ETALON IMAGES: UNDERSTANDING THE CONVOLUTION NEURAL NETWORKS

Vladimir V. Molchanov[1], Boris V. Vishnyakov[1], Vladimir S. Gorbatsevich[1], Yury V. Vizilter[1]

[1] FGUP «State Research Institute of Aviation Systems», Russia, 125319, Moscow, Viktorenko street, 7 (vmolchanov, vishnyakov, gvs, viz)@gosniias.ru

Commission II, WG II/5

KEY WORDS: CNN, deep learning, manifold learning, affine transformations, graphs, etalons

ABSTRACT:

In this paper we propose a new technic called etalons, which allows us to interpret the way how convolution network makes its predictions. This mechanism is very similar to voting among different experts. Thereby CNN could be interpreted as a variety of experts, but it acts not like a sum or product of them, but rather represent a complicated hierarchy. We implement algorithm for etalon acquisition based on well-known properties of affine maps. We show that neural net has two high-level mechanisms of voting: first, based on attention to input image regions, specific to current input, and second, based on ignoring specific input regions. We also make an assumption that there is a connection between complexity of the underlying data manifold and the number of etalon images and their quality.

1. INTRODUCTION

For the last past years in computer vision society there were introduced tremendous variety of different neural networks architectures (Redmon et al., 2016), (Girshick, 2015), (He et al., 2016). There are different ideas behind these nets, which are explained by intuition and a good guess rather than strict theory, but the core blocks all of them are the same – they all utilize convolution and pooling layers. Such first CNNs as AlexNet and VGG, using only simple convolutions and pooling blocks without any additional connections (that exist in ResNet and DenseNet), haven’t shown such good quality, compared to the modern CNN architectures. But they have demonstrated the ability to fit data and to outperform previous state of the art algorithms. Given this, it is rather obvious that a good start point for deep neural network understanding is the first, simple CNNs.

To analyze CNN behavior, we select MNIST challenge and one of the common nets with good results on it (LeNet (LeCun et al., 1988), but we use its slight modification, replacing sigmoid activation units with ReLU). This net has simple consecutive convolution and pooling layers with ReLU activation functions. It also has 10 output neurons for final predictions (each neuron predicts class specific probability for input digit). The schematic network’s figure present in Fig. 1.

For explaining the notion of etalon it’s useful to consider CNN from functional point of view. According this interpretation CNN represent parametric family of functions which depends on net architecture and neuron’s activation functions. During training network parameters are tuned and as a result we get one instance form this family. For our particular network with ReLU activations if we don’t take into account the last SoftMax layer, then whole network represents the piecewise linear function in high dimensional input space. Each input image lies in some flat region where network behaves like affine transformation. This means that there is small neighborhood around any input image where CNN behaves like an affine transformation. Its size and topology depend on the complexity and form of the data manifold. As known CNN gives effective implicit representation of this manifold and etalons give way to look at it.

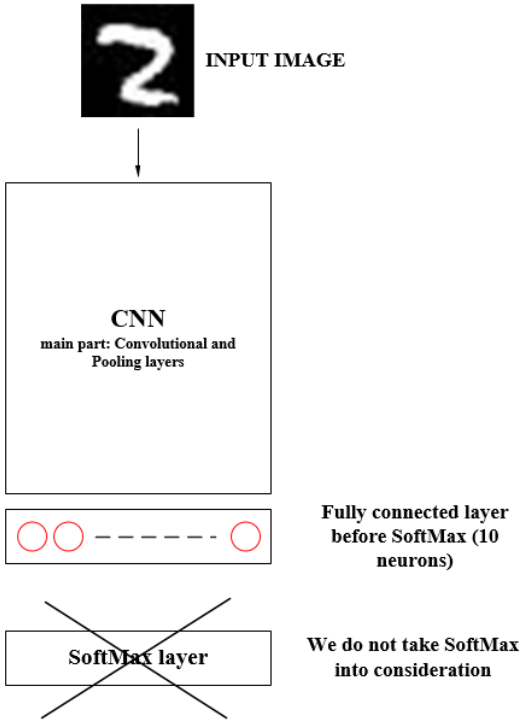

Figure 1. Schematic view of LeNet. We exclude SoftMax layer from further consideration

2. RELATED WORKS

For now, deep learning methods, especially based on deep neural networks, play an important role in information processing of visual data. At the heart of these models lies a hypothesis, that deep models can be exponentially more efficient at representing some functions than their shallow counterparts (Bengio, 2009). There is no strong theoretical justification, but there is a lot of practical experiments and promising results. It based on assumption that higher layers in a deep model can use features, constructed by the previous layers in order to build more complex functions and not to learn low level features again. For example, in CNNs for image classification (object detections and other vision tasks), first layer can learn Gabor filters that are capable to detect edges of different orientation. These edges are then put together at the second layer to form part-of-object shapes. On higher layers, these part-of-object shapes are combined further to obtain detectors for more complex part-of-object shapes or objects. Such a behavior is empirically illustrated, for instance, in (Zeiler and Fergus, 2013), (Lee et al. 2009). On the other hand, a shallow model has to construct detectors of target objects based only on the detectors learnt by the first layer.

There are also some theoretical justifications which prove the possibility for Deep Neural Nets to reconstruct functions with exponential number of regions. One example of such work is (Montufar et al., 2014). In this work authors investigate deep feed forward neural net with piecewise linear activation units. The intermediary layers of these models are able to map several pieces of their inputs into the same output. The layer-wise composition of the functions computed in this way re-uses low-level computations exponentially often as the number of layers increases. As a result, deep networks are able to identify an exponential number of input neighborhoods by mapping them to a common output of some intermediary hidden layer. The computations of the activations of this intermediary layer are replicated many times, once in each of the identified neighborhoods. This allows the networks to compute very complex looking functions even when they are defined with relatively few parameters. There are also some works dedicated to understanding how neural networks perceive images.

One well known method, described in work (Zeiler and Fergus, 2013), shows the way of getting regions in input image which are responsible for activation one or other neuron. In our work we try to investigate the other side of the problem. We concentrate on the mechanism which neural net obtains during training to perform its tasks (in this work we consider the classification task). We don't consider the problem of representation capability, however we suppose that there is a connection between complexity of the underlying data manifold and the number of etalon images and their quality. We guess that analyzing etalons can give some insight into problem of under and overfitting and models comparison based on last ones. Our work is similar to work (Zeiler and Fergus, 2013) in sense of visualization of regions, which neural network concentrates on.

However, etalon images aren't regions in input image. They are affine maps, defined on the input space and represents the behavior of the network on particular image.

3. METHODS

3.1 Image etalon definition

As was mentioned above, we consider CNN without last SoftMax layer. The output of such net is 10-dimensional vector with unnormalized class scores (one for each digit class). Let x be any input example – a vector of size m (for our consideration we flatten it's dimensions but one can take in mind that it is a grayscale picture with spatial sizes), NET – the function, represented by CNN that we describe earlier. According to introduction part, if we use ReLU as an activation function, for activated neurons we can write network transformation for vector X as:

$$NET(X) = Aff_X(X) = W_X^{NET} * X + b_X^{NET} \quad (1)$$

where Aff_X = affine transformation specific to X
W_X^{NET} = transformation matrix of the Aff_X
b_X^{NET} = bias of the Aff_X
$*$ = matrix product

This transformation take place between input space of images (m-dimensional) and output 10 dimensional space (class specific unnormalized scores), so W_X^{NET} is a $10 \times m$ matrix. Next we emphasize very simple fact which is important for further study: if we consider the whole net as an affine transformation, then we can treat any neuron as an affine map from input space to the real line $\mathbb{R}$. We show this for the output neuron from the last layer, but, as we will see further, this is also valid for all other neurons in any layer.

Let us look at one neuron p from the 10 output neurons and denote its output as $NET(X)_p$. For this neuron we can do all the steps above and write its affine map:

$$NET(X)_p = \langle (W_X)_p, X \rangle + (b_X)_p \quad (2)$$

where $(W_X)_p$ = transformation matrix specific to neuron p
$(b_X)_p$= bias term specific to neuron p like in (1)
$\langle \cdot \rangle$ = scalar product.

For output neurons there is obvious relationship between matrixes and biases terms in equations (2) and (1). It is easy to see that $(W_X)_p$ is just row p in matrix W_X^{NET} and also $(b_X)_p$ is p-th component in vector b_X^{NET}. Then we can write:

$$(W_X)_p = (W_X^{NET})_p, (b_X)_p = (b_X^{NET})_p \quad (3)$$

As was mentioned above such affine function could be defined for any neuron in CNN.

We call $(W_X)_p$ an etalon image for the input image X and the neuron p, because it has the same dimensions as the input vector

X. The work of the whole CNN can be presented as the scalar product between the image and its etalon. Despite the fact that each neuron has its own etalon for every image, effective number of etalons depends on the data manifold complexity and also on the training algorithm and network architecture. But if we suppose that network architecture isn’t too overabundant and train procedure is affective then first factor is dominant. Consequently, for more complicated dataset we shall have more etalons, and CNNs should have capacity to keep all of them, and that is exactly what they are good at, according to many practical and some theoretical results (Montufar, 2014). In the next paragraph we’ll give you more constructive view on etalons and it would be clear that CNN can keep the large number of them. That’s why etalons are another argument why CNN outperforms previous methods.

3.2 Etalon subgraph

In this paper we develop a method for getting the etalon from an input image. This method is based on another etalon interpretation like a subgraph in CNN. We can consider a neural net as graph, where neurons are vertexes and interlayer connections are edges. When a new image is passed to the network input, it goes successively through all layers. One can interpret this as some kind of flow, spreading through the network graph. But in contrast to classical flow, where there are sources and stokes, we don’t have stokes here, however this interpretation is helpful. There is also one important point, connected with ReLU activations: the flow spreads through only some subgraph with non-zero activations. When ReLU gives us zero activation, it does not affect all further calculations, so we can freeze edges between this neuron and successive layer. Summarizing all the above, each etalon can be associated with its own subgraph.

For better understanding how it works, let us consider the following example. Net structure is shown in the picture below (Fig.2). For simplicity we consider an example with two-dimensional input and two fully connected layers (activation functions are ReLU).

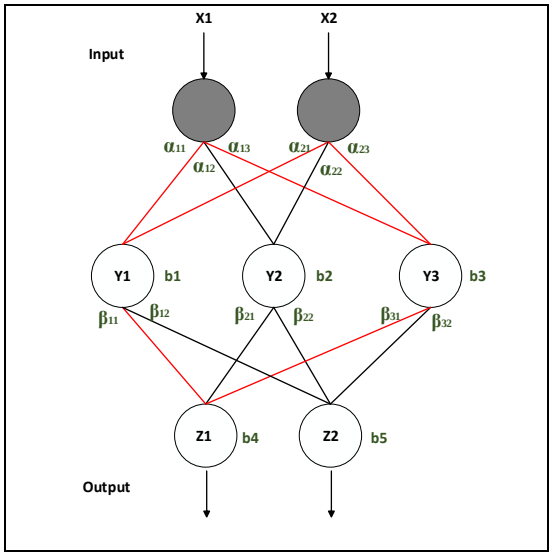

Figure 2. vector $\mathrm{X} = (\mathrm{X}_1, \mathrm{X}_2)$ is passed as an input to the CNN. We marked in red connections, that are active for a current input. One can see that connections with Y_2 are inactive, because $\mathrm{Y}_2 = 0$ ($\mathrm{ReLU}(\cdot) = 0$) and does not affect the final outcome.

Let us take the vector $X = (X_1, X_2)$ as an input and calculate the network output on it. Suppose that activation Y_2 argument is less than 0, so $Y_2 = \mathrm{ReLU}(\cdot) = 0$ and does not affect the result. We also assume that Z_2 argument is also less than 0. Then we have:

$$\begin{gathered} Y_1 = \mathrm{ReLU}(\alpha_{11}X_1 + \alpha_{21}X_2 + b_1) = \alpha_{11}X_1 + \alpha_{21}X_2 + b_1 \\ Y_2 = 0 \\ Y_3 = \mathrm{ReLU}(\alpha_{13}X_1 + \alpha_{23}X_2 + b_3) = \alpha_{13}X_1 + \alpha_{23}X_2 + b_3 \\ Z_1 = \mathrm{ReLU}(\beta_{11}Y_1 + \beta_{31}Y_3 + b_4) = \beta_{11}Y_1 + \beta_{31}Y_3 + b_4 \\ Z_2 = 0 \end{gathered} \quad (4)$$

where α_{ij} and β_{lk} = weights of the corresponding neurons,
b_s = biases of the neurons.

After putting Y_1, Y_2 expressions in the formula for Z_1 calculation, we have:

$$Z_1 = \left\langle \begin{pmatrix} \beta_{11}\alpha_{11} + \beta_{31}\alpha_{13} \\ \beta_{11}\alpha_{21} + \beta_{31}\alpha_{23} \end{pmatrix}, \begin{pmatrix} X_1 \\ X_2 \end{pmatrix} \right\rangle + (\beta_{11}b_1 + \beta_{31}b_3 + b_4) \quad (5)$$

where $(W_X)_{Z_1} = \begin{pmatrix} \beta_{11}\alpha_{11} + \beta_{31}\alpha_{13} \\ \beta_{11}\alpha_{21} + \beta_{31}\alpha_{23} \end{pmatrix}$ = matrix of the affine map,

$(b_X)_{Z_1} = \beta_{11}b_1 + \beta_{31}b_3 + b_4$ = bias of the affine map.

As a result we get the explicit view of the Z_1 neuron affine map for the given input, according to equation (2). One can see the relation between affine map and subgraph, which is obtained as a result of the input vector passing through the network. This graph only includes those neurons, that have non-zero activations and its highlighted with red connections in Fig. 2. It is rather obvious that all the results are valid for CNN because they are just specific type of fully connected networks with most of the connections equal to zero. We can conclude that one possible way of getting neuron etalons is through reconstructing the graph of its affine map. In next section we describe how to do this.

3.3 Etalon acquisition

The above results prove that if subgraph and input vector are known, then it is possible to reconstruct the etalon image. However, in arbitrary CNN it is rather difficult to get separate components of an etalon image altogether simultaneously. But it is possible to get these components separately. We know that network processes any image as scalar product between its etalon and image itself plus bias term. Using linearity of the scalar product, we can write down next statement for the input image X and any neuron p:

$$\begin{gathered} \langle (W_X)_p, X \rangle = (\omega_1, \omega_2, \dots, \omega_m)_p \cdot (X_1, X_2, \dots, X_m)^T = \\ \sum_{i=1}^{m} X_i \cdot (\omega_1, \omega_2, \dots, \omega_m)_p \cdot e_i \end{gathered} \quad (6)$$

where e_i = the basis coordinate vector (coordinate image) with component i, equal to 1, having all other components equal to 0,

$(W_X)_p = (\omega_1, \omega_2, \dots, \omega_m)_p$ is the etalon image for the neuron p of the input image X(we flatten etalon as input image).

From (6) we can conclude that if we feed coordinate vector e_i as input, then we get component ω_i of the etalon vector $(W_X)_p$. However we note that simply feeding this vector does not make sense because there is no guaranty that etalon subgraphs for image X and coordinate image e_i are coincide. But if we freeze all connections in the network except those, which belong to subgraph of the $(W_X)_p$ etalon and pass e_i through the network, we get reliable result.

It is obvious that for freezing we need to know etalon subgraph for image X. In principle it is not a problem, because we can track active connections in layers to form our graph structure and know the graph, then perform calculations for each coordinate image e_i using simple traverse of the graph. However, we decided to go the other way and freeze unused connections (those where ReLU equals to zero) on the flight. It is very easy – we can set activations of one particular neuron to zero to freeze its connections. Then they will not affect further computations, and it is equivalent to removing these connections.

3.4 Etalon reconstruction algorithm

In this section we propose an algorithm for etalon reconstruction, which outputs etalon image $(W_X)_p$ for the input image X and the neuron p. You can see the algorithm 1 steps in Fig. 3.

Algorithm 1. Etalon reconstruction for input image X and neuron p

Input: image X, network NET, neuron p

1. pass X to NET and remember non-zero activations
2. nullify biases of all neurons in NET
3. **for each** coordinate image e_i **do:**
   - 3.1 pass e_i to NET zeroing all activations which were stored in step 2
   - 3.2 save neuron p activation to i-th component of the $(W_X)_p$

Output: etalon image $(W_X)_p$

Figure 3. Etalon reconstruction algorithm for input image X and neuron p

Here we comment all steps in details.

- Algorithm 1 requires input image X, neuron p and network NET
- Algorithm 1 returns $(W_X)_p$ etalon image for X and p
- In step 1 we pass input image X to network NET and remember those activations that aren’t equal zero. We implement two layers – first ReLU, and second Pooling. ReLU layer – remembers those activations which are positive (active) in step 1 and during passing coordinate images e_i in step 3 this layer set to zero all activations except those that were stored. Pooling layer follows similar logic – it remembers neurons that were active (neurons with max values) in step 1, and in step 3 use activations of these neurons as outputs ignoring max pooling operation.
- In step 2 we nullify all biases in the network, because they influence the neuron activations and, as a result, we’ll get incorrect values for components of the $(W_X)_p$ (which would be equal to the sum of i-th component and bias term). However, from equations (2), (5) we see that affine transformation has two independent parts – one for linear transformation and other for translation (bias term). Consequently, we can zero biases for reconstruction etalon image.
- In step 3.1 we pass each coordinate image and according to our implementation of ReLU and Pooling layers they flow through required subgraph.
- In step 3.2 we take activation of the given neuron p which equals to i-th component of the etalon image

For better insight we visualize the work of algorithm 1 on a simple example of etalon reconstruction for Z_1 neuron. We also assume that Z_2 argument is also less than 0. For simplicity the net from Fig 2. was taken again, and the same proposals are suggested: it is a simple fully connected net with two layers and two-dimensional input $X = (X_1, X_2)$. Suppose that activation Y_2 argument is less than 0, so $Y_2 = \mathrm{ReLU}(\cdot) = 0$ and does not affect the result. According to step 1 input image $X = (X_1, X_2)$ is passed to the net, and we remember non-zero activations (see Fig. 4).

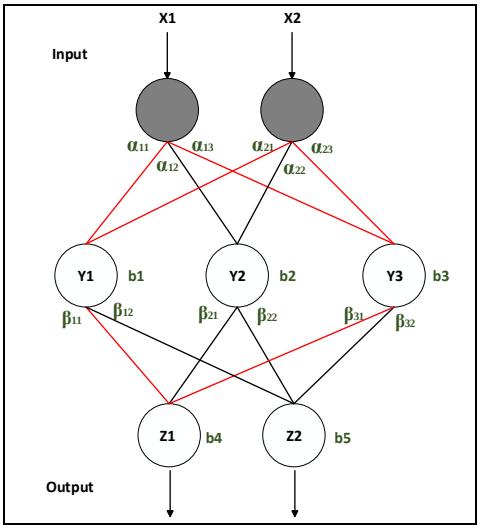

Figure 4. vector $\mathrm{X} = (\mathrm{X}_1, \mathrm{X}_2)$ is passed as an input to the CNN. We marked in red connections, that are active for a current input. Since neurons Y_2, Z_2 activations are zero, their connections are freezed.

Then in step 2 we zero all biases:

$$b_1 = b_2 = b_3 = b_4 = b_5 = 0 \qquad (7)$$

In steps 3, 3.1, 3.2 we pass all coordinate images e_i. For this example there are only two:

$$e_1 = (1,0), e_2 = (0,1) \qquad (8)$$

For e_1, according to system (4), we have the first component of Z_1 (see Fig. 5):

$$\begin{aligned} Y_1 &= \mathrm{ReLU}(\alpha_{11} * 1 + \alpha_{21} * 0) = \alpha_{11} \\ Y_3 &= \mathrm{ReLU}(\alpha_{13} * 1 + \alpha_{23} * 0) = \alpha_{13} \\ Z_{1_1} &= \mathrm{ReLU}(\beta_{11} Y_1 + \beta_{31} Y_3 + b_4) = \beta_{11} * \alpha_{11} + \beta_{31} * \alpha_{13} \end{aligned} \quad (9)$$

Comparing (5) with (9), we see that for e_1 coordinate image neuron Z_1 activation contains first component of the $(W_X)_{Z_1}$ etalon image.

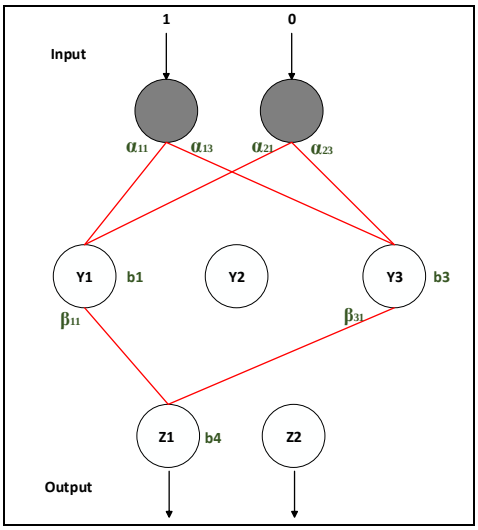

Figure 5. Feeding the first coordinate vector (1,0) to get the first component of the etalon $(W_X)_{Z_1}$. Components that were frizzed during step 1 of the algorithm 1 aren’t shown.

After feeding e_2 coordinate image we get next result for second component of the Z_1 (see Fig 6.):

$$Z_{1_2} = \beta_{11}\alpha_{21} + \beta_{31}\alpha_{23} \quad (10)$$

Summarizing equations (9) and (10), we can reconstruct etalon image $(W_X)_{Z_1}$:

$$(W_X)_{Z_1} = \begin{pmatrix} \beta_{11}\alpha_{11} + \beta_{31}\alpha_{13} \\ \beta_{11}\alpha_{21} + \beta_{31}\alpha_{23} \end{pmatrix}$$

In this algorithm we don’t reconstruct bias term in (2), however it is straightforward to do. Between steps 1 and 2 in algorithm 1 we can feed zero vector to the net, which gives us bias as activation value of the neuron p. Indeed, according to system (4), if vector $(0,0)$ is passed to the net, we have:

$$\begin{aligned} Y_1 &= \mathrm{ReLU}(\alpha_{11} * 0 + \alpha_{21} * 0 + b_1) = b_1 \\ Y_3 &= \mathrm{ReLU}(\alpha_{13} * 1 + \alpha_{23} * 1 + b_3) = b_3 \\ Z_1 &= \mathrm{ReLU}(\beta_{11} Y_1 + \beta_{31} Y_3 + b_4) = \beta_{11} b_1 + \beta_{31} b_3 + b_4 \end{aligned} \quad (11)$$

Comparing (5) and (11), we see neuron Z_1 activation contains bias term of the $(W_X)_{Z_1}$ etalon image affine map. But for etalon images this doesn’t make sense, so we ignore this step.

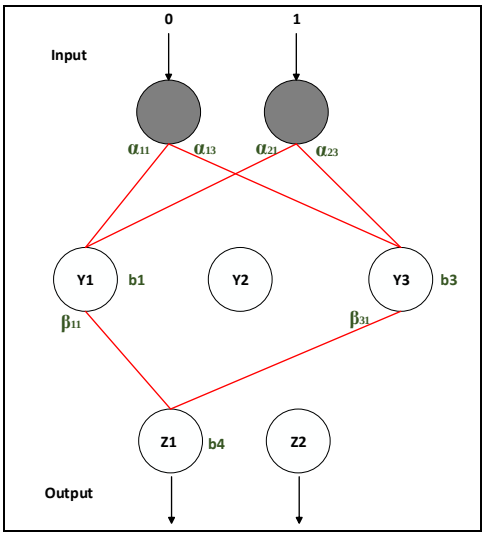

Figure 6. feeding the second coordinate vector (0,1) to get the second component of the etalon $(W_X)_{Z_1}$. Components that were frizzed during step 1 of the algorithm 1 aren’t shown.

3.5 Etalon reconstruction algorithm limitations

This algorithm is time consuming – for LeNet network with input image’s resolution $n \times n$ we need to make n^2 forward passes. Cutting subgraph from network and performing calculations with it can reduce computation cost because of reducing the number of active connections, but neural nets use effective implementation of their operations in CUDA. In addition, we also need perform n^2 forward passes. One possible solution is try to reconstruct etalon for one forward pass. It could be done by noting that convolution operation could be represented as linear mapping. ReLU activations after each mapping just nullifies some rows of transformation matrix of that mapping (concrete rows depends on input image). As a result, we can consider network as a product of affine mapping (different for each image). Multiplying all affine transformation, we’ll get another one which rows represent etalon images. However, representing convolution operation as matrix product requires a lot of memory. We didn’t investigate this way in practice.

4. EXPERIMENTAL RESULTS

4.1 LeNet etalon reconstruction

As was mentioned above, we choose LeNet for our experiments. We experimented with different images and show our results in Fig. 11. For experiments we concentrate on penultimate layer and visualize etalons for each neuron from this layer for different input images. As a result, for each image there are ten etalon images (one per each neuron). For better interpretation we consider several examples.

From example 1 we can see, that neural net tries to give attention to definite regions of input image and does it depending on the input. In the next example we show another mechanism, which is reversed to mentioned above.

In example 2 we demonstrate another mechanism, different from example 1. Here, our net is strongly sure that particular image doesn’t belong to given class, it simply votes against it, and again we see that it concentrates its attention on concrete regions.

Example 1: Input image class – 0 (Fig. 7.)

Figure 7

For this image here we visualize etalon images from several neurons (Fig. 8)

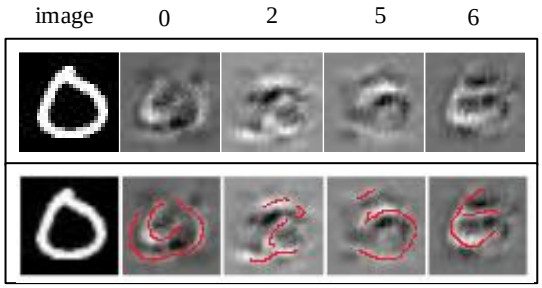

Figure 8. Etalon images for neurons 0, 2, 5, 6

Numbers in captions are neurons, responsible for corresponding image class. White spots in etalon images can be interpreted as most important regions in input image on which neural net concentrates its attention. And, vice versa, black spots are regions which net prefers to ignore. We highlighted with red those regions that neural net makes attention on. From neuron 0 etalon image one can conclude that net tries to concentrate its attention on places which are responsible for zero number. Also it should be emphasized that it is not a simple tracking of most possible regions, but it is specific for concrete input picture. There are many possible zeros in train set which are located near boundaries, but net puts some weights on positions, that are specific to concrete image. From the other hand, if we look at neuron etalon 5 image, then one can conclude that the net has some imagination. From this picture we can see how our net tries to propose possible look of number 5 in this image. We see that it tries to use bottom part of zero number in the picture to draw number five. On neurons 2 and 6 we see similar behavior. However, when this etalons images multiplied with input one it is easy to see, that biggest response is for zero etalon image.

Example 2: Input images classes – 4, 6 (Fig. 9.)

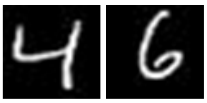

Figure 9

For this example, we take two images from classes 4, 6 and visualize etalon images for neurons 0 and 1 correspondently (Fig. 10).

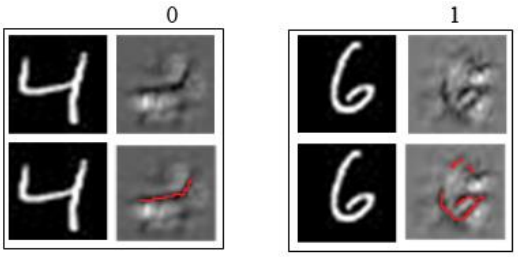

Figure 10.

We highlighted with red those regions which neural net tries to ignore. From neuron zero etalon image you can see, that net votes against this class. It puts very low weights on places where number four is located and, as a result, this class has very small score. The same situation we can see for etalon image of neuron one when number six is processed – neural net is completely sure that this picture can’t below to class one, so it votes against it.

5. CONCLUSIONS

In this work we represent a new notion – etalon images, which are defined as affine maps for any neuron and given input image. From the construction point of view etalons can be considered as subgraphs in the whole neural network graph. We implement algorithm for their acquisition based on well-known properties of the affine maps. Their analyses have shown that neural net has two high-level mechanisms of voting: first, based on attention to the input image regions and specific to current input, and second, based on ignoring specific input regions. We also suppose that there is a connection between complexity of the underlying data manifold and the number of etalon images and their quality. We guess that analyzing etalons can give some insight into problem of under and overfitting and models comparison. We concentrate on these problems in our future works.

6. ANKNOLEGMENTS

This work was supported by Russian Science Foundation (RSF) under Grant 16-11-00082; by Russian Foundation
For Basic Research (RFBR) under Grant 16-08-01260 A.

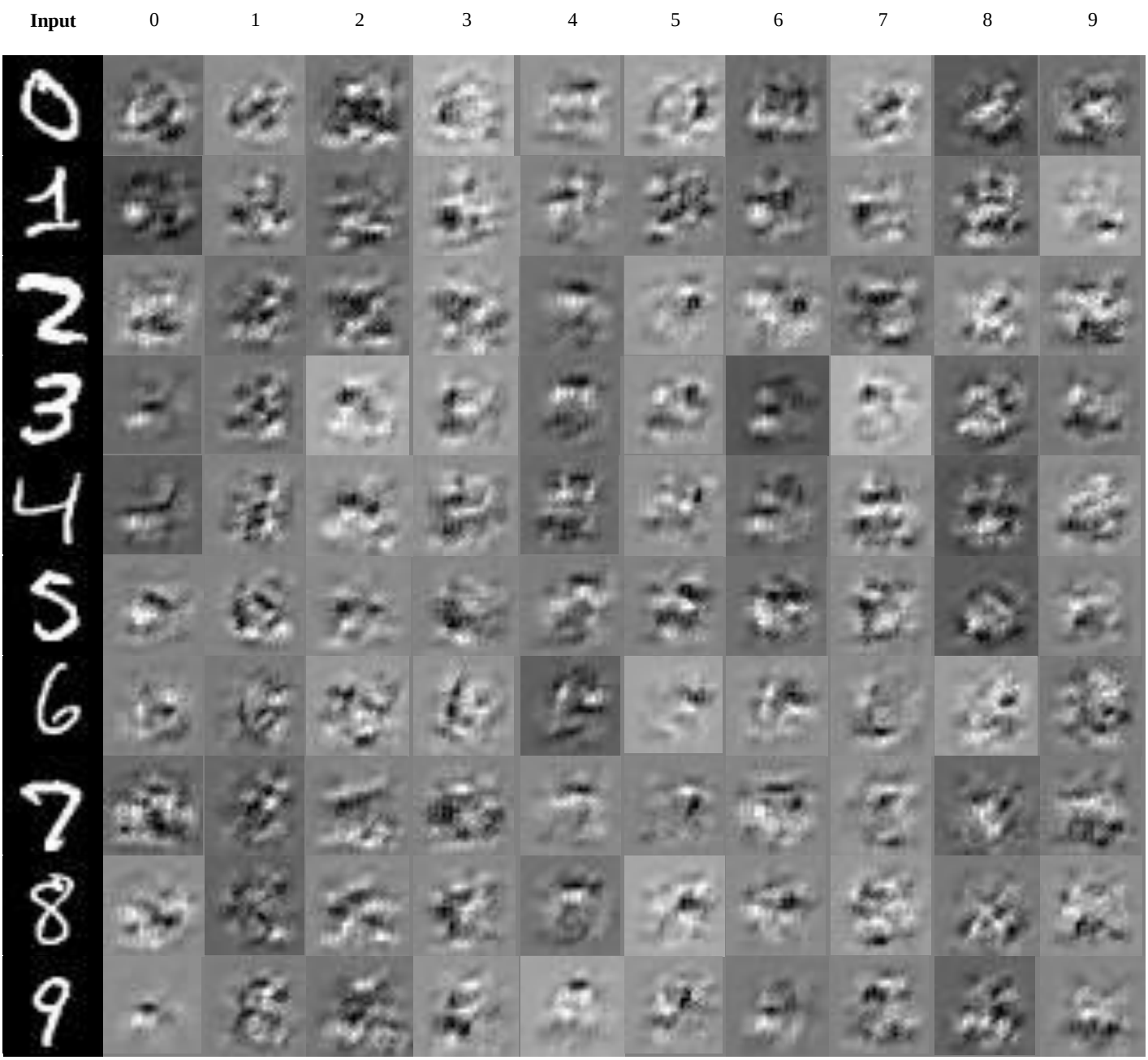

Figure 11. First column contains ten input images of the different classes from MNIST dataset. Rows from 1 to 10 contains their etalon images, row *i* contains etalon image of the corresponding neuron (each neuron has its own class for which it is responsible for). For example, neuron 8 for digit 8 has very clear etalon - one can see specific cross-figure of the digit 8.

7. REFERENCES

Bengio Y., 2009. Learning deep architectures for AI. *Foundations and trends in Machine Learning*, 2(1): pp.1-127.

Bing Xu, Naiyan Wang, Tianqi Chen, Mu Li., 2015. Empirical Evaluation of Rectified Activations in Convolutional Network. eprint arXiv:1505.00853

Girshick R., 2015. Fast R-CNN In: *2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 1440-1448.

He, K., Zhang, X., Ren, S., Sun, J., 2016. Deep residual learning for image recognition. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, NV, 2016, pp. 770-778. https://doi.org/10.1109/CVPR.2016.90

Lee H., Grosse R., Ranganath R., and A. Y. Ng., 2009. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In: *International Conference on Machine Learning*.

LeCun Y., Bottou L., Bengio Y., and Haffner P., 1998. Gradient-based learning applied to document recognition In: *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278-2324, Nov 1998. https://doi.org/10.1109/5.726791

Montúfar G., Pascanu K., Cho K., Bengio Y., 2014. On the Number of Linear Regions of Deep Neural Networks. In: *Proceeding NIPS'14*, Volume 2, pp. 2924-2932.

Redmon J., Divvala S., Girshick R., Farhadi A., 2016. You only look once: Unified, real-time object detection. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, NV, 2016, pp. 779-788.

Zeiler M.D., Fergus R., 2014. Visualizing and Understanding Convolutional Networks. In: *Fleet D., Pajdla T., Schiele B., Tuytelaars T. (eds) Computer Vision – ECCV 2014*. Lecture Notes in Computer Science, vol 8689. Springer, Zurich, pp. 818-833.