Deep learning final project

Yana Hasson

March 5, 2017

1 Questions

1.1 top-5 vs top-1

The top-5 prediction includes the 5 highest probability classes and therefore allows for some additional flexibility in the result analysis compared to top-1. Using top-5 prediction emphasis the ability to assign a high probability to the correct class even if the highest prediction score leads to the wrong prediction.

This is also relevant when several similar classes are included (such as various breeds of a same species) as in the ImageNet dataset. In this case, a top-5 error rate would allow to not penalize the network for confusing similar breeds.

1.2 Data augmentation

Data corruption and data augmentation have the effect of artificially increasing the size of the training dataset. This can prevent **overfitting** by avoiding the network to overspecialize by training it on a more varied dataset and decreasing the number of epochs during training. This prevents the network from recognizing specific caracteristics of the samples that are not usefull for generalization, and can therefore prevent overfitting.

The object recognition task relies on images that are taken with a subjective framing. Therefore, a network should be robust to global translations of small magnitude of the picture, provided the relevant content of the image is still visible. Using data augmentation by adding translations and horizontal flips can therefore make the network more **robust** as it will encourage the network to focus on the visual features of the object rather then on its position.

By providing translated, noisy and horizontally flipped images, we encourage the model to be **invariant** to these transformations, as the network will have to learn to predict the correct class regardless.

1.3 Adagrad

Adagrad replaces the fixed learning rate in each gradient step by a value that depends both on the iteration step and on the historical gradient values.

The exact formula is : $\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{G_{t,i}+\epsilon} * g_{t,i}$.

In this formulat, t is the time (iteration number), $\theta_{t,i}$ is the i^th parameter's value at time t, $G_{t,i}$ is the sum of the squares of the gradients for the parameter θ_i up to time t and $g_{t,i}$ is the gradient value for parameter $\theta_{t,i}$ at time t, η is the initial learning rate which is shared among parameters, ϵ is chosen to be a small numerical value to allow for numerical stability.

Adagrad allows for an adaptative learning rate that is adapted to each parameter, and large updates for a parameter produces a reduction of the next parameters.

As $G_{t,i}$ increases over time, the update of the parameter is weighted by smaller values, this is both an advantage, as gradient values are used to update the parameter, the influence of additionnal is reduced, therefore replacing a manual tuning of a learning parameter.

By adapting to each parameter, the adagrad optimization allows for large parameter updates for parameters that seldom have important gradients. Convergence is improved as learning is boosted for those parameters without damaging the more often solicitated parameters. The learning is fastened as this avoids globally decreasing the learning rate which is necessarily non-optimal for the learning performances either of the parameters that often have high gradient values or the ones that tend to often have small gradients.

1.4 Sketch a Net - stroke ordering

The order of the strokes is incorporated in the following way :

- the strokes are broken up evenly into 3 chronological parts that sum up to the full image
- 3 additional parts are obtained by combining the chronologically contiguous layers (first part with the second, second with third, and final sketch composed of all three parts)
- these 6 parts of the image are entered in the neural network as separate channels

As mentionned in the article the stroke ordering information is interesting as it provides additionnal information to the network compared with the raw final image.

Generally, we start by drawing a general outline of the object and then focus on details. Extracting various snapshots of the strokes at various points in the drawing process would therefore often extract coarse to fine information about the sketched object.

One advantage of this basic approach for the segmentation is that it generates a fixed-channel number representation of the original image which can be fed to a fix input size neural network.

Still, some limitations are noticeable. As the arbitrary level of detail seems to be one of the main challenges of the sketch recognition task, we might consider that the ordering itself is not as important as the separation in the sketch of the general outline from the details. If the user adds a large quantity of details, with a uniform splitting based on the total number of strokes, some level of detail will be incorporated even in the first representation of the sketch. To cope with this issue we could for instance bias the first chronological separation to favour keeping only large strokes in the first part, which would more directly focus on capturing the outline rather then small details.

Also, even if we keep a fixed-size number of strokes in each part, we could experiment with various number of splits, for instance from 2 to 5, to see if additionnal granularity in the temporal ordering helps the net learn.

Furthermore, the fixed-size number of channels is a limitation, as some sketches with various levels of detail would probably benefit from an splitting that adapts to the variety in detail levels.

1.5 Sketch a Net - Scale fusion strategies

To capture information about the sketches' various scales, several networks are trained independantly on increasingly blurred versions of the sketches. The blurring is obtained by downsampling the image to a certain scale and then upsampling it back to the original size.

As it is mentionned in the article, a more basic fusion strategy would be to average the softmax scores and pick the class with the highest score, but this strategy does not take advantage of the knowledge we have of the various blur levels that were applied to each image. A simple way to take that knowledge into account could be to apply a weighted average that emphasizes for instance the classification porobility scores of the original image and takes less into account the blurred versions. This strategy would allow to take into account important probability scores from the blurred versions to bias the fine-grained prediction.

If we consider the last full connected layer's activations as a feature representative of the input for each network, other fusion strategies can be applied.

A possible fusion strategy presented is feeding a concatenation of those features (concatenation of the activations of the independent networks) to a classifier. But in this simple case, all the activations are weighted equally.

The method used in the article relies on a joint bayesian model which learns the intra ensemble and inter ensemble correlations. Once the metric is learned, a 5-nearest neighbor algorithm is applied to various crops and reflections of the input to find the closest training samples in the training dataset, and the by a majority voting, the class is assigned.

Other options would be to train the neural networks jointly by concatenating the outputs of the second fully connected layer together and adding several layers above the concatenation. We could for instance add a fully connected layer on top of the concatenation before a last fully connected layer which would have as many outputs as classes. We would therefore obtain a variant of a siamese network with 6 base networks instead of the usual 2.

2 Deep dream

2.1 Computation of the Jacobian

$$z = ||x^{l+1}||_2 = (x^{l+1})^T \cdot x^{l+1}$$

$$||x^{l+1} + h||_2 - ||x^{l+1}||_2 = (x^{l+1} + h)^T \cdot (x^{l+1} + h) - (x^{l+1})^T \cdot x^{l+1}$$

$$||x^{l+1} + h||_2 - ||x^{l+1}||_2 = (x^{l+1})^T \cdot h + h^T \cdot x^{l+1} + h^T \cdot h$$

$$||x^{l+1} + h||_2 - ||x^{l+1}||_2 = 2 * (x^{l+1})^T \cdot h + o(||h||_2)$$

Therefore we obtain the Jocobian :

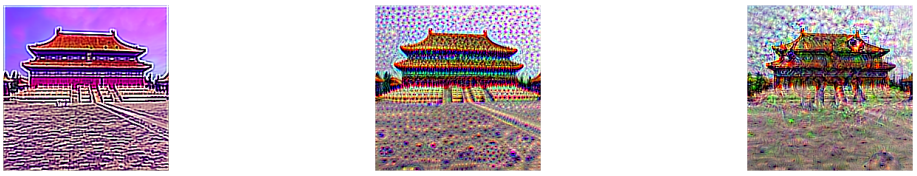
$$\frac{dz}{dx^{(l+1)}} = 2 * (x^{l+1})^T$$

2.2 Network analysis

The last five layers before the softmax one are fc6, relu6, fc7, relu7 and fc8. Before those layers all the layers are either convolutions with kernels of size 3x3 (and Relu non-linearity) or 2x2 max-pooling layers. In the final layers the localization information that was preserved by the convolutional layers is lost and the network can take advantage of the activations in various localizations of the image to make it's prediction.

The last layer is a **softmax** layer that takes the final 1000 activations and converts them to 1000 values between 0 and 1 that sum to 1. This allows to interpret the output of the softmax layer as a probability of the input being classified in each class.

2.3 First experiments



(a) step size : 0.0001
layer : 3
iteration : 9

(b) step size : 0.00001
layer : 15
iteration : 6

(c) step size : 0.001
layer : 27
iteration : 6

Figure 1: Various layers

Looking at 1, we can observe the impact of the layer choice on the granularity of the patterns that are enhanced by the networks.

For the 3rd layer, as we can see in 3a, small scale details are emphasized, especially the detail's borders.

For the 15th layer, we can see in 3b that larger scale patterns are produced.

For the choice of the 27th layer, the patterns produced have an even larger characteristic scale, as we can see in 3c.

Observations when we vary the step size

Varying the step size allows us to choose how big a step we take in the direction of the gradient to modify the initial image. Large step sizes will have the effect of changing the image more drastically.

Exagerately large step sizes could produce erratic variations of the l2 norm, as the information contained in the gradient is relevant on an infinitesimal scale.

We therefore have to find a compromise in the step size that allow us to enhance patterns that effectively increase the l2 norm of the response layer in a reasonable number of iterations.

Looking at figure 2 we see the impact of the variation of the step size. For larger values the increase of the response layer values occurs faster but less regularly.

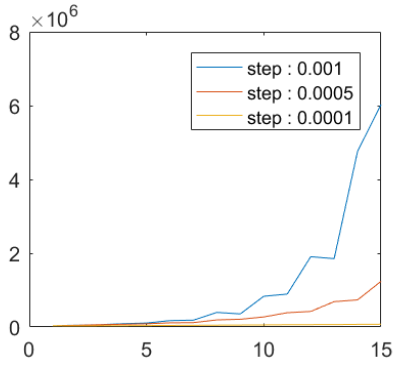


Figure 2: l2 norm of 27th layer for various step sizes

We can also evaluate the effect of varying the step size by looking at a fixed iteration result for varying step sizes. We display the results of this approach in figure 3. We can see that the feature emphasizing effect increases when the step size increase.

(a) step size : 1e-3
layer : 27

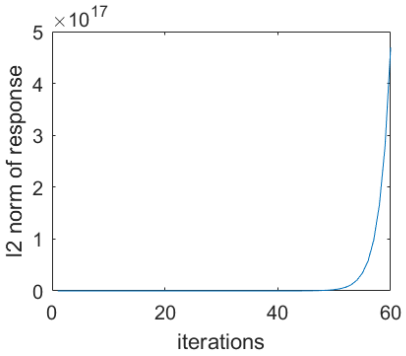
(b) step size : 5e-4
layer : 27

(c) step size : 1e-4
layer : 27

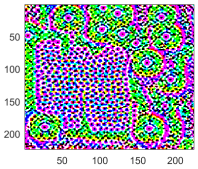
Figure 3: Results at 15th iteration

It is also noticeable that each layer needs a specific tuning of the step size. To avoid this layer-specific tuning, we adaptively tune the learning rate at each iteration by dividing the learning rate by the mean of the absolute values of the gradient. This normalization allows in most cases to keep a consistent learning rate accross layers.

The l2norm does not seem to reach a stable value, on the contrary, it tends to diverge as we can observe in figure 4a. In this case, we obtain compeletely saturated images such as the example 4b.



(a) L2 increase with iterations



(b) Image after 60 iterations

Figure 4: L2 divergence

2.4 Bigger input sizes

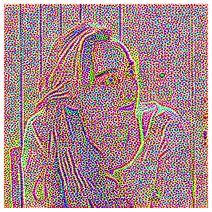
In this implementation of vgg the fully connected layers are convolutional layers with a kernel of size 1. Therefore, in the case when the input is larger then the expected input, we obtain a spatially dense output that correspond to various local subimages in the original image.

We can then backpropagate as before to optimize the input image.

In figure 6b we used the same code for the forward and backward step to perform the 'dreaming' on a n image of input size 512x512 instead of the original 224x224. We used the 15th layer, and therefore medium-scale patterns are generated in the original image.

(a) Original image

(b) step size : 1e-5
layer : 15
iteration : 6



(c) step size : 1e-5
layer : 15
iteration : 10

Figure 5: Optimization on larger input image

2.5 Multi scale experimentations

The first multi-scale experimentation we made was the to take 224x224 contiguous patches from the original image and process them in a cycle for a given layer response (in our case the 15th layer).

We can see in figure 6 that borders are visible at the limit between the patches, and that this effect intensifies as the iterations progress as no continuity condition is taken into account.

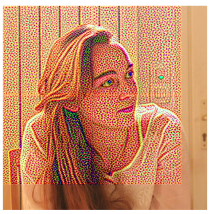
We can notice that part of the image is not updated.

The second experimentation we made was by for each update cycle changing the chosen layer and the corresponding step size.

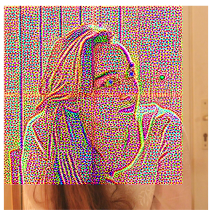
Each iteration corresponded to the following cycle :

- layer 3, step size : 1e-4
- layer 15, step size : 1e-5
- layer 27, step-size : 1e-3

For each cycle, various sizes of pattern are created, as we can see in figures 7a and 7.



(a) step size : 1e-5
layer : 15
iteration : 6



(b) step size : 1e-5
layer : 15
iteration : 9

Figure 6: 4 patches optimization

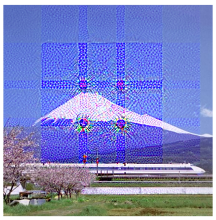
(a) iteration cycle : 1

(b) iteration cycle : 3

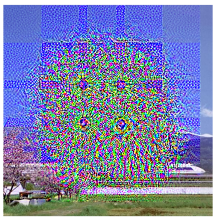
Figure 7: Multi-layer optimization

In our last experimentation, we allowed for some overlap during a given image update cycle (for a given layer response). More specifically, we allowed for a 124 pixels overlap. Over one image update for a given layers response, parts of the image are updated up to 3 times, which creates more visible enhancements in the parts of the image which are at intersections of many patch updates.

(a) Original image



(b) iteration cycle : 1



(c) iteration cycle : 2

Figure 8: Multi-layer optimization with overlap

2.6 Starting from pure noise

We start by generating a noisy image that has in each channel a random value between 0 and 255, we obtain 9.

We then optimize the original image for the l2 norm of the responses of the 3rd, 15th and 27th layer. The results are displayed in 10. As before, the size of the patterns increases with the index of the layer. We have purposefully let the algorithm do extra iterations that saturate the image and therefore make the size of the pattern very visible.

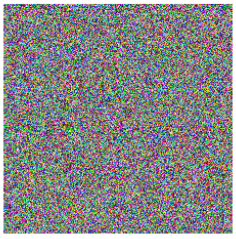
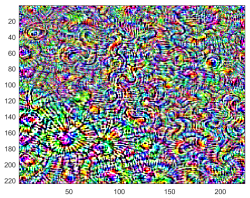
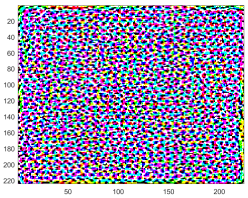


Figure 9: Initial noise



(a) step size : 1e-3
layer : 27
iteration : 14



(b) step size : 1e-5
layer : 15
iteration : 17

(c) step size : 1e-4
layer : 3
iteration : 10

Figure 10: Patterns created from noise

3 Experiment 1

In this experiment we try to dream images using the 'sketch-a-net' network.

More specifically, in this experiment I focused on the network that takes the image at full resolution and fed it 256*256 black and white images.

I preprocessed natural image in the following way :

- converted the image to grey scale using the matlab function rgb2gray
- resized to 256*256
- affine transformation to map the values to [-1,1]

During the 'dreaming' I tried the following regularization options :

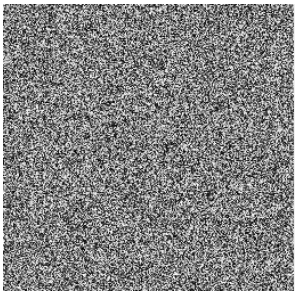
- scale the weights by a factor < 1 after each pass to avoid some pixels to take extreme values (strongly abose 1 or below -1)
- clamp between -1 and 1
- apply a gaussian blur at varying iteration intervals

I observed similar performances with the clamping and the scaling whereas I did not find a good combination of parameters for the gaussian blur to provide regularization without overly blurring the image.

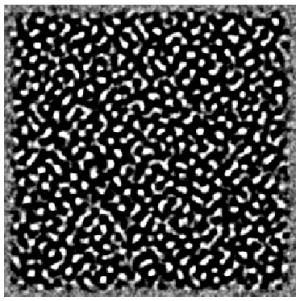
3.1 Starting from noise

We first work with random noise (uniformly sampled colors of gray). We display an example with clamping and scaling in 12 for maximization of the 3rd layer. We notice that the clamping emphasizes the sketching effect, but that overall in white noise, the sketching effect is not much put forward.

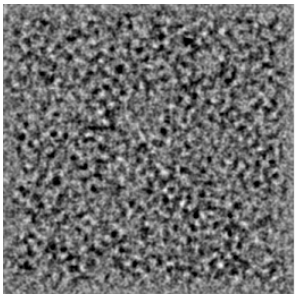
For early layer activation, the visualized features resemble gray versions of the colored ones obtained for vgg. On the other hand, the higher layer features seem to diverge quiet a bit as can be seen in 11.



(a) Original noise



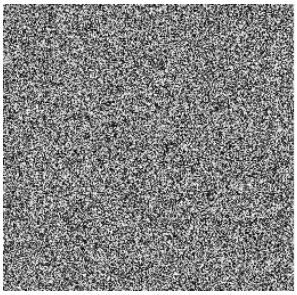
(b) Clamped
step size : 1e-2
layer : 3
iteration : 50



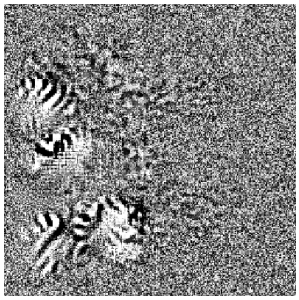
(c) Scaled
step size : 1e-1
layer : 3
iteration : 200

Figure 11: Patterns created from noise at layer 3

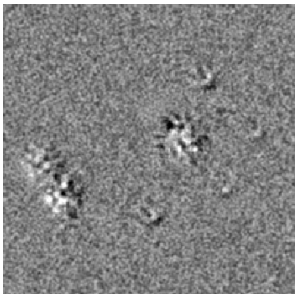
3.2 starting from noise



(a) Original noise



(b) Clamped
step size : 1e-2
layer : 20
iteration : 100



(c) Scaled
step size : 2e-3
layer : 20
iteration : 130

Figure 12: Patterns created from noise at layer 20

3.3 Natural images

We will now present some results we obtained with clamping for one of the natural images we experimented with. To allow for easy comparison we used the lion image that we experimented with earlier.

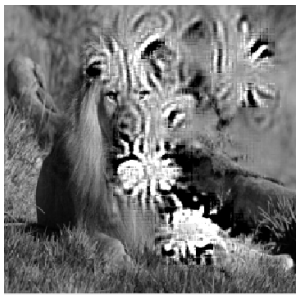
(a) Original lion

(b) Clamped
step size : 1e-2
layer : 20
iteration : 20

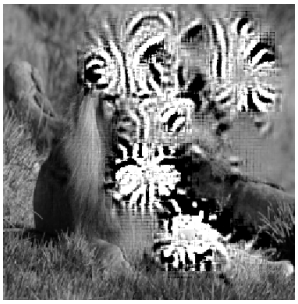
(c) Clamped
step size : 1e-2
layer : 20
iteration : 50

Figure 13: Patterns created from natural image at layer 20

(a) Original lion



(b) Clamped
step size : 1e-2
layer : 10
iteration : 20

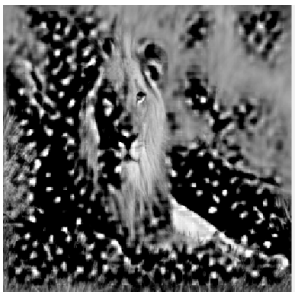


(c) Clamped
step size : 1e-2
layer : 10
iteration : 50

Figure 14: Patterns created from natural image at layer 10

(a) Original lion

(b) Clamped
step size : 1e-2
layer : 3
iteration : 20



(c) Clamped
step size : 1e-2
layer : 3
iteration : 50

Figure 15: Patterns created from natural image at layer 10

Looking at he emphasized features at the 10th layer presented in 18 we can clearly see that 'sketch-like' contours are visible.

Using scaling makes this effect less visible as we can see in 16. But we can still distinguish something that looks like sketch strokes in the features.

Figure 16: Scaled
step size : 1e-1
layer : 10
iteration : 100

4 Experiment 2

4.1 Neuron activation

In this case we decided to monitor the activation of the neuron that codes for a given class before softmax. We therefore used the 36th layer. We kept some gaussian blur to introduce some regularization, and occasionnaly also applied a l2 regularization to see if it improved the visual results.

We decided to maximaxe the square of the activation value of the neuron.

We tried to enhance the features both starting from noise and from a natural image representing mountains and sky.

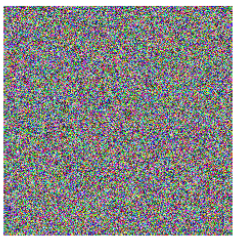
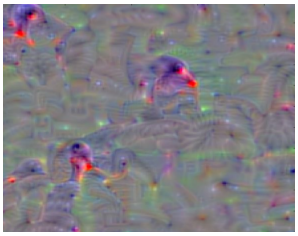
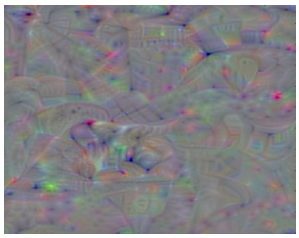


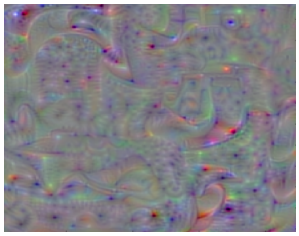
Figure 17: White noise



(a) Black swan
step size : 4
iteration : 160



(b) Ground beetle
step size : 4
iteration : 140



(c) Coffe mug
step size : 4
iteration : 140

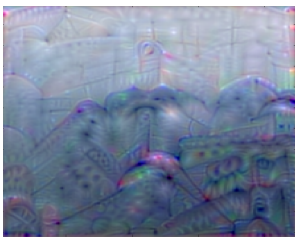
Figure 18: Patterns created from natural image at layer 10

Let's now look at some results starting from natural image 19.

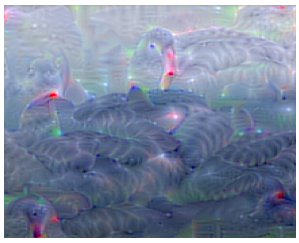
Figure 19: Natural image

It is not always easy to recognize the class by looking at the enhanced features, but looking at the results in 20, we can see that for different classes very different visual features are revealed. For the black-swan class, we can recognize black-swan's heads in some parts of the image, which are marked by a red beak.

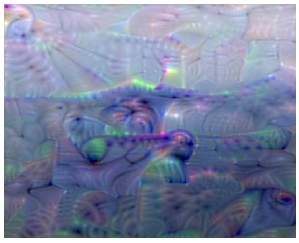
To emphasize the fact that the network indeed learns to recognize class-level features we present the dreamed features on the natural image and the first google image results for a class for which the dreamt features closely resemble the original class images. We display the results for the 650th



(a) Movie theatre
step size : 4
iteration : 95
blur step : 10



(b) Black swan
step size : 4
iteration : 100
blur step : 10



(c) Ground beetle
step size : 4
iteration : 200
blur step : 10

Figure 20: Patterns created from natural image at layer 10

(a) Megalith images from google

(b) Megalith (650)
step size : 1
iteration : 95
blur step : 10

Figure 21: Megalith dreaming

class of imagenet : megalith. We can see in 21 the relationship between the emphasized features and the characteristics of the class, as we can distinguish megalith-looking features in the dreamt image.

On the example presented in 22 we show that even for various breads of dogs, the learnt features are pretty different, which helps us understand how the network manages to do a good job for fine-grained classification. To help us visualize the learnt features we provide an example of image for the two classes of bread we examined : borzoi and boston terrier.

We can notice that this procedure effectively produces adverserial examples. We could monitor the classification score which should increase with the iterations.

4.2 Guided dreaming

We want to minimize the dot product for a given layer between the input of a guiding image and the activation of another image. To achieve this we operate gradient descent on this second image, we therefore use the activations as values for our gradient descent. (As taking the derivative of the dot product gives us the activation values of the guiding image)

To produce visually appealing results we use periodic gaussian blurring every 10 iterations.

We also use a normalized step size by dividing the step by the mean of the absolute gradient values.

We look at the results for various targeted layers.

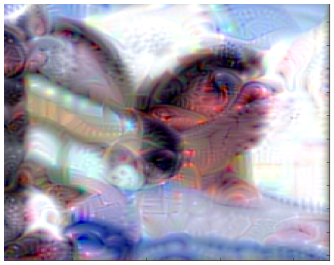
Looking at 23 several interesting observations can be made.

We can clearly see the effect of loss of localization introduced by the fully connected layers. When we guide for layers above the first connected layers we loose the localization information that we can clearly see is preserved for the consecutive convolutive layers.

We can also see the various scales associated with various convolutional layers. Maximizing the output of the first convolutional layer (3rd layer in the matconvnet implementation), we keep

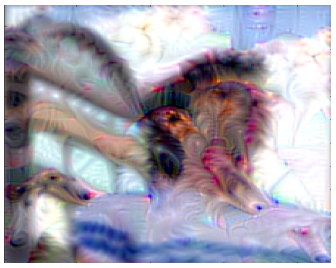
(a) Original portrait

(b) Boston terrier



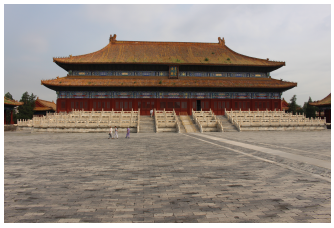
(c) Boston Terrier
step size : 1
iteration : 130
blur step : 10

(d) Borzoi



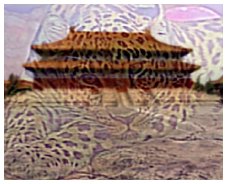
(e) Borzoi
step size : 1
iteration : 130
blur step : 10

Figure 22: Dreaming from portrait

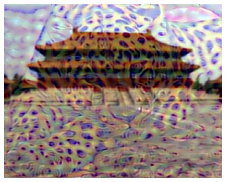


(a) Original image

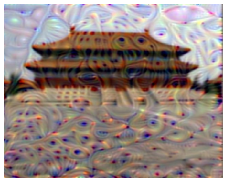
(b) Guiding image



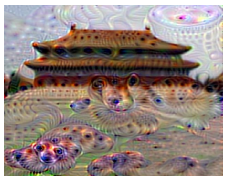
(c) layer 3
iteration : 20



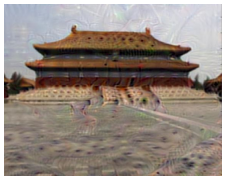
(d) layer 9
iteration : 20



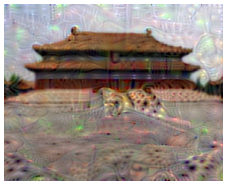
(e) layer : 17
iteration : 20



(f) layer : 25
iteration : 40



(g) layer : 32
iteration : 10



(h) layer : 36
iteration : 40

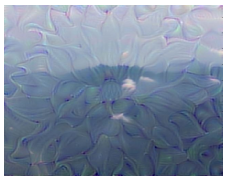
Figure 23: Guided dreaming

(a) Original image

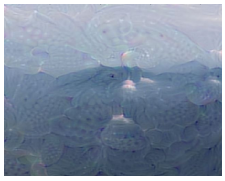
(b) Guiding image

(c) layer 3
iteration : 10

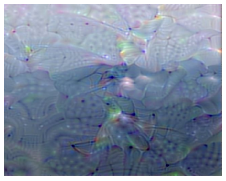
(d) layer 9
iteration : 10



(e) layer : 17
iteration : 15



(f) layer : 32
iteration : 15



(g) layer : 32
iteration : 45

Figure 24: Guided dreaming

mostly the edges and contours, which is consistent with our knowledge that early filters usually function as gabor filters or edge detectors.

As we move forward in the layers, we observe that more a general outline with blurrend edges is produced by the gradient descent.