Department of Computer Science and Engineering

CSC405A: Seminar

Designs and Patterns in Mathematics and Deep Learning, Fractals for Image Classification, and more ! (and what a failed paper taught me)

Name:	Satyajit Ghana
Reg No:	17ETCS002159
Dept:	CSE
Batch:	2017

Outline

- Introduction
- Motivation for choosing the Topic
- Presentation of Topic
- Student's opinion/perspective
- Relevance to society
- Conclusions
- References

HEADS UP  : This is going to be an interactive seminar

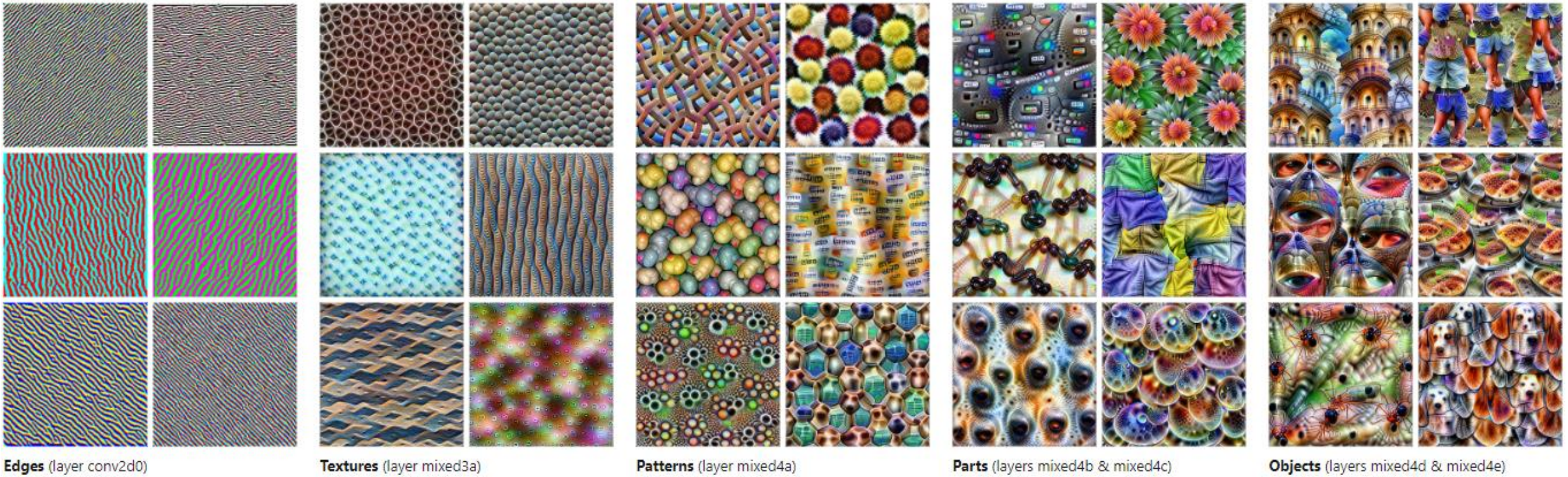
Introduction

- This Seminar is about Patterns, Designs in Mathematics and how they can be leveraged for better Image Classification
- Image Classification is the task of identifying a given Image.
- It is difficult to get a generalized model in Image Classification, hence we use Image Augmentation Techniques like Rotation, Scaling, Shifting etc. which helps in generating more examples and helping in building more generalized model.
- We are going to see how Fractals and Deep Learning based Image Generation work.

Motivation for choosing the topic

- I've been doing Deep Learning for quite some time now, from Classification, Segmentation to Generation, everything Vision !
- The most difficult part is to get the data in the correct augmented way such that the model does not overfit and generalizes well.
- Seeing Nature, and how beautiful patterns are, I had in mind of leveraging it to augment my data and making a better classifier from that augmentation.

Feature Visualization



Feature visualization allows us to see how GoogLeNet, trained on the ImageNet dataset, builds up its understanding of images over many layers.

Neural networks are, generally speaking, differentiable with respect to their inputs. If we want to find out what kind of input would cause a certain behavior — whether that's an internal neuron firing or the final output behavior — we can use derivatives to iteratively tweak the input towards that goal.

Translation: We can change the inputs to the layers such that the output from this layer has a specific output (in our case we maximize this output or in simple words we try to get the input which will make the layer's kernel most happy)

What if we try to directly create Patterns/Parts Layer?

Fractals

Fractals are infinitely complex patterns that are self-similar across different scales. They are created by repeating a simple process over and over in an ongoing feedback loop. Driven by recursion, fractals are images of dynamic systems – the pictures of Chaos.

Mandelbrot Set

- It's a very simple equation
- $f_c(z) = z^2 + c$
- Let's see how it plays with different values of x_0 and c
- x_0 is initial value of z
- c is some constant

Mandelbrot Set

```
[ ]  1 c = 0
     2 x0 = 0
     3 x = x0
     4 for iter in range(10):
     5     x = f(x, c)
     6     print(x)
```

```
0
0
0
0
0
0
0
0
0
0
```

Remains Finite

```
[ ]  1 c = 1
     2 x0 = 0
     3 x = x0
     4 for iter in range(10):
     5     x = f(x, c)
     6     print(x)
```

```
1
2
5
26
677
458330
210066388901
44127887745906175987802
1947270476915296449559703445493848930452791205
3791862310265926082868235028027893277370233152247388584761734150717768254410341175325352026
```

Explodes

Mandelbrot Set

```
[ ]  1 c = -3
     2 x0 = 0
     3 x = x0
     4 for iter in range(10):
     5     x = f(x, c)
     6     print(x)
```

```
-3
6
33
1086
1179393
1390967848446
193479155541049442461 4913
37434183628877603174075412715593584918683419975 66
14013181039605279591264794847619800328827486486508228624585427223970679113386904690548911149924353
19636924284875290450457734975738303761298515391392640369512757169616146185867159037867617261318310050851596568815927129039874491281406331270877
```

Explodes

```
[ ]  1 c = -0.28
     2 x0 = 0
     3 x = x0
     4 for iter in range(20):
     5     x = f(x, c)
     6     print(x)
```

```
-0.28
-0.2016
-0.23935744000000003
-0.2227080159166464
-0.2304011396464708
-0.22691531484960747
-0.22850943988670352
-0.22778343588266506
-0.22811470633748784
-0.2279636807525617
-0.22803256025774415
-0.22800115146229832
-0.22801547493186614
-0.22800894319159554
-0.2280119218246518
-0.2280105635058289
-0.2280111829297544
-0.2280109004589741
-0.22801102927188782
-0.22801097053037434
```

Finite but chaotic

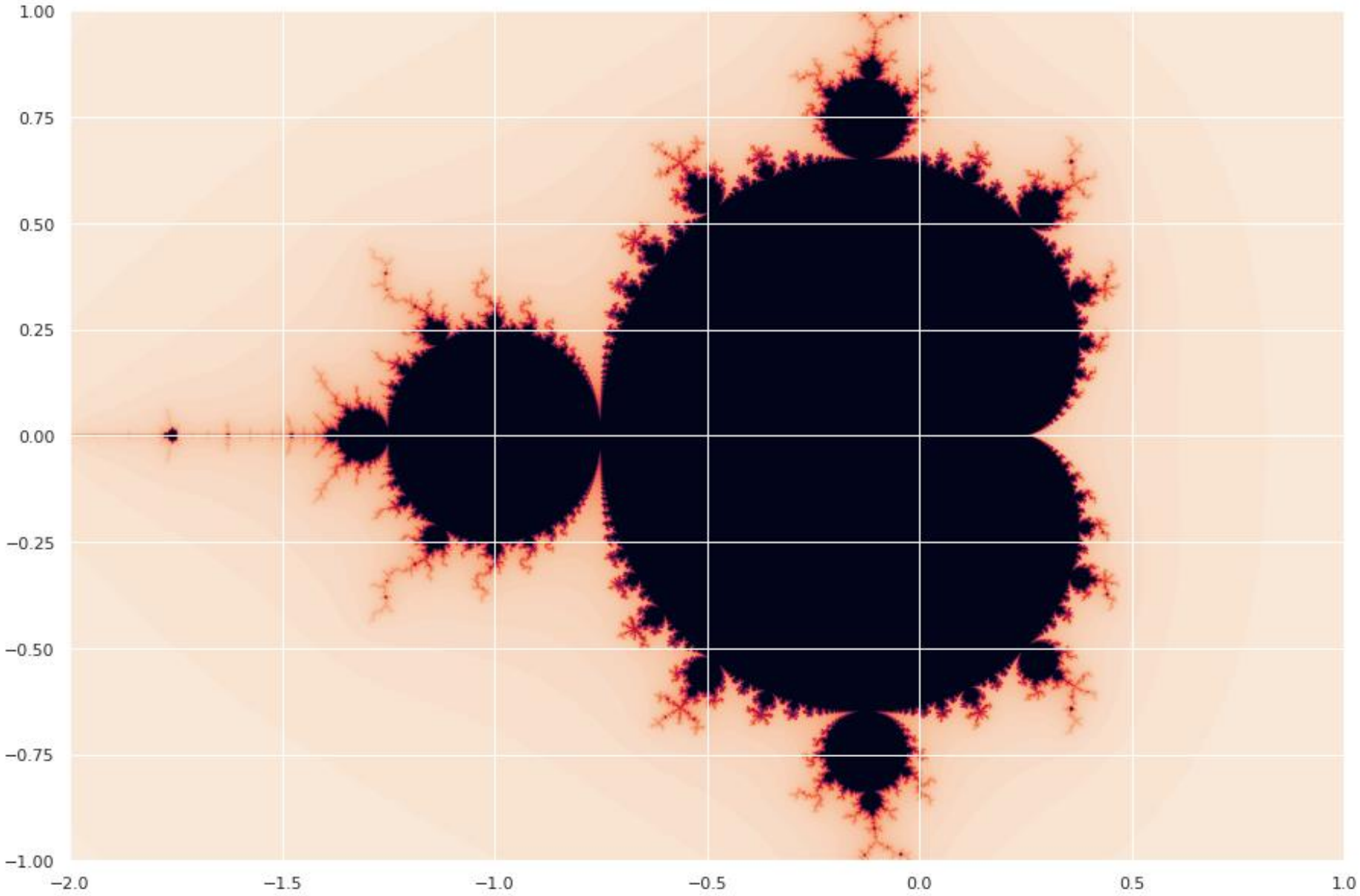
What if we do something with how quickly it explodes

Mandelbrot Set

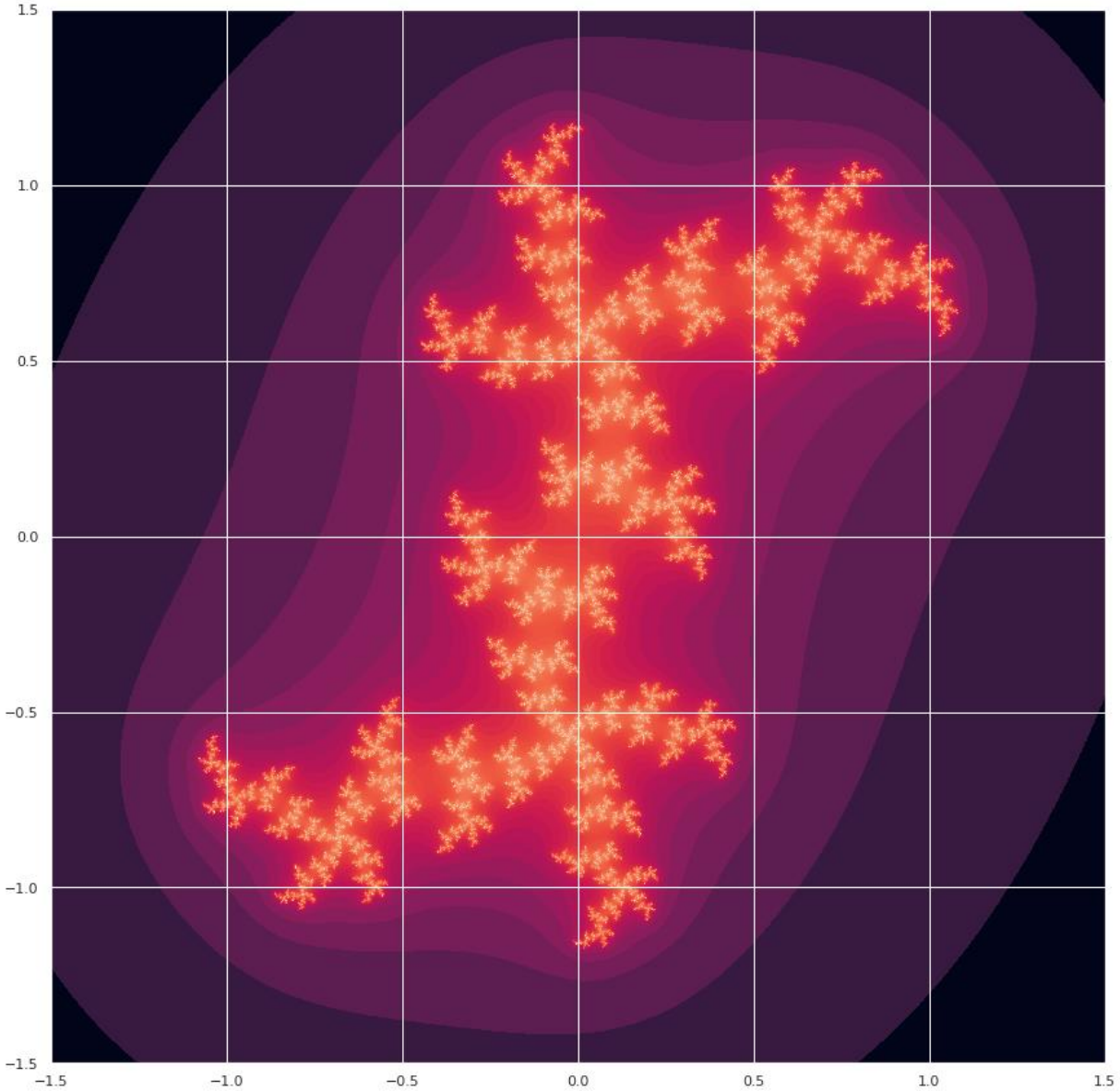
- Create a Grid [-2, 1] with 0.001 delta
- $x_0 = 0$
- $c = \text{grid}$
- Assign the Color based on how quickly it exploded

```
[ ]  1 x_min, x_max = -2, 1
     2 y_min, y_max = -1, 1
     3
     4 delta = 0.001
     5
     6 re, im = np.mgrid[x_min:x_max:delta, y_min:y_max:delta]
     7
     8 c = (re + 1j*im).T
     9 escape = np.zeros_like(np.abs(c))
    10 x0 = np.zeros_like(c)
    11 x = x0
    12
    13 for i in range(100):
    14     x = f(x, c)
    15     idx = (np.abs(x) > 10) & (escape == 0)
    16     # print(idx, idx.shape, c.shape)
    17     # break
    18     escape[idx] = 100 - i
```

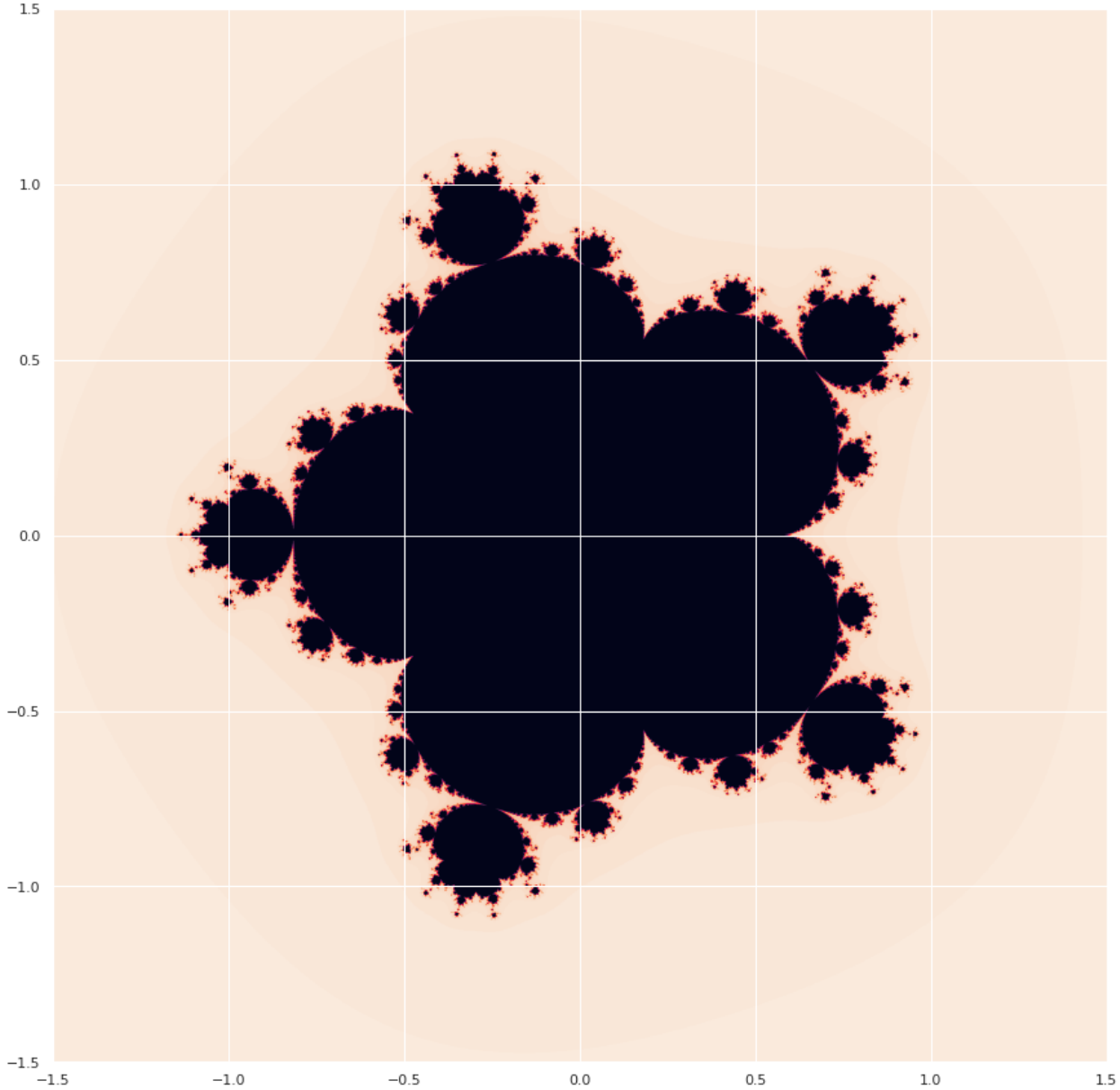
Result



With $C = 0.3 + 0.6j$



With $f(z) = z^6 + c$



Ehh but we deviated from our main topic still it was pretty cool right ?

What if we can create Fractals out of Images ?

Orbit Trap Fractals

In mathematics, an orbit trap is a method of colouring fractal images based upon how **close an iterative function**, used to create the fractal, approaches a geometric shape, called a "trap". Typical traps are points, lines, circles, flower shapes and **even raster images**. Orbit traps are typically used to **colour two dimensional fractals** representing the complex plane.

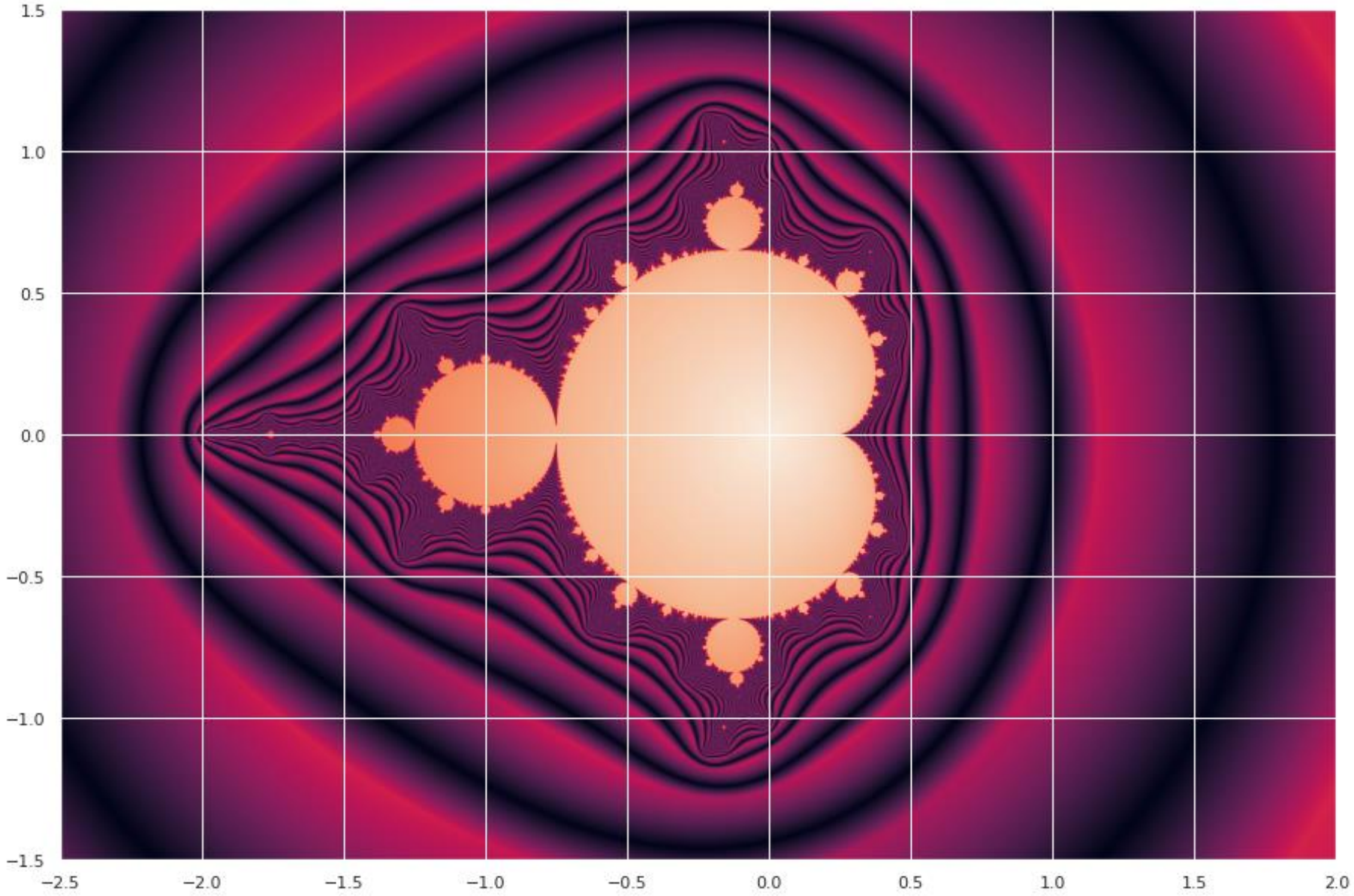
Using Distance function of Circle

```
[ ]  1 # shortest distance of a point from a circle
     2 def circle_dist(x, rad = 5.0):
     3     return np.abs(np.abs(x) - rad)
```

```
[ ]  1 def create_orbit_fractal(c, deg = 2, dist_func = circle_dist):
     2     x0 = np.zeros_like(c) # x0 = 0
     3     x = x0
     4
     5     distances = np.ones_like(np.abs(c)) * 10
     6
     7     for iter in range(100):
     8         x = f(x, c, deg=deg)
     9
    10         z_minus = x.copy()
    11         z_minus = dist_func(z_minus)
    12         idx = np.abs(z_minus) < 10
    13         distances[idx] = np.minimum(distances[idx], np.abs(z_minus[idx]))
    14
    15     return distances
    16
```

$$\text{Shortest Distance from Circle} = \left|\sqrt{x^2 + y^2} - c\right|$$

Result



Orbit Trap Fractal

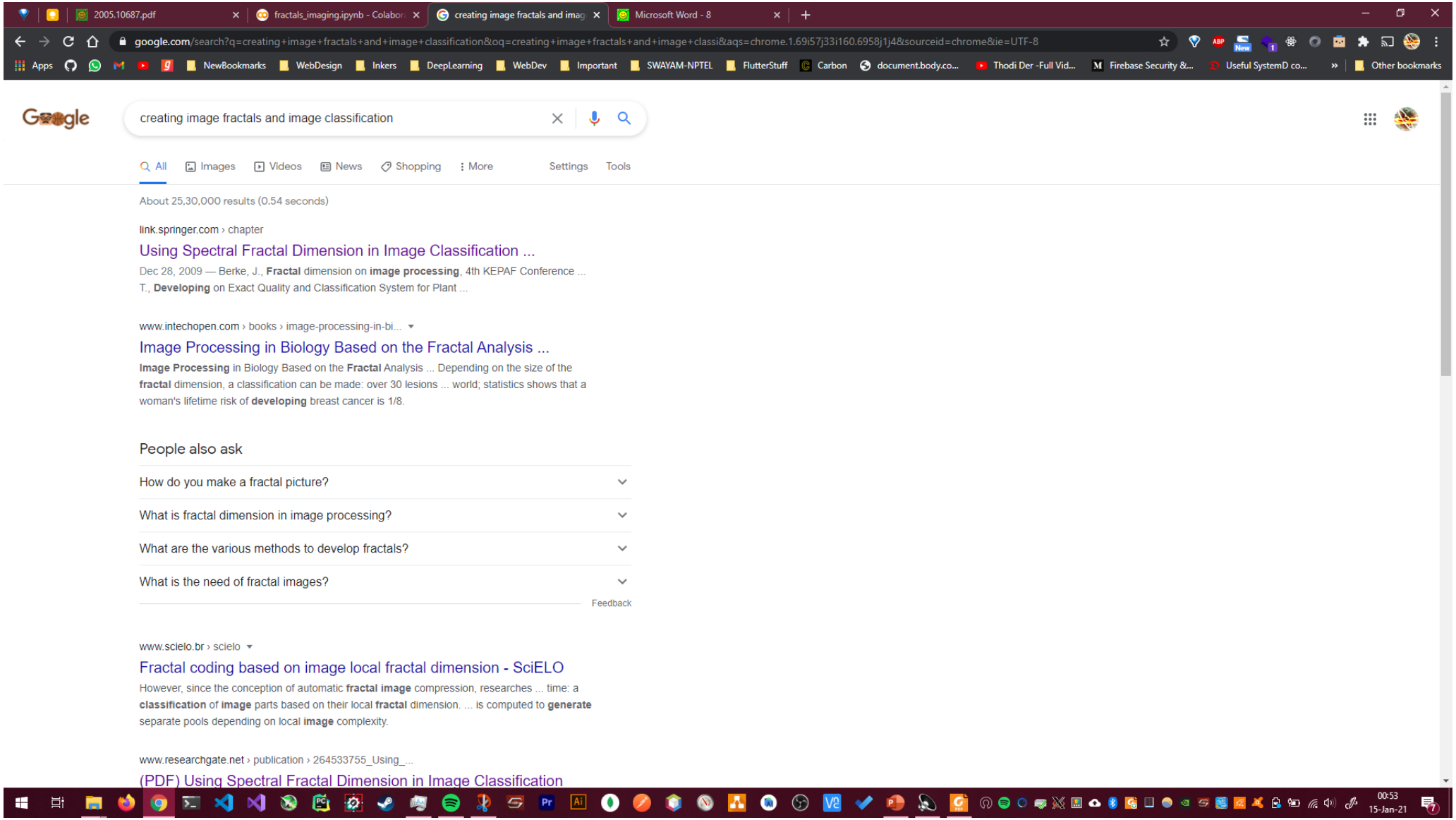
```python
def create_image_fractal3(c, image, deg = 2):
    x0 = np.zeros_like(c) # x0 = 0
    x = x0

    btmleft = (-0.5, -0.5)
    topright = (0.5, 0.5)
    trap_size = (1.0, 1.0)

    fractal = np.zeros((*(np.abs(c).shape), 3))
    trapped = np.zeros_like(np.abs(c))

    with np.nditer(x, flags=['multi_index'], op_flags=['readwrite']) as it:
        for point in tqdm(it, total=x.size):
            for iter in range(100):
                point[...] = f(point, c[it.multi_index], deg)

                # if this point was trapped previously then break
                if (abs(point) > 10) or trapped[it.multi_index] == 1:
                    break

                # check if this point can be trapped
                if (point.real > btmleft[0] and point.real < topright[0]) and (point.imag > btmleft[1] and point.imag < topright[1]):
                    trapped[it.multi_index] = 1
                    fractal[it.multi_index] = np.array(
                            image.getpixel(
                                (
                                    int( ( point.real + 0.5 ) * image.width ),
                                    int( ( point.imag + 0.5 ) * image.height  )
                                )
                            )
                        )

    return fractal
```

Result

What if we used this Image for a Classification Task ?

- From our previous knowledge we know that channels try to create patterns for the objects it wants to detect
- Fractals are Naturally Occurring Patterns
- In an way we can imagine like we are try to do half of the task for the Model by directly feeding it the pattern from the input stage
- It should result in better accuracies Right?
- Righttt ???

I couldn't find a paper that was trying to do this



What I found

- Fractal Analysis
  - Fractal analysis is assessing fractal characteristics of data. It consists of several methods to assign a fractal dimension and other fractal characteristics to a dataset which may be a theoretical dataset, or a pattern or signal extracted from phenomena including natural geometric objects, ecology and aquatic sciences, sound, market fluctuations, heart rates, frequency domain in electroencephalography signals, digital images, molecular motion, and data science. Fractal analysis is now widely used in all areas of science.
- But this is not what I am trying to do !

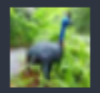
Do you think this would make a really good paper?

Image Classifier using Fractal Augmentation

I used the Fractal Augmentation to Generate CIFAR10 Fractal Dataset

CIFAR10 Peacock

Mandelbrot
CIFAR10 Peacock

```
[71]: 1 train_iter = iter(train_dataset)

[88]: 1 img = next(iter(train_iter))[0]
      2 img

[88]: 

[95]: 1 trans_img = Image.fromarray(transform(image=np.array(img))['image'])
      2 trans_img

[95]: 
```

Training a Simple Model WITH Fractal Augmentation

[45]:

```
dm = CIFAR10MandelbrotDataModule(
    data_dir="../dev_nbs/cifar10_mandelbrot",
    transform=A.Compose(
        [
            A.Resize(96, 96),
            A.Normalize(mean=CIFAR10.mean, std=CIFAR10.std),
            AT.ToTensor(),
        ]
    ),
    test_transform=A.Compose(
        [
            A.Resize(96, 96),
            A.Normalize(mean=CIFAR10.mean, std=CIFAR10.std),
            AT.ToTensor(),
        ]
    ),
)
model = LitModel(3, 96, 96, dm.num_classes, hidden_size=64)
trainer = pl.Trainer(max_epochs=1, progress_bar_refresh_rate=20)
trainer.fit(model, dm)
```

Training a Model WITHOUT any augmentations

[37]:

```
dm = CIFAR10DataModule(
    transform=A.Compose(
        [
            A.Resize(32, 32),
            A.Normalize(mean=CIFAR10.mean, std=CIFAR10.std),
            AT.ToTensor(),
        ]
    )
)
model = LitModel(*dm.size(), dm.num_classes, hidden_size=32)
trainer = pl.Trainer(max_epochs=1, progress_bar_refresh_rate=20)
trainer.fit(model, dm)
```

Who would win ? What could be the test Accuracies of each of them ? Any Guesses ?

	With Fractal Augmentation	Without Fractal Augmentation
Validation Accuracy	21.21%	41.60%
Test Accuracy	18.13%	41.43%
Parameters	1.8M	394K
Epochs	1	1

Why I didn't proceed further

- Fractals require more processing power than usual augmentations.
- For getting descent results (if any) it would takes much more time and energy
- Instead I could get away with better accuracies simply by using augmentation techniques like rotate, shear, scale, cut out !
- This wasn't a total waste of time though !

Couldn't I have avoided so much effort ? If someone had published a paper on this with their findings?

Relevance to Society

- Fractals like Orbit Trap Fractals are used widely to generate uniquely generating patterns and using them for printing on clothes.
- Marlies Bugmann from Tasmania is using the orbit trap fractal generator to make scarves. Her wearable art can be seen at tasmanianartist.yolasite.com

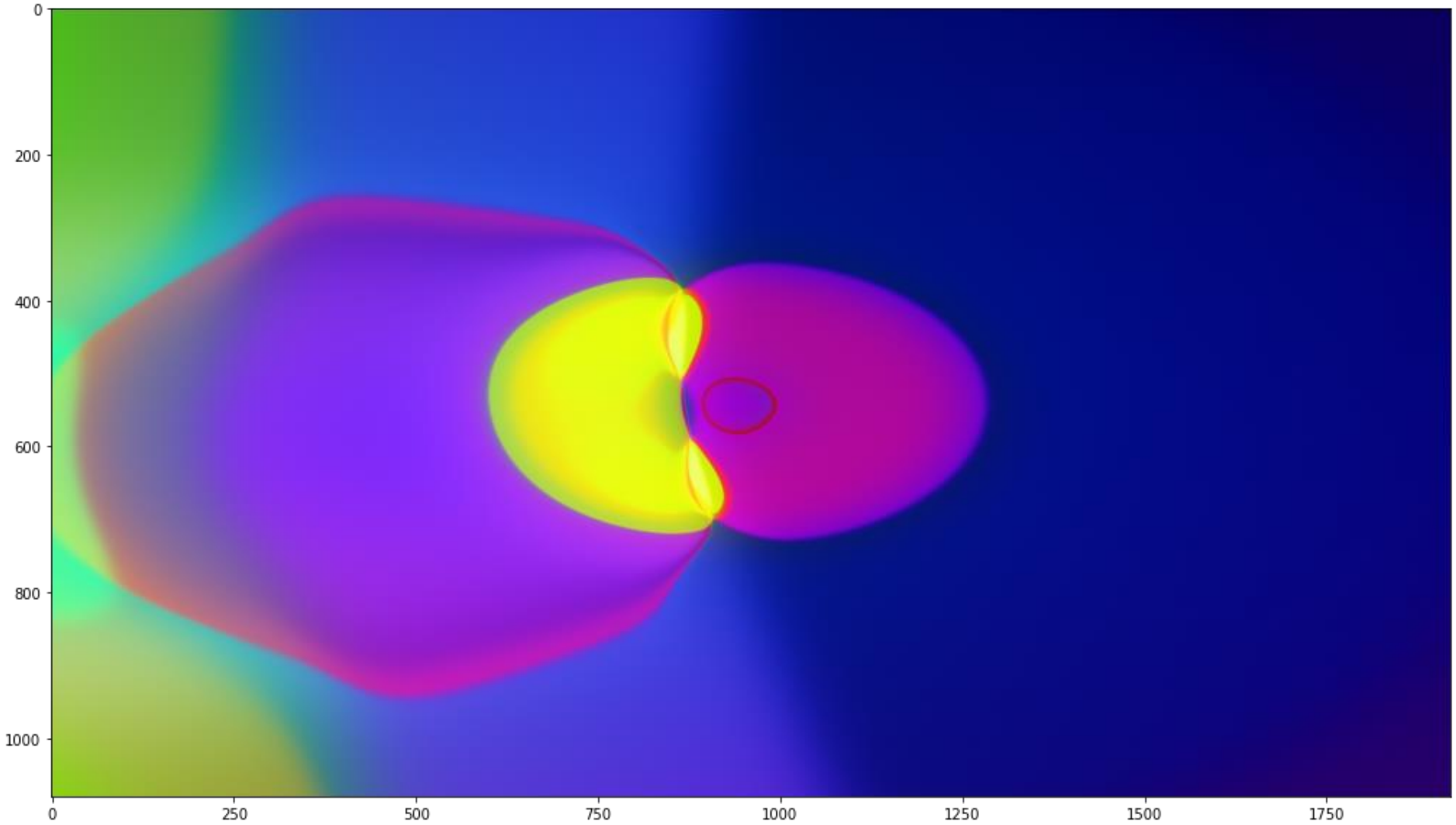
Orbit Trap Fractal Scarf

Deep Learning based Image Generator

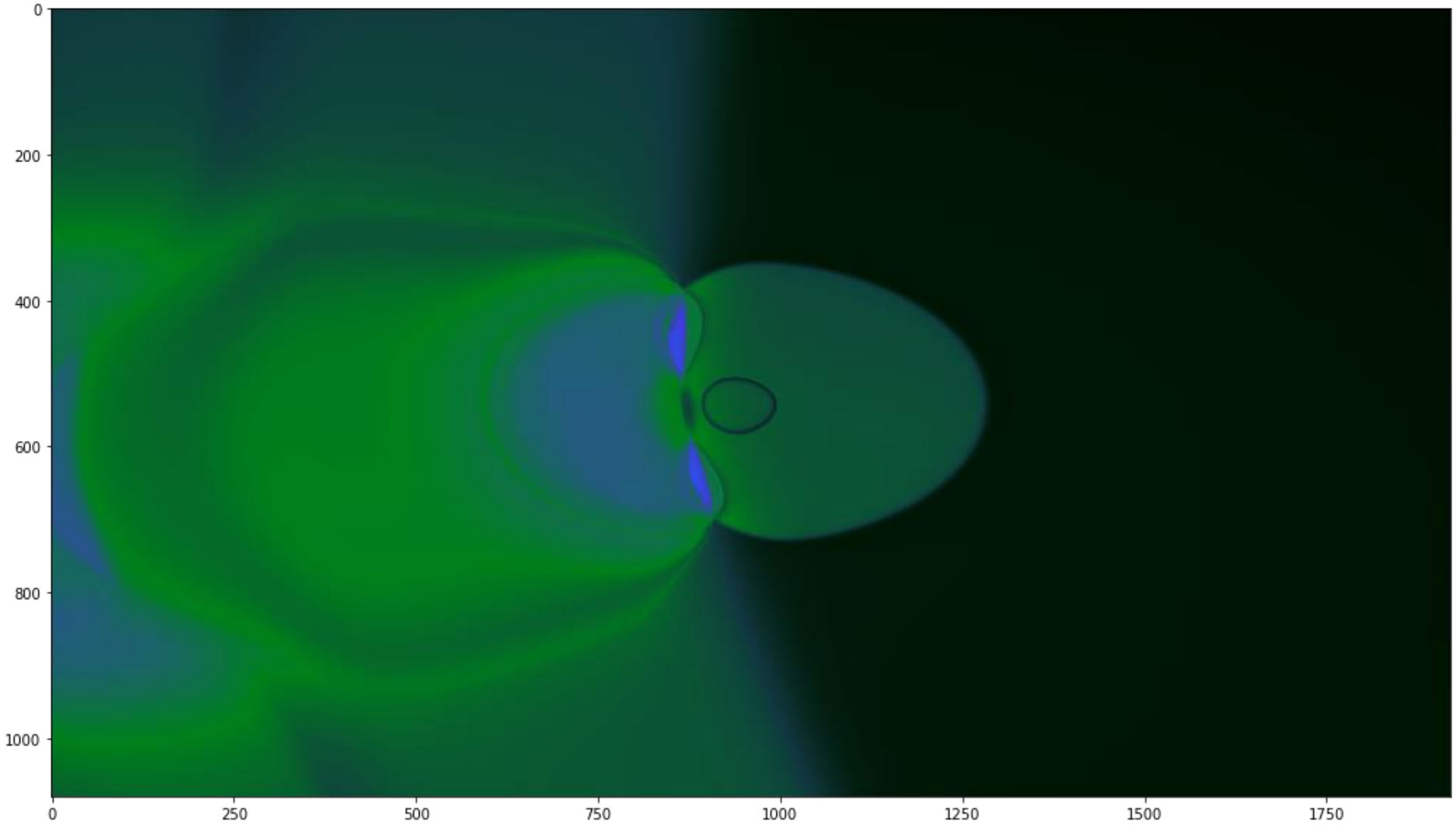
- A Very Simple Neural Network

```
[ ]
def build_densenet(width=4, depth=4, variance=400, activation='tanh', seed=42):
    assert width > 0 and depth > 0 and variance > 0
    tf.random.set_seed(seed)
    input_shape = (5,)  # number of parameters in input space defined above
    initializer = keras.initializers.VarianceScaling(scale=variance,
                                                     mode='fan_in',
                                                     distribution='normal',
                                                     seed=seed)
    inputs = keras.Input(shape=input_shape)
    x = inputs
    for _ in range(depth):
        y = keras.layers.Dense(width, kernel_initializer=initializer, activation=activation)(x)
        x = keras.layers.Concatenate()([x, y])
    bottleneck_initializer = keras.initializers.GlorotNormal(seed)
    bottleneck = keras.layers.Dense(3,  # The number of channels in RGB image
                                    activation='tanh',
                                    kernel_initializer=bottleneck_initializer)(x)
    model = keras.Model(inputs=inputs, outputs=bottleneck)
    return model
```

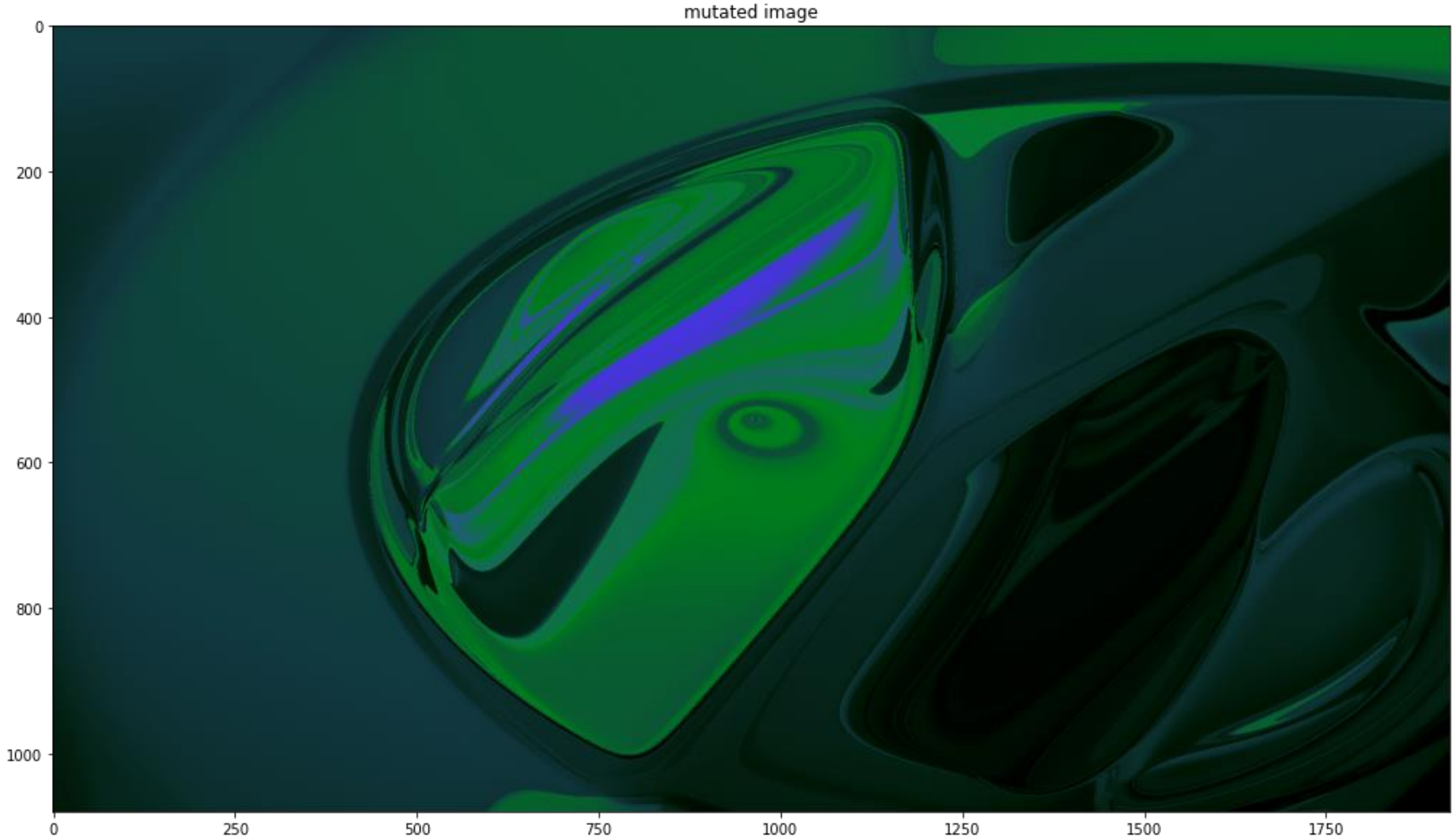
Deep Learning based Image Generator



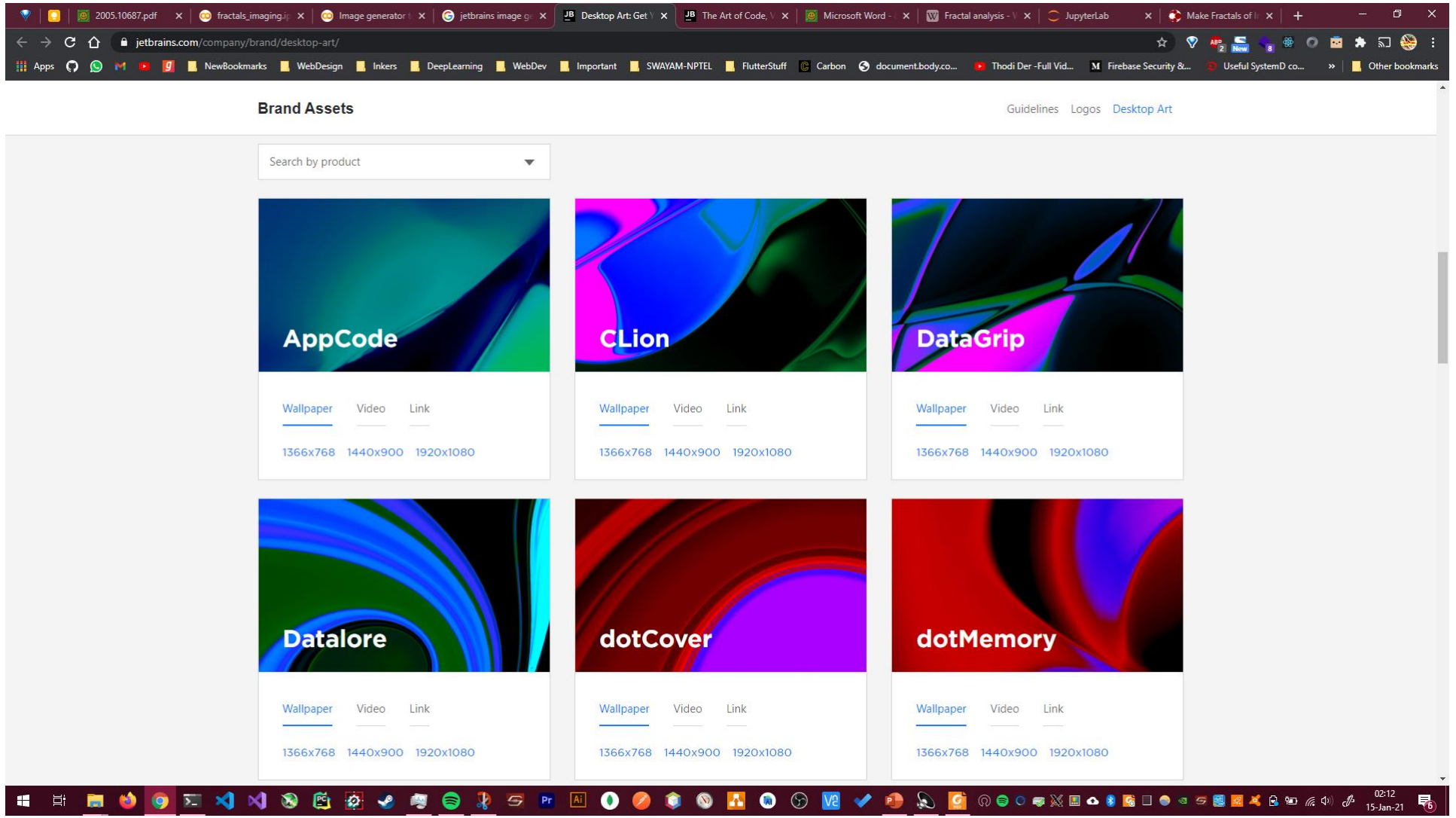
Deep Learning based Image Generator



Deep Learning based Image Generator



JetBrains uses it to generate Unique Product Images



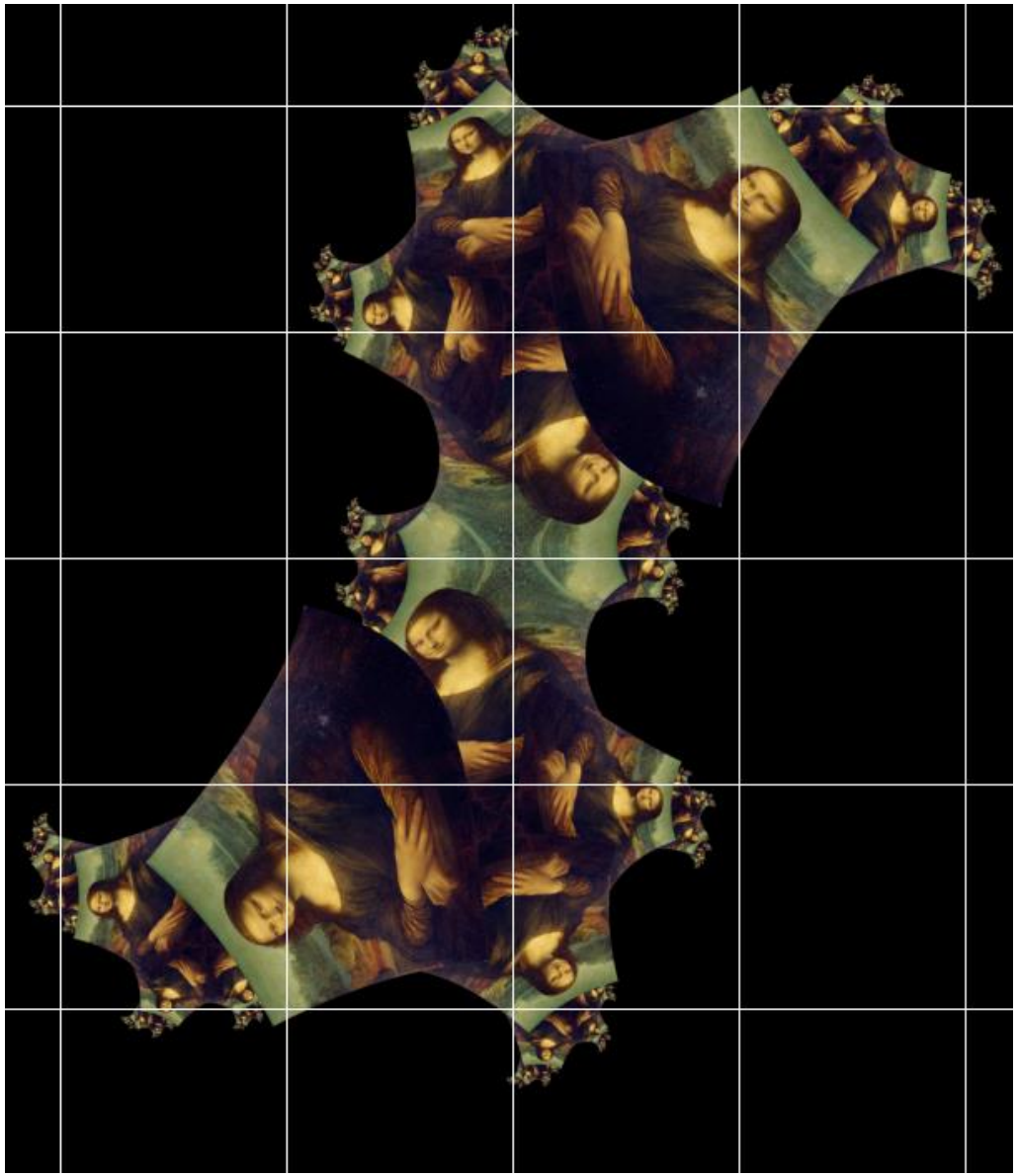
Conclusions

- Fractals are amazing !
- There are infinitely many kinds of patterns you can generate with Fractals, and even more if you involve Deep Learning for generating Patterns.
- In this world every brands wants to be unique, Unique Logo, Unique Theme, Unique Color Scheme, Unique design.
- Deep Learning can help in generating these and making it easy for designers
- Everyone is after beating the SOTA models, but people should also publish their failed experiments also, saves a lot of time for us beginners.

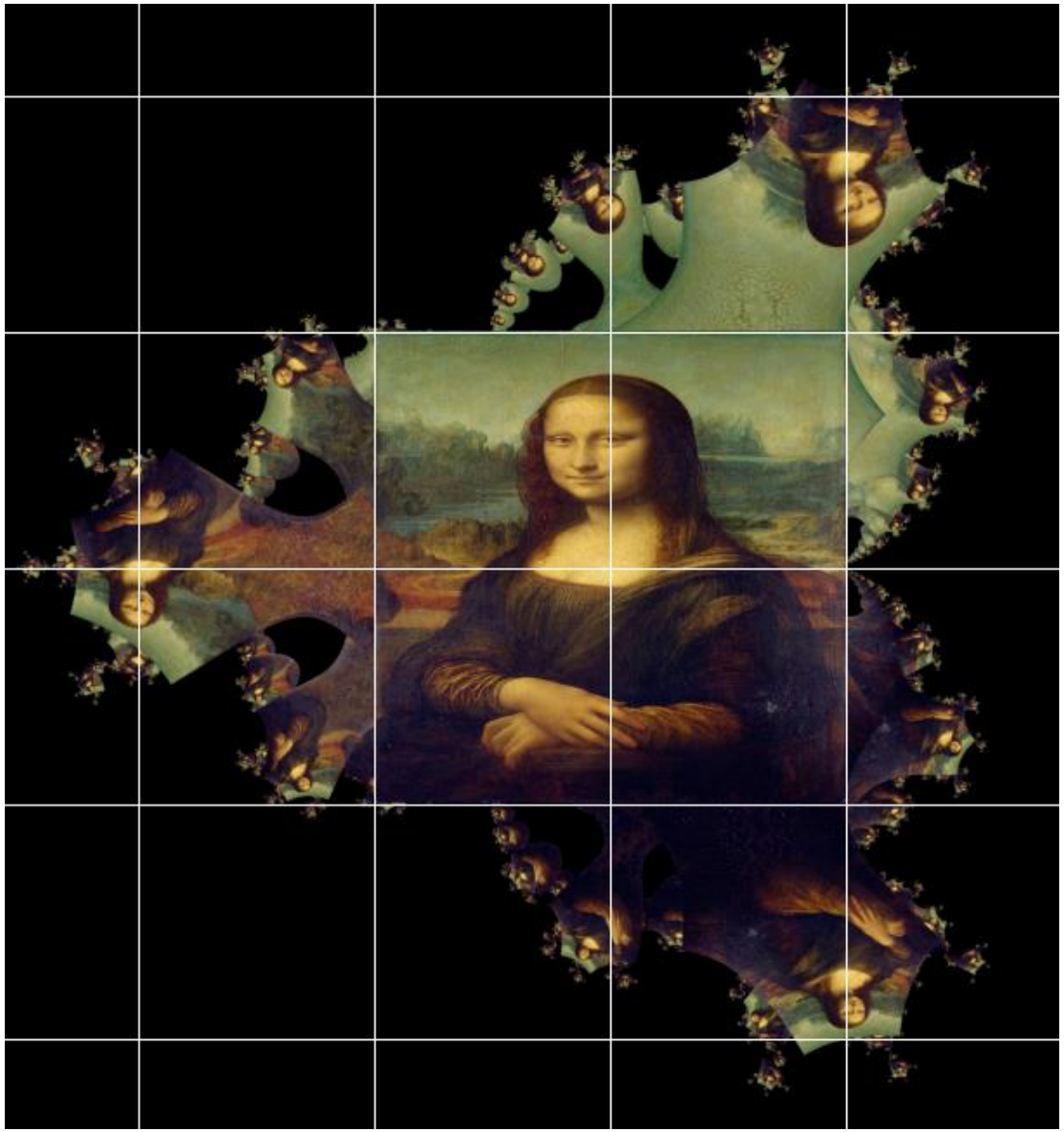
References

- Olah, et al., "Feature Visualization", Distill, 2017.
- http://www.malinc.se/m/ImageFractals.php
- Seuront, Laurent (2009-10-12). Fractals and Multifractals in Ecology and Aquatic Science. CRC Press. doi:10.1201/9781420004243. ISBN 9780849327827.
- Fractal Analysis of Digital Images http://rsbweb.nih.gov/ij/plugins/fraclac/FLHelp/Fractals.htm
- Benoît B. Mandelbrot (1983). The fractal geometry of nature. Macmillan. ISBN 978-0-7167-1186-5. Retrieved 1 February 2012.

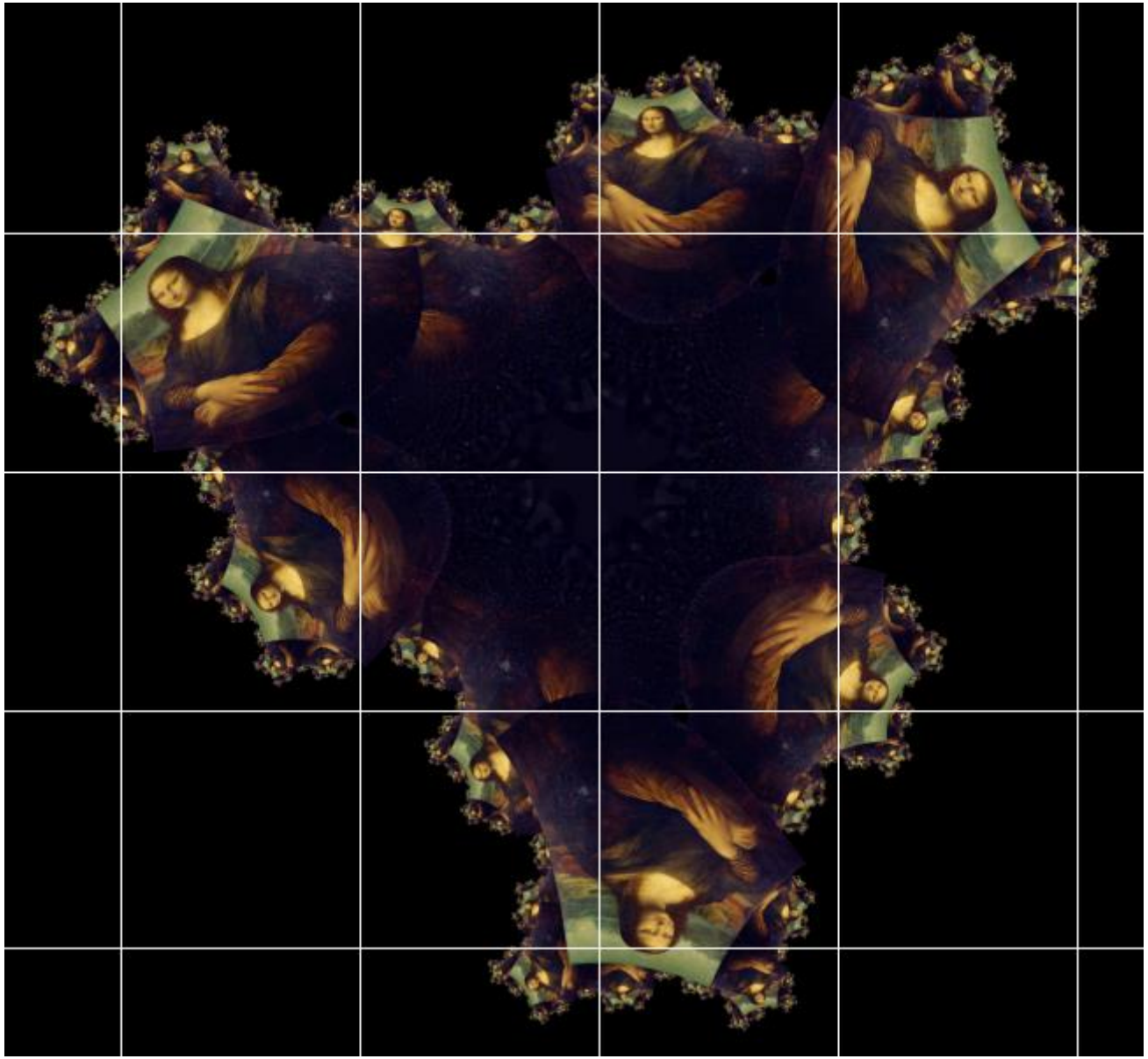
Appendix



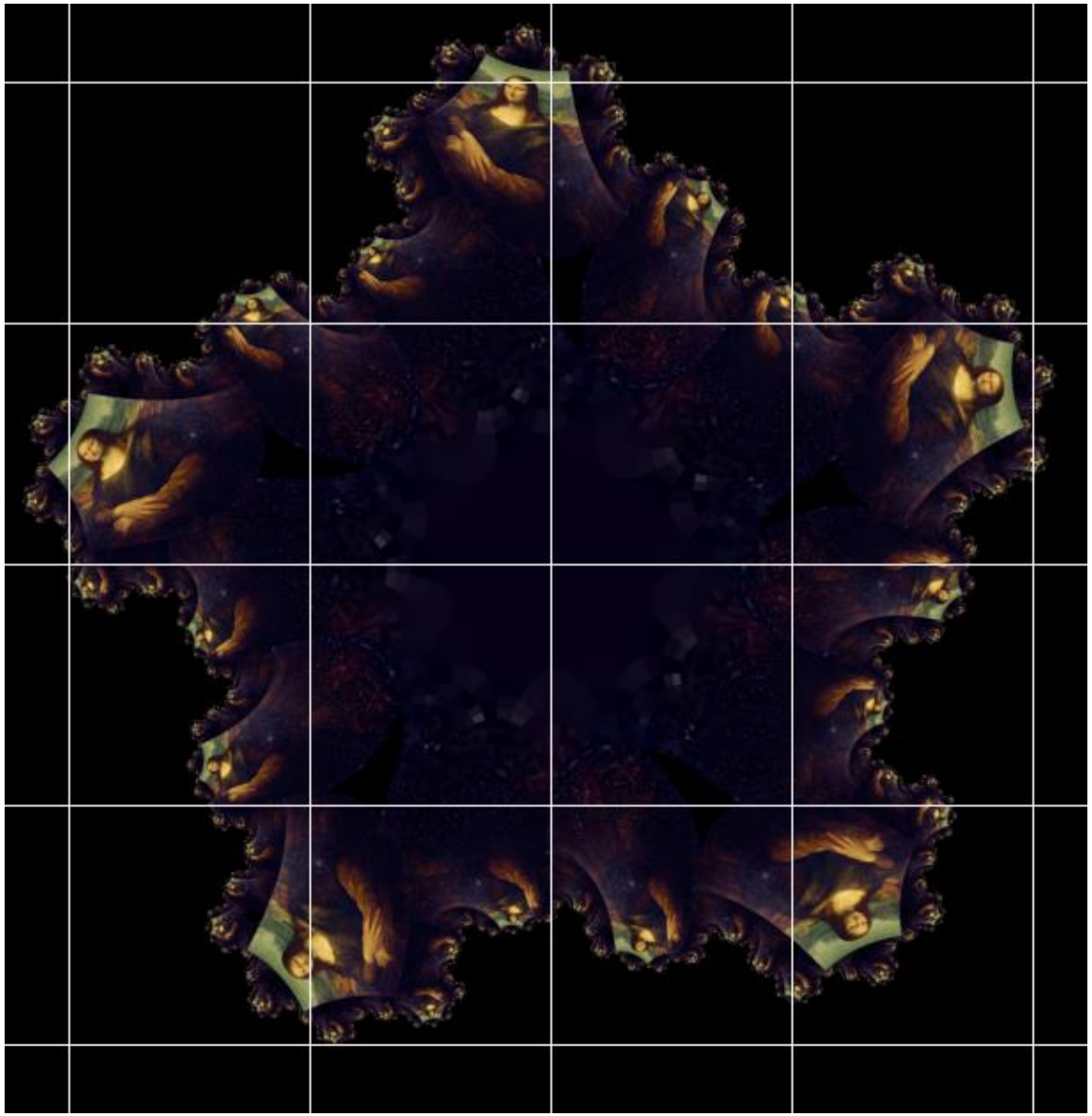
Appendix



Appendix



Appendix



Appendix