HAVC User Guide

Ver 1.5.6 – February 2025

(based on HAVC 5.0.4 and Hybrid 2025.02.12.1)

Table of Contents

1.0 Introduction

This guide has been written to describe the Vapoursynth filter [Hybrid Automatic Video Colorizer] available on GitHub under MIT License.

The filter (*HAVC* in short)[1] was developed to provide a simple way to coloring black and white movies. Due to the technical limitations lots of videos filmed in the last century are in black and white, making them less visually appealing, but most of these videos have historical values and colorizing them could help to restore their appeal especially to younger audiences. In order to add coloring capability to Vapoursynth, the filter is able to combine the results provided by *DeOldify* and *DDColor* , and in alternative to DDcolor can use the models provided in the project Colorization[2]. These models are some of the best models available for coloring pictures, and by combining them, the filter HAVC is able to obtain final colorized images, that often are better than the images obtained from the individual models.

Unfortunately, directly applying existing image colorization methods does not generate satisfactory colorized videos, as minor perturbations in consecutive input video frames, may lead to substantial differences in colorized video results. To overcome this problem, additional specialized filters have been developed for HAVC, that help improve the final quality of the videos.

In addition, to further improve the temporal stability of the colors, has been added the ability to provide the frames colored with HAVC directly as reference images to ColorMNet , Deep Exemplar based Video Colorization model (*DeepEx* in short) and DeepRemaster. *DeepEx*, *DeepRemaster* and *ColorMNet* are exemplar-based video colorization models, which allow to colorize a movie starting from an external-colored reference image. They allow to colorize a Video in sequence based on the colorization history, enforcing its coherency by using a temporal consistency loss. ColorMNet is more recent and advanced respect to DeepEx[3] and it is suggested to use it as default exemplar-based model. DeepRemaster has the interesting feature to be able store the reference images, so that is able to manages situations where the reference images not are synchronized with the movie to colorize. Conversely ColorMNet is not storing the full reference frame image (like DeepRemaster) but it stores only the key points (e.g., representative pixels in each frame). This imply that the colored frames could have some colors that are very different from the reference image. DeepRemaster has not this problem since it stores the full reference image. Unfortunately, the number of reference images that DeepRemaster is able to use depends on GPU memory and power, because the time required for inference increase with the number of reference images provided. Instead ColorMNet has some interpolation capability while DeepRemaster is very basic and is unable to properly colorize a frame if is missing a reference image very similar and it need a lot of reference images to be able to properly colorize a movie (the time resolution of DeepRemaster is 15 frames). So, the choice of which exemplar-based video colorization model to use depends on the source to colorize and the number of reference image available.

This guide contains some useful tips, but as everyone knows, the best way to learn is to personally experiment with the functions and parameters present in HAVC[4]. Unfortunately, was not possible to provide a *one size fits-all solution* for coloring the movies. The HAVC parameters suggested in this guide were defined to address the most common situation, but for obtaining the best results, the filter parameters need to be adjusted depending on the specific type of video to be colored.

---

[1] The first coloring filters added in Hybrid were *DDColor* and *DeOldify*. Subsequently, the DeOldify filter was extended adding the possibility to use *DDColor* to improve the color quality and the filter was renamed **DDeoldify**, then the filter was extended to use more coloring methods, including ColorMNet and the Deep Exemplar based Video Colorization, hence the name of the filter was changed in **HAVC**, because is using a mixture of models (currently are implemented 4 picture-base and 3 exemplar-based models).
[2] The project *Colorization* includes 2 models: *Real-Time User-Guided Image Colorization with Learned Deep Priors* (Zhang, 2017) and *Colorful Image Colorization* (Zhang, 2016). The Zhang (Siggraph, 2017) model is a fairly famous model also adopted by some commercial software. These 2 models have been added as alternative models to *DDcolor* (named: **siggraph17, eccv16**) in short Zhang's models. These models have the same problem of temporal stability of colors observed in DDcolor and thus share the same tweaks developed for DDcolor and in some cases could provide a valid alternative.
[3] In the chapter on HAVC speed-up it will be shown that the model Deep-Exemplar can be used to HAVC coloring speed.
[4] There is a thread on Selur forum that can be used to post questions on HAVC filter: DeOldify Vapoursynth filter

The guide is organized as follows: the first chapter is dedicated to the description of Hybrid installation (the installation of Hybrid is a mandatory requirement for this guide), then there is a chapter describing the HAVC filter and the main parameters of the filter and their use (the sub-chapters 3.1, 3.2, 3.3, 3.4 are quite technical and could be skipped on first reading). Next there is the, much more interesting, chapter describing how to use Hybrid to colorize movies (the sub-chapter 4.1 is quite technical and could be skipped on first reading). Then there is a chapter on using external filters to improve final HAVC quality. Unfortunately, as often happens, these filters can significantly improve some frames and worsen the color of others, so they should be used with caution. Finally, there is a chapter with the reference to the HAVC internal functions. This chapter is useful for the most advanced users, who want to better understand the scripts generated by Hybrid or want to modify them.

2.0 Installation

This filter is distributed with the torch package provided with the **Hybrid Windows Addons**. To use it on Desktop (Windows) it is necessary install Hybrid and the related Addons. **Hybrid** is a Qt-based frontend for other tools (including this filter) which can convert most input formats to common audio & video formats and containers. It represents the easiest way to colorize images with the HAVC filter using VapourSynth and for this reason in this guide will be provided detailed information on how to install and use this filter using Hybrid[5].

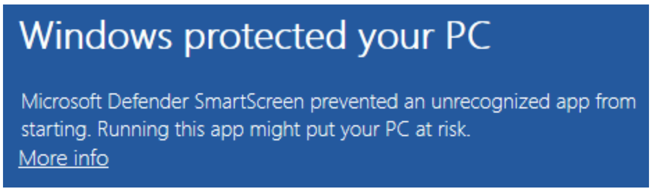
The main advantages of using Hybrid are:

- The availability of a complete working torch package with all the necessary decencies already installed (something that for some users could be a nightmare to complete successfully);
- Hybrid is able to automatically generate all the Python/Vapoursynth code to allow to all filters available in Hybrid to work properly (is not necessary a knowledge of Python/Vapoursynth to be able to use Hybrid);
- Easy access to all the HAVC functions with all parameters properly filled (HAVC as almost 50 parameters, and it can be very difficult to use it without a good knowledge of the filter or without Hybrid).

To install Hybrid is necessary to download it from https://www.selur.de/downloads, opening the link will be displayed the following page:

It is necessary to download and install the installer (see point 1).

If is displayed the blue window of Microsoft Defender SmartScreen it is possible to install anyway by following the instructions provided in the previous link or by clicking on More Info and clicking on **Run anyway**.



It is suggested to install Hybrid in a writable path, like “C:\Hybrid” or “D:\Programs\Hybrid”. Once installed, in the installation folder create a new subfolder called `Settings`, then create the file `misc.ini` with the following lines:

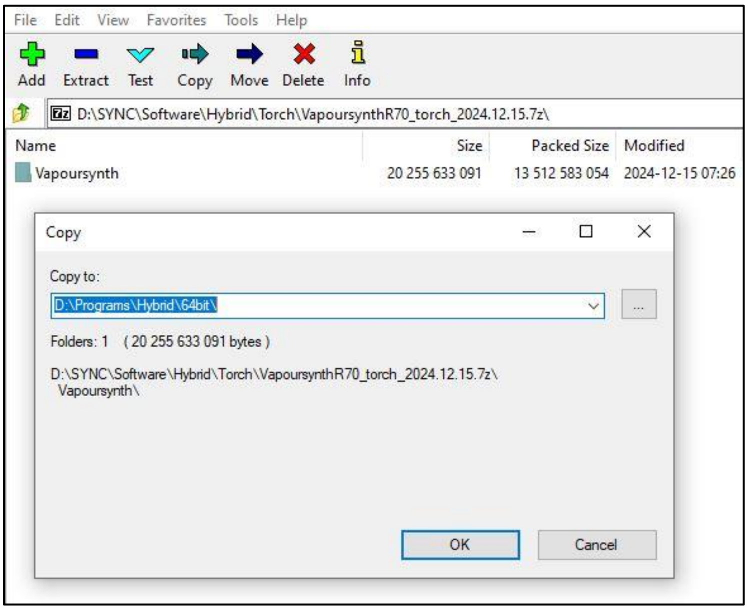
```
[General]

settingPath=.\\Settings

niceness=0
```

In this way Hybrid will run in portable mode[6].

[5] For manual installation see the GitHub page: https://github.com/dan64/vs-deoldify
[6] For more useful settings see the page: https://forum.selur.net/thread-10.html

After having installed Hybrid, it is necessary to click on the link GoogleDrive (see point 2), it will be displayed the following page:

The most important file to download is the archive containing the torch packages which are necessary to use HAVC. In this case the file is named: `VapoursynthR70_torch_2024.12.27.7z`.



By opening it with 7-zip will be displayed the following window.

It is necessary to extract the folder `Vapoursynth` on the related location in the installation folder. In this case it is assumed the Hybrid has been installed in "D:\Programs\Hybrid", in the case Hybrid was installed in a different folder it is necessary to change the destination path (highlighted in blue in the picture on the left) accordingly.

2.1 Installation of Development Version

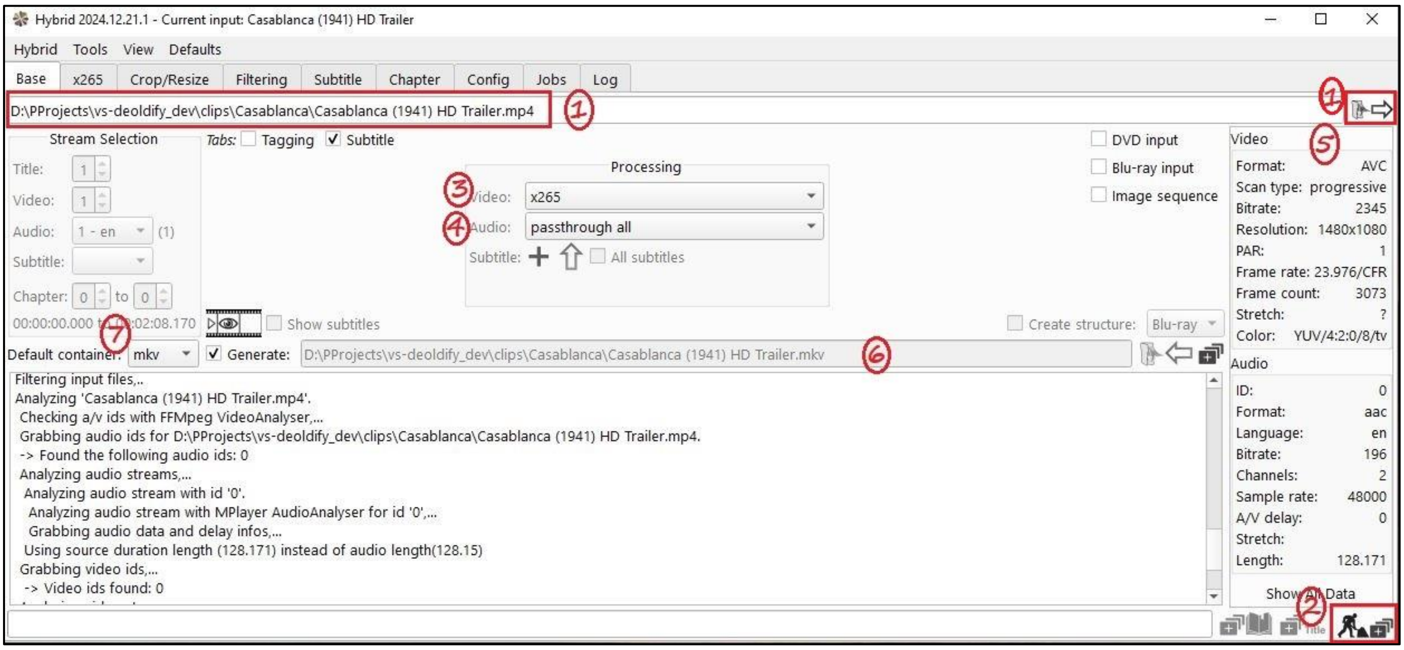
Sometime to get the most updated version of HAVC filter is necessary to install the Development version of Hybrid.

In this case all the files to be downloaded are available in the folder **experimental** on GoogleDrive, as shown in the following picture:

It is necessary first to download and run the installer (see point 1) and then to download and extract the torch addon archive (see point 2) as described previously.

3.0 Using the Filter

Once Hybrid is installed it is possible to use it to coloring B&W movies. The clip to be colored can be added in input to Hybrid by using drag-and-drop. In the following picture is displayed the Hybrid main GUI window.

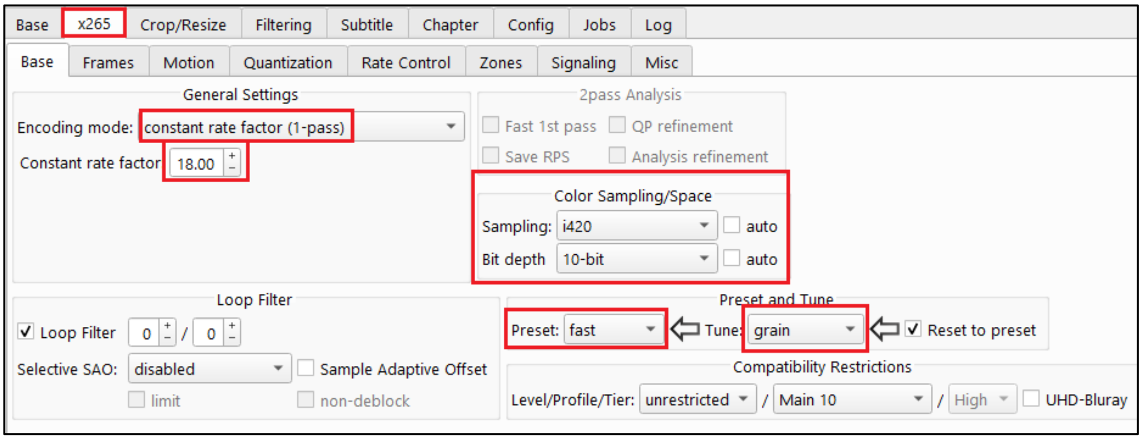


GUI Explanation[7]:

1) Input field, the clip can be inserted with drag-and-drop or by selecting the big arrow on the right of the text box
2) Encoding button, by pressing it Hybrid will start to encode the clip
3) Video encoder, in this case has been selected **x265** (the encoder options are available in the tab "x265")
4) Audio encoder, in this case has been selected "passthrough all", all the audio tracks will be included in the container untouched.
5) Media information page
6) Name to be used for the new encoded clip (in this case is auto generated).
7) The container used to store the encoded clip, in this case "mkv".

3.0.1 Suggested settings for x265 encoding

In the following picture are shown the suggested settings for encoding with h265 at 10-bits (tab "x265")



[7] In the following post (a little outdated) is available a small guide to Hybrid: https://forum.selur. net/thread-282.html

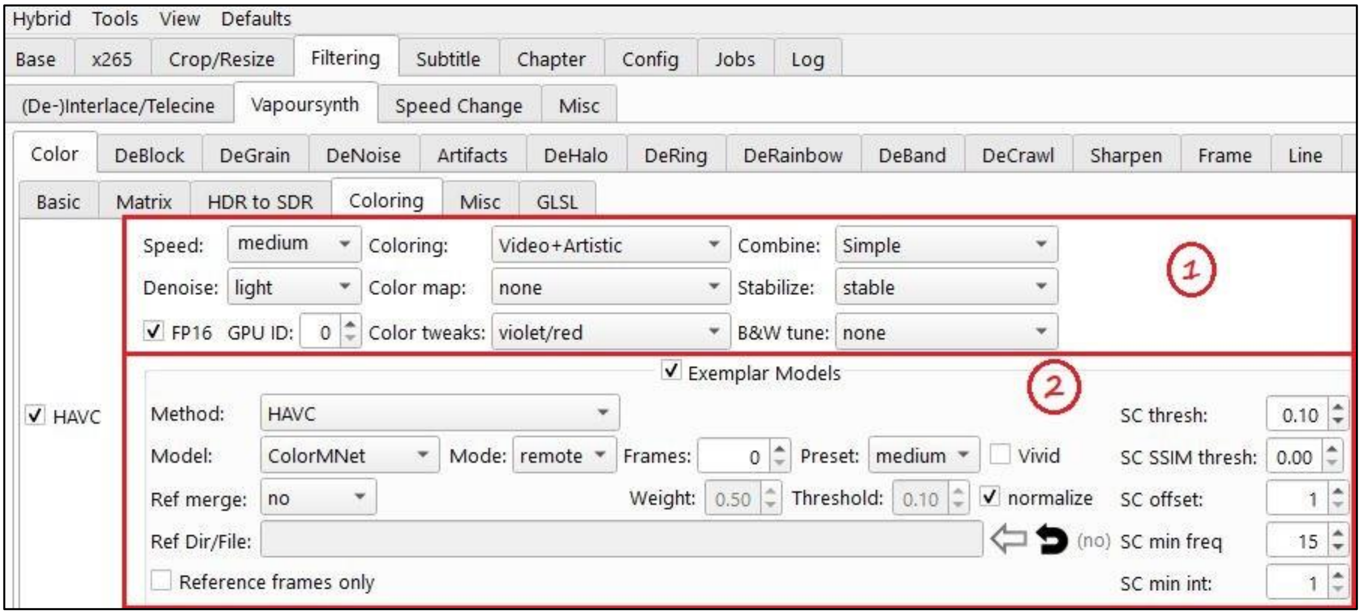
With these settings the movie will be encoded at 10 bits, this will increase the color accuracy. When is selected **fast** and **grain** (in this order) it is necessary to click on the big arrows on the right to apply them (in the same order). The constant factor (CF) of 18.00 should be good enough. To get more quality the CF can be decreased till 15.00 (lower values will increase only the size with very little quality improvement).

3.0.2 Coloring filters configuration page

In Hybrid there are a lot of filters, the coloring filters are available at: **Filtering->Vapoursynth->Color->Coloring**.

The HAVC filter is available under the checkbox **HAVC**. The filter was developed having in mind to use it mainly to colorize movies. Both DeOldify and DDcolor are good models for coloring pictures, but when are used for coloring movies they are introducing artifacts that usually are not noticeable in the images but are well observable in the colored movie. Especially in dark scenes both DeOldify and DDcolor are not able to understand what it is the dark area and what color to give it, they often decide to color these dark areas with blue, then in the next frame this area could become red and then in the next frame return to blue, introducing a *flashing psychedelic effect* when all the frames are put in a movie. To try to solve this problem has been developed *pre-* and *post-* process filters.

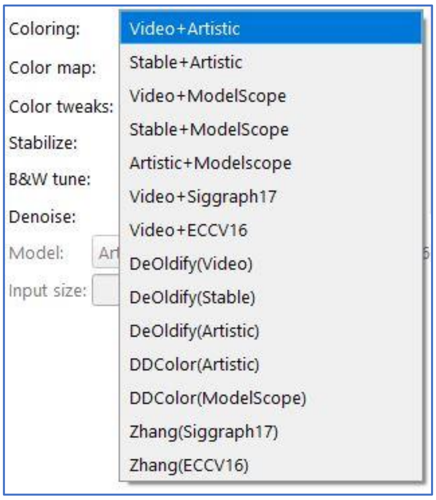
The HAVC filter panel can be divided in 2 group-box (see red rectangles on the picture below):



1) The **Presets group-box**, that allows to set all the filter parameters of the custom section (which are more than 35). It represents, the easiest way to use the filter to get good results. The parameters in this group box are used as input for the internal filter function HAVC_main.

2) The **Exemplar-based Models** group-box. Using this section is possible to change all the filter parameters related to the exemplar-base color models: ColorMNet, DeepRemaster and DeepEx. These parameters are used as input to the internal filter function HAVC_deepex.

The **Coloring** field in the group-box 1 (see picture on the right), allows to select the pictures-based color models: DeOldify, DDColor and Zhang (siggraph17, eccv16) and their combination. The DeOldify has 3 networks where the Video network is the most stable. DeOldify is the only model with a network trained for coloring videos and using it with the other color models will improve the color stability and quality as shown in the section [Comparison of Models]() on GitHub.



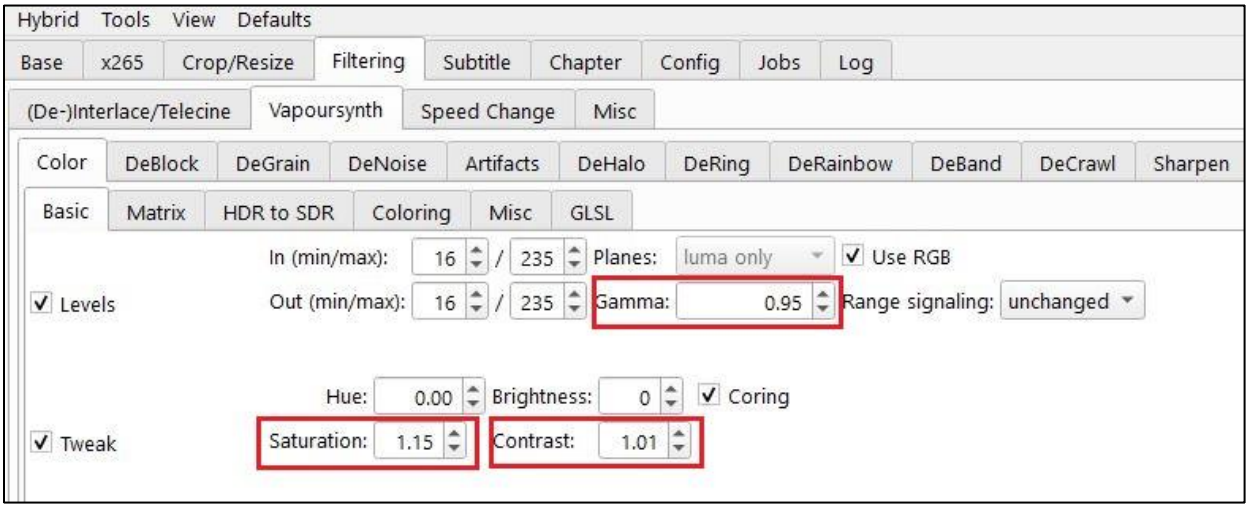
The meaning of remaining group-box filters is the following:

- **Color map**: this field allows to select the [Color Mapping]() presets;
- **Color tweaks**: the fields *ColorFix* and *Denoise* (ColorTune) are the [Tweaks]() parameters developed for the pictures-based models. It is possible to disable the tweaks by setting this field to **none.** The parameter *Denoise* has effect also on the strength of **Color map**;
- **Stabilize**: it allows to define the weight to assign to the DeOldify Video stabilization network when used in combination with DDColor and the Alternative color models (suggested presets are: Stable and MoreStable).
- **Combine**: it allows to define the method adopted by HAVC to [merge]() the frames colored by 2 models.
- **Speed**: it allows to select the most appropriate parameters to get the best balance between quality and speed (suggested presets is: medium). This table display the full mapping of Speed to RenderFactor:

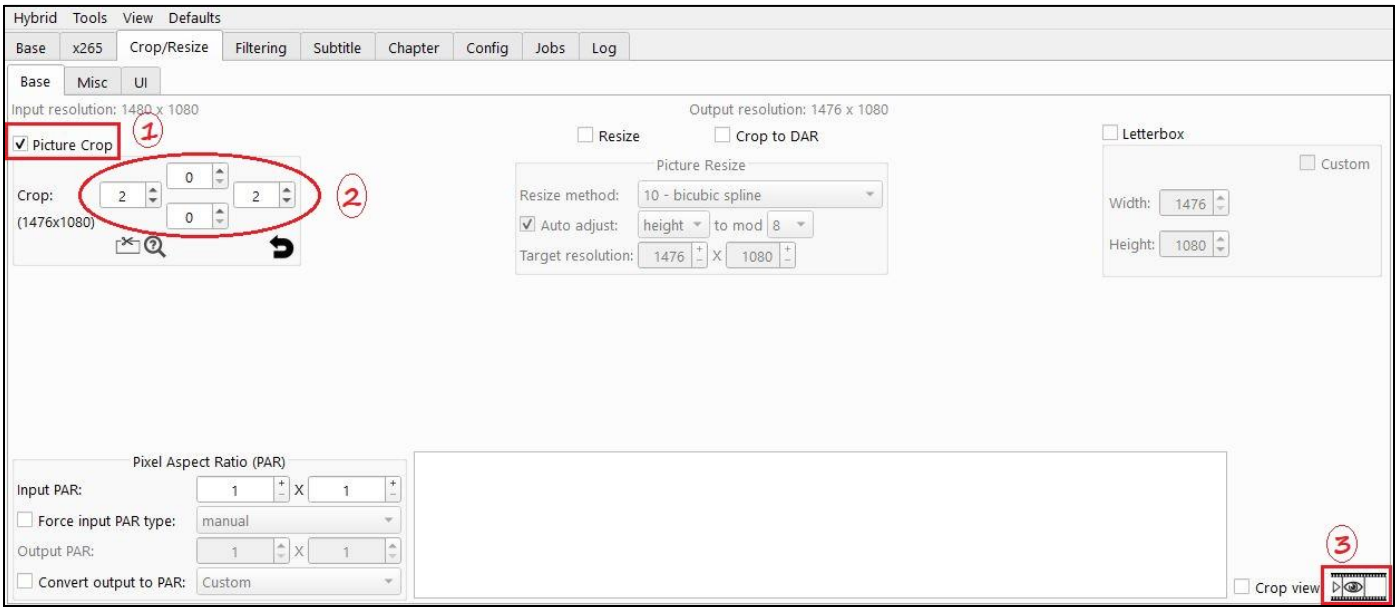
Model	placebo	veryslow	slower	slow	medium	fast	faster	veryfast
DeOldify (RF)	32	30	28	26	24	22	20	16
DDColor (RF)	44	36	32	28	24	22	20	16

The most colorful movies can be obtained using the single pictures-based color models. But the resulting video will be almost unwatchable and this is the main reason behind the development of HAVC. It is possible to improve the color stability using the combined color models proposed in the field **Speed** above and setting the **Color tweaks** to medium.

In the chapter [Coloring using Hybrid]() will be suggested some settings that will allow to improve further the color temporal stability and, in the chapter, [Best settings for color temporal stability]() will be provided what could be the optimal settings to get movies with colors very stable. Unfortunately, the color stability has a cost, not only in term of computational speed but also and above all, in terms of variety of colors. With the increase of color stability will decrease the variety and saturation of the colors. Some useful filters, which will be possible to add as post-process to improve the saturation of the colors, can be found in the panel: **Filtering->Vapoursynth->Color->Basic** as shown in the picture below (with some suggested settings):



It is recommended to always remove all the black bars (if any) before applying the coloring filters. It is possible to remove the black bar in Hybrid, using the dedicated page **Crop/Resize** (see picture below). It is necessary to enable the **Picture Crop (1)** and then insert the appropriate number of pixels in the **Crop box (2)**. It is possible to preview the crop by clicking on the **Preview Crop button (3)**.



3.1 HAVC *pre*- and *post*- process filters

The main filters introduced are[8]:

3.1.1 Post-process filters

Chroma Smoothing: This filter allows to reduce the *vibrancy* of colors assigned by DeOldify/DDcolor by using the parameters *de-saturation* and *de-vibrancy*, the effect on *vibrancy* will be visible only if the option **chroma resize** is enabled (default), otherwise this parameter has effect on the *luminosity*. The area impacted by the filter is defined by the thresholds dark/white. All the pixels with luma below the dark threshold will be impacted by the filter, while the pixels above the white threshold will be left untouched. All the pixels in the middle will be gradually impacted depending on the luma value (see related parameters).

Chroma Stabilization: This filter will try to stabilize the frames' colors. As explained previously since the frames are colored individually, the colors can change significantly from one frame to the next, introducing a disturbing psychedelic flashing effect. This filter tries to reduce this by averaging the chroma component of the frames. The average is performed using a number of frames specified in the *Frames* parameter. Are implemented 2 averaging methods:

1. *Arithmetic average*: the current frame is averaged using equal weights on the past and future frames
2. *Weighted average*: the current frame is averaged using a weighed mean of the past and future frames, where the weight decreases with the time (far frames have lower weight respect to the nearest frames).

As explained previously the stabilization is performed by averaging the past/future frames. Since the non-matched areas of past/future frames are *gray* because is missing in the past/future the *color information*, the filter will apply a *color restore* procedure that fills the gray areas with the pixels of current frames (eventually de-saturated with the parameter "sat"). The image restored in this way is blended with the non-restored image using the parameter "weight". The gray areas are selected by the threshold parameter "tht". All the pixels in the HSV color space with "S" < "tht" will be considered gray. If is detected a scene change (controlled by the parameter "tht_scen"), the *color restore* is not applied (see related parameters).

Darkeness: this post process filter will force the dark areas of a frame, identified by the region of pixels having a luma below the *dark_threshold*, to have a dark color, the dark color is obtained by de-saturating the pixels by an amount specified by the parameter *dark_amount* (see related parameters).

3.1.2 Pre-process Filters

DDColor Tweaks: This filter is available only for DDColor and has been added because has been observed that the DDcolor's *inference* is quite poor on dark/bright scenes depending on the luma value. This filter will force the luma of input image to don't be below the threshold defined by the parameter *luma_min*. Moreover, this filter allows to apply a **dynamic gamma correction**. The gamma adjustment will be applied when the average luma is below the parameter *gamma_luma_min*. The adjustment applied to gamma is defined by the following expression:

gamma_new = MAX[gamma*(luma/*gamma_luma_min*)^*gamma_alpha*, *gamma_min*]

A *gamma* value > 2.0 improves the DDColor stability on bright scenes, while a *gamma* < 1 improves the DDColor stability on dark scenes (see related parameters). Using the dynamic gamma correction is possible to apply a high tweak gamma (parameter [2] in the tweak parameter list) and then thanks to the dynamic gamma correction decreasing it with the luma, so that on dark scenes the gamma will < 1. At the following link there is a comparison between using a gamma = 1 and gamma = 2: https://imgsli.com/MjUyNjY0. For this sample a DDcolor Tweak like this: ddtweak_p=[0.0, 1.0, 2.8, True, 0.3, 0.6, 0.7, 0.5] is appropriate. It is also possible to specify the Chroma Adjustment to be applied to the frames colored with DDcolor (and Alternative models) by adding a string parameter. An example of full tweak is the following:

[8] These filters are automatically applied by Hybrid if is selected a rendering speed above *faster*.

ddtweak_p=[0.0, 1.0, 2.5, True, 0.3, 0.6, 0.7, 0.5, "300:360|0.5,0.1"]

In this example the last string parameter represent the Chroma Adjustment that will be explained in the next chapter.

3.2 Chroma Adjustment

Unfortunately, when are applied to movies the color models are subject to assign unstable colors to the frames especially on the red/violet chroma range. This problem is more visible on DDColor than on DeOldify. To mitigate this issue was necessary to implement some kind of chroma adjustment. This adjustment allows to de-saturate all the colors included in a given color range. The color range must be specified in the HSV color space. This color space is useful because all the chroma is represented by only the parameter "Hue". In this color space the colors are specified in degree (from 0 to 360), as shown in the HAVC Hue Wheel. It is possible to apply this adjustment on all filters described previously. Depending on the filter the adjustment can be enabled using the following syntax:

Croma Range

chroma_range = "hue_start:hue_end" or "hue_wheel_name"

for example, this assignment:

chroma_range = "290:330,rose"

specify the range of hue colors: 290-360, because "rose" is hue wheel name that correspond to the range:330-360.

It is possible to specify more ranges by using the comma "," separator.

In HAVC are defined the following hue wheel names:

Name	Chroma Range
red	“0:30”
orange	“30:60”
yellow	“60:90”
yellow-green	“90:120”
green	“120:150”
blue-green	“150:180”
cyan	“180:210”
blue	“210:240”
blue-violet	“240:270”
violet	“270:300”
red-violet	“300:330”
rose	“330:360”

Chroma Adjustment

When the de-saturation information is not already available in the filter's parameters, it necessary to use the following syntax:

chroma_adjustment = "chroma_range|sat,weight"

in this case it is necessary to specify also the de-saturation parameter "sat" and the blending parameter "weight".

for example, with this assignment:

chroma_adjustment = "300:340|0.4,0.2"

the hue colors in the range 300-340 will be de-saturated by the amount 0.4 and the final frame will be blended (with weight 0.8=1-0.2) with the frame obtained by applying a de-saturation of 0.4 an all the pixels. (if weight=0, no blending is applied). The weight can also be negative, as shown in the example below:

chroma_adjustment = "300:340|0.4,-0.2"

In this case the hue colors in the range 300-340 will be de-saturated by the amount 0.4 as in the previous example, but the final frame will be blended (with weight 0.8) with the original (non de-saturated) frame (i.e. will be avoid the merge with the de-saturated frame in all pixels).

To simplify the usage of this filter has been added the Preset *ColorFix* which allows to fix a given range of chroma combination. The strength of the filter (i.e. de-saturation) is controlled by the Preset *ColorTune*.

3.3 Color Mapping

Using an approach similar to *Chroma Adjustment* has been introduced the possibility to remap a given range of colors in another chroma range. This remapping is controlled by the Preset *ColorMap*. For example, the preset "blue->brown" allows to remap all the chroma combinations of *blue* in the color *brown*. It is not expected that this filter can be applied on a full movie, but it could be useful to remap the color on some portion of a movie.

To use the color mapping feature is necessary to use the following syntax:

colormap = "chroma_range|hue_shifit,weight"

The color mapping is similar to the chroma adjustment, the difference instead to apply a desaturation to the given color range is applied a chroma hue shift.

For example, with this setting:

colormap ="30:90|+250,0.8"

the color range "30:90" (corresponding to yellow) will be shifted by +250 degrees, the original will be retained at 80%, because has been specified the weight=0.8 (the weight given to the adjusted frame is 0.2=1-0.8).

In the chapter HAVC Color Mapping/Chroma Adjustment are provided useful tips on how to use both the *Chroma Adjustment* and *Color Mapping* features provided by this filter.

In HAVC are defined the following color mapping names:

Name	**Color Mapping**
blue->brown	180:280|+140,0.8
blue->red	180:280|+100,0.8
blue->green	180:280|+220,0.8
green->brown	80:180|+260,0.8
green->red	80:180|+220,0.8
green->blue	80:180|+140,0.8
redrose->brown	300:360,0:20|+40,0.8
redrose->blue	300:360,0:20|+260,0.8
red->brown	320:360,0:15|+50,0.8
yellow->rose	30:90|+300,0.8

3.4 Merging the models

As explained previously, this filter is able to combine the results provided by DeOldify and DDColor, to perform this combination has been implemented 6 methods:

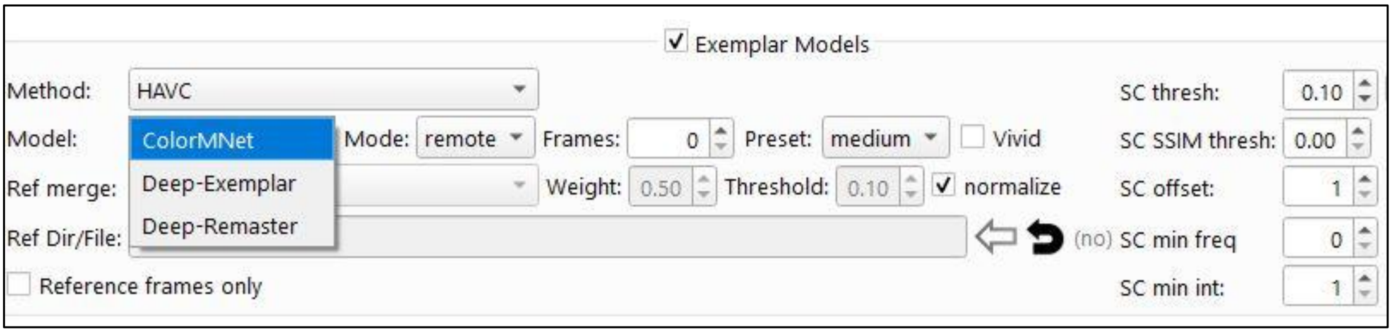
0. *DeOldify* only coloring model (no merge).
1. *DDColor* only color model (no merge).
2. *Simple Merge*: the frames are combined using a *weighted merge*, where the parameter *merge_weight* represent the weight assigned to the frames provided by the DDcolor model, using the following weighted sum: f_out = f_deoldify*(1- *merge_weight*) + *merge_weight* *f_ddcolor (see related parameter).
3. *Constrained Chroma Merge*: given that the colors provided by DeOldify *Video* model are more conservative and stable than the colors obtained with DDcolor. The frames are combined by assigning a limit to the amount of difference in chroma values between DeOldify and DDcolor. This limit is defined by the parameter *threshold*. The limit is applied to the frame converted to "YUV". For example, when threshold=0.1, the chroma values "U","V" of DDcolor frame will be constrained to have an absolute percentage difference respect to "U","V" provided by DeOldify not higher than 10%. If *merge_weight* is < 1.0, the chroma limited DDColor frames will be will be merged again with the frames of DeOldify using the *Simple Merge* (see related parameter).
4. *Luma Masked Merge*: the behavior is similar to the method *Adaptive Luma Merge*. With this method the frames are combined using a *masked merge*. The pixels of DDcolor's frame with luma < *luma_limit* will be filled with the (de-saturated) pixels of DeOldify, while the pixels above the *white_limit* threshold will be left untouched. All the pixels in the middle will be gradually replaced depending on the luma value. If the parameter *merge_weight* is < 1.0, the resulting masked frames will be merged again with the non-de-saturated frames of DeOldify using the *Simple Merge* (see related parameter).
5. *Adaptive Luma Merge*: given that the DDcolor performance is quite bad on dark scenes, with this method the images are combined by decreasing the weight assigned to DDcolor frames when the luma is below the *luma_threshold*. For example, with: *luma_threshold* = 0.6 and *alpha* = 1, the weight assigned to DDcolor frames will start to decrease linearly when the luma < 60% till *min_weight*. For *alpha*=2, the weight begins to decrease quadratically, because the formula applied is: ddcolor_weight = MAX[weight * (luma/*luma_threshold*)^*alpha*, *min_weight*] (see related parameter).

The merging methods 2-5 are leveraging on the fact that usually the DeOldify *Video* model provides frames which are more stable, this feature is exploited to stabilize also DDColor. The methods 3 and 4 are similar to *Simple Merge*, but before the merge with *DeOldify* the *DDColor* frame is limited in the chroma changes (method 3) or limited based on the luma (method 4). The method 5 is a *Simple Merge* where the weight decreases with luma.

3.5 Exemplar-based Models

As stated previously to stabilize further the colorized videos it is possible to use the frames colored by HAVC as reference frames (exemplar) as input to the supported exemplar-based models: ColorMNet, Deep Exemplar based Video Colorization and DeepRemaster.

In Hybrid the *Exemplar Models* have their own panel, as shown in the following picture:



For the ColorMNet models there are 2 implementations defined, by the field **Mode**:

- 'remote' (has not memory frames limitation but it uses a remote process for the inference)
- 'local' (the inference is performed inside the VapourSynth local thread but has memory limitation)

The field **Preset** control the render method and speed, allowed values are:

- 'Fast' (faster but colors are more washed out)
- 'Medium' (colors are a little washed out)
- 'Slow' (slower but colors are a little more vivid)

The field **SC thresh** define the sensitivity for the scene detection (suggested value **0.10**), while the field **SC min freq** allows to specify the minimum number of reference frames that have to be generated.

The flag **Vivid** has 2 different meanings depending on the *Exemplar Model* used:

- **ColorMNet** (the frames memory is reset at every reference frame update)
- **DeepEx** (given that the colors generated by the inference are a little washed-out, the saturation of colored frames will be increased by about 25%).
- **DeepRemaster** (given that the colors generated by the inference are a little washed-out, the saturation of colored frames will be increased by about 15% and the Hue by 8).

The field **Method** allows to specify the type of reference frames (RF) provided in input to the *Exemplar-based Models*, allowed values are:

- 0 = HAVC same as video (default)
- 1 = HAVC + RF same as video
- 2 = HAVC + RF different from video
- 3 = external RF same as video
- 4 = external RF different from video
- 5 = external ClipRef same as video
- 6 = external ClipRef different from video

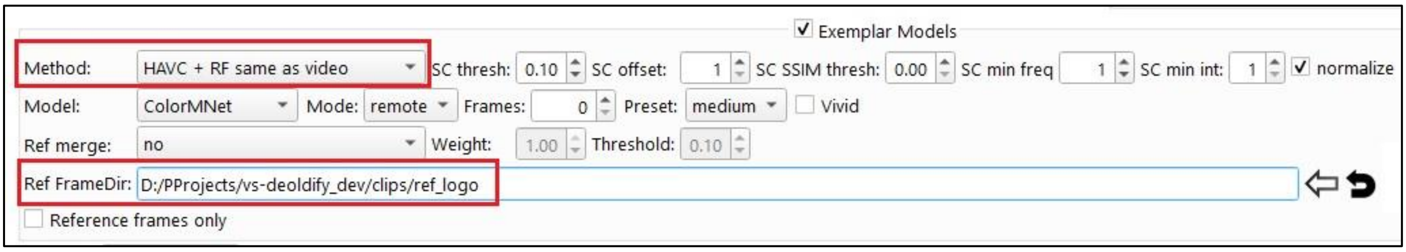
Where "same as video" implies that the reference frame provided (from file or video clip) is the same as the frame to be colored with the only exception of the colors. The model **DeepRemaster**, since it needs to access to the future frames, can be used only with methods: 3, 4, 5, 6.

It is possible to specify the directory containing the external reference frames by using the field **Ref FrameDir**. The frames must be named using the following format: *ref_nnnnnn.[png | jpg]*. With the methods 5 and 6 it is possible to select as source of reference frames the path to a video clip. This feature is useful in the case a colored movie is already available but is necessary to improve the stabilization of colors using the exemplar-based models.

Finally, the flag **Reference frames only** can be used to export the reference frames generated with the method **HAVC** and defined by the parameters **SC thresh**, **SC min freq** fields. The fields methods: HAVC *+ RF same as video* and *HAVC + RF different from video*, can be used to correct the colors applied by the HAVC filter.

Supposing, for example that the movie to be colored starts with the logo of film producer. In this case it could be possible that the frames colored by the color model will apply different colors to the logo. To correct this problem, it is possible to create a folder that contains the logo reference frames as shown in the picture below:

supposing to have named the folder *ref_logo* it is possible to use it to properly colorize the logo frames using the following settings



In this case has been selected the **Method** *HAVC + RF same as video* because the reference frames were almost equal (with the only exception of the colors) to the B&W frames to colorize. To use this method is necessary to provide in the field **Ref FrameDir** the path of the directory containing the reference frames to be used by the exemplar-based model.

When is selected the Method **HAVC different from video**, it is possible to selected in **Ref FrameDir** a colored video (instead of a directory), that will be used to extract the reference frames to be used for the color inference by ColorMNet[remote all-ref]. This method cannot be used with **Ref merge** because frames different from video cannot be safely merged.

3.5.1 The new features problem

Unfortunately all the Deep-Exemplar methods have the problem that are unable to properly colorize the new "features" (new elements not available in the reference frame) so that often these new elements are colored with implausible colors (see for an example: New "features" are not properly colored)[9]. To try to fix this problem has been introduced the possibility to merge the frames propagated by DeepEx with the frames colored with DDColor and/or DeOldify. The merge is controlled by the field **Ref merge**, allowed values are:

- 0 = no merge
- 1 = reference frames are merged with very-low weight
- 2 = reference frames are merged with low weight
- 3 = reference frames are merged with medium weight
- 4 = reference frames are merged with high weight
- 5 = reference frames are merged with very-high weight

When the field **Ref merge** is set to a value greater than 0, the field **SC min freq** is set =1, to allows the merge for every frame. This parameter has been added to fix the problem reported in the post New "features" are not properly colored. For example, in the picture on the left below there is the frame #20 obtained by merging the propagated frame with the frame colored using DDColor and/or DeOldify. In the middle there is the propagate frame with no merge (the new features added in the frame were the hands). The reference image used for coloring the frame provided in input to the model DeepEx is displayed in the picture on the right:



Using ColorMNet the colored frame (with no merge) is a little better[10] as shown in the following picture:


0020.png


0021.png


0022.png

The code used to generate the merged frame #20 was:

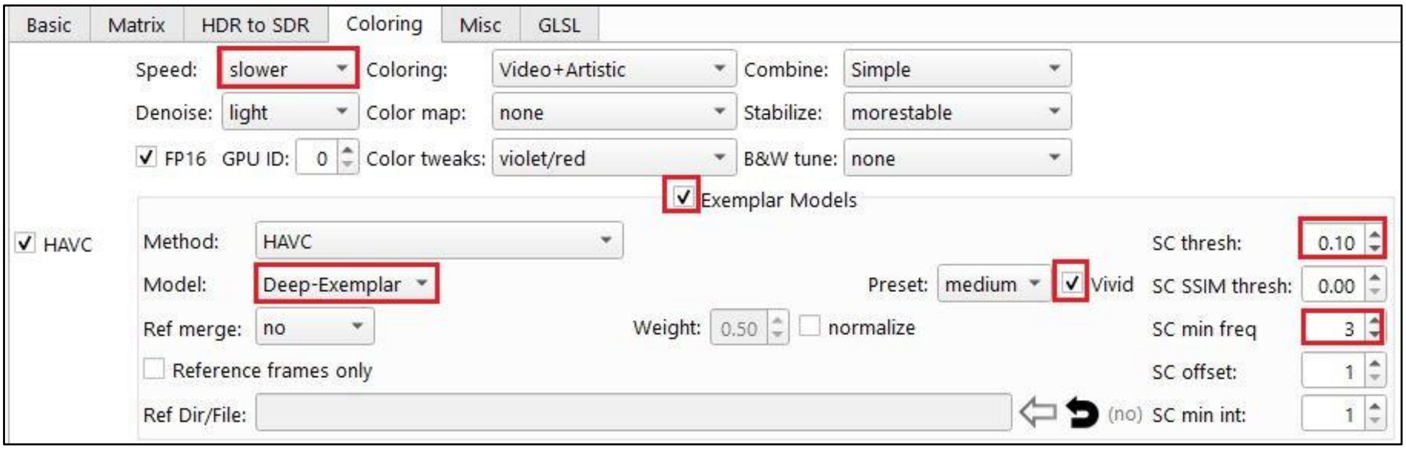
```
clip = HAVC_main(clip,Preset='Medium',ColorFix='Violet/Red',ColorTune='Light',
EnableDeepEx=True, DeepExMethod=1, DeepExPreset='Medium', DeepExRefMerge=2,
DeepExModel=1, ScFrameDir="D:/Tests/Green/ref_color")
```

[9] The problem was mitigated with the release of ColorMNet.

[10] It is interesting to observe that ColorMNet was able to colorize 1 hand because was a little visible on the reference image.

3.5.2 Using Deep-Exemplar to speed-up the encoding

The model Deep-Exemplar (*DeepEx*) has not the memory storage capability of ColorMNet of DeepRemaster, it is just able to propagate the colors of the reference frames received as input. But its simplicity can be used to speed-up the HAVC coloring process. An example of this use is shown in the picture below:



In the example above it was selected the model **Deep-Exemplar**, with the method **HAVC**. In this case the trick to speed-up HAVC is to set **SC min freq** equal to 3. In this way the coloring filter will be called *only* every 3 frames, thus providing a theoretical speed-up of about 3x on the coloring process. The actual speed improvement is much less because it is necessary to introduce the overhead of using *DeepEx* to propagate the colors, so that the total encoding speed will increase only of about 40%. Using a **SC min freq** equal to 4 or 5, will increase further the speed but DeepEx could have some problems in propagating correctly the colors because of new features problem described previously.

This increase in speed can be used, to use the HAVC preset **Speed** *slower* so that the colors generated by HAVC will be more accurate. This preset is about 40% slower than the suggested preset *medium*, but using this trick is possible to obtain an encoding speed similar to the one obtained using the preset **Speed** *medium.*

4.0 Coloring using Hybrid

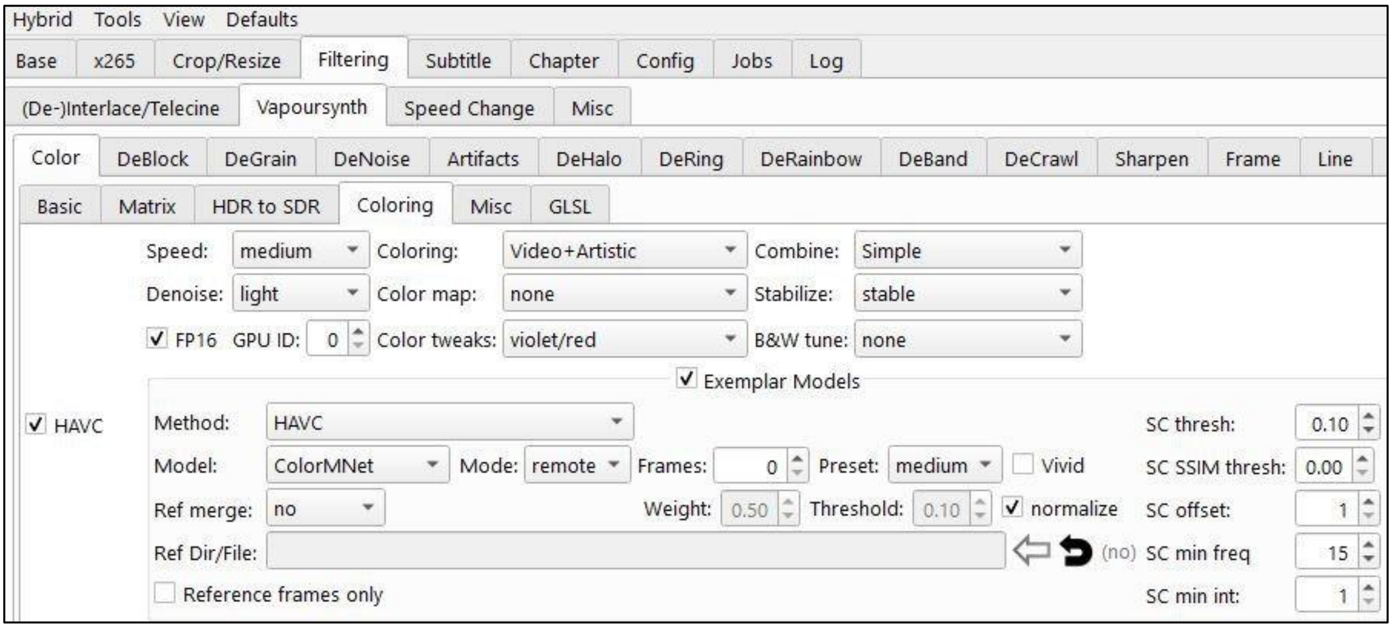
As stated previously the simplest way to colorize images with the HAVC filter it to use Hybrid. To simplify the usage has been introduced standard Presets that automatically apply all the filter's settings. A set of parameters that are able to provide a satisfactory colorization are the following:

- **Speed:** medium (*slower* will increase the color accuracy but with a decrease of about 40% in encoding speed)
- **Color map:** none
- **Color tweaks:** violet/red
- **Denoise:** light
- **Stabilize:** Stable (or MoreStable)

then enable the *Exemplar Models* check box and set

- **Method:** HAVC
- **SC thresh:** 0.10
- **SC SSIM thresh:** 0.0
- **SC min freq:** 15 (5 if is used the *local* mode)
- **normalize:** checked
- **Mode:** remote
- **Frames:** 0
- **Preset:** medium (*slow* will increase the color accuracy but the speed will decrease of 40%)
- **Vivid:** checked

In the following picture are shown the suggested parameters:

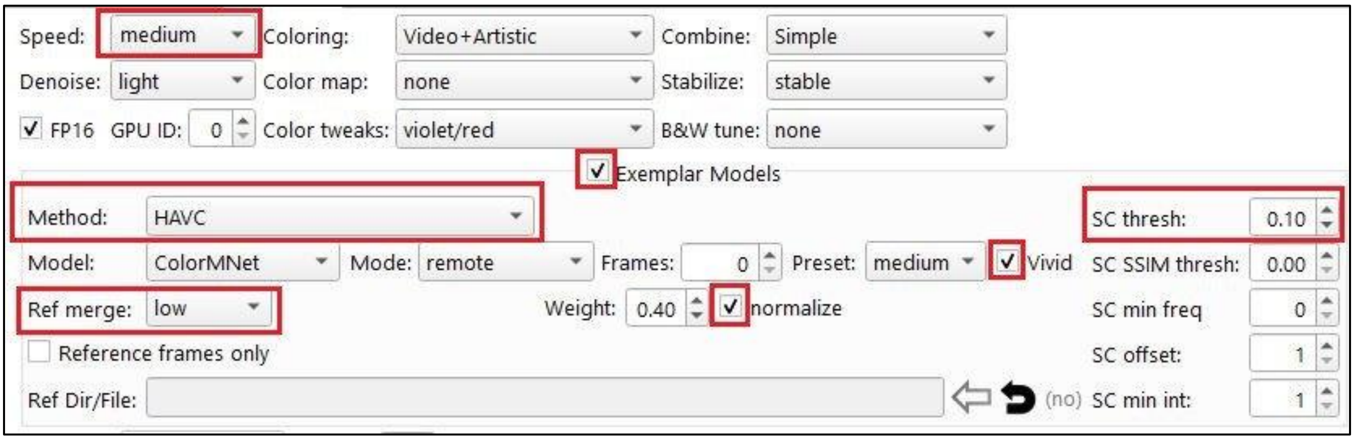


In ColorMNet using the flag **Vivid** the internal memory will be reset at every new reference frame; this will allow to assign the maximum weight to the last reference frame during the inference and the color propagation will depend only from the last reference added. By unchecking **Vivid** the last reference frame will be added to the internal memory and the inference will depend from all the reference frames provided in input, this will produce sometime unwanted blend effect where the propagate colors can be different from the last reference image.

Once all the filter parameters are set it is possible start the encoding process by pressing the Encoding button 2 as described in the section GUI Explanation.

4.0.1 Best settings for colors temporal stability

Willing to obtain the best results, but at cost of speed it is possible to improve further the color stability using the settings shown in the following picture:



Respect to the previous configuration has been changed the following parameters (highlighted in red box):

- **Speed:** medium
- **Ref merge**: low
- **Weight**: 0.40 (automatically filled)
- **Threshold**: 0.10
- **Vivid:** checked
- **Normalize:** checked

As explained in The new features problem in this way the frames propagated by exemplar-based models will be merged with the frames colored with DDColor and/or DeOldify. In practice will be merged 3 frames. This will improve further the temporal color stability while maintaining the color accuracy, and thus avoiding the color degradation (observed in the frames propagated using the exemplar-based models) when new features are introduced. The strategy adopted by HAVC in this case is the following:

1) all the clip frames are colored using DDColor and/or DeOldify (because the parameter SC min freq = 1)
2) using the threshold specified (in this case 0.10) a subset of colored frames is sampled by the scene change detection algorithm and these frames are used as reference images for the selected exemplar-based model (in this case ColorMNet).
3) the frames obtained at step 1 are merged with the weight specified (in this case 0.50) with the frames propagated by the selected exemplar-based model.

This approach will allow to maintain the temporal color consistency provided by the exemplar-based models[11] and at the same time to keep the color quality provided by DDColor and/or DeOldify.

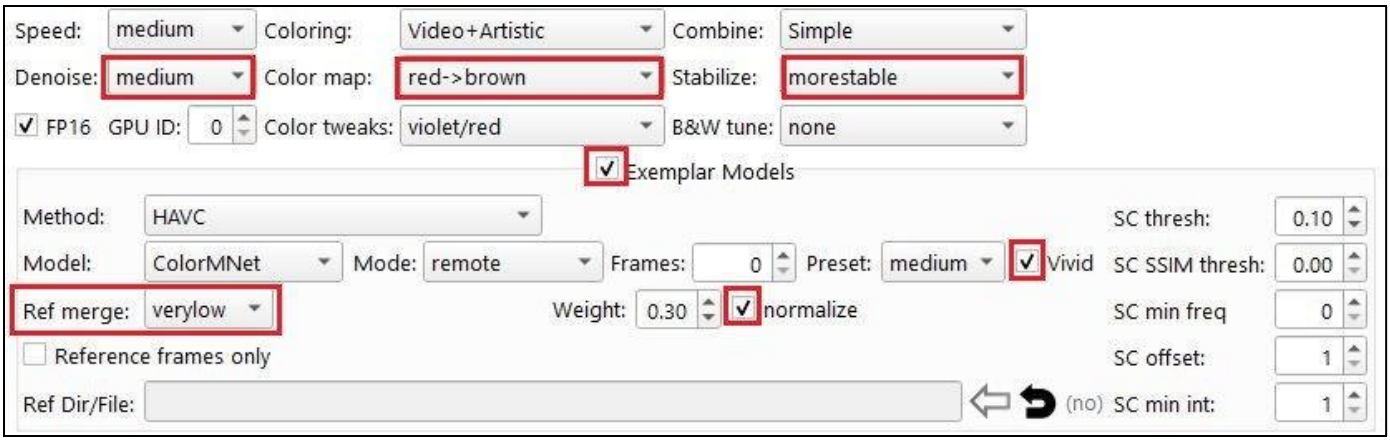
The only methods that can be used with **Ref merge** are **HAVC** and **HAVC + RF same as video**, in this case it is just enough providing a valid path in the field **Ref FrameDir** containing the reference images.

[11] In ColorMNet the effect is reinforced by setting the parameter **Vivid** unchecked.

Unfortunately, this quality improvement has a cost, and using this approach the encoding speed will decrease about of 50% respect to the approach suggested previously. The only way to increase the speed is to upgrade the GPU (Nvidia RTX4070 or above).

4.0.2 Best settings to remove colors shifting towards red

Unfortunately, all the pictures-based color models: DeOldify, DDColor and Zhang (siggraph17, eccv16), suffer from the problem that they tend to shift the colors towards red. By changing some of the settings described in the previous chapter it is possible to mitigate the problem. The suggested settings are shown in the following picture:

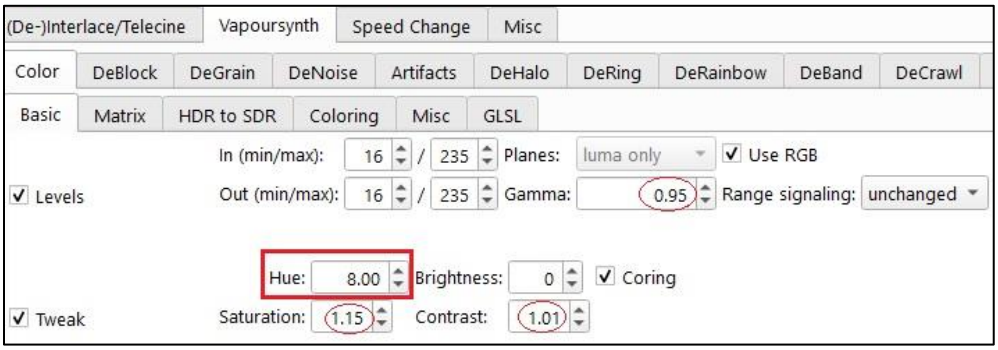


Respect to the previous configuration has been changed the following parameters (highlighted in red box):

- **Color map**: red->brown
- **Denoise**: medium
- **Stabilize**: morestable
- **Ref merge**: verylow
- **Weight**: 0.30 (automatically filled)

As explained previously the Color Mapping allows to change a range of colors in another range, by shifting the Hue color component (in HSV color space). Using the preset *red->brown* the pixels in the frames having the specified range will be shifted toward the brown color. Using the **Denoise** equal to *strong* the *violet/red* components will be strongly desaturated and this will contribute to mitigate the red-shift problem (if the effect is too strong it is possible to use the preset *medium*). To enforce the effect was selected the **Stabilize** preset *morestable*, this will lower the weight of color models DDColor and Zhang (siggraph17, eccv16) which are strongly affected by this problem. Finally, has been selected the preset **Ref merge** equal to *verylow*, so that the frames colorized by ColorMNet will have more weight. The exemplar-based model ColorMNet is almost not affected by the problem of shifting the colors towards red, and so by given to it more weight will contribute to mitigate further the problem.

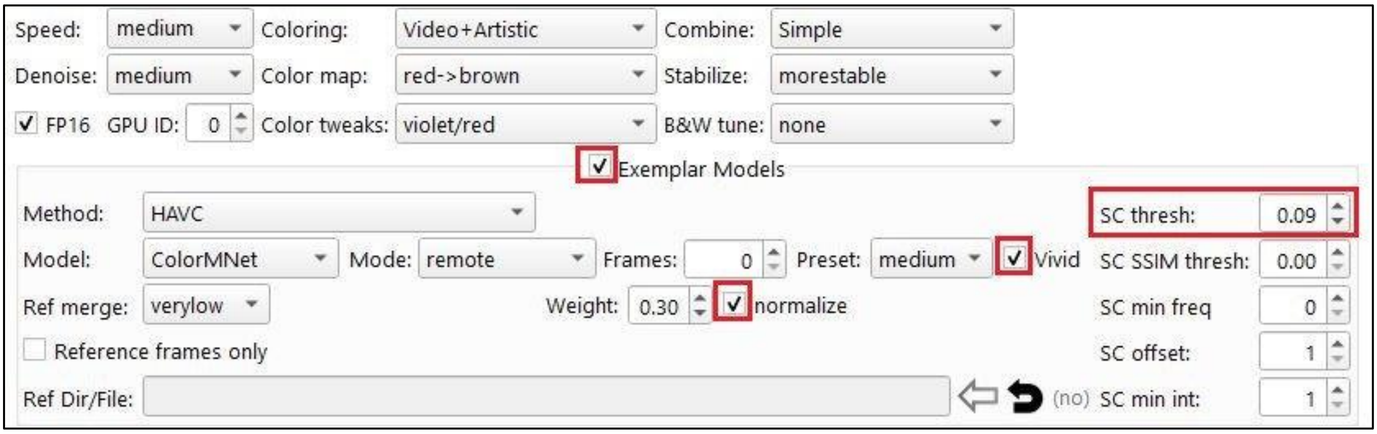
Unfortunately, by using the preset **Color map** equal to *red->brown* will change also the color of the skin toward the yellow-brown. To fix this *side effect* it is possible to add as coloring post-process, the tweak filter to shift the Hue by a value of 8.0 (or higher). This filter can be found in the panel: **Filtering->Vapoursynth->Color->Basic** as shown in the picture on the right (with some



suggested settings highlighted in red).

At the following links are available some examples of using the proposed settings to reduce the problem of colors shifting towards red: https://imgsli.com/MzM5MjQ1, https://imgsli.com/MzM5MjQ2, https://imgsli.com/MzM5MjQ3.

The approach described previously works well if there are few environments in the movie. In the case in the movie there are a lot of different environments indoor/outdoor, the result obtained could be not satisfactory. In this case it is possible to adopt a different approach by using the settings displayed in the following picture:



Respect to the previous configuration has been changed the following parameter (highlighted in red box):

- **SC thresh**: 0.09

The parameter **SC thresh** was lowered to 0.09. For low thresholds below 0.10 the filter will adopt another algorithm that is able to properly handle this low sensitivity levels[12].

For some movies the reference images produced by using a threshold of 0.10 are not enough to allow to ColorMNet to properly propagate the colors. By lowering the thresholds, the generated reference images will increase, allowing ColorMNet to improving the color propagation.

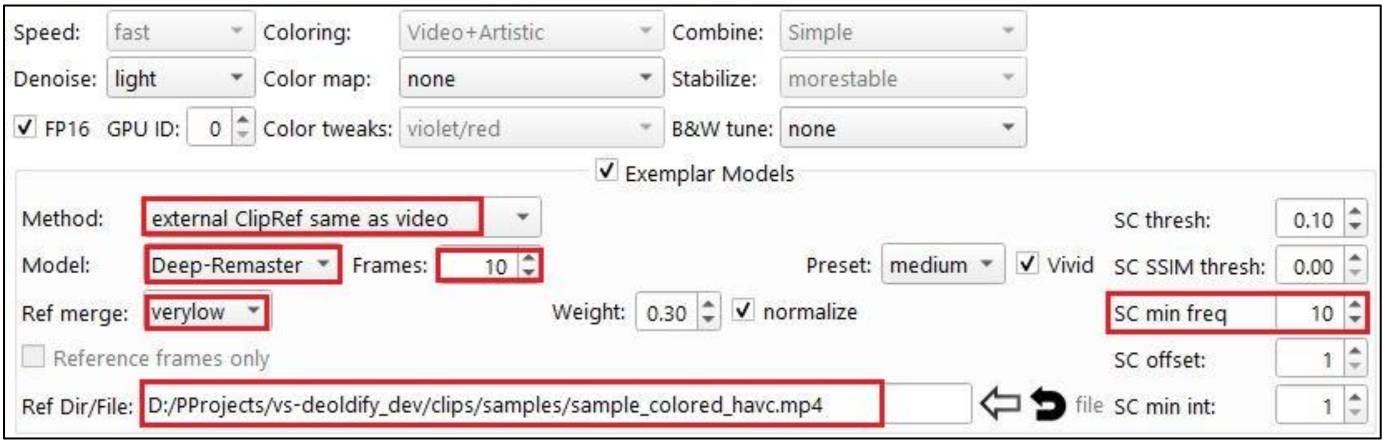
It is not suggested to decrease the threshold too much, because this could generate too many reference frames canceling the stabilizing effect of ColorMNet. This is the reason why it is suggested to start with a threshold of 0.10, because it is the threshold that will produce the maximum stabilization, and to lower it to 0.09 only if there are video sequences not properly colored[13]. Of course, is not guaranteed that by lowering the threshold the problem will be solved, but it's worth giving it a try.

Not knowing in advance if in the movie there are a lot of different environments indoor/outdoor, and not having time to do some test, probably the proposed settings using the threshold of 0.09, is the best option to produce a satisfactory colorized movie.

In the case the movie colored with the settings previously suggested is still not satisfactory because there are still frames with colors that shift towards red. It is possible to adopt a more conservative approach which consists in using the colored movies as source for an exemplar-based model. Among all the exemplar-based models the one that provides the most stable colors is DeepRemaster. Unfortunately, in some cases ColorMNet it is affected by the same DDcolor problem of introducing red artifacts in the frames, and so it is not suggested using it in this combination.

In the following picture are shown the settings with the more conservative approach available in HAVC to remove the problem of frames with colors that shift towards red:

[12] Unfortunately, the function *SCDetect* implemented in Vapoursynth is not working properly with thresholds below 0.10.
[13] If the video sequences that are not properly colored are few, it is possible to follow the approach described in the chapter on Advanced coloring using adjusted reference frames, to provide the adjusted reference frames to properly colorize these sequences.



Respect to the previous configuration has been changed the following parameter (highlighted in red box):

- **Method**: external ClipRef same as video
- **Model**: Deep-Remaster
- **Frames**: 10
- **Ref merge**: verylow (or low)
- **SC min freq**: 10
- **Ref File**: path to the colored movie colored with HAVC with unsatisfactory colored frames

For DeepRemaster a value of frames equal to 10 will allow the model to provide more conservative colors because having more reference frames will allow DeepRemaster to smooth the fast color transitions usually observed when picture-based models are used to colorize movies. A frame number of 4 is the minimum suggested, with a low number of reference frames the color inference will be about 40% faster but probably the estimated colors will be less conservative. An extension of this approach will be described in the chapter on using HAVC to restore colored videos.

4.1 HAVC Color Mapping/Chroma Adjustment

In this chapter will be described the usage of parameters colormap and chroma adjustment. As suggested in the previous chapter, the colormap parameter can be used to change a range of colors in another range, and in the previous chapter this feature was used to mitigate the problem of colors shifting towards red. In this chapter will be shown how to use it to change the color of a specific portion of a frame. While this chapter can be useful to understand how to create new color mapping, in the chapter on advanced coloring will be proposed a better way to propagate the color adjustments.

4.1.1 Example of Color Mapping

Let's start with a simple example.

Here a frame obtained by using the HAVC with the following code

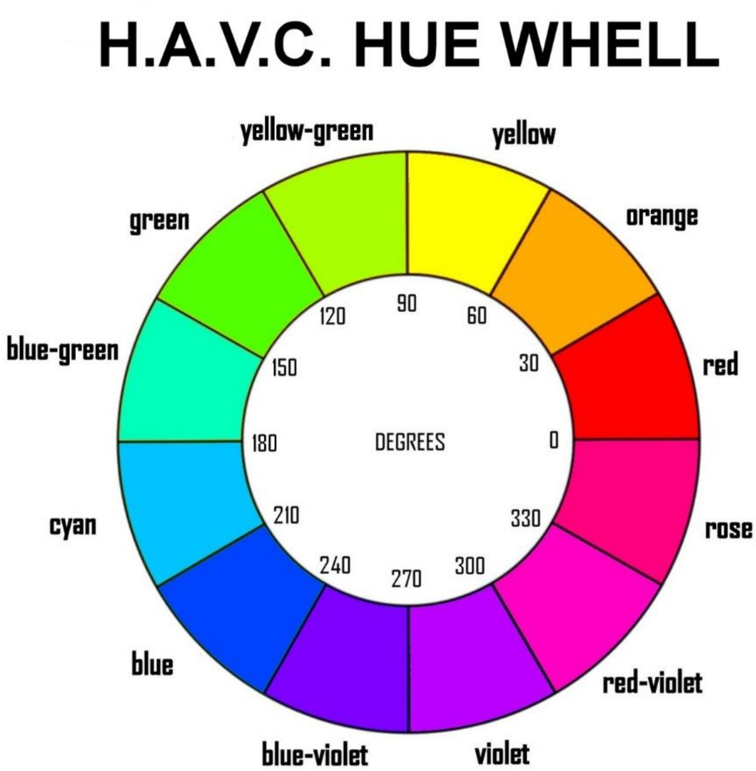
```
clip = HAVC_colorizer(clip=clip, ddtweak=True)
```

The colored frame is quite good, but the woman's hand is almost yellow.

Using the "Color Mapping" feature it is possible to correct this defect.

The "Color Mapping" feature allows to change a given range of colors.

To be able to perform the change in necessary to specify the target range of colors using the HUE defined in HSV color space.



There are a lot of Painting programs that allows to see the HUE of a range of pixels, but the simpler approach is to use the following **HAVC COLOR WHEEL**:

In the HSV color space the colors are specified in degrees from 0 to 360. In this case it is possible to see the yellow range is between 60 and 90. But to be more conservative it is better to include also the orange, so that the chroma range that we want adjust is the following: "30:90".

Now that we have selected the range; to change the colors we need to define the HUE SHIFT that need to be added to arrive to our preferred range of colors. We want to arrive in the range that in the HUE WHEEL is called "red-violet"/"rose". To arrive in this range, we need to add about 250 degrees (250+30=280->"violet", 250+90=340->"rose").

To perform this mapping is necessary to run this code after **HAVC_colorizer**()

```
clip = HAVC_stabilizer(clip=clip, dark=True, colormap="30:90|+250,0.0")
```

In the picture below it is possible to see the result obtained. The result is quite bad, but it is useful to see the range of colors that has been changed.

Now we need to specify the last parameter, the "weight". Using this parameter, it is possible to merge the image obtained using the color mapping with the original image.

In this way it is possible to blend the color differences and obtain a more realistic effect.

Since we want to apply only a little change in color, we can try to retain the 80% of the original image. This can be done by using the following code

```
clip = HAVC_stabilizer(clip=clip, dark=True, colormap="30:90|+250,0.8")
```

Here the new image:

Now the image is more realistic!

It is possible try to increase the HUE SHIFT to include also the RED component, this can be obtained by increasing the shift to 300 degrees.

Let's try the following command

```
clip = HAVC_stabilizer(clip=clip, dark=True, colormap="30:90|+300,0.8")
```

Here the image obtained:

Even this image is quite good.

To simplify the comparison was created the following album: https://imgsli.com/MjYxNjY5

4.1.2 Example of Chroma Adjustment

The "Chroma Adjustment" is similar to the "Color Mapping" the difference is that instead to apply a HUE SHIFT to the selected hue range, the selected colors are de-saturated.

Suppose, for example that in some frames the "Violet/Red" component is too strong. In this case the color is correct but it is necessary to reduce its intensity, to do that is necessary to de-saturate the color.

For example with this command

```
clip = HAVC_colorizer(clip=clip, ddtweak=True, ddtweak_p=[0.0, 1.0, 2.5, True, 0.3, 0.6, 0.7, 0.5, "300:360|0.5,0.1"])
```

the saturation of colors in the range "300:360" (that correspond to "red-violet/rose" of HUE WHEEL) will be reduced by 50% (parameter "|0.5") the final image will be blended at 10% (parameter ",0.1" after the de-saturation parameter "|0.5"). In this case the chroma adjustment will be applied only to the frames colored by DDColor.

Willing to apply the de-saturation on the final-colored frame, it is possible to use the following command

```
clip = HAVC_stabilizer(clip=clip, dark=True, smooth=True, smooth_p=[0.3, 0.7, 0.9, 0.1, "300:360|0.5,0.1"])
```

To apply the adjustments to the frames colored by `HAVC_colorizer()` it is necessary to apply the post-process filter `HAVC_stabilizer()`.

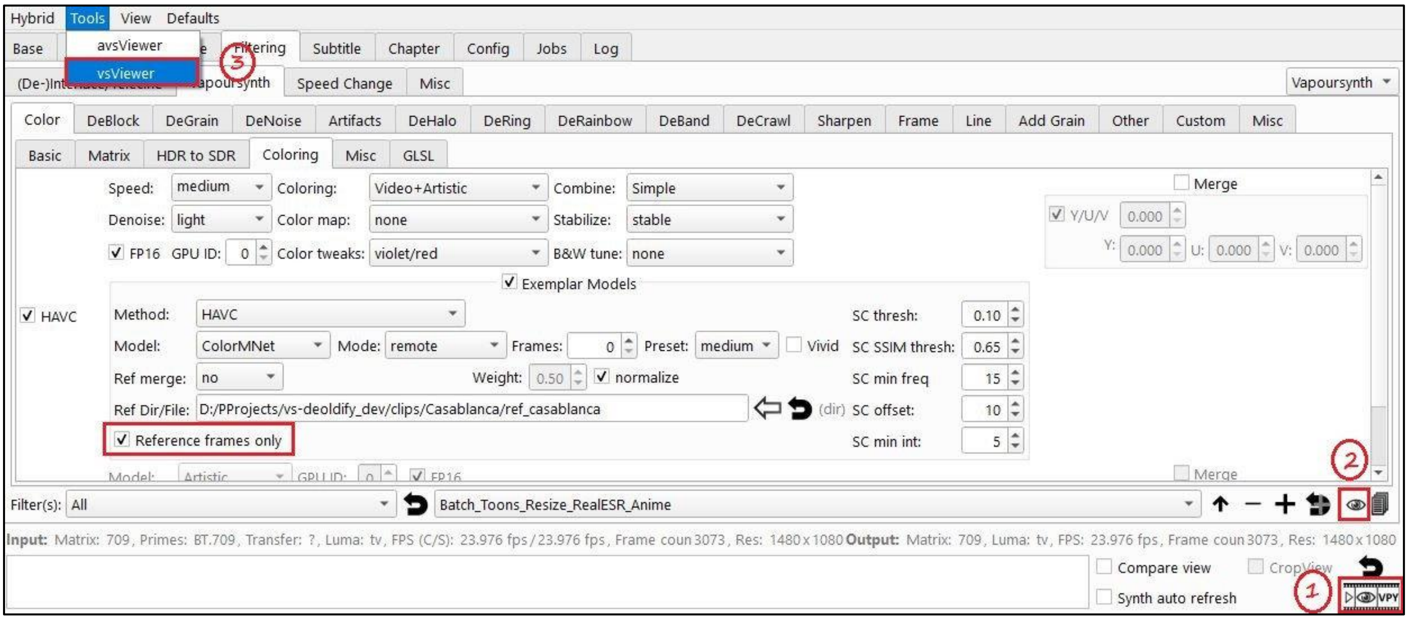
A helpful way to learn how to use these adjustments is to use the Presets.
In Hybrid when is selected a Preset different from "custom" the filter parameters will be disabled, but their values will be updated with the setting defined by the Preset.

The Preset that control the "Color Mapping" is *Color map* while the presets controlling the "Chroma adjustment" are *ColorFix* and *ColorTune*.

4.2 Advanced coloring using adjusted reference frames

In this chapter will be described how to improve the coloring process by manually adjusting the reference frames. In this guide will be used as sample movie to colorize the following clip[14]: https://archive.org/details/casablanca-1941-hd-trailer

Having downloaded the test clip, it is possible to add it in input to Hybrid using drag-and-drop. In order to be able to manually adjust the reference frames, it necessary first to generate and export them in a folder. In the following picture are shown the filter settings necessary to perform the export:



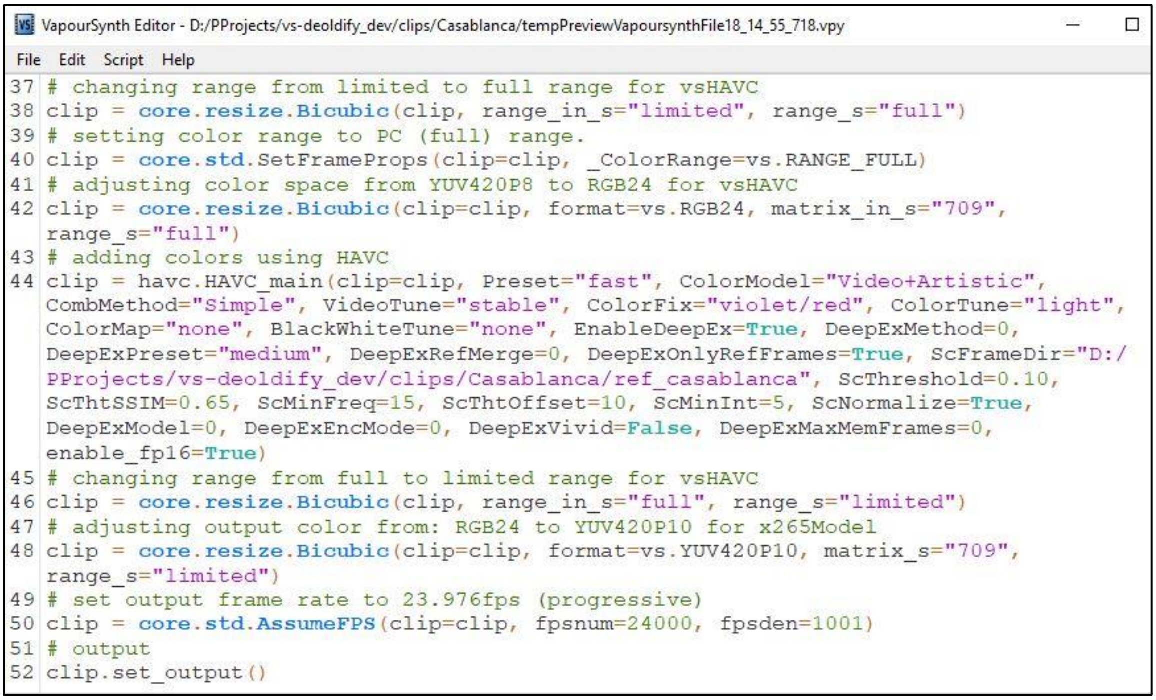
In the GUI there are 2 preview buttons. The **button (1)** will allow to preview the filtered frames, while the **button (2)** will allow to preview the code automatically generated by Hybrid. The check box **Reference frames** only is necessary to enable the export of frames that will be used by ColorMNet to propagate the colors. It is interesting to observe that in this case **SC SSIM thresh** has been set to 0.65, this is necessary to filter-out the frames that are similar, since will

[14] It is suggested to download it using the TORRENT link.

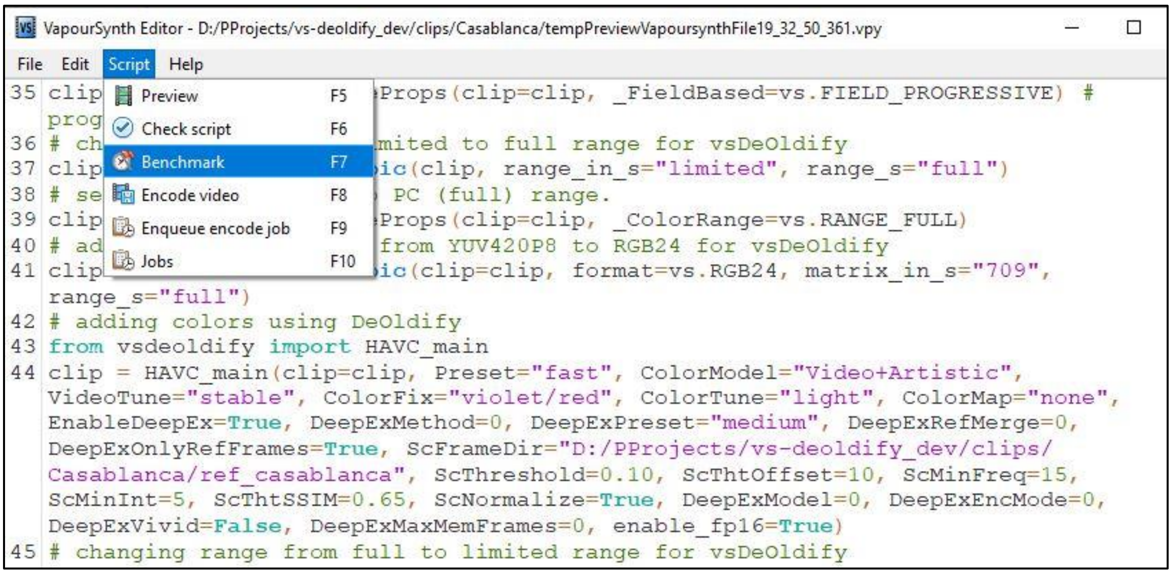
need to check manually the generated frames, is better to reduce the number of exported frames to the minimum necessary. For the same reason has been set **SC min int** to 5, this setting will guarantee that the minimum distance between 2 consecutive reference images is at least 5 frames. The parameter **SC Offset** has been set to 10 to increase the sensitivity of scene changes detection in the case of blended frames.

Once all the parameters are set it is necessary to press the **Preview button 1** (shown in the previous image) and wait (it could be necessary to wait more than 40sec) till is displayed the preview window (in this case the first frame is black).

Then is necessary open **vsViewer** (see **point 3** in the previous image) that will show the code of preview file (in this case is `tempPreviewVapoursynthFile18_14_55_718.vpy`) as shown in the following picture:

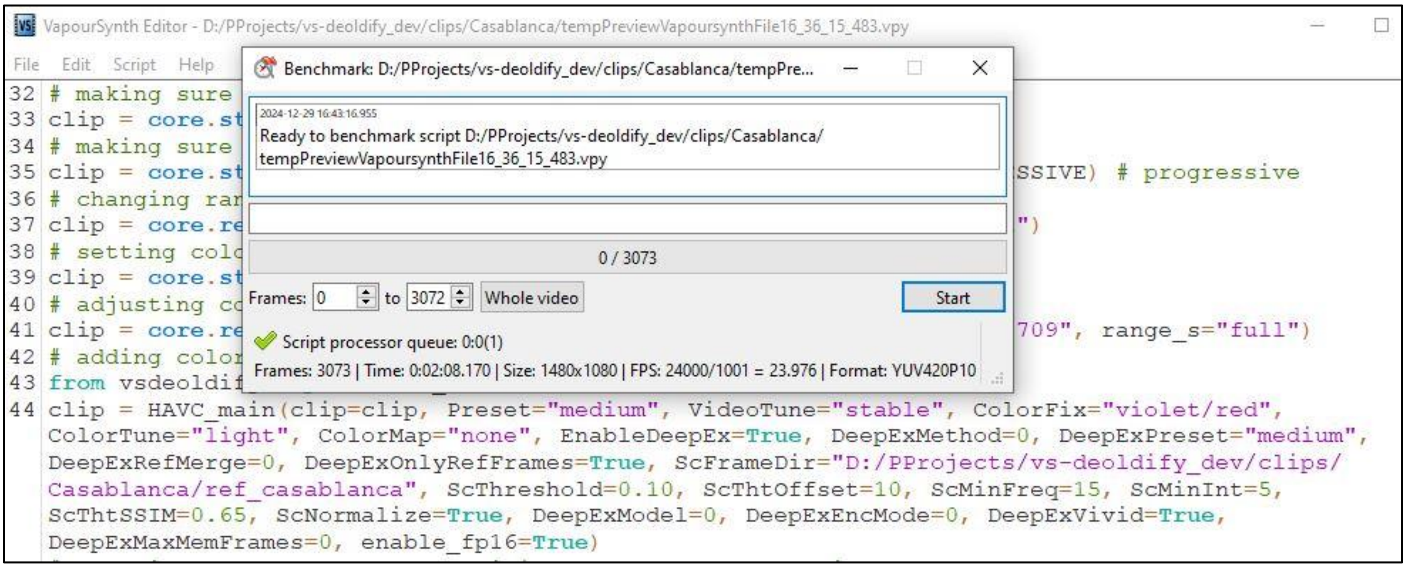


Then is necessary to select **Script-Benchmark** or press **F7**, as shown in the picture below:

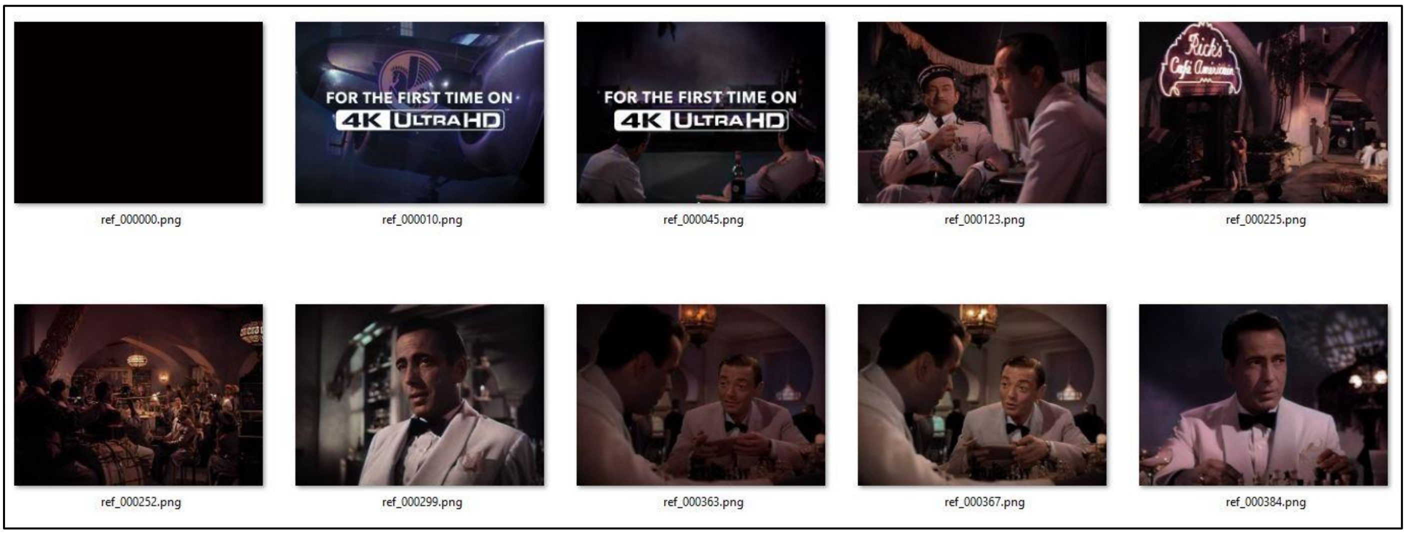


The Benchmark will run the script but will not call the encoding process, to that will not be generated any movie file. This is useful because in this case, what is necessary to generate, are the reference frames and not the clip.

Once the Benchmark is selected will be displayed a window like this:



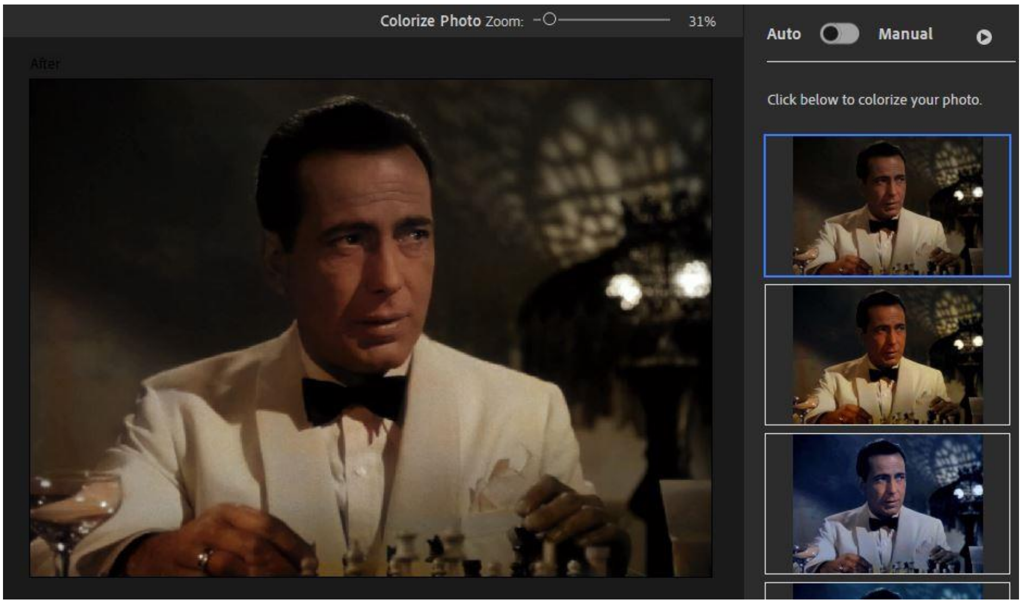
At the end of the Benchmark in the directory defined in the parameter ScFrameDir will be available all the reference frames that will be used by HAVC for coloring the B&W clip. In this case should be available 108 frames out of 3073 frames contained in clip, so about 3.5% of the frames were selected as reference frames for the selected coloring model (in this case ColorMNet). Then is possible to look at the reference frames that will used for coloring the clip as shown is the following picture:



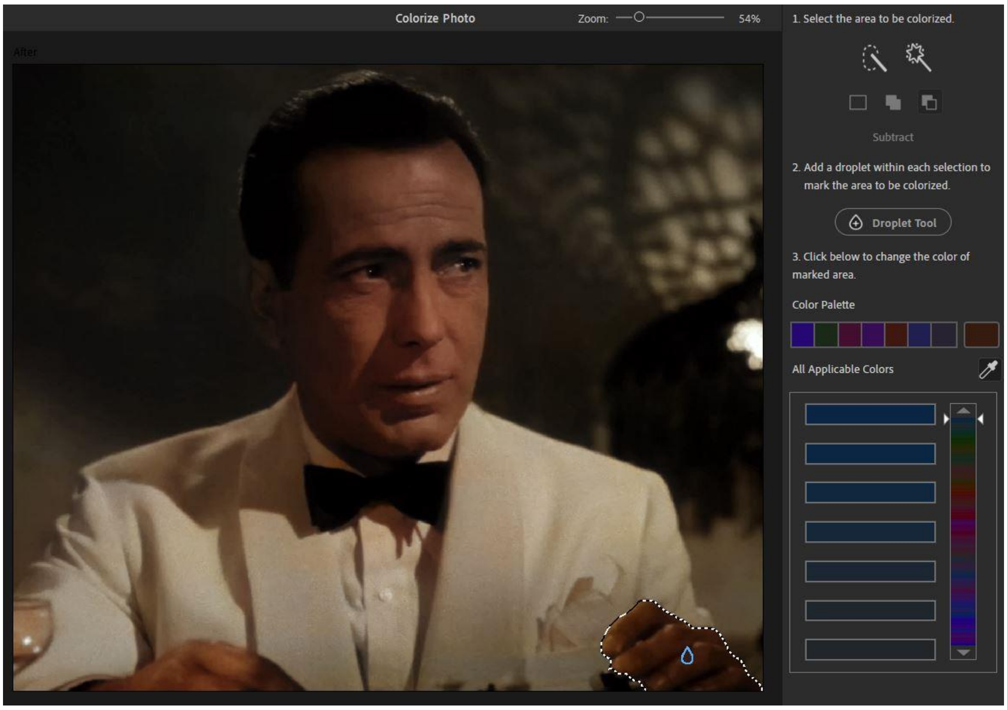
In the sample provided above is possible to see that the frames 363 and 367 are very similar, but 367 has better colors, since in this case we are interested in providing the best *exemplars* to ColorMNet, it is possible to delete the frames 363 and rename the frame 367 in ref_000363.png. When there are similar frames, like in this case, it is better to keep always the frame that appears first, eventually replacing it with a better colored frame.

Then is possible to see another common situation, a frame that has wrong or not appropriate colors: in the frame 384, the Humphrey Bogart's jacket is almost pink and not white as it should be. In this case, it is necessary to correct the color. To correct the color, it could be possible to user the Color Mapping procedure described in the previous chapter, but is quite complex because is missing a dedicated GUI to perform this kind of mapping. A simplest way is to use a dedicated software as suggested chapter Useful companion software. In this case it will be used Photoshop Elements 2024 (see software for coloring pictures).

Using Photoshop Elements, the proposed colored image is quite good, as is possible to see in the following picture:



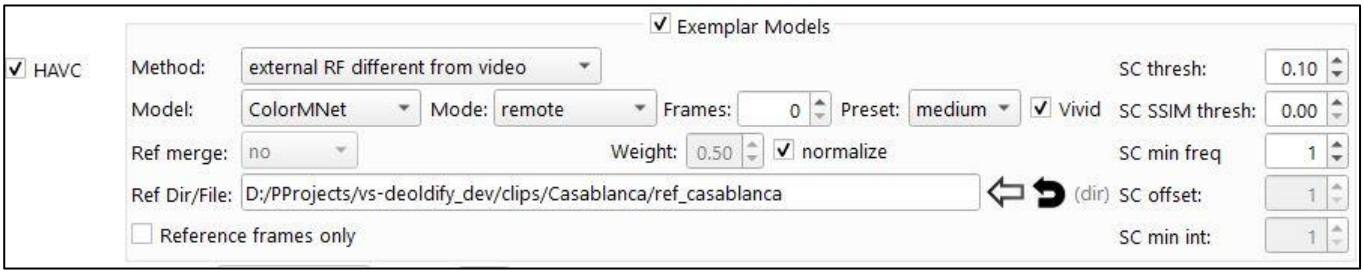
Buth the hand on the right is not colored well and is necessary to adjust the color manually as shown in the picture below:



The situations described previously represent the most common cases that need to be addressed:

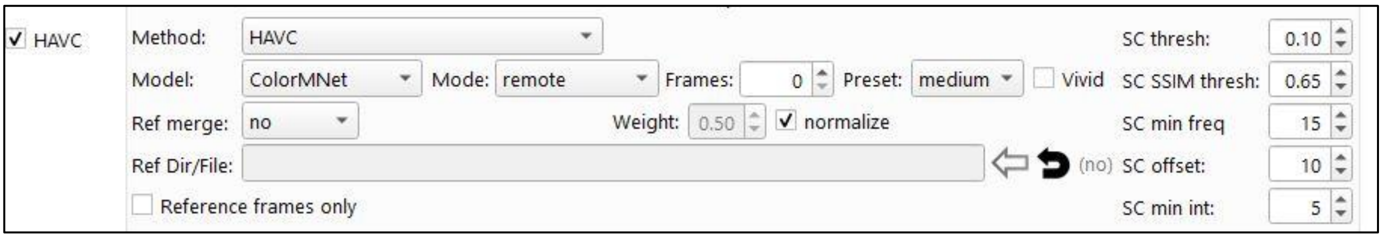
1) similar or duplicated frames: in this case is necessary to selected the best frame (eventually by renaming it) and delete all the remaining frames.
2) frame with wrong colors, in this case it is necessary to adjust the colors.

After having adjusted all the reference frames is possible to finally start to coloring the clip using the settings shown in the following picture:



Having selected the method **external RF different from video**, the clip will be colored using only ColorMNet and the reference frames provided in the folder: “ref_casablanca”.

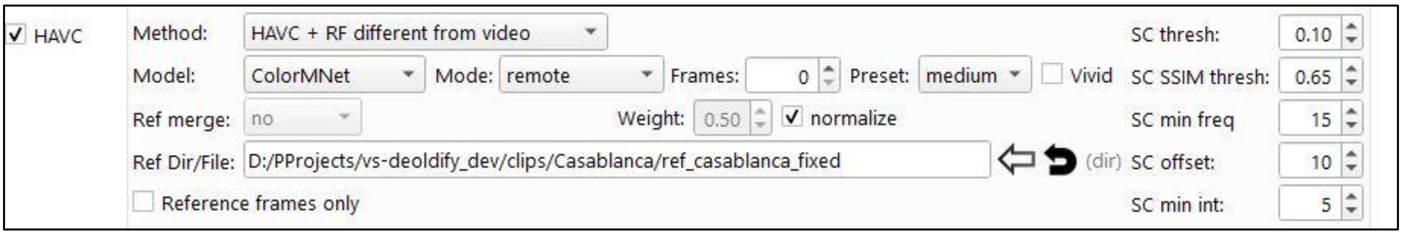
Using the standard method HAVC as shown in the following picture:



the clip will be colored using the previous unadjusted reference frames, so the clip will show the color artifact observed previously, for this reason has been shown how to adjust the reference frames to improve the color quality.

Alternatively, is possible to create a folder that contains only the fixed/adjusted reference frames, as shown in the following picture:

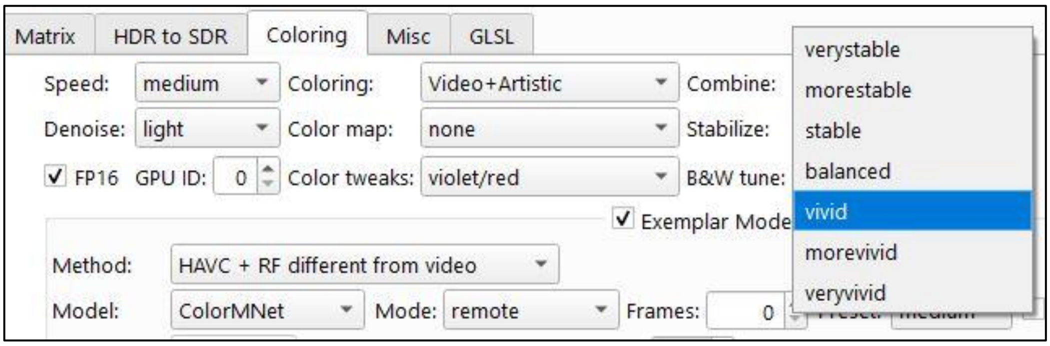
Supposing that the folder is named “ref_casablanca_fixed”, it is possible to color the clip using the following settings:



In this case the filter will use as reference frames the ones colored using HAVC but the frames found in the folder specified by the parameter **Ref FrameDir** will have higher priority and eventually will override the frames generated by HAVC. It is suggested to use **HAVC + RF different from video** even if in this case is appropriate to select **HAVC + RF same as video** since the reference frames were obtained from the same clip that HVC will colorize.

By using **RF same as video** ColorMNet will skip the inference and will provide in output exactly the same colors specified in the reference frame, but the next frame will be colored using the inference and this could lead in some color discontinuity between the reference frame and the next frames. By selecting **RF different from video**, ColorMNet will apply the inference even on the reference images and this will assure more color uniformity between the reference image and the next frames.

The reference frames were obtained using the suggested settings for HAVC (see picture below), with the parameter Stabilize set to **stable**, but it could be also possible to set it equal to **vivid**, in this case will probably necessary to perform more color adjustments.



It is necessary to clarify that by using *AI automatic colorizers* is not possible to get colorful movies with a great variety of colors, because this will increase the instability of colors at a level that the colored movies will be almost unwatchable.

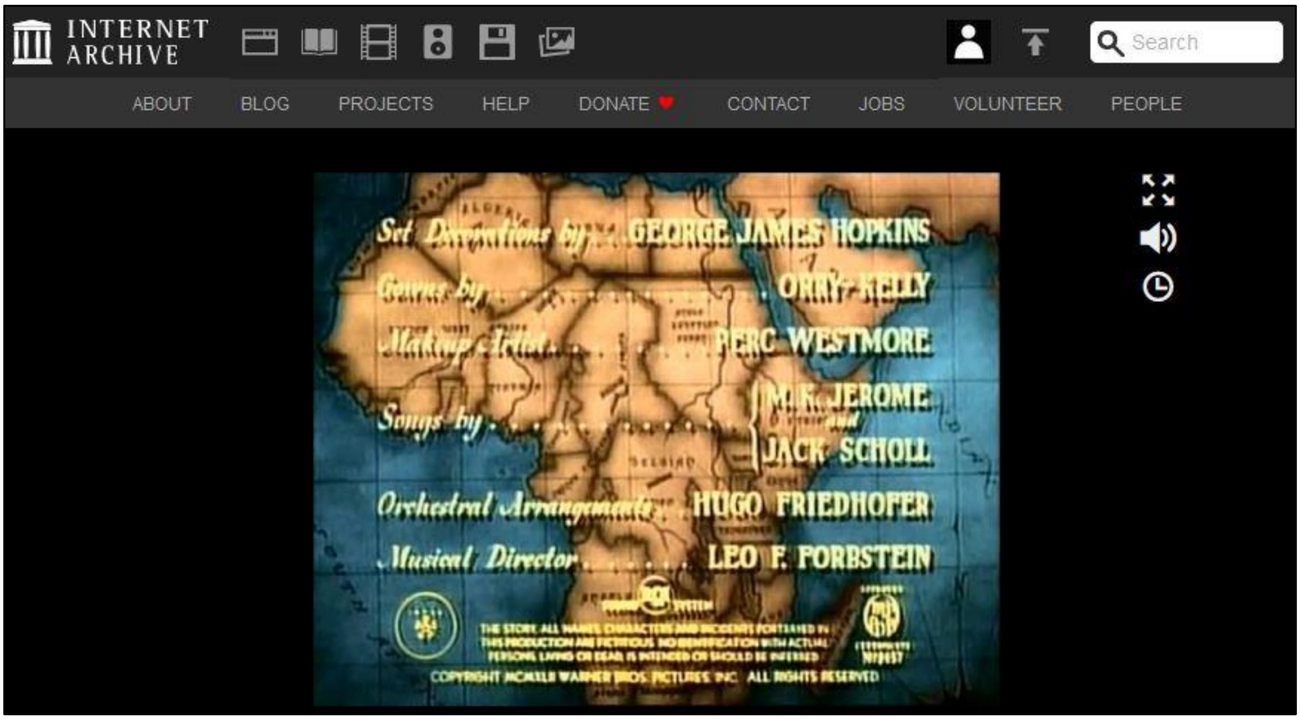
To get colorful movies with a great variety of colors is still necessary a lot of manual work. Using ColorMNet with the method **external RF different from Video** will be possible to obtain colorful movies by providing colored reference images with a great variety of colors.

At the following link there is an example of movie obtained using this approach: [The Thing (Colorized, 1951)].

Several thousand low-resolution reference images (already available) and hundreds of manually colored images were used to color this clip. But this is a very time-consuming task and *AI automatic colorizers* have been developed precisely to avoid this (boring) manual task.

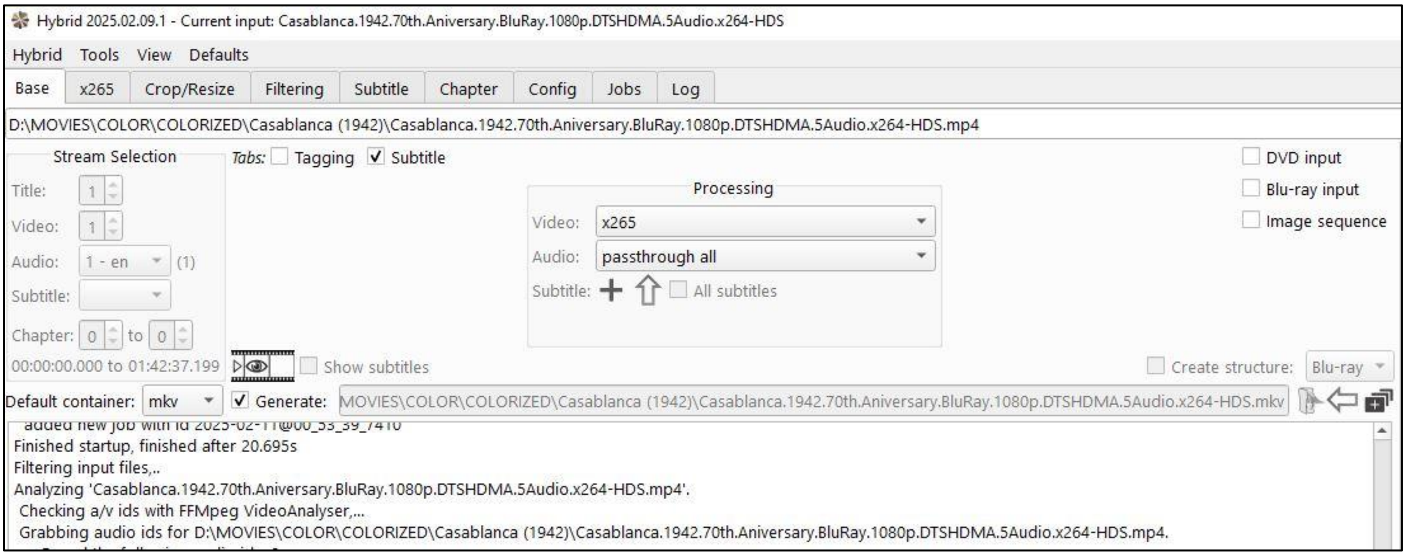
4.3 Using HAVC to restore old colored videos.

Starting from HAVC 5.0 it is possible to use the filter to restore old colored videos. To explain how to use this feature will be used as example the colored video: [Casablanca InColor](#).



This is a low-quality movie, and is almost unwatchable, but with HAVC 5.0 it is possible to restore it to full HD resolution. To do that is necessary the availability of a good HD copy of the movie in B&W. Fortunately in this case is available this HD version of the movie: [Casablanca (1942)](#).

Now to restore the colored movie it is necessary to load in the main Hybrid page the HD version as shown below:

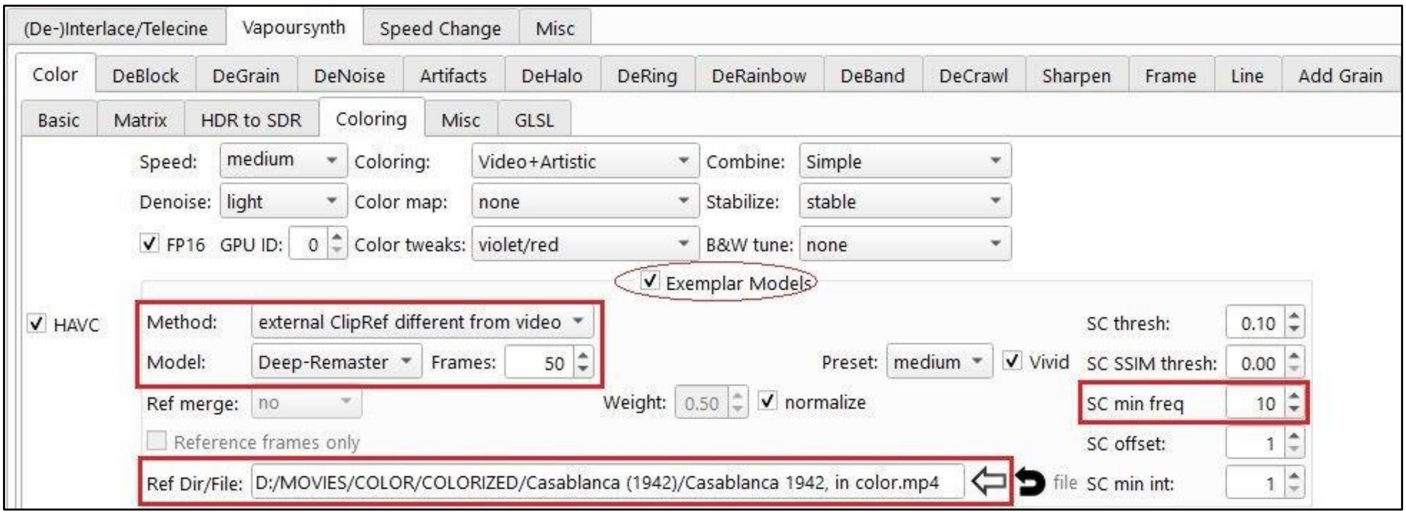


To restore the video, it is necessary to move to the page at: **Filtering->Vapoursynth->Color->Coloring**.

In HAVC 5.0 has been added the possibility to use an external clip as source for the reference images to be used by the exemplar-based models to colorize a B&W movie. In HAVC 5.0 are available 3 models: ColorMNet, DeepEx and DeepRemaster. In the case the B&W movie and the reference-colored movie are perfectly in sync all the 3 models are suitable to be used to propagate the colors of the refence clip in the B&W movie. But in the case, they are not perfectly in sync, as often happens, only DeepRemaster is able to properly propagate the colors. This is possible because in the

HAVC implementation, has been adopted the strategy to provide in input to DeepRemaster 50% of past reference images and 50% of future reference images, respect to the frame to be colored. In this way DeepRemaster is able to manage the situation where the reference frames are either ahead or behind the frame to colorize. DeepRemaster store the full frame in a tensor array and in this way is able to properly apply the colors without compromise. The others models are not able to manage the reference frames in this way.

Since in this case the 2 movies are not in sync, there is a difference of about 200 frames randomly distributed between the 2 movies. It will be used DeepRemaster with the following settings:

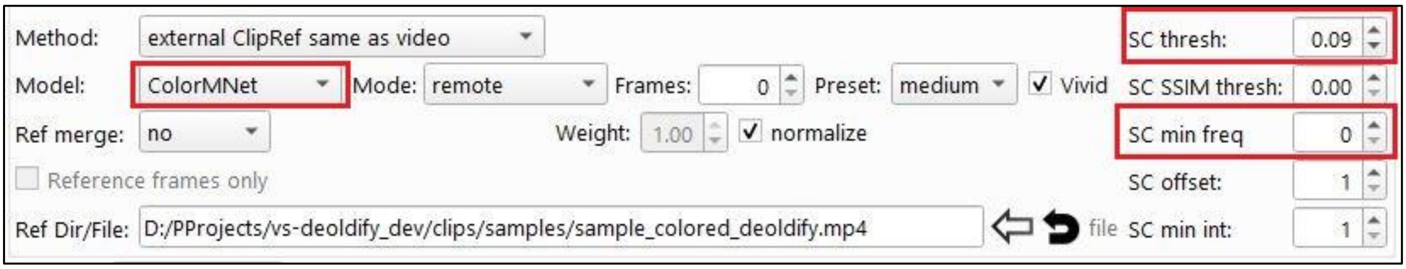


First all is necessary to check the box **Exemplar Models**. Then is necessary to select the model **Deep-Remaster** with Frames equal to 50[15], finally is necessary to select the Method **external ClipRef different from video** and click on the big arrow on the right of the box **Ref Dir/File**. DeepRemaster need a lot of reference frames to properly colorize the movie so it is necessary to set the parameter **SC min freq** to a value between 5 and 15 (in this case was set 10).

Now it is possible to start the encoding and if all goes well after a few hours the colorized HD version should be available[16].

A version of the movie in half-HD colored using this approach is available at: casablanca-remastered-colorized-1942.

The color restoration can be applied also if is available a HD version of a movie colored with another tool (for example DeOldify) in this case could be used the model ColorMNet to provide more stability to the movie colors as shown below:



In this case it is possible to use the movie also as main clip in Hybrid to colorize. In this way the movie will be (automatically) converted in B&W and then (re)colored using ColorMNet. From the colored movie will be taken only the reference images, the remaining frames will be colored by ColorMNet, thus providing more color stability.

[15] The maximum suggested value for the frames to be used with DeepRemaster is 50. If the source is perfectly in sync can be used a lower number of frames 4 or 10. If the difference in frames between the 2 movie is above 200, it is necessary to split the movies in chunks to reduce the difference on the single chunk.
[16] See the Example1 for a VapourSynth script using this approach.

4.3.1 Fixing DeepRemaster problems

As stated previously DeepRemaster is the only model that is able to restore old colored movies. But depending on the quality of old colored source, DeepRemaster can be affected by the following problems.

1. Flickering

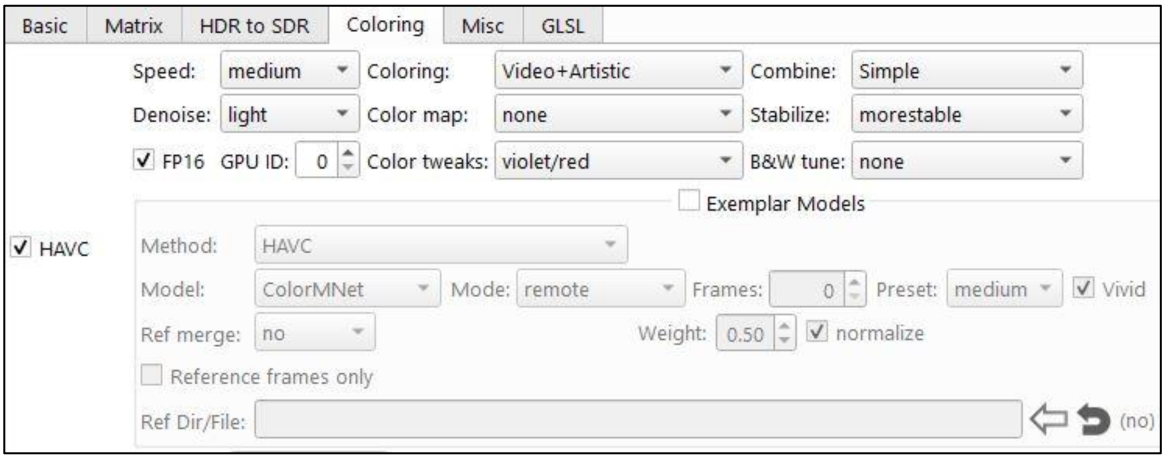
It could happen that if the source is affected by some small flickering effect, DeepRemaster will amplify the effect more than 10 times. In this case is necessary to apply a de-flicker filter. In the Example3 there is a Vapoursynth script which uses in combination 2 of most effective filters to move the flickering: HAVC_stabilizer and ReduceFlicker.

2. Inability to apply colors correctly in dark scenes

If the quality of colors in the source movie is bad, DeepRemaster will not be able to properly colorize the frames in dark scenes. An example of such problem is shown in the picture below.

(colored source frame) (DeepRemaster output) (frame fixed with Example4)

In this case to fix the frame it is necessary to colorize the full movie with HAVC using for example the Coloring model **Video+Artistic** and the Combine method **Simple** as shown in the picture below.



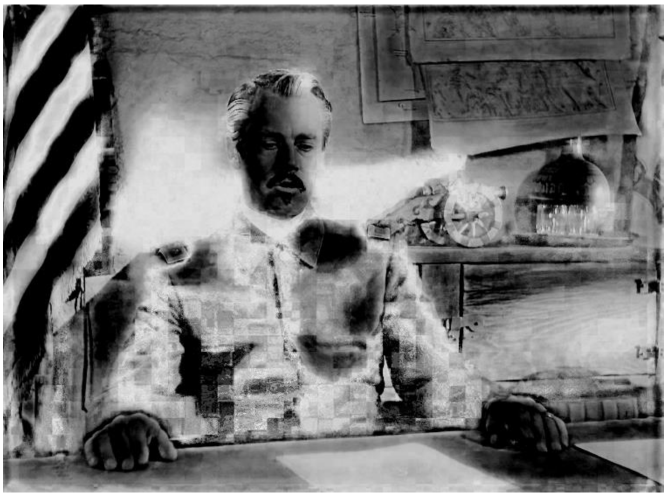
Once the clip colored with HAVC is available it is possible to use it to get the proper colors to apply in the dark scenes. Since in this case the problem is mainly limited to the dark scenes is not suggested to apply a simple merge as shown in the Example2. In this case is better to use the Merging Methods developed in HAVC. In this case the most appropriate merging method to use are Luma Masked Merge the Adaptive Luma. To use these methods is necessary to write a simple Vapoursynth script as shown in the Example4 which uses the function HAVC_merge to combine the 2 clips. In this case was used the Adaptive Luma method, because is able to change the weight adaptively with the Luma, providing more weight to clipa in the dark scenes and more weight to clipb in the bright scenes.

3. Inability to apply colors correctly both in bright and dark scenes

If the quality of colors in the source movie is very bad, DeepRemaster could not be able to properly colorize the frames both in bright and dark scenes. In this case the most frequent artifact is that some portion of the frame is desaturated, almost gray as shown in the pictures below:

Even in this case is necessary to colorize the full movie with HAVC and is not possible to use a simple merge. In effect, to fix this problem is necessary to use a more sophisticated approach which consists in identifying the desaturated regions of the frame and apply a color substitution only on these regions. To do that is necessary to build a mask where the identified regions are white with a gray gradient around the white region to be able to apply a smooth color substitution. To identify the regions is necessary to provide a threshold that defines when a pixel in desaturated. In the HSV color space the saturation has value in the range [0, 255], so a reasonable threshold can be in the range [15, 60]. In the picture below are shown the masks built with a threshold of 60 in the HSV color space.

(Binary Mask) (Gradient Mask)



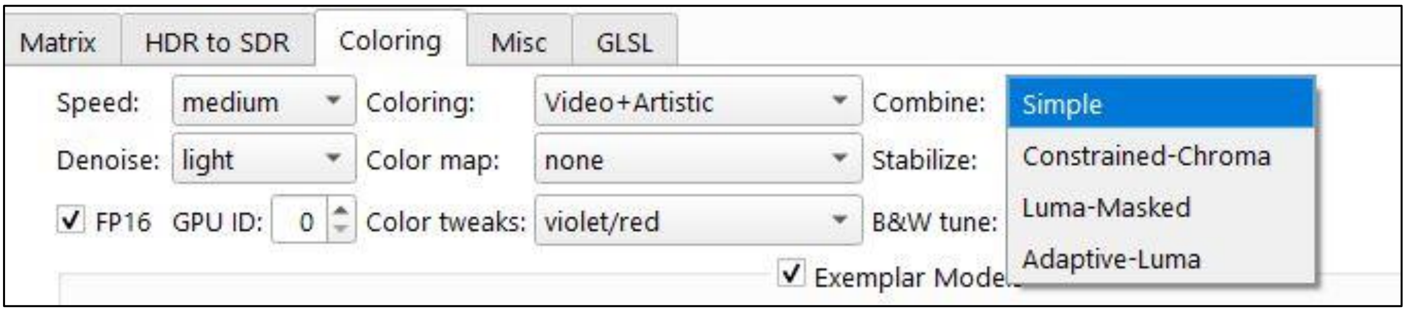
For this type of problem is not possible to use a simple binary mask where the white pixels are substituted by the colored pixels, but the binary mask could be useful to easily visualize the desaturated pixels identified using the assigned threshold. As it is possible to see, in this example, has not been identified only the right side of the face, but also the wall because is almost white and the uniform. As stated previously, to apply a smooth color substitution is necessary to build a gradient mask as shown in the picture above on the right.

To implement this special case of color substitution has been implemented a new HAVC filter function that is available with HAVC version 5.0.4 and is called: HAVC_recover_clip_color. The main parameters of this function are the threshold level tht (previously explained) and the acceleration parameter alpha (values above 2.0 will preserve more pixels, but could introduce some artifacts).

In the Example5 there is the Vapoursynth code using the proposed approach. In the picture below are shown the frames (re)colored using the given example.

4.4 Using HAVC Models merging

With HAVC it is possible to merge 2 frames generated by different models. To perform this merge are available 4 methods, that can be selected with the parameter **Combine**.



The parameter **Combine** allows to select the merging methods used by HAVC to merge the frames colored with DeOldify and DDcolor. The simplest method is **Simple** that merge the frames using the weight defined in the parameter Merge weight.

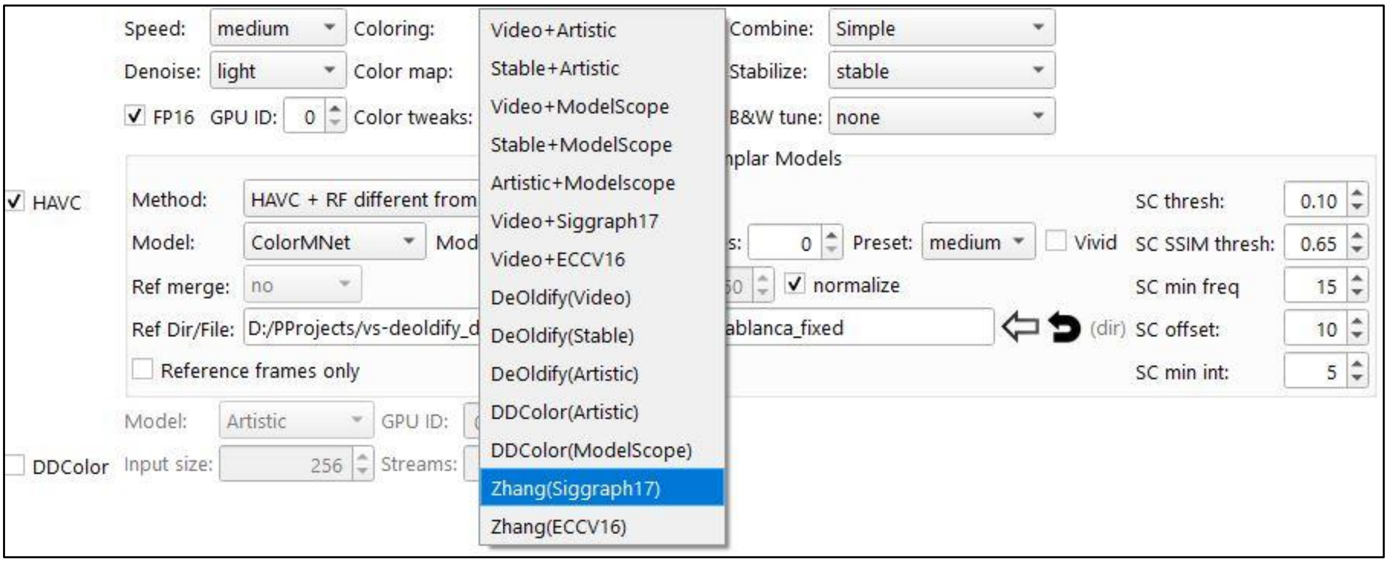
With the method Constrained Chroma, the frames are combined by assigning a limit to the amount of difference in chroma values between DeOldify and DDcolor. This limit is defined by the parameter *Threshold*, as shown in the picture on the right.

With the method Luma Masked, the frames are combined using a masked merge. The pixels of DDcolor's frame with luma < luma_limit (called *Luma* on the GUI) will be filled with the de-saturated (parameter *Sat* on the GUI) pixels of DeOldify, while the pixels above the white_limit threshold (called *White* on the GUI) will be left untouched. All the pixels in the middle will be gradually replaced depending on the luma value. If the parameter *merge_weight* is < 1.0, the resulting masked frames will be merged again with the non-de-saturated frames of DeOldify using the *Simple Merge*.

With the method Adaptive Luma, the frames are combined by decreasing the weight assigned to DDcolor frames when the luma is below the *luma_threshold* (called *Thresh* on the GUI). For example, with: *luma_threshold* = 0.6 and *alpha* = 1 (called *Exp* on the GUI), the weight assigned to DDcolor frames will start to decrease linearly when the luma < 60% till *min_weight* (called *Weight* on the GUI). In practice this method is a *Simple Merge* where the weight decreases with luma.

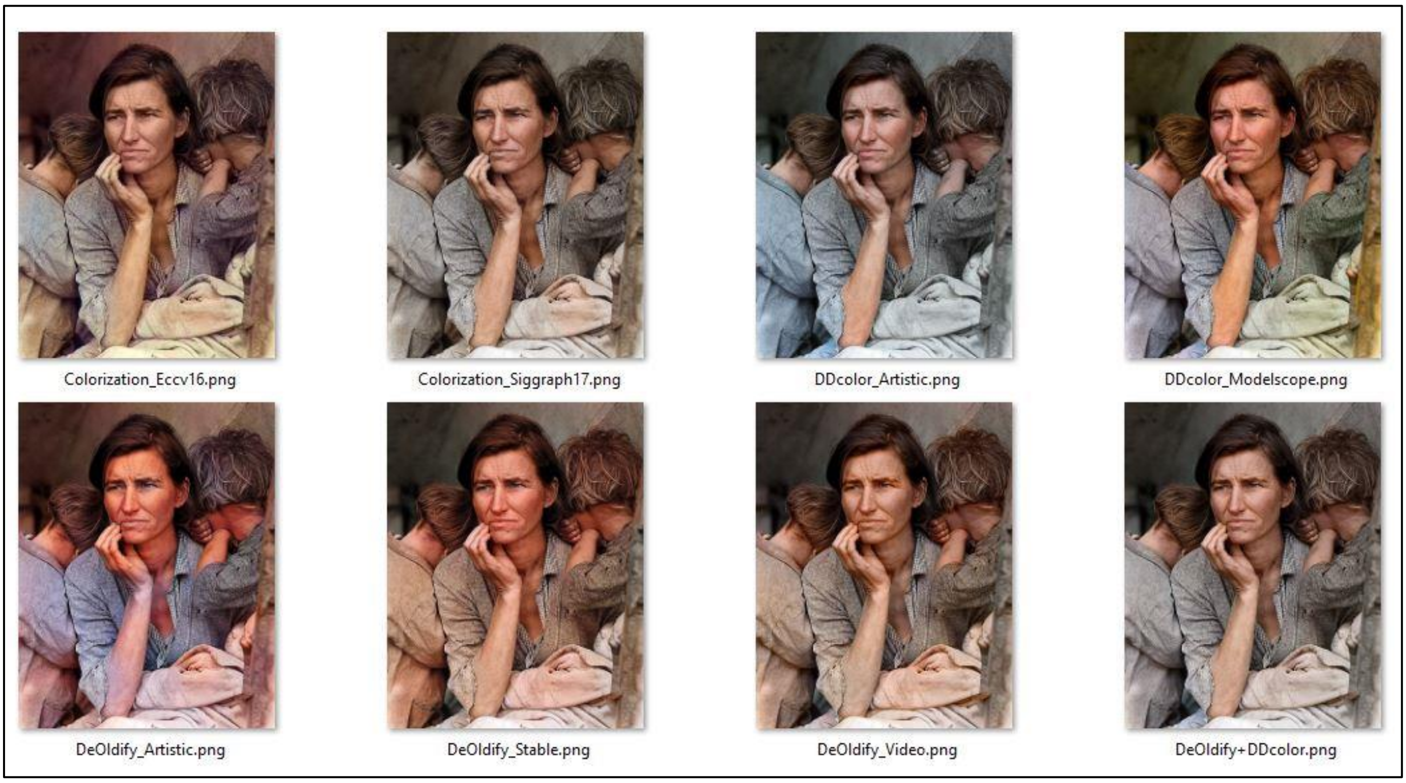
4.4.1 Alternative inference models to DDcolor

In alternative to DDcolor, starting from version 4.6.0 of HAVC is also possible, to use for the color inference the 2 models



provide in the project Colorization: [Real-Time User-Guided Image Colorization with Learned Deep Priors]() (named: **siggraph17**) and [Colorful Image Colorization]() (named: **eccv16**). These models have the same color instability observed in DDcolor and hence have in common the same settings and tweaks of DDcolor. It is suggested to try them without the Tweaks activated (*Denoise* and *Color tweaks* set to **none**) to see the improvement of this post-process filter on the colored frames.

In the picture below is possible to see a comparison between the 2 alternative models **siggraph17** and **eccv16** with the other models implemented in HAVC:



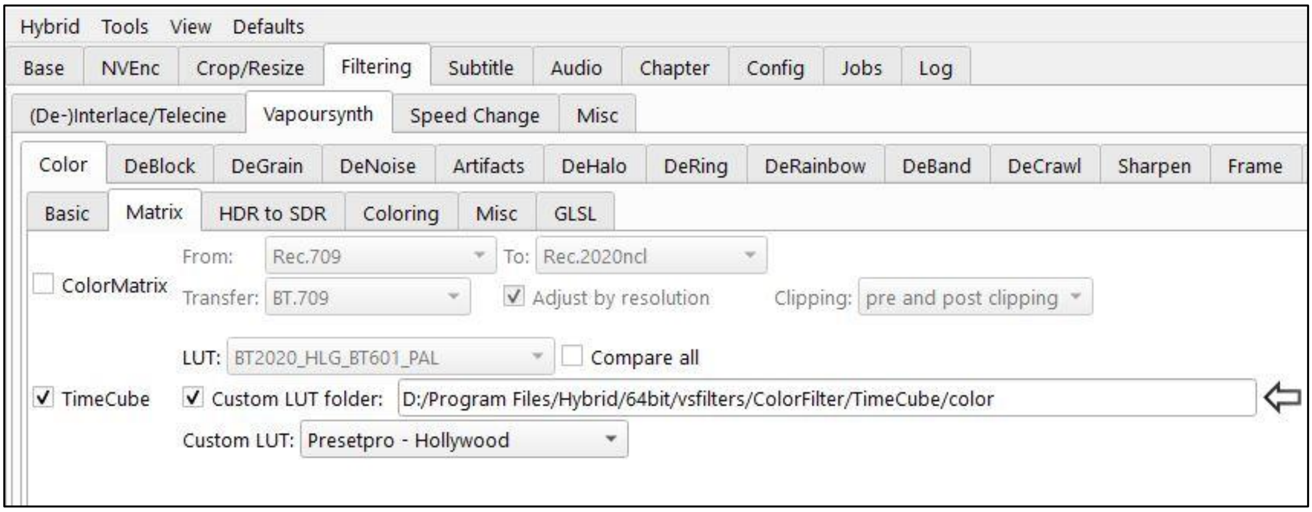
5.0 Using external filters to improve final HAVC color quality

In some cases, it is possible to improve the final color quality of the movies colorized with HAVC using other filters available in Hybrid. As explained previously, the color stability has a cost in terms of variety of colors. With the increase of color stability will decrease the variety and saturation of the colors. Some useful filters, which will possible to add as post-process to improve the saturation of the colors, can be found in the panel: **Filtering->Vapoursynth->Color->Basic**. In this chapter will be provide a guide on how to use a couple of more advanced filters to improve the final color quality. Unlike the Basic color filters, the proposed filters are not suitable for all the movies. It is better to check the impact of the filters on a small clip sample to see if are suitable to be used for the specific movie to colorize.

5.1 Using LUT (Lookup Tables) as post-process filter

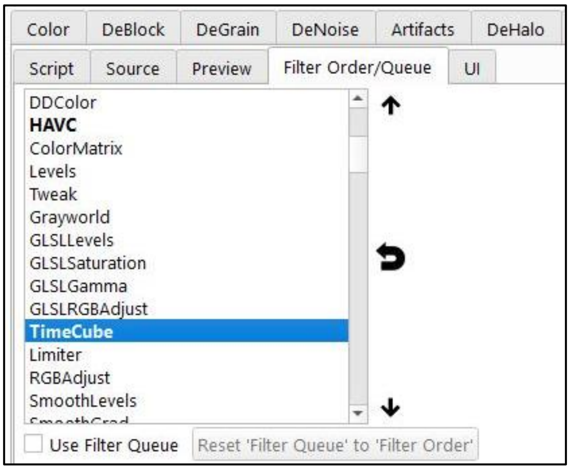
LUTs (Lookup Tables) are a kind of post-process color filter that can be used to alter the colors of final clip colored with HAVC. They apply predetermined sets of mathematical formulas to video's existing colors to change those colors and achieve a desired result. They make adjustments to gamma, contrast, saturation, luminance, and hue, essentially taking the original set of colors and changing them into a new set of colors. And they do so completely automatically. Simply put, LUT are powerful tools that can be used to elevate the color correction and color grading of HAVC colored clips.

In Hybrid there are already some interesting color LUTs, they can be selected using: **Filtering->Vapoursynth->Color->Matrix**, as shown in the following picture:



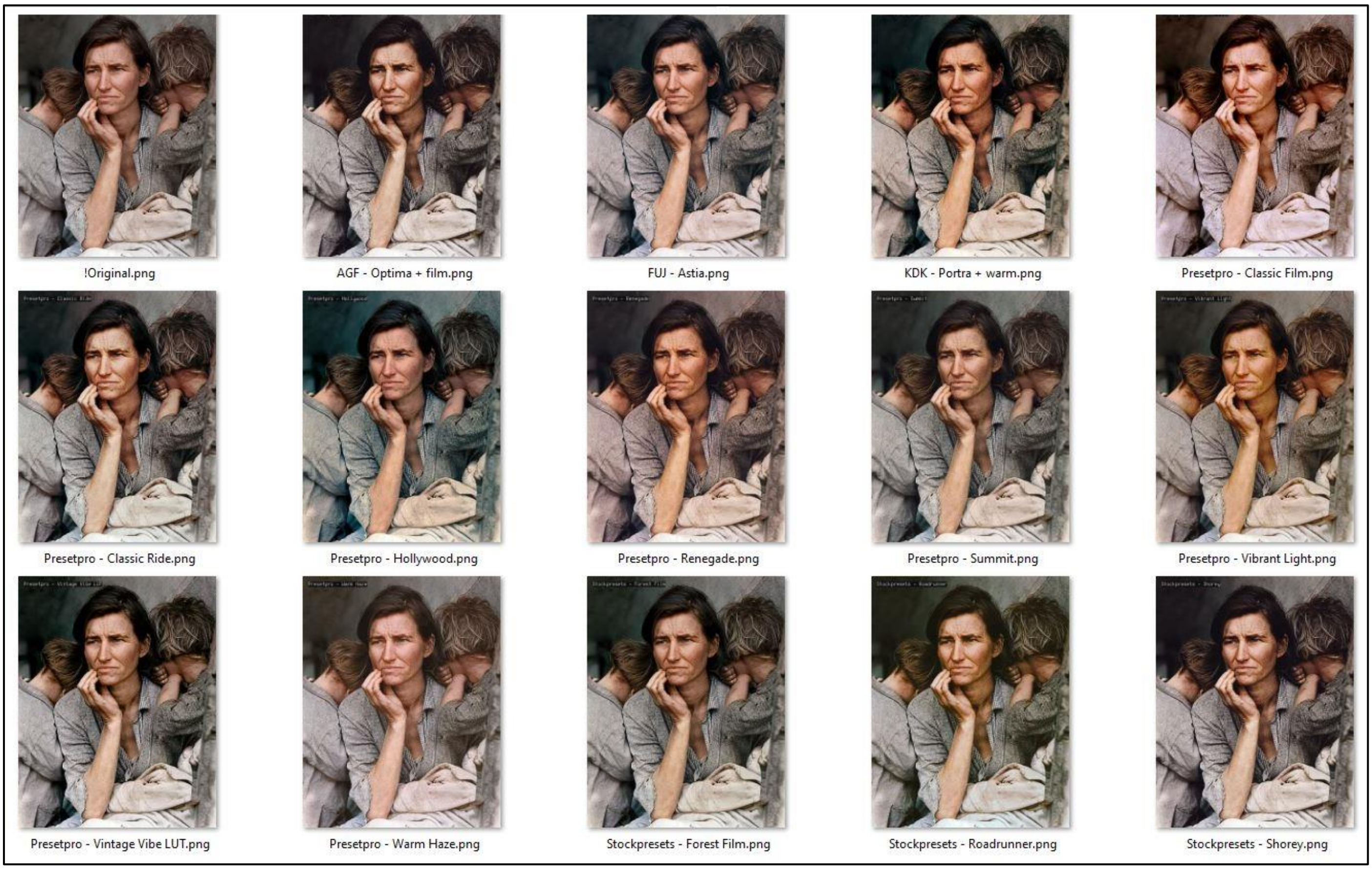
The LUT can be applied to the final video already encoded, and in this case, it is just enough to apply the above settings, where has been selected the **Custom LUT folder**. In the field **Custom LUT** is possible to select the LUT table to apply (in this example was selected the LUT named: *Presetpro – Hollywood*).

If one has already a clear idea of the LUT table to apply, it is possible to add the LUT mapping as a post-process color filter by modifying the filter Order/Queue, as shown in the picture on the right.



It is necessary to verify that the filter TimeCube is located after the filter DeOldify in the Order/Queue. By using the up and down arrows shown on the GUI.

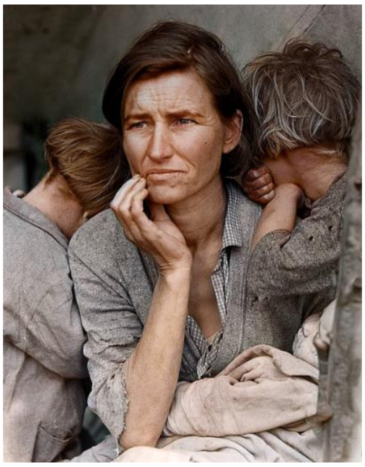
In the following picture is shown a sample of the effects obtained using the LUTs available in Hybrid:



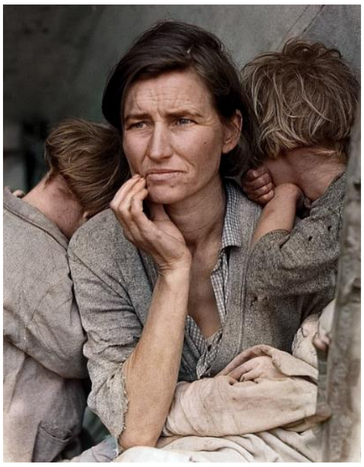
Some of them are able to change the final color of the image in an interesting way.

In the next chapter will be proposed to use **Retinex** as pre-process filter, the following picture show the result obtained using this approach with the reference image already used for the LUT comparison.

(Retinex + HAVC)



(HAVC only)



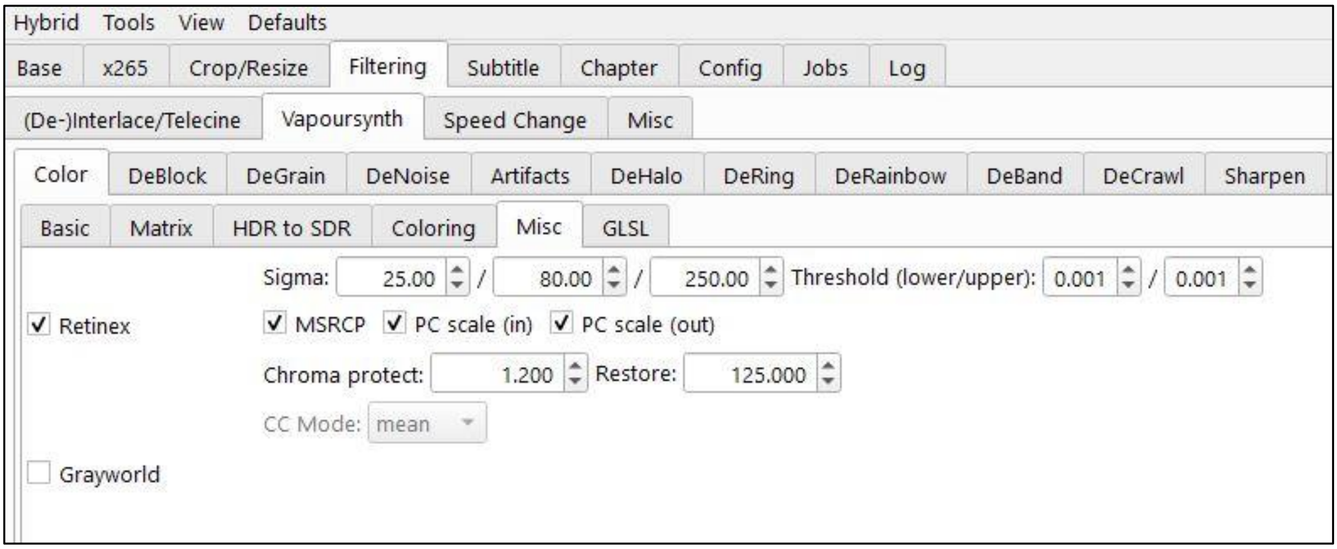
In this case the difference is similar to the result that could be obtained using LUTs, because the starting luminosity of the image was good. The differences with dark images are more significant and [Retinex could introducing artifacts] in the colored frames.

5.2 Using Retinex as pre-process filter

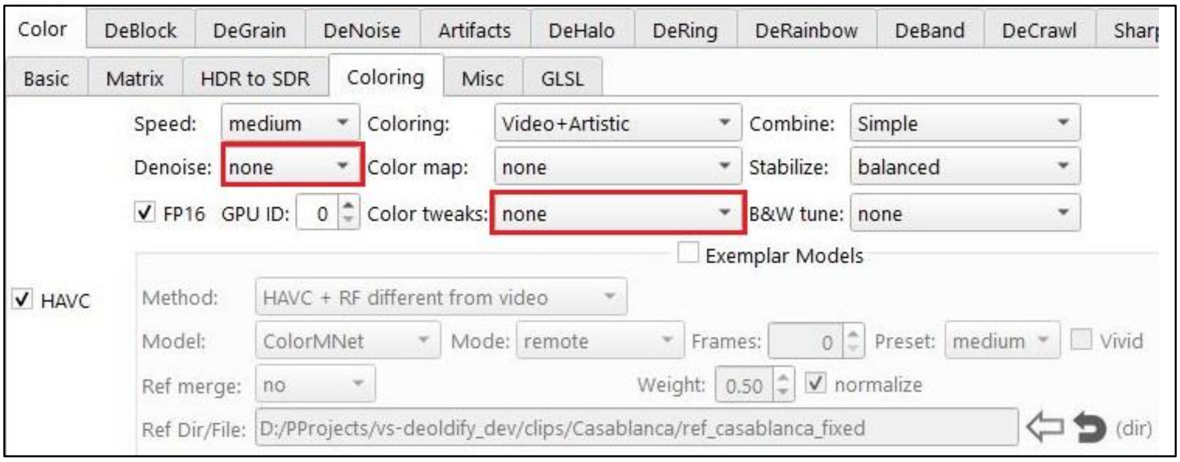
The Retinex filter available in Hybrid is the implementation of the theory of human color vision proposed by Edwin Land to account for color sensations in real scenes. The basic Retinex theory is that the color of an object is determined by the ability of the object to reflect light in long (red), medium (green), and short (blue) light, rather than by the absolute value of the intensity of the reflected light.

The color of the object is not affected by the illumination non-uniformity, and the Retinex filter is based on the consistency of color perception (color constancy), in this way the Retinex filter can balance dynamic range compression, edge enhancement and color constancy, so that it can be used successfully as a pre-filter for the color models implemented in HAVC. Given that the Retinex filter change in significant manner the images it is necessary, after having used the filter to colorize them, restore the original luminance of the colorized movie. To do that is necessary to manually change the script. This is a task quite complex and, in this guide, will be provided all the steps necessary to obtain the correct result.

After having provided the input clip in Hybrid is necessary first to activate the Retinex filter. Before using **Retinex** is necessary to remove all the black bars (if are present) as explained at the beginning of this guide. The Retinex filter is available in **Filtering->Vapoursynth->Color->Misc**, as shown in the following picture:

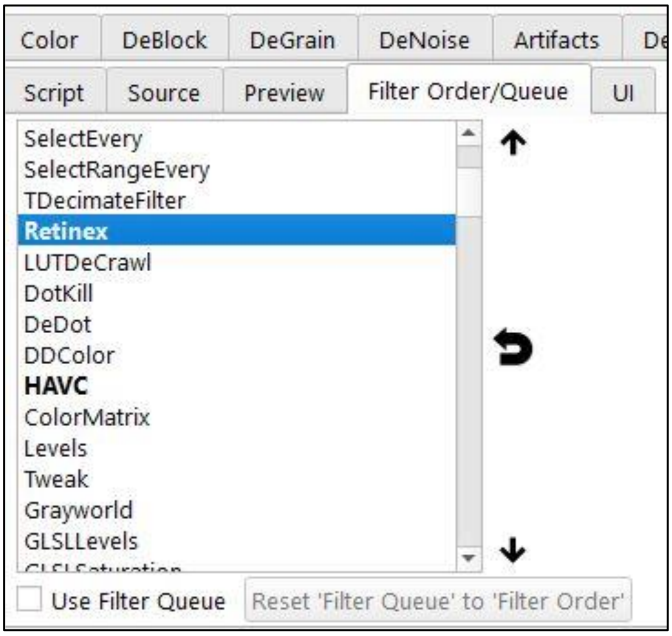


After having activated the filter is necessary to activate the HAVC filter as usual, but this time is necessary to disable the tweaks as shown in the picture below:



In this case tweaks are not more necessary because the Retinex filter is already changing significantly the luma, gamma and contrast of images.

Then is necessary to select the tab Filter Order/Queue and change the position of the Retinex filter using the up-down arrows as shown in the picture on the right. To be used as a pre-process filter the Retinex filter must be executed before DeOldify, and its order must be change so that it appears before the DeOldify filter.



But in this way the original luma, gamma and contrast of the original clip will not be restored. To restore the luminance of the images is necessary to manually change the script.

To do that it will be necessary to follow the steps already described in the chapter regarding Advanced coloring:

- click on preview button.
- select the menu Tools->vsViewer, this will open the tool vsViewer to show the preview script.

Having opened the script code of the preview file should be visible something like this:

```
File  Edit  Script  Help
24 # setting color transfer (vs.TRANSFER_BT709), if it is not set.
25 if validate.transferIsInvalid(clip):
26   clip = core.std.SetFrameProps(clip=clip, _Transfer=vs.TRANSFER_BT709)
27 # setting color primaries info (to vs.PRIMARIES_BT709), if it is not set.
28 if validate.primariesIsInvalid(clip):
29   clip = core.std.SetFrameProps(clip=clip, _Primaries=vs.PRIMARIES_BT709)
30 # setting color range to TV (limited) range.
31 clip = core.std.SetFrameProps(clip=clip, _ColorRange=vs.RANGE_LIMITED)
32 # making sure frame rate is set to 23.976fps
33 clip = core.std.AssumeFPS(clip=clip, fpsnum=24000, fpsden=1001)
34 # making sure the detected scan type is set (detected: progressive)
35 clip = core.std.SetFrameProps(clip=clip, _FieldBased=vs.FIELD_PROGRESSIVE) # progressive
36 # adjusting color space from YUV420P10 to YUV444P10 for vsRetinex
37 clip = core.resize.Bicubic(clip=clip, format=vs.YUV444P10, range_s="limited")
38 # color adjustment using Retinex
39 clip = core.retinex.MSRCP(input=clip, sigma=[25,80,250], lower_thr=0.001, upper_thr=0.001, fulls=True,
   fulld=True, chroma_protect=1.200)
40 # adjusting color space from YUV444P10 to RGB24 for vsHAVC
41 clip = core.resize.Bicubic(clip=clip, format=vs.RGB24, matrix_in_s="709", range_s="full",
   dither_type="error_diffusion")
42 # adding colors using HAVC
43 clip = havc.HAVC_main(clip=clip, Preset="fast", ColorModel="Video+Artistic", CombMethod="Simple",
   VideoTune="balanced", ColorFix="none", ColorTune="none", ColorMap="none", BlackWhiteTune="none",
   EnableDeepEx=False, enable_fp16=True)
44 # adjusting output color from: RGB24 to YUV420P10 for x265Model
45 clip = core.resize.Bicubic(clip=clip, format=vs.YUV420P10, matrix_s="709", range_s="full")
46 # set output frame rate to 23.976fps (progressive)
47 clip = core.std.AssumeFPS(clip=clip, fpsnum=24000, fpsden=1001)
48 # output
49 clip.set_output()
```

It is possible to see that at row 39 is called the Retinex filter, and at row 43 is called the HAVC functions HAVC_main.

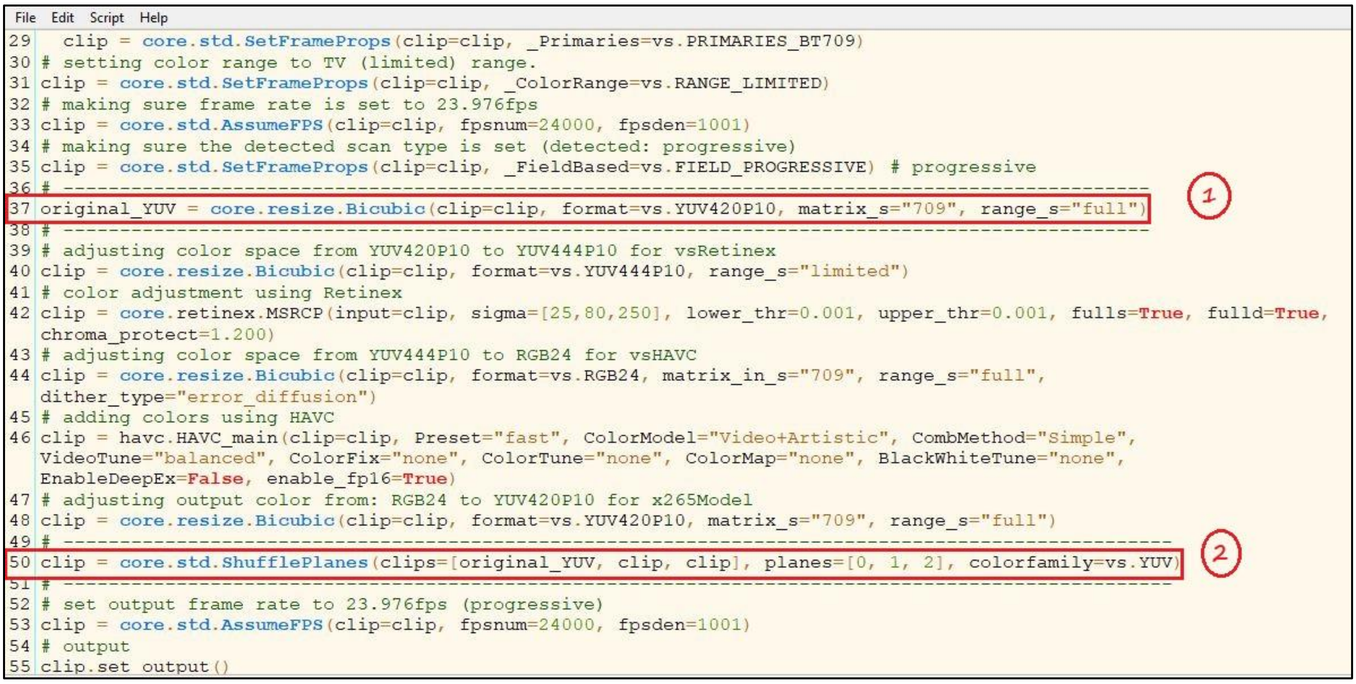
Now is necessary to manually change the script to add at row 37 (1), the following code to save the original clip:

```
original_YUV = core.resize.Bicubic(clip=clip, format=vs.YUV420P10, matrix_s="709",
range_s="full")
```

Then at row 50 (2) is necessary to add the code to restore the original luminosity of the clip:

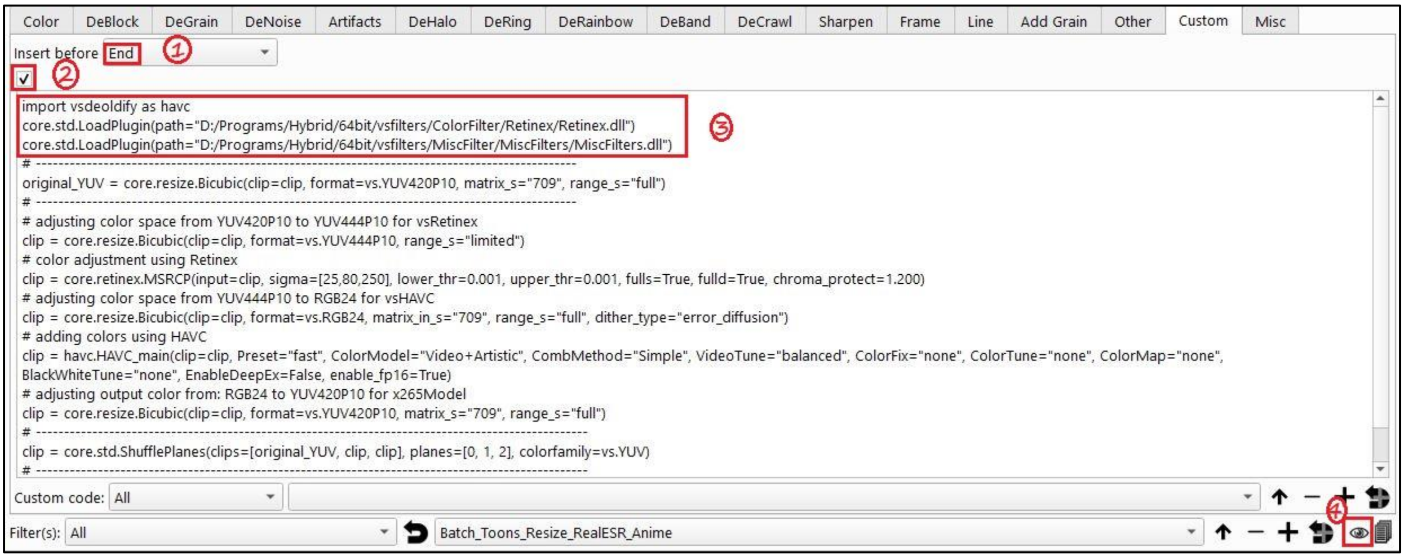
```
clip = core.std.ShufflePlanes(clips=[original_YUV, clip, clip], planes=[0, 1, 2],
colorfamily=vs.YUV)
```

The described changes are shown in the following picture:



To be able to use the modified script in Hybrid is necessary to uncheck all the filters previously activated so that Hybrid will contain only the code to preview the clip.

Then is necessary to go at panel: **Filtering->Vapoursynth->Custom** (see picture below)



and perform the following selections:

1) Select from the drop-down box: **End**
2) Check the box to allow the editing of code window
3) Paste the code (from row 35 to 55) previously changed (shown in the previous picture) and insert at the beginning, the code to load the necessary Plugins (path to be adjusted to Hybrid location):
```
import vsdeoldify as havc
core.std.LoadPlugin(path="D:/Programs/Hybrid/64bit/vsfilters/ColorFilter/Retinex/Retinex.dll")
core.std.LoadPlugin(path="D:/Programs/Hybrid/64bit/vsfilters/MiscFilter/MiscFilters/MiscFilters.dll")
```
4) Check by pressing the **preview button (4)** if the code has been properly inserted in Hybrid.

After having pressed the preview button, should be displayed the following Preview window:

VapoursynthScriptView

```
core.std.LoadPlugin(path="D:/Programs/Hybrid/64bit/vsfilters/SourceFilter/LSmashSource/LSMASHSource.dll")
# defining beforeEnd-function - START
def beforeEnd(clip):
 import vsdeoldify as havc
 core.std.LoadPlugin(path="D:/Programs/Hybrid/64bit/vsfilters/ColorFilter/Retinex/Retinex.dll")
 core.std.LoadPlugin(path="D:/Programs/Hybrid/64bit/vsfilters/MiscFilter/MiscFilters/MiscFilters.dll")
 # ------------------------------------------------------------------------------------------------
 original_YUV = core.resize.Bicubic(clip=clip, format=vs.YUV420P10, matrix_s="709", range_s="full")
 # ------------------------------------------------------------------------------------------------
 # adjusting color space from YUV420P10 to YUV444P10 for vsRetinex
 clip = core.resize.Bicubic(clip=clip, format=vs.YUV444P10, range_s="limited")
 # color adjustment using Retinex
 clip = core.retinex.MSRCP(input=clip, sigma=[25,80,250], lower_thr=0.001, upper_thr=0.001, fulls=True, fulld=True, chroma_protect=1.200)
 # adjusting color space from YUV444P10 to RGB24 for vsHAVC
 clip = core.resize.Bicubic(clip=clip, format=vs.RGB24, matrix_in_s="709", range_s="full", dither_type="error_diffusion")
 # adding colors using HAVC
 clip = havc.HAVC_main(clip=clip, Preset="fast", ColorModel="Video+Artistic", CombMethod="Simple", VideoTune="balanced", ColorFix="none", ColorTune="none", ColorMap="none",
BlackWhiteTune="none", EnableDeepEx=False, enable_fp16=True)
 # adjusting output color from: RGB24 to YUV420P10 for x265Model
 clip = core.resize.Bicubic(clip=clip, format=vs.YUV420P10, matrix_s="709", range_s="full")
 # ------------------------------------------------------------------------------------------------
 clip = core.std.ShufflePlanes(clips=[original_YUV, clip, clip], planes=[0, 1, 2], colorfamily=vs.YUV)
 # ------------------------------------------------------------------------------------------------
 return clip
# defining beforeEnd-function - END

# Import scripts
import validate
# Source: 'D:\PProjects\vs-deoldify_dev\clips\The.Invisible.Man\TEST1\The Invisible Man_TEST1.mkv'
# Current color space: YUV420P10, bit depth: 10, resolution: 1480x1080, frame rate: 23.976fps, scanorder: progressive, yuv luminance scale: limited, matrix: 709, format: HEVC
# Loading D:\PProjects\vs-deoldify_dev\clips\The.Invisible.Man\TEST1\The Invisible Man_TEST1.mkv using LWLibavSource
clip = core.lsmas.LWLibavSource(source="D:/PProjects/vs-deoldify_dev/clips/The.Invisible.Man/TEST1/The Invisible Man_TEST1.mkv", format="YUV420P10", stream_index=0, cache=0,
fpsnum=24000, fpsden=1001, prefer_hw=0)
frame = clip.get_frame(0)
# setting color matrix to 709.
clip = core.std.SetFrameProps(clip, _Matrix=vs.MATRIX_BT709)
# setting color transfer (vs.TRANSFER_BT709), if it is not set.
if validate.transferIsInvalid(clip):
  clip = core.std.SetFrameProps(clip=clip, _Transfer=vs.TRANSFER_BT709)
# setting color primaries info (to vs.PRIMARIES_BT709), if it is not set.
if validate.primariesIsInvalid(clip):
  clip = core.std.SetFrameProps(clip=clip, _Primaries=vs.PRIMARIES_BT709)
# setting color range to TV (limited) range.
clip = core.std.SetFrameProps(clip=clip, _ColorRange=vs.RANGE_LIMITED)
# making sure frame rate is set to 23.976fps
clip = core.std.AssumeFPS(clip=clip, fpsnum=24000, fpsden=1001)
# making sure the detected scan type is set (detected: progressive)
clip = core.std.SetFrameProps(clip=clip, _FieldBased=vs.FIELD_PROGRESSIVE)# progressive
clip = beforeEnd(clip)
# set output frame rate to 23.976fps (progressive)
clip = core.std.AssumeFPS(clip=clip, fpsnum=24000, fpsden=1001)
# output
clip.set_output()
```

In the box 1 it is possible to see the function **beforeEnd** automatically created by Hybrid with the coded previously inserted, while in the box 2 is possible to see the call to the function **beforeEnd** so we are certain that the code is properly inserted and called. Using this custom code will be possible to use Hybrid to encode the clip as explained at the beginning of this guide.

As stated previously not always Retinex is able to improve the color quality of final image, and it is better to apply it to a small sample of the video to be colorized to see if the use of Retinex is introducing artifacts instead of improve the color quality.

In the following pictures are shown some cases where Retinex is introducing artifact in the colored frames. Only the model DeOldify (with Video network) was able to provide a satisfactory result with Retinex.


!Original.png
DDcolor_Artistic.png
DDcolor_ModelScope.png
Deoldify_Video.png
Zhang_ECCV16.png
Zhang_Siggraph17.png


!Original.png
DDcolor_Artistic.png
DDcolor_ModelScope.png
Deoldify_Video.png
Zhang_ECCV16.png
Zhang_Siggraph17.png

6.0 HAVC Functions reference

In this chapter will be described the most useful functions available in the filter HAVC.

6.1 HAVC_main

This is the main HAVC function, with the support of Presets, it is a wrapper to the more specialized HAVC and it represents the easier way to use the filter. The header of the function is the following:

HAVC_main(clip: vs.VideoNode, Preset: str = 'medium', ColorModel: str = 'Video+Artistic',

CombMethod: str = 'Simple', VideoTune: str = 'Stable', ColorFix: str = 'Violet/Red', ColorTune: str = 'Light',

ColorMap: str = 'None', BlackWhiteTune: str = 'None', EnableDeepEx: bool = False, DeepExMethod: int = 0,

DeepExPreset: str = 'Medium', DeepExRefMerge: int = 0, DeepExOnlyRefFrames: bool = False,

ScFrameDir: str = **None**, ScThreshold: float = 0.10, ScThtOffset: int = 1, ScMinFreq: int = 0,

ScMinInt: int = 1, ScThtSSIM: float = 0.0, ScNormalize: bool = False, DeepExModel: int = 0,

DeepExVivid: bool = False, DeepExEncMode: int = 0, DeepExMaxMemFrames=0,

RefRange: tuple[int, int] = (0, 0), enable_fp16: bool = True, sc_debug: bool = False) -> vs.VideoNode:

Where:

clip: clip to process, only RGB24 format is supported.

Preset: Preset to control the encoding speed/quality. Allowed values are: 'Placebo', 'VerySlow', 'Slower', 'Slow', 'Medium' (default), 'Fast', 'Faster', 'VeryFast'.

ColorModel: Preset to control the Color Models to be used for the color inference. Allowed values are: 'Video+Artistic' (default), 'Stable+Artistic', 'Video+ModelScope', 'Stable+ModelScope', 'Video+ECCV16', 'Artistic+Modelscope', 'Video+Siggraph17', 'DeOldify(Video)', 'DeOldify(Stable)', 'DeOldify(Artistic)', 'DDColor(Artistic)', 'DDColor(ModelScope)', 'Zhang(Siggraph17)', 'Zhang(ECCV16)'.

CombMethod: Method used to combine coloring models with (+): Allowed values are: 'Simple' (default), 'Constrained-Chroma', 'Luma-Masked', 'Adaptive-Luma'.

VideoTune: Preset to control the output video color stability. Allowed values are: 'DeOldify', 'VeryStable', 'MoreStable', 'Stable', 'Balanced', 'Vivid', 'MoreVivid', 'VeryVivid', 'DDColor'.

ColorFix: This parameter allows to reduce color noise on specific chroma ranges. Allowed values are: 'None', 'Magenta', 'Magenta/Violet', 'Violet', 'Violet/Red' (default), 'Blue/Magenta', 'Yellow', 'Yellow/Orange', 'Yellow/Green'.

ColorTune: This parameter allows to define the intensity of noise reduction applied by ColorFix. Allowed values are: 'None', 'Light' (default), 'Medium', 'Strong'.

ColorMap: This parameter allows to change a given color range to another color. Allowed values are: 'None' (default), 'Blue->Brown', 'Blue->Red', 'Blue->Green', 'Green->Brown', 'Green->Red', 'Green->Blue', 'Red->Brown', 'Red->Blue', 'Yellow->Rose'.

BlackWhiteTune: This parameter allows to improve contrast and luminosity of Black & White input clip to be colored with HAVC. Allowed values are: 'None' (default), 'Light', 'Medium', 'Strong'.

EnableDeepEx: Enable coloring using Exemplar-based Video Colorization models.

DeepExMethod: Method to use to generate reference frames. Allowed values are: 0 = HAVC same as video (default), 1 = HAVC + RF same as video, 2 = HAVC + RF different from video, 3 = external RF same as video, 4 = external RF different from video, 5 = external ClipRef same as video, 6 = external ClipRef different from video.

DeepExPreset: Preset to control the render method and speed. Allowed values are: 'Fast' (colors are more washed out), 'Medium' (colors are a little washed out), 'Slow' (colors are a little more vivid).

DeepExRefMerge: Method used by DeepEx to merge the reference frames with the frames propagated by DeepEx. It is applicable only with DeepEx method: 0, 1, 2. Allowed values are: 0 = No RF merge (reference frames can be produced with any frequency), 1 = RF-Merge VeryLow (reference frames are merged with weight=0.3), 2 = RF-Merge Low (reference frames are merged with weight=0.4), 3 = RF-Merge Med. (reference frames are merged with medium weight=0.5), 4 = RF-Merge High (reference frames are merged with weight=0.6), 5 = RF-Merge VeryHigh (reference frames are merged with weight=0.7).

DeepExOnlyRefFrames: If enabled the filter will output in ScFrameDir the reference frames. Useful to check and eventually correct the frames with wrong colors (can be used only if DeepExMethod = 0).

DeepExModel: Exemplar Model used by DeepEx to propagate color frames. Allowed values are: 0: ColorMNet (default), 1: Deep-Exemplar, 2 : Deep-Remaster.

DeepExVivid: Depending on selected DeepExModel, if enabled (True): (0) ColorMNet: the frames memory is reset at every reference frame update, (1) Deep-Exemplar: the saturation will be increased by about 25%. (2) Deep-Remaster: the saturation will be increased by about 20% and Hue by +10. Range [True, False].

DeepExEncMode: Parameter used by ColorMNet to define the encode mode strategy. Available values are:

0: <u>remote encoding</u>. The frames will be colored by a thread outside Vapoursynth.

This option doesn't have any GPU memory limitation and will allow to fully use the long-term frame memory. It is the faster encode method (default)

1: <u>local encoding</u>. The frames will be colored inside the Vapoursynth environment.

In this case the max_memory will be limited by the size of GPU memory (max 15 frames for 24GB GPU). Useful for coloring clips with a lot of smooth transitions, since in this case is better to use a short frame memory or the Deep-Exemplar model, which is faster.

2: <u>remote all-ref</u>. Same as "remote encoding" but all the available reference frames will be used for the inference at the beginning of encoding.

DeepExMaxMemFrames: Parameter used by ColorMNet/DeepRemaster models.

For ColorMNet specify the max number of encoded frames to keep in memory. Its value depends on encode mode and must be defined manually following the suggested values.

DeepExEncMode =0: there is no memory limit (it could be all the frames in the clip).

Suggested values are: min=150, max=10000

If = 0 will be filled with the value of 10000 or the clip length if lower.

DeepExEncMode =1: the max memory frames are limited by available GPU memory.

Suggested values are:

min=1, max=4: for 8GB GPU

min=1, max=8: for 12GB GPU

min=1, max=15: for 24GB GPU

If = 0 will be filled with the max value (depending on total GPU RAM available).

For DeepRemaster represent the number to reference frames to keep in memory.

Suggested values are:

min=4, max=50

If = 0 will be filled with the value of 20.

ScFrameDir: if set, define the directory where are stored the reference frames that will be used by Exemplar-based Video Colorization models. With DeepExMethod 5,6 this parameter can be the path to a video clip.

ScThreshold: Scene changes threshold used to generate the reference frames to be used by Exemplar-based Video Colorization. It is a percentage of the luma change between the previous and the current frame. Range [0-1], default 0.10. If =0 the reference frames are not generated.

ScThtOffset: Offset index used for the Scene change detection. The comparison will be performed, between frame[n] and frame[n-offset]. An offset > 1 is useful to detect blended scene change. Range [1, 25]. Default = 1.

ScMinInt: Minimum number of frame interval between scene changes, Range [1, 25]. Default = 1.

ScMinFreq: if > 0 will be generated at least a reference frame every ScMinFreq frames. Range [0-1500], default: 0.

ScThtSSIM: Threshold used by the SSIM (Structural Similarity Index Metric) selection filter. If > 0, will be activated a filter that will improve the scene-change detection, by discarding images that are similar. Suggested values are between 0.35 and 0.75. Range [0-1], default 0.0 (deactivated).

ScNormalize: If true the B&W frames are normalized before scene detection. The normalization will increase the sensitivity to smooth scene changes. Range [True, False], default: True.

RefRange: Parameter used only with DeepExMethod in (5, 6). With this parameter it is possible to provide the frame number of clip start and end. For example RefRange = (100, 500) will return the clip's slice: clip[100:500], if RefRange=(0, 0) will be considered all clip's frames.

enable_fp16: Enable/disable FP16 in DDcolor inference. Range [True, False], default: True.

sc_debug: Print debug messages regarding the scene change detection process.

6.2 HAVC_deepex

This is the HAVC function that perform the color inference using the exemplar-based models: ColorMNet, Deep-Exemplar. Some of parameters in input are accepting lists in order to minimize the number of parameters managed by Hybrid. The header of the function is the following:

```
HAVC_deepex(clip: vs.VideoNode = None, clip_ref: vs.VideoNode = None, method: int = 0,
        render_speed: str = 'medium', render_vivid: bool = True, ref_merge: int = 0, sc_framedir: str = None,
        ref_norm: bool = False, only_ref_frames: bool = False, dark: bool = False, dark_p: list = (0.2, 0.8),
        smooth: bool = False, smooth_p: list = (0.3, 0.7, 0.9, 0.0, "none"), colormap: str = "none",
        ref_weight: float = None, ref_thresh: float = None, ref_freq: int = None, ex_model: int = 0,
        encode_mode: int = 0, max_memory_frames: int = 0, torch_dir: str = model_dir) -> vs.VideoNode:
```

Where:

clip: Clip to process. Only RGB24 format is supported

clip_ref: Clip containing the reference frames, it is necessary if method in (0,1,2,5,6).

method: Method to use to generate reference frames (RF). Allowed values are: 0 = HAVC same as video (default), 1 = HAVC + RF same as video, 2 = HAVC + RF different from video, 3 = external RF same as video, 4 = external RF different from video, 5 = external ClipRef same as video, 6 = external ClipRef different from video.

render_speed: Preset to control the render method and speed. Allowed values are: 'Fast' (colors are more washed out), 'Medium' (colors are a little washed out), 'Slow' (colors are a little more vivid).

render_vivid Depending on selected ex_model, if enabled (True): (0) ColorMNet: the frames memory is reset at every reference frame update, (1) Deep-Exemplar: the saturation will be increased by about 25%. (2) Deep-Remaster: the saturation will be increased by about 20% and Hue by +10. Range [True, False].

ref_merge Method used by DeepEx to merge the reference frames with the frames propagated by DeepEx.
It is applicable only with DeepEx method: 0, 1, 2. Allowed values are: 0 = No RF merge (reference frames can be produced with any frequency), 1 = RF-Merge VeryLow (reference frames are merged with weight=0.3), 2 = RF-Merge Low (reference frames are merged with weight=0.4), 3 = RF-Merge Med. (reference frames are merged with medium weight=0.5), 4 = RF-Merge High (reference frames are merged with weight=0.6), 5 = RF-Merge VeryHigh (reference frames are merged with weight=0.7).

ref_weight: If (ref_merge > 0), represent the weight used to merge the reference frames. If is not set, is assigned automatically a value depending on ref_merge value.

ref_thresh: Represent the threshold used to create the reference frames. If is not set, is assigned automatically a value of 0.10.

ref_freq: If > 0 will be generated at least a reference frame every "ref_freq" frames. range [0-1500]. If is not set, is assigned automatically a value depending on ref_merge/method values.

ref_norm: If true the B&W frames are normalized before apply the Scene Detection to generate the reference frames. The normalization will increase the sensitivity to smooth scene changes, range [True, False], default: False

sc_framedir: If set, define the directory where are stored the reference frames. If only_ref_frames=True, and method=0 this directory will be written with the reference frames used by the filter. If method!=0 the directory will be read to create the reference frames that will be used by Exemplar-based Video Colorization. The reference frame name must be in the format: ref_nnnnnn.[jpg | png], for example the reference frame 897 must be named: ref_000897.jpg or ref_000897.png. With method 5,6 this parameter can be the path to a video clip.

only_ref_frames: If enabled the filter will output in sc_framedir the reference frames. Useful to check and eventually correct the frames with wrong colors.

dark: Enable/disable darkness filter (only on ref-frames). Range [True, False]

dark_p: List of parameters for darken the clip's dark portions, which sometimes are wrongly colored by the color models, the positional parameters in the list are the following:

[0]: dark_threshold, luma threshold to select the dark area, range [0.1-0.5] (0.01=1%)

[1]: dark_amount: amount of desaturation to apply to the dark area, range [0-1]

[2]: chroma_range (optional), if="none" is disabled, otherwise the filter desaturation will be applied only on the region defined in the chroma range with the requested syntax.

smooth: Enable/disable chroma smoothing (only on ref-frames). Range [True, False]

smooth_p: List of parameters to adjust the saturation and "vibrancy" of the clip, the positional parameters in the list are the following:

[0]: dark_threshold, luma threshold to select the dark area. Range [0-1] (0.01=1%)

[1]: white_threshold, if > dark_threshold will be applied a gradient till white_threshold, range [0-1] (0.01=1%)

[2]: dark_sat, amount of de-saturation to apply to the dark area. Range [0-1]

[3]: dark_bright, darkness parameter it used to reduce the "V" component in "HSV" color-space. Range [0, 1]

[4]: chroma_range (optional), if="none" is disabled, otherwise the filter desaturation will be applied only on the region defined in the chroma range with the requested syntax.

colormap: Direct hue/color mapping (only on ref-frames), without luma filtering, using the "chroma adjustment" parameter, if="none" is disabled.

ex_model: Exemplar-based model to use for the color propagation of reference images, available models are: 0 = ColorMNet (default), 1 = Deep-Exemplar, 2 : Deep-Remaster.

encode_mode: Parameter used by ColorMNet to define the encode mode strategy. Available values are:

0: remote encoding. The frames will be colored by a thread outside Vapoursynth. This option doesn't have any GPU memory limitation and will allow to fully use the long-term frame memory. It is the faster encode method (default)

1: local encoding. The frames will be colored inside the Vapoursynth environment. In this case the max_memory will be limited by the size of GPU memory (max 15 frames for 24GB GPU). Useful for coloring clips with a lot of smooth transitions, since in this case is better to use a short frame memory or the Deep-Exemplar model, which is faster.

2: remote all-ref. Same as "remote encoding" but all the available reference frames will be used for the inference at the beginning of encoding.

max_memory_frames: Parameter used by ColorMNet/DeepRemaster models.

For ColorMNet specify the max number of encoded frames to keep in memory. Its value depends on encode mode and must be defined manually following the suggested values.

encode_mode =0: there is no memory limit (it could be all the frames in the clip).

Suggested values are: min=150, max=10000

If = 0 will be filled with the value of 10000 or the clip length if lower.

encode_mode =1: the max memory frames are limited by available GPU memory.

Suggested values are:

- min=1, max=4: for 8GB GPU
- min=1, max=8: for 12GB GPU
- min=1, max=15: for 24GB GPU

If = 0 will be filled with the max value (depending on total GPU RAM available).

For DeepRemaster represent the number to reference frames to keep in memory.

Suggested values are:

- min=4, max=50

If = 0 will be filled with the value of 20.

torch_dir: torch hub directory location, default is model directory, if set to None will switch to torch cache dir.

6.3 HAVC_colorizer

This is the HAVC function that perform the color inference using the single frame-based models: DeOldify, DDColor and Zhang models. Some of parameters in input are accepting lists in order to minimize the number of parameters managed by Hybrid. The header of the function is the following:

HAVC_colorizer(clip: vs.VideoNode, method: int = 2, mweight: float = 0.4, deoldify_p: list = (0, 24, 1.0, 0.0), ddcolor_p: list = (1, 24, 1.0, 0.0, True), ddtweak: bool = False, ddtweak_p: list = (0.0, 1.0, 2.5, True, 0.3, 0.6, 1.5, 0.5, "300:360|0.8,0.1"), cmc_tresh: float = 0.2, lmm_p: list = (0.2, 0.8, 1.0), alm_p: list = (0.8, 1.0, 0.15), cmb_sw: bool = False, sc_threshold: float = 0.0, sc_tht_offset: int = 1, sc_min_freq: int = 0, sc_tht_ssim: float = 0.0, sc_normalize: bool = True, sc_min_int: int = 1, sc_tht_white: float = DEF_THT_WHITE, sc_tht_black: float = DEF_THT_BLACK, device_index: int = 0, torch_dir: str = model_dir, sc_debug: bool = False) -> vs.VideoNode:

Where:

clip: clip to process, only RGB24 format is supported

method: method used to combine DeOldify with DDColor (default = 2):

0: **DeOldify only** (no merge)

1: **DDcolor only** (no merge)

2: **Simple Merge** (default):

the frames are combined using a weighted merge, where the parameter mweight represent the weight assigned to the colors provided by the DDColor frames. With this method is suggested a starting weight < 50% (ex. = 40%).

3: **Constrained Chroma Merge**:

given that the colors provided by DeOldify are more conservative and stable than the colors obtained with DDcolor. The frames are combined by assigning a limit to the amount of difference in chroma values between DeOldify and DDcolor this limit is defined by the threshold parameter cmc_tresh. The limit is applied to the image converted to "YUV". For example, when cmc_tresh=0.2, the chroma values "U","V" of DDcolor frame will be constrained to have an absolute percentage difference respect to "U","V" provided by DeOldify not higher than 20%. The final limited frame will be merged again with the DeOldify frame. With this method is suggested a starting weight > 50% (ex. = 60%).

4: **Luma Masked Merge**:

the frames are combined using a masked merge, the pixels of DDcolor with luma < luma_mask_limit will be filled with the pixels of DeOldify. If luma_white_limit > luma_mask_limit the mask will apply a gradient till luma_white_limit. If the parameter mweight > 0 the final masked frame will be merged again, with the DeOldify frame. With this method is suggested a starting weight > 60% (ex. = 70%).

5: **Adaptive Luma Merge**:

given that the DDcolor performance is quite bad on dark scenes, the images are combined by decreasing the weight assigned to DDcolor when the luma is below a given threshold given by: luma_threshold. The weight is calculated using the formula:

merge_weight = MAX(mweight * (luma/luma_threshold)^alpha, min_weight).

For example, with: luma_threshold = 0.6 and alpha = 1, the weight assigned to DDColor will start to decrease linearly when the luma < 60% till min_weight. For alpha=2, begins to decrease quadratically (because luma/luma_threshold < 1). With this method is suggested a starting weight > 70% (ex. = 80%).

The methods 3 and 4 are similar to **Simple Merge**, but before the merge with DeOldify the DDcolor frame is limited in the chroma changes (method 3) or limited based on the luma (method 4).

The method 5 is a **Simple Merge** where the weight decrease with luma.

mweight: weight given to DDcolor clip in all merge methods, range [0-1] (0.01=1%), the final frame is obtained performing the following weighted sum: f_out = f_deoldify*(1-mweight) + mweight*f_ddcolor

deoldify_p: List of parameters for the DeOldify color inference:

[0] DeOldify-model to use (default = 0):

0 = ColorizeVideo_gen

1 = ColorizeStable_gen

2 = ColorizeArtistic_gen

[1] render-factor for the model. Range: 10-44 (default = 24).

[2] saturation parameter to apply to DeOldify color model (default = 1)

[3] hue parameter to apply to DeOldify color model (default = 0)

ddcolor_p: List of parameters for DDcolor inference:

[0] DDColor-model to use (default = 1):

0 = ddcolor_modelscope,

1 = ddcolor_artistic

2 = colorization_siggraph17

3 = colorization_eccv16

[1] render-factor for the model, if=0 will be auto selected (default = 24). Range: [0, 10-64]

[2] saturation parameter to apply to DDcolor model (default = 1)

[3] hue parameter to apply to DDcolor model (default = 0)

[4] FP16: enable/disable FP16 in DDcolor inference

ddtweak: enabled/disable tweak parameters for DDcolor. Range [True, False]

ddtweak_p: List of DDcolor tweak parameters:

[0]: bright (default = 0)

[1]: contrast (default = 1), if < 1 DDcolor provides de-saturated frames

[2]: gamma tweak for DDcolor (default = 1)

[3]: luma_constrained_gamma: luma constrained gamma correction enabled (default = False).

Range: [True, False]. When enabled the average luma of a video clip will be forced to don't be below the value defined by the parameter luma_min. The function allows to modify the gamma (g) of the clip if the average luma is below the parameter gamma_luma_min.

A gamma (g) value > 2.0 improves the DDcolor stability on bright scenes, while a gamma (g) < 1 improves the DDcolor stability on dark scenes.

The decrease of the gamma with luma is activated using a gamma_alpha!= 0.

[4]: luma_min: luma (%) min value for tweak activation (default = 0.2), if=0 is not activated, range [0-1]

[5]: gamma_luma_min: luma (%) min value for gamma tweak activation (default = 0.5), if=0 is not Activated. Range [0-1]

[6]: gamma_alpha: the gamma (g) will decrease with the luma using the following expression:

g = MAX(gamma * pow(luma/gamma_luma_min, gamma_alpha), gamma_min),

for a movie with a lot of dark scenes is suggested alpha > 1, if=0 is not activated. Range [>=0]

[7]: gamma_min: minimum value for gamma. Range (default=0.5) [>0.1]

[8]: chroma_adjustment (optional), if="none" is disabled, otherwise will be applied the specified chroma adjustment defined with the requested syntax.

cmc_tresh: chroma threshold (%), used by **Constrained Chroma Merge** (see method=3 for a full explanation). Range [0-1] (0.01=1%)

lmm_p: List of parameters for method: **Luma Masked Merge** (see method=4 for a full explanation)

[0]: luma_mask_limit: luma limit for build the mask used in Luma Masked Merge. Range [0-1] (0.01=1%)

[1]: luma_white_limit: the mask will apply a gradient till luma_white_limit. Range [0-1] (0.01=1%)

[2]: luma_mask_sat: if < 1 the DDcolor dark pixels will be substituted with the desaturated DeOldify Pixels. Range [0-1] (0.01=1%)

alm_p: List of parameters for method: **Adaptive Luma Merge** (see method=5 for a full explanation)

[0]: luma_threshold: threshold for the gradient merge, range [0-1] (0.01=1%)

[1]: alpha: exponent parameter used for the weight calculation. Range [>0]

[2]: min_weight: min merge weight. Range [0-1] (0.01=1%)

cmb_sw: if true switch the clip order in all the combining methods. Range [True, False]

sc_threshold: Scene changes threshold used to generate the reference frames to be used by Exemplar-based Video Colorization. It is a percentage of the luma change between the previous and the current frame.

Range [0-1], default 0.0. If =0 the reference frames are not generated and will be colorized all the frames.

sc_tht_offset: Offset index used for the Scene change detection. The comparison will be performed, between frame[n] and frame[n-offset]. An offset > 1 is useful to detect blended scene change. Range [1, 25]. Default = 1.

sc_tht_ssim: Threshold used by the SSIM (Structural Similarity Index Metric) selection filter. If > 0, will be activated a filter that will improve the scene-change detection, by discarding images that are similar. Suggested values are between 0.35 and 0.85. Range [0-1], default 0.0 (deactivated).

sc_normalize: If true the B&W frames are normalized before scene detection. The normalization will increase the sensitivity to smooth scene changes. Range [True, False], default: True.

sc_min_int: Minimum number of frame interval between scene changes. Range [1, 25]. Default = 1.

sc_min_freq: If > 0 will be generate at least a reference frame every sc_min_freq frames. Range [0-1500], default: 0.

sc_tht_white: Threshold to identify white frames. Range [0-1], default 0.88.

sc_tht_black: Threshold to identify dark frames. Range [0-1], default 0.12.

device_index: device ordinal of the GPU, choices: GPU0...GPU7, CPU=99 (default = 0)

torch_dir: torch hub directory location, default is model directory, if set to None will switch to torch cache dir.

sc_debug: Print debug messages regarding the scene change detection process.

6.4 HAVC_stabilizer

This is the HAVC function that allows to apply to the input clip the color stabilization filters, which can be applied to stabilize the chroma components in colored clips. Some of parameters in input are accepting lists in order to minimize the number of parameters managed by Hybrid. The header of the function is the following:

HAVC_stabilizer(clip: vs.VideoNode, dark: bool = False, dark_p: list = (0.2, 0.8), smooth: bool = False,

smooth_p: list = (0.3, 0.7, 0.9, 0.0, "none"), stab: bool = False, stab_p: list = (5, 'A', 1, 15, 0.2, 0.8),

colormap: str = "none", render_factor: int = 24) -> vs.VideoNode:

Where:

clip: clip to process, only RGB24 format is supported.

dark: enable/disable darkness filter. Range [True, False]

dark_p: List of parameters for darken the clip's dark portions, which sometimes are wrongly colored by the color models:

[0]: dark_threshold, luma threshold to select the dark area. Range [0.1-0.5] (0.01=1%), default = 0.2

[1]: dark_amount: amount of desaturation to apply to the dark area. Range [0-1], where a value of 0 will not apply any desaturation, default = 0.8

[2]: chroma_range (optional), if="none" is disabled, otherwise the filter desaturation will be applied only on the region defined in the chroma range with the requested syntax.

smooth: enable/disable chroma smoothing. Range [True, False]

smooth_p: List of parameters to adjust the saturation and "vibrancy" of the clip.

[0]: dark_threshold, luma threshold to select the dark area, range [0-1] (0.01=1%)

[1]: white_threshold, if > dark_threshold will be applied a gradient till white_threshold, range [0-1] (0.01=1%)

[2]: dark_sat, amount of de-saturation to apply to the dark area. Range [0-1]

[3]: dark_bright, darkness parameter it used to reduce the "V" component in "HSV" color-space. Range [0, 1]

[4]: chroma_range (optional), if="none" is disabled, otherwise the filter desaturation will be applied only on The region defined in the chroma range with the requested syntax.

stab: enable/disable chroma stabilizer. Range [True, False]

stab_p: List of parameters for the temporal color stabilizer:

[0]: nframes, number of frames to be used in the stabilizer. Range [3-15]

[1]: mode, type of average used by the stabilizer. Range ['A'='arithmetic', 'W'='weighted']

[2]: sat: saturation applied to the restored gray pixels. Range [0,1]

[3]: tht, threshold to detect gray pixels. Range [0,255], if=0 is not applied the restore.

its value depends on merge method used; suggested values are:

method 0: tht = 5

method 1: tht = 60 (DDcolor provides very saturated frames)

method 2: tht = 15

method 3: tht = 20

method 4: tht = 5

method 5: tht = 10

[4]: weight, weight to blend the restored image (default=0.2), range [0-1], if=0 is not applied the blending

[5]: tht_scen, threshold for scene change detection (default = 0.8), if=0 is not activated, range [0.01-0.50]

[6]: chroma_adjustment (optional), if="none" is disabled, otherwise will be applied the specified chroma adjustment defined with the requested syntax.

colormap: direct hue/color mapping, without luma filtering, using the color mapping syntax, if="none" is disabled.

render_factor: render_factor to apply to the filters, the frame size will be reduced to speed-up the filters, but the final resolution will be the one of the original clip. If = 0 will be auto selected. This approach takes advantage of the fact that human eyes are much less sensitive to imperfections in chrominance compared to luminance. This means that it is possible to speed-up the chroma filters and ultimately get a great high-resolution result. Range: [0, 10-64]

6.5 HAVC_read_video

This is a utility HAVC function used to read a movie to be used as input for the function HAVC_restore_video, The clip provided in output will be already in RGB24 format. The header of the function is the following:

HAVC_read_video(source: str, fpsnum: int = 0, fpsden: int = 1) -> vs.VideoNode:

Where:

source: Full path to the video to read

fpsnum: FPS numerator, for using it in HAVC, must be provided the same value of clip to be colored: **clip.fps_num**

fpsden: FPS denominator, for using it in HAVC, must be provided the same value of clip to be colored: **clip.fps_den**

6.6 HAVC_restore_video

This is the HAVC function used to colorize a movie using a video clip as source of reference images. Usually, the reference clip to provide in input is obtained by a previous call to the function HAVC_read_video. The header of the function is the following:

HAVC_restore_video(clip: vs.VideoNode = **None**, clip_ref: vs.VideoNode = **None**, method: int = 6,

render_speed: str = 'medium', ex_model: int = 0, ref_merge: int = 0, ref_weight: float = **None**,

ref_thresh: float = **None**, ref_freq: int = **None**, ref_norm: bool = False, max_memory_frames: int = 0,

render_vivid: bool = False, encode_mode: int = 2, torch_dir: str = model_dir) -> vs.VideoNode:

Where:

clip: Clip to process. Only RGB24 format is supported

clip_ref: Clip containing the reference frames, it is necessary if method in (0,1,2,5,6).

method: Method to use to generate reference frames (RF). Allowed values are: 0 = HAVC same as video (default), 1 = HAVC + RF same as video, 2 = HAVC + RF different from video, 3 = external RF same as video, 4 = external RF different from video, 5 = external ClipRef same as video, 6 = external ClipRef different from video.

render_speed: Preset to control the render method and speed. Allowed values are: 'Fast' (colors are more washed out), 'Medium' (colors are a little washed out), 'Slow' (colors are a little more vivid).

render_vivid Depending on selected ex_model, if enabled (True): (0) ColorMNet: the frames memory is reset at every reference frame update, (1) Deep-Exemplar: the saturation will be increased by about 25%. (2) Deep-Remaster: the saturation will be increased by about 20% and Hue by +10. Range [True, False].

ref_merge Method used by DeepEx to merge the reference frames with the frames propagated by DeepEx. It is applicable only with DeepEx method: 0, 1, 2. Allowed values are: 0 = No RF merge (reference frames can be produced with any frequency), 1 = RF-Merge VeryLow (reference frames are merged with weight=0.3), 2 = RF-Merge Low (reference frames are merged with weight=0.4), 3 = RF-Merge Med. (reference frames are merged with medium weight=0.5), 4 = RF-Merge High (reference frames are merged with weight=0.6), 5 = RF-Merge VeryHigh (reference frames are merged with weight=0.7).

ref_weight: If (ref_merge > 0), represent the weight used to merge the reference frames. If is not set, is assigned automatically a value depending on ref_merge value.

ref_thresh: Represent the threshold used to create the reference frames. If is not set, is assigned automatically a value of 0.10.

ref_freq: If > 0 will be generated at least a reference frame every "ref_freq" frames. range [0-1500]. If is not set, is assigned automatically a value depending on ref_merge/method values.

ref_norm: If true the B&W frames are normalized before apply the Scene Detection to generate the reference frames. The normalization will increase the sensitivity to smooth scene changes, range [True, False], default: False

ex_model: Exemplar-based model to use for the color propagation of reference images, available models are: 0 = ColorMNet (default), 1 = Deep-Exemplar, 2 = Deep-Remaster.

encode_mode: Parameter used by ColorMNet to define the encode mode strategy. Available values are:

0: remote encoding. The frames will be colored by a thread outside Vapoursynth.

This option doesn't have any GPU memory limitation and will allow to fully use the long-term frame memory. It is the faster encode method (default)

1: local encoding. The frames will be colored inside the Vapoursynth environment.

In this case the max_memory will be limited by the size of GPU memory (max 15 frames for 24GB GPU). Useful for coloring clips with a lot of smooth transitions, since in this case is better to use a short frame memory or the Deep-Exemplar model, which is faster.

2: remote all-ref. Same as "remote encoding" but all the available reference frames will be used for the inference at the beginning of encoding.

max_memory_frames: Parameter used by ColorMNet/DeepRemaster models.

For ColorMNet specify the max number of encoded frames to keep in memory. Its value depends on encode mode and must be defined manually following the suggested values.

encode_mode =0: there is no memory limit (it could be all the frames in the clip).

Suggested values are: min=150, max=10000

If = 0 will be filled with the value of 10000 or the clip length if lower.

encode_mode =1: the max memory frames are limited by available GPU memory.

Suggested values are:

min=1, max=4: for 8GB GPU

min=1, max=8: for 12GB GPU

min=1, max=15: for 24GB GPU

If = 0 will be filled with the max value (depending on total GPU RAM available).

For DeepRemaster represent the number to reference frames to keep in memory.

Suggested values are:

min=4, max=50

If = 0 will be filled with the value of 20.

torch_dir: torch hub directory location, default is model directory, if set to None will switch to torch cache dir.

6.7 HAVC_SceneDetect

This is the HAVC function to set the scene-change frames in the clip. When is detected a scene change, the frame property '_SceneChangePrev' is set = 1 and '_SceneChangeNext' is set = 0. The header of the function is the following:

HAVC_SceneDetect(clip: vs.VideoNode, sc_threshold: float = DEF_THRESHOLD, sc_tht_offset: int = 1,

sc_tht_ssim: float = 0.0, sc_min_int: int = 1, sc_min_freq: int = 0, sc_normalize: bool = True,

sc_tht_white: float = DEF_THT_WHITE, sc_tht_black: float = DEF_THT_BLACK,

sc_debug: bool = False) -> vs.VideoNode:

Where:

clip: clip to process, only RGB24 format is supported.

sc_threshold: Scene changes threshold used to generate the reference frames.

It is a percentage of the luma change between the previous n-frame (n=sc_the_offset)

and the current frame. range [0-1], default 0.10.

sc_tht_offset: Offset index used for the Scene change detection. The comparison will be performed,

between frame[n] and frame[n-sc_tht_offset]. An sc_tht_offset > 1 is useful to detect blended scene

change. Range [1, 25], default = 1.

sc_normalize: If true the B&W frames are normalized before apply scene detection filter, the normalization will

increase the sensitivity to smooth scene changes.

sc_tht_white: Threshold to identify white frames, range [0-1], default 0.88.

sc_tht_black: Threshold to identify dark frames, range [0-1], default 0.12.

sc_tht_ssim: Threshold used by the SSIM (Structural Similarity Index Metric) selection filter.

If > 0, will be activated a filter that will improve the scene-change detection,

by discarding images that are similar.

Suggested values are between 0.35 and 0.85. Range [0-1], default = 0.0 (deactivated)

sc_min_int: Minimum number of frame interval between scene changes. Range [1, 25], default = 1.

sc_min_freq: If > 0 will be generated at least a reference frame every sc_min_freq frames.

Range [0-1500], default = 0.

sc_debug: If True will enable scene changes debug messages. Range [True, False], default = False

6.8 HAVC_extract_reference_frames

This is an HAVC utility function that perform Scene change detection and the export the reference frames. The header of the function is the following:

HAVC_extract_reference_frames(clip: vs.VideoNode, sc_threshold: float = DEF_THRESHOLD, sc_tht_offset: int = 1,

sc_tht_ssim: float = 0.0, sc_min_int: int = 1, sc_min_freq: int = 0, sc_framedir: str = "./",

sc_normalize: bool = True, ref_offset: int = 0, sc_tht_white: float = DEF_THT_WHITE,

sc_tht_black: float = DEF_THT_BLACK, ref_ext: str = "jpg", ref_jpg_quality: int = DEF_JPG_QUALITY,

ref_override: bool = True, sc_debug: bool = False) -> vs.VideoNode:

Where:

clip: clip to process, only RGB24 format is supported.

sc_threshold: Scene changes threshold used to generate the reference frames.

It is a percentage of the luma change between the previous n-frame (n=sc_the_offset)

and the current frame. range [0-1], default 0.10.

sc_tht_offset: Offset index used for the Scene change detection. The comparison will be performed,

between frame[n] and frame[n-sc_tht_offset]. An sc_tht_offset > 1 is useful to detect blended scene

change. Range [1, 25], default = 1.

sc_normalize: If true the B&W frames are normalized before apply scene detection filter, the normalization will

increase the sensitivity to smooth scene changes.

sc_tht_white: Threshold to identify white frames, range [0-1], default 0.88.

sc_tht_black: Threshold to identify dark frames, range [0-1], default 0.12.

sc_tht_ssim: Threshold used by the SSIM (Structural Similarity Index Metric) selection filter.

If > 0, will be activated a filter that will improve the scene-change detection,

by discarding images that are similar.

Suggested values are between 0.35 and 0.85. Range [0-1], default = 0.0 (deactivated)

sc_min_int: Minimum number of frame interval between scene changes. Range [1, 25], default = 1.

sc_min_freq: If > 0 will be generated at least a reference frame every sc_min_freq frames.

Range [0-1500], default = 0.

sc_framedir: If set, define the directory where are stored the reference frames.

The reference frames are named as: ref_nnnnnn.[jpg | png].], for example the reference frame 897

must be named: ref_000897.jpg or ref_000897.png.

ref_offset: Offset number that will be added to the number of generated frames, default = 0.

ref_ext: File extension and format of saved frames. Range ["jpg", "png"] , default = "jpg" ref_jpg_quality: Quality of jpg compression. Range [0, 100], default = 95

ref_override: If True, the reference frames with the same name will be overridden, otherwise will be discarded. Range [True, False], default = True

sc_debug: If True will enable scene changes debug messages. Range [True, False], default = False

6.9 HAVC_export_reference_frames

This is an HAVC utility function that export the reference frames of a clip. The clip must have the frame property '_SceneChangePrev' and '_SceneChangeNext' set. The header of the function is the following:

HAVC_export_reference_frames(clip: vs.VideoNode, sc_framedir: str = "./", ref_offset: int = 0,

ref_ext: str = "jpg", ref_jpg_quality: int = DEF_JPG_QUALITY, ref_override: bool = True) -> vs.VideoNode:

Where:

clip: clip to process, only RGB24 format is supported.

sc_framedir: If set, define the directory where are stored the reference frames.

The reference frames are named as: ref_nnnnnn.[jpg | png].], for example the reference frame 897

must be named: ref_000897.jpg or ref_000897.png.

ref_offset: Offset number that will be added to the number of generated frames, default = 0.

ref_ext: File extension and format of saved frames. Range ["jpg", "png"] , default = "jpg"

ref_jpg_quality: Quality of jpg compression. Range [0, 100], default = 95

ref_override: If True, the reference frames with the same name will be overridden, otherwise will

be discarded. Range [True, False], default = True

6.10 HAVC_bw_tune

Pre/post - process filter for improving contrast and luminosity of clips to be colored with HAVC.

HAVC_bw_tune(clip: vs.VideoNode = None, bw_tune: str = 'none', action: str = **None**,

hue: float = 0, sat: float = 1, bright: float = 0, cont: float = 1) -> vs.VideoNode

Where:

clip: Clip to process. Only RGB24 format is supported.

bw_tune: This parameter allows to improve contrast and luminosity of input clip to be colored with HAVC. Allowed values are:

'None' (default)

'Light',

'Medium',

'Strong'

'Custom' (it allows to adjust: hue, saturation, brightness, contrast)

action: This parameter allows to apply the improvement to input clip and to revert the adjustments.

Allowed values are:

'ON': the adjustments are applied on the input clip

'OFF', the adjustments previous applied are almost reverted

None, allowed only for bw_tune = 'Custom'

Parameters available only with bw_tune = 'Custom' mode:

hue: Adjust the color hue of the image. hue>0.0 shifts the image towards red. hue<0.0 shifts the image towards green. Range -180.0 to +180.0, default = 0.0

sat: Adjust the color saturation of the image by controlling gain of the color channels. sat>1.0 increases the saturation. sat<1.0 reduces the saturation. Use sat=0 to convert to *GreyScale*. Range 0.0 to 10.0, default = 1.0

bright: Change the brightness of the image by applying a constant bias to the luma channel. bright>0.0 increases the brightness. bright<0.0 decreases the brightness. Range -255.0 to 255.0, default = 0.0

cont: Change the contrast of the image by multiplying the luma values by a constant. cont>1.0 increase the contrast (the luma range will be stretched). cont<1.0 decrease the contrast (the luma range will be contracted). Range 0.0 to 10.0, default = 1.0.

6.11 HAVC_merge

Utility function with the implementation of HAVC merge methods.

HAVC_merge(clipa: vs.VideoNode, clipb: vs.VideoNode, clip_luma: vs.VideoNode = **None**, weight: float = 0.5,

method: int = 2, cmc_tresh: float = 0.2, lmm_p: list = (0.2, 0.8, 1.0), alm_p: list = (0.8, 1.0, 0.25)

) -> vs.VideoNode:

clipa: first clip to merge, only RGB24 format is supported

clipb: second clip to merge, only RGB24 format is supported

method: method used to combine clipa with clipb (default = 2):

0 : clipa only (no merge)

1 : clipb only (no merge)

2 : Simple Merge (default):

the frames are combined using a weighted merge, where the parameter weight represent the weight assigned to the colors provided by the clipb frames.

If weight = 0 will be returned clipa, if = 1 will be returned clipb.

3 : Constrained Chroma Merge:

The frames are combined by assigning a limit to the amount of difference in chroma values between clipa and clipb this limit is defined by the threshold parameter cmc_tresh.

The limit is applied to the image converted to "YUV". For example, when cmc_tresh=0.2, the chroma values "U","V" of clipb frame will be constrained to have an absolute percentage difference respect to "U","V" provided by clipa not higher than 20%. The final limited frame will be merged again with the clipa frame. With this method is suggested a starting weight > 50% (ex. = 60%).

4 : Luma Masked Merge:

the frames are combined using a masked merge, the pixels of clipb with luma < "luma_mask_limit" will be filled with the pixels of clipa.

If "luma_white_limit" > "luma_mask_limit" the mask will apply a gradient till "luma_white_limit". If the parameter weight > 0 the final masked frame will be merged again with the clipa frame.

5 : Adaptive Luma Merge:

The frames are combined by decreasing the weight assigned to clipb when the luma is below a given threshold given by: luma_threshold. The weight is calculated using the formula:

merge_weight = max(weight * (luma/luma_threshold)^alpha, min_weight).

For example, with: luma_threshold = 0.6 and alpha = 1, the weight assigned to clipb will start to decrease linearly when the luma < 60% till "min_weight".

For alpha=2, begins to decrease quadratically (because luma/luma_threshold < 1).

The methods 3 and 4 are similar to Simple Merge, but before the merge with clipa the clipb frame is limited in the chroma changes (method 3) or limited based on the luma (method 4).

The method 5 is a Simple Merge where the weight decreases with luma.

weight: weight given to clipb in all merge methods. If weight = 0 will be returned clipa, if = 1 will be returned clipb. range [0-1] (0.01=1%)

cmc_tresh: chroma_threshold (%), used by: Constrained Chroma Merge, range [0-1] (0.01=1%)

lmm_p: parameters for method: Luma Masked Merge (see method=4 for a full explanation)

[0] : luma_mask_limit: luma limit for build the mask used in Luma Masked Merge, range [0-1] (0.01=1%)

[1] : luma_white_limit: the mask will apply a gradient till luma_white_limit, range [0-1] (0.01=1%)

[2] : luma_mask_sat: if < 1 the clipb dark pixels will substitute with the desaturated clipa pixels, range [0-1] (0.01=1%)

alm_p: parameters for method: Adaptive Luma Merge (see method=5 for a full explanation)

[0] : luma_threshold: threshold for the gradient merge, range [0-1] (0.01=1%)

[1] : alpha: exponent parameter used for the weight calculation, range [>0]

[2] : min_weight: min merge weight, range [0-1] (0.01=1%)

clip_luma: if specified, clip_luma will be used as source of luma component for the merge. It is an optional parameter, and it is suggested to provide the clip with the best luma resolution between clipa and clipb. It is used only with the methods: 3, 4, 5 and can speed up the filter when it uses these methods.

6.12 HAVC_recover_clip_color

Utility function to restore the colors of gray pixels in the input clip by using the colors provided in the clip: clip_color. Useful to repair the clips colored with DeepRemaster.

HAVC_recover_clip_color(clip: vs.VideoNode = **None**, clip_color: vs.VideoNode = **None**, sat: float = 0.8, tht: int = 30,

weight: float = 0.0, alpha: float = 2.0, chroma_resize: bool = True, return_mask: bool = False,

binary_mask: bool = False) -> vs.VideoNode:

Where:

clip: clip to repair the colors, only RGB24 format is supported

clip_color: clip with the colors to restore, only RGB24 format is supported

sat: this parameter allows to change the saturation of colored clip (default = 0.8)

tht: threshold to identify gray pixels, range[0, 255] (default = 30)

weight: if > 0, the restored frame will be merged with clip_color frame. (default = 0.0)

alpha: parameter used to control the steepness of gradient curve, values above the default value, will preserve more pixels, but could introduce some artifacts, range[1, 10] (default = 2)

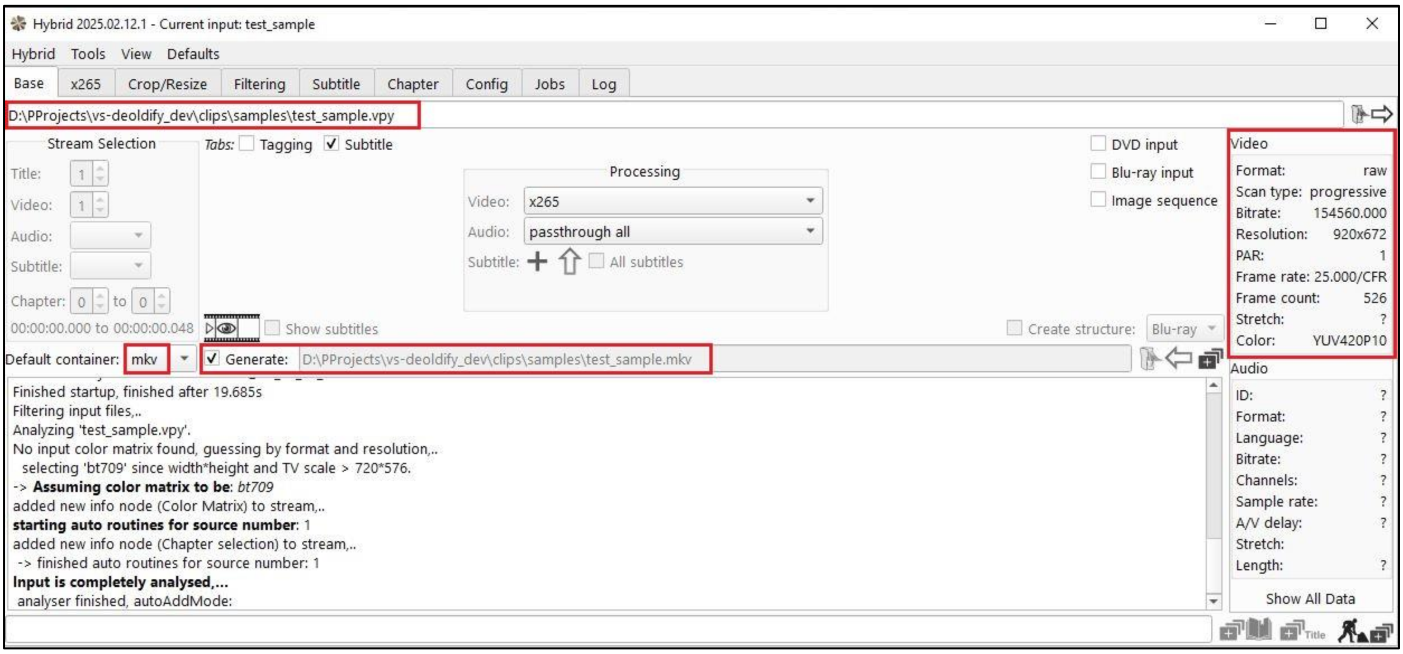
chroma_resize: if True, the frames will be resized to improve the filter speed (default = True)

return_mask: if True, will be returned the mask used to identify the gray pixels (white region), could be useful to visualize the gradient mask for debugging, (default = false).

binary_mask: if True, will be used a binary mask instead of a gradient mask, could be useful to get a clear view on the selected desaturated regions for debugging, (default = false).

7.0 Sample scripts

In this chapter will be shown some useful Vapoursynth scripts using the HAVC functions. In Hybrid is possible to provide in input, not only movies, but also Vapoursynth scripts as shown in the picture below:



Once the script has been loaded by Hybrid it is possible to encode it directly or to add some filters before start the encoding process.

All the filters that will be added in Hybrid after the loading of Vapoursynth script will be applied on the clip generated by the script in input. The script that will be executed is the following:


```
from importlib.machinery import SourceFileLoader
# Imports
import vapoursynth as vs
# getting Vapoursynth core
core = vs.core
# loading plugins
core.std.LoadPlugin(path="D:/Programs/Hybrid/64bit/vsfilters/GrainFilter/AddGrain/AddGrain.dll")
core.std.LoadPlugin(path="D:/Programs/Hybrid/64bit/vsfilters/SharpenFilter/CAS/CAS.dll")
# Source: 'D:\PProjects\vs-deoldify_dev\clips\samples\test_sample.vpy'
# Current color space: YUV420P8, bit depth: 8, resolution: 920x672, frame rate: 25fps, scanorder: progressive, yuv luminance scale: full,
matrix: 709, format: raw
output = SourceFileLoader('%script', 'D:/PProjects/vs-deoldify_dev/clips/samples/test_sample.vpy').load_module().vs.get_output()
try:
  if isinstance(output, vs.VideoOutputTuple):
    clip = output.clip
  else:
    clip = output
except AttributeError:
  clip = output
# contrast sharpening using CAS
clip = core.cas.CAS(clip=clip, sharpness=0.600)
# adding Grain using AddGrain
clip = core.grain.Add(clip=clip, var=2.00)
# adjusting output color from: YUV420P8 to YUV420P10 for x265Model
clip = core.resize.Bicubic(clip=clip, format=vs.YUV420P10, range_s="full")
# set output frame rate to 25fps (progressive)
clip = core.std.AssumeFPS(clip=clip, fpsnum=25, fpsden=1)
# output
clip.set_output()
```

In this example were added the filters: **CAS** (to increase the sharpness) and **AddGrain** (to add some grain).

As is possible to see these filters are applied on the script output, which must be a clip.

Example 1: script to restore a colored video using DeepRemaster

```
# Imports
import vapoursynth as vs
# getting Vapoursynth core
import sys
import os
core = vs.core
# Import scripts folder
scriptPath = 'D:/Programs/Hybrid/64bit/vsscripts'
sys.path.insert(0, os.path.abspath(scriptPath))
# loading plugins
core.std.LoadPlugin(path="D:/Programs/Hybrid/64bit/vsfilters/MiscFilter/MiscFilters/MiscFilters.dll")
core.std.LoadPlugin(path="D:/Programs/Hybrid/64bit/vsfilters/SourceFilter/LSmashSource/LSMASHSource.dll")
# Import scripts
import validate
import vsdeoldify as havc
import adjust
clip = havc.HAVC_read_video(source="source_bw_clip.mp4")
clipRef = havc.HAVC_read_video(source="reference_colored_clip.mp4", fpsnum=clip.fps_num, fpsden=clip.fps_den)
# change of hue, saturation, contrast and bright (optional)
clipRef = havc.HAVC_bw_tune(clipRef, bw_tune="custom", hue=5.00, sat=1.05, cont=0.80, bright=-1.1)
# ---------------------------------------------------------------------------------
clip = havc.HAVC_restore_video(clip, clipRef, ex_model = 2, ref_thresh = 0.10, ref_freq = 10,
max_memory_frames = 50, render_vivid = True)
# adjusting output color to YUV420P10 for x265Model
clip = core.resize.Bicubic(clip=clip, format=vs.YUV420P10, matrix_s="709", range_s="limited")
# set output frame rate (progressive)
clip = core.std.AssumeFPS(clip=clip, fpsnum=clip.fps_num, fpsden=clip.fps_den)
# output
clip.set_output()
```

Example 2: Merging of 2 colored clips and restore of the original luma e resolution of B&W clip

```
# Imports
import vapoursynth as vs
# getting Vapoursynth core
import sys
import os
core = vs.core
# Import scripts folder
scriptPath = 'D:/Programs/Hybrid/64bit/vsscripts'
sys.path.insert(0, os.path.abspath(scriptPath))
core.std.LoadPlugin(path="D:/Programs/Hybrid/64bit/vsfilters/MiscFilter/MiscFilters/MiscFilters.dll")
core.std.LoadPlugin(path="D:/Programs/Hybrid/64bit/vsfilters/SourceFilter/LSmashSource/LSMASHSource.dll")
# Import scripts
import validate
import vsdeoldify as havc
# clip_YUV = B&W Clip
clip = havc.HAVC_read_video(source="source_bw_clip.mp4")
# Resize clip
clip = core.resize.Spline36(clip=clip, width=1280, height=784)
# adjusting output color to YUV420P10 for x265Model
clip_YUV = core.resize.Bicubic(clip=clip, format=vs.YUV420P10, matrix_s="709", range_s="limited",
dither_type="error_diffusion")
# clip = CLIP COLORED WITH HAVC
clip = havc.HAVC_read_video(source="colored_havc_clip.mp4", fpsnum=clip_YUV.fps_num, fpsden=clip_YUV.fps_den)
# clip = CLIP COLORED WITH DEEPREMASTER
clip_ref = havc.HAVC_read_video(source="colored_deepremaster_clip.mp4",fpsnum=clip_YUV.fps_num,
fpsden=clip_YUV.fps_den)
clip_ref = havc.HAVC_bw_tune(clip=clip_ref, bw_tune="custom", hue=0.00, sat=0.70, cont=1.0, bright=0)
# ---------------------- START MERGING & RESTORE LUMA -------------------------------------
clip = core.std.Merge(clipa=clip,clipb=clip_ref,weight=0.60)
# adjusting output color YUV420P10 for x265Model
clip = core.resize.Bicubic(clip=clip, format=vs.YUV420P10, matrix_s="709", range_s="limited",
dither_type="error_diffusion")
clip = core.std.ShufflePlanes(clips=[clip_YUV, clip, clip], planes=[0, 1, 2], colorfamily= vs.YUV)
# set output frame rate to (progressive)
clip = core.std.AssumeFPS(clip=clip, fpsnum=clip_YUV.fps_num, fpsden=clip_YUV.fps_den)
# output
clip.set_output()
```

Example 3: Remove the flickering produced by DeepRemaster

```
# Imports
import vapoursynth as vs
# getting Vapoursynth core
import sys
import os
```

```
core = vs.core
# Import scripts folder
scriptPath = 'D:/Programs/Hybrid/64bit/vsscripts'
sys.path.insert(0, os.path.abspath(scriptPath))
# loading plugins
core.std.LoadPlugin(path="D:/Programs/Hybrid/64bit/vsfilters/SourceFilter/LSmashSource/LSMASHSource.dll")
core.std.LoadPlugin(path="D:/Programs/Hybrid/64bit/vsfilters/FrameFilter/ReduceFlicker/ReduceFlicker.dll")
# Import scripts
import validate
import vsdeoldify as havc
clip = havc.HAVC_read_video(source="sample_deepremaster_flicker.mp4")
# ------------------------------------------------------
clip = havc.HAVC_stabilizer(clip, dark=True, stab=True, stab_p = (5, 'A', 1, 0, 0, 0))
# adjusting output color to YUV420P10 for x265Model
clip = core.resize.Bicubic(clip=clip, format=vs.YUV420P10, matrix_s="709", range_s="limited")
# removing flickering using ReduceFlicker
clip = core.rdfl.ReduceFlicker(clip=clip, strength=2, aggressive=0)
# ------------------------------------------------------
# set output frame rate (progressive)
clip = core.std.AssumeFPS(clip=clip, fpsnum=clip.fps_num, fpsden=clip.fps_den)
# output
clip.set_output()
```

Example 4: Merging a DeepRemaster clip with a simple colored clip with HAVC_merge

```
# Imports
import vapoursynth as vs
# getting Vapoursynth core
import sys
import os
core = vs.core
# Import scripts folder
scriptPath = 'D:/Programs/Hybrid/64bit/vsscripts'
sys.path.insert(0, os.path.abspath(scriptPath))
# loading plugins
core.std.LoadPlugin(path="D:/Programs/Hybrid/64bit/vsfilters/MiscFilter/MiscFilters/MiscFilters.dll")
core.std.LoadPlugin(path="D:/Programs/Hybrid/64bit/vsfilters/SourceFilter/LSmashSource/LSMASHSource.dll")
# Import scripts
import validate
import vsdeoldify as havc
clip1 = havc.HAVC_read_video(source="sample_deepremaster.mkv")
# decrease the saturation of deepremaster clip (optional)
clip1 = havc.HAVC_bw_tune(clip1, bw_tune="custom", hue=0.00, sat=0.90, cont=1.0, bright=0.0)
clip2 = havc.HAVC_read_video(source="sample_simple_colored.mkv")
# merging the 2 clips using the method: Adaptive Luma Merge
clip = havc.HAVC_merge(clipa=clip2, clipb=clip1, weight=0.80, method=5, alm_p=(0.9, 1.0, 0.65))
# adjusting output color from: RGB24 to YUV420P10 for x265Model
clip = core.resize.Bicubic(clip=clip, format=vs.YUV420P10, matrix_s="709", range_s="limited")
# set output frame rate (progressive)
clip = core.std.AssumeFPS(clip=clip, fpsnum=clip.fps_num, fpsden=clip.fps_den)
# output
clip.set_output()
```

Example 5: Recover DeepRemaster gray colors using a colored clip with HAVC_recover_clip_color

```
# Imports
import vapoursynth as vs
# getting Vapoursynth core
import sys
import os
core = vs.core
# Import scripts folder
scriptPath = 'D:/Programs/Hybrid/64bit/vsscripts'
sys.path.insert(0, os.path.abspath(scriptPath))
# loading plugins
core.std.LoadPlugin(path="D:/Programs/Hybrid/64bit/vsfilters/MiscFilter/MiscFilters/MiscFilters.dll")
core.std.LoadPlugin(path="D:/Programs/Hybrid/64bit/vsfilters/SourceFilter/LSmashSource/LSMASHSource.dll")
# Import scripts
import validate
import vsdeoldify as havc
clip1 = havc.HAVC_read_video(source="sample_deepremaster.mkv")
# decrease the saturation of deepremaster clip (optional)
clip1 = havc.HAVC_bw_tune(clip1, bw_tune="custom", hue=0.00, sat=0.90, cont=1.0, bright=0.0)
clip2 = havc.HAVC_read_video(source="sample_simple_colored.mkv")
# recover the gray colors
clip = havc.HAVC_recover_clip_color(clip1, clip2, tht=60, weight=0.0, alpha=2, return_mask=False)
# adjusting output color from: RGB24 to YUV420P10 for x265Model
clip = core.resize.Bicubic(clip=clip, format=vs.YUV420P10, matrix_s="709", range_s="limited")
# set output frame rate (progressive)
clip = core.std.AssumeFPS(clip=clip, fpsnum=clip.fps_num, fpsden=clip.fps_den)
# output
clip.set_output()
```

8.0 Useful companion software

To perform advanced coloring could be useful the following software:

8.0.1 Software for coloring pictures

The project interactive-deep-colorization , whose automatic colorization model is included in HAVC with the name of siggraph17, provides a useful tool that help to interactively colorize pictures. The installation of this software is quite complex, fortunately this software has been added in Photoshop Elements since version 2020, see this link for more details: Automatically colorize your photos.

8.0.2 Software for processing batch of pictures

Sometime it will be necessary to process a significant number of reference frames, for example to change the size and recompress in jpg. There are a lot of software to perform this task. I found that XnView is good tool to perform these tasks and I suggest to use it.

Sometime it will be also, necessary to rename a significant number of reference frames. There are a lot of software to perform this task. I found that Advanced Renamer is a good tool to perform this task and I suggest to use it.

8.2 Useful Web Links

In this chapter are provided some useful links related to the colorization

- The Hybrid forum has a specific thread for the filter HAVC: https://forum.selur.net/thread-3595.html
    - It has also a specific thread for general questions: https://forum.selur.net/forum-3.html
- On GitHub there is a specific project in collecting colorization papers: https://github.com/MarkMoHR/Awesome-Image-Colorization
- The Internet Archive Site is a useful resource to get interesting B&W movie to colorize.
    - huge list of movies: https://archive.org/details/opensource_movies
    - list of contributed AI colored movies[17]: https://archive.org/details/colorized-movies
    - list of contributed movies colored with HAVC: havc-colorized-movies
- For the users that want to understand better the scripts generated by Hybrid, using the Vapoursynth functions, is suggested to read the Vapoursynth documentation: http://www.vapoursynth.com/doc/

[17] Most of them are based on DeOldify with additional contrast and color correction via Avidemux.