Scientific report

Anass BOUKHEMS, Abderrahim ELHAZMI, Artem BUZLUKOV, Kawtar ABIDINE, Arthur TODER, Jean-Baptiste DEROUET, Sébastien BATTY

Advanced Project S9 - Signal and Image Processing

ENSEIRB-MATMECA

January 22, 2023

Keywords: Deep Learning, Hair Segmentation, Texture Analysis, Python, Raspberry PI, Flask

ABSTRACT

This project aims to use the U-Net architecture to create a real-time hair segmentation and color change application. The color change technique used is the HSV technique for realistic results. The application will be able to run on Raspberry PI devices, and a Flask interface on a Web application, or on a computer with a graphics card to enhance inference performance. It is important to note that this project is inspired by previous work published in scientific articles on the use of the U-Net architecture for image segmentation and color change application.

1 INTRODUCTION

This scientific project aims to use advancements in deep learning to create a real-time hair color change application. By using neural networks for hair segmentation and color transfer in video, we aim to offer an interactive experience for users who want to try different hair colors without permanent commitment. The growth of deep learning usage in computer vision applications and the increasing computational power of portable devices make this topic an area of great importance. However, there are still gaps in the optimization of performance in the development of such embedded system applications. Our main goal is to develop a complete deep learning-based pipeline for hair segmentation and color transfer in video, targeting real-time results with moving hair segmentation and integrating it into a lightweight embedded application.

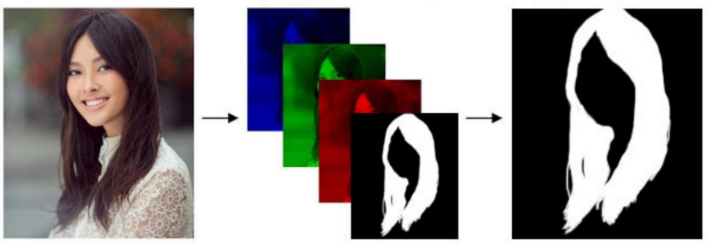

Figure 1. Hair segmentation on a colour image

To accomplish this project, our group consists of 7 engineering students in Artificial Intelligence and Image Processing, all with a background in machine learning, image processing, and computer programming. We will use deep learning techniques for image and video analysis, and then test our pipeline on hair testing dataset. We will use U-Net architecture for hair segmentation and HSV technique for realistic color change results. The final deployment of the application on a lightweight device such as a Raspberry PI equipped with a camera will allow for practical and interactive use, also a Flask web application is available. Neverthless a computer with a graphics card improves the inference performance in terms of FPS. This project is inspired by previous works published in scientific articles on the use of deep learning for image segmentation and color change application. In summary, we hope to contribute to the optimization of these deep learning applications on embedded systems.

2 STATE OF THE ART

We first started by making an inventory of the type of architecture, dataset and implementations currently used in the real-time image processing of human hair.

The article *"Hair detection, segmentation, and hairstyle classification in the wild"* [1] explains that detecting human hair is a challenging task because of various factors such as noise, wind, hair textures, and head movements. Additionally, there are few image databases of hair with annotated labels. The article proposes an approach based on texture analysis using CNNs to obtain a fine segmentation of hair. To achieve this, they used the "Figaro1k" multi-class image database for human hair detection, which contains 1050 images of people that are divided into 7 different classes (straight, wavy, curly, kinky, braids, dreadlocks, short) with the corresponding hair mask, which was segmented manually. They were able to obtain satisfactory results.

Figure 2. Figaro1k database (1050 images)

The article describes the proposed method for detecting and segmenting hair in a given environment. The first step of detection aims to create a probability map of the presence of hair on the image. This is done by a texture-based hair classification pipeline. Then, hair segmentation is performed. Finally, the hairstyle of the person represented in the image is identified using the classes of Figaro1k (straight, wavy, curly, kinky, braids, dreadlocks or short). The article also describes the different layers, including the convolution layers, and mentions that it obtained scores of at least 80 % on the Figaro1k database.

The second article *"Real-time hair segmentation and recoloring on mobile gpus"* [2] describes a novel approach for real-time hair

segmentation and recoloring on mobile GPUs. The approach uses a neural network-based technique for hair segmentation from a single camera input that is designed for real-time, mobile application. The proposed model is small, but produces high-quality hair segmentation masks that are well-suited for AR (Augmented Reality) effects, such as virtual hair recoloring. It highlights also the importance of temporal consistency of the computed masks across frames to enable realistic hair recoloring effects in AR applications. The approach achieves real-time inference speeds on mobile GPUs with high accuracy, and has been deployed in major AR applications and is used by millions of users. The article also describes the network input, dataset, and ground truth annotation used in the approach.

We also consulted other articles on hair segmentation, including Cedric ROUSSET's scientific thesis [3], which provides a detailed description of several distinctions to be made in the color, location and texture of hair. It also presents optimal contour and region approach techniques.
We finally carried out research on the use of the U-Net architecture [4].

In our research, we initially worked with the Tensorflow library, but we then transitioned to Pytorch as it was recommended by the two articles we were referencing and we also noticed better performance.

In our caseline, the input is a tensor of size (C,W,H)=(3,256,256), and the output is in the form of (1,256,256). The figure below illustrates an input and output of a trained model (section 4.4)

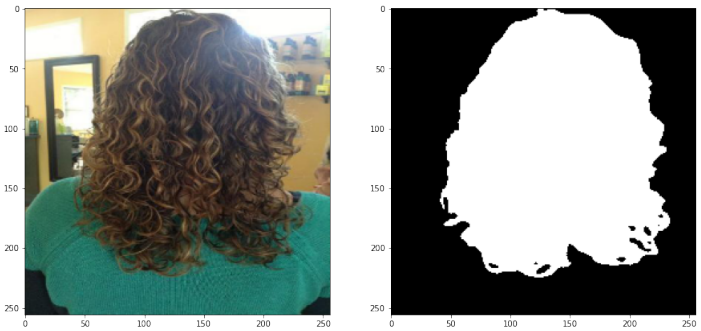

Figure 4. Input and output of U-net Architecture

3 ARCHITECTURE

3.1 U-Net

Hair segmentation is a process requiring details attention and precise segmentation. The U-Net architecture revolves around three blocks :

1. The contraction: many contraction blocks constitued from a repeated application of convolutions (3x3), each followed by a rectified linear unit (ReLU) and a max pooling operation (2x2).
2. The bottommost: links between The contraction and the expansion.
3. The expansion: many expansion blocks formed from a repeated application of Upsampling and convolutions.

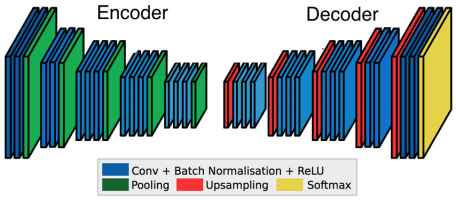

Figure 3. U-net Architecture

The contraction part is also known as the encoding part where at each downsampling step the channels are doubled, while the spatial dimension is cut in half. This process of encoding the input image helps the model learn from a lot of details in the training set as the channels expands each time with a factor of 2. **The bottommost** job is to link the encoder and decoder (Expansion) and preserve the loss from the previous layers. **The expansion** part, also known as the decoder relies on reconstructing the input that has been reducted by half in the encoder part.
Through this process of cutting down and reconstructing, through phases of converting to tensor then back to image, the model learns precise frontiers and can segment the desired region according to the input and ground truth relation.

3.2 Dataset

Dataset quality can change drastically the accuracy of model segmentation. To provide high quality data for our machine learning pipeline, we used every database that we found. In fact, we augmented Figaro1k that contains only 1050 images with different sizes (between 400 and 1000) by adding 7 images for each image by applying random rotations and mirrors (total of 8400 images). Figaro1K is a good dataset thanks to the variety of head positions that it provides unlike CELEBA dataset which have the numbers(30000 images of size 512x512) but all its images are frontal with eyes aligned which is not the only position that we want experiment on. Therefore, we applied random rotations on it, and we combined it with Figaro1k augmented. The final dataset is a large calibrated database with 38400 images & masks. (80%) of it was used for training and (20%) for validation.

Before the training and in order to emulate the motion on a video context we added some transformations as blur and motion-blur to enhance our model's performance for real-time tasks.

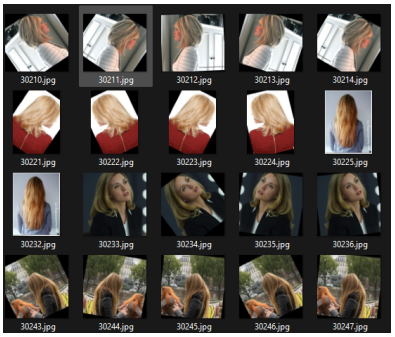

Figure 5. Figaro1k augmented

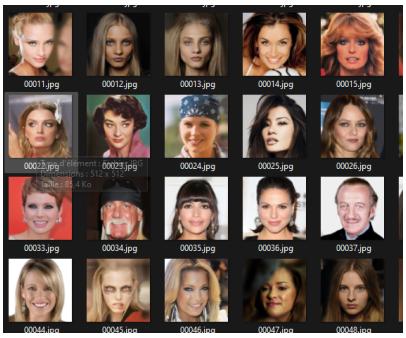

Figure 6. CELEBA dataset

3.3 Optimization

The accuracy that we got after training is great (97%) but that is a direct consequence of the number of parameters that our model uses which is $31,037,633$. In order to optimize it, we applied layer reduction technique: deleting some layers in the U-net architecture. The new model had 95% accuracy and $1,864,129$ parameters which is 1664% smaller. In order to lightweight the model, instead of using double convolution we used only one and continued reducing layers. We were deemed to have a final model with only $226,913$ parameters and 92% accuracy.

The three models work fine with GPU with a high FPS rate. However, it's not the case with CPU as shown in the table below.

Model Total params	31,037,633	1,864,129	226,913
Model Accuracy	97%	95%	92%
Device = Cuda(RTX 2070)	30.16 FPS	45.18 FPS	68.80 FPS
Device = CPU(i7 8th Gen)	1.74 FPS	2.67 FPS	10.04 FPS

3.4 Metrics

In this section we are going to evaluate the performance of the second model that has $1,864,129$ parameters. In order to achieve acceptable results the training followed this process:

- Epoch = 50
- Batch size = 16
- Loss function = BCEwithlogits(y,p)=$-(ylog(\sigma(p)) + (1-y)log(1-\sigma(p))$
- Dice$_{coeff} = \frac{(y_{ppredict} \cdot y_{true})}{(y_{predict} + y_{true})}$
- Optimiser = Adam with a fixed learning rate at 10^{-4}

Adam optimizer is known to be cost-effective with large datasets and also requires less memory and is memory-efficient, which suits our use case.

Before jumping to inference, we evaluate the hair segmentation model based on three metrics that are expressed below:

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \quad (1)$$

Where:

1. TP: True Positive is the number of objects we correctly identified in image;
2. FP: False Positive are objects we identified but there is no such object in ground-truth;
3. TN: True Negative is when algorithm doesn't identify any object;
4. FN: False Negative is when our algorithm failed to identify objects.

BCEwithlogitsloss is more numerically stable than using a plain Sigmoid followed by a BCELoss because it combines a Sigmoid layer and the BCELoss in one single class as it is shown bellow.

$$BCEwithlogits(y,p) = -(ylog(\sigma(p)) + (1-y)log(1-\sigma(p)) \quad (2)$$

where y is the validation and σ(p) is the sigmoid function applied to the prediction.

Dice: is the most commonly used metric for semantic segmentation since it considers the loss information both locally and globally, which is critical for high accuracy.

$$Dice_{coeff} = \frac{(y_{predict} \cdot y_{true})}{(y_{predict} + y_{true})} \quad (3)$$

Where y predict is the predicted image and y true is the correct image. The training and validation phase gave us two graphes shown bellow :

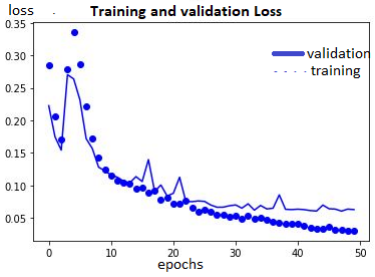

Figure 7. Training and validation loss with BCEwithlogitsloss

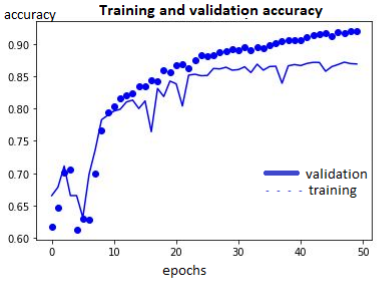

Figure 8. Training and validation accuracy

We remark that our model is not over-fitting as we can notice on the plots above. The validation loss is as low as it can achieve in our validation dataset. We can conclude that our model can segment hair in new fed images with precise accuracy. (table below)

The table below gives the metrics of our model on evaluation mode.

	BCEWithlogitsloss	Accuracy	Dice_{coeff}
Validation data	6%	86%	95%
Training data	1%	95%	98%

The Dice loss coefficient indicates that we have a loss of 5 % between the predicted image and the real one. How ever, we can say that the hair segmentation model is capable to predict any new image.

the figure below gives an idea on the output of the latest model.

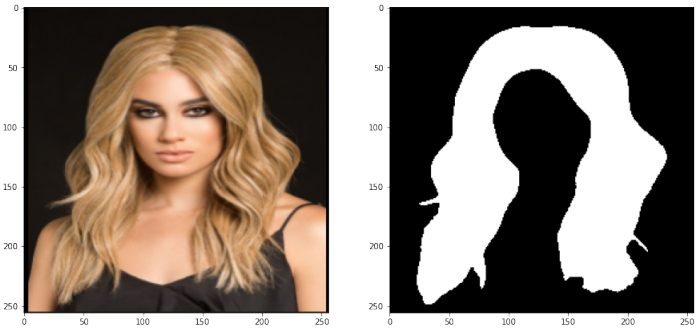

Figure 9. Input and output of the latest model

We can notice that our model can detect and segment with precision details.

4 INFERENCE

Each development of a Machine Learning model can be divided in two separate parts: Training and Inference. The first stage, in which a ML model is created or trained by running a specified subset of data into the model. ML inference is the second phase, in which the model is put into action on live data to produce actionable output.

In this section we will present what approaches we used to deploy our model on various devices and environments. The performances we were able to achieve vary depending on hardware ressources and limitations.

4.1 Deployment on PC & Raspberry Pi

As we developed our machine learning model on PC, the first logical support to try an inference was a computer. We use the camera of our own computers to get new images and videos thanks to the library OpenCV, especially ***cv2.videocapture()***. This function allow us to directly transform an input video into multiple images so we can use it as input for the inference. As we are working with a GPU, the inference's quality is really high with masks that corresponds really good to our hair's reality.

We can observe the quality of the transformation that fit to the hairs. The color change is made in the color domain HSV instead of RGB as we only want to modify the hue of the hair and keep all the change of luminosity to maintain the hair texture. The HSV model is perfect for this matter as the three channels corresponds respectively to: Hue, Saturation and Value. It means we only had to modify the first channel by the hue we wanted, defined thanks to a color written in RGB. To counter the problem of dark hair on which saturation is low and so the change of colors not that visible, we first tried to amplify the saturation of each hair pixel. However, by doing this we are losing some hair details. So, a solution would have been to higher the total saturation's mean of the hairs.

The figure below presents the inference phase on one single image. The left image is the input, the mask in the middle is the prediction of our model, the output being in the right with the colorized hair.

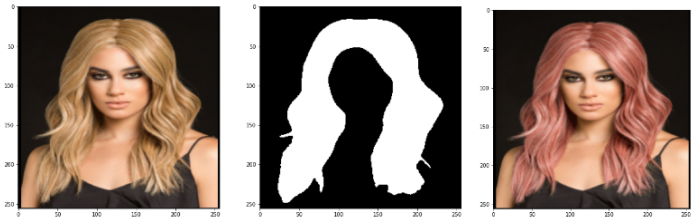

Figure 10. Hair segmentation by the first model. Left: input, middle: predicted mask, right: output with color change

The second support we used to make inference was a Raspberry PI 4 Model B. A Raspberry Pi can be likened to a mini-computer. The functioning is overall the same, except the RAM which is smaller, we got 4 GB of RAM instead of, usually 16 GB. Moreover, the inference on the Raspberry PI, beside the problems of libraries, we also had to deal with speed-processing problem, as there is no GPU to accelerate the process.

For the deployment on Raspberry PI, we looked through various libraries that could suit our needs, such as OpenCV for Raspberry PI or PiCamera. Finally, we chose the Picamera2 library, which is a recent library developed because of the move to Bullseye. It was released in September 2022 with the Raspberry PI OS.

From there we could interact with the High Quality Camera by initializing it as ***picam2 = Picamera2()***, also specifying the size of images, and such tools as the ***picam2.capture_buffer("main")*** that allowed to decompose the video input in frames for further processing. The step of combining the input frame with the predicted mask was realized with the ***picam2.set_overlay()*** function, where each channel of the overlay contained the mask with respective RGB values, as the implementation was done only using RGB color system.

Also we tried to evaluate which trained model would suit the most this application. Where the 2nd model presented really good segmentation performances, the mask prediction had a really low fps rate, meaning the hair color change was operating really slowly, and head movements would not been taken into account on real-time. For that, we decided to use the lightest model for the Rapsberry PI deployment. Some additional processing was added to the predicted mask, for example keeping the mask region with the highest area, thus eliminating other incorrectly segmented parts of the image as hair.

4.2 Deployment on Flask

Web applications are used to deploy a machine learning model. There are several frameworks available for this matter. **Flask** is widely known to be easy-to-use.

Python, Flask framework & Javascript are the backbones of the web application used for deployment of **Color Change Hair Segmentation** as Python & Flask are server-side baselines and Javascript along with HTML & CSS are client-side baselines.

The web page displays as follows:

Figure 11. Web Interface with Flask

In order to have a steady and control communication between server and client side, Flask provides ways of mapping the URLs to a specific function that handle the logic for that URL. The diagram below explains the mapping between server and client.

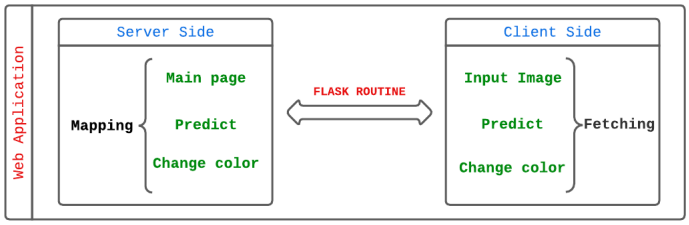

Figure 12. Server and Client connection using Flask

The upload form allows users to upload an image file, which is then processed and displayed on the page. The JavaScript is used to handle file uploads, image processing, and displaying segmented image on its side, then the colored hair to the extreme right as soon as you click on the desired color button. The script uses the FileReader API to read the image file that is uploaded and the result is displayed on the page. The Flask web application developed using the Flask framework to handle HTTP requests and responses. The backend logic is based on receiving an image file via a POST request, performing image processing on the image using the pre-trained model, and returns the segmented image. The Flask server then renders the HTML template to display the result.

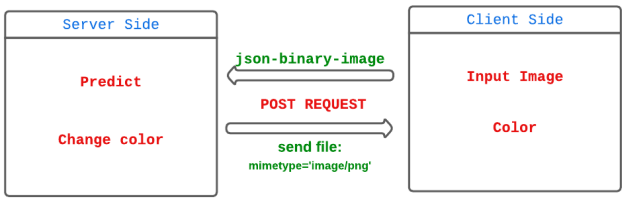

Figure 13. Sending and receiving data

The input image is in form of a base64 encoded string as part of a JSON object in the POST request. It then decodes the base64 string into binary image data using b64decode method. It then stores the image using PIL in order to feed it the pre-trained model, when the predicted image is available after computation, it is saved in a temporary file-like object, and sent back to the user as a PNG image using the Flask's send_file method. The mimetype is set to "image/png" to indicate that the response is an image in the PNG format.

5 CONCLUSION

In this paper, we started by making a state of the art of existing algorithms and architectures currently in deep learning on the detection and segmentation of human hair [1] [2] [3] [4]. This project allowed to set up a functional U-Net neural network architecture based on deep learning for hair segmentation and color transfer on images and video. We then managed to deploy the CNN on the Raspberry PI equipped with a camera to extract images in real time, and to create a graphical interface for color selection on Flask.

The U-Net shows good performance with an accuracy above 90 % thanks to the increased database with the fusion of Figaro1k and CELEBA.

There may be several possible improvements such as reducing the number of input parameters in order to obtain better real-time performance, or even the use of advanced optimization techniques (batch optimization, regularization and recurrent neural network techniques).

Facing high demand and fierce competition, advanced hair color changer app have emerged as "Perception - CV4AR/VR" developed by Google researchers in 2019 [5] at segments hair on a selfie image taken by a smartphone camera. The model, which is an encoder-decoder network with skip connections, provides better performance with an Intersection over Union (IOU) metric.

REFERENCES

[1] Riccardo LEONARDI Umar RIAZ MUHAMMAD, Michele SVANERA and Sergio BENINI. Hair detection, segmentation, and hairstyle classification in the wild. Image and Vision Computing, 2018. Accessed: 2023-01-21.

[2] Andrey VAKUNOV Yury KARTYNNIK Artsiom ABLAVATSKI Valentin BAZAREVSKY Andrei TKACHENKA, Gregory KARPIAK and Siargey PISARCHYK. Real-time hair segmentation and recoloring on mobile gpus. Retreived from `https://doi.org/10.48550/arXiv.1907.06740`, 2018. Accessed: 2023-01-21.

[3] Cedric ROUSSET. Segmentation du masque capillaire dans un visage. Retreived from `https://theses.hal.science/tel-00530635/file/These.pdf`, 2010. Accessed: 2023-01-22.

[4] ArcGIS Developers. How u-net works? Retreived from `https://developers.arcgis.com/python/guide/how-unet-works/`, 2019. Accessed: 2023-01-22.

[5] Google Research. Google research perception cv4ar/vr. Retreived from `https://sites.google.com/view/perception-cv4arvr/`, 2019. Accessed: 2023-01-22.