

Projet chef d'œuvre

Alyes_tracker

Sommaire

<u>Introduction.....</u>	<u>3</u>
<u>Problématique.....</u>	<u>3</u>
<u>Analyse des besoins.....</u>	<u>3</u>
<u>Gestion de projet.....</u>	<u>4</u>
<u>Intelligence artificielle.....</u>	<u>5</u>
<u>Dataset.....</u>	<u>5</u>
<u>Méthodes.....</u>	<u>9</u>
<u>Résultat.....</u>	<u>11</u>
<u>Application.....</u>	<u>13</u>
<u>Base de données.....</u>	<u>14</u>
<u>Backend.....</u>	<u>18</u>
<u>Frontend.....</u>	<u>20</u>
<u>Qualité et Monitoring.....</u>	<u>22</u>
<u>Axes d'amélioration.....</u>	<u>23</u>
<u>Conclusion.....</u>	<u>24</u>

Introduction

Ce projet de fin d'étude a pour but de mettre en pratique toutes les connaissances apprises durant ce cursus de Data et IA d'une durée 19 mois (4 mois de formation intensive puis 15 mois d'alternance). Ce projet m'a notamment permis de consolider mes acquis mais aussi de combler mes lacunes, Il m'a en outre offert la possibilité d'aborder un domaine spécifique auquel je souhaitais m'intéresser.

Problématique

Nous utilisons tous au quotidien les ordinateurs. Ils n'ont cessé de se perfectionner au fil des années. Cependant les différents moyens d'interagir avec eux, tel que la souris ou le clavier n'ont pas connu une évolution aussi fulgurante. C'est pourquoi depuis plusieurs années, des entreprises essaient de proposer de nouvelles manières d'interagir avec la machine, tels que la réalité virtuelle, la commande vocale ou encore le suivi du regard. Nous allons nous intéresser à ce dernier. Sur le marché, il en existe une très grande variété. Les plus connus sont ceux de la marque Tobii ou encore Steelseries. Cependant tous les produits sur le marché sont basés sur le même principe. En effet ils utilisent tous du Computer Vision pour localiser la pupille et détecter les reflets dans l'œil et ce afin de montrer à l'utilisateur l'endroit de l'écran qu'il regarde. De plus ces dispositifs nécessitent une caméra spéciale qu'on place sous l'écran pour certains d'entre eux, ou directement devant les yeux pour d'autres. Cela nécessite au préalable d'étalonner le dispositif. Il est relativement onéreux puisqu'il coûte environ 200 Euros, ce qui freine la démocratisation de cette technologie. En outre il faut souligner que les résultats ne sont pas toujours de bonne qualité.

C'est pourquoi j'ai choisi de développer mon propre Eye tracker : « Alyes_tracker »

L'application sera centrée autour d'une IA capable de prédire à partir d'un flux d'image les endroits successifs où se fixent les regards de la personne présente dans ce flux. Notre application ne nécessitera pas l'achat d'une caméra spéciale, elle se contentera de la webcam de base présente sur chaque ordinateur. L'utilisateur pourra consulter les différents résultats grâce à de la visualisation de données. Il pourra en outre accéder à l'application sur son navigateur web grâce à une application Flask.

Analyse des besoins

L'application est conçue de telle manière qu'il n'y ait qu'un seul utilisateur : la personne désirant faire suivre son regard sur l'écran.

L'application doit respecter les scénarios d'utilisation suivants :

- en tant qu'utilisateur je veux savoir où je regarde en temps réel sur l'écran,
- en tant qu'utilisateur je veux enregistrer tous les endroits où j'ai regardé,
- en tant qu'utilisateur je veux pouvoir avoir accès à tous les enregistrements que j'ai effectués,
- en tant qu'utilisateur je veux avoir un rendu visuel sur un enregistrement,
- en tant qu'utilisateur je veux être protégé par un système d'authentification afin d'être le seul à pouvoir consulter mes enregistrements,

Gestion de projet

Ce projet a été réalisé grâce à la méthode kanban. Cette méthode a été retenue car elle a apporté de nombreux avantages comme une bonne visualisation, un suivi des différentes étapes, des tâches facilitées ou encore l'instauration d'une WIP LIMIT (limitée à 2 dans mon cas). Le déroulement s'est fait en 4 étapes principales :

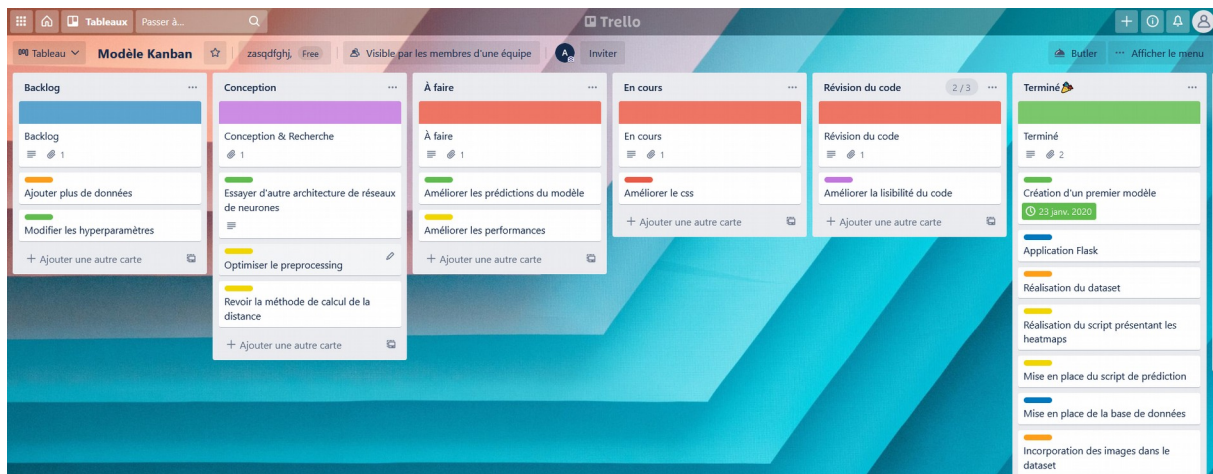
1 – Conception : c'est ici qu'a été imaginé l'architecture du projet avec notamment les différentes étapes ainsi que l'anticipation des futurs points bloquants.

2 – Collecte de données : c'est à cette étape que la collecte de données a été effectuée afin de pouvoir réaliser notre dataset.

3 – Réalisation du réseau de neurone : maintenant que nous avons nos données, nous pouvons créer et entraîner notre IA.

4 – Mise en production : réalisation des scripts afin de pouvoir utiliser notre IA et notamment l'implémentation au sein d'une application Flask.

Cette méthodologie a été implémentée sur l'outil de gestion de projet en ligne Trello, dont voici un extrait :



Dans cette méthodologie j'ai joué le double rôle du développeur et du client. En effet, d'une part je développais l'application avec toutes ses fonctionnalités et de l'autre en tant que client je regardais si l'avancement allait bien dans le sens de la création du produit final.

Pour cela je me suis basé sur une Definition Of Done telle que : toute l'équipe doit considérer que la tâche est fonctionnelle et peut-être intégrée facilement dans l'application. En outre je faisais un point quotidien sur l'avancement afin de ne pas me disperser et rester « focus » sur les tâches en cours.

	Septembre	Octobre	Novembre	Décembre	Janvier	Février	Mars
Conception							
Dataset							
Model							
Mise en production							

Sur ce retro planning on peut voir que le projet s'est déroulé sur 7 mois. La partie création du model a été particulièrement longue en raison du fait qu'il s'agissait de la partie que je maîtrisais le moins, d'autant plus que le model que j'ai utilisé était plus complexe que ce que j'avais anticipé.

Intelligence artificielle

Le but de notre IA est de prédire, à partir de l'image d'un œil ainsi que d'informations s'y référant, les coordonnées où regarde l'utilisateur. Il s'agit ici d'un mixed data model effectuant une double régression. Notre IA doit être capable de faire des prédictions en temps réel, ce qui nous oblige à faire un model le plus léger possible.

Dataset

Afin de pouvoir entraîner cette IA il a bien fallu créer un dataset. Pour constituer ce dataset, je me suis pris en photo grâce à la caméra de l'ordinateur et de la library open-cv. Les photos ont été enregistrées avec comme nom un identifiant ainsi que les coordonnées où je regardais lors de leurs prises. Pour ce faire j'ai utilisé le script suivant :

```
import cv2
import pyautogui

cam = cv2.VideoCapture(0)
cv2.namedWindow("test")
img_counter = 0




















































while True:
    ret, frame = cam.read()
    if not ret:
        print("failed to grab frame")
        break
    cv2.imshow("test", frame)
    k = cv2.waitKey(1)
    if k%256 == 27:
        # ESC pressed
        print("Escape hit, closing...")
        break
    elif k%256 == 32:
        # SPACE pressed
        x, y = pyautogui.position()
        #print(x,y)
        img_name = str(img_counter)+'-'+str(x)+'-'+str(y)
        img_name = "opencv_frame_{}.png".format(img_name)
        cv2.imwrite(img_name, frame)
        print("{} written!".format(img_name))
        img_counter += 1

cam.release()

cv2.destroyAllWindows()
```

Grâce à ce script j'ai pu enregistrer 1000 photos. Je me suis arrangé pour regarder partout sur l'écran afin qu'il n'y ait pas de zones qui ne soient pas couvertes. J'ai par ailleurs varié la luminosité de la pièce où je prenais les clichés dans le but d'obtenir

des données d'entraînement couvrant un plus large panel. Afin de gagner en performance pendant l'entraînement du model mais aussi lors de la mise en production, je n'ai utilisé que des images en noir et blanc.

Par la suite, j'ai pris chaque photo et grâce à 2 classifieurs (un détectant le visage et l'autre détectant les yeux : haarcascade_frontalface_default.xml et haarcascade_eye.xml) j'ai pu obtenir ainsi 2 nouvelles photos (uniquement les yeux) dont j'ai changé la taille en 32 x 32 pixels. Les photos sont de petite taille afin que notre réseau de neurone puisse traiter rapidement les images et produire la prédiction la plus rapide possible, idéalement en temps réel.

```
ret, img = cap.read()
#ret, img2 = cap.read()
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
faces = face_cascade.detectMultiScale(gray, scaleFactor = 1.1, minNeighbors = 5)
for (ex,ey,ew,eh) in faces:
    cv2.rectangle(img, (ex,ey), (ex+ew,ey+eh), (255,0,0),5)
eyes = eye_cascade.detectMultiScale(image, scaleFactor = 1.1, minNeighbors = 5)
for (eex,eey,eeew,eeeh) in eyes:
    if eey < image.shape[0]/2:
```

```
width = 32
height = 32

dim = (width, height)
img_crop1 = cv2.resize(img_crop1, dim, interpolation = cv2.INTER_AREA)
img_crop2 = cv2.resize(img_crop2, dim, interpolation = cv2.INTER_AREA)
```

Les résultats sont les suivants :



Par la suite j'ai utilisé ces photos ainsi qu'un fichier comprenant divers features collectés durant les phases de détections, pour créer un dataset contenant tout ce dont j'avais besoin.

	pixel_0_0	pixel_0_1	pixel_0_2	pixel_0_3	pixel_0_4	pixel_0_5	pixel_0_6	pixel_0_7	pixel_0_8	pixel_0_9	...	pixel_31_31	milieu_crop_x	milieu_crop_y	distance	milieu_oeil_x	milieu_oeil_y	taille_original	cote	prediction_x	prediction_y
0	0.294118	0.282353	0.258824	0.239216	0.219608	0.211765	0.219608	0.211765	0.211765	0.231373	...	0.639216	0.492188	0.418750	0.501	0.431250	0.383333	42	0.0	0.050000	0.050000
1	0.674510	0.666967	0.658824	0.600000	0.529412	0.482353	0.458824	0.427451	0.411765	0.403922	...	0.721569	0.492188	0.418750	0.501	0.546875	0.383333	36	0.5	0.050000	0.050000
2	0.490196	0.490196	0.443137	0.368627	0.329412	0.317647	0.301961	0.286275	0.258824	0.231373	...	0.658824	0.495312	0.422917	0.501	0.437500	0.370833	40	0.0	0.050000	0.150000
3	0.650980	0.603922	0.505882	0.439216	0.423529	0.407843	0.392157	0.372549	0.376471	0.380392	...	0.725490	0.495312	0.422917	0.501	0.543750	0.383333	38	0.5	0.050000	0.150000
4	0.458824	0.443137	0.372549	0.317647	0.305882	0.294118	0.266667	0.247059	0.215686	0.207843	...	0.627451	0.503125	0.425000	0.515	0.448437	0.379167	40	0.0	0.150000	0.050000
...
1995	0.529412	0.560784	0.588235	0.611765	0.611765	0.600000	0.560784	0.525490	0.505882	0.501961	...	0.666667	0.495312	0.560417	0.462	0.553125	0.520833	40	0.5	0.063542	0.837037
1996	0.760784	0.733333	0.717647	0.733333	0.745098	0.745098	0.729412	0.701961	0.678431	0.647059	...	0.854902	0.479687	0.558333	0.469	0.410938	0.508333	48	0.0	0.761979	0.283333
1997	0.458824	0.490196	0.533333	0.500784	0.592157	0.611765	0.627451	0.627451	0.607843	0.584314	...	0.650980	0.479687	0.558333	0.469	0.535937	0.516667	43	0.5	0.761979	0.283333
1998	0.745098	0.756863	0.756863	0.737255	0.721569	0.705882	0.690196	0.662745	0.631373	0.603922	...	0.890196	0.454688	0.562500	0.467	0.392188	0.516667	46	0.0	0.953125	0.531481
1999	0.419608	0.486275	0.517647	0.513725	0.474510	0.435294	0.411765	0.400000	0.415686	0.427451	...	0.556863	0.454688	0.562500	0.467	0.514062	0.518750	42	0.5	0.953125	0.531481

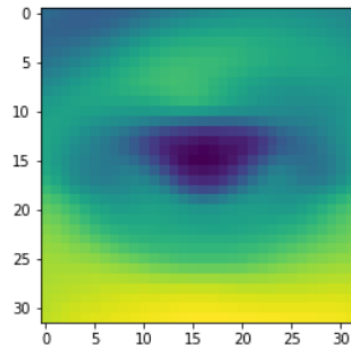
2000 rows x 1033 columns

Ce dataset est composé de 1033 colonnes et 2000 lignes. 1024 colonnes sont consacrées aux valeurs des pixels de l'image, 7 colonnes contiennent des features :

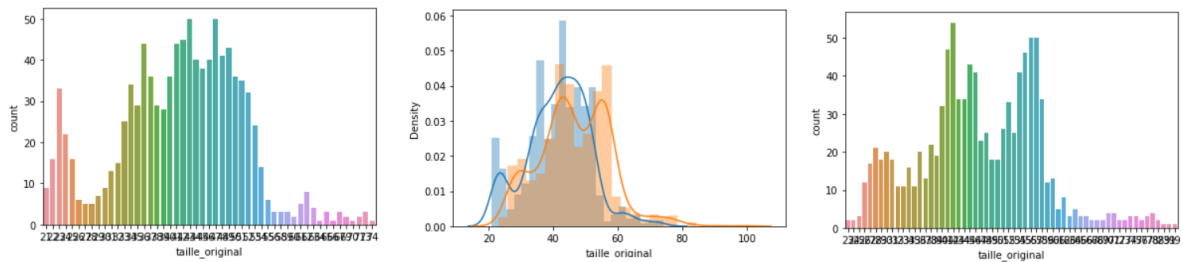
coordonnées x et y du visage, distance, coordonnées x et y du centre de l'œil, taille originale de l'œil ainsi que l'identification de l'œil (droit ou gauche).

Enfin les 2 dernières colonnes correspondent aux 2 valeurs que nous voulons prédire : il s'agit des coordonnées x et y de l'endroit que nous regardons sur l'écran. Au total 1000 photos ont été prises pour un total de 2000 yeux. Ici toutes les données ont été normalisées.

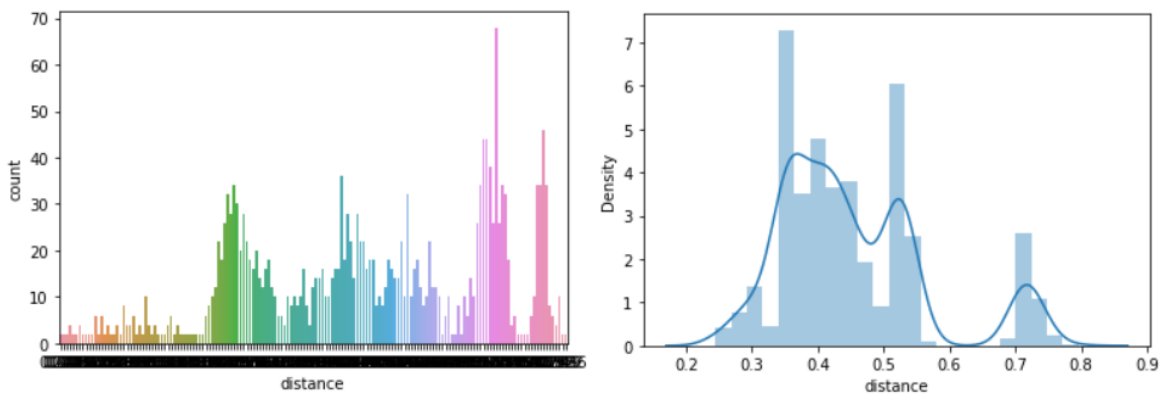
Voici quelques exemples d'explorations de données de ce dataset.



Ici l'image de l'œil correspondant à la moyenne de tous les yeux du dataset. Nous remarquons à mi-hauteur à gauche et à droite un léger trait bleu horizontal correspondant aux coins de chaque œil.



Ici nous comparons la taille des yeux selon qu'il s'agit du gauche ou du droit. Nous remarquons que les yeux gauche (graphique de gauche et couleur bleue dans le graphique du milieu) ont une taille dans l'ensemble plus petite. Ceci peut être expliqué soit par le fait que j'ai un œil plus grand que l'autre soit que je ne me trouvais pas exactement en face de la caméra lors de la prise des clichés.



Ces graphiques représentent la distance à laquelle ont été pris les clichés.

Une fois ce dataset réalisé, je n'ai pas eu besoin de faire de nettoyage de données. En effet ayant créé ce dataset de toute pièce avec mes propres clichés, je me suis arrangé pour qu'à chaque étape de transformation d'image et de collecte de features il n'y ait pas de pertes d'information où de données erronées enregistrées, telles qu'une narine prise pour un œil ou encore un œil détecté plusieurs fois comme ci-dessous :



Méthodes

Grâce à ce dataset nous avons pu entraîner notre IA. Pour rappel il s'agit d'un mixed data model effectuant une double régression.

Pour entraîner notre IA nous avons utilisé toutes les données de notre dataset que nous avons séparé avec un ratio de 80% - 20% en train et test. Une fois cela effectué, nous avons de nouveau fait une séparation mais cette fois, nous avons séparé les 1031 colonnes du dataset en 1024 et 7 (les 2 colonnes de prédiction ayant été retirées au préalable pendant l'utilisation de la méthode `train_test_split()`). Ainsi nous avons nos 2 inputs pour chacune de nos branches aussi bien en train qu'en test.

```
y = df.iloc[0:2000, 1031:1033]
x = df.iloc[0:2000, 0:1031]
x_train, x_val, y_train, y_val = train_test_split(x, y, test_size=0.2, random_state=1)

x_train = x_train.to_numpy()
x_val = x_val.to_numpy()
y_train = y_train.to_numpy()
y_val = y_val.to_numpy()

X1_train = x_train[0:x_train.shape[0], 0:1024]
X2_train = x_train[0:x_train.shape[0], 1024:1031]

X1_val = x_val[0:x_val.shape[0], 0:1024]
X2_val = x_val[0:x_val.shape[0], 1024:1031]

X1_train = X1_train.reshape(1600, 32, 32)
X1_val = X1_val.reshape(400, 32, 32)
```

Cette IA faisant un traitement sur des images, j'ai choisi de partir directement sur des couches convolutives d'une part mais aussi des couches denses pour la partie ne traitant pas d'images.



Notre premier model avec la 1ère branche (haut) constituée de 5 fois les couches :

```
x = Conv2D(3, (3, 3), padding="same") (x)
x = Activation("relu") (x)
x = BatchNormalization(axis=1) (x)
x = MaxPooling2D(pool_size=(2, 2)) (x)
```

La 2ème branche est beaucoup plus simple, elle est constituée de 4 couches denses avec activation relu.

Puis nous réunissons les 2 branches grâce à la méthode concatenate() après avoir fait un Flatten().

Cette réunion est constituée de 5 couches denses avec activation relu.

Durant cet entraînement nous utilisons la mae en tant loss, l'optimizer adam et nous observons aussi par ailleurs l'évolution de la metric mse.

$$mae = \frac{\sum_{i=1}^n abs(y_i - \lambda(x_i))}{n}$$

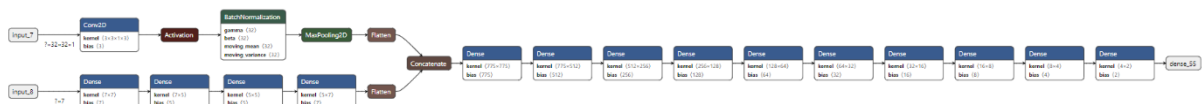
Mae : Mean absolute error. Cette metric mesure la dispersion des prédictions ; plus la valeur est élevée plus les prédictions sont éloignées de ce qu'elles devraient être et inversement. Dans notre cas nous voulons prédire les coordonnées des pixels où l'on regarde ; il faut donc que cette valeur soit faible afin de pouvoir prédire des coordonnées les plus proche du réel.

En tout l'entraînement est programmé sur 2000 epochs avec des batchs de 128. Heureusement une epoch durant environ 1,5s, l'entraînement n'est pas extrêmement long.

Comme l'on peut s'en douter cette première IA n'a pas de bons résultats avec notamment une val_loss à 0.28, d'autant plus que pour l'instant il n'y a pas encore d'utilisation de callbacks.

Par la suite il a fallu améliorer les résultats. Ils se sont améliorés au fur et à mesure que nous avons procédé à des changements :

Changement d'architecture :



Ici le schéma final de notre IA

Cette architecture comprend moins de couches convolutives sur la 1ère branche et contient plus de couches denses après le l'utilisation de la méthode concatenate().

Dans les couches denses l'activation relu a été remplacée par l'activation swish. Swish est une version améliorée de relu développée récemment par Google.

Modification des hyperparamètres ainsi que l'ajout de callbacks :

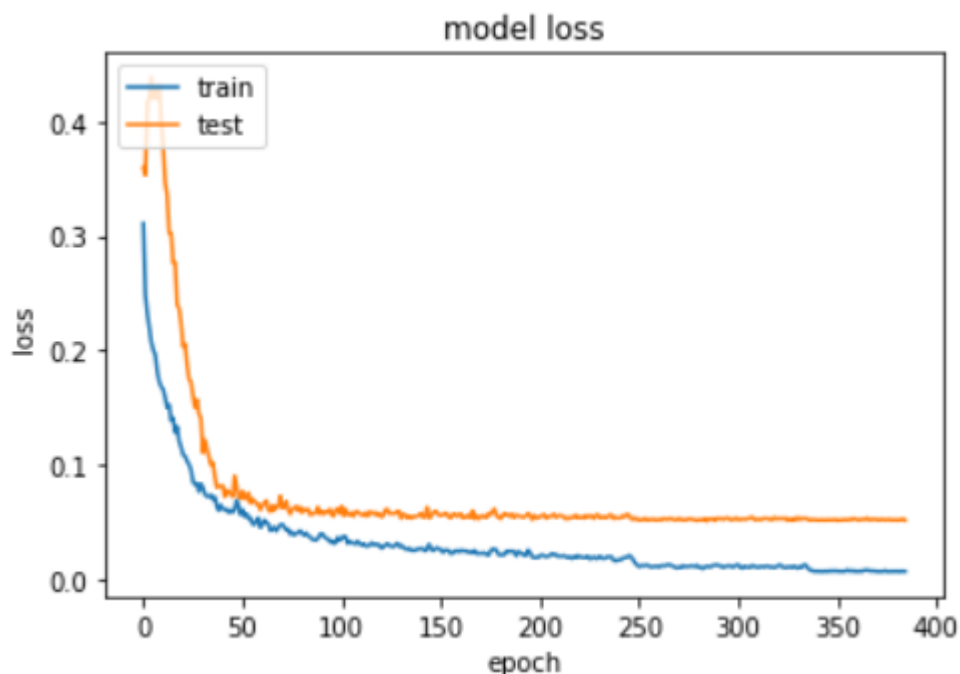
- EarlyStopping afin que notre entraînement ne dure plus que nécessaire et ainsi limiter l'overfitting. Ici l'entraînement s'arrête si les résultats ne se sont pas améliorés pendant les 100 dernières epochs.
- ModelCheckpoint afin de garder le model au moment où il était le plus performant pendant l'entraînement.
- ReduceLROnPlateau afin de changer le learning rate lorsque le modèle n'arrive plus à améliorer les résultats. Ici, si les résultats ne se sont pas améliorés durant les 50 dernières epochs, le learning rate va être divisé par 2.

```
model.compile(loss=['mae'], optimizer='adam', metrics=['mse'])
history = model.fit(x=[X1_train, X2_train], y=y_train, batch_size=128, epochs=2000,
                    verbose=1, callbacks=[earlyStopping, mcp_save, reduce_lr_loss],
                    validation_data = ([X1_val, X2_val], y_val))
```

```
earlyStopping = EarlyStopping(monitor='val_loss', patience=100, verbose=1, mode='min')
mcp_save = ModelCheckpoint('checkpoint.hdf5', save_best_only=True, monitor='val_loss', mode='min', verbose=0)
reduce_lr_loss = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=50,
                                   verbose=1, mode='min', min_lr=0.000001)
```

Résultat

Tous ces changements nous ont permis d'atteindre une val_loss de 0.0514 soit des résultats 5.44 fois meilleurs.



Ce schéma montre la différence de résultats entre les prédictions faites sur le train et le test. Nous remarquons que les résultats sont très similaires aux alentours de 50 -100 epochs. Pour le test les résultats continuent de s'améliorer très légèrement jusqu'à ce que le callback EarlyStopping ne termine l'entraînement. En revanche les résultats continuent de bien s'améliorer pour le train. Cette différence entre le train et

le test nous laisse supposer que notre model commence à overfitter, en particulier au-delà de l'époch 250.

Nous avons vu que la meilleure val_loss que nous avons obtenue était de 0.0514. Ceci, correspond à une erreur d'environ 77 pixels. Cette erreur peut sembler énorme quand on compare à un écran de 1920 x 1080 pixels. Cependant elle n'est pas si élevée quand on relativise et que l'on regarde les contraintes que l'on a, ainsi que les axes d'améliorations possibles :

- 1^{ère} amélioration possible : La caméra. En effet le but de notre application est de pouvoir fonctionner sur n'importe quel ordinateur, sans acheter une caméra spéciale. Pour cela nous utilisons la caméra de base de l'ordinateur et celle-ci est très souvent de mauvaise qualité. Acheter une meilleure caméra nous donnerait des clichés avec une meilleure définition que ceux de notre caméra de base (640 x 480 pixels). Ainsi la position de la pupille, iris, etc... serait beaucoup plus précise. Cependant ceci irait à l'encontre de l'objectif de ce projet.
- 2^{ème} amélioration possible : La détection du visage et des yeux : La détection se fait grâce à 2 classifieurs. L'avantage de ces classifieurs est qu'ils détectent rapidement les parties de l'image nous intéressant, cependant ils ne sont pas très précis. Cette partie pourrait être facilement améliorée en utilisant les model de la série yolo. En effet ils sont bien plus performants. Cependant ce sont des models plutôt lourds ce qui entraînerait un temps de preprocessing plus long et aurait pour conséquence de perdre l'avantage du temps réel avec notre application, ce que nous ne voulons pas.
- 3^{ème} amélioration possible : Le dataset : Les images ne présentent pas nécessairement toutes les possibilités auxquelles notre IA sera confrontée. La répartition des coordonnées regardées sur l'écran peut être hétérogène : Par exemple il peut y avoir plus d'échantillons présents regardant le milieu de l'écran que d'échantillons regardant en haut à droite, ou encore des échantillons n'ayant pas une bonne diversification de luminosité. Ceci peut introduire des biais qui empêcheront l'exactitude des prédictions. Lors de la création du dataset j'ai essayé au maximum de limiter ces possibles erreurs.
- 4^{ème} amélioration possible : Les features : Plus de features pourraient être ajoutés comme notamment l'orientation du visage. En outre certains des features existants ne sont pas très précis comme la distance du visage. En effet cette distance est calculée avec une marge d'erreur pouvant aller jusqu'à 2 centimètres (quand la distance est proche d'un mètre). En effet mesurer la distance sur une image est bien plus complexe qu'il n'y paraît.
- 5^{ème} amélioration possible : Le redimensionnement des images : En effet lors de notre preprocessing, nous redimensionnons les images en 32 x 32 pixels. Cela peut conduire à une perte d'information dans l'image, ce qui détériore la qualité des prédictions. Pour y remédier nous pourrions utiliser des images de plus grande taille, cela nous éviterait de redimensionner les images et ainsi nous n'aurions plus de perte éventuelle d'information. Cependant travailler avec des images de grande taille conduirait à un entraînement du modèle plus

long ainsi que des prédictions plus lentes, ce qui nous éloignerait une nouvelle fois des prédictions en temps réel.

- 6^{ème} amélioration possible : Le model : Notre model a été fait en suivant une approche empirique, rien ne garantit qu'il s'agit du meilleur model possible. Changer un hyperparamètre, ajouter ou enlever une couche de neurones, ou encore avoir de la chance lors de l'entraînement tel que ne pas tomber dans un minimum local ou ne pas converger vers le minimum global. Pour cet axe d'amélioration, il n'y a pas réellement de solution à part continuer à expérimenter.
- 7^{ème} amélioration possible : Utiliser un accélérateur logiciel : J'ai eu l'occasion en entreprise de travailler sur un projet dont le but était de comparer les performances de différents models. J'ai utilisé en particulier onnxruntime qui permettait a minima d'accélérer l'exécution de notre model de 50 % (sur la série des models yolo). Dans notre cas, utiliser cet accélérateur aura un impact peu significatif. En effet notre model est léger, c'est le preprocessing qui prend le plus de temps (0.28s pour une prédiction en comptant le preprocessing avec seulement 0.003s consacrées au model)

Application

L'application Alyes_tracker a pour but d'offrir sans l'achat de matériel supplémentaire la possibilité d'interagir avec son ordinateur ainsi que de pouvoir accéder à des statistiques basées sur cette interaction. On peut facilement imaginer un service commercial utiliser l'application pour identifier les produits qui attirent le regard sur leur site, ou encore repositionner certains éléments d'une page web pour qu'ils soient plus visibles. Initialement l'application devait proposer la possibilité à l'utilisateur de cliquer là où il regarde. Cependant en raison des résultats obtenus, cette fonctionnalité n'a pas été implémentée. Elle aurait permis de pouvoir utiliser l'ordinateur en se passant de souris, d'offrir un accès aux personnes en statut de handicap où encore de ne pas toucher l'ordinateur physiquement et ainsi ne pas contaminer (avec le coronavirus par exemple) les collègues qui seraient amenés à utiliser le même ordinateur.

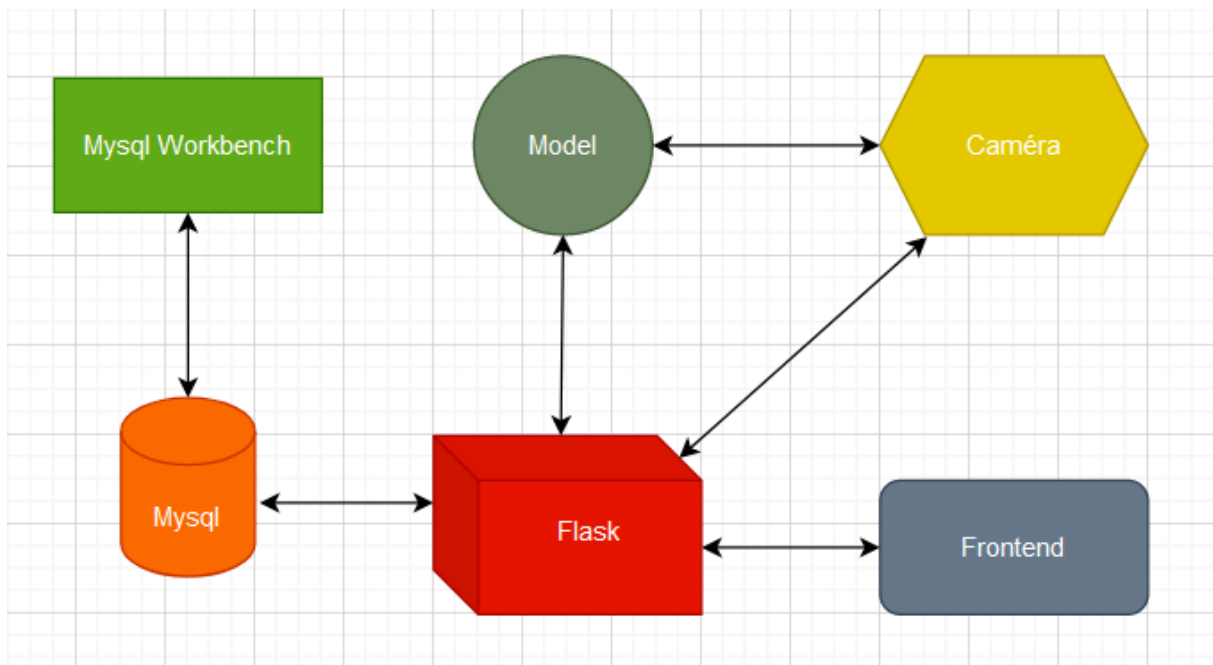


Schéma fonctionnel de notre application

Nous utilisons Mysql Workbench afin de faciliter l'administration de notre base de données Mysql. Celle-ci contient les informations nécessaires à l'authentification, mais aussi des données concernant l'affichage ou encore diverse statistiques. Notre application est basée sur Flask, qui intègre notre base de données ainsi que notre IA. Flask s'occupe de transmettre le contenu visuel. Pour que l'application puisse prédire où nous regardons, Flask est en relation avec notre caméra qui elle-même est en relation avec notre IA.

Base de données

Mysql a été retenue pour notre projet, car il s'agit d'une base de données relationnelle qui a démontré sa fiabilité et on peut facilement trouver de la documentation s'y référant en raison du fait qu'elle est très utilisée.

Dans notre application elle permet de gérer principalement l'authentification et d'enregistrer les diverses statistiques liées aux prédictions. Elle permet aussi de gérer des éléments graphiques ou encore l'éventuelle tarification du service.

Afin de concevoir cette base de données le langage UML a été utilisé, ce qui a permis d'arriver facilement à la fin de cette conception.

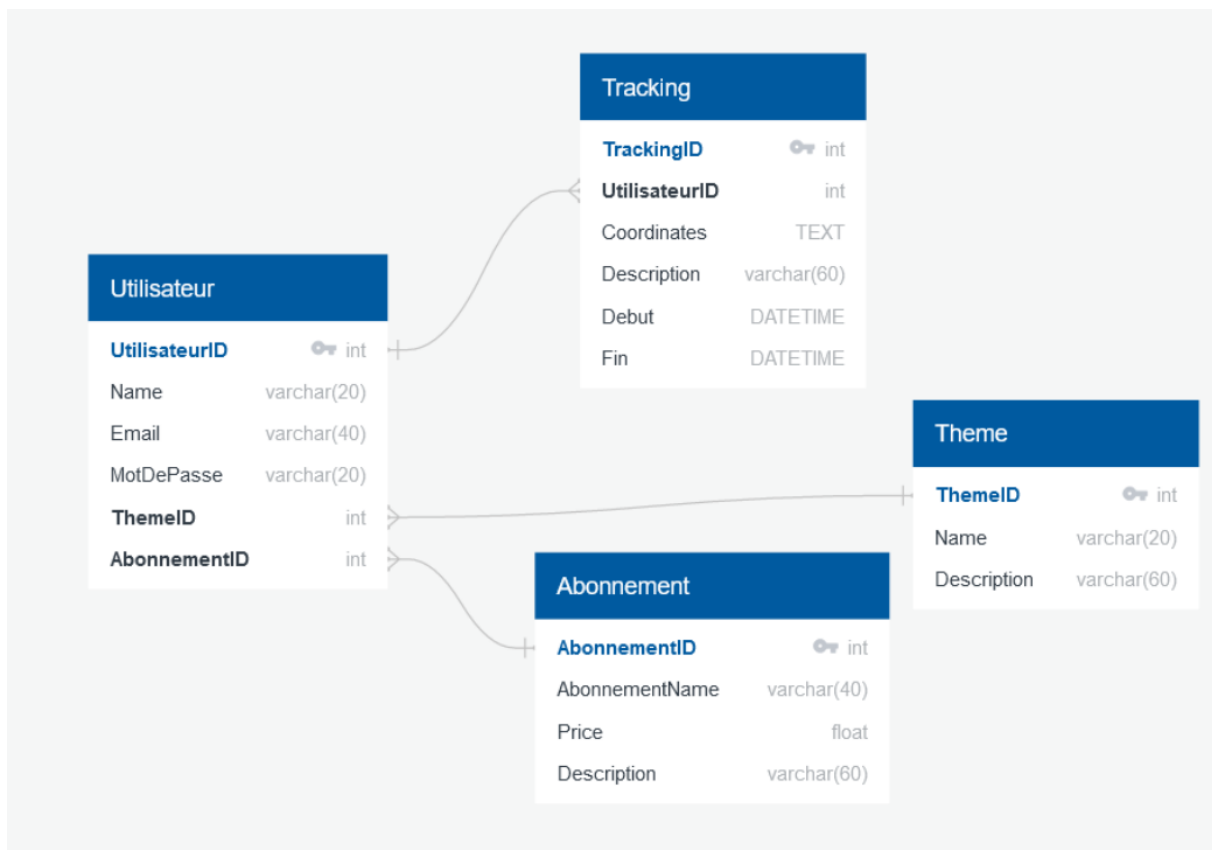


Schéma de la base de données

Notre base de données est composée de 4 tables : Utilisateur, Tracking, Abonnement et Thème. La table Utilisateur enregistre les données d'authentification des utilisateurs de l'application. Cette table est reliée à la table Abonnement par une relation 1, 1 – 1, n (un utilisateur a un seul abonnement, mais un abonnement peut correspondre à plusieurs utilisateurs). La table Abonnement enregistre le tarif que l'utilisateur doit payer pour pouvoir utiliser l'application. La table Thème enregistre le thème choisi par l'utilisateur afin que son UI lui corresponde mieux. Il s'agit ici aussi d'une relation 1, 1 – 1, n. La dernière table, Tracking enregistre, si l'utilisateur le désire, tous les endroits où il a regardé sur une session donnée. Cette table permet à l'utilisateur de faire des statistiques sur les endroits où il a regardé. Cette table est reliée à la table utilisateur par un lien 1, 1 – 1, n. Le champ Coordinates est de type Text car en pratique il s'agira d'enregistrer de très longues numpy array, ainsi il faut un champ capable d'enregistrer beaucoup de caractères.

Exemple d'utilisation de requête sql dans notre application :

Lors de l'inscription, sur la page nous trouvons 2 liste déroulantes proposant les abonnements mais aussi les thèmes disponibles. Afin de pouvoir remplir ces 2 listes déroulantes nous utilisons les requêtes sql suivantes :

```
mycursor.execute("SELECT abonnementID,abonnementName FROM eye_tracker.abonnement")
```

```
mycursor.execute("SELECT ThemeID,Name FROM eye_tracker.theme")
```

Une fois le formulaire d'inscription complété, nous insérons les informations de l'utilisateur dans la base de données afin qu'il puisse s'identifier :

```
query = ("INSERT INTO eye_tracker.Utilisateur (Name, Email, MotDePasse, ThemeID, AbonnementID) VALUES(%s, %s, %s, %s, %s)")
mycursor.execute(query, (name, email, mdp, theme, abonnement))
```

Si la personne désire s'authentifier dans l'application, l'utilisateur va devoir renseigner dans le formulaire son adresse mail ainsi que son mot de passe. S'ils correspondent à ceux déjà présents dans la base de données l'authentification est validée. Pour faire cette vérification nous utilisons la requête suivante :

```
query = ("SELECT email FROM eye_tracker.utilisateur where email = %s and motdepasse= %s")
mycursor.execute(query, (email,mdp))
```

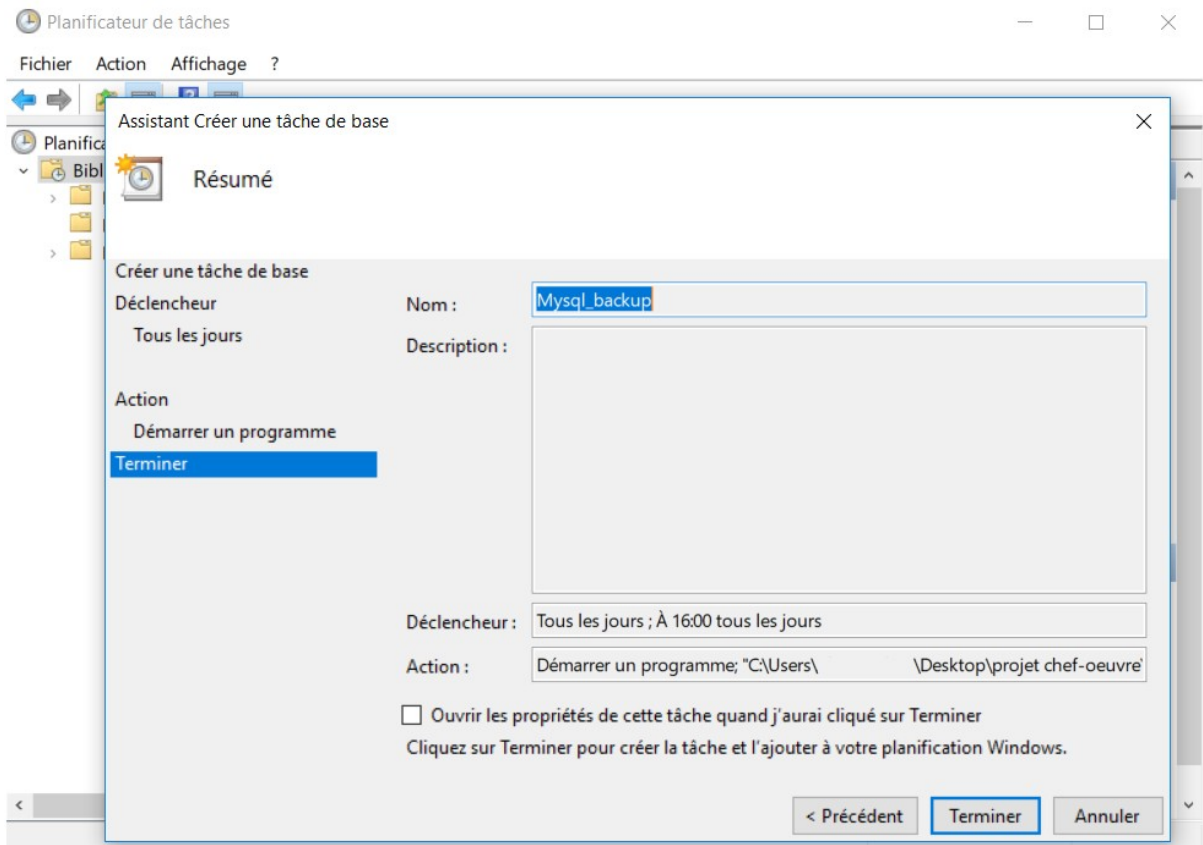
La mise en place de la base de données s'est faite facilement. En effet l'outil de modélisation que j'ai utilisé m'a directement fourni le script sql permettant l'implémentation. Il m'a suffi ensuite d'utiliser Mysql Workbench et d'utiliser la fonctionnalité d'import.

La base de données a dans les applications une place primordiale. En effet il faut s'assurer de ne jamais perdre nos données, car nous avons la responsabilité de leur conservation. Pour ce faire nous mettons en place une méthode de backup automatisé.

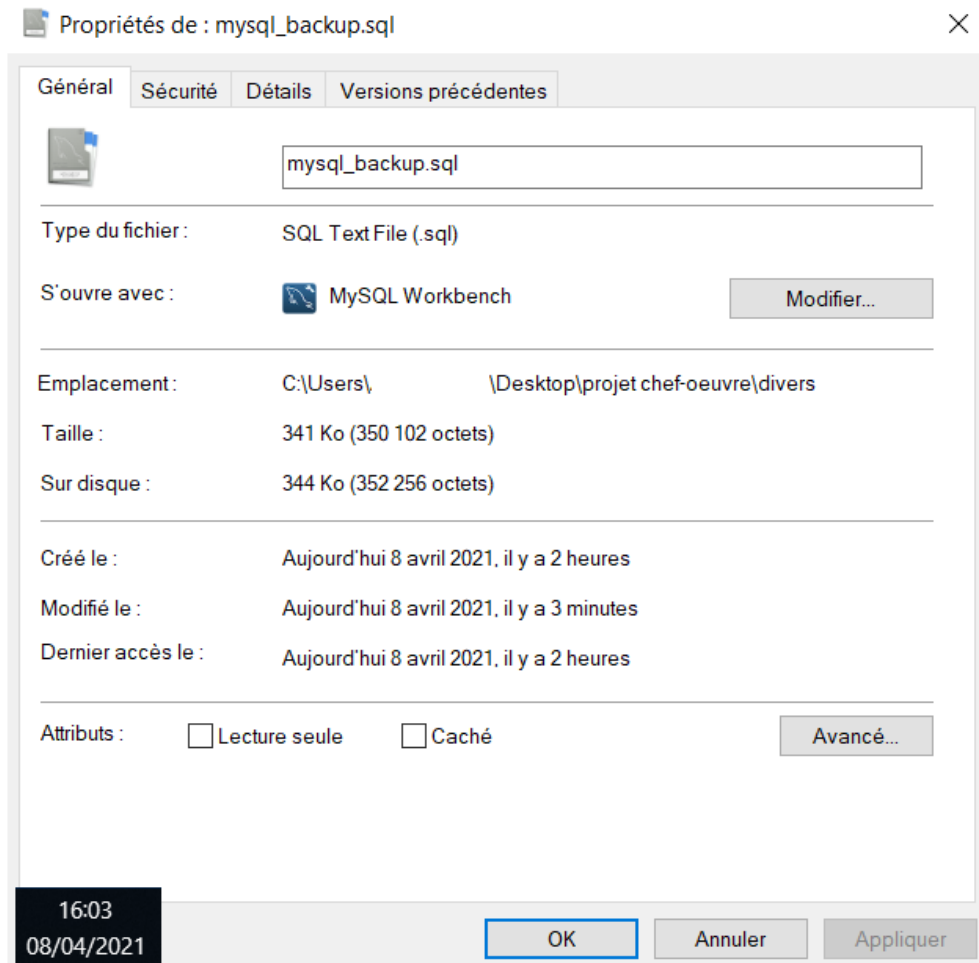
Nous créons un script mysql_backup.bat qui aura pour but de sauvegarder notre base de données :

```
cd C:\Program Files\MySQL\MySQL Server 8.0\bin
mysqldump -hlocalhost -uroot -proot eye_tracker > "C:\Users\Desktop\projet chef-oeuvre\divers\mysql_backup.sql"
```

Afin qu'il soit exécuté tous les jours à la même heure nous créons une task dans le planificateur de tâche de Windows.



Nous avons maintenant une sauvegarde automatisée tous les jours à la même heure.



Les requêtes de notre application s'exécutent rapidement principalement en raison du fait que notre base de données ne contient pas encore beaucoup de données.

#	Time	Action	Message	Duration / Fetch
1	15:19:58	SELECT * FROM eye_trackertracking LIMIT 0, 1000	10 row(s) returned	0.000 sec / 0.000 sec

Si les requêtes s'avèrent trop longues, on pourrait utiliser Explain Analyze afin de connaître le temps passé par Mysql sur chacune des étapes de la requête. Le résultat serait de la forme :

1	-> Table scan on tracking (cost=19.21 rows=10) (actual time=0.017..0.086 rows=10 loops=1)
---	---

On pourrait par exemple rajouter ensuite des index afin que les résultats des requêtes nous parviennent plus rapidement.

Backend

Afin de concevoir notre application Web, nous avons utilisé le Framework Flask. Ce Framework a été retenu principalement pour le fait qu'il utilise le langage python mais aussi car il est plutôt simple d'utilisation.

La partie backend est composée principalement de 2 scripts :

- App.py qui est le script principal de l'application. Ce script contient principalement les routes de l'application, mais aussi les imports des principales librairies, la connexion à la base de données, des requêtes SQL ou encore des calculs nécessaires au fonctionnement de l'application.
- Camera.py qui est le fichier qui gère le preprocessing, la prédiction, ou encore la visualisation de notre IA

```

60 > def index(): ...
64
65 @app.route('/inscription', methods=['GET', 'POST'])
66 def inscription():
67     print(session['user'])
68     print(session['theme'])
69     mycursor = mydb.cursor()
70     abonnement = []
71     theme = []
72     mycursor.execute("SELECT abonnementID,abonnementName FROM eye_tracker.abonnement")
73     for x in mycursor:
74         abonnement.append(x)
75     mycursor.execute("SELECT ThemeID,Name FROM eye_tracker.theme")
76     for x in mycursor:
77         theme.append(x)
78     mycursor.close()
79     return render_template('inscription.html', post=[abonnement,theme])
80
81 @app.route('/video_feed')
82 > def video_feed(): ...
89
90 @app.route('/aiyestracker', methods=['GET', 'POST'])
91 > def aiyestracker(): ...
96
97 @app.route('/resultats', methods=['GET', 'POST'])

```

Exemple de routes dans le script principale App.py

```

7 class VideoCamera(object):
8     def __init__(self):
9         self.video = cv2.VideoCapture(0)
10        self.loaded_model = load_model('./my_eyes.h5', compile = False)
11        self.face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
12        self.eye_cascade = cv2.CascadeClassifier('haarcascade_eye.xml')
13        self.list_heatmap = []
14
15    def __del__(self):
16        self.video.release()
17
18    def get_frame(self,capture_heatmap=False):
19
20        loaded_model = self.loaded_model
21        face_cascade = self.face_cascade
22        eye_cascade = self.eye_cascade
23        list_heatmap = self.list_heatmap
24
25        #start_time = time.time()
26        stop = 0
27        while stop == 0:
28
29            ret, img = self.video.read()
30
31            gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

```

Début du script Camera.py

Nous pouvons grâce à l'appel de ce script directement dans le script principal App.py afficher en temps réel nos prédictions.

```
32 def gen(camera,capture_heatmap=False):
33     list_heatmap = []
34     while True:
35         frame,list_heatmap = camera.get_frame(capture_heatmap)
36         file = open('./static/list_heatmap.txt','w')
37         file.write(str(list_heatmap))
38         file.close()
39         yield (b'--frame\r\n'
40               + b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n\r\n')
```

Cette fonction est ensuite appelée dans la route video_feed :

```
81 @app.route('/video_feed')
82 def video_feed():
83     video_stream = VideoCamera()
84     capture = request.args.get("capture",False)
85     frame = gen(video_stream, capture)
86     return Response(frame,
87                     mimetype='multipart/x-mixed-replace; boundary=frame')
```

Frontend

Le frontend est rendu grâce au module Jinja directement disponible avec Flask., auquel nous ajoutons du CSS afin d'avoir un meilleur rendu visuel. Les différentes pages de notre application sont dans le dossier Template. La route pour accéder à ces pages est incorporée dans notre fichier principal app.py. Les fichier css quant à eux sont dans le dossier static de notre application.

```
▼ templates
  <> aiyestracker.html
  <> confirmation.html
  <> connexion.html
  <> footer.html
  <> header.html
  <> inscription.html
  <> resultats.html
```

```
▼ static
  # style_clair.css
  # style_sombre.css
```

Comme nous pouvons le constater, notre application est composée de 5 pages :

- Aytracker.html est la page à laquelle nous avons accès à nos prédictions en temps réel et qui permet en plus d'enregistrer la session de prédiction.
- Connexion.html est la page qui contient le formulaire qui permet de s'authentifier.

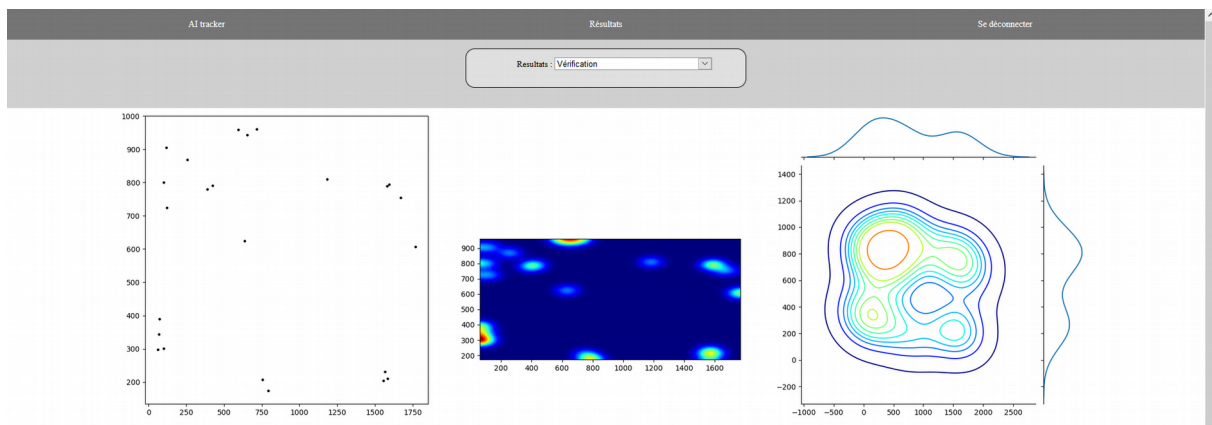
- Inscription.html est la page qui permet de s'inscrire sur l'application grâce au formulaire mis à disposition.
- Résultats.html est la page sur laquelle l'utilisateur peut retrouver les résultats de sa session de prédictions.
- Confirmation.html est la page qui confirme ou infirme que l'inscription, la connexion ou la déconnexion se sont bien déroulées sans encombre.

Enfin nous trouvons le Header qui gère le menu de notre application ainsi que le footer qui s'occupe de notre bas de page.

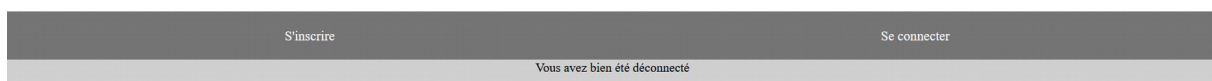
Page d'inscription.

Page de connexion.

Page de prédiction et d'enregistrement de la session de prédiction.



Page de résultats de session.



Page de confirmation (Déconnexion dans ce cas-ci)

Qualité et Monitoring

Afin de s'assurer que l'application fonctionne et ne comprend pas de bugs, il nous a fallu mettre en place différents procédés. Idéalement nous partons d'une version simplifiée de la fonction que nous voulons implémenter, puis nous la complexifions petit à petit jusqu'à atteindre le résultat escompté. Ensuite nous devons vérifier si une fonction ou méthode fait bien ce qu'elle est sensée faire. A cet effet une méthode rapide est de mettre des `print()` des différentes variables de notre algorithme, ce qui permet de voir si leurs valeurs varient bien comme cela se doit durant le processus.

```

160 @app.route('/confirmation_inscription', methods=['GET', 'POST'])
161 def confirmation_inscription():
162     print(session['user'])
163     print(session['theme'])
164     name = request.form['name']
165     email = request.form['email']
166     mdp = request.form['mdp']
167     theme = request.form['theme']
168     abonnement = request.form['abonnement']
169     print(name,email,mdp,theme,abonnement)

```

Ici par exemple on vérifie le contenu de plusieurs variables provenant d'une autre page

On peut aussi faire des tests plus rigoureux, notamment quand le bug est plus difficile à détecter. Pour cela nous utilisons les tests unitaires, dans notre cas nous utilisons Pytest.

```
import pytest
mycursor = mydb.cursor()
mycursor.execute("SELECT ThemeID FROM eye_tracker.theme Where theme.Name = 'Clair' ")
print(mycursor)
for z in mycursor:
    resultat = str(z) [1:-2]
mycursor.close()
assert resultat == 2
```

Une fois exécuté dans notre terminal, avec la commande `pytest theme_test.py` :

```
===== ERRORS =====
ERROR collecting test_theme.py
test_theme.py:20: in <module>
    assert resultat == 2
E   AssertionError: assert '1' == 2
```

Nous constatons une erreur, en effet le thème Clair a pour Id 1 et non 2

Grâce aux tests unitaires nous pouvons ainsi bien suivre le déroulement de notre algorithme que voir où se situent les éventuels bugs.

A cela nous pouvons ajouter les principes du monitoring une fois notre application mise en production. Cela nous permettra d'avoir un suivi des performances, la mesure de la disponibilité, avoir accès aux logs d'activités mais aussi avoir les mesures de changements. L'application comprend d'ailleurs de nombreux `print()` permettant d'obtenir des logs. Elle assure en outre la continuité de service lors de problèmes de connexion avec la base de données par exemple.

Grâce à tout ceci nous pourrions garantir la bonne conservation de notre application dans le temps.

Axes d'amélioration

Bien que notre application ait atteint un stade de développement où elle répond à nos attentes, elle peut être encore améliorée :

- améliorer notre IA, ce point a été traité antérieurement,
- améliorer le visuel de l'application avec notamment du CSS plus abouti.
- améliorer l'expérience utilisateur avec des interfaces plus ergonomiques ainsi que des menus plus intuitifs.
- améliorer la qualité du code, que ce soit sur la lisibilité ou l'optimisation.
- renforcer la sécurité de l'application afin que les données ne puissent être accessibles par une personne mal attentionnée.

Conclusion

Ce projet a été l'occasion pour moi de mettre en pratique tout ce que j'ai appris pendant la formation, ainsi qu'en entreprise. Il m'a aussi permis d'approfondir certains domaines et d'en expérimenter d'autres. La réalisation rigoureuse de ce projet constitue une première expérience complète qui a nécessité la mise en œuvre des compétences nécessaires à l'exercice du métier visé par cette formation. Ce rapport illustre donc la concrétisation de mon entrée dans le monde professionnel. Car selon l'adage «c'est en forgeant que l'on devient forgeron» !