Visualisation of maximising images for deep neural networks.

Abstract.

Deep neural networks have already proved their efficiency in solving various types of machine learning problems, especially related to recognising natural images. However, we still don’t have a deep understanding of how this networks work, especially in deep hidden layers. Developing methods of visualising neural networks would help to reveal what kind of features hidden layers have learned and analyse what each neuron is actually responsible for. There are two main approaches in visualising neural networks: deconvolutional and optimisation. The first one is often used for because of its’ high speed and low difficulty, but reconstructed images do not pretend to have high accuracy. The other one is quite accurate: it is formulated as an optimisation problem of maximising activity of the definite neuron but takes a lot of time to converge for deep network. We’ve tried to combine these two methods in order to have a possibility to visualise fast having quite high accuracy. We used regularisation based on neurons with specific activations to make images more interpretable.

1. Introduction.

Last years have produced a huge progress in creating deep neural networks models for solving various challenging machine learning tasks (). The leading and most popular example is training extremely deep convolutional neural networks for classifying natural images (). This breakthrough occurred thanks to developing GPU’s for faster computing of neural networks, new models of neurons (ReLUs ()), training tricks and algorithms (), soft for constructing neural networks faster and easily (Theano, Lasagne, Keras, Torch, Caffe, TensorFlow and etc.), collecting large labeled datasets ().

However, we don't still understand how deep networks work, particularly the hidden layers. Neurons visualisation is a way to reveal what features hidden neurons learn. Understanding of what each neuron is capable for can even provide better performance of the whole model. For example, deconvolutional approach helped to find out changes in small convolutional filters that led to the first place of the ImageNet competition (2013).

The easiest way to visualise neurons in is to simply look at the top-n dataset examples that result in the highest activity of every neuron (). Sometimes it is sufficient for defining what is neuron responsible for, but more often it is very difficult to generalise.

Another easy way to visualise neurons in deep convolutional networks is to directly plot an image of their activations (). It makes sense only for convolutions neural networks because of their possibility to save spatial structure and orientation of the data given to input. It is possible to plot such images online for several neurons in any convolutional layer because of low difficulty of this method. Then some of them could be found to be local and responsible for the definite kind of objects such as texts, human or animal faces, flowers, fruits and etc. This type of visualisation is easy to implement, but for the most part of neurons seems to be very difficult to interpret.

The other well-known way of visualising the activities of neurons is setting pixels’ intensity in some squared area (or even one pixel) to zero and tracking the difference in neurons’ activities () . While using this method, such square of zeros is slides on the picture so the spatial structure of the neurons activity can be recorded and then visualised. This method is quite slow and is used rarely, but it works for any kind of neural networks unlike the previous one.

One of the mostly used methods in visualising hidden layers of deep neural networks is deconvolutional approach. It was offered for neural networks that contain neurons with ReLU activation functions (), but this approach can be implemented for any types of neurons. . Plotting the image starts with finding in dataset the example that provides high activity to the neuron to be visualised. Then, the gradient of neuron's activity is calculated by passing the signal from the particular neuron (the array of 0 that contains 1 on the index of the neuron for the whole layer) backwards to the input layer. Some tricks to do it more efficient can be used for ReLUs (). This method is quite fast, but reconstructed images appear to be visually uninterpretable.

The most accurate way for visualising the neural network is the optimisation method. While plotting the images the optimisation problem is solving. The algorithm tries to maximise the activity of the definite neuron for a given set of neural network's parameters (weights and biases) by adjusting the intensity of every pixel of the input image. It starts from a fully black image with zero intensity for visualising any of neurons. Then, the image is reconstructed using gradient ascent. Pure optimisation method appears to be very noisy and uninterpretable, so different kinds

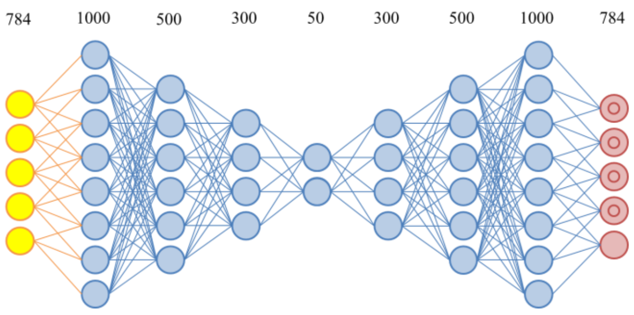

FIGURE 1 | The architecture of the stacked autoencoder. The first hidden layer has sparse connections (each neuron is connected with 7×7 square area of the input image), all other layers are fully connected. The network was constructed by sequential training of four autoencoders. Sparse autoencoder (784–1000–784) was trained at first, then values of the hidden layer were set as inputs to autoencoder (1000–500–1000). The same method was used for learning deeper layers (500–300–500, 300–50–300).

of regularisations are usually applied to the reconstructed images (). It provides the similarity between images in dataset and the maximising activity image.

In this work we've tried to combine the deconvolutional and optimisation methods to create the algorithm that can visualise precisely and fast. We used neurons with specific activations to adjust some hyperparameters of the algorithm and then applied it to non-specific neurons. Two different neural network models were used: RBM and autoencoder, both learned unsupervised. We describe the architecture of neural networks in Section 2. Neurons with specific activations are analysed n Section 3. Section 4 contains the description of the combined algorithm. Conclusions are presented is Section 5.

2. Architecture of the model

Two different models with the same deep architecture were trained. Networks' shape was chosen according to the model of ventral stream of visual cortex: V1, V2, V4 and IT areas. Neurons in the first area are known to detect small-size and low-level features like edges at different angles. Deeper we go into the brain, more complex features neurons are able to detect. Both of our models obtain this property by using sparse connections: neurons of the first hidden layer are connected with 7×7 square area of the input image, all other layers are fully connected.

Deep autoencoder was constructed by sequential training of four autoencoders (Fig [1]) with sigmoid activation function for each neuron. Sparse autoencoder 784–1000–784 was trained at first, and then the array of values of the hidden layer was set as inputs to autoencoder 1000–500–1000. The same method was used for learning deeper layers 500–300–500 and 300–50–300. Finally, stacked autoencoder was fine-tuned with a smaller learning rate. We notice that such greedy algorithm was used for training the deep network in order to increase the amount of neurons with the specific activation in deep layers. The comparison of the greedy algorithm and

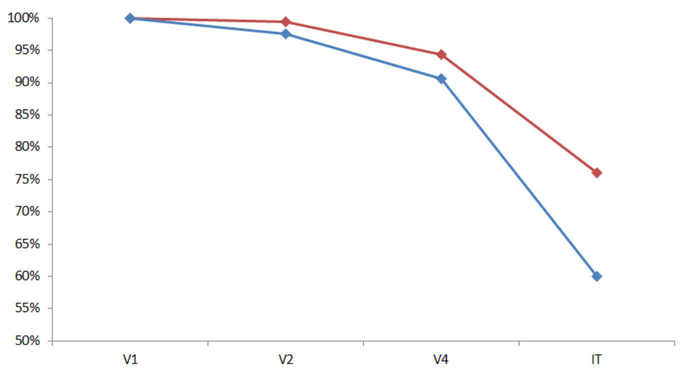

FIGURE 2 | Comparison of the greedy training algorithm and backpropogation. Parts of neurons with non-specific activation per each layer are shown for regular backpropogation through all deep network (red line) and greedy training algorithm (blue line). Layer-by-layer learning increases the number of neurons with specific activation from 12 up to 20. Three different models were trained during the research; best results for each approach are shown.

ordinary backpropogation algorithm is shown in the Fig. [2].

The same layer-by-layer approach was used for constructing deep RBM model (with the same architecture), using CD-1 rule for each layer. We used RBMs with binary probabilistic neurons, but on the last layer of the network we considered that output is not binary (we used probabilities of neurons as the output for further classification task). No extra fine-tuning was made for deep RBM model.

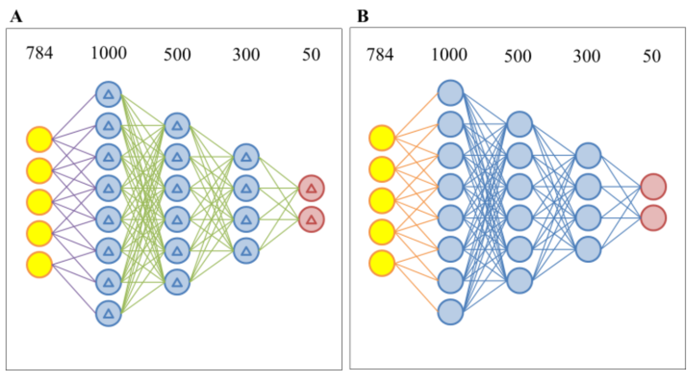

FIGURE 3 | Architectures of deep networks. (A) Deep RBM architecture with 7×7 sparse first hidden layer. All hidden neurons are probabilistic and binary. **(B)** Deep network based on stacked autoencoder (Fig [1]). Four last layers were pruned, because they don't take part in classification task and are necessary only for unsupervised

Finally, two models with the same architecture were chosen. Four last layers of deep autoencoder were pruned, because they are necessary only for training and don't take part in classification.

3. Neurons with specific activations.

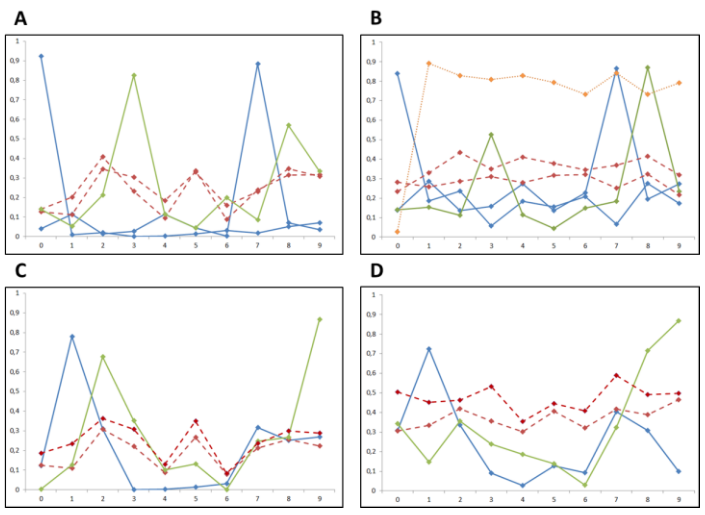

FIGURE 4 | Neurons with specific activations. The neuron is considered to have specific activation on the input signal of definite class, if its activation on test data with this label exceeds average value on more than 40%. The activities of unimodal (blue solid lines), bimodal (green solid line) and non-specific (red dash lines) neurons are shown for V4 and IT layers SAE (A, C) and DRBM (B, D).

For developing any new method of visualisation of neurons it is necessary to check if the reconstructed image is a real image of particular neuron. This is difficult to make decision if the image is good or bad for a random neuron, so we focused on neurons with specific activations on images with concrete labels, which images are quite understandable.

After the network had been trained, we observed neurons with specific activations by passing testing data through the network. We consider neuron to have specific activation on the input images of definite class, if its activation on input images with this class label exceeds average value on more than 40%. The activities of unimodal (blue solid lines), bimodal (green solid line) and non-specific (red dash lines) neurons are shown for V4 and IT layers SAE (A, C) and DRBM (B, D) in the Fig. [4].

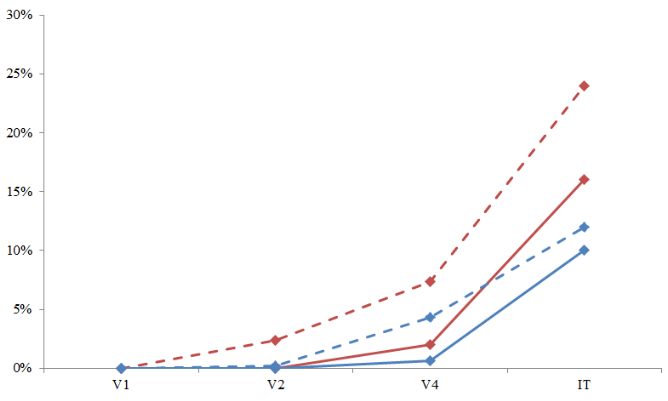

FIGURE 5 | Neurons with specific activation in different layers. Parts of the unimodal (solid lines) and bimodal (dash lines) neurons in the hidden layers of deep networks. Autoencoders (red lines) seem to have more neurons with specific activations on input images of the same class than RBM (blue lines). Four different SAE and DRBM were constructed; highest results are shown on the graph.

All outputs for neurons are in range from 0 to 1 (sigmoid function for autoencoder and probabilities for RBM). For example, we can see that in DRBM there are neurons that obtain significant average activation only for images of zero and seven, only eight or even all labels besides zero (orange line on Fig.[4B]) although the network was trained totally unsupervised.

Another question to answer is which of two unsupervised trained models is better for visualisation experiments, or, in another words, which of two networks has more neurons with specific activations in last layers. We trained four different DRBMs and four deep autoencoders (by using greedy algorithm) and found out that autoencoders seem to have approximately two times more specific neurons than DRBMs. Part of neurons with specific activations per each layer for both models is shown on Fig. [5].

4. Visualisation of the maximising images.

The basic idea of our approach of visualisation of neurons is a mix of deconvolutional and optimisation methods. Our aim was to find a way to visualise fast and quite precisely. Deconvolutional method is very fast because of starting to visualise from a definite image that provides the highest activity for the neuron to be visualised. The image reconstructed by deconvolutional algorithm is an image of gradients of the neurons activity while chosen image is given as an input (Fig[6A, 6B] column 2). Unlike deconvolutional approach, optimisation algorithm is doesn’t depend on the initial input image and starts from an empty image. It maximises the neurons activity by moving in direction of activity’s gradient step by step, as standard gradient descend does. In this case parameters of the model fixed, input image is fitted. Reconstructed by optimisation approach images for our models are shown on Fig[6A, 6B], column 3. Disadvantage of pure optimisation method is a huge amount of noise, so sometimes extra regularisation is used in function to be maximised.

We’ve tried to combine this two approaches and used an algorithm, that converges about 100 times faster than the pure optimisation algorithm and is robust to noise. We start from the definite image that provides high activity to the chosen neuron (as in deconvolution) and start to move in direction of the gradient for a fixed amount of steps (epochs). Finally, we pass each pixel of the image through the step-filter that returns 1 if pixel intensity is more than fixed threshold and 0 otherwise. Three hyperparameters are necessarily to be optimised: epochs (number of steps to take), learning rate and threshold. Learning rate was chosen by calculating the gradient’s norm and setting the step of the pixel intensity no more than 1%. Threshold was chosen to be 0.9 (all intensities were transformed to be in range from 0 to 1). Number of steps was evaluated using neurons with unimodal activations: *epochs* parameter was set as the average maximum step before any noise appears.

To write the algorithm in a compact form, let’s define for vector *x* matrix *A* mathematical operation:

$$x \otimes A = x_i \cdot A_{ij}$$

When hyperparameters are already evaluated, the algorithm can be described as follows:

```
Algorithm for calculating the maximising image:

1. Choose the input image x from the dataset, for which neuron l has the highest activity.

2. While (epoch < epochs):
        1. Compute the gradient of the activation of neuron.
           $\frac{\partial a_l}{\partial x_k} = \left( \prod_i^{N-1} \sigma'(y^{(i)}) \otimes W^{(i)} \right)_{lk}$
        2. Update the image x :
           $x_k = x_k + \varepsilon \cdot \frac{\partial a_l}{\partial x_k}$
           $epoch \mathrel{+}= 1$
        3. Update x using the step function for fixed threshold.
           $x_k = 1 \ if \ x_k > threshold \ else \ x_k = 0$
```

Algorithm with evaluated value of *epochs* can be applied to bimodal or non-specific neurons. Reconstructed images for bimodal neurons that both have high activity on label “9” for SAE (green line on Fig.[4C]) and DRBM (green line on Fig.[4D]) are shown on Fig.[6A] and Fig.[6B] relatively (row with label 9). First four rows (digits 3, 0, 7, 8) relate to the 4th layer, next three (digits 1, 4, 8) - to the 3d, next three (digits 6, 5, 2) - to the 2nd.

5. Conclusion.

We have introduced new method for dealing with problem of visualisation and interpreting neurons of hidden layers. It is a data-driven optimisation approach that can lead to quite interpretable maximising images. One of the advantages of this algorithm is observing neurons

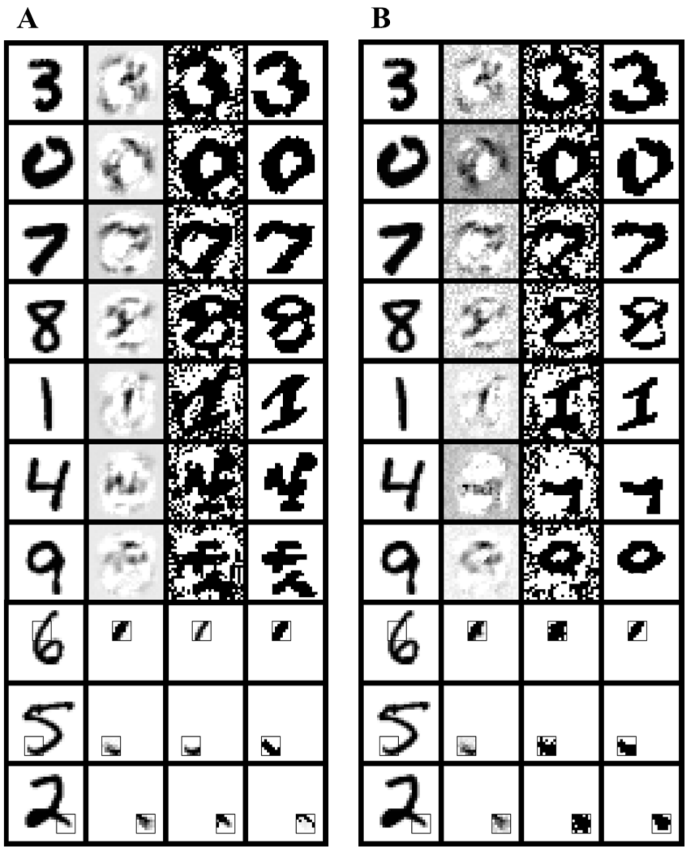

FIGURE 6 | . Visualisation of the images that activate neurons of V1, V2 and IT layers. The first column contains input images that are used for deconvolutional method of the gradients visualisation (the second column) and optimised visualisation (the fourth column). The third column contains images, computed by the optimisation method without regularisation. (A, B). Digits [3, 0, 7, 8] are visualised for the IT layer, [1, 4, 9] and [6, 5, 2] to the V2 and V1 layers relatively. Images for SAE neurons (A) and RBM neurons (B) are presented.

with specific activations, so we can further expect well-interpretable images for the found batch of neurons. Data-based character of our method leads to fast convergence compared with pure optimisation one.

We are planning to try it on more difficult datasets and deeper models in the future.