Deep Seasonal Color Analysis System (DSCAS)

Baraldi Francesco
256745@studenti.unimore.it

Michelini Marco
211728@studenti.unimore.it

Pagliani Matteo
259320@studenti.unimore.it

Abstract

Seasonal color analysis (SCA) is a discipline which deals with how the colors match with people's physical characteristics. According to this theory, some palettes fit a person better than others and the right matching can enhance beauty. In this report we present a comprehensive computer vision pipeline for assigning a color palette to a user, given a picture of its face, according to seasonal color analysis principles. The assigned palette, together with configurable filters, is then used to recommend to the user compatible clothing items retrieved from a dataset. The code is available at https://github.com/mrcmich/deep-seasonal-color-analysis-system

1. Introduction

According to seasonal color analysis each person can be associated with one among four palettes that correspond to the four seasons (spring, summer, autumn, winter), each of these referencing the colors of nature in that specific period of the year. Depending on some key facial characteristics, some palettes fit a person better than others and the right matching can bring out and highlight natural beauty. This theory can help people in choosing clothes and accessories, but also make-up and hair coloring: for these reasons it is widely used by stylists and models and represents the first, essential, step of image consulting.

This is not a new concept. In fact, our modern understanding of harmonious colors comes from 19th-century impressionist painters' understanding of the seasons. To accurately depict each season, they needed to understand the colors that are reflective of each one. However, it was not until the 1980s that this subject spread considerably, not only among fashion professionals but also among young people, gaining mainstream popularity.

It is not a simple analysis: it requires technique, experience and professional materials, but above all, the main challenge faced by this project is that this subject is a qualitative one. It is hard to define in an objective manner the palette that fits a person best, because personal taste, aesthetics, psychology and creativity are involved. Historically, people have been placed into their corresponding season by expert practitioners. Interestingly, these practitioners often disagree into which season a person is to place, especially in edge cases. Our goal has been therefore to transform subjective principles into objective, measurable ones and build an AI-driven consultant.

In the following sections we present our attempt to get seasonal color analysis as close as possible to a deterministic science, introducing Deep Seasonal Color Analysis System (DSCAS), a comprehensive computer vision pipeline developed to support the two parties involved in a purchase in a shop: on one side the customer can use the application to find its best matching palette given a picture of its face, while on the other side the shop assistants can recommend specific outfits tailored to the customer's palette (and so to its physical traits) from the shop database of clothing images. Hence, this project can have interesting applications in fashion design, marketing, and retailing business. As far as we know, at the time of writing this is the first comprehensive computational approach to the topic.

1.1. Color theory

SCA is built upon the Munsell color system, a colorimetry model created by Professor Albert H. Munsell in the first decade of the 20th century that specifies colors based on three properties: hue, saturation (or chroma), and value.

SCA applies Munsell's theory to human facial features and extends it with the concept of contrast. To better grasp the following considerations we provide the reader with the following definitions:

- **Hue** refers to color temperature. Cool tones are those with a blue base, warm tones are those with a red base.
- **Saturation** (or **chroma**) defines how bright or muted a color is, or in other words, how far or close to grey a color is. Neon colors have high saturation, whereas dusty colors have low saturation.
- **Value** defines how much light is reflected off of the color. Light colors have had white added to them, while dark colors have a higher percentage of black.
- **Contrast** is the level of brightness difference between two different colors.

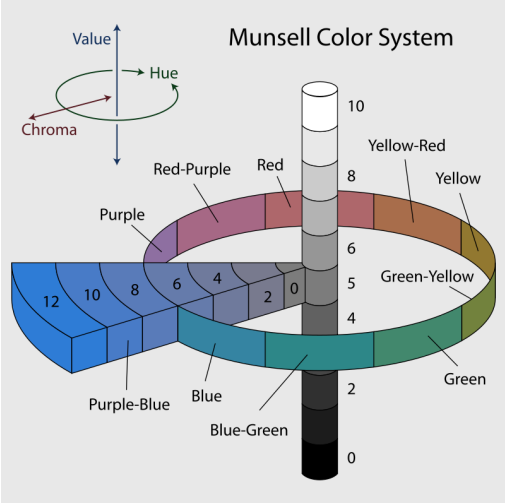

Figure 1: Munsell color theory

1.2. Palettes

The key facial features we consider to detect the matching palette are the **skin** and the **lips** undertone, the color of the **eyes** and the tone of the **hair**. According to SCA theory, popularized by Carole Jackson in her book "Color Me Beautiful" [4], seasons are divided based on four factors: hue (warm, cool), value (light, dark), saturation (bright, muted) and contrast (low, high). So, the four seasons palettes have the following properties:

Season	Hue	Saturation	Value	Contrast
Spring	warm	bright	light	high
Summer	cool	muted	light	low
Autumn	warm	muted	dark	low
Winter	cool	bright	dark	high

Table 1: Color properties for each season palette

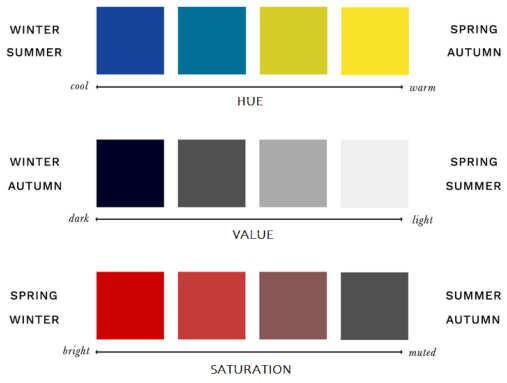

Figure 2: Hue, saturation and value across seasons

Each of these four metrics is referred to specific facial features as will be detailed in the Face palette matching section. For example, users associated with cool seasons will have a cool or bluish lips undertone, while warm seasons will have a undertone with warmer, golden shades.

Of course, not every person will fall exactly into one of the four categories. This is the reason why more sophisticated sub-categories might be identified increasing the number to twelve, but we will stick to the four categories aforementioned. Rather than computing the exact palette we will therefore compute the palette "closer" to the user according to a specific measure of distance, as will be detailed in the following sections.

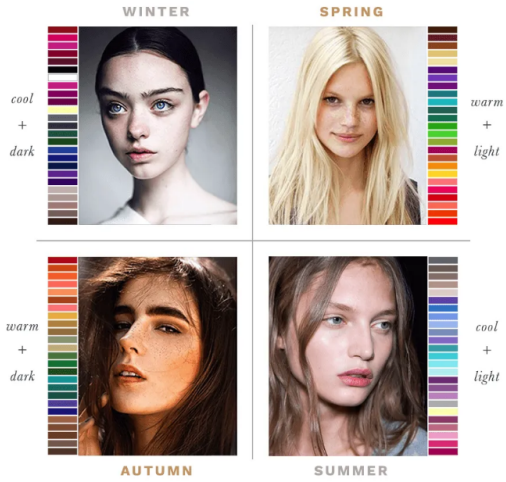

Figure 3: Examples of palette matching

The four seasons palettes considered as a reference contain 10 colors, each one represented by specific RGB values computed taking into account general principles shared by the seasonal color analysts' community. Once identified the palette of the user according to the four color metrics presented before, these are the colors (for example of clothes) that can be paired with that person to enhance natural beauty.

Figure 4: Spring palette

Figure 5: Summer palette

Figure 6: Autumn palette

Figure 7: Winter palette

1.3. Pipeline

The project follows this pipeline:

1. **Face segmentation**: acquisition of user's picture and segmentation of face salient features (e.g. skin, hair, lips, eyes) using deep neural networks.
2. **Face palette matching**: extraction of dominant colors from face salient features and association of a palette using rules derived from seasonal color analysis theory.
3. **Clothing segmentation**: segmentation of clothing item images belonging to the dress dataset in two classes (clothing, background) for background color removal.
4. **Clothing palette matching**: extraction of a variable length embedding representing the colors of each clothing item and association of a palette based on euclidean distance computation.
5. **Clothing retrieval**: recommendation of clothes from the dress dataset according to the previously computed matching palette and to the type of clothing specified by the user.

In the following sections each component of the pipeline will be analyzed. Before that, an overview about the datasets used will be given to the reader.

2. Datasets

2.1. Face segmentation dataset

The dataset [6] used for segmenting face salient features contains 16557 fully pixel-level labeled segmentation images. Facial images are included from different ethnicities, ages and genders making it a well balanced dataset. Also, there is a wide variety of facial poses and different camera angles to provide a good coverage from -90 to 90 degrees face orientations. Each label image can encode up to 11 different areas: background, lips, eyes, nose, skin, hair, eyebrows, ears, teeth, beard, specs/sunglasses. For our purposes, only skin, hair, lips and eyes were necessary.

2.2. Clothing retrieval dataset

The dataset of clothes used for retrieval is the test set of the **DressCode Dataset** [5], from which we ignored the images containing models, since we were simply interested in recommending clothing items to the user. We therefore obtained 5400 high resolution clothing item images equally divided into three different categories: dresses, upper-body clothes, lower-body clothes.

3. Approach

3.1. Face segmentation

The first element of the pipeline consists in the semantic segmentation of user's face, previously acquired by the system. As stated before, each label image contained in the face segmentation dataset can encode up to 11 different semantic areas. Hence, we will apply the segmentation process considering all of them and after that we proceed in the pipeline taking into account only skin, hair, lips and eyes. Images from the face segmentation dataset are normalized using mean and standard deviation computed on the dataset itself before passing them to a segmentation model.

For this task we decided to use deep neural networks as segmentation models. Five models of different complexity have been tested:

- **CGNet** (Context Guided Network) [10]
- **LedNet** (Lightweight Encoder-Decoder Network) [9]
- **FastSCNN** (Fast Semantic Segmentation Network) [7]
- **UNet** [1]
- **DeepLab** [2]

After several tests that will be detailed in the Results section, a low-complexity model and a high-complexity one were selected: FastSCNN and UNet respectively. The idea behind this approach is to use the lighter model for a local use on devices with average performance, while the heavier one could be put in the cloud and run on more powerful hardware. The local model uses less memory, has lower training and inference times but is a bit less accurate.

3.2. Face palette matching

Given a set of segmentation masks for a user's image, dominants color for skin, hair, lips and eyes have to be found. In fact, color is not uniform because of light reflections in the image and natural shades. In order to compute the dominant color for each of the four areas, first we computed a masked image for each area, where all pixels not in area are filled with black - represented by the RGB triplet *(0, 0, 0)* - then a k-means **clustering** algorithm with $k=3$ was applied to each masked image, giving us three candidate

dominants per area. From these, candidates too dark or too bright are removed. Finally, for each couple *(area, candidate)* we computed a reconstruction image, where all pixels in area are filled with the candidate color and all other pixels are filled with black. The dominant color for each area will be the one minimizing the RMSE (root mean square error) between the masked image and the reconstruction one over all reconstruction images for that area. When evaluating candidates, brighter colors are favored for skin, hair and lips dominant while darker colors are favored for eyes dominant: this is done by appropriately weighting the computed RMSEs. Please observe that, having decided not to consider candidates too dark or too bright as explained above, we don't consider black when selecting dominants.

Given the four extracted dominants, the following rules are then applied based on SCA principles:

- Hue represents the **undertone type**. It is computed by comparing lips dominant color with colors peach and purple: if lips color is closest to peach color then hue is warm, otherwise if lips color is closest to purple then hue is cool. The distance considered is the euclidean distance computed between the two colors in CIELab color space.
- Saturation refers to **skin color saturation**. It is set to bright if its value is above a certain threshold, it is set to muted if it is under the threshold.
- Value is defined as the mean **brightness of skin, hair and eyes**. If the person is bald then the value is computed using only skin and eyes. It is set to light if its value is above a certain threshold, it is set to dark if it is under the threshold.
- Contrast is defined as the **brightness difference between hair and eyes**. If the person is bald, this value is not computed. If the contrast level is computed, then it is set to high if its value is above a certain threshold, it is set to low if it is under the threshold.

Determining which threshold to use for each metric is not an easy task. We decided to use hand-selected thresholds based on the median values of saturation, value and contrast computed on a portion of the face segmentation dataset, as to be more robust to potential outliers with respect to the mean. After thresholding, the resulting four binary values are saved in a metric vector. Then, for each season palette, a metric vector is computed too. It contains for each metric the respective boolean value according to properties of season colors specified in Palettes section.

Finally, user's season palette is the one that minimizes the **Hamming distance** between the two metric vectors. Please note that the user's undertone computed is considered discriminant: if it is warm we will compute the distance with respect to spring and autumn only (seasons characterized by a warm undertone), while if it is cool the distance is computed with respect to summer and winter only (seasons characterized by a cool undertone).

If it wasn't possible to compute the contrast, then the distance is computed considering only the other three metrics.

3.3. Clothing segmentation

In order to assign a SCA palette to clothes it is necessary to segment the garment to distinguish its region in the image with respect to the background. This part of the pipeline has been done with classical computer vision methods. Given that clothes images from the DressCode Dataset have all the same structure, with the garment at the center of the image and a neutral background, it is not necessary to use a complex neural network for the segmentation, hence some classical operators are enough.

The process of segmentation is composed by the following steps, implemented using *OpenCV*:

- Conversion to grayscale and **thresholding** of the image with an adaptive threshold. This first step is needed to binarize the image and to find a first approximation of the edges. The adaptive threshold is computed as the Gaussian of a *7x7* window with constant factor *C=2*.
- The **edges** are computed over the thresholded image with Canny edge detector and then a dilation is applied on the resulting image in order to obtain thicker edges. This step is important because allows the cloth's contour to be more evident and connected making easier the next step.
- The **contours** of the dilated edges image are detected with a contour tracing algorithm, and only the longest contour is kept to create a new binary image that depicts only the longest one.
- Finally, a ***floodfill*** algorithm from *OpenCV* is applied to distinguish all the pixels outside the contour with respect to the pixels inside the contour. In particular, all the pixels outside the border are given the white color, while the inside ones remains black. The negative of the resulting image is the cloth segmentation mask.

3.4. Clothing palette matching

In order to retrieve only clothing items matching with the user's palette, we first need to assign the appropriate palette to each clothing item. We decided to use an approach similar to the one followed during face palette matching, but in this case clustering is applied, with $k=8$, to extract all colors present in each clothing item rather than extracting dominant colors. In this way we can compute a variable length embedding from which to assign a palette to a clothing item. Its length is variable because the clustering algorithm could find less than $k=8$ clusters (colors). For this

to work correctly it is vital that the clothing segmentation is reasonably precise, otherwise the embeddings could include the original background color, possibly influencing the palette assigned.

Given that our classical approach to clothing segmentation presented sub-optimal results in some cases, we had to tackle the problem by applying the erosion operator to the masks in order to remove potential background pixels incorrectly marked as clothing pixels in the previous step.

We then assign a palette to each clothing embedding (and thus to each clothing item) through the following steps, repeated for each palette:

- Because each palette is composed of 10 colors while embeddings contain at most 8, we first align the two by matching each color in the embedding to the color in the palette with closest hue: the remaining colors of the palette will not be used during the subsequent steps.
- For each match *(embedding color, palette color)*, we compute the euclidean distance between the two colors in CIELab space.
- We thus get a set of (at most) 8 distances, from which we take the maximum value to represent the distance between palette and embedding.

Finally, the palette assigned to the clothing embedding will be the one closest to the embedding, where "closeness" is defined in terms of the previously computed distance.

Thus we have obtained a set of static mappings linking each clothing image to its assigned palette. Obviously, it would not make much sense to compute said mappings at run-time, and for this reason we decided to compute and store on disk all clothing mappings as JSON files that can be quickly loaded in memory at run-time, divided by category of clothing.

3.5. Clothing retrieval

The recommendation system is based on the DressCode Dataset and its three types of clothes: upper-body, lower-body and dresses. The user can specify one of these classes and the system suggests garments that belong to the specified category and that match with the user's palette, previously computed. In this section the retrieval system will be explained in detail, that is the part that retrieves clothes of the type specified by the user.

We chose a text-to-image approach using the **CLIP** [8] model (Contrastive Language-Image Pre-Training) proposed by OpenAI. It is a neural network that learns how to associate language concepts with visual ones: it takes as input the text and the image and encodes them in a common embedding space in which the distance of the two embedding vectors reflects the similarity of the original inputs. This approach is flexible because it is possible to perform classification tasks as well as retrieval tasks. In our case the latter has been used.

The test set of the DressCode Dataset has been extracted assigning to each cloth image a textual class of type *"a cloth of type ⟨class⟩"*, so we obtained one textual class for each cloth. Then the retrieval task is done by giving as input to the model the textual class specified by the user, and considering the k images of the dataset with the highest cosine similarity between the text embedding and the images embeddings.

We experimented with different versions of the open source implementation of CLIP [3], all of them pre-trained on different datasets and tested on DressCode "zero-shot". Initially we evaluated the performance of the models in both image classification task and retrieval task, then we selected the best one for the retrieval part of our pipeline. We tested three models:

- ViT-B-16, pre-trained on *laion400m_e31*, *laion400m_e32* and *laion2b_s34b_b88k*
- ViT-B-32, pre-trained on *laion400m_e31*, *laion400m_e32*, *laion2b_e16* and *laion2b_s34b_b79k*
- ViT-L-14, pre-trained on *laion400m_e31*, *laion400m_e32* and *laion2b_s32b_b82k*

After several tests, detailed in the Results section, **ViT-B-32** pre-trained on *laion2b_s34b_b79k* has been selected as the best model and used for the retrieval part of the pipeline.

The final step is to suggest to the user the retrieved clothes that also match the user's palette, as explained in the Clothing palette matching section.

4. Results

4.1. Face segmentation

For each training and test phase the face segmentation dataset has been split in this way: 80% for the training set and 20% for the test set. When the validation process is considered, the training set is split further in 85% for the actual training set and 15% for the validation set.

For training and validation phases, a loss function and a score are computed and will be detailed in the following. Moreover, in order to be able to evaluate the scores while also considering that small facial features should be considered as more important than large ones, we decided to implement a weighted version of the score, which will be used only at test time. The weighted score weights the score of each class according to the area occupied in an image, such that the score is increased for small scale classes (e.g. eyes, eyebrows) and decreased for large scale classes (e.g. hair, skin). Weights were computed on the face segmentation dataset.

This phase is carried out through the following steps:

1. Demo
2. Pre-processing
3. Loss selection
4. Hyperparameters optimization (HPO)
5. Training on the full training set and test

4.1.1 Demo

We started out with a "demo" of the five models, just to have an overall idea of performance differences. Performance has been monitored throughout the process computing cross entropy loss and mIoU (mean intersection over union) score both on training and validation set.

We trained each model for 20 epochs on the training set using the following parameters:

Learning rate	Batch size	Optimizer
0.01	32	Adam

Table 2: Training parameters for "demo" configuration

Finally the performance has been measured on a test set with a weighted version of the mIoU, where weights were computed as stated before.

These are the overall results of the demo phase:

Model	T_loss	T_score	V_loss	V_score	w_mIoU
CGNet	0.071	0.690	0.108	0.681	0.684
LedNet	0.113	0.634	0.106	0.669	0.673
FastSCNN	0.088	0.654	0.131	0.641	0.645
UNet	0.072	0.684	0.140	0.681	0.686
DeepLab	0.046	0.730	0.104	0.727	0.728

Table 3: "Demo" results

Computational demand for each model has been carefully analyzed measuring training time, inference time and memory usage:

Model	T_time[min/epc]	I_time[s]	Mem[MB]
CGNet	8.3	69	6266
LedNet	7.1	53	5023
FastSCNN	3.8	57	947
UNet	6.1	58	8997
DeepLab	10.2	89	14136

Table 4: "Demo" models computational demand

We chose the best model for local use and the best model for cloud use according to the highest *performance/computational demand* ratio. **FastSCNN** was the model chosen for local use because it was by far the lightest though accurate enough, while **UNet** was the one chosen for cloud use because of good performance and acceptable computational demand. Therefore, we proceeded considering the demo results of these two models as a baseline from which to improve.

4.1.2 Pre-processing

In this phase we wanted to test how some classical computer vision operators could have impacted the results. Performance has been monitored throughout the process computing cross entropy loss and mIoU score both on training and validation set. We trained the two selected models for 10 epochs on the training set using the same parameters used during the demo phase. We started out testing **no pre-processing**:

Model	T_loss	T_score	V_loss	V_score
FastSCNN	0.134	0.603	0.131	0.618
UNet	0.150	0.571	0.160	0.594

Table 5: No pre-processing

Then we tested **color jitter**:

Model	T_loss	T_score	V_loss	V_score
FastSCNN	0.124	0.608	0.146	0.606
UNet	0.143	0.584	0.162	0.588

Table 6: Color jitter pre-processing

We tried to apply **horizontal flip**:

Model	T_loss	T_score	V_loss	V_score
FastSCNN	0.142	0.589	0.149	0.600
UNet	0.188	0.535	0.201	0.510

Table 7: Horizontal flip pre-processing

Finally we tested **center crop**:

Model	T_loss	T_score	V_loss	V_score
FastSCNN	0.129	0.615	0.142	0.617
UNet	0.160	0.580	0.168	0.594

Table 8: Center crop pre-processing

Regarding FastSCNN, we decided to proceed without pre-processing because this version showed the best results.

For UNet instead, the results were very similar; we decided to proceed with color jitter because it affects the color and therefore it has a major impact on the segmentation compared to the other pre-processing techniques.

4.1.3 Loss selection

So far we had used cross entropy as loss function. In this phase we wanted to test the weighted version of the same function. The weights used are the same as the weighted mIoU ones. We trained the two selected models for 10 epochs on the training set using the same parameters used during the demo phase and no pre-processing.

Model	T_loss	T_score	V_loss	V_score
FastSCNN	0.121	0.616	0.143	0.607
UNet	0.146	0.583	0.161	0.582

Table 9: Standard cross entropy

Model	T_loss	T_score	V_loss	V_score
FastSCNN	0.146	0.583	0.161	0.582
UNet	0.106	0.588	0.114	0.576

Table 10: Weighted cross entropy

From now on, we will consider weighted cross entropy as loss function because, although it had a slightly worse score, it showed better results for classes with less pixels, such as eyes and lips, which are very important for our purposes.

4.1.4 Hyperparameters optimization (HPO)

We decided to apply a hyperparameter grid search among some values of learning rate and batch size, testing also the contribute of a learning rate scheduler. The optimizer has been fixed to Adam. This results in 16 combinations for each model. We trained each combination for 10 epochs on the training set, evaluating loss with weighted cross entropy and using no pre-processing. Exhaustive HPO results can be found in Table 14.

The more promising combination for FastSCNN is:

Lr	Batch size	Optimizer	Lr_scheduler
10^{-3}	16	Adam	None

Table 11: FastSCNN best hyperparameter combination

The more promising combination for UNet is:

Lr	Batch size	Optimizer	Lr_scheduler
10^{-4}	16	Adam	None

Table 12: UNet best hyperparameter combination

Training and validation loss and score during HPO for both models are shown in section Plots.

4.1.5 Training on the full training set and test

Finally, we trained each model best hyperparameter combination for 20 epochs on the whole training set and evaluating loss with weighted cross entropy. As pointed out before, we chose to use no pre-processing for FastSCNN and color jitter for UNet. Besides that, we applied a bilateral filter on training images for both models, as a way of smoothing images while preserving edges.

Model	Train loss	Train score
FastSCNN	0.052	0.671
Unet	0.035	0.748

Table 13: Final training results

We tested both models on the test set evaluating IoU scores for each of the 11 masks available. Results can be found in Table 15. These are the plots of loss and score during training on the full training set of FastSCNN:

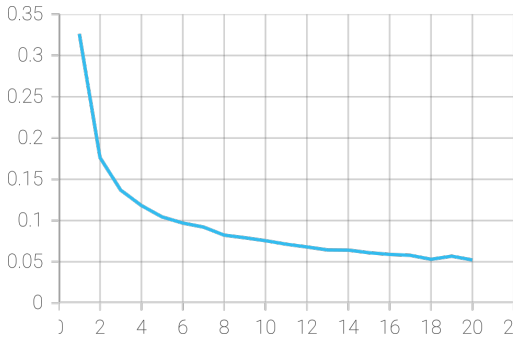

Figure 8: Training loss during full training of FastSCNN model

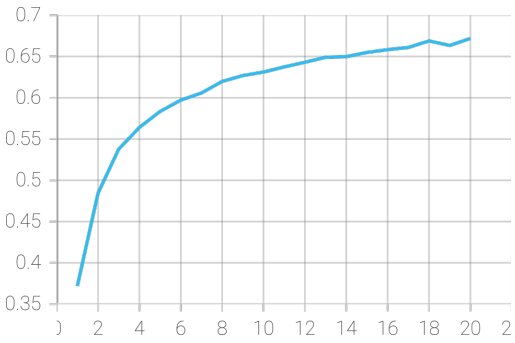

Figure 9: Training score during full training of FastSCNN model

These are the plots of loss and score during training on the full training set of UNet:

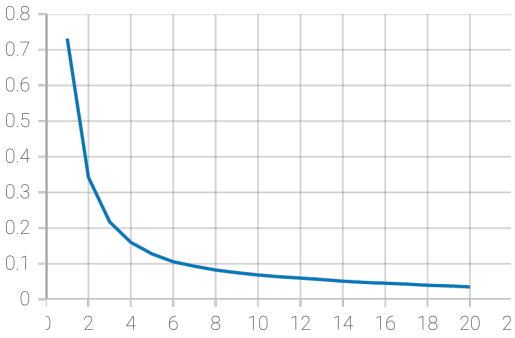

Figure 10: Training loss during full training of UNet model

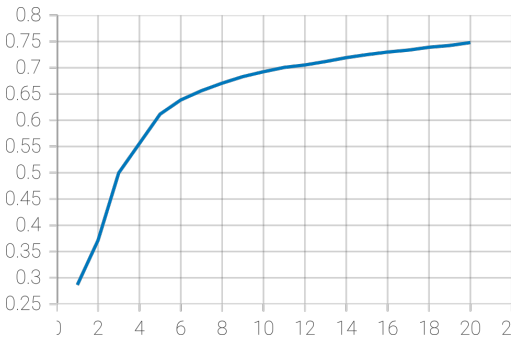

Figure 11: Training score during full training of UNet model

We can state that performance is increased for both models with respect to the demo test considered as a baseline. FastSCNN and UNet weighted mIoU have improved by 3.3% and 5.1% respectively.

This is an example of the face segmentation process using the trained UNet model on a real person:

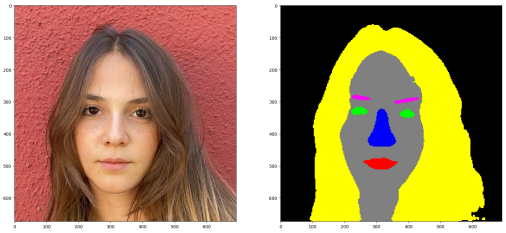

Figure 12: Input image and face segmentation phase output

Going ahead presenting the pipeline results we will use this exemplifying user to show all the steps from face segmentation to the final retrieval process.

4.2. Face palette matching

Given the set of segmentation masks, dominants color for skin, hair, lips and eyes are computed using *k-means*. The dominant color for each area will be the one minimizing the RMSE between the masked image and the reconstruction one over all reconstruction images for that area.

Let's analyze for example the output for the hair. The first candidate is represented by the RGB values *(134, 97, 80)* and has a weighted reconstruction error of 44.15.

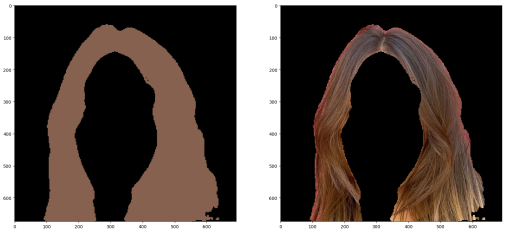

Figure 13: First cluster (color) identified and real hair color

The second candidate is represented by the RGB values *(93, 59, 40)* and has a weighted reconstruction error of 61.26.

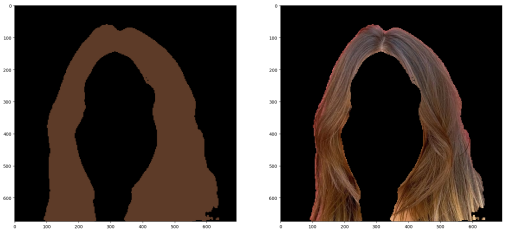

Figure 14: Second cluster (color) identified and real hair color

The third candidate, represented by the RGB values *(0, 0, 0)*, is not considered as a potential dominant due to being too dark and thus is not shown. So, the candidate for the hair which minimizes the weighted reconstruction error is the first one.

An analogous process is done for skin, eyes and lips. After these steps we obtain the following dominants color for each area:

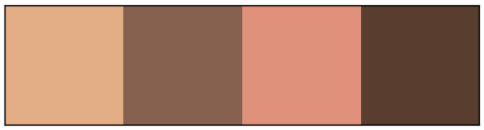

Figure 15: Dominants color for skin, hair, lips and eyes (from left to right in this order)

Then, SCA rules are applied and we obtain this binary vector: *(1, 0, 1, 0)*. This means that the computed undertone is warm, saturation is low (muted), value is high (bright) and contrast is low. Then, this vector is compared with the two seasons palettes metric vectors (previously computed) which are characterized by warm undertone (spring and autumn). The season palette given as output is **autumn** because it has a lower Hamming distance.

Interestingly, autumn corresponds to the palette associated to the real person from a professional image consultant after an accurate analysis.

4.3. Clothing segmentation

In order to perform the retrieval process we had to segment the clothes. This is an example of the intermediate steps that lead to the final segmentation of a dress using classical computer vision methods:

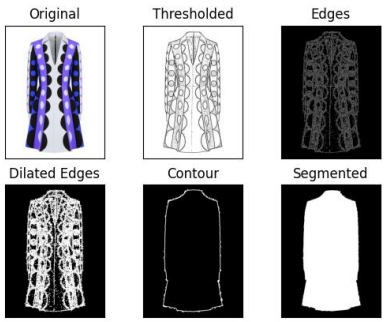

Figure 16: Intermediate steps of clothing segmentation

4.4. Clothing retrieval

Several versions of pre-trained open CLIP model have been tested on the test set of DressCode Dataset in order to select the best performing one for our purposes. We evaluated both the image classification task and the text-to-image retrieval task, which is the one we are more interested in, though the image classification performance can be a useful indicator about the performance of the model too. The retrieval accuracy is computed with a threshold of 0.5 on the similarity scores as:

$$acc = \frac{TP + TN}{TP + TN + FP + FN}$$

Given the results, which are detailed in Table 16, we selected **ViT-B-32** pre-trained on *laion2b_s34b_b79k* as the best model. Although it is not the best in classification, it has a high retrieval accuracy and with reasonable inference times, which is an important aspect to consider because the recommendation process has to be quick.

Coming back to our exemplifying user, we retrieve clothes belonging to the autumn palette and matching with the user's query *"a cloth of type dress"*, completing the pipeline.

Figure 17: First ten matching clothes retrieved

5. Future works

This project could certainly benefit from the advice of a professional image consultant. The rules used to assign a palette to the user could be refined and extended to take into account season subcategories.

Additional improvements could be gained by experimenting with higher-quality data for the face segmentation part and different embeddings for clothing items.

The retrieval process could become practically useful if combined with a dataset in which labels are specific kinds of clothes: jumpers, trousers etc.

Then the pipeline could be completed with a virtual try-on phase, in which retrieved clothes can be virtually tried by the users given a full body image. This would be icing on the cake for e-commerce applications.

Another interesting idea is to implement augmented reality capabilities, such as a live video stream recorded from users' phone in a clothing store, showing in real time if the framed clothes match with the user's palette.

6. Conclusions

As far as we know, this is the first comprehensive experiment of building an AI-driven image consultant built upon seasonal color analysis principles. After several tests we can

affirm that the system assigns reasonable season palettes to the user and recommends compatible clothes that respect the user's query. The tests were done either using famous actress images, of which the palette was known, or real users who previously had had their palette assigned by a professional image consultant.

This field is growing more and more in popularity and we think that, with appropriate enhancements, interested people could take advantage of this system.

References

[1] Mateusz Buda, Ashirbani Saha, and Maciej A Mazurowski. Association of genomic subtypes of lower-grade gliomas with shape features automatically extracted by a deep learning algorithm. *Computers in Biology and Medicine*, 109, 2019.

[2] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *CoRR*, abs/1606.00915, 2016.

[3] Gabriel Ilharco, Mitchell Wortsman, Ross Wightman, Cade Gordon, Nicholas Carlini, Rohan Taori, Achal Dave, Vaishaal Shankar, Hongseok Namkoong, John Miller, Hannaneh Hajishirzi, Ali Farhadi, and Ludwig Schmidt. Openclip, July 2021. If you use this software, please cite it as below.

[4] Carole Jackson. *Color Me Beautiful: Discover Your Natural Beauty Through the Colors That Make You Look Great and Feel Fabulous*. 1973.

[5] Davide Morelli, Matteo Fincato, Marcella Cornia, Federico Landi, Fabio Cesari, and Rita Cucchiara. Dress code: High-resolution multi-category virtual try-on. In *Proceedings of the European Conference on Computer Vision*, 2022.

[6] Mut1ny. Face/head segmentation dataset community edition.

[7] Rudra P. K. Poudel, Stephan Liwicki, and Roberto Cipolla. Fast-scnn: Fast semantic segmentation network. In *British Machine Vision Conference*, 2019.

[8] Alec Radford, Jong Wook Kim, Chris Hallacy, A. Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In *ICML*, 2021.

[9] Yu Wang, Quan Zhou, Jia Liu, Jian Xiong, Guangwei Gao, Xiaofu Wu, and Jan Longin Latecki. Lednet: A lightweight encoder-decoder network for real-time semantic segmentation. *arXiv preprint arXiv:1905.02423*, 2019.

[10] Tianyi Wu, Sheng Tang, Rui Zhang, Juan Cao, and Yongdong Zhang. Cgnet: A light-weight context guided network for semantic segmentation. *IEEE Transactions on Image Processing*, 30:1169–1179, 2020.

Tables

We add here large tables to provide a better visualization.

Model	Lr	Batch size	Optimizer	Lr_scheduler	T_loss	T_score	V_loss	V_score
FastSCNN	10^{-5}	16	Adam	None	0.410	0.356	0.384	0.358
	10^{-5}	16	Adam	Linear	0.445	0.355	0.416	0.356
	10^{-4}	16	Adam	None	0.133	0.516	0.139	0.536
	10^{-4}	16	Adam	Linear	0.156	0.484	0.152	0.502
	$\mathbf{10^{-3}}$	**16**	**Adam**	**None**	**0.078**	**0.626**	**0.088**	**0.639**
	10^{-3}	16	Adam	Linear	0.087	0.612	0.089	0.625
	10^{-2}	16	Adam	None	0.092	0.602	0.096	0.616
	10^{-2}	16	Adam	Linear	0.095	0.606	0.098	0.601
	10^{-5}	32	Adam	None	0.484	0.355	0.463	0.354
	10^{-5}	32	Adam	Linear	0.547	0.346	0.517	0.349
	10^{-4}	32	Adam	None	0.149	0.485	0.159	0.493
	10^{-4}	32	Adam	Linear	0.174	0.458	0.174	0.471
	10^{-3}	32	Adam	None	0.078	0.623	0.092	0.616
	10^{-3}	32	Adam	Linear	0.085	0.609	0.095	0.614
	10^{-2}	32	Adam	None	0.085	0.618	0.090	0.616
	10^{-2}	32	Adam	Linear	0.085	0.622	0.099	0.622
UNet	10^{-5}	16	Adam	None	0.373	0.349	0.341	0.367
	10^{-5}	16	Adam	Linear	0.461	0.291	0.430	0.296
	$\mathbf{10^{-4}}$	**16**	**Adam**	**None**	**0.066**	**0.685**	**0.071**	**0.698**
	10^{-4}	16	Adam	Linear	0.074	0.672	0.081	0.662
	10^{-3}	16	Adam	None	0.075	0.656	0.086	0.662
	10^{-3}	16	Adam	Linear	0.068	0.687	0.086	0.686
	10^{-2}	16	Adam	None	0.094	0.595	0.102	0.608
	10^{-2}	16	Adam	Linear	0.068	0.667	0.086	0.633
	10^{-5}	32	Adam	None	0.551	0.248	0.528	0.268
	10^{-5}	32	Adam	Linear	0.612	0.247	0.590	0.257
	10^{-4}	32	Adam	None	0.078	0.682	0.081	0.686
	10^{-4}	32	Adam	Linear	0.097	0.617	0.098	0.645
	10^{-3}	32	Adam	None	0.078	0.653	0.091	0.633
	10^{-3}	32	Adam	Linear	0.069	0.690	0.079	0.681
	10^{-2}	32	Adam	None	0.082	0.623	0.090	0.653
	10^{-2}	32	Adam	Linear	0.081	0.643	0.087	0.634

Table 14: HPO combinations

Model	mIoU	w_mIoU	bground	lips	eyes	nose	skin	hair	brows	ears	teeth	beard	specs
FastSCNN	0.698	0.678	0.953	0.510	0.386	0.706	0.852	0.788	0.384	0.643	0.679	0.871	0.909
Unet	0.753	0.737	0.958	0.647	0.568	0.793	0.879	0.811	0.554	0.725	0.768	0.732	0.852

Table 15: Face segmentation test results

Model	Pre-trained	Class. accuracy (%)	Retrieval accuracy (%)	Inference time (s)
ViT-B-16	laion400m_e31	90.06	66.67	250
	laion400m_e32	89.22	66.67	
	laion2b_s34b_b88k	86.00	66.68	
ViT-B-32	laion400m_e31	90.70	66.69	**160**
	laion400m_e32	90.67	66.67	
	laion2b_e16	81.00	66.67	
	laion2b_s34b_b79k	**85.81**	**66.73**	
ViT-L-14	laion400m_e31	88.96	69.56	822
	laion400m_e32	88.98	69.35	
	laion2b_s32b_b82k	73.17	69.29	

Table 16: Retrieval models test results

Plots

We add here large plots to provide a better visualization.

FastSCNN

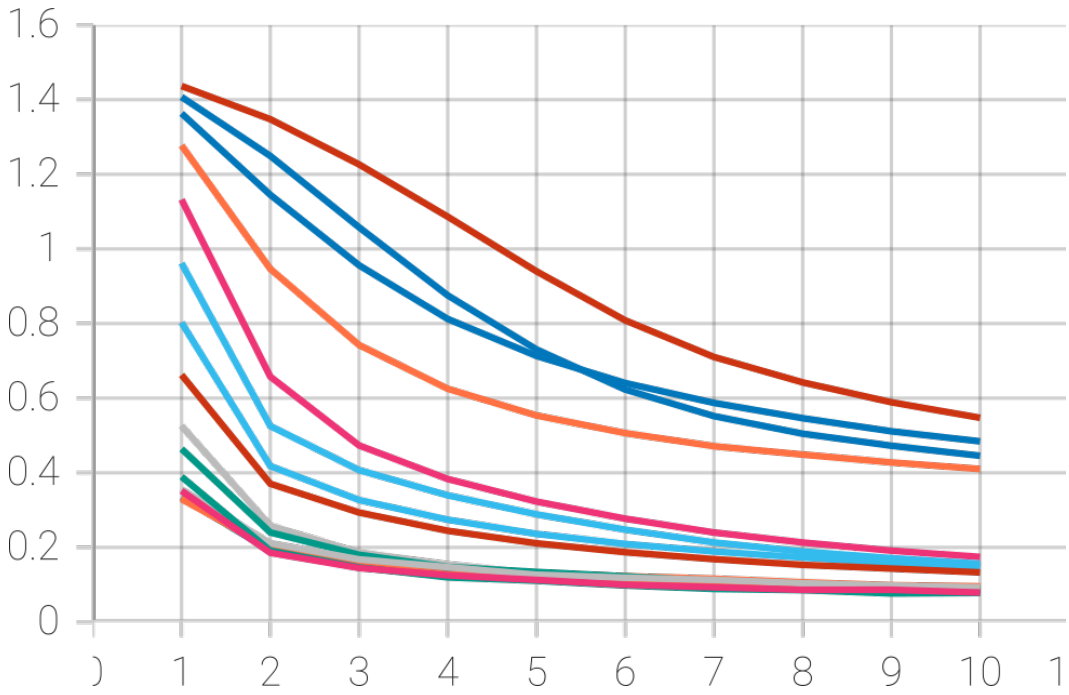

Figure 18: Train loss during HPO of FastSCNN model

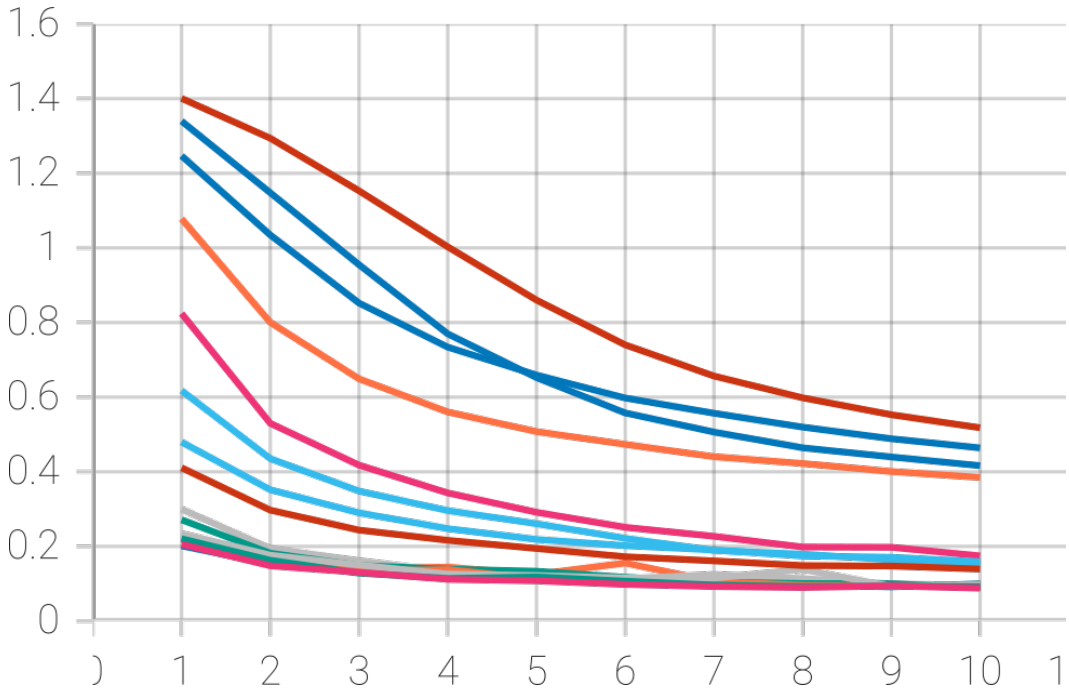

Figure 19: Validation loss during HPO of FastSCNN model

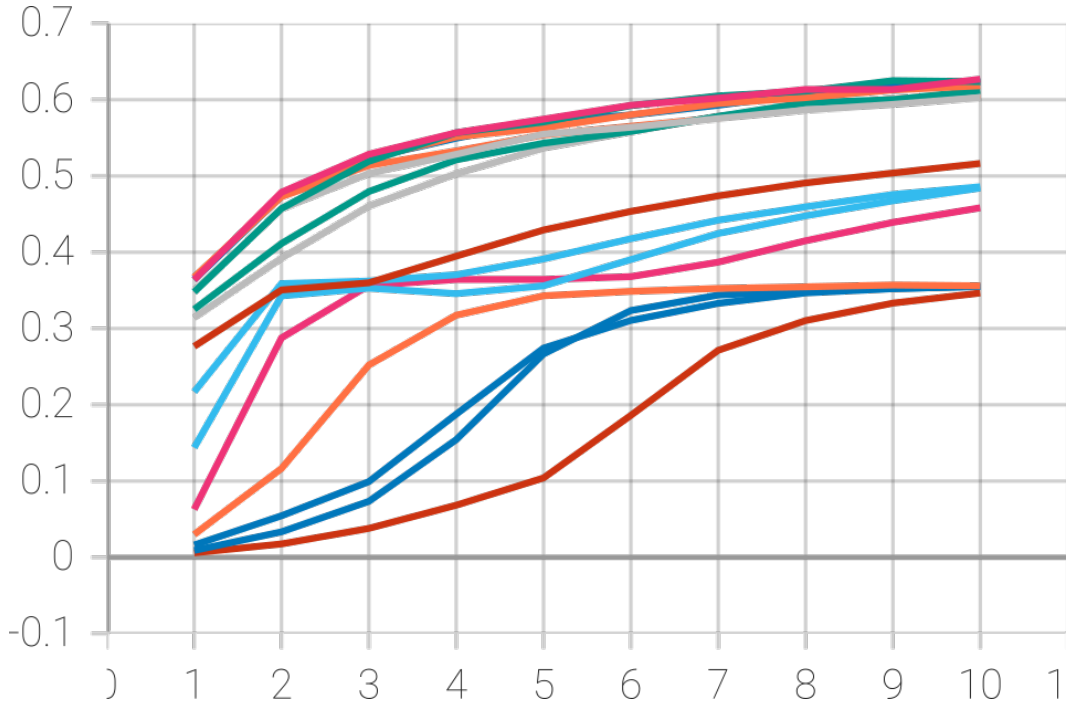

Figure 20: Train score during HPO of FastSCNN model

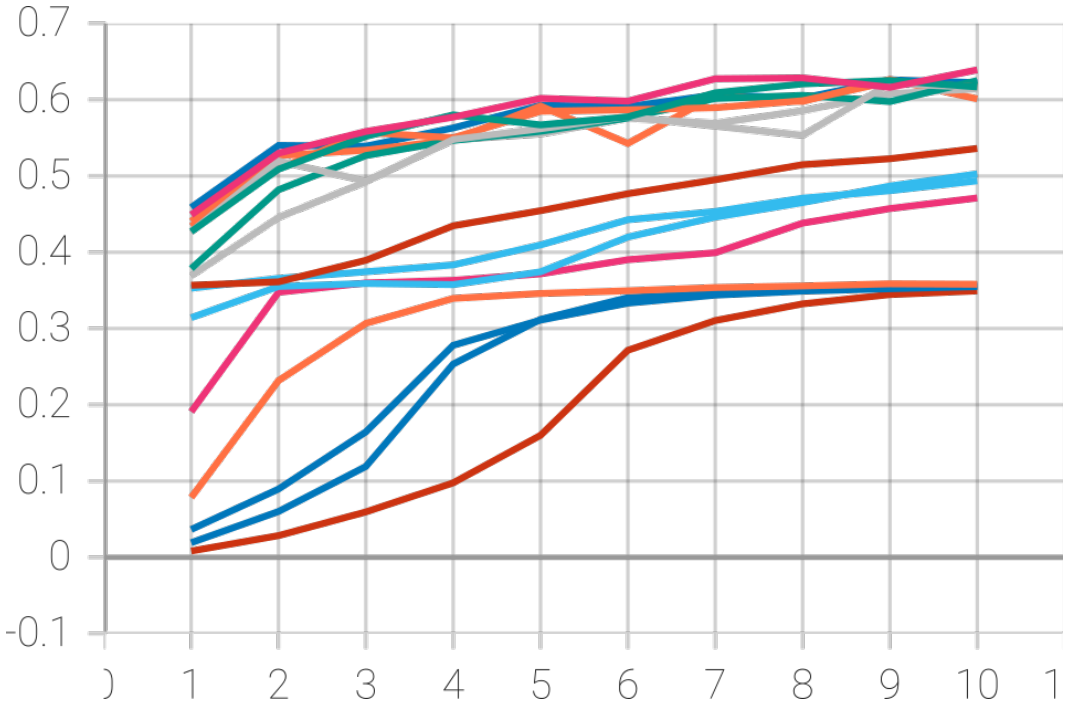

Figure 21: Validation score during HPO of FastSCNN model

UNet

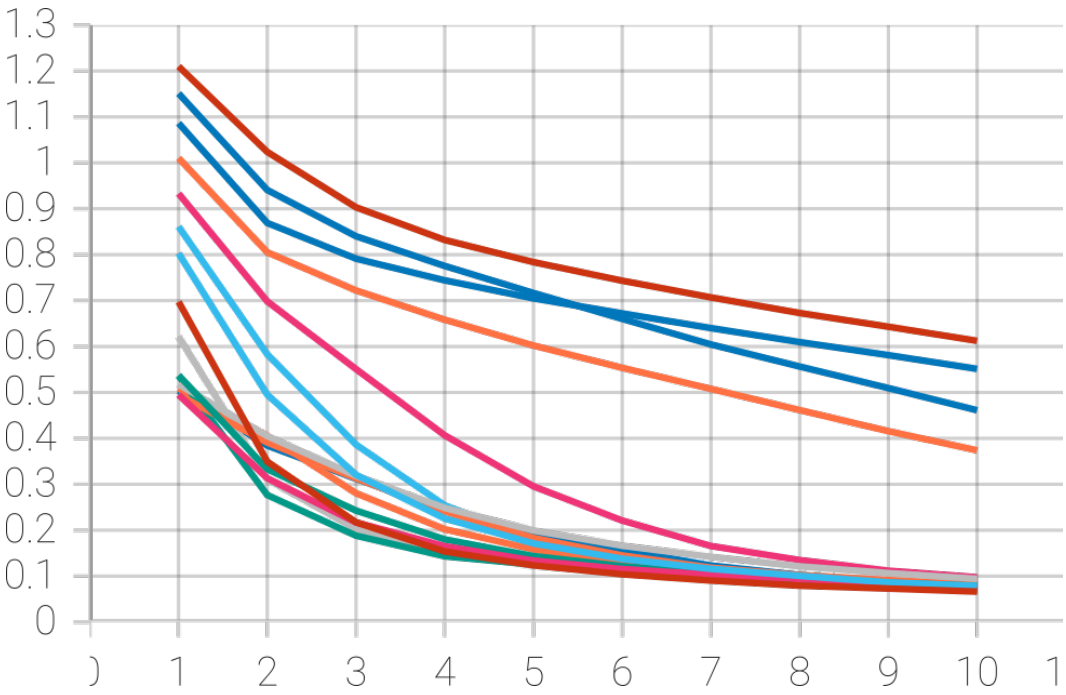

Figure 22: Train loss during HPO of UNet model

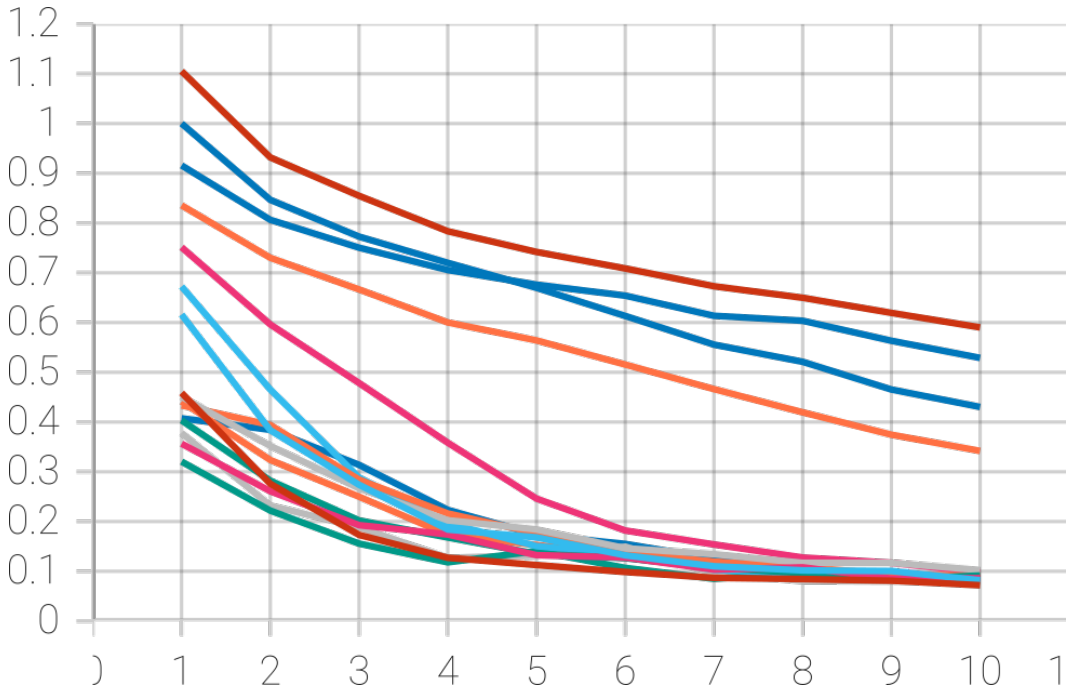

Figure 23: Validation loss during HPO of UNet model

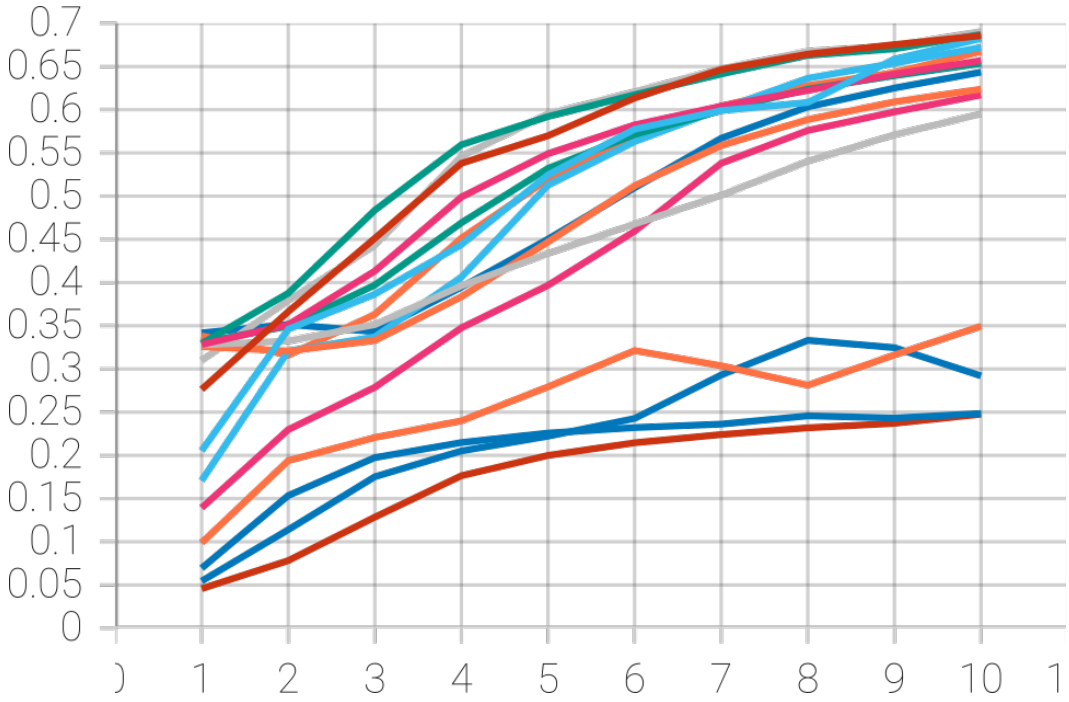

Figure 24: Train score during HPO of UNet model

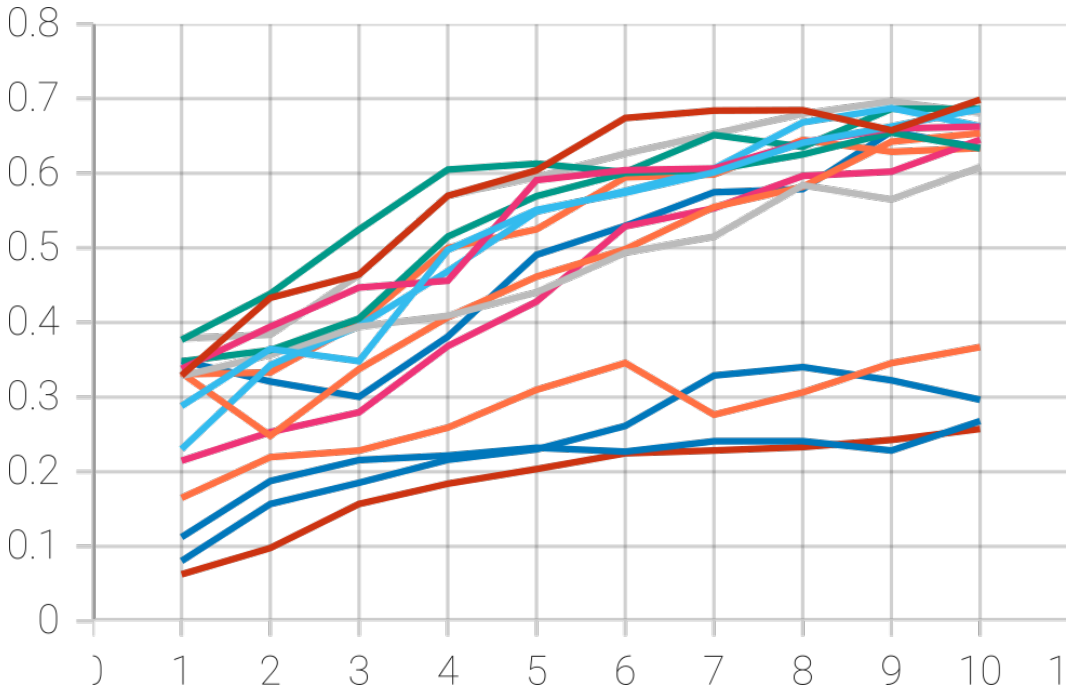

Figure 25: Validation score during HPO of UNet model