Introduction Générale

Le rythme soutenu et accéléré de la vie mène à des situations de fatigue qui touchent tous les secteurs d'activité. Selon le degré de fatigue que pourrait ressentir l'humain, celui-ci peut être confronté à des dangers et des accidents qui pourraient bouleverser le courant de sa vie. La fatigue est aussi un sentiment de mal-être, un état d'esprit pénible affectant aussi bien la productivité de l'employé que le rendement de l'élève que la vie de famille ou au sein de la société.

Un des objectifs de la communauté scientifique est de relever le défi visant à orienter les avancées technologiques vers le bien-être de l'individu au sein de la société et ce en lui fournissant les services adéquats aux moments opportuns. Pour ce faire, des techniques de détection de malaise, qu'il soit physique ou mental, doivent être offertes.

Tout particulièrement, la fatigue de conducteurs de véhicules cause chaque année un grand nombre d'accidents automobiles. D'après des récentes statistiques chaque année 1 200 décès et 76 000 blessures sont liés à la fatigue du conducteur sur la route. Le développement de technologies de détection ou de prévention de la fatigue au volant est un défi majeur dans le domaine de la sécurité routière et la prévention des accidents, il faut donc élaborer des méthodes pour contrer ses effets. En outre, la fatigue peut concerner les individus dans le milieu professionnel, universitaire ou scolaire. La détection de cette situation pourrait aider à améliorer la productivité et l'efficacité des uns et des autres. Notre objectif dans ce projet est de mettre un prototype de système de détection de la fatigue.

Notre Projet de Conception et de Développement est centré sur la localisation de visage par un auto-algorithme de traitement d'image développé. Une fois la position du visage est déterminée, le système se charge de déterminer si ce visage est fatigué ou non.

Le rapport est articulé autour de 4 chapitres. Le premier est étude préalable,le deuxieme est analyse et spécification des besoins,le troisieme est conception d'un détecteur de fatigue et le quatrieme est réalisation.

Chapitre 1

Etude préalable

Durant cette phase, nous allons présenter la notion de la détection de fatigue ainsi qu'une vue générale sur les fonctions assurées par cette techniques et leurs usages actuels dans le monde.

1.1 Etude théorique

Cette section forme la base théorique de notre projet. Nous consacrons cette partie à aborder de plus près le contexte du projet en détaillant les techniques et les méthodes.

1.1.1 Intelligence Artificielle

L'intelligence artificielle (IA) se réfère à la simulation de l'intelligence humaine dans des machines programmées pour penser comme les humains et imiter leurs actions. Le terme peut également être appliqué à toute machine qui présente des traits associés à un esprit humain tels que l'apprentissage et la résolution de problèmes. Les applications de l'intelligence artificielle sont infinies. La technologie peut être appliquée à de nombreux secteurs et industries différents. L'IA est testée et utilisée dans l'industrie de santé ,l'industrie d'électronique etc [B1].

1.1.2 Apprentissage Automatique (Machine Learning)

Machine Learning ou l'Apprentissage Automatique est un domaine de l'intelligence artificielle (IA) qui concerne la conception, l'analyse, le développement et l'implémentation de méthodes permettant à un ordinateur d'évoluer par un processus systématique.
Le Machine Learning permet à un programme informatique d'effectuer une tâche pour laquelle il n'est pas programmé explicitement ,il est plutôt programmé pour apprendre à la faire c'est à dire qu'on donne au programme de nombreuses données et il apprend à partir de ces données[B1].

1.1.3 Apprentissage profond (Deep Learning)

Le Deep Learning, ou l'apprentissage profond, est une approche d'intelligence artificielle qui imite le fonctionnement du cerveau humain dans le traitement des données et la création de modèles à utiliser dans la prise de décision. Le deep learning est un sous-ensemble du machine learning qui dispose de réseaux capables d'apprendre de manière non supervisée à partir de données non structurées ou non étiquetées. Le deep learning a provoqué une explosion des données sous toutes leurs formes et dans toutes les régions du monde. Ces données, appelées simplement big data, sont extraites de sources telles que les médias sociaux, les moteurs de recherche Internet, les plateformes de commerce électronique et les cinémas en ligne, entre autres. Cette énorme quantité de données est facilement accessible et peut être partagée via des applications avancées comme le cloud computing [B1].

1.1.3.1 Convolutional Neural Network (CNN)

Convolutional Neural Network est un réseau neuronal à action directe qui est généralement utilisé pour analyser des images visuelles en traitant des données avec une topologie en forme de grille. Il est également connu sous le nom de ConvNet. Convolutional Neural Network est utilisé pour détecter et classer les objets dans une image[8].

L'architecture de CNN se compose d'une couche d'entrée (Fully connected input layer) et d'une couche de sortie (Fully connected output layer) séparées par trois types de couches : couche convolutionnelle (Convolutional layer), couche Pooling(Pooling layer) et couche entièrement connectée (Fully connected layer) (cf. figure 1.1).

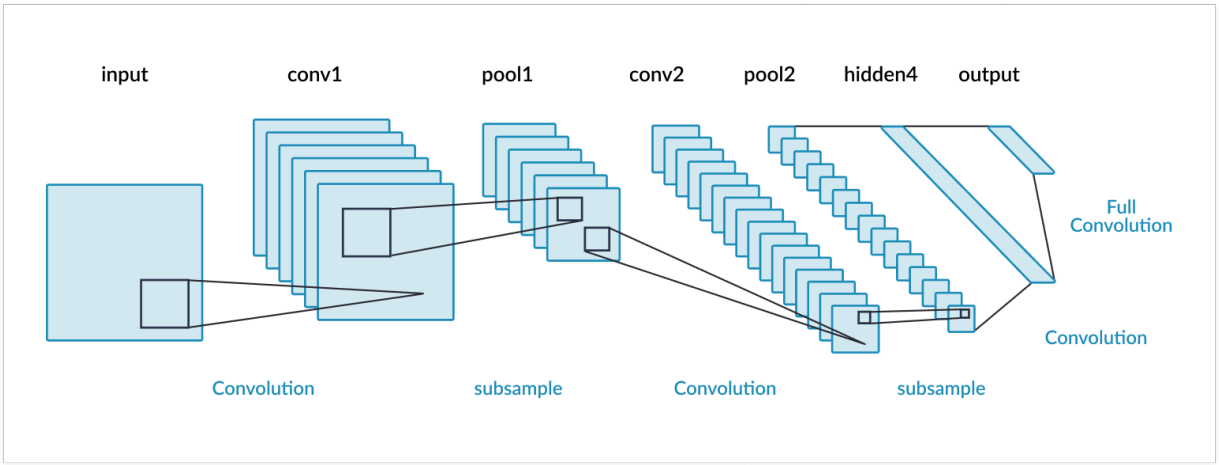

FIGURE 1.1 – Architecture du CNN [1]

— **Couche convolutionnelle** (Convolutional layer) :
La couche de convolution est la composante principales des réseaux de neurones

convolutifs. Son but est de repérer la présence d'un ensemble de caractéristiques (que l'on appellera features tout au long du rapport) dans les images reçues en entrée. Pour cela, on réalise un filtrage par convolution, le principe étant de faire "glisser" une fenêtre représentant la feature sur l'image, et de calculer le produit de convolution entre la feature et chaque portion de l'image balayée (cf. figure 1.2).

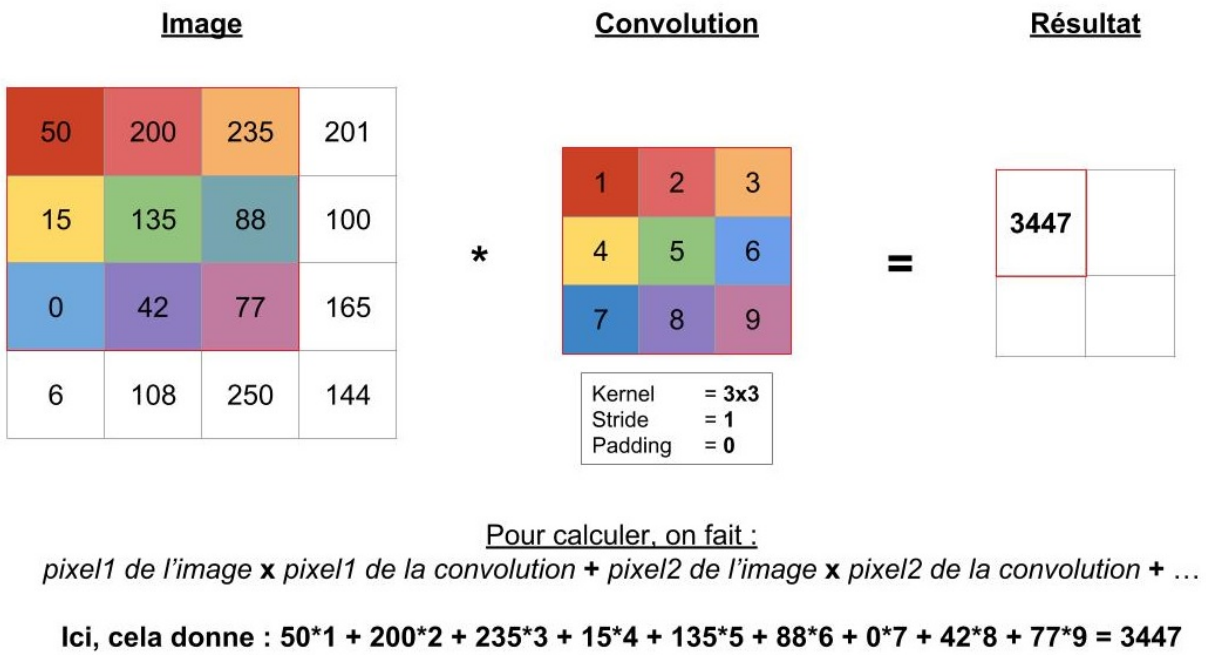

FIGURE 1.2 – Exemple du principe de la couche convolutionnelle [2]

— **Couche de pooling** (Pooling Layer) : Sa fonction est de réduire progressivement la taille spatiale de la représentation pour réduire la quantité de paramètres et de calcul dans le réseau. La couche de regroupement fonctionne indépendamment sur chaque carte d'entités (cf. figure 1.3).

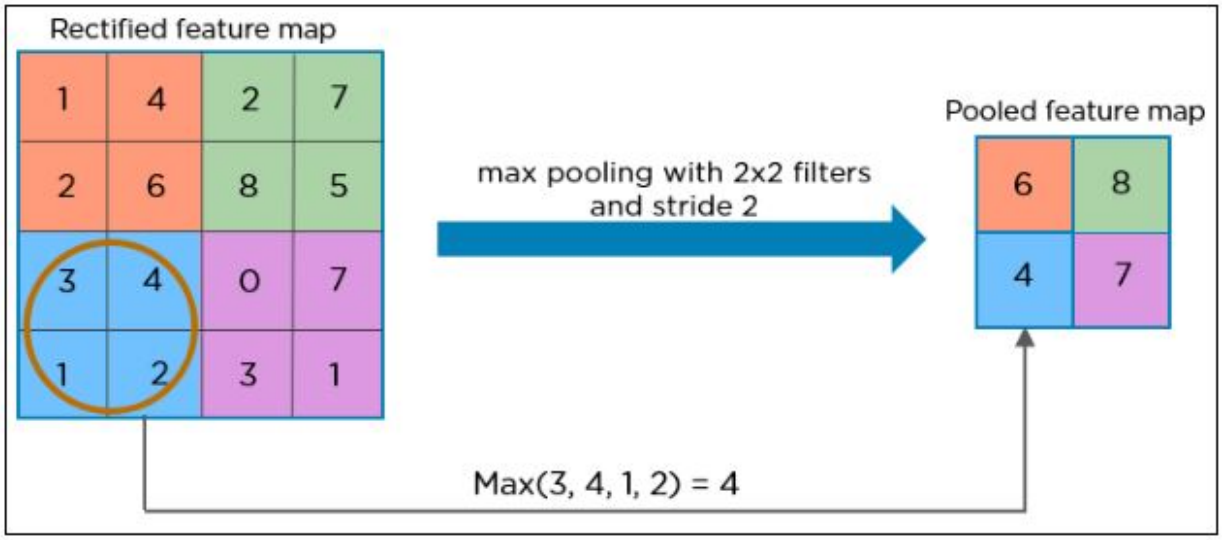

FIGURE 1.3 – Exemple de principe de la couche de pooling [3]

— **Couche entièrement connectée** (Fully connected layer) :
Les couches entièrement connectées sont une composante essentielle des réseaux neuronaux convolutionnels (CNN), qui se sont révélés très efficaces pour reconnaître et classer les images pour la vision par ordinateur (c'est la couche de classification) (cf. figure 1.4).

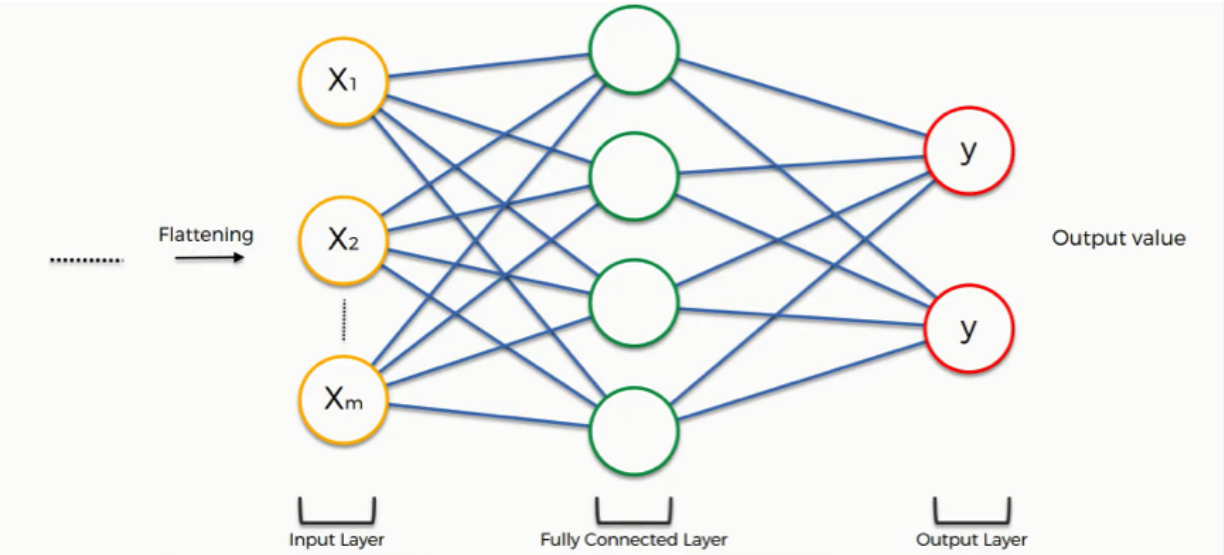

FIGURE 1.4 – Exemple d'une couche entièrement connectée [4]

1.1.3.2 Multi-task Cascaded Convolutional Neural Networks(MTCNN)

MTCNN (Multi-task Cascaded Convolutional Neural Networks) est un algorithme qui détecte les visages individuels, localise les repères faciaux (c'est-à-dire deux yeux, le nez et les extrémités de la bouche) et dessine un box qui contient le visage.

FIGURE 1.5 – Exemple de MTCNN Face Detector [5]

1.1.4 Transfer learning

Lorsque nous avons un ensemble de données relativement petit, une technique super efficace consiste à utiliser Transfer Learning où nous utilisons un modèle pré-formé. Ce modèle a été formé sur un ensemble de données extrêmement volumineux, et nous pourrions transférer des poids qui ont été appris grâce à des centaines d'heures de formation sur plusieurs GPU haute puissance[9].

L'apprentissage par transfert est devenu extrêmement populaire car il réduit considérablement le temps de formation et nécessite beaucoup moins de données sur lesquelles s'entraîner pour augmenter les performances[B2].

1.1.5 Cropping

Le recadrage consiste à supprimer les zones externes indésirables d'une image photographique ou illustrée. Le processus consiste généralement à supprimer les déchets étrangers de l'image, pour améliorer son cadrage ou pour accentuer ou isoler le sujet de son arrière-plan.

1.1.6 Processus de détection de la fatigue

Le processus de détection de la fatigue comprend la détection et le suivi du visage, la détection de l'emplacement de l'œil humain, le suivi de l'œil humain, la reconnaissance de l'état des yeux et les tests de fatigue d'un individu. Les aspects clés du système de détection comprend la détection et la localisation des yeux humains et les tests de fatigue d'un individu. Le système utilise une petite caméra qui pointe directement vers le visage d'un personne et surveille ces yeux afin de détecter la fatigue.
Ce projet décrit comment trouver le visage et aussi comment déterminer à partir de ce visage si le personne est fatiguée ou pas.

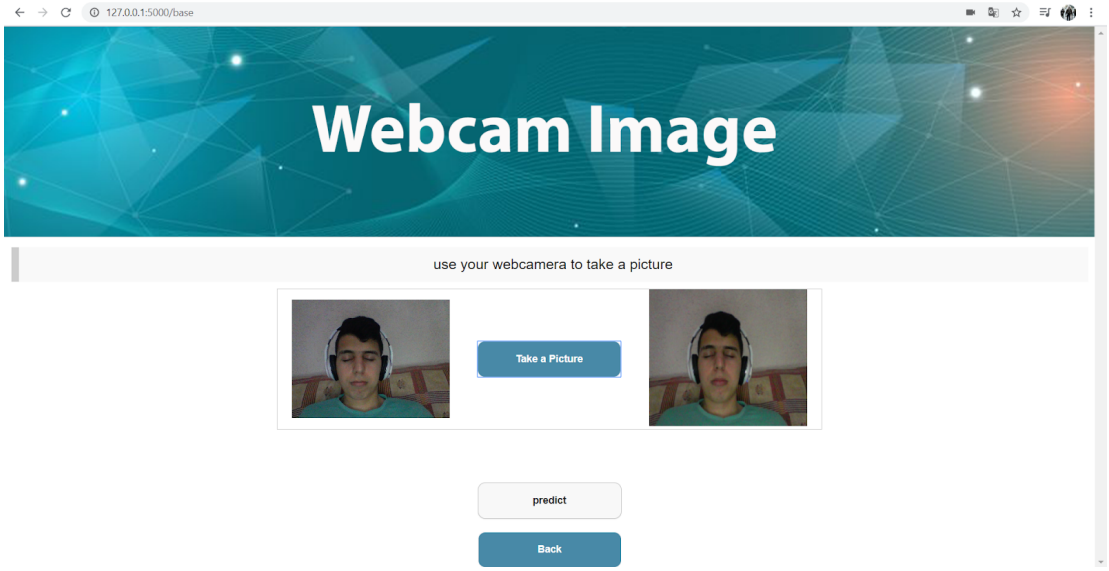

FIGURE 1.6 – Saisie de l'image

Résultat :

FIGURE 1.7 – Résultat de détection

1.2 Étude de l'existant

Afin d'atteindre les objectifs de notre projet, Il est nécessaire d'effectuer une étude des systèmes existants afin d'étudier les systèmes déjà développés et de mettre en relief leurs limites. Après une étude approfondie sur l'existence des applications de détection de la fatigue, nous avons constaté que de nombreux chercheurs ont travaillé ces dernières années sur des systèmes de détection de fatigue et plusieurs innovations ont été lancé :

Il existe un «Driver Fatigue Monitor System» créé par Attention Technologies, Inc. Il est basé sur le PERCLOS (PERcent of the Time Eyelids are CLOSed) mesure de la fatigue du conducteur. Ils ont découvert que le système basé sur PERCLOS fonctionne mais présente quelques défauts.la figure 1.8 présente ce système[B3].

Plusieure entreprises du secteur Industrie automobile comme Bosch , Mercedes ,Volvo et Volkswagen ont développé un systeme de détection de la fatigue qui pourrait se faire à l'aide d'une caméra intégrée dans le tableau de bord et qui analysera le comportement du conducteur (fermeture des oeils, inclinaison de la tête, direction du regard, . . .).

FIGURE 1.8 – Système de détection de fatigue Créé par Attention Technologie [6]

1.3 Solution proposée

Notre solution proposée consiste à développer une application web à l'aide de Flask [B4] et d'algorithmes d'apprentissage automatique supervisé. Notre application assurera la classification des photos en deux classes fatigue et non fatigue. Il offre également une interface sophistiquée pour permettre au utilisateur de choisir le mode de saisie d'image .Il existe 3 modes, le premier est le saisie à partir de votre appareil, le deuxième est prendre une photo avec la webcam et le troisième consiste à lancer un streaming vidéo et détecter chaque frame de cette vidéo fatigue ou pas.

1.4 Conclusion

Dans ce premier chapitre, nous avons commencé, tout d'abord, par mettre l'accent sur les méthodes et les techniques afin de mieux comprendre notre projet. Ceci prépare le

terrain à une étude profonde des besoins de notre système qui seront étudiés et modélisés dans le chapitre suivant.

Chapitre 2

Analyse et spécification des besoins

La partie analyse et spécification de besoin est une partie évidente afin d'avoir une compréhension claire et précise de toutes les caractéristiques du projet et de respecter les critères fixés par le cahier des charges.

Ainsi, dans cette section, nous commençons d'abord par un analyse des besoins auxquelles l'application doit réagir,passant ensuite à la spécification de ces besoins à travers l'introduction des acteurs et les diagrammes de cas d'utilisation relatifs à ces acteurs.

2.1 Analyse des besoins

Dans cette étape on va déterminer les différents besoins que notre application doit remplir.

2.1.1 Besoin fonctionnels

Le système détecte si un individu est fatigué ou pas selon une approche de classification supervisée.

2.1.2 Besoins non fonctionnels

— **Rapidité :**

La durée de réponse des traitements s'approche le plus possible du temps réel

— **Fiabilité** :

Fréquence d'apparition d'erreurs faible (good accuracy)

— **Disponibilité :**

Notre système doit être toujours disponible pour un utilisateur.

2.2 Spécification des besoins

Au niveau de cette partie nous allons commencer par définir les acteurs qui sont en relation avec le système. On établit ensuite le diagramme des cas d'utilisation et les diagrammes de séquence de notre application conforme au langage de modélisation unifié (UML).Ce dernier donne une vision globale du comportement fonctionnel de la solution.

2.2.1 Définition des acteurs

Un acteur est une entité externe qui interagit avec le système afin d'accomplir un objectif. Les acteurs de notre système sont : Un individu pouvant être

- un conducteur de véhicule
- un employé
- un étudiant
- un élève
- un pilote

2.2.2 Langage de modélisation

UML « Langage de modélisation unifié ». L'UML est un langage visuel constitué d'un ensemble des diagrammes, qui donnent chacun une vision alternative du projet à traiter.

2.2.2.1 Diagramme de cas d'utilisation global

Un diagramme de cas d'utilisation est une représentation graphique des interactions entre les éléments d'un système.Il est généralement simple. Il ne montre pas le détail des cas d'utilisation.il ne résume que certaines des relations entre les cas d'utilisation, les acteurs et les systèmes.Il ne montre pas l'ordre dans lequel les étapes sont effectuées pour atteindre les objectifs de chaque cas d'utilisation. Il ne nécessite aucune connaissance informatique et l'idéal serait qu'il soit réalisé par le client.
Entrer une image : Ce cas d'utilisation est associé à un individu . Ce dernier peut prendre une photo avec webcam, importer une photo et lancer une vidéo stream.La figure 2.1 représente le diagramme de cas d'utilisation.

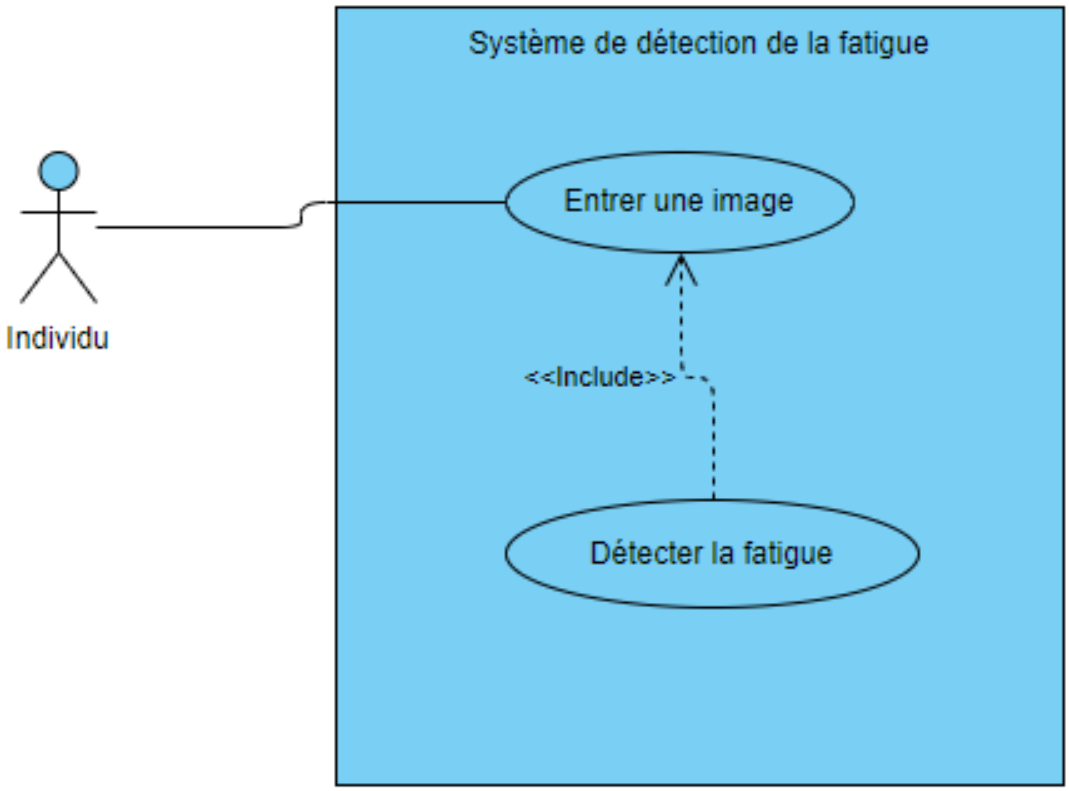

FIGURE 2.1 – Diagramme de cas d'utilisation

2.2.2.2 Diagramme de séquence système pour le scénario "détection de la fatigue "

Dans le diagramme de séquence nous avons introduit comment un individu peut interagir avec notre système. tout d'abord l'utilisateur devrait saisir une image puis le système va traiter cette image et affiche le résultat sur l'interface de l'application web .la figure 2.2 présente le diagramme de séquence système.

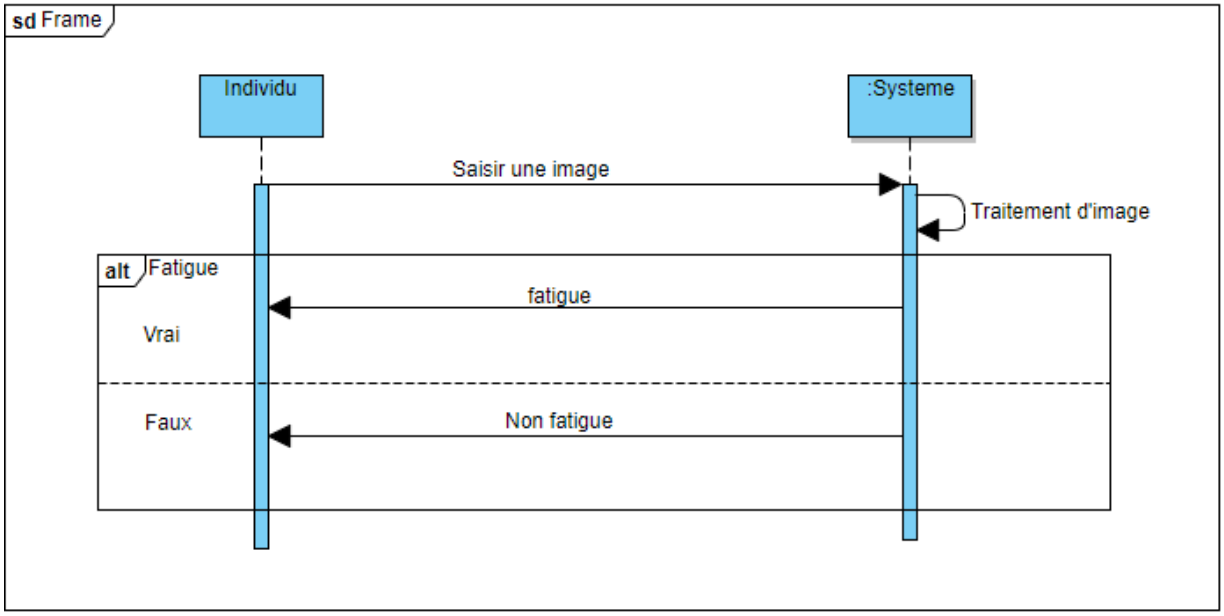

FIGURE 2.2 – Diagramme de séquence pour la détection de la fatigue

2.3 Conclusion

Au cours de ce chapitre, nous avons précisé et analysé les besoins fonctionnels et non fonctionnels que notre application satisfera et nous avons présenté les principaux cas d'utilisation et scénarios de notre projet. Dans le chapitre suivant, nous allons plus loin dans la conception de notre application.

Chapitre 3

Conception d'un détecteur de fatigue

Dans ce chapitre ,nous allons présenter la partie conception et architecture .dans la première partie ,nous allons aborder la conception architecture globale de notre application.Dans la deuxième partie,nous allons donner une analyse conceptuelle détaillée

3.1 Conception globale

Dans cette partie nous allons présenter les architectures matérielle et logiciel que nous avons choisies pour notre application.

3.1.1 Architecture physique

L'architecture physique la plus appropriée que nous considérons pour notre application est l'architecture à deux niveaux elle est appelée aussi architecture client-serveur. cette architecture est composée de deux couches client et serveur . le Client envoie une requête au serveur et le serveur reçoit cette requête et fait un traitement puis retourner une réponse au client comme l'indique la figure 3.1

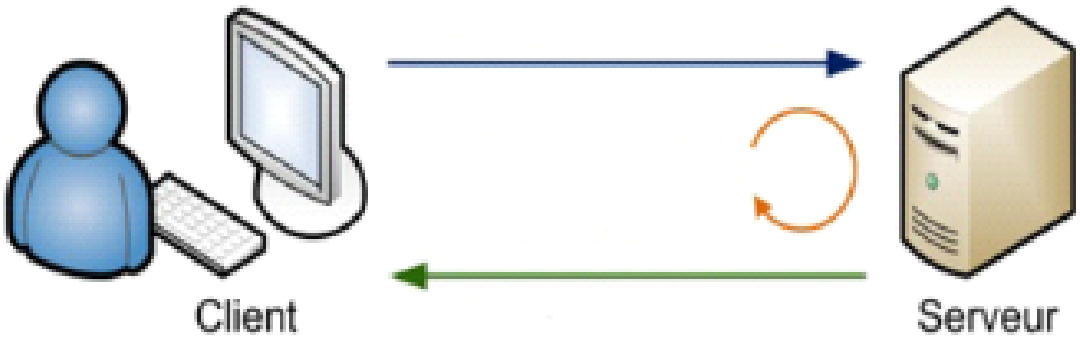

FIGURE 3.1 – Architecture 2-tiers[10]

3.1.2 Architecture logique

Le modèle MVC, qui signifie Modèle-Vue-Contrôleur,est une architecture de développement conçue pour séparer le code source en 3 modules données,présentation et traitement . Cette architecture est la plupart du temps utilisé dans les applications Web, mais elle existe également dans les applications lourdes. La figure 3.2 présente le modèle MVC

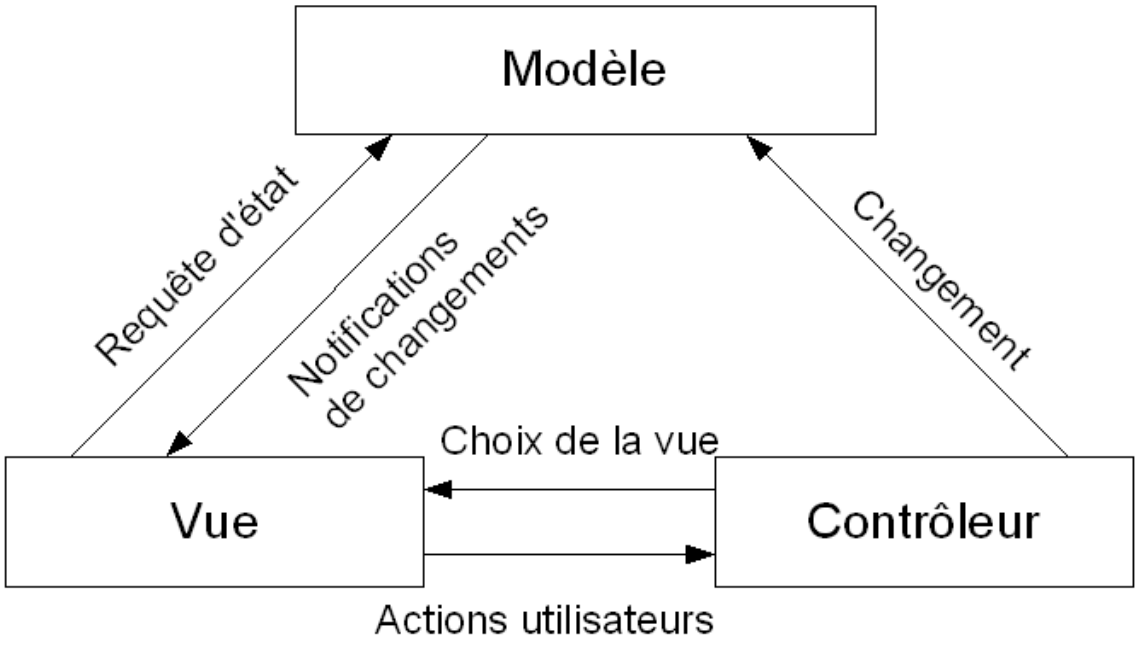

FIGURE 3.2 – Modèle MVC [11]

— **le Modèle** : représente et structure l'ensemble des composants qui vont être manipulés et qui doivent mettre les résultats de l'appel à la disposition de la Vue.
— **la Vue** : représente l'interface utilisateur (IHM).
— **le Contrôleur** : gère la synchronisation entre la Vue et le Modèle. Le contrôleur réagit aux actions de l'utilisateur en effectuant les actions nécessaires sur le Modèle. Le Contrôleur surveille les modifications du modèle et informe la Vue des mises à jour nécessaires.

Avantages :
-Séparation entre les données, la représentation et comportement
-Possibilités de vues différentes pour un même modèle
-Réutilisabilité

Inconvénients :
-Mise en place complexe dans le cas d'applications importantes

3.2 Conception détaillée

Dans cette partie, nous allons détailler la conception en utilisant des pipelines. Nous allons commencer par présenter l'architecture du modèle InceptionV3 qu'on a utilisé comme pré-trained modèle puis nous allons présenter un pipeline de préparation de données et un pipeline de formation de modèle et on finit par le pipeline de classification des images en deux classes fatigue et non fatigue

3.2.1 Architecture du modèle InceptionV3

Inception V3 de Google est la 3éme version d'une série d'architectures convolutionnelles d'apprentissage profond.

Inception V3 est un modèle de reconnaissance d'images qui a été formé à l'aide d'un ensemble de données de 1 000 classes à partir de l'ensemble de données ImageNet d'origine qui comporte plus qu'un million d'images d'entraînement.Ce modèle est l'aboutissement de nombreuses idées développées par plusieurs chercheurs au fil des ans.

Le modèle lui-même est constitué de 42 layers incluant des layers de convolutions, average pooling, maximum pooling, concatenations et fully connected et il utilise la fonction softmax pour calculer la perte .[13]

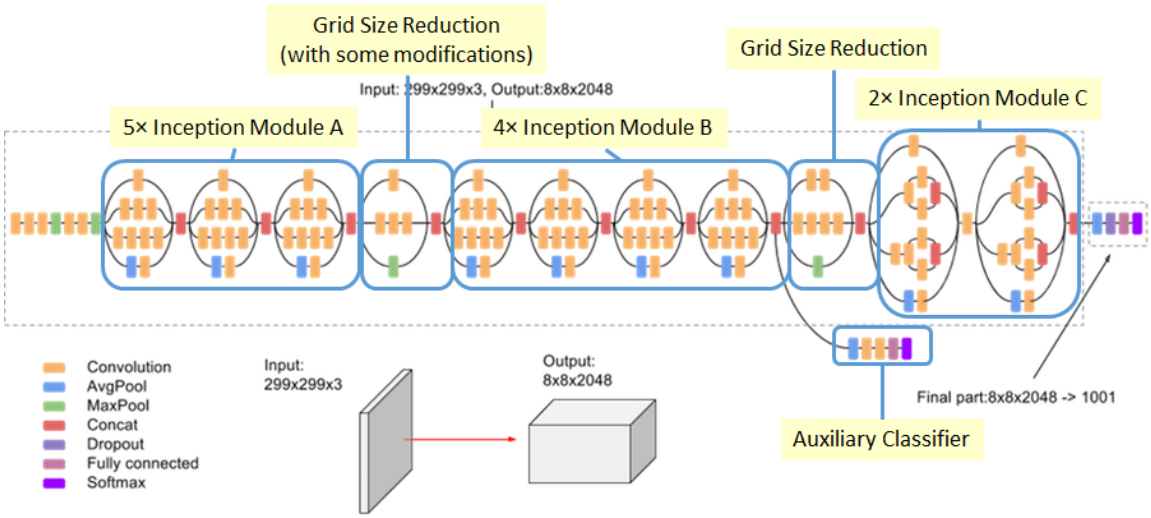

FIGURE 3.3 – Architecture inceptionV3 [12]

3.2.2 Préparation des données

sous cette partie, nous allons montrer étape par étape comment nous avons fait le prétraitement de données

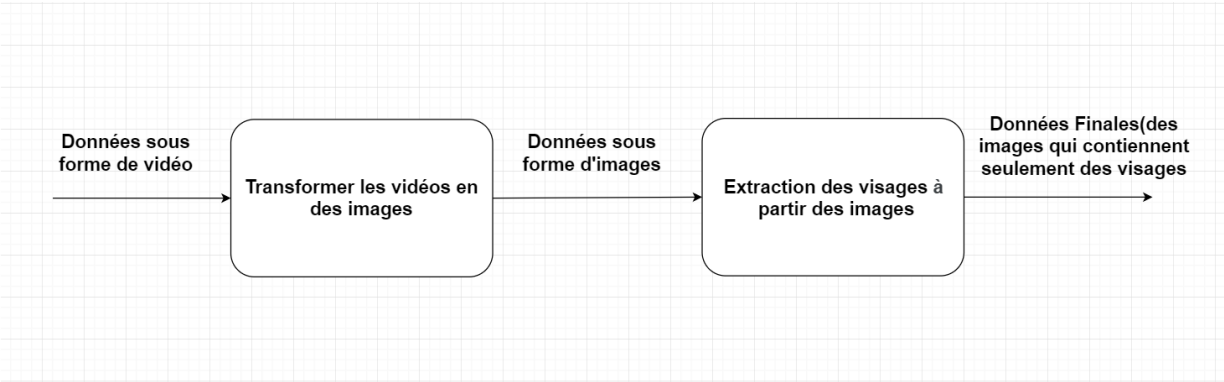

FIGURE 3.4 – Préparation des données pipeline.

3.2.3 Prédiction

Cette figure montre comment fonctionne le processus de prédiction :

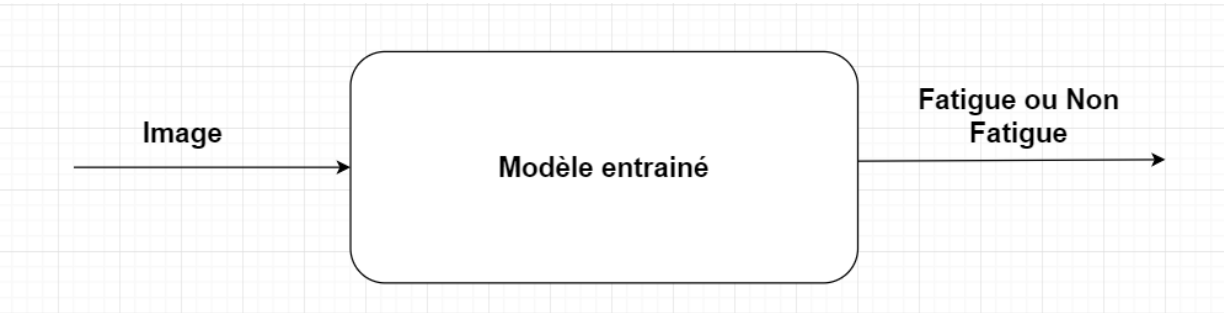

FIGURE 3.5 – Prédiction.

3.3 Conclusion

Dans ce chapitre, nous avons exposé la structure globale de notre application. Nous avons proposé les architectures logiques et physiques, ainsi que la conception détaillée de notre système. Dans le prochain chapitre, nous détaillons les outils et les langages utilisés ainsi que l'interface de notre application.

Chapitre 4

Réalisation

Après avoir détaillé la partie conceptuelle du notre projet, nous allons décrire l'aspect implémentation. Nous commençons par présenter l'environnement matériel,la plateforme logicielle utilisée.Ensuite nous expliquerons les différentes phases de notre projet et fournirons des captures d'écran du travail réalisé.

4.1 Environnement de travail

Dans cette partie nous allons présenter l'environnement matériel et logiciels utilisé pour implémenter notre solution.

4.1.1 Environnement matériel

Tout le long de ce travail nous avons utilisés l'ensemble d'outils matériels suivant :

— Ordinateur :DELL G5 5587
  Processeur : Intel Core i7-8750H 8ème Génération
  Mémoire physique (RAM) installé : 16,00 Go
  Disque dur :1 TO
  SE :Windows

— Ordinateur : ASUS X555L
  Processeur : Intel Core i7-4510U 4ème Génération
  Mémoire physique (RAM) installé : 8,00 Go
  Disque dur :1 TO
  SE :Windows

— Ordinateur : HP pavillon
  Processeur : Intel Core i5-8250 U 8ème Génération
  Mémoire physique (RAM) installé : 16,00 Go

Disque dur :1 TO
SE :Windows

4.1.2 Environnement logiciel

Dans cette section, nous présenterons l'environnement logiciel utilisé pour implémenter notre application :

— **Jupyter Notebook**
Le Jupyter Notebook est une application Web open source qui permet de créer et de partager des documents contenant du code vivant, des équations, des visualisations et du texte explicatif. Cette application a été choisie afin de faciliter le test d'algorithmes avant de les intégrer dans notre projet.

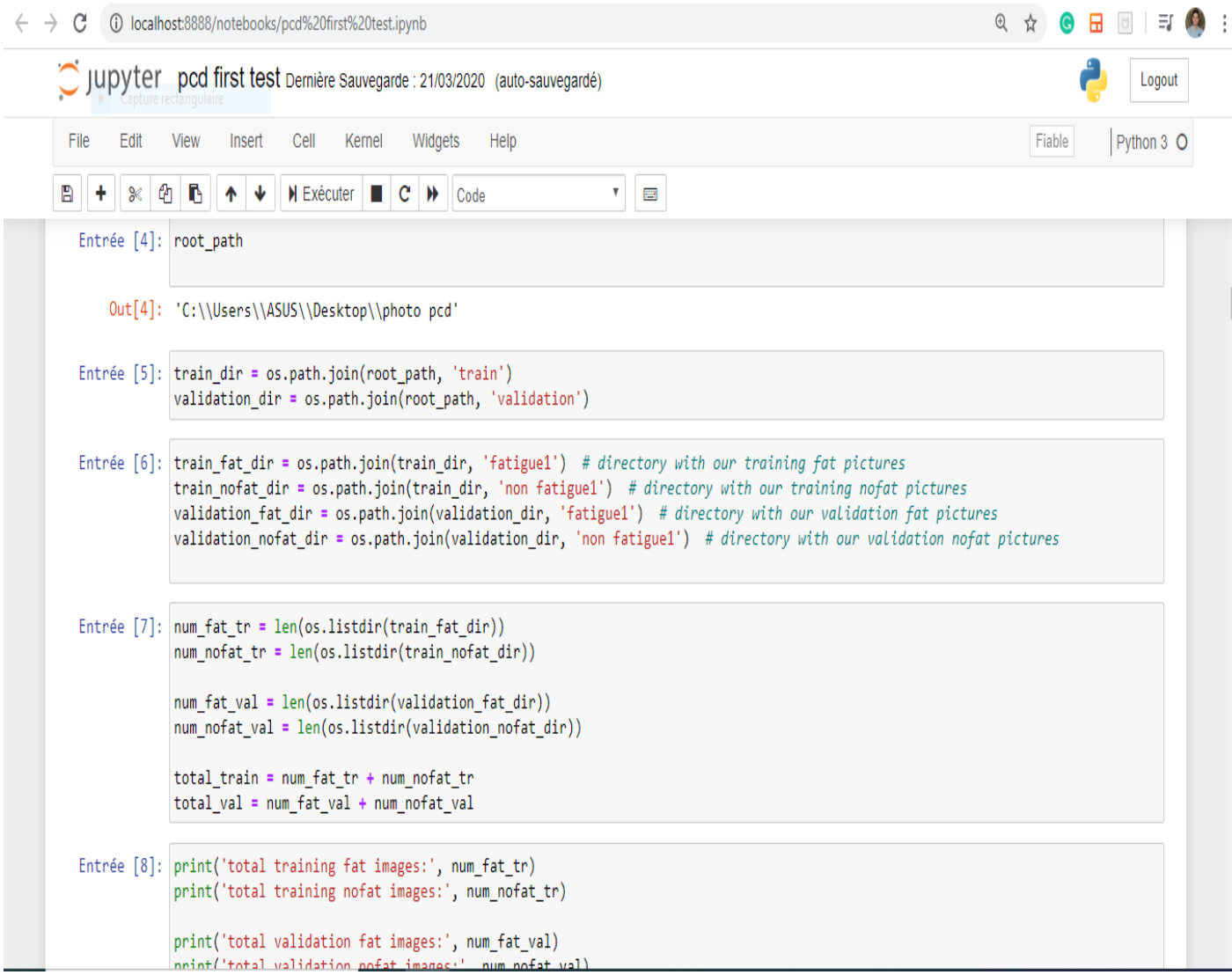

FIGURE 4.1 – Jupyter Notebook.

— **Spyder**
Spyder est un environnement de développement intégré (IDE) multiplateforme open source pour la programmation scientifique en langage Python.

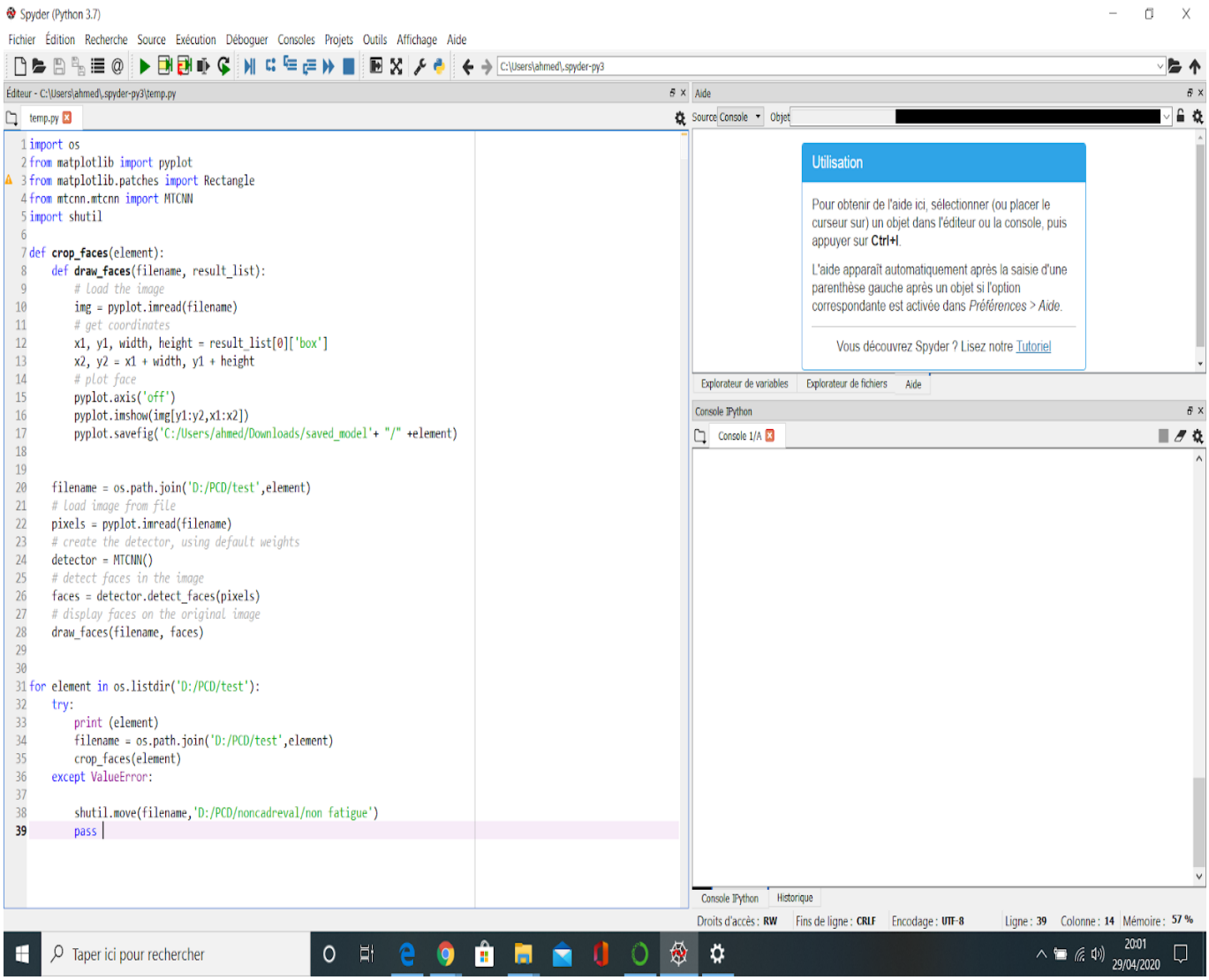

FIGURE 4.2 – Spyder.

— **Google Colab**
Google Colab est un service cloud , vous pouvez développer en langage de programmation Python, développer des applications de deep learning à l'aide de bibliothèques populaires telles que Keras, TensorFlow, PyTorch et OpenCV.

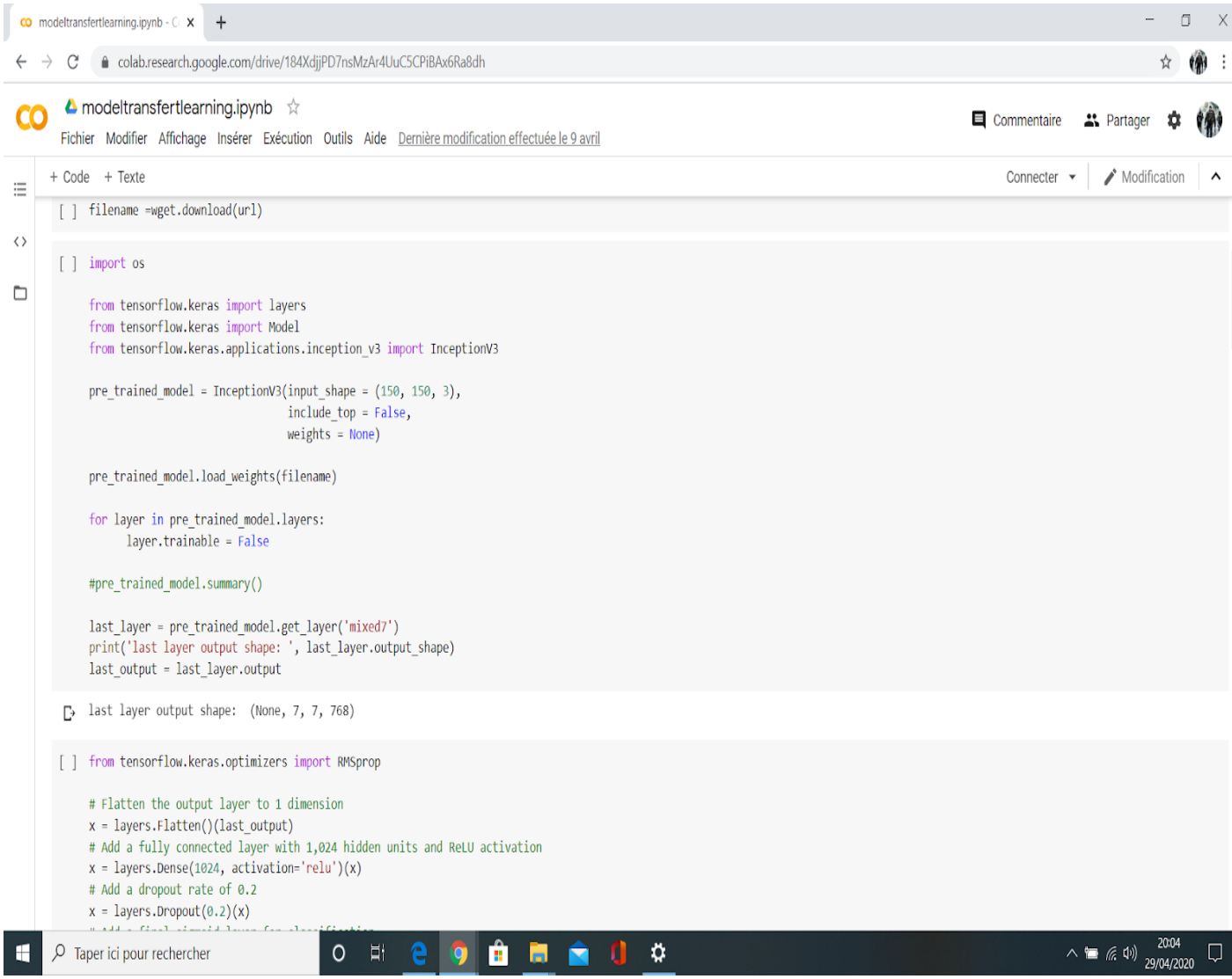

FIGURE 4.3 – Google colab.

4.1.3 Justification des choix technique

Dans cette section, nous allons effectuer une étude sur les Frameworks et bibliothèques qu'on a utilisé pour ce projet ainsi les raisons du choix de ces technologies :

— **Langages de programmation**

Python :

Python est un langage de programmation interprété, orienté objet,multiplateforme et multi-paradigme . C'est un langage de programmation polyvalent et populaire qui peut être utilisé pour tout, du développement Web au développement de logiciels et aux machine learning et deep learning applications.

Python a été notre choix pour mettre en œuvre ce projet car il est facile à utiliser

aussi Python nous permet de construire nos modèles d'apprentissage automatique très facilement grâce aux nombreux frameworks, bibliothèques et extensions qui simplifient l'implémentation de différentes fonctionnalités.

— **Frameworks**
1. Flask :
Flask est un micro framework de développement web écrit en Python.Il est classé un micro framework parce qu'il ne nécessite ni outils ni bibliothèques. il se différencie par sa légèreté permettant de disposer d'une solide base de développement tout en conservant la flexibilité et la lisibilité du langage de programmation Python.
Cependant, la caractéristique la plus attrayante qui nous a fait choisir le flask comme notre framework est le fait qu'il est facile à utiliser et à apprendre grâce à sa documentation complète et sa grande flexibilité.

2.Tensorflow :
TensorFlow est un outil open source d'apprentissage automatique développé par Google.Il est également utilisée pour des applications d'apprentissage automatique telles que les réseaux de neurones.
Il offre une approche abstraite de très haut niveau pour organiser la programmation numérique de bas niveau.

— **Bibliothèques**
1.keras :
keras est une bibliothèque machine learning open source écrite en python qui permet d'interagir avec les algorithmes de réseaux de neurones profonds.
Dans notre cas nous avons utilisé keras pour implémenter un modèle CNN qui fait la classification d'une image en deux classes fatigue ou bien non fatigue.

2.OpenCV :
OpenCV est une bibliothèque graphique.Elle est spécialisée dans le traitement d'images et des vidéos.
On a utilisé la bibliothèque OpenCv pour l'extraction des images à partir d'une vidéo.

4.2 Aperçu sur le travail réalisé

Après avoir préparé le terrain pour la mise en œuvre, nous entamons la phase de mise en œuvre qui se décompose en 3 parties. Premièrement, nous avons collecté les données à utiliser pour la construction de notre modèle de classification. Ensuite, nous préparons

le terrain pour la phase de prédiction en réalisant les modèles appropriés. Enfin, nous présenterons les interfaces de notre application.

4.2.1 Préparation de dataset

La qualité de notre modèle dépend de la quantité et de la qualité des données collectées. Cette étape est donc la plus importante.

4.2.1.1 Collection des données

Le processus de collecte des données était fastidieux et pénible. Aucun data set ne correspondait à nos attentes. De longues recherches nous ont menées à un data set de grand volume (136 Gb), son contenu était inaccessible mais sa description convenait bien. Le lien de téléchargement associé change toutes les 4 heures et le débit de téléchargement dont on disposait ne nous le permettait pas. La solution était de tester le téléchargement sur Jupyter Cloud dont le débit était élevé (il peut atteindre 40 Mo/s). Après le téléchargement de ce dossier sur le serveur Jupyter, nous avons trouvé seulement 1000 images exprimant la fatigue et 2000 images exprimant la non fatigue. Ce nombre d'observations s'avère insuffisant et une autre recherche de data set s'est lancée. Notre dernier recours était un data set comportant des vidéos d'enregistrement de personnes devant un écran. Après un certain temps, la personne sur la vidéo commence à être fatiguée. La taille des dossiers est importante (minimum 20 Gb), ils ont été téléchargés sur le Drive à partir de Google Colab dont le stockage est limité à 100 Gb.

4.2.1.2 Extraction des images à partir d'un vidéo

Après avoir stocké le dataset dans le google drive on a commencé de transformer les vidéos en des images on utilisant un algorithme créé avec python et des fonctionnalités de bibliothèque OpenCV, ci-dessous vous trouvez une capture de code.

```
[ ] import cv2
    import math

    videoFile = "/content/drive/Shared drives/Drowsy Driving Dataset/DrowsyDrivingDataset/Watching_Simulation/225.AVI"
    imagesFolder = "/content/drive/My Drive/drowsy/video25"
    cap = cv2.VideoCapture(videoFile)
    frameRate = cap.get(5) #frame rate
    while(cap.isOpened()):
        frameId = cap.get(1) #current frame number
        ret, frame = cap.read()
        if (ret != True):
            break
        if (frameId % math.floor(frameRate) == 0):
            filename = imagesFolder + "/video25" +  str(int(frameId)) + ".jpg"
            cv2.imwrite(filename, frame)
        print('Saved image ', frameId)
    cap.release()

    Saved image  0.0
    Saved image  1.0
    Saved image  2.0
    Saved image  3.0
    Saved image  4.0
    Saved image  5.0
    Saved image  6.0
    Saved image  7.0
    Saved image  8.0
```

FIGURE 4.4 – Extraction des image a partir d'un vidéo algorithme

4.2.1.3 Traitement des données

Après l'extraction des images à partir des vidéos, nous avons procédé à la sélection des images qui correspondent à nos besoins et à leur séparation en deux dossiers. Un dossier qui contient les images de classe Fatigue, et un autre qui contient les images de classe Non Fatigue. Deux semaines ont été consacrées à cette étape. La plupart des images obtenues étaient floues, elles ont, par conséquent, subi un traitement sous Photoshop. Suite au traitement des images en question, nous avons obtenu 5000 images dans un dossier Fatigue et 10000 images dans un dossier Non Fatigue.

4.2.1.4 Nettoyage des données

Nous avons lancé le training sur le data set décrit plus haut, nous avons obtenu un mauvais résultat. Nous avons choisi de nous concentrer uniquement sur le visage et supprimer l'arrière-plan de chaque image afin de minimiser le nombre de features. Ainsi, au lieu d'avoir l'image entière comme feature, notre feature devient seulement le visage. Cette étape a pour but d'avoir un data set qui contient seulement des visages fatigués et des visages non fatigués, ce qui nous permet d'éliminer le bruit et nous aide à améliorer l'accuracy de notre modèle.

```
Entrée [1]:
import mtcnn
import os
from matplotlib import pyplot
from matplotlib.patches import Rectangle
from mtcnn.mtcnn import MTCNN
import shutil

Using TensorFlow backend.
```

```
Entrée [2]:
import mtcnn
import os
from matplotlib import pyplot
from matplotlib.patches import Rectangle
from mtcnn.mtcnn import MTCNN
import shutil

def crop_faces(element):
    def draw_faces(filename, result_list):
        # load the image
        img = pyplot.imread(filename)
        # get coordinates
        x1, y1, width, height = result_list[0]['box']
        x2, y2 = x1 + width, y1 + height
        # plot face
        pyplot.axis('off')
        pyplot.imshow(img[y1:y2,x1:x2])
        pyplot.savefig('D:/PCD/validation/non fatigue'+ "/" +element)

    filename = os.path.join('D:/PCD/dataset/validation/non fatigue',element)
    # load image from file
    pixels = pyplot.imread(filename)
    # create the detector, using default weights
    detector = MTCNN()
    # detect faces in the image
    faces = detector.detect_faces(pixels)
    # display faces on the original image
    draw_faces(filename, faces)


for element in os.listdir('D:/PCD/dataset/validation/non fatigue'):
        try:
            print (element)
            filename = os.path.join('D:/PCD/dataset/validation/non fatigue',element)
            crop_faces(element)
            shutil.move(filename,'D:/PCD/Nouveau dossier');
        except ValueError:

            shutil.move(filename,'D:/PCD/noncadreval/non fatigue')
            pass
```

```
video21800.jpg
video21920.jpg
video21950.jpg
video22010.jpg
video22220.jpg
video22280.jpg
video2240.jpg
video22820.jpg
video22850.jpg
video22970.jpg
video2300.jpg
video23000.jpg
video23090.jpg
```

FIGURE 4.5 – Algorithme de cropping

4.2.2 Classification

Après avoir nettoyé et bien préparé notre dataset nous pouvons travailler avec nos données pour construire notre modèle de classification.

4.2.2.1 Organisation de données

Pour évaluer la performance du notre modèle de classification , nous devons tester le modèle sur des données invisibles. Par conséquent, avant de construire un modèle, nous

divisons nos données en deux parties : 80% pour le training et 20% pour le test. Nous avons utilisé l'ensemble de training pour former le modèle. Ensuite, nous avons utilisé le modèle entraîné pour faire des prédictions sur l'ensemble de test invisible. Cette approche nous donne une idée des performances du modèle.

4.2.2.2 Construction de modèle

Dans cette étape nous avons construit deux modèle un sans transfert learning et l'autre avec transfert learning.

1.Modèle sans transfert learning

— **Importation des packages**

Commençons par importer les packages requis. Le package os est utilisé pour lire les fichiers et la structure des répertoires, NumPy est utilisé pour convertir la liste python en tableau numpy et pour effectuer les opérations de matrice requises et matplotlib.pyplot pour tracer le graphique et afficher les images dans les données de formation et de validation. Importez les classes Tensorflow et Keras nécessaires pour construire notre modèle.

```
Entrée [2]: import tensorflow as tf
            from tensorflow.keras.models import Sequential
            from tensorflow.keras.layers import Dense, Conv2D, Flatten, Dropout, MaxPooling2D
            from tensorflow.keras.preprocessing.image import ImageDataGenerator

            import os
            import numpy as np
            import matplotlib.pyplot as plt
```

FIGURE 4.6 – Importation des bibliotheques.

— **Charger les données**

cette partie de code consiste à charger les dossier train et validation

```
Entrée [5]: train_dir = os.path.join(root_path, 'train')
            validation_dir = os.path.join(root_path, 'validation')

Entrée [6]: train_fat_dir=os.path.join(train_dir, 'fatigue')
            train_nofat_dir=os.path.join(train_dir, 'non fatigue')
            validation_fat_dir =os.path.join(validation_dir, 'fatigue')
            validation_nofat_dir=os.path.join(validation_dir, 'non fatigue')
```

FIGURE 4.7 – Chargement des données

— **Préparation des données**

Etant donné que chaque image est composée d'un ensemble de pixels de valeurs entre 0 et 255, le but de cette partie du code est de rendre ces valeurs entre 0 et 1 car les réseaux de neurones préfèrent traiter des petites valeurs. Cette tâche peut

être effectuée avec la classe ImageDataGenerator fournie par tf.keras.

```
Entrée [11]: train_image_generator = ImageDataGenerator(rescale=1./255) # Generator for our training data
             validation_image_generator = ImageDataGenerator(rescale=1./255) # Generator for our validation data
```

FIGURE 4.8 – Image generator

Après avoir défini les générateurs pour les images de training et de validation, la méthode flow_from_directory charge les images à partir du disque et redimensionne les images avec les dimensions requises(IMG_HEIGHT et IMG_WIDTH).

```
Entrée [12]: train_data_gen = train_image_generator.flow_from_directory(batch_size=batch_size,
                                                                        directory=train_dir,
                                                                        shuffle=True,
                                                                        target_size=(IMG_HEIGHT, IMG_WIDTH),
                                                                        class_mode='binary')
             Found 12891 images belonging to 2 classes.

Entrée [13]: val_data_gen = validation_image_generator.flow_from_directory(batch_size=batch_size,
                                                                           directory=validation_dir,
                                                                           target_size=(IMG_HEIGHT, IMG_WIDTH),
                                                                           class_mode='binary')
             Found 2880 images belonging to 2 classes.
```

FIGURE 4.9 – Data generator

— **creation de modele**
Après avoir préparé notre data, nous avons créé notre modèle avec la classe Sequential . ce modèle se compose de trois blocs de convolution avec une couche de pool max dans chacun d'eux. Après, on aplatit (flatten) ces matrices dans une seule colonne .ensuite on passe par une couche entièrement connectée avec 512 unités au-dessus qui est activée par une fonction d'activation ReLu.Enfin, on termine par une layer qui contient un seul neurone pour avoir le résultat.

```
Entrée [17]: model = Sequential([
                 Conv2D(16, 3, padding='same', activation='relu', input_shape=(IMG_HEIGHT, IMG_WIDTH ,3)),
                 MaxPooling2D(),
                 Conv2D(32, 3, padding='same', activation='relu'),
                 MaxPooling2D(),
                 Conv2D(64, 3, padding='same', activation='relu'),
                 MaxPooling2D(),
                 Flatten(),
                 Dense(512, activation='relu'),
                 Dense(1)
             ])
```

FIGURE 4.10 – Création de model

— **compiler le modèle**
Dans cette partie nous avons utilisé ADAM comme "Optimizer" et Binary Cross

entropy comme "Loss function" pour compiler notre modèle.

```
Entrée [18]: model.compile(optimizer='adam',
                           loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
                           metrics=['accuracy'])
```

FIGURE 4.11 – Compiler le modèle

— **Résumé du modèle**
nous avons utilisé model.summary() pour afficher toutes les couches du réseau.

```
Entrée [19]: model.summary()

Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 180, 180, 16)      448
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 90, 90, 16)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 90, 90, 32)        4640
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 45, 45, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 45, 45, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 22, 22, 64)        0
_________________________________________________________________
flatten (Flatten)            (None, 30976)             0
_________________________________________________________________
dense (Dense)                (None, 512)               15860224
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 513
=================================================================
Total params: 15,884,321
Trainable params: 15,884,321
Non-trainable params: 0
_________________________________________________________________
```

FIGURE 4.12 – Model summary

— **Former le modèle**
Après avoir préparé notre data et la structure de notre modèle nous avons commencé le training en utilisant la fonction fitgenerator qui prend en paramètre le donné de training,le donné de validation et le nombre d'epochs.

```
Entrée [20]: history = model.fit_generator(
                 train_data_gen,
                 steps_per_epoch=total_train // batch_size,
                 epochs=epochs,
                 validation_data=val_data_gen,
                 validation_steps=total_val // batch_size
             )
             201/201 [==============================] - 291s 1s/step - loss: 0.0171 - accuracy: 0.9942 - val_loss: 1.4786 - val_accuracy:
             0.8122
             Epoch 10/15
             201/201 [==============================] - 291s 1s/step - loss: 0.0124 - accuracy: 0.9961 - val_loss: 1.1141 - val_accuracy:
             0.8566
             Epoch 11/15
             201/201 [==============================] - 308s 2s/step - loss: 0.0109 - accuracy: 0.9966 - val_loss: 1.1354 - val_accuracy:
             0.8441
             Epoch 12/15
             201/201 [==============================] - 289s 1s/step - loss: 0.0130 - accuracy: 0.9959 - val_loss: 1.1406 - val_accuracy:
             0.8694
             Epoch 13/15
             201/201 [==============================] - 308s 2s/step - loss: 0.0086 - accuracy: 0.9970 - val_loss: 0.9415 - val_accuracy:
             0.8677
             Epoch 14/15
             201/201 [==============================] - 305s 2s/step - loss: 0.0062 - accuracy: 0.9974 - val_loss: 1.3898 - val_accuracy:
             0.8431
             Epoch 15/15
             201/201 [==============================] - 282s 1s/step - loss: 0.0175 - accuracy: 0.9944 - val_loss: 1.1865 - val_accuracy:
             0.8573
```

FIGURE 4.13 – Model training

— **Visualisez les résultats de la “training”**
Cette partie de code consiste à afficher le résultat de progression de training sous forme de courbe qui affiche l’accuracy et le loss pour le training et la validation.

```
Entrée [22]: acc = history.history['accuracy']
             val_acc = history.history['val_accuracy']

             loss=history.history['loss']
             val_loss=history.history['val_loss']

             epochs_range = range(epochs)

             plt.figure(figsize=(8, 8))
             plt.subplot(1, 2, 1)
             plt.plot(epochs_range, acc, label='Training Accuracy')
             plt.plot(epochs_range, val_acc, label='Validation Accuracy')
             plt.legend(loc='lower right')
             plt.title('Training and Validation Accuracy')

             plt.subplot(1, 2, 2)
             plt.plot(epochs_range, loss, label='Training Loss')
             plt.plot(epochs_range, val_loss, label='Validation Loss')
             plt.legend(loc='upper right')
             plt.title('Training and Validation Loss')
             plt.show()
```

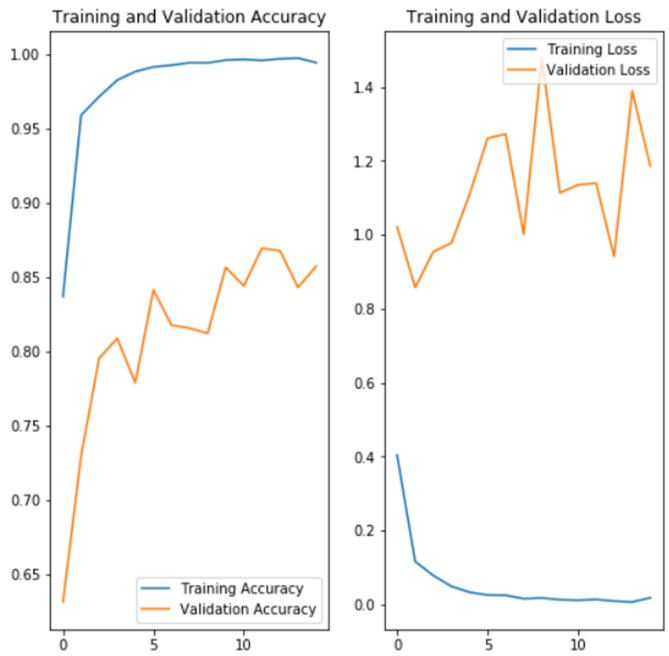

FIGURE 4.14 – Training result

Comme vous pouvez le voir sur la figure 4.14, la précision d'entraînement de modèle augmente linéairement au fil du temps jusqu'à atteindre 99% tandis que la précision de la validation stagne autour de 85%. En outre, la différence entre la précision de l'entraînement et de la validation est perceptible, signe d'un "overfitting".

2.Modèle avec transfert learning

— **Création du modèle**

Dans le modèle précédent, nous avons travaillé sur un sous-ensemble de données

(5000 images fatigue et 10000 non fatigue) pour construire un classificateur d'images capable de classer les images de fatigue et non fatigue avec une précision de 85 % . Maintenant nous avons essayé d'améliorer nos performances en utilisant le transfert learning.Nous avons utilisé le modèle Inception-v3 dans ce projet parce qu'on a trouvé dans l'internet que ce modèle donne un bon résultat dans les problèmes de classification[9].
Premièrement , nous avons commencé par télécharger les weights du modèle **inceptionV3**,puis nous avons défini la structure de notre modèle pre-trained en utilisant la fonction InceptionV3 qui prend comme paramètre :
inputshape : pour spécifier la hauteur et la largeur des images.
includetop : égale False pour supprimer les couches de sortie entièrement connectées du modèle qui ont utilisées pour faire des prédictions et pour qu'on puisse ajoutée et formée une nouvelle couche de sortie .
weights : égale NULL pour une initialisation aléatoire du poids.

Deuxièmement, nous fixons les poids de notre modèle pre-trained aux valeurs téléchargées .
Troisièmement,nous avons mis layer.trainable égale false pour indiquer que cette couche ne doit pas être entraînée (pour geler les weights d'une couche).

```
[ ] url = 'https://storage.googleapis.com/mledu-datasets/inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5'

[ ] import wget

[ ] filename =wget.download(url)

[ ] import os

    from tensorflow.keras import layers
    from tensorflow.keras import Model
    from tensorflow.keras.applications.inception_v3 import InceptionV3

    pre_trained_model = InceptionV3(input_shape = (150, 150, 3),
                                    include_top = False,
                                    weights = None)

    pre_trained_model.load_weights(filename)

    for layer in pre_trained_model.layers:
          layer.trainable = False

    #pre_trained_model.summary()

    last_layer = pre_trained_model.get_layer('mixed7')
    print('last layer output shape: ', last_layer.output_shape)
    last_output = last_layer.output

    last layer output shape:  (None, 7, 7, 768)
```

FIGURE 4.15 – InceptionV3 pre trained model

Après avoir préparé le pretrained modèle nous avons ajouté des dernières couches du réseau à ce pretrained modèle pour obtenir notre modèle final . pour faire ceci on a utilisé :
layers.flatten :aplatit des matrices en une simple vecteur.
layers.dropout : est une technique utilisée pour empêcher un modèle de overfitting.

Dropout fonctionne en définissant au hasard les edges sortants des unités cachées (neurones qui composent les couches cachées) à 0 à chaque mise à jour de la phase d'entraînement.

```
[ ] from tensorflow.keras.optimizers import RMSprop

    # Flatten the output layer to 1 dimension
    x = layers.Flatten()(last_output)
    # Add a fully connected layer with 1,024 hidden units and ReLU activation
    x = layers.Dense(1024, activation='relu')(x)
    # Add a dropout rate of 0.2
    x = layers.Dropout(0.2)(x)
    # Add a final sigmoid layer for classification
    x = layers.Dense  (1, activation='sigmoid')(x)

    model = Model( pre_trained_model.input, x)

    model.compile(optimizer = RMSprop(lr=0.0001),
                  loss = 'binary_crossentropy',
                  metrics = ['accuracy'])
```

FIGURE 4.16 – Deuxième modèle structure

— **Formation du modèle et visualisation de résultat**
Après avoir préparé le modèle , on exécute le training avec la fonction fit,puis on a affiché le résultat sous forme de courbe qui affiche l'accuracy et le loss.

```
[ ] Epoch 1/20
    100/100 [==============================] - 5309s 53s/step - loss: 0.2806 - accuracy: 0.9065 - val_loss: 0.2750 - val_accuracy: 0.8906
    Epoch 2/20
    100/100 [==============================] - 597s 6s/step - loss: 0.0834 - accuracy: 0.9699 - val_loss: 1.0738 - val_accuracy: 0.7660
    Epoch 3/20
    100/100 [==============================] - 611s 6s/step - loss: 0.0552 - accuracy: 0.9825 - val_loss: 0.2867 - val_accuracy: 0.8825
    Epoch 4/20
    100/100 [==============================] - 609s 6s/step - loss: 0.0422 - accuracy: 0.9857 - val_loss: 0.2519 - val_accuracy: 0.9009
    Epoch 5/20
    100/100 [==============================] - 611s 6s/step - loss: 0.0351 - accuracy: 0.9875 - val_loss: 0.3858 - val_accuracy: 0.8540
    Epoch 6/20
    100/100 [==============================] - 608s 6s/step - loss: 0.0283 - accuracy: 0.9900 - val_loss: 0.2718 - val_accuracy: 0.9041
    Epoch 7/20
    100/100 [==============================] - 600s 6s/step - loss: 0.0224 - accuracy: 0.9926 - val_loss: 0.3143 - val_accuracy: 0.8881
    Epoch 8/20
    100/100 [==============================] - 572s 6s/step - loss: 0.0194 - accuracy: 0.9931 - val_loss: 0.3455 - val_accuracy: 0.8835
    Epoch 9/20
    100/100 [==============================] - 593s 6s/step - loss: 0.0149 - accuracy: 0.9954 - val_loss: 0.3863 - val_accuracy: 0.8899
    Epoch 10/20
    100/100 [==============================] - 598s 6s/step - loss: 0.0190 - accuracy: 0.9935 - val_loss: 0.3023 - val_accuracy: 0.9020
    Epoch 11/20
    100/100 [==============================] - 584s 6s/step - loss: 0.0135 - accuracy: 0.9955 - val_loss: 0.3379 - val_accuracy: 0.9013
    Epoch 12/20
    100/100 [==============================] - 599s 6s/step - loss: 0.0120 - accuracy: 0.9961 - val_loss: 0.3233 - val_accuracy: 0.9013
    Epoch 13/20
    100/100 [==============================] - 609s 6s/step - loss: 0.0110 - accuracy: 0.9965 - val_loss: 0.3569 - val_accuracy: 0.9016
    Epoch 14/20
    100/100 [==============================] - 604s 6s/step - loss: 0.0117 - accuracy: 0.9959 - val_loss: 0.3807 - val_accuracy: 0.8924
    Epoch 15/20
    100/100 [==============================] - 602s 6s/step - loss: 0.0133 - accuracy: 0.9951 - val_loss: 0.3998 - val_accuracy: 0.8991
    Epoch 16/20
    100/100 [==============================] - 604s 6s/step - loss: 0.0079 - accuracy: 0.9966 - val_loss: 0.3931 - val_accuracy: 0.9009
    Epoch 17/20
    100/100 [==============================] - 597s 6s/step - loss: 0.0074 - accuracy: 0.9977 - val_loss: 0.3772 - val_accuracy: 0.9013
    Epoch 18/20
    100/100 [==============================] - 599s 6s/step - loss: 0.0088 - accuracy: 0.9976 - val_loss: 0.4413 - val_accuracy: 0.8999
    Epoch 19/20
    100/100 [==============================] - 598s 6s/step - loss: 0.0066 - accuracy: 0.9983 - val_loss: 0.4099 - val_accuracy: 0.9009
    Epoch 20/20
    100/100 [==============================] - 585s 6s/step - loss: 0.0081 - accuracy: 0.9978 - val_loss: 0.4580 - val_accuracy: 0.9006
```

FIGURE 4.17 – Deuxième modèle training

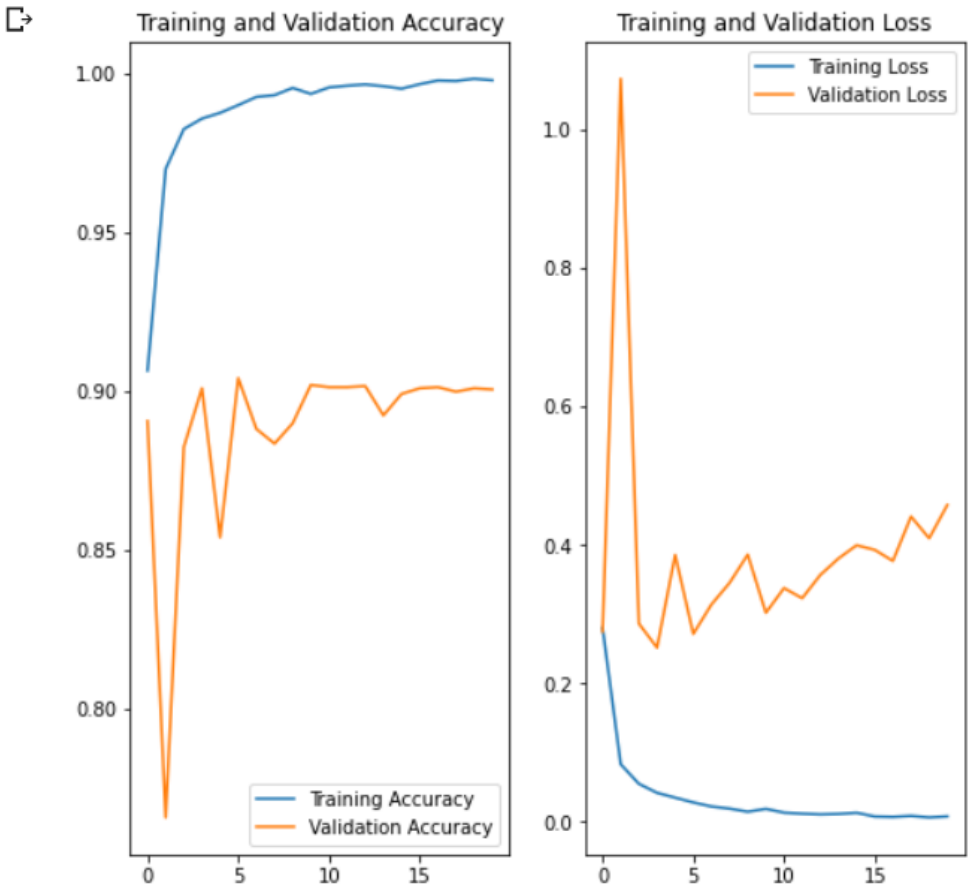

FIGURE 4.18 – Résultat de training de deuxième modèle

Comme vous pouvez le voir sur la figure 4.18, pour le modèle avec le transfert learning,les courbes montrent qu'il y a beaucoup moins de "overfitting" qu'auparavant et le modèle a atteint une précision d'environ 90% sur l'ensemble de validation donc on a choisi le modèle avec transfert learning.

4.2.3 Application Web

Cette phase est dédiée à présenter notre interface d'application web qui est développée pour analyser le visage d'un individu et détecter si il est fatigué ou pas en utilisant principalement des approches basées sur l'apprentissage en profondeur,HTML,CSS,JAVASCRIPT et Flask.

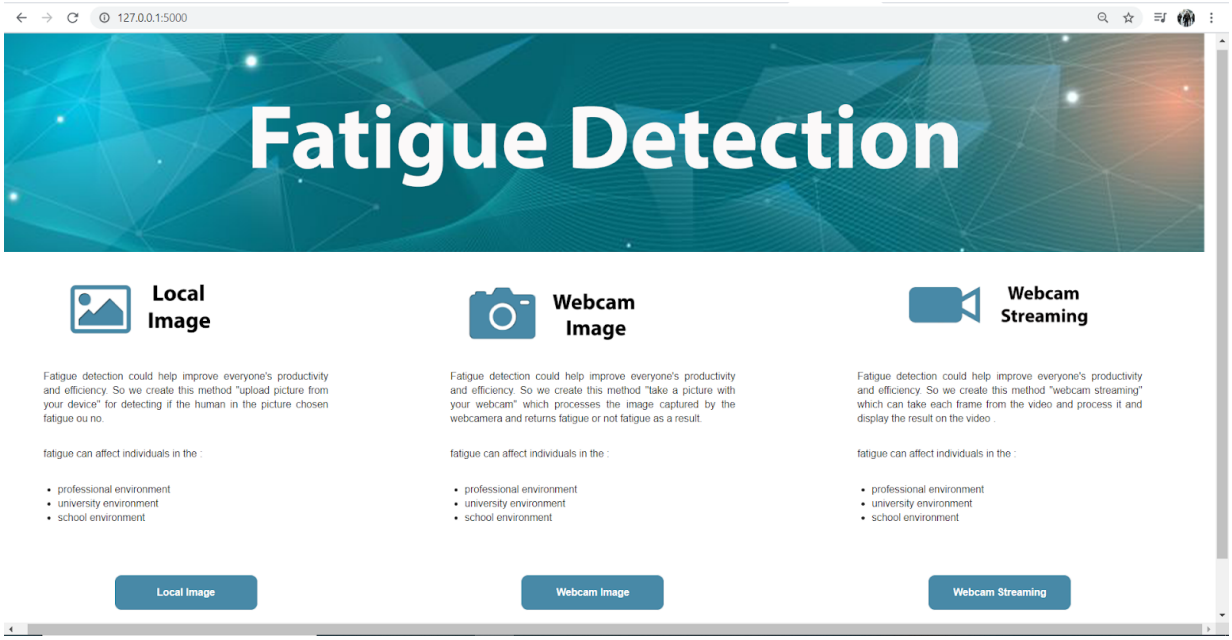

FIGURE 4.19 – page d'accueil

Comme la figure ci dessus l'indique cette application web composée de trois parties :

— **LOCAL IMAGE**
cette partie de l'application web consiste à télécharger une image à partir de la machine d'utilisateur puis la traiter et afficher le résultat de ce traitement qui peut être fatigué ou non fatigué.

— **WebCam Image**
cette partie de l'application web consiste à prendre une photo à partir de webcaméra puis le traiter et afficher le résultat.

— **WebCam Streaming**
cette partie consiste à lancer une vidéo à partir d'une webcam et faire un traitement de chaque frame de cette vidéo en temps réel et affiche le résultat sur la vidéo.

4.2.4 Conclusion

Dans ce chapitre, nous avons présenté notre environnement de travail. Ensuite, nous avons passés à la présentation de l'implémentation de notre solution.

Conclusion Générale et perspectives

Vu l'augmentation du nombre d'accidents chaque année à cause de la fatigue, nous avons proposé la conception et le développement d'une application de détection de fatigue. Cette application sert à classer une image capturée à partir d'une webcam dans l'une des deux classes Fatigue ou Non Fatigue. Elle permet aussi de détecter la fatigue en temps réel à partir d'une vidéo en direct, le rôle étant de traiter chaque frame de cette vidéo et afficher le résultat au même instant.

Des technologies avancées, comme le deep learning, le transfer learning et les environnements de développement dédiés, ont été utilisées pour la classification d'une image/vidéo selon qu'un individu présente un état de fatigue ou pas. Cet individu pourrait être un conducteur/pilote, un employé, un élève, etc., l'intérêt étant d'assurer un certain contrôle pouvant minimiser les risques d'accidents dus à la somnolence, maximiser la productivité des employés d'une entreprise, fiabiliser davantage les services offerts par ceux-ci et assurer un bon apprentissage des lycéens et étudiants.

Pour bien formuler notre problème, nous avons décomposé le projet en quatre chapitres. Nous avons présenté dans le premier chapitre une étude préalable qui nous a permis d'introduire des concepts clés en intelligence artificielle et d'étudier quelques solutions existantes . Puis, nous avons analysé les besoins et spécifier les cas d'utilisations afin de fixer toutes les fonctionnalités que le système doit satisfaire. Ensuite, nous avons présenté une conception globale et détaillée, une description de la démarche suivie pour la collecte des données et l'entraînement des modèles pour une solution efficace. Enfin, dans le dernier chapitre nous avons détaillé la phase de réalisation de l'application et nous avons évalué la performance des algorithmes utilisés en se basant sur les critères d'accuracy et de loss.

Finalement, nous tenons à préciser qu'il existe plusieurs moyens pour améliorer ce travail. À court terme on peut travailler sur l'augmentation des données, à moyen terme on peut intégrer ce système de détection de fatigue dans un système embarqué (un véhicule par exemple). Nous proposons aussi d'expérimenter d'autres modèles qui pourraient offrir de meilleures performances.

Bibliographie

Netographie

Bibliography

[B1] Deep Learning with Python FRANÇOIS CHOLLET

[B2] Hands-On Transfer Learning with Python

[B3]https ://www.researchgate.net/publication/232657899_A_PERCLOS-Based_Driver_Fatigue_

[B4]https ://flask.palletsprojects.com/en/1.1.x/

Netography

[1]https ://missinglink.ai/guides/convolutional-neural-networks/convolutional-neural-network-architecture-forging-pathways-future/

[2]https ://penseeartificielle.fr/wp-content/uploads/2019/01/Convolution-11.jpg

[3]https ://www.simplilearn.com/ice9/free_resources_article_t_humb/rectified.JPG

[4]https ://sds-platform-private.s3-us-east-2.amazonaws.com/uploads/74_blog_image_1.png

[5]https ://pypi.org/project/mtcnn/

[6]https ://www.amazon.fr/Appareil-davertissement-fatigue-conduite-clignotants/dp/B07MHVMV59

[7]https ://www.usine-digitale.fr/article/innov-detecte-le-coup-de-fatigue-du-conducteur.N372860

[8]https ://missinglink.ai/guides/convolutional-neural-networks/convolutional-neural-network-architecture-forging-pathways-future/ ?fbclid=IwAR0IIoOMCFNSb2pU8EKVVoteC7Npoc
zxC1ydf7b7E

[9]https ://medium.com/analytics-vidhya/transfer-learning-using-inception-v3-for-image-classification-86700411251b

[10]https ://uploads.lightcode.fr/books/01-les_reseaux/client-serveur.png

[11]https ://www.supinfo.com/articles/single/8729-architecture-mvc-qu-est-ce-que-c-est

[12]https ://miro.medium.com/max/1400/1*gqKM5V-uo2sMFFPDS84yJw.png

[13]https ://medium.com/@sh.tsang/review-inception-v3-1st-runner-up-image-classification-in-ilsvrc-2015-17915421f77c

تلخيص

الهدف من هذا المشروع هو الكشف عن تعب الإنسان من خلال إنشاء نموذج ذكي لتصنيف الصور إلى فئتين من التعب وغير التعب بالإضافة إلى تطبيق ويب يمكن للمستخدم من خلاله الأخير نتيجة اختبار هذا النموذج في التقاط صورة أو تشغيل فيديو مباشر وسيعرض هذا التصنيف في الوقت الحقيقي

Résumé

le but de ce projet est de détecter la fatigue d'un être humain par la création d'un modele intelligent de classification des images en deux classes fatigue et non fatigue en plus d'une application web sur laquelle un utilisateur peut tester ce modele en saisissant une image ou lancer une vidéo en direct et ce dernier permet d'afficher le résultat de classification en temps réel.
.

Abstract

the objective of this project is to detect the fatigue of a human being by creating an intelligent model for classifying images into two classes fatigue and no-fatigue in addition to a web application on which a user can test this model with input an image or launching a video stream and the result will be displayed on the site in real time.