Part 1: Purpose of the Project:

1. Utilize AI for generating innovative art.
2. Grasp the principles of the DeepDream algorithm.
3. Build, train, and evaluate the DeepDream algorithm for producing art with AI, using the Keras API and TensorFlow 2.0.
4. Use the DeepDream algorithm to add psychedelic effects to images and videos.
5. Comprehend gradient ascent and the maximization of activations.
6. Explore the Inception Net deep neural network model developed by Google.

Creative AI is a new branch within AI wherein it can create paintings, produce new music, write stories. In this project, we will apply AI to generate art for us. We will use the Deep Dream algorithm to create almost creepy/trippy images. We will also understand what the model views/what truly happens within the hidden layers in the model to generate art.

When we try to run the Deep Dream algorithm, we try to maximize the activations or what the hidden layers can see.

1. Deep Dream is an algorithm created by Google that produces images with a surreal, dream-like quality, reminiscent of the effects of potent hallucinogens.
2. The algorithm is trained on millions of images to enhance and create increasingly unusual features.
3. It amplifies the patterns within an image based on its prior training, emphasizing elements it recognizes.
4. For instance, if the algorithm has learned to identify airplanes, it will attempt to highlight airplane-like characteristics in any given image.

Part 2: Import Model with Pre-Trained Weights

In this section, we will import a pre-trained model and investigate some activations within that model.

We will first import the relevant libraries and packages.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import random
import seaborn as sns
import PIL.Image #Pillow is primarily used for image manipulation, reading, and saving images
import os
import PIL.Image
from PIL import Image
import cv2
```

```
np.__version__
```

```
'1.25.2'
```

```
#We want to make sure we are working with TensorFlow 2.0
import tensorflow as tf
tf.__version__
```

```
'2.15.0'
```

```
#We will load a pre-trained inception net model. When we load inceptionNet V3, we will apply transfer
#The idea behind transfer learning is to transfer information/intelligence from a pre-trained network
#We will leverage the Keras API in TensorFlow here.
pre_trained_model = tf.keras.applications.InceptionV3(include_top = False, weights = 'imagenet') #inc
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/inception_v3/i
87910968/87910968 [==============================] - 1s 0us/step
```

So, we are downloading a model that has been pre-trained on the imagenet dataset. The imagenet dataset is a repository for millions of images (around 14 million+ images and thousands of classes). Generally, the error rate for a model like pre_trained_model is around 3.6%, which is pretty good (often even less than that of humans)!

If we include the dense layer in the pre_trained_model, we will have more total and trainable parameters.

```
pre_trained_model.summary()
```

Show hidden output

The total number of parameters is 21.8 million parameters, with 21.7 million trainable parameters.

Part 3: Extract a Sample Image and Perform Pre-Processing

In this section, we will extract an image, blend it with an additional image, and finally perform data pre-processing on the merged image.

```
#We will first unzip the loaded zipped file
!unzip '/content/Creative+AI+Dataset.zip'
```

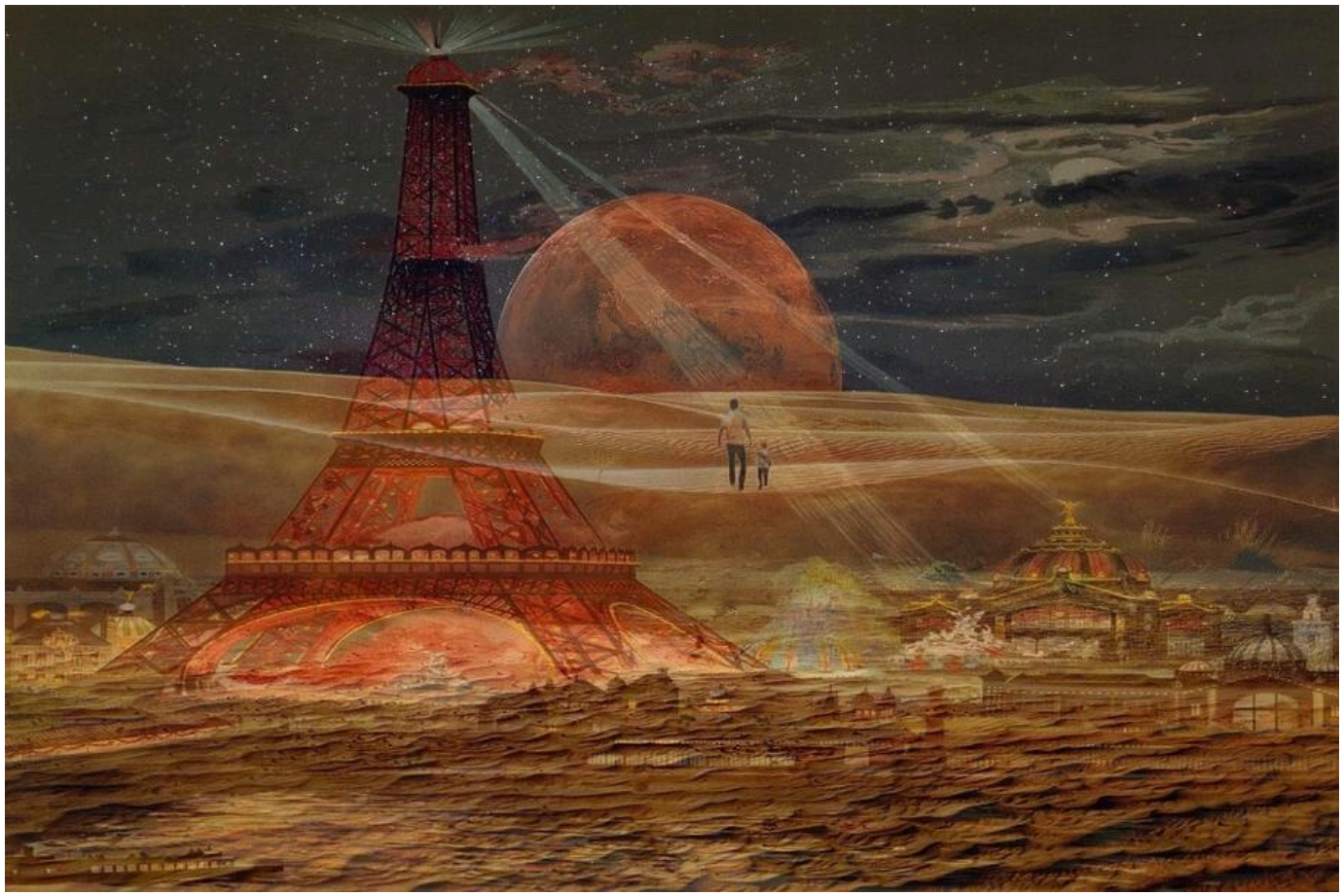
```
Archive:  /content/Creative+AI+Dataset.zip
   creating: Creative AI Dataset/
  inflating: Creative AI Dataset/eiffel.jpg
  inflating: Creative AI Dataset/mars.jpg
  inflating: Creative AI Dataset/mars_eiffel.avi
  inflating: Creative AI Dataset/mars_eiffel.zip
```

```
#Let's load two sample images
#Open the first image from source: https://www.pxfuel.com/en/free-photo-xxgfs
#Load picture of Mars
image_1 = Image.open('/content/Creative AI Dataset/mars.jpg')

#Open the second picture from source: https://commons.wikimedia.org/wiki/File:Georges_Garen_embrasemen
image_2 = Image.open('/content/Creative AI Dataset/eiffel.jpg')

#Blend both the images
image_blend = Image.blend(image_1, image_2, alpha = 0.5)

#Save blended image
image_blend.save("image_0.jpg")
```

Here, the value of alpha is important. Alpha is the interpolation factor. An alpha of 0.0 will return the copy of the first image, and an alpha of 1.0 will return a copy of the second image.

```
#Load the blended image
image_blend_load = tf.keras.preprocessing.image.load_img("image_0.jpg")
```

```
image_blend_load
```



```
#Get shape of image
np.shape(image_blend_load)
#The third dimension (3) indicates the color of the picture with components red, green, and blue
```

Show hidden output

```
#type of image
type(image_blend_load) #this is a jpeg image file
```

PIL.JpegImagePlugin.JpegImageFile
def __init__(fp=None, filename=None)

Base class for image file format handlers.

```
#Let's convert this image into a numpy array
image_blend_load = tf.keras.preprocessing.image.img_to_array(image_blend_load)
```

```
#now checking and confirming image_blend_load is of numpy array type
type(image_blend_load)
```

Show hidden output

```
#checking the minimum and maximum values of the image
#if the image is not normalized, we should be seeing values between 0 and 255.
#If the image is normalized, we should be seeing values between 0 and 1 or between -1 and 1
print("Minimum pixel value: ", image_blend_load.min())
print("Maximum pixel value: ", image_blend_load.max())
```

Show hidden output

From the above output, the image is not normalize. Therefore, we will normalize the image:

```
image_blend_load = np.array(image_blend_load)/255
```

```
#Checking the minimum and maximum values of pixels of the image after normalization
print("Minimum pixel value: ", image_blend_load.min())
print("Maximum pixel value: ", image_blend_load.max())
```

Show hidden output

We will now add an additional dimension to our image. The idea here is that we don't feed one image to the model, we feed batches of images. Adding this new dimension will help us feed in batches of images to the model.

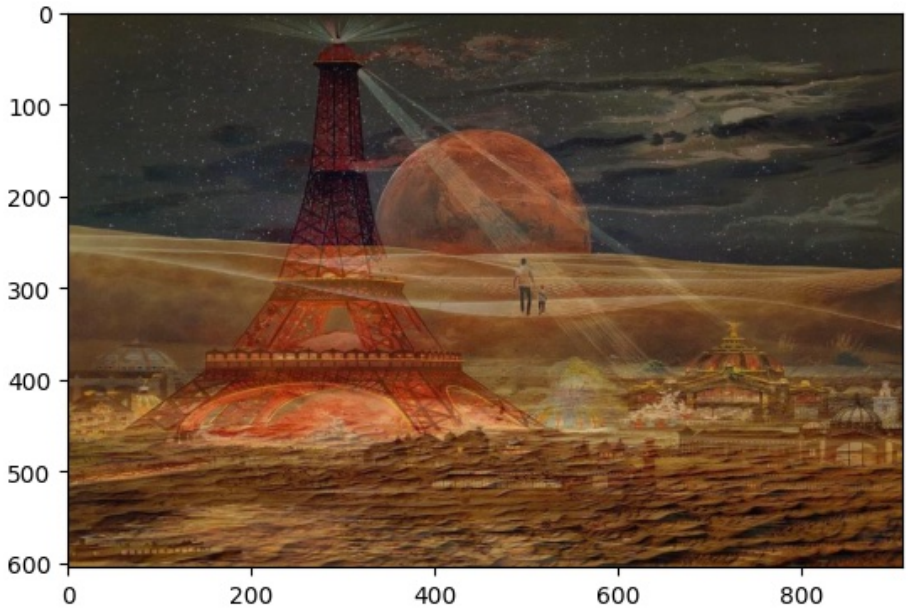
```
image_blend_load = tf.expand_dims(image_blend_load, axis = 0)
```

```
image_blend_load.shape #this new dimension will be used when we start running the artificial neural n
```

Show hidden output

```
#What will happen if we want to remove our dimension? Let's try removing a dimension and plotting the

print(np.squeeze(image_blend_load).shape)
plt.imshow(np.squeeze(image_blend_load))
```

```
(605, 910, 3)
<matplotlib.image.AxesImage at 0x7a5ea015e5f0>
```



Part 4: Run Pre-Trained Model and Explore Activations:

We want to obtain our InceptionNet model, feed in our image, and assess what is happening in different layers by maximizing the activations of those layers.

```
#Maximize the activations of layers:
names = ['mixed3', 'mixed5', 'mixed7']
layers = [pre_trained_model.get_layer(name).output for name in names]

#Create a new model that performs feature extraction
deep_dream = tf.keras.Model(inputs = pre_trained_model.input, outputs = layers)
```

```
deep_dream.summary()
```

Show hidden output

```
#Let's feed our image into this model and see the activations in the hidden layers
activations = deep_dream(image_blend_load)
activations
#We can take a look at the hidden layers in the model and explore neuron communications and connectic
#Outputs have different dimensions because they come from different layers.
```

Show hidden output

```
len(activations) #for each layer
```

```
3
```

Part 5: Understand the Deep Dream Algorithm

1. If we take an image and feed it to a pre-trained convolutional neural network, we will find that the first layers generally capture low-level features, called edges/curves.
2. As we go deeper in the network, we will find that higher level features are detected, like faces and cars.
3. The final few layers very deep into the model assemble into complete inferences. These neurons are generally activated in response to complex things such as buildings, etc.
4. The kind of layers we select will impact the response of the network.

How does the deep dream algorithm work?

1. If we have a pre-trained network and an input image and we pass this image to the network, a series of activations will be generated.
2. We want to take these activations and manipulate the input image in a way to maximize the activations coming from the corresponding layers.
3. So, if we ask the first couple of layers and the last couple of layers to maximize what they see/detect, a phenomeon called "inceptionism" occurs, wherein the results become dreamy.
4. So, the model superimposes what it has already been trained to see on the original image, churning out trippy images.
5. So, when we feed an image to a trained artificial neural network, the neurons fire and generate activation. The model tries to change the input image so that some of these neurons can fire more and maximize activations (we can select which particular neurons in which specific layers should fire more as well).
6. This process is repeated continuously until the input image contains the features the layer was initially looking for (for example, detecting cats in a pink sky until the model starts creating cat faces on top of the pink sky, until the layer is satisfied with the results).

The following are the specific steps of the deep dream algorithm:

1. Feed an image through a pre-trained artificial neural network or convolutional neural network or a residual neural network or inception net neural network, or any other appropriate neural network.
2. Then, we will select a layer (the initial layers capture lines/edges/basic shapes, while the deep layers capture full shapes, like faces, buildings, and animals).
3. Calculate the activations generating from the selected layer.
4. Calculate the activation gradient with respect to the input image fed to the network.
5. Modify the image until the activations are maximized, increasng the patterns seen by the network.
6. Repeat steps 1-5 across multiple scales.

Part 6: Understand Gradient Calculations

We will be using tf.GradientTape() in TF 2.0 for automatic differentiation (first order derivative dy/dx). This is done for a sample expression x^3 to understand that the output received will be 3x^2.

Since the value of x being fed is 2, the answer dy_dx = 3x^2 = 3*2*2 = 12

```
x = tf.constant(2.0) #putting x=2


with tf.GradientTape() as gt:
  gt.watch(x) #we are now watching the x parameter
  y = x**3
dy_dx = gt.gradient(y,x)
```

```
dy_dx #Here, the gradient works and we get the answer as 12.
```

```
<tf.Tensor: shape=(), dtype=float32, numpy=12.0>
```

```
#Let's try another equation: x^4 + x^5 at x = 5
x_new = tf.constant(5.0)
with tf.GradientTape() as gt_new:
  gt_new.watch(x_new) #we are now watching the x parameter
  y_new = (x_new * x_new * x_new * x_new) + (x_new * x_new * x_new * x_new * x_new)
dy_dx_new = gt_new.gradient(y_new,x_new)
```

```
dy_dx_new
```

```
<tf.Tensor: shape=(), dtype=float32, numpy=3625.0>
```

Part 7: Implement Deep Dream Algorithm to Calculate Loss

We will use the Keras documentation to implement the algorithm.

```
image_blend_load.shape
```

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

```
#we will take the sample image and squeeze it back to 3 dimensions
image_blend_load = tf.squeeze(image_blend_load, axis = 0)
```

```
#checking shape of the image once again
image_blend_load.shape
```

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

```
# This function will help us with loss calculations. It will take the model, feedforward the input im
def calculate_loss(image, model):
  image_batch = tf.expand_dims(image, axis = 0)
  activations_layers = model(image_batch)
  print('Activation Layers =\n', activations_layers)

  losses = []
  for act in activations_layers:
    loss = tf.math.reduce_mean(act)
    losses.append(loss)

  print('Losses from the activation layers: ', losses)
  print('Shape of Losses: ', tf.shape(losses))
  print('Sum of Losses: ', tf.reduce_sum(losses))

  #printing the sum of losses: this parameter returned will be eventually fed back into a new model t
  return tf.reduce_sum(losses)
```

```
#calculate the losses with our model and input image
loss = calculate_loss(tf.Variable(image_blend_load), deep_dream)
```

Show hidden output

```
loss #we have the losses summed up from both the activations
```

```
<tf.Tensor: shape=(), dtype=float32, numpy=0.5578056>
```

Part 8: Implement Deep Dream Algorithm to Calculate the Gradient

In this section, we will use the loss calculated in Part 7 to calculate the gradient of the input image and add this gradient to the input image.

When we repeat this process multiple times, we feed images continuously to excite the neurons and generate more dream-like visuals.

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

@tf.function
def deep_dream_gradient(model, image, step_size):
  with tf.GradientTape() as tape:
    tape.watch(image) #watching tf.Variable, in this case our input image
    loss = calculate_loss(image, model)

  #Calculate the gradient as in task 6
  gradients = tape.gradient(loss, image)
  #Normalize the gradient based on standard deviation
  print('Gradients =\n', gradients)
  print('Shape of Gradients = \n', np.shape(gradients))
  gradients /= tf.math.reduce_std(gradients)

  #maximize the activations in gradient ascent so that neurons can fire up more, the input image exci
  image += gradients * step_size
  image = tf.clip_by_value(image, -1, 1) #normalizing the image pixel values
  return loss, image



def deprocess(image):
  #Pre-processing function
  image = 255*(image + 1.0)/2.0
  return tf.cast(image, tf.uint8)


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 = deep_dream_gradient(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)


#Run the model after loading the image again
#We will pause running this, due to its 4000 steps
image_blend_load = (tf.keras.preprocessing.image.load_img('/content/image_0.jpg'))

# Convert PIL Image to a NumPy array
image_blend_load = tf.keras.preprocessing.image.img_to_array(image_blend_load)

dreamy = run_deep_dream_simple(model = deep_dream, image = image_blend_load, steps = 4000, step_size
```

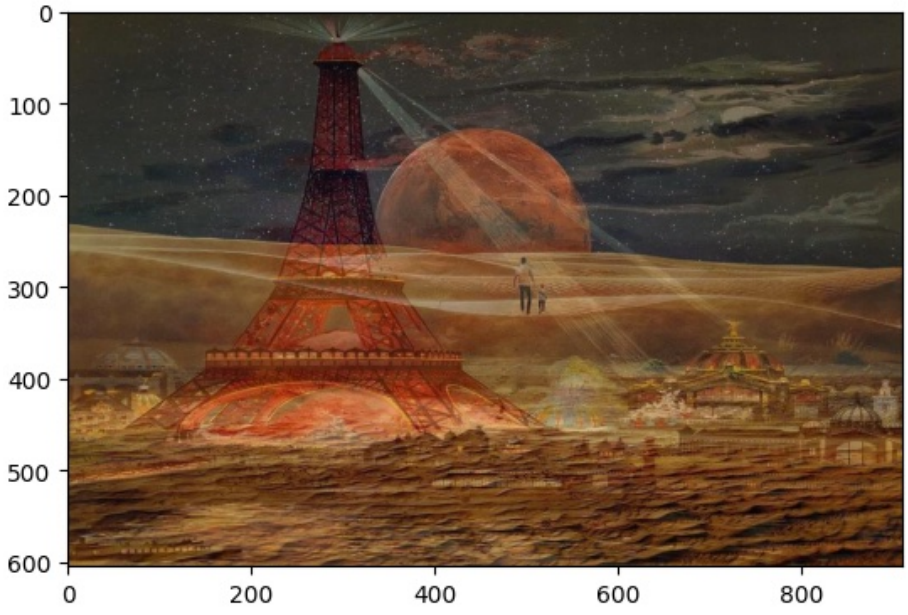
Show hidden output

An increasing loss is a good sign, it means the activation is getting maximized, leading to greater firing of neurons and eventually more dreamy images.

Apply Deep Drea, Algorithm to Generate a Video (Series of Images)

```
video_image = tf.keras.preprocessing.image.load_img("/content/Creative AI Dataset/mars_eiffel/image_0

plt.imshow(video_image) #This is our blended original image
```

```
<matplotlib.image.AxesImage at 0x7a5e30587ee0>
```

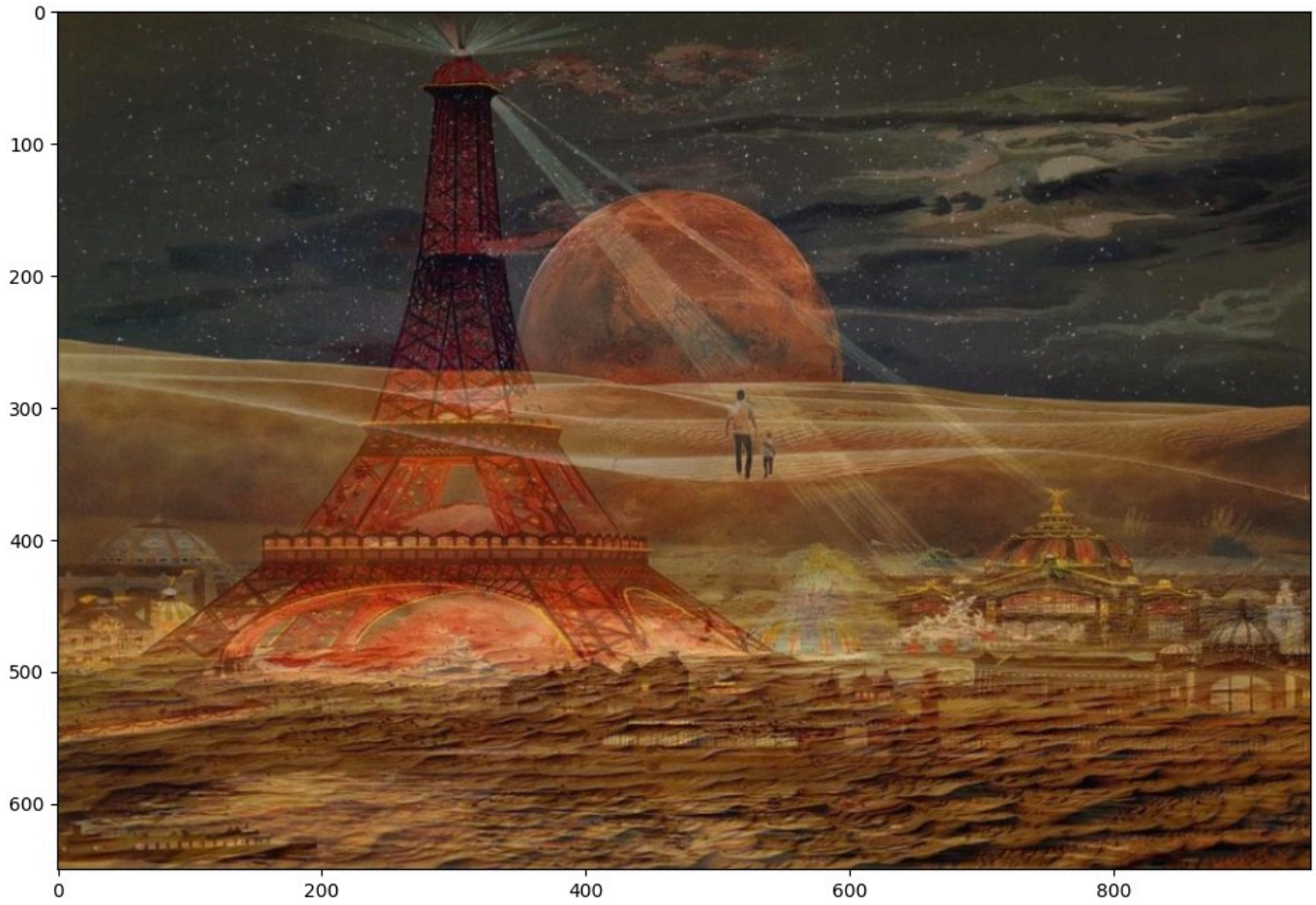


Now, we will be zooming into the image and creating a series of images(video) out of it.

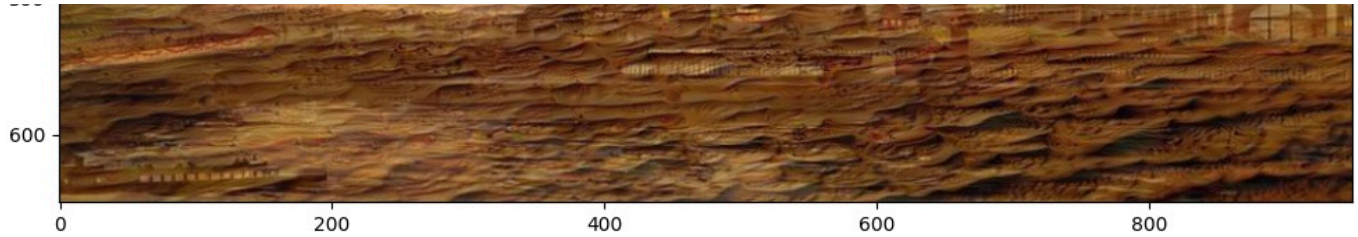
```
#Create a folder
dream_name = 'mars_eiffel'
```

```
#Specify the dimensions of the larger image to be used
x_size = 950
y_size = 650
#When the image is larger, it would take longer to get to the frames


#Counters to decide how many images we would generate
max_count = 50
min_count = 0


#Load the image from the filename and return the floating points of the image as a numpy array
def loading_image(filename):
  image = PIL.Image.open(filename)
  return np.float32(image)


for i in range(0, 50):
  #Ensuring we have a folder created called 'mars_eiffel' with the sample input image image_0.jpg loa
  if os.path.isfile("/content/Creative AI Dataset/{}/image_{}.jpg".format(dream_name, i+1)):
    print("{} is there already, continue getting to the frames ".format(i+1))

  else:
    #call the loading_image function
    # Add missing '/' in the file path
    resulting_image = loading_image(r'/content/Creative AI Dataset/{}/image_{}.jpg'.format(dream_name

    #Zoom the image
    x_zoom = 2
    y_zoom = 2 # this indicates how quickly we zoom into the image

    # Chop off the edges of the image and resize the image back to the original shape.
    #Doing so will create a zoom effect
    resulting_image = resulting_image[0+x_zoom : y_size-y_zoom, 0+y_zoom : x_size-x_zoom]
    resulting_image = cv2.resize(resulting_image, (x_size, y_size))

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

    # Deep dream model
    resulting_image = run_deep_dream_simple(model = deep_dream, image = resulting_image, steps = 500,

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

    # Save all the frames in the dream location
    final_result.save(r'/content/Creative AI Dataset/{}/image_{}.jpg'.format(dream_name, i+1))

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

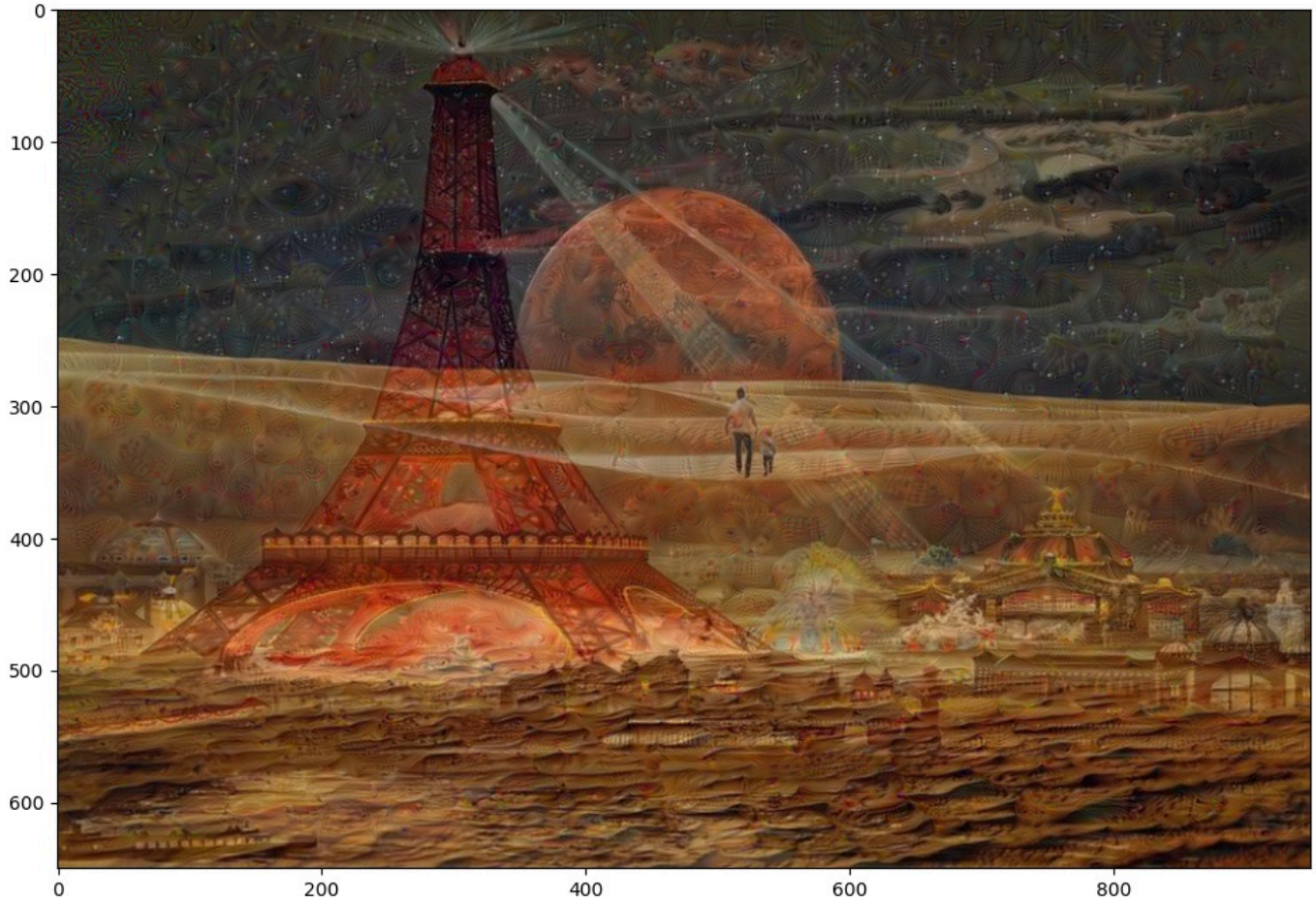
```
Activation Layers =
 [<tf.Tensor 'model/mixed3/concat:0' shape=(1, 38, 57, 768) dtype=float32>, <tf.Tensor 'model/mix
Losses from the activation layers:  [<tf.Tensor 'Mean:0' shape=() dtype=float32>, <tf.Tensor 'Mea
Shape of Losses:  Tensor("Shape:0", shape=(1,), dtype=int32)
Sum of Losses:  Tensor("Sum:0", shape=(), dtype=float32)
Gradients =
 Tensor("gradient_tape/Reshape_4:0", shape=(650, 950, 3), dtype=float32)
Shape of Gradients =
 (650, 950, 3)
```




```
Step 0, loss 0.5673832893371582
```




```
Step 100, loss 1.2989927530288696
```



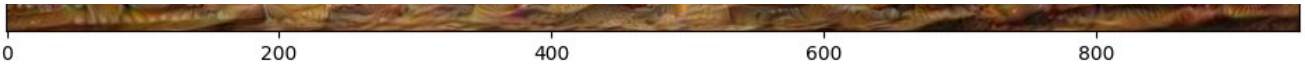

```
Step 200, loss 1.5107022523880005
```


```
Step 100, loss 1.8735296726226807
```


```
Step 200, loss 2.0152204036712646
```


```
Step 100, loss 1.9815341234207153
```

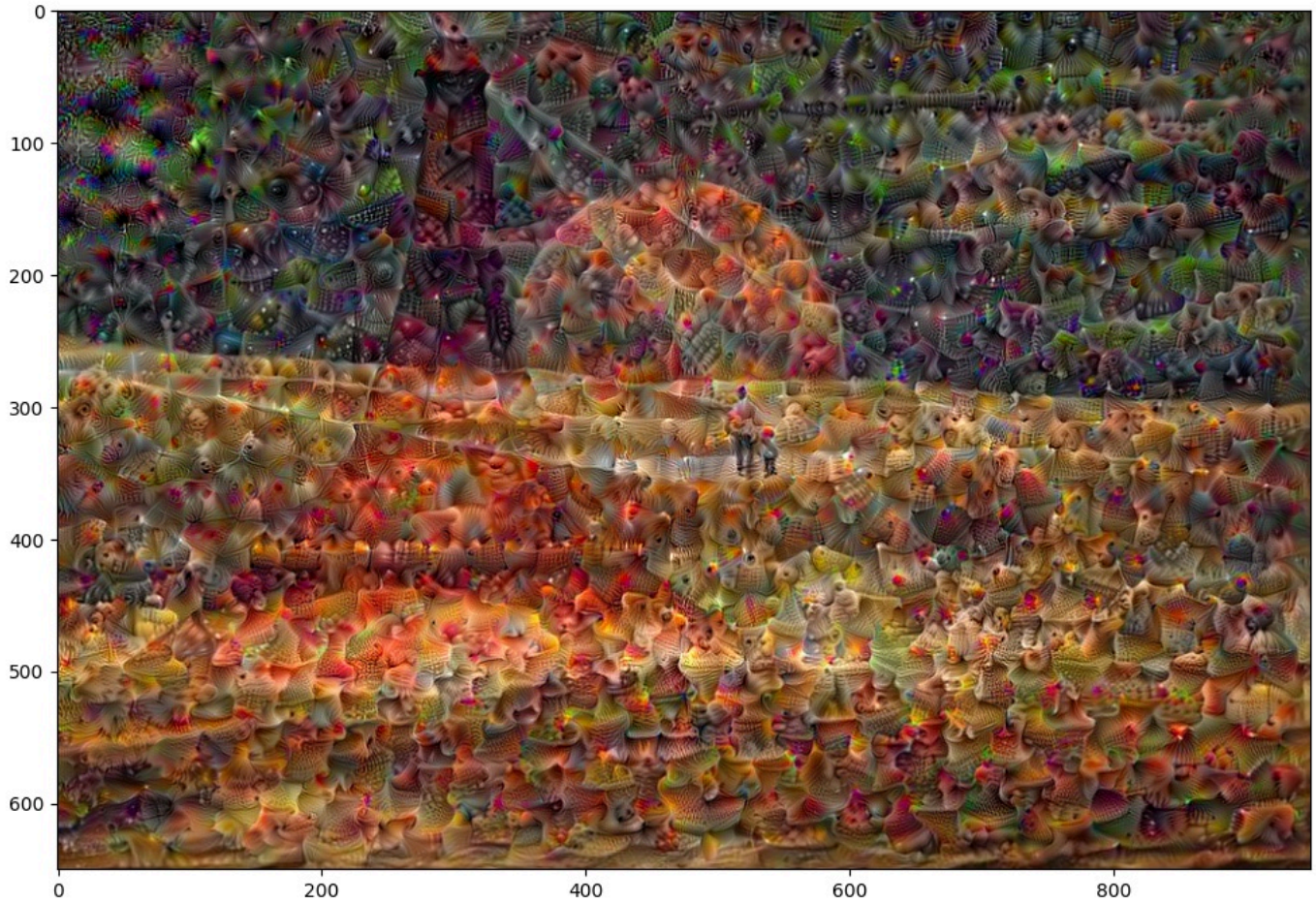

```
Step 200, loss 2.1248137950897217
```




```
Step 0, loss 1.2270585298538208
```

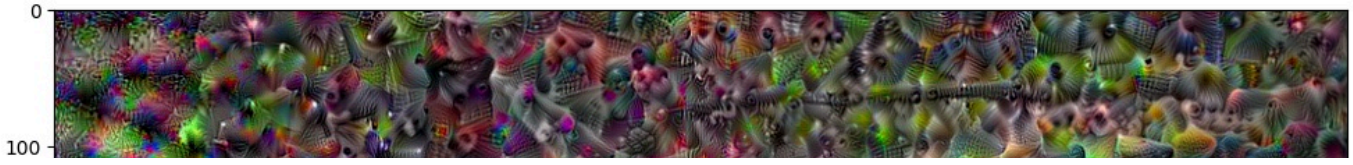
```
Step 100, loss 2.214198589324951
```


```
Step 200, loss 2.372666835784912
```

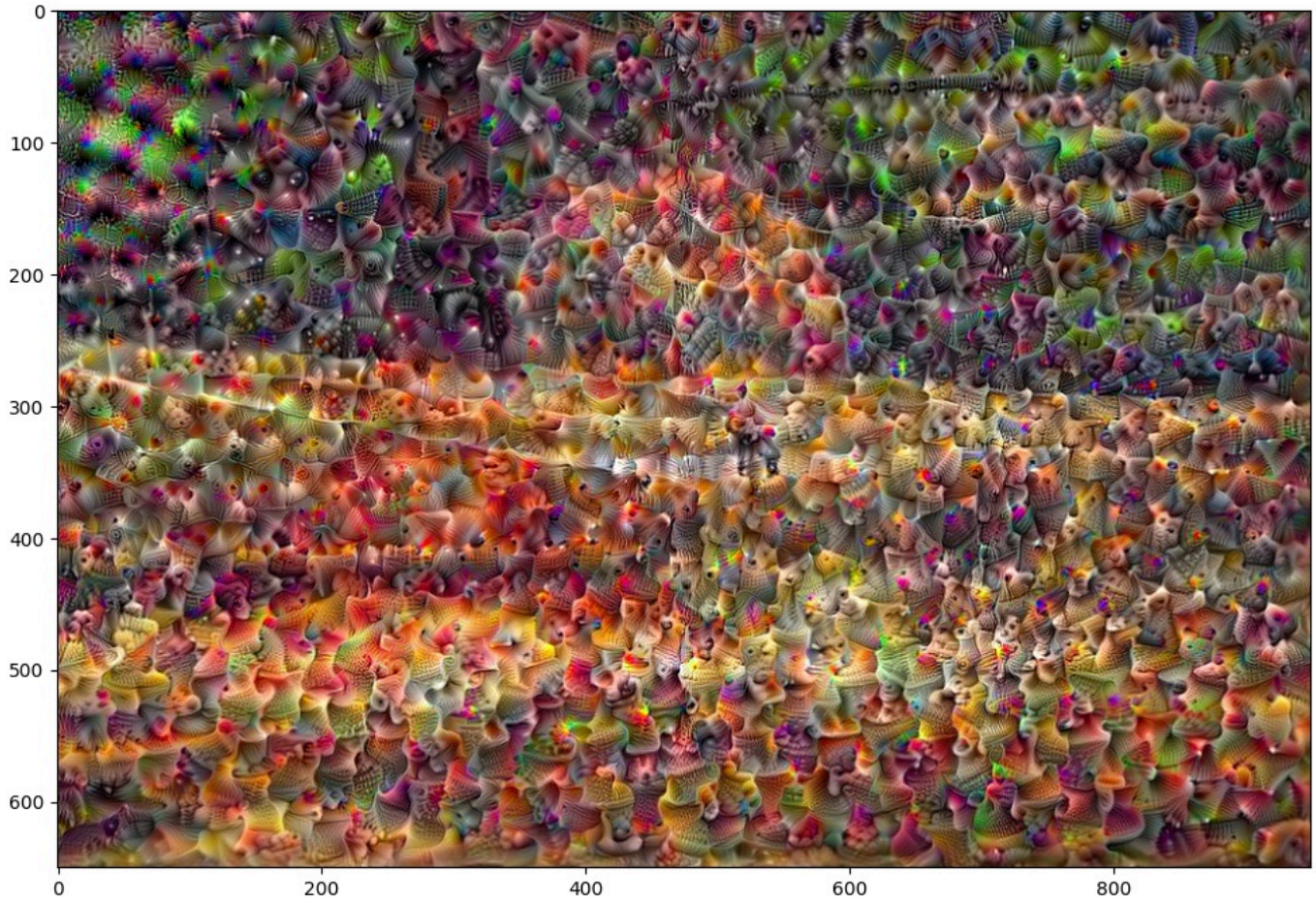



```
Step 200, loss 2.5864460468292236
```

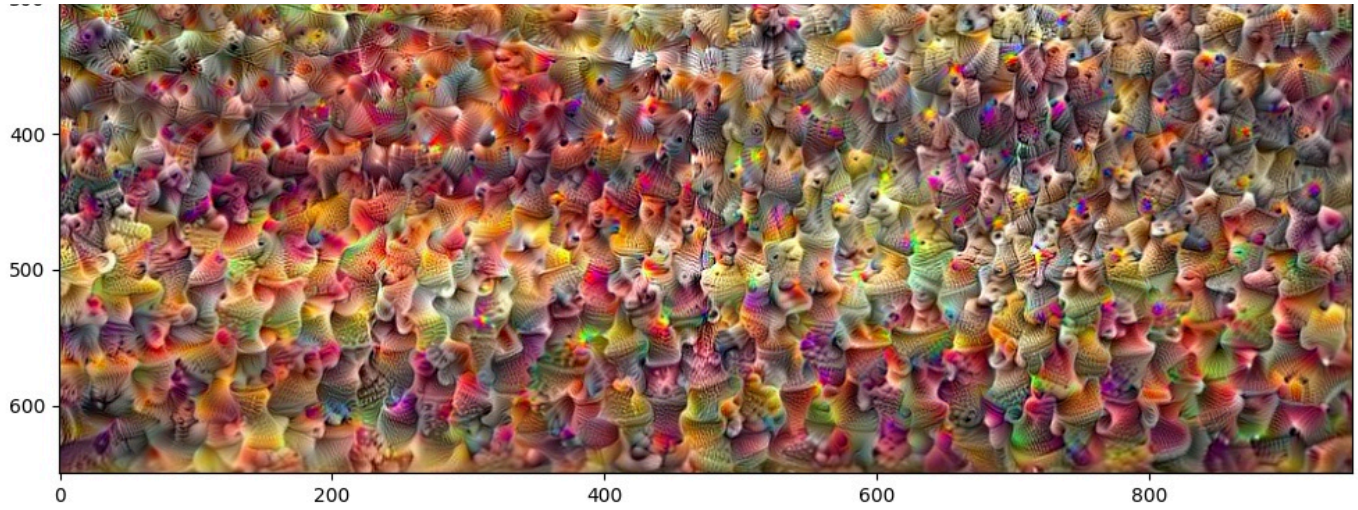

```
Step 300, loss 2.6898610591888428
```



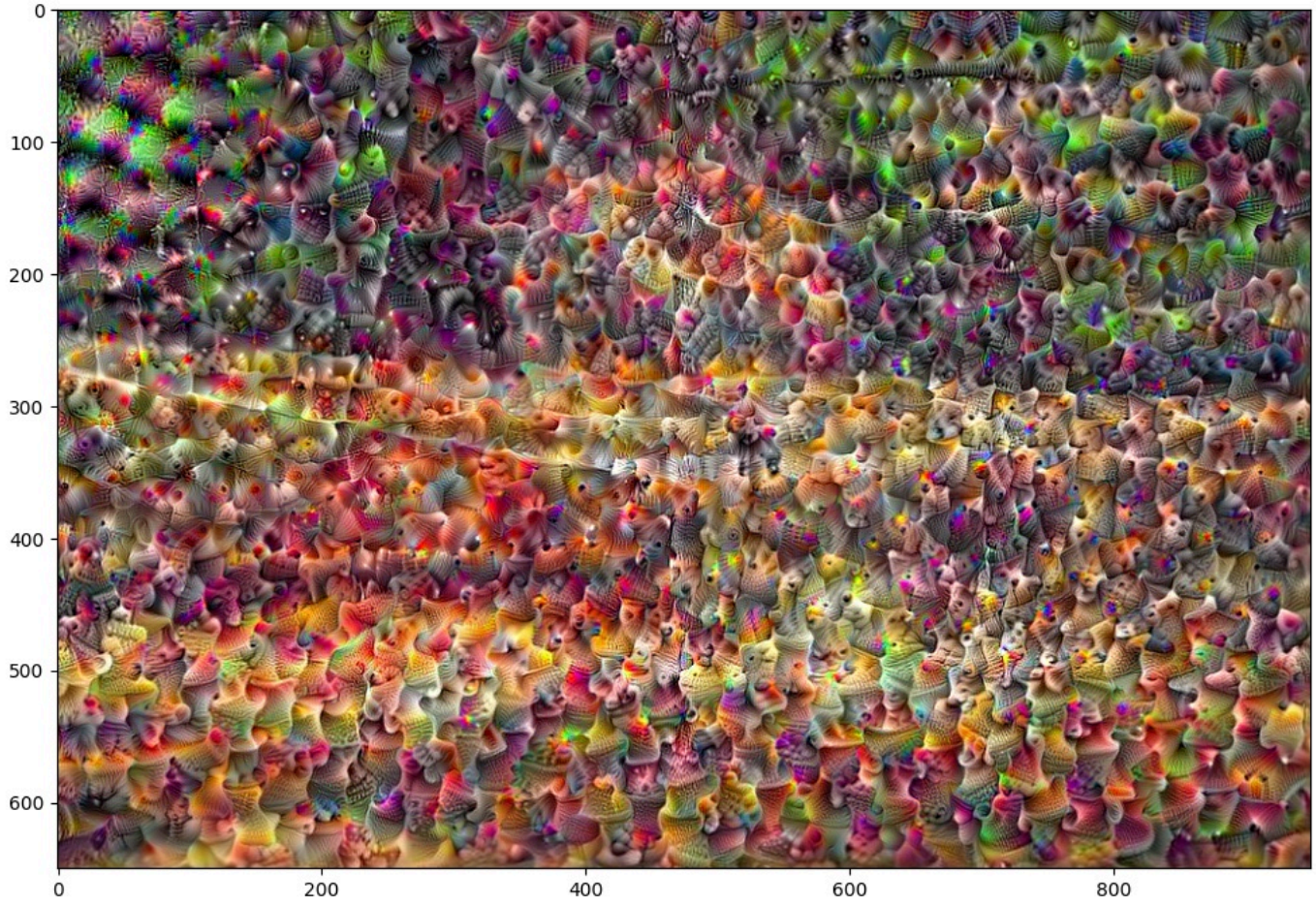

```
Step 200, loss 2.668839454650879
```

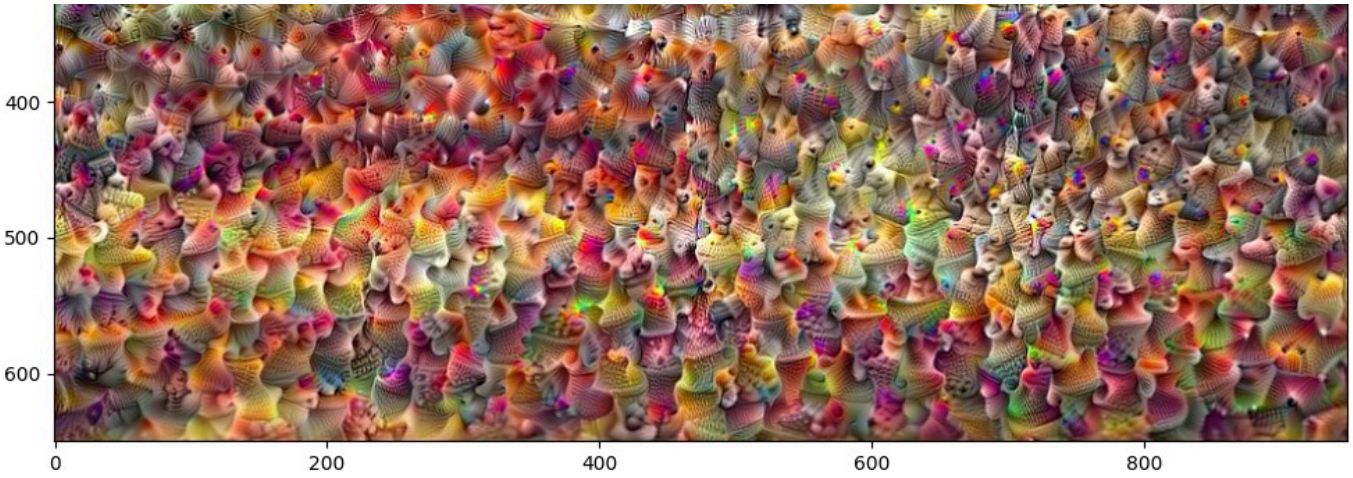



```
Step 300, loss 2.7744908332824707
```



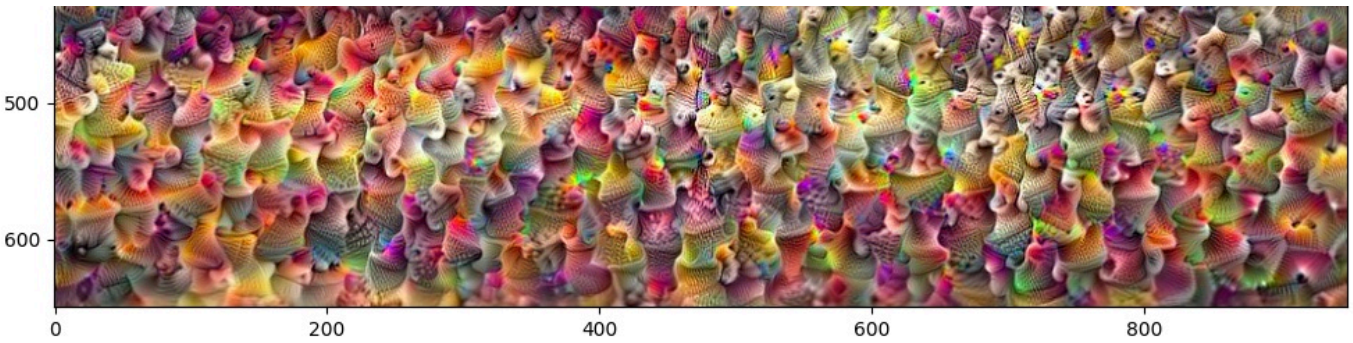

```
Step 400, loss 2.9625957012176514
```





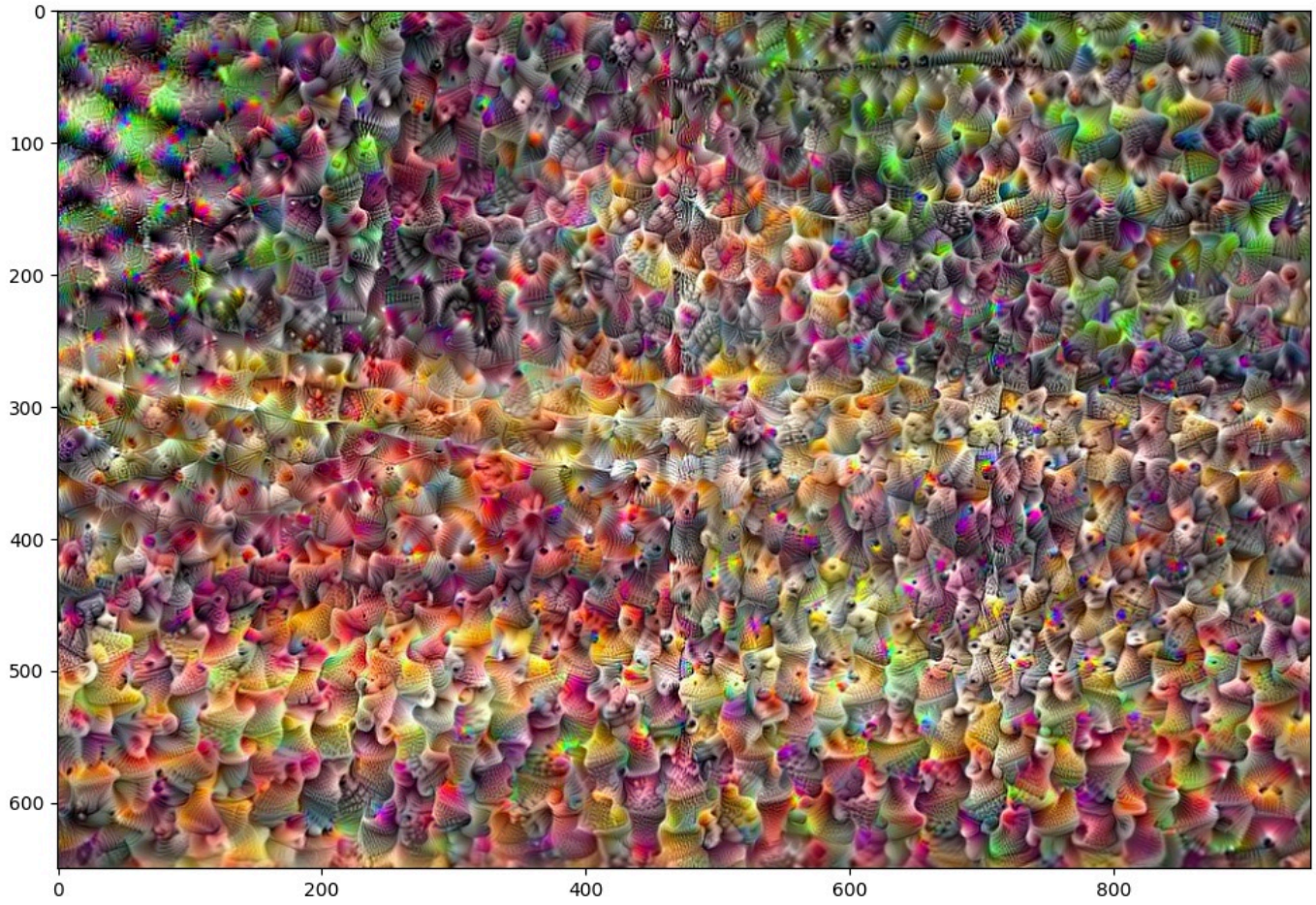

```
Step 200, loss 2.7808127403259277
```


```
Step 300, loss 2.891728401184082
```

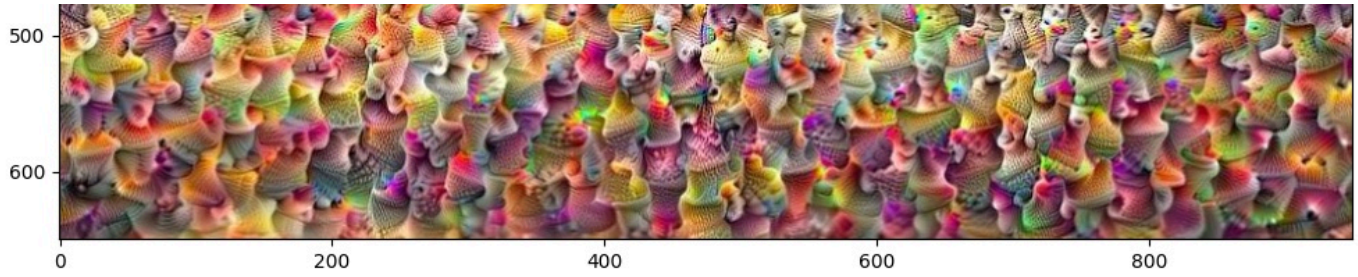



```
Step 400, loss 3.0301716327667236
```

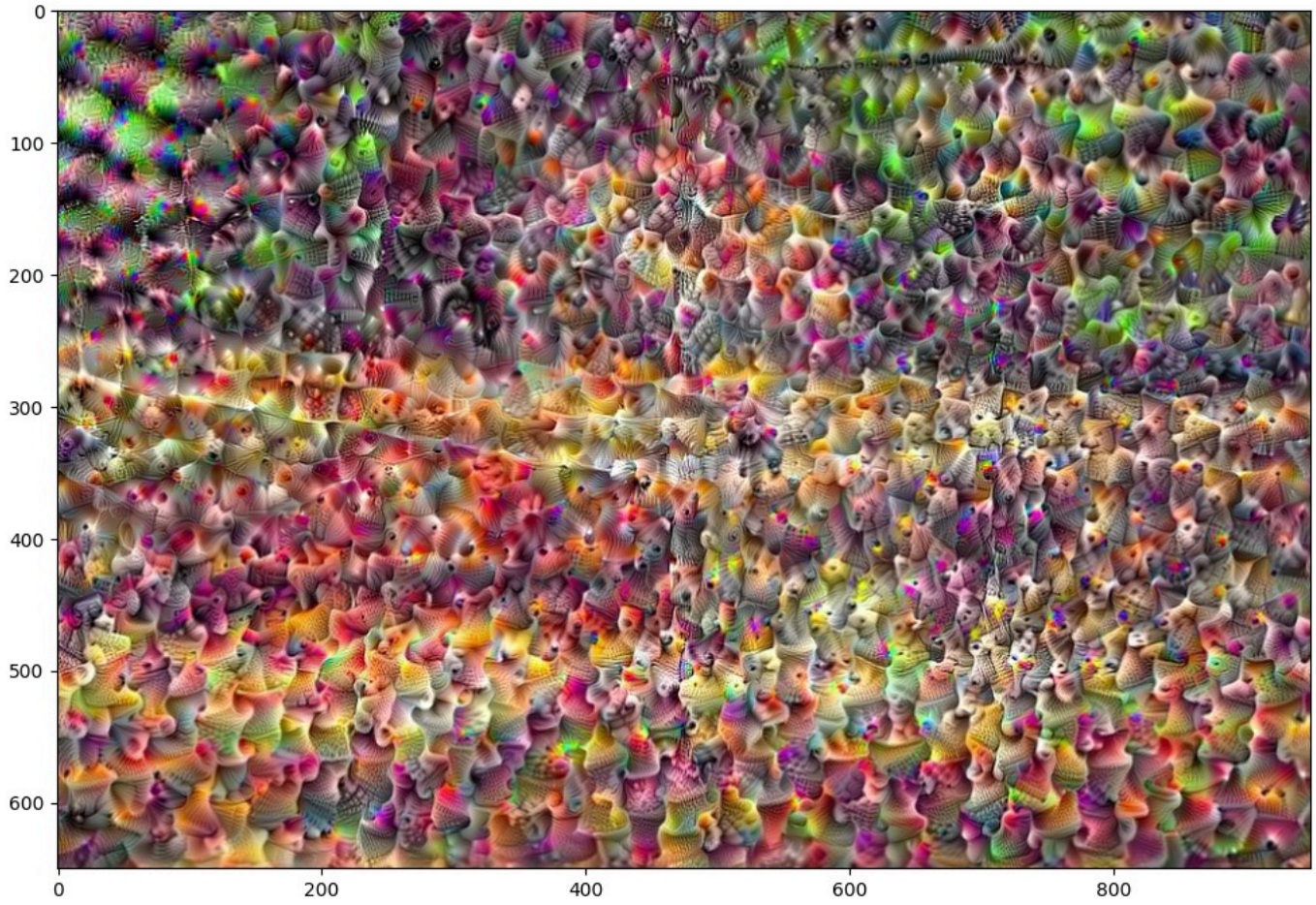

```
Step 200, loss 2.842731237411499
```



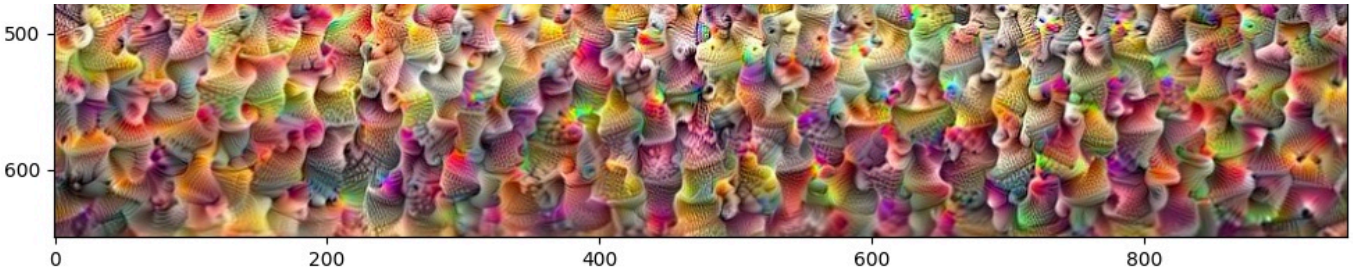

```
Step 300, loss 2.9596850872039795
```



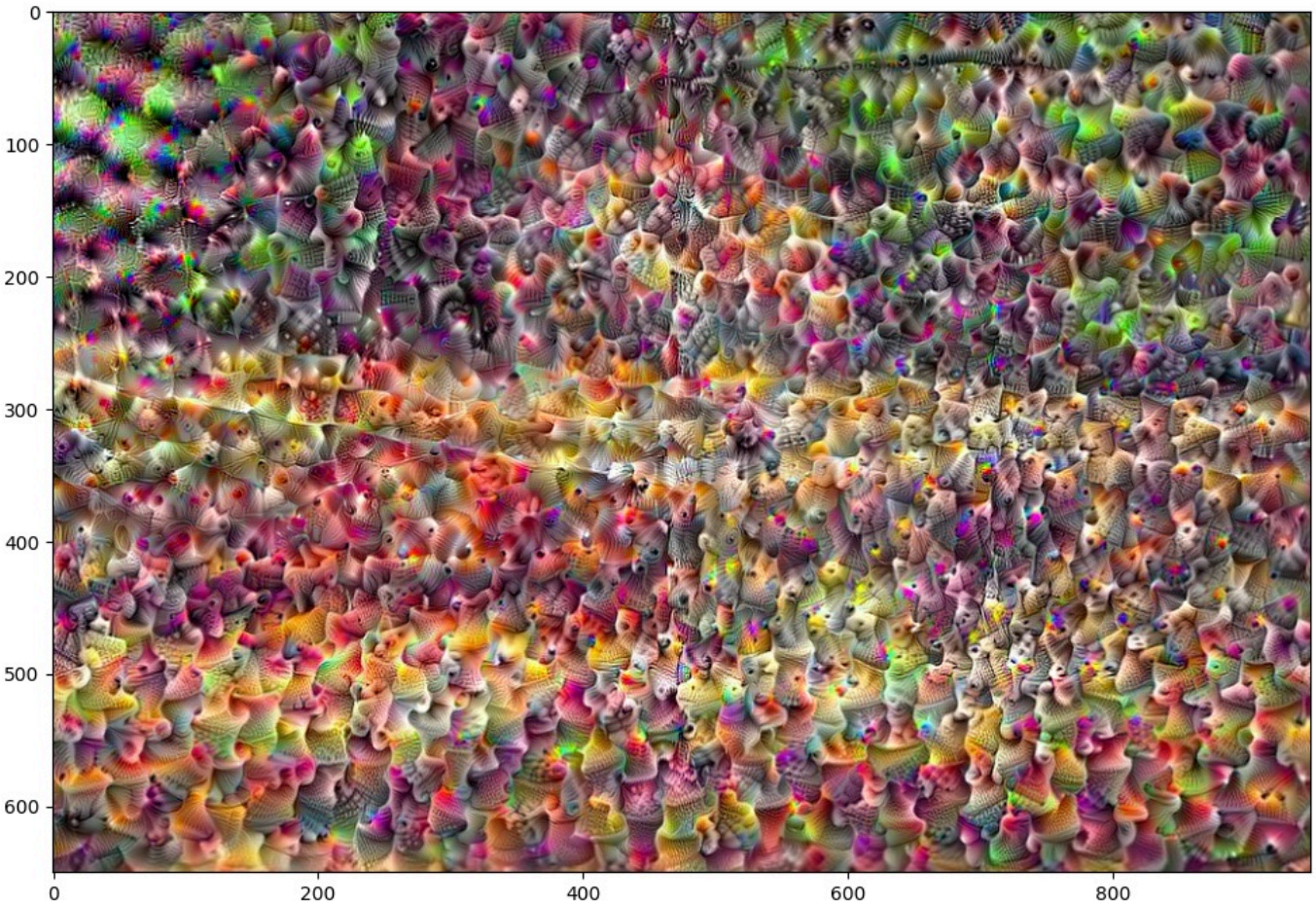

```
Step 0, loss 1.4508227109909058
```



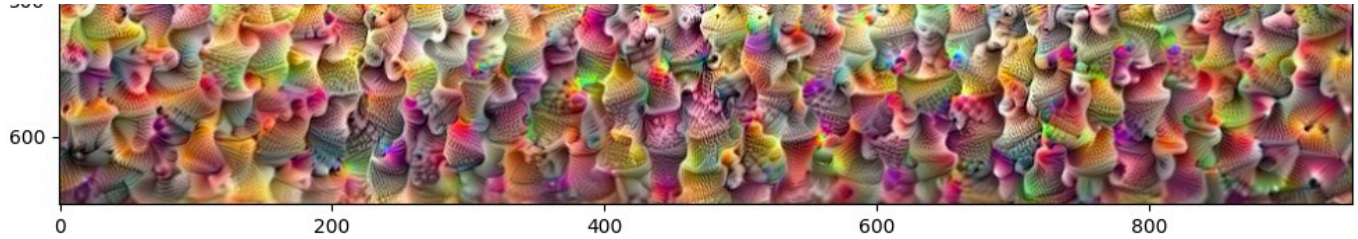

```
Step 100, loss 2.657562017440796
```



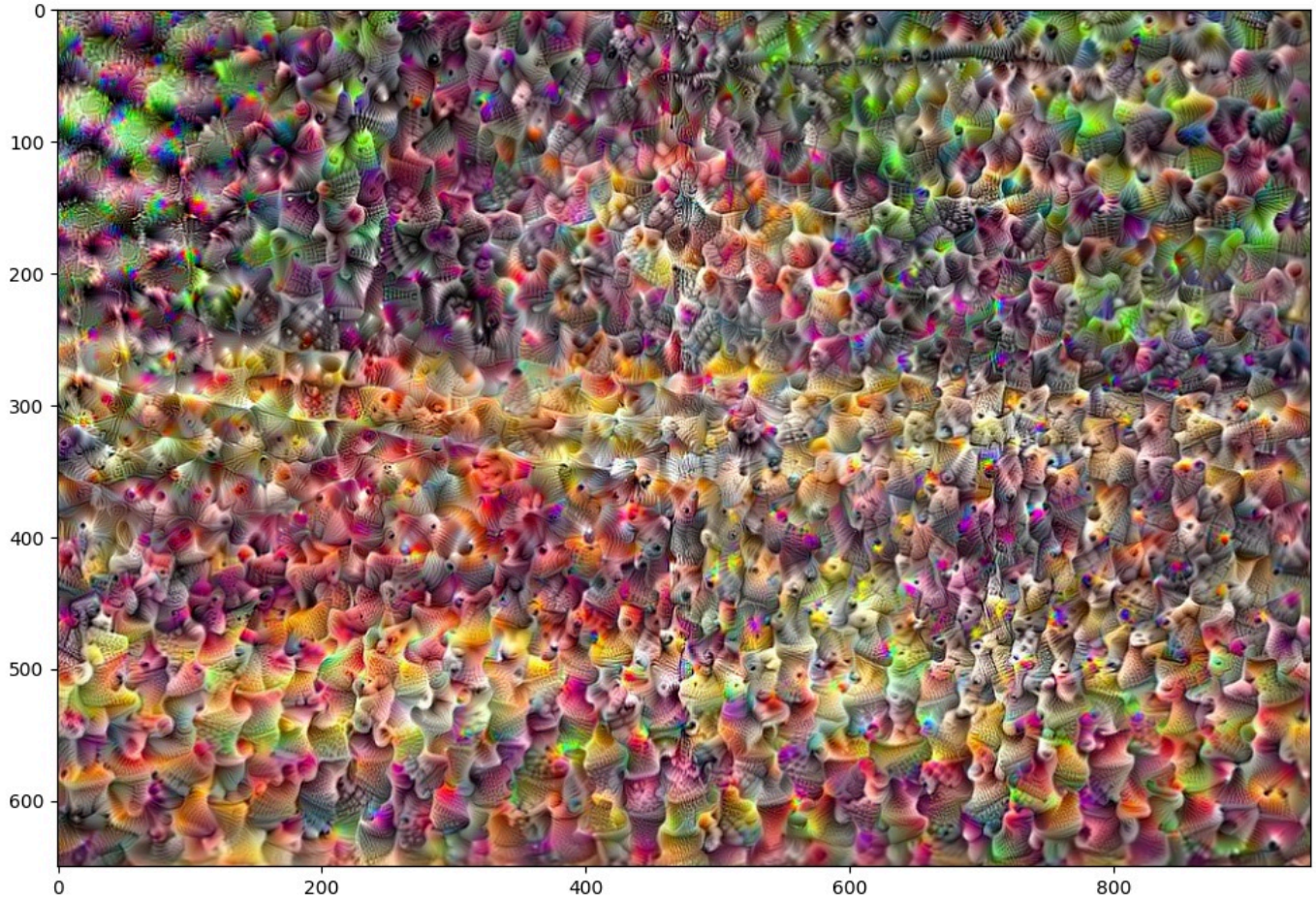

```
Step 200, loss 2.854915142059326
```



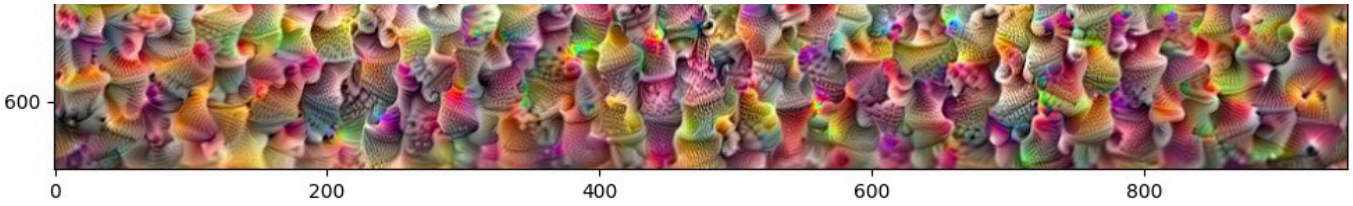

```
Step 300, loss 2.9706015586853027
```



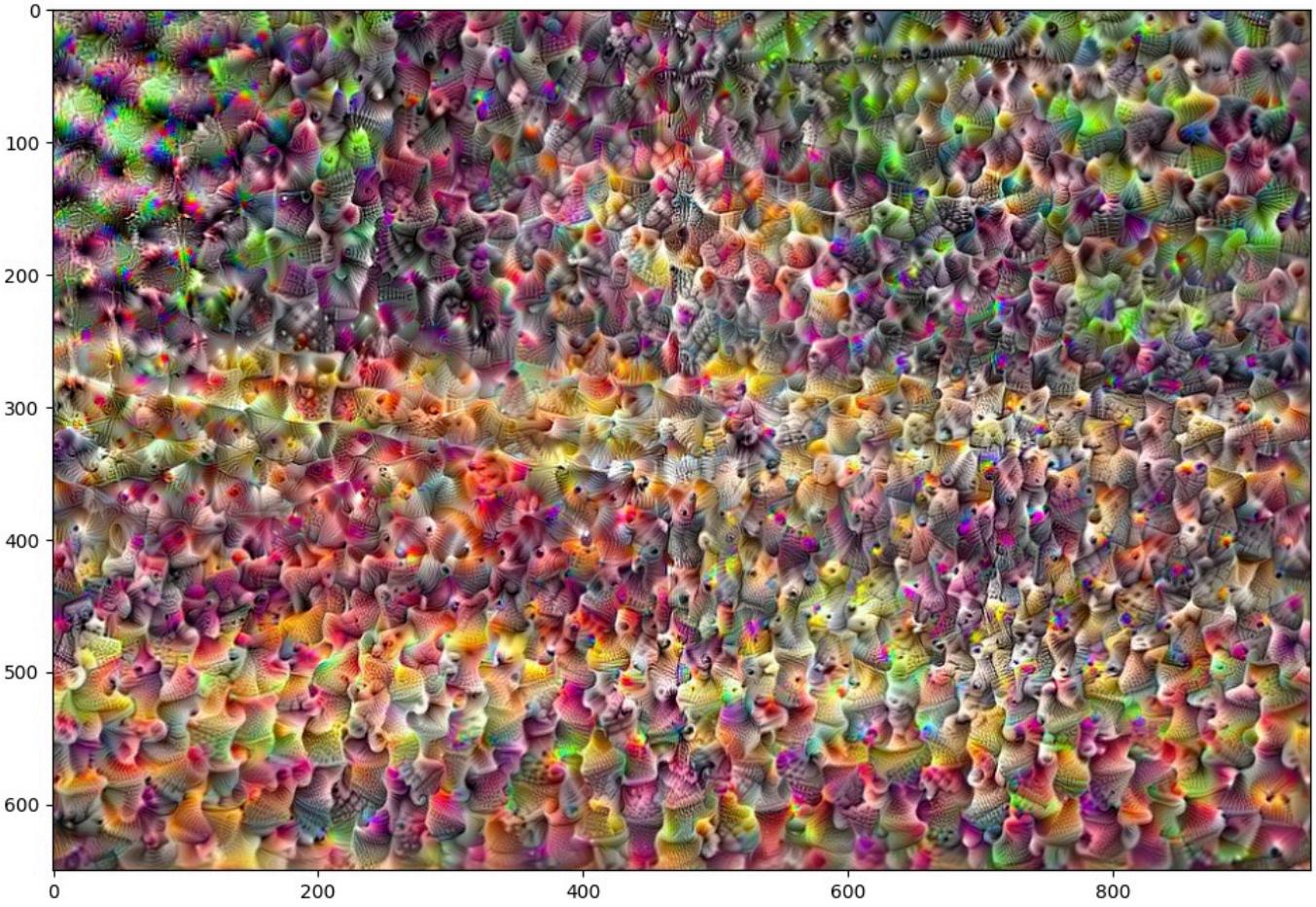

```
Step 200, loss 2.8588719367980957
```

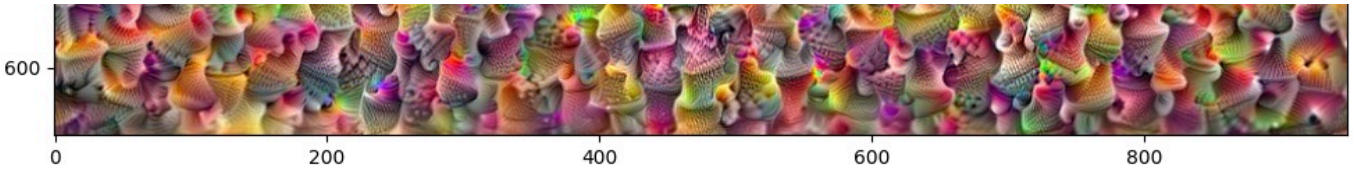



```
Step 300, loss 2.9752116203308105
```

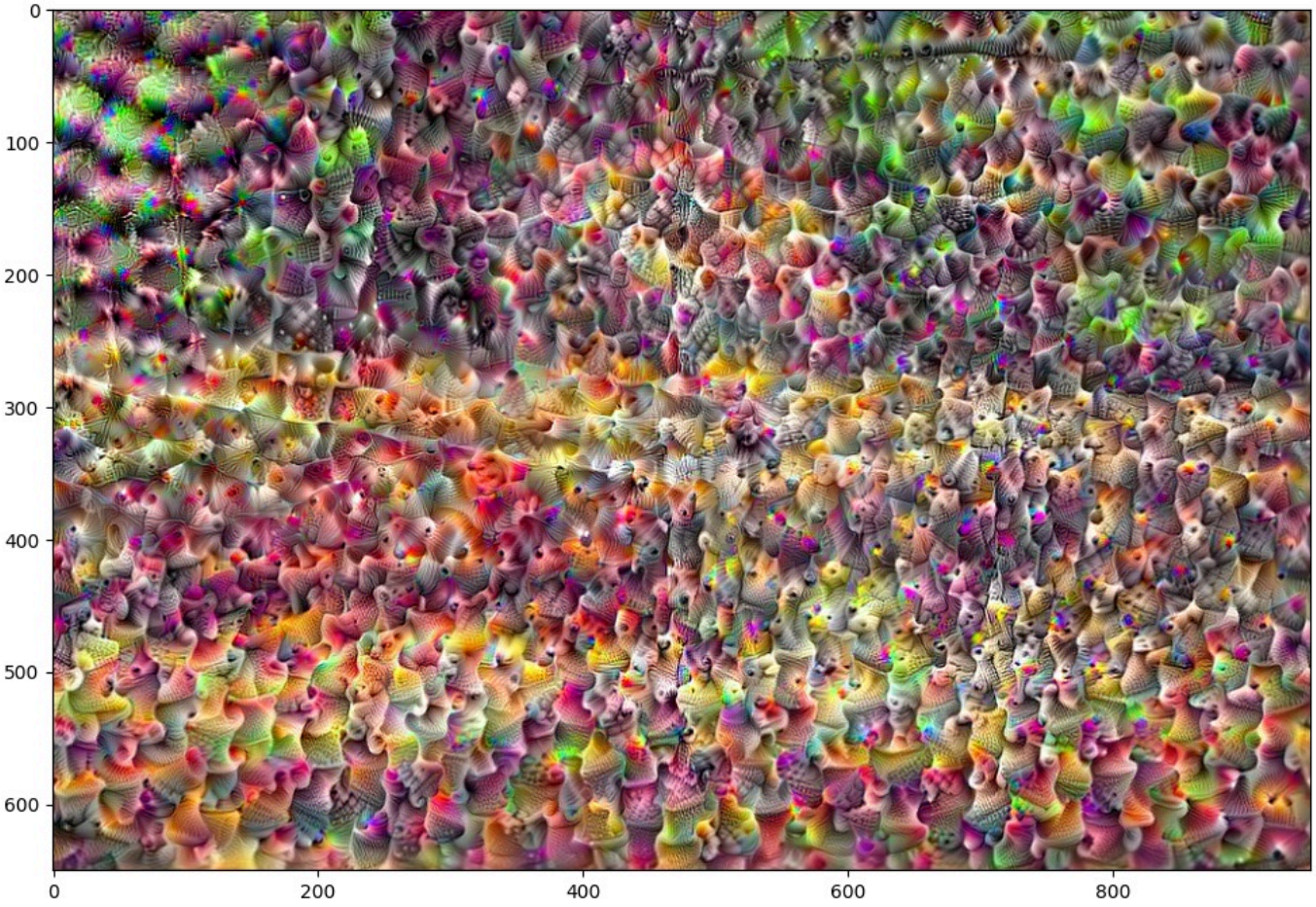



```
Step 400, loss 3.078650951385498
```

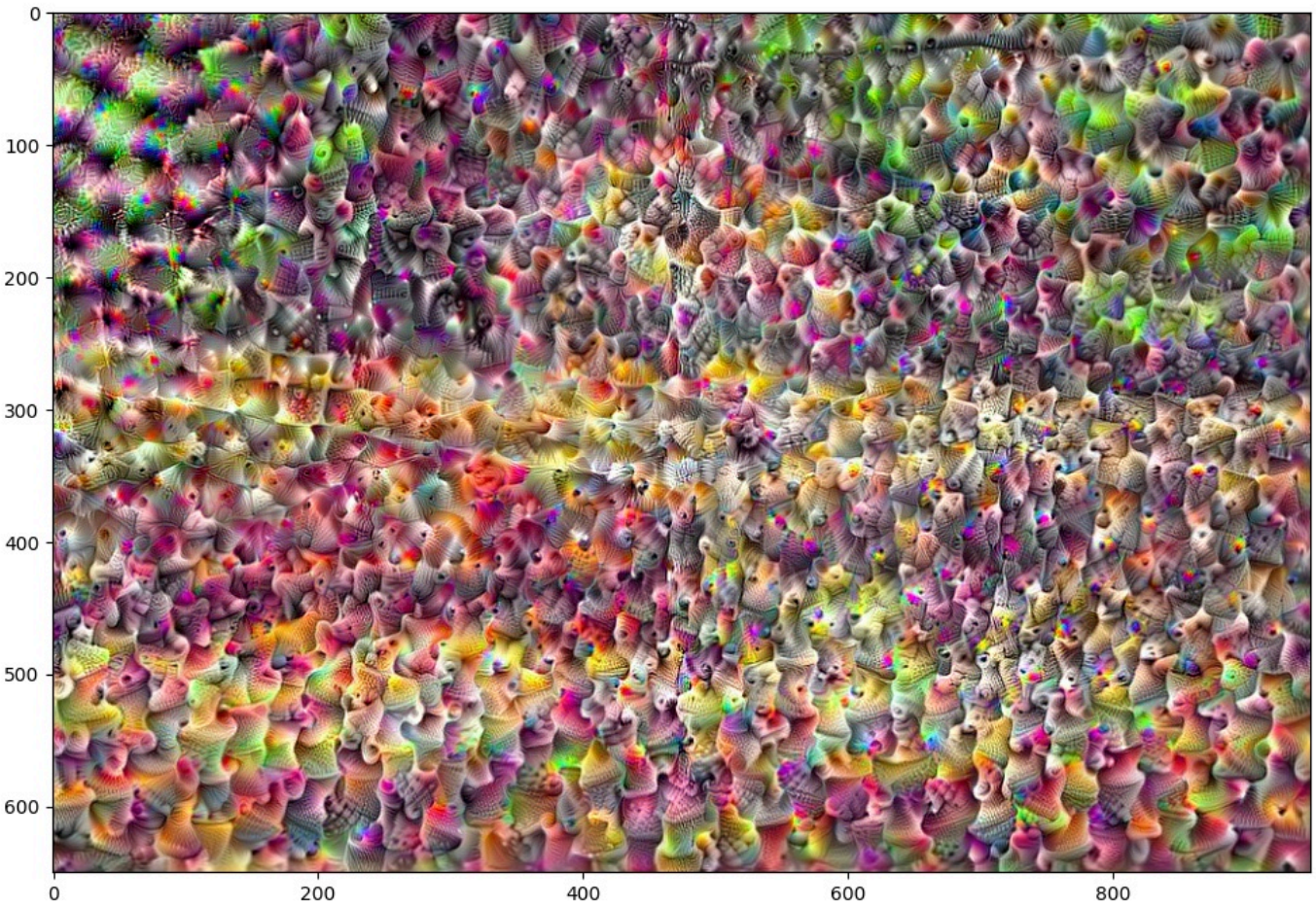




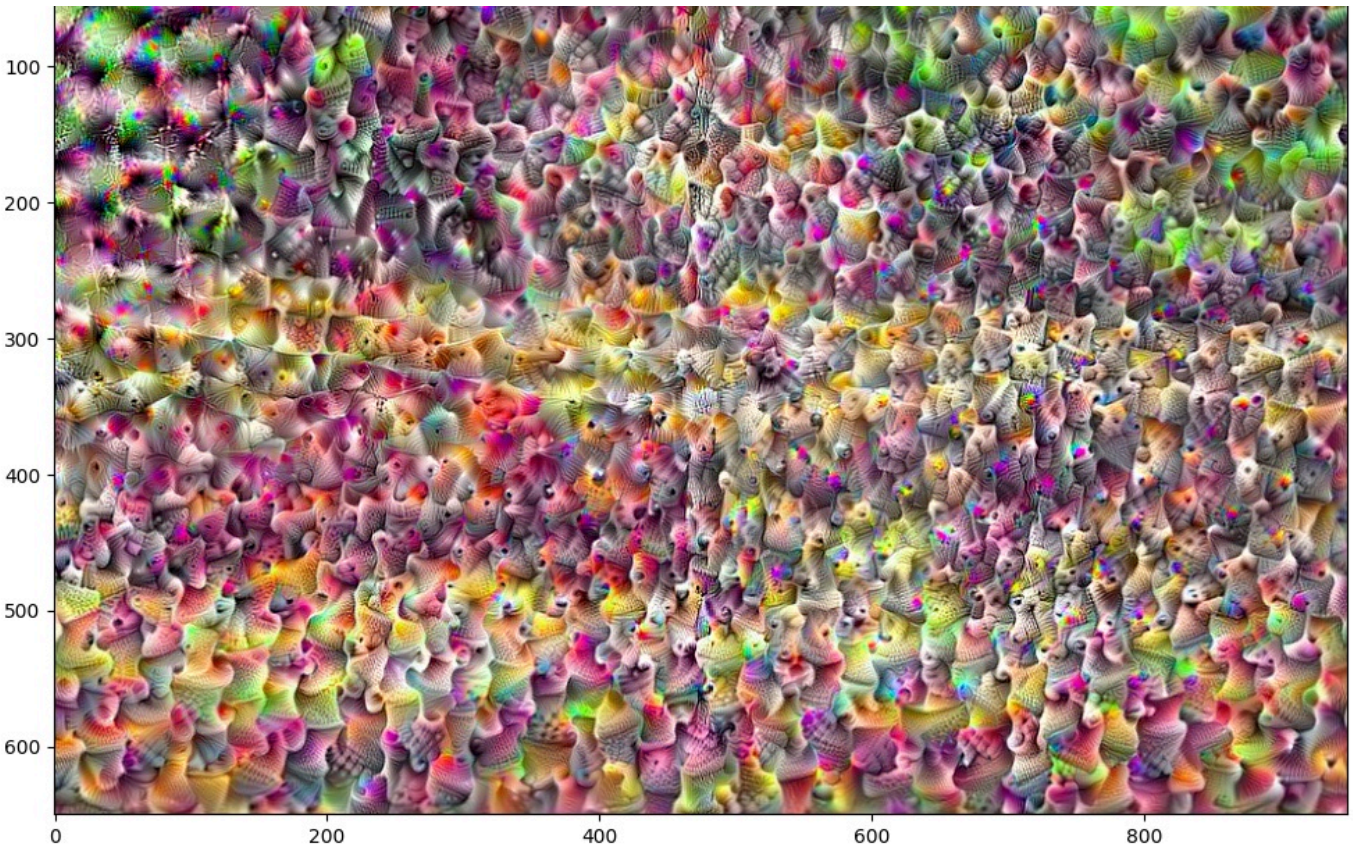

```
Step 400, loss 3.08994197845459
```



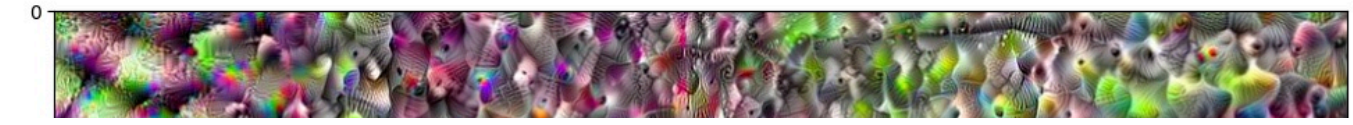
```
Step 400, loss 3.141254425048828
```




```
Step 0, loss 1.4775099754333496
```




```
Step 0, loss 1.4871001243591309
```

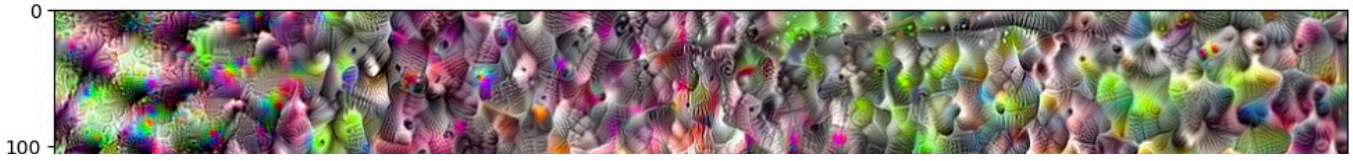



```
Step 100, loss 2.7789130210876465
```


```
Step 200, loss 2.9885005950927734
```


```
Step 300, loss 3.1171085834503174
```


```
Step 400, loss 3.2032694816589355
```



Part 10: Creating a Video From All The Frames in Part 9

```
path_dream = '/content/Creative AI Dataset/mars_eiffel'


#Define codec and create the VideoWriter object
#Download FFmeg to create the video
# Define the codec and create VideoWriter object
# Download FFmeg

fourcc = cv2.VideoWriter_fourcc(*'mp4v') # FourCC is a 4-byte code used to specify the video codec
```