IMPORT MODEL WITH PRE-TRAINED WEIGHTS

```
import tensorflow as tf

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import random
import os
import PIL.Image
import cv2
import random
from PIL import Image # Python Image Library is a library that adds support for opening, m
                      # image file formats
```

```
tf.__version__
```

```
'2.4.1'
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
# Load trained inceptionNet model, for more information on Transfer Learning, check previo
base_model = tf.keras.applications.InceptionV3(include_top = False, weights = 'imagenet')
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/i
87916544/87910968 [==============================] - 1s 0us/step
```

GET AN IMAGE AND PRE-PROCESS IT

```
# Open the first image
# Source: https://www.pxfuel.com/en/free-photo-xxgfs
img_1 = Image.open("/content/drive/MyDrive/Colab Notebooks/deep dream/mars.jpg")

# Open the second image
# Source: https://commons.wikimedia.org/wiki/File:Georges_Garen_embrasement_tour_Eiffel.jp
img_2 = Image.open('/content/drive/MyDrive/Colab Notebooks/deep dream/eiffel.jpg')

# Blend the two images

image = Image.blend(img_1, img_2, 0.5) # alpha --> The interpolation alpha factor. If alph
# If alpha is 1.0, a copy of the second image is returned.
```

```
# Save the blended image
image.save("img_0.jpg")
```

```
plt.subplots(figsize=(20, 5))
plt.subplot(1, 2, 1)  # row 1, column 2, count 1
plt.imshow(img_1)
plt.title('Mars Image')

# using subplot function and creating plot two
# row 1, column 2, count 2
plt.subplot(1, 2, 2)

plt.imshow(img_2)
plt.title('Eiffel Tower Image')

# show plot
plt.show()
```


```
# Load the image

Sample_Image = tf.keras.preprocessing.image.load_img('img_0.jpg')
```

```
Sample_Image
```

```
# Get the shape of the image
np.shape(Sample_Image)
```

```
(605, 910, 3)
```

```
# Check out the type of the image
type(Sample_Image)
```

```
PIL.JpegImagePlugin.JpegImageFile
```

```
# Convert to numpy array
Sample_Image = tf.keras.preprocessing.image.img_to_array(Sample_Image)

# Sample_Image = np.array(Sample_Image)
```

```
# Confirm that the image is converted to Numpy array
type(Sample_Image)
```

```
numpy.ndarray
```

```
# Obtain the max and min values
print('min pixel values = {}, max pixel values = {}'.format(Sample_Image.min(), Sample_Ima
```

```
min pixel values = 0.0, max pixel values = 255.0
```

```
# Normalize the input image
Sample_Image = np.array(Sample_Image)/255.0
Sample_Image.shape
```

```
(605, 910, 3)
```

```
# Let's verify normalized images values!
print('min pixel values = {}, max pixel values = {}'.format(Sample_Image.min(), Sample_Ima
```

```
min pixel values = 0.0, max pixel values = 1.0
```

```
Sample_Image = tf.expand_dims(Sample_Image, axis = 0)
```

```
np.shape(Sample_Image)
```

```
TensorShape([1, 605, 910, 3])
```

RUN THE PRETRAINED MODEL AND EXPLORE ACTIVATIONS

NOTES:

- Select a layer and attempt at maximizing the loss which is the activations generated by the layer of interest.
- We can select any layer we choose, early layers generate simple features such as edges and deep layers generate more complex features such as entire face, car or tree.
- Inception network has multiple concatenated layers named 'mixed'

```
base_model.summary()
```

```
Model: "inception_v3"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, None, None,  0
__________________________________________________________________________________________________
conv2d (Conv2D)                 (None, None, None, 3 864         input_1[0][0]
__________________________________________________________________________________________________
batch_normalization (BatchNorma (None, None, None, 3 96          conv2d[0][0]
__________________________________________________________________________________________________
activation (Activation)         (None, None, None, 3 0           batch_normalizat
__________________________________________________________________________________________________
conv2d_1 (Conv2D)               (None, None, None, 3 9216        activation[0][0]
__________________________________________________________________________________________________
batch_normalization_1 (BatchNor (None, None, None, 3 96          conv2d_1[0][0]
__________________________________________________________________________________________________
activation_1 (Activation)       (None, None, None, 3 0           batch_normalizat
__________________________________________________________________________________________________
conv2d_2 (Conv2D)               (None, None, None, 6 18432       activation_1[0][
__________________________________________________________________________________________________
batch_normalization_2 (BatchNor (None, None, None, 6 192         conv2d_2[0][0]
__________________________________________________________________________________________________
activation_2 (Activation)       (None, None, None, 6 0           batch_normalizat
__________________________________________________________________________________________________
max_pooling2d (MaxPooling2D)    (None, None, None, 6 0           activation_2[0][
__________________________________________________________________________________________________
conv2d_3 (Conv2D)               (None, None, None, 8 5120        max_pooling2d[0]
__________________________________________________________________________________________________
```

```
batch_normalization_3 (BatchNor (None, None, None, 8 240         conv2d_3[0][0]
__________________________________________________________________________________
activation_3 (Activation)       (None, None, None, 8 0           batch_normalizat
__________________________________________________________________________________
conv2d_4 (Conv2D)               (None, None, None, 1 138240      activation_3[0][
__________________________________________________________________________________
batch_normalization_4 (BatchNor (None, None, None, 1 576         conv2d_4[0][0]
__________________________________________________________________________________
activation_4 (Activation)       (None, None, None, 1 0           batch_normalizat
__________________________________________________________________________________
max_pooling2d_1 (MaxPooling2D)  (None, None, None, 1 0           activation_4[0][
__________________________________________________________________________________
conv2d_8 (Conv2D)               (None, None, None, 6 12288       max_pooling2d_1[
__________________________________________________________________________________
batch_normalization_8 (BatchNor (None, None, None, 6 192         conv2d_8[0][0]
__________________________________________________________________________________
activation_8 (Activation)       (None, None, None, 6 0           batch_normalizat
__________________________________________________________________________________
conv2d_6 (Conv2D)               (None, None, None, 4 9216        max_pooling2d_1[
__________________________________________________________________________________
conv2d_9 (Conv2D)               (None, None, None, 9 55296       activation_8[0][
__________________________________________________________________________________
batch_normalization_6 (BatchNor (None, None, None, 4 144         conv2d_6[0][0]
__________________________________________________________________________________
batch_normalization_9 (BatchNor (None, None, None, 9 288         conv2d_9[0][0]
__________________________________________________________________________________
activation_6 (Activation)       (None, None, None, 4 0           batch_normalizat
__________________________________________________________________________________
activation_9 (Activation)       (None, None, None, 9 0           batch_normalizat
```

```
# Maximize the activations of these layers

names = ['mixed3', 'mixed5', 'mixed7']

# names = ['mixed3']

layers = [base_model.get_layer(name).output for name in names]

# Create the feature extraction model
deepdream_model = tf.keras.Model(inputs = base_model.input, outputs = layers)
```

```
deepdream_model.summary()
```

```
Model: "model"
__________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================
input_1 (InputLayer)            [(None, None, None,  0
__________________________________________________________________________________
conv2d (Conv2D)                 (None, None, None, 3 864         input_1[0][0]
__________________________________________________________________________________
batch_normalization (BatchNorma (None, None, None, 3 96          conv2d[0][0]
__________________________________________________________________________________
activation (Activation)         (None, None, None, 3 0           batch_normalizat
__________________________________________________________________________________
conv2d_1 (Conv2D)               (None, None, None, 3 9216        activation[0][0]
__________________________________________________________________________________
```

```
batch_normalization_1 (BatchNor (None, None, None, 3 96          conv2d_1[0][0]
__________________________________________________________________________________
activation_1 (Activation)       (None, None, None, 3 0           batch_normalizat
__________________________________________________________________________________
conv2d_2 (Conv2D)               (None, None, None, 6 18432       activation_1[0][
__________________________________________________________________________________
batch_normalization_2 (BatchNor (None, None, None, 6 192         conv2d_2[0][0]
__________________________________________________________________________________
activation_2 (Activation)       (None, None, None, 6 0           batch_normalizat
__________________________________________________________________________________
max_pooling2d (MaxPooling2D)    (None, None, None, 6 0           activation_2[0][
__________________________________________________________________________________
conv2d_3 (Conv2D)               (None, None, None, 8 5120        max_pooling2d[0]
__________________________________________________________________________________
batch_normalization_3 (BatchNor (None, None, None, 8 240         conv2d_3[0][0]
__________________________________________________________________________________
activation_3 (Activation)       (None, None, None, 8 0           batch_normalizat
__________________________________________________________________________________
conv2d_4 (Conv2D)               (None, None, None, 1 138240      activation_3[0][
__________________________________________________________________________________
batch_normalization_4 (BatchNor (None, None, None, 1 576         conv2d_4[0][0]
__________________________________________________________________________________
activation_4 (Activation)       (None, None, None, 1 0           batch_normalizat
__________________________________________________________________________________
max_pooling2d_1 (MaxPooling2D)  (None, None, None, 1 0           activation_4[0][
__________________________________________________________________________________
conv2d_8 (Conv2D)               (None, None, None, 6 12288       max_pooling2d_1[
__________________________________________________________________________________
batch_normalization_8 (BatchNor (None, None, None, 6 192         conv2d_8[0][0]
__________________________________________________________________________________
activation_8 (Activation)       (None, None, None, 6 0           batch_normalizat
__________________________________________________________________________________
conv2d_6 (Conv2D)               (None, None, None, 4 9216        max_pooling2d_1[
__________________________________________________________________________________
conv2d_9 (Conv2D)               (None, None, None, 9 55296       activation_8[0][
__________________________________________________________________________________
batch_normalization_6 (BatchNor (None, None, None, 4 144         conv2d_6[0][0]
__________________________________________________________________________________
batch_normalization_9 (BatchNor (None, None, None, 9 288         conv2d_9[0][0]
__________________________________________________________________________________
activation_6 (Activation)       (None, None, None, 4 0           batch_normalizat
__________________________________________________________________________________
activation_9 (Activation)       (None, None, None, 9 0           batch_normalizat
```

```
# Let's run the model by feeding in our input image and taking a look at the activations "
activations = deepdream_model(Sample_Image)
activations
```

```
[<tf.Tensor: shape=(1, 36, 55, 768), dtype=float32, numpy=
 array([[[[0.        , 0.        , 0.        , ..., 0.02111739,
           0.42713785, 0.30679742],
          [0.        , 0.13667153, 0.        , ..., 0.        ,
           0.        , 0.        ],
          [0.        , 0.25661832, 0.        , ..., 0.        ,
           0.07347696, 0.        ],
          ...,
          [0.        , 0.        , 0.        , ..., 0.        ,
           0.        , 0.        ],
```

```
       [0.        , 0.        , 0.        , ..., 0.42646617,
        0.        , 0.        ],
       [0.05945809, 0.        , 0.        , ..., 0.42646617,
        0.6412622 , 0.        ]],

      [[0.6751412 , 0.        , 0.        , ..., 0.02111739,
        0.28741294, 0.65448624],
       [0.        , 0.        , 0.        , ..., 0.        ,
        0.        , 0.22241248],
       [0.        , 0.        , 0.        , ..., 0.        ,
        0.        , 0.        ],
       ...,
       [0.        , 0.        , 0.        , ..., 0.        ,
        0.        , 0.        ],
       [0.        , 0.        , 0.        , ..., 0.07401298,
        0.        , 0.02371312],
       [0.33377698, 0.        , 0.        , ..., 0.07401298,
        0.13015401, 0.6882228 ]],

      [[0.        , 0.44130546, 0.        , ..., 0.        ,
        0.        , 0.50054014],
       [0.46142423, 0.2867318 , 0.05729262, ..., 0.        ,
        0.        , 0.36714223],
       [0.        , 0.        , 0.        , ..., 0.        ,
        0.        , 0.16450064],
       ...,
       [0.        , 0.        , 0.        , ..., 0.18158405,
        0.        , 0.        ],
       [0.        , 0.        , 0.        , ..., 1.2826656 ,
        0.        , 0.02371312],
       [0.5771486 , 0.        , 0.        , ..., 1.2826656 ,
        0.        , 0.6882228 ]],

      ...,

      [[0.8527035 , 0.        , 0.        , ..., 0.75638807,
        0.45838654, 0.        ],
       [0.        , 0.25250068, 0.        , ..., 0.2939361 ,
        0.15084678, 0.        ],
       [0.        , 0.62371   , 0.        , ..., 0.12852539,
        0.        , 0.        ],
       ...,
       [2.1394928 , 0.        , 0.        , ..., 0.74185884,
        0.2701201 , 0.        ],
       [1.0512463 , 0.        , 0.        , ..., 0.7103812 ,
        0.41722578, 0.        ],
       [0.        , 0.        , 0.        , ..., 1.236742  ,
        1.3979324 , 0.        ]],
```

```
len(activations)
```

```
3
```

- tf.GradientTape() is used to record operations for automatic differentiation
- For example, Let's assume we have the following functions y = x^3.
- The gradient at x = 2 can be computed as follows: dy_dx = 3 * x^2 = 3 * 2^2 = 12.

IMPLEMENT DEEP DREAM ALGORITHM - STEP #1 LOSS CALCULATION

- CREDITS: The DeepDream Code has been adopted from Keras Documentation:
- https://www.tensorflow.org/tutorials/generative/deepdream

```
# Since the cal_closs function includes expand dimension, let's squeeze the image (reduce_
Sample_Image.shape
```

```
TensorShape([1, 605, 910, 3])
```

```
Sample_Image = tf.squeeze(Sample_Image, axis = 0)
```

```
Sample_Image.shape
```

```
TensorShape([605, 910, 3])
```

```
def calc_loss(image, model):
# Function used for loss calculations
# It works by feedforwarding the input image through the network and generate activations
# Then obtain the average and sum of those outputs

  img_batch = tf.expand_dims(image, axis=0) # Convert into batch format
  layer_activations = model(img_batch) # Run the model
  print('ACTIVATION VALUES (LAYER OUTPUT) =\n', layer_activations)
  # print('ACTIVATION SHAPE =\n', np.shape(layer_activations))

  losses = [] # accumulator to hold all the losses
  for act in layer_activations:
    loss = tf.math.reduce_mean(act) # calculate mean of each activation
    losses.append(loss)

  print('LOSSES (FROM MULTIPLE ACTIVATION LAYERS) = ', losses)
  print('LOSSES SHAPE (FROM MULTIPLE ACTIVATION LAYERS) = ', np.shape(losses))
  print('SUM OF ALL LOSSES (FROM ALL SELECTED LAYERS)= ', tf.reduce_sum(losses))

  return  tf.reduce_sum(losses) # Calculate sum
```

```
loss = calc_loss(tf.Variable(Sample_Image), deepdream_model)
```

```
ACTIVATION VALUES (LAYER OUTPUT) =
 [<tf.Tensor: shape=(1, 36, 55, 768), dtype=float32, numpy=
array([[[[0.        , 0.        , 0.        , ..., 0.02111739,
          0.42713785, 0.30679742],
         [0.        , 0.13667153, 0.        , ..., 0.        ,
          0.        , 0.        ],
         [0.        , 0.25661832, 0.        , ..., 0.        ,
          0.07347696, 0.        ],
```

```
 ...,
 [0.        , 0.        , 0.        , ..., 0.        ,
  0.        , 0.        ],
 [0.        , 0.        , 0.        , ..., 0.42646617,
  0.        , 0.        ],
 [0.05945809, 0.        , 0.        , ..., 0.42646617,
  0.6412622 , 0.        ]],

[[0.6751412 , 0.        , 0.        , ..., 0.02111739,
  0.28741294, 0.65448624],
 [0.        , 0.        , 0.        , ..., 0.        ,
  0.        , 0.22241248],
 [0.        , 0.        , 0.        , ..., 0.        ,
  0.        , 0.        ],
 ...,
 [0.        , 0.        , 0.        , ..., 0.        ,
  0.        , 0.        ],
 [0.        , 0.        , 0.        , ..., 0.07401298,
  0.        , 0.02371312],
 [0.33377698, 0.        , 0.        , ..., 0.07401298,
  0.13015401, 0.6882228 ]],

[[0.        , 0.44130546, 0.        , ..., 0.        ,
  0.        , 0.50054014],
 [0.46142423, 0.2867318 , 0.05729262, ..., 0.        ,
  0.        , 0.36714223],
 [0.        , 0.        , 0.        , ..., 0.        ,
  0.        , 0.16450064],
 ...,
 [0.        , 0.        , 0.        , ..., 0.18158405,
  0.        , 0.        ],
 [0.        , 0.        , 0.        , ..., 1.2826656 ,
  0.        , 0.02371312],
 [0.5771486 , 0.        , 0.        , ..., 1.2826656 ,
  0.        , 0.6882228 ]],

...,

[[0.8527035 , 0.        , 0.        , ..., 0.75638807,
  0.45838654, 0.        ],
 [0.        , 0.25250068, 0.        , ..., 0.2939361 ,
  0.15084678, 0.        ],
 [0.        , 0.62371   , 0.        , ..., 0.12852539,
  0.        , 0.        ],
 ...,
 [2.1394928 , 0.        , 0.        , ..., 0.74185884,
  0.2701201 , 0.        ],
 [1.0512463 , 0.        , 0.        , ..., 0.7103812 ,
  0.41722578, 0.        ],
 [0.        , 0.        , 0.        , ..., 1.236742  ,
```

```
loss # Sum up the losses from both activations
```

```
<tf.Tensor: shape=(), dtype=float32, numpy=0.5577076>
```

MINI CHALLENGE #5:

- What is the sum of all losses when 'mixed3' layer is the only layer used for activations

IMPLEMENT DEEP DREAM ALGORITHM - STEP #2 (CALCULATE THE GRADIENT)

- In this step, we will rely on the loss that has been calculated in the previous step and calculate the gradient with respect to the given input image and then add it to the input original image.
- Doing so iteratively will result in feeding images that continiously and increasingly excite the neurons and generate more dreamy like images!

```
# When you annotate a function with tf.function, the function can be called like any other
# The benefit is that it will be compiled into a graph so it will be much faster and could

@tf.function
def deepdream(model, image, step_size):
    with tf.GradientTape() as tape:
      # This needs gradients relative to `img`
      # `GradientTape` only watches `tf.Variable`s by default
      tape.watch(image)
      loss = calc_loss(image, model) # call the function that calculate the loss

    # Calculate the gradient of the loss with respect to the pixels of the input image.
    # The syntax is as follows: dy_dx = g.gradient(y, x)
    gradients = tape.gradient(loss, image)

    print('GRADIENTS =\n', gradients)
    print('GRADIENTS SHAPE =\n', np.shape(gradients))

    # tf.math.reduce_std computes the standard deviation of elements across dimensions of
    gradients /= tf.math.reduce_std(gradients)

    # In gradient ascent, the "loss" is maximized so that the input image increasingly "ex
    # You can update the image by directly adding the gradients (because they're the same
    image = image + gradients * step_size
    image = tf.clip_by_value(image, -1, 1)

    return loss, image
```

```
def run_deep_dream_simple(model, image, steps = 100, step_size = 0.01):
  # Convert from uint8 to the range expected by the model.
  image = tf.keras.applications.inception_v3.preprocess_input(image)

  for step in range(steps):
    loss, image = deepdream(model, image, step_size)
```

```
    if step % 100 == 0:
      plt.figure(figsize=(12,12))
      plt.imshow(deprocess(image))
      plt.show()
      print ("Step {}, loss {}".format(step, loss))

  # clear_output(wait=True)
  plt.figure(figsize=(12,12))
  plt.imshow(deprocess(image))
  plt.show()

  return deprocess(image)
```

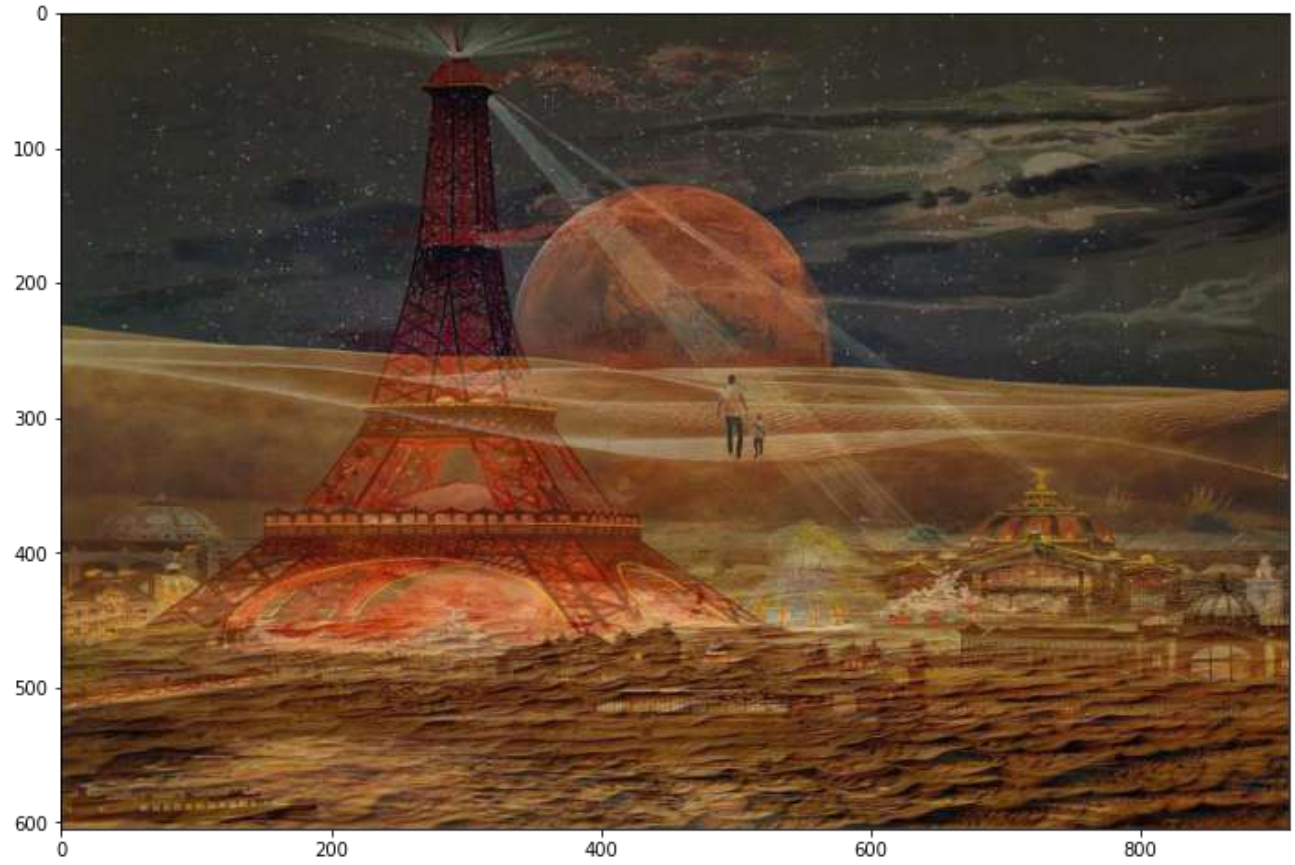
```
def deprocess(image):
  image = 255*(image + 1.0)/2.0
  return tf.cast(image, tf.uint8)
```

```
Sample_Image.shape
```

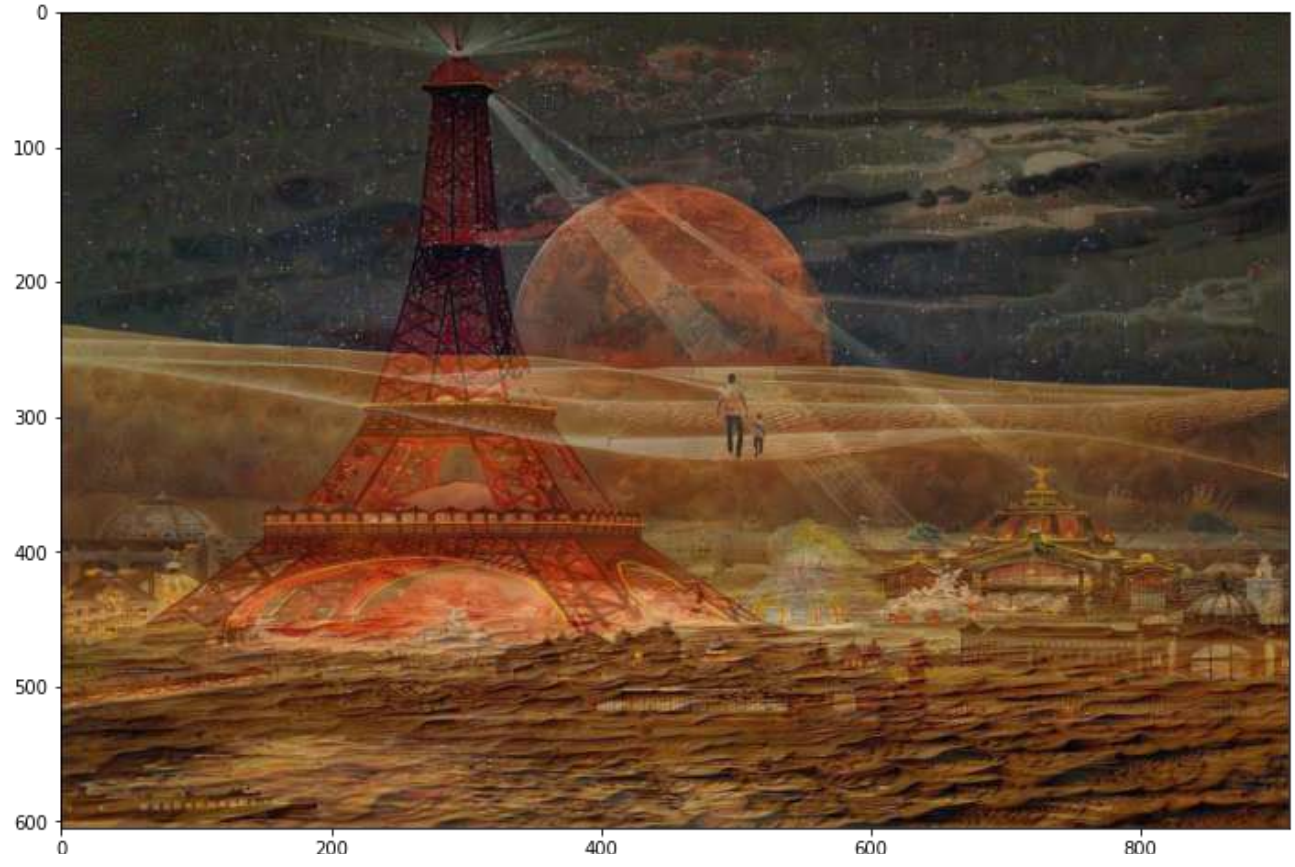
```
TensorShape([605, 910, 3])
```

```
# Let's Load the image again and convert it to Numpy array
Sample_Image = np.array(tf.keras.preprocessing.image.load_img('img_0.jpg'))
dream_img = run_deep_dream_simple(model = deepdream_model, image = Sample_Image, steps = 4
```

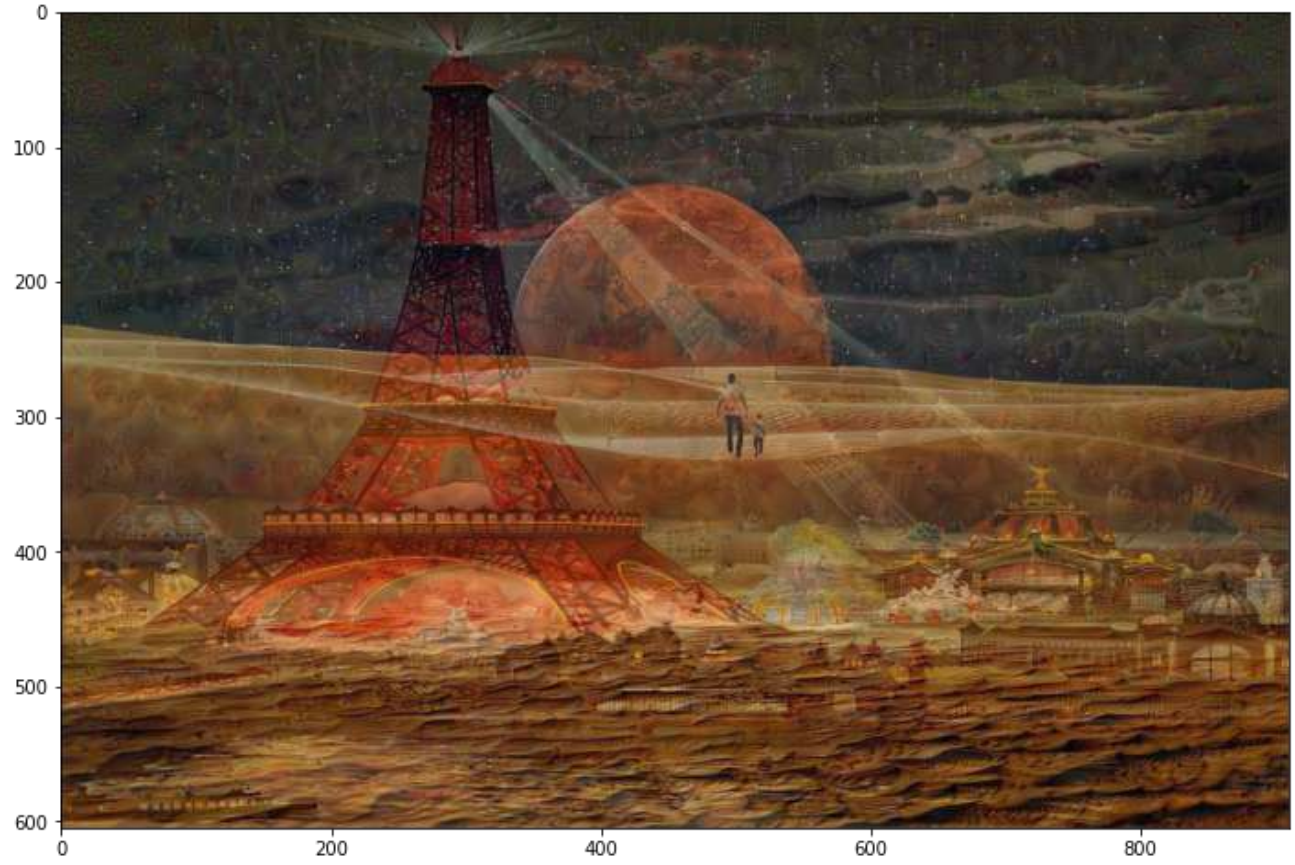
```
ACTIVATION VALUES (LAYER OUTPUT) =
 [<tf.Tensor 'model/mixed3/concat:0' shape=(1, 36, 55, 768) dtype=float32>, <tf.Tensc
LOSSES (FROM MULTIPLE ACTIVATION LAYERS) =  [<tf.Tensor 'Mean:0' shape=() dtype=float
LOSSES SHAPE (FROM MULTIPLE ACTIVATION LAYERS) =  (3,)
SUM OF ALL LOSSES (FROM ALL SELECTED LAYERS)=  Tensor("Sum:0", shape=(), dtype=float3
GRADIENTS =
 Tensor("gradient_tape/Reshape_4:0", shape=(605, 910, 3), dtype=float32)
GRADIENTS SHAPE =
 (605, 910, 3)
```



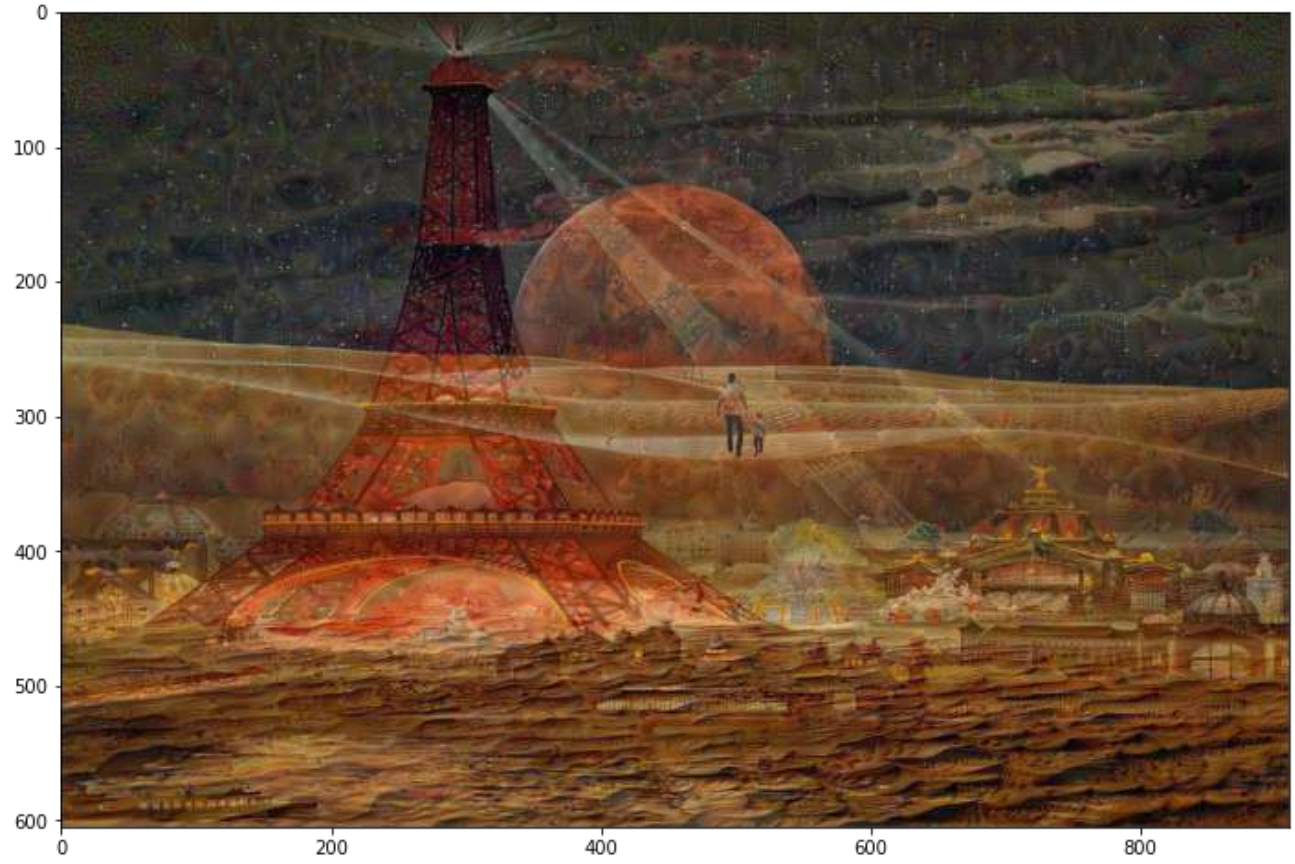

```
Step 0, loss 0.5804305076599121
```



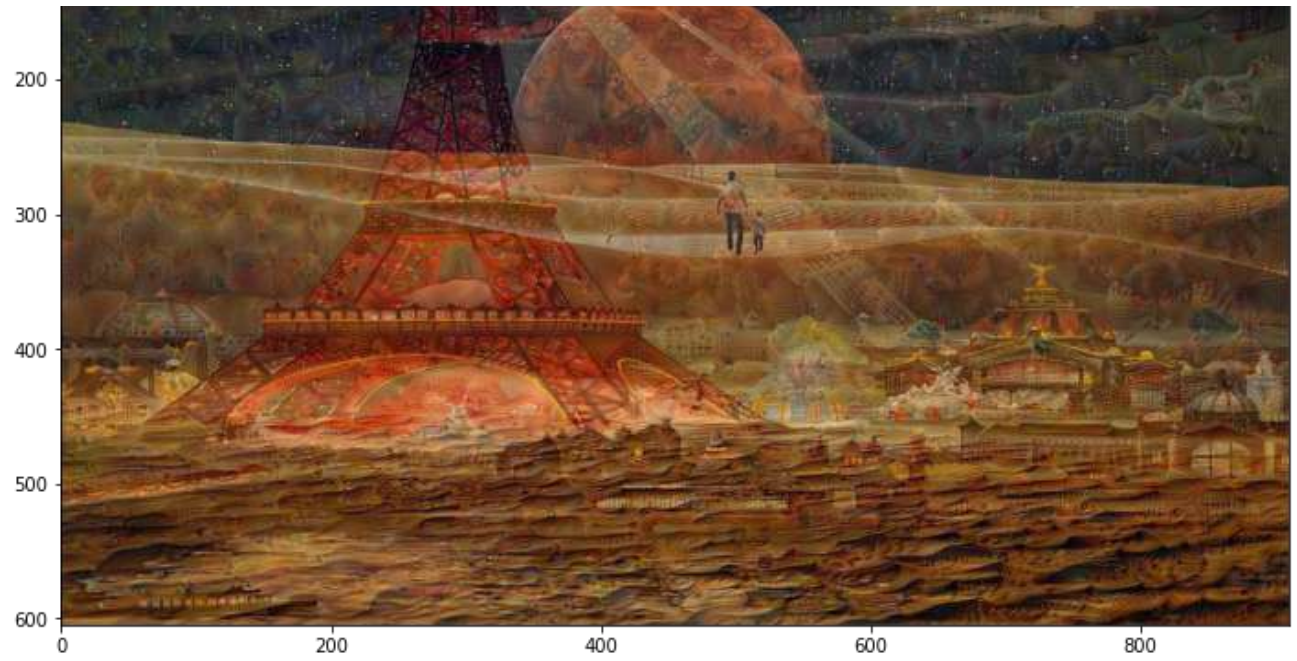
```
Step 100, loss 1.2811204195022583
```




```
Step 200, loss 1.4991165399551392
```

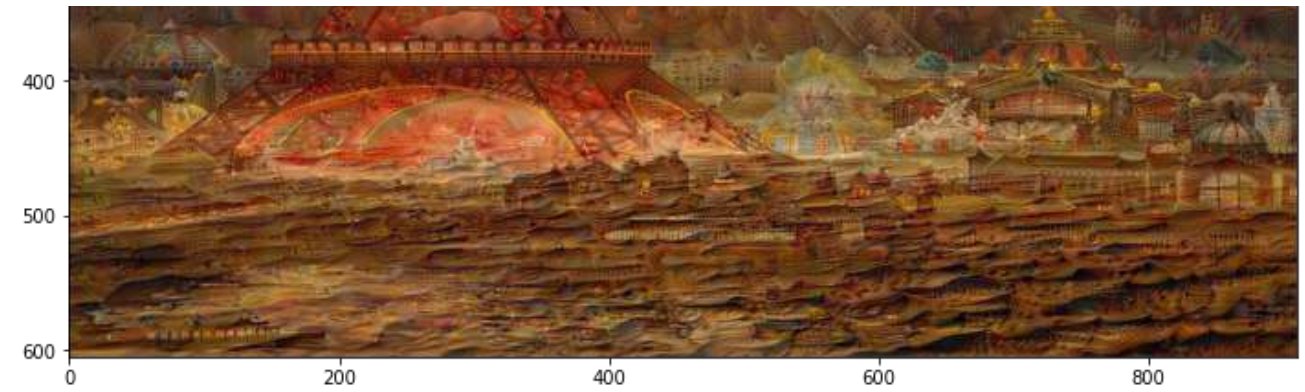



```
Step 300, loss 1.6373804807662964
```




```
Step 400, loss 1.7380015850067139
```


```
Step 500, loss 1.8196983337402344
```



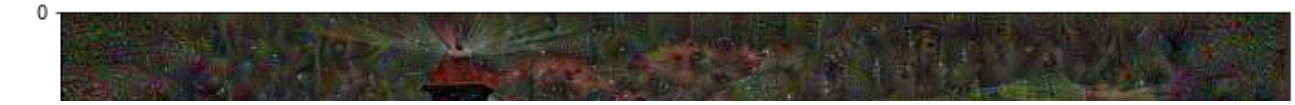

```
Step 600, loss 1.8892912864685059
```


```
Step 700, loss 1.949694275856018
```


```
Step 800, loss 2.0028796195983887
```

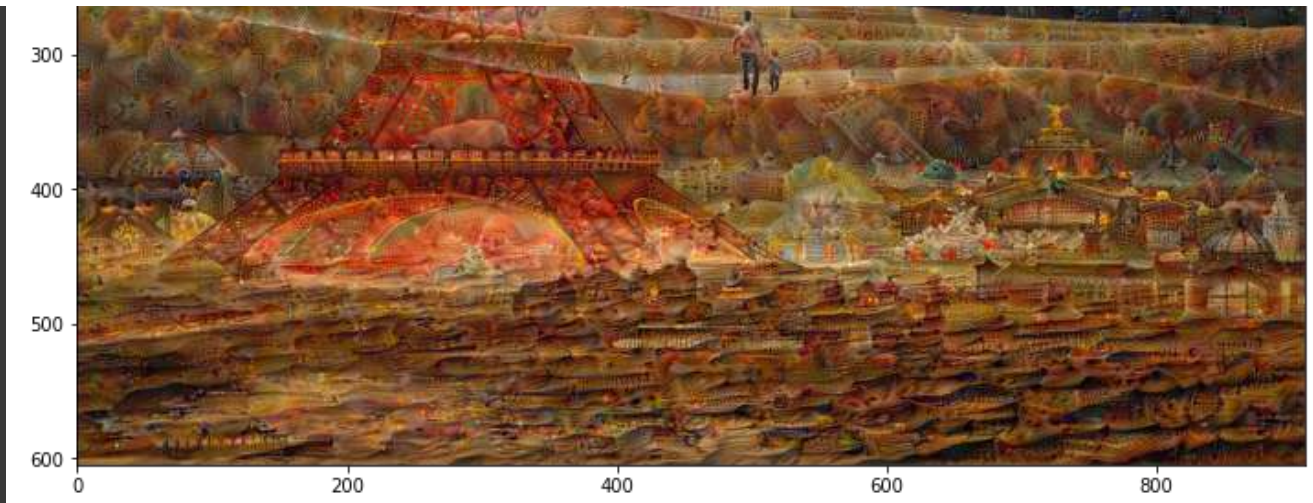

```
Step 900, loss 2.050954818725586
```


```
Step 1000, loss 2.094006061553955
```



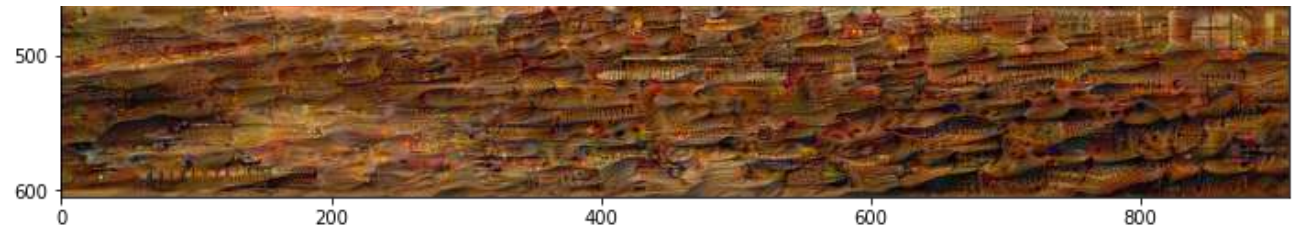

```
Step 1100, loss 2.1331350803375244
```


```
Step 1200, loss 2.168748617172241
```




```
Step 1300, loss 2.201577663421631
```


```
Step 1400, loss 2.2322490215301514
```

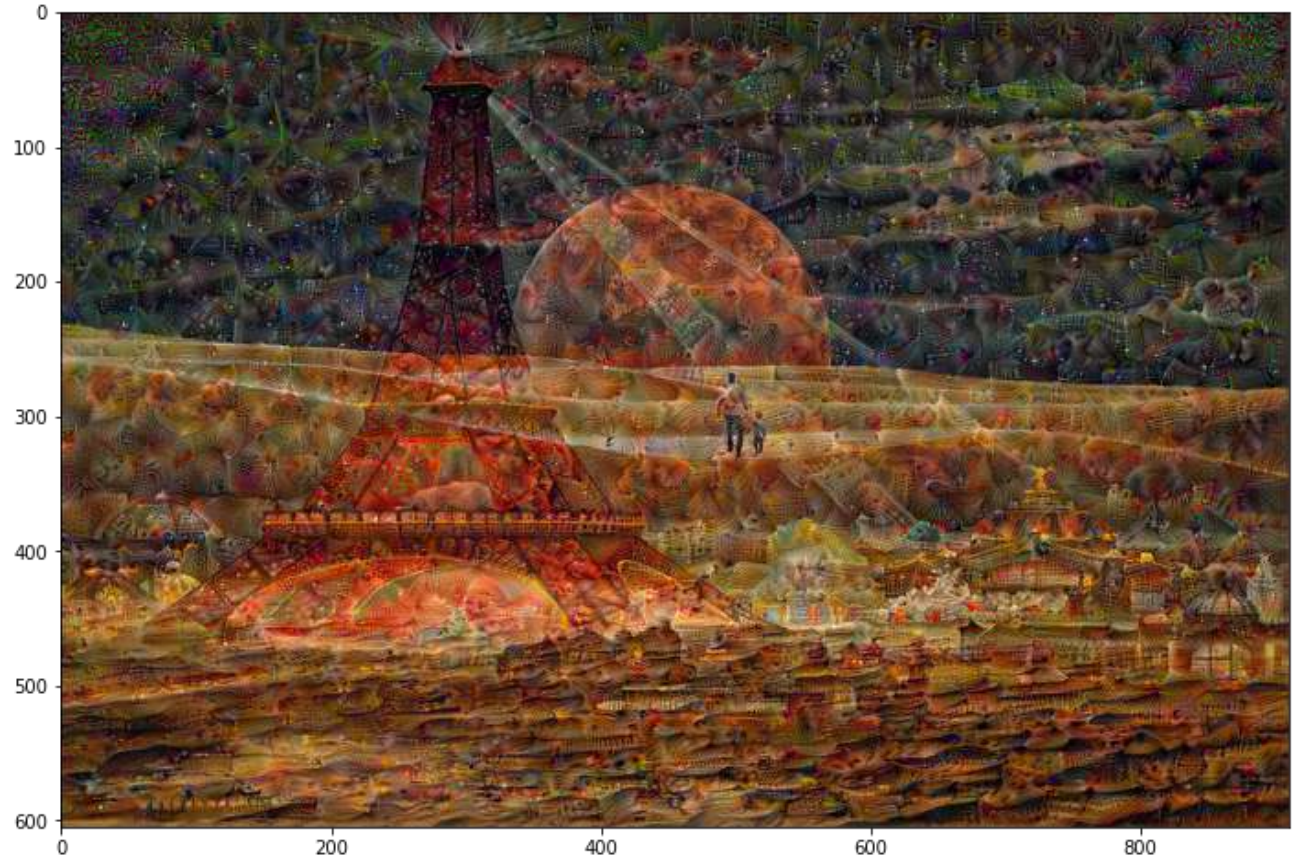



```
Step 1500, loss 2.2606256008148193
```

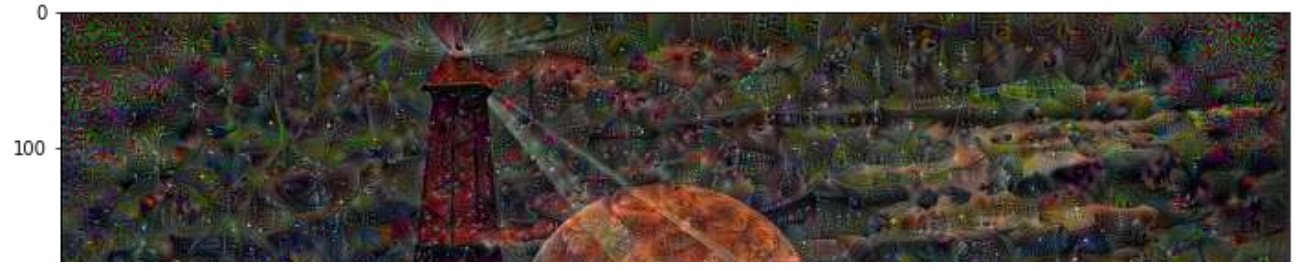

```
Step 1600, loss 2.2874693870544434
```

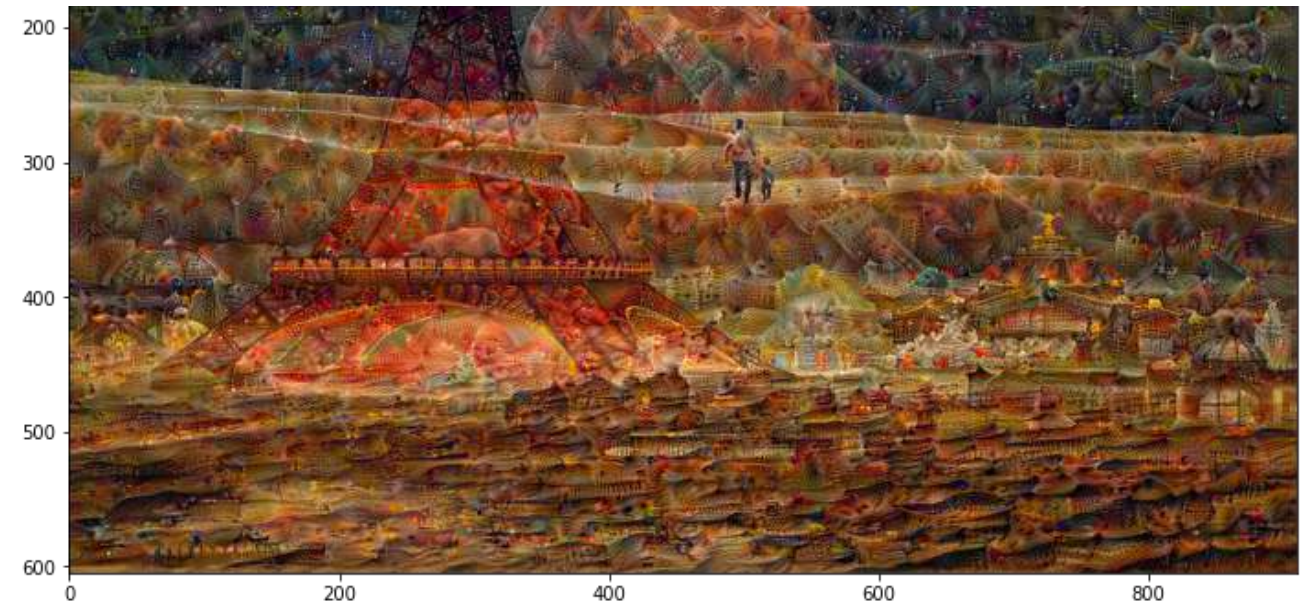

```
Step 1700, loss 2.313199281692505
```

Step 1800, loss 2.3378443717956543

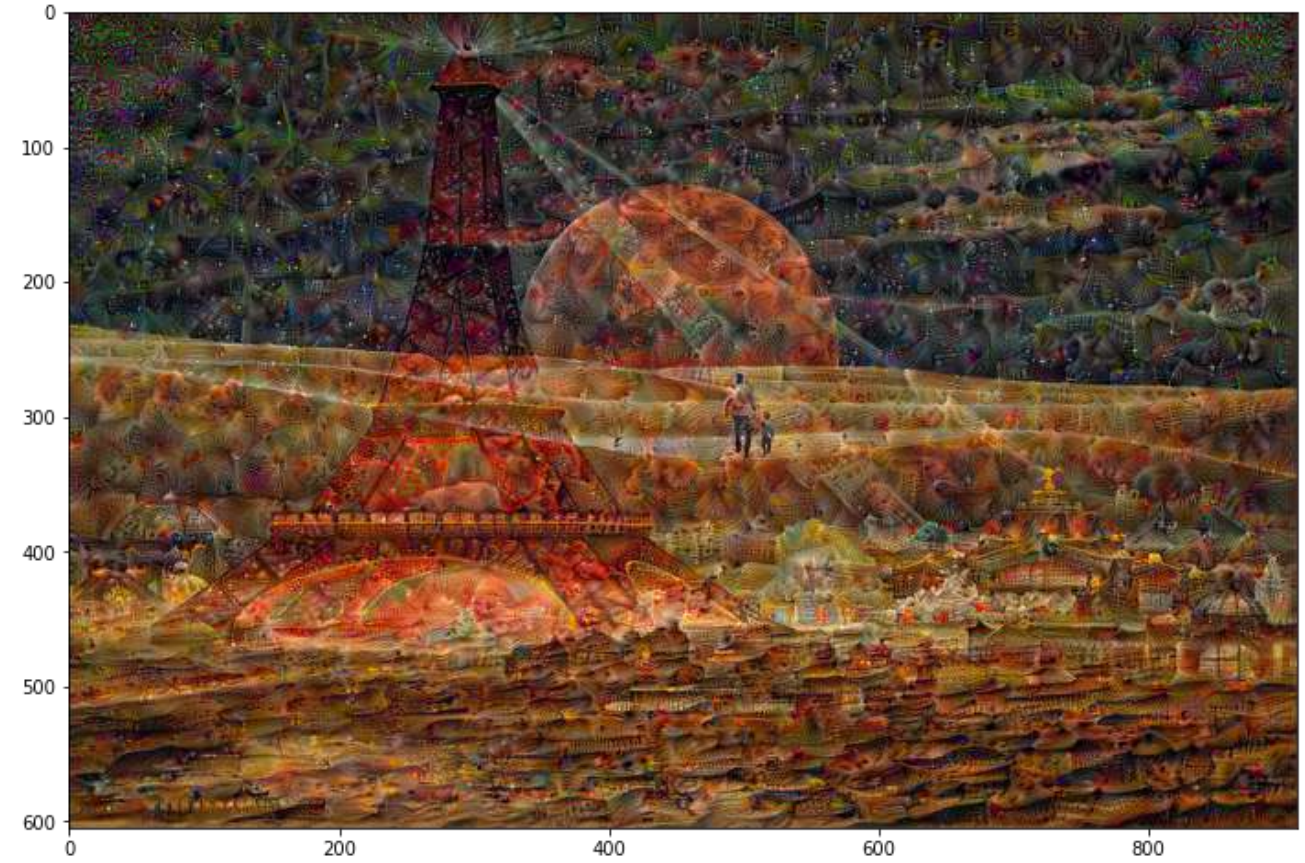


Step 1900, loss 2.3610715866088867

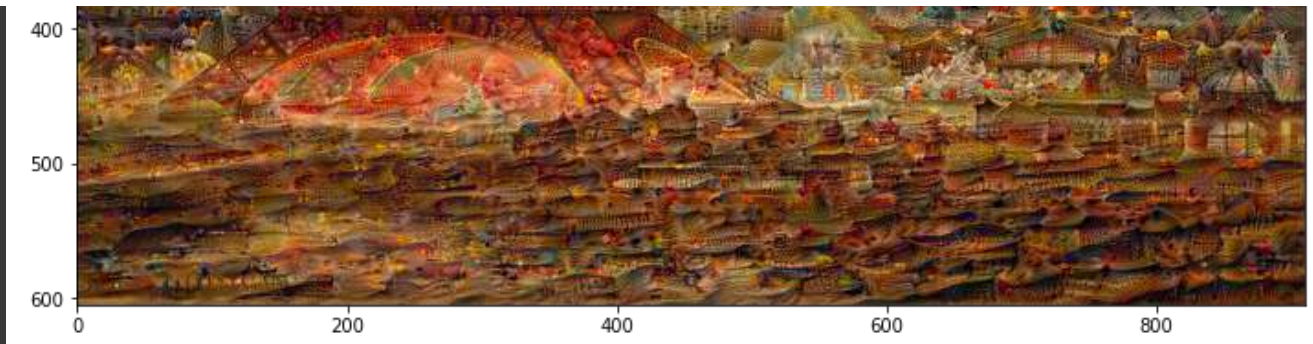




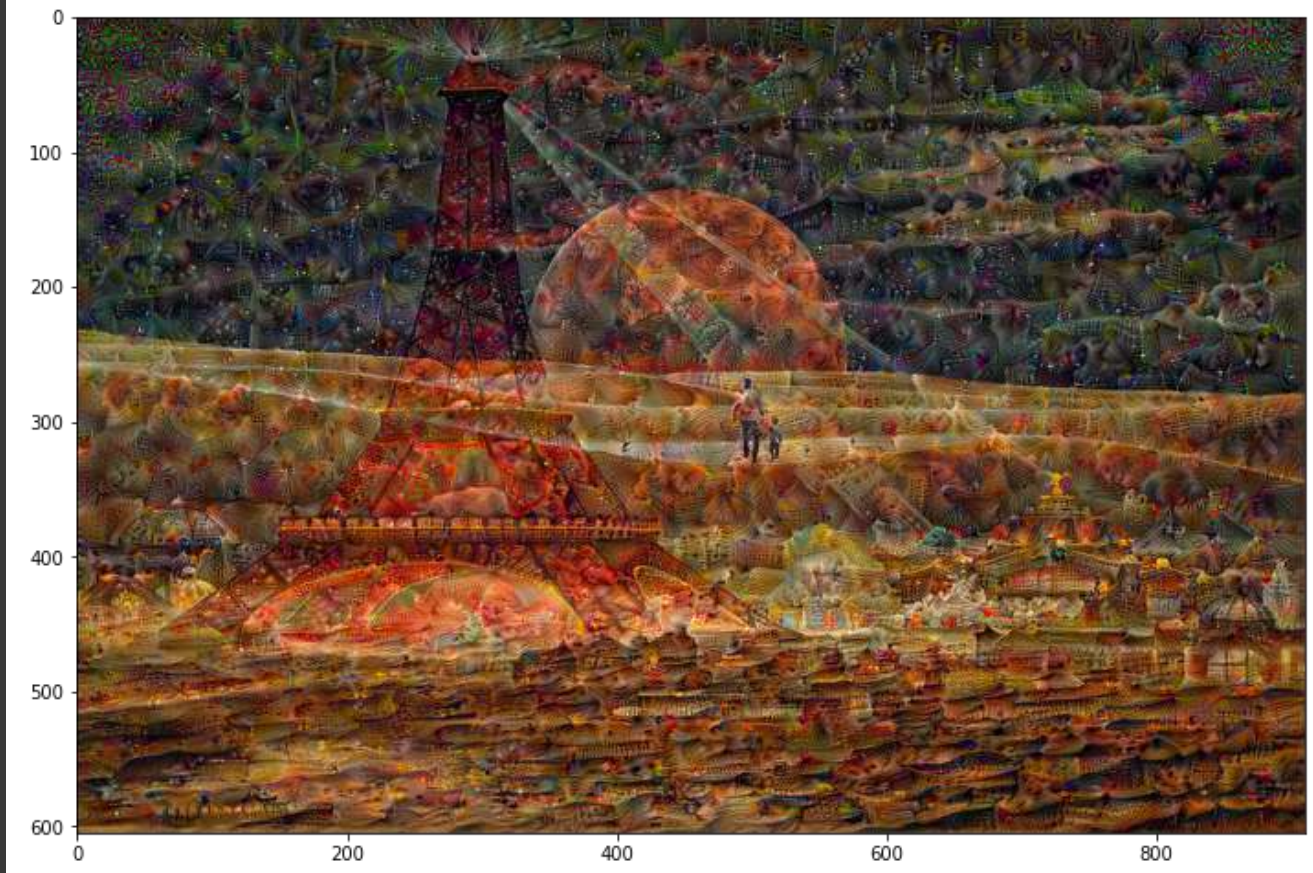

```
Step 2000, loss 2.3835103511810303
```



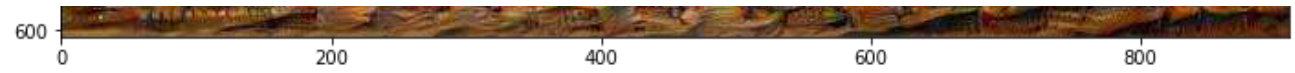

```
Step 2100, loss 2.404651165008545
```




```
Step 2200, loss 2.4254605770111084
```

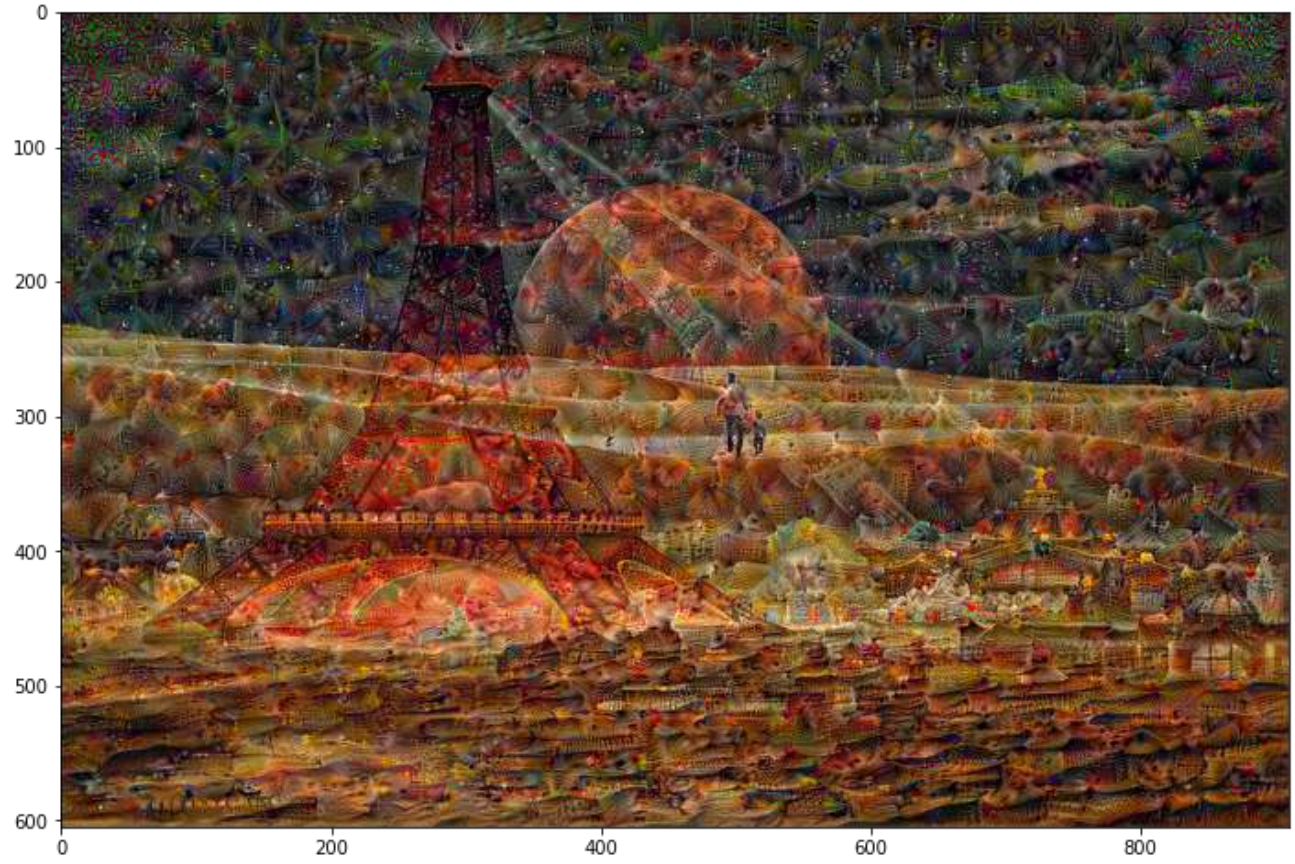



```
Step 2300, loss 2.445452928543091
```

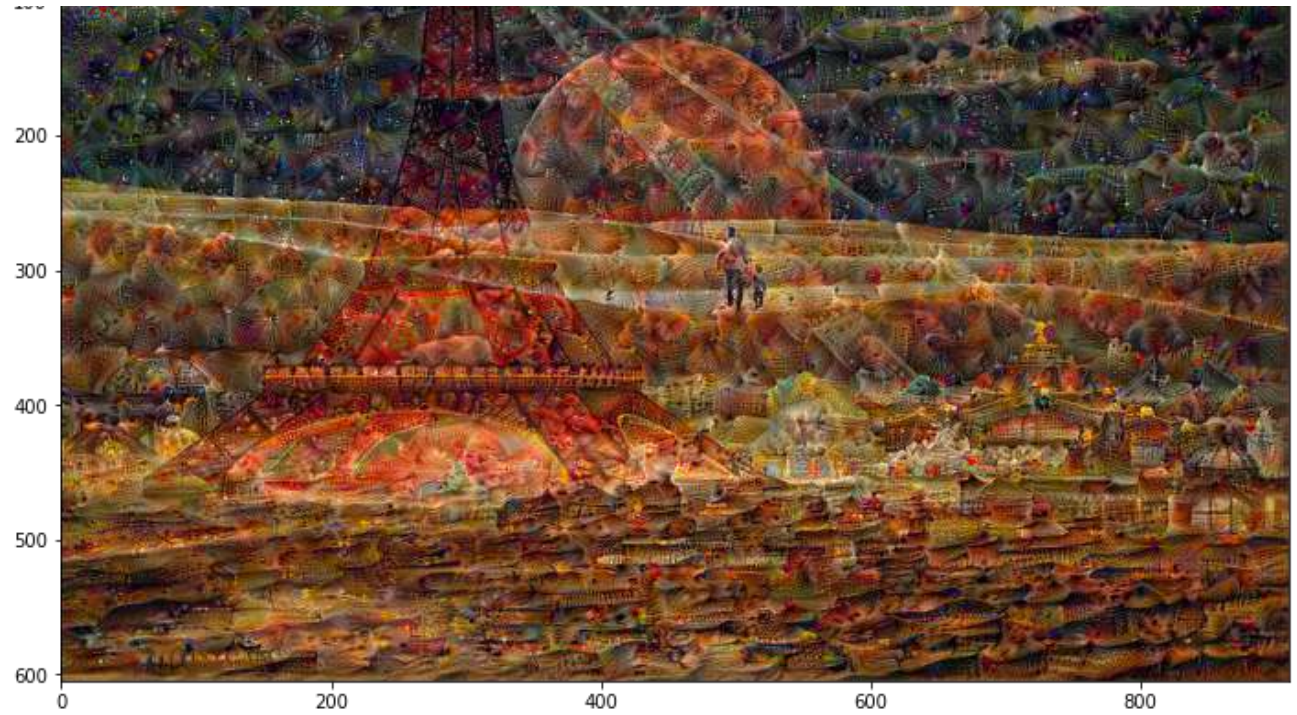



```
Step 2400, loss 2.464843988418579
```


```
Step 2500, loss 2.4832167625427246
```

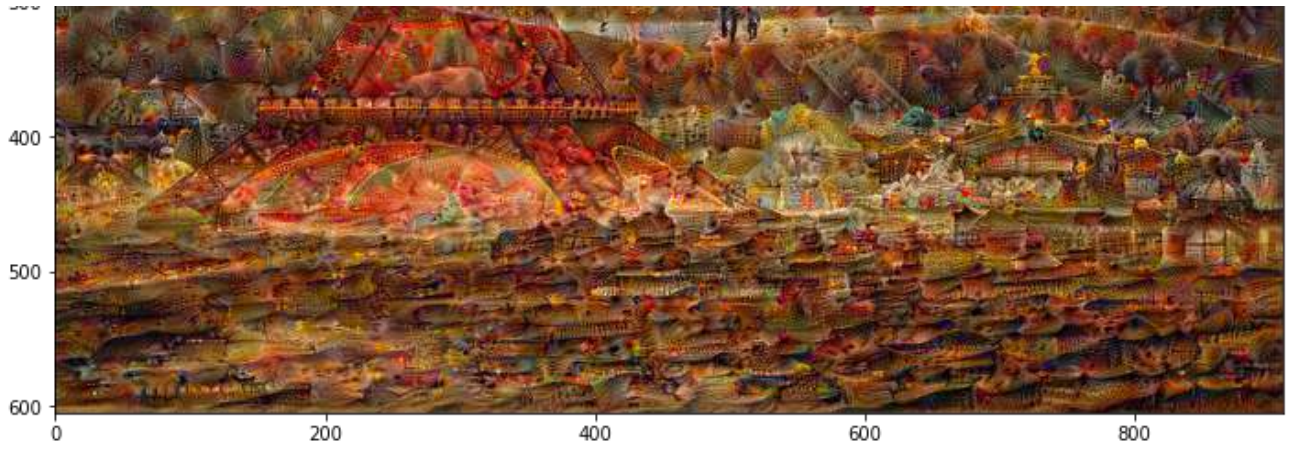



```
Step 2600, loss 2.501070737838745
```

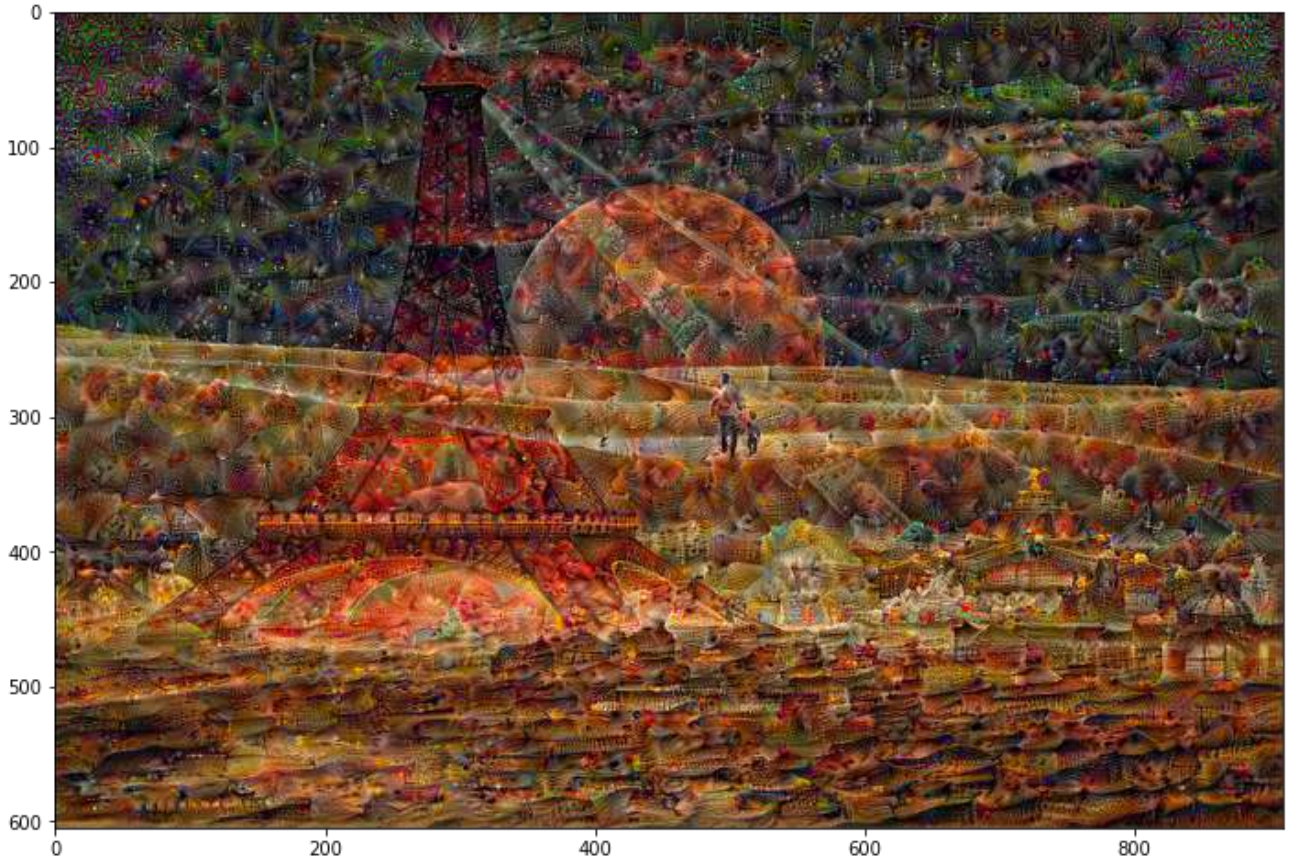



```
Step 2700, loss 2.518592596054077
```

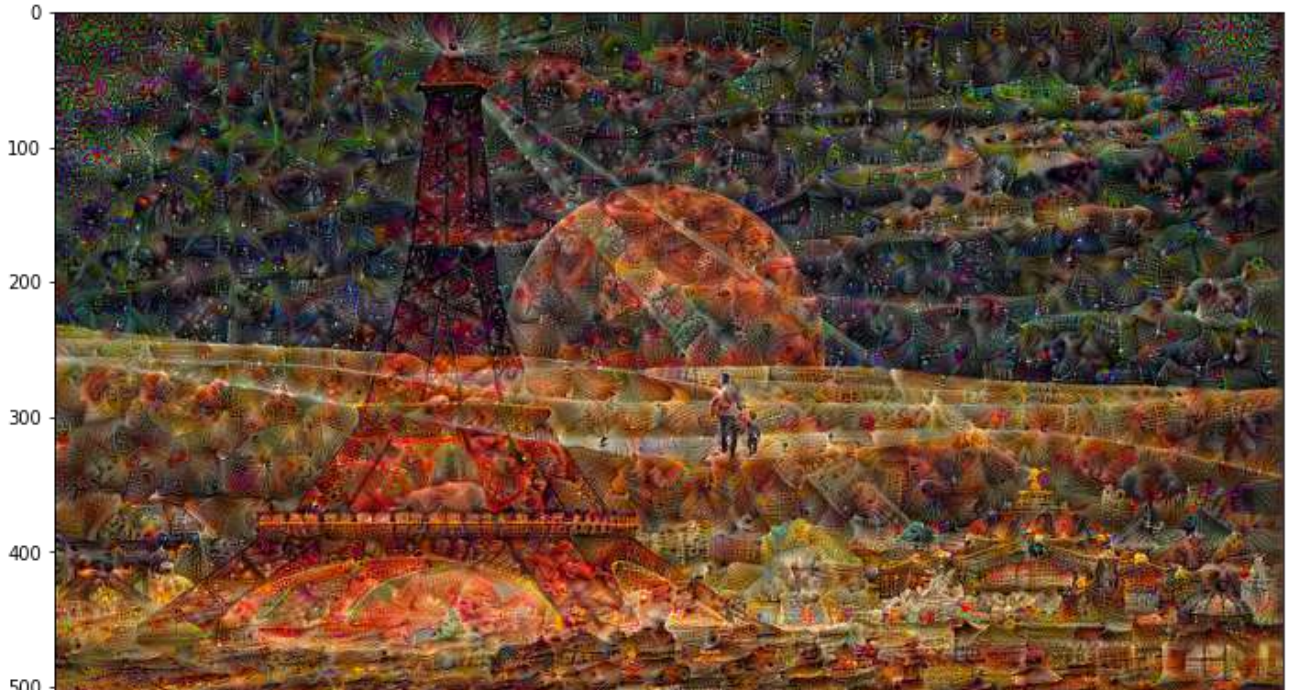

```
Step 2800, loss 2.5352697372436523
```

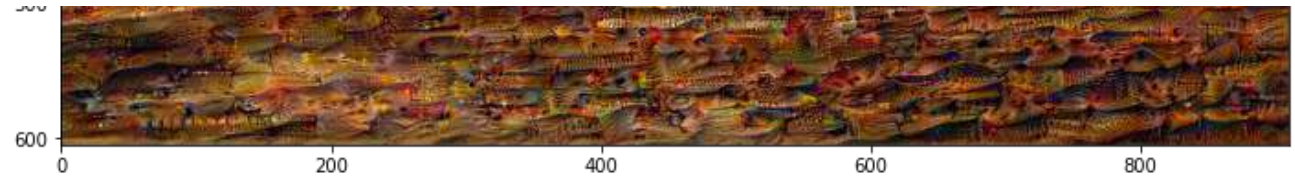



```
Step 2900, loss 2.5516860485076904
```




```
Step 3000, loss 2.5676088333129883
```

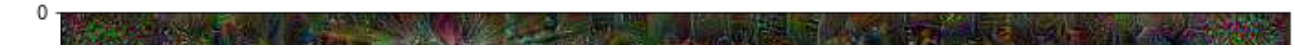




Step 3100, loss 2.582650899887085

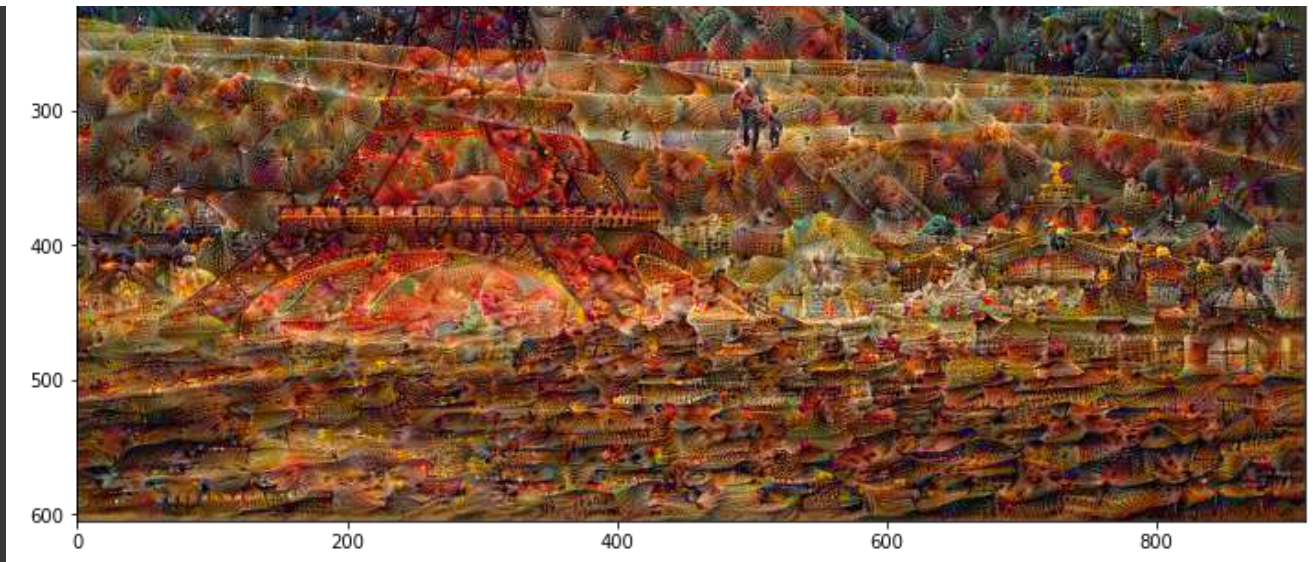
Step 3200, loss 2.597588539123535

Step 3300, loss 2.6123812198638916

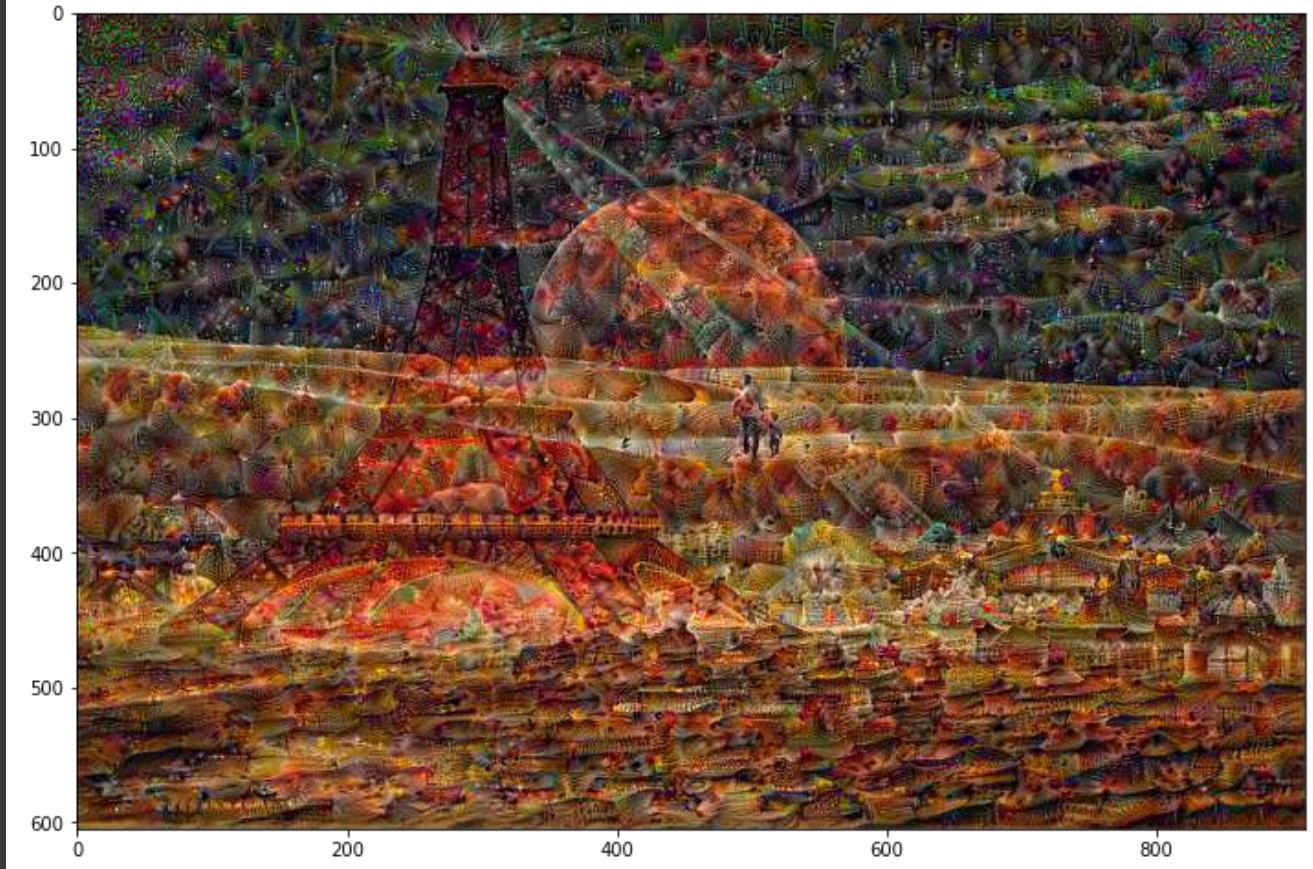



```
Step 3400, loss 2.6267123222351074
```

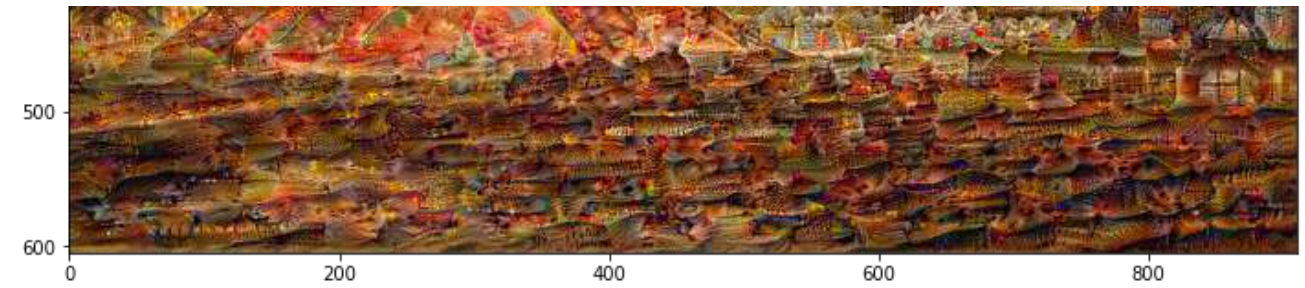

```
Step 3500, loss 2.6407463550567627
```




```
Step 3600, loss 2.654707908630371
```




```
Step 3700, loss 2.668416976928711
```




```
Step 3800, loss 2.6814424991607666
```


```
Step 3900, loss 2.6946818828582764
```

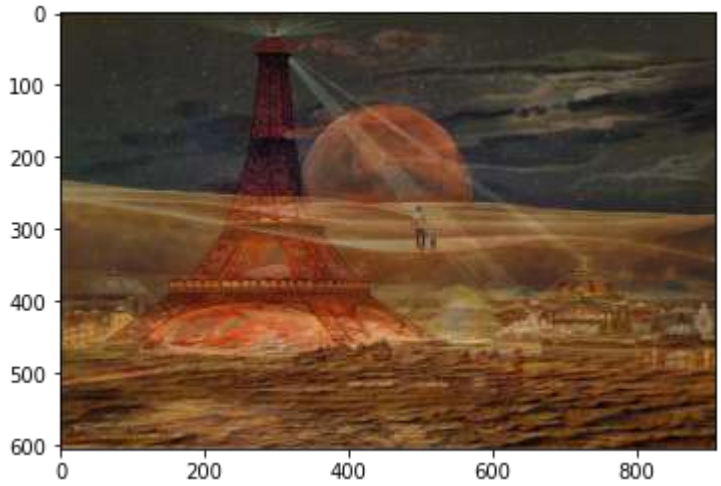
0
200
400
600
800

(VIDEO) APPLY DEEPDREAM TO GENERATE A SERIES OF IMAGES

```
image = tf.keras.preprocessing.image.load_img("img_0.jpg")
```

```
plt.imshow(image)
```

```
<matplotlib.image.AxesImage at 0x7f85244f0dd0>
```




```
# Name of the folder
dream_name = 'mars_eiffel'
```

```
# Blended image dimension

x_size = 910 # larger the image longer is going to take to fetch the frames
y_size = 605
```

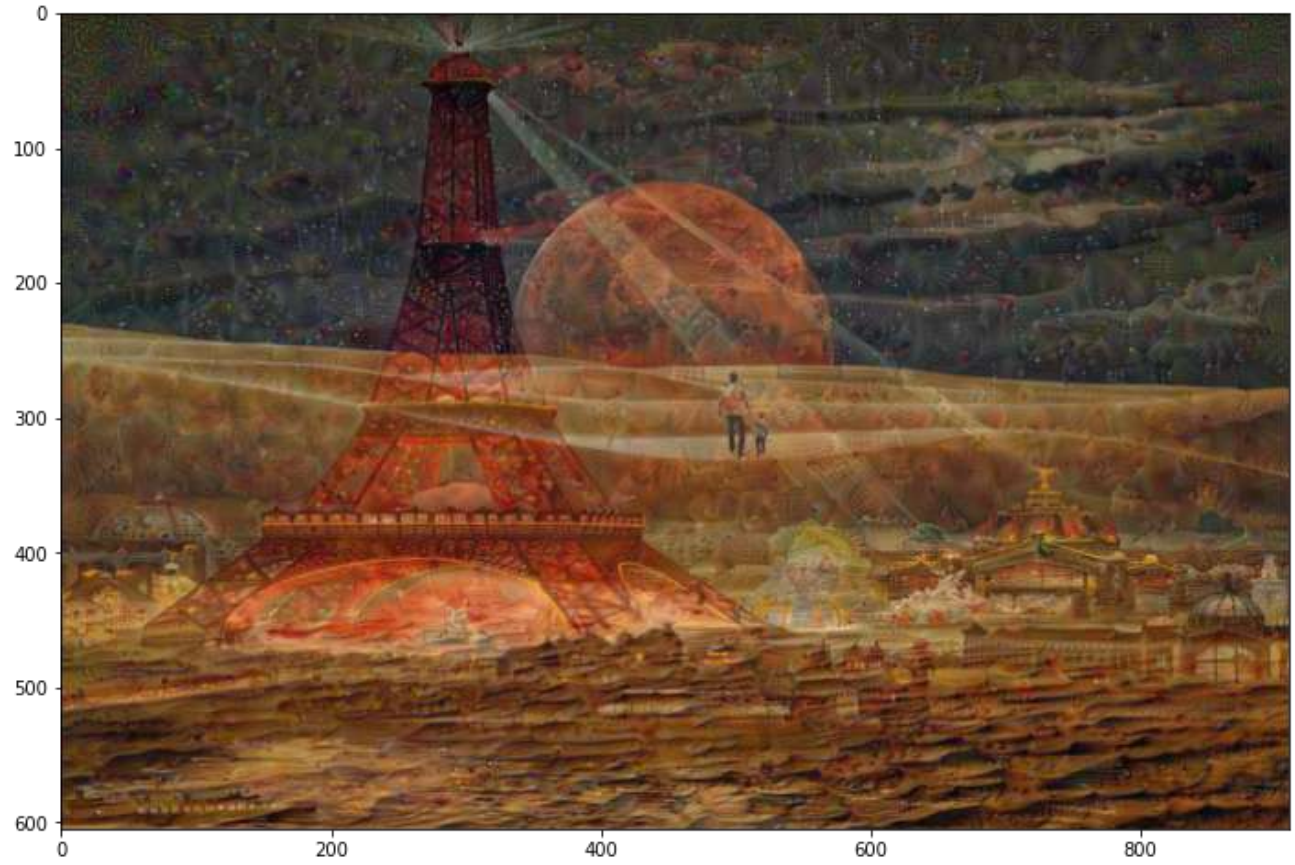
```
# Define Counters
created_count = 0
max_count = 50
```

```
# This helper function loads an image and returns it as a numpy array of floating points

def load_image(filename):
```

```
def load_image(filename):
    image = PIL.Image.open(filename)
    return np.float32(image)
```

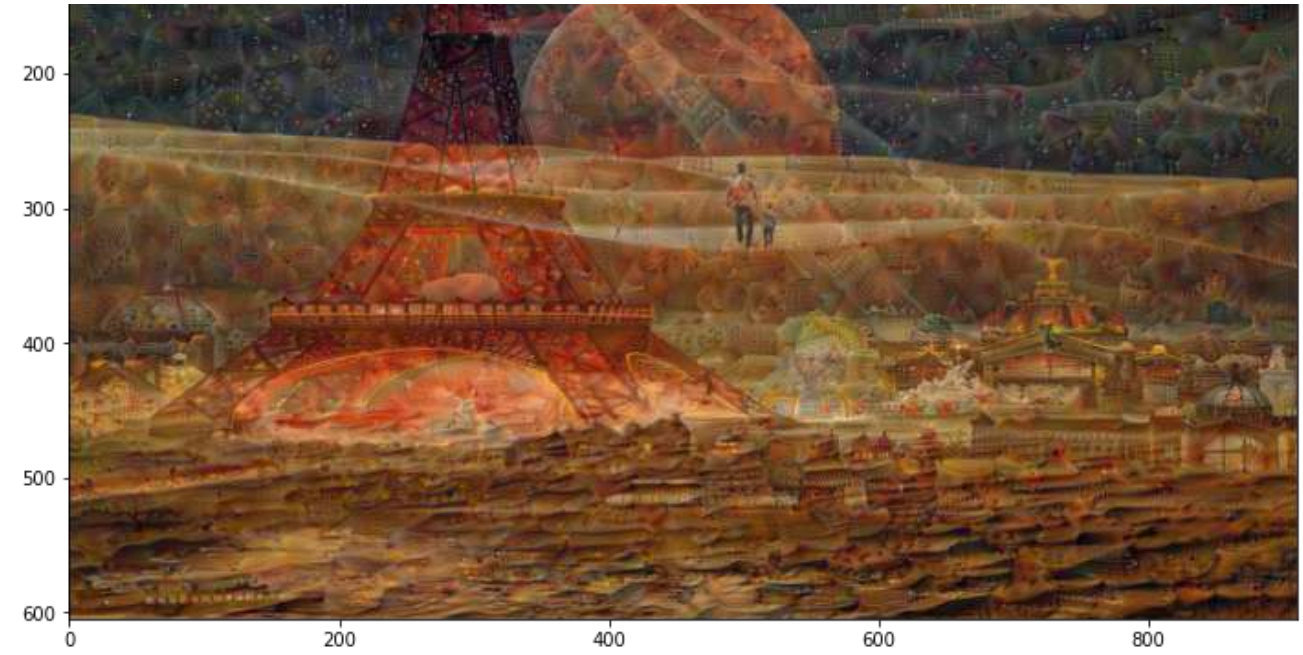
```
for i in range(0, 50):
    # Make sure to create a new folder entitled 'mars_eiffel' and place img_0 in it
    # Get into the dream directory and look for the number of images and then figure out w
    # image we are going to start with and let it dream on and on

    if os.path.isfile('/content/drive/My Drive/Colab Notebooks/deep dream/{}/img_{}.jpg'.f
        print("{} present already, continue fetching the frames...".format(i+1))

    else:
        # Call the load image funtion
        img_result = load_image(r'/content/drive/My Drive/Colab Notebooks/deep dream/{}/im



        # Zoom the image
        x_zoom = 2 # this indicates how quick the zoom is
        y_zoom = 1

        # Chop off the edges of the image and resize the image back to the original shape.
        img_result = img_result[0+x_zoom : y_size-y_zoom, 0+y_zoom : x_size-x_zoom]
        img_result = cv2.resize(img_result, (x_size, y_size))

        # Adjust the RGB value of the image
        img_result[:, :, 0] += 2  # red
        img_result[:, :, 1] += 2  # green
        img_result[:, :, 2] += 2  # blue

        # Deep dream model
        img_result = run_deep_dream_simple(model = deepdream_model, image = img_result, st

        # Clip the image, convert the datatype of the array, and then convert to an actual
        img_result = np.clip(img_result, 0.0, 255.0)
        img_result = img_result.astype(np.uint8)
        result = PIL.Image.fromarray(img_result, mode='RGB')

        # Save all the frames in the dream location
        result.save(r'/content/drive/My Drive/Colab Notebooks/deep dream/{}/img_{}.jpg'.fo

        created_count += 1
        if created_count > max_count:
            break
```

```
1 present already, continue fetching the frames...
```



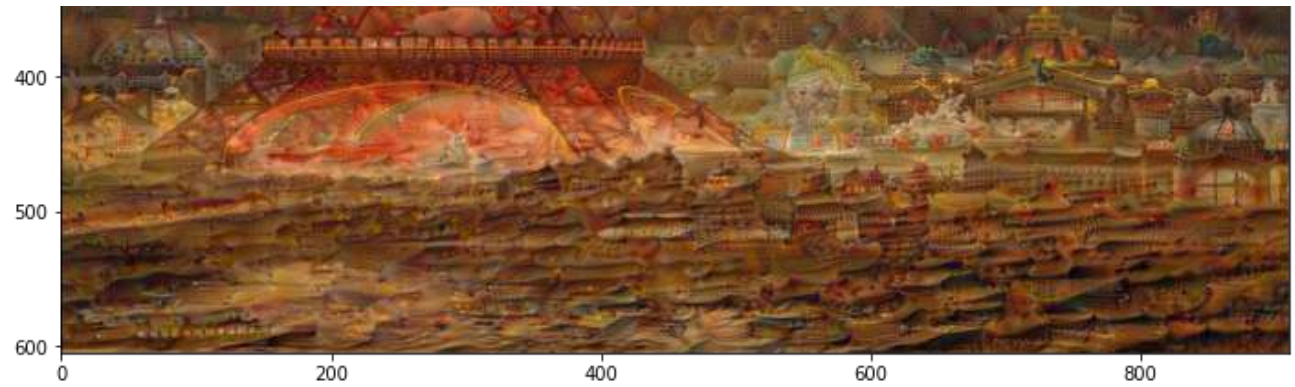

```
Step 0, loss 0.8976892828941345
```


```
Step 100, loss 1.6634790897369385
```




```
Step 200, loss 1.8038506507873535
```


```
Step 300, loss 1.891886591911316
```




```
Step 400, loss 1.958195686340332
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-43-5240cfd82e9d> in <module>()
     26
     27         # Deep dream model
---> 28         img_result = run_deep_dream_simple(model = deepdream_model, image =
img_result, steps = 500, step_size = 0.001)
     29
     30         # Clip the image, convert the datatype of the array, and then
convert to an actual image.

                                    6 frames
/usr/local/lib/python3.7/dist-packages/tensorflow/python/eager/execute.py in
quick_execute(op_name, num_outputs, inputs, attrs, ctx, name)
     58     ctx.ensure_initialized()
     59     tensors = pywrap_tfe.TFE_Py_Execute(ctx._handle, device_name, op_name,
---> 60                                         inputs, attrs, num_outputs)
     61   except core._NotOkStatusException as e:
     62     if name is not None:

KeyboardInterrupt:
```

SEARCH STACK OVERFLOW

(VIDEO) CREATE A VIDEO FROM ALL THE FRAMES

```
from google.colab import files
uploaded = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

Saving mars_eiffel.zip to mars_eiffel.zip

```
# Unzip the folder

from zipfile import ZipFile
file_name = "mars_eiffel.zip"

with ZipFile(file_name, 'r') as zip:
  zip.extractall()
  print('Done')
```

```
Done
```

```
# Path of all the frames

dream_path = 'mars_eiffel'
```

```
# Define the codec and create VideoWriter object
# Download FFmeg

fourcc = cv2.VideoWriter_fourcc(*'XVID') # FourCC is a 4-byte code used to specify the vid

out = cv2.VideoWriter('deepdreamvideo.avi', fourcc , 5.0, (910, 605)) # Specify the fourCC
                                                                      # and frame si
# The frames per second value is depends on few important things
# 1. The number of frames we have created. Less number of frames brings small fps
# 2. The larger the image the bigger the fps value. For example, 1080 pixel image can brin
```

```
for i in range(9999999999999):

    # Get into the dream directory and looks for the number of images and then figure out
    # this image we are going to start with and let it dream on and on
    if os.path.isfile('mars_eiffel/img_{}.jpg'.format(i+1)):
        pass
    # Figure out how long the dream is
    else:
        dream_length = i
        break
```

```
dream_length
```

```
200
```

```
for i in range(dream_length):

    # Build the frames of cv2.VideoWriter
    img_path = os.path.join(dream_path,'img_{}.jpg'.format(i)) # join the dream path

    print(img_path) # print the image path

    frame = cv2.imread(img_path)
    out.write(frame)

out.release()
```

```
mars_eiffel/img_0.jpg
mars_eiffel/img_1.jpg
mars_eiffel/img_2.jpg
mars_eiffel/img_3.jpg
mars_eiffel/img_4.jpg
mars_eiffel/img_5.jpg
mars_eiffel/img_6.jpg
mars_eiffel/img_7.jpg
mars_eiffel/img_8.jpg
mars_eiffel/img_9.jpg
mars_eiffel/img_10.jpg
```

```
mars_eiffel/img_11.jpg
mars_eiffel/img_12.jpg
mars_eiffel/img_13.jpg
mars_eiffel/img_14.jpg
mars_eiffel/img_15.jpg
mars_eiffel/img_16.jpg
mars_eiffel/img_17.jpg
mars_eiffel/img_18.jpg
mars_eiffel/img_19.jpg
mars_eiffel/img_20.jpg
mars_eiffel/img_21.jpg
mars_eiffel/img_22.jpg
mars_eiffel/img_23.jpg
mars_eiffel/img_24.jpg
mars_eiffel/img_25.jpg
mars_eiffel/img_26.jpg
mars_eiffel/img_27.jpg
mars_eiffel/img_28.jpg
mars_eiffel/img_29.jpg
mars_eiffel/img_30.jpg
mars_eiffel/img_31.jpg
mars_eiffel/img_32.jpg
mars_eiffel/img_33.jpg
mars_eiffel/img_34.jpg
mars_eiffel/img_35.jpg
mars_eiffel/img_36.jpg
mars_eiffel/img_37.jpg
mars_eiffel/img_38.jpg
mars_eiffel/img_39.jpg
mars_eiffel/img_40.jpg
mars_eiffel/img_41.jpg
mars_eiffel/img_42.jpg
mars_eiffel/img_43.jpg
mars_eiffel/img_44.jpg
mars_eiffel/img_45.jpg
mars_eiffel/img_46.jpg
mars_eiffel/img_47.jpg
mars_eiffel/img_48.jpg
mars_eiffel/img_49.jpg
mars_eiffel/img_50.jpg
mars_eiffel/img_51.jpg
mars_eiffel/img_52.jpg
mars_eiffel/img_53.jpg
mars_eiffel/img_54.jpg
mars_eiffel/img_55.jpg
mars_eiffel/img_56.jpg
mars_eiffel/img_57.jpg
mars_eiffel/img_58.jpg
```