PROJET THEMATIQUE

IA - Segmentation de cheveux et transfert de couleur

Filière électronique
Groupe Majeur TSI 1

BELASRI Sami
PECORARO Gabriel

Supervisé par
GIRAUD Rémi

20/03/2024

Table des matières

1 Introduction

L'avènement de l'apprentissage profond a révolutionné de nombreux domaines de l'informatique, en particulier la vision par ordinateur. Cette technologie a permis le développement d'applications innovantes et interactives, transformant notre manière d'interagir avec les appareils numériques. Parmi ces avancées, l'une des plus récentes et les plus fascinantes est l'utilisation de l'apprentissage profond pour le changement de couleur de cheveux en temps réel. Le changement de couleur de cheveux a gagné en popularité notamment avec l'arrivée de partage de photos sur réseaux sociaux comme Snapchat, Instagram ou autre. De nos jours, toute application faisant intervenir une caméra propose un changement de couleur de cheveux en direct ou après traitement.

Le projet présenté ici, intitulé "Changement de couleur de cheveux par apprentissage profond", s'inscrit dans cette dynamique d'innovation en exploitant les capacités de l'apprentissage profond pour segmenter les cheveux dans des images et des vidéos, puis appliquer des modifications de couleur de manière réaliste. Concrètement, ce projet vise à créer un pipeline fonctionnel basé sur des réseaux de neurones profonds pour la segmentation des cheveux et le transfert de couleur en temps réel, avec une application pratique sur des dispositifs embarqués tels que le Raspberry Pi. Nous travaillerons en binômes tout au long de ce projet. Tout d'abord, nous entreprendrons de rechercher une base de données plus exhaustive que celle dont nous disposons actuellement. Ensuite, nous nous attaquerons à la prise en main du Raspberry Pi et à sa configuration afin de permettre l'acquisition en temps réel de vidéos et enfin nous essayerons de proposer un algorithme de changement de couleur réaliste. L'objectif est de réaliser un algorithme portatifs en pouvant l'implémenter sur tout type de systèmes embarqués tels que des webcams ou des Raspberry.

2 L'algorithme initiale

Tout les codes réalisés au long du projet seront réalisé avec le langage de programmation Python. Ce dernier est actuellement très en vogue dans le monde de l'intelligence artificielle avec les nombreuses librairies incluses telles que Pytorch que nous avons utilisé tout au long du projet. L'architecture de notre inteligence artificielle est appelée U-Net. Le U-Net est une architecture de réseau de neurones convolutifs utilisée principalement pour la segmentation d'images, notamment dans le domaine biomédical. Sa structure en forme de U comprend un encodeur à gauche pour extraire les caractéristiques de l'image et un décodeur à droite pour reconstruire l'image segmentée. Les sauts de connexions entre l'encodeur et le décodeur permettent de préserver les détails spatiaux et améliorent la précision de la segmentation. Cette architecture est populaire en raison de sa capacité à produire des segmentations précises, même avec des ensembles de données limités.

Cette architecture est celle que nous allons entraîner tout au long du projet. Le but est simple, qu'elle soit très précise lors de la segmentation de cheveux peut importe la coupe présente sur la photo. Il y a néanmoins un risque, il faut contrôler son apprentissage pour éviter l'apprentissage par coeur qui vient nuire au modèle car il le rend moins performant. La base de donnée fournie comprenait environ 800 images, ce qui était plutôt correct. Nous avions entraîné notre intelligence artificiel avec cette base de donnée et nous avions obtenu ces résultats :

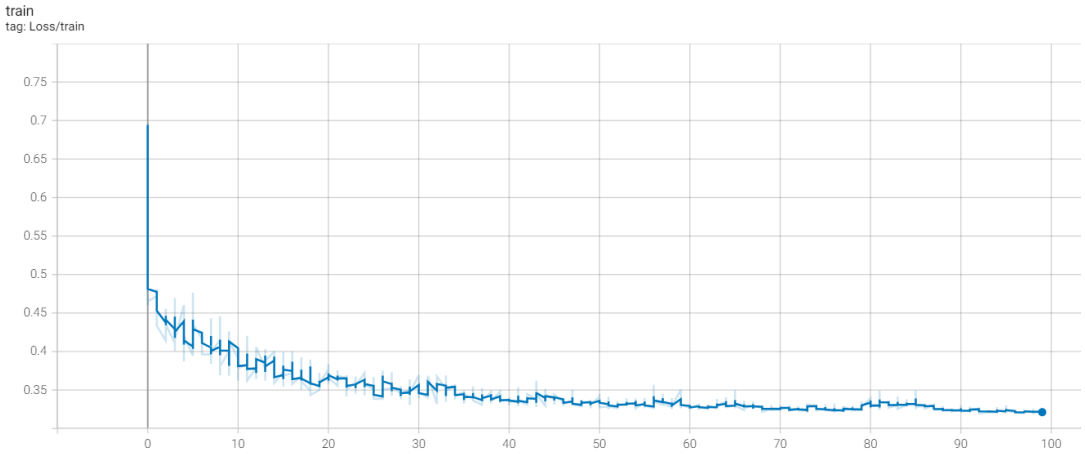

FIGURE 1 – Courbe Loss en fonction des epochs de la partie Train

FIGURE 2 – Image à segmenter

FIGURE 3 – Image prévue

FIGURE 4 – Image obtenue par notre IA

A première vue, notre IA n'a pas l'air d'être en sur-apprentissage d'après la figure 1. De plus, la précision de la validation était de 0.9470.

```
Validation loss: 0.3650, Validation acc: 0.9470
```

FIGURE 5 – Résultats avec la base de donnée Figaro1K

Néanmoins, lorsque l'on regarde d'un peu plus près les résultats directement sur nos images (cf. figure 4), nous nous sommes rendus compte que notre IA commettait trop d'erreurs et d'imprécisions, des segmentations du background qui n'était absolument pas demandées, en somme, notre segmentation n'était pas celle recherchée.

3 Recherche d'une base de donnée adéquate

À la suite de cela, nous nous sommes tournés vers la recherche d'une nouvelle base de donnée plus complète et exhaustive. Une base de donnée facile d'accès, gratuite et directement utilisable a particulièrement retenu notre attention : la base de donnée CelebMask-HQ (disponible ici : CelebMask-HQ).

CelebAMask-HQ est un ensemble de données d'images faciales à grande échelle comprenant 30 000 images de visages haute résolution sélectionnées à partir de l'ensemble de données CelebA en suivant CelebA-HQ. Chaque image est accompagnée d'un masque de segmentation des attributs faciaux correspondant à CelebA. Les masques de CelebAMask-HQ ont été annotés manuellement avec une taille de 512 x 512 pixels et comportent 19 classes, incluant tous les composants faciaux et accessoires tels que la peau, le nez, les yeux, les sourcils, les oreilles, la bouche, les lèvres, les cheveux, le chapeau, les lunettes, les boucles d'oreilles, le collier, le cou et les vêtements.

Dans notre cas, nous avons utilisé uniquement 8000 images sans compter les rotations et les retournements, car nos ordinateurs n'étaient pas assez puissants pour traiter davantage d'images. De plus, il est à noter que les images ont été redimensionnées en 256 par 256 pixels, par choix.

CelebAMask-HQ peut être utilisé pour entraîner et évaluer des algorithmes de segmentation faciale, de reconnaissance faciale et de GANs pour la génération et l'édition de visages.

Avec cette nouvelle base de donnée, nous obtenons ces résultats :

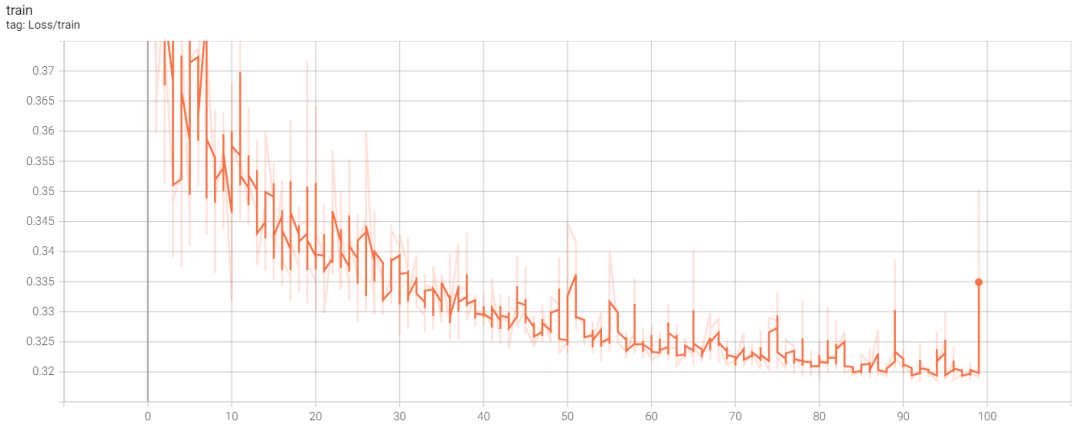

FIGURE 6 – Courbe Loss en fonction des epochs de la partie Train

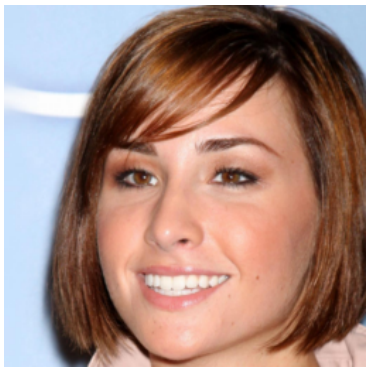

FIGURE 7 – Image à segmenter

FIGURE 8 – Image prévue

FIGURE 9 – Image obtenue par notre IA

De prime abord, lorsque l'on regarde la figure 6, nous pouvons dire que notre IA n'est a priori pas dans un régime de sur-apprentissage. Pour compléter, nous avions obtenu une précision sur la phase de validation de 0.965, ce qui est une petite amélioration par

rapport à la base de donnée précédente mais tout de même appréciable. De plus, en comparant les résultats directement sur les images (cf. figure 7-9), nous pouvons aussi dire que la segmentation est plutôt précise.

Néanmoins, elle est précise mais pas robuste puisque toutes nos images ont été entraînées sur une base de donnée sans background, avec des visages et cheveux centrés et prenant 80% de l'image. Or, ce n'est pas très réaliste puisque dans la pratique, nous savons pertinemment que ce ne sera pas le cas et donc qu'il y aura beaucoup d'imprécision et de concession à devoir faire. En effet :

FIGURE 10 – Image à segmenter

FIGURE 11 – Image prévue

FIGURE 12 – Image obtenue par notre IA

En somme, on a pu remarque sur une plus grande base de données, notre architecture a amélioré sa segmentation sur les photos ce qui nous est plutôt favorable. Néanmoins, bien que nous atteignons des résultats jusqu'à 96%, cela relève quasiment du hasard pour l'intelligence artificielle, nous allons donc plus tard essayer de nous repencher sur notre modèle pour le rendre encore plus robuste.

4 Raspberry PI

Pour tester la portabilité de notre intelligence artificielle nous avons eu a notre disposition un Raspberry PI qui a fait office de système embarquer. Nous allons donc expliciter étape par étape comment nous avons fait pour essayer de implémenter notre réseau de neurones pour changer la couleur des cheveux sur un système embarqué telle que on le ferait avec un site web ou une application pour le grand public.

4.1 Prise en Main

Le but tout d'abord est de prendre en main l'interface Raspberry et de comprendre comment elle fonctionne. Elle se présente de la manière suivante, on a notre disposition un terminal UNIX, un éditeur de script et un accés à internet. Il est possible de se connecter à distance sur le Raspberry en exploitant l'adresse IP qui est fournie et en utilisant la commande bash suivante :

```
ssh pi@"IP_de_notre_raspberry"
```

On nous a fourni avec ce Raspberry une caméra que l'on peut connecter pour prendre différente photo ou vidéo. Pour pouvoir utiliser la caméra, il faut effectuer les commandes bash suivantes pour pouvoir configurer les différents paramètres de la caméra tels que la

résolution, le nombre d'image par seconde si jamais on fait une vidéo, le temps de capture, le nom du fichier et autre. Les commandes sont les suivantes :

```
sudo raspi-config
```

Cette commande permet d'accéder au menu de configuration du Raspberry pour pouvoir activer la caméra. Par la suite il est possible de prendre une photo ou une vidéo avec les commandes suivantes :

```
raspistill -o image.jpg
raspivid -o video.h264
```

4.2 Gestion de la prise de photo

Cette étape consiste en la mise en oeuvre d'un code qui permettrait à un potentiel utilisateur de gérer le nom de fichier, le type de capture voulu entre photo ou vidéo et le temps d'exécution. On viendrai directement enregistrer le fichier dans les documents du Raspberry. La finalité de la chose est que la code de notre intelligence artificielle aille chercher par elle même la photo d'origine et la traiter pour pouvoir afficher les deux images, celle d'origine et celle modifiée côte à côte.

Pour faire ce code, plusieurs options se présenter à nous quand au langage à utiliser et aussi la librairie pour prendre la photo. Comme ce code va faire appel à interactions entre le script et la ligne de commande lors de l'exécution, on pensait que le langage C serait une bonne solution, mais la question qui se posa fut, comment prendre des photos avec un script en langage C. Donc, pour la simplicité et la cohérence du projet, on a utilisé le langage Python. Enfin, nous avons utilisé la librairie OpenCV pour prendre la photo sur le Raspberry, ce choix a été fait à la suite d'un test non-fructueux pour transposer le code de changement de couleur de cheveux sur Raspberry. Initialement, nous avons lu dans la documentation du Rasberry PI que la libraire Python "PiCamera" est nécessaire pour prendre des photos depuis une caméra connectée sur Raspberry PI. Toutefois, lors de l'éxecution de notre code, on a eu un conflit entre les librairies de PyTorch et du Raspberry, nous avons donc opté pour la librarie OpenCV pour avoir l'homogénéité. Le code obtenu est le suivant :

```
import cv2
import time
import sys
import os
def capture_media(file_name, capture_time):
    cap = cv2.VideoCapture(0)
    if not cap.isOpened():
        print("Erreur: Impossible d'ouvrir la webcam.")
        return
    if capture_time > 1:
        capture_video(cap, file_name,capture_time)
    else:
        capture_photo(cap, file_name)
    cap.release()

def capture_video(cap, file_name,capture_time):
```

```
    # Definir la resolution de la video
    cap.set(cv2.CAP_PROP_FRAME_WIDTH, 640)
    cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 480)

    # Definir le codec et creer l'objet VideoWriter
    fourcc = cv2.VideoWriter_fourcc(*'XVID')
    out = cv2.VideoWriter(file_name + '.mp4', fourcc, 20.0, (640, 480))

    start_time = time.time()
    while (time.time() - start_time) < capture_time:  # Capturer pendant 10
        ret, frame = cap.read()
        if ret:
            out.write(frame)
        else:
            break

    out.release()

def capture_photo(cap, file_name):
    ret, frame = cap.read()
    if ret:
        cv2.imwrite(file_name + '.jpg', frame)
```

Pour mettre en place ce code python, nous avons du nous documenter sur différentes librairies. Tout d'abord, Python dispose d'une librairie qui permet d'interagir avec la ligne de commande d'un terminal. C'est la librairie "system" et "os". Nous avons aussi mis la librarie OpenCV pour la gestion de la photo et de la vidéo bien plus efficace que la librairie PiCamera. Ce code est un élément clé pour la réalisation de l'implémentation de l'intelligence artificielle sur le Raspberry car il permet à l'utilisateur d'avoir la liberté sur ce qu'il veut faire.

4.3 Implantation sur Raspberry

L'étape finale consiste a amené notre intelligence artificielle sur le Raspberry. Pour ce faire, nous avons importer le module PyTorch sur Rasperry pour que ce dernier puisse supporter notre code. Enfin, il fallut fusionner le code qui s'occupe de la gestion de photo et celui qui segmentera la photo. Le défi dans ce code final était de reussir à créer le flux de données associés à la photo pour pouvoir faire la segmentation des cheveux tout en créant en dur la photo elle même en lui associant un chemin prédéfini sur le Raspberry. Enfin, l'objectif est de renvoyer les deux photos, celle d'origine et celle modifiée. Encore une fois, l'utilisation ici uniquement d'OpenCV est obligatoire car aucune fonction du module PiCamera ne permet de créer un flux de données et de venir écrire dedans. Les résultats sont les suivants :

FIGURE 13 – Image initiale

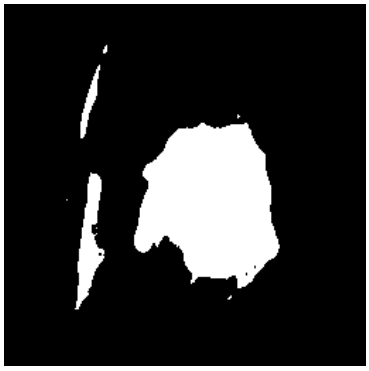

FIGURE 14 – Segmentation des cheveux

FIGURE 15 – Segmentation des cheveux et transfert de couleur avec le RaspBerry

On remarque que la segmentation des cheveux est plus que satisfaisante bien qu'on ait une segmentation d'un arrière plan du sûrement à un début d'apprentissage par coeur.

5 Transfert de Couleur

Une première approche que nous avions eu était de travailler et faire le transfert de couleur dans l'espace de couleur RGB. Voici ce que nous avions obtenu :

FIGURE 16 – Image à segmenter

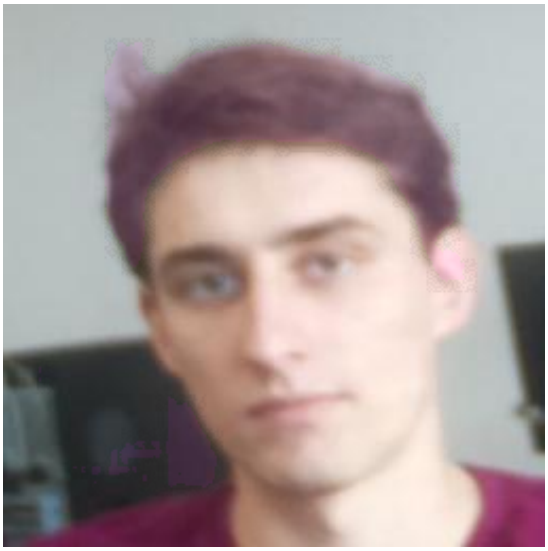

FIGURE 17 – Image obtenue par notre IA en RGB

Nous avons compris que pour des couleurs telles que le magenta et le bleu, le changement de couleur était plutôt réaliste puisque la sensibilité humaine est plutôt faible pour ce genre de longueurs d'onde. Néanmoins, pour des couleurs tels que le vert, le jaune ou même le rouge pétant, nous nous sommes rendus compte que les résultats n'étaient pas suffisants et réalistes comme représentés ci-dessous :

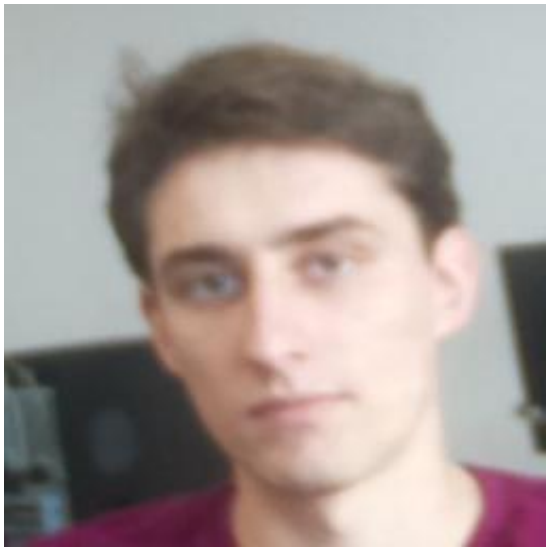

FIGURE 18 – Image à segmenter

FIGURE 19 – Image obtenue par notre IA

Comme nous pouvons le voir à la figure 16, le résultat obtenu n'est pas très concluant. Donc nous avons décidé d'abandonner cette idée et d'explorer d'autres perspectives. D'après des recherches a propos de changement de couleur sur des images, un chercheur nommé Alvy a démontré que le meilleur espace pour effectuer un changement de couleur était l'eapce HSV. En effet, Alvy Ray Smith a introduit en 1978 le modèle TSV ou HSV, offrant une transformation non linéaire de l'espace de couleur RVB des ordinateurs. Utilisé pour sa progression colorique, ce modèle est largement employé dans les applications graphiques. Il permet aux utilisateurs de choisir des couleurs en sélectionnant d'abord la Teinte sur une roue circulaire, puis la Saturation et la Valeur sur un triangle.

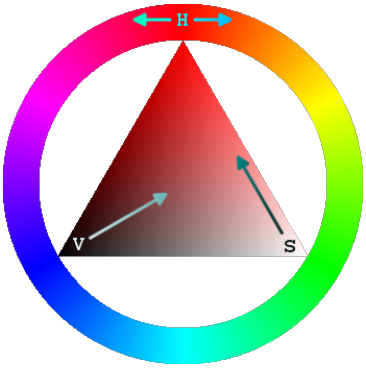

FIGURE 20 – roue de couleurs HSV

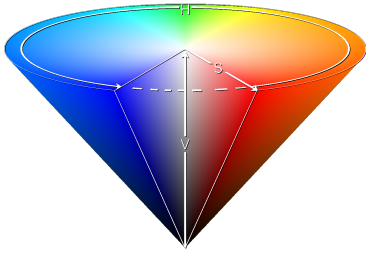

FIGURE 21 – représentation conique en HSV

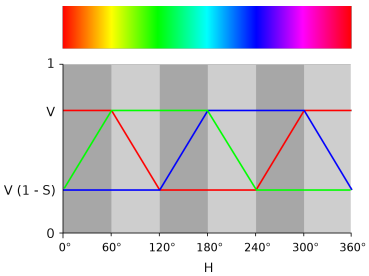

FIGURE 22 – Illustration de la relation entre les espaces de couleurs HSV et RGB.

Notons que la représentation conique est bien adaptée pour représenter tout l'espace HSV en un seul objet.

En essayant cette idée, voici les résultats obtenus :

FIGURE 23 – Image à segmenter

FIGURE 24 – Image obtenue par notre IA en HSV avec valeur de saturation maximale

Ce que l'on peut dire c'est que les résultats sont plutôt convaincants puisque les contrastes sont beaucoup mieux représenter que l'espace colorimétrique RGB.

En conclusion, on continuera de travailler dans l'espace HSV.

Dernière chose, pour pouvoir trouver une valeur de translation optimale des valeurs de saturation idéale, il faudrait pouvoir vérifier l'allure des histogrammes, comme suit :

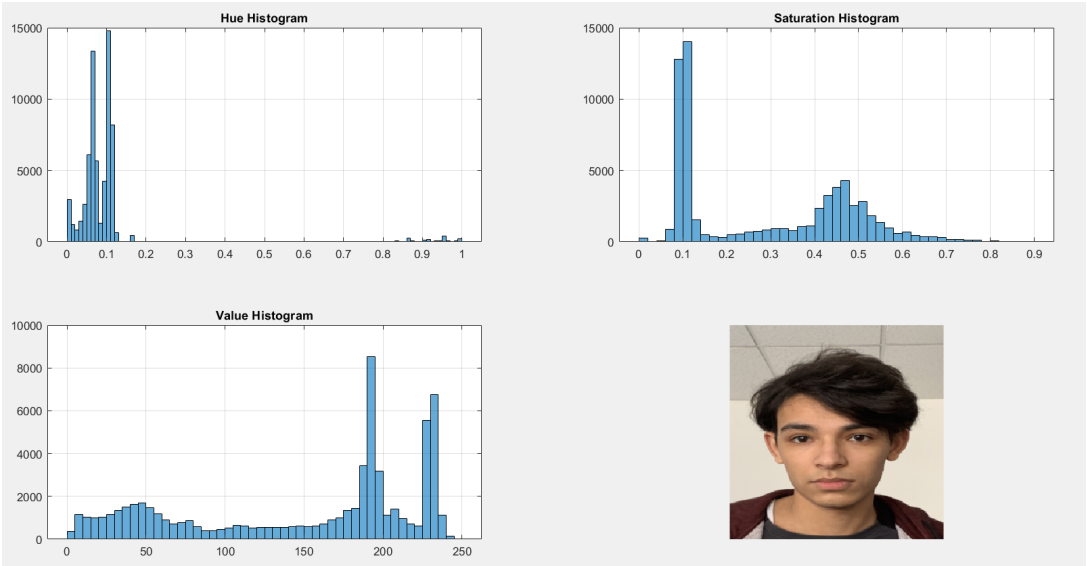

FIGURE 25 – Histogramme en HSV de l'image en bas à droite

Nous avons choisi de centrer les histogrammes autour de leur moyenne, notamment l'histogramme de saturation qui permet vraiment de modifier le contraste. Cette dernière est une moyenne calculée uniquement sur les pixels composant les cheveux, comme précisée plus haut.

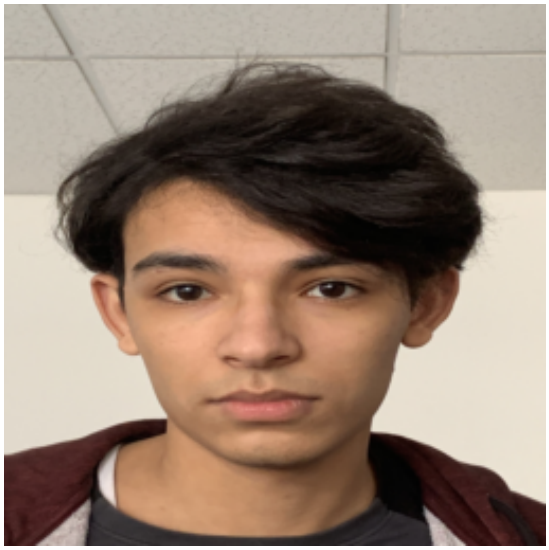

FIGURE 26 – Image à initiale

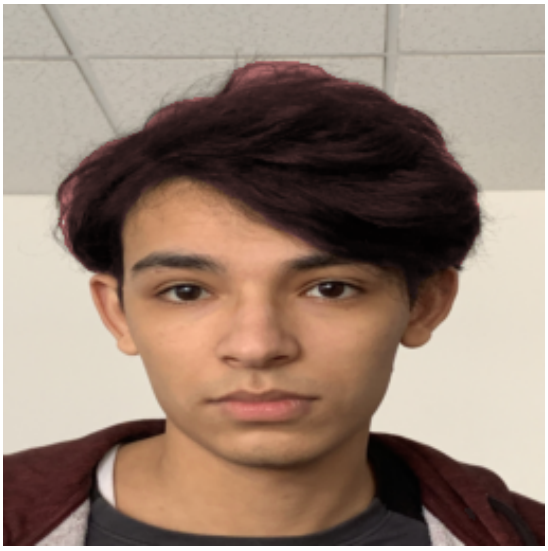

FIGURE 27 – Image obtenue par moyennage de la saturation autour de la moyenne

Néanmoins, nous nous sommes rendus compte que pour une même couleur souhaitée, les personnes brunes et blondes obtenaient des couleurs totalement différentes, comme le montre la figure ci-dessous avec les mêmes configurations que la figure ci-dessus :

FIGURE 28 – Image à initiale avec personne blonde

FIGURE 29 – Image obtenue par moyennage de la saturation autour de la moyenne

Nous avons donc caractérisé les personnes avec les cheveux noirs et claires/blonds pour mieux généraliser à tout type de cheveux car les valeurs de teinte changent en fonction de notre couleur de cheveux naturels. Pour cela, nous avions calculer la moyenne des intensités des cheveux (=Value histogram) pour des personnes ayant des cheveux foncés et clairs pour pouvoir caractériser et dissocier numériquement la couleur naturelle des cheveux pour ensuite adapter la couleur de la teinte, pour obtenir le même rendu visuel. Ainsi, nous en avons conclut que les personnes de couleur de cheveux foncés auront une

moyenne de luminosité de pixels compris entre 0 et 50, pour les cheveux châtains la moyenne sera plutôt comprise entre 50 et 100 et puis pour des cheveux clairs/blonds nous serons plutôt dans l'intervalle]100 ;255[.

Ainsi, dans cette recherche de meilleur constraste, nous avons obtenu :

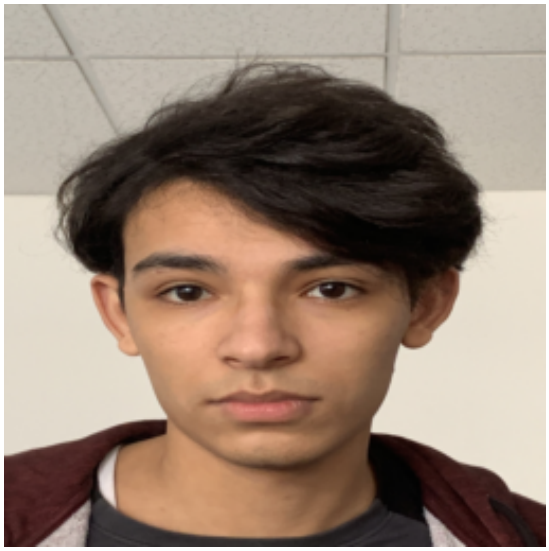

FIGURE 30 – Image à initiale avec personne brune

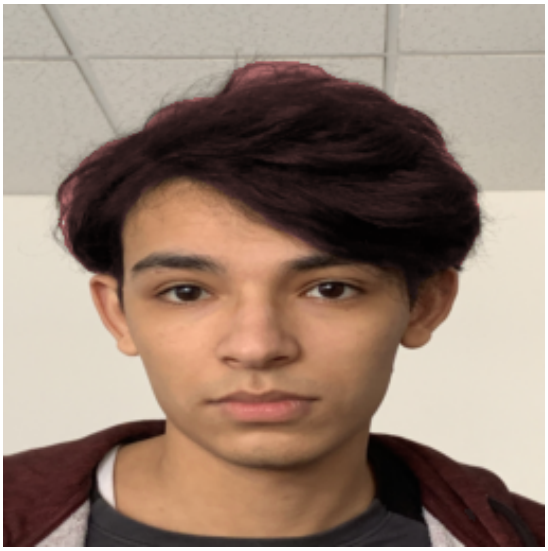

FIGURE 31 – Image obtenue en s'adaptant par rapport à la couleur de cheveux

FIGURE 32 – Image à initiale avec personne blonde

FIGURE 33 – Image obtenue en s'adaptant par rapport à la couleur de cheveux

Nous remarquons une nette amélioration du constraste au niveau des cheveux, et ceci quels que soient les types de cheveux utiliser.

La méthode utilisée est la suivante :

```
# Fonction pour ajuster la teinte, la saturation et la luminosite
def adjust_hsv(i,j, hsv, hair_color):
    # Recuperer les composantes HSV
```

```
    h, s, v = hsv[i,j,0], hsv[i,j,1], hsv[i,j,2]

    # Ajuster en fonction de la couleur initiale des cheveux
    if hair_color == 'brun':
        # Ajuster la teinte, la saturation et la luminosite pour les cheveu
        # Exemple : pas de modification de teinte, augmenter la saturation
        h_adjusted = np.mod(h, 180)  # 180 est la plage de teinte dans Open
        s_adjusted = np.clip(s * 1.2, 0, 255)  # Clip pour s'assurer que la
        v_adjusted = np.clip(v * 0.9, 0, 255)  # Clip pour s'assurer que la
    elif hair_color == 'chatain':
        # Ajuster la teinte, la saturation et la luminosite pour les cheveu
        # Exemple : pas de modification de teinte, saturation augmentee de
        h_adjusted = h
        s_adjusted = np.clip(s * 1.1, 0, 255)
        v_adjusted = np.clip(v * 0.95, 0, 255)
    elif hair_color == 'blond':
        # Ajuster la teinte, la saturation et la luminosite pour les cheveu
        # Exemple : pas de modification de teinte, saturation augmentee de
        h_adjusted = np.mod(h, 180)  # 180 est la plage de teinte dans Open
        s_adjusted = np.clip(s * 1.4, 0, 255)
        v_adjusted = np.clip(v * 0.8, 0, 255)


    # Combinez les composantes ajustees pour former l'image HSV ajustee
    hsv_adjusted = np.stack((h_adjusted, s_adjusted, v_adjusted), axis=-1)

    return hsv_adjusted
```

Notre programme est donc prêt à être implémenter dans une interface Web.

6 Tentative de Mise en Place de l'Interface Web

Nous avions pensé étant donné que le projet avance bien qu'il pourrait être pertinent de créer une interface web qui viendrait par l'intermédiaire de la caméra embarquée de l'ordinateur prendre une photo ou une de l'utilisateur pour la traiter devant lui. On pourrait aussi ajouter une option sur la manière dont l'utilisateur souhaite télécharger son contenu en demandant si il souhaite prendre une photo, en télécharger une ou connecter un Raspberry pour faire la photo via une webcam. C'était ambitieux et nous ne sommes pas arrivé à notre dessein par manque de temps et de compétences en la matière.

La page web utiliserait le code HTML et CSS pour la face avant et la face arrière serait codée en Javascript, on verra plus tard que ce fut une erreur de passer par Javascript car ce n'est pas spécialement adéquat vis-à-vis de notre problème. Toute les pages crées débutent par la même syntaxe qui est la suivante :

```
    <!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
    <link rel="stylesheet" href="web.css">
    <title>TITRE DE LA PAGE</title>
</head>
```

Cette syntaxe précise juste le type de document avec la langue, caractères que l'on va utiliser. Mais ce qui est essentiel c'est qu'elle indique quel style nous allons utiliser en spécifiant le fichier CSS source qui dicte le style de la page. De ce fait, dans le fichier CSS se trouve les styles pour les titres, paragraphes, labels, case de choix et autre. En somme, cela dicte l'apparence de la page.

6.1 Première Page

La première page est en somme très classique, elle demande à l'utilisateur si il souhaite prendre traiter une photo ou une vidéo, en fonction de son choix. Pour ce faire, on créer une cellule ou on définit deux boutons, la fonction "on click" de HTML qui permet de créer une action est utilisé ici pour dérouter sur des pages annexes en fonction du choix. La page se présente comme ci-dessous avec son code source associée :

```
<body>
<div class = "container">
    <h1>Projet Thematique</h1>
        <label for = "Filetype">Choisissez un type de fichier</label>
        <button type="button"  value = "photo" onclick =
        "window.location.href=./Photo.html>
        <button type="button" value="video" onclick
        ="window.location.href = ./Video.html>
</div>
</body>
```

Le "body" signifie que l'on commence un nouveau paragraphe et le "div" signifie que l'on créer une case. Cette case va être associé au à la classe container de notre fichier CSS comme l'indique le "class = container". Ainsi avons nous le résultat associé à cette page :

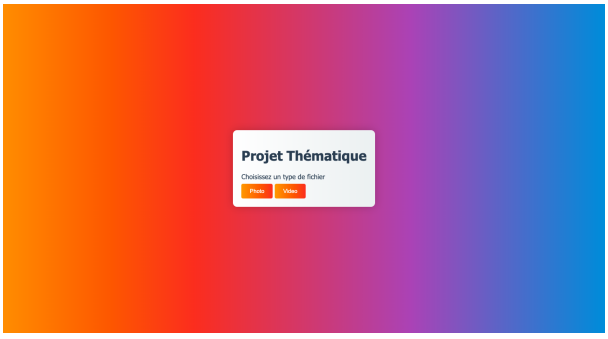

FIGURE 34 – Acceuil de la page Web

6.2 Deuxième Page

Le but de cette deuxième page est de demander à l'utilisateur comment il souhaite télécharger sa photo par trois choix, caméra, téléchargement ou connexion à Raspberry. Nous avons pu étoffer que la première option. Pour ce faire, on va utiliser un menu

défilant avec un choix unique qui est donné par la fonction en "toggleNameInput" codée en JavaScript et donné par la code suivant :

```
document.getElementById("fname").style.display = "block";
function toggleNameInput(type) {
    if(type.value == '1'){
        document.getElementById("fname").style.display = "block";
    }
    else{
        document.getElementById("fname").style.display = "none";
    }
}
```

Cette fonction doit actualiser le menu en fonction du en fonction du choix de l'utilisateur, c'est à dire, si il veut prendre une photo avec sa caméra, il doit d'abord rentrer un nom de fichier adéquat. Si il veut télécharger une photo on lui permet de choisir parmi ses fichiers etc. Pour pouvoir interagir entre la fonction et le code HTML, on donne une valeur à chaque choix, ce qui permet de faire la disjonction entre chaque choix comme vu précédemment. Le code HTML pour les valeurs est le suivant :

```
<div class = "container">
    <h1>Parametre pour la photo</h1>
    <select name = "mcharge" id="mcharge" onchange="toggleNameInput(this);"
    <label for="uploadname">Facon de telechargement</label><br>
    <option value = "1">Prendre une photo avec la camera</option>
    <option value = "2">Charger une photo</option>
    <option value = "3">Prendre une photo avec Rasberry</option>
    </select>
```

Lorsque le choix est sélectionné, on fait appel à une autre fonction en Javascript qui va nous permettre de rentrer un nom pour notre fichier si le choix 1 est sélectionné. Il faut que cette fonction permette de soumettre le nom du fichier si le champ est rempli et accède à la page suivante et interdit toute action si le champ n'est pas rempli. La fonction en JavaScript se présente comme tel :

```
function submitForm()
{
var fname = document.getElementById("fname").value;
var mcharge = document.getElementById("mcharge").value;
alert(fname);
if (fname == "")
        {
    alert("Veuillez Remplir le Champ")
        return false;
        }
alert("ok");
if(mcharge == "1")
    {
        document.location.href = "./Photoself.html";
    }
}
```

L'idée dans ce code est de venir récupérer la valeur rentrée dans le champ, si la valeur rentrée n'est pas correct, on signal à l'utilisateur avec la fonction "alert" qu'il doit rentrer une valeur. Lorsque le champ est juste, on regarde le choix fait par l'utilisateur et on le réoriente vers la page web associé à son choix. Cette fonction est appelée en HTML par l'intermédiaire d'un bouton, le code est le suivant :

```
<div class = "info-box">
    <label for = "fname">Veuillez rentrer le nom de votre
    fichier</label>
    <input type = "text" id = "fname" name="fname" value = "" /><br>
    <div class = "button">
    <input type = "button" id = "button" value = "Soumettre" onclick
    ="submitForm()" /><br>
    </div>
</div>
```

On a donc le résultat suivant :

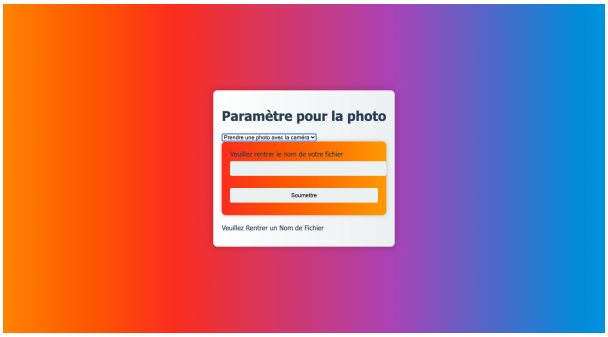

FIGURE 35 – Choix de la photo

Suite à cela, nous voulions mettre en place des algorithmes Python qui nous permettrait de faire des choses que Javascript ne peut pas faire comme la prise de photo mais aussi de relier notre réseau de neurones sur Python à notre page web pour que l'utilisateur puisse modifier sa couleur de cheveux. Sur papier cela semble plutôt simple mais en réalité cela dépassait largement nos capacité car relier un code Python à une interface web necessite de créer ce qu'on appelle un API. Un API (Applicative Programmation Interface) est essentiellement un ensemble de règles et de protocoles qui permettent à différents logiciels de communiquer entre eux. Cela permet aux développeurs d'intégrer des fonctionnalités prêtes à l'emploi dans leurs propres applications, favorisant ainsi la réutilisation du code, la modularité et l'interopérabilité entre les systèmes. La création d'un API depassait largement nos compétences, nous avons préferrer abandonner pour étayer notre projet.

7 Conclusion

Pour conclure, nous souhaitons exprimer notre gratitude envers Monsieur Rémi Giraud pour son encadrement et ses conseils précieux tout au long du projet. En ce qui concerne le projet lui-même, nous identifions une opportunité d'amélioration concernant le transfert de couleur, qui se révèle peu robuste. De plus, nous avons rencontré des

contraintes de temps et de connaissances, ce qui nous a parfois empêchés de finaliser la page web selon nos aspirations, telles que décrites précédemment.