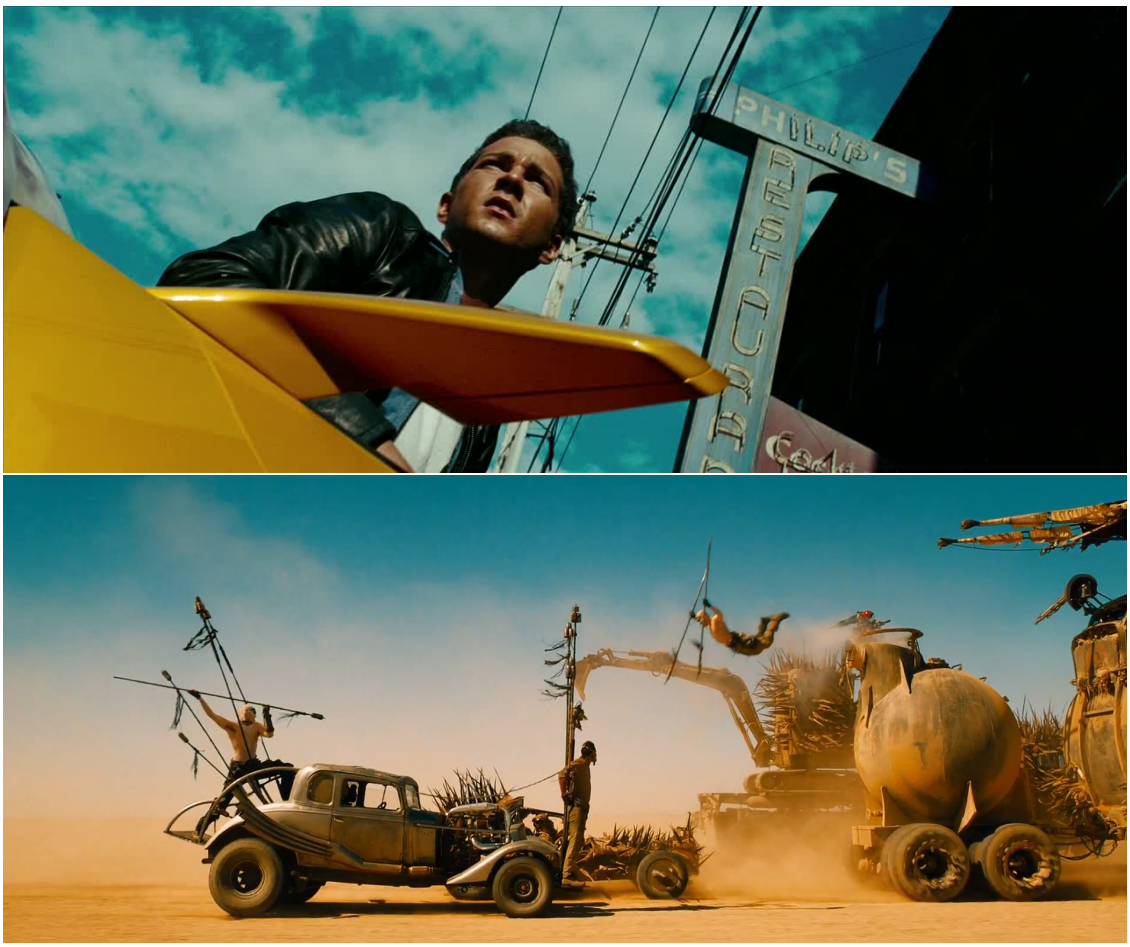
PCM

February 27, 2025

1 1. Introduction

In a world of ubiquitous digital color grading, certain artistic choices can seem to be repeated ad nauseam in blockbuster movies. Specifically, the orange and teal color palette is especially prevalent in the industry. In films like Mad Max: Fury Road and Transformers 2 the color palette is a cornerstone of the film's cinematography.



The goal of this notebook is to research this trend further, to limit the search to more manageable levels, I will be focusing on action movies of the past twenty years. A sample of movies will be taken and analyzed to find if there is a trend in the usage of specific color palettes.

To help us research this question further, I develop image analysis algorithms to extract dominant

colors in film frames. These will be part of a larger video analysis algorithm that will extract the dominant colors of entire films.

Finally, these techniques will be applied to dozens of movie trailers to potentially find patterns in color palette usage.

2 2. Extraction of dominant colors in a video

In order to extract dominant colors from videos we must first be able to extract them from single frames. In this section we take an iterative approach and build an increasingly more sophisticated algorithm.

2.1 References

I took inspiration from several sources during the making of this notebook. Some are cited directly in the paragraphs bellow, but I want to highlight the wikipedia page for K-means clustering, which helped me re-familiarize myself with the technique. Additionally, this blogpost was in interesting read from which I took inspiration and this paper helped me understand types of advanced methods that were at my disposal.

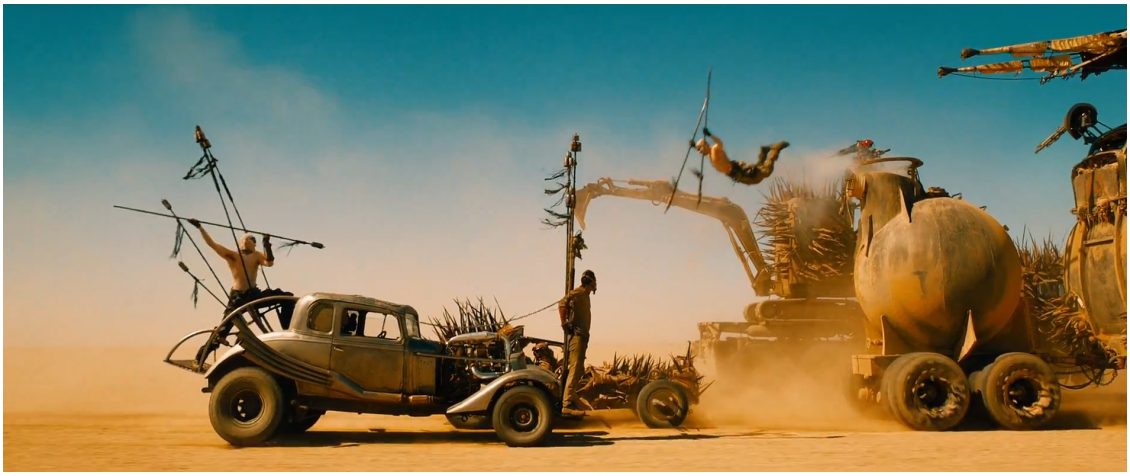
2.2 Extraction of dominant frame colors through average color analysis

Let us create a simple algorithm that receives an image and calculates the average color of all pixels and run it by some sample images.

[2]:
```
from pathlib import Path
from PIL import Image
from IPython.display import display
import numpy as np

notebook_dir = Path().resolve()
image_location = notebook_dir / 'frames'

for image_path in sorted(image_location.glob('*.jpg')):

    # import, open and display images
    img = Image.open(str(image_path))
    display(img)

    # convert the image to a NumPy array and find mean pixel color
    img_array = np.array(img)
    average_color = img_array.mean(axis=(0, 1))
    print(f'Average color (R, G, B): {average_color}')

    # Display average color
    average_color_image = Image.new('RGB', (300, 100), tuple(map(int,␣
 ↪average_color)))
    display(average_color_image)
```

Average color (R, G, B): [49.27523206 47.03413516 40.26889857]

Average color (R, G, B): [139.6036682 55.97235933 21.70142661]



Average color (R, G, B): [127.28436243 126.47306529 100.61911925]

Average color (R, G, B): [71.0555642 89.45905633 68.75083852]

Average color (R, G, B): [53.01252315 14.72578704 13.31689352]

This algorithm is not very accurate. The first two images are rather monochromatic. One is mostly composed of greys and the other of orange tones. In both, the average pixel color is only vaguely representative of the actual most noticeable tonalities in the images. Even worse, when used in pictures with contrasting colors, the average color does not represent any color properly.

The next step is improving the algorithm such that more than one dominant color can be identified. To do this, we will use a k means clustering algorithm. This will allow us to find groups of similar colors and calculate the means of those groups.

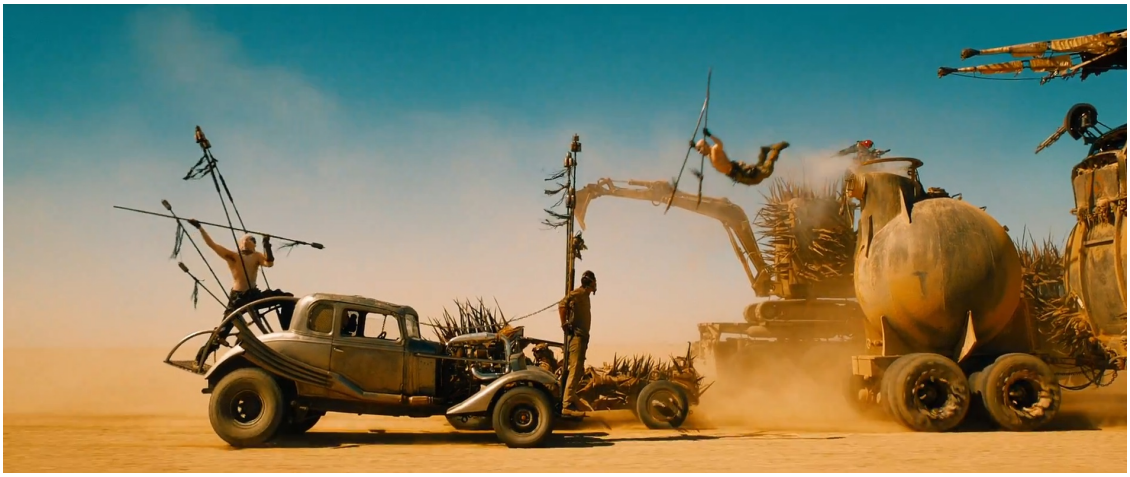
The following code uses this machine learning library and its K-means implementation. It is important to note that this implementation is *not deterministic*, and so results may vary.

[3]:
```
from pathlib import Path
from PIL import Image
from IPython.display import display
import numpy as np
from sklearn.cluster import KMeans # had some trouble importing, numpy must be␣
 ↪v1.x

notebook_dir = Path().resolve()
image_location = notebook_dir / 'frames'

def kmeans(k, image_path, display_frames=True):
    # import, open and display images
    img = Image.open(str(image_path))
    if display_frames:
        display(img)

    # convert image to array of pixels
    pixels = np.array(img).reshape(-1, 3)

    # perform k-means
    kmeans = KMeans(n_clusters=k)
    kmeans.fit(pixels)
    return kmeans.cluster_centers_

def print_colors(colors):
    print(f'Dominant colors (R, G, B): \n {colors}')
```

```
    dominant_color_images = [Image.new('RGB', (300, 100), tuple(map(int,␣
 ↪color))) for color in colors]
    for color_image in dominant_color_images:
        display(color_image)

for image_path in sorted(image_location.glob('*.jpg')):
    print_colors(kmeans(2, image_path))
```

```
Dominant colors (R, G, B):
 [[ 28.61938598  27.56567011  22.67482236]
 [114.65538215 108.65597152  95.95789523]]
```

```
Dominant colors (R, G, B):
 [[ 35.34742828   4.05251119   1.3706596 ]
 [205.99451896  89.03515626  34.64815209]]
```



```
Dominant colors (R, G, B):
 [[ 62.55535679  89.25496558  76.78868886]
 [189.503831    162.248223    123.52564625]]
```


```
Dominant colors (R, G, B):
 [[108.99561623 142.95646785 111.72453951]
 [ 15.09727962  10.55501645   5.3683585 ]]
```

```
Dominant colors (R, G, B):
 [[ 35.29818905  16.91801997  10.70251924]
 [132.122324     4.93557366  24.99233261]]
```

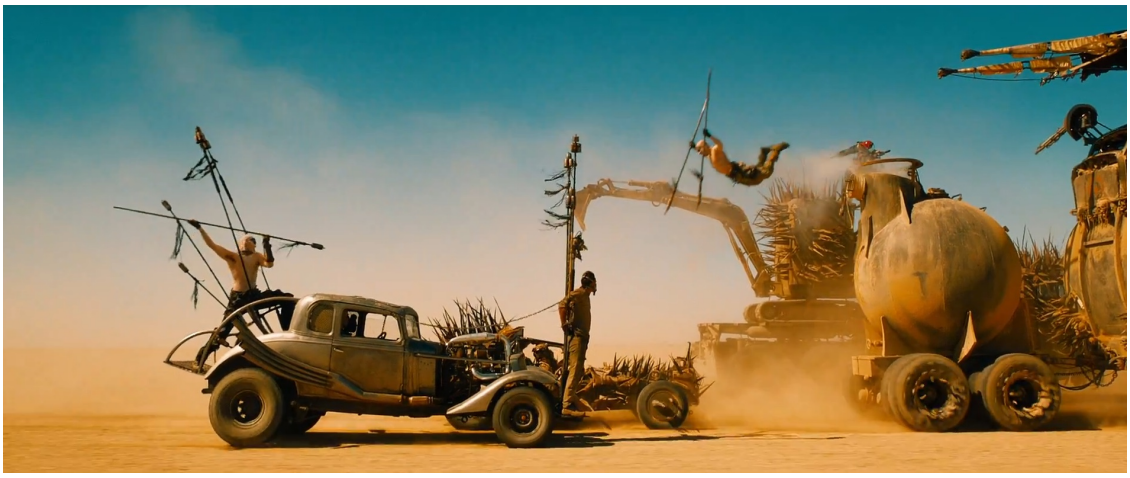
This is slightly better. There were improvements in the monochromatic images, with a clearer and accurate distinction between the shades. However, the bi-color images continue to have inaccurate results, with clearly prominent colors not being detected.

Let us increase the number of clusters used in the algorithm.

[4]:
```
for image_path in sorted(image_location.glob('*.jpg')):
    print_colors(kmeans(5, image_path))
```

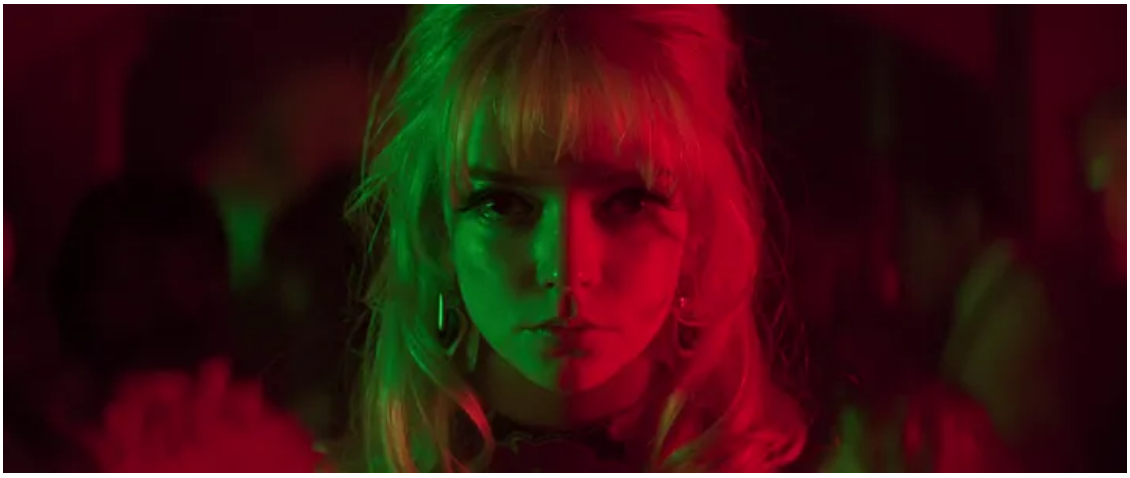
```
Dominant colors (R, G, B):
 [[ 20.67609303  20.74260693  17.75391286]
 [162.10320957 156.97650514 142.36637298]
 [ 80.8331248   74.21761898  60.70930071]
 [ 53.30518611  48.48080871  37.28960993]
 [113.77493748 108.03873298  96.95598777]]
```

```
Dominant colors (R, G, B):
 [[ 25.79008015   1.99061624   1.13366578]
 [215.8087547   80.53798716  21.45958564]
 [237.28867138 238.30460948 236.09351233]
 [243.58778576 134.60919216  62.03063329]
 [153.26953378  43.95170174   7.00787835]]
```



```
Dominant colors (R, G, B):
 [[219.72643858 156.53511872  78.47554096]
 [ 46.61615951 124.77306461 136.15511936]
 [181.12107048 172.62650304 157.11901151]
 [141.62717276  91.17147343  29.04366   ]
 [ 56.11287275  34.30142112   8.6921627 ]]
```


```
Dominant colors (R, G, B):
 [[ 30.66739055 126.28518853 127.02691509]
 [171.17517196 126.36530381  14.56861118]
 [  3.01652086   5.87540183   3.27233265]
 [119.81679624 161.98980379 152.78369515]
 [ 75.15240411  38.09320436  20.63926917]]
```



```
Dominant colors (R, G, B):
 [[ 25.8930584    4.55806635   7.18785503]
 [161.85105791   2.82032294  29.65100223]
 [ 24.02085917  79.47905226  19.14712135]
 [104.06005296   7.29525362  20.53966185]
 [ 50.37297082  12.00758407  12.31821289]]
```

Much better! In the bi-color frames, we now have a color palette representative of the images. All main colors are present, the orange and teal pattern is clear, just like the red and green. It is important to note that adding more clusters did not add much information to the monochromatic frame results. In fact, the white result for the second image might even be misleading.

Since we want to investigate prominent *color* combinations in film, black and white color results do not interest us much. Additionally, 5 clusters does give us all dominant colors in the images, but it also provides several shades of the same color. In order to trim down the results to two main colors, we can eliminate very dark or bright shades, and focus on keeping colors that are substantially different from each other (ex. light and deep red vs yellow and pink).

The following are some functions that will help us implement these restrictions:

[5]:
```
import numpy as np
from scipy.spatial.distance import euclidean

def is_valid_color(color, min_brightness=40, max_brightness=215):
    brightness = np.mean(color)
    return min_brightness <= brightness <= max_brightness

def color_distance(color1, color2):
    return euclidean(color1, color2)

def select_distinct_colors(colors, n=2):
    # filter very dark or bright colors
    valid_colors = [color for color in colors if is_valid_color(color)]

    #if all colors are dark and bright, choose from among them
    if len(valid_colors) < n:
        valid_colors = colors

    # if only two valid colors, return them directly
    if len(valid_colors) == n:
        return valid_colors

    # find the two most distinct colors
```

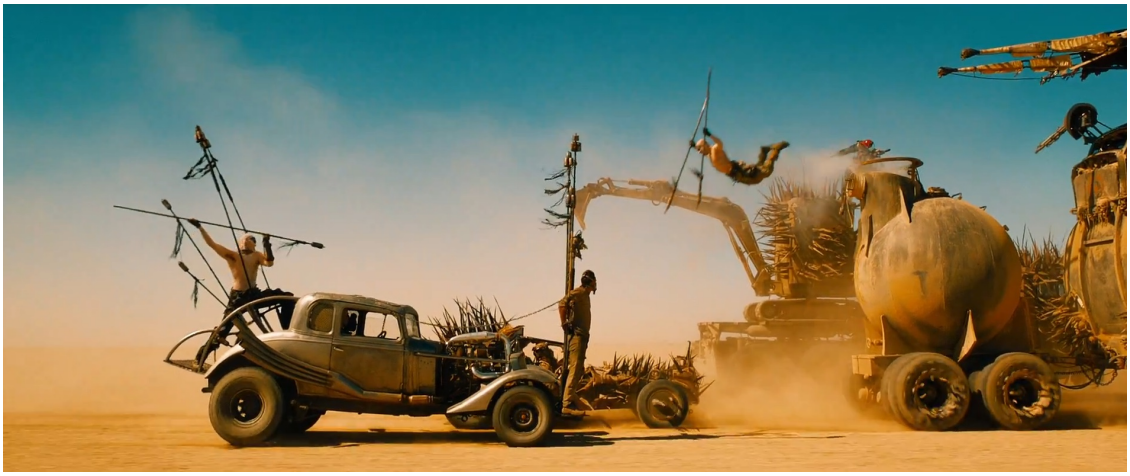
```
    max_distance = 0
    color_pair = None
    for i in range(len(valid_colors)):
        for j in range(i + 1, len(valid_colors)):
            distance = color_distance(valid_colors[i], valid_colors[j])
            if distance > max_distance:
                max_distance = distance
                color_pair = (valid_colors[i], valid_colors[j])

    return color_pair
```

select_distinct_colors() eliminates shades that are too bright or too dark, additionally it performs the selection of the two most distinct colors out of the group. There are three main adjustable parameters: min_brightness, max_brightness and the k value (number of clusters) when the k-means algorithm is called. Feel free to experiment with them. I have found that setting (k=7, min_brightness=40, max_brightness=215) yields good results.

[6]:
```
for image_path in sorted(image_location.glob('*.jpg')):
    print_colors(select_distinct_colors(kmeans(7, image_path)))
```

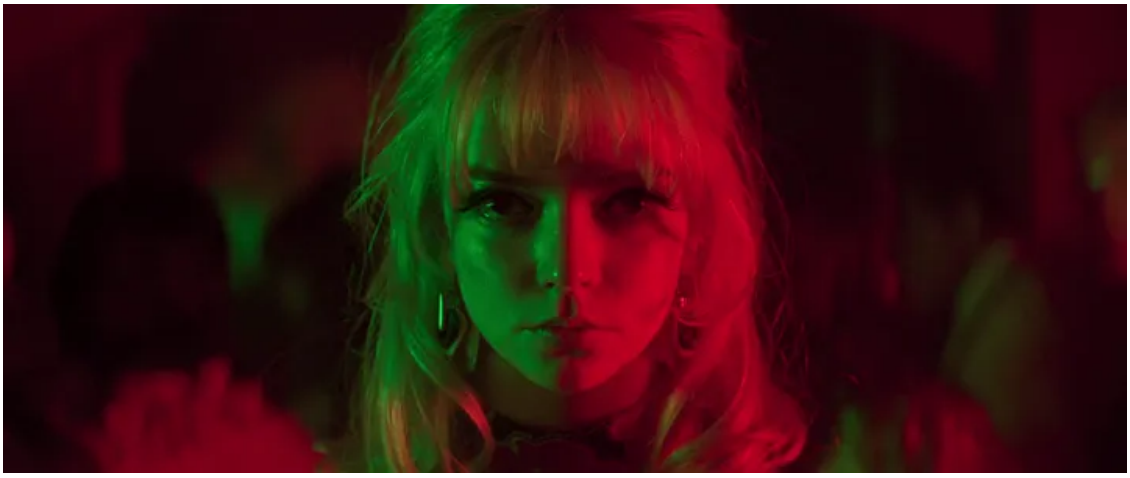
```
Dominant colors (R, G, B):
 (array([47.47643437, 43.35776167, 33.21331662]), array([182.15878576,
176.53998832, 159.73759486]))
```

```
Dominant colors (R, G, B):
 (array([245.15042488, 157.54065377,  82.03784884]), array([150.57504301,
42.52123859,   6.41628512]))
```



```
Dominant colors (R, G, B):
 (array([219.79466662, 156.05268487,  77.10968059]), array([ 23.8223163 ,
117.28496421, 131.39961569]))
```


```
Dominant colors (R, G, B):
 (array([172.92497112, 127.29606646,  10.88053034]), array([ 22.73463726,
125.28776506, 127.54608247]))
```



```
Dominant colors (R, G, B):
 (array([13.9550173 , 92.31392181, 21.53353457]), array([162.59277467,
2.83491483,  29.80593346]))
```

We now have very good representations of the dominant colors in the images. In the bi-color frames the teal and orange patters are obvious, and the red and green one is extremely appropriate, notice how the strongest, most representative, red shade was chosen.

In the monochrome images, the elimination of the brightest and darkest tones is evident, allowing the algorithm to select more midrange tones, which better represent the dominant colors in the image.

2.3 Extraction of the dominant colors of a video

Now that we are able to extract the dominant colors from an image, the next step is doing the same to a full video. To do so, we must be able to extract frames from videos so that individual image analysis may take place.

The following function extracts frames from a certain video, at a specific time interval, and places them in the desired folder.

[21]:
```
import cv2

def extract_frames(video_path, output_folder, frame_interval=5):
    # open video file
    video_capture = cv2.VideoCapture(video_path)
    if not video_capture.isOpened():
        print("Error: Could not open video.")
        return

    # get video fps and find number of frames per time interval
    fps = int(video_capture.get(cv2.CAP_PROP_FPS))
```

```
    interval_frame_number = frame_interval * fps

    frame_count = 0
    saved_frame_count = 0

    # iterate over frames and extract some
    while True:
        success, frame = video_capture.read()
        if not success:
            break

        if frame_count % interval_frame_number == 0:
            frame_filename = output_path / f"frame_{saved_frame_count:04d}.jpg"
            cv2.imwrite(frame_filename, frame)
            saved_frame_count += 1

        frame_count += 1

    video_capture.release()
    print(f"Extracted {saved_frame_count} frames from the video.")


# Example usage
notebook_dir = Path().resolve()
video_path = notebook_dir / 'test_trailer/madmax.mp4'
output_path = notebook_dir / 'test_trailer/frames'
extract_frames(video_path, output_path)
```

```
Extracted 32 frames from the video.
```

This algorithm works well when blindly extracting frames from a video. However, after running it on the test Mad Max trailer, a couple of details become noticeable.

First, some frames are entirely black, or, at least, mostly made up of dark shadows. These should not count towards the color palette of the entire video. We will fix this by finding the mean pixel value of the frame, and discarding the frame if the value is too low.

Secondly, some frames are advertisements for the movie, and not part of the movie themselves. We will correct this by ignoring the first and last 17 seconds of the trailer.

Finally, there are black stripes above and bellow the image, these need to be removed.

The following script implements these changes:

[20]:
```
def frame_is_not_too_dark(frame):
    img_array = np.array(frame)
    average_color = img_array.mean(axis=(0, 1))
    brightness = np.mean(average_color)
    return 20 < brightness
```

```
def cut_out_black_stripes(frame):
    img_array = np.array(frame)

    gray = cv2.cvtColor(img_array, cv2.COLOR_BGR2GRAY)
    _, binary = cv2.threshold(gray, 10, 255, cv2.THRESH_BINARY)

    # find first row from top and bottom that are not black
    top_index = 0
    for i in range(binary.shape[0]):
        if np.sum(binary[i]) > 0:
            top_index = i
            break

    bottom_index = binary.shape[0]
    for i in range(binary.shape[0] - 1, -1, -1):
        if np.sum(binary[i]) > 0:
            bottom_index = i
            break

    # crop the image
    cropped_img = img_array[top_index:bottom_index+1, :]

    return cropped_img

def extract_frames(video_path, output_folder, frame_interval=5):
    # open video file
    video_capture = cv2.VideoCapture(video_path)
    if not video_capture.isOpened():
        print("Error: Could not open video.")
        return

    # get video fps and find number of frames per time interval
    fps = int(video_capture.get(cv2.CAP_PROP_FPS))
    total_frames = int(video_capture.get(cv2.CAP_PROP_FRAME_COUNT))
    duration = total_frames / fps

    interval_frame_number = frame_interval * fps

    frame_count = 0
    saved_frame_count = 0

    start_frame = 15 * fps  # starting frame (first 15 seconds excluded)
    end_frame = (duration - 15) * fps  # ending frame (last 15 seconds excluded)

    # Iterate over frames and extract some
    while True:
        success, frame = video_capture.read()
```

```
        if not success:
            break

        if start_frame <= frame_count < end_frame and frame_count %␣
↪interval_frame_number == 0 and frame_is_not_too_dark(frame):
                frame_filename = Path(output_folder) /␣
↪f"frame_{saved_frame_count:04d}.jpg"
                cv2.imwrite(frame_filename, cut_out_black_stripes(frame))
                saved_frame_count += 1

        frame_count += 1

    video_capture.release()
    print(f"Extracted {saved_frame_count} frames from the video.")

# Example usage
notebook_dir = Path().resolve()
video_path = notebook_dir / 'test_trailer/madmax.mp4'
output_path = notebook_dir / 'test_trailer/frames'
extract_frames(video_path, output_path)
```

```
Extracted 22 frames from the video.
```

This frame selection algorithm now selects a good sample of frames from the video, while ignoring the first and last few seconds, where studio branding, marketing images and other types of media will skew results.

Let us combine it with the image analysis algorithm and check the results!

[12]:
```
notebook_dir = Path().resolve()
video_path = notebook_dir / 'test_trailer/madmax.mp4'
output_path = notebook_dir / 'test_trailer/frames'

extract_frames(video_path, output_path, 8) # frame every 8 seconds

for image_path in sorted(output_path.glob('*.jpg')):
    print_colors(select_distinct_colors(kmeans(5, image_path)))

for image_path in sorted(output_path.glob('*.jpg')):
    print_colors(kmeans(5, image_path))
```

```
Extracted 14 frames from the video.
```

```
Dominant colors (R, G, B):
 (array([217.33694379, 155.86468091, 101.14798009]), array([119.1018    ,
59.69353333,  16.5542    ]))
```

```
Dominant colors (R, G, B):
 (array([101.45952086,  20.45215476,   3.57276607]), array([230.21053804,
139.92480784,  19.24206305]))
```

```
Dominant colors (R, G, B):
 (array([82.60095256, 41.58911889, 16.17314526]), array([199.02072539,
172.04964911, 169.58739424]))
```

```
Dominant colors (R, G, B):
 [array([111.22115112, 150.38502977, 167.49645591]), array([ 34.77030319,
114.0029778 , 128.5468327 ])]
```

```
Dominant colors (R, G, B):
 (array([93.73714323, 60.11793772, 30.68409076]), array([162.58368298,
151.33177933, 148.25827506]))
```

```
Dominant colors (R, G, B):
 (array([244.37281966, 189.74078568, 104.22736235]), array([132.40897619,
57.59058488,  15.78733677]))
```

```
Dominant colors (R, G, B):
 (array([56.66397677, 58.08095299, 69.95106955]), array([208.44056399,
192.57989877, 212.27078814]))
```

```
Dominant colors (R, G, B):
 (array([223.31937442, 196.37545015, 189.33938677]), array([120.94949911,
77.6355153 ,  39.61767531]))
```


```
Dominant colors (R, G, B):
 (array([208.84330666, 180.9553474 , 171.99731721]), array([79.99478677,
40.68318947, 11.14571404]))
```

```
Dominant colors (R, G, B):
 (array([222.11696478, 147.04129402,  87.31092789]), array([116.52124059,
61.12758168,  22.11045106]))
```

```
Dominant colors (R, G, B):
 (array([228.63075975, 181.3923563 , 148.10556326]), array([80.34213266,
36.28327311, 15.48288112]))
```

```
Dominant colors (R, G, B):
 (array([192.67787669, 160.39358721, 133.48269041]), array([106.00330649,
73.4902925 ,  41.90394235]))
```

```
Dominant colors (R, G, B):
 (array([206.24160572, 179.22742132, 175.52354832]), array([109.46990085,
65.59897309,  36.65509915]))
```

```
Dominant colors (R, G, B):
 (array([84.38386748, 53.60622974, 30.72767375]), array([173.07194801,
137.79993811, 109.37459384]))
```

```
Dominant colors (R, G, B):
 (array([223.63290995, 173.79859885, 149.1172182 ]), array([92.10438171,
47.30672003, 21.08773963]))
```

```
Dominant colors (R, G, B):
 (array([84.88417768, 46.75385463, 21.05066079]), array([168.58785269,
164.1705753 , 167.75311469]))
```

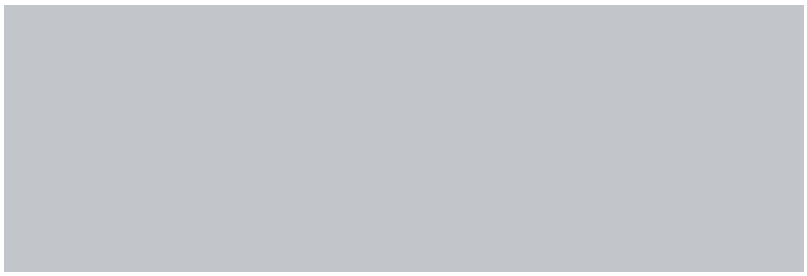
```
Dominant colors (R, G, B):
 (array([ 61.66306584, 136.39866255, 194.36676955]), array([  7.12048568,
61.19209215, 125.02047011]))
```

```
Dominant colors (R, G, B):
 (array([169.0344446 , 161.69327247, 149.39101766]), array([102.89855456,
68.83518421,  36.64242905]))
```

```
Dominant colors (R, G, B):
 (array([69.21640552, 39.23150358, 16.90095465]), array([238.01341238,
178.73875012, 136.50296433]))
```

```
Dominant colors (R, G, B):
 (array([180.84742647, 141.14485294, 102.65955882]), array([68.88473321,
40.29568835, 19.37894929]))
```

```
Dominant colors (R, G, B):
 (array([98.16251503, 57.59520128, 39.33024108]), array([194.51551124,
198.31054131, 203.38398227]))
```



```
Dominant colors (R, G, B):
 (array([242.1639833 , 161.11846551,  50.80938183]), array([144.66056597,
37.41205439,  12.61889011]))
```

```
Dominant colors (R, G, B):
 (array([139.16902763, 152.45710913, 163.35187128]), array([90.97344358,
51.28832685, 24.23472763]))
```

```
Dominant colors (R, G, B):
 [array([  7.44633133,  66.14837717, 131.04694662]), array([ 27.76933896,
112.55399281, 174.54805438])]
```

```
Dominant colors (R, G, B):
 (array([92.41771269, 63.46345886, 36.94588563]), array([155.57399693,
158.15895637, 149.42512607]))
```

```
Dominant colors (R, G, B):
 (array([120.87576773,  72.69430486,  33.87046343]), array([237.01057875,
177.02944418, 134.23590603]))
```

```
Dominant colors (R, G, B):
 (array([176.48661599, 137.78321216, 100.05700595]), array([76.23718636,
65.0542033 , 51.50123311]))
```

```
Dominant colors (R, G, B):
 (array([162.01923942, 184.85233122, 196.71695567]), array([110.64398254,
64.91996449,  43.8184777 ]))
```

```
Dominant colors (R, G, B):
 (array([77.9269174 , 39.86327434, 17.19985251]), array([159.02463964,
137.99637861, 137.10367109]))
```


```
Dominant colors (R, G, B):
 [array([168.56602113,  74.75132042,  17.04137324]), array([221.34342161,
135.54880202,  16.73383775])]
```


```
Dominant colors (R, G, B):
 [array([216.36624929, 137.85590969,  43.04904193]), array([163.25728155,
82.39232127,  16.8592233 ])]
```


```
Dominant colors (R, G, B):
 [array([208.56196902, 129.40404798,  28.45402299]), array([158.15725518,
164.10436026, 179.36812009])]
```

```
Dominant colors (R, G, B):
 [[ 82.11455457  44.78306229  22.37987113]
 [217.89672862 156.4597026  101.42944238]
 [143.48059323  76.48212973  21.64793907]
 [ 26.91515648  17.63372158  13.41253303]
 [195.49442349 122.04798203  41.44234935]]
```

```
Dominant colors (R, G, B):
 [[230.26291132 139.98371445  19.23513664]
 [ 12.11463806  10.20023909   3.72007424]
 [101.75141633  20.48880752   3.58297637]
 [157.32359551  50.43356742   7.97949438]
 [ 53.23245257  18.83401084   3.16246612]]
```

```
Dominant colors (R, G, B):
 [[187.06588999 152.05299847 136.91233766]
 [ 86.09647426  43.78495983  17.1326986 ]
 [168.93260252 107.55399919  62.6591555 ]
 [211.86036088 183.87012987 182.46316515]
 [ 43.68388917  23.75216245  13.88778389]]
```

```
Dominant colors (R, G, B):
 [[ 10.17615788  45.83287926  59.2071234 ]
 [205.73359324 210.70890188 229.68421053]
 [ 34.49385848 112.99799733 127.50293725]
 [110.21107073 150.03214985 167.11601901]
 [  2.84399613  21.9397296   32.67765236]]
```

```
Dominant colors (R, G, B):
 [[ 36.95263401  20.1073079    7.50745973]
 [ 45.26621588 116.19659404 126.16662493]
 [162.69836321 151.32657833 148.19064692]
 [ 93.59873065  59.96859012  30.55145392]
 [ 99.95058979 136.82265116 147.55473068]]
```

```
Dominant colors (R, G, B):
 [[183.39316834 107.66692575  43.55770096]
 [ 88.32649869  36.65316648   9.67785575]
 [115.06449371 148.27265003 159.73059396]
 [244.90954634 186.01433343  91.85555246]
 [169.29849462 158.46913978 161.61591398]]
```

```
Dominant colors (R, G, B):
 [[ 57.91084448  59.15135608  71.09215515]
 [162.64874225 153.53654758 174.38816989]
 [102.07291507  97.52318286 112.94781943]
 [ 18.58541325  19.81441671  27.0163357 ]
 [209.11096237 193.17246765 212.78201696]]
```

```
Dominant colors (R, G, B):
 [[ 95.34741968  57.49406959  25.03660357]
 [204.19882424 162.4391016  131.56270727]
 [ 24.86856156  16.7227544    9.39074278]
 [186.60518732 130.14889529  74.55818581]
 [224.6174672  199.73668586 195.57026658]]
```

```
Dominant colors (R, G, B):
 [[ 78.68163339  39.88427139  10.9215828 ]
 [190.21340426 139.65329787  97.73074468]
 [166.59839768  99.73604689  33.3280998 ]
 [208.83101211 180.90758971 171.91003947]
 [126.27595087  68.10789752  17.25515055]]
```

```
Dominant colors (R, G, B):
 [[147.10709789  76.52455146  25.09222537]
 [ 43.98960271  30.43362154  17.60161979]
 [217.5223003  140.97055309  79.67565371]
 [ 95.53554502  52.79421302  22.36467947]
 [188.46568513 103.95489913  25.05092485]]
```

```
Dominant colors (R, G, B):
 [[ 13.87129148   9.04479525   7.28792762]
 [228.43435733 181.16796576 147.83524377]
 [133.68589564  72.78904135  34.95426118]
 [ 80.35666953  36.2880108   15.48196098]
 [187.0828679  128.5579296   77.94557735]]
```

```
Dominant colors (R, G, B):
 [[156.16131582 158.4517985  149.59102288]
 [ 29.87589674  18.56501021  11.53542441]
 [207.71963366 147.12776188  87.99404694]
 [105.37415882  73.96887618  42.9051144 ]
 [198.96517753 123.22207019  27.319491  ]]
```

```
Dominant colors (R, G, B):
 [[120.92906199 143.1877824  151.49674679]
 [206.23707744 179.21764168 175.49973698]
 [ 40.70632644  18.03646384   9.43983089]
 [108.31151515  64.54839827  35.98164502]
 [200.35564332 135.0565371   74.43234255]]
```

```
Dominant colors (R, G, B):
 [[246.46988631 223.47270676 229.68215742]
 [104.39646109  71.27911453  44.1006145 ]
 [156.34420073 120.74609542  89.83511138]
 [ 43.94940288  25.07979527  13.80034121]
 [206.18744142 201.11648146 211.3057973 ]]
```


```
Dominant colors (R, G, B):
 [[ 23.60374193  16.18663887  10.11820631]
 [190.07787532 131.91985094  88.39032401]
 [223.53830908 173.67897389 148.94348491]
 [ 91.64163204  46.81271972  20.64482795]
 [145.81046687  96.36001506  60.16302711]]
```

```
Dominant colors (R, G, B):
 [[149.57050313  80.77531851  17.31278342]
 [138.97445592 152.39459222 163.39854913]
 [ 78.63581167  43.50724329  20.7645931 ]
 [174.92503278 125.76854145  86.62669387]
 [181.37639446 105.25131598  23.02289081]]
```

```
Dominant colors (R, G, B):
 [[  7.12003269  61.19059666 125.01860429]
 [  3.53108663  11.82991038  31.50569455]
 [ 61.57414058 136.35864546 194.32221652]
 [  4.30632565  32.25784199  75.33723806]
 [ 10.8937574   89.49297919 155.04347826]]
```

```
Dominant colors (R, G, B):
 [[194.42927334 159.92948884 131.23170416]
 [ 29.67029729  18.44719811  11.4899237 ]
 [200.77934944 127.05059335  37.00190718]
 [105.55985205  73.25756557  41.85591796]
 [142.196147   155.9228764  150.50813835]]
```

```
Dominant colors (R, G, B):
 [[238.01185675 178.73798532 136.50269692]
 [ 69.22232606  39.23343718  16.90093458]
 [145.34618236  91.84785026  46.73202372]
 [213.96919991 149.22999764  98.79136181]
 [ 27.03726974  16.14083002   8.38598966]]
```

```
Dominant colors (R, G, B):
 [[ 35.76181803  21.02278889  10.07984581]
 [128.71881275  74.9998254   28.5582715 ]
 [180.66502193 140.96546053 102.45723684]
 [ 79.89176278  70.75963966  57.58609159]
 [ 68.63979378  40.1654348   19.35331691]]
```

```
Dominant colors (R, G, B):
 [[112.25198673  65.8428362   44.37065041]
 [161.96370726 184.87457508 196.7759948 ]
 [ 15.73288946  11.45998925  10.60659195]
 [167.03646446 126.6926567   98.92791128]
 [ 56.70968797  35.56858769  26.63105907]]
```

```
Dominant colors (R, G, B):
 [[131.85351271  77.23824307  39.86087156]
 [ 61.91236852  29.57062024  10.85948495]
 [159.24900815 138.22679074 137.67005699]
 [155.4219697   24.21439394   5.40292208]
 [196.2289114  117.61170606  51.82337996]]
```

```
Dominant colors (R, G, B):
 [[ 93.48615933  52.3229802   23.70451347]
 [115.97586514 143.6321105  160.61042626]
 [  2.65552653   1.86457653   1.19261195]
 [176.74582838 104.60670852  31.86201143]
 [170.56976515 152.46358792 143.77488323]]
```

```
Dominant colors (R, G, B):
 [[ 23.47920328 106.20515524 169.28611599]
 [  3.63561949  12.27220769  32.38307663]
 [  7.2408709   64.56641276 129.26973749]
 [  4.66472831  32.74342573  75.83162112]
 [  1.11966065   1.25873375   1.90364168]]
```

```
Dominant colors (R, G, B):
 [[168.44206223 161.50897116 149.3489439 ]
 [  8.78979505   5.58647181   4.07247066]
 [113.08735632 121.21043324 105.95915119]
 [201.45638888 128.64698915  41.71336523]
 [ 87.74981484  57.11820471  30.353281  ]]
```

```
Dominant colors (R, G, B):
 [[  8.84409177   5.48172362   3.30719876]
 [237.99448102 178.68299768 136.42476762]
 [ 59.71682791  34.18063488  15.17200917]
 [138.56385936  86.25069501  42.96500409]
 [213.01898439 148.51735383  98.04332635]]
```

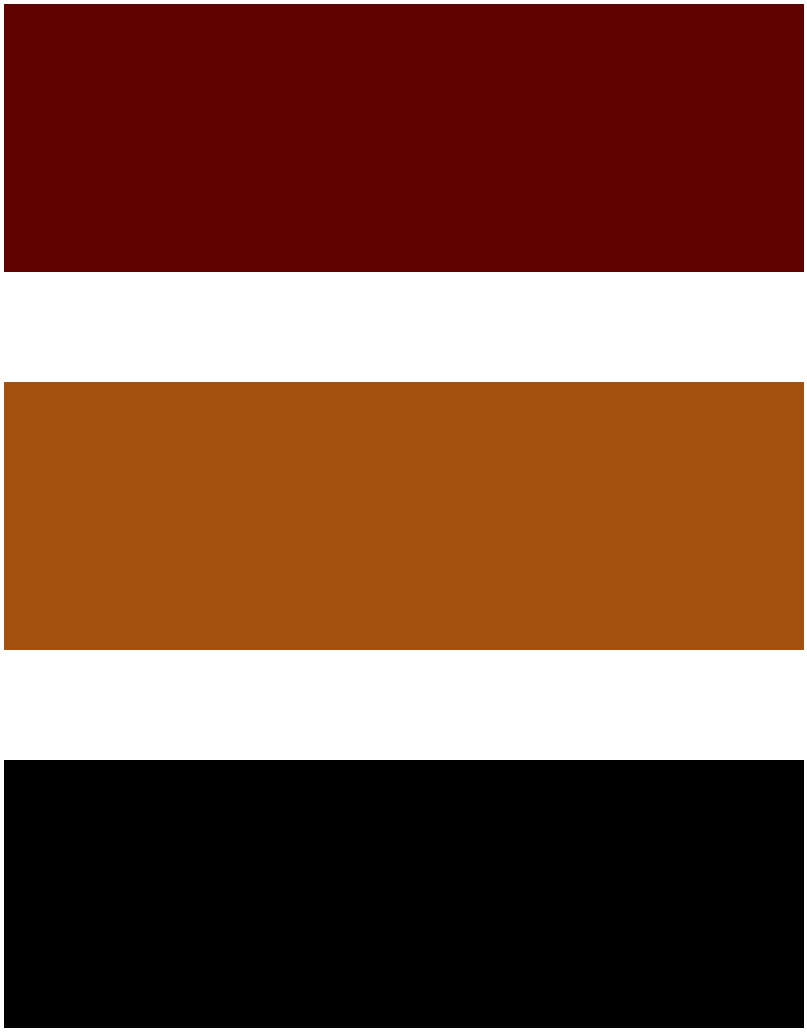
```
Dominant colors (R, G, B):
 [[ 51.82179052  29.35217934  13.07425153]
 [ 76.25191123  64.8001168   51.1210979 ]
 [177.39673358 138.77917659 100.99438585]
 [124.83791272  72.26445009  27.48987917]
 [  4.13473551   2.82120548   1.8062201 ]]
```

```
Dominant colors (R, G, B):
 [[  7.17983585   5.31850287   5.21691201]
 [166.49792806 126.14337809  98.46693187]
 [162.01923942 184.85233122 196.71695567]
 [ 52.14083691  32.88072331  24.91126695]
 [110.69855502  64.94094109  43.82326788]]
```

```
Dominant colors (R, G, B):
 [[  4.06667369   1.99596236   1.74366266]
 [192.79817197 115.63784685  51.3778135 ]
 [152.12480026  37.62496005  12.2464845 ]
 [159.00618381 138.01194115 137.14443102]
 [ 78.10953257  39.96643357  17.24688995]]
```


```
Dominant colors (R, G, B):
 [[ 85.42773335   1.56717201   1.1103838 ]
 [221.35579107 135.56480781  16.73210531]
 [  1.61804143   1.06444444   1.07702448]
 [168.6032967   74.78967033  17.04549451]
 [ 30.25554646   7.88579512   2.84533204]]
```


```
Dominant colors (R, G, B):
 [[ 32.52114912   7.66863741   2.79719864]
 [215.94455335 137.39988799  42.83795389]
 [ 95.48984695   3.51937214   1.25391661]
 [162.19817666  81.37542968  16.41862203]
 [  1.38117389   1.04863475   0.96895091]]
```




```
Dominant colors (R, G, B):
 [[ 88.13540573   1.81036028   1.54507976]
 [  1.29652638   1.32554263   1.07117694]
 [208.54659006 129.38920809  28.44691481]
 [158.15725518 164.10436026 179.36812009]
 [ 31.87873574   8.15156439   3.29611929]]
```

It is extremely interesting to notice that color detection is not being as accurate as previously thought. For example, the 5th image has a strong blue presence, however, the closest picked color is a very light bluish grey. Nonetheless, scrolling down, one can see that the stronger blues were detected by the K-means in that image.

The color detection algorithm seems to be preferring browns, beige and greys, while ignoring other strong colors in the frame. Since we are more interested in detecting those strong colors, which, to the human eye are more prevalent in the frame, we shall make one last modification to the algorithm.

[13]:
```
from scipy.spatial import distance

# Define PST colors (primary, secondary, tertiary colors in RGB format)
pst_colors = [
    (255, 0, 0),    # Primary Red
    (0, 255, 0),    # Primary Green
```

```
    (0, 0, 255),    # Primary Blue
    (255, 255, 0),  # Secondary Yellow
    (0, 255, 255),  # Secondary Cyan
    (255, 0, 255),  # Secondary Magenta
    #(255, 128, 0),  # Tertiary Orange
    #(128, 255, 0),  # Tertiary Chartreuse
    #(0, 255, 128),  # Tertiary Spring Green
    #(0, 128, 255),  # Tertiary Azure
    #(128, 0, 255),  # Tertiary Violet
    #(255, 0, 128),  # Tertiary Rose
]

# UNCOMMENT TO SEE PST COLORS USED
#for color in pst_colors:
#    image = Image.new('RGB', (300, 100), tuple(map(int, color)))
#    display(image)

def select_dominant_colors(input_colors):
    def color_distance(c1, c2):
        # Euclidean distance between two RGB colors
        return distance.euclidean(c1, c2)

    closest = []
    used_pst_indexes = set()

    for input_color in input_colors:
        closest_pst = None
        closest_dist = float('inf')

        for i, pst_color in enumerate(pst_colors):
            if i in used_pst_indexes:
                continue
            dist = color_distance(input_color, pst_color)
            if dist < closest_dist:
                closest_dist = dist
                closest_pst = i

        if closest_pst is not None:
            closest.append((input_color, closest_pst))
            used_pst_indexes.add(closest_pst)

    # Return only the two closest colors
    closest.sort(key=lambda x: color_distance(x[0], pst_colors[x[1]]))
    return [color[0] for color in closest[:2]]
```

The select_dominant_colors function takes an array of input colors and compares each color to a predefined set of primary, secondary, and tertiary (PST) colors. This is based on this website. It

calculates the Euclidean distance between each input color and each PST color to determine the closest matches. The function ensures that the two closest input colors correspond to different PST colors by tracking which PST colors have already been matched. It returns the two input colors that are closest to different PST colors.

After some tweaking I have found better results by only comparing to primary and secondary colors and setting k=6.

The results are marginally better, we can now detect more blues in the frames. However, some blues are still not detected and the results vary depending on the randomness of the K-means algorithm.

[14]:
```
notebook_dir = Path().resolve()
video_path = notebook_dir / 'test_trailer/madmax.mp4'
output_path = notebook_dir / 'test_trailer/frames'

extract_frames(video_path, output_path, 8) # frame every 8 seconds

for image_path in sorted(output_path.glob('*.jpg')):
    print_colors(select_dominant_colors(kmeans(6, image_path)))

for image_path in sorted(output_path.glob('*.jpg')):
    print_colors(kmeans(6, image_path))
```

```
Extracted 14 frames from the video.
```

```
Dominant colors (R, G, B):
 [array([176.94842369, 119.92533966,  75.03442818]), array([223.62925352,
160.91911634, 102.88482095])]
```

```
Dominant colors (R, G, B):
 [array([231.20625645, 141.14460869,  19.20293343]), array([11.24430246,
9.96765907,  3.75635621])]
```

```
Dominant colors (R, G, B):
 [array([200.96049346, 134.98193567,  85.54692319]), array([177.42158476,
153.54490948, 151.41829297])]
```

```
Dominant colors (R, G, B):
 [array([36.52309666, 80.98866553, 94.27887083]), array([206.01013734,
210.93459778, 229.90745585])]
```

```
Dominant colors (R, G, B):
 [array([ 44.18203654, 117.14943811, 127.62065712]), array([74.35271002,
40.8190109 , 15.56424705])]
```

```
Dominant colors (R, G, B):
 [array([227.63971377, 138.07406082,  27.72683363]), array([132.8077179 ,
57.05683078,  15.04313595])]
```

```
Dominant colors (R, G, B):
 [array([170.22440128, 160.21091835, 181.86771639]), array([52.08560292,
53.93783191, 65.65442144])]
```

```
Dominant colors (R, G, B):
 [array([189.61665575, 132.16117056,  75.32954794]), array([224.59129562,
199.64208102, 195.38449274])]
```

```
Dominant colors (R, G, B):
 [array([122.85030701,  66.1083351 ,  17.04171078]), array([208.67244817,
182.32919657, 175.10007974])]
```

```
Dominant colors (R, G, B):
 [array([185.51667797,  98.71423729,  13.86501695]), array([221.64765434,
146.32971614,  86.33434791])]
```

```
Dominant colors (R, G, B):
 [array([235.19666948, 189.2727656 , 158.69962057]), array([199.81996171,
141.16742081,  89.55412461])]
```

```
Dominant colors (R, G, B):
 [array([198.07735939, 122.66370532,  27.28668079]), array([111.00209996,
127.25619488, 115.29105418])]
```

```
Dominant colors (R, G, B):
 [array([108.60718646,  64.60962847,  35.79242979]), array([ 94.78150036,
134.20611799, 145.58612527])]
```

```
Dominant colors (R, G, B):
 [array([129.41261167,  95.06375228,  64.07346691]), array([170.87473557,
135.05152614, 105.60803868])]
```

```
Dominant colors (R, G, B):
 [array([137.34489257,  90.97804372,  57.40539012]), array([228.61677247,
180.3422987 , 157.96316405])]
```

```
Dominant colors (R, G, B):
 [array([124.6044332 ,  70.57468015,  26.03778637]), array([175.72324857,
125.22726943,  84.39230602])]
```

```
Dominant colors (R, G, B):
 [array([ 4.02895248, 31.33328865, 73.860063  ]), array([  8.71068412,
75.24746921, 141.95912051])]
```

```
Dominant colors (R, G, B):
 [array([68.31678314, 44.48419735, 26.15990404]), array([171.89237668,
161.72104699, 148.22660363])]
```

```
Dominant colors (R, G, B):
 [array([221.06145368, 155.11053343, 105.28543205]), array([55.86797027,
32.10353372, 14.64221073])]
```

```
Dominant colors (R, G, B):
 [array([133.24618878, 101.78657152,  73.22315926]), array([198.44248531,
153.92555276, 110.44472432])]
```

```
Dominant colors (R, G, B):
 [array([111.36002754,  65.19259599,  43.89031666]), array([148.16671382,
178.57334842, 193.00735606])]
```

```
Dominant colors (R, G, B):
 [array([210.16129981,  99.46096162,  16.64519924]), array([180.17524491,
110.4501244 ,  58.98382833])]
```

```
Dominant colors (R, G, B):
 [array([174.89012282,  99.09906597,  19.5956461 ]), array([114.20078582,
141.90842755, 158.38713816])]
```

```
Dominant colors (R, G, B):
 [array([ 4.50696232, 32.08420775, 74.79324226]), array([ 10.82178385,
89.54704097, 155.14314618])]
```

```
Dominant colors (R, G, B):
 [array([169.66971365, 161.68351833, 149.17876131]), array([55.92718902,
35.55452298, 20.79248018])]
```

```
Dominant colors (R, G, B):
 [array([157.93221035, 103.24075596,  54.85846343]), array([238.58461458,
179.83591331, 138.0251876 ])]
```

```
Dominant colors (R, G, B):
 [array([49.61306066, 28.07994491, 12.51688724]), array([131.26850672,
99.56673744,  70.1004954 ])]
```

```
Dominant colors (R, G, B):
 [array([148.15532398, 178.55125479, 192.99922019]), array([164.85023349,
124.3478986 ,  96.49899933])]
```

```
Dominant colors (R, G, B):
 [array([183.54312347, 107.65089125,  51.7817825 ]), array([155.96213743,
136.01180899, 137.86707506])]
```


```
Dominant colors (R, G, B):
 [array([224.64886715, 139.61513125,  15.76085438]), array([1.61804143,
1.06444444, 1.07702448])]
```


```
Dominant colors (R, G, B):
 [array([94.73465076,  2.36872359,  1.1790952 ]), array([139.13888228,
58.90463734,   9.46777646])]
```


```
Dominant colors (R, G, B):
 [array([161.72650879, 168.45760122, 183.90068755]), array([139.98871878,
70.58259468,  43.12731668])]
```

```
Dominant colors (R, G, B):
 [[ 26.84282996  17.5940147   13.39016381]
 [218.36948241 156.56591232 101.19493614]
 [141.71440093  69.46276548  11.58759745]
 [209.95548884 129.31967106  31.79480486]
 [ 81.27878162  44.80072277  22.9832731 ]
 [160.75223442 103.4655504   53.71557869]]
```

```
Dominant colors (R, G, B):
 [[ 11.9739585   10.16698875   3.73125297]
 [208.3611188  110.51813902  20.17280532]
 [100.48079072  20.10924839   3.51948868]
 [238.91476892 152.29822505  18.65606162]
 [153.09998555  46.00852478   6.83268314]
 [ 52.49392025  18.81563752   3.14761225]]
```

```
Dominant colors (R, G, B):
 [[ 83.04677689  41.81549322  16.19543006]
 [178.08049678 153.9323827  151.50839466]
 [ 42.24976489  23.16210464  13.7896847 ]
 [199.69311035 133.79445397  84.4366781 ]
 [215.1930605  185.78762074 182.7632181 ]
 [145.67887365  89.40122556  49.44120222]]
```

```
Dominant colors (R, G, B):
 [[  3.19843393  24.41684503  35.45956407]
 [ 93.06267113 138.68938242 154.91238211]
 [217.85507901 219.52641084 238.54085779]
 [159.47123288 176.54246575 194.69917808]
 [ 28.60441455 117.84411105 132.72874634]
 [ 19.93965287  59.6200267   73.68331108]]
```

```
Dominant colors (R, G, B):
 [[ 74.36496404  40.83794203  15.57913856]
 [ 99.93754243 137.71817719 148.98243381]
 [ 31.53211711  17.50151021   6.8092304 ]
 [ 44.18203654 117.14943811 127.62065712]
 [105.10714286  76.73317191  47.69104116]
 [162.91801701 152.21963741 149.77570993]]
```

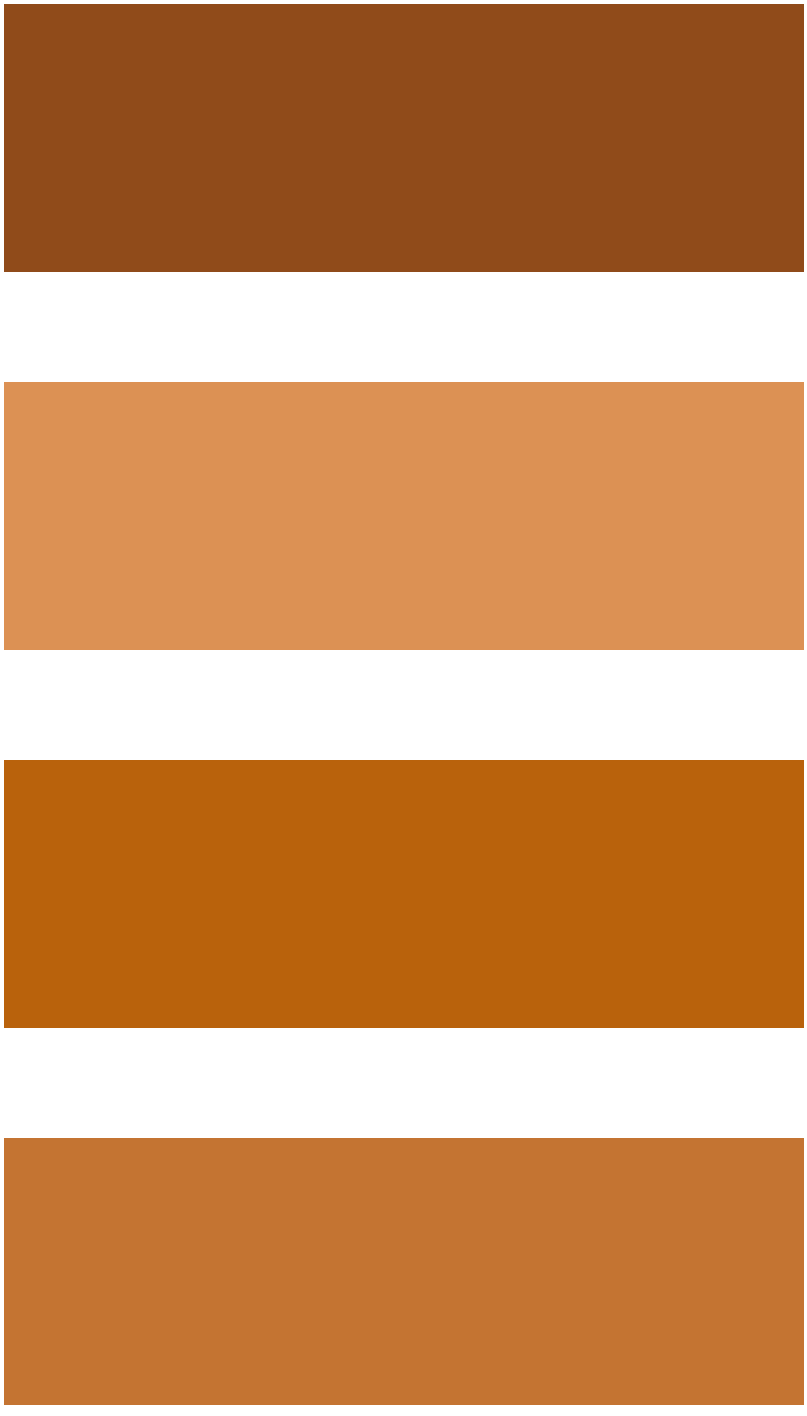
```
Dominant colors (R, G, B):
 [[ 49.08036697  20.50733945   5.55357798]
 [193.00119355 119.24259001  49.95285459]
 [246.89097624 191.3002569   99.99791265]
 [114.85132099 148.11172232 159.56326025]
 [132.48858071  57.48990149  15.61427549]
 [168.48991104 158.39173356 162.05673682]]
```

```
Dominant colors (R, G, B):
 [[ 17.09628647  18.19787798  24.94774536]
 [212.47385763 196.08870199 215.34024105]
 [ 87.24500727  84.16328717  98.71681077]
 [ 52.88162452  54.6792238   66.41367367]
 [128.14316956 122.06210818 138.44015163]
 [172.57862669 162.20299807 184.01953578]]
```

```
Dominant colors (R, G, B):
 [[ 60.81179652  35.72138576  14.51140184]
 [211.49992421 170.0741246  141.4553585 ]
 [189.10254004 133.36230816  78.82199584]
 [ 14.56689216  11.26018896   7.93380668]
 [224.96645501 200.9680585  197.90981977]
 [117.19772967  74.07246568  36.42779831]]
```

```
Dominant colors (R, G, B):
 [[120.42726758  64.73043724  16.95783218]
 [208.70573431 182.21037308 174.82017371]
 [179.50391366 122.09274194  69.49252846]
 [161.71279874  93.99389428  26.75344537]
 [201.60024676 155.01927822 120.88679827]
 [ 75.93708303  37.97885662  10.05983783]]
```

```
Dominant colors (R, G, B):
 [[144.27487542  75.18879601  26.09781386]
 [220.92821615 145.2743015   84.96941971]
 [185.29544316  98.18270263  12.31711853]
 [196.4711841  116.54063387  50.36581997]
 [ 43.08124362  30.02507659  17.4471803 ]
 [ 93.36500342  52.00017088  22.41840396]]
```



```
Dominant colors (R, G, B):
 [[ 10.6462585    7.48766495   6.66855176]
 [199.85616021 141.20835873  89.60339573]
 [117.16092398  52.68858158  20.05740111]
 [235.22255068 189.31271115 158.76097973]
 [158.64961664 104.26929944  61.7108497 ]
 [ 60.74024001  30.93141425  15.37080358]]
```

```
Dominant colors (R, G, B):
 [[209.3395371  148.06669053  88.34693391]
 [ 69.10651134  44.44539214  25.47867693]
 [157.09628574 158.97871203 150.11472343]
 [121.6658511   88.79739019  53.19736689]
 [ 19.73724984  12.45183272   8.19890969]
 [200.65390603 124.12432777  26.69806114]]
```

```
Dominant colors (R, G, B):
 [[108.69530295  64.67941331  35.82870613]
 [210.65601136 181.09766488 176.6304828 ]
 [ 40.80728696  18.09763625   9.47613671]
 [ 94.90458915 134.24886632 145.62252857]
 [200.47953764 135.12194106  74.4116422 ]
 [158.93617021 157.96294077 161.66976326]]
```

```
Dominant colors (R, G, B):
 [[ 39.47302312  22.07205253  12.13788182]
 [207.02316973 202.79380313 214.24006032]
 [131.69815303  97.0182058   65.63254178]
 [246.5982794  223.54644308 229.72832875]
 [ 86.58733048  55.54801129  32.24201329]
 [172.3630635  136.72467281 107.70415253]]
```

```
Dominant colors (R, G, B):
 [[ 23.38179772  16.10830306  10.11623267]
 [181.93578755 123.39700717  78.40754531]
 [210.51105089 156.2793632  124.07799136]
 [ 90.22178131  45.35758377  19.39748677]
 [230.61356874 182.87690934 160.95526681]
 [139.08787441  92.12351137  58.07407073]]
```

```
Dominant colors (R, G, B):
 [[172.66124278  96.82920822  17.32333347]
 [168.85245265 165.38591549 169.8537154 ]
 [ 84.14669904  46.32990942  20.99224951]
 [183.96593378 120.29130095  62.90058225]
 [158.62928392 129.22610755 109.6519922 ]
 [111.67868449 142.43747802 160.41030601]]
```

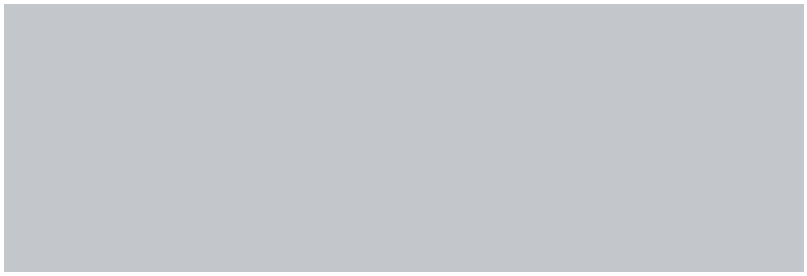
```
Dominant colors (R, G, B):
 [[ 10.94169197  89.91982646 155.41084599]
 [  2.39325153  20.25378323  52.58210634]
 [  7.04415194  61.66575754 125.89039092]
 [  5.21688562  36.053552    81.82696009]
 [ 61.70957775 136.40885685 194.376931  ]
 [  4.03909374   9.82198388  26.06543124]]
```

```
Dominant colors (R, G, B):
 [[ 20.37243361  12.77745077   8.36073155]
 [193.41417879 160.25481457 132.65208982]
 [203.12888741 128.3167027   36.44958146]
 [ 70.27179272  45.78768488  26.61669988]
 [126.03977409  88.54013458  50.4444845 ]
 [141.38877076 156.27893688 151.26983389]]
```

```
Dominant colors (R, G, B):
 [[238.96380015 180.66094108 139.17316623]
 [ 56.39047619  32.31235955  14.65222044]
 [163.98661774 108.91603712  59.298079  ]
 [ 23.04008293  13.80463559   7.31104141]
 [221.99068292 155.99573793 106.36832193]
 [100.36562054  57.1639087   24.74094151]]
```

```
Dominant colors (R, G, B):
 [[162.2351552   97.46390013  27.84075574]
 [ 70.0513501   59.53929148  46.84006462]
 [118.18091537  94.66018998  73.85290731]
 [196.54584282 154.19276765 113.71156036]
 [106.30889649  56.14636441  16.85124038]
 [ 44.23840162  25.03133295  11.24965378]]
```

```
Dominant colors (R, G, B):
 [[111.35671665  65.187806    43.88570991]
 [148.23338267 178.62853316 193.06019595]
 [195.45972037 198.84770306 203.81741012]
 [ 15.58280905  11.3719845   10.5384359 ]
 [165.38366588 124.79455529  96.80267163]
 [ 55.990396    35.15524273  26.38626497]]
```



```
Dominant colors (R, G, B):
 [[155.14752826  23.23176679   5.27466194]
 [156.4466468  136.39590164 138.22019374]
 [ 58.30775405  27.60788249  10.10290576]
 [186.45620892 109.19339329  52.29273608]
 [242.82186985 162.45840198  51.61490939]
 [118.59907662  68.23546294  34.37347143]]
```

```
Dominant colors (R, G, B):
 [[  1.7880314    1.46415346   1.01118353]
 [177.13230294 100.73898764  20.01062597]
 [139.06737869 152.53103024 163.55300633]
 [ 75.98896659  38.02517483  14.596892  ]
 [176.36817428 125.99271442  85.23883719]
```

```
[117.80748493  71.52632414  34.63812242]]
```

```
Dominant colors (R, G, B):
 [[  6.93253983  61.28769438 125.33703206]
 [  3.64756868  12.17501591  32.13006219]
 [ 10.86709567  89.94886315 155.48803089]
 [  4.49931044  32.05372037  74.72666973]
 [  1.11887205   1.25656798   1.90065587]
```

```
[ 62.44566929 136.68818898 194.64619423]]
```

```
Dominant colors (R, G, B):
 [[ 57.64716312  36.58628842  21.16244512]
 [  4.98734      3.19660903   2.76995845]
 [208.8214664  148.13640108  89.19842297]
 [156.37437722 158.72744662 150.0294484 ]
 [200.41303286 124.03265192  27.17491104]
```

```
[117.66370746  84.44261256  49.74910167]]
```

```
Dominant colors (R, G, B):
 [[238.40005105 179.45941807 137.49561001]
 [ 45.28798331  26.58742122  12.88272867]
 [  5.74814132   3.67169769   2.40340433]
 [158.07186512 103.33403583  54.8170706 ]
 [ 89.6603352   50.10837989  20.84123921]
```

[218.33712207 152.73187088 102.49215731]]

```
Dominant colors (R, G, B):
 [[ 38.61973746  22.65831174  10.70487276]
 [140.32549925  99.72843364  59.9391387 ]
 [ 95.90230004  67.43550559  41.3427269 ]
 [  1.51553215   1.11192178   0.96235427]
 [197.09792368 154.09539843 112.38271605]
```

```
[ 67.36153002  42.17927977  22.89283429]]
```

```
Dominant colors (R, G, B):
 [[ 51.57219438  32.546265    24.7003263 ]
 [147.67917544 178.22798868 192.68991798]
 [109.89662789  64.36342218  43.39302017]
 [193.7773791  198.41232449 203.86599064]
 [164.93451091 124.39793368  96.53724379]
```

```
[  7.11368853   5.27478591   5.18230776]]
```

```
Dominant colors (R, G, B):
 [[155.78512905  27.81698075   7.27130274]
 [183.51640229 107.59408013  51.73694195]
 [  4.72542256   2.36181442   1.91134874]
 [156.4331054  136.34160844 138.17087855]
 [ 83.29866889  43.03702163  18.71873267]
```

```
[242.09437711 161.25014902  50.70296046]]
```


```
Dominant colors (R, G, B):
 [[224.63880299 139.60448878  15.75541147]
 [ 30.25348075   7.88586107   2.84554389]
 [  1.61804143   1.06444444   1.07702448]
 [ 85.15438182   1.34631765   1.06310815]
 [195.18508584 103.73310086  20.99114807]
```

```
[153.67709766  59.57152682  14.4590784 ]]
```


```
Dominant colors (R, G, B):
 [[  1.38117389   1.04863475   0.96895091]
 [184.34393838 103.41711319  23.90053583]
 [137.14464023  56.98604993   8.87347039]
 [220.80528676 142.81838565  46.46967194]
 [ 94.66963896   2.26183201   1.17281911]
```

```
[ 32.5089711    7.67255478   2.79862655]]
```


```
Dominant colors (R, G, B):
 [[  1.29596939   1.32528979   1.07121752]
 [ 87.57354926   1.10689816   1.14263469]
 [216.84255195 137.10974539  25.87884109]
 [161.75152905 168.50076453 183.94801223]
 [ 31.84389653   8.11849702   3.25456923]
```

```
[140.4732577   71.31523501  43.36061588]]
```

2.3.1 Dominant colors of a video

To find the average color of the video we will use K-means again. Feeding it with the dominant colors of the collected frames.

[15]:
```
def video_kmeans(color_array, k=5):

    kmeans = KMeans(n_clusters=k)
    kmeans.fit(color_array)
    return kmeans.cluster_centers_
```

[16]:
```
notebook_dir = Path().resolve()
video_path = notebook_dir / 'test_trailer/madmax.mp4'
output_path = notebook_dir / 'test_trailer/frames'

extract_frames(video_path, output_path, 8) # frame every 8 seconds

def get_video_colors(frames_path):
    # dominant_color_array will be an array of all detected dominant colors
    dominant_color_array = []

    for image_path in frames_path.glob('*.jpg'):
        dominant_pair = select_dominant_colors(kmeans(6, image_path, False))
        for color in dominant_pair:
            dominant_color_tuple = tuple(color)
            dominant_color_array.append(dominant_color_tuple)
```

```
        #print_colors(dominant_pair)

    print_colors(video_kmeans(dominant_color_array))

get_video_colors(output_path)
```

```
Extracted 14 frames from the video.
Dominant colors (R, G, B):
 [[111.15074554 148.24415262 165.49643645]
 [ 85.77853347  46.38087797  23.23261934]
 [206.86236692 173.68868438 155.4567054 ]
 [189.88674917 123.99457516  58.65765402]
 [ 30.83006911  82.50398774 126.21071753]]
```

The colors above represent the main groups of colors present in the trailer. As would be expected, brown and beige dirt tones are the majority, but the aforementioned gold and teal tones are also present.

3 3. Applying algorithms to action movies of the past decade

We now have all the tools to investigate color patterns in films over the years. I am particularly curious about the prevalence of the orange and teal pattern. Thus we will perform a small experiment. In the trailers folder I have trailers for films from 2015 to 2023. They were picked at random from the IMDb list of best rated action movies for that year. We will get the dominant colors of the films and see how common gold and teal has been for the past decade in the action genre.

First we will extract the frames from the videos and into their respective folders. After running the script bellow I commented out process_all_trailers. I have manually deleted marketing frames that slipped past the filter. Running the script bellow will overwrite the frame folders and replace the marketing frames.

[17]:
```
notebook_dir = Path().resolve()
trailers_path = notebook_dir / 'trailers'


video_path = notebook_dir / 'test_trailer/madmax.mp4'
output_path = notebook_dir / 'test_trailer/frames'
extract_frames(video_path, output_path)

def process_all_trailers(trailers_dir):
    trailers_dir = Path(trailers_dir)
```

```
    for year_folder in trailers_dir.iterdir():
        if year_folder.is_dir():
            for video_file in year_folder.glob('*.mp4'):
                video_number = video_file.stem  # Get the number from the file␣
↪name (e.g., "1" from "1.mp4")
                output_folder = year_folder / video_number

                if not output_folder.exists():
                    output_folder.mkdir(parents=True, exist_ok=True)

                extract_frames(video_file, output_folder, 2)
                print(f"Processed {video_file} and saved frames to␣
↪{output_folder}")

# Process all videos
#process_all_trailers(trailers_path)
```

```
Extracted 22 frames from the video.
```

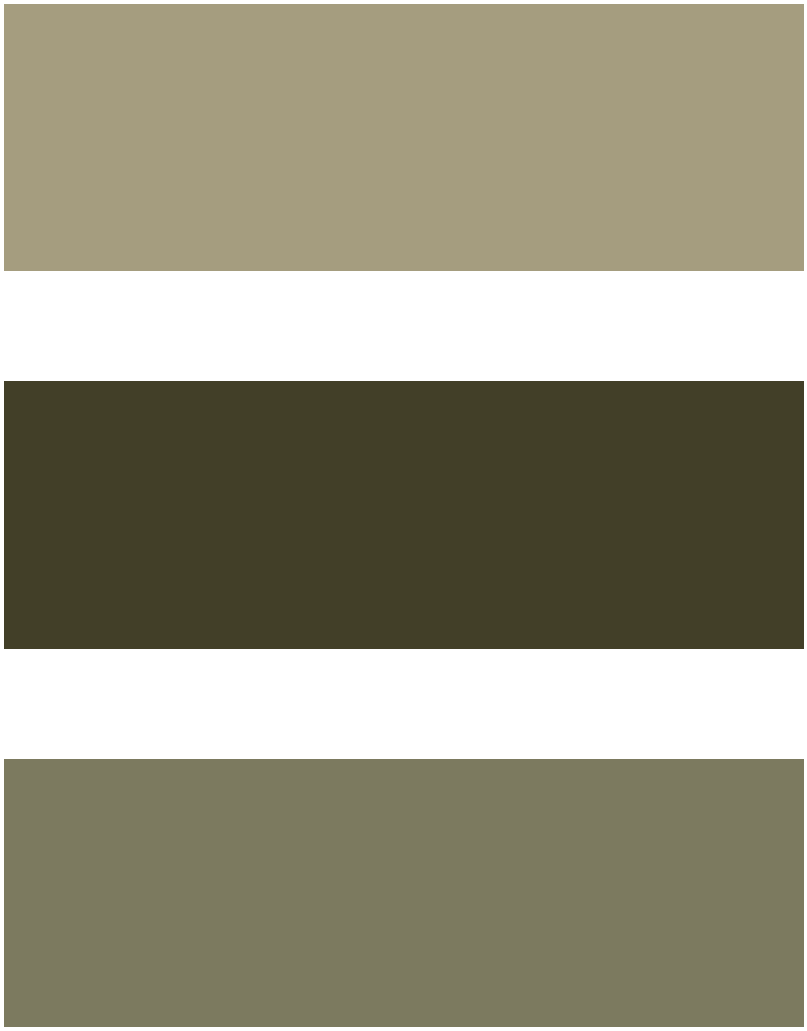
Finally, let us run the color detection on the frames.

[18]:
```
import re

def process_frames_all_directories(trailers_dir):
    trailers_dir = Path(trailers_dir)
    pattern = re.compile(r'\d$')  # Regex pattern to match directories ending␣
↪with a single digit

    for year_folder in trailers_dir.iterdir():
        if year_folder.is_dir() and year_folder.name.isdigit():  # Ensure the␣
↪year folder name is numeric
            year = year_folder.name
            print(f"Processing year: {year}")

            for frame_dir in year_folder.iterdir():
                if frame_dir.is_dir() and pattern.search(frame_dir.name):  #␣
↪Ensure frame_dir ends with a digit
                    # Check the contents of the frame directory
                    frame_files = list(frame_dir.glob('*'))
                    if not frame_files:
                        print(f"No frames found in directory: {frame_dir}")
                        continue

                    print(f"Found {len(frame_files)} frames in directory:␣
↪{frame_dir}")
                    get_video_colors(frame_dir)
```

```
                    print(f"Processed frames in directory: {frame_dir}")

# Set the path to your "trailers" directory
trailers_dir = Path().resolve() / 'trailers'

# Process all frame directories
process_frames_all_directories(trailers_dir)
```

```
Processing year: 2016
Found 47 frames in directory: /home/andre/PCMprojectEE/trailers/2016/1
Dominant colors (R, G, B):
 [[ 99.8764111   91.03676923  97.26932745]
 [156.21512024 161.23837657 168.56691366]
 [ 56.72485586  56.72746052  63.07850971]
 [189.56258809 192.88040461 194.99745168]
 [148.17840269 133.0348157  131.27710045]]
```

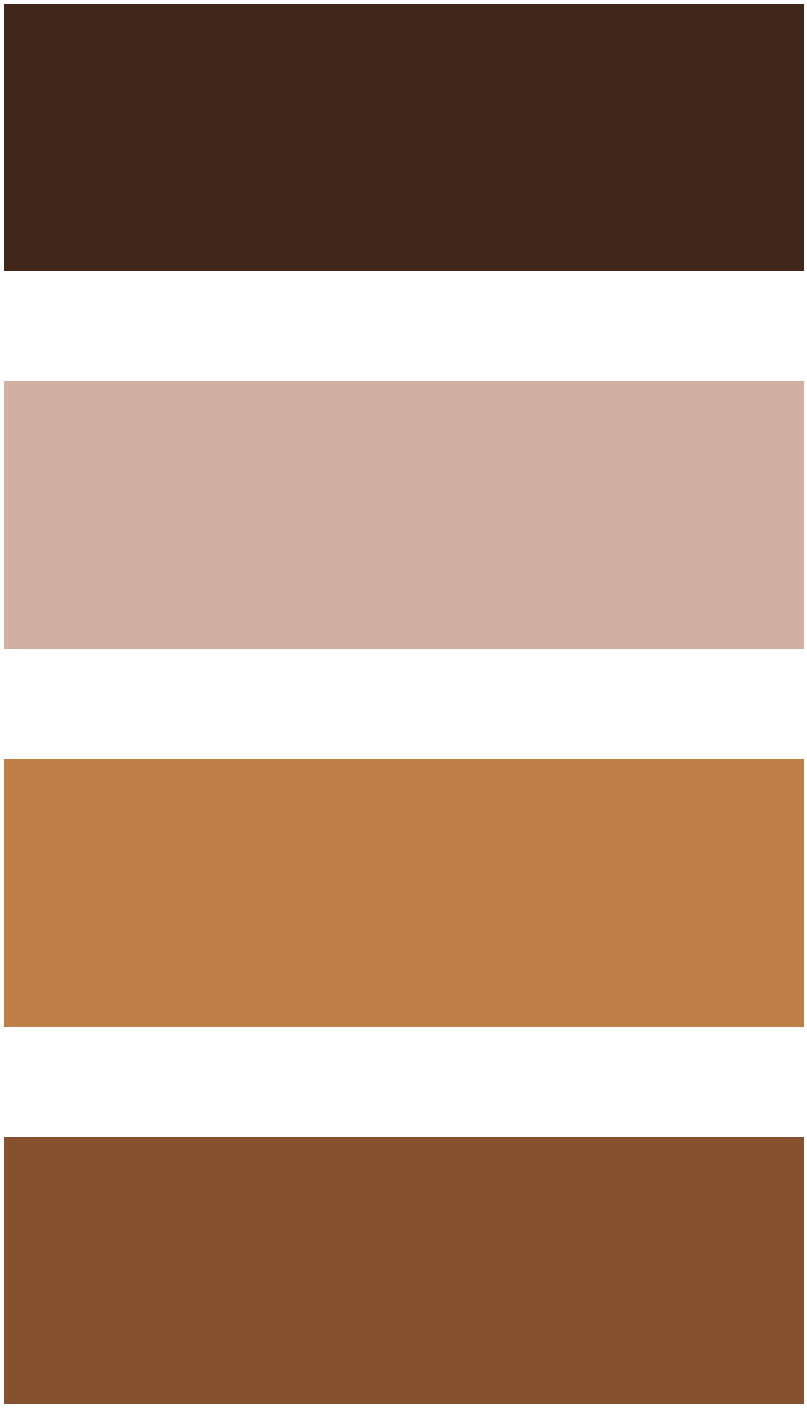
```
Processed frames in directory: /home/andre/PCMprojectEE/trailers/2016/1
Found 51 frames in directory: /home/andre/PCMprojectEE/trailers/2016/2
Dominant colors (R, G, B):
 [[106.75323173  96.41968936  59.98967731]
 [203.99237909 198.14915566 173.02172974]
 [165.39503829 157.51714495 127.77689039]
 [ 66.21473653  63.10403242  40.93373748]
 [124.58613239 122.18017727  95.88584639]]
```



```
Processed frames in directory: /home/andre/PCMprojectEE/trailers/2016/2
Found 34 frames in directory: /home/andre/PCMprojectEE/trailers/2016/3
Dominant colors (R, G, B):
 [[109.01972705 148.89845777 151.99227248]
 [ 84.24901419  78.63396717  70.93049894]
 [205.63027271 186.8233681  173.16927283]
 [ 54.30548438  34.23645899  26.29248835]
 [139.53740137 119.59489136 102.85281483]]
```

```
Processed frames in directory: /home/andre/PCMprojectEE/trailers/2016/3
Processing year: 2015
Found 43 frames in directory: /home/andre/PCMprojectEE/trailers/2015/1
Dominant colors (R, G, B):
 [[216.61992659 205.27533499 175.17858451]
 [103.0087646   93.03155351  73.50749967]
 [156.14840634 141.66807387 113.22950094]
 [ 60.25732361  45.99167768  29.11662075]
 [181.51054136 180.95125498 160.96036849]]
```


```
Processed frames in directory: /home/andre/PCMprojectEE/trailers/2015/1
Found 59 frames in directory: /home/andre/PCMprojectEE/trailers/2015/2
Dominant colors (R, G, B):
 [[101.27533517 139.44443774 151.0550932 ]
 [ 64.53333224  39.94695239  25.86495551]
 [209.08598407 175.24648974 161.0628504 ]
 [191.61964997 128.85928798  72.39965132]
 [133.11608325  81.95875934  46.20575113]]
```



```
Processed frames in directory: /home/andre/PCMprojectEE/trailers/2015/2
Found 53 frames in directory: /home/andre/PCMprojectEE/trailers/2015/3
Dominant colors (R, G, B):
```

```
[[171.68166996 157.79868484 130.45522248]
[114.85227622 101.17275361  78.08158338]
[209.16162965 203.91166313 176.37040053]
[ 64.56826597  54.21764012  40.78129147]
[ 33.87344392 123.4843026  138.71130238]]
```

```
Processed frames in directory: /home/andre/PCMprojectEE/trailers/2015/3
Processing year: 2021
Found 45 frames in directory: /home/andre/PCMprojectEE/trailers/2021/1
Dominant colors (R, G, B):
 [[101.87835033  82.85075159  68.13988691]
 [211.35111984 205.10384292 192.1838482 ]
 [ 47.18606484  41.00246573  37.45526067]
 [198.08552734 179.39512455 139.28379335]
 [152.45694676 141.07202109 127.28990774]]
```

```
Processed frames in directory: /home/andre/PCMprojectEE/trailers/2021/1
Found 49 frames in directory: /home/andre/PCMprojectEE/trailers/2021/2
Dominant colors (R, G, B):
 [[ 79.86635046 107.75558108 113.64963528]
 [196.72392246 196.08479771 183.81708756]
 [ 55.05039226  63.86509837  65.61764015]
 [144.88253634 147.81767547 141.97061507]
 [ 51.37455962 183.07491386 196.8134144 ]]
```

```
Processed frames in directory: /home/andre/PCMprojectEE/trailers/2021/2
Found 57 frames in directory: /home/andre/PCMprojectEE/trailers/2021/3
Dominant colors (R, G, B):
```

```
[[ 60.34481112  51.16671079  45.46015485]
[175.73609332 165.55286474 148.32626694]
[202.41786275 214.93410874 211.51455374]
[123.07760648  94.17522457  71.71456743]
[ 45.13049418 108.2831125  122.44914595]]
```

```
Processed frames in directory: /home/andre/PCMprojectEE/trailers/2021/3
Processing year: 2023
Found 46 frames in directory: /home/andre/PCMprojectEE/trailers/2023/1
Dominant colors (R, G, B):
 [[ 39.00047711  66.03610967  75.11482627]
 [ 99.46035361 127.60125527 139.03405849]
 [175.67851288 175.86703206 171.23763307]
 [177.53498706 146.30207228 110.72368006]
 [ 89.09501399  69.80884323  51.35452127]]
```

```
Processed frames in directory: /home/andre/PCMprojectEE/trailers/2023/1
Found 45 frames in directory: /home/andre/PCMprojectEE/trailers/2023/2
Dominant colors (R, G, B):
 [[219.05281524 157.81608976 102.94460498]
 [ 95.38944552  83.85401724  76.62459355]
 [184.16410484 187.62241793 181.22501751]
 [ 50.41359912  51.89990603  49.04158086]
 [140.28026919 144.43654489 136.99409948]]
```

```
Processed frames in directory: /home/andre/PCMprojectEE/trailers/2023/2
Found 49 frames in directory: /home/andre/PCMprojectEE/trailers/2023/3
Dominant colors (R, G, B):
```

```
[[ 34.81796466 105.33214142 101.23065497]
[229.18898985 212.16140302 170.42982493]
[ 52.12203435  58.99976671  49.35800639]
[153.99585933  87.28146723  42.95218486]
[136.69529686 152.84418335 136.09485826]]
```

```
Processed frames in directory: /home/andre/PCMprojectEE/trailers/2023/3
Processing year: 2020
Found 38 frames in directory: /home/andre/PCMprojectEE/trailers/2020/1
Dominant colors (R, G, B):
 [[ 53.66519377  33.51386373  21.80775713]
 [218.49541562 202.4962784  172.29637068]
 [ 86.27504562  77.2516196   64.82002335]
 [177.88095414 155.78216605 106.46514632]
 [124.43182352 122.92851324  98.16618226]]
```

```
Processed frames in directory: /home/andre/PCMprojectEE/trailers/2020/1
Found 55 frames in directory: /home/andre/PCMprojectEE/trailers/2020/2
Dominant colors (R, G, B):
 [[214.92180743 197.73959762 167.01533381]
 [127.05178618  96.21580396  48.64162887]
 [ 69.65883956  54.09668086  21.10397782]
 [184.81562144 151.13943606 101.38555475]
 [114.67126803 111.87426787  96.06923994]]
```

```
Processed frames in directory: /home/andre/PCMprojectEE/trailers/2020/2
Found 52 frames in directory: /home/andre/PCMprojectEE/trailers/2020/3
Dominant colors (R, G, B):
```

```
[[140.85732239 131.07558549  99.7946259 ]
[ 88.79546447  89.36281653  70.2226716 ]
[196.47176304 200.37644556 172.07637008]
[169.96717343 167.44734413 139.74786939]
[ 49.41053897  46.22417912  33.16489405]]
```

```
Processed frames in directory: /home/andre/PCMprojectEE/trailers/2020/3
Processing year: 2017
Found 57 frames in directory: /home/andre/PCMprojectEE/trailers/2017/1
Dominant colors (R, G, B):
 [[171.16244017 134.03346719 127.00078542]
 [ 68.4220025   53.69180677  50.52726168]
 [118.28671566  96.84131554  93.23780605]
 [207.6998821  179.69391297 172.24778325]
 [105.10964861   3.40142905  13.0696341 ]]
```

```
Processed frames in directory: /home/andre/PCMprojectEE/trailers/2017/1
Found 47 frames in directory: /home/andre/PCMprojectEE/trailers/2017/2
Dominant colors (R, G, B):
 [[138.04243499  77.14851303  16.78256806]
 [192.44629655 185.44918857 164.79747871]
 [189.17905255 137.08788947  70.22023326]
 [ 96.070908    95.24537899  98.85668727]
 [ 53.22250615  41.38895537  29.92871062]]
```

```
Processed frames in directory: /home/andre/PCMprojectEE/trailers/2017/2
Found 32 frames in directory: /home/andre/PCMprojectEE/trailers/2017/3
Dominant colors (R, G, B):
```

```
[[ 51.87206005  50.28022766  39.99516361]
[198.32165361 164.73524687 136.73542354]
[ 93.49196457  91.45963027  80.67832308]
[203.01826923 213.17853082 205.38588765]
[137.38050664 141.72568053 134.56471344]]
```

```
Processed frames in directory: /home/andre/PCMprojectEE/trailers/2017/3
Processing year: 2019
Found 43 frames in directory: /home/andre/PCMprojectEE/trailers/2019/1
Dominant colors (R, G, B):
 [[115.98805788 118.26619091  97.09238384]
 [184.04507733 190.64853034 177.09400094]
 [ 50.64098186  50.7685264   41.79509207]
 [186.31028973 153.86043241  90.69689867]
 [ 80.47989005  83.82768375  71.34615703]]
```

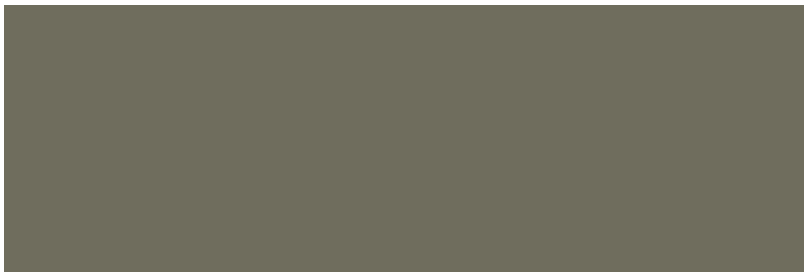
```
Processed frames in directory: /home/andre/PCMprojectEE/trailers/2019/1
Found 40 frames in directory: /home/andre/PCMprojectEE/trailers/2019/2
Dominant colors (R, G, B):
 [[ 83.91865801  90.8500422   90.58876088]
 [190.57944486 196.27755355 190.89734834]
 [154.89923983 151.84306991 147.14091455]
 [ 58.3548558   53.12216038  51.17009897]
 [135.88145453 117.81366555  94.36378162]]
```

```
Processed frames in directory: /home/andre/PCMprojectEE/trailers/2019/2
Found 54 frames in directory: /home/andre/PCMprojectEE/trailers/2019/3
Dominant colors (R, G, B):
```

```
[[ 90.65710503  73.56593348  73.93615002]
[166.60967573 153.11649376 142.44218275]
[213.64357708 195.25704926 198.27385411]
[126.40160165 109.5588959  103.34346024]
[ 52.39850642  40.14924598  38.80938831]]
```

```
Processed frames in directory: /home/andre/PCMprojectEE/trailers/2019/3
Processing year: 2022
Found 56 frames in directory: /home/andre/PCMprojectEE/trailers/2022/1
Dominant colors (R, G, B):
 [[159.0624389  144.55146081 112.62078322]
 [ 61.7887496   63.78779465  55.55363788]
 [212.70019585 224.6554914  196.67534648]
 [115.16627777 101.4208351   79.0888916 ]
 [168.69503378 183.62398231 169.21706113]]
```

```
Processed frames in directory: /home/andre/PCMprojectEE/trailers/2022/1
Found 60 frames in directory: /home/andre/PCMprojectEE/trailers/2022/2
Dominant colors (R, G, B):
 [[188.64149436 198.46698246 192.02952935]
 [ 77.96168096  61.77435244  47.80384083]
 [155.99244497 150.30619294 130.43187029]
 [ 40.07645773  61.40217434  62.13514084]
 [111.04557655 109.5477791   93.76493163]]
```

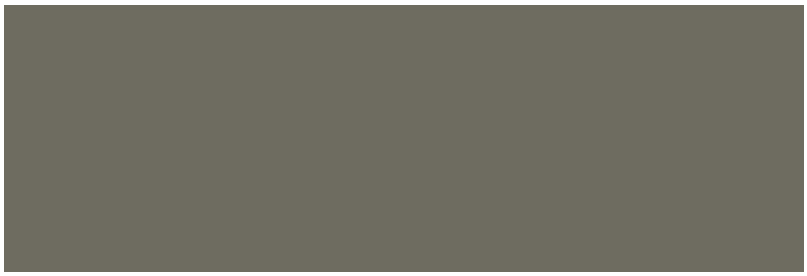


```
Processed frames in directory: /home/andre/PCMprojectEE/trailers/2022/2
Found 63 frames in directory: /home/andre/PCMprojectEE/trailers/2022/3
Dominant colors (R, G, B):
```

```
[[178.89851403 135.22297     66.78036409]
 [ 38.78954708 135.76530708 137.50444331]
 [ 74.31342138  63.39349839  59.02236716]
 [203.68774156 195.94542594 180.59326472]
 [141.34081384 143.01765832 146.71628996]]
```

```
Processed frames in directory: /home/andre/PCMprojectEE/trailers/2022/3
Processing year: 2018
Found 34 frames in directory: /home/andre/PCMprojectEE/trailers/2018/1
Dominant colors (R, G, B):
 [[ 72.87395725  77.95614828  69.76958229]
 [ 83.62590551 151.24738817 155.45551769]
 [132.09778106 122.90707858 102.5823211 ]
 [176.4739321  180.26473196 165.20133315]
 [ 19.98789867  56.06249199  62.30014082]]
```

```
Processed frames in directory: /home/andre/PCMprojectEE/trailers/2018/1
Found 54 frames in directory: /home/andre/PCMprojectEE/trailers/2018/2
Dominant colors (R, G, B):
 [[110.90361292 108.80183556  96.2344816 ]
 [160.90363186 158.85920246 143.4106299 ]
 [204.71854886 200.04974146 185.67855544]
 [ 45.26612877  33.52322781  28.84948075]
 [ 72.66769363  67.77435341  58.40594199]]
```



```
Processed frames in directory: /home/andre/PCMprojectEE/trailers/2018/2
Found 53 frames in directory: /home/andre/PCMprojectEE/trailers/2018/3
Dominant colors (R, G, B):
```

```
[[ 32.64022151  47.79830121  77.20713084]
[111.14309669 140.91776507 169.92082243]
[168.72062112 160.47984892 146.41126072]
[ 90.49340458  89.60312707  86.31996169]
[185.2482799  195.09971867 206.1086779 ]]
```

```
Processed frames in directory: /home/andre/PCMprojectEE/trailers/2018/3
```

4 4. Analysis of results

These results are extremely interesting. First of all, the orange and teal pattern seems to not be as prevalent as once thought. In 27 trailers, only 4 (2022/3, 2023/3, 2023/1, 2015/2) show a palette that could indicate orange and teal color grading, that is 15% of analyzed videos. On the other hand, a surprising number of trailers have green tones (2018/1, 2022/2, 2022/1, 2019/1, 2020/3, 2020/1, 2015/3, 2015/1, 2016/2). That is 9 out of 27 or 33%. Films with overall blue shades are also present, but in less quantity (2018/3, 2018/1, 2023/3, 2021/2). However 4 out of 27 is the same amount as teal and orange. This analysis leads me to believe that, according to this sample, teal and orange is not as prevalent as it might initially seem, with other patterns, like soft greens, having been more popular with filmmakers in the past decade.