HAVC User Guide

Ver 1.1.5 – January 2025

(based on HAVC 4.6.1 and Hybrid 2025.01.04.1)

Table of Contents

1.0 Introduction

This guide has been written to describe the Vapoursynth filter Hybrid Automatic Video Colorizer (aka DDeoldify) available on GitHub under MIT License. This filter (*HAVC* in short) 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*, (in alternative to DDcolor can use the models provided in the project Colorization[1]) that are some of the best models available for coloring pictures, providing often a final colorized image that is better than the image 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 this filter that help improve the 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 and the Deep Exemplar based Video Colorization model (*DeepEx* in short). Both *DeepEx* and *ColorMNet* are exemplar-based video colorization models and 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 and it is suggested to use it as default exemplar-based model.

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[2].

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**.

---
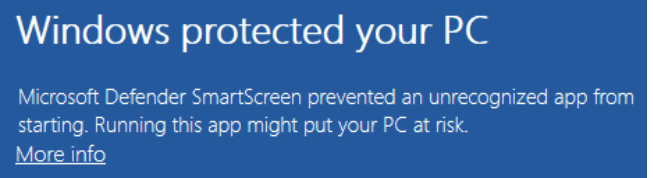
[1] The project *Colorization* includes 2 models: *Real-Time User-Guided Image Colorization with Learned Deep Priors* (Zhang, 2017) and *Colorful Image Colorization* (Zhang, 2016), these 2 models has been added as alternative models (named: **siggraph17, eccv16**) to *DDcolor*. These models have the same problem of temporal stability of colors observed in DDcolor and are slower than DDcolor. They can be used only with the Preset custom.

[2] For manual installation see the GitHub page: https://github.com/dan64/vs-deoldify

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[3].

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

📁 experimental	4️⃣ 4Selur	23 giu 2024 4Selur	—
🗜 vsgan_models_2023.07.10.7z 🤝	4️⃣ 4Selur	10 lug 2023 4Selur	7,61 GB
🗜 vs-mlrt_2024.11.22.7z 🤝	4️⃣ 4Selur	22 nov 2024 4Selur	5,61 GB
🗜 VapoursynthR70_torch_2024.12.27.7z 🤝	4️⃣ 4Selur	08:37 4Selur	12,59 GB
📄 README.md 🤝	4️⃣ 4Selur	11 ago 2024 4Selur	2 kB
🗜 onnx_models_2023.12.05.7z 🤝	4️⃣ 4Selur	5 dic 2023 4Selur	4,55 GB

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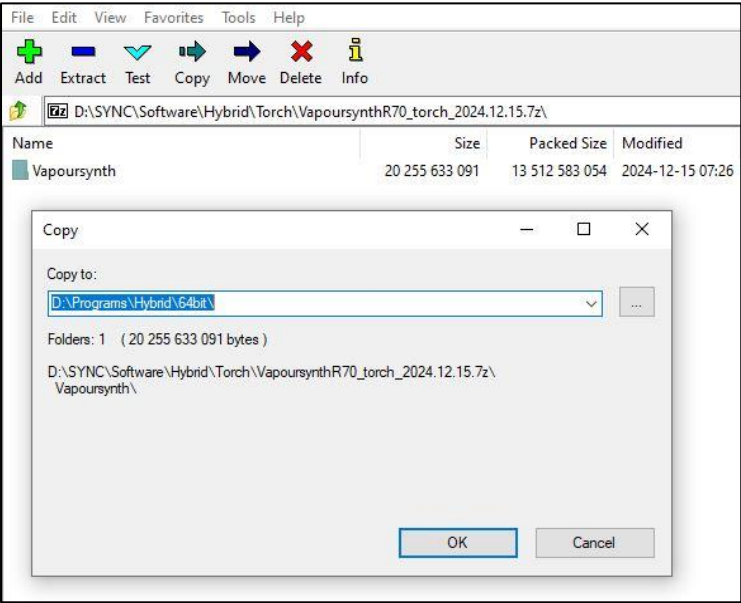
By opening it with 7-zip will be displayed the following window.

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 above) accordingly.

---

[3] For more useful settings see the page: https://forum.selur.net/thread-10.html

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.
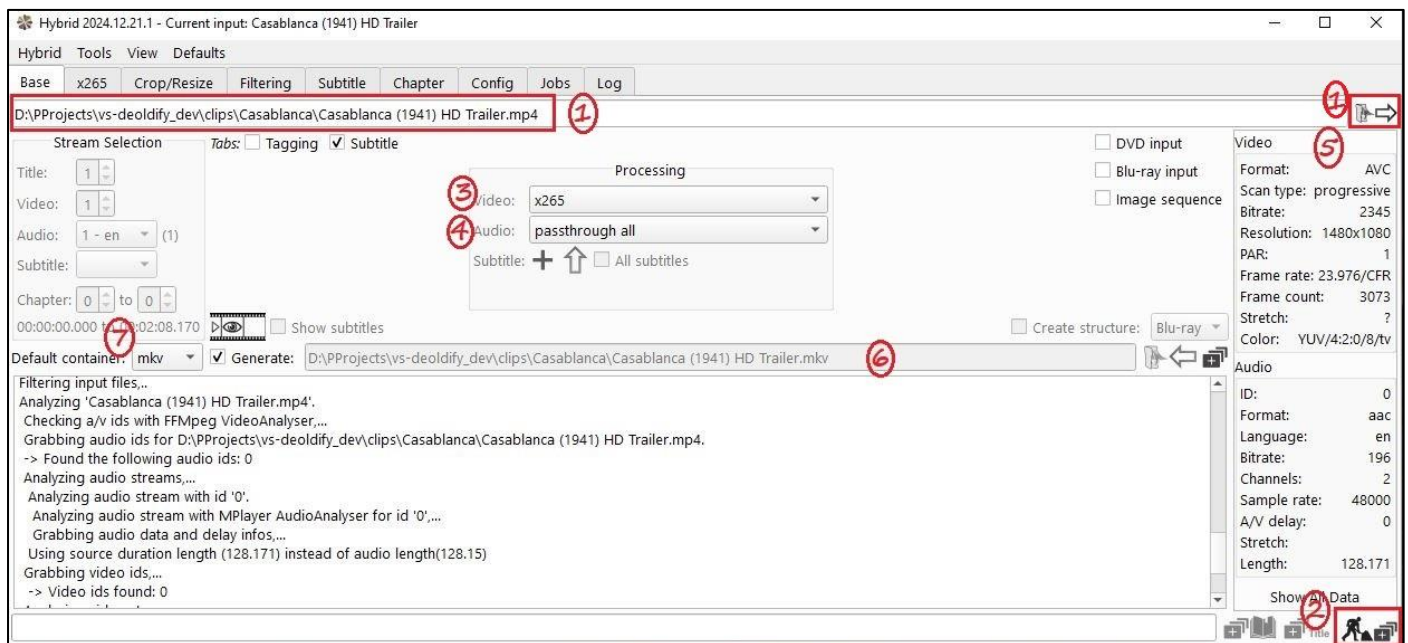
GUI Explanation[4]:

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".

---

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

In Hybrid there are a lot of filters, the coloring filters are available at: **Filtering->Vapoursynth->Color->Coloring**, as shown in the following picture:
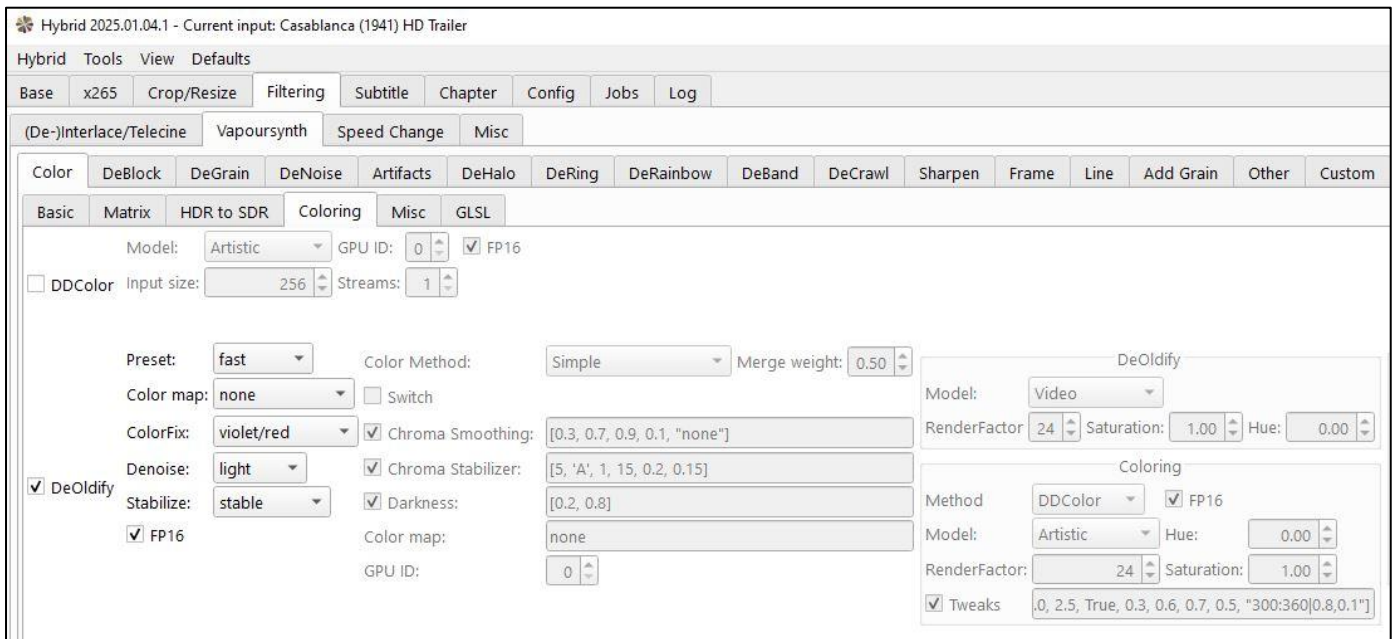
The HAVC filter is available under the checkbox DeOldify. In effect the first coloring filters added were *DeOldify* and *DDColor*, but then the DeOldify filter was extended adding more coloring methods, including ColorMNet and the Deep Exemplar based Video Colorization, hence the name of the filter was changed in HAVC, but for historical reason in Hybrid is still referenced as DeOldify. 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.

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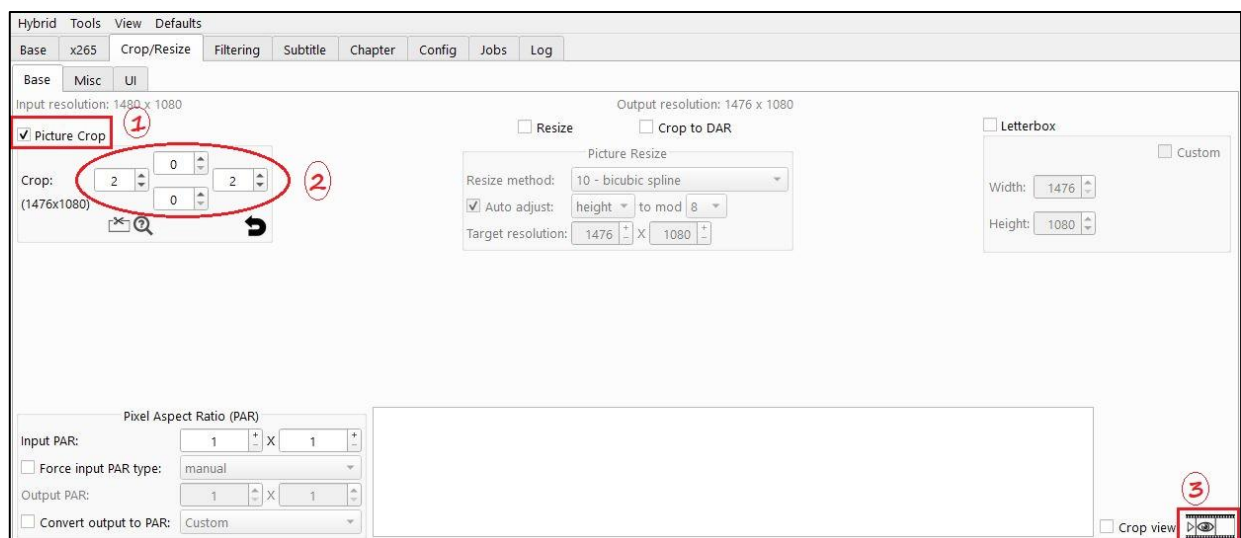
3.1 HAVC *pre-* and *post-* process filters

The main filters introduced are:

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 decrease 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).

Darkness: 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.

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 DDeoldify 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 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.

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 hue ranges:

Name	Color Mapping
blue->brown	180:280|+140,0.4
blue->red	180:280|+100,0.4
blue->green	180:280|+220,0.4
green->brown	80:180|+260,0.4
green->red	80:180|+220,0.4
green->blue	80:180|+140,0.4
red->brown	300:360,0:20|+40,0.6
red->blue	300:360,0:20|+260,0.6
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:

1. *DeOldify* only coloring model.

2. *DDColor* only color model.

3. *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](#)).

4. *Constrained Chroma Merge*: given that the colors provided by DeOldify's *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](#)).

5. *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](#)).
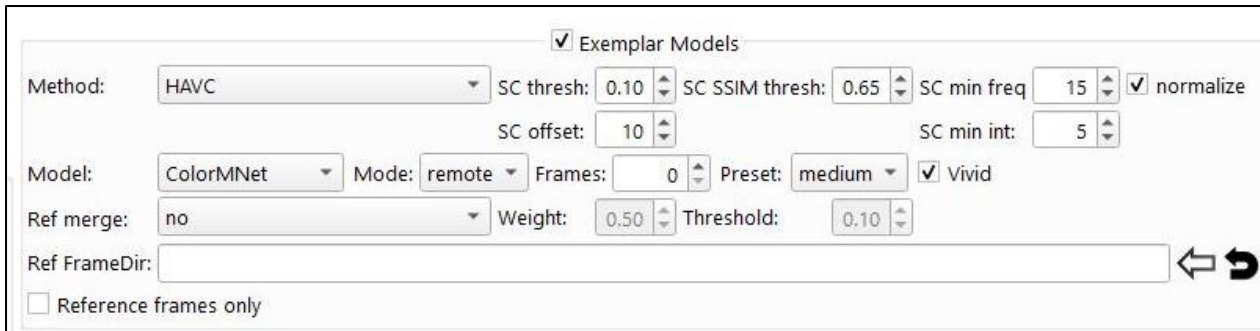
Adaptive Luma Merge: given that the DDcolor performance is quite bad on dark scenes, with this method the images are combined by decreasing the weight assigned to DDcolor frames when the luma is below the *luma_threshold*. For example, with: *luma_threshold* = 0.6 and *alpha* = 1, the weight assigned 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 decrease 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 and Deep Exemplar based Video Colorization model.

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 **0.05**), 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%).

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 = HAVC different from video

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]*. 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.

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)[5]. 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 low weight

- 2 = reference frames are merged with medium weight

- 3 = reference frames are merged with 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 as shown in the following picture:

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

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

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

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:

- **Preset:** medium (*fast* will increase the speed with a little decrease in color accuracy)

- **Color map:** none

- **ColorFix:** 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 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.

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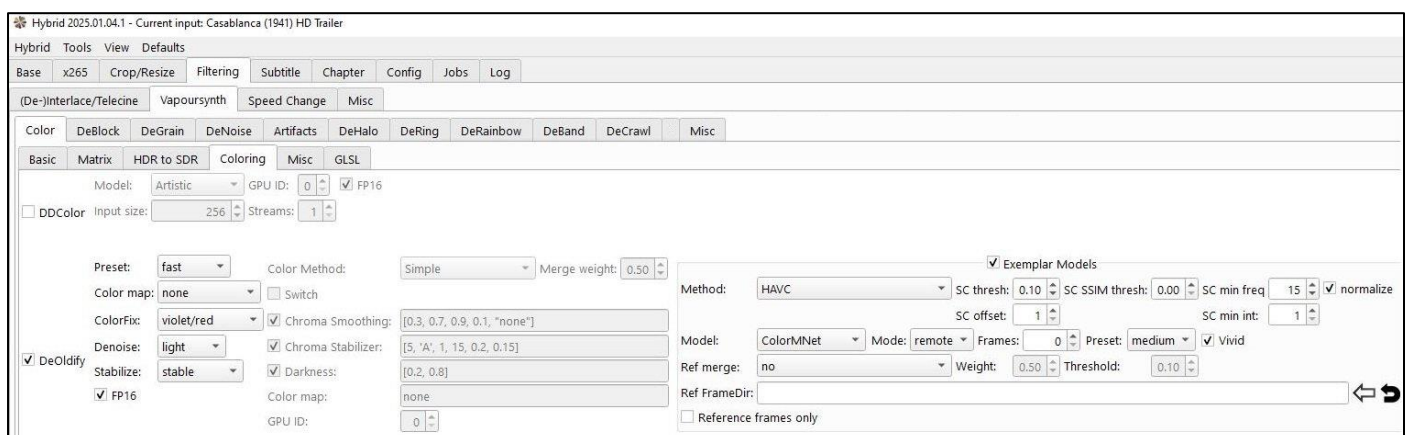
Once all the filter parameters are set it is possible start the encoding process by pressing the Encoding button 2 as described in the section GUI Explanation.

Willing to obtain the best results, but at cost of speed it is possible to improve further the color stability using the setting shown in the following picture:
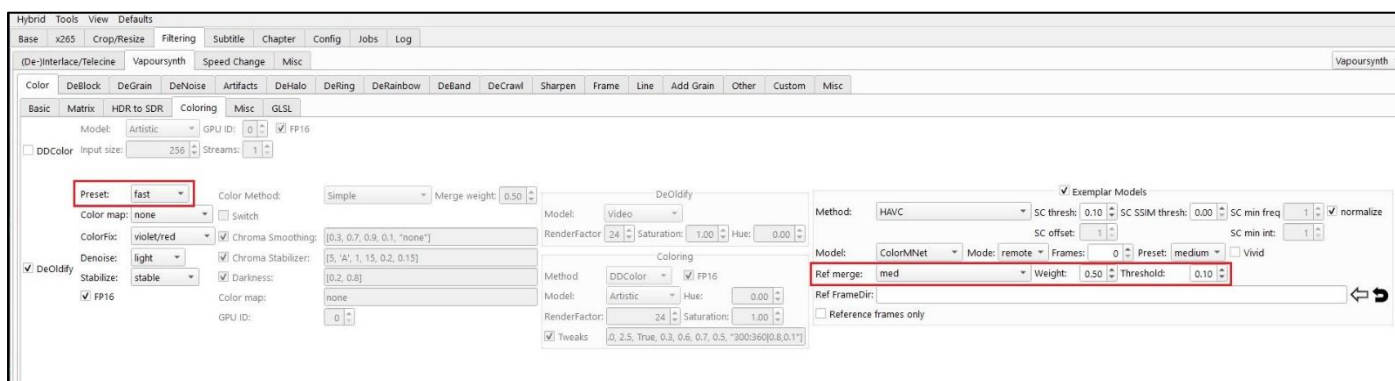
Respect to the previous configuration has been changed the following parameters (highlighted in red box):

- **Preset:** fast
- **Ref merge**: med
- **Weight**: 0.50
- **Threshold**: 0.10
- **Vivid:** unchecked

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[6] and at the same time to keep the color quality provided by DDColor and/or DeOldify. 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.

---

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

4.1 HAVC Color Mapping/Chroma Adjustment

In this chapter will be described the usage of parameters colormap and chroma adjustment.

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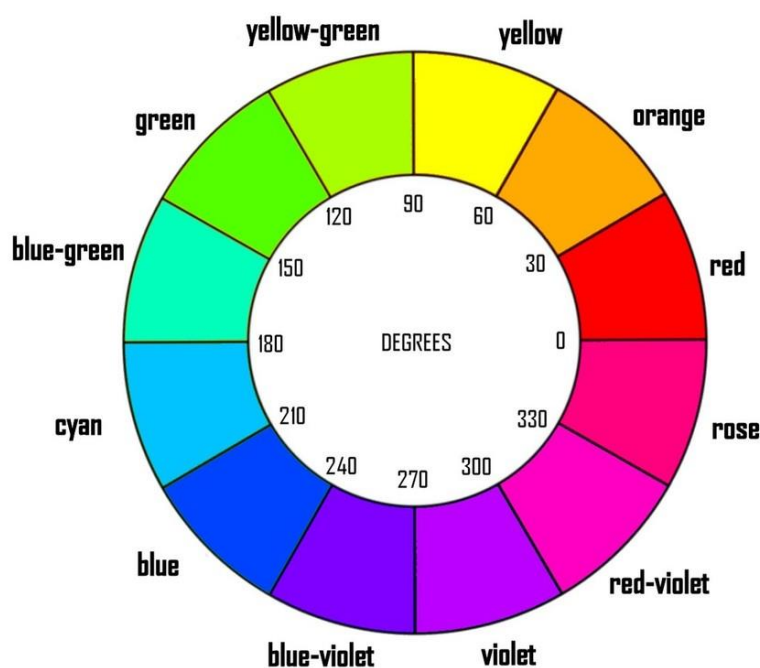
In the HSV color space the colors are specified in degrees from 0 to 360. In this case it is possible to see the yellow range is between 60 and 90. But to be more conservative it is better to include also the orange, so that the chroma range that we want adjust is the following: "30:90".

Now that we have select 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_ddeoldify()`

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

Here the result

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!

We can 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.

I created the following album to simplify the comparison: 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_ddeoldify(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_ddeoldify()` 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[7]: 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:
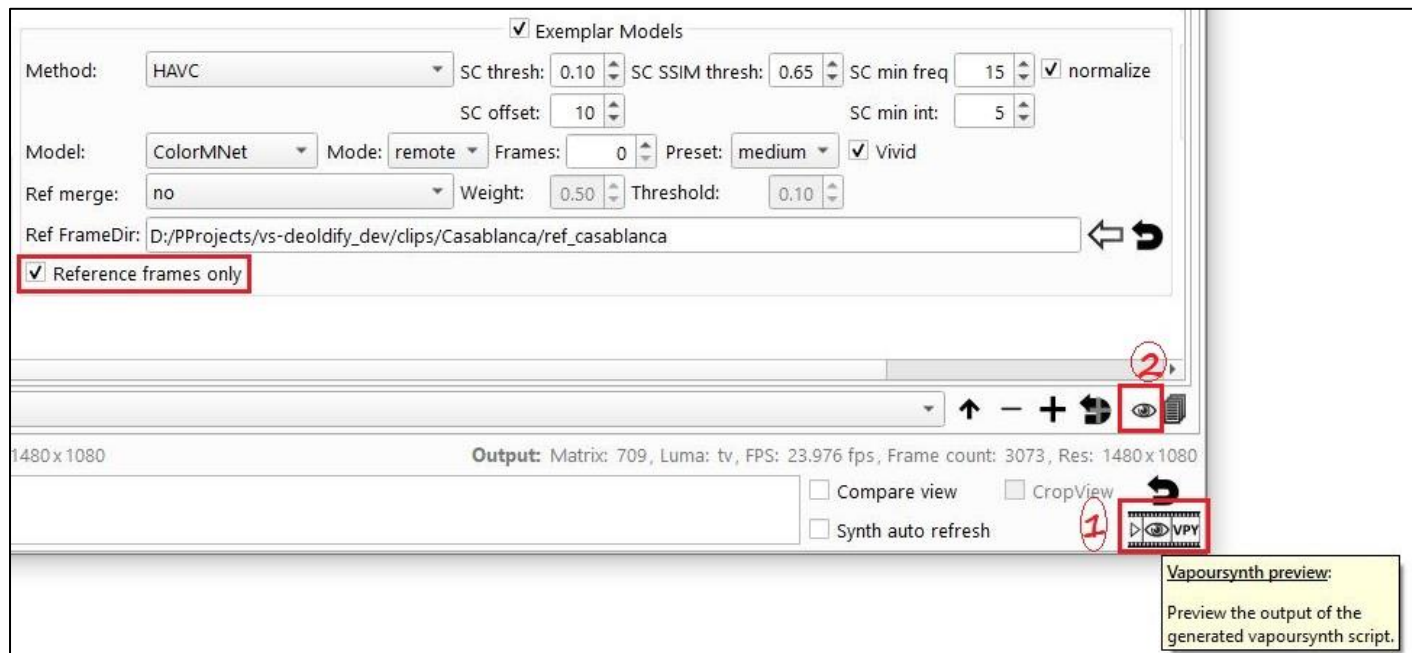
In the GUI there are 2 preview buttons. The **button (1)** will allow to preview the filtered frames, while the **button (2)** will allow to preview the code automatically generated by Hybrid. The check box **Reference frames** only is necessary to enable the export of frames that will be used by ColorMNet to propagate the colors. It is interesting to observe that

---

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

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 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 30sec) till is displayed the preview window (in this case the first frame is black).

Then is necessary open windows file explorer in the location folder of the clip provided in input to Hybrid to locate the preview script file (in this case is tempPreviewVapoursynthFile18_49_09_321.vpy). Should be visible something like this:
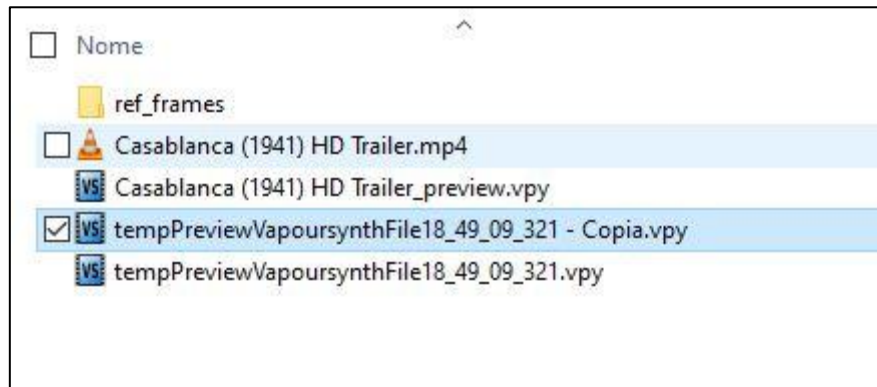
The files with ".vpy" extension should be associated to the executable "vsViewer.exe" stored in: ".\Hybrid\64bit\Vapoursynth". Once the association is applied is possible to open the preview file as shown in the following picture:


```
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 # changing range from limited to full range for vsDeOldify
37 clip = core.resize.Bicubic(clip, range_in_s="limited", range_s="full")
38 # setting color range to PC (full) range.
39 clip = core.std.SetFrameProps(clip=clip, _ColorRange=vs.RANGE_FULL)
40 # adjusting color space from YUV420P8 to RGB24 for vsDeOldify
41 clip = core.resize.Bicubic(clip=clip, format=vs.RGB24, matrix_in_s="709", range_s="full")
42 # adding colors using DeOldify
43 from vsdeoldify import HAVC_main
44 clip = HAVC_main(clip=clip, Preset="medium", VideoTune="stable", ColorFix="violet/red",
   ColorTune="light", ColorMap="none", EnableDeepEx=True, DeepExMethod=0, DeepExPreset="medium",
   DeepExRefMerge=0, DeepExOnlyRefFrames=True, ScFrameDir="D:/PProjects/vs-deoldify_dev/clips/
   Casablanca/ref_casablanca", ScThreshold=0.10, ScThtOffset=10, ScMinFreq=15, ScMinInt=5,
   ScThtSSIM=0.65, ScNormalize=True, DeepExModel=0, DeepExEncMode=0, DeepExVivid=True,
   DeepExMaxMemFrames=0, enable_fp16=True)
45 # changing range from full to limited range for vsDeOldify
46 clip = core.resize.Bicubic(clip, range_in_s="full", range_s="limited")
```
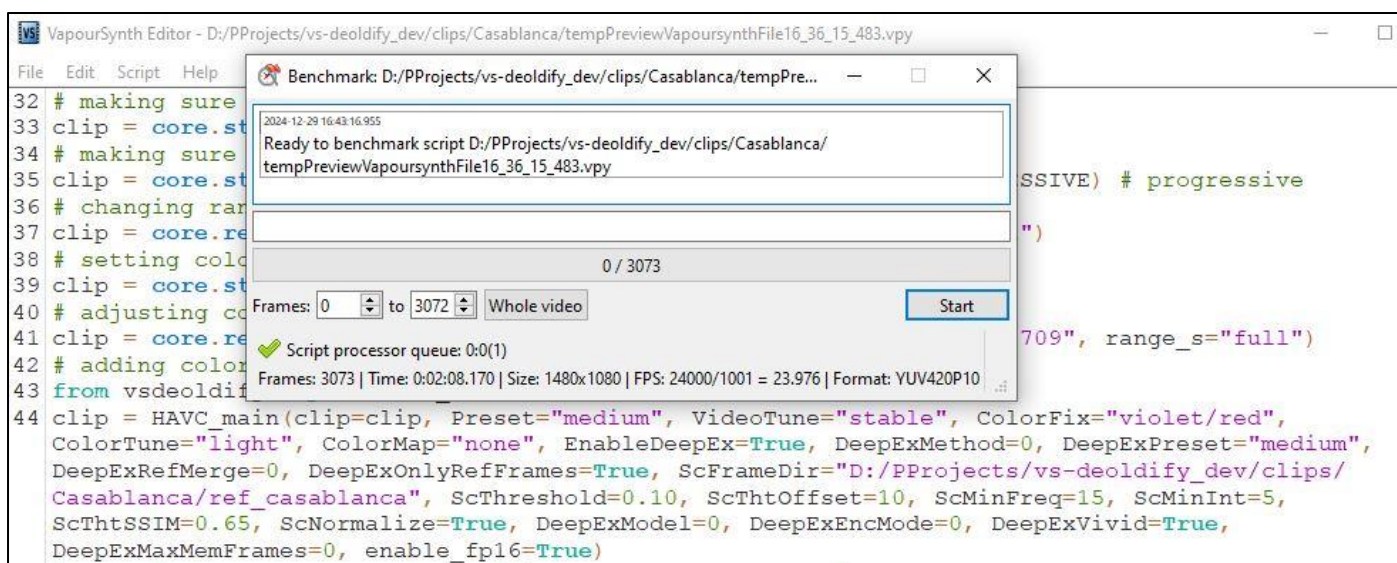
Then is necessary to select Script-Benchmark or press F7, as shown in the picture below:
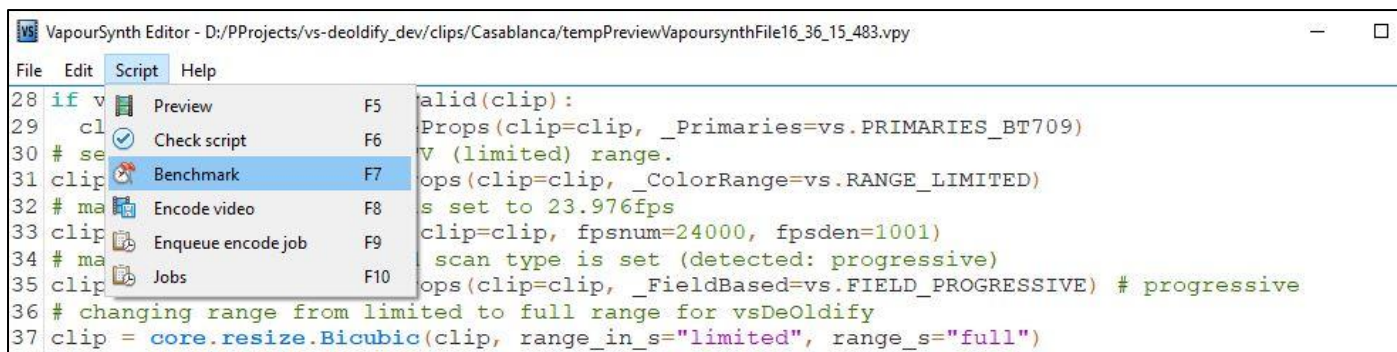
The Benchmark will run the script but will not call the encoding process, to that will not be generated any movie file. This is useful because in this case, what is necessary to generate are the reference frames.

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:
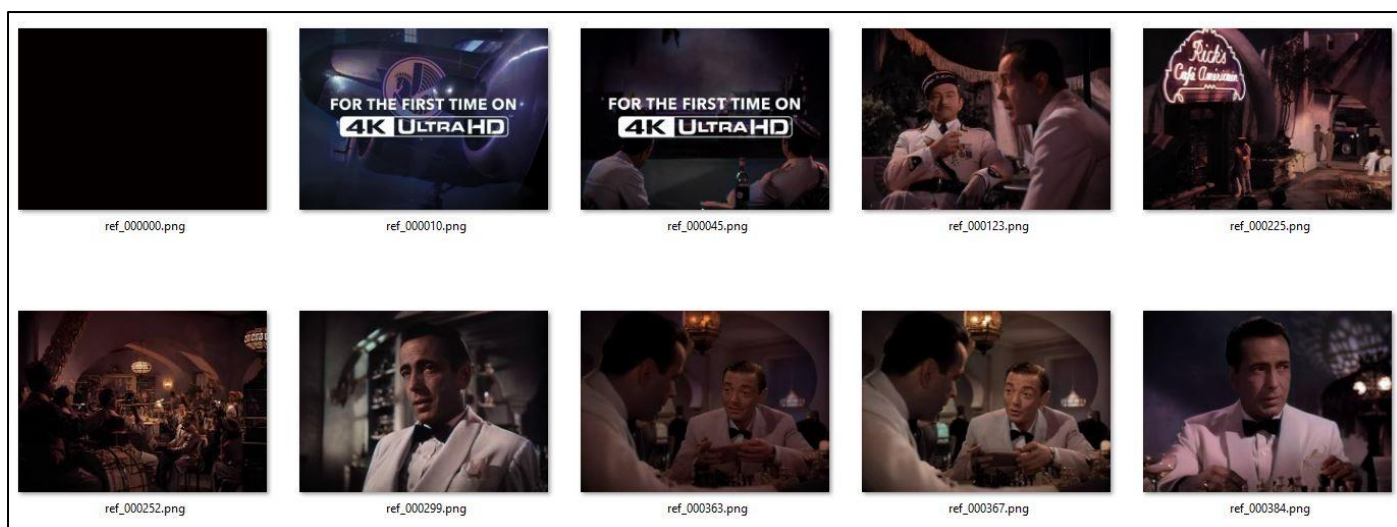
In the sample provided above is possible to see that the frames 363 and 367 are very similar, but 367 has better colors, since in this case we are interested in providing the best *exemplars* to ColorMNet, it is possible to delete the frames 363 and rename the frame 367 in ref_000363.png. When there 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 I will use 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:
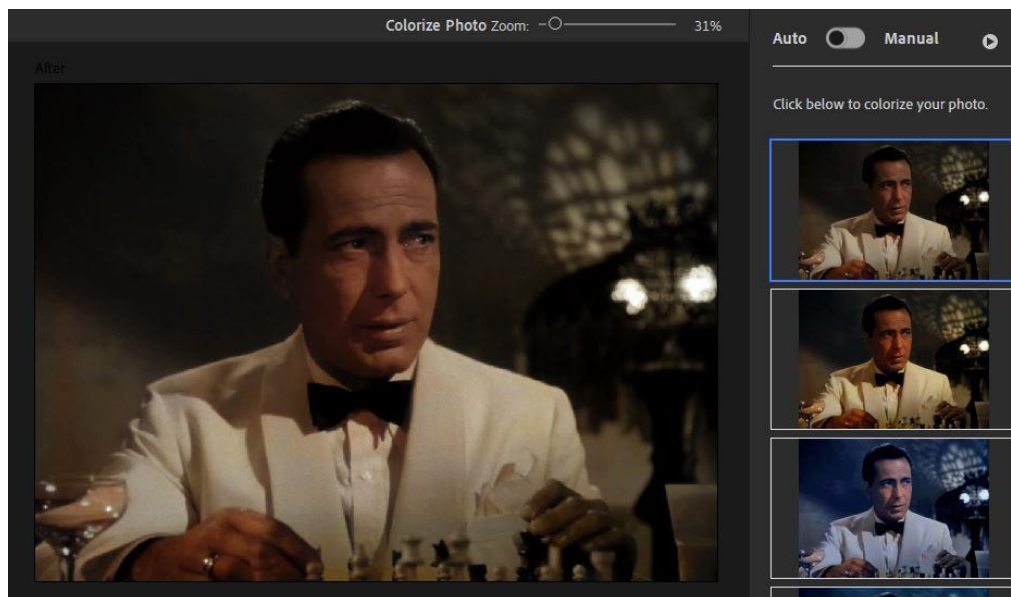
Buth the hand on the right is not colored well and is necessary to adjust the color manually as shown in the picture below:
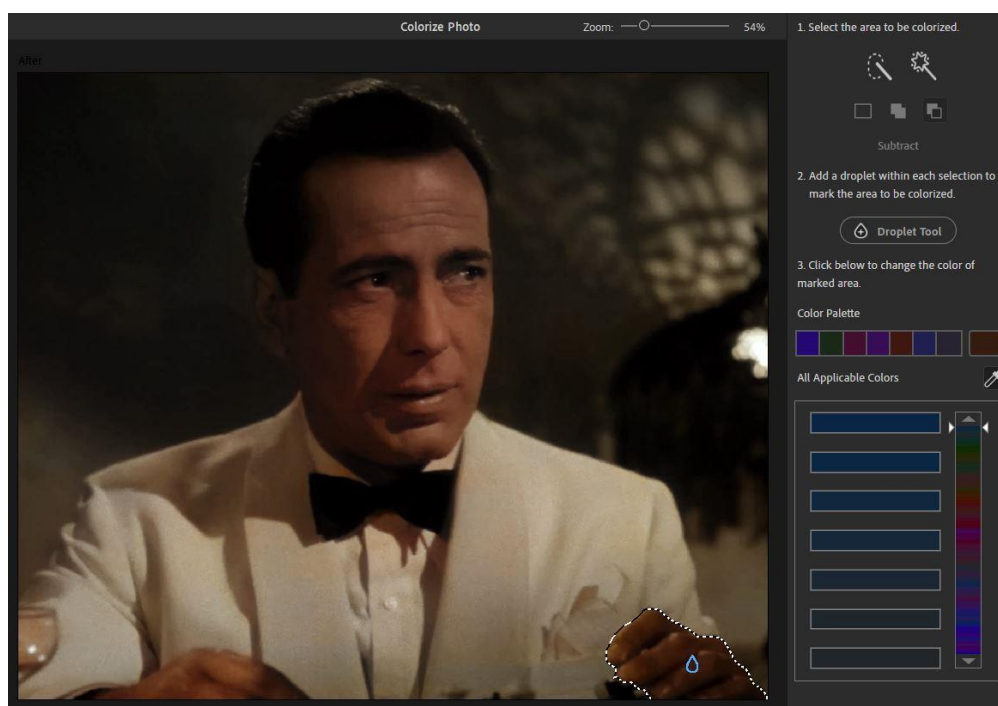
I have shown the most common cases that need to be addressed:

4) similar or duplicated frames: in this case is necessary to selected the best frame (eventually by renaming it) and delete all the remaining frames.
5) 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 pictures:

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 method HAVC as shown in the following picture:

the clip will be colored using the previous unadjusted reference frames, so the clip will show some color artifact. Alternatively, is possible to create a folder that contains only the fixed/adjusted reference frames, as shown in the following picture:
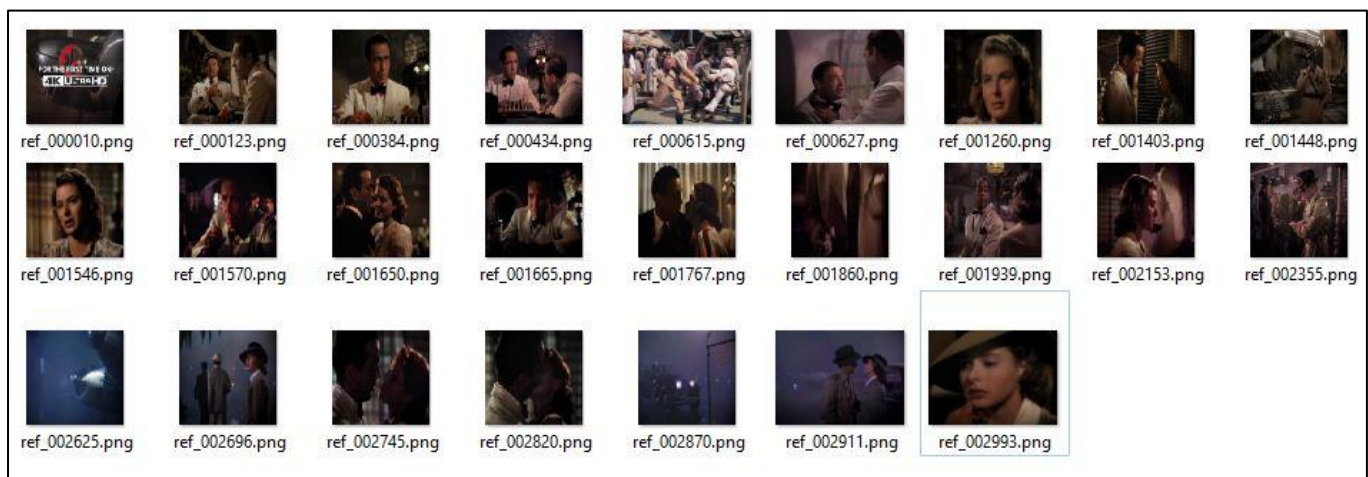
 Supposing that the folder is named "ref_casablanca_fixed", it is possible to color the clip using the following settings:

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.

4.3 Using HAVC custom settings

By selecting the Preset custom (see picture below) it is possible to manually set all the HAVC parameters.

The most important parameter is the Color Method this parameter 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 decrease with luma.

Finally, there are the 2 basic methods: **DeOldify only** and **DDColor only**. When are selected the frames will be colored using only DeOldify or DDcolor. For both the model the quality of colored frames is defined by the parameter *RenderFactor*, this parameter will affect both the quality and the speed, high values should provide better colored frames but will decrease significantly the inference speed, a good compromise value is between 24 and 28. For both the models is possible to change for the colored frames the *Saturation* and the *Hue*. For the model DDcolor are available a list of parameters called *Tweaks*. These parameters have been added because the DDcolor's inference is quite poor on dark/bright scenes. In DDcolor a gamma > 2.0 improves the quality, but on dark scenes the quality is worse, so has been added also a dynamic gamma correction, that allows to decrease the gamma with the luma, so that on dark scenes the gamma will be < 1. Also, the contrast can impact the DDcolor output. A contrast < 1 DDcolor images are less saturated, while with a contrast > 1 the images are more saturated (this effect can be obtained also with the parameter *Saturation*).

Using the **custom** preset It is suggested to enable the *DDColor Tweaks* (to apply the dynamic gamma correction) and the post-process filters: *Chroma Smoothing*, *Chroma Stabilization* and *Darkeness*. Unfortunately, is not possible provide a *one size fits-all solution* and the filter parameters need to be adjusted depending on the type of video to be colored.

4.3.1 Alternative inference models to DDcolor

In alternative to DDcolor in HAVC version 4.6.0 is also possible, by selecting the Method **Alternative**, 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 possible to use these models only with the Preset **custom**.

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:
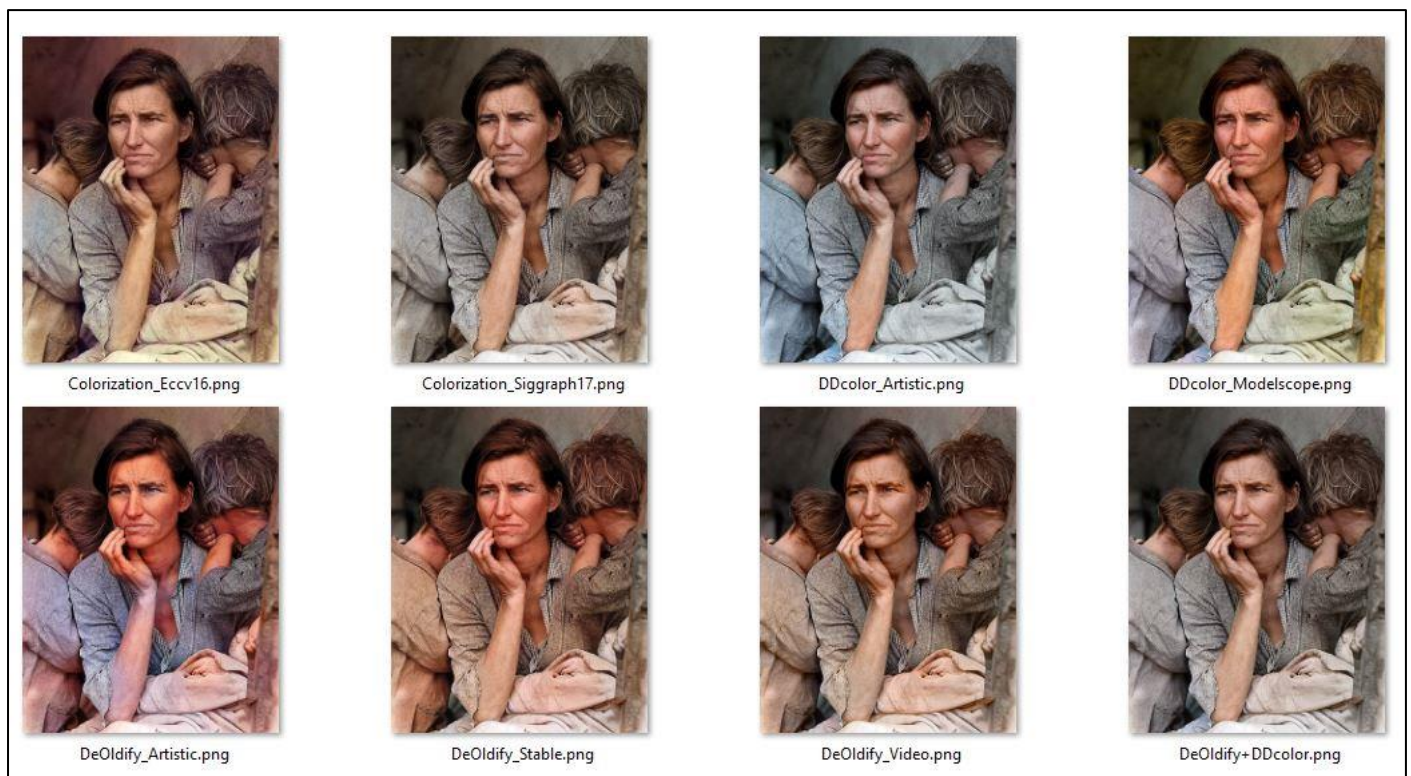
5.0 Using external filters to improve final HAVC color quality

In some cases, it is possible to improve the final color quality of the movies colorized using HAVC. In this chapter will be provide a guide on how to use a couple of filters already available in Hybrid to improve the final color quality.

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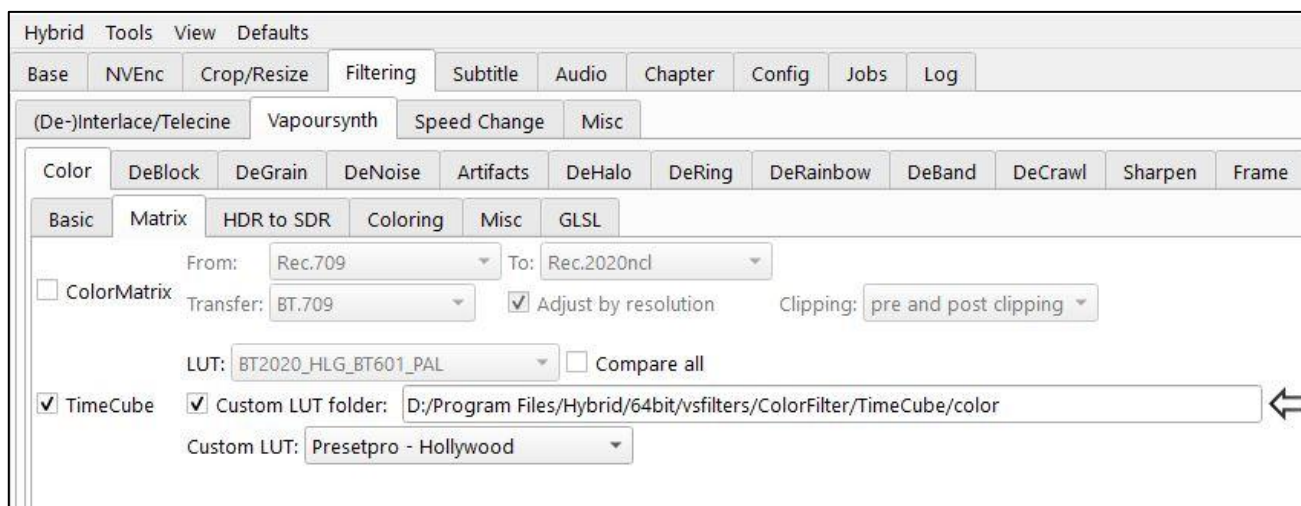
The LUT can be applied to the final video already encoded, and in this case, it is just enough to apply the above settings, where in the field **Custom LUT** is possible to select the LUT table to apply (in this example was selected the LUT named *Presetpro – Hollywood*).

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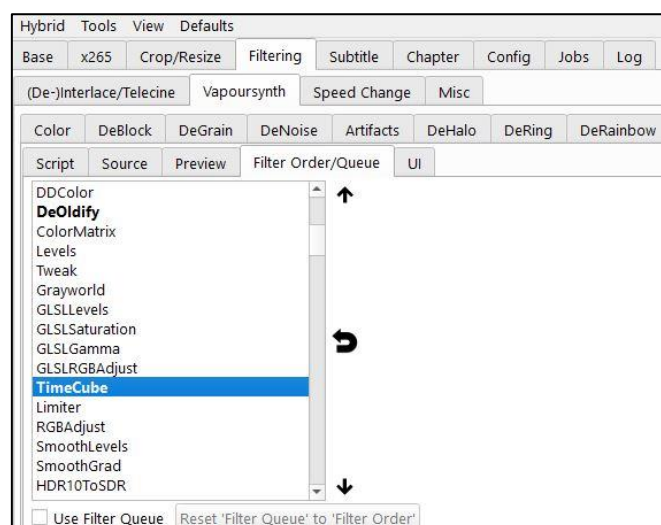
It is necessary to verify that the filter TimeCube is located after the filter DeOldify in the Order/Queue. By using the up and down arrows shown on the GUI.

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

!Original.png	AGF - Optima + film.png	FUJ - Astia.png	KDK - Portra + warm.png	Presetpro - Classic Film.png
Presetpro - Classic Ride.png	Presetpro - Hollywood.png	Presetpro - Renegade.png	Presetpro - Summit.png	Presetpro - Vibrant Light.png
Presetpro - Vintage Vibe LUT.png	Presetpro - Warm Haze.png	Stockpresets - Forest Film.png	Stockpresets - Roadrunner.png	Stockpresets - Shorey.png

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, using as reference the same image used for the LUT comparison.
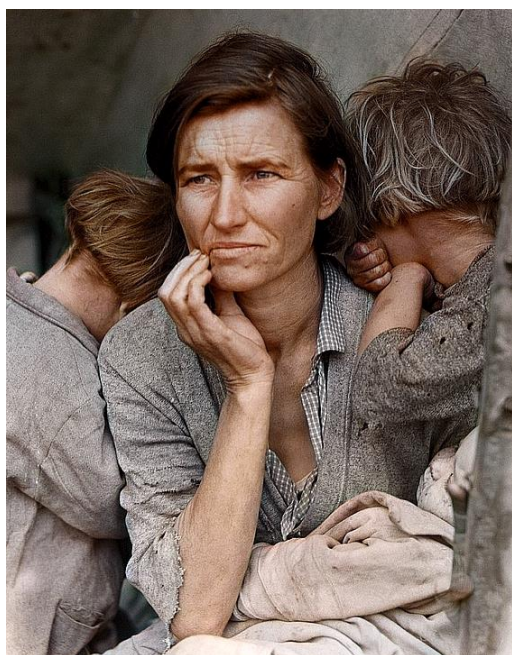
(Retinex + HAVC)   (HAVC only)

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 theory of 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.

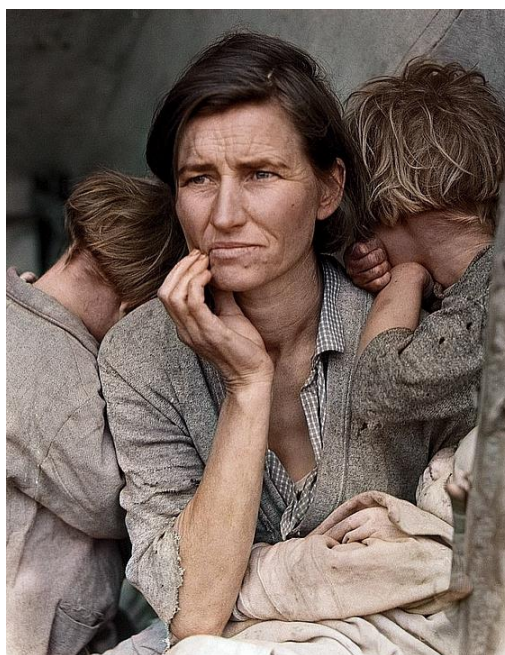The color of the object is not affected by the illumination non-uniformity, and the Retinex filter is based on the consistency of color perception (color constancy), in this way the Retinex filter can balance dynamic range compression, edge enhancement and color constancy, so that it can be used successfully as a pre-filter for the color models implemented in HAVC.

Given that the Retinex filter change in significantly manner the images it is necessary, after having used the filter to colorize the images, 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, as shown in the following image.
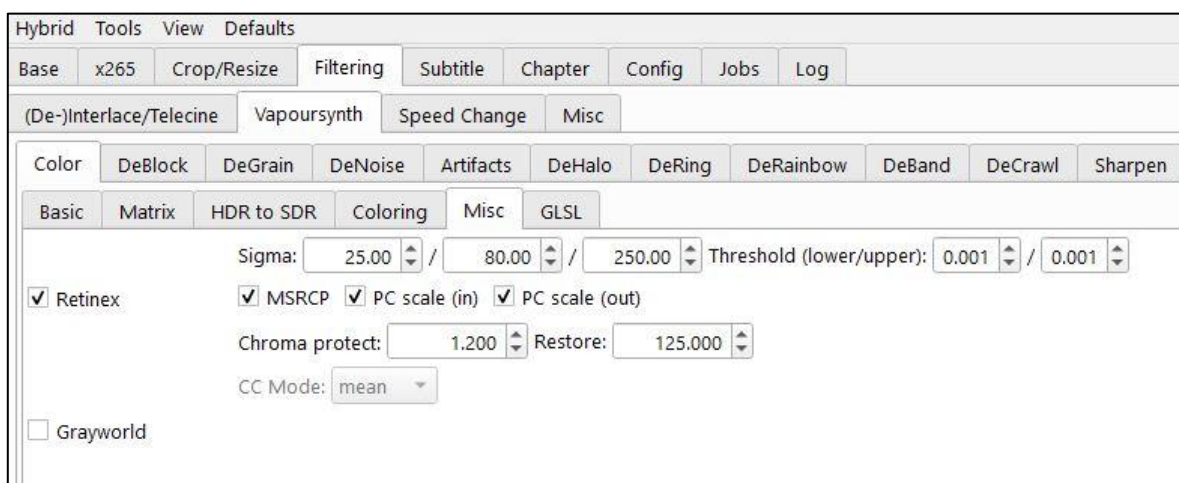
The is necessary to activate the DeOldify filter as usual, but this time is necessary to select the Preset custom and disable the tweaks as shown in the picture below:
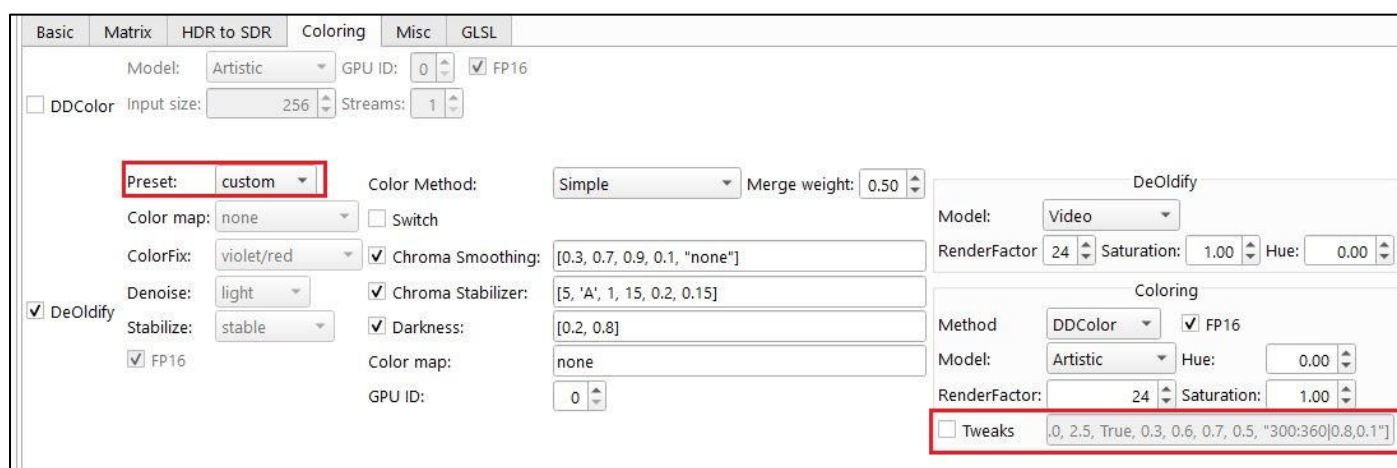
In this case tweaks are not more necessary because the Retinex filter is already changing significantly the luma, gamma and contrast of images. Before using **Retinex** is necessary to remove all the black bars (if are present) as explained at the beginning of this guide.

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.
- open windows file explorer to locate the preview script file and make a copy of it.
- open the copy of the preview script file (the original will be automatically deleted by Hybrid after having closed the Preview Window) with the associated editor "vsViewer.exe".
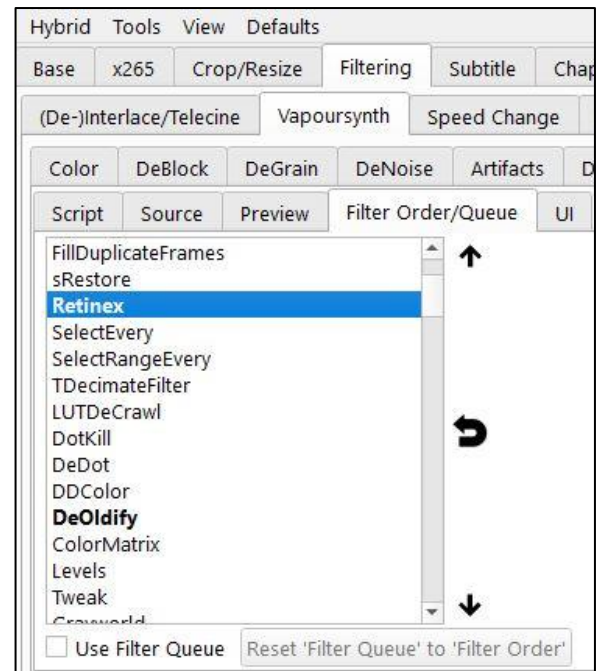
Having opened the copy of the preview file should be visible something like this:

```
File  Edit  Script  Help
27  if validate.primariesIsInvalid(clip):
28    clip = core.std.SetFrameProps(clip=clip, _Primaries=vs.PRIMARIES_BT709)
29  # setting color range to TV (limited) range.
30  clip = core.std.SetFrameProps(clip=clip, _ColorRange=vs.RANGE_LIMITED)
31  # making sure frame rate is set to 23.976fps
32  clip = core.std.AssumeFPS(clip=clip, fpsnum=24000, fpsden=1001)
33  # making sure the detected scan type is set (detected: progressive)
34  clip = core.std.SetFrameProps(clip=clip, _FieldBased=vs.FIELD_PROGRESSIVE) # progressive
35  # adjusting color space from YUV420P8 to YUV444P10 for vsRetinex
36  clip = core.resize.Bicubic(clip=clip, format=vs.YUV444P10, range_s="limited")
37  # color adjustment using Retinex
38  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)
39  # adjusting color space from YUV444P10 to RGB24 for vsDeOldify
40  clip = core.resize.Bicubic(clip=clip, format=vs.RGB24, matrix_in_s="709", range_s="full", dither_type="error_diffusion")
41  # adding colors using DeOldify
42  from vsdeoldify import HAVC_ddeoldify
43  clipRef = HAVC_ddeoldify(clip=clip, method=2, mweight=0.40, deoldify_p=[0, 24, 1, 0], ddcolor_p=[1, 24, 1, 0, True],
    ddtweak=False, sc_threshold=0.10, sc_tht_offset=1, sc_min_freq=1, sc_min_int=1, sc_tht_ssim=0.00, sc_normalize=True,
    device_index=0)
44  from vsdeoldify import HAVC_deepex
45  clip = HAVC_deepex(clip=clip, clip_ref=clipRef, method=0, render_speed="medium", render_vivid=False, ref_merge=1,
    dark=True, dark_p=[0.2, 0.8], smooth=True, smooth_p=[0.3, 0.7, 0.9, 0.1, "none"], colormap="none", sc_framedir=None,
    only_ref_frames=False, ref_weight=0.50, ref_thresh=0.10, ex_model=0, max_memory_frames=0, encode_mode=0)
46  from vsdeoldify import HAVC_stabilizer
47  clip = HAVC_stabilizer(clip=clip, stab=True, stab_p=[5, 'A', 1, 15, 0.2, 0.15], colormap="none", render_factor=24)
48  # adjusting output color from: RGB24 to YUV420P10 for x265Model
49  clip = core.resize.Bicubic(clip=clip, format=vs.YUV420P10, matrix_s="709", range_s="full")
50  # set output frame rate to 23.976fps (progressive)
51  clip = core.std.AssumeFPS(clip=clip, fpsnum=24000, fpsden=1001)
52  # output
```

It is possible to see that at row 38 is called the Retinex filter, and at row 43, 45, and 46 are called the HAVC functions: HAVC_ddeoldify, HAVC_deepex and HAVC_stabilizer. The use exemplar-based models implemented by the function HAVC_deepex with ref_merge > 0 is optional but can be used to improve the video color stability.

Now is necessary to manually change the script to add at row 36 (1), the following code:
```
original_YUV = core.resize.Bicubic(clip=clip, format=vs.YUV420P10, matrix_s="709",
range_s="full")
```

Then at row 36 (2) is necessary to add: `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:


```
File  Edit  Script  Help
33 # making sure the detected scan type is set (detected: progressive)
34 clip = core.std.SetFrameProps(clip=clip, _FieldBased=vs.FIELD_PROGRESSIVE) # progressive
35 # -----------------------------------------------------------------------------
36 original_YUV = core.resize.Bicubic(clip=clip, format=vs.YUV420P10, matrix_s="709", range_s="full")    1
37 # -----------------------------------------------------------------------------
38 # adjusting color space from YUV420P8 to YUV444P10 for vsRetinex
39 clip = core.resize.Bicubic(clip=clip, format=vs.YUV444P10, range_s="limited")
40 # color adjustment using Retinex
41 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)
42 # adjusting color space from YUV444P10 to RGB24 for vsDeOldify
43 clip = core.resize.Bicubic(clip=clip, format=vs.RGB24, matrix_in_s="709", range_s="full", dither_type="error_diffusion")
44 # adding colors using DeOldify
45 from vsdeoldify import HAVC_ddeoldify
46 clipRef = HAVC_ddeoldify(clip=clip, method=2, mweight=0.40, deoldify_p=[0, 24, 1, 0], ddcolor_p=[1, 24, 1, 0, True],
   ddtweak=False, sc_threshold=0.10, sc_tht_offset=1, sc_min_freq=1, sc_min_int=1, sc_tht_ssim=0.00, sc_normalize=True,
   device_index=0)
47 from vsdeoldify import HAVC_deepex
48 clip = HAVC_deepex(clip=clip, clip_ref=clipRef, method=0, render_speed="medium", render_vivid=False, ref_merge=1,
   dark=True, dark_p=[0.2, 0.8], smooth=True, smooth_p=[0.3, 0.7, 0.9, 0.1, "none"], colormap="none", sc_framedir=None,
   only_ref_frames=False, ref_weight=0.50, ref_thresh=0.10, ex_model=0, max_memory_frames=0, encode_mode=0)
49 from vsdeoldify import HAVC_stabilizer
50 clip = HAVC_stabilizer(clip=clip, stab=True, stab_p=[5, 'A', 1, 15, 0.2, 0.15], colormap="none", render_factor=24)
51 # adjusting output color from: RGB24 to YUV420P10 for x265Model
52 clip = core.resize.Bicubic(clip=clip, format=vs.YUV420P10, matrix_s="709", range_s="full")
53 # -----------------------------------------------------------------------------
54 clip = core.std.ShufflePlanes(clips=[original_YUV, clip, clip], planes=[0, 1, 2], colorfamily=vs.YUV)    2
55 # -----------------------------------------------------------------------------
56 # set output frame rate to 23.976fps (progressive)
57 clip = core.std.AssumeFPS(clip=clip, fpsnum=24000, fpsden=1001)
58 # output
```

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)
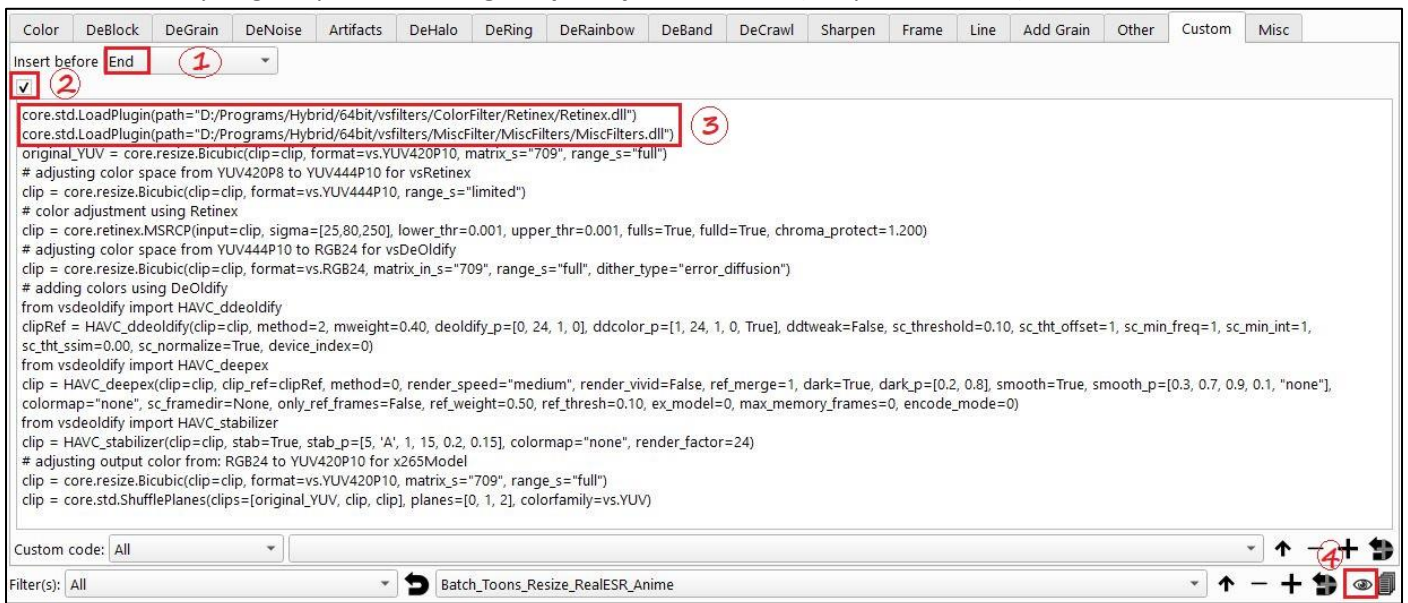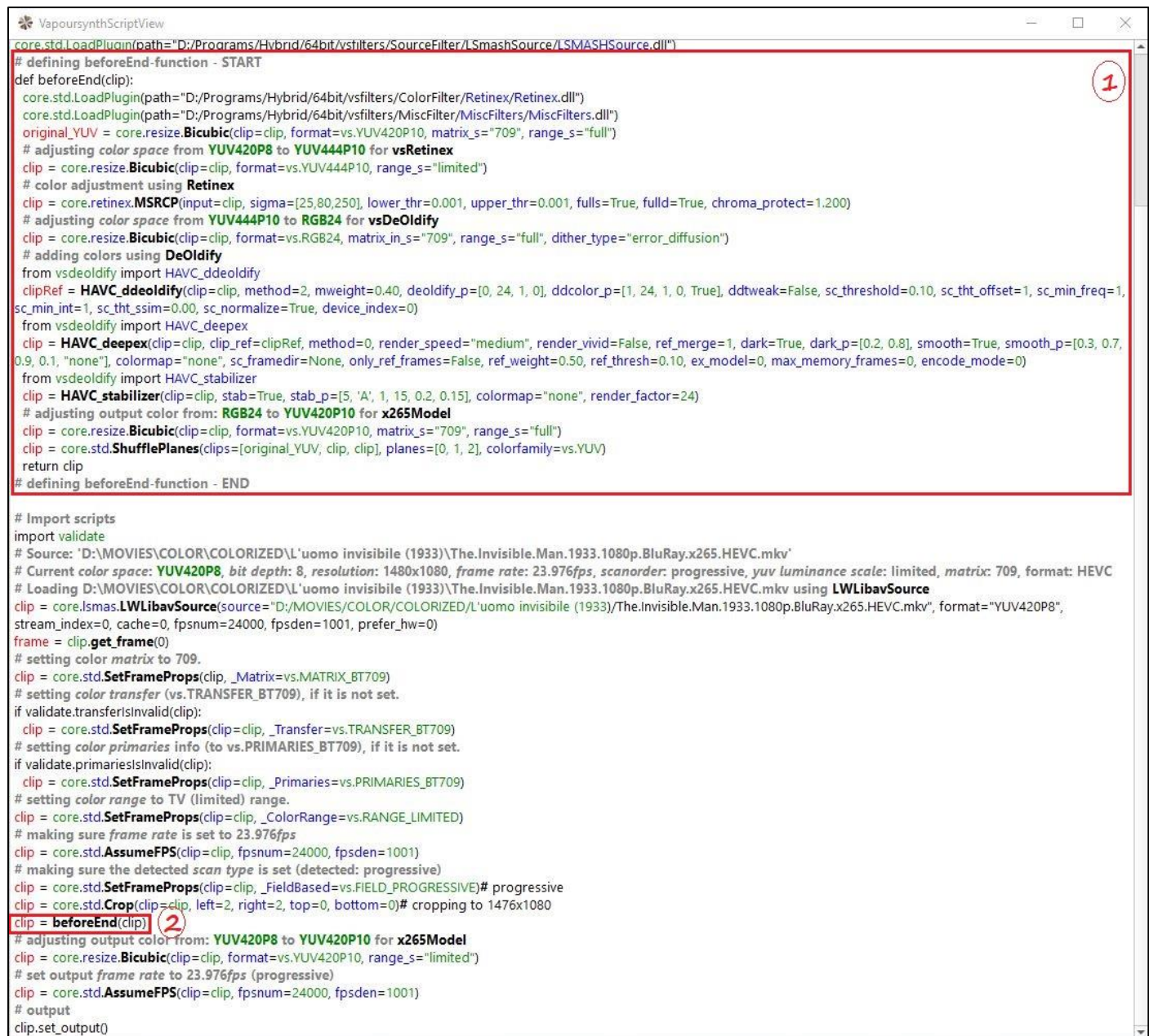

and perform the following selections:

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:
   ```
   core.std.LoadPlugin(path="D:/Programs/Hybrid/64bit/vsfilters/ColorFilter/Retinex/Retinex.dll")
   core.std.LoadPlugin(path="D:/Programs/Hybrid/64bit/vsfilters/MiscFilter/MiscFilters/MiscFilter
   s.dll")
   ```
4) Check by pressing the preview button (4) if the code has been properly inserted in Hybrid

Should be displayed the following Preview window:

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.

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 = 'Fast', VideoTune: str = 'Stable', ColorFix: str = 'Violet/Red',

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

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

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

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

DeepExVivid: bool = True, DeepExEncMode: int = 0, DeepExMaxMemFrames=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', 'Fast' (default), 'Faster', 'VeryFast'.

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: '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'.

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 = HAVC 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 Low (reference frames are merged with low weight), 2 = RF-Merge Med. (reference frames are merged with medium weight), 3 = RF-Merge High (reference frames are merged with high weight).

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 in [0,5]).

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

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%. Range [True, False].

DeepExEncMode: 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.

DeepExMaxMemFrames: Parameter used by ColorMNet model, 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 higher.
>
> 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).

ScFrameDir: if set, define the directory where are stored the reference frames that will be used by Exemplar-based Video Colorization models.

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.

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,

     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, 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).

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 = HAVC 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%. 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 Low (reference frames are merged with low weight), 2 = RF-Merge Med. (reference frames are merged with medium weight), 3 = RF-Merge High (reference frames are merged with high weight).

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: If (ref_merge > 0), represent the threshold used to create the reference frames. If is not set, is assigned automatically a value of 0.10.

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.

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

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.

max_memory_frames: Parameter used by ColorMNet model, 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 higher.

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).

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

6.3 HAVC_ddeoldify

This is the HAVC function that perform the color inference using the single frame-based models: DeOldify and DDColor. 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_ddeoldify(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, 0.15),

     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 = 0.15), 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_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 0.05.

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

6.6 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 0.05.

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

6.7 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

7.0 Useful companion software

To perform advanced coloring could be useful the following software:

7.1 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.

7.2 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.

7.3 Software for renaming a batch of pictures

Sometime it will be 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.