Designs and Patterns in Mathematics and Deep Learning, Fractals for Image Classification, and more!

Seminar Member (s)

Sl. No.	Reg. No.	Student Name
01	17ETCS002159	Satyajit Ghana

Supervisor: Mrs. Chaitra S.

Jan – 2021

B. Tech. in Computer Science and Engineering

FACULTY OF ENGINEERING AND TECHNOLOGY

M. S. RAMAIAH UNIVERSITY OF APPLIED SCIENCES

BENGALURU -560 054

FACULTY OF ENGINEERING AND TECHNOLOGY

Certificate

This is to certify that the Seminar titled "Designs and Patterns in Mathematics and Deep Learning, Fractals for Image Classification, and more!" is a bonafide work carried out in the Department of Computer Science and Engineering by Mr. Satyajit Ghana bearing Reg. No. 17ETCS002159 in partial fulfilment of requirements of the Course curriculum of 7th Sem Computer Science and Engineering of Ramaiah University of Applied Sciences.

Jan – 2021

Mrs. Chaitra S
Designation: Professor - Dept of CSE
Place: Bangalore
Date: 31-Jan-2020

Acknowledgements

I would like to thank my mentor Mrs. Chaitra S, who proposed the seminar topic and was passionate towards the project and the driving force and an important factor in the completion of this seminar. She guided me throughout the presentations, seminar and reports along with suggestions during the presentation.

I would also like to thank Dr. S. V. Aruna Kumar for being the co-opted faculty in assessing the seminar, and his valuable inputs on the results obtained in this experiment.

I would like to acknowledge our Dean of Faculty of Engineering and Technology Dr. H.M. Rajashekara Swamy and the head of CSE department Prof. P.V.R Murthy who gave us this opportunity to give a seminar.

I would also like to appreciate the guidance given by the faculty members in the seminar panel which improved my presentation skills by their suggestions and comments.

Summary

Maths and Deep Learning is all about numbers, and findings patterns in them, this seminar is based off of the patterns we can generate using simple mathematical functions, and how we can use them in improving image classification task, and followed by generating patterns using Deep Learning models.

The most difficult part in Image Classification 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, we had in mind of leveraging it to augment image data and making a better classifier from that augmentation. This is followed by how people have leveraged image fractals as their livelihood, by printing it on cloths, each of these fabrics now have a unique pattern, different from other prints, since fractals have this infinite capacity to create infinitely many unique designs.

The second important part of this Seminar was the Pattern Generation using DL Models. Often when creating a product, we require some unique patterns, colour schemes to go along with it, and the demand of "unique" designs is always high, we try to generate these unique patterns for products.

Table of Contents

List of Figures

Abbreviation and Acronyms

DIP	Digital Image Processing
DL	Deep Learning
CNN	Convolutional Neural Network
OTF	Orbit Trap Fractal

1. Introduction

This Chapter gives a brief Introduction of the topics that will be essential to know to understand this report, the explanation and elaboration will continue in Background Theory of this Report. Followed by the organization of this report.

This Project started with an attempt to create Patterns and Designs using standard Mathematical techniques and followed by Deep Learning, this led into an attempt to use these patterns for improvisation in Image Classification.

1.1. Deep Learning for Computer Vision

Deep learning is a subset of Machine Learning that has pushed the limits to what was possible in Digital Image Processing (DIP), its used in the domain of image colorization, classification, segmentation, and detection. Methods that involve Convolutional Neural Network (CNN) have take in a lot of data and computational resources to push the boundaries.

DL is all about learning or 'credit assessment' across many layers of the neural network, self-organisation and the exploitation of interactions between small units have proven to perform better than central control, particularly for complex non-linear process models in that better fault tolerance and adaptability to new data is achievable. (Koehn P. et al, 1994).

The development of CNNs has had a tremendous influence in the field of CV in recent years and is responsible for a big jump in the ability to recognize objects (Voulodimos et al, 2018). This burst in progress has been enabled by an increase in computing power, as well as an increase in the amount of data available for

training neural networks. The recent explosion in and wide-spread adoption of various deep-neural network architectures for CV is apparent in the fact that the seminal paper ImageNet Classification with Deep Convolutional Neural Networks has been cited over 3000 times. (Nash et. al, 2018)

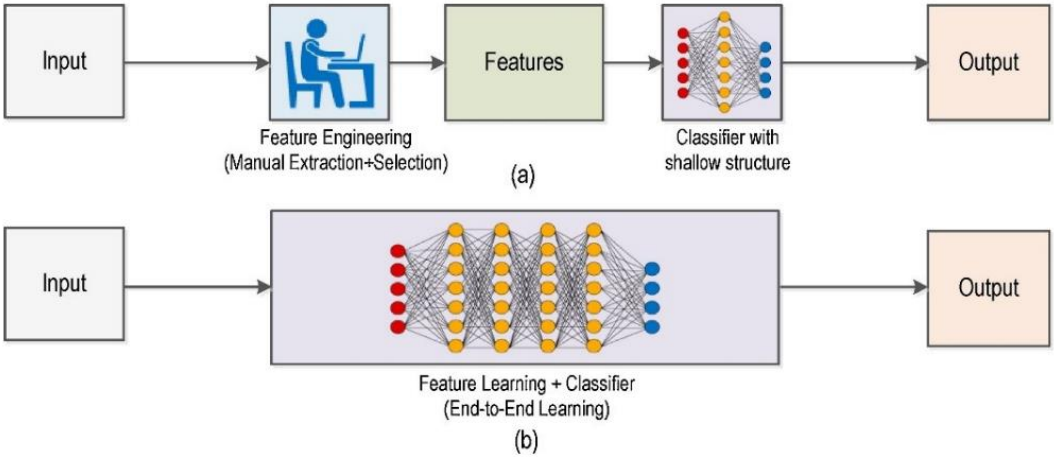


Figure 1 (a) Traditional Computer Vision workflow vs (b) Deep Learning workflow (Wang J et al, 2018)

1.2. Regularization in Deep Learning

Regularization is one of the key elements of machine learning, particularly of deep learning (Goodfellow et al., 2016), allowing to generalize well to unseen data even when training on a finite training set or with an imperfect optimization procedure. In the traditional sense of optimization and also in older neural networks literature, the term "regularization" is reserved solely for a penalty term in the loss function (Bishop, 1995a). Recently, the term has adopted a broader meaning: Goodfellow et al. (2016, Chap. 5) loosely define it as "any modification we make to a learning algorithm that is intended to reduce its test error but not its training error".

The quality of a trained model depends largely on the training data. Apart from acquisition/selection of appropriate training data, it is possible to employ

regularization via data. This is done by applying some transformation to the training set $\mathcal{D}$, resulting in a new set $\mathcal{DR}$. Some transformations perform feature extraction or pre-processing, modifying the feature space or the distribution of the data to some representation simplifying the learning task. Other methods allow generating new samples to create a larger, possibly infinite, augmented dataset. These two principles are somewhat independent and may be combined. (J Kukačka et al, 2018)

1.3. Fractals as an Augmentation Regularizer

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.

Figure 2 Example of Fractals in Nature

The aim of this experiment is to use Fractals as augmentation technique for images, and speculation is that it would increase the classification accuracy of such images. This speculation is based on the fact, that instead of performing a lot of different augmentations such as Scaling, Shearing and Cropping, all of these augmentations can be done in a single shot by using Image Fractals.

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.

Orbit Trap Fractals can thus be used in a Raster Image to create Image Fractals. Which we are calling the Fractal Image Augmentation as of this report.

1.4. Pattern Generation using Deep Learning

From the idea of Fractals, and creating beautiful patterns using simple Mathematical function, we further explore the idea of generating patterns using Deep Learning models, this is something that is very unique, and aims to generate unique product background images, something that JetBrains has been doing for quite sometime in all of their products.

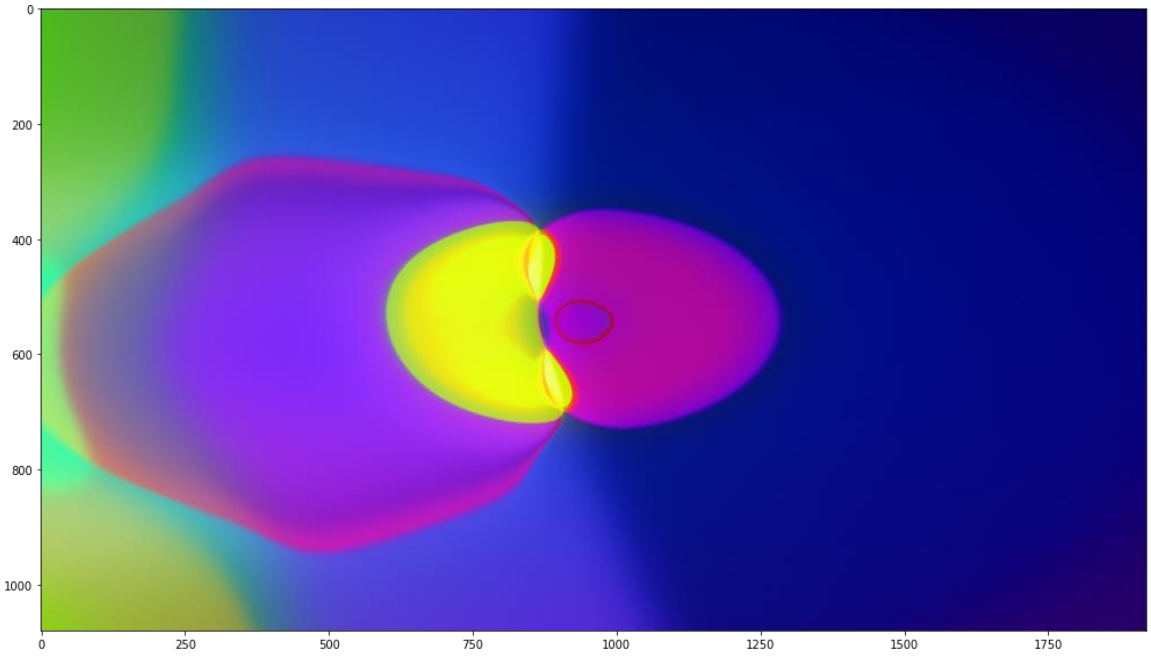


Figure 3 Deep Learning based Generated Image/Pattern

1.5. Organization of the report

This report is further organized into,

a. Background Theory: Discussion of various Regularization Techniques in Deep Learning for Computer Vision, and various other related work in concern of this report.

b. Aim and Objectives: Description and Summary of the various experiments done in this Report, and Methodologies used to achieve that Aim.

c. Discussion and Results: Here we describe and elaborate of the results obtained of the aims discussed in the above chapter, and try to explain the behaviour that was exhibited by the results.

d. Conclusion and Suggestion for Future Work: At the end we conclude the report by stating the conclusions and possible future work to improve the results that was obtained in this report.

2. Background Theory

This chapter briefly describes the different key concepts important to understand the underlying principle of the idea being discussed in this report.

2.1. Survey of Related Work

Convolutional Neural Networks (CNN) are capable of learning really complex patterns and objects, however due the enormous model capacities, and their learning abilities, often models tend to overfit on the dataset, and therefore require some rigorous regularization methods to overcome this issue.

In this paper, (DeVries et al, 2017) show that the simple regularization technique of randomly masking out square regions of input during training, which we call cutout, can be used to improve the robustness and overall performance of convolutional neural networks.

Not only is this method extremely easy to implement, but they also demonstrate that it can be used in conjunction with existing forms of data augmentation and other regularizers to further improve model performance. (DeVries et al, 2017)

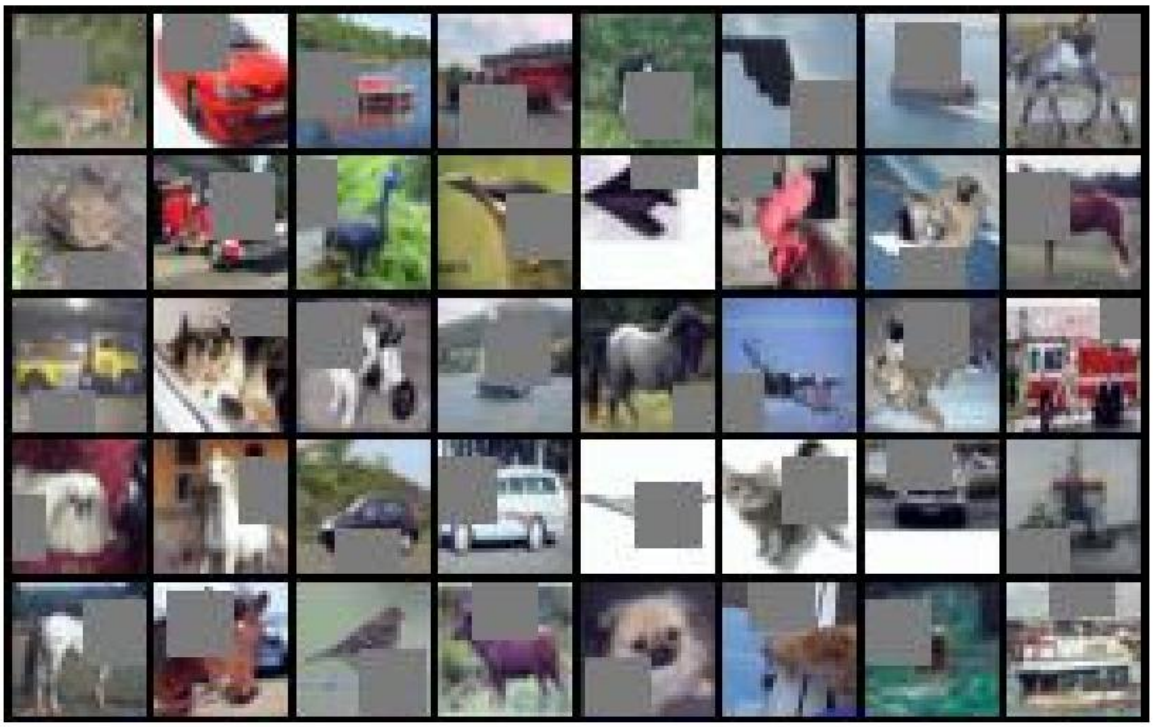


Figure 4 Cutout applied to images from CIFAR10

Albumentations: fast and flexible image augmentations (Buslaev et al, 2018) is another really good paper on the image transformations, albumentations is a fast and flexible library for image augmentations with many various image transform operations available, that is also an easy-to-use wrapper around other augmentation libraries.

Figure 5 Some Examples of Image Transformations from the Albumentations Library

2.2. Assumptions and Principles

To understand the assumptions and principles we have to first look into how actually a neural network learns to identify different features of a given image and thus given a prediction on what that image might be, Olah et al, 2017, published a really good article in distill.pub that explains exactly this.

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.

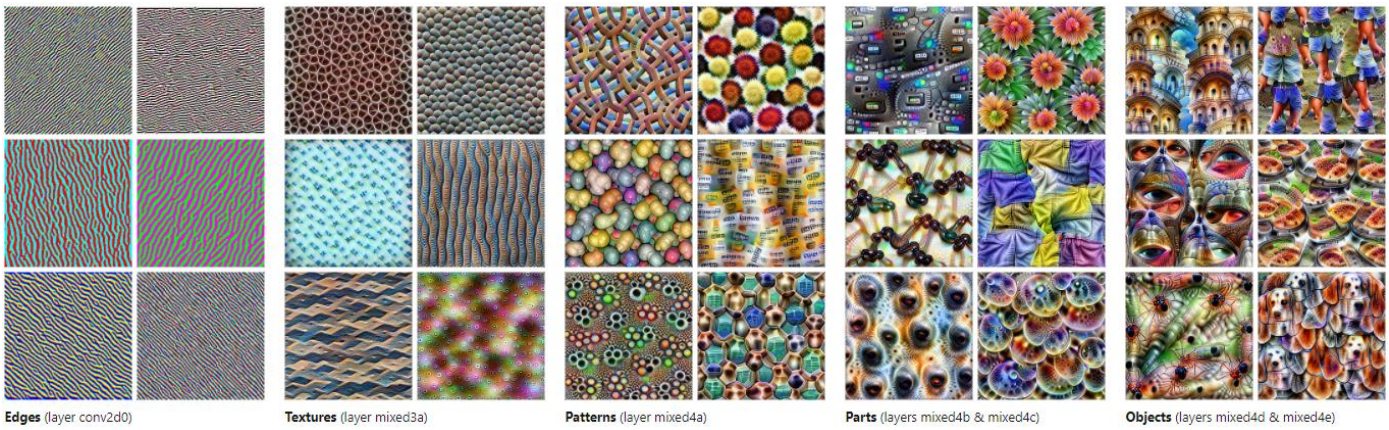


Figure 6 Feature visualization allows us to see how GoogLeNet, trained on the ImageNet dataset, builds up its understanding of images over many layers.

Here we can see how different layers of LeNet learns to classify images. It forms Edges, Textures, Patterns, Parts of Objects and finally the Objects.

The assumptions to creating fractals out of images, is give more and more data to the network so that it can more efficiently create this Parts of Objects and Object,

the speculation is that fractals will make even better Object Channels, and the Model will be able to generalize much better than before.

2.3. Originality of Work

The only work close to this (in terms of google search results based on name) is Fractal Analysis, which is nowhere close to what our main idea is, Fractal Analysis 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.

3. Aim and Objectives

This Chapter describes the different aims of the experiments conducted towards our goal of Fractals for Image Classification, and also generating Patterns/Designs using Deep Learning Models and Frameworks.

Title

- Designs and Patterns in Mathematics and Deep Learning, Fractals for Image Classification

Aim

- The main aim is to see how well Fractal Image Augmentation works for the task of Image Classification of Computer Vision
- The secondary aim is to create a deep learning model that can generate unique product background images

Objectives

- To create fractals (Julia Fractal, Mandelbrot fractal)
- To create orbit trap image fractals
- To use image fractals as an Image Augmentation Technique
- Evaluating Fractal Image Augmentation on CIFAR10 dataset and comparison with a basic DL model
- Creating Patterns and Designs using Deep Learning Model

Objective No.	Statement of the Objective	Method/ Methodology	Resources Utilised
1	Create Fractals	Use Mandelbrot and Julia Set parameters to create Fractals	Numpy and PIL Python Libraries
2	Create Orbit Trap Fractal	Use Raster Images, and Basic Shapes to demonstrate OTF	Numpy, and PIL Python Libraries
3	Fractal as an Image Augmentation Technique	Create Fractal Augmentation using Albumentation Method	Albumentations Python Library
4	Evaluate models on CIFAR10 and CIFAR10 Fractal Augmentation	Compare models on CIFAR10 Dataset with and without the Fractal Augmentation	PyTorch Lightning Python Library
5	Creating Patterns using DL Model	Create different types of pattern using a DL Model, and vary the different parameters to obtain different patterns	Tensorflow and Keras Python Libraries

4. Discussion and Results

This chapter shows the process of fulfilling the aims and objectives discussed earlier

4.1. Results & Discussion

This report is structured according to the aims and objectives defined earlier, we will take those objectives one-by-one and discuss on how they were achieved along with the results obtained on each of them.

4.1.1. Create Fractals

We can start by creating Mandelbrot Fractals, which is a very simple Equation,

$$f_c(z) = z^2 + c$$

where,

x_0 is initial value of z

c is some constant

```
x_min, x_max = -2, 1
y_min, y_max = -1, 1

delta = 0.001

re, im = np.mgrid[x_min:x_max:delta, y_min:y_max:delta]

c = (re + 1j*im).T
escape = np.zeros_like(np.abs(c))
x0 = np.zeros_like(c)
x = x0

for i in range(100):
    x = f(x, c)
    idx = (np.abs(x) > 10) & (escape == 0)
    escape[idx] = 100 - i

plt.figure(figsize=(15, 15))
plt.imshow(escape, extent=(x_min, x_max, y_min, y_max))
```

The above code simply applies our Mandelbrot function, over and over again, 100 times at max, and assigns a colour value to the pixel based on how quickly it has exploded. This produces the following result,

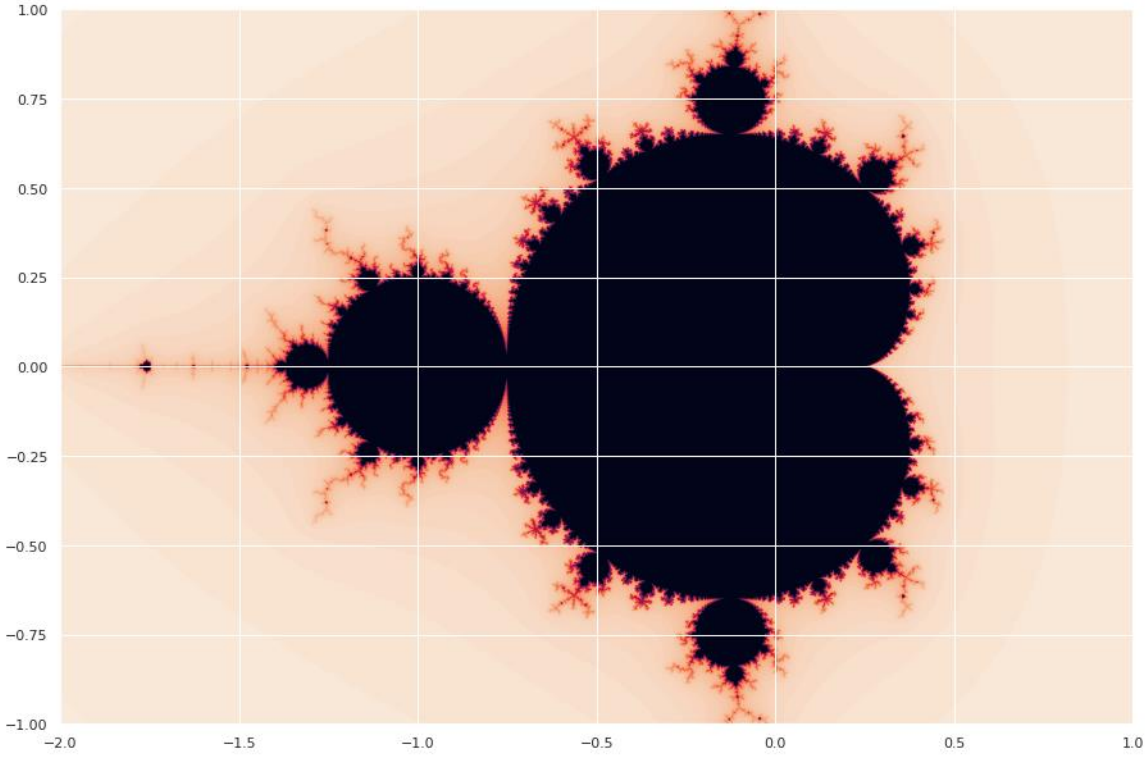


Figure 7 Mandelbrot Fractal

If we apply the same logic, but for different initial value and constant value (there are some predefined constants, that produce nice looking fractals) we can end up with Julia Fractals,

$$c = 0.3 + 0.6j$$

```
x_min, x_max = -1.5, 1.5
y_min, y_max = -1.5, 1.5

delta = 0.001

re, im = np.mgrid[x_min:x_max:delta, y_min:y_max:delta]

x = (re + 1j*im).T
escape = np.zeros_like(np.abs(x))
c = np.zeros_like(x)
c.fill(0.3 + 0.6j)

for i in range(100):
    x = f(x, c)
    idx = (np.abs(x) > 10) & (escape == 0)
    escape[idx] = i

plt.figure(figsize=(15, 15))
plt.imshow(escape, extent=(x_min, x_max, y_min, y_max), interpolation = 'none', norm = LogNorm())
```

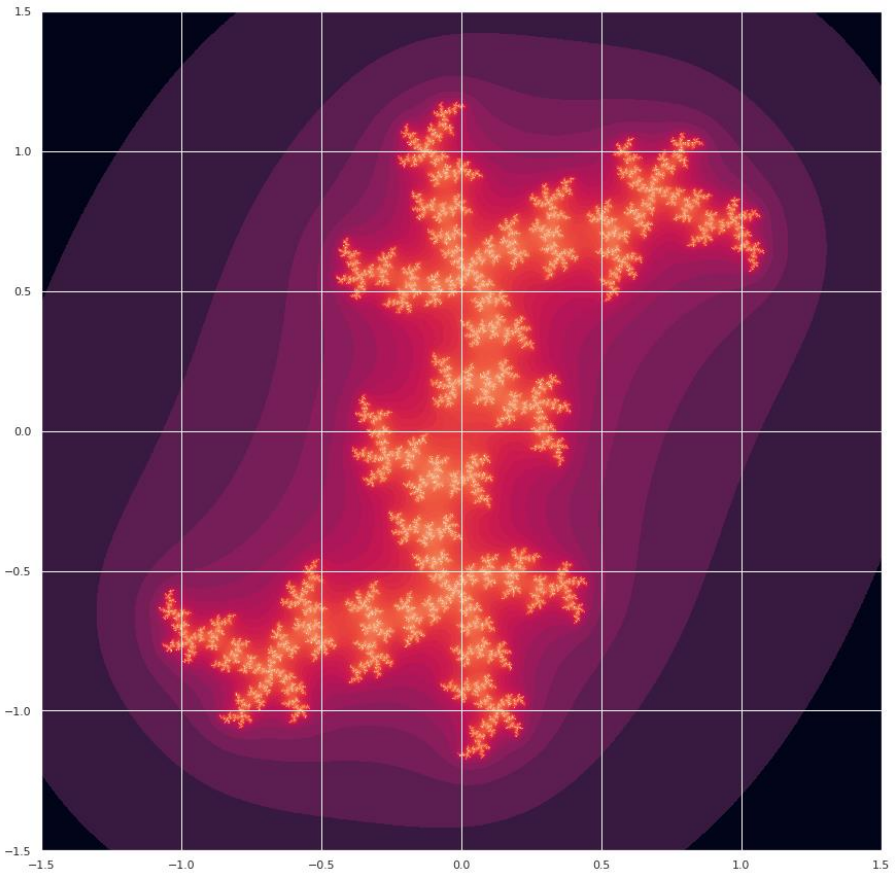


Figure 8 Julia Set Fractal

4.1.2. Orbit Trap Fractal

Orbit Trap Fractal is basically trapping each of the point that tries to escape a specific region, this region can be any geometric figure, or even an Image, below we try to use a circle to trap the exploding points from the fractal loop,

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

def create_orbit_fractal(c, deg = 2, dist_func = circle_dist):
    x0 = np.zeros_like(c) # x0 = 0
    x = x0

    distances = np.ones_like(np.abs(c)) * 10

    for iter in range(100):
        x = f(x, c, deg=deg)

        z_minus = x.copy()
        z_minus = dist_func(z_minus)
        idx = np.abs(z_minus) < 10
        distances[idx] = np.minimum(distances[idx], np.abs(z_minus[idx]))

    return distances

x_min, x_max = -2.5, 2
y_min, y_max = -1.5, 1.5

delta = 0.001

re, im = np.mgrid[x_min:x_max:delta, y_min:y_max:delta]

c = (re + 1j*im).T

distances = create_orbit_fractal(c)

plt.figure(figsize=(15, 15))
plt.imshow(distances, extent=(x_min, x_max, y_min, y_max))
```

This results in the following figure,

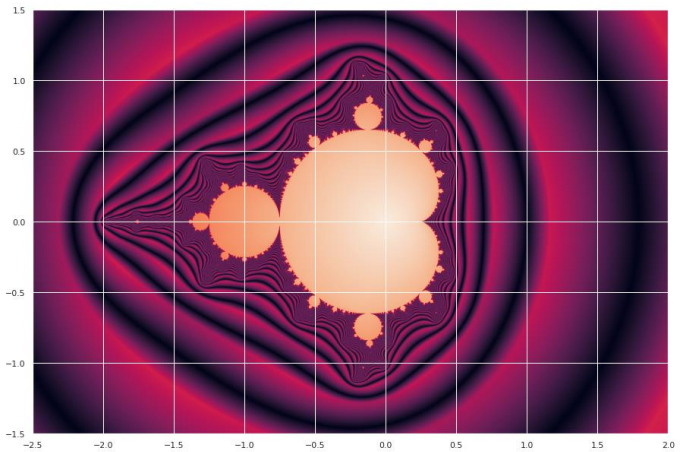


Figure 9 Orbit Trap Fractal - Circle

Now we try the same logic, but on a raster image,

```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

x_min, x_max = -2.5, 2
y_min, y_max = -1.5, 1.5

delta = 0.001

re, im = np.mgrid[x_min:x_max:delta, y_min:y_max:delta]

c = (re + 1j*im).T

fractal = create_image_fractal3(c, img)

plt.figure(figsize=(15, 15))
plt.imshow(fractal.astype(np.uint8), extent=(x_min, x_max, y_min, y_max))
```

Which results in the following image,

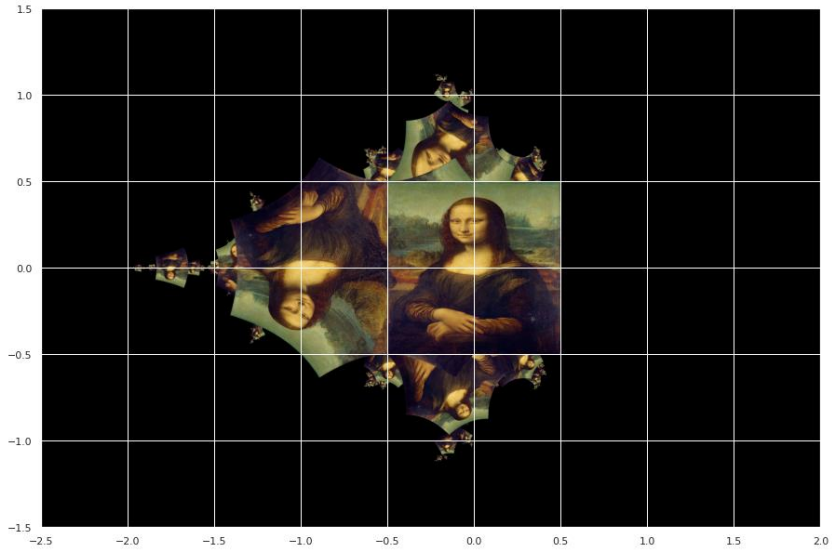


Figure 10 Orbit Trap Fractal on Image (Mandelbrot degree 2)

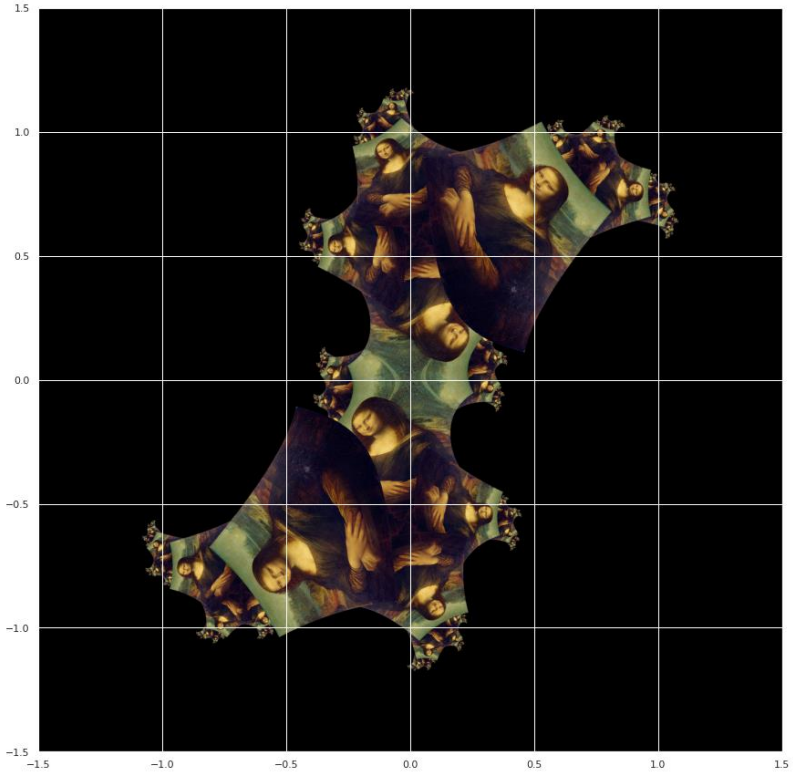


Figure 11 Orbit Trap Fractal on Image (Julia degree 2)

4.1.3. Image Fractal Augmentation

First, we have to convert the images in CIFAR10 into Mandelbrot Fractal Images, this is being done prior to training the model, since the fractal generation processing is time and compute consuming, and doing it on the fly while training the model makes the process really slow and non-feasible.

```
transform = A.Compose([
    MandelBrotFractalTransform(deg=2, delta=0.04),
    A.Resize(96, 96)
])


train_dataset = CIFAR10('~/data', train=True, download=True)
test_dataset = CIFAR10('~/data', train=False, download=True)


for idx, (image, class_idx) in tqdm(enumerate(train_dataset), total=len(train_dataset)):
    save_path: Path = dset_path / 'train' / f'{class_idx}'
    save_path.mkdir(exist_ok=True, parents=True)

    transformed = transform(image=np.array(image))['image']
    trans_image = Image.fromarray(transformed)

    trans_image.save(str(save_path / f"img_{idx}.jpg"))

for idx, (image, class_idx) in tqdm(enumerate(test_dataset), total=len(test_dataset)):
    save_path: Path = dset_path / 'test' / f'{class_idx}'
    save_path.mkdir(exist_ok=True, parents=True)

    transformed = transform(image=np.array(image))['image']
    trans_image = Image.fromarray(transformed)

    trans_image.save(str(save_path / f"img_{idx}.jpg"))

// Photo by Samuel Ferrara on Unsplash
```

Once this is done, we will get images like the following,

Original Image

Mandelbrot Fractal Image

4.1.4. Evaluating a DL model

The Model is a really simply, fully connected layers,

```
class LitModel(pl.LightningModule):
    def __init__(
        self, channels, width, height, num_classes, hidden_size=64, learning_rate=2e-4
    ):
        ...

        self.model = nn.Sequential(
            nn.Flatten(),
            nn.Linear(channels * width * height, hidden_size),
            nn.ReLU(),
            nn.Dropout(0.1),
            nn.Linear(hidden_size, hidden_size),
            nn.ReLU(),
            nn.Dropout(0.1),
            nn.Linear(hidden_size, num_classes),
        )

    def forward(self, x):
        x = self.model(x)
        return F.log_softmax(x, dim=1)
```

And now we can evaluate it on our fractal CIFAR10 dataset, and normal CIFAR10 dataset, to make things simple and quick, we train both of them for a single epoch, and with equal amount of data split. Both of them take a resized image of 96X96 and forward passthrough the network, get the log-softmax of the output, this is then used in Cross Entropy Loss to evaluate the loss.

```python
# Evaluating the model on Mandelbrot Fractal Image CIFAR10
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)

# Evaluting the model on regular non-augmented CIFAR10
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)

// Photo by Matt Artz on Unsplash
```

Results

	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

4.1.5. Creating Patterns using DL Model

Images that produces our generator are essentially landscapes of (fairly simple) feed-forward neural networks' mapping functions! Although there's a couple of tricks that make them more beautiful, most of them are based on what exact data we are passing to our networks. Actually, there's 5 float inputs corresponding to each pixel in the image:

- `x_pos` - this one is linearly proportional to the coordinate of the current pixel over X axis of the image (exact values depend on other image generation parameters that are out of scope of this demo);
- `y_pos` - the same as the previous one but corresponds to Y axis;
- `alpha` - this one has some constant value across all the pixels in the image and used by the animation engine;
- `beta` - the same as the previous one
- `f` - this one allows to apply some general patterns to images by mapping some functions to `x_pos` and `y_pos`
- $f = F(x_{pos}, y_{pos})$. For Example, $f = \sqrt{x_{pos}^2 + y_{pos}^2}$ produces some circular shape patterns

We'll build a simple DenseNet Model with just 258 parameters

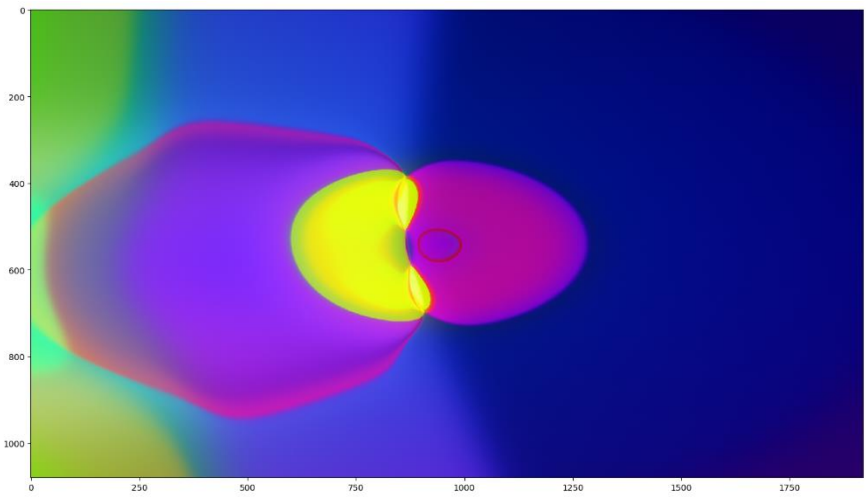
```python
def build_densenet(width=4, depth=4, variance=400, activation='tanh', seed=42):
    assert width > 0 and depth > 0 and variance > 0
    tf.random.set_seed(seed)
    input_shape = (5,)  # number of parameters in input space defined above
    initializer = keras.initializers.VarianceScaling(scale=variance,
                                                     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

# Some images could contain gradient bandings. Let's add a grain effect:
def dither_image(img: np.ndarray, noise_factor: float = 1):
    img += (np.random.random(img.shape) - 0.5) * (noise_factor / 256)
    img[img < 0] = 0
    img[img > 1] = 1
    return img

# Finally, let's write a method for inferencing the network:
def render_image(model, input_space, out_shape):
    pred = model.predict(input_space, batch_size=65536)
    # Normalizing the predictions
    pred_normed = (pred - pred.min(0)) / (pred.ptp(0) + 1e-10)
    img_arr = pred_normed.reshape(*out_shape)
    img = dither_image(img_arr, noise_factor=2)
    img_arr = (img_arr * 255).astype(np.uint8)
    return img_arr

example_image = render_image(example_model, input_space, (y_resolution, x_resolution, 3))
plt.figure(figsize=(19, 10))
plt.imshow(example_image)
plt.show()
```

Which produces,



5. Conclusions and Suggestions for Future Work

This chapter concludes the results obtained in the above chapter, with its interpretation. Continued by suggestions for further experiments that should be carried out to get more results out of this new form of augmentation technique.

5.1. 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 brand 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

From the results obtained, we (not whole heartedly) conclude that Fractals though may seem like a really good augmentation technique, does not work that well, and a major downside of using fractals is that it takes a lot of time and CPU to make the fractals, even though we could have use GPU to speed up the process, it requires the knowledge of CUDA programming.

To make the comparison we had chosen the same model architecture, for both w/ fractal augmentation and w/o augmentation. The non-augmented dataset model performed well above (41.3% accuracy) when compared to the fractal augmented

dataset model (18.13% accuracy), this as per us does not give enough motivation to do further work and experimentation on this kind of augmentation.

Coming to the part of Pattern Generation using Deep Learning, this is the area where DL models really shine, from the results we conclude that DL models can generate "unique" art, and this process is already being used by JetBrains, and various other companies for their products.

A novel uses of Orbit Trap Fractals

Marlies Bugmann from Tasmania is using the orbit trap fractal generator to make scarves. Her wearable art can be seen at tasmanianartist.yolasite.com and she sells her creations at Redbubble's 'd1g1tal m00dz'.

Figure 12 An Orbit Trap Fractal Scarf

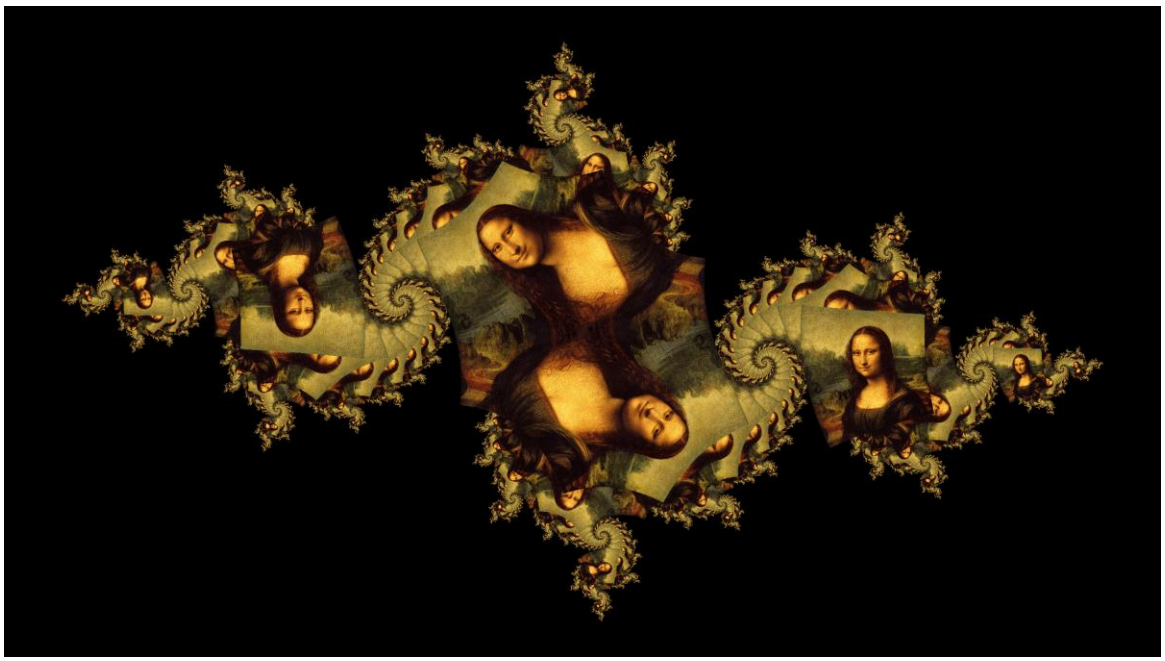
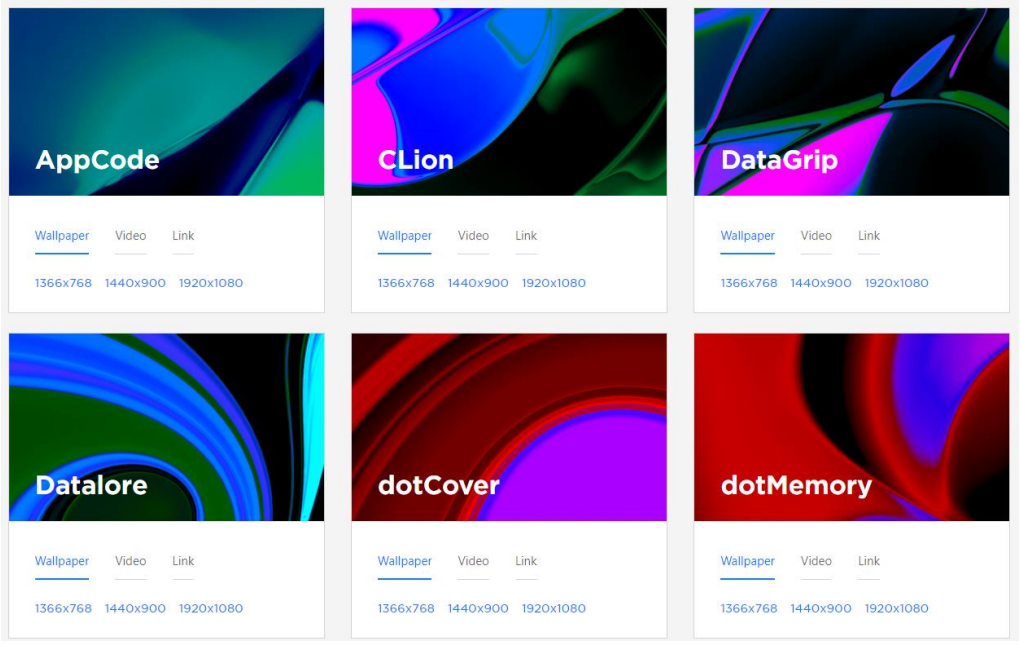


Figure 13 Orbit Trap Fractal Wallpaper

Novel Use for Pattern Generation using Deep Learning Models



JetBrains used Deep Learning to generate their product unique background art.

5.2. Future Work

The fractal augmentation experimentation can be further taken up to verify its results, also there's also discussion needed on whether or not the test dataset should also be augmented the same way as of the train dataset, in case of fractals. Also, various other fractal methods can be used, like trying our different Julia Sets, along with moving the computation to GPU to speed up the process. Furthermore, additional augmentation can be applied over the Fractal Images, to improve regularization effects, like for example cutouts.

DL Models are not just limited to generating 2D art, they can also be used to generate 3D objects,

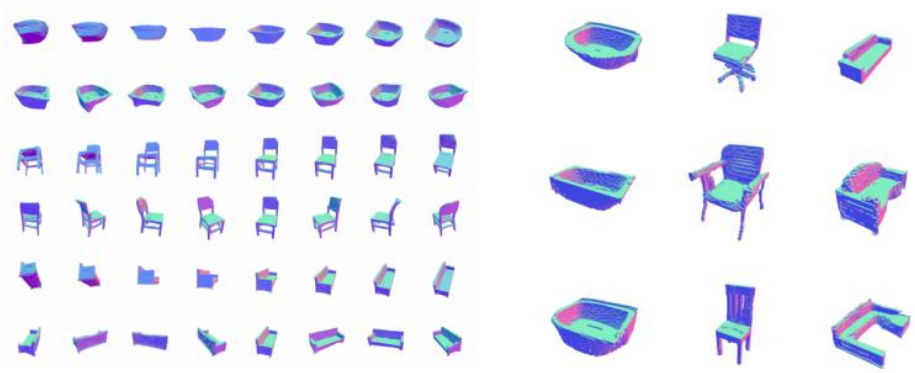


Figure 14 AI Generated 3D Images

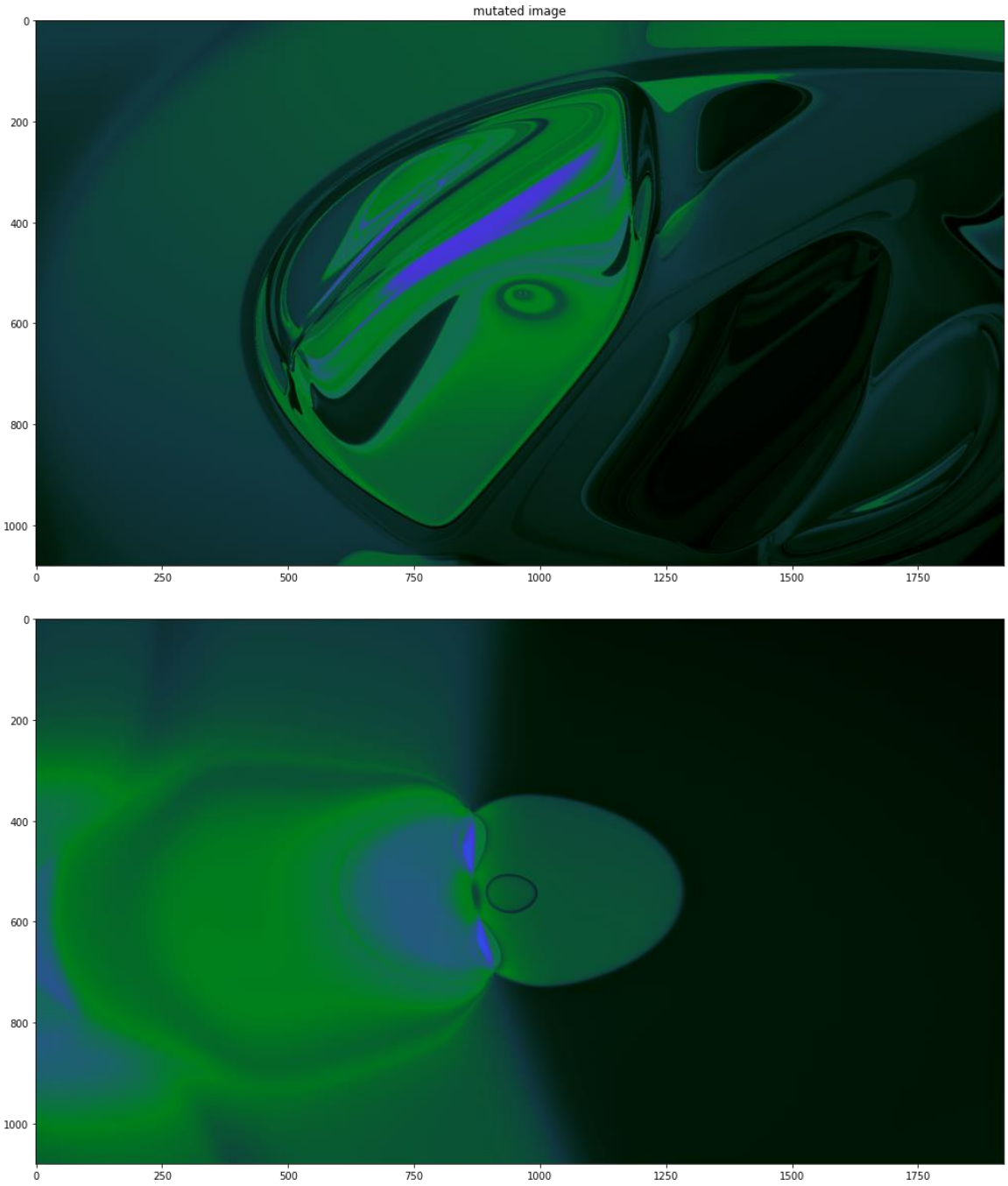
References

1. Koehn, P., 1994. Combining genetic algorithms and neural networks: The encoding problem.
2. Nash, W., Drummond, T. and Birbilis, N., 2018. A review of deep learning in the study of materials degradation. npj Materials Degradation, 2(1), pp.1-12.
3. Voulodimos, A., Doulamis, N., Doulamis, A. and Protopapadakis, E., 2018. Deep learning for computer vision: A brief review. Computational intelligence and neuroscience, 2018.
4. Wang, J., Ma, Y., Zhang, L., Gao, R.X. and Wu, D., 2018. Deep learning for smart manufacturing: Methods and applications. Journal of Manufacturing Systems, 48, pp.144-156.
5. Goodfellow, I., Bengio, Y., Courville, A. and Bengio, Y., 2016. Deep learning (Vol. 1, No. 2). Cambridge: MIT press.
6. Kukačka, J., Golkov, V. and Cremers, D., 2017. Regularization for deep learning: A taxonomy. arXiv preprint arXiv:1710.10686.
7. DeVries, T. and Taylor, G.W., 2017. Improved regularization of convolutional neural networks with cutout. arXiv preprint arXiv:1708.04552.
8. Buslaev, A., Iglovikov, V.I., Khvedchenya, E., Parinov, A., Druzhinin, M. and Kalinin, A.A., 2020. Albumentations: fast and flexible image augmentations. Information, 11(2), p.125.
9. Olah, C., Mordvintsev, A. and Schubert, L., 2017. Feature visualization. Distill, 2(11), p.e7.

Appendix

Appendix-A

Here are some of the patterns generated by the DL Model,



Here are some more fractal images generated,

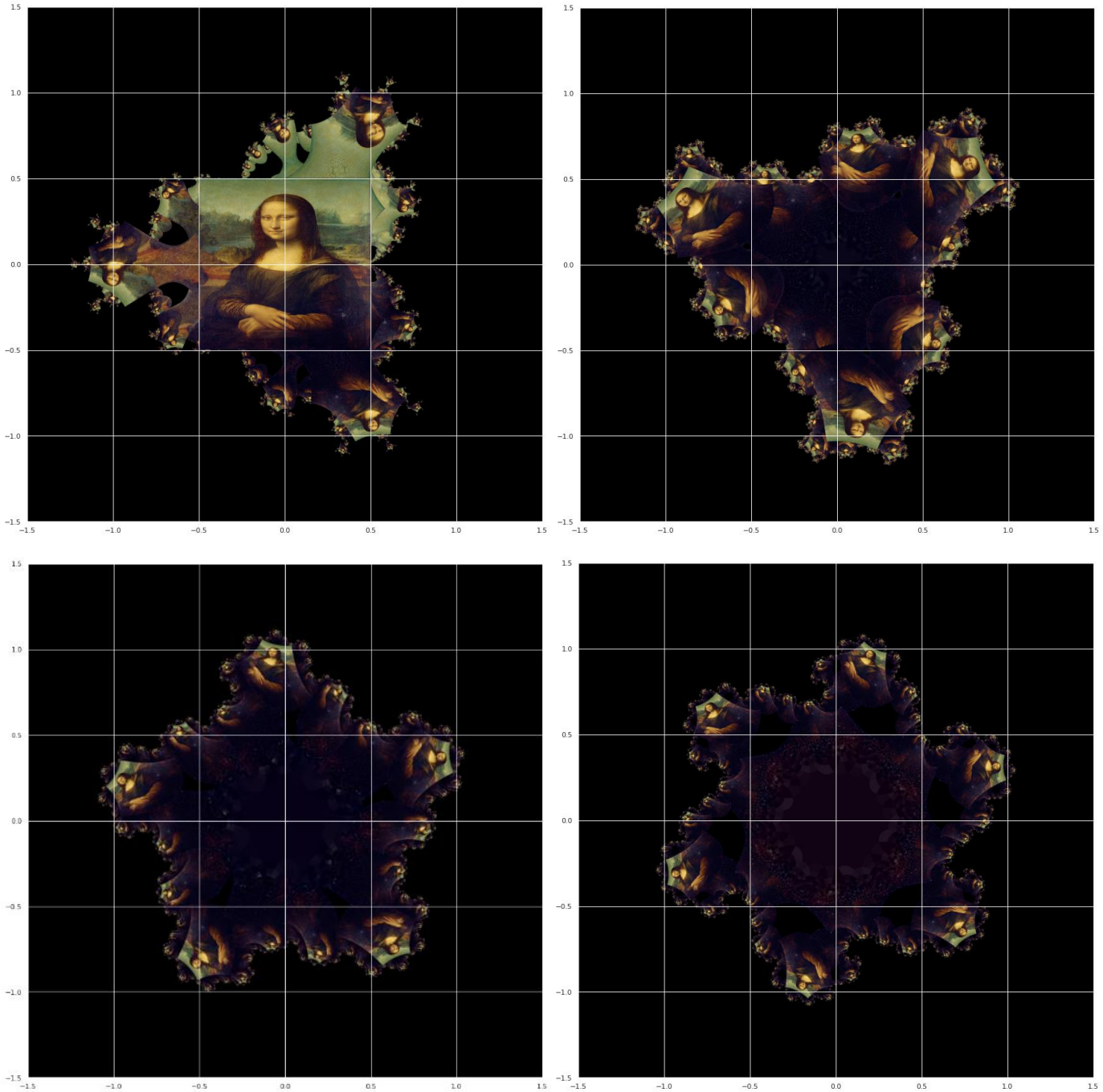


Figure 15 MandelBrot (deg=4), Julia (deg=3), Julia (deg=5), Julia (deg=6) from left to right, top to bottom