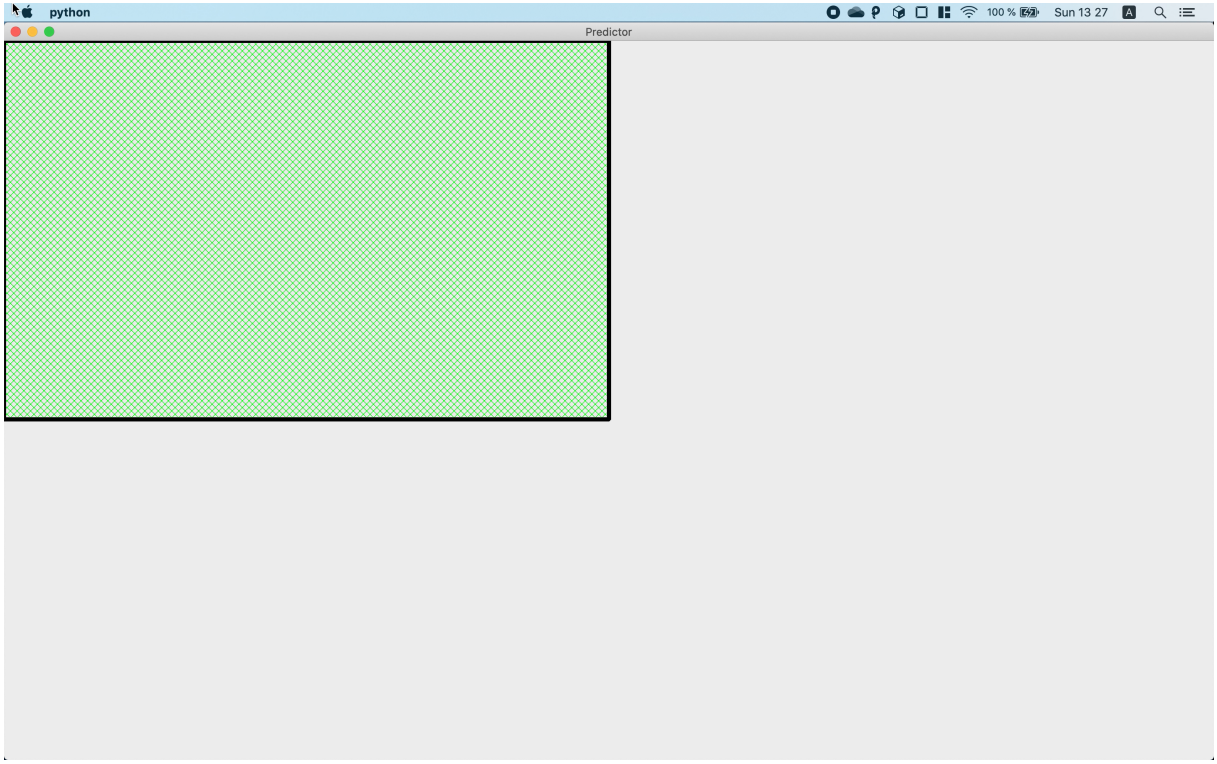


**FIGURE 6.1**
Prévision sur une grille 2x2 avec MLP
En fait, je vraiment regardais dans la partie supérieure gauche de l'écran, croyez-moi !

## 6.2 RÉSEAUX NEURONAUX CONVOLUTIFS

Il s'agit actuellement d'un travail en cours.

# BIBLIOGRAPHIE

WIKIPEDIA (2020). *Eye Tracking*. Wikimedia Foundation. URL : https://en.wikipedia.org/wiki/Eye_tracking (visité le 15 mar. 2020).