



ICLICKER

PROJET NUMÉRO 73

Étudiant : Sergiu Iacob
Encadrant : Marius Bilasco

avril, 2020

Table des matières

1	Introduction	2
1.1	Le contexte	2
1.2	L'idée	2
1.3	Motivation	2
1.4	Le but	3
1.4.1	Objectifs essentiels	3
1.4.2	Objectifs préférables	3
2	Plan de développement	5
2.1	Comment résoudre le problème	5
2.1.1	Définir le problème	5
2.1.2	Le neurone artificiel	5
2.1.3	Les réseaux de neurones	6
2.2	Stratégie	6
2.2.1	Obtenir des données	6
2.2.2	Perceptrons multicouches	7
2.2.3	Réseau neurones convolutifs	7
3	Informations préliminaires	8
3.1	Manière de travailler	8
3.2	Configuration utilisée	8
3.3	Informations techniques	8
3.4	Limites	9
4	La collecte de données	10
4.1	Données nécessaires	10
4.2	Acquisition de données	10
4.2.1	Moyens d'acquérir des données	10
4.2.2	Sauvegarde des données	10
5	Traitement de l'information	12
5.1	Avantages du traitement des données	12
5.2	Travailler uniquement avec les yeux	12
5.3	En utilisant seulement le visage	13
5.4	“Bandea oculaire”	14
6	Expérimentations	16
6.1	Perceptrons multicouches	16
6.2	Réseaux neuronaux convolutifs	18
6.2.1	Utilisation des visages extraits comme données	19
6.2.2	Utilisation de “bandes oculaire” comme données	20
6.2.3	Améliorer l'architecture de CNN	21
6.3	Régression avec CNN	21
7	Conclusion	23

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.

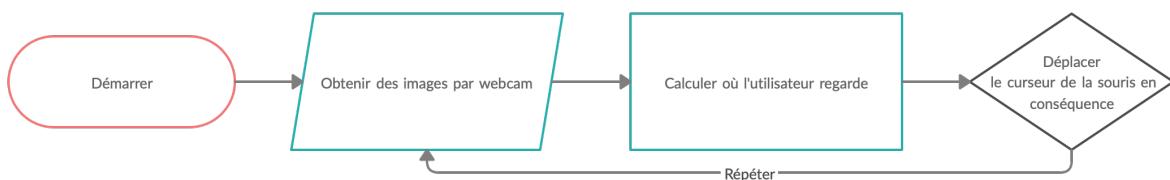


FIGURE 1.1
Procédure générale

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. Je vais expérimenter avec différentes tailles de grille, comme 2x2, 3x3 et 4x4.

		0	1	2	3
		0	1	2	3
		3	4	5	7
2	3	6	7	8	11
		12	13	14	15

FIGURE 1.2
Exemples de grilles

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.

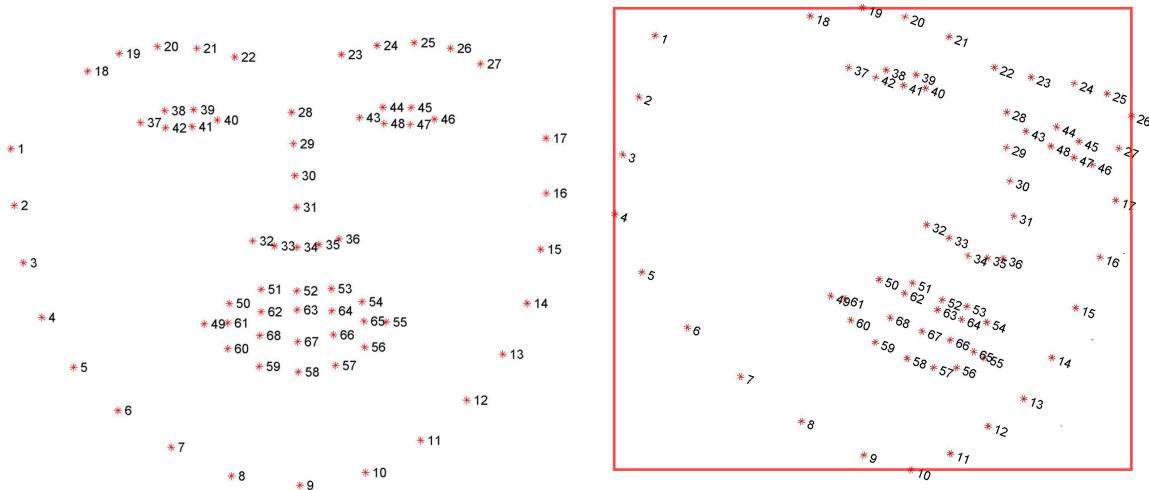


FIGURE 1.3
Repères faciaux
Source : ibug

CHAPITRE 2

PLAN DE DÉVELOPPEMENT

2.1 COMMENT RÉSOUTRE 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 φ , 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.

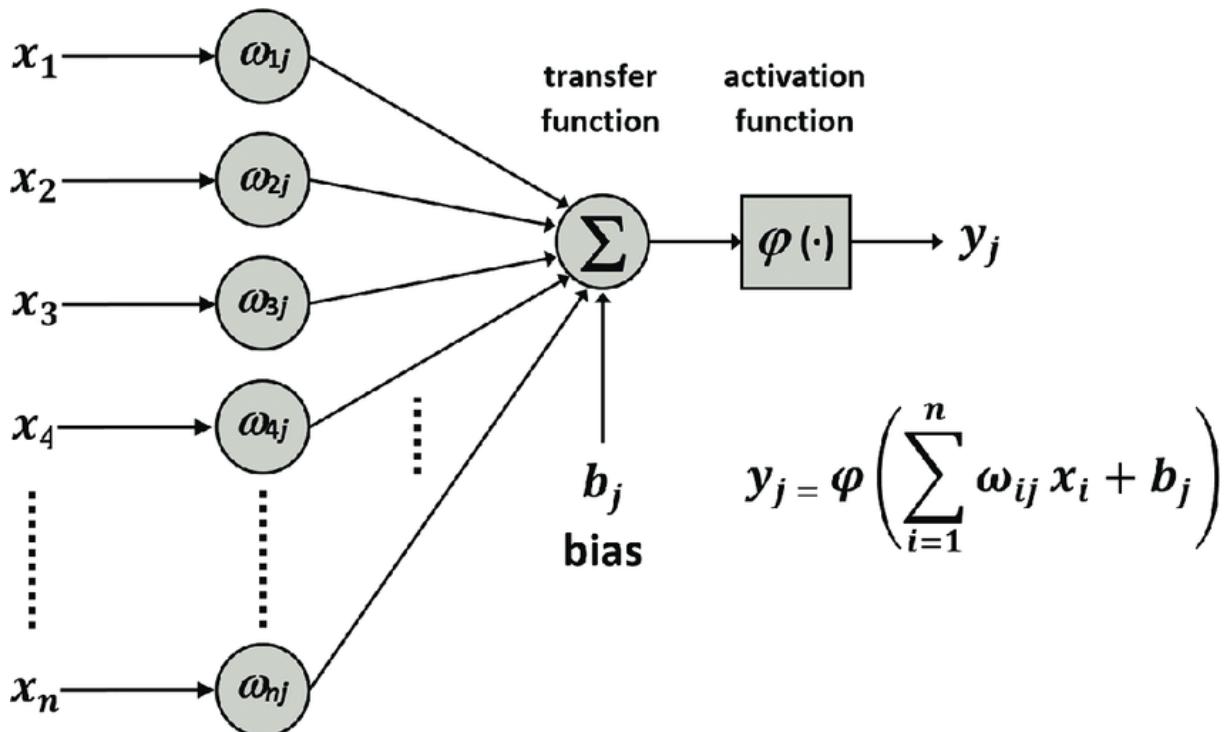


FIGURE 2.1
Schéma d'un neurone artificiel
Source : ResearchGate

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.

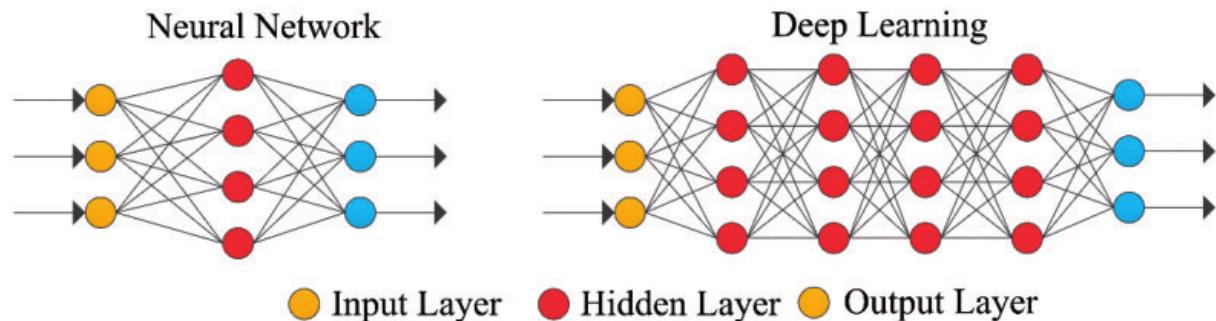


FIGURE 2.2
Schéma d'un neurone artificiel
Source : ResearchGate

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.

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.

```
1 def start_collecting(self, collection_type="background"):
2     print(f'Start collecting data in {collection_type} mode')
3     WebcamCapturer.start_capturing()
4     self.gui.start()
5     print('DataCollectorGUI started')
6     if collection_type == "background":
7         self.mouse_listener.start_listening()
8         print('Mouse listener started')
9     else:
10        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.

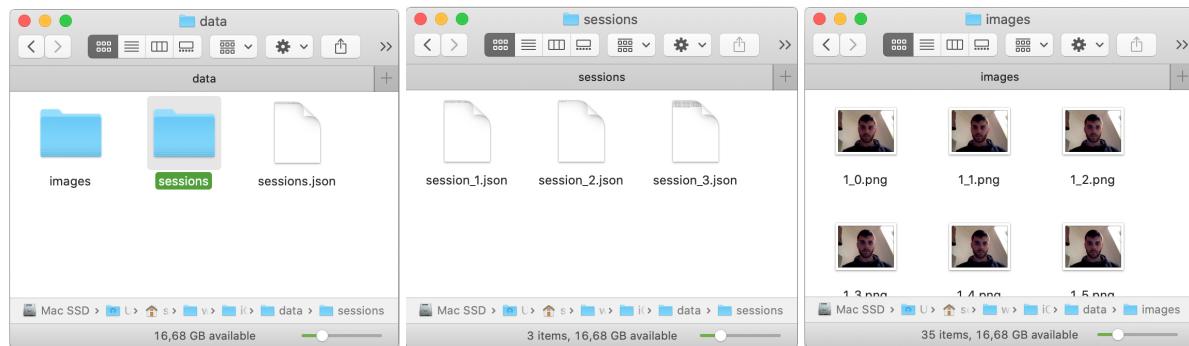


FIGURE 4.1
How data is structured

```

1 def save_collected_data(self):
2     if len(self.collected_data) == 0:
3         return
4     self.collect_data_lock.acquire()
5     session_no = self.get_session_number()
6     print(f"Saving data for session_{session_no}")
7     self.save_session_info(session_no)
8     self.save_images_info(session_no)
9     self.save_images(session_no)
10    self.collect_data_lock.release()
11    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.

```
1 def extract_eyes(cv2_image):
2     """Returns a list of images that contain the eyes extracted from the original
3     image.
4
5     First result is the left eye, second result is the right eye."""
6     global stuff_was_initialized, face_detector, face_predictor
7     if stuff_was_initialized == False:
8         initialize_stuff()
9
10    gray_image = Utils.convert_to_gray_image(cv2_image)
11    rects = face_detector(gray_image, 0)
12    if len(rects) > 0:
13        shape = face_predictor(gray_image, rects[0])
14        shape = face_utils.shape_to_np(shape)
15
16    eyes = []
17    for eye in ["left_eye", "right_eye"]:
18        # get the points for the contour
19        (eye_start, eye_end) = face_utils.FACIAL_LANDMARKS_IDXS[eye]
20        contour = shape[eye_start:eye_end]
21        # get the upper left point, lower right point for this eye
22        start = [min(contour, key=lambda x: x[0])[0],
23                 min(contour, key=lambda x: x[1])[1]]
24        end = [max(contour, key=lambda x: x[0])[0],
25                 max(contour, key=lambda x: x[1])[1]]
26        # extract the current eye
27        eyes.append(cv2_image[start[1]:end[1], start[0]:end[0]])
28
29    return eyes
30
31 return None
```

LISTING 5.1
Extraire les 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.

```

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

```

LISTING 5.2
Application d’un seuil binaire

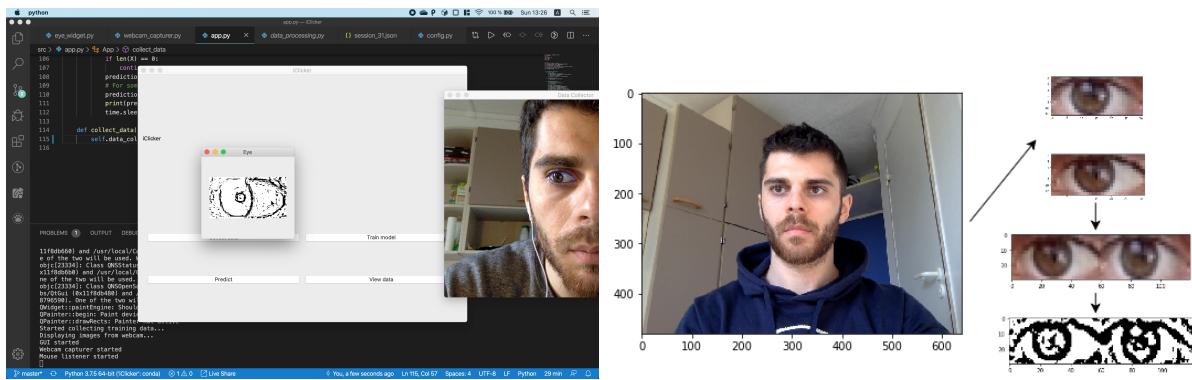


FIGURE 5.1
Obtenir des données sur les yeux

5.3 EN UTILISANT SEULEMENT LE VISAGE

Mon idée suivante était d’utiliser le visage et de le donner en entrée à une CNN. Voici à quoi ressemble le traitement des données :



FIGURE 5.2
Extraire le visage

L’image finale est un carré contenant le visage extrait des images, converti en niveaux de gris. Elle a également été normalisée pour prendre des valeurs entre 0 et 1. Ensuite, en fonction de la taille de la grille 1.2, les images ont été étiquetées avec le numéro de la cellule correspondant à l’endroit où l’utilisateur regardait.

```

1 def extract_face(cv2_image):
2     """Returns the face part extracted from the image"""
3     global stuff_was_initialized, face_detector
4     if stuff_was_initialized == False:
5         initialize_stuff()
6
7     gray_image = Utils.convert_to_gray_image(cv2_image)
8     rects = face_detector(gray_image, 0)
9     if len(rects) > 0:
10         # only for the first face found
11         (x, y, w, h) = face_utils.rect_to_bb(rects[0])
12         return cv2_image[y:y+h, x:x+w]
13     return None

```

LISTING 5.3
Extraction du visage en Python3

5.4 “BANDEAU OCULAIRE”

Next, I’ll try to use only the eyes as data, but without modifying them. Here’s how that looks :

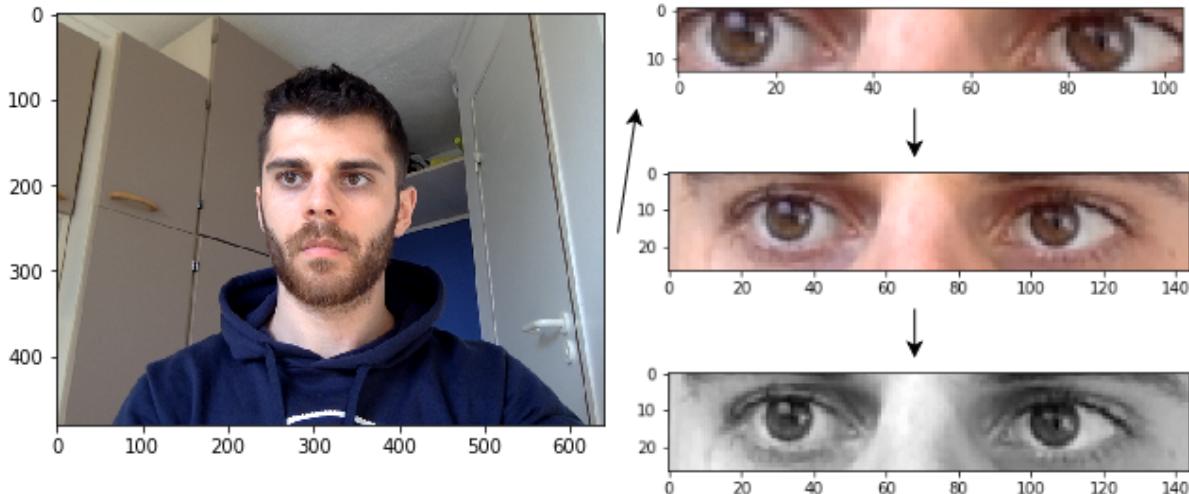


FIGURE 5.3
Extraction de “bandeau oculaire”

I extract both eyes from the detected face, I expand the bounding box to have more information, then I convert it to grayscale and normalise it.

```

1 def extract_eye_strip(cv2_image):
2     """Returns a horizontal image with the two eyes extracted from the image"""
3     global stuff_was_initialized, face_detector, face_predictor
4     if stuff_was_initialized == False:
5         initialize_stuff()
6
7     gray_image = Utils.convert_to_gray_image(cv2_image)
8     rects = face_detector(gray_image, 0)
9     if len(rects) > 0:
10         # only for the first face found
11         shape = face_predictor(gray_image, rects[0])
12         shape = face_utils.shape_to_np(shape)
13         (left_eye_start,
14          left_eye_end) = face_utils.FACIAL_LANDMARKS_IDXS["left_eye"]
15         (right_eye_start,
16          right_eye_end) = face_utils.FACIAL_LANDMARKS_IDXS["right_eye"]
17         # get the contour
18         start, end = min(left_eye_start, right_eye_start), max(

```

```
19         left_eye_end, right_eye_end)
20     strip = shape[start:end]
21     # get the upper left point, lower right point
22     start = [min(strip, key=lambda x: x[0])[0],
23               min(strip, key=lambda x: x[1])[1]]
24     end = [max(strip, key=lambda x: x[0])[0],
25             max(strip, key=lambda x: x[1])[1]]
26     # go a little outside the bounding box, to capture more details
27     distance = (end[0] - start[0], end[1] - start[1])
28     # 20 percent more details on the X axis, 60% more details on the Y axis
29     percents = [20, 60]
30     for i in range(0, 2):
31         start[i] -= int(percents[i]/100 * distance[i])
32         end[i] += int(percents[i]/100 * distance[i])
33     return cv2_image[start[1]:end[1], start[0]:end[0]]
34 return None
```

LISTING 5.4
Extraction de “bandeau oculaire” en Python3

CHAPITRE 6

EXPÉRIMENTATIONS

Toutes ces expériences sont également bien détaillées ici : <https://github.com/sergiuiacob1/iClicker/wiki>

6.1 PERCEPTRONS MULTICOUCHES

Les premières expériences que j'ai faites ont consisté à utiliser des images dont j'ai extrait les yeux, comme mentionné dans la section 5.2 du traitement des données. Chaque œil a été redimensionné à 60px par 30px puis ils ont été fusionnés horizontalement, formant une image noir et blanc de 120px par 30px. De plus, chaque image a ensuite été étiquetée avec un numéro de cellule, correspondant à la cellule que l'utilisateur regardait sur la grille donnée 1.2. Nous allons essayer de prédire ce numéro de cellule. Voici la première architecture que j'ai essayée :

```
1 model = Sequential([
2     Dense(100, input_shape=(n,), kernel_initializer='glorot_uniform'),
3     Dropout(0.5),
4     ReLU(),
5     Dense(128, kernel_initializer='glorot_uniform'),
6     Dropout(0.5),
7     ReLU(),
8     Dense(64, kernel_initializer='glorot_uniform'),
9     # Dropout(0.5),
10    ReLU(),
11    Dense(Config.grid_size * Config.grid_size, activation='softmax')
12])
```

LISTING 6.1
MLP 1

IMPORTANT Pour les premières expériences de cette section, je n'ai utilisé que les images qui correspondaient à l'utilisateur regardant une des cellules dans les coins. Par exemple, pour une grille de 3x3, je n'ai choisi que les images dans lesquelles l'utilisateur regardait les cellules 0, 2, 6 ou 8. De plus, les données d'entraînement représentaient 80% des données totales, le reste étant des données de test.

Expérience	Grille taille	Taille des données	Epoques	Optimizer	Learning rate	Batch size
1	2x2	2184	300	Adam	0.001	32
2	3x3	1247	300	Adam	0.001	32
3	4x4	711	300	Adam	0.001	32

Pour les expériences 2 et 3, je n'ai pris que les images correspondant aux cellules dans les coins, le nombre d'images est donc inférieur à celui de la grille 2x2. Pour les 3 premières expériences, seule la première a bien prédit la cellule que je regardais. Le réseau MLP s'est très bien comporté lorsque j'ai ouvert davantage les yeux, car il a mieux réussi à construire le contour de l'œil.

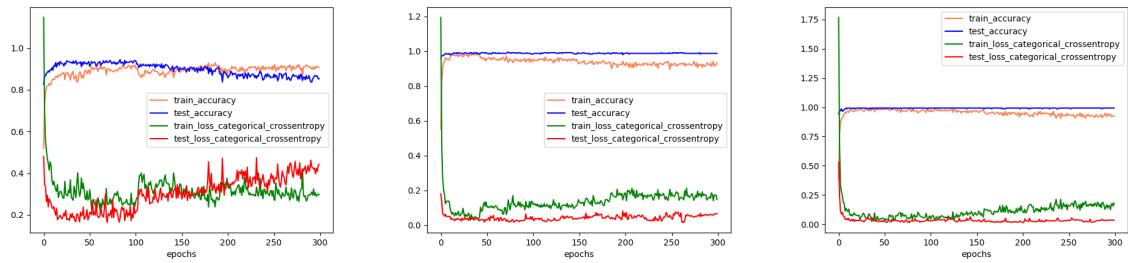


FIGURE 6.1
Modèles 1, 2 et 3

Nous pouvons voir que la précision est vraiment bonne pour tous les modèles, mais dans la vie réelle, les modèles ne sont pas aussi précis. Les conditions d'éclairage et la façon dont l'utilisateur est positionné devant la caméra sont très importantes.

Un élément important à noter, qui sera présent dans la plupart des graphiques, sont les pics, la précision allant de haut en bas. Cela est dû à la façon dont la couche “Dropout” fonctionne : “by randomly dropping units during training to prevent their co-adaptation”, comme mentionné par PIERRE BALDI 2020. Elle élimine les neurones aléatoires du réseau afin d'aider les autres neurones à apprendre. Lorsqu'elle élimine les “bons” neurones, c'est-à-dire ceux qui donnent le plus d'informations, la précision est plus faible et l'erreur plus élevée. Lorsqu'il élimine les neurones qui n'aident pas beaucoup à la prédiction, la précision ne sera pas trop affectée.

J'ai procédé à la formation du même modèle sur la grille 3x3, mais cette fois en utilisant toutes les données dont je disposais. Malheureusement, cela n'a pas beaucoup aidé. J'ai également observé qu'après une certaine période, le réseau ne serait plus précis sur les données de test et qu'il serait surdimensionné.

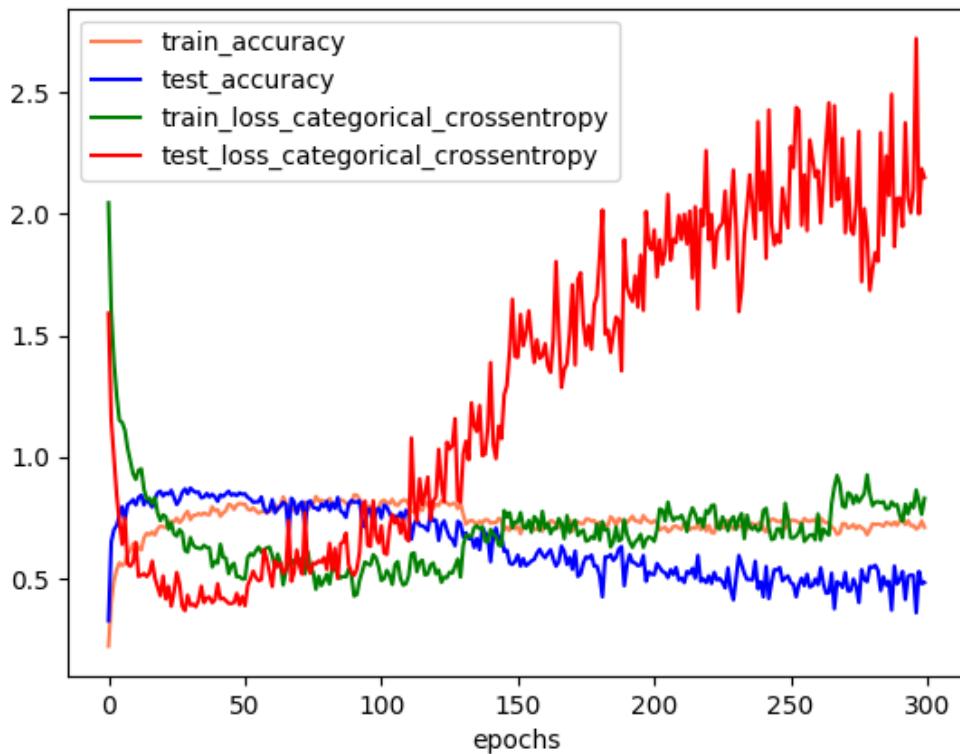


FIGURE 6.2
Modèle 4

Pour résoudre le problème du “overfit”, j’ai introduit un *L2 regularizer* qui, selon KARIM 2020, devrait aider à résoudre ce problème. Cela a en effet aidé à résoudre le problème du “overfit”, car, si l’on regarde les graphiques de précision, on peut voir que l’erreur pour les données du train est maintenant plus proche de l’erreur des données de test. Les modèles ont également mieux prédit où je regardais.

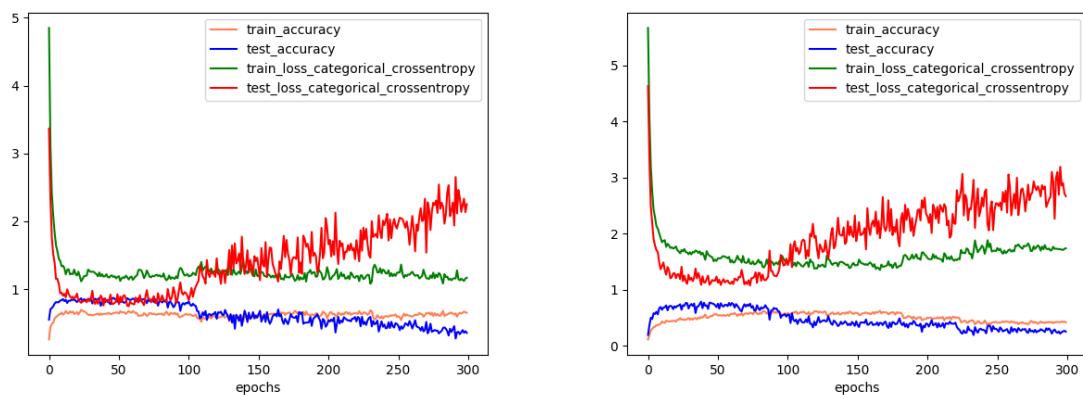


FIGURE 6.3
Modèles 5 et 6

6.2 RÉSEAUX NEURONNAUX CONVOLUTIFS

6.2.1 UTILISATION DES VISAGES EXTRAITS COMME DONNÉES

La première expérience que j'ai faite consistait à utiliser les visages extraits 5.2 sur une grille de 2x2 1.2. Malheureusement, ce fut un échec. Après avoir analysé le code, j'ai découvert que j'avais un bug et que les données n'étaient pas normalisées, ce qui empêchait la CNN d'apprendre. J'ai résolu ce problème et j'ai commencé à expérimenter.

Voici l'architecture réseau que j'ai utilisée :

```

1 model = Sequential()
2 model.add(Conv2D(16, kernel_size=(3, 3),
3                  input_shape=input_shape))
4 model.add(MaxPooling2D(pool_size=(2, 2)))
5 model.add(ReLU())
6 model.add(Conv2D(32, kernel_size=(3, 3)))
7 model.add(MaxPooling2D(pool_size=(2, 2)))
8 model.add(ReLU())
9 model.add(Flatten())
10 model.add(Dense(128, activation='relu'))
11 model.add(Dense(4, activation='softmax'))
12 # optimizer used
13 opt = Adam()

```

LISTING 6.2
Première architecture de CNN

IMPORTANT Pour les premières expériences de cette section, je n'ai utilisé que les images qui correspondaient à l'utilisateur regardant une des cellules dans les coins. Par exemple, pour une grille de 3x3, je n'ai choisi que les images dans lesquelles l'utilisateur regardait les cellules 0, 2, 6 ou 8. De plus, les données d'entraînement représentaient 80% des données totales, le reste étant des données de test.

Voici les paramètres :

Expérience	Grille taille	Taille des données	Epoques	Optimizer	Learning rate	Batch size
7	2x2	2184	50	Adam	0.001	32
8	3x3	1247	50	Adam	0.001	32
9	4x4	711	50	Adam	0.001	32

Ici, nous pouvons voir comment les modèles se comportent :

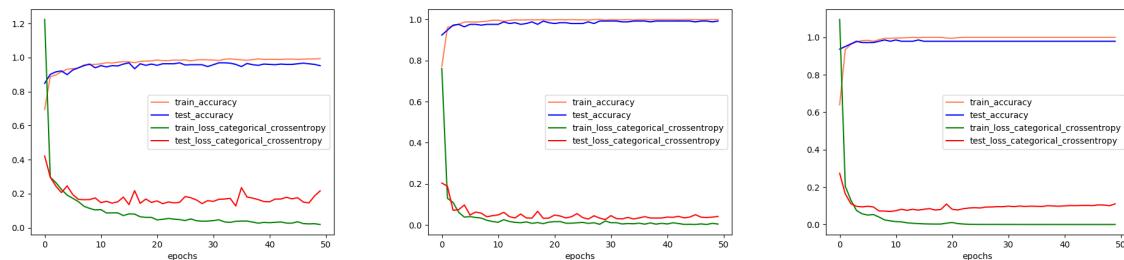


FIGURE 6.4
Modèles 7, 8 et 9

Les modèles 8 et 9 ont eu de bons scores, mais ils étaient instables quand il s'agissait de prédire où je regardais. Le modèle 7 a fait mieux : si je m'assieds dans le bon angle, il peut prédire assez bien où je regarde. Cela est probablement dû au fait que le modèle a été entraîné avec plus de données.

Pour les expériences 10 et 11, j'ai rassemblé plus d'images et j'ai formé la même CNN sur l'ensemble des données (pas seulement les données de coin), ce qui représente environ 2736 d'images, pour des grilles de 3x3 et 4x4. Les prévisions avec ces modèles étaient meilleures, ce qui m'a poussé à la conclusion que plus j'avais de données, mieux c'était. On peut également voir un signe de "surcharge" dans ces modèles.

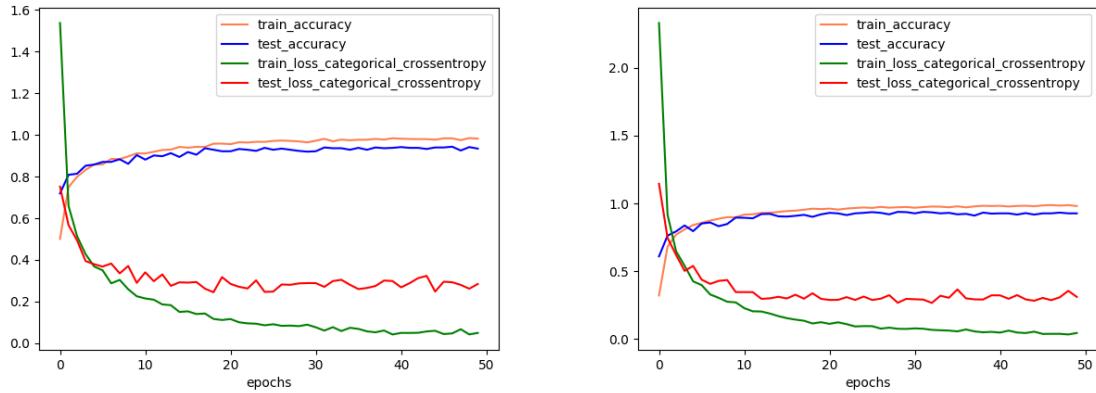


FIGURE 6.5
Modèles 10 et 11

6.2.2 UTILISATION DE “BANDES OCULAIRE” COMME DONNÉES

J'ai utilisé la même architecture CNN que celle ci-dessus, mais cette fois en utilisant des données différentes 5.3. Les premières expériences n'utilisaient que des images dans lesquelles l'utilisateur regardait près des coins de l'écran.

Expérience	Grille taille	Taille des données	Epoques	Optimizer	Learning rate	Batch size
12	2x2	2184	50	Adam	0.001	32
13	3x3	1247	50	Adam	0.001	32
14	4x4	711	50	Adam	0.001	32

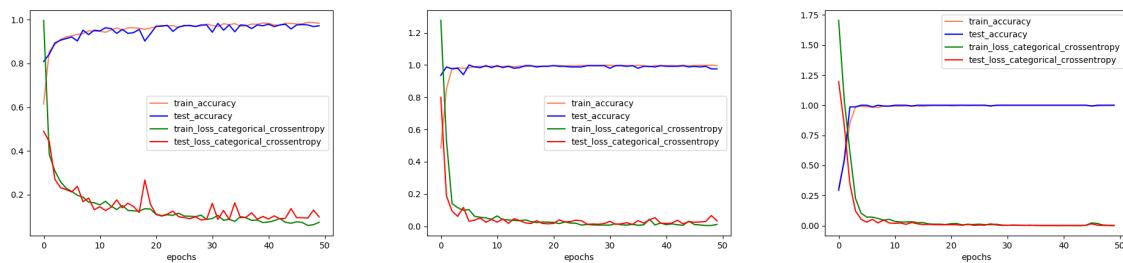


FIGURE 6.6
Modèles 12, 13 et 14

L'expérience 12 était bonne, meilleure que la plupart des autres jusqu'à présent. Elle pouvait assez bien prédire où je regardais, même si je m'éloignais de l'écran. Les expériences 13 et 14 n'étaient pas si bonnes, même si elles avaient une bonne précision et de faibles erreurs. Une fois de plus, cela s'est probablement produit parce que le modèle formé sur une grille 2x2 obtient plus de données, car pour ces expériences, j'ai seulement sélectionné les images dans lesquelles l'utilisateur regarde une cellule de coin, et sur une grille 2x2, chaque image correspond à un coin.

Pour les expériences 15 et 16, j'ai entraîné la même CNN sur des grilles de 3x3 et 4x4, mais cette fois en utilisant toutes les données dont je dispose. Cette fois, ils ont fait de meilleures prévisions, surtout

quand je regardais dans les coins. Cependant, avec les modèles 15 et 16, lorsque je regardais au centre de l'écran, on disait parfois que je regardais l'un des coins.

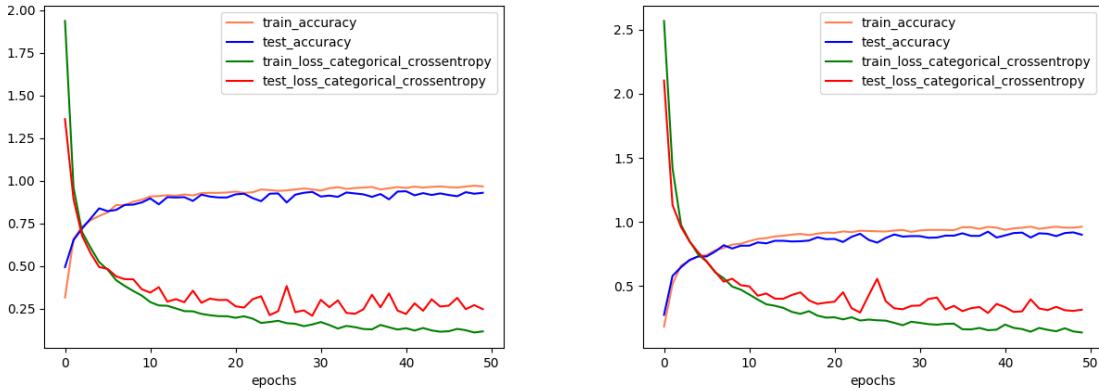


FIGURE 6.7
Modèles 15 et 16

COMMENTAIRE JUSQU'À PRÉSENT D'après les expériences menées jusqu'à présent, j'ai conclu que CNN en combinaison avec des "bandeau oculaire" est la meilleure option. De plus, lorsque j'ai entraîné le même modèle sur un plus grand nombre de données, il a toujours donné de meilleurs résultats. Il est donc extrêmement important d'avoir le plus de données possible et aussi une certaine variété dans les images que j'utilise.

6.2.3 AMÉLIORER L'ARCHITECTURE DE CNN

Les expériences 17 – 22 consistaient à essayer d'améliorer la première architecture de CNN 6.2.1. J'ai essayé de supprimer une des couches de convolution et de modifier le nombre de filtres de convolution.

J'ai remarqué que lorsque j'ai retiré la deuxième couche de convolution, le réseau ne pouvait pas prédire certaines cellules qu'il avait correctement prédites auparavant (par exemple la cellule numéro 2 sur une grille 3x3). De plus, en augmentant le nombre de filtres de convolution, les prédictions s'amélioraient, j'ai donc décidé de continuer avec cette architecture :

```

1 model = Sequential()
2 model.add(Conv2D(64, kernel_size=(3, 3),
3                  input_shape=input_shape))
4 model.add(MaxPooling2D(pool_size=(2, 2)))
5 model.add(ReLU())
6 model.add(Conv2D(128, kernel_size=(3, 3)))
7 model.add(MaxPooling2D(pool_size=(2, 2)))
8 model.add(ReLU())
9 model.add(Flatten())
10 model.add(Dense(128, activation='relu'))
11 model.add(Dense(Config.grid_size * Config.grid_size, activation='softmax'))
```

LISTING 6.3
Nouvelle architecture de la cnn

6.3 RÉGRESSION AVEC CNN

Pour mes prochaines expériences, j'ai essayé de prédire la position exacte de la souris et, sur cette base, de voir quelle cellule l'utilisateur regarde.

Expérience	Grille taille	Taille des données	Epoques	Optimizer	Learning rate	Batch size
22	3x3	2736	50	Adam	0.001	32
23	3x3	2736	100	Adam decay= 10^{-4}	0.001	32
24	3x3	2736	100	Adam decay= 10^{-4}	0.01	32

Au début, l'expérience 22 était chaotique, mais j'ai découvert que si je tiens l'ordinateur portable dans le bon angle et à la bonne hauteur, les prédictions sont assez bonnes. Par la suite, j'ai formé le même modèle pour d'autres époques et j'ai également introduit une diminution du taux d'apprentissage, qui actualise le taux d'apprentissage par cette formule : $lr = \text{initial_lr} * (1 / (1 + \text{decay} * \text{iteration}))$. Cela a permis au réseau d'apprendre rapidement au début, puis d'apprendre plus lentement, mais de s'améliorer encore à la fin. Ces modèles étaient les meilleurs jusqu'à présent.

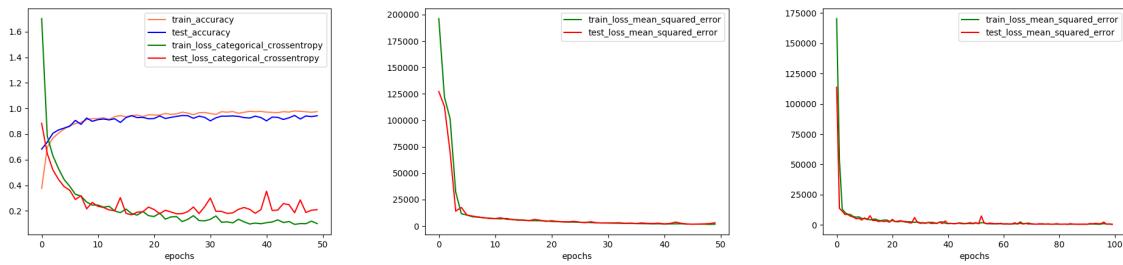


FIGURE 6.8
Modèles 12, 13 et 14

CHAPITRE 7

CONCLUSION

J'ai commencé ce projet en essayant de prédire où un utilisateur regarde à l'écran. Au fil du temps, j'ai constaté que ce problème n'est pas très facile à résoudre, car il dépend de nombreux facteurs : la qualité de l'éclairage de la pièce, la position de l'utilisateur devant la caméra, la distance, etc. J'ai commencé par expérimenter les technologies actuelles, les méthodes connues pour résoudre ce problème et voir comment chaque méthode fonctionne, tout en essayant de comprendre comment et pourquoi l'une est meilleure que l'autre.

À un moment donné, j'ai dû faire des recherches scientifiques pour comprendre comment les choses fonctionnent, ce qui s'est avéré être une tâche difficile. J'ai appris que je dois documenter les progrès que je fais et que je dois écrire comment je l'ai fait et quelles sont mes observations. Dans ce problème, il était également très important de savoir comment les petites modifications que j'ai apportées à l'architecture d'un réseau m'ont aidé à obtenir de meilleurs résultats. Par exemple, l'ajout de filtres convolutifs sur un réseau CNN a permis d'obtenir de meilleurs résultats, parce que le réseau pouvait calculer plus de détails sur les images que je lui fournissais, ou qu'en utilisant plus de données, les réseaux en apprenaient davantage.

Certains travaux sont encore en cours, notamment pour comprendre comment fonctionnent les bibliothèques que j'ai utilisées, telles que `dlib` ou `imutils`, et comment les repères faciaux 1.3 sont extraits des images. Néanmoins, ce projet m'a été très utile pour apprendre davantage de choses dans le domaine de l'Apprentissage Automatique.

BIBLIOGRAPHIE

- WIKIPEDIA (2020). *Eye Tracking*. Wikimedia Foundation. URL : https://en.wikipedia.org/wiki/Eye_tracking (visité le 15 mar. 2020).
- PIERRE BALDI, Peter Sadowski (2020). *The dropout learning algorithm*. Science Direct. URL : <https://www.sciencedirect.com/science/article/pii/S0004370214000216> (visité le 10 avr. 2020).
- KARIM, Raimi (2020). *Intuitions on L1 and L2 Regularisation*. Towards Data Science. URL : <https://towardsdatascience.com/intuitions-on-l1-and-l2-regularisation-235f2db4c261> (visité le 17 avr. 2020).