arXiv:1910.04857v1 [cs.CV] 27 Sep 2019

Sampling the "Inverse Set" of a Neuron: An Approach to Understanding Neural Nets

Suryabhan Singh Hada Miguel Á. Carreira-Perpiñán
Electrical Engineering and Computer Science, University of California, Merced
http://eecs.ucmerced.edu

September 25, 2019

Abstract

With the recent success of deep neural networks in computer vision, it is important to understand the internal working of these networks. What does a given neuron represent? The concepts captured by a neuron may be hard to understand or express in simple terms. The approach we propose in this paper is to characterize the region of input space that excites a given neuron to a certain level; we call this the *inverse set*. This inverse set is a complicated high dimensional object that we explore by an optimization-based sampling approach. Inspection of samples of this set by a human can reveal regularities that help to understand the neuron. This goes beyond approaches which were limited to finding an image which maximally activates the neuron (Simonyan et al., 2014) or using Markov chain Monte Carlo to sample images (Nguyen et al., 2017), but this is very slow, generates samples with little diversity and lacks control over the activation value of the generated samples. Our approach also allows us to explore the intersection of inverse sets of several neurons and other variations.

1 Introduction

Recently, deep neural networks have shown great results in solving problems in the field of computer vision. This leads to a surge in the usage of deep neural networks in real life applications, which makes it very important to understand the working of deep neural networks. Some machine learning models are readily interpretable, that is, by looking at the parameters and structure of the model, we can understand the prediction for a given input. For instance, in the case of the univariate decision tree, the prediction for a given input can be written as a sequence of single-feature tests. Another example is the nearest-neighbor classifier, where the model is the training data, and the prediction is given by the most similar training data to the input. However, this is not the case with deep neural networks, where the parameters of the model are interleaved in a very complex way. It is very difficult to make the prediction just by looking at the parameters of the model. In this paper, we propose a general approach to interpret deep neural networks and other complex machine learning models. We achieve this by characterizing the region of input space that excites a given neuron to a certain level; we call this the *inverse set*. That is, rather than looking at the model directly, we propose to treat the model as a black box and characterize its observed behavior on data.

In biology, one puts an electrode into the neuron (Electrophysiology) and records its response while a number of stimuli are fed to it. This is the classic Hubel and Wiesel experiment. We can replicate the same experiment by passing the training data to the network and record its activation for the neuron of interest. However, there are issues with it, as there is no guarantee that we know the data on which network is trained. Even if we know the training data, this approach will never characterize the input space accurately, because the training data is limited. This leads to incomplete information about the nature of neuron. Besides, real-life images contain a lot of irrelevant data, so there is no way to tell which part of the image is important to the neuron. Therefore, we will characterize our input space with synthetic inputs.

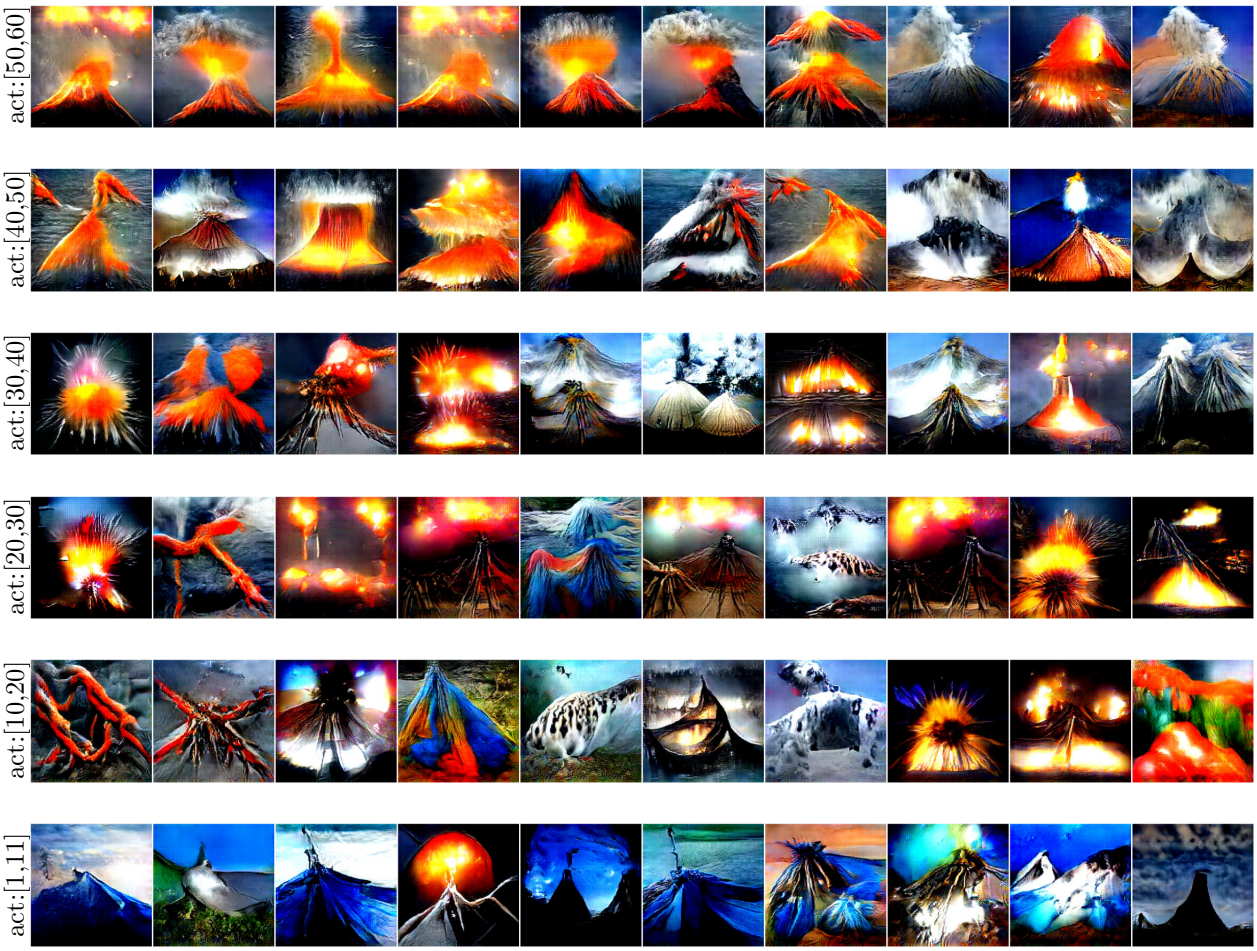

Figure 1: Samples generated using our sampling approach for the neuron number 981 in the fc8 layer of the CaffeNet (Jia et al., 2014), which represents volcano class. Each row contains 10 samples picked from 500 samples generated with activation range mentioned on the left side. Generated samples are not just photo-realistic but also very diverse in nature, characterizing input space around certain activation very well. Unlike previous approaches (Nguyen et al., 2017); generated samples not just only contains volcanoes but they also contain lava flowing through water and on land, and the ash cloud after the volcanic eruption. These kind of samples never been seen before. The first row that contains samples of high activation and the last row which have samples with low activation, both have volcanoes but the presence of lava and smoke create a huge difference in activation value of the neuron. Giving us a clear understanding how the amount of the lava and smoke impact the activation of the neuron, higher the amount of lava and smoke more the activation.

2 Related work

In the last few years, significant effort has been made to understand what parameters of deep neural networks have learned. Currently available techniques that are used to understand the working of deep neural networks involve two major approaches: feature inversion and activation maximization. Feature inversion attempts to project the output of a layer to the input space, to understand what each layer of the network has learned. This is done by back propagation, gradient descent or training a neural network (Mahendran and Vedaldi, 2015; Wei et al., 2015; Zeiler and Fergus, 2014; Dosovitskiy and Brox, 2016a). Xia et al. (2016) used feature inversion to visualize what every filter has learned.

On the other hand, activation maximization involves maximizing the response of a neuron in deep neural network for an input image, as done by Erhan et al. (2009), Simonyan et al. (2014), Yosinski et al. (2015),

Nguyen et al. (2016), Nguyen et al. and many more. Activation maximization is done by taking a random image and then back propagating it through the network to maximize the activation of the neuron of interest.

Both feature inversion and activation maximization suffer from a major problem: the result is a very noisy image or fooling image (Nguyen et al., 2015; Szegedy et al., 2014), which mostly does not make sense to humans. So, to tackle this problem many handcrafted regularizers are used like α-norm (Simonyan et al., 2014), Gaussian blur (Yosinski et al., 2015), total variation (Mahendran and Vedaldi, 2015), jitter (Mordvintsev et al., 2015), data-driven patch priors (Wei et al., 2015), and center-biased regularization (Nguyen et al.), but there are more. All these methods have helped to improve the quality of the generated images, but still not enough to create natural-looking images. Recently, Dosovitskiy and Brox (2016b) trained a neural network using GAN (Goodfellow et al., 2014) to generate realistic images from a feature vector, which is later used by Nguyen et al. (2016) to generate realistic images for activation maximization.

Although these approaches are really good, they do not consider the fact that there are multiple images which can maximally activate the same neuron. Nguyen et al. tried to address this problem by initializing the input image with the mean of a cluster of training images, for activation maximization. Then the authors repeated this process to generate different images using activation maximization. They called it *multifaceted feature visualization*. However, the generated images were very noisy and very little diverse. Later, Nguyen et al. (2017) used their plug and play model to generate comparatively diverse and more realistic images using Markov chain Monte Carlo based sampling approach.

Activation maximization is a good way to visualize the neuron of interest, but it does not consider the fact that real life images do not usually high activations (Nguyen et al., 2016). So, producing images which have very high activation value does not solve the problem entirely to understand what real life images are preferred by a neuron.

So, to better understand the nature of a neuron in a deep neural network, we need an approach that can generate multiple natural looking images, and those images excite a given neuron in a certain activation range. To our best knowledge, this problem hasn't been addressed yet. Although authors in Nguyen et al. (2017) generate more diverse samples for a single neuron compared to all the previous work, they do not address the issue mentioned above. There is no control over the activation value of the generated samples. Besides their Markov chain Monte Carlo based sampling approach is very slow.

In the next section, we describe our approach to address the issue mentioned above.

3 The inverse set of a neuron, and how to sample it

3.1 The inverse set of a neuron: definition

We say an input $\mathbf{x}$ is in the inverse set of a given neuron having a real-valued activation function f if it satisfies the following two properties:

$$z_1 \leq f(\mathbf{x}) \leq z_2 \qquad \text{and} \qquad \mathbf{x} \text{ is a valid input}$$

where z_1, $z_2 \in \mathbb{R}$ are activation values of the neuron. $\mathbf{x}$ being a valid input means the image features are in the valid range (say, pixel values in [0,1]) and it is a natural looking image.

For a simple model, the inverse set can be calculated analytically. For example, consider a linear model with logistic activation function $\sigma(\mathbf{w}^T\mathbf{x} + c)$ and all valid inputs to have pixel values between [0,1]. For $z_2 = 1$ (maximum activation value) and $0 < z_1 < z_2$, the inverse set will be the intersection of the half space $\mathbf{w}^T\mathbf{x} + c \geq \sigma^{-1}(z_1)$ and the [0,1] hypercube.

One way to look at this inverse set is as feasibility problem which has general form as:

$$\arg\min_{\mathbf{x}} \; 1 \quad \text{such that: set of constraints}(\mathbf{x}) \quad \Leftrightarrow \quad \mathbf{X} = \{\mathbf{x} : \text{set of constraints}(\mathbf{x})\}. \tag{1}$$

In the case of deep neural networks, these constraints are nonlinear. So, we approximate the $\mathbf{X}$ with a sample S that covers it in a representative way. The difficult part is to find an efficient algorithm to construct S. A simple way to do this is to select all the images in the training set that satisfy eq. (1), but this may rule out all images. A neuron may "like" certain aspects of a training image without being sufficiently activated by it, or, in other words, the images that activate a given neuron need not look like any specific training image. Therefore, we need an efficient algorithm to sample the inverse set.

3.2 Sampling the inverse set of a neuron: an optimization approach

To generate the maximum activation image, the problem can be mathematically formulated as:

$$\arg\max_{\mathbf{x}} f(\mathbf{x}) + \mathcal{R}(\mathbf{x}) \tag{2}$$

where real-valued function f gives the activation value of the given neuron for an input image $\mathbf{x}$. $\mathcal{R}$ is a regularizer which makes sure that the generated image $\mathbf{x}$ looks like a real image. This is the same objective function used in Simonyan et al. (2014), Yosinski et al. (2015), Nguyen et al., and others, where, $\mathcal{R}$ is replaced by their hand-crafted regularizers. As mentioned in Nguyen et al., it mostly produces the same images for a given neuron. However, to construct the sample $S = \{\mathbf{x}_1, \cdots, \mathbf{x}_n\}$ that covers the inverse set, generated images should be different from each other. So, we propose the following formulation to construct S of size n as a constraint optimization problem:

$$\arg\max_{\mathbf{x}_1,\mathbf{x}_2,\cdots,\mathbf{x}_n} \sum_{i,j=1}^{n} \|\mathbf{x}_i - \mathbf{x}_j\|_2^2 \qquad \text{s.t.} \quad z_1 \leq f(\mathbf{x}_1), \ldots, f(\mathbf{x}_n) \leq z_2. \tag{3}$$

The objective function makes sure that the samples are different from each other and also satisfy eq. (1). However, this generates noisy-looking samples. To make them realistic we use an image generator network $\mathbf{G}$, which has been empirically shown to produce realistic images (Dosovitskiy and Brox, 2016b) when a feature vector $\mathbf{c}$ is passed as an input. Then we get:

$$\arg\max_{\mathbf{c}_1,\mathbf{c}_2,\cdots,\mathbf{c}_n} \sum_{i,j=1}^{n} \|\mathbf{G}(\mathbf{c}_i) - \mathbf{G}(\mathbf{c}_j)\|_2^2 \qquad \text{s.t.} \quad z_1 \leq f(\mathbf{G}(\mathbf{c}_1)), \ldots, f(\mathbf{G}(\mathbf{c}_n)) \leq z_2. \tag{4}$$

We observe that using Euclidean distances directly on the generated images $\mathbf{G}(\mathbf{c})$ is very sensitive to small changes in their pixels. Instead, we compute distances on a low-dimensional encoding $\mathbf{E}(\mathbf{G}(\mathbf{c}))$ of the generated images, where $\mathbf{E}$ is obtained from the first layers of a deep neural network trained for classification. Then we have our final formulation of the optimization problem over the n samples $\mathbf{G}(\mathbf{c}_1), \ldots, \mathbf{G}(\mathbf{c}_n)$:

$$\arg\max_{\mathbf{c}_1,\mathbf{c}_2,\cdots,\mathbf{c}_n} \sum_{i,j=1}^{n} \|\mathbf{E}(\mathbf{G}(\mathbf{c}_i)) - \mathbf{E}(\mathbf{G}(\mathbf{c}_j))\|_2^2 \quad \text{s.t.} \quad z_1 \leq f(\mathbf{G}(\mathbf{c}_1)), \ldots, f(\mathbf{G}(\mathbf{c}_n)) \leq z_2. \tag{5}$$

Now to generate the samples, initialize $\mathbf{c}$ with random values and then optimize eq. (5) using augmented Lagrangian (Nocedal and Wright, 2006).

In theory eq. (5) is enough to generate the sample S. But in practice, as the n gets larger which is required to correctly sample the inverse set, eq. (5) pose two issues: First, because of the quadratic complexity of the objective function over the number of samples n, it is computationally expensive to generate many samples. Second, since it involves optimizing all codes together, for larger n it is not possible to fit all in the GPU memory. In the next section, we describe a much faster and less computationally expensive approach to create the inverse set.

3.3 Sampling in feasible region

In this section, we solve the problem for sampling the set S described in eq: (5) with a faster approach. For this, we apply two approximations.

First, we solve the problem in an inexact but good enough way. The sum-of-all-pairs objective is not a strict necessity; it is really a mechanism to ensure the diversity of the samples and coverage of the inverse set. We observe that this is already achieved by stopping the optimization algorithm once the samples enter the feasible set, by which time they already are sufficiently separated.

Second, we create the samples incrementally, K samples at a time (with $K \ll n$). For the first K samples (which we call *seeds* ($\mathbf{C}_0$)) we optimize eq. (5), initializing the code vectors $\mathbf{c}$ with random values and stopping as soon as all K samples are in the feasible region. These samples are then fixed. The next K samples are generated by the following equation:

$$\underset{\mathbf{c}_1,\mathbf{c}_2,\cdots,\mathbf{c}_K}{\arg\max} \sum_{i,j=1}^{K} \|\mathbf{E}(\mathbf{G}(\mathbf{c}_i)) - \mathbf{E}(\mathbf{G}(\mathbf{c}_j))\|_2^2 + \sum_{i=1}^{K}\sum_{y=1}^{|C_0|} \|\mathbf{E}(\mathbf{G}(\mathbf{c}_i)) - \mathbf{E}(\mathbf{G}(\mathbf{c}_y))\|_2^2$$
$$\text{s.t.} \quad z_1 \leq f(\mathbf{G}(\mathbf{c}_1)), \ldots, f(\mathbf{G}(\mathbf{c}_K)) \leq z_2 \text{ and } \mathbf{c}_y \in \mathbf{C}_0. \tag{6}$$

The first part of the equation "$\sum_{i,j=1}^{K} \|\mathbf{E}(\mathbf{G}(\mathbf{c}_i)) - \mathbf{E}(\mathbf{G}(\mathbf{c}_j))\|_2^2$" is similar to that of eq: (5), means samples should be apart from each other. On the other hand, the second part of the equation $\sum_{i=1}^{K}\sum_{y=1}^{|C_0|} \|\mathbf{E}(\mathbf{G}(\mathbf{c}_i)) - \mathbf{E}(\mathbf{G}(\mathbf{c}_y))\|_2^2$ makes sure that the generated samples should be far apart from the previous ones. The presence of constraints makes sure that generated samples stay in the feasible region.

Now to pick next K samples, we initialize them to the previous K samples ($\mathbf{C}_0$) and take a single gradient step in the augmented Lagrangian optimization of eq: (6). This gives K new samples ($\mathbf{G}(\mathbf{c}_i)$) which we fix, and the process is repeated until we generate the desired n samples.

Note that, here we are not trying to optimize anything. We are using eq: (6) to take steps inside the feasible region. It is like taking a random walk, but here we are taking steps inside the feasible region in a way that the next step gives the samples which are different each other as well as from the one we already have. Another good part of this approach is that unlike the previous sampling approaches eq: (6), this approach allows us to pick samples in parallel which further increase the speed of sampling. In most of our experiments $K = 10$ and $n = 500$. It took almost 85 gradient step of eq: (6) to generate rest of the 490 $(n - K)$ samples with $K = 10$ seeds. These are just a few extra gradient steps than the minimum required 49 gradient steps.

3.4 Optimization details

As described above creating the inverse set of a neuron takes place in two steps:

- Finding seeds in the feasible region.
- Use these seeds to pick samples inside the feasible region.

For finding seeds we optimize eq: (5), that can be transformed into:

$$\underset{\mathbf{c}_1,\mathbf{c}_2,\cdots,\mathbf{c}_K}{\arg\max} \sum_{i,j=1}^{K} \|\mathbf{E}(\mathbf{G}(\mathbf{c}_i)) - \mathbf{E}(\mathbf{G}(\mathbf{c}_j))\|_2^2$$
$$\text{s.t.} \quad z_2 - f(\mathbf{G}(\mathbf{c}_i)) \leq \epsilon \quad \forall i \in \{1, \cdots, K\} \text{ and } \quad \epsilon = z_2 - z_1. \tag{7}$$

This is a constraint optimization problem with inequality constraints. We can solve it by introducing slack variables (s) and then using bound-constrained augmented Lagrangian. So, eq: (7) will transform to:

$$\begin{aligned}\mathcal{L}_{\text{seed}}(\mathbf{C}; \mathbf{s}, \boldsymbol{\lambda}, \mu) = \underset{\mathbf{C}}{\arg\min} & - \sum_{i,j=1}^{K} \|\mathbf{E}(\mathbf{G}(\mathbf{c}_i)) - \mathbf{E}(\mathbf{G}(\mathbf{c}_j))\|_2^2 \\ & - \sum_{i=1}^{K} \lambda_i(\epsilon_0 - z_2 + f(\mathbf{G}(\mathbf{c}_i)) - s_i) \\ & + \sum_{i=1}^{K} \frac{\mu}{2}(\epsilon_0 - z_2 + f(\mathbf{G}(\mathbf{c}_i)) - s_i)^2 \\ & \text{s.t.} \qquad s_i \geq 0 \quad \forall i \in \{1, \cdots, K\}.\end{aligned} \tag{8}$$

Here, $\epsilon_0 = \frac{\epsilon}{2}$, $\mathbf{s} = \{s_1, s_2, \cdots, s_K\}$ and $\mathbf{C} = \{\mathbf{c}_1, \mathbf{c}_2, \cdots, \mathbf{c}_K\}$. The reason to make $\epsilon_0 = \frac{\epsilon}{2}$ is that once we have seeds($\mathbf{C}_0$), they should be in well inside the feasible region rather on the boundary. This helps generate samples to get started from the middle of the feasible region, which pushes them to both sides of the feasible region one towards the z_2 and other away from z_2. It helps generated samples to cover the inverse set at much faster pace.

Now, to optimize eq: (8), initialize $\mathbf{C}$ by passing K initial images $\mathbf{X}_0$ (we used random values) through the encoder($\mathbf{E}$) , $\boldsymbol{\lambda} \in \mathbb{R}^K$ with zero column vector of size K and $\mu \in \mathbb{R}$ with small scalar value $\mu_0 > 0$.

Now apply coordinate descent first in $\mathbf{s}$, then in $\mathbf{C}$:

- **In s**:

$$\arg\min_{\mathbf{s}} \mathcal{L}_{\text{seed}}(\mathbf{C}; \mathbf{s}, \boldsymbol{\lambda}, \mu) \quad \text{s.t.} \quad s_i \geq 0 \quad \forall i \in K$$
$$\implies s_i^r = \max\left(0, \epsilon_0 - z_2 + f(\mathbf{G}(\mathbf{c}_i)) - \frac{1}{\mu_r}\lambda_i^r\right).$$

  Since $\mathcal{L}_{\text{seed}}$ is a convex, separable quadratic form on $\mathbf{s}$.

- **In $\mathcal{L}_{\text{seed}}$**: substitute the value of $\mathbf{s}$ in $\mathcal{L}_{\text{seed}}$ and then solve it approximately using gradient descent as unconstrained problem.

After each step of coordinate descent first update $\boldsymbol{\lambda}$ then μ as follows:

- $\lambda_i^{r+1} \leftarrow \max(\lambda_i^r - \mu^r(\epsilon_0 - z_2 + f(\mathbf{G}(\mathbf{c}_i)), 0)$.
- $\mu^{r+1} \leftarrow a\mu^r$ where $a \in \mathbb{R}$ and $a > 1$.

We stop the optimization process when all $\mathbf{c}$ reach to the feasible region, means

$$z_2 - f(\mathbf{G}(\mathbf{c}_i)) \leq \epsilon \quad \forall i \in \{1, \cdots, K\}.$$

The codes $\mathbf{C}$ which we get after this optimization act as seeds($\mathbf{C}_0$) for doing sampling in the feasible region using eq: (6).

Let us say the value of augmented Lagrangian parameters at the end of optimization is $\boldsymbol{\lambda}^*$ and μ^*.

In most of our experiments value of μ starts with 10 and updated with a multiplication factor of 10 that is $\alpha = 10$. For each iteration of coordinate descent we run 100 steps of gradient descent to approximately optimize $\mathcal{L}_{\text{seed}}$ with $K = 10$. It takes around 4 iterations of coordinate descent to find the seeds in the feasible region which takes approximately $4 - 6$ minutes on NVIDIA Quadro P5000-16GB GPU, that we used for all our experiments.

Now, to pick samples in the feasible region using seeds we use following eq: (6), and that can be transformed into:

$$\arg\max_{\mathbf{c}_1, \mathbf{c}_2, \cdots, \mathbf{c}_K} \sum_{i,j=1}^{K} \|\mathbf{E}(\mathbf{G}(\mathbf{c}_i)) - \mathbf{E}(\mathbf{G}(\mathbf{c}_j))\|_2^2 + \sum_{i=1}^{K}\sum_{y=1}^{|C_0|} \|\mathbf{E}(\mathbf{G}(\mathbf{c}_i)) - \mathbf{E}(\mathbf{G}(\mathbf{c}_j))\|_2^2$$
$$\text{s.t.} \quad z_2 - f(\mathbf{G}(\mathbf{c}_i)) \leq \epsilon \quad \forall i \in \{1, \cdots, K\} \text{ and } \quad \epsilon = z_2 - z_1. \tag{9}$$

As described in the paper we need to take the gradient of eq: (9) with respect to $\mathbf{C} = \{\mathbf{c}_1, \mathbf{c}_2, \cdots, \mathbf{c}_K\}$ to generate rest of the samples. But eq: (9) contains inequality constraints so we transform it into bound-constrained augmented Lagrangian as we did for eq: (7), but the augmented Lagrangian parameters $\boldsymbol{\lambda}$ and μ are initialized with $\boldsymbol{\lambda}^*$ and μ^* and never changed again throughout the sampling process.

$$\mathcal{L}_{\text{sample}}(\mathbf{C}; \mathbf{C}_0, \mathbf{s}, \boldsymbol{\lambda}^*, \mu^*) = \arg\min_{\mathbf{C}} - \sum_{i,j=1}^{K} \|\mathbf{E}(\mathbf{G}(\mathbf{c}_i)) - \mathbf{E}(\mathbf{G}(\mathbf{c}_j))\|_2^2 - \sum_{i=1}^{K}\sum_{y=1}^{|C_0|} \|\mathbf{E}(\mathbf{G}(\mathbf{c}_i)) - \mathbf{E}(\mathbf{G}(\mathbf{c}_j))\|_2^2$$
$$- \sum_{i=1}^{K} \lambda_i^*(\epsilon_0 - z_2 + f(\mathbf{G}(\mathbf{c}_i)) - s_i) + \sum_{i=1}^{K} \frac{\mu^*}{2}(\epsilon_0 - z_2 + f(\mathbf{G}(\mathbf{c}_i)) - s_i)^2$$
$$\text{s.t.} \quad s_i \geq 0 \quad \forall i \in \{1, \cdots, K\}. \tag{10}$$

Now to do the sampling we apply following steps:

1. Initialize $\mathbf{C}$ with $\mathbf{C}_0$.

2. Update $\mathbf{s}$ as follows:

$$s_i^r \leftarrow \max\left(0, \epsilon_0 - z_2 + f(\mathbf{G}(\mathbf{c}_i)) - \frac{1}{\mu^*}\lambda_i^*\right).$$

3. Take **single** gradient step of $\mathcal{L}_{\text{sample}}$ with respect to $\mathbf{C}$ and update $\mathbf{C}$ to $\mathbf{C}'$ as follows:

$$\mathbf{C}' \leftarrow \mathbf{C} - \beta\frac{\partial\mathcal{L}_{\text{sample}}}{\partial\mathbf{C}}$$

where $\beta \in \mathbb{R}$ is the step length.

4. Replace $\mathbf{C}_0$ with $\mathbf{c}_i'$ which are in feasible region that is for which $z_2 - f(\mathbf{G}(\mathbf{c}_i')) \leq \epsilon$. These $\mathbf{G}(\mathbf{C}_0)$ are the new samples.

5. Assign $\mathbf{C}'$ to $\mathbf{C}$.

6. Repeat step 2, 3, 4 and 5 until all n desired are not generated.

Step 3, not only helps the algorithm to move inside the feasible region but it also provides a direction where the generated samples are different from the one we already have. The presence of constraint terms in eq: (10) play a very important role in the sampling process. In absence of the constraint the eq: (10) will be as below:

$$\mathcal{L}_{\text{sample}}(\mathbf{C}; \mathbf{C}_0) = \arg\min_{\mathbf{C}} - \sum_{i,j=1}^{K} \|\mathbf{E}(\mathbf{G}(\mathbf{c}_i)) - \mathbf{E}(\mathbf{G}(\mathbf{c}_j))\|_2^2 - \sum_{i=1}^{K}\sum_{y=1}^{|C_0|} \|\mathbf{E}(\mathbf{G}(\mathbf{c}_i)) - \mathbf{E}(\mathbf{G}(\mathbf{c}_y))\|_2^2$$

which simply means the new codes $\mathbf{c}_i$ should be far apart from each other as well as from the one we already have ($\mathbf{C}_0$). This leads the gradient (step 3) to move the new samples ($\mathbf{G}(\mathbf{C}')$) outside the feasible region. After few iterations, all the generated samples ($\mathbf{G}(\mathbf{C}')$) will be out of the feasible region and there is no way to bring them back to the feasible region, which makes it impossible to generate a large number of samples. Infact , the constraint terms act as a penalty in case any of the $\mathbf{G}(\mathbf{c}_i')$ goes outside the feasible region which pushes it back to the feasible region in next gradient update steps(step 3).

3.5 Intersection of inverse-sets

Our method also allows us to visualize the intersection of multiple inverse sets. This can be achieved by modifying the constraints in eq: (5) as:

$$\begin{aligned}
&\arg\max_{\mathbf{c}_1,\mathbf{c}_2,\cdots,\mathbf{c}_n} \sum_{i,j=1}^{n} \|\mathbf{E}(\mathbf{G}(\mathbf{c}_i)) - \mathbf{E}(\mathbf{G}(\mathbf{c}_j))\|_2^2 \\
&\qquad \text{s.t.} \qquad z_1^{(1)} \leq f_1(\mathbf{G}(\mathbf{c}_1)), \ldots, f_1(\mathbf{G}(\mathbf{c}_n)) \leq z_2^{(1)}, \\
&\qquad\qquad\qquad\qquad \vdots \\
&\qquad\qquad\qquad z_1^{(p)} \leq f_p(\mathbf{G}(\mathbf{c}_1)), \ldots, f_p(\mathbf{G}(\mathbf{c}_n)) \leq z_2^{(p)}
\end{aligned} \tag{11}$$

and in eq: (6) as follows:

$$\begin{aligned}
&\arg\max_{\mathbf{c}_1,\mathbf{c}_2,\cdots,\mathbf{c}_K} \sum_{i,j=1}^{K} \|\mathbf{E}(\mathbf{G}(\mathbf{c}_i)) - \mathbf{E}(\mathbf{G}(\mathbf{c}_j))\|_2^2 + \sum_{i=1}^{K}\sum_{y=1}^{|C_0|} \|\mathbf{E}(\mathbf{G}(\mathbf{c}_i)) - \mathbf{E}(\mathbf{G}(\mathbf{c}_y))\|_2^2 \\
&\qquad \text{s.t.} \qquad z_1^{(1)} \leq f_1(\mathbf{G}(\mathbf{c}_1)), \ldots, f_1(\mathbf{G}(\mathbf{c}_K)) \leq z_2^{(1)}, \\
&\qquad\qquad\qquad\qquad \vdots \\
&\qquad\qquad\qquad z_1^{(p)} \leq f_p(\mathbf{G}(\mathbf{c}_1)), \ldots, f_p(\mathbf{G}(\mathbf{c}_K)) \leq z_2^{(p)}, \\
&\qquad \text{and} \qquad \mathbf{c}_y \in \mathbf{C}_0
\end{aligned} \tag{12}$$

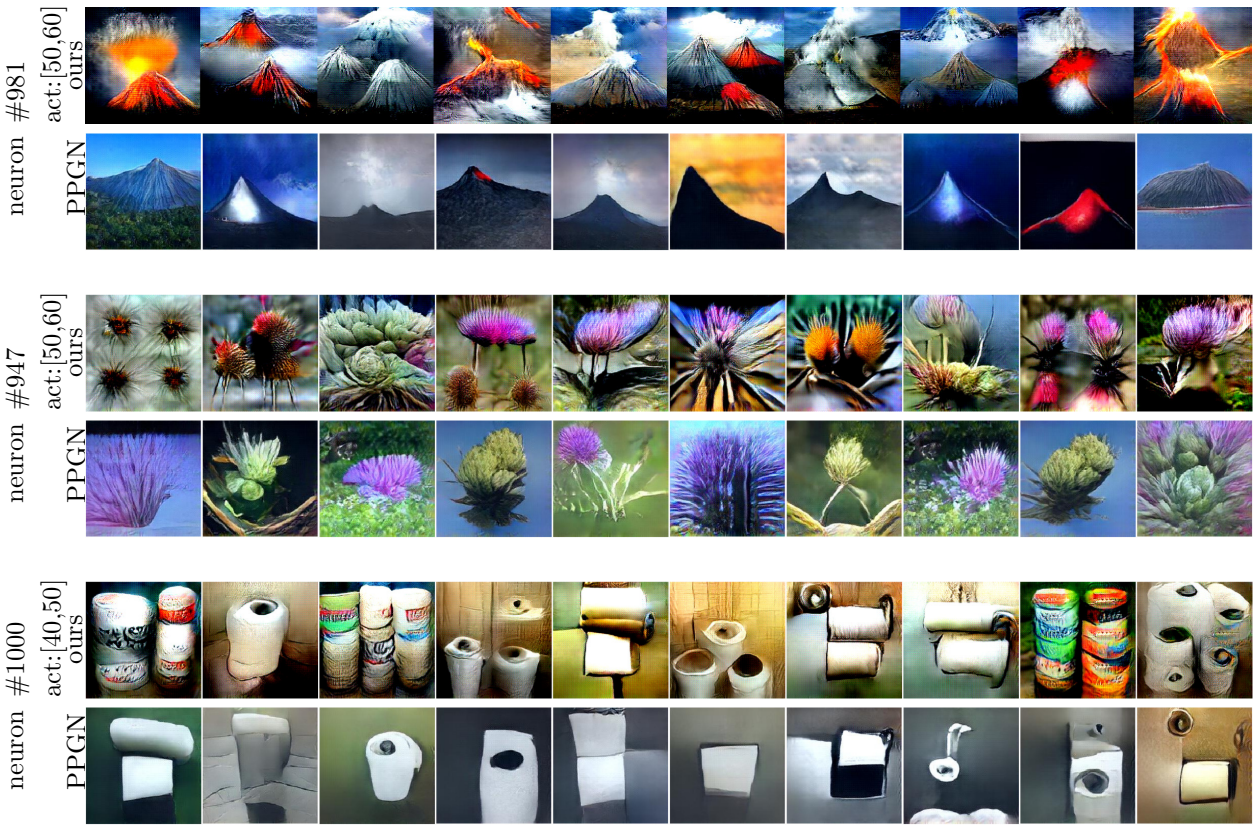

Figure 2: The first, third and fifth row contains 10 samples picked from 500 samples generated by our sampling approach to cover the inverse set for the neuron number 981 (represents volcano class), 947 (represents cardoon class) and 1000 (represents toilet paper class) respectively. All three neurons are from layer fc8 of CaffeNet (Jia et al., 2014). For the first row (volcano class) the activation range is [50,60], for the third (cardoon class) and the fifth row (toilet paper class) the range is [40,50]. The second, fourth and sixth row shows samples generated for the same neurons by sampling approach from Nguyen et al. (2017). Activation range for the samples from Nguyen et al. (2017) is not guaranteed to be in any fixed range like ours.

where, $f_1, \cdots, f_p$ are real valued activation functions for different neurons and $(z_1^{(1)}, z_2^{(1)}), \cdots, (z_1^{(p)}, z_2^{(p)})$ are corresponding activation values.

Figure: 4 shows samples from the intersection of two different inverse sets, corresponding to neurons in the same layer. Eq: (12) can also be used to generate samples from inverse set intersection of neurons in different layers, as shown in figure: 5. Generated samples have a face in the middle of the images. When these samples passed through the network, they excite both neurons in their corresponding activation range.

4 Experiments

All previous works to understand the neural networks took CaffeNet (Jia et al., 2014) a minor variant of AlexNet (Krizhevsky et al., 2012) as the main subject. So, we also used a pre-trained CaffeNet (Jia et al., 2014) for our experiments, which had been trained on ImageNet dataset (Deng et al., 2009). We use Matcovnet (Vedaldi and Lenc, 2015) for all our experiments. Using previous papers (Nguyen et al., 2016, 2017), naming convention we will also call last three fully connected layers in CaffeNet (Jia et al., 2014) as fc6, fc7, and fc8. fc8 is the last layer before softmax. In all our experiments **E** is pre-trained CaffeNet (Jia et al., 2014). However, the network has been shortened to the fc6 layer which is the first fully connected layer. The output of the **E** is a vector of size 4096. **G** is a pre-trained generative network which we picked

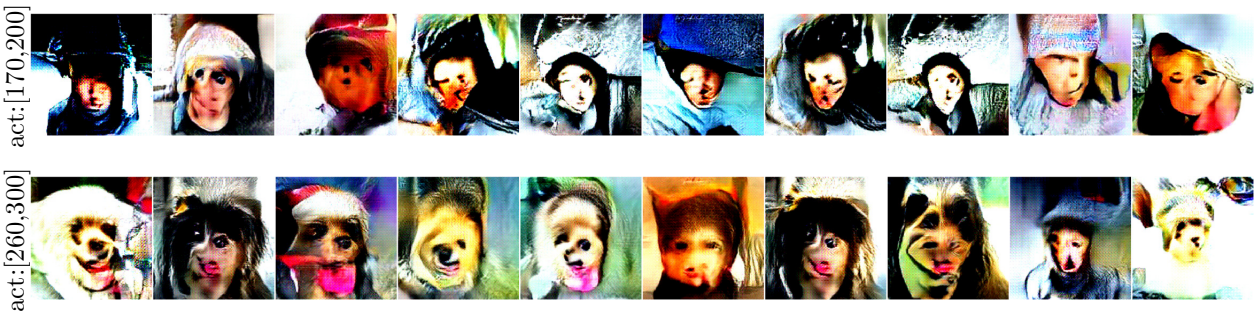

Figure 3: Both rows contain 10 samples out of 500 samples which are generated to cover the inverse set for neuron number 197 in the conv5 layer of CaffeNet (Jia et al., 2014). This neuron is known to detect faces (Yosinski et al., 2015). In the first row, the activation range is $[170, 200]$, and samples are mostly human faces. On the other hand, in the second row which has higher activation range($[260, 300]$), contains dog faces with fur. This shows the neuron prefers faces with fur compare to clean faces.

Figure 4: First two rows show the samples generated by our sampling method for neuron number 664 (represents **Monastery**) and 862 (represents **Toilet seat**) respectively. Both neurons are from layer fc8 of CaffeNet (Jia et al., 2014). Last row contains the samples from their intersection set. All three rows have activation range of [40,50]

from the code provided by (Nguyen et al., 2017) at https://github.com/Evolving-AI-Lab/ppgn. $\mathbf{G}$ has been trained to generate images from a feature vector of size 4096 more specifically output of fc6 layer of CaffeNet (Jia et al., 2014).

To do a fair comparison of our results with (Nguyen et al., 2017), we pick the neuron 981 which represents "Volcano" class. We run our algorithm for six times to generate 500 samples to cover inverse set for this neuron corresponding to different activation levels. Figure:1 shows results of this experiment. We pick 10 samples from generated 500 samples for each $[z_2, z_1]$.

Unlike the previous visualization approaches (Nguyen et al., 2017, 2016; Nguyen et al.) (second row of figure:2), the generated samples are far more diverse and rich in information. Our approach allows us to look at the samples which excite the neuron at different activation levels; this was not possible before. In doing so, it uncovers samples which have never been seen before. For instance, images which do not even contain volcano, instead contain lava flowing through water, but still excite the neuron.

Figure: 2 shows some of the samples generated by our algorithm for a certain activation range with some other neurons and their comparison with Nguyen et al. (2017). To do a fair comparison with Nguyen et al. (2017), we generate samples as it is described in the supplementary section of the paper. We run 10 sampling chains conditioned on various classes each for 200 steps, to produce 2000 samples for each class. We picked 1 sample from each 200 steps to provide as much as diversity possible in the samples.

Figure 5: Above figure contains samples from the intersection of the inverse set of a hidden neuron and a neuron in the fc8 layer of CaffeNet (Jia et al., 2014). The hidden neuron in both rows is neuron number 197 in the conv5 layer, which is known to detect faces. The activation range for the hidden neuron is $[260, 300]$. In the first row, the neuron from the fc8 layer is neuron number 1000 (toilet paper class), and the activation range is $[20, 50]$. The generated samples have activation value in the range of $[260, 300]$ for the hidden neuron and $[20, 50]$ for the neuron number 1000 in the fc8 layer. In the second row the neuron in the fc8 layer is neuron number 862 (toilet seat class), and activation range is $[40, 50]$. The generated samples have activation value in the range of $[260, 300]$ for the hidden neuron and $[40, 50]$ for the neuron number 862 in the fc8 layer.

Figure 6: Figure shows the results of our sampling process over a different network. The network is a variant of AlexNet trained on MIT Places dataset (Zhou et al., 2002). The first row contains samples from the inverse set for neuron number 88 (Gas station class), and the second row contains samples from the inverse set for neuron number 141 (Phone booth class). Both neurons are from the last layer before softmax, and the activation range is $[40, 50]$ in both cases.

Second row shows the generated samples for neuron number 947 which represents "Cardoon" class for $z_2 = 50$ and $z_1 = 40$. The generated samples not only have more diverse color distribution compared to the Nguyen et al. (2017) (fourth row in the figure:2) but also showing samples of different shapes and sizes, that can only be seen in real life images.

To further test whether our sampling truly generates diverse samples or not, we pick a rather difficult class: "Toilet paper", neuron number 1000. The reason for the difficulty is as the realistic looking samples cannot just differ by having a different color like in the case of "Cardoon" class; they all should have the white color. Sampling method should pick the samples which are of different shape or quantity. The fifth row in figure: 2 shows our results. The generated samples are very diverse; they not only show just a toilet roll or toilet rolls hanged with a hanger but also show packed toilet rolls. In fact, samples contain packages with different shapes, labels, and even quantity. The first and ninth sample in the fifth row of figure: 2 shows rolls packed in two packages but have altogether different packaging labels, while the third sample shows toilet rolls packed in 3 packages which also have different packaging label than other two. Our sampling method was even able to pick samples such that the number of toilet rolls is different, as shown in the second, fourth and tenth sample. These type of samples can only be seen in real life images thus showing how perfectly our sampling method covers the inverse set. On the other hand, samples generated by using PPGN (Plug and Play Generative Networks) (Nguyen et al., 2017) (sixth row of figure:2) mostly contains rolls which are standalone or attached to a holder clearly not much diverse.

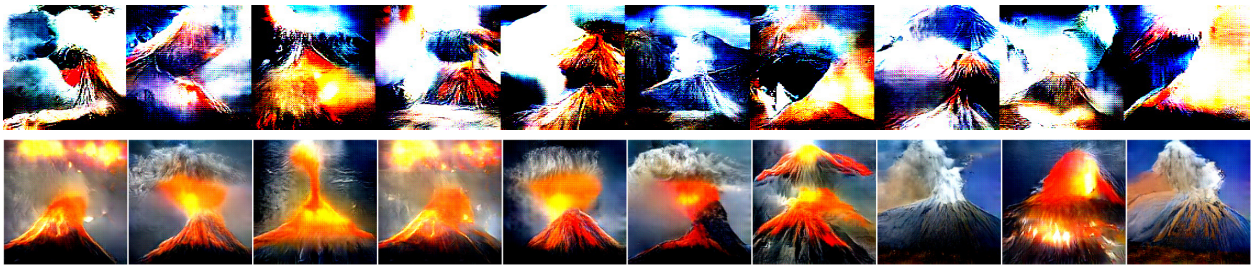

Figure 7: All images are generated for the neuron number 981 in the fc8 layer, which represents Volcano class. In both cases activation range is [50,60]. The first row samples are generated using eq: (13) which looks really bad visually unlike the second row samples which are generated by using eq: (5) that are much closer to real life image.

We also tested our method for hidden neurons. We apply our algorithm for neuron number 197 in the conv5 layer (last convolution layer in the CaffeNet (Jia et al., 2014)). This neuron is known to detect faces (Yosinski et al., 2015). We first apply our approach with activation range $[200, 170]$, most of the generated samples produce human faces as shown in the first row of figure:3. But as we apply the same process with activation range $[300, 260]$, dog's faces started to appear in the samples as shown in the second row of the figure:3. Hence showing this neuron starts getting more activation towards the faces with more fur compared to that of clean faces.

4.1 Need of encoder(E) in sampling process

Both in eq: (5) and eq: (6) samples are separated by taking L2 distance in code space. Mathematically speaking the objective function is just a distance between two vectors. But in our case these vectors(codes $\mathbf{c}$) go through a nonlinear transform (First $\mathbf{G}$ and then $\mathbf{E}$) before L2 distance is applied on them. For instance, in eq: (5) one can argue why we need this transform afterall it is just a distance between two vectors. So, instead of involving encoder($\mathbf{E}$) and generator network($\mathbf{G}$) in eq: (5), we generated samples by optimizing directly over codes in following way:

$$\underset{\mathbf{c}_1,\mathbf{c}_2,\cdots,\mathbf{c}_K}{\arg\max} \sum_{i,j=1}^{K} \|\mathbf{c}_i - \mathbf{c}_j\|_2^2 \quad \text{s.t.} \quad z_1 \leq f(\mathbf{G}(\mathbf{c}_1)), \ldots, f(\mathbf{G}(\mathbf{c}_K)) \leq z_2. \tag{13}$$

Although, this seems to be much cleaner approach then eq: (5) but this results in generating samples which are visually very bad as compare to eq: (5) as shown in the first row of figure: 7. Generated samples have large white spots and they barely look close to the real life images unlike the samples generated by the eq: (5) (second row in figure: 7) which looks visually much better.

The possible reason for this difference in the quality of the generated images is the presence of certain extreme values (large negative and positive values) in the code vector($\mathbf{c}$) when samples are generated using eq: (13). But when samples are generated using eq: (5) the presence of the encoder($\mathbf{E}$) limits the values of the code vector $\mathbf{c}$ in a certain range, which results in more smooth and more natural looking generated images. However, when codes has been optimized using eq: (13) there is no restriction what values of code vector ($\mathbf{c}$) can take, which make them to take certain extreme values (large negative and positive values). This leads to generating images which have big white spots and barely natural looking images as shown in first row of figure: 7.

4.2 Generating diverse samples from a single initial value

To check the effect of term $\sum_{i,j=1}^{K} \|\mathbf{E}(\mathbf{G}(\mathbf{c}_i)) - \mathbf{E}(\mathbf{G}(\mathbf{c}_j))\|_2^2$ in eq: (5) for generating diverse samples we perform following experiment.

First we modify the eq: (5) as follows:

$$\underset{\mathbf{c}_1,\mathbf{c}_2,\cdots,\mathbf{c}_K}{\arg\max} \quad 1 \quad \text{s.t.} \quad z_1 \leq f(\mathbf{G}(\mathbf{c}_1)), \ldots, f(\mathbf{G}(\mathbf{c}_K)) \leq z_2. \tag{14}$$

Figure 8: All images are generated for the neuron number 981 in the fc8 layer, which represents Volcano class. In both cases activation range is [50,60]. The first row samples are generated using eq: (14) such that all $\mathbf{c}$ are initialized with the same random vector. Generated samples look very similar unlike the second row samples that are comparatively more diverse and are generated by using eq: (5) with all $\mathbf{c}$ are initialized with the same value as in the first row.

Then we initialize all $\mathbf{c}$ from same random vector and optimize eq: (14) using coordinate as described in section: 3.4. First row of figure: 4.2 shows the generated samples. All samples are very little diverse. Then we repeat the same experiment to generate samples using eq: (5) initializing all $\mathbf{c}$ with the same value as above. The results are shown in second row of figure : 4.2. As expected generated are comparatively diverse, showing the need of the term $\sum_{i,j=1}^{K} \|\mathbf{E}(\mathbf{G}(\mathbf{c}_i)) - \mathbf{E}(\mathbf{G}(\mathbf{c}_j))\|_2^2$ in eq: (5) for generating diverse samples.

4.3 Proposed fast sampling process is a good approximation of Eq 5

Eq: (5) is actually enough to generate samples which are different from each other and look like real life images. But as described in paper it is very slow and may not be be possible due to memory constraints for large number of samples. So we try to approximate this whole process by our modified sampling process as described in section: 3.4. There is one more approximation we apply while optimizing eq: (5). We stop our optimization as soon as all our samples reach to the feasible region.

To test how good is our approximation we perform following experiment:

1. First we generate n samples instead of K ($K \ll n$) samples at a time, using eq: (5) as follows:

$$\underset{\mathbf{c}_1,\mathbf{c}_2,\cdots,\mathbf{c}_n}{\arg\max} \sum_{i,j=1}^{n} \|\mathbf{E}(\mathbf{G}(\mathbf{c}_i)) - \mathbf{E}(\mathbf{G}(\mathbf{c}_j))\|_2^2 \quad \text{s.t.} \quad z_1 \leq f(\mathbf{G}(\mathbf{c}_1)), \ldots, f(\mathbf{G}(\mathbf{c}_n)) \leq z_2. \tag{15}$$

2. Next, using same initial values (picked first K initial images from n used in above experiment), we first generate K seeds using eq: (5). But instead of stopping the optimization as soon as we reach to feasible region we optimize eq: (5) all the way. After that we generate rest of the $n-K$ samples using eq: (6) as described in section: 3.4.

3. Last we generate all n samples with the same parameters and same initial values for K seeds in above step using sampling process described in section: 3.4.

For these three experiments, the total number of samples generated are $n = 100$ and number of seeds are $K = 10$. The total number of gradient steps took to optimize eq: (5) for experiment in 1 was around 13000. Compare to this in experiment 2, the total number of gradient steps required for eq: (5) were around 800 and rest $n - K$ samples were generated under of 35 steps. Whereas the last experiment took only around 400 gradient steps which take almost 5-6 minutes over a GPU and rest $n - K$ samples were also generated under 35 steps. Figure: 9 shows results of this experiment. After looking at the results it is pretty clear that the proposed sampling process does a good job at approximating the solution of eq: (15) in much fewer iterations.

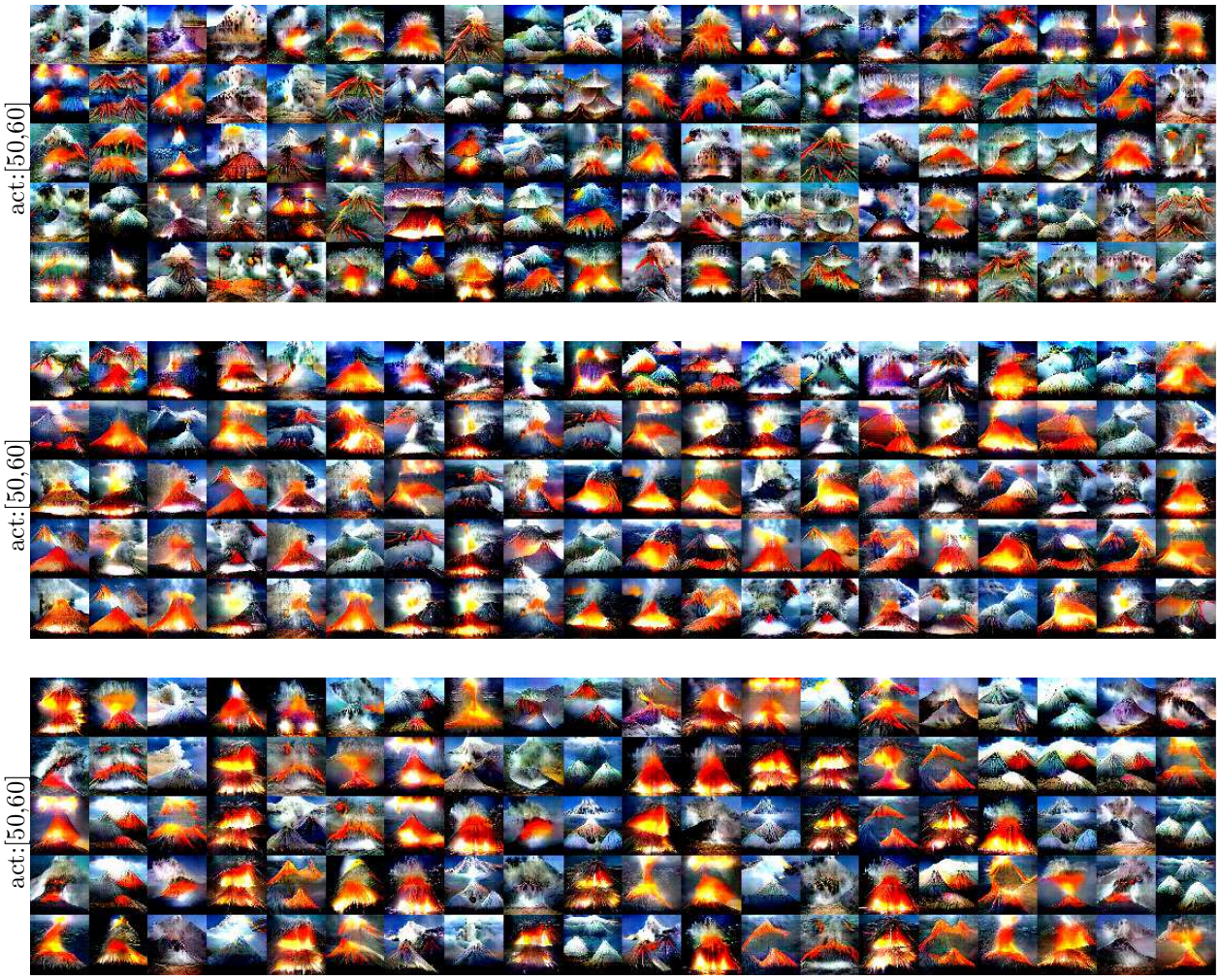

Figure 9: In all three cases neuron in concern is number 981 in layer fc8 of CaffeNet (Jia et al., 2014) with activation range $[50, 60]$. *First block:* All 100 samples (first 5 rows) are generated by fully optimizing eq: (15). *Second block:* the first row (sixth row in total) represents the seeds which are generated by fully optimizing using eq: (5). That is, unlike proposed sampling approach we didn't stop optimization as soon as all samples were in feasible region. Rest of the samples are generated by our fast sampling method (eq: (6)). *Third block:* the first row (11^{th} row in total) represents the seeds which are generated as per the optimization details described in section: 3.4 and rest of the images are generated by our fast sampling method (eq: (6)). From the figure, it is pretty clear that the proposed sampling approach approximate eq: (15) pretty well in much fewer iterations.

5 Discussion

Admittedly, the goal of understanding what a neuron in a deep neural network may be representing is not a well defined problem. It may well be that a neuron does represent a specific concept, but one which is very difficult to grasp for a human; or that one should look at what a group of neurons may be representing. That said, for some neurons their preferred response does correlate well with intuitive concepts or classes, such as the volcano or monastery examples we give. Our approach is to characterize a neuron's preference by a diverse set of examples it likes, which is something that people sometimes do in order to explain a subjective concept to each other. Also, it may be possible to extract specific concepts from this set of examples using data analysis techniques.

6 Conclusion

In conclusion, we propose a very simple, yet effective generalized approach to understand the neurons in a deep neural network: an approach that does not involve activation maximization or feature inversion, but instead characterizes the region of input space around different activation values of the neuron of interest. Thus overcoming the shortcomings of current visualization approaches and providing a great deal of understanding what parts of an image are responsible for impacting the activation of a neuron. This eventually helps us to have a much better understanding about the nature of a neuron in deep neural networks.

We also provide a simpler, faster, and hyper-parameter free sampling method which generates far more diverse samples than previously proposed methods. Our sampling method is also very generalized; just by modifying the constraints, it can also be used for high dimensional sampling in other domains. Although we performed our experiments with only image classification models, our approach for creating inverse sets can also be applied to deep neural networks in other domains, as well as to other complex machine learning models.

In this work, we propose an effective and efficient arbitrary style transfer approach that does not require learning for every individual style. By applying rigid alignment to style features with respect to content features, we solve the problem of content distortion without sacrificing style patterns in the styled image. Our method can seamlessly adapt the existing multi-layer stylization pipeline and capture style information from those layers too. Our method can also seamlessly perform video stylization, merely by per-frame style transfer. Experimental results demonstrate that the proposed algorithm achieves favorable performance against the state-of-the-art methods in arbitrary style transfer. As further direction, one may replace multiple autoencoders for multi-level style transfer, by training an hourglass architecture similar to Avatar-Net for better efficiency.

References

J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A large-scale hierarchical image database. In *Proc. of the 2009 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'09)*, pages 248–255, Miami, FL, June 20–26 2009.

A. Dosovitskiy and T. Brox. Inverting visual representations with convolutional networks. In *Proc. of the 2016 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'16)*, Las Vegas, NV, June 26 – July 1 2016a.

A. Dosovitskiy and T. Brox. Generating images with perceptual similarity metrics based on deep networks. In D. D. Lee, M. Sugiyama, U. von Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 29, pages 658–666. MIT Press, Cambridge, MA, 2016b.

D. Erhan, Y. Bengio, A. Courville, and P. Vincent. Visualizing higher-layer features of a deep network. Technical Report 1341, Université de Montréal, June 2009.

I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 27, pages 2672–2680. MIT Press, Cambridge, MA, 2014.

Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. arXiv:1408.5093 [cs.CV], June 20 2014.

A. Krizhevsky, I. Sutskever, and G. Hinton. ImageNet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 25, pages 1106–1114. MIT Press, Cambridge, MA, 2012.

A. Mahendran and A. Vedaldi. Understanding deep image representations by inverting them. In *Proc. of the 2015 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'15)*, Boston, MA, June 7–12 2015.

A. Mordvintsev, C. Olah, and M. Tyka. Inceptionism: Going deeper into neural networks, June 17 2015.

A. Nguyen, J. Yosinski, and J. Clune. Multifaceted feature visualization: Uncovering the different types of features learned by each neuron in deep neural networks. In *Visualization for Deep Learning workshop, ICML 2016*.

A. Nguyen, J. Yosinski, and J. Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *Proc. of the 2015 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'15)*, pages 427–436, Boston, MA, June 7–12 2015.

A. Nguyen, A. Dosovitskiy, J. Yosinski, T. Brox, and J. Clune. Synthesizing the preferred inputs for neurons in neural networks via deep generator networks. In D. D. Lee, M. Sugiyama, U. von Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 29, pages 3387–3395. MIT Press, Cambridge, MA, 2016.

A. Nguyen, J. Clune, Y. Bengio, A. Dosovitskiy, and J. Yosinski. Plug & play generative networks: Conditional iterative generation of images in latent space. In *Proc. of the 2017 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'17)*, pages 3510–3520, Honolulu, HI, July 21–26 2017.

J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. Springer-Verlag, New York, second edition, 2006.

K. Simonyan, A. Vedaldi, and A. Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. In *Proc. of the 2nd Int. Conf. Learning Representations (ICLR 2014)*, Banff, Canada, Apr. 14–16 2014.

C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. In *Proc. of the 2nd Int. Conf. Learning Representations (ICLR 2014)*, Banff, Canada, Apr. 14–16 2014.

A. Vedaldi and K. Lenc. MatConvNet: Convolutional neural networks for MATLAB. In *Proc. 23rd ACM Int. Conference on Multimedia*, pages 689–692, 2015.

D. Wei, B. Zhou, A. Torralba, and W. Freeman. Understanding intra-class knowledge inside CNN. arXiv:1507.02379, July 15 2015.

Z. Xia, C. Zhu, Z. Wang, Q. Guo, and Y. Liu. Every filter extracts a specific texture in convolutional neural networks. arXiv:1608.04170, Aug. 16 2016.

J. Yosinski, J. Clune, A. Nguyen, T. Fuchs, and H. Lipson. Understanding neural networks through deep visualization. arXiv:1506.06579, June 15 2015.

M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *Proc. 13th European Conf. Computer Vision (ECCV'14)*, pages 818–833, Zürich, Switzerland, Sept. 6–12 2014.

B. Zhou, A. Lapedriza, J. Xiao, A. Torralba, and A. Oliva. Learning deep features for scene recognition using places database. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 14. MIT Press, Cambridge, MA, 2002.