**DEEP DREAM**

Machine learning project report submitted in partial fulfilment of the requirement for the degree of Bachelor of Technology

in

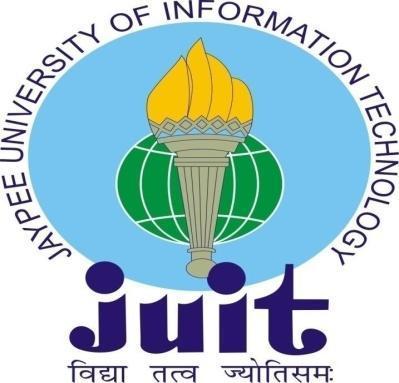
# **Computer Science and Engineering**

By

| **NAME** | **ROLL NO** |
| --- | --- |
| DIVYANSH MANDHAN | 191270 |
| KRITIKA PATHAK | 191285 |

**PRESENTED TO**

DR. MONIKA BHARTI



Department of Computer Science & Engineering

and Information Technology

# **Jaypee University of Information Technology, Waknaghat, 173234, Himachal Pradesh, INDIA**

**TABLE OF CONTENT**

| **Title** | **Page No.** |
| --- | --- |
| **Declaration** | **III** |
| **Certificate** | **IV** |
| **Acknowledgement** | **05** |
| **Abstract** | **06** |
| **Problem Definition** | **07** |
| **Feature Dataset** | **10** |
| **Algorithm/Pseudo code** | **12** |
| **Alternative Approach** | **16** |
| **Results and References** | **18** |

**DECLARATION**

I hereby declare that this project has been done by me and my classmate.I also declare that neither this project nor any part of this project has been submitted elsewhere for award of any degree or diploma.

**Submitted by:**

**Divyansh Mandhan**

(191270)

**Kritika Pathak**

(191285)

Department of Computer Science & Engineering and Information Technology

Ⅲ

**CERTIFICATE**

This is to certify that the work which is being presented in the project report titled “**Deep Dream**” in partial fulfilment of the requirements for the award of the degree of B.Tech in Computer Science And Engineering and submitted to the Department of Computer Science And Engineering, Jaypee University of Information Technology, Waknaghat is an authentic record of work carried out by “**Divyansh Mandhan, 191270**” & “**Kritika Pathak, 191285**” during the period from January 2022 to May 2022 under the course coordinator **Dr. Monika Bharti**, Department of Computer Science and Engineering, Jaypee University of Information Technology, Waknaghat.

**Divyansh Mandhan**

(191270)

**Kritika Pathak**

(191285)

The above statement made is correct to the best of my knowledge.

**Dr. Monika Bharti**

Assistant Professor (SG)

Department of Computer Science & Engineering and Information Technology

Jaypee University of Information Technology

Ⅳ

**ACKNOWLEDGEMENT**

Firstly, I express my heartiest thanks and gratefulness to almighty God for His divine blessing making it possible for us to complete the project work successfully.

I am really grateful and wish my profound indebtedness to Course Coordinator **Dr. Monika Bharti, Assistant Professor (SG)**, Department of CSE Jaypee University of Information Technology,Wakhnaghat. Deep Knowledge & keen interest of my supervisor in the field of “**Neural Networks**” to carry out this project. Her endless patience, scholarly guidance, continual encouragement, constant and energetic supervision, constructive criticism, valuable advice, reading many inferior drafts and correcting them at all stages have made it possible to complete this project.

I would like to express my heartiest gratitude to **Dr. Monika Bharti,** Department of CSE, for his kind help to finish my project.

I would also generously welcome each one of those individuals who have helped me straightforwardly or in a roundabout way in making this project a win. In this unique situation, I might want to thank the various staff individuals, both educating and non-instructing, which have developed their convenient help and facilitated my undertaking.

Finally, I must acknowledge with due respect the constant support and patients of my parents.

**Divyansh Mandhan**

(191270)

**Kritika Pathak**

(191285)

**ABSTRACT**

DeepDream is an artistic algorithm where a pretrained CNN is fed an image and optimised to amplify the features it "sees" in the image.

We do that by doing a gradient ascent on the (usually) MSE loss constructed over the activations coming from a certain layer.

Depending on the neural network layer the features amplified will either be low level (like edges, certain geometric patterns, etc.) or high level (like dog snouts, eyes, etc.) that heavily depends on the dataset on which the net was pretrained!

**Problem Definition**

The idea behind this algorithm so given an input image the thing that happens is that the neural network we run pre-trained model on some like classification test or something like on imagenet or mit places 365 basically sees some something in that image certain features and whatever the network sees we just tell it to just amplify those exact features and by doing that you get the image. [Artificial Neural Networks](http://en.wikipedia.org/wiki/Artificial_neural_network) have spurred remarkable recent progress in [image classification](http://googleresearch.blogspot.com/2014/09/building-deeper-understanding-of-images.html#uds-search-results) and [speech recognition](https://www.youtube.com/watch?v=yxxRAHVtafI). But even though these are very useful tools based on well-known mathematical methods, we actually understand surprisingly little of why certain models work and others don’t. So let’s take a look at some simple techniques for peeking inside these networks. We train an artificial neural network by showing it millions of training examples and [gradually adjusting the network parameters](https://en.wikipedia.org/?title=Backpropagation) until it gives the classifications we want. The network typically consists of 10-30 stacked layers of artificial neurons. Each image is fed into the input layer, which then talks to the next layer, until eventually the “output” layer is reached. The network’s “answer” comes from this final output layer. One of the challenges of neural networks is understanding what exactly goes on at each layer. We know that after training, each layer progressively extracts higher and higher-level features of the image, until the final layer essentially makes a decision on what the image shows. For example, the first layer may look for edges or corners. Intermediate layers interpret the basic features to look for overall shapes or components, like a door or a leaf. The final few layers assemble those into complete interpretations—these neurons activate in response to very complex things such as entire buildings or trees. One way to visualise what goes on is to turn the network upside down and ask it to enhance an input image in such a way as to elicit a particular interpretation. So here’s one surprise: neural networks that were trained to discriminate between different kinds of images have quite a bit of the information needed to generate images too. Check out some more examples across different classes: Instead of exactly prescribing which feature we want the network to amplify, we can also let the network make that decision. In this case we simply feed the network an arbitrary image or photo and let the network analyse the picture. We then pick a layer and ask the network to enhance whatever it detected. Each layer of the network deals with features at a different level of abstraction, so the complexity of features we generate depends on which layer we choose to enhance. For example, lower layers tend to produce strokes or simple ornament-like patterns, because those layers are sensitive to basic features such as edges and their orientations.

| **Fig1.1 *Original photo by*** [***Zachi Evenor***](https://www.flickr.com/photos/zachievenor/8258092492/in/set-72157630014410078)***. Right: processed by Günther Noack, Software Engineer*** |
| --- |
| If we choose higher-level layers, which identify more sophisticated features in images, complex features or even whole objects tend to emerge. Again, we just start with an existing image and give it to our neural net. We ask the network: “Whatever you see there, I want more of it!” This creates a feedback loop: if a cloud looks a little bit like a bird, the network will make it look more like a bird. This in turn will make the network recognize the bird even more strongly on the next pass and so forth, until a highly detailed bird appears, seemingly out of nowhere.  This technique gives us a qualitative sense of the level of abstraction that a particular layer has achieved in its understanding of images. We call this technique “Inceptionism” in reference to the [neural net architecture](http://arxiv.org/pdf/1409.4842.pdf) used. See our [Inceptionism gallery](https://goo.gl/photos/fFcivHZ2CDhqCkZdA) for more pairs of images and their processed results, plus some cool video animations. We must go deeper: Iterations  If we apply the algorithm iteratively on its own outputs and apply some zooming after each iteration, we get an endless stream of new impressions, exploring the set of things the network knows about. We can even start this process from a random-noise image, so that the result becomes purely the result of the neural network, as seen in the following images:   |  | | --- | | ***Fig 1.2 Neural net “dreams”— generated purely from random noise, using a network trained on places by*** [***MIT Computer Science and AI Laboratory***](http://places.csail.mit.edu/)***. See our*** [***Inceptionism gallery***](https://goo.gl/photos/fFcivHZ2CDhqCkZdA) ***for hi-res versions of the images above and more (Images marked “Places205-GoogLeNet” were made using this network).*** |   The techniques presented here help us understand and visualise how neural networks are able to carry out difficult classification tasks, improve network architecture, and check what the network has learned during training. It also makes us wonder whether neural networks could become a tool for artists—a new way to remix visual concepts—or perhaps even shed a little light on the roots of the creative process in general.  (Alexander Mordvintsev, p.1).  **Feature Dataset**  Our dataset is nothing but some images  Some examples would be:  **Fig2.1 Lion.jpg**    **Fig2.2 Figure.jpg**    **Fig2.3 toys.jpg**  **Algorithm Used**  We have used gradient ascent on Mean Squared Error(MSE) loss.  Gradient ascent works in the same manner as gradient descent, with one difference. The task it fulfils isn’t minimization, but rather maximisation of some function. The reason for the difference is that, at times, we may want to reach the maximum, not the minimum of some function; this is the case, for instance, if we’re maximising the distance between [separation hyperplanes and observations](https://www.baeldung.com/cs/ml-support-vector-machines#2-separation-hyperplane-and-support-vectors).  For this reason, the formula that describes gradient ascent is similar to the one for gradient descent. Only, with a flipped sign:    If gradient descent indicates an iterative movement towards the closest minimum, gradient ascent, conversely, indicates a movement towards the nearest maximum. In this sense, for any function f on which we apply gradient descent, there is a symmetric function -f on which we can apply gradient ascent.  This means also that a problem tackled through gradient descent also has solutions that we can find through gradient ascent, if only we reflect it upon the axis of the independent variable. This image shows the same function of the previous graph, but reflected along the x axis:    **Fig 3.1 Gradient ascent**  If we use a positive log-likelihood, then the objective function is concave and we must use gradient ascent.  Basically in , gradient ascent so basically the only difference is you just change the sign uh when you do the update for a single pixel you don't do the minus where you use the learning rate and the the gradients you just switch it to plus.  The model we used, single CNN : The VGG16  VGG16 is a variant of the VGG model with 16 convolution layers and we have explored the VGG16 architecture in depth.  VGGNet-16 consists of 16 convolutional layers and is very appealing because of its very uniform Architecture. Similar to AlexNet, it has only 3x3 convolutions, but lots of filters. It can be trained on 4 GPUs for 2–3 weeks. It is currently the most preferred choice in the community for extracting features from images. The weight configuration of the VGGNet is publicly available and has been used in many other applications and challenges as a baseline feature extractor.  However, VGGNet consists of 138 million parameters, which can be a bit challenging to handle. VGG can be achieved through transfer Learning. In which the model is pretrained on a dataset and the parameters are updated for better accuracy and you can use the parameters values.    **Fig 3.2: VGG16 Layers**    **Fig 3.3: VGG16 architecture**  **More concepts such as gradient smoothing and Image Pyramid are used:**  **CascadeGaussianSmoothing**  The Gaussian smoothing operator is a 2-D convolution operator that is used to `blur' images and remove detail and noise. In this sense it is similar to the mean filter, but it uses a different kernel that represents the shape of a Gaussian (`bell-shaped') hump. This kernel has some special properties.    **Fig 3.4 Cascade Gaussian Smoothing**  **Image Pyramid**  We'll be feeding the image in various resolutions into the CNN. By doing that the network will see different things each time and that will in return give us a richer output.  Note: ***that happens because the ratio between the receptive field of the CNN and the input image changes and thus sometimes the net will see the entire image and thus can create global features and in other cases it will see a small portion of the image and it can focus more on the texture.***  In order to feed multiple resolutions in, we'll need to define something called an **image pyramid**.    **Fig 3.4 Image Pyramid**  **Alternative Approaches**  **If we used another , minimization techniques you would end up with would be that the input image would either become black or more probably just some random noise image.**  **Gradient Descent**  Gradient descent is an iterative process through which we optimise the parameters of a machine learning model. It’s particularly used in neural networks, but also in logistic regression and support vector machines.  It’s the most typical method for iterative minimization of a cost function. Its major limitation, though, consists of its guaranteed convergence to a local, not necessarily global, minimum:    **Fig 4.1 Gradient descent approach**  A hyperparameter , also called the learning rate, allows the fine-tuning of the process of descent. In particular, with an appropriate choice of , we can escape the convergence to a local minimum, and descend towards a global minimum instead. The gradient is calculated with respect to a vector of parameters for the model, typically the weights w. In neural networks, the process of applying gradient descent to the weight matrix takes the name of the backpropagation of the error.  Backpropagation uses the sign of the gradient to determine whether the weights should increase or decrease. The sign of the gradient allows us to decide the direction of the closest minimum to the cost function. For a given parameter \alpha, we iteratively optimise the vector w by computing:      **Fig 4.2 Gradient descent on convex fn**    **Results**  **After feeding image**    **Fig 4.3 Input.jpg**    **Fig 4.4 Output.jpg**  **References:**  **Alexander Mordvintsev, Software Engineer, Christopher Olah, Software Engineering Intern** |