N^4-Fields: Neural Network Nearest Neighbor Fields for Image Transforms

Yaroslav Ganin Victor Lempitsky

Skolkovo Institute of Science and Technology (Skoltech)
{ganin,lempitsky}@skoltech.ru

Abstract. We propose a new architecture for difficult image processing operations, such as natural edge detection or thin object segmentation. The architecture is based on a simple combination of convolutional neural networks with the nearest neighbor search.
We focus our attention on the situations when the desired image transformation is too hard for a neural network to learn explicitly. We show that in such situations the use of the nearest neighbor search on top of the network output allows to improve the results considerably and to account for the underfitting effect during the neural network training. The approach is validated on three challenging benchmarks, where the performance of the proposed architecture matches or exceeds the state-of-the-art.

1 Introduction

Deep convolutional neural networks (CNNs) [1] have recently achieved a breakthrough in a variety of computer vision benchmarks and are attracting a very strong interest within the computer vision community. The most impressive results have been attained for image [2] or pixel [3] classification results. The key to these results was the sheer size of the trained CNNs and the power of modern GPU used to train those architectures.

In this work, we demonstrate that convolutional neural networks can achieve state-of-the-art results for sophisticated image processing tasks. The complexity of these tasks defies the straightforward application of CNNs, which perform reasonably well, but clearly below state-of-the-art. In particular, we show that by pairing convolutional networks with a simple non-parametric transform based on nearest-neighbor search state-of-the-art performance is achievable. This approach is evaluated on three challenging and competitive benchmarks (edge detection on Berkeley Segmentation dataset [4], edge detection on the NYU RGBD dataset [5], retina vessel segmentation on the DRIVE dataset [6]). All the results are obtained with the same meta-parameters, such as the configuration of a CNN, thus demonstrating the universality of the proposed approach.

The two approaches, namely convolutional **N**eural **N**etworks and **N**earest **N**eighbor search are applied sequentially and in a patch-by-patch manner, hence we call the architecture N^4-*fields*. At test time, an N^4-field first passes each

Fig. 1: N^4-Fields can be applied to a range of complex image processing tasks, such as natural edge detection (left) or vessel segmentation (right). The proposed architecture combines the convolutional neural networks with the nearest neighbor search and is generic. E.g. it achieves state-of-the-art performance on standard benchmarks for these two rather different applications with very little customization or parameter tuning.

patch through a CNN. For a given patch, the output of the first stage is a low-dimensional vector corresponding to the activations of the top layer in the CNN. At the second stage we use the nearest neighbor search within the CNN activations corresponding to patches sampled from the training data. Thus, we retrieve a patch with a known pixel-level annotation that has a similar CNN activation, and transfer its annotation to the output. By averaging the outputs of the overlapping patches, the transformation of the input image is obtained.

Below, we first review the related works (Section 2), describe the proposed architecture and the associated training procedures in detail (Section 3), and discuss the results of applying it on sample problems (Section 4). We conclude with a short discussion of the merits and the potential of the proposed approach (Section 5).

2 Related Work

There is a very large body of related approaches, as both neural networks and nearest neighbor methods have been used heavily as components within image processing systems. Here, we only review several works that are arguably most related to ours.

Neural Networks for Image Processing. The use of neural networks for image processing goes back for decades [7]. Several recent works have investigated large-scale training of deep architectures for complex edge detection and segmentation tasks. Thus, Mnih and Hinton [8] have used a cascade of two deep networks to segment roads in aerial images, while Shulz et al. [9] use CNNs to perform semantic segmentation on standard datasets. Kivinen et al. [10] proposed using unsupervised features extraction via deep belief net extension of mcRBM [11] followed by supervised neural net training for boundary prediction in natural images. State-of-the-art results on several semantic segmentation

datasets were obtained by Farabet et al. [12] by using a combination of a CNN classifier and superpixelization-based smoothing. Finally, a large body of work, e.g. [13, 3] simply frame the segmentation problem as patch classification, making generic CNN-based classification easily applicable and successful. Below, we compare N^4-fields against such baseline and find them to achieve better results for our applications.

Another series of works [14, 15] investigate the use of convolutional neural networks for image denoising. In this specific application, CNNs benefit greatly from virtually unlimited training data that can be synthesized, while the gap between synthetic and real data for this application is small.

Neural networks have also been applied for descriptor learning, which resembles the way they are used within N^4-fields. Thus, Chopra et al. [16] introduced a general scheme for learning CNNs that map input images to multi-dimensional descriptors, suitable among other things for nearest neighbor retrieval or similarity verification. The learning in that case is performed on a large set of pairs of matching images. N^4-fields are different from this group of the approaches in terms of their purpose (image processing) and the type of the training data (annotated images).

Non-parametric Approaches to Image Processing. Nearest neighbor methods have been applied to image processing with a considerable success. Most methods use nearest neighbor relations within the same image, e.g. Dabov et al. [17] for denoising or Criminisi et al. [18] for inpainting. More related to our work, Freeman et al. [19] match patches in a given image to a large dataset of patches from different images, to infer the missing high-frequencies and to achieve super-resolution. All these works use the patches themselves or their band-passed versions to perform the matching.

Another popular non-parametric framework to perform operations with patches are random forests. Our work was in many ways inspired by the recent impressive results in Dollár et al. [20], where random forests are trained on patches with structured annotations. Their emphasis is on natural edge detection, and their system represent the state-of-the-art for this task. N^4-fields match the accuracy of [20] for natural edge detection, and perform considerably better for the task of vessel segmentation in micrographs, thus demonstrating the ability to adapt to new domains.

3 N^4-Fields

3.1 Architecture

We start by introducing the notation, and discussing the way our architecture is applied to images. The N^4-Fields transform images patch-by-patch. Given an image transform application, we wish to map a single or multi-channel (e.g. RGB) image patch $\mathcal{P}$ of size $M \times M$ to a segmentation, an edge map, or some other semantically-meaningful annotation $\mathbf{A}(\mathcal{P})$, which in itself is a single or multi-channel image patch of size $N \times N$. We take N to be smaller than M,

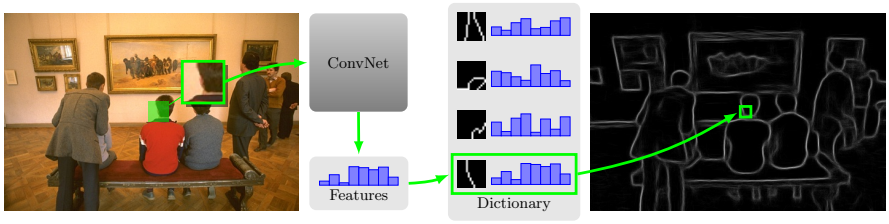


Fig. 2: The N^4 architecture for natural edge detection. The input image is processed patch-by-patch. An input patch is first passed through a pretrained convolutional neural network (CNN). Then, the output of the CNN is matched against the dictionary of sample CNN outputs that correspond to training patches with known annotations. The annotation corresponding to the nearest neighbor is transferred to the output. Overall, the output is obtained by averaging the overlapping transferred annotations.

so that $\mathbf{A}(\mathcal{P})$ represents a desired annotation for the central part of $\mathcal{P}$. For the simplicity of comparisons, in our experiments we use the sizes proposed in [20], in particular, $M = 34$ and $N = 16$.

Given the annotated data, we learn a mapping $\mathbf{F}$ that maps patches to the desired annotations. At test time, the mapping is applied to all image patches and their outputs are combined by averaging, thus resulting in an output image. The output of the processing for a pixel $p = (x, y)$ is the average of the outputs of N^2 patches that contain this pixel. More formally, the output of the mapping on the input image I is defined as:

$$\mathbf{F}(I)[x, y] = \frac{1}{N^2} \sum_{\substack{i,j:|i-x|\le N/2 \\ |j-y|\le N/2}} \mathbf{F}\left(I(i, j|M)\right)[x - i, y - i], \tag{1}$$

where $\mathbf{F}(I)[x, y]$ denotes the value of image transform at pixel (x, y), $I(i, j|M)$ denotes the image patch of size $M \times M$ centered at (i, j), and $\mathbf{F}\left(I(i, j|M)\right)[x - i, y - i]$ is a pixel in the output patch at the position $(x - i, y - j)$ assuming the origin in the center of the patch.

Obviously, the accuracy of the transform depends on the way the transform $\mathbf{F}$ is defined and learned. Convolutional neural networks (CNNs) provide a generic architecture for learning functions of the multi-channel images and patches exploiting the translational invariance properties of natural images. The direct approach is then to learn a mapping $\mathcal{P} \to \mathbf{A}(\mathcal{P})$ in the form of a CNN. In practice, we found the flexibility of CNNs to be insufficient to learn the corresponding mapping even when a large number of layers with large number of parameters are considered. For complex transforms, e.g. natural edge detection, we observe a strong underfitting during the training, which results in a suboptimal performance at test time.

Convolutional neural network can be regarded as a parametric model, albeit with a very large number of parameters. A straightforward way to increase the fitting capacity of the mapping is to consider a non-parametric model. We thus

combine a simple non-parametric mapping (nearest neighbor) and a complex parametric mapping (convolutional neural network). The input patch $\mathcal{P}$ is first mapped to an intermediate representation $\mathrm{CNN}(\mathcal{P};\Theta)$, where Θ denotes the parameters of the CNN. The output $\mathrm{CNN}(\mathcal{P};\Theta)$ of the CNN mapping (we call it a *neural code*) is then compared to a dictionary dataset of CNN outputs, computed for T patches $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_T$ taken from the training images, and thus having known annotations $\mathbf{A}(\mathcal{P}_1), \mathbf{A}(\mathcal{P}_2), \ldots, \mathbf{A}(\mathcal{P}_T)$. The input patch is then assigned the annotation from the dictionary patch with the closest CNN output, i.e. $\mathbf{A}(\mathcal{P}_k)$, where $k = \arg\min_{i=1}^{T} ||\mathrm{CNN}(\mathcal{P}_i) - \mathrm{CNN}(\mathcal{P})||$ (Figure 2). If we denote such nearest neighbor mapping as NNB, then the full two-stage mapping is defined as:

$$\mathbf{F}(\mathcal{P}) = \mathrm{NNB}\left(\mathrm{CNN}(\mathcal{P};\,\Theta) \,|\, \{(\mathrm{CNN}(\mathcal{P}_i;\Theta)\,;\,\mathbf{A}(\mathcal{P}_i)) \,|\, i = 1..T\}\right)\,, \qquad (2)$$

where $\mathrm{NNB}(x \,|\, M = \{(a_i|b_i)\})$ denotes the nearest-neighbor transform that maps x to the value b_i corresponding to the key a_i that is closest to x over the dataset M. In our experiments, the dimensionality of the intermediate representation (i.e. the space of CNN outputs) is rather low (16 dimensions), which makes nearest neighbor search reasonably easy.

In the experiments, we observe that such a two-stage architecture can successfully rectify the underfitting effect of the CNN and result in better generalization and overall transform quality compared to single stage architectures that include either CNN alone or nearest neighbor search on hand-crafted features alone.

3.2 Training

The training procedure for an N^4-field requires learning the parameters Θ of the convolutional neural network. Note, that the second stage (nearest neighbor mapping) does not require any training apart from sampling T patches from the training images.

The CNN training is performed in a standard supervised way on the patches drawn from the training images $I_1, I_2, \ldots I_R$. For that, we define the surrogate target output for each input patch. Since for each training patch $\mathcal{P}$, the desired annotation $\mathbf{A}(\mathcal{P})$ is known, it is natural to take this annotation itself as such a target (although other variants are possible as described in Section 3.3), i.e. to train the network on the input-output pairs of the form $(\mathcal{P}, \mathbf{A}(\mathcal{P}))$. However, such output can be rather high-dimensional (when the output patch size is large) and vary non-smoothly w.r.t. small translations and jitter, in particular when our model applications of edge detection or thin object segmentations are considered. To address both problems, we perform dimensionality reduction of the output annotations using PCA. Experimentally, we found that the target dimensionality can be taken rather small, e.g. 16 dimensions for 16×16 patches.

Thus, the overall training process includes the following steps:

1. Learn the PCA projection on a subset of $N \times N$ patches extracted from the training image annotations.

Fig. 3: The CNN architecture used in our experiments. See Section 3.3 for details.

2. Train the convolutional neural network on the input-output pairs $\{(\mathcal{P}, \mathrm{PCA}(\mathbf{A}(\mathcal{P}))\}$ sampled from the training images.
3. Construct a dictionary $\{(\mathrm{CNN}(\mathcal{P}_i; \Theta); \mathbf{A}(\mathcal{P}_i))|i = 1..T\}$ by drawing T random patches from the training images and passing them through the trained network.

After the training, the N^4-field can be applied to new images as discussed above.

3.3 Implementation Details

Training the CNN. We use the heavily modified cuda-convnet CNN toolbox[1]. The CNN architecture that was used in our experiments is loosely inspired by [2] (it comprises the layers shown in Figure 3). We also tried a dozen of other CNN designs (deeper ones and wider ones) but the performance always stayed roughly the same, which suggests that our system is somewhat insensitive to the choice of the architecture given the sufficient number of free parameters.

The model was trained on 34×34 patches extracted at randomly sampled locations of the training images. Each patch is preprocessed by subtracting the per-channel mean (across all images). Those patches are packed into mini-batches of size 128 (due to the software/hardware restrictions) and presented to the network. The initial weights in the CNN are drawn from Gaussian distribution with zero mean and $\sigma = 10^{-2}$. They are then updated using stochastic gradient descent with momentum set to 0.9. The starting learning rate η is set to 10^{-1} (below in Section 4 we introduce an alternative target function which demands smaller initial $\eta = 10^{-3}$). As commonly done, we anneal η throughout training when the validation error reaches its plateau.

As the amount of the training data was rather limited, we observed overfitting (validation error increasing, while training error decreasing) alongside with underfitting (training error staying high). To reduce overfitting, we enrich the training set with various artificial transformations of input patches such as random rotations and horizontal reflections. Those transformations are computed on-the-fly during the training procedure (new batches are prepared in parallel with the network training).

Along with data augmentation we apply two regularization techniques which have become quite common for CNNs, namely dropout [21] (we randomly discard half of activations in the first two fully-connected layers) and ℓ_2-norm restriction of the filters in the first layer [21, 22].

[1] https://code.google.com/p/cuda-convnet/

Testing Procedure. At test time we want to calculate activations for patches centered at all possible locations within input images. A naive approach would be to apply a CNN in the sliding window fashion (separate invocation for each location). However this solution may be computationally expensive especially in case of deep architectures. Luckily it is rather easy to avoid redundant calculations and to make dense applications efficient by feeding the network with a sequence of shifted test images [23].

After neural codes for all patches are computed, nearest-neighbors search is done by means of k-d trees provided as a part of VLFeat package [24]. We use default settings except for maximum number of comparisons which we set to 30.

Our proof-of-concept implementation runs reasonably fast taking about 6 seconds to process an image of size 480×320, although we were not focusing on speed. Computational performance may be brought closer to the real-time by, for example, applying the system in a strided fashion [20] and/or finding a simpler design for the CNN.

Multi-scale Operation. Following the works [20, 25] we apply our scheme at different scales. For each input image we combine detections produced for original, half and double resolutions to get the final output. While various blending strategies may be employed, in our case even simple averaging gave remarkably good results.

Committee of N^4-Fields. CNNs are shown [2, 23, 3] to perform better if outputs of multiple models are averaged. We found that this technique works quite well for our system too. One rationale would be that different instances of the neural network produce slightly different neural codes hence nearest-neighbor search may return different annotation patches for the same input patch. In practice we observe that averaging amplifies relevant edges and smooths the noisy regions. The latter is especially important for the natural edge detection benchmarks, as the output of N^4-fields is passed through the non-maximum suppression.

4 Experiments

We evaluate our approach on three datasets. Within two of them (BSDS500 and NYU RGBD), the processing task is to detect natural edges, and in the remaining case (DRIVE) the task is to segment thin vessels in retinal micrographs. Across the datasets, we provide comparison with baseline methods, with the state-of-the-art on those datasets, illustrate the operation of the method, and demonstrate characteristic results.

CNN Baselines. All three tasks correspond to binary labeling of pixels in the input photographs (boundary/not boundary, vessel/no vessel). It is therefore natural to compare our approach to CNNs that directly predict pixel labels. Given the input patch, a CNN can produce a decision either for the single central pixel or for multiple pixels (e.g. a central patch of size 16×16) hence we have two *CNN baselines*. We call them *CNN-central* and *CNN-patch* respectively. Each of

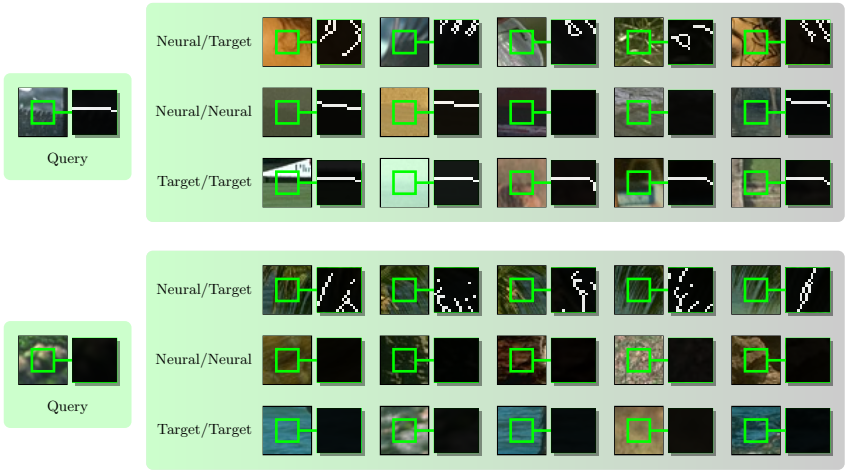


Fig. 4: Examples of nearest neighbor matchings of query patches to dictionary patches. For all patches, the ground truth annotations (edge sets) of the central parts are shown alongside. The righthand panels show the results of the nearest neighbor searches for different combinations of the query encoding and the dictionary patch encoding. *"Neural"* corresponds to the encoding with top-layer activations $\mathrm{CNN}(\mathcal{P})$ of the CNN, while *"Target"* corresponds to the "ground truth"' encoding $\mathrm{PCA}(\mathbf{A}(\mathcal{P}))$ that the CNN is being trained to replicate. Matching neural codes to target codes (Neural/Target) works poorly thus highlighting the gap between the neural codes and the target PCA codes (which is the manifestation of the underfitting during the CNN training). By using neural codes for both the queries and the dictionary patches, our approach is able to overcome such underfitting and to match many patches to correct annotations (see Neural/Neural matching).

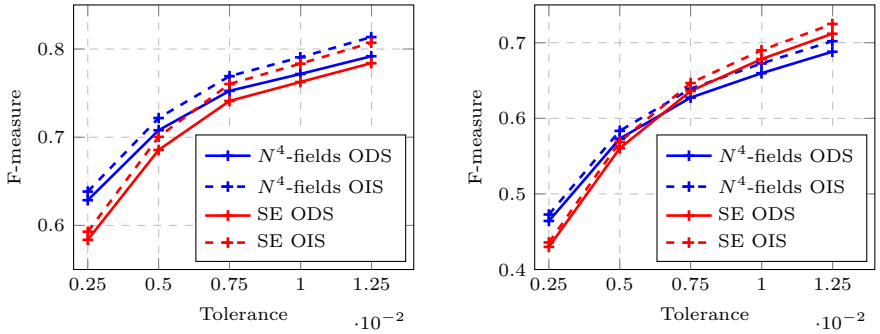
the CNNs has the same architecture as the CNN we use within N^4-fields, except that the size of the last layer is no longer 16 but equals the number of pixels we wish to produce predictions for (i.e. 1 for CNN-central and 256 for CNN-patch). At test time, we run the baseline on every patch and annotate chosen subsets of pixels with the output of the CNN classifier applying averaging in the overlapping regions. As with our main system, to assess the performance of the baseline, we use a committee of three CNN classifiers at three scales.

Nearest Neighbor Baseline. We have also evaluated a baseline that replaces the learned neural codes with "hand-crafted" features. For this, we used SIFT vectors computed over the input $M \times M$ patches as descriptors and use these vectors to perform the nearest-neighbor search in the training dataset. Since SIFT was designed mainly for natural RGB photographs, we evaluate this baseline for the BSDS500 edge detection only.

Alternative Encoding (AE). Given the impressive results of [20] on edge detection, we experimented with a variation of our method inspired by their method. We annotate each patch with a long binary vector that looks at the pairs of pixels in the output $N \times N$ patch and assigns it 1 or 0 depending whether it belongs to the object segment. We then apply PCA dimensionality reduction to 16 components. More formally, we define the target annotation vector during the CNN training to be:

$$\mathbf{B}(P) = \mathrm{PCA}((v_1, v_2, \ldots, v_L)), \tag{3}$$

where $L = \binom{N^2}{2}$ and v_i is defined for i-th pair (p_l, p_m) of pixels in the ground truth segmentation $\mathbf{S}(\mathcal{P})$ and is equal to $\mathbf{1}\{\mathbf{S}(\mathcal{P})[p_l] = \mathbf{S}(\mathcal{P})[p_m]\}$. In the experiments, we observe a small improvement for such alternative encoding.



(a) BSDS500 [4] (b) NYU RGBD [5]

Fig. 5: Performance scores for different tolerance thresholds (default value is $\mathbf{0.75 \cdot 10^{-2}}$) used in the BSDS500 benchmark [4]. Algorithms' performance (ODS and OIS measures plotted as *dashed* and *solid* lines respectively) is going down as the tolerance threshold is decreased. N^4-fields (*blue* lines) handles more stringent thresholds better, which suggests that cleaner edges are produced, as is also evidenced by the qualitative results. See Section 4 for details.

BSDS500 Experiments. The first dataset is Berkley Segmentation Dataset and Benchmark (BSDS500) [4]. It contains 500 color images divided into three subsets: 200 for training, 100 for validation and 200 for testing. Edge detection accuracy is measured using three scores: fixed contour threshold (ODS), per-image threshold (OIS), and average precision (AP) [4, 20]. In order to be evaluated properly, test edges must be thinned to one pixel width before running the benchmark code. We use the non-maximum suppression algorithm from [20] for that.

In general, N^4-fields perform similarly to the best previously published methods [20, 10, 26]. In particular, the full version of the system (the committee of three N^4-fields applied at three scales) matches the performance of the mentioned algorithms , with the alternative encoding performing marginally better

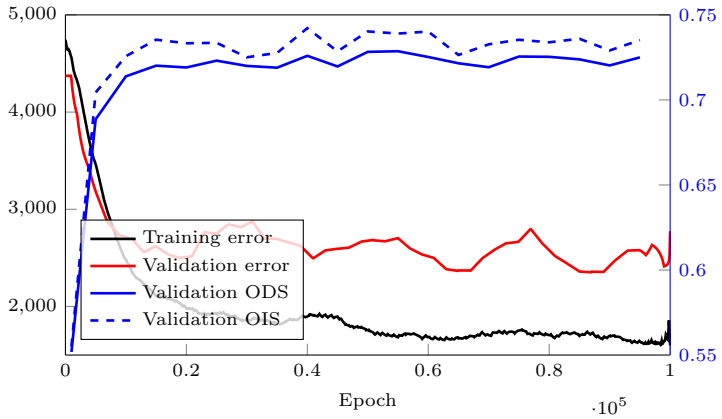


Fig. 6: The validation score (average precision) of the full N^4-fields and error rates (loss) of the underlying CNN measured throughout the training process. The strong correlation between the values suggests the importance of large-scale learning for the good performance of N^4-fields. This experiment was performed for the BSDS500 edge detection (hold out validation set included 20 images).

N^4-fields Input Structured Edge [20]

Fig. 7: Representative results on the BSDS500 dataset. For comparison, we give the results of the best previously published method [20]. The red numbers correspond to Recall/Precision/F-measure. We give two examples where N^4-fields perform better than [20], and one example (bottom row) where [20] performs markedly better according to the quantitative measure.

	ODS	OIS	AP
SIFT + NNB	.59	.60	.60
CNN-central	.72	.74	.75
CNN-patch	.73	.75	.74
gPb-owt-ucm [4]	.73	.76	.73
SCG [25]	.74	.76	.77
SE-MS, $T = 4$ [20]	.74	.76	**.78**
DeepNet [10]	.74	.76	.76
PMI + sPb, MS [26]	.74	**.77**	**.78**
N^4-fields	**.75**	.76	.77
N^4-fields, AE	**.75**	**.77**	**.78**

(a) BSDS500 [4] (Any)

	ODS	OIS	AP
SE-MS, $T = 4$ [20]	.59	.62	.59
DeepNet [10]	.61	.64	.61
PMI + sPb, MS [26]	.61	**.68**	.56
N^4-fields, AE	**.64**	.67	**.64**

(b) BSDS500 [4] (Consensus)

	ODS	OIS	AP
CNN, central	.60	.62	.55
CNN, patch	.58	.59	.49
gPb [4]	.53	.54	.40
SCG [25]	.62	.63	.54
SE-MS, $T = 4$ [20]	**.64**	**.65**	**.59**
N^4-fields	.61	.62	.56
N^4-fields, AE	.63	.64	.58

(c) NYU RGBD [5]

Table 1: Edge detection results on BSDS500 [4] (both for the original ground-truth annotation and "consensus" labels) and NYU RGBD [5]. Our approach (N^4-fields) achieves performance which is better or comparable to the state-of-the-art. We also observe that the relative performance of the methods in terms of perceptual quality are not adequately reflected by the standard performance measures.

(Table 1-a). Following [27] in order to account for the inherent problems of the dataset we also test our approach against the so-called "consensus" subset of the ground-truth labels. Within this setting our method significantly outperforms other algorithms in terms of ODS and AP (Table 1-b).

The benchmark evaluation procedure does not perform strict comparison of binary edge masks but rather tries to find the matching between pixels within certain tolerance level and then analyzes unmatched pixels [4]. We observed that the default distance matching tolerance threshold, while accounting for natural uncertainty in the exact position of the boundary, often ignores noticeable and unnatural segmentation mistakes such as spurious boundary pixels. Therefore, in addition to the accuracy evaluated for the default matching threshold, we report results for more stringent thresholds (Figure 5a-left).

It is also useful to investigate how successful is the deep learning, and what is its role within the N^4-fields. It is insightful to see whether the outputs of the CNN within the N^4-fields, i.e. $\text{CNN}(\mathcal{P})$ are reasonably close to the codes $\text{PCA}(\mathbf{A}(\mathcal{P}))$ that were used as target during the learning. To show this, in Fig-

ure 4 we give several representative results of the nearest neighbor searches where different types of codes are used on the query and on the dictionary dataset sides (alongside the corresponding patches). It can be seen, that there are very accurate matches (in terms of similarity between true annotations) between PCA codes on both sides, and reasonably good matches between neural (CNN) codes on both sides. However, when matching the neural code of an input patch to PCA codes on the dataset side the results are poor. This is especially noticeable for patches without natural boundaries in them as we force our neural network to map all such patches into one point (empty annotation is always encoded with the same vector). This qualitative performance results in a notoriously bad quantitative performance of the system that uses such matching (from the neural codes in the test image to the PCA codes in the training dataset).

While CNN is clearly unable to learn to reproduce the target codes closely, there is still a strong correlation between the training error (the value of the loss function within the CNN) and the performance of the N^4-fields (Figure 6). The efficiency of the learned codes and its importance for the good performance of N^4-fields is also highlighted by the fact that the nearest neighbor baseline using SIFT codes performs very poorly. Thus, optimizing the loss functions introduced above really makes edge maps produced by our algorithm agree with ground truth annotations.

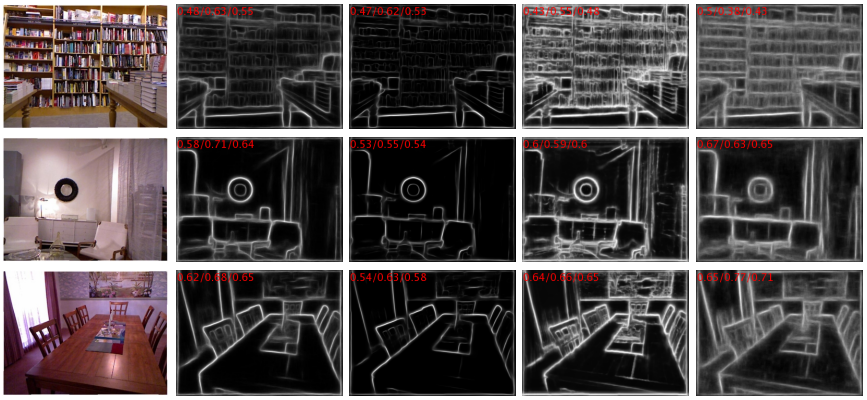


Fig. 8: Results on the NYU RGBD dataset. For comparison, we give the results of the best previously published method [20] and the CNN baseline. We show a representative result where the N^4-fields perform better (top), similarly (middle), or worse (bottom) than the baseline, according to the numberic measures shown in red (recall/precision/F-measure format). We argue that the numerical measures do not adequately reflect the relative perceptual performance of the methods.

NYU RGBD Experiments. We also show results for the NYU Depth dataset (v2) [5]. It contains 1,449 RGBD images with corresponding semantic segmentations. We use Ren and Bo [25] script to translate the data into BSDS500 format and use the same evaluation procedure, following training/testing split proposed by [25]. The CNN architecture stays the same except for the number of input channels which is now equal to four (RGBD) instead of three (RGB).

The results are summarized in Table 1-c. Our approach almost ties the state-of-the-art method by [20] for the default matching threshold. However, just like in the case of the BSDS500 dataset this difference in scores may be due to the peculiarity of the benchmark. Indeed, Figure 5b-right shows that for smaller values of matching thresholds, N^4-fields match or outperform the accuracy of Structured Edge detector [20].

Note on the Quantitative Performance. During the experiments, we observed a clear disconnect between the relative performance of the methods according to the quantitative measures, and according to the actual perceptual quality. This was especially noticeable for the NYU RGBD dataset (Figure 8). We provide extended uniformly-sampled qualitative results at the project website[2].

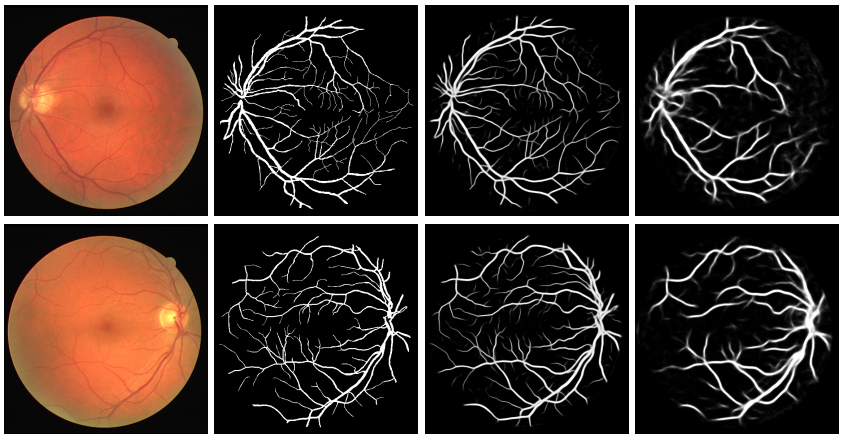


Fig. 9: Representative results on the DRIVE dataset. A close match to the human expert annotation is observed.

DRIVE Dataset. In order to demonstrate wide applicability of our method, we evaluate it on the DRIVE dataset [6] of the micrographs obtained within the diabetic retinopathy screening program. It includes forty 768 × 584 images split evenly into a training and a test sets. Ground truth annotations include manually segmented vasculature as well as ROI masks.

[2] `http://sites.skoltech.ru/compvision/projects/n4/` at the moment of publication.

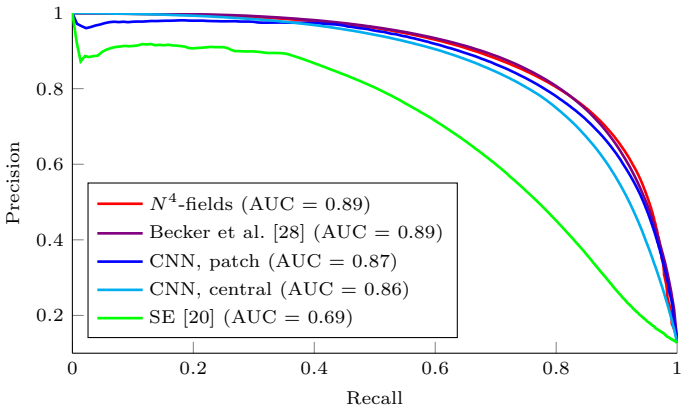


Fig. 10: Results for the DRIVE dataset [6] in the form of the recall/precision curves. Our approach matches the performance of the current state-of-the-art method by Becker et al [28] and performs much better than baselines and [20].

We use exactly the same CNN architecture as in the BSDS500 experiment. Without any further tuning our system achieves state-of-the-art performance comparable to the algorithm proposed by Becker et al. [28]. Precision/recall curves for both approaches as well as for the baseline neural networks and [20] (obtained using the authors' code) are shown in Figure 10. Notably, there is once again a clear advantage over the CNN baselines. Poor performance of [20] is likely to be due to the use of default features that are not suitable for this particular imaging modality. This provides an extra evidence for the benefits of fully data-driven approach.

5 Conclusion

We have presented a new approach to machine-learning based image processing. We have demonstrated how convolutional neural networks can be efficiently combined with the nearest neighbor search, and how such combination can improve the performance of standalone CNNs in the situation when CNN training underfits due to the complexity of a problem at hand. State-of-the-art results are demonstrated for natural edge detection in RGB and RGBD images, as well as for thin object (vessel) segmentation. Compared to the structured forests method [20], the proposed approach is slower, but can be adapted to new domains (e.g. micrographs) without retuning.

The future work may concern the fact that we use a PCA compression to define the target output during the CNN training. A natural idea is then to learn some non-linear transformation in the label space in parallel to the CNN training on the image patch input, so that the gap between the neural codes of the input patches and the target codes is minimized, and a closer match between the neural and the target codes is obtained. It remains to be seen whether this will bring the improvement to the overall performance of the system.

References

1. LeCun, Y., Boser, B.E., Denker, J.S., Henderson, D., Howard, R.E., Hubbard, W.E., Jackel, L.D.: Handwritten digit recognition with a back-propagation network. In: NIPS. (1989) 396–404
2. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: NIPS. (2012)
3. Ciresan, D.C., Giusti, A., Gambardella, L.M., Schmidhuber, J.: Deep neural networks segment neuronal membranes in electron microscopy images. In: NIPS. (2012) 2852–2860
4. Arbeláez, P., Maire, M., Fowlkes, C., Malik, J.: Contour detection and hierarchical image segmentation. IEEE Trans. Pattern Anal. Mach. Intell. **33**(5) (May 2011) 898–916
5. Silberman, N., Fergus, R.: Indoor scene segmentation using a structured light sensor. In: ICCV Workshops, IEEE (2011) 601–608
6. Staal, J., Abrmoff, M.D., Niemeijer, M., Viergever, M.A., van Ginneken, B.: Ridge-based vessel segmentation in color images of the retina. IEEE Trans. Med. Imaging **23**(4) (2004) 501–509
7. Egmont-Petersen, M., de Ridder, D., Handels, H.: Image processing with neural networks - a review. Pattern recognition **35**(10) (2002) 2279–2301
8. Mnih, V., Hinton, G.E.: Learning to detect roads in high-resolution aerial images. In: ECCV. Springer (2010) 210–223
9. Schulz, H., Behnke, S.: Learning object-class segmentation with convolutional neural networks. In: ESANN. Volume 3. (2012)
10. Kivinen, J.J., Williams, C.K.I., Heess, N.: Visual boundary prediction: A deep neural prediction network and quality dissection. In: AISTATS. (2014) 512–521
11. Ranzato, M., Hinton, G.E.: Modeling pixel means and covariances using factorized third-order boltzmann machines. In: CVPR. (2010) 2551–2558
12. Farabet, C., Couprie, C., Najman, L., LeCun, Y.: Learning hierarchical features for scene labeling. Pattern Analysis and Machine Intelligence, IEEE Transactions on **35**(8) (2013) 1915–1929
13. Jain, V., Murray, J.F., Roth, F., Turaga, S.C., Zhigulin, V.P., Briggman, K.L., Helmstaedter, M., Denk, W., Seung, H.S.: Supervised learning of image restoration with convolutional networks. In: ICCV. (2007) 1–8
14. Jain, V., Seung, H.S.: Natural image denoising with convolutional networks. In: NIPS. (2008) 769–776
15. Burger, H.C., Schuler, C.J., Harmeling, S.: Image denoising: Can plain neural networks compete with bm3d? In: CVPR. (2012) 2392–2399
16. Chopra, S., Hadsell, R., LeCun, Y.: Learning a similarity metric discriminatively, with application to face verification. In: CVPR. Volume 1. (2005) 539–546
17. Dabov, K., Foi, A., Katkovnik, V., Egiazarian, K.: Image restoration by sparse 3d transform-domain collaborative filtering. In: Electronic Imaging 2008, International Society for Optics and Photonics (2008) 681207–681207
18. Criminisi, A., Pérez, P., Toyama, K.: Region filling and object removal by exemplar-based image inpainting. Image Processing, IEEE Transactions on **13**(9) (2004) 1200–1212
19. Freeman, W.T., Pasztor, E.C., Carmichael, O.T.: Learning low-level vision. International Journal of Computer Vision **40**(1) (2000) 25–47
20. Dollár, P., Zitnick, C.L.: Structured forests for fast edge detection. In: ICCV. (2013)

21. Hinton, G.E., Srivastava, N., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Improving neural networks by preventing co-adaptation of feature detectors. CoRR **abs/1207.0580** (2012)
22. Zeiler, M.D., Fergus, R.: Visualizing and understanding convolutional networks. CoRR **abs/1311.2901** (2013)
23. Sermanet, P., Eigen, D., Zhang, X., Mathieu, M., Fergus, R., LeCun, Y.: Overfeat: Integrated recognition, localization and detection using convolutional networks. CoRR **abs/1312.6229** (2013)
24. Vedaldi, A., Fulkerson, B.: VLFeat: An open and portable library of computer vision algorithms. http://www.vlfeat.org/ (2008)
25. Xiaofeng, R., Bo, L.: Discriminatively trained sparse code gradients for contour detection. In: NIPS. (2012) 593–601
26. Isola, P., Zoran, D., Krishnan, D., Adelson, E.H.: Crisp boundary detection using pointwise mutual information. In: ECCV. (2014)
27. Hou, X., Yuille, A., Koch, C.: Boundary detection benchmarking: Beyond f-measures. In: CVPR. Volume 2013., IEEE (2013) 1–8
28. Becker, C.J., Rigamonti, R., Lepetit, V., Fua, P.: Supervised feature learning for curvilinear structure segmentation. In: MICCAI. Volume 8149 of Lecture Notes in Computer Science. (2013) 526–533