CS7015 (Deep Learning) : Lecture 13

Visualizing Convolutional Neural Networks, Guided Backpropagation, Deep Dream, Deep Art, Fooling Convolutional Neural Networks

Mitesh M. Khapra

Department of Computer Science and Engineering
Indian Institute of Technology Madras

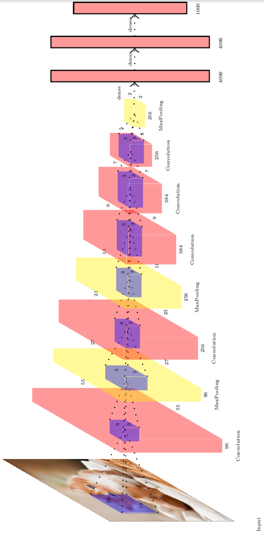
Acknowledgements

- Andrej Karpathy Video Lecture on Visualization and Deep Dream*

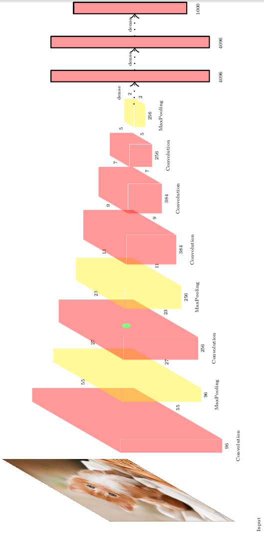
---

*Visualization, Deep Dream, Neural Style, Adversarial Examples

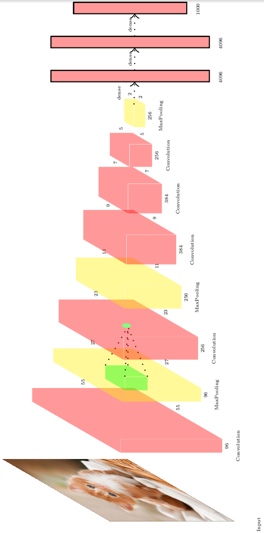
Module 13.1: Visualizing patches which maximally activate a neuron



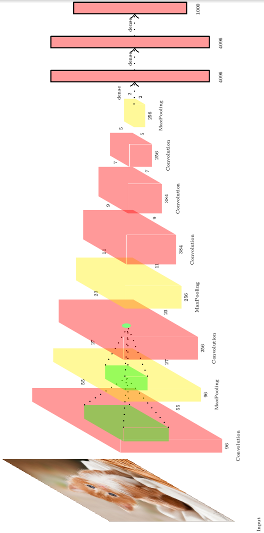
- Consider some neurons in a given layer of a CNN



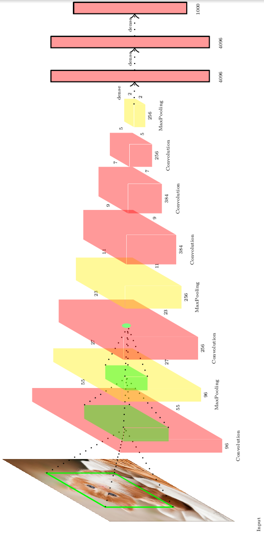
- Consider some neurons in a given layer of a CNN
- We can feed in images to this CNN and identify the images which cause these neurons to fire



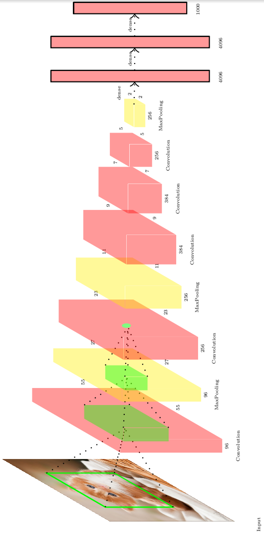
- Consider some neurons in a given layer of a CNN
- We can feed in images to this CNN and identify the images which cause these neurons to fire
- We can then trace back to the patch in the image which causes these neurons to fire



- Consider some neurons in a given layer of a CNN
- We can feed in images to this CNN and identify the images which cause these neurons to fire
- We can then trace back to the patch in the image which causes these neurons to fire



- Consider some neurons in a given layer of a CNN
- We can feed in images to this CNN and identify the images which cause these neurons to fire
- We can then trace back to the patch in the image which causes these neurons to fire



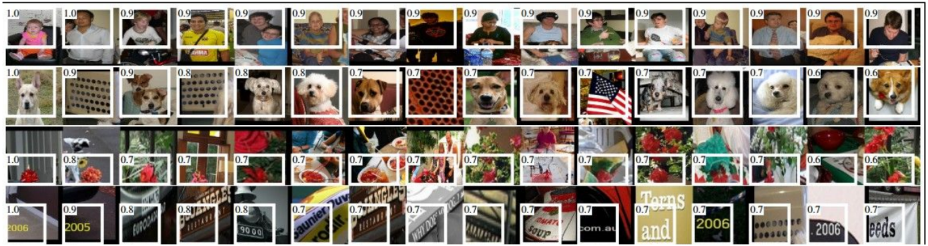
- Consider some neurons in a given layer of a CNN
- We can feed in images to this CNN and identify the images which cause these neurons to fire
- We can then trace back to the patch in the image which causes these neurons to fire
- Let us look at the result of one of such experiment conducted by Grishick et al., 2014

- They consider 6 neurons in the pool5 layer and find the image patches which cause these neurons to fire
- One neuron fires for people faces

- They consider 6 neurons in the pool5 layer and find the image patches which cause these neurons to fire
- Another neuron fires for dog faces

- They consider 6 neurons in the pool5 layer and find the image patches which cause these neurons to fire
- Another neuron fires for flowers

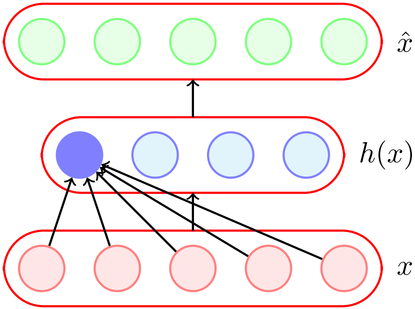
- They consider 6 neurons in the pool5 layer and find the image patches which cause these neurons to fire
- Another neuron fires for numbers



- They consider 6 neurons in the pool5 layer and find the image patches which cause these neurons to fire
- Another neuron fires for houses

- They consider 6 neurons in the pool5 layer and find the image patches which cause these neurons to fire
- Another neuron fires for shiny surfaces

Module 13.2: Visualizing filters of a CNN

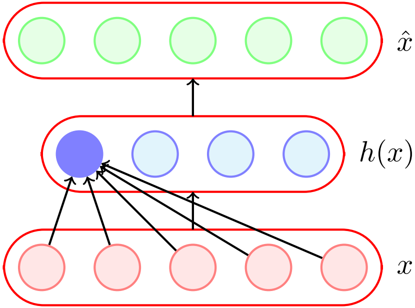


$$\max_{x} \quad \{w^T x\}$$

$$s.t. \quad ||x||^2 = x^T x = 1$$

$$\text{Solution:} \quad x = \frac{w_1}{\sqrt{w_1^T w_1}}$$

- Recall that we had done something similar while discussing autoencoders

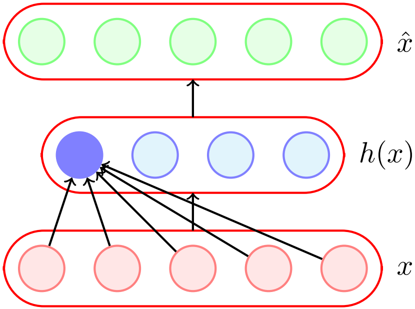


$$\max_x \quad \{w^T x\}$$

$$s.t. \quad ||x||^2 = x^T x = 1$$

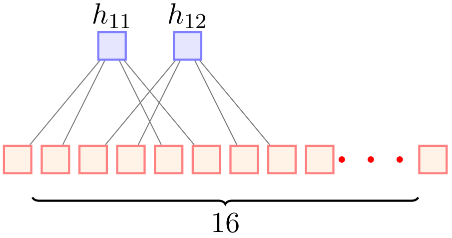
$$\text{Solution:} \quad x = \frac{w_1}{\sqrt{w_1^T w_1}}$$

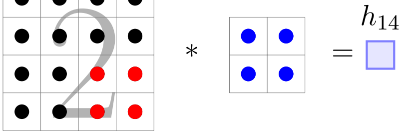
- Recall that we had done something similar while discussing autoencoders
- We are interested in finding an input which maximally excites a neuron



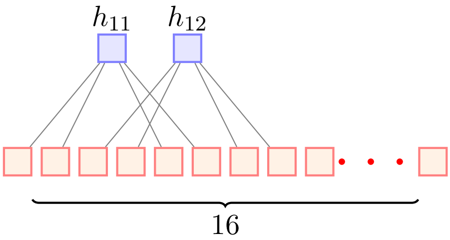
$$\max_{x} \quad \{w^T x\}$$
$$s.t. \quad ||x||^2 = x^T x = 1$$
$$\text{Solution:} \quad x = \frac{w_1}{\sqrt{w_1^T w_1}}$$

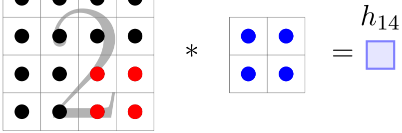
- Recall that we had done something similar while discussing autoencoders
- We are interested in finding an input which maximally excites a neuron
- Turns out that the input which will maximally activate a neuron is $\frac{W}{\|W\|}$





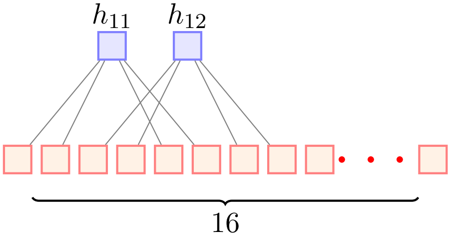
- Now recall that we can think of a CNN also as a feed-forward network with sparse connections and weight sharing

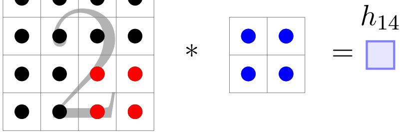




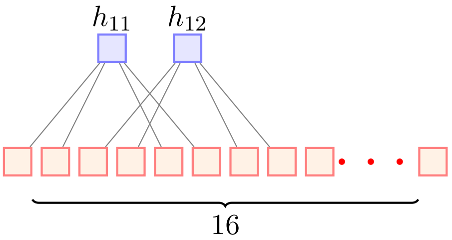
- Now recall that we can think of a CNN also as a feed-forward network with sparse connections and weight sharing

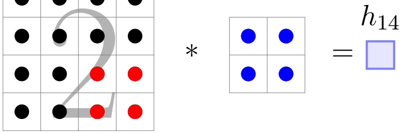
- Once again, we are interested in knowing what kind of inputs will cause a given neuron to fire



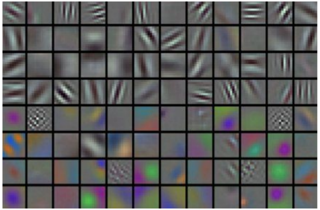


- Now recall that we can think of a CNN also as a feed-forward network with sparse connections and weight sharing
- Once again, we are interested in knowing what kind of inputs will cause a given neuron to fire
- The solution would be the same ($\frac{W}{\|W\|}$) where W is the filter(2×2, in this case)





- Now recall that we can think of a CNN also as a feed-forward network with sparse connections and weight sharing
- Once again, we are interested in knowing what kind of inputs will cause a given neuron to fire
- The solution would be the same $(\frac{W}{\|W\|})$ where W is the filter(2×2, in this case)
- We can thus think of these filters as pattern detectors

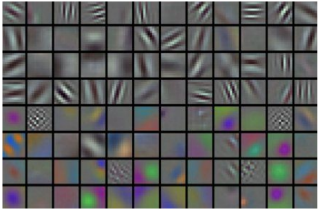


- We can simply plot the $K \times K$ weights (filters) as images & visualize them as patterns

$$\max_{x} \quad \{w^T x\}$$

$$s.t. \quad ||x||^2 = x^T x = 1$$

$$\text{Solution:} \quad x = \frac{w_1}{\sqrt{w_1^T w_1}}$$

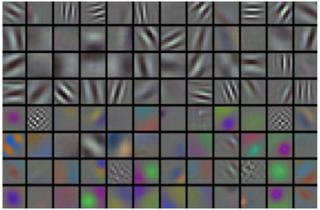


- We can simply plot the $K \times K$ weights (filters) as images & visualize them as patterns
- The filters essentially detect these patterns (by causing the neurons to maximally fire)

$$\max_{x} \quad \{w^T x\}$$

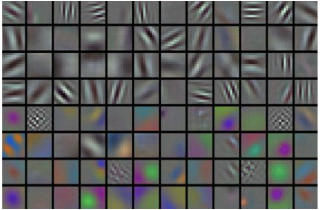
$$s.t. \quad ||x||^2 = x^T x = 1$$

$$\text{Solution:} \quad x = \frac{w_1}{\sqrt{w_1^T w_1}}$$



$$\max_{x} \quad \{w^T x\}$$
$$s.t. \quad ||x||^2 = x^T x = 1$$
$$\text{Solution:} \quad x = \frac{w_1}{\sqrt{w_1^T w_1}}$$

- We can simply plot the $K \times K$ weights (filters) as images & visualize them as patterns
- The filters essentially detect these patterns (by causing the neurons to maximally fire)
- This is only interpretable for the filters in the first convolution layer



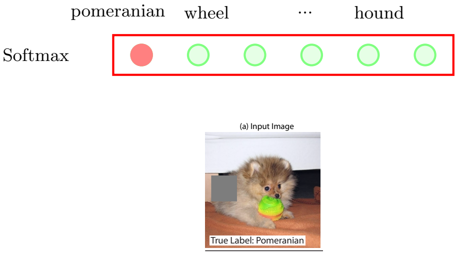
$$\max_{x} \quad \{w^T x\}$$

$$s.t. \quad ||x||^2 = x^T x = 1$$

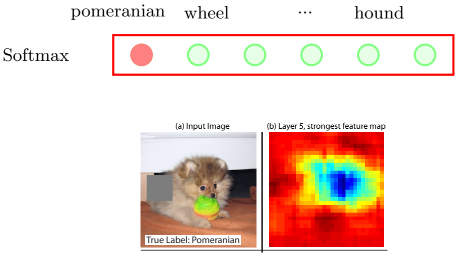
$$\text{Solution:} \quad x = \frac{w_1}{\sqrt{w_1^T w_1}}$$

- We can simply plot the $K \times K$ weights (filters) as images & visualize them as patterns
- The filters essentially detect these patterns (by causing the neurons to maximally fire)
- This is only interpretable for the filters in the first convolution layer (Why?)

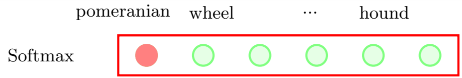
Module 13.3: Occlusion experiments

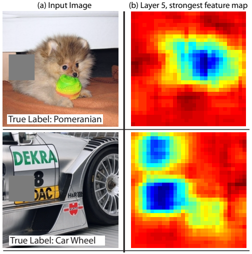


- Typically we are interested in understanding which portions of the image are responsible for maximizing the probability of a certain class

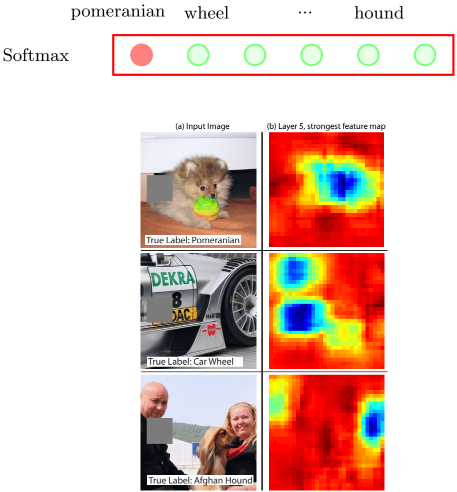


- Typically we are interested in understanding which portions of the image are responsible for maximizing the probability of a certain class
- We could occlude (gray out) different patches in the image and see the effect on the predicted probability of the correct class



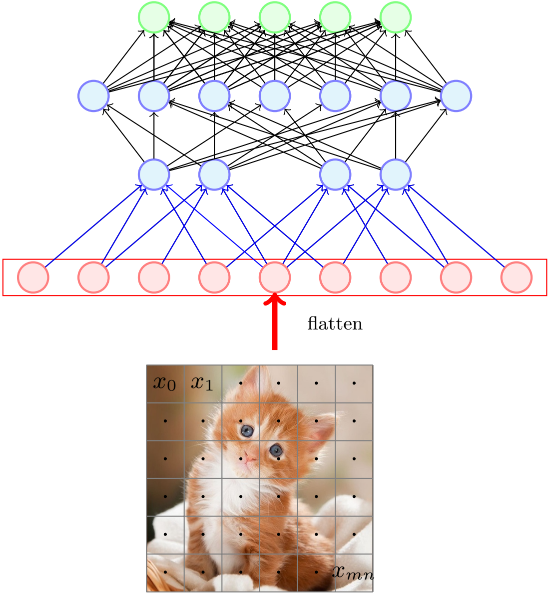


- Typically we are interested in understanding which portions of the image are responsible for maximizing the probability of a certain class
- We could occlude (gray out) different patches in the image and see the effect on the predicted probability of the correct class
- For example this heat map shows that occluding the face of the dog causes a maximum drop in the prediction probability

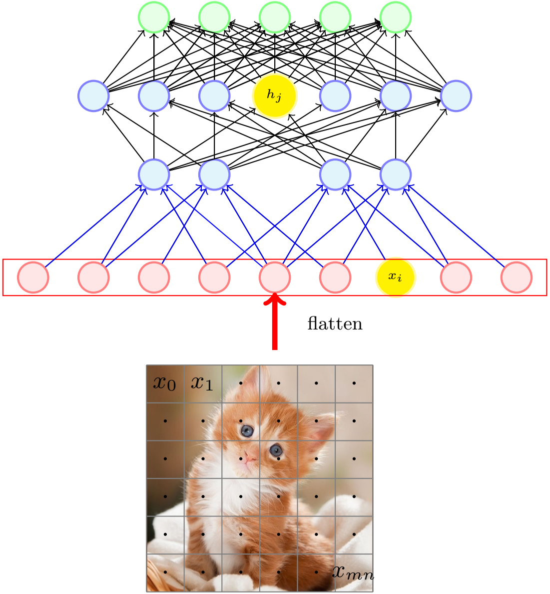


- Typically we are interested in understanding which portions of the image are responsible for maximizing the probability of a certain class
- We could occlude (gray out) different patches in the image and see the effect on the predicted probability of the correct class
- For example this heat map shows that occluding the face of the dog causes a maximum drop in the prediction probability
- Similar observations are made for other images

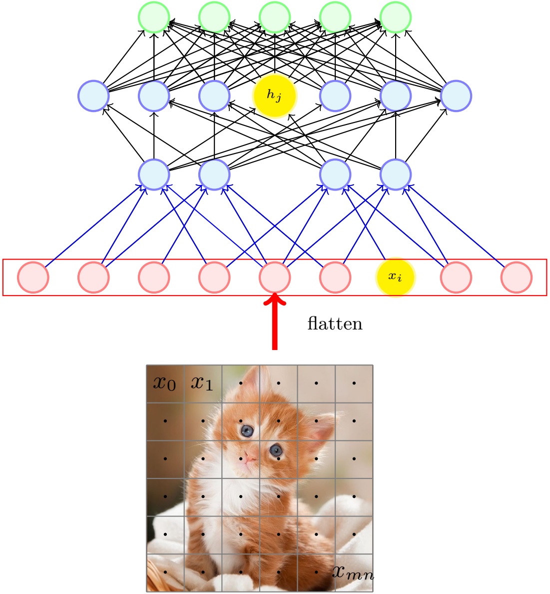
Module 13.4: Finding influence of input pixels using backpropagation



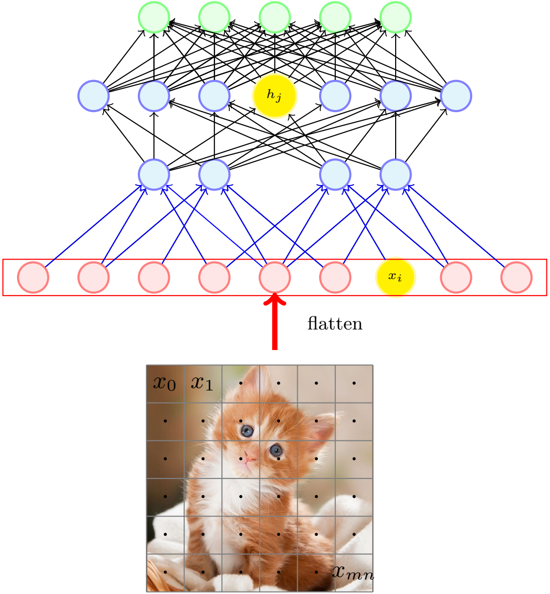
- We can think of an image as a $m \times n$ inputs $x_0, x_1, \ldots, x_{m\times n}$



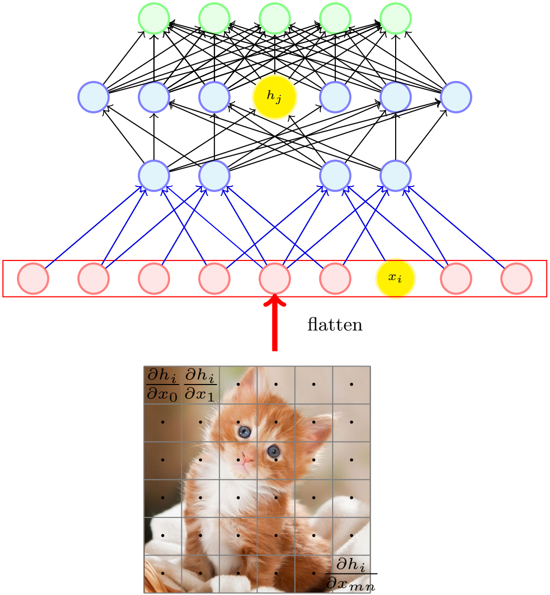
- We can think of an image as a $m \times n$ inputs $x_0, x_1, \ldots, x_{m\times n}$
- We are interested in finding the influence of each of these inputs(x_i) on a given neuron(h_j)



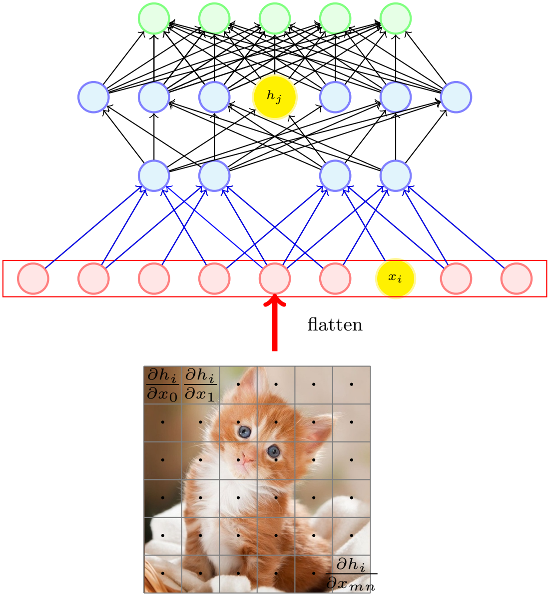
- We can think of an image as a $m \times n$ inputs $x_0, x_1, \ldots, x_{m\times n}$
- We are interested in finding the influence of each of these inputs(x_i) on a given neuron(h_j)
- If a small change in x_i causes a large change in h_j then we can say that x_i has a lot of influence of h_j



- We can think of an image as a $m \times n$ inputs $x_0, x_1, \ldots, x_{m \times n}$
- We are interested in finding the influence of each of these inputs(x_i) on a given neuron(h_j)
- If a small change in x_i causes a large change in h_j then we can say that x_i has a lot of influence of h_j
- In other words the gradient $\frac{\partial h_j}{\partial x_i}$ could tell us about the influence

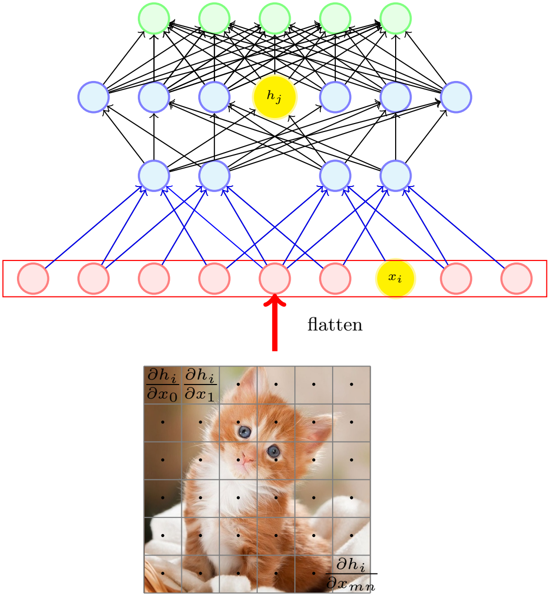


$$\frac{\partial h_j}{\partial x_i} = 0 \qquad \longrightarrow \quad \text{no influence}$$



$$\frac{\partial h_j}{\partial x_i} = 0 \quad \longrightarrow \quad \text{no influence}$$

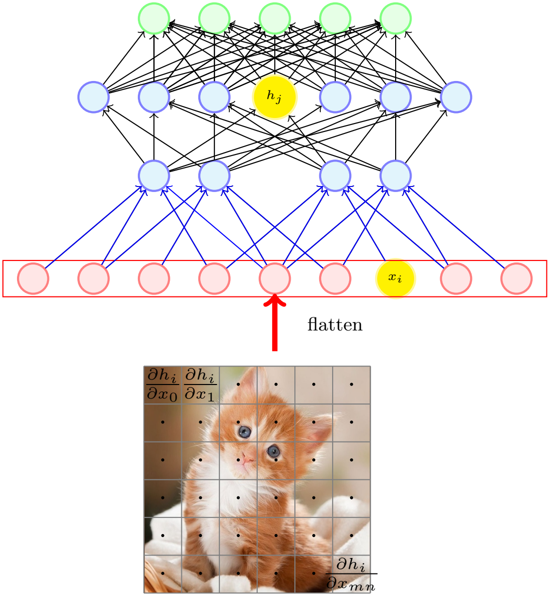
$$\frac{\partial h_j}{\partial x_i} = large \quad \longrightarrow \quad \text{high influence}$$



$$\frac{\partial h_j}{\partial x_i} = 0 \quad \longrightarrow \quad \text{no influence}$$

$$\frac{\partial h_j}{\partial x_i} = large \quad \longrightarrow \quad \text{high influence}$$

$$\frac{\partial h_j}{\partial x_i} = small \quad \longrightarrow \quad \text{low influence}$$

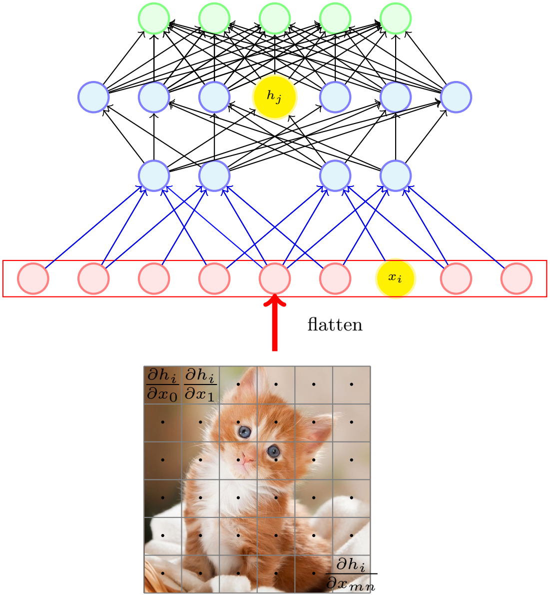


$$\frac{\partial h_j}{\partial x_i} = 0 \longrightarrow \text{no influence}$$

$$\frac{\partial h_j}{\partial x_i} = large \longrightarrow \text{high influence}$$

$$\frac{\partial h_j}{\partial x_i} = small \longrightarrow \text{low influence}$$

- We could just compute these partial derivatives w.r.t all the inputs

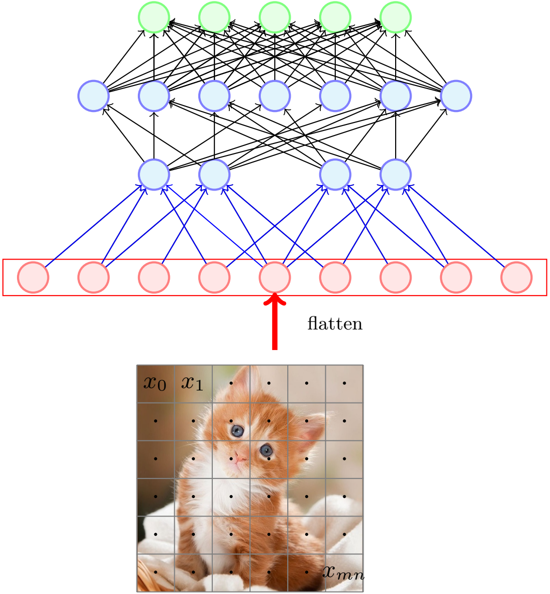


$$\frac{\partial h_j}{\partial x_i} = 0 \qquad \longrightarrow \quad \text{no influence}$$

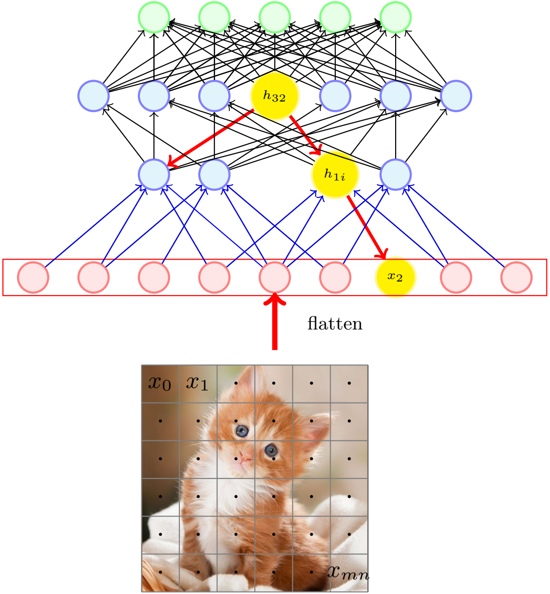
$$\frac{\partial h_j}{\partial x_i} = large \qquad \longrightarrow \quad \text{high influence}$$

$$\frac{\partial h_j}{\partial x_i} = small \qquad \longrightarrow \quad \text{low influence}$$

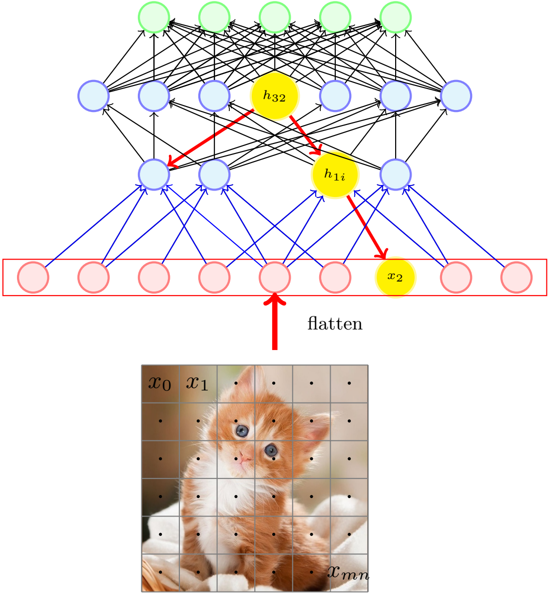
- We could just compute these partial derivatives w.r.t all the inputs
- And then visualize this gradient matrix as an image itself



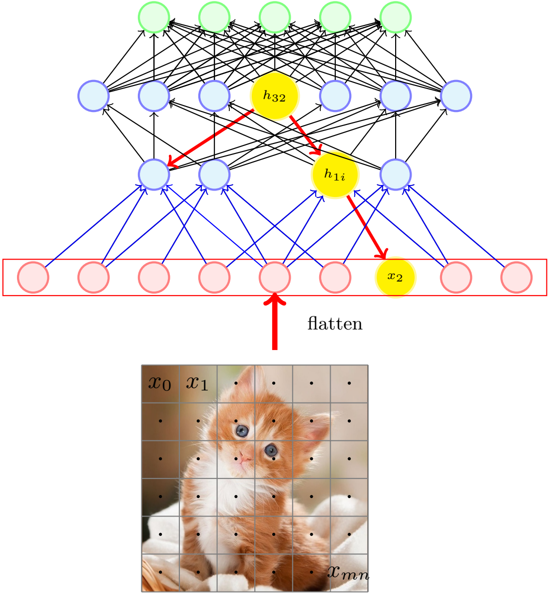
- But how do we compute these gradients?



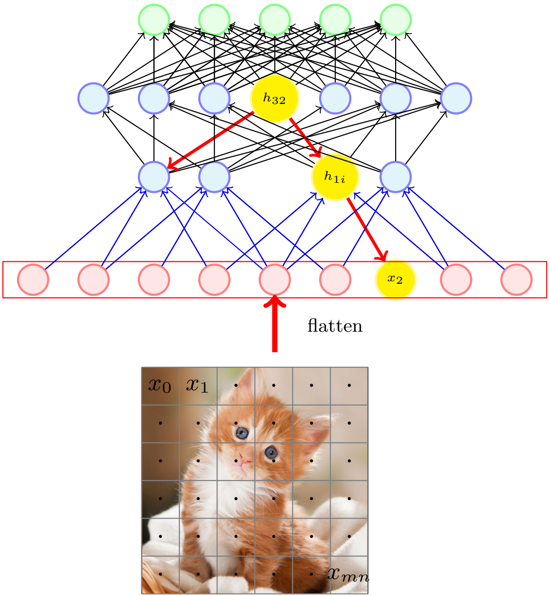
- But how do we compute these gradients?
- Recall that we can represent CNNs by feedforward neural network



- But how do we compute these gradients?
- Recall that we can represent CNNs by feedforward neural network
- Then we already know how to compute influences (gradient) using backpropogation



- But how do we compute these gradients?
- Recall that we can represent CNNs by feedforward neural network
- Then we already know how to compute influences (gradient) using backpropogation
- For example, we know how to backprop the gradients till the first hidden layer



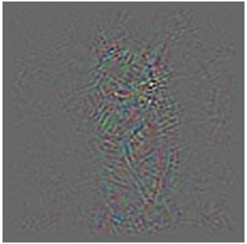
- But how do we compute these gradients?
- Recall that we can represent CNNs by feedforward neural network
- Then we already know how to compute influences (gradient) using backpropogation
- For example, we know how to backprop the gradients till the first hidden layer

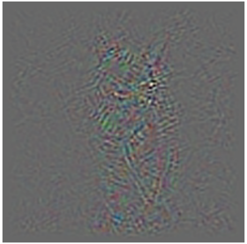
$$\frac{\partial h_{32}}{\partial x_2} = \sum_{i=1}^{3} \frac{\partial h_{32}}{\partial h_{1i}} \frac{\partial h_{1i}}{\partial x_2}$$

$$h_{1i} = \sum_{j=1}^{4} w_{ji} x_j$$

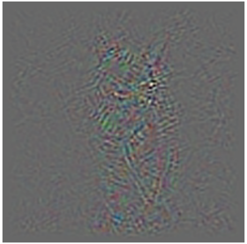
$$\frac{\partial h_{1i}}{\partial x_2} = w_{12}$$

- This is what we get if we compute the gradients and plot it as an image



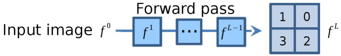


- This is what we get if we compute the gradients and plot it as an image
- The above procedure does not show very sharp influences

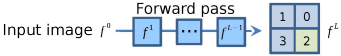


- This is what we get if we compute the gradients and plot it as an image
- The above procedure does not show very sharp influences
- Springenberg et al. proposed "guided back propagation" which gives a better idea about the influences

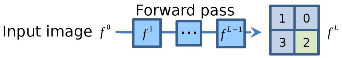
Module 13.5: Guided Backpropagation



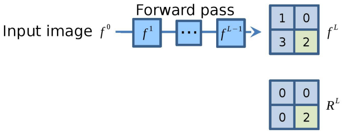
- We feed an input to the CNN and do a forward pass



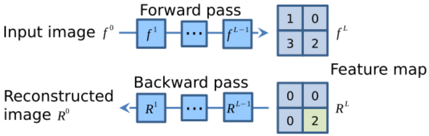
- We feed an input to the CNN and do a forward pass
- We consider one neuron in some feature map at some layer



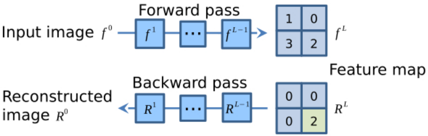
- We feed an input to the CNN and do a forward pass
- We consider one neuron in some feature map at some layer
- We are interested in finding the influence of the input on this neuron



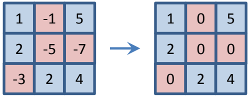
- We feed an input to the CNN and do a forward pass
- We consider one neuron in some feature map at some layer
- We are interested in finding the influence of the input on this neuron
- We retain this neuron and set all other neurons in the layer to zero



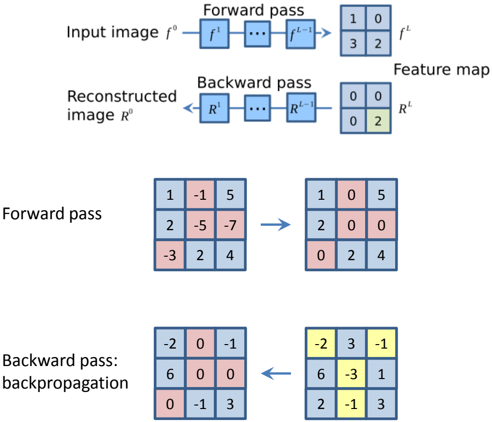
- We now backpropogate all the way to the inputs



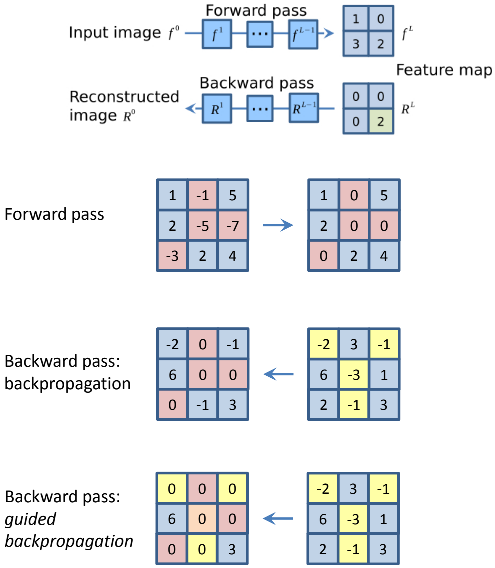
Forward pass



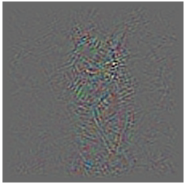
- We now backpropogate all the way to the inputs
- Recall that during forward pass relu activation allows only positive values to pass & clamps $-ve$ values to zero



- We now backpropogate all the way to the inputs
- Recall that during forward pass relu activation allows only positive values to pass & clamps $-ve$ values to zero
- Similarly during backward pass no gradient passes through the dead relu neurons

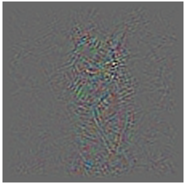


- We now backpropogate all the way to the inputs
- Recall that during forward pass relu activation allows only positive values to pass & clamps $-ve$ values to zero
- Similarly during backward pass no gradient passes through the dead relu neurons
- In guided back propagation any -ve gradients flowing from the upper layer are also set to 0



Backpropagation

- **Intuition:** Neglect all the negative influences (gradients) and focus only on the positive influences (gradients)

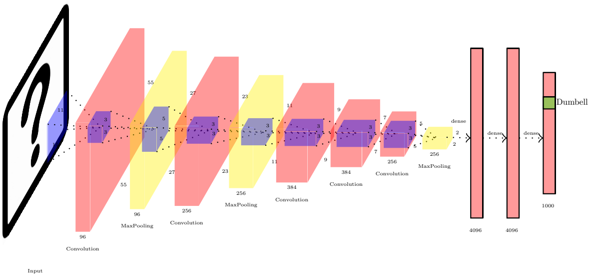


Backpropagation

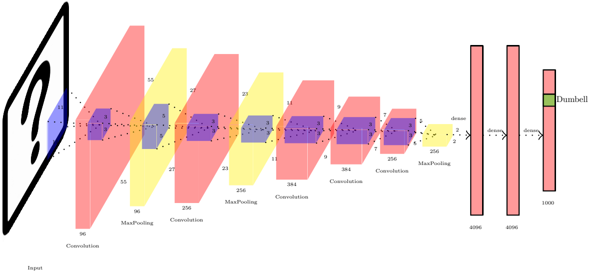
Guided Backpropagation

- **Intuition:** Neglect all the negative influences (gradients) and focus only on the positive influences (gradients)
- This gives a better picture of the true influence of the input

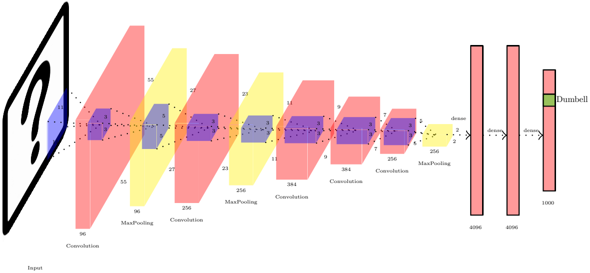
Module 13.6: Optimization over images



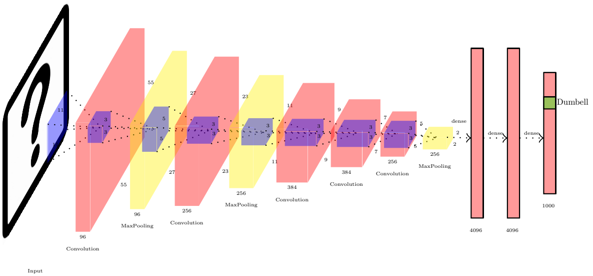
- Suppose we want to create an image which looks like a dumbell (or an ostrich, or a car, or just anything)



- Suppose we want to create an image which looks like a dumbell (or an ostrich, or a car, or just anything)
- In other words we want to create an image such that if we pass it through a trained ConvNet it should maximize the probability of the class dumbell



- Suppose we want to create an image which looks like a dumbell (or an ostrich, or a car, or just anything)
- In other words we want to create an image such that if we pass it through a trained ConvNet it should maximize the probability of the class dumbell
- We could pose this as an optimization problem w.r.t I $(i_0, i_1, \ldots, i_{mn})$

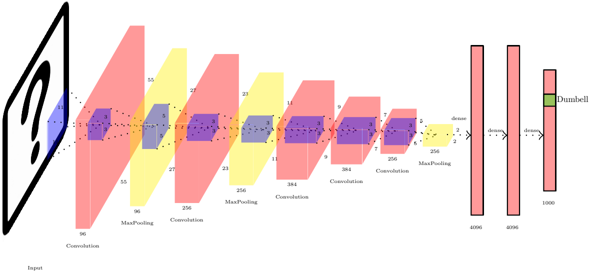


- Suppose we want to create an image which looks like a dumbell (or an ostrich, or a car, or just anything)
- In other words we want to create an image such that if we pass it through a trained ConvNet it should maximize the probability of the class dumbell
- We could pose this as an optimization problem w.r.t I $(i_0, i_1, \ldots, i_{mn})$

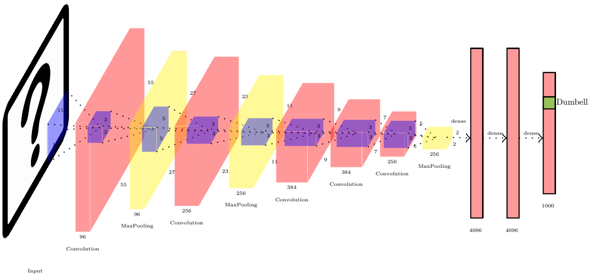
$$\arg\max_I (S_c(I) - \lambda\Omega(I))$$

$$S_c(I) = \text{Score for class C before softmax}$$

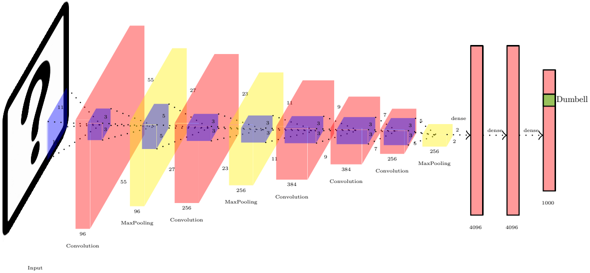
$$\Omega(I) = \text{Some regularizer to ensure that } I \text{ looks like an image}$$



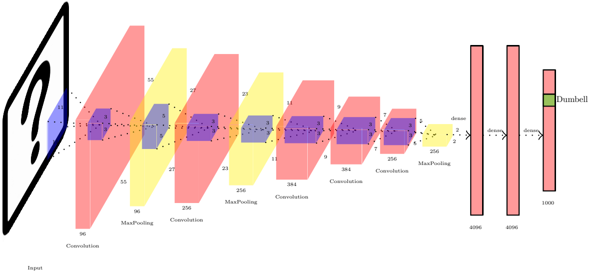
- We can essentially think of the image as a collection of parameters



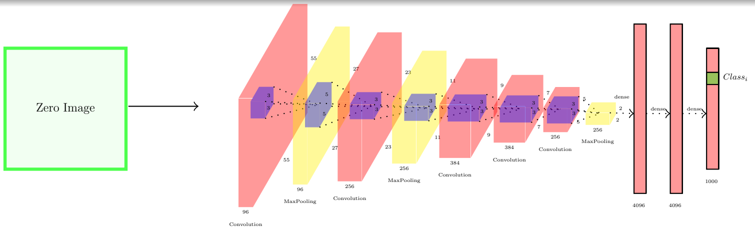
- We can essentially think of the image as a collection of parameters
- Keep the weights of trained convolutional neural network fixed



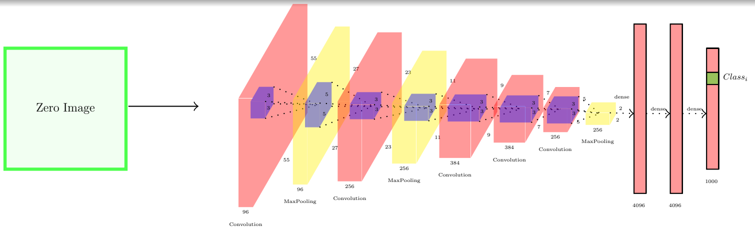
- We can essentially think of the image as a collection of parameters
- Keep the weights of trained convolutional neural network fixed
- Now adjust these parameters(image pixels) so that the score of a class is maximized



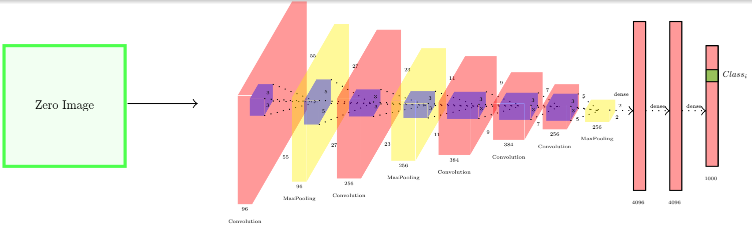
- We can essentially think of the image as a collection of parameters
- Keep the weights of trained convolutional neural network fixed
- Now adjust these parameters(image pixels) so that the score of a class is maximized
- Let us see how



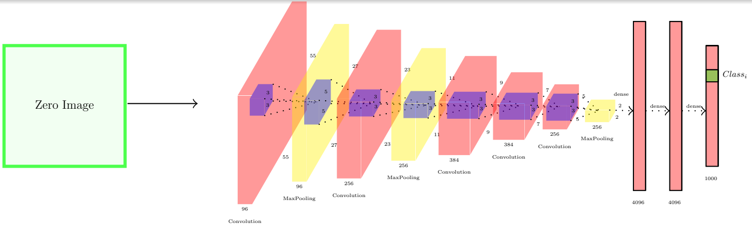
1. Start with a zero image



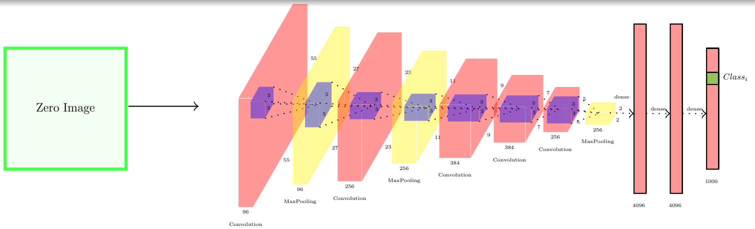
1. Start with a zero image
2. Set the score vector to be $[0, 0, \ldots 1, 0, 0]$



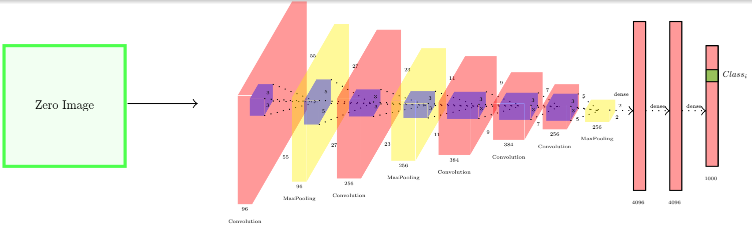
1. Start with a zero image
2. Set the score vector to be $[0, 0, \ldots 1, 0, 0]$
3. Compute the gradient $\frac{\partial S_c(I)}{\partial i_k}$



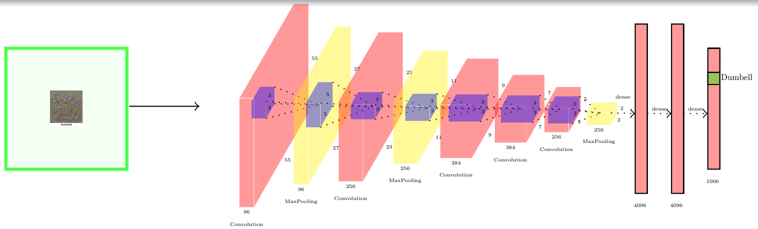
1. Start with a zero image
2. Set the score vector to be $[0, 0, \ldots 1, 0, 0]$
3. Compute the gradient $\frac{\partial S_c(I)}{\partial i_k}$
4. Now update the pixel $i_k = i_k - \eta \frac{\partial S_c(I)}{\partial i_k}$



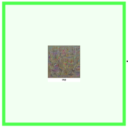
1. Start with a zero image
2. Set the score vector to be $[0, 0, \ldots 1, 0, 0]$
3. Compute the gradient $\frac{\partial S_c(I)}{\partial i_k}$
4. Now update the pixel $i_k = i_k - \eta \frac{\partial S_c(I)}{\partial i_k}$
5. Now again do a forward pass through the network

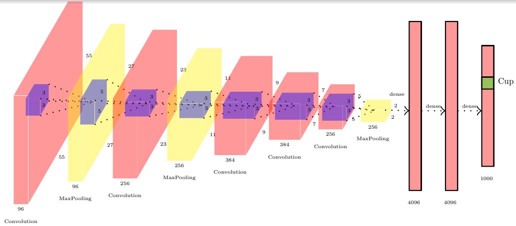


1. Start with a zero image
2. Set the score vector to be $[0, 0, \ldots 1, 0, 0]$
3. Compute the gradient $\frac{\partial S_c(I)}{\partial i_k}$
4. Now update the pixel $i_k = i_k - \eta \frac{\partial S_c(I)}{\partial i_k}$
5. Now again do a forward pass through the network
6. Go to step 2

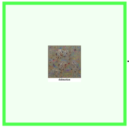


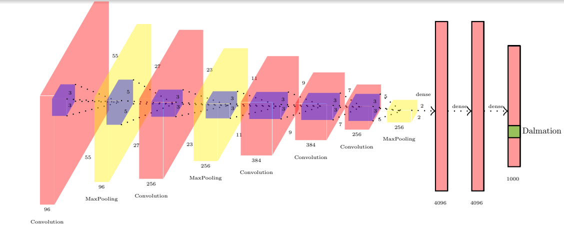
- Lets look at the images obtained for maximizing some class scores



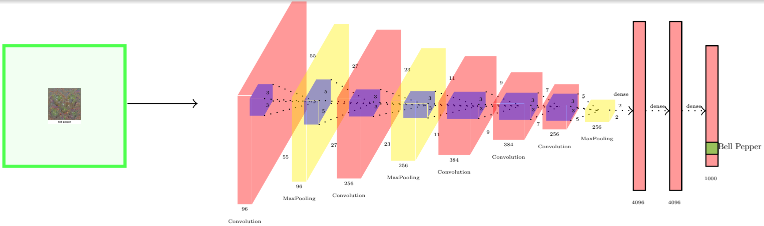


- Lets look at the images obtained for maximizing some class scores

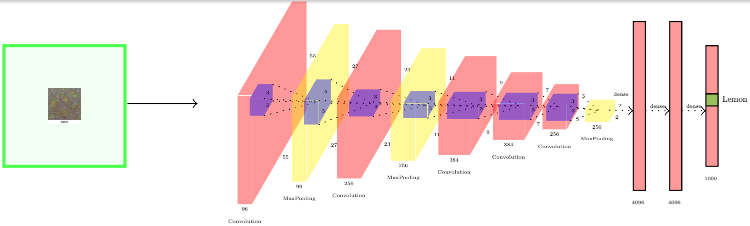




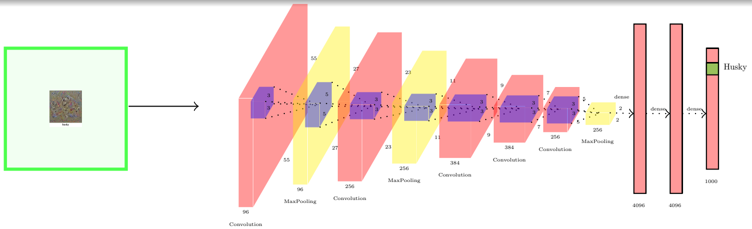
- Lets look at the images obtained for maximizing some class scores



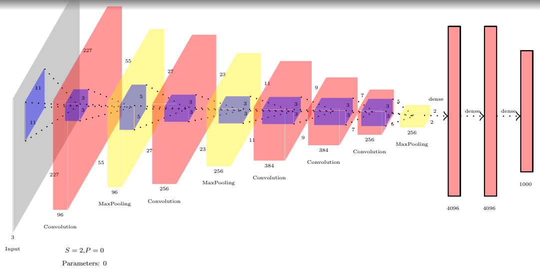
- Lets look at the images obtained for maximizing some class scores



- Lets look at the images obtained for maximizing some class scores



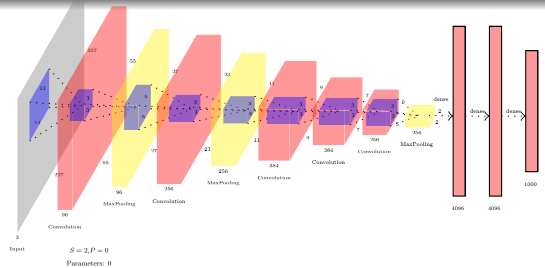
- Lets look at the images obtained for maximizing some class scores



Repeat:

- Feed an image through the network

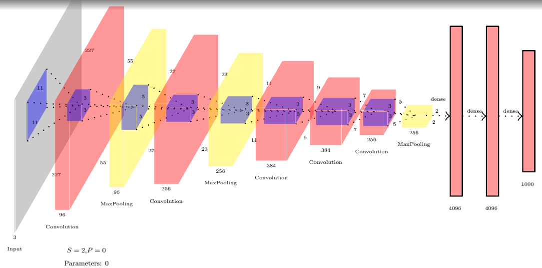
- We can actually do this for any arbitrary neuron in the convnet



- We can actually do this for any arbitrary neuron in the convnet

Repeat:

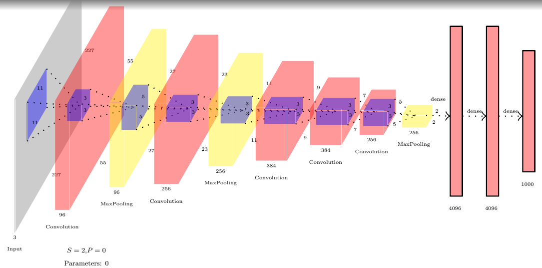
- Feed an image through the network
- Set activation in layer of interest to all zero, except for a neuron of interest



- We can actually do this for any arbitrary neuron in the convnet

Repeat:

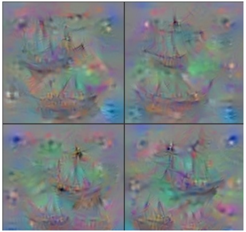
- Feed an image through the network
- Set activation in layer of interest to all zero, except for a neuron of interest
- Backprop to image



- We can actually do this for any arbitrary neuron in the convnet

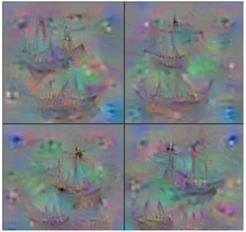
Repeat:

- Feed an image through the network
- Set activation in layer of interest to all zero, except for a neuron of interest
- Backprop to image
- $i_k = i_k - \eta \frac{\partial A(I)}{\partial i_k}$, A(I) is the activation of the i^{th} neuron in some layer



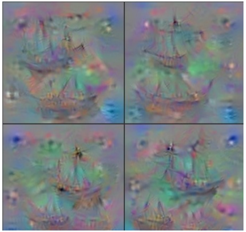
Layer-8

- Let us look at some "updated" images which excite certain neurons in some layer



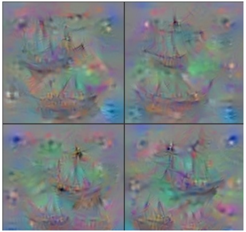
Layer-8

- Let us look at some “updated” images which excite certain neurons in some layer
- Starting with different initializations instead of using a zero image we can get different insights



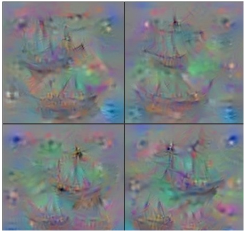
Layer-8

- Let us look at some "updated" images which excite certain neurons in some layer
- Starting with different initializations instead of using a zero image we can get different insights
- Each of these 4 images are obtained by focusing on one neuron in layer 8 and starting with different initializations

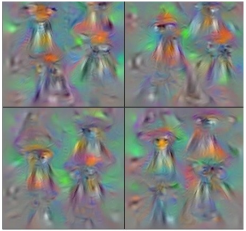


Layer-8

- Let us look at some "updated" images which excite certain neurons in some layer
- Starting with different initializations instead of using a zero image we can get different insights
- Each of these 4 images are obtained by focusing on one neuron in layer 8 and starting with different initializations
- We can do a similar analysis with other layers



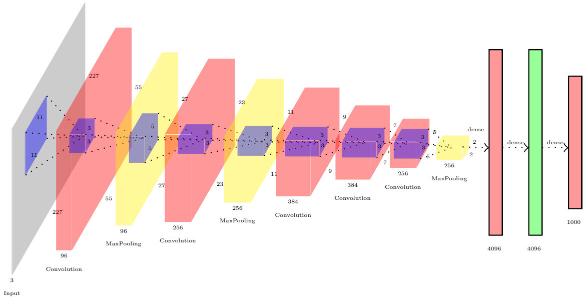
Layer-8



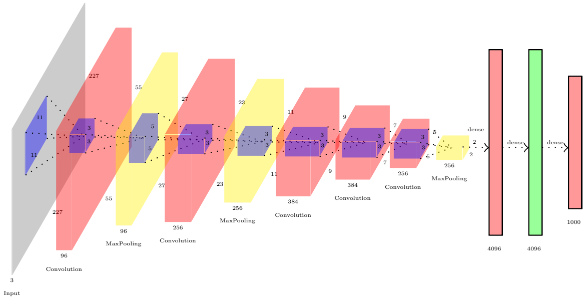
Layer-7

- Let us look at some "updated" images which excite certain neurons in some layer
- Starting with different initializations instead of using a zero image we can get different insights
- Each of these 4 images are obtained by focusing on one neuron in layer 8 and starting with different initializations
- We can do a similar analysis with other layers

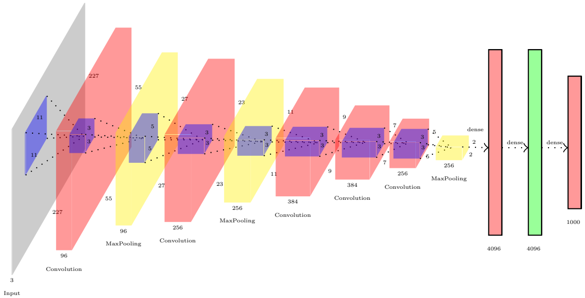
Module 13.7: Creating images from embeddings



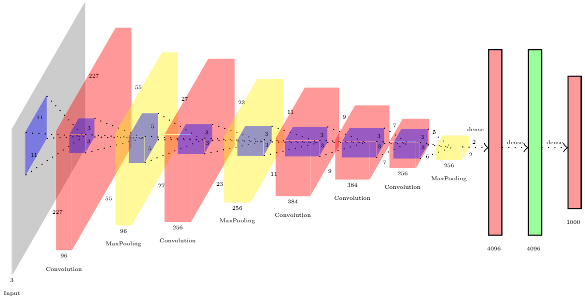
- We could think of the fc7 layer as some kind of an embedding for the image



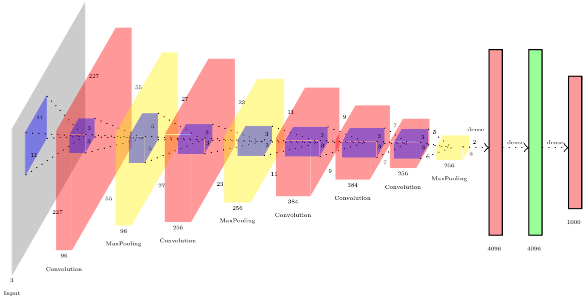
- We could think of the fc7 layer as some kind of an embedding for the image
- **Question:** Given this embedding can we reconstruct the image?



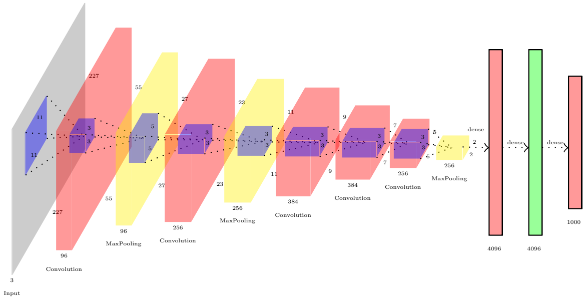
- We could think of the fc7 layer as some kind of an embedding for the image
- **Question:** Given this embedding can we reconstruct the image?
- We can pose this as an optimization problem



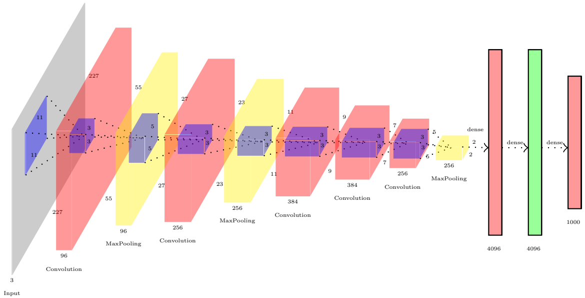
- Find an image such that



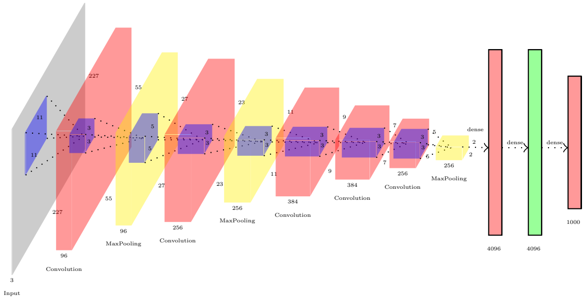
- Find an image such that
- Its embedding is similar to a given embedding



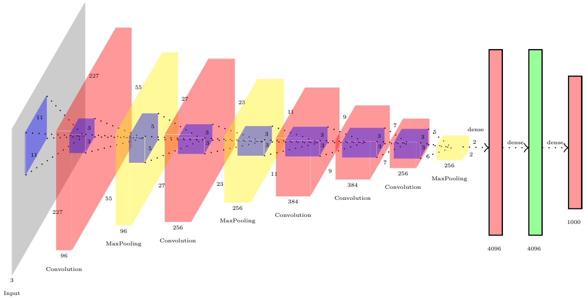
- Find an image such that
- Its embedding is similar to a given embedding
- It looks natural (some prior regularization)



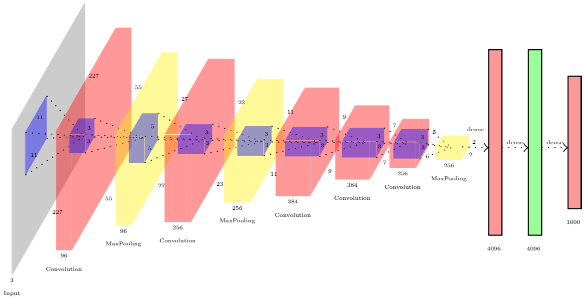
- ϕ_0 :Embedding of an image of interest



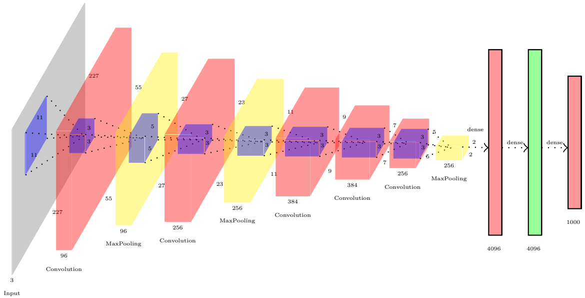
- ϕ_0 :Embedding of an image of interest
- X :Random image (say zero image)



- ϕ_0 :Embedding of an image of interest
- X :Random image (say zero image)
- Repeat

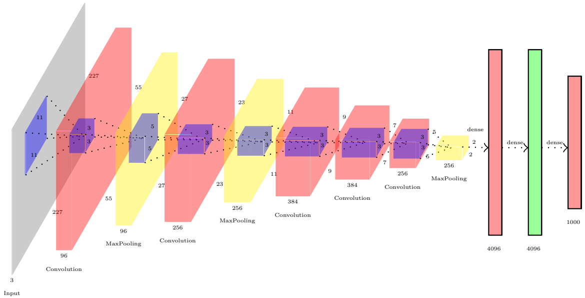


- ϕ_0 :Embedding of an image of interest
- X :Random image (say zero image)
- Repeat
  - Forward pass using X and compute $\phi(x)$.



- ϕ_0 :Embedding of an image of interest
- X :Random image (say zero image)
- Repeat
  - Forward pass using X and compute $\phi(x)$.
  - Compute

$$\mathscr{L}(i) = ||\phi(x) - \phi_0||^2 + \lambda||\phi(x)||_6^6$$



- ϕ_0 :Embedding of an image of interest
- X :Random image (say zero image)
- Repeat
  - Forward pass using X and compute $\phi(x)$.
  - Compute

$$\mathscr{L}(i) = ||\phi(x) - \phi_0||^2 + \lambda||\phi(x)||_6^6$$

  - $i_k = i_k - \eta \frac{\mathscr{L}(i)}{\partial i_k}$

Original Image

Conv-1

Original Image

Relu-1

Original Image

Mpool-1

Original Image

Norm-1

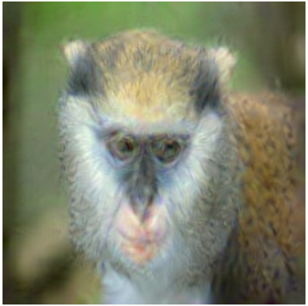
Original Image

Conv-2

Original Image

Relu-2

Original Image



Mpool-2

Original Image

Norm-2

Original Image

Conv-3

Original Image

Relu-3

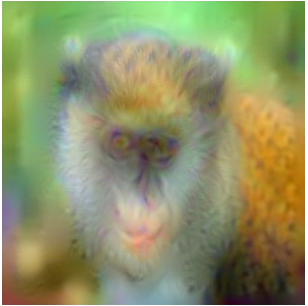
Original Image

Conv-4

Original Image

Relu-4

Original Image



Conv-5

Original Image

Relu-5

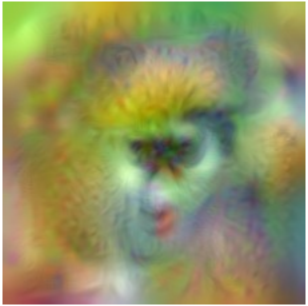
Original Image

Mpool-5

Original Image

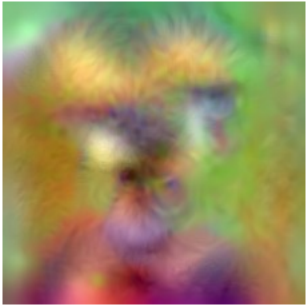
FC-6

Original Image



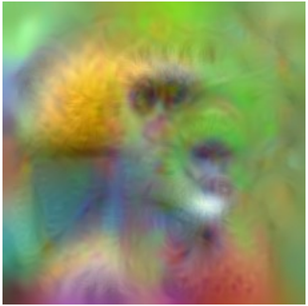
Relu-6

Original Image



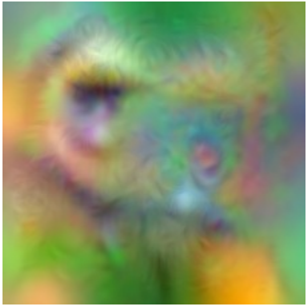
FC-7

Original Image



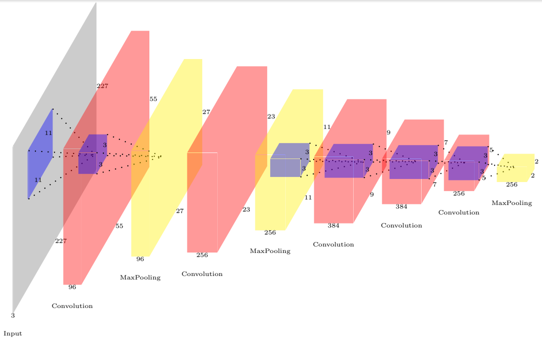
Relu-7

Original Image

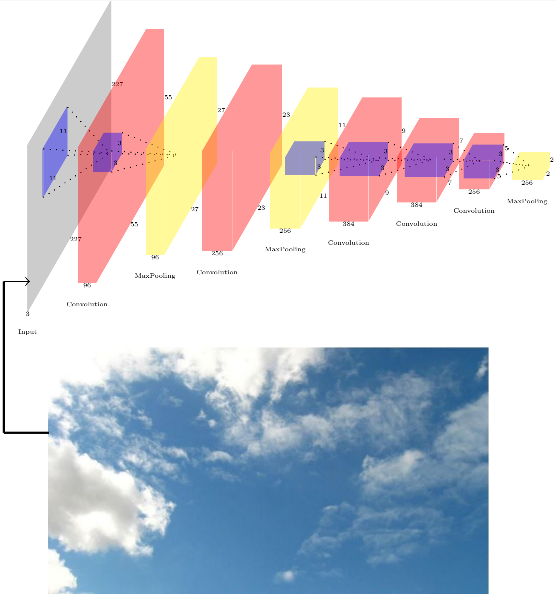


FC-8

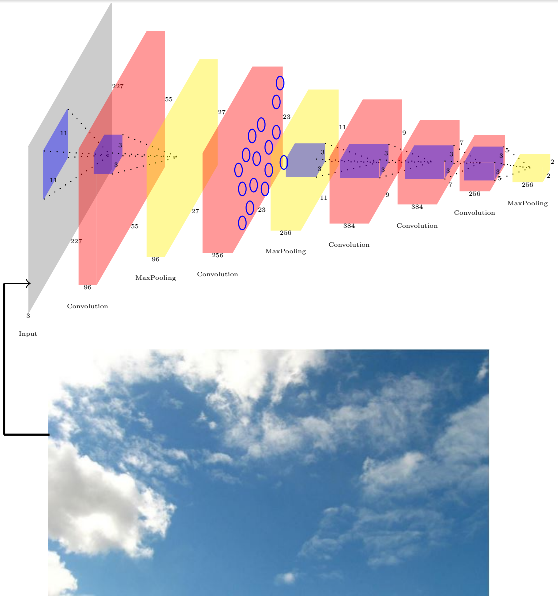
Module 13.8: Deep Dream



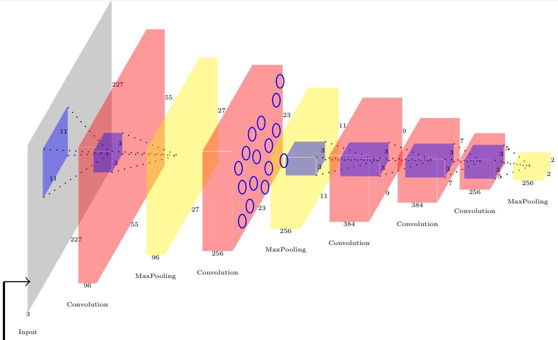
- Suppose instead of starting with a blank (zero) image we start with an actual image.



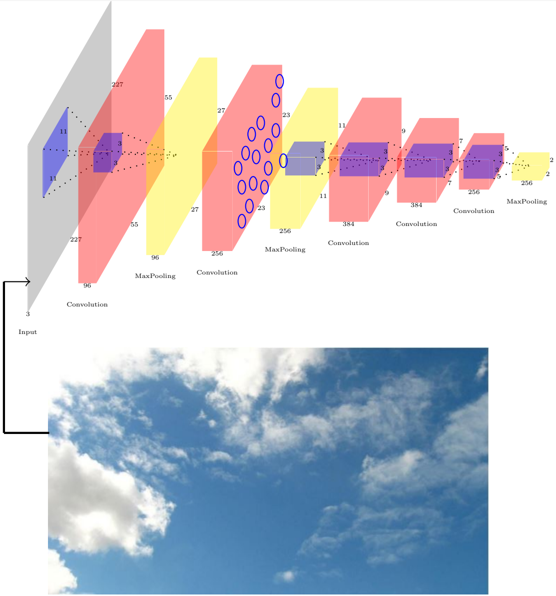
- Suppose instead of starting with a blank (zero) image we start with an actual image.



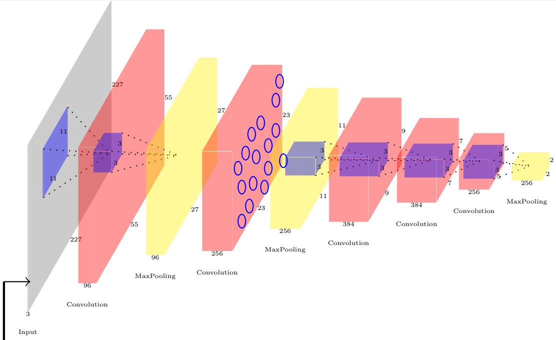
- Suppose instead of starting with a blank (zero) image we start with an actual image.
- We focus on some layer and check the activations of the neurons



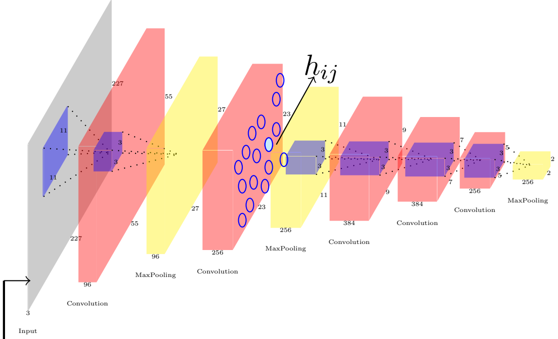
- Suppose instead of starting with a blank (zero) image we start with an actual image.
- We focus on some layer and check the activations of the neurons
- We want to change the image so that these neurons fire even more



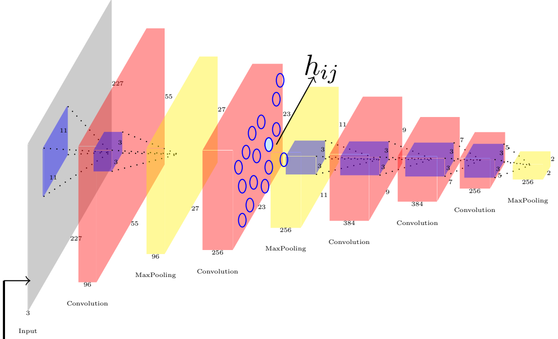
- How would we achieve this?



- How would we achieve this?
- Suppose we want to boost the activation h_{ij} (some neuron in some layer)



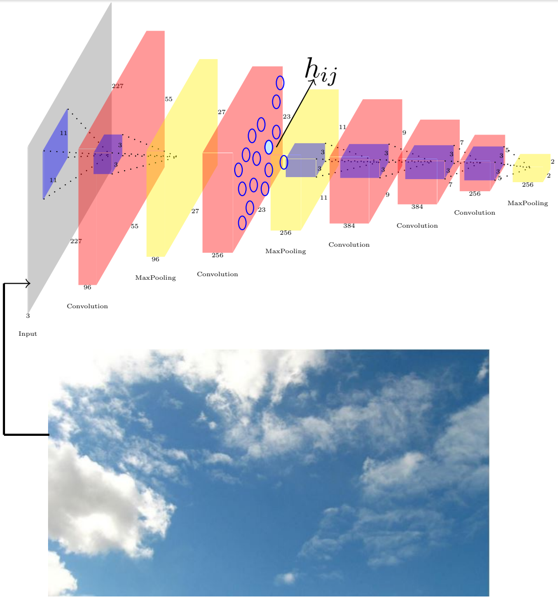
- How would we achieve this?
- Suppose we want to boost the activation h_{ij} (some neuron in some layer)
- We can formulate this as the following optimization problem



- How would we achieve this?
- Suppose we want to boost the activation h_{ij} (some neuron in some layer)
- We can formulate this as the following optimization problem

$$\max_{I} \mathscr{L}(I)$$

$$\mathscr{L}(I) = h_{ij}^2$$



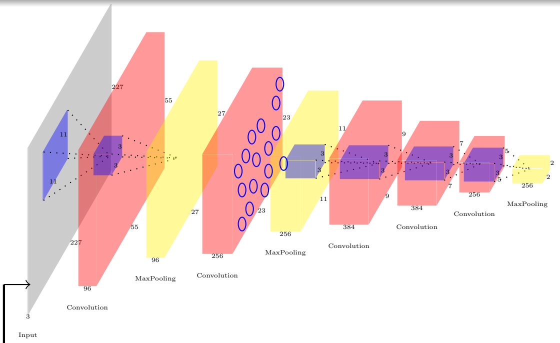
- How would we achieve this?
- Suppose we want to boost the activation h_{ij} (some neuron in some layer)
- We can formulate this as the following optimization problem

$$\max_{I} \mathscr{L}(I)$$

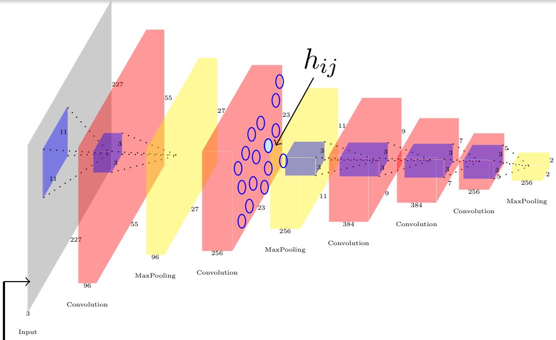
$$\mathscr{L}(I) = h_{ij}^2$$

- Consider a pixel i_{mn} in the image

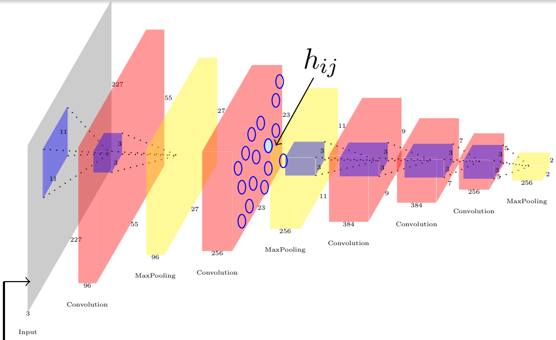
$$\frac{\partial \mathscr{L}(I)}{\partial i_{mn}} = \frac{\partial \mathscr{L}(I)}{\partial h_{ij}} \frac{\partial h_{ij}}{\partial i_{mn}}$$



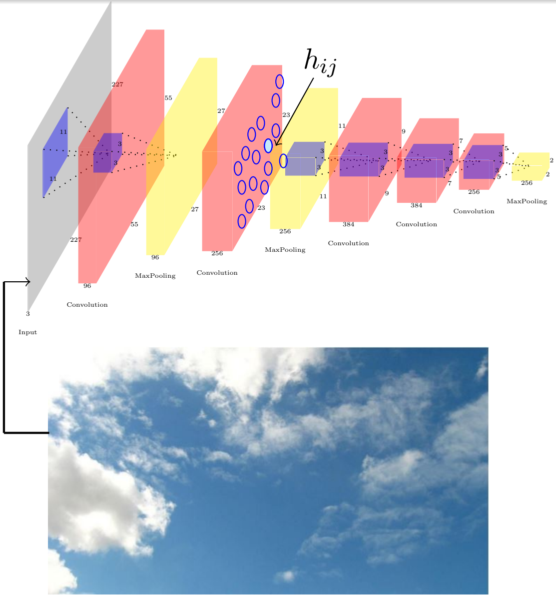
- Once the image is updated $\left(i_{mn} = i_{mn} + \frac{\partial \mathscr{L}(I)}{\partial i_{mn}}\right)$ we feed it back to the network



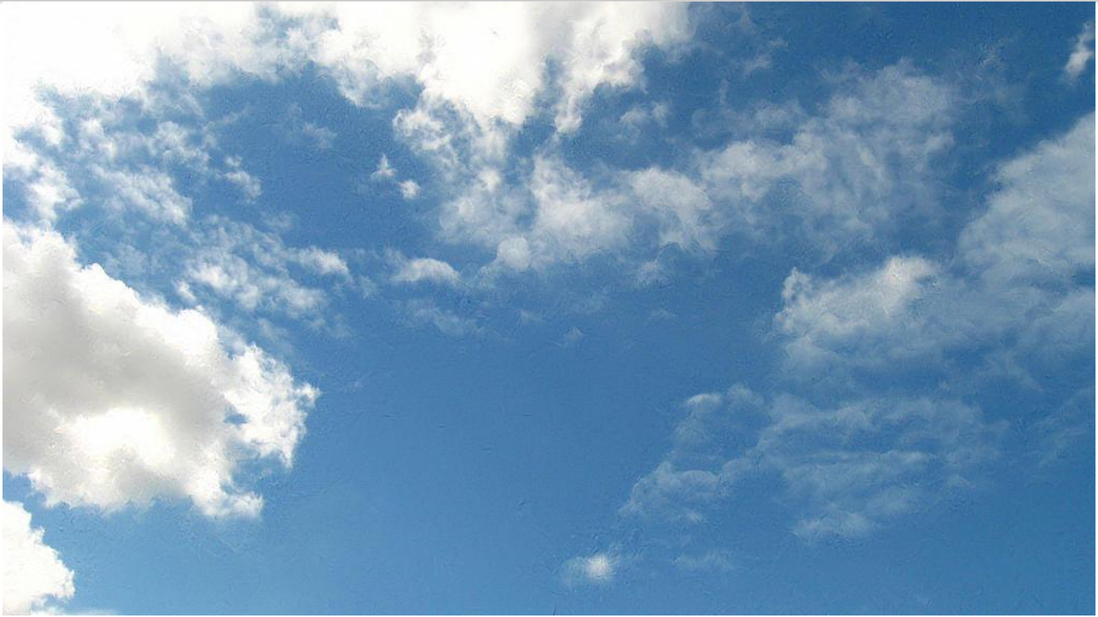
- Once the image is updated $\left(i_{mn} = i_{mn} + \frac{\partial \mathscr{L}(I)}{\partial i_{mn}}\right)$ we feed it back to the network
- This time the target neurons should fire even more (because we have precisely modified the image to achieve this)

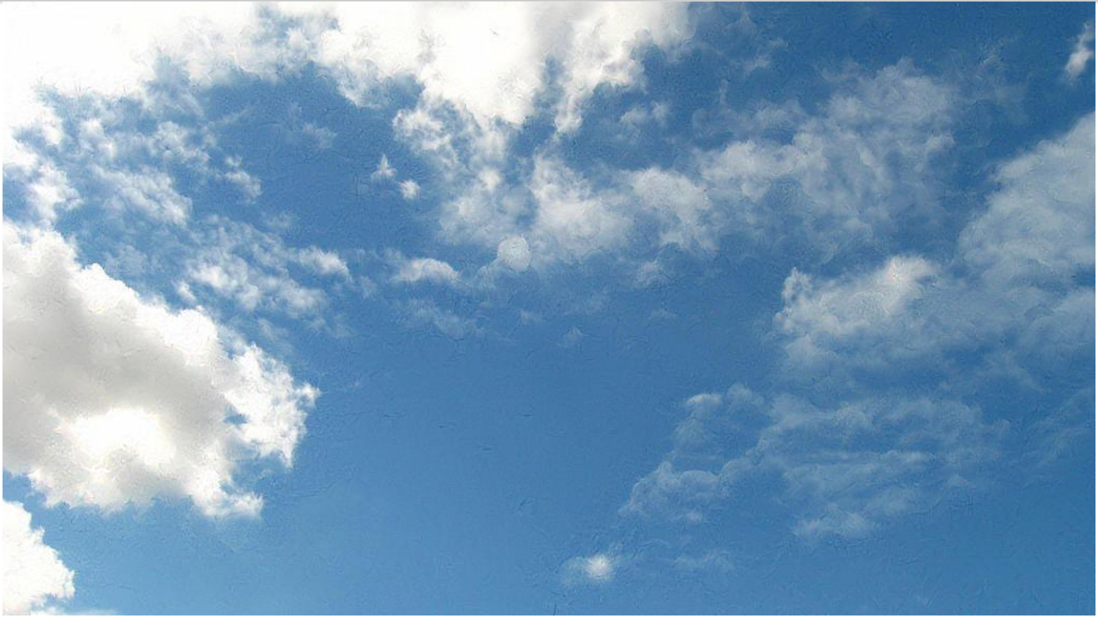


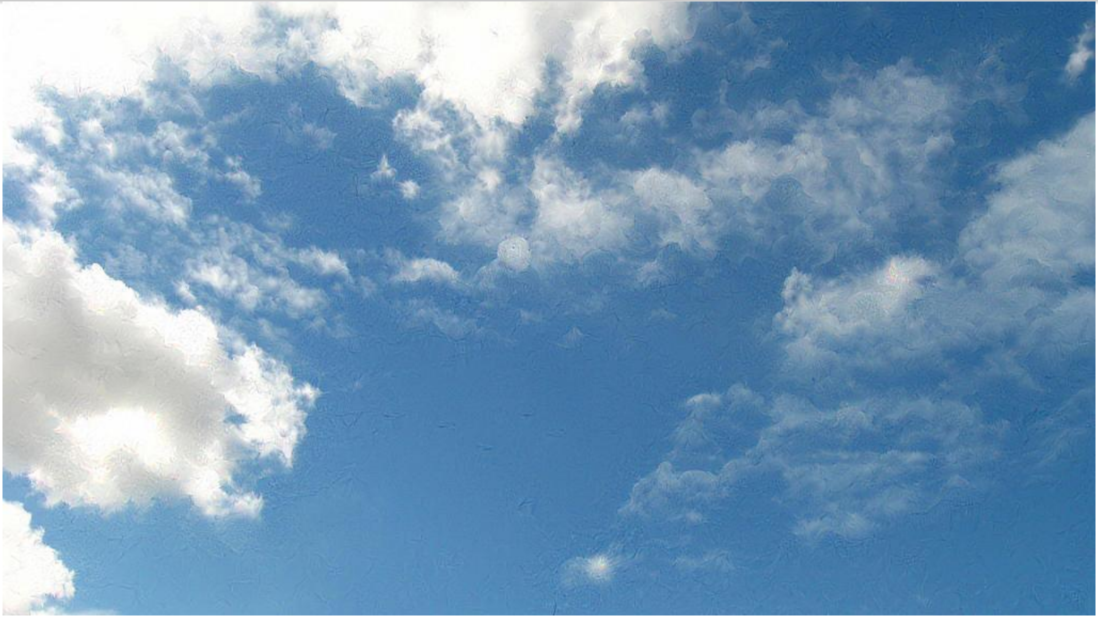
- Once the image is updated $\left(i_{mn} = i_{mn} + \frac{\partial \mathscr{L}(I)}{\partial i_{mn}}\right)$ we feed it back to the network
- This time the target neurons should fire even more (because we have precisely modified the image to achieve this)
- Doing this iteratively would make the image more and more like the patterns that cause the neuron to fire

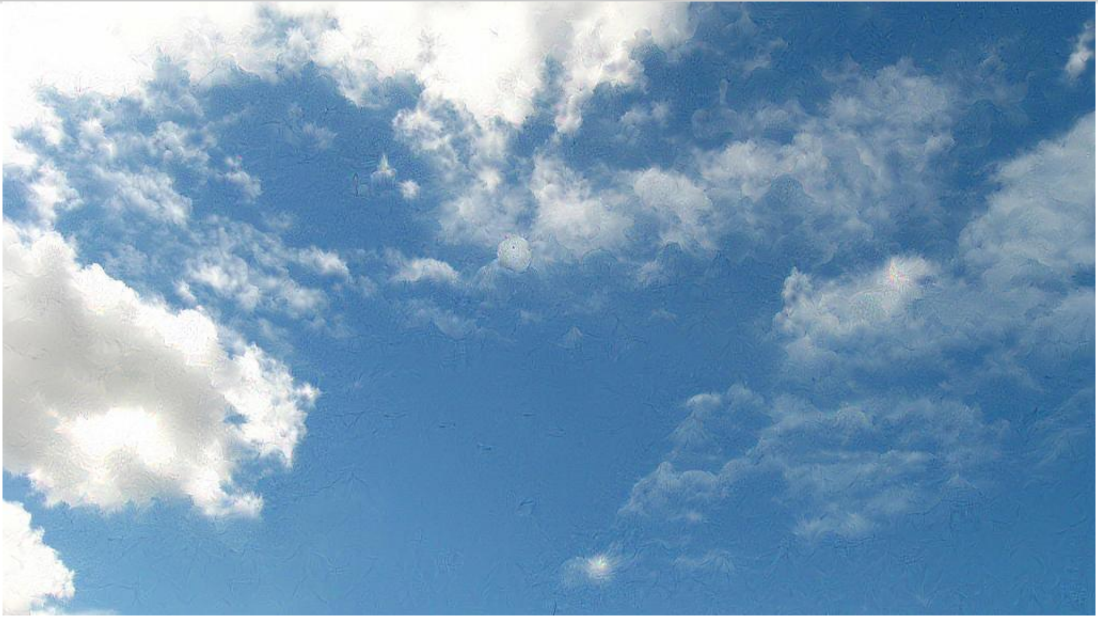


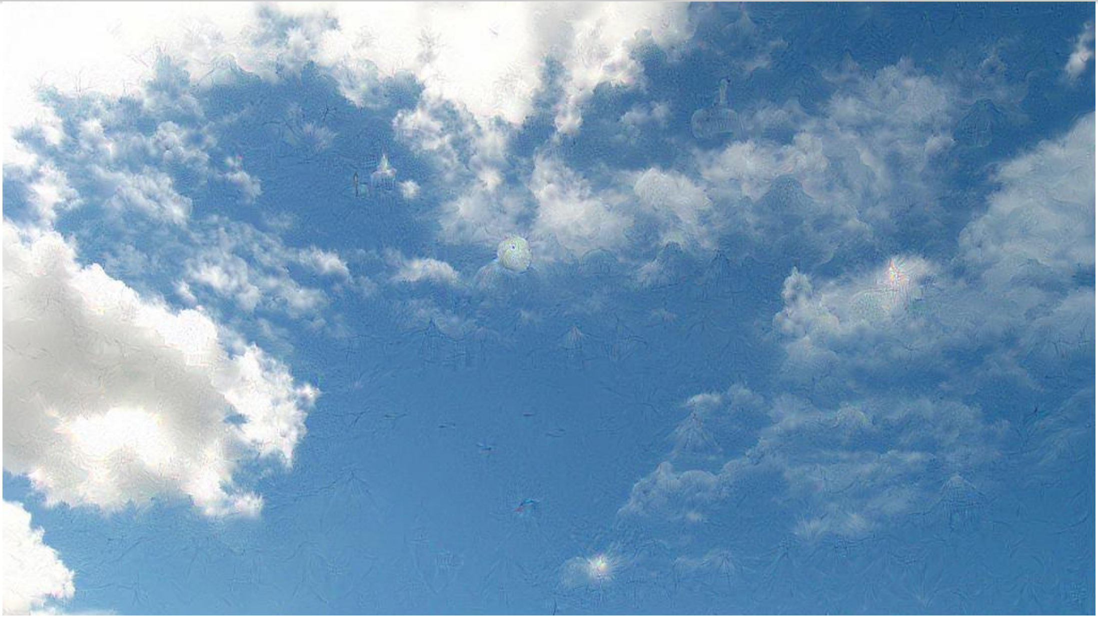
- Once the image is updated $\left(i_{mn} = i_{mn} + \frac{\partial \mathscr{L}(I)}{\partial i_{mn}}\right)$ we feed it back to the network
- This time the target neurons should fire even more (because we have precisely modified the image to achieve this)
- Doing this iteratively would make the image more and more like the patterns that cause the neuron to fire
- Let us run this algorithm

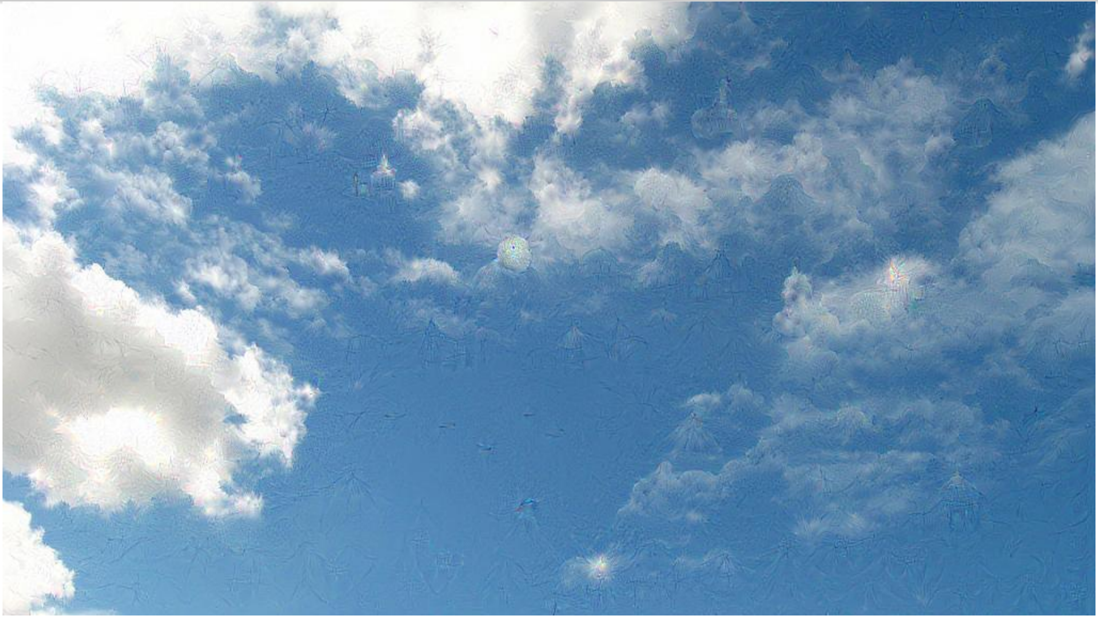


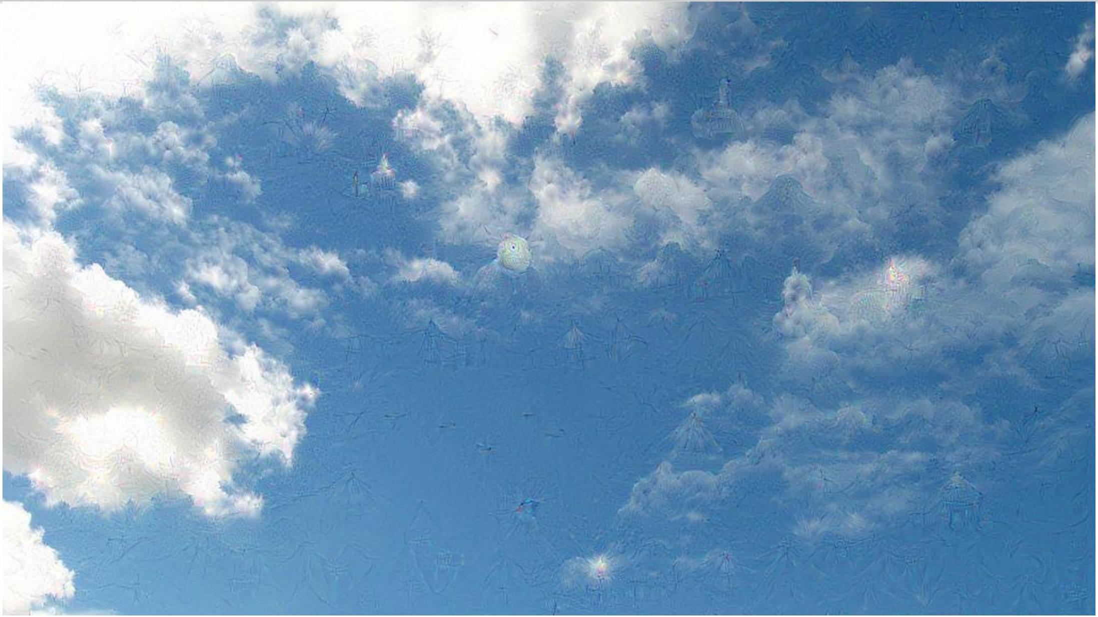


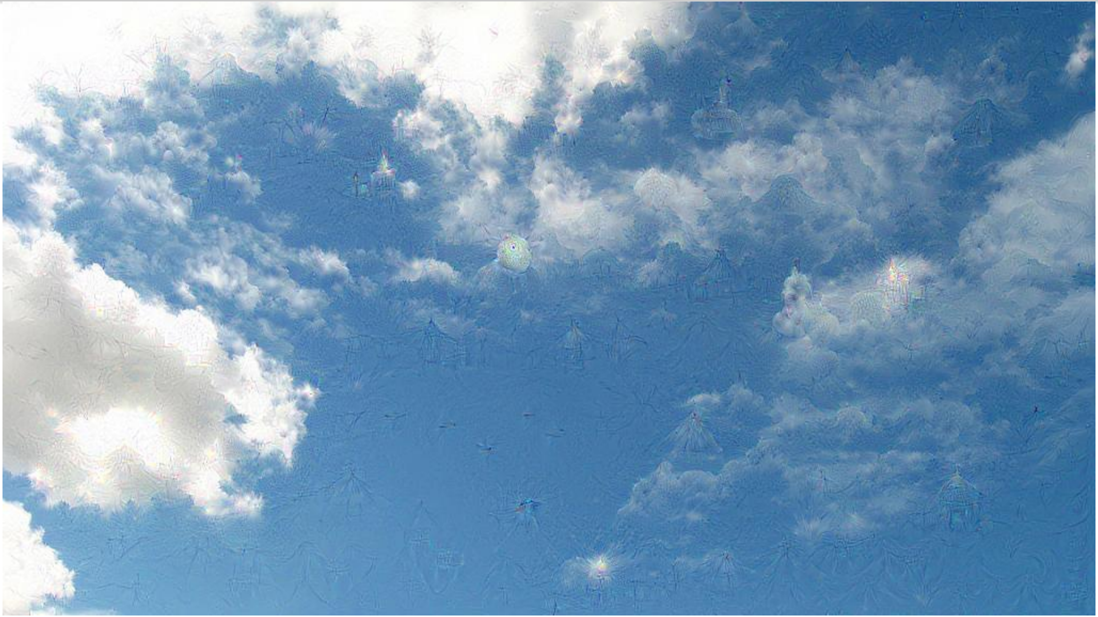


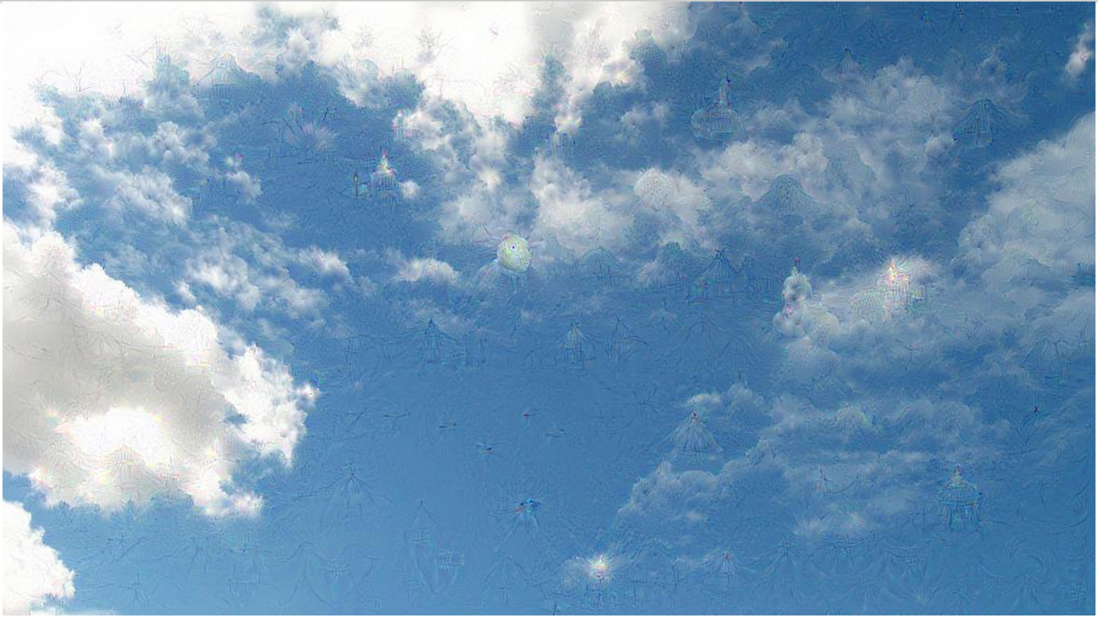


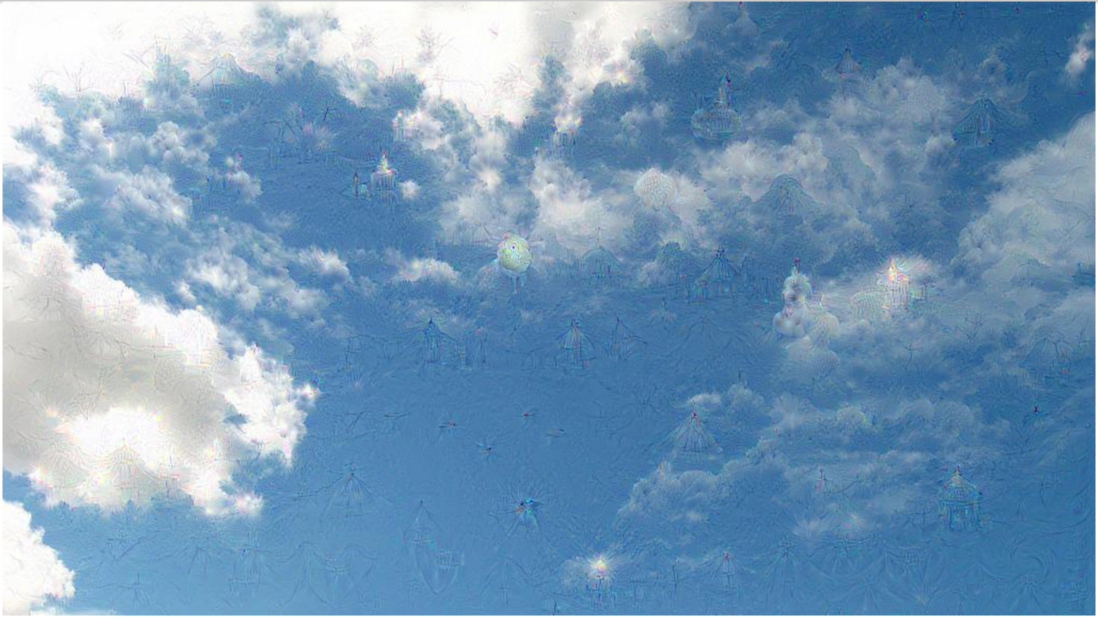


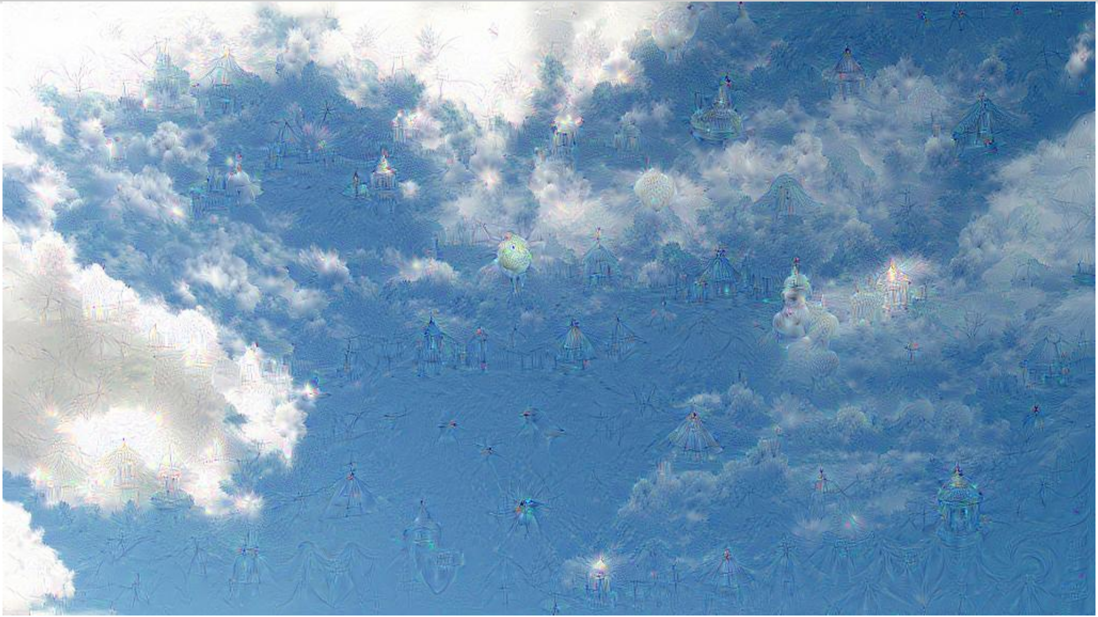


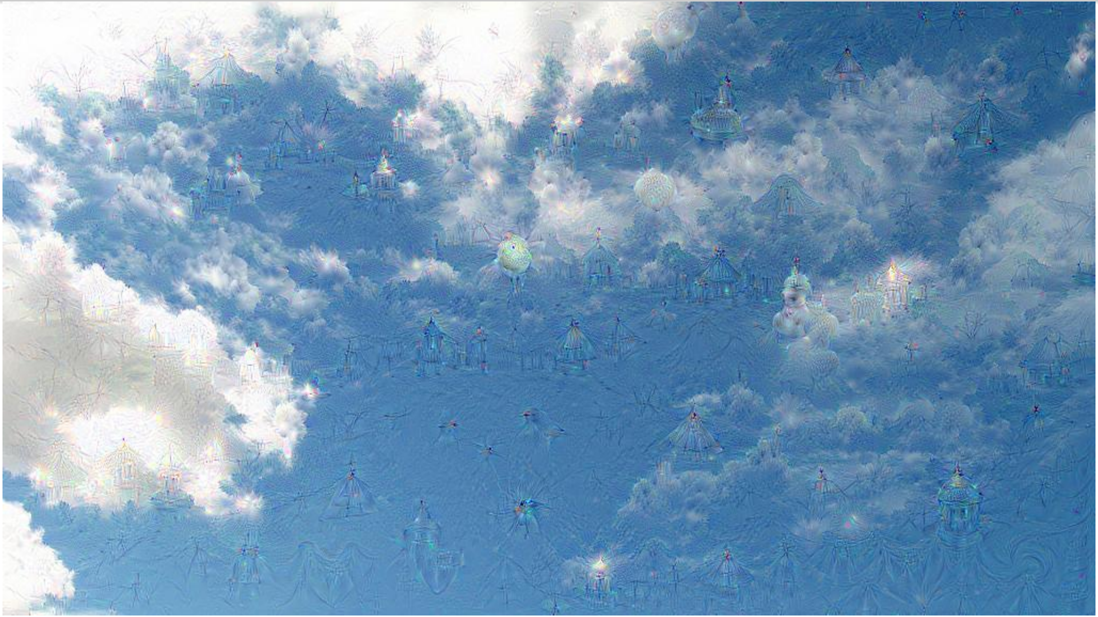


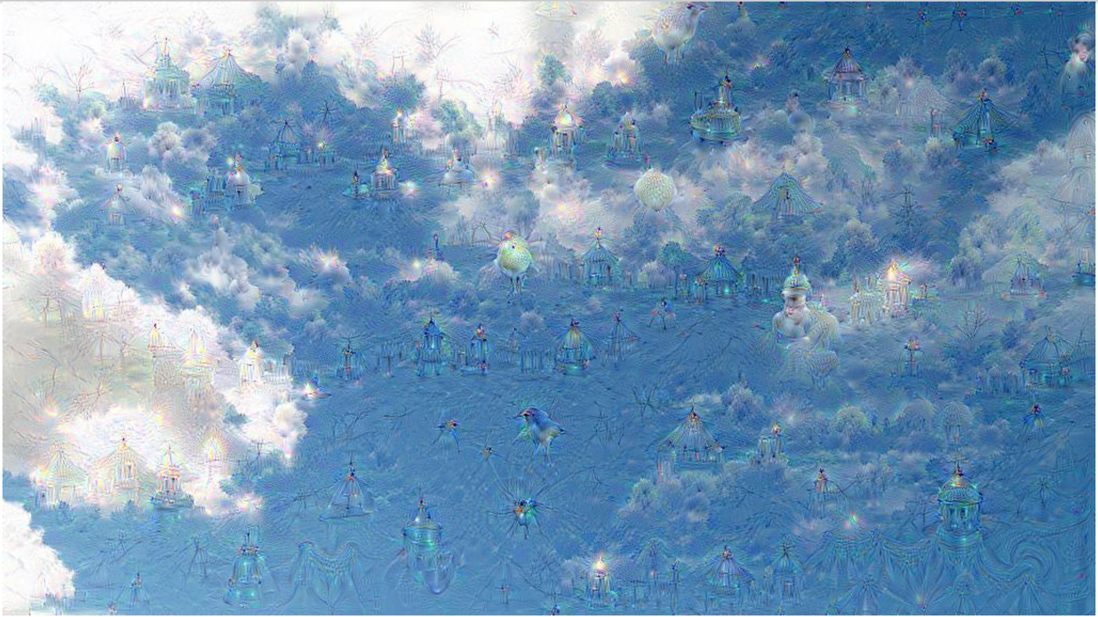


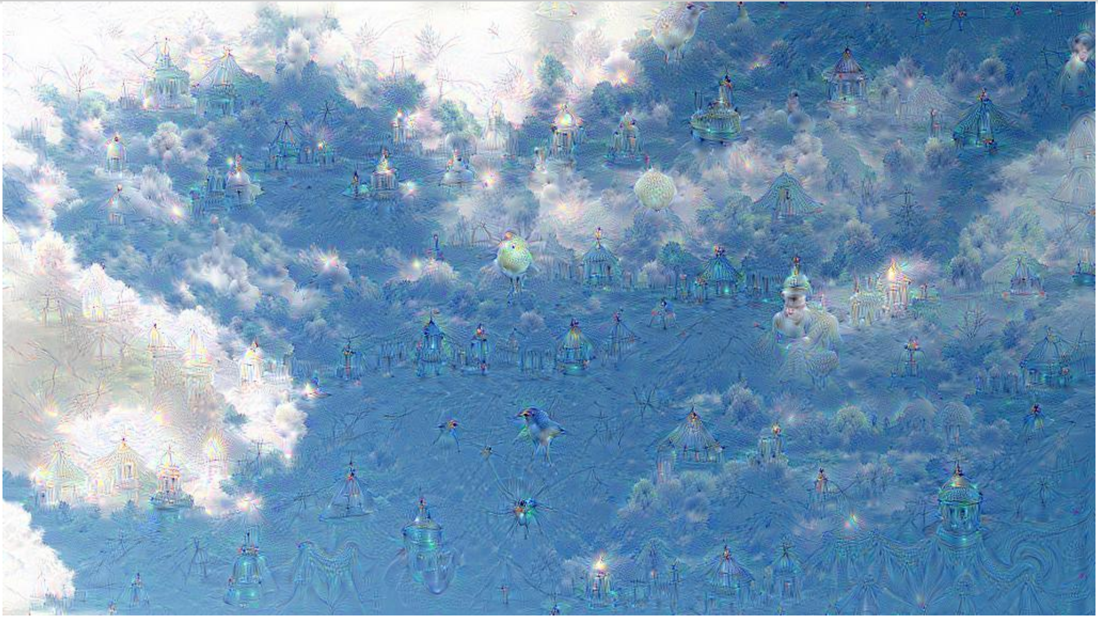


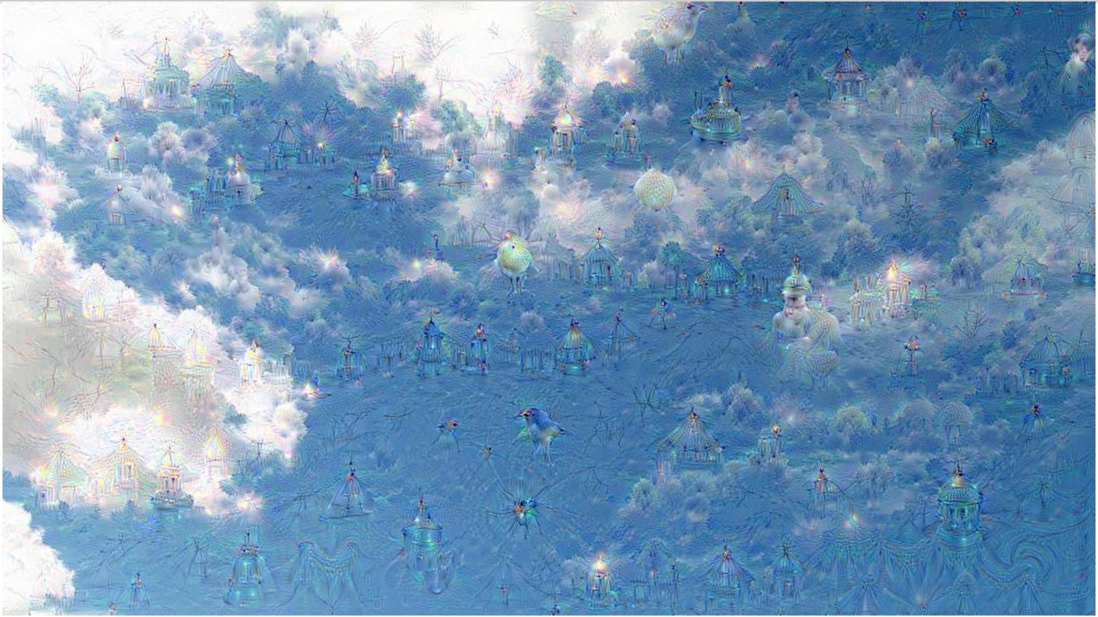


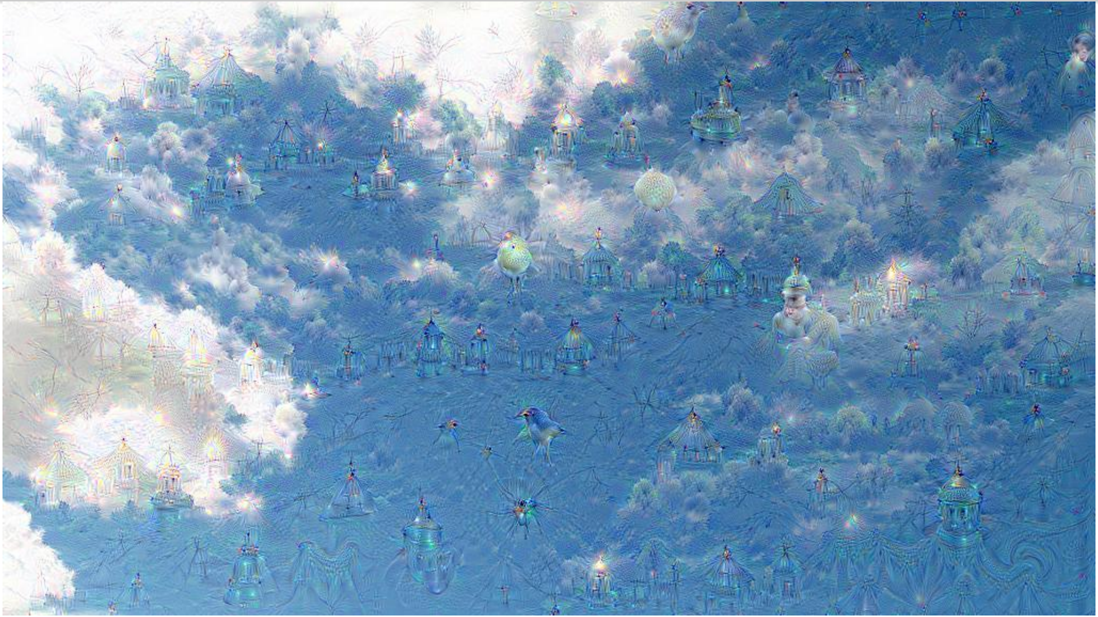


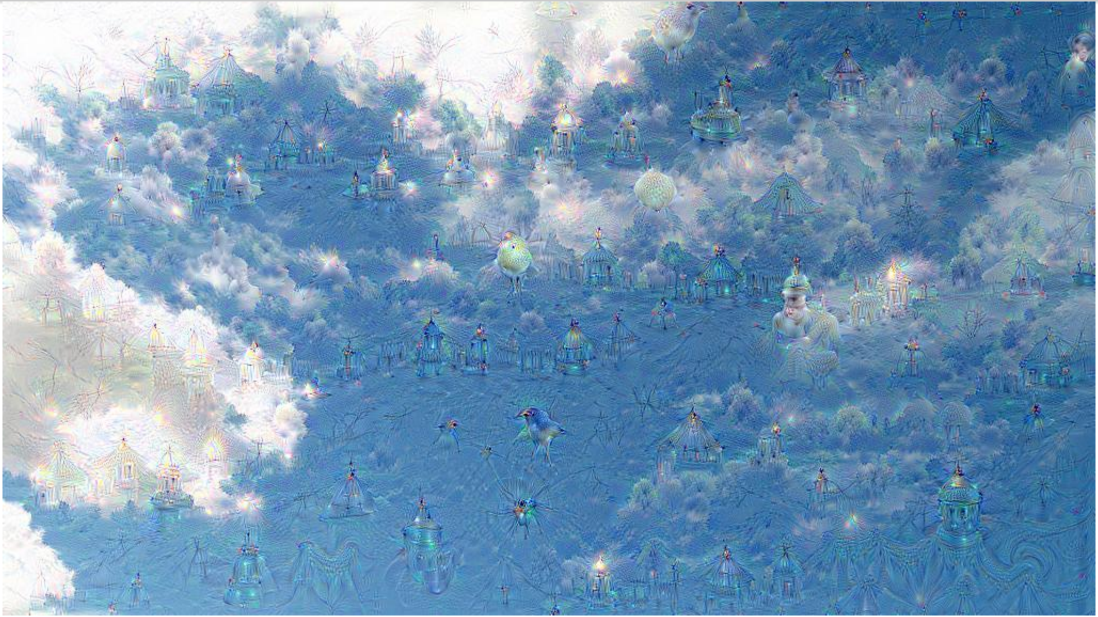


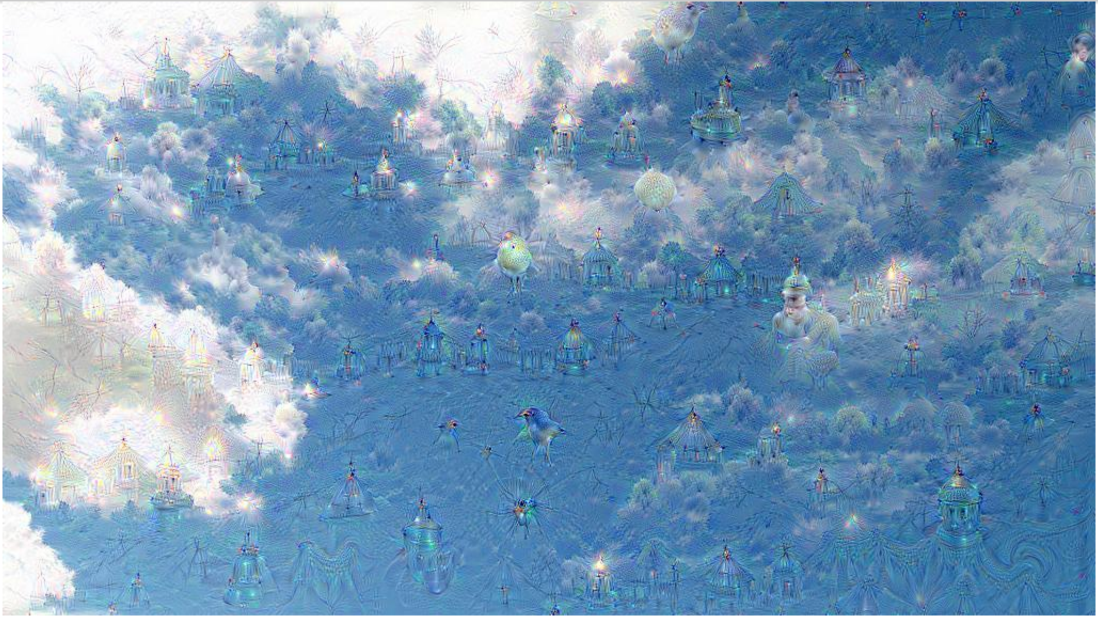


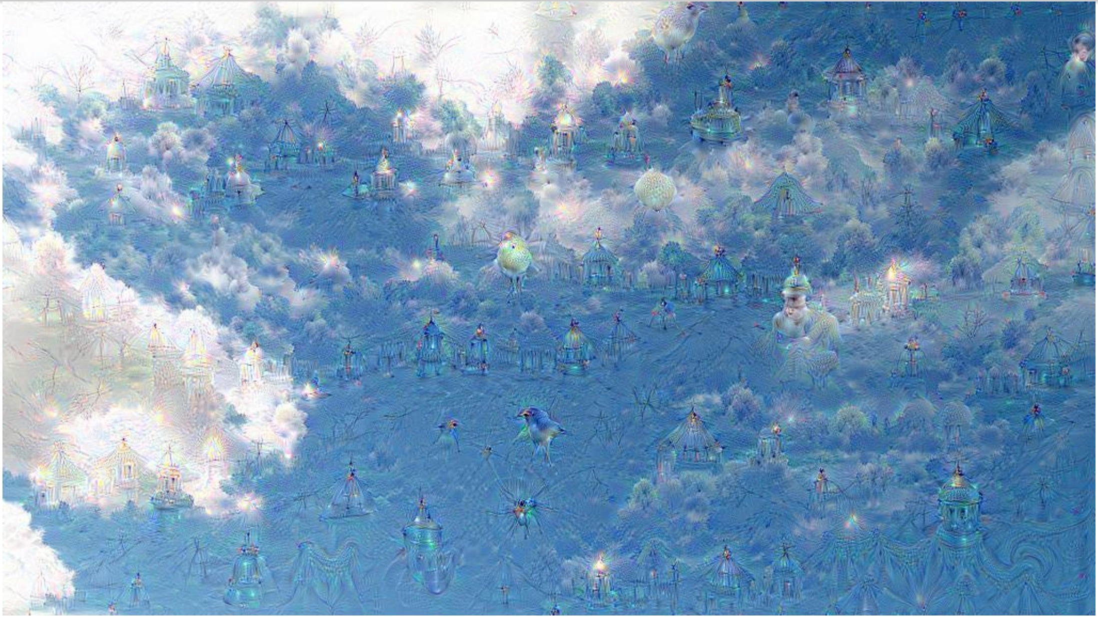


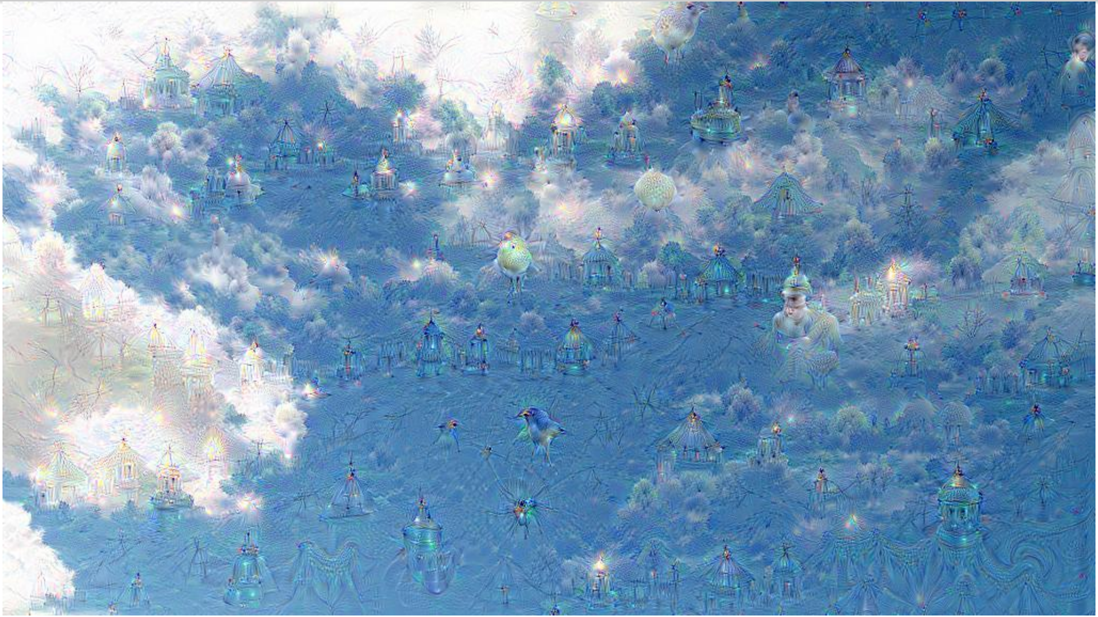


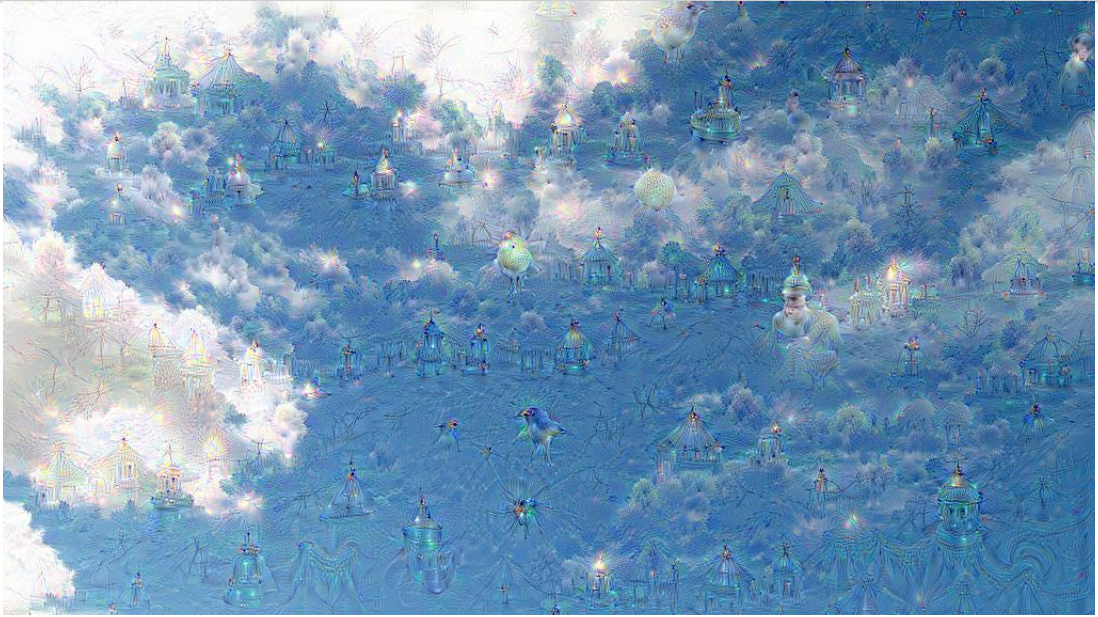


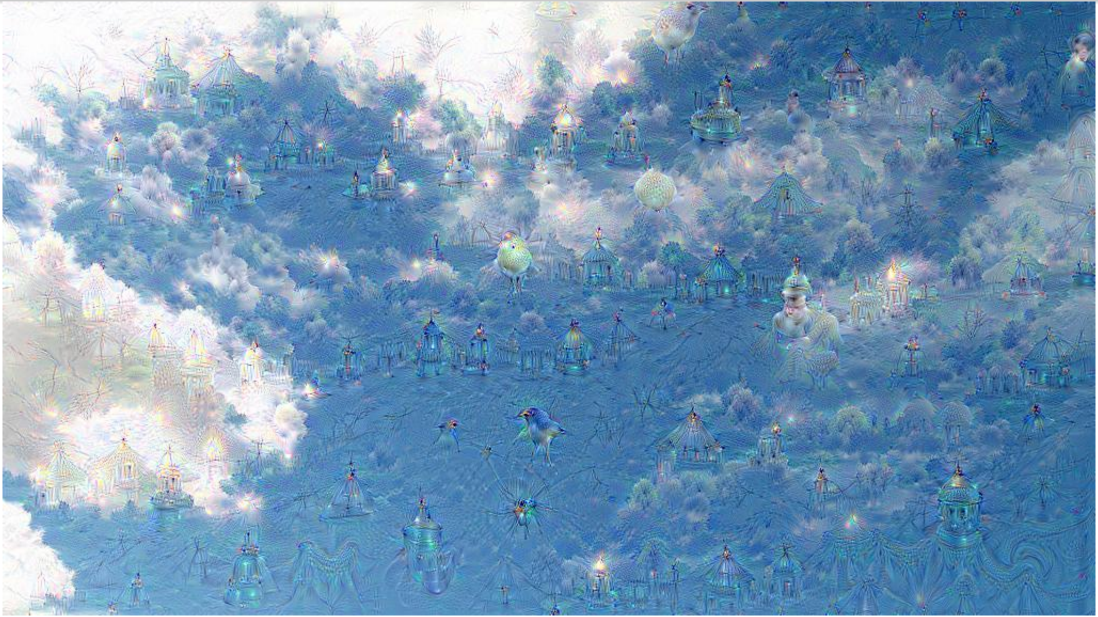


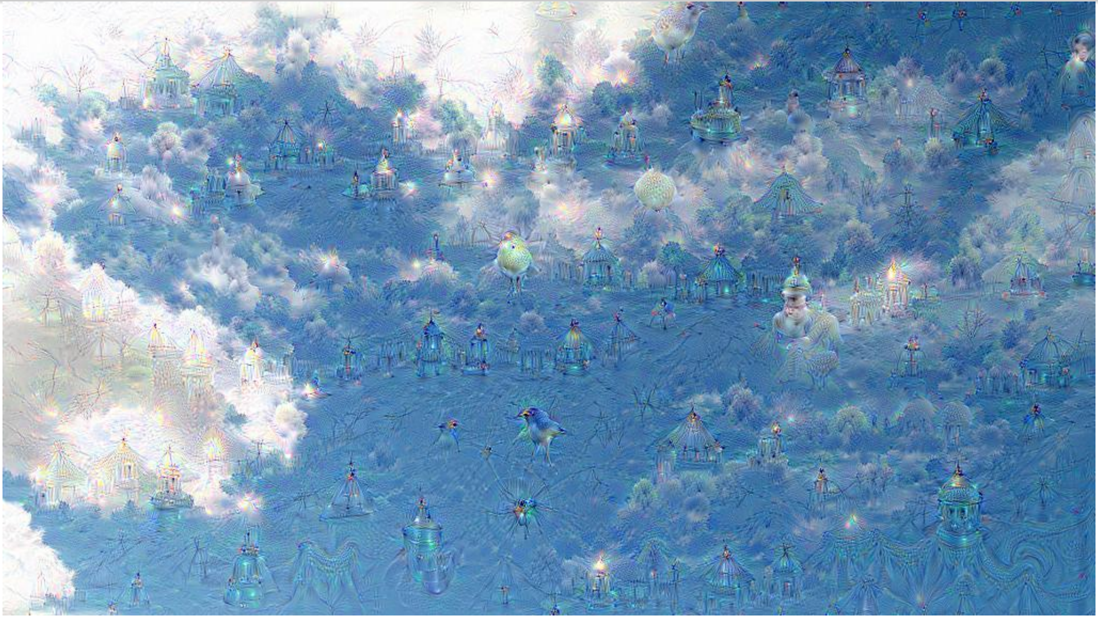


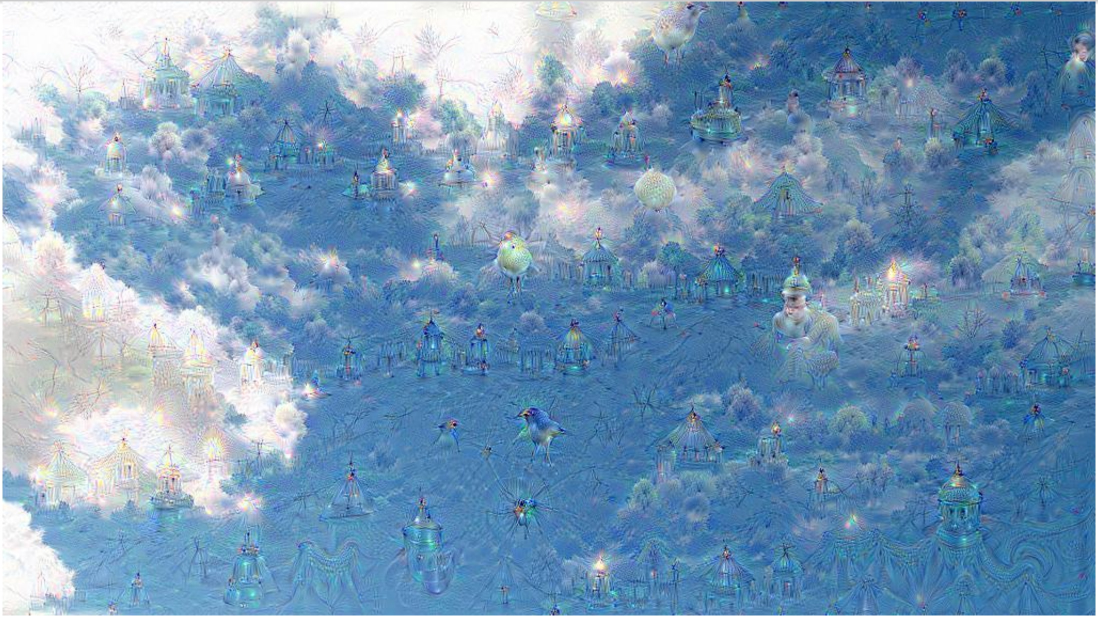


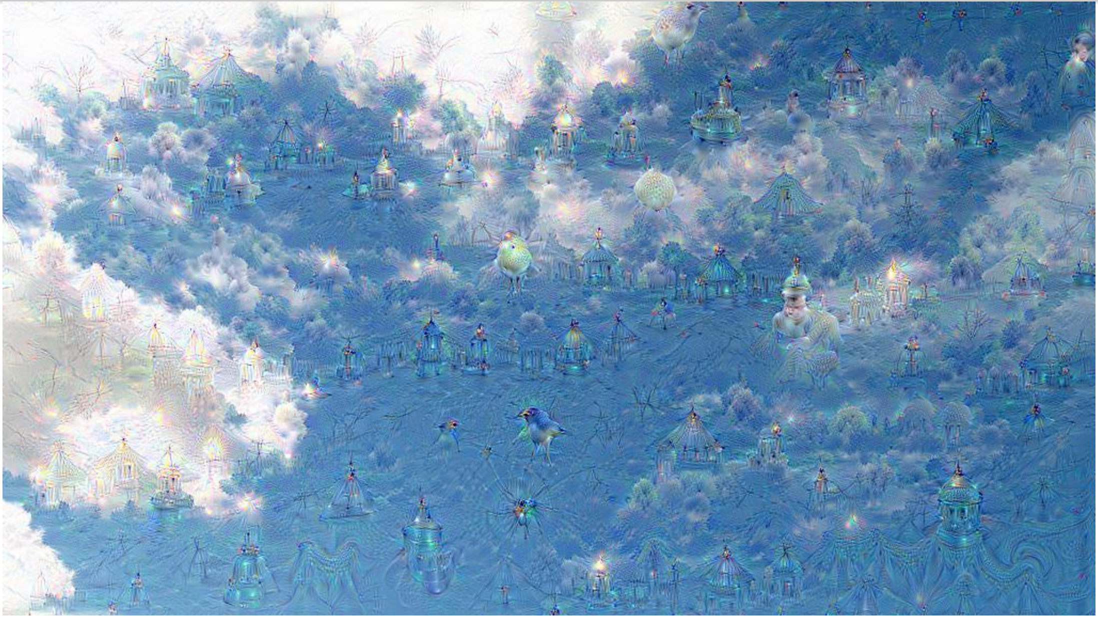


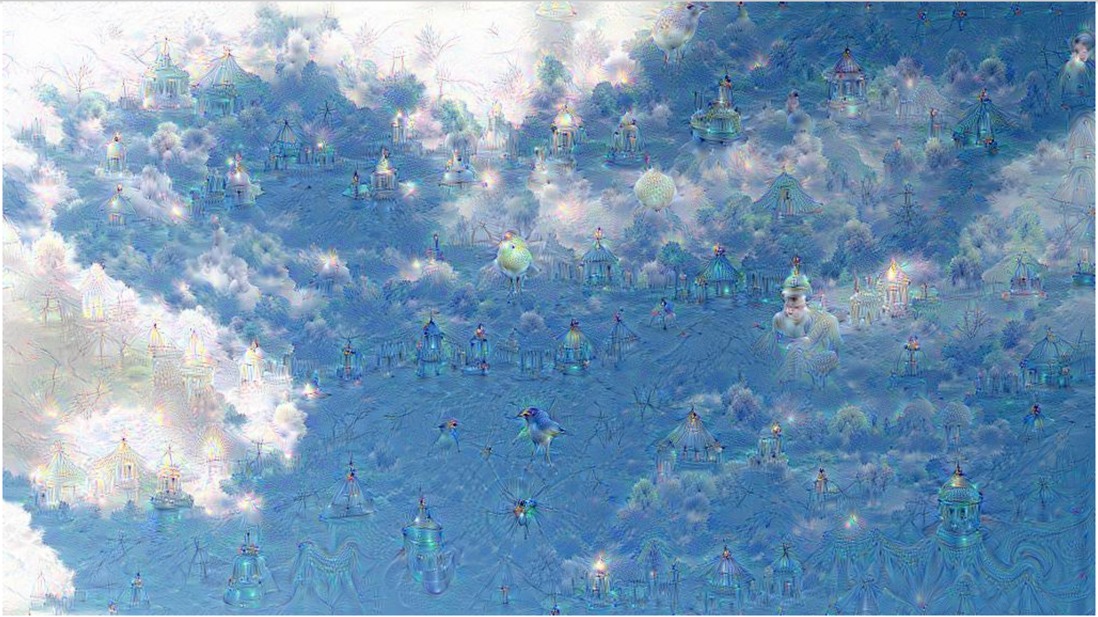


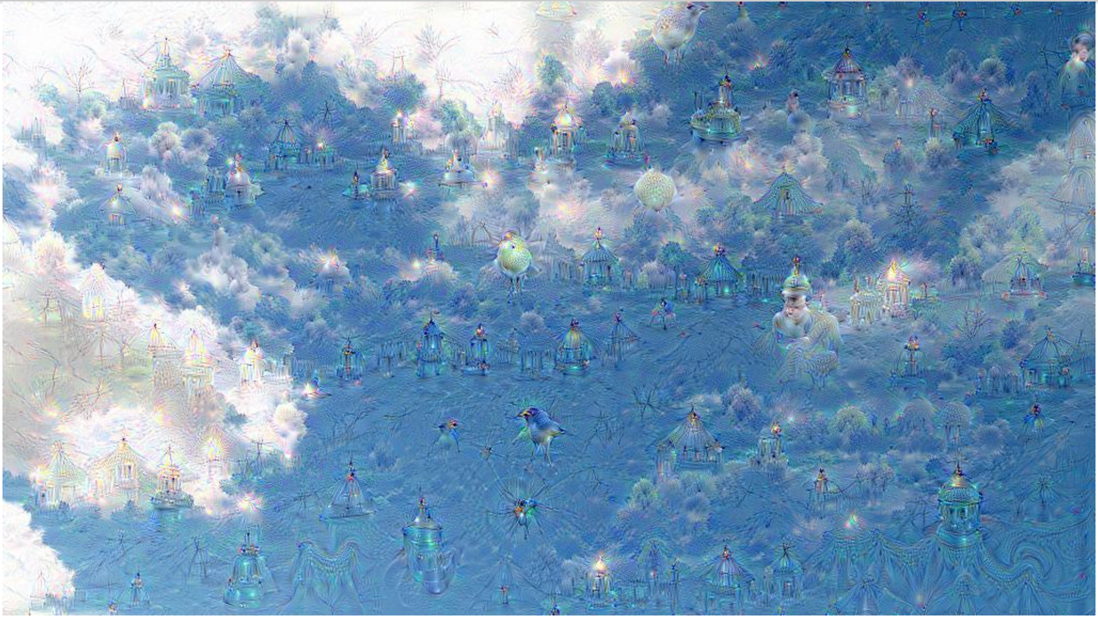


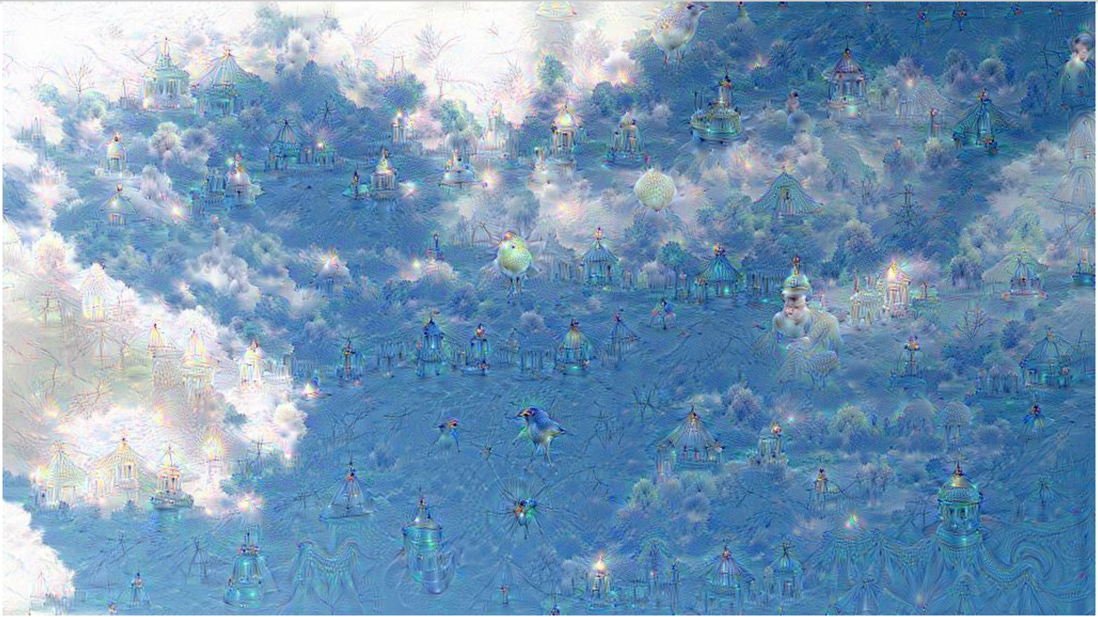


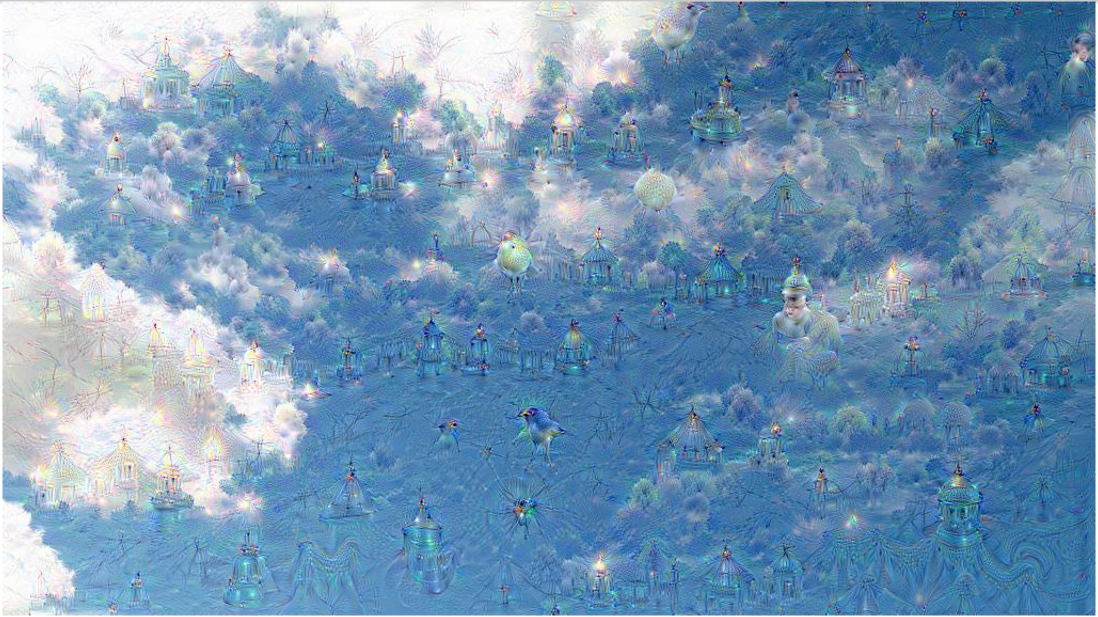


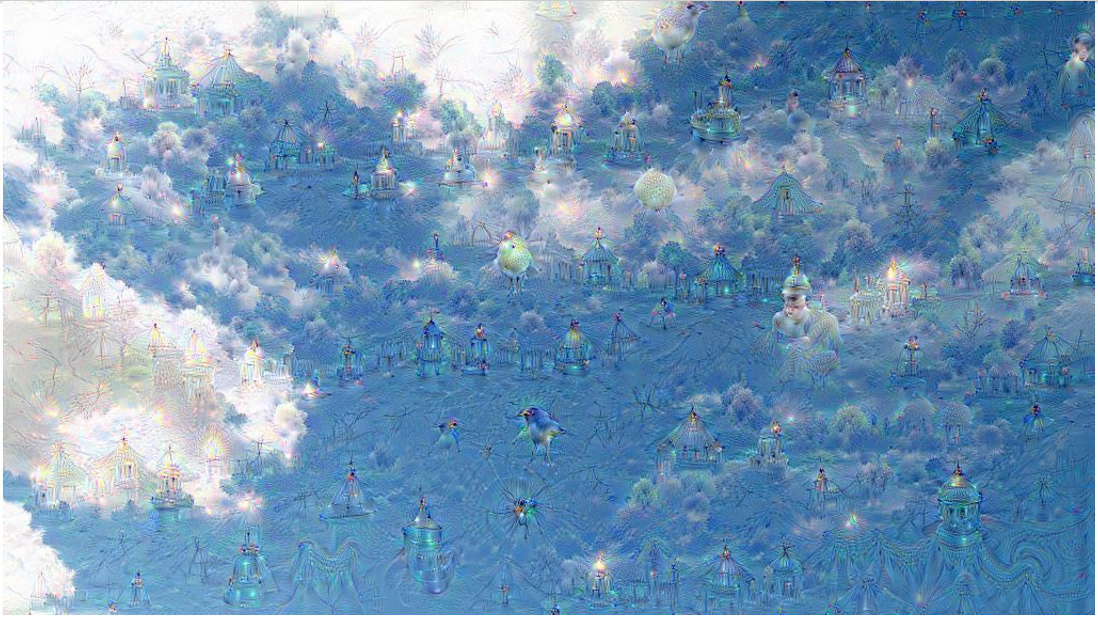


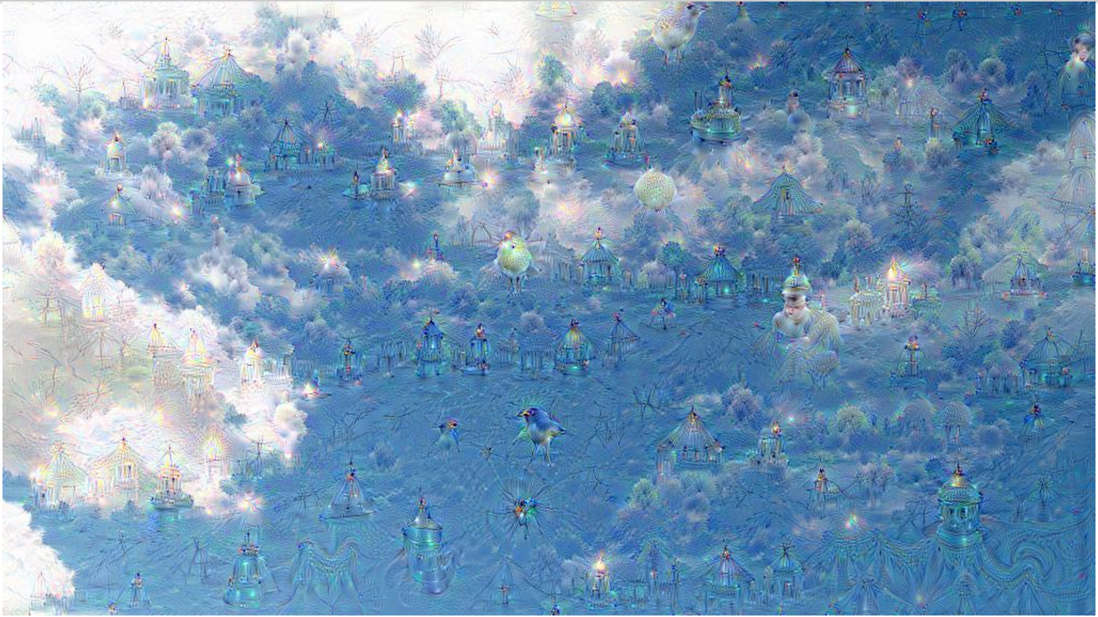


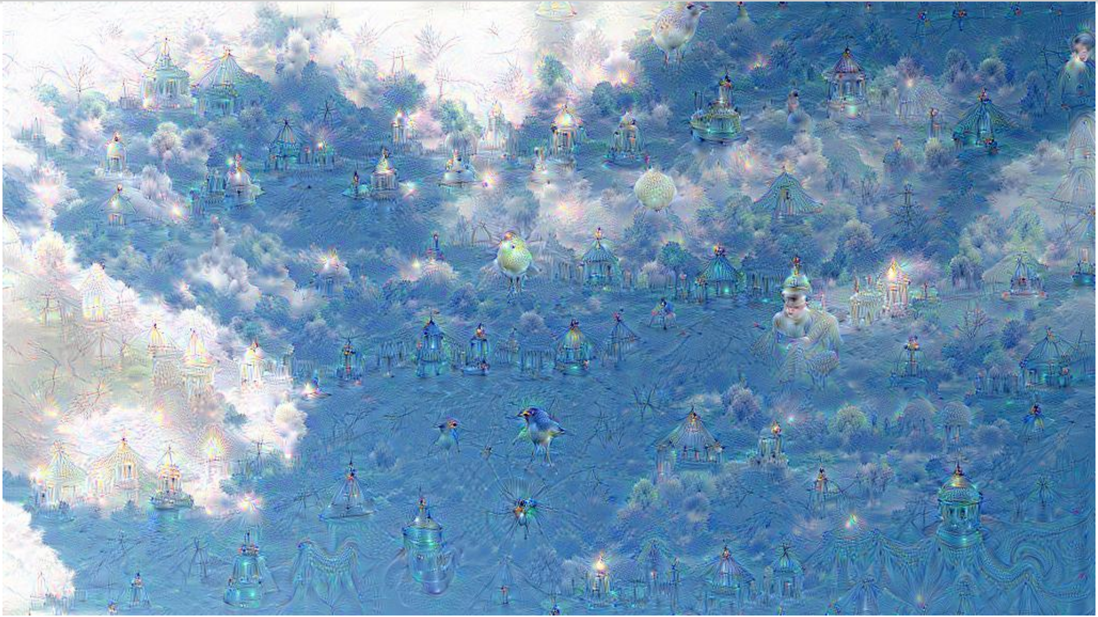


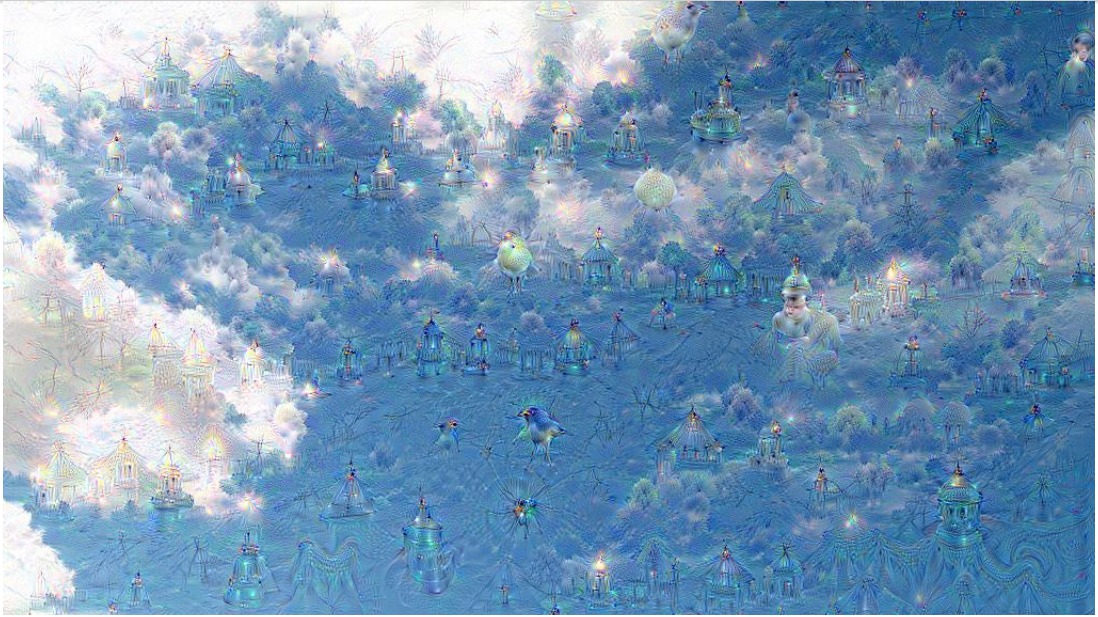


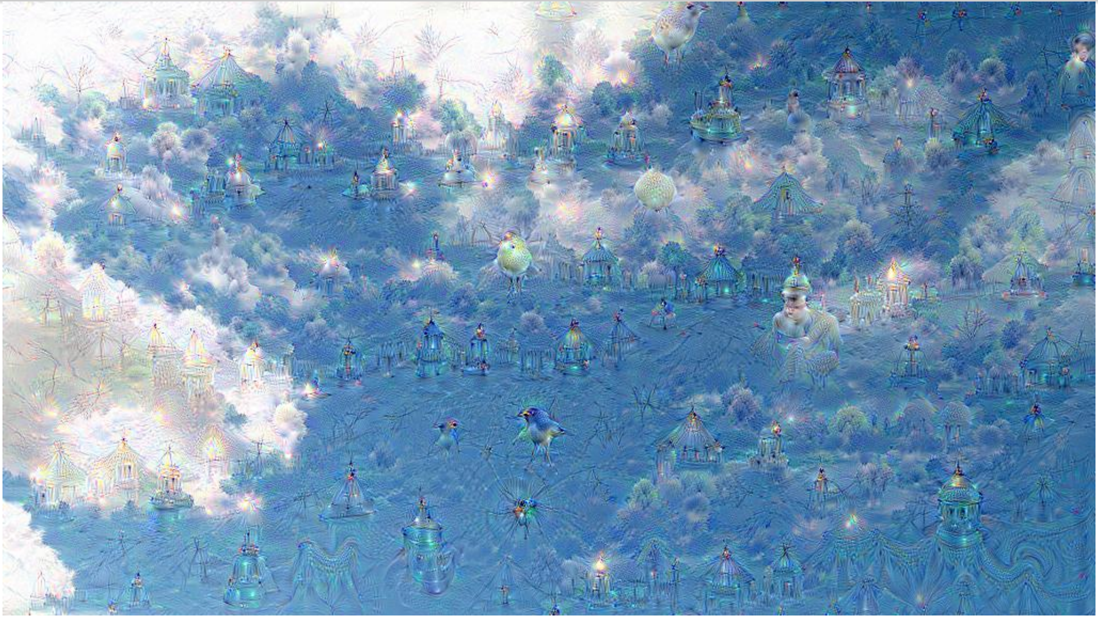


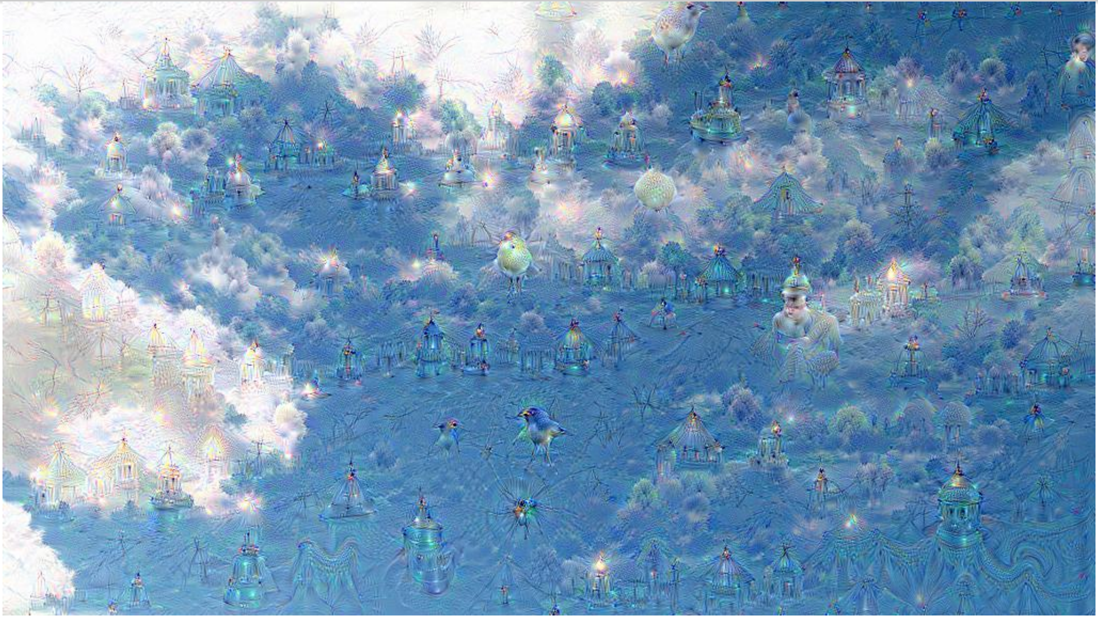


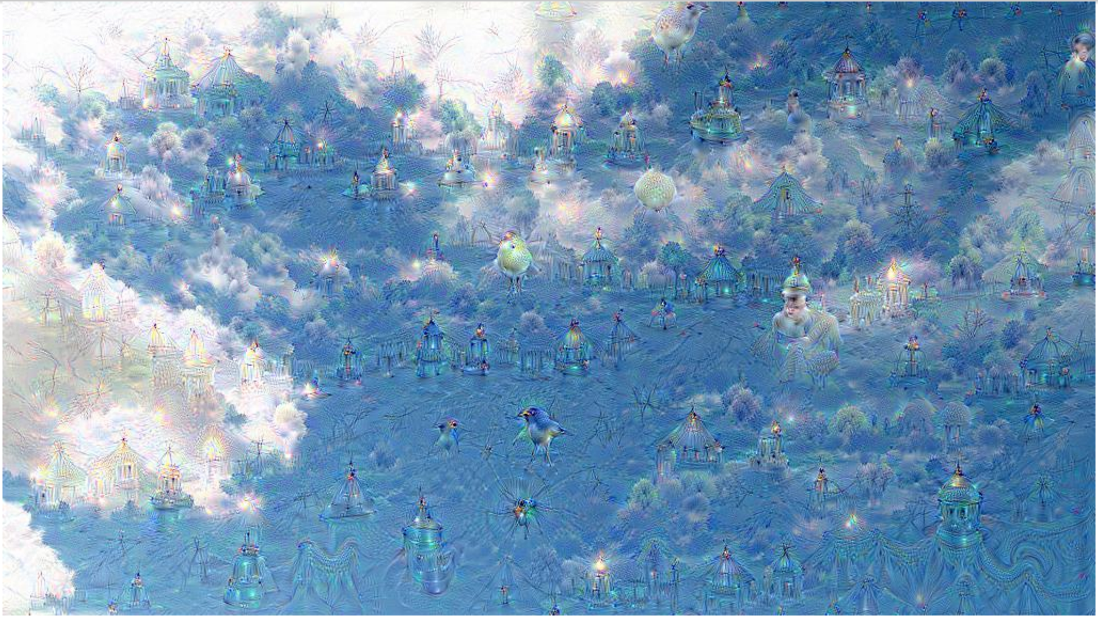


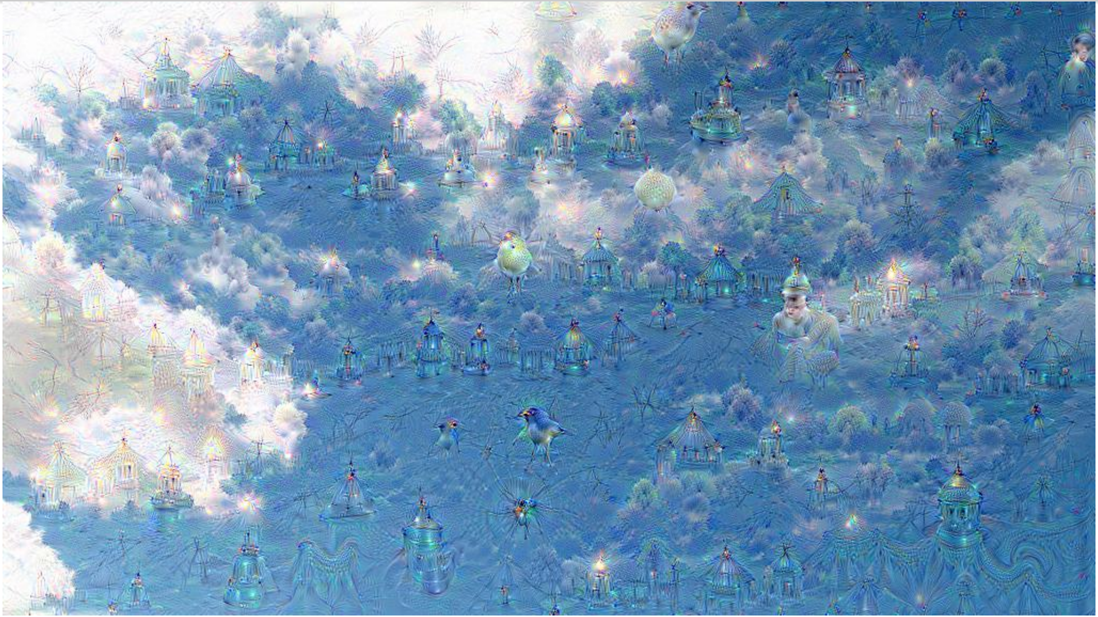


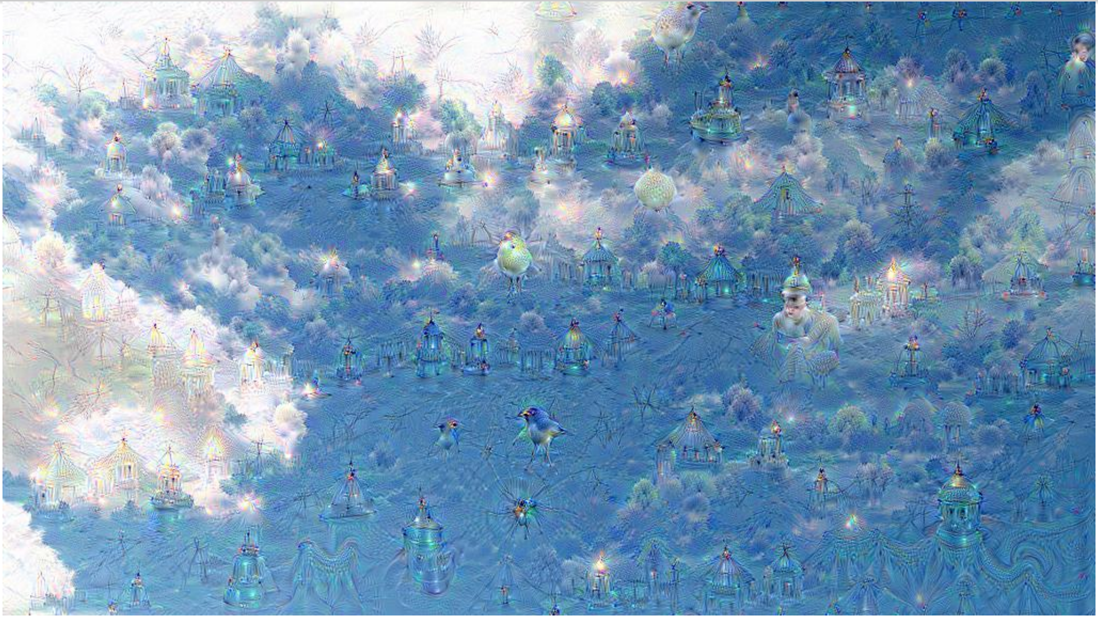


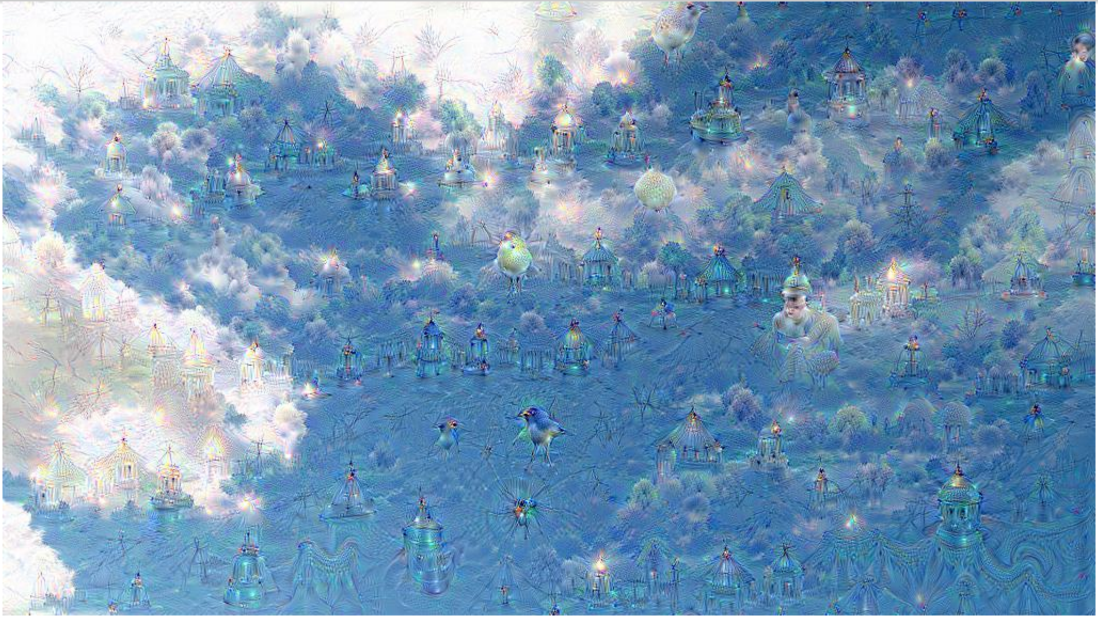


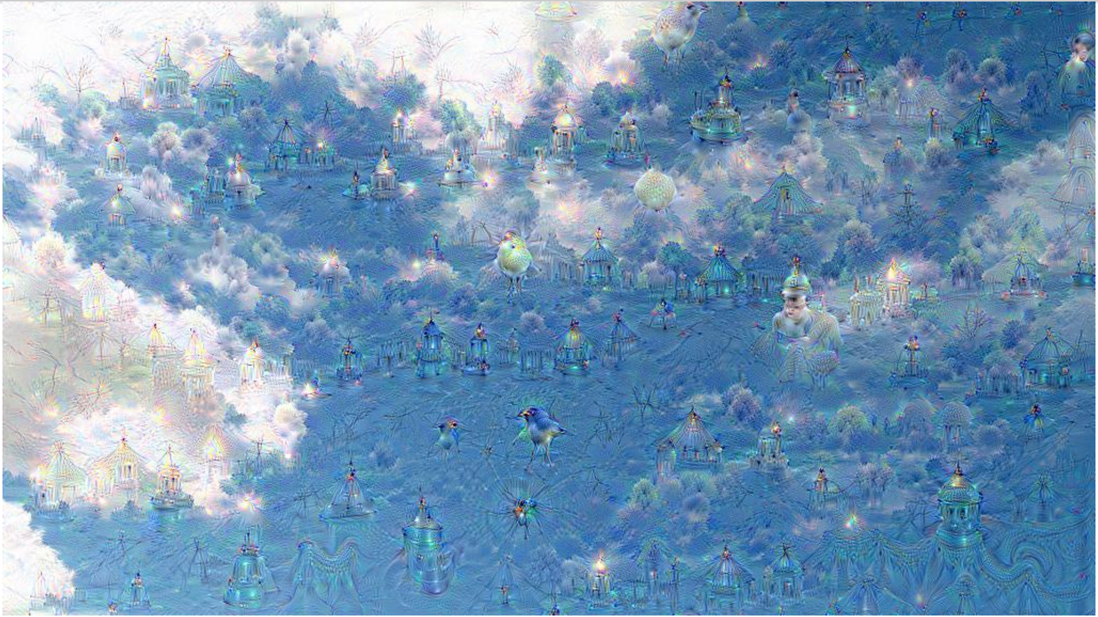


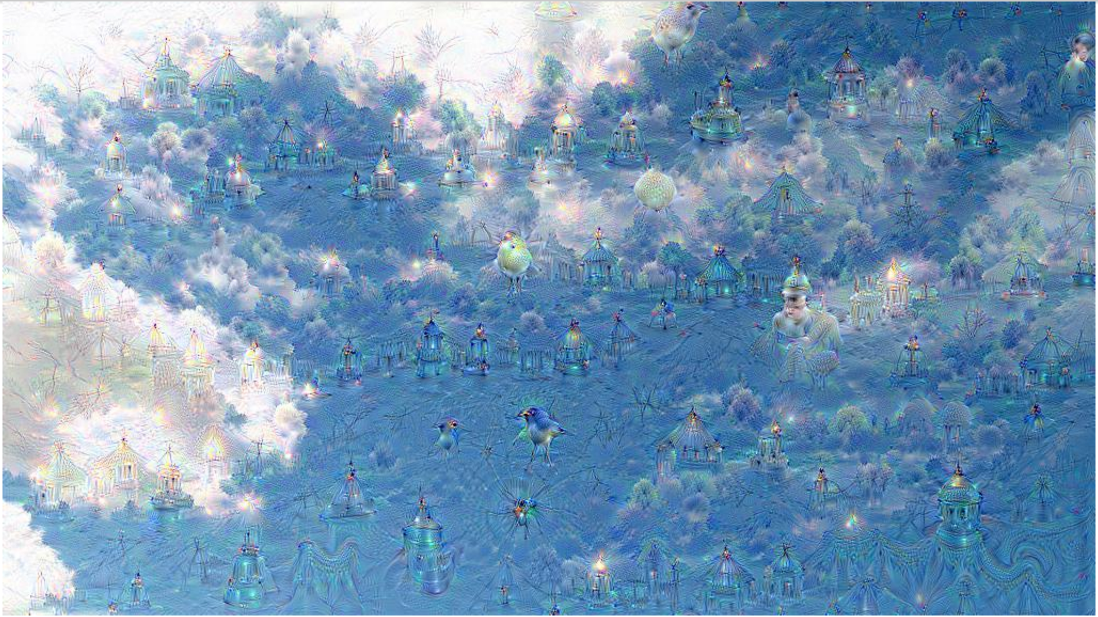


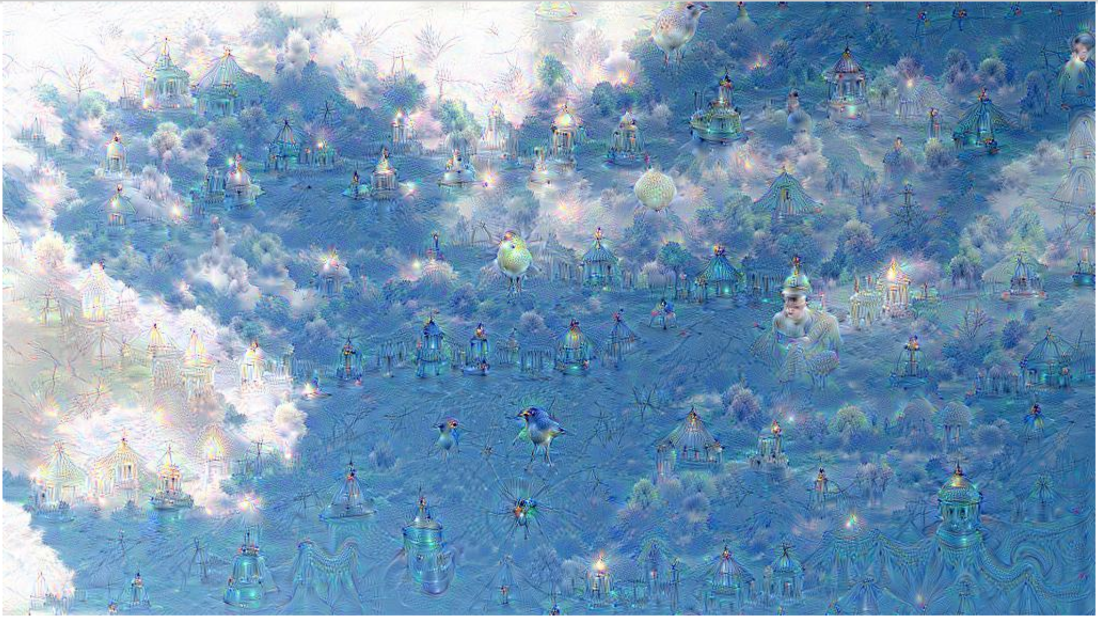


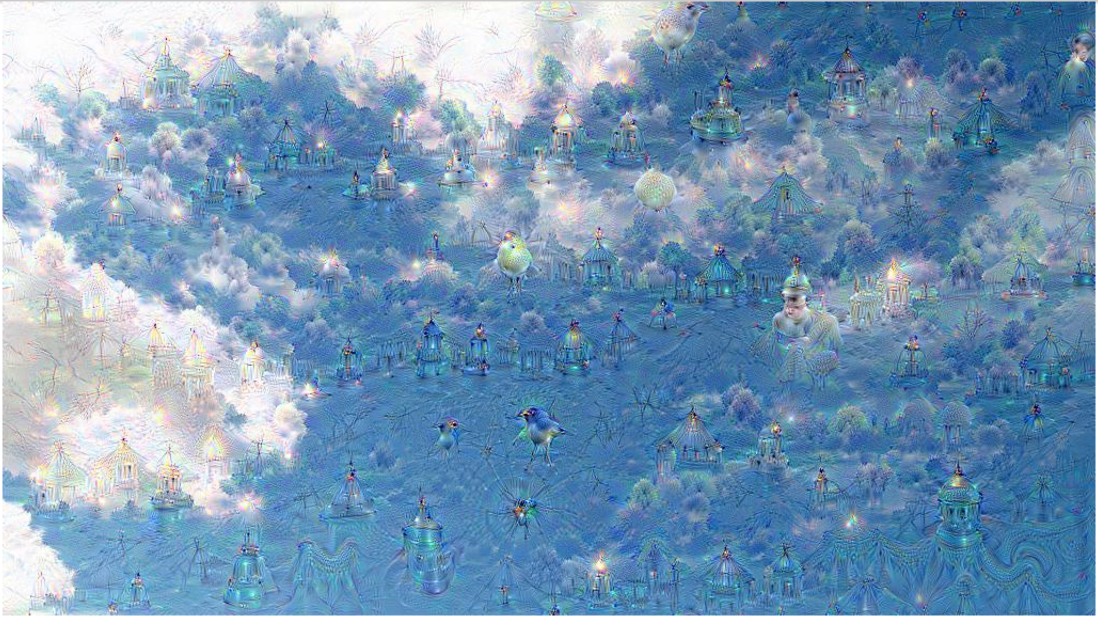


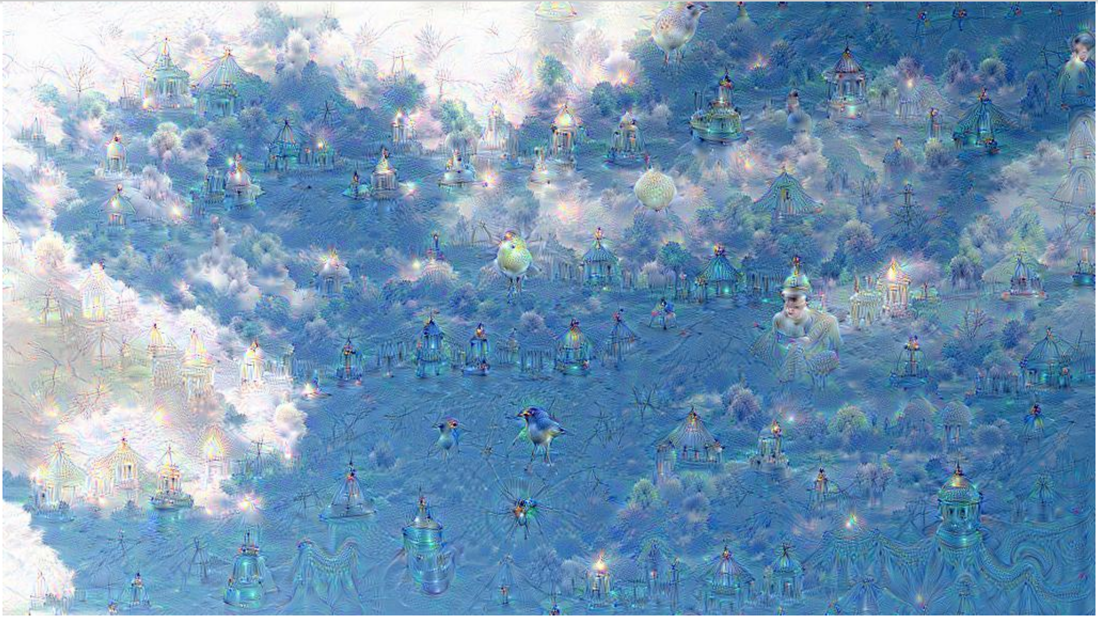


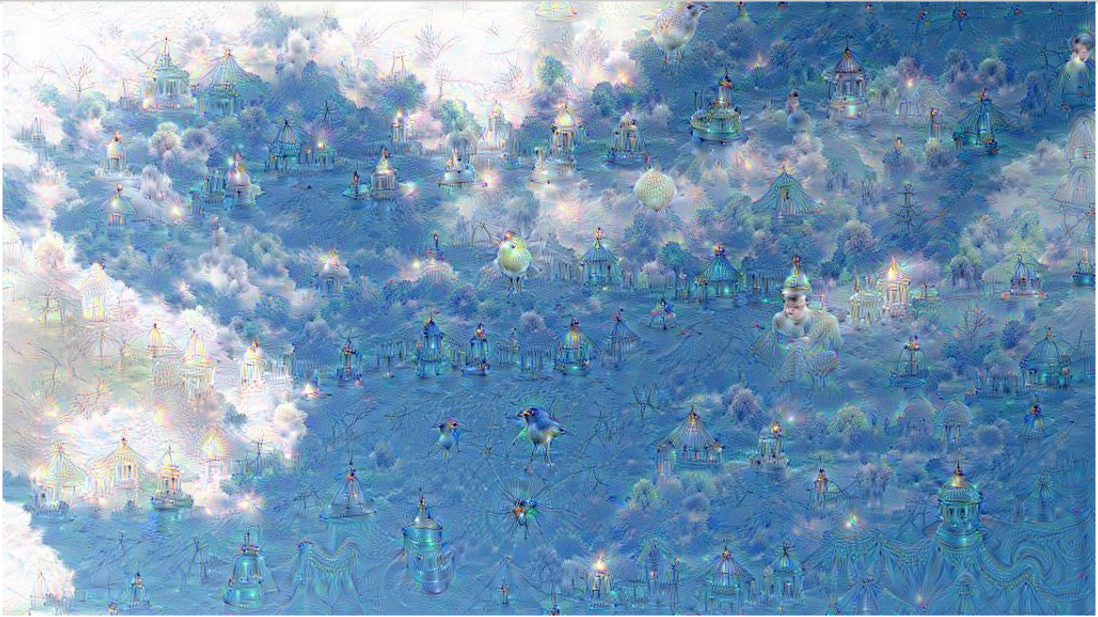


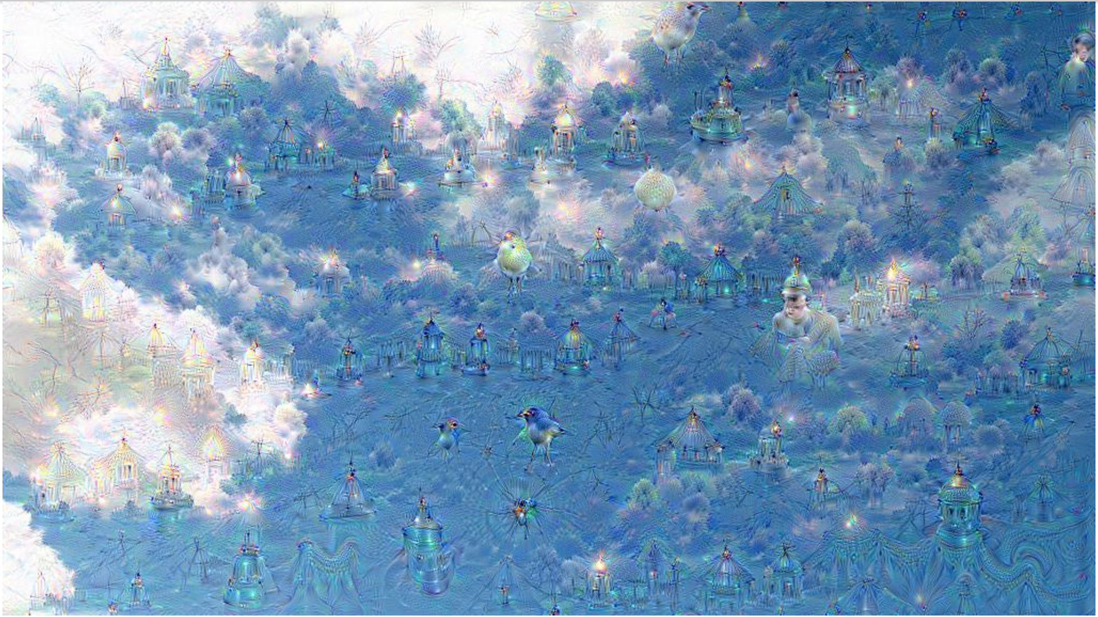


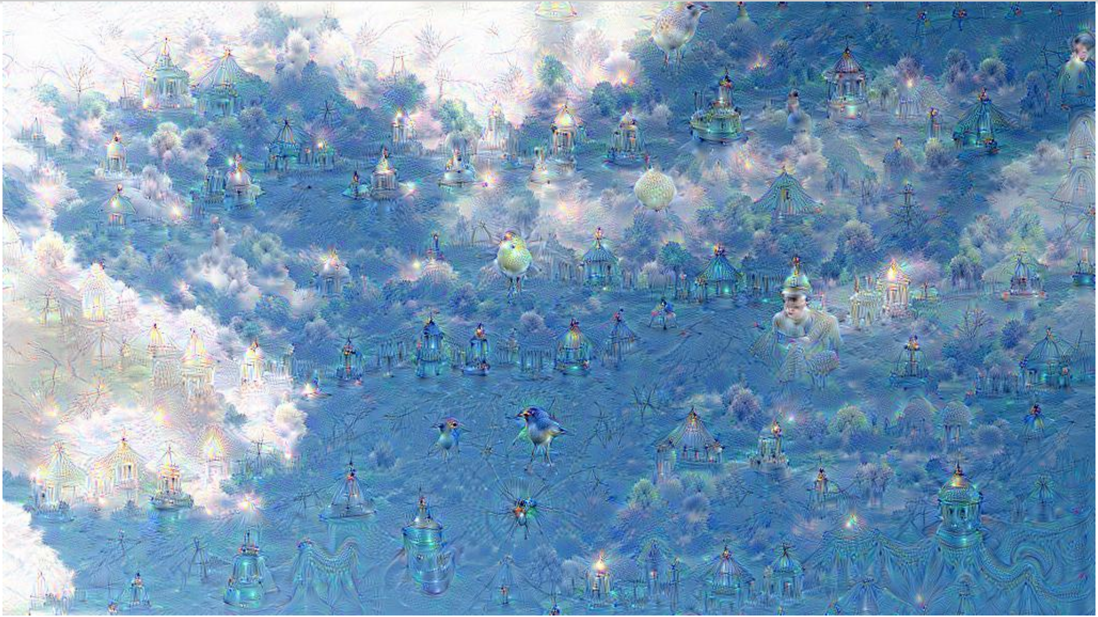


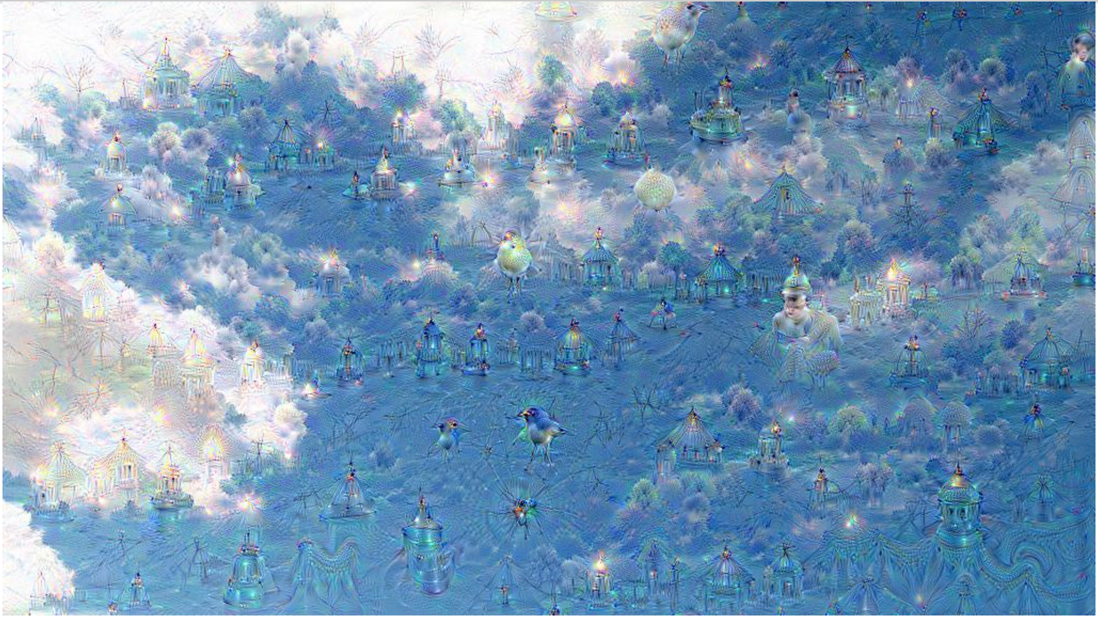


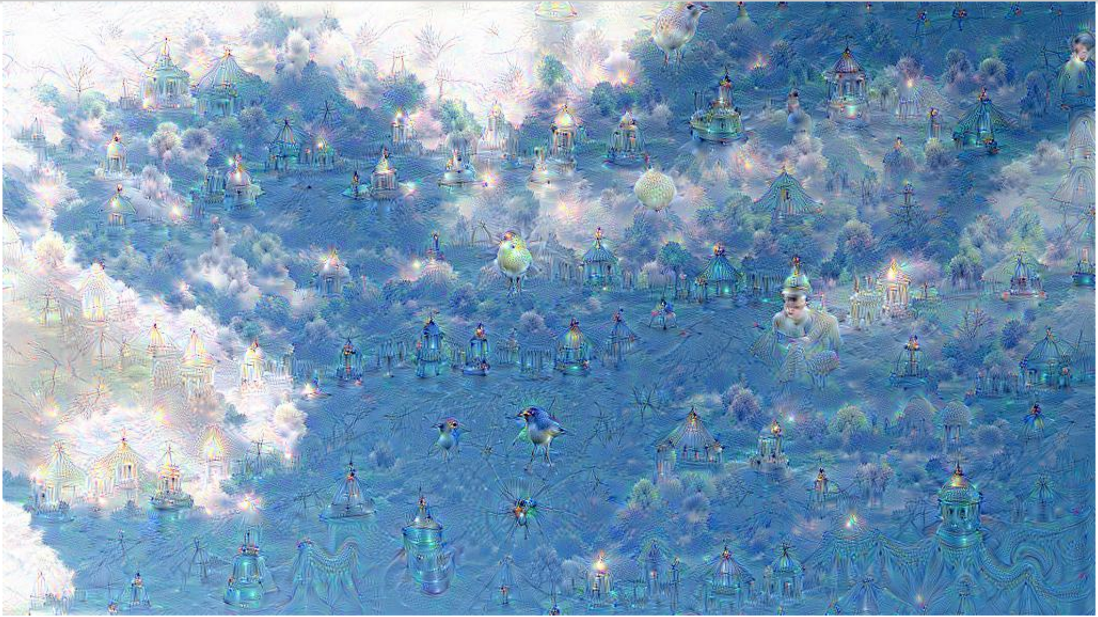


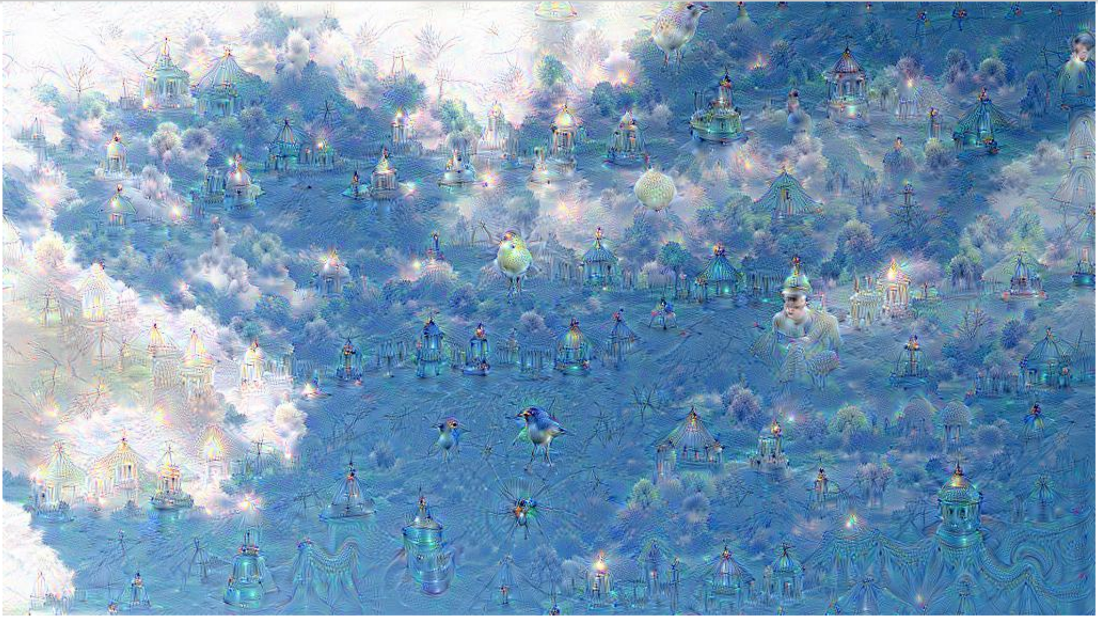


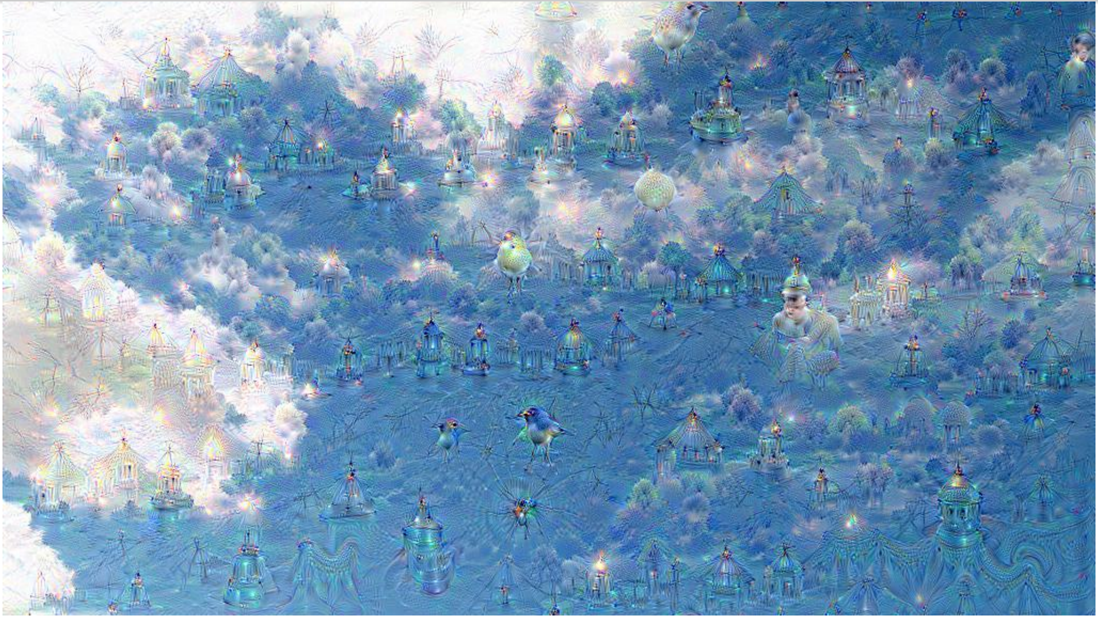


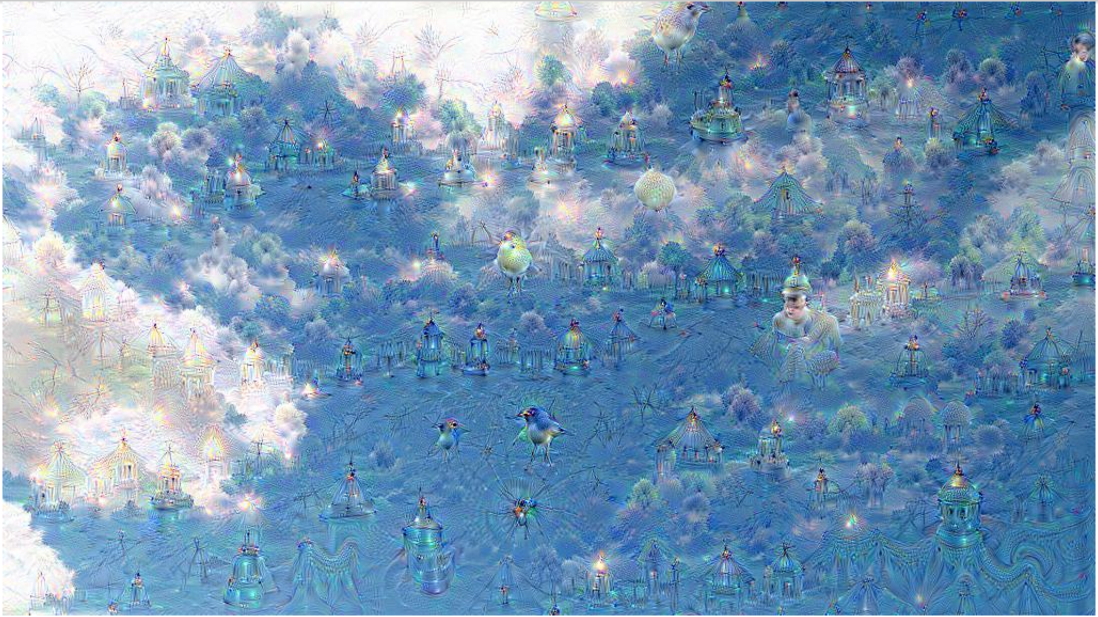


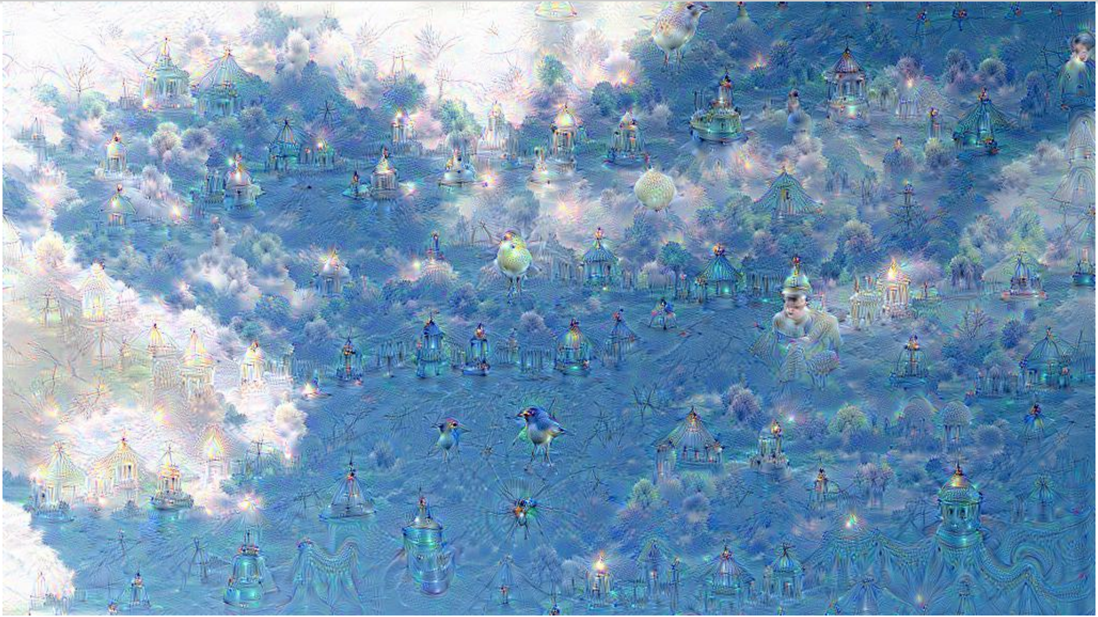


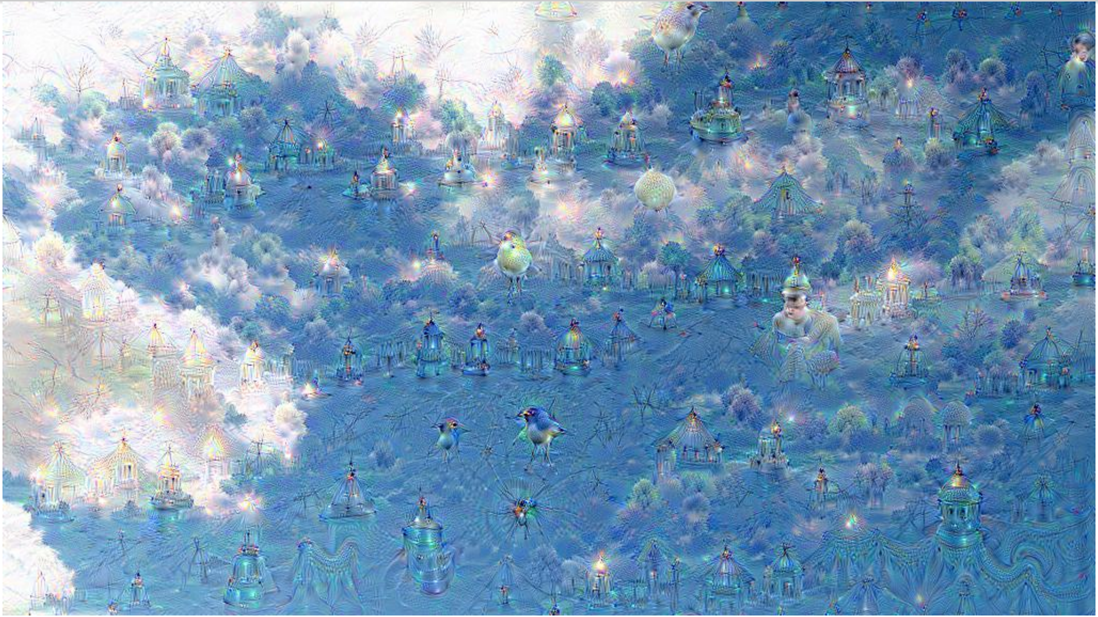


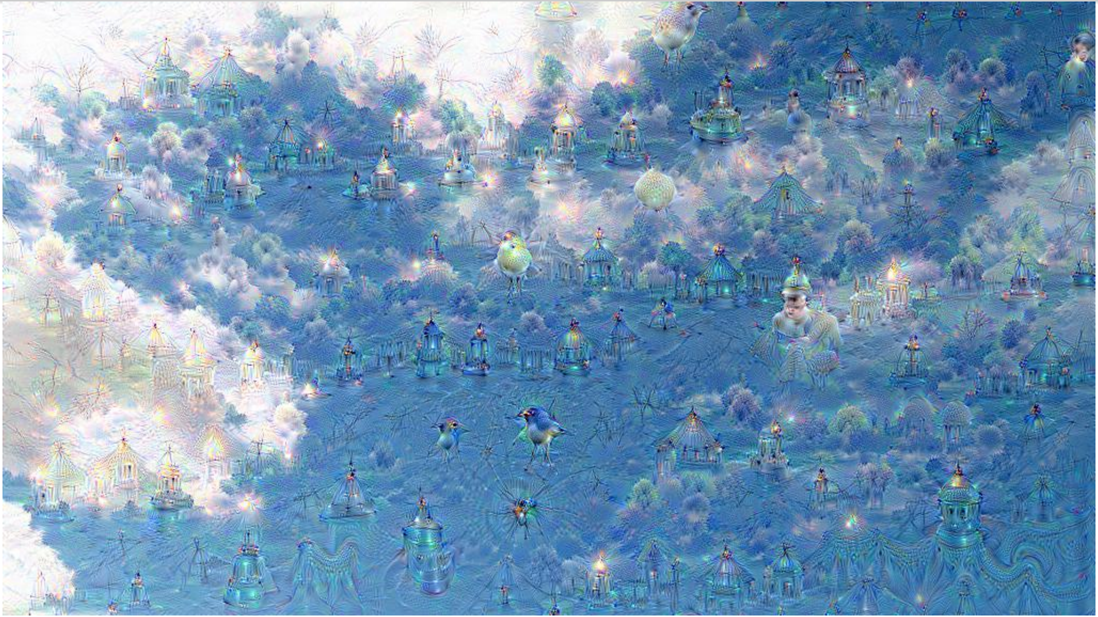


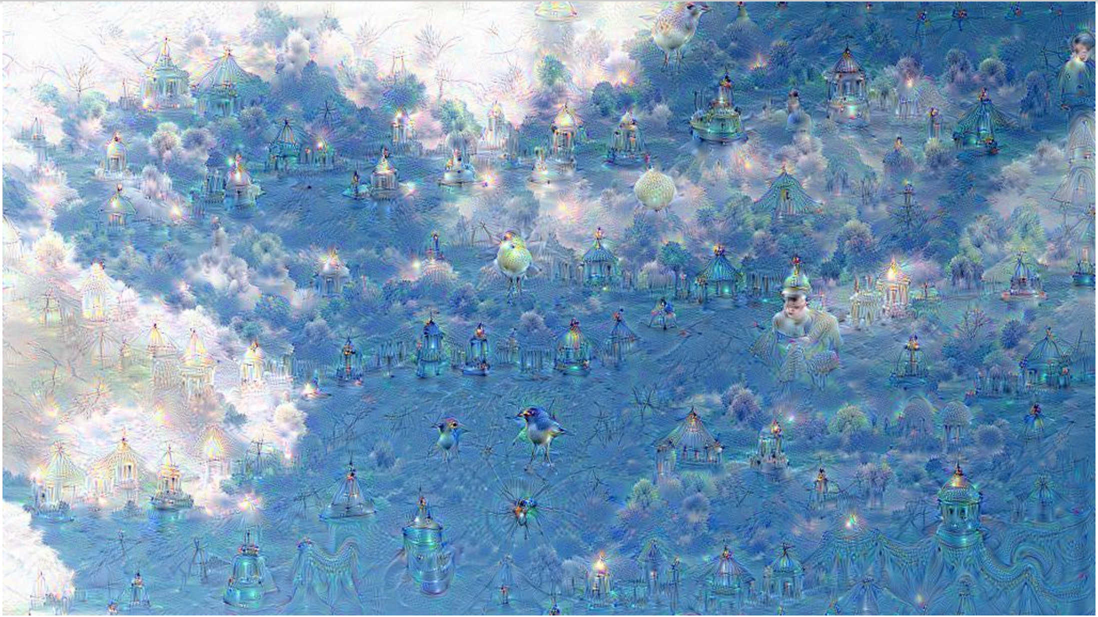


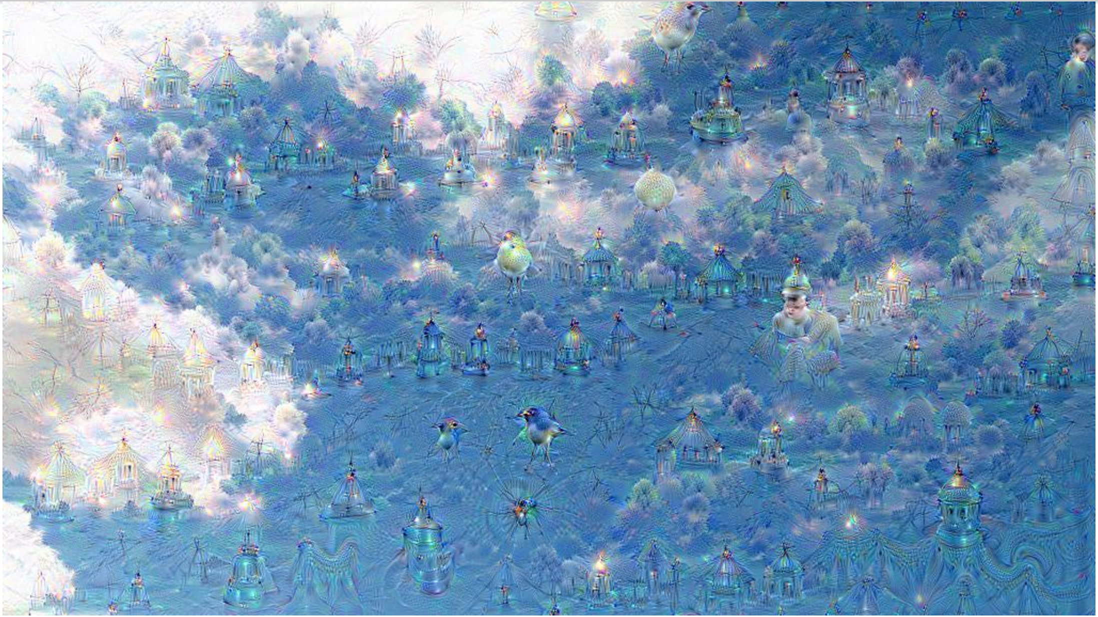


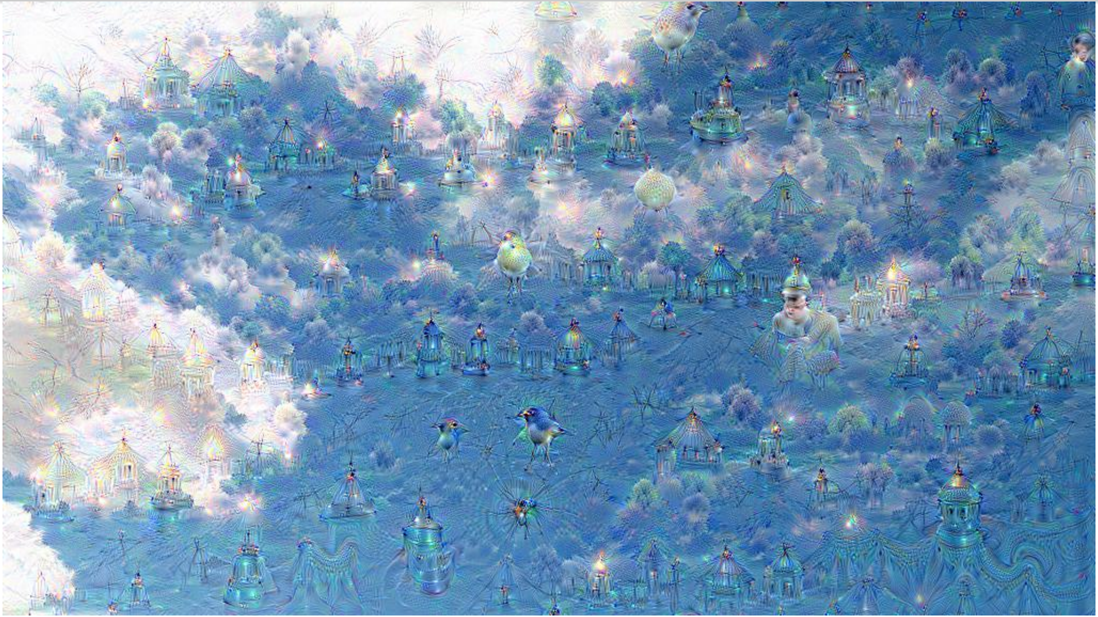


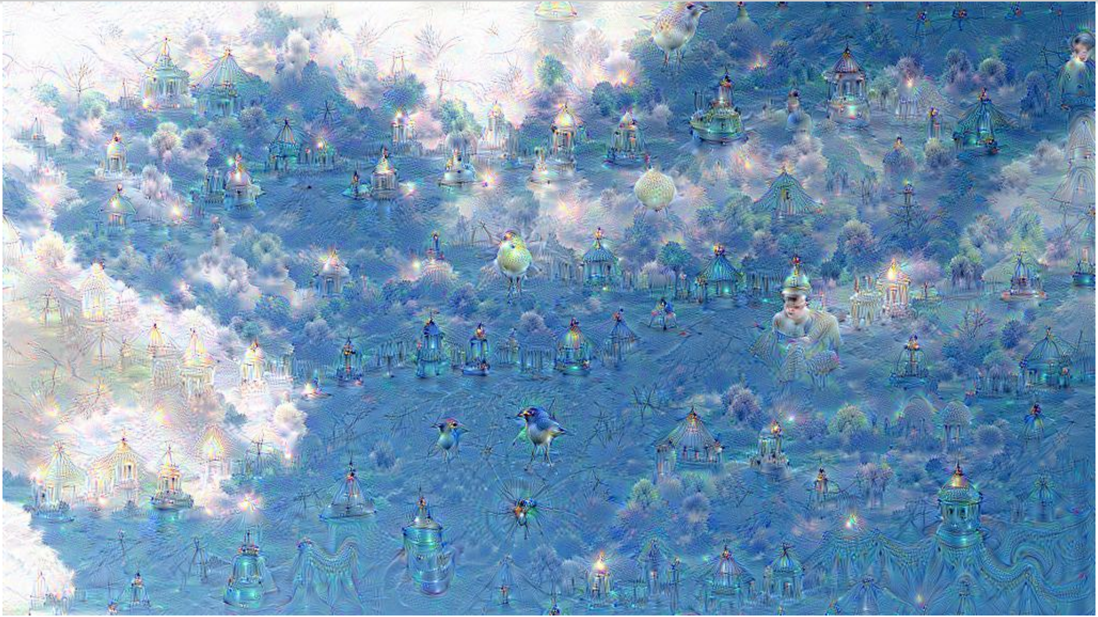


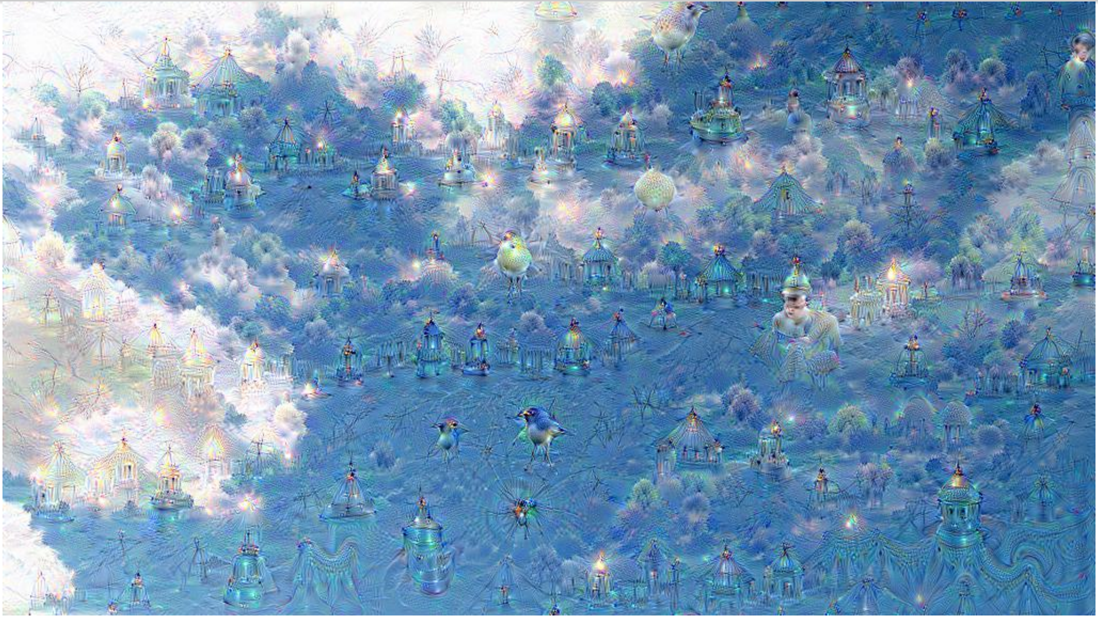


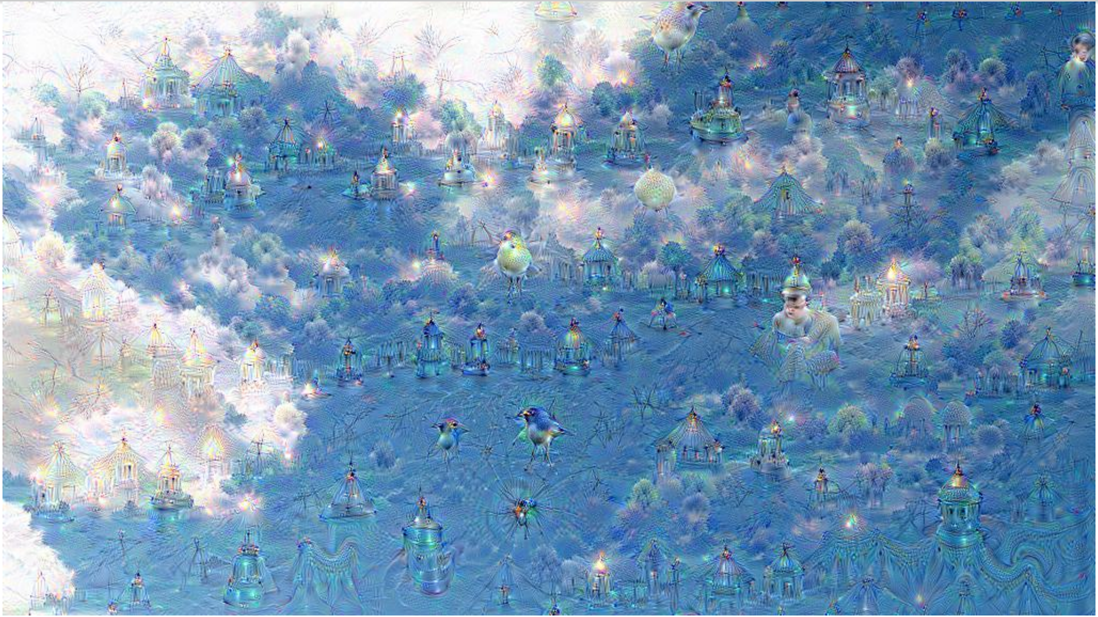


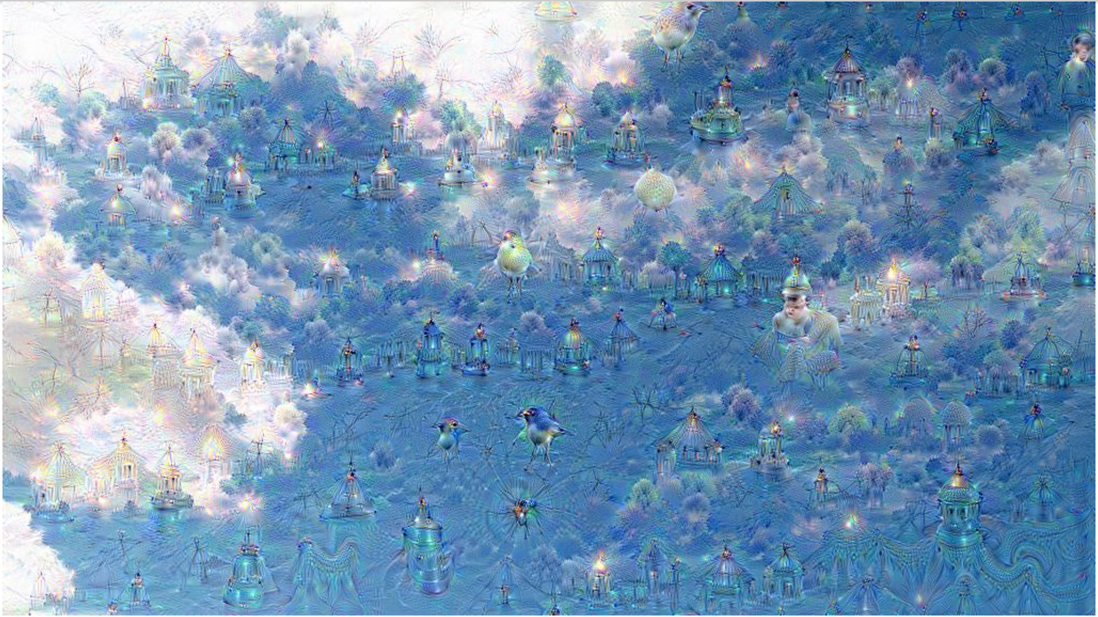


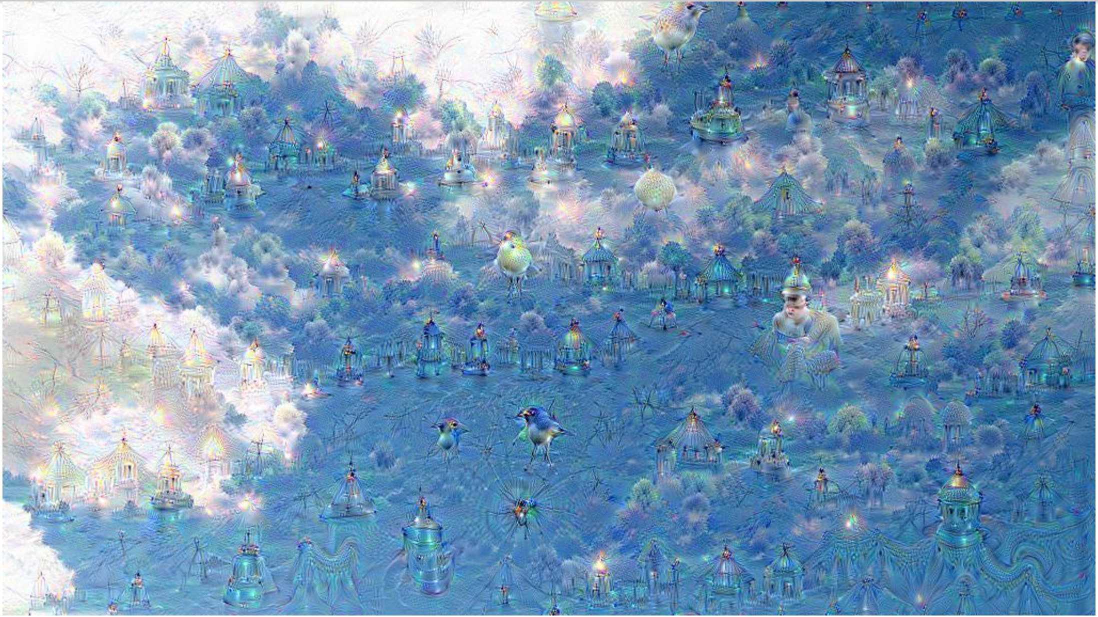


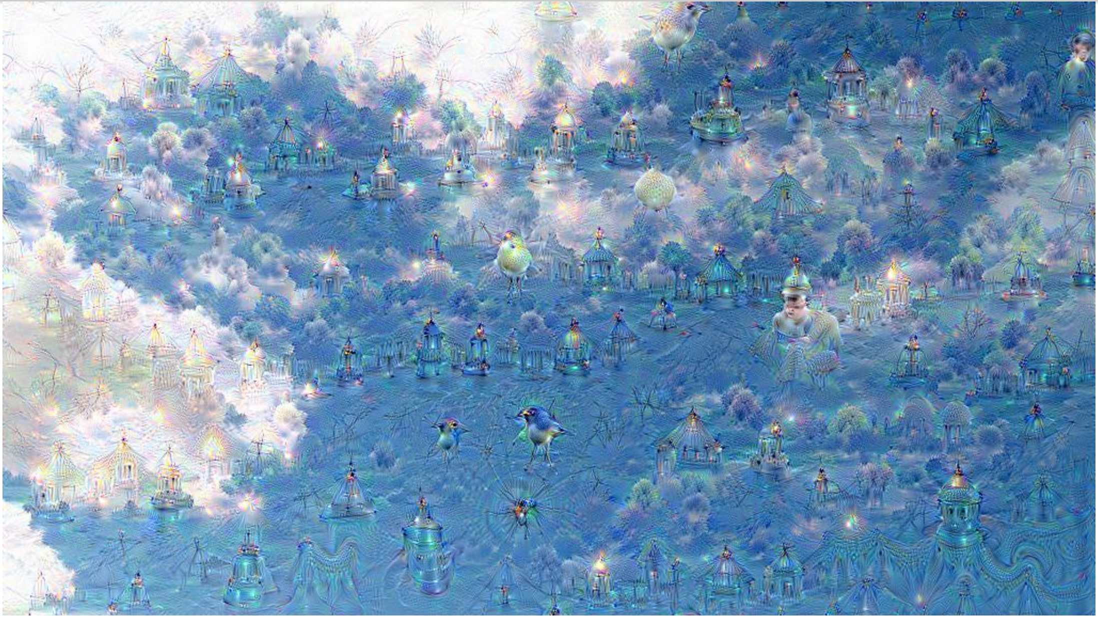


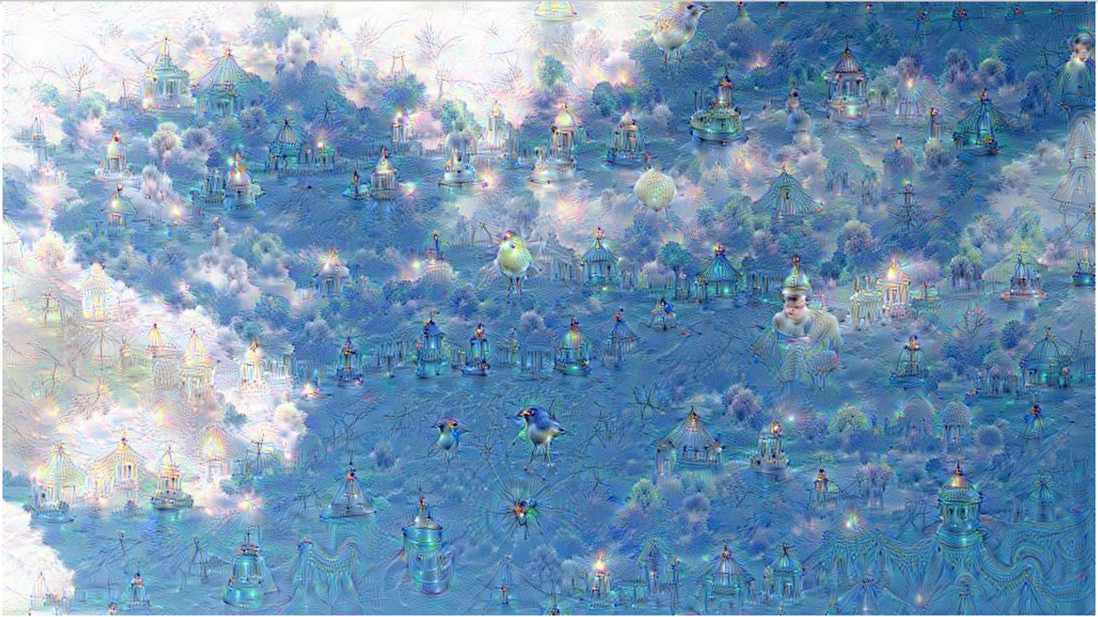


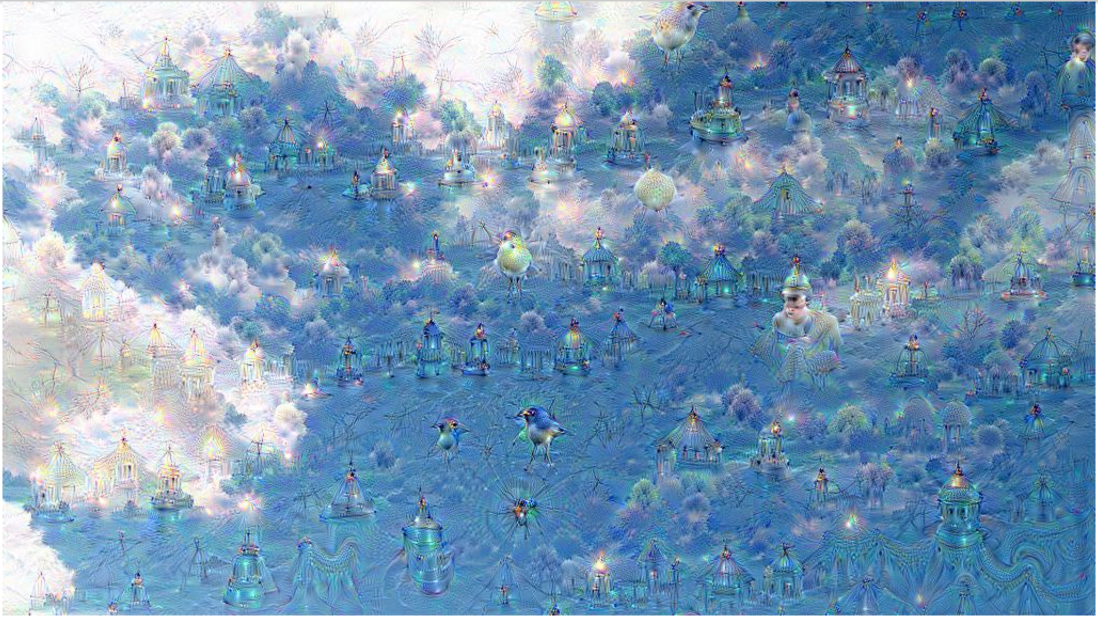


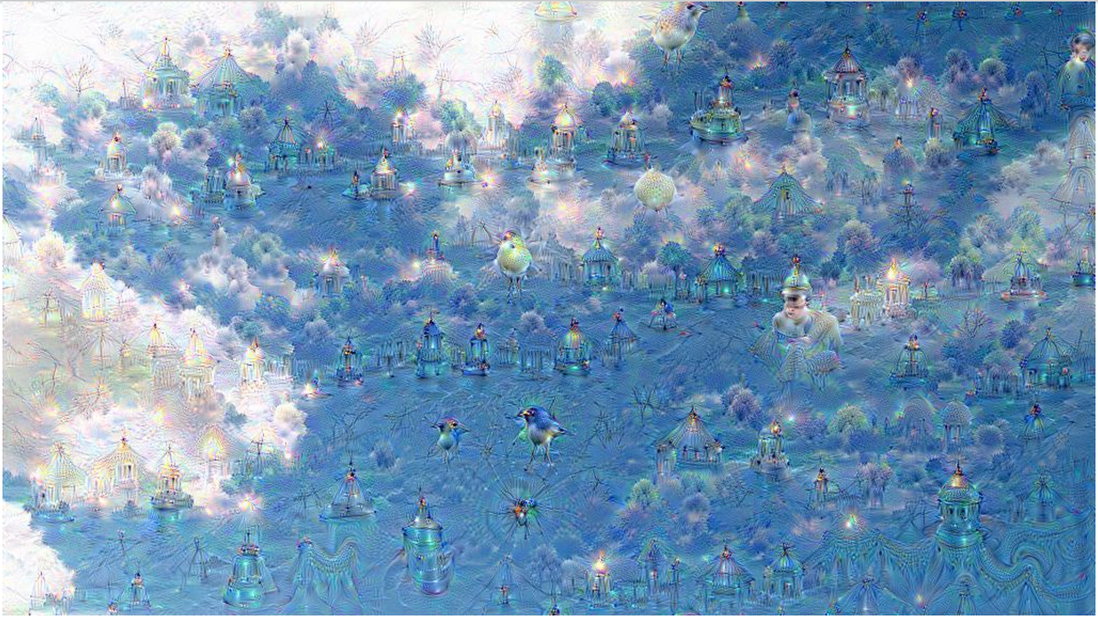


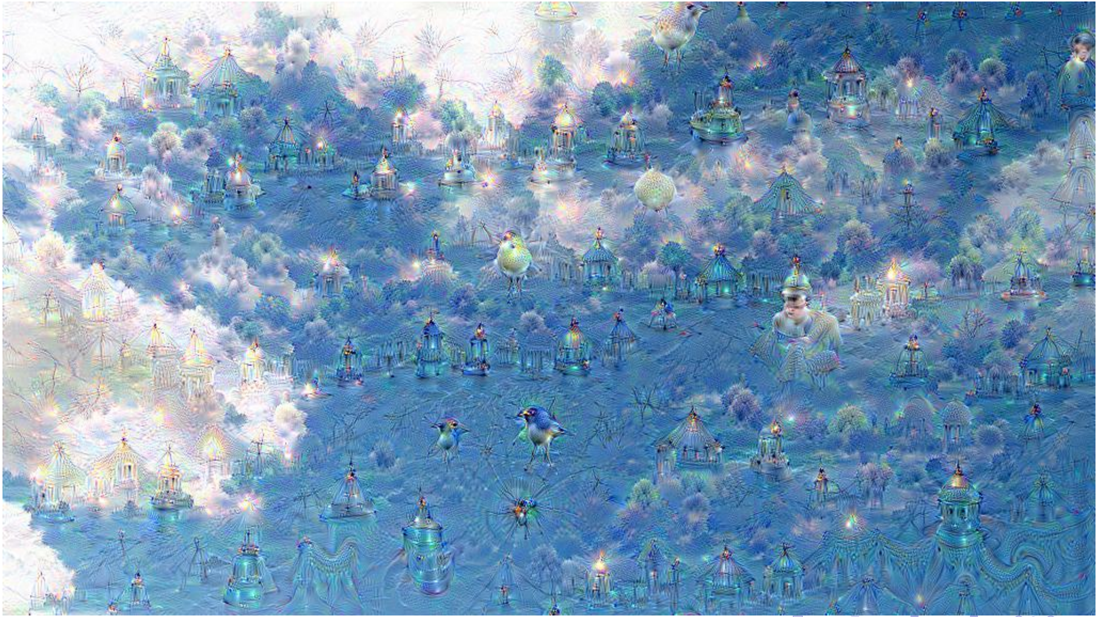













JUST JARED


JUST JARED


JUST JARED


JUST JARED


JUST JARED


JUST JARED


JUST JARED


JUST JARED


JUST JARED


JUST JARED


JUST JARED


JUST JARED


JUST JARED


JUST JARED


JUST JARED


JUST JARED


JUST JARED


JUST JARED


JUST JARED


JUST JARED


JUST JARED


JUST JARED


JUST
JARED


JUST JARED


JUST
JARED


JUST JARED


JUST JARED


JUST JARED


JUST JARED


JUST JARED


JUST JARED


JUST JARED


JUST JARED


JUST JARED


JUST JARED


JUST JARED


JUST JARED


JUST JARED


JUST JARED


JUST JARED


JUST JARED


JUST JARED


JUST JARED


JUST JARED

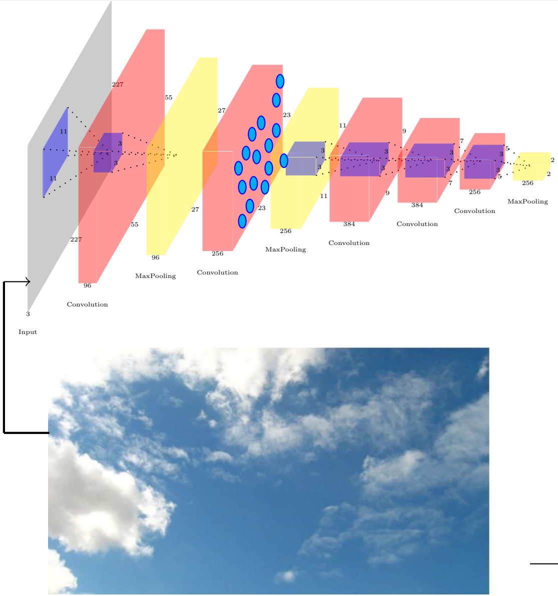

JUST JARED


JUST JARED


JUST JARED

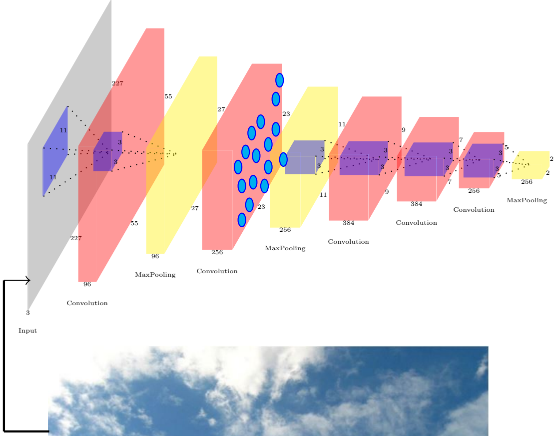

JUST JARED


JUST JARED



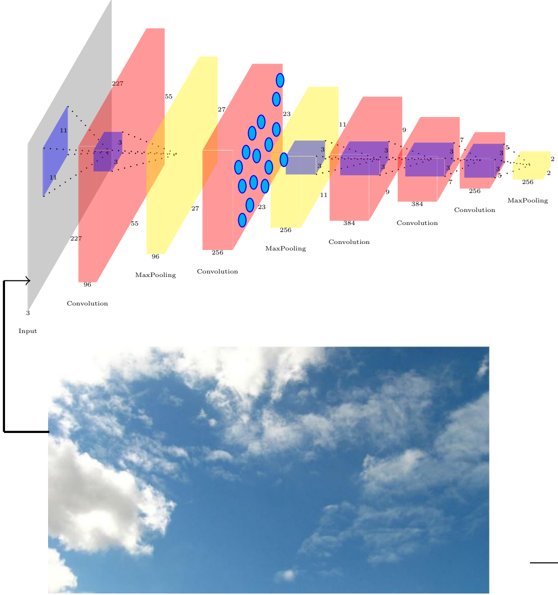
- So what exactly is happening here?

*research.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html



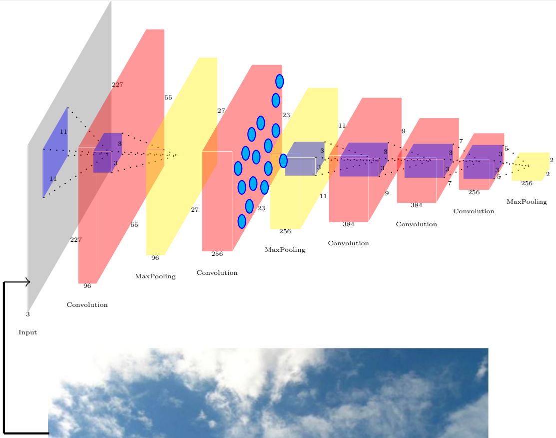
- So what exactly is happening here?
- The network has been trained to detect certain patterns (dogs, cat, birds etc.) which appear frequently in the ImageNet data

*research.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html



- So what exactly is happening here?
- The network has been trained to detect certain patterns (dogs, cat, birds etc.) which appear frequently in the ImageNet data
- It starts seeing these patterns even when they hardly exist

*research.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html



- So what exactly is happening here?
- The network has been trained to detect certain patterns (dogs, cat, birds etc.) which appear frequently in the ImageNet data
- It starts seeing these patterns even when they hardly exist
- *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. - Google**

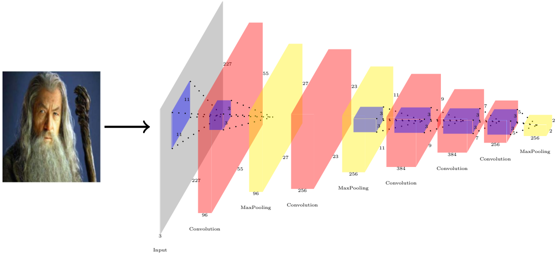
*research.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html

Module 13.9: Deep Art

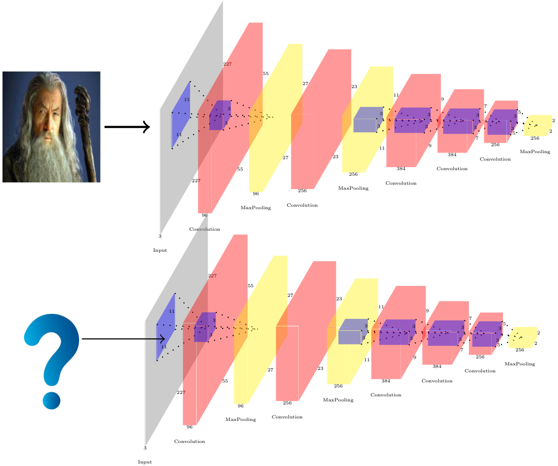
+

=

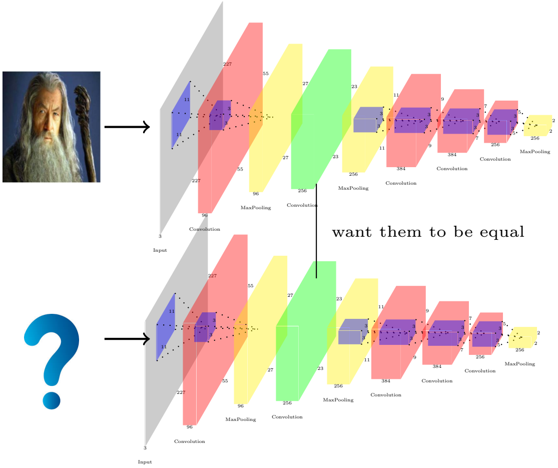
- To design a network which can do this, we first define two quantities



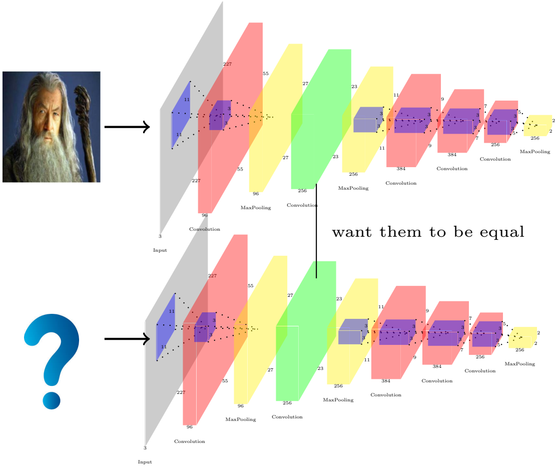
- To design a network which can do this, we first define two quantities
- **Content Targets** : The activations of all layers for the given content image



- To design a network which can do this, we first define two quantities
- **Content Targets** : The activations of all layers for the given content image
- Ideally, we would want the new image to be such that

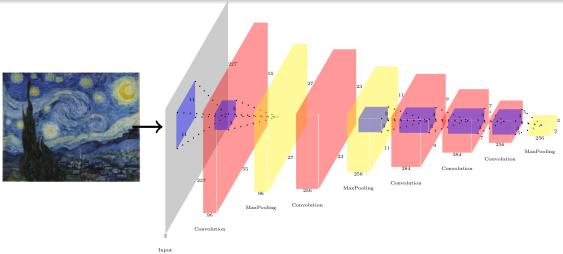


- To design a network which can do this, we first define two quantities
- **Content Targets** : The activations of all layers for the given content image
- Ideally, we would want the new image to be such that it's activations are also close to those of the original content image

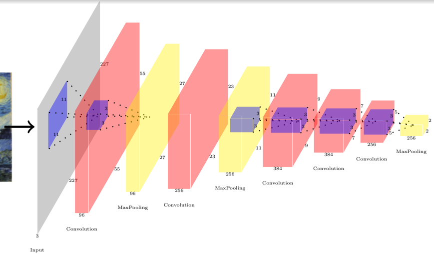


- To design a network which can do this, we first define two quantities
- **Content Targets** : The activations of all layers for the given content image
- Ideally, we would want the new image to be such that it's activations are also close to those of the original content image
- Let $\vec{p}, \vec{x}$ be the activations of the content image and the new image (to be generated) respectively

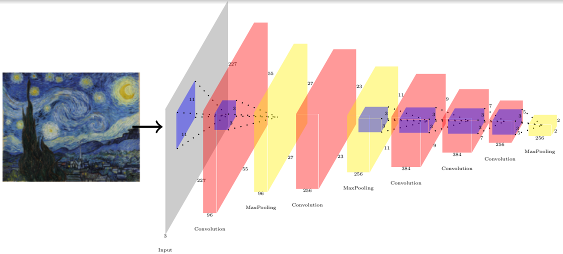
$$\mathscr{L}_{content}(\vec{p}, \vec{x}) = \sum_{ijk} (\vec{p}_{ijk} - \vec{x}_{ijk})^2$$



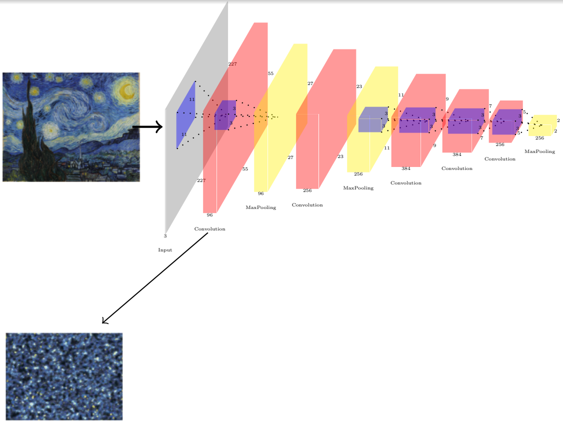
- Next we would want the style of the generated image to be the same as the style image



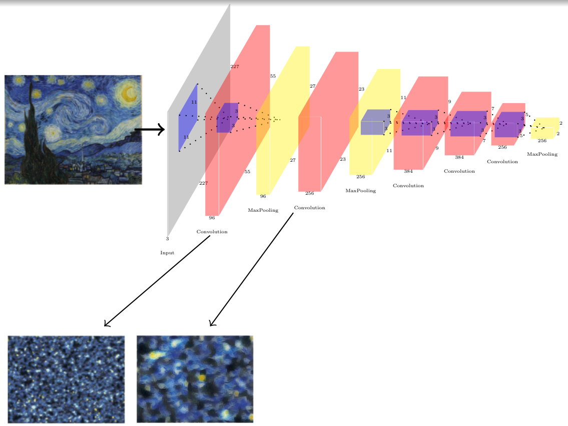
- Next we would want the style of the generated image to be the same as the style image
- How do we capture the style of the image?



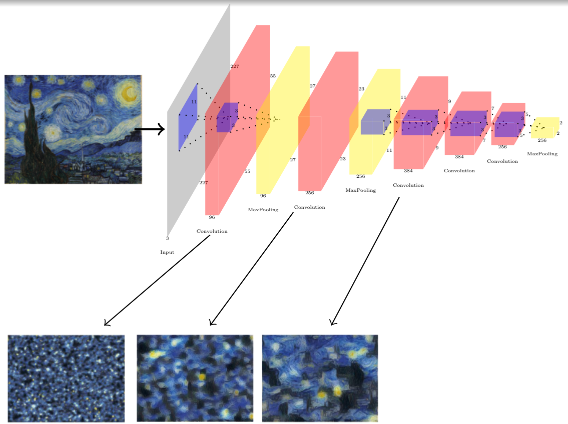
- Next we would want the style of the generated image to be the same as the style image
- How do we capture the style of the image?
- Turns out that if $V \in \mathbb{R}^{64\times(256\times256)}$ is the activation at a layer then $V^TV \in \mathbb{R}^{64\times64}$ captures the style of the image



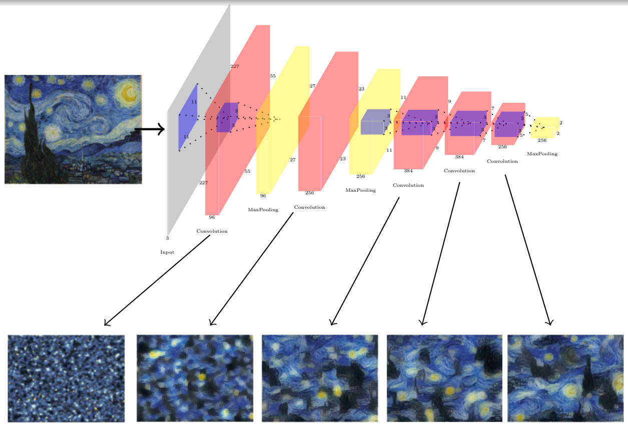
- Next we would want the style of the generated image to be the same as the style image
- How do we capture the style of the image?
- Turns out that if $V \in \mathbb{R}^{64\times(256\times256)}$ is the activation at a layer then $V^TV \in \mathbb{R}^{64\times64}$ captures the style of the image
- The deeper layers capture more of this style information



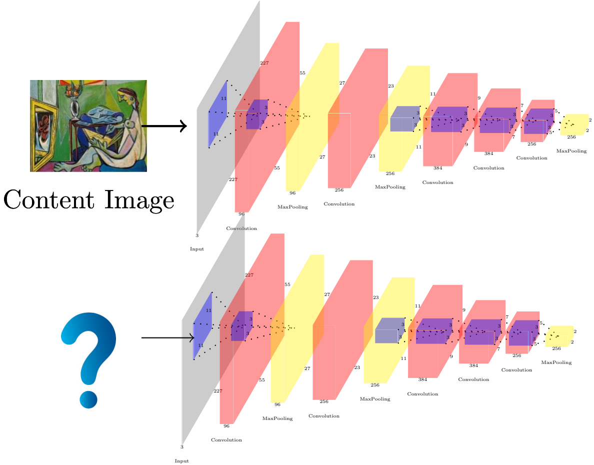
- Next we would want the style of the generated image to be the same as the style image
- How do we capture the style of the image?
- Turns out that if $V \in \mathbb{R}^{64\times(256\times256)}$ is the activation at a layer then $V^TV \in \mathbb{R}^{64\times64}$ captures the style of the image
- The deeper layers capture more of this style information



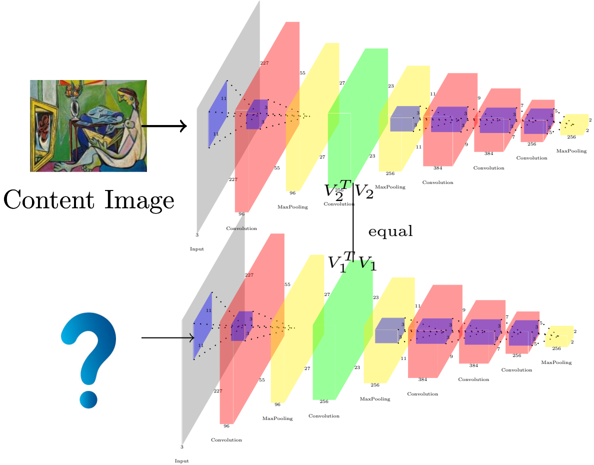
- Next we would want the style of the generated image to be the same as the style image
- How do we capture the style of the image?
- Turns out that if $V \in \mathbb{R}^{64\times(256\times256)}$ is the activation at a layer then $V^TV \in \mathbb{R}^{64\times64}$ captures the style of the image
- The deeper layers capture more of this style information



- Next we would want the style of the generated image to be the same as the style image
- How do we capture the style of the image?
- Turns out that if $V \in \mathbb{R}^{64\times(256\times256)}$ is the activation at a layer then $V^TV \in \mathbb{R}^{64\times64}$ captures the style of the image
- The deeper layers capture more of this style information



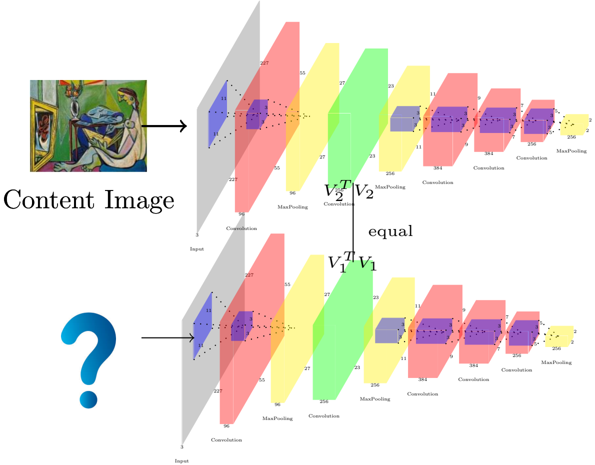
- To ensure that the style of the new image captured by layer ℓ matches the style of the style image, we can use the following objective function :



- To ensure that the style of the new image captured by layer ℓ matches the style of the style image, we can use the following objective function :

$$E_\ell = \sum_{ij} (G^\ell_j - A^\ell_{ij})^2$$

where G^ℓ and A^ℓ are the style gram matrices computed at layer ℓ for the style image and new image respectively.

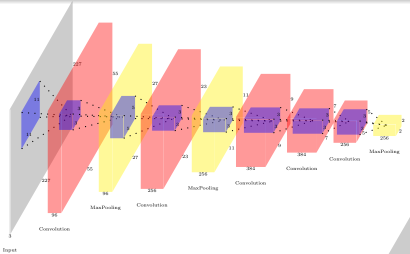


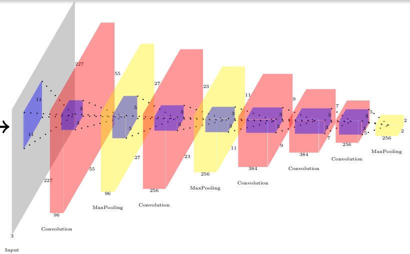
- To ensure that the style of the new image captured by layer ℓ matches the style of the style image, we can use the following objective function :

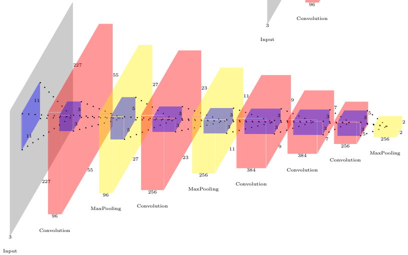
$$E_\ell = \sum_{ij} (G^\ell_j - A^\ell_{ij})^2$$

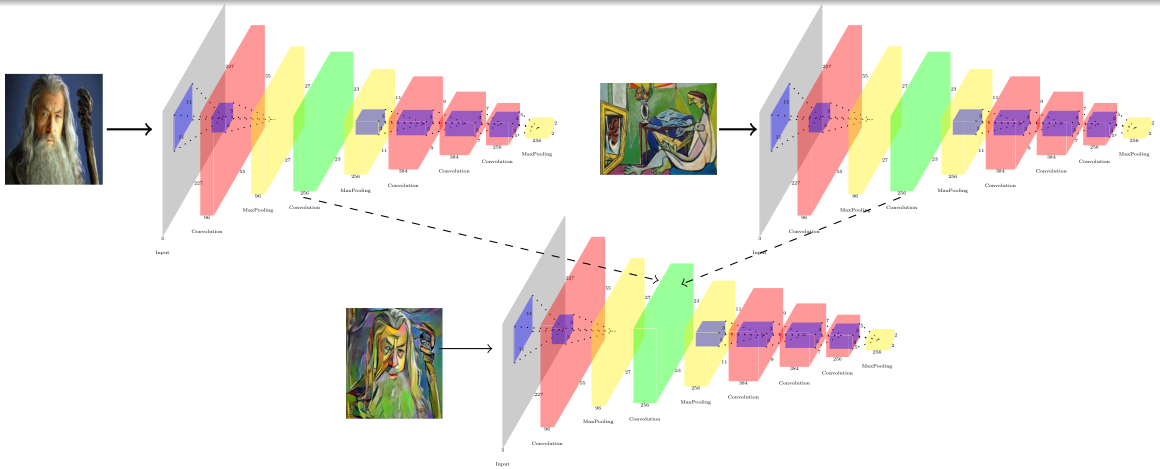
where G^ℓ and A^ℓ are the style gram matrices computed at layer ℓ for the style image and new image respectively.

$$\mathscr{L}_{style}(\vec{a}, \bar{x}) = \sum_{\ell=0}^{L} w_\ell E_\ell$$

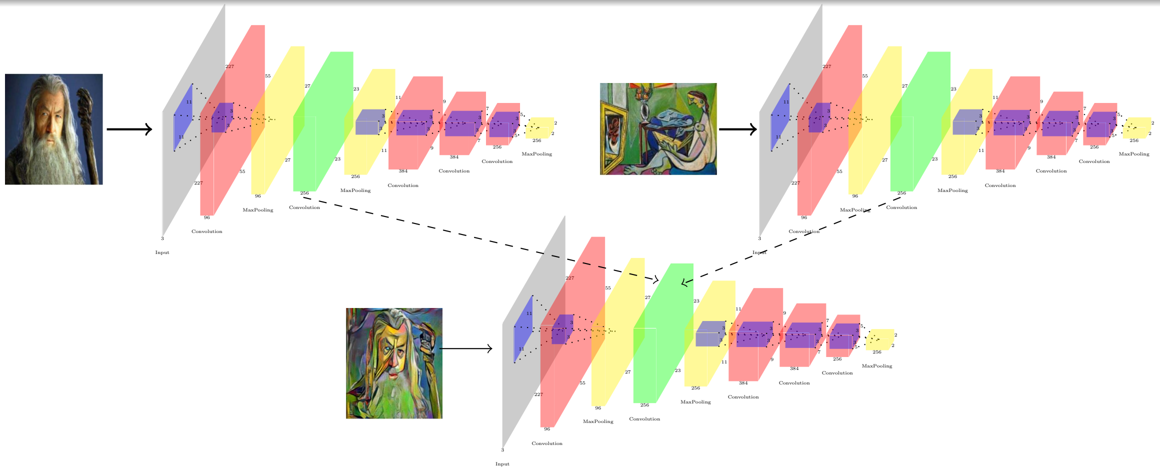

Input
Convolution
MaxPooling
Convolution
MaxPooling
Convolution
Convolution
Convolution
MaxPooling


Input
Convolution
MaxPooling
Convolution
MaxPooling
Convolution
Convolution
Convolution
MaxPooling


Input
Convolution
MaxPooling
Convolution
MaxPooling
Convolution
Convolution
Convolution
MaxPooling



Input
Convolution
MaxPooling
Convolution
MaxPooling
Convolution
Convolution
Convolution
MaxPooling



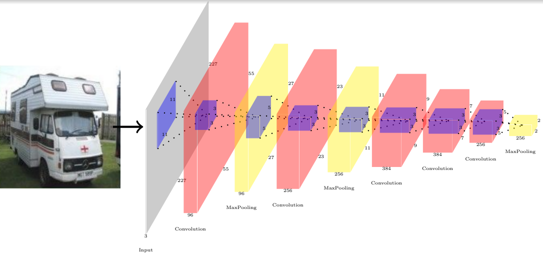
- The total loss is given by :-

$$\mathscr{L}_{total}(\vec{p}, \vec{a}, \vec{x}) = \alpha \mathscr{L}_{content}(\vec{p}, \vec{x}) + \beta \mathscr{L}_{style}(\vec{a}, \vec{x})$$

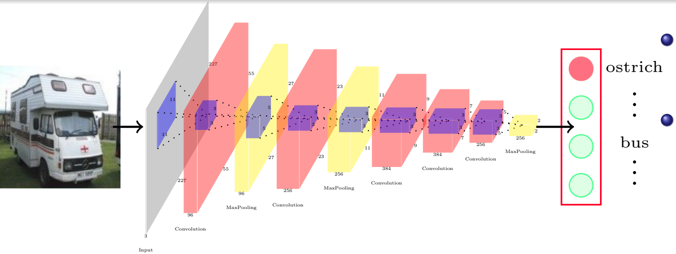
Module 13.10: Fooling Deep Convolution Neural Networks

- Turns out that using this idea of optimizing over the input, we can also "fool" ConvNets

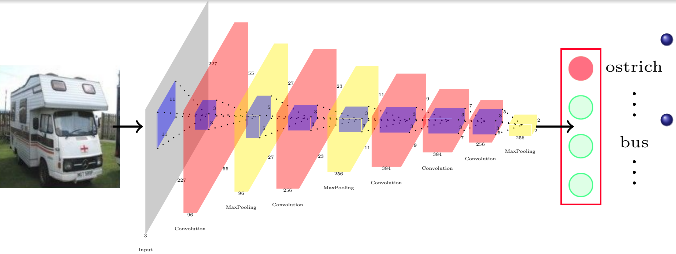
- Turns out that using this idea of optimizing over the input, we can also “fool” ConvNets
- Let us see how



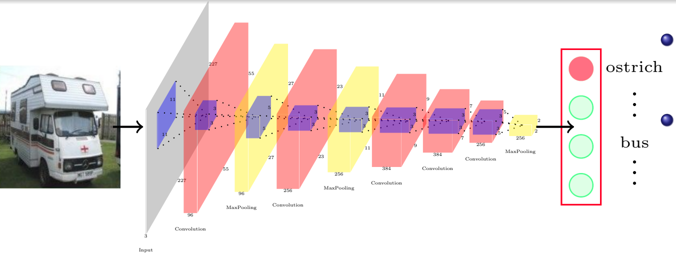
- Suppose we feed in an image to a Convnet.



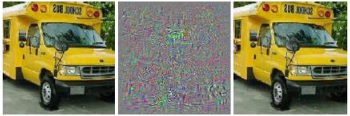
- Suppose we feed in an image to a Convnet.
- Now instead of maximizing the log-likelihood of the correct class (bus) we set the objective to maximize some incorrect class (say, ostrich)



- Suppose we feed in an image to a Convnet.
- Now instead of maximizing the log-likelihood of the correct class (bus) we set the objective to maximize some incorrect class (say, ostrich)
- Turns out that with minimal changes to the image (using backprop) we can soon convince the Convnet that this is an ostrich.

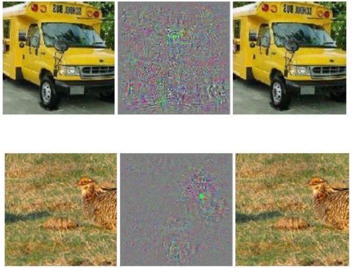


- Suppose we feed in an image to a Convnet.
- Now instead of maximizing the log-likelihood of the correct class (bus) we set the objective to maximize some incorrect class (say, ostrich)
- Turns out that with minimal changes to the image (using backprop) we can soon convince the Convnet that this is an ostrich.
- Let us see some examples



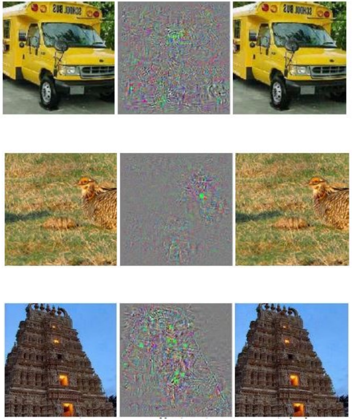
- Notice that the changes are so minimal that the two images are indistinguishable to humans

*Intriguing properties of neural networks, Szegedy et al., 2013



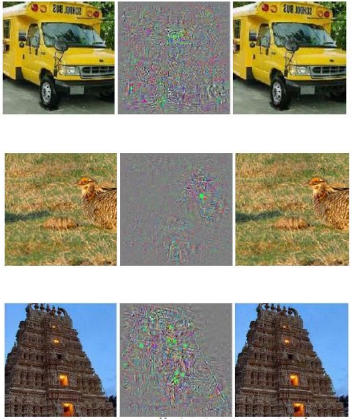
- Notice that the changes are so minimal that the two images are indistinguishable to humans
- But the ConvNet thinks that the third image obtained by adding the first image to the second image is an ostrich

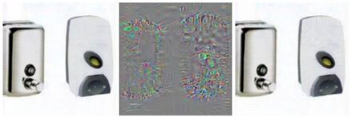
*Intriguing properties of neural networks, Szegedy et al., 2013



- Notice that the changes are so minimal that the two images are indistinguishable to humans
- But the ConvNet thinks that the third image obtained by adding the first image to the second image is an ostrich

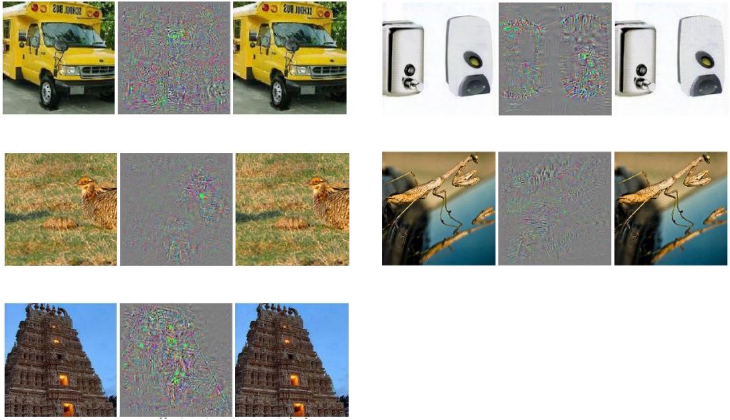
*Intriguing properties of neural networks, Szegedy et al., 2013





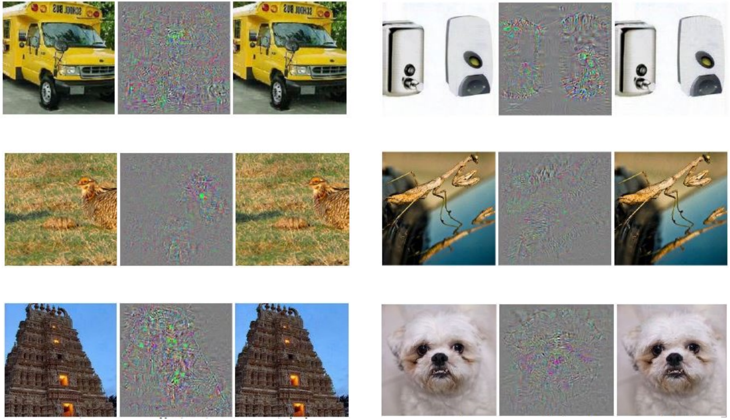
- Notice that the changes are so minimal that the two images are indistinguishable to humans
- But the ConvNet thinks that the third image obtained by adding the first image to the second image is an ostrich

*Intriguing properties of neural networks, Szegedy et al., 2013



- Notice that the changes are so minimal that the two images are indistinguishable to humans
- But the ConvNet thinks that the third image obtained by adding the first image to the second image is an ostrich

*Intriguing properties of neural networks, Szegedy et al., 2013

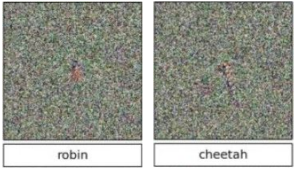


- Notice that the changes are so minimal that the two images are indistinguishable to humans
- But the ConvNet thinks that the third image obtained by adding the first image to the second image is an ostrich

*Intriguing properties of neural networks, Szegedy et al., 2013

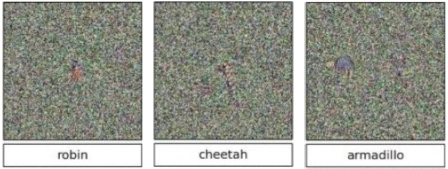
- We can also do this starting with random images and then optimizing them to predict some class.

*Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images Nguyen, Yosinski, Clune, 2014



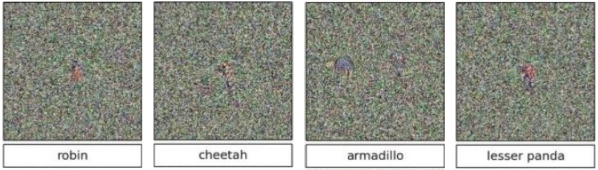
- We can also do this starting with random images and then optimizing them to predict some class.
- In all these cases the classifier is 99.6% confident of the class

*Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images Nguyen, Yosinski, Clune, 2014



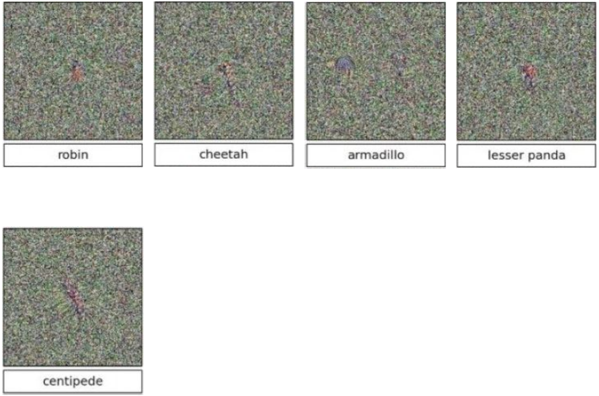
- We can also do this starting with random images and then optimizing them to predict some class.
- In all these cases the classifier is 99.6% confident of the class

*Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images Nguyen, Yosinski, Clune, 2014



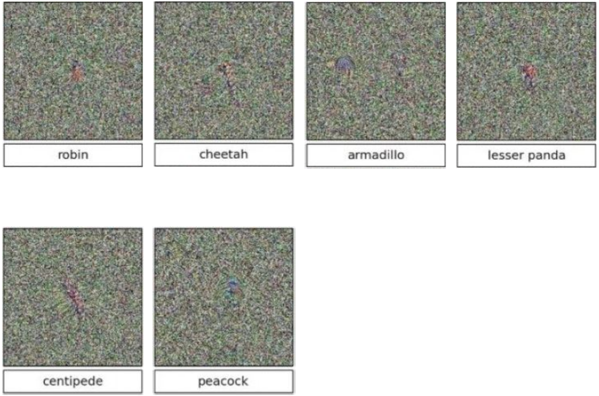
- We can also do this starting with random images and then optimizing them to predict some class.
- In all these cases the classifier is 99.6% confident of the class

*Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images Nguyen, Yosinski, Clune, 2014



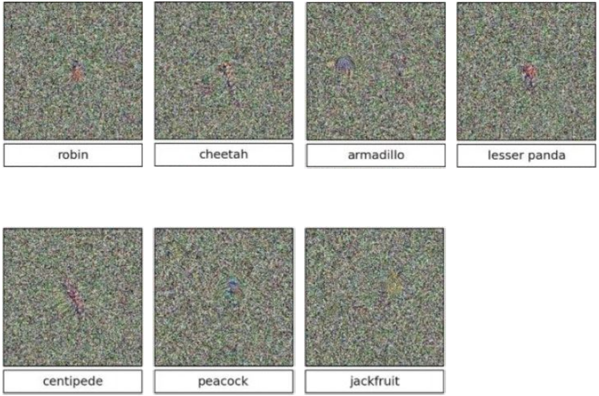
- We can also do this starting with random images and then optimizing them to predict some class.
- In all these cases the classifier is 99.6% confident of the class

*Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images Nguyen, Yosinski, Clune, 2014



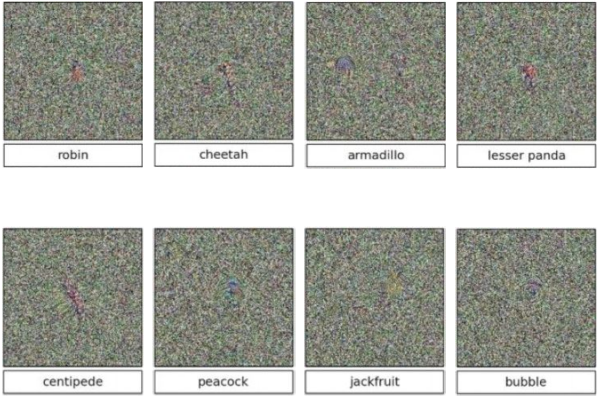
- We can also do this starting with random images and then optimizing them to predict some class.
- In all these cases the classifier is 99.6% confident of the class

*Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images Nguyen, Yosinski, Clune, 2014



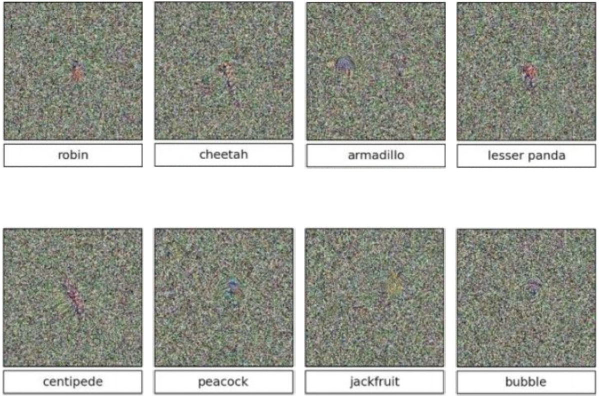
- We can also do this starting with random images and then optimizing them to predict some class.
- In all these cases the classifier is 99.6% confident of the class

*Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images Nguyen, Yosinski, Clune, 2014



- We can also do this starting with random images and then optimizing them to predict some class.
- In all these cases the classifier is 99.6% confident of the class

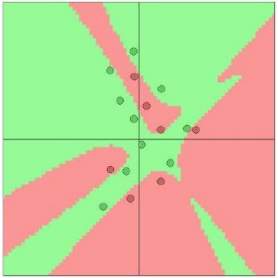
*Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images Nguyen, Yosinski, Clune, 2014



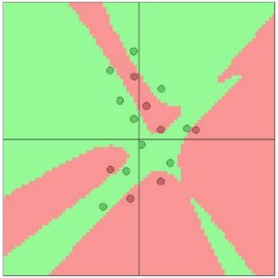
- We can also do this starting with random images and then optimizing them to predict some class.
- In all these cases the classifier is 99.6% confident of the class
- Let us see an intuitive explanation of why this happens

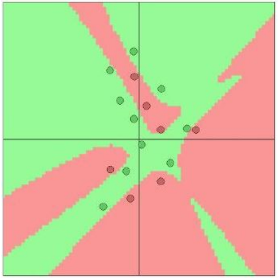
*Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images Nguyen, Yosinski, Clune, 2014

- Images are extremely high dimensional objects ($\mathcal{R}^{227 \times 227}$)

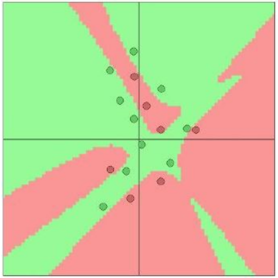


- Images are extremely high dimensional objects ($\mathcal{R}^{227\times227}$)
- There are many many many points in this high dimensional space

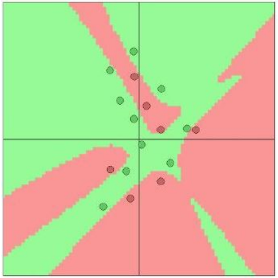




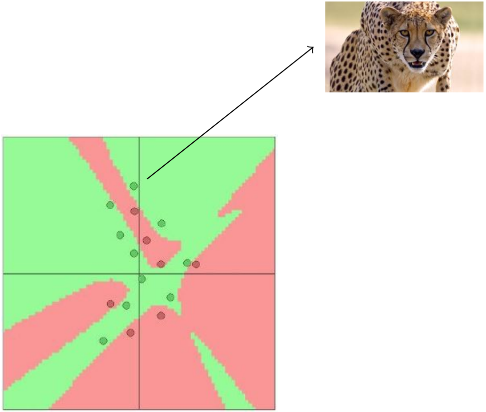
- Images are extremely high dimensional objects ($\mathcal{R}^{227 \times 227}$)
- There are many many many points in this high dimensional space
- Of these only a few are images (of which we see some during training)



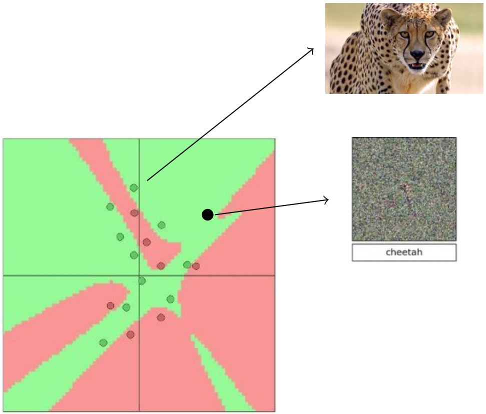
- Images are extremely high dimensional objects ($\mathcal{R}^{227\times227}$)
- There are many many many points in this high dimensional space
- Of these only a few are images (of which we see some during training)
- Using these training images we fit some decision boundaries



- Images are extremely high dimensional objects ($\mathcal{R}^{227\times227}$)
- There are many many many points in this high dimensional space
- Of these only a few are images (of which we see some during training)
- Using these training images we fit some decision boundaries
- While doing so we also end up taking decisions about the many many unseen points in this high dimensional space (Notice the large green and red regions which do not contain any training points)



- Images are extremely high dimensional objects ($\mathcal{R}^{227 \times 227}$)
- There are many many many points in this high dimensional space
- Of these only a few are images (of which we see some during training)
- Using these training images we fit some decision boundaries
- While doing so we also end up taking decisions about the many many unseen points in this high dimensional space (Notice the large green and red regions which do not contain any training points)



- Images are extremely high dimensional objects ($\mathcal{R}^{227\times227}$)
- There are many many many points in this high dimensional space
- Of these only a few are images (of which we see some during training)
- Using these training images we fit some decision boundaries
- While doing so we also end up taking decisions about the many many unseen points in this high dimensional space (Notice the large green and red regions which do not contain any training points)