HAL
open science

# Learning to Compare Image Patches via Convolutional Neural Networks

Sergey Zagoruyko, Nikos Komodakis

## To cite this version:

Sergey Zagoruyko, Nikos Komodakis. Learning to Compare Image Patches via Convolutional Neural Networks. IEEE Conference on Computer Vision and Pattern Recognition 2015, Jun 2015, Boston, United States. 10.1109/CVPR.2015.7299064 . hal-01246261

**HAL Id: hal-01246261**

**https://hal.science/hal-01246261v1**

Submitted on 18 Dec 2015

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

arXiv:1504.03641v1 [cs.CV] 14 Apr 2015

# Learning to Compare Image Patches via Convolutional Neural Networks

Sergey Zagoruyko
Universite Paris Est, Ecole des Ponts ParisTech
sergey.zagoruyko@imagine.enpc.fr

Nikos Komodakis
Universite Paris Est, Ecole des Ponts ParisTech
nikos.komodakis@enpc.fr

## Abstract

*In this paper we show how to learn directly from image data (*i.e.*, without resorting to manually-designed features) a general similarity function for comparing image patches, which is a task of fundamental importance for many computer vision problems. To encode such a function, we opt for a CNN-based model that is trained to account for a wide variety of changes in image appearance. To that end, we explore and study multiple neural network architectures, which are specifically adapted to this task. We show that such an approach can significantly outperform the state-of-the-art on several problems and benchmark datasets.*

## 1. Introduction

Comparing patches across images is probably one of the most fundamental tasks in computer vision and image analysis. It is often used as a subroutine that plays an important role in a wide variety of vision tasks. These can range from low-level tasks such as structure from motion, wide baseline matching, building panoramas, and image super-resolution, up to higher-level tasks such as object recognition, image retrieval, and classification of object categories, to mention a few characteristic examples.

Of course, the problem of deciding if two patches correspond to each other or not is quite challenging as there exist far too many factors that affect the final appearance of an image [17]. These can include changes in viewpoint, variations in the overall illumination of a scene, occlusions, shading, differences in camera settings, *etc*. In fact, this need of comparing patches has given rise to the development of many hand-designed feature descriptors over the past years, including SIFT [15], that had a huge impact in the computer vision community. Yet, such manually designed descriptors may be unable to take into account in an optimal manner all of the aforementioned factors that determine the appearance of a patch. On the other hand, nowadays one can easily gain access to (or even generate using available software) large datasets that contain patch correspondences between images [22]. This begs the following question: can we make proper use of such datasets to automatically learn a similarity function for image patches ?

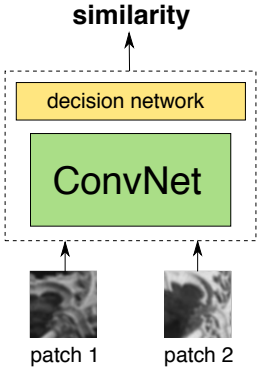



Figure 1. Our goal is to learn a general similarity function for image patches. To encode such a function, here we make use of and explore convolutional neural network architectures.

The goal of this paper is to affirmatively address the above question. Our aim is thus to be able to generate a patch similarity function *from scratch*, *i.e.*, without attempting to use any manually designed features but instead directly learn this function from annotated pairs of raw image patches. To that end, inspired also by the recent advances in neural architectures and deep learning, we choose to represent such a function in terms of a deep convolutional neural network [14, 13] (Fig. 1). In doing so, we are also interested in addressing the issue of what network architecture should be best used in a task like this. We thus explore and propose various types of networks, having architectures that exhibit different trade-offs and advantages. In all cases, to train these networks, we are using as sole input a large database that contains pairs of raw image patches (both matching and non-matching). This allows to further improve the performance of our method simply by enriching this database with more samples (as software for automatically generating such samples is readily available [21]).

To conclude this section, the paper's main contributions are as follows: (i) We learn directly from image data (*i.e.*, without any manually-designed features) a general similar-

Source code and trained models are available online at http://imagine.enpc.fr/~zagoruys/deepcompare.html (work supported by EC project FP7-ICT-611145 ROBOSPECT).

ity function for patches that can implicitly take into account various types of transformations and effects (due to *e.g.*, a wide baseline, illumination, *etc.*). (ii) We explore and propose a variety of different neural network models adapted for representing such a function, highlighting at the same time network architectures that offer improved performance. as in [19]. (iii) We apply our approach on several problems and benchmark datasets, showing that it significantly outperforms the state-of-the-art and that it leads to feature descriptors with much better performance than manually designed descriptors (*e.g.*, SIFT, DAISY) or other learnt descriptors as in [19]. Importantly, due to their convolutional nature, the resulting descriptors are very efficient to compute even in a dense manner.

## 2. Related work

The conventional approach to compare patches is to use descriptors and a squared euclidean distance. Most feature descriptors are hand-crafted as SIFT [15] or DAISY [26]. Recently, methods for learning a descriptor have been proposed [27] (*e.g.*, DAISY-like descriptors learn pooling regions and dimensionality reduction [3]). Simonyan *et al.* [19] proposed a convex procedure for training on both tasks.

Our approach, however, is inspired by the recent success of convolutional neural networks [18, 25, 24, 9]. Although these models involve a highly non-convex objective function during training, they have shown outstanding results in various tasks [18]. Fischer *et al.* [10] analysed the performance of convolutional descriptors from AlexNet network (that was trained on Imagenet dataset [13]) on the well-known Mikolajczyk dataset [16] and showed that these convolutional descriptors outperform SIFT in most cases except blur. They also proposed an unsupervised training approach for deriving descriptors that outperform both SIFT and Imagenet trained network.

Zbontar and LeCun in [28] have recently proposed a CNN-based approach to compare patches for computing cost in small baseline stereo problem and shown the best performance in KITTI dataset. However, the focus of that work was only on comparing pairs that consist of very small patches like the ones in narrow baseline stereo. In contrast, here we aim for a similarity function that can account for a broader set of appearance changes and can be used in a much wider and more challenging set of applications, including, *e.g.*, wide baseline stereo, feature matching and image retrieval.

## 3. Architectures

As already mentioned, the input to the neural network is considered to be a pair of image patches. Our models do not impose any limitations with respect to the number of channels in the input patches, *i.e.*, given a dataset with colour patches the networks could be trained to further increase performance. However, to be able to compare our approach with state-of-the-art methods on existing datasets, we chose to use only grayscale patches during training. Furthermore, with the exception of the SPP model described in section 3.2, in all other cases the patches given as input to the network are assumed to have a fixed size of $64 \times 64$ (this means that original patches may need to be resized to the above spatial dimensions).

There are several ways in which patch pairs can be processed by the network and how the information sharing can take place in this case. For this reason, we explored and tested a variety of models. We start in section 3.1 by describing the three basic neural network architectures that we studied, *i.e.*, 2-channel, Siamese, Pseudo-siamese (see Fig. 2), which offer different trade-offs in terms of speed and accuracy (note that, as usually, applied patch-matching techniques imply testing a patch against a big number of other patches, and so re-using computed information is always useful). Essentially these architectures stem from the different way that each of them attempts to address the following question: when composing a similarity function for comparing image patches, do we first choose to compute a descriptor for each patch and then create a similarity on top of these descriptors or do we perhaps choose to skip the part related to the descriptor computation and directly proceed with the similarity estimation?

In addition to the above basic models, we also describe in section 3.2 some extra variations concerning the network architecture. These variations, which are not mutually exclusive to each other, can be used in conjunction with any of the basic models described in section 3.1. Overall, this leads to a variety of models that is possible to be used for the task of comparing image patches.

### 3.1. Basic models

**Siamese:** This type of network resembles the idea of having a descriptor [2, 6]. There are two branches in the network that share exactly the same architecture and the same set of weights. Each branch takes as input one of the two patches and then applies a series of convolutional, ReLU and max-pooling layers. Branch outputs are concatenated and given to a top network that consists of linear fully connected and ReLU layers. In our tests we used a top network consisting of 2 linear fully connected layers (each with 512 hidden units) that are separated by a ReLU activation layer.

Branches of the siamese network can be viewed as descriptor computation modules and the top network - as a similarity function. For the task of matching two sets of patches at test time, descriptors can first be computed independently using the branches and then matched with the top network (or even with a distance function like $l_2$).

**Pseudo-siamese:** In terms of complexity, this architec-

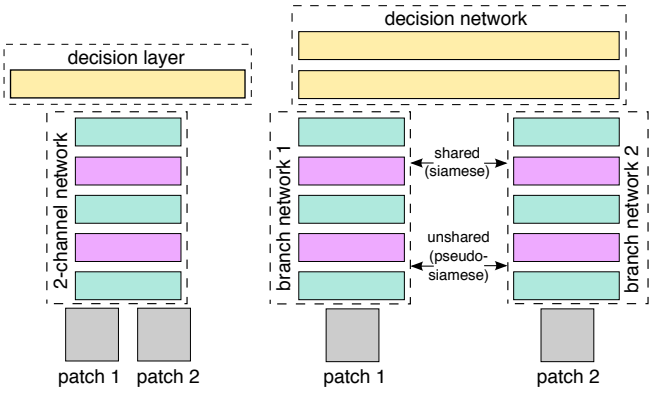



Figure 2. Three basic network architectures: 2-channel on the left, siamese and pseudo-siamese on the right (the difference between siamese and pseudo-siamese is that the latter does not have shared branches). Color code used: cyan = Conv+ReLU, purple = max pooling, yellow = fully connected layer (ReLU exists between fully connected layers as well).

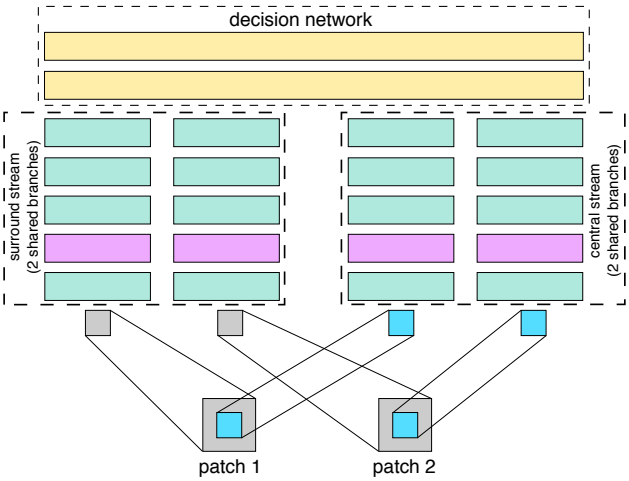



Figure 3. A central-surround two-stream network that uses a siamese-type architecture to process each stream. This results in 4 branches in total that are given as input to the top decision layer (the two branches in each stream are shared in this case).

ture can be considered as being in-between the siamese and the 2-channel networks. More specifically, it has the structure of the siamese net described above except that the weights of the two branches are uncoupled, *i.e.*, not shared. This increases the number of parameters that can be adjusted during training and provides more flexibility than a restricted siamese network, but not as much as the 2-channel network described next. On the other hand, it maintains the efficiency of siamese network at test time.

**2-channel:** unlike the previous models, here there is no direct notion of descriptor in the architecture. We simply consider the two patches of an input pair as a 2-channel image, which is directly fed to the first convolutional layer of the network. In this case, the bottom part of the network consists of a series of convolutional, ReLU and max-pooling layers. The output of this part is then given as input to a top module that consists simply of a fully connected linear decision layer with 1 output. This network provides greater flexibility compared to the above models as it starts by processing the two patches jointly. Furthermore, it is fast to train, but in general at test time it is more expensive as it requires all combinations of patches to be tested against each other in a brute-force manner.

### 3.2. Additional models

**Deep network.** We apply the technique proposed by Simonyan and Zisserman in [20] advising to break up bigger convolutional layers into smaller 3x3 kernels, separated by ReLU activations, which is supposed to increase the non-linearities inside the network and make the decision function more discriminative. They also report that it might be difficult to initialise such a network, we, however, do not observe this behavior and train the network from scratch as usual. In our case, when applying this technique to our model, the convolutional part of the final architecture turns out to consist of one convolutional 4x4 layer and 6 convolutional layers with 3x3 layers, separated by ReLU activations. As we shall also see later in the experimental results, such a change in the network architecture can contribute in further improving performance, which is in accordance with analogous observations made in [20].

**Central-surround two-stream network.** As its name suggests, the proposed architecture consists of two separate streams, central and surround, which enable a processing in the spatial domain that takes place over two different resolutions. More specifically, the central high-resolution stream receives as input two $32 \times 32$ patches that are generetad by cropping (at the original resolution) the central $32 \times 32$ part of each input $64 \times 64$ patch. Furthermore, the surround low-resolution stream receives as input two $32 \times 32$ patches, which are generated by downsampling at half the original pair of input patches. The resulting two streams can then be processed by using any of the basic architectures described in section 3.1 (see Fig. 3 for an example that uses a siamese architecture for each stream).

One reason to make use of such a two-stream architecture is because multi-resolution information is known to be important in improving the performance of image matching. Furthermore, by considering the central part of a patch twice (*i.e.*, in both the high-resolution and low-resolution streams) we implicitly put more focus on the pixels closer to the center of a patch and less focus on the pixels in the periphery, which can also help for improving the precision of matching (essentially, since pooling is applied to the down-sampled image, pixels in the periphery are allowed to have more variance during matching). Note that the total input dimenionality is reduced by a factor of two in this case. As a result, training proceeds faster, which is also one other

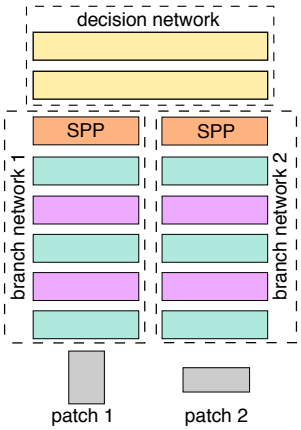



Figure 4. SPP network for a siamese architecture: SPP layers (orange) are inserted immediately after the 2 branches of the network so that the top decision layer has an input of fixed dimensionality for any size of the input patches.

practical advantage.

**Spatial pyramid pooling (SPP) network for comparing patches.** Up to this point we have been assuming that the network requires the input patches to have a fixed size of $64 \times 64$. This requirement comes from the fact that the output of the last convolutional layer of the network needs to have a predefined dimensionality. Therefore, when we need to compare patches of arbitrary sizes, this means that we first have to resize them to the above spatial dimensions. However, if we look at the example of descriptors like SIFT, for instance, we can see that another possible way to deal with patches of arbitrary sizes is via adjusting the size of the spatial pooling regions to be proportional to the size of the input patch so that we can still maintain the required fixed output dimensionality for the last convolutional layer without deteriorating the resolution of the input patches.

This is also the idea behind the recently proposed SPP-net architecture [11], which essentially amounts to inserting a spatial pyramid pooling layer between the convolutional layers and the fully-connected layers of the network. Such a layer aggregates the features of the last convolutional layer through spatial pooling, where the size of the pooling regions is dependent on the size of the input. Inspired by this, we propose to also consider adapting the network models of section 3.1 according to the above SPP-architecture. This can be easily achieved for all the considered models (*e.g.*, see Fig. 4 for an example with a siamese model).

## 4. Learning

**Optimization.** We train all models in strongly supervised manner. We use a hinge-based loss term and squared $l_2$-norm regularization that leads to the following learning objective function

$$\min_w \frac{\lambda}{2}\|w\|_2 + \sum_{i=1}^{N} \max(0, 1 - y_i o_i^{net}) \ , \qquad (1)$$

where $w$ are the weights of the neural network, $o_i^{net}$ is the network output for the $i$-th training sample, and $y_i \in \{-1, 1\}$ the corresponding label (with $-1$ and $1$ denoting a non-matching and a matching pair, respectively).

ASGD with constant learning rate 1.0, momentum 0.9 and weight decay $\lambda = 0.0005$ is used to train the models. Training is done in mini-batches of size 128. Weights are initialised randomly and all models are trained from scratch.

**Data Augmentation and preprocessing.** To combat overfitting we augment training data by flipping both patches in pairs horizontally and vertically and rotating to 90, 180, 270 degrees. As we don't notice overfitting while training in such manner we train models for a certain number of iterations, usually for 2 days, and then test performance on test set.

Training dataset size allows us to store all the images directly in GPU memory and very efficiently retrieve patch pairs during training. Images are augmented "on-the fly". We use Titan GPU in Torch [7] and convolution routines are taken from Nvidia cuDNN library [5]. Our siamese descriptors on GPU are just 2 times slower than computing SIFT descriptors on CPU and 2 times faster than Imagenet descriptors on GPU according to [10].

## 5. Experiments

We applied our models to a variety of problems and datasets. In the following we report results, and also provide comparisons with the state-of-the-art.

### 5.1. Local image patches benchmark

For a first evaluation of our models, we used the standard benchmark dataset from [3] that consists of three subsets, Yosemite, Notre Dame, and Liberty, each of which contains more than 450,000 image patches (64 x 64 pixels) sampled around Difference of Gaussians feature points. The patches are scale and orientation normalized. Each of the subsets was generated using actual 3D correspondences obtained via multi-view stereo depth maps. These maps were used to produce 500,000 ground-truth feature pairs for each dataset, with equal number of positive (correct) and negative (incorrect) matches.

For evaluating our models, we use the evaluation protocol of [4] and generate ROC curves by thresholding the distance between feature pairs in the descriptor space. We report the false positive rate at 95% recall (FPR95) on each of the six combinations of training and test sets, as well as the mean across all combinations. We also report the mean, denoted as $\mathrm{mean}(1, 4)$, for only those 4 combinations that

were used in [1], [3] (in which case training takes place on Yosemite or Notre Dame, but not Liberty).

Table 1 reports the performance of several models, and also details their architecture (we have also experimented with smaller kernels, less max-pooling layers, as well as adding normalisations, without noticing any significant improvement in performance). We briefly summarize some of the conclusions that can be drawn from this table. A first important conclusion is that 2-channel-based architectures (*e.g.*, `2ch`, `2ch-deep`, `2ch-2stream`) exhibit clearly the best performance among all models. This is something that indicates that it is important to *jointly* use information from both patches right from the *first* layer of the network.

`2ch-2stram` network was the top-performing network on this dataset, with `2ch-deep` following closely (this verifies the importance of multi-resolution information during matching and that also increasing the network depth helps). In fact, `2ch-2stream` managed to outperform the previous state-of-the-art by a large margin, achieving 2.45 times better score than [19]! The difference with SIFT was even larger, with our model giving 6.65 times better score in this case (SIFT score on `mean(1,4)` was 31.2 according to [3]).

Regarding siamese-based architectures, these too manage to achieve better performance than existing state-of-the-art systems. This is quite interesting because, *e.g.*, none of these siamese networks tries to learn the shape, size or placement of the pooling regions (like, *e.g.*, [19, 3] do), but instead utilizes just standard max-pooling layers. Among the siamese models, the two-stream network (`siam-2stream`) had the best performance, verifying once more the importance of multi-resolution information when it comes to comparing image patches. Furthermore, the pseudo-siamese network (`pseudo-siam`) was better than the corresponding siamese one (`siam`).

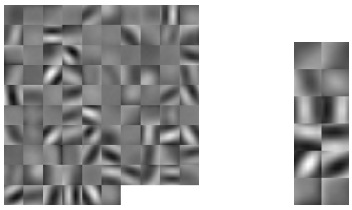
We also conducted additional experiments, in which we tested the performance of siamese models when their top decision layer is replaced with the $l_2$ Euclidean distance of the two convolutional descriptors produced by the two branches of the network (denoted with the suffix $l_2$ in the name). In this case, prior to applying the Euclidean distance, the descriptors are $l_2$-normalized (we also tested $l_1$ normalization). For pseudo-siamese only one branch was used to extract descriptors. As expected, in this case the two-stream network (`siam-2stream-`$l_2$) computes better distances than the siamese network (`siam-`$l_2$), which, in turn, computes better distances than the pseudo-siamese model (`pseudo-siam-`$l_2$). In fact, the `siam-2stream-`$l_2$ network manages to outperform even the previous state-of-the-art descriptor [19], which is quite surprising given that these siamese models have never been trained using $l_2$ distances.

For a more detailed comparison of the various models,

| | $\text{conv3}_{(3456)}$ | $\text{conv4}_{(3456)}$ | $\text{conv5}_{(2304)}$ |
|---|---|---|---|
| Notredame | 12.22 | 9.64 | 19.384 |
| Liberty | 16.25 | 14.26 | 21.592 |
| Yosemite | 33.25 | 30.22 | 43.262 |
| mean | 20.57 | 17.98 | 28.08 |

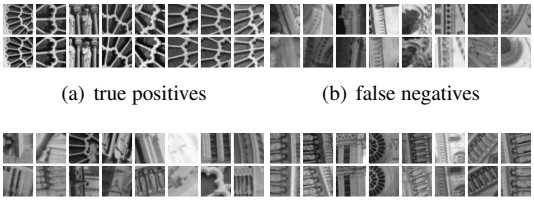
Table 2. FPR95 for imagenet-trained features (dimensionality of each feature appears as subscript).

we provide the corresponding ROC curves in Fig. 5. Furthermore, we show in Table 2 the performance of imagenet-trained CNN features (these were $l_2$-normalized to improve results). Among these, `conv4` gives the best FPR95 score, which is equal to 17.98. This makes it better than SIFT but still much worse than our models.



(a) (b)

Figure 6. (a) Filters of the first convolutional layer of `siam` network. (b) Rows correspond to first layer filters from `2ch` network (only a subset shown), depicting left and right part of each filter.



(a) true positives (b) false negatives

(c) true negatives (d) false positives

Figure 7. Top-ranking false and true matches by `2ch-deep`.

Fig. 6(a) displays the filters of the first convolutional layer learnt by the siamese network. Furthermore, Fig. 6(b) shows the left and right parts for a subset of the first layer filters of the 2-channel network `2ch`. It is worth mentioning that corresponding left and right parts look like being negative to each other, which basically means that the network has learned to compute differences of features between the two patches (note, though, that not all first layer filters of `2ch` look like this). Last, we show in Fig. 7 some top ranking false and correct matches as computed by the `2ch-deep` network. We observe that false matches could be easily mistaken even by a human (notice, for instance, how similar the two patches in false positive examples look like).

For the rest of the experiments, we note that we use models trained on the Liberty dataset.

| Train | Test | 2ch-2stream | 2ch-deep | 2ch | siam | siam-$l_2$ | pseudo-siam | pseudo-siam-$l_2$ | siam-2stream | siam-2stream-$l_2$ | [19] |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Yos | ND | **2.11** | 2.52 | 3.05 | 5.75 | 8.38 | 5.44 | 8.95 | 5.29 | 5.58 | 6.82 |
| Yos | Lib | **7.2** | 7.4 | 8.59 | 13.48 | 17.25 | 10.35 | 18.37 | 11.51 | 12.84 | 14.58 |
| ND | Yos | **4.1** | 4.38 | 6.04 | 13.23 | 15.89 | 12.64 | 15.62 | 10.44 | 13.02 | 10.08 |
| ND | Lib | 4.85 | **4.55** | 6.05 | 8.77 | 13.24 | 12.87 | 16.58 | 6.45 | 8.79 | 12.42 |
| Lib | Yos | 5 | **4.75** | 7 | 14.89 | 19.91 | 12.5 | 17.83 | 9.02 | 13.24 | 11.18 |
| Lib | ND | **1.9** | 2.01 | 3.03 | 4.33 | 6.01 | 3.93 | 6.58 | 3.05 | 4.54 | 7.22 |
| mean | | **4.19** | 4.27 | 5.63 | 10.07 | 13.45 | 9.62 | 13.99 | 7.63 | 9.67 | 10.38 |
| mean(1,4) | | **4.56** | 4.71 | 5.93 | 10.31 | 13.69 | 10.33 | 14.88 | 8.42 | 10.06 | 10.98 |

Table 1. Performance of several models on the "local image patches" benchmark. The models architecture is as follows: (i) 2ch-2stream consists of two branches $\mathrm{C}(95,5,1)$-ReLU-$\mathrm{P}(2,2)$-$\mathrm{C}(96,3,1)$-ReLU-$\mathrm{P}(2,2)$-$\mathrm{C}(192,3,1)$-ReLU-$\mathrm{C}(192,3,1)$-ReLU, one for central and one for surround parts, followed by $\mathrm{F}(768)$-ReLU-$\mathrm{F}(1)$ (ii) 2ch-deep $=$ $\mathrm{C}(96,4,3)$-Stack$(96)$-$\mathrm{P}(2,2)$-Stack$(192)$-$\mathrm{F}(1)$, where Stack$(n)$ $=$ $\mathrm{C}(n,3,1)$-ReLU-$\mathrm{C}(n,3,1)$-ReLU-$\mathrm{C}(n,3,1)$-ReLU. (iii) 2ch $=$ $\mathrm{C}(96,7,3)$-ReLU-$\mathrm{P}(2,2)$-$\mathrm{C}(192,5,1)$-ReLU-$\mathrm{P}(2,2)$-$\mathrm{C}(256,3,1)$-ReLU-$\mathrm{F}(256)$-ReLU-$\mathrm{F}(1)$ (iv) siam has two branches $\mathrm{C}(96,7,3)$-ReLU-$\mathrm{P}(2,2)$-$\mathrm{C}(192,5,1)$-ReLU-$\mathrm{P}(2,2)$-$\mathrm{C}(256,3,1)$-ReLU and decision layer $\mathrm{F}(512)$-ReLU-$\mathrm{F}(1)$ (v) siam-$l_2$ reduces to a single branch of siam (vi) pseudo-siam is uncoupled version of siam (vii) pseudo-siam-$l_2$ reduces to a single branch of pseudo-siam (viii) siam-2stream has 4 branches $\mathrm{C}(96,4,2)$-ReLU-$\mathrm{P}(2,2)$-$\mathrm{C}(192,3,1)$-ReLU-$\mathrm{C}(256,3,1)$-ReLU-$\mathrm{C}(256,3,1)$-ReLU (coupled in pairs for central and surround streams), and decision layer $\mathrm{F}(512)$-ReLU-$\mathrm{F}(1)$ (ix) siam-2stream-$l_2$ consists of one central and one surround branch of siam-2stream. The shorthand notation used was the following: $\mathrm{C}(n,k,s)$ is a convolutional layer with $n$ filters of spatial size $k \times k$ applied with stride $s$, $\mathrm{P}(k,s)$ is a max-pooling layer of size $k \times k$ applied with stride $s$, and $\mathrm{F}(n)$ denotes a fully connected linear layer with $n$ output units.

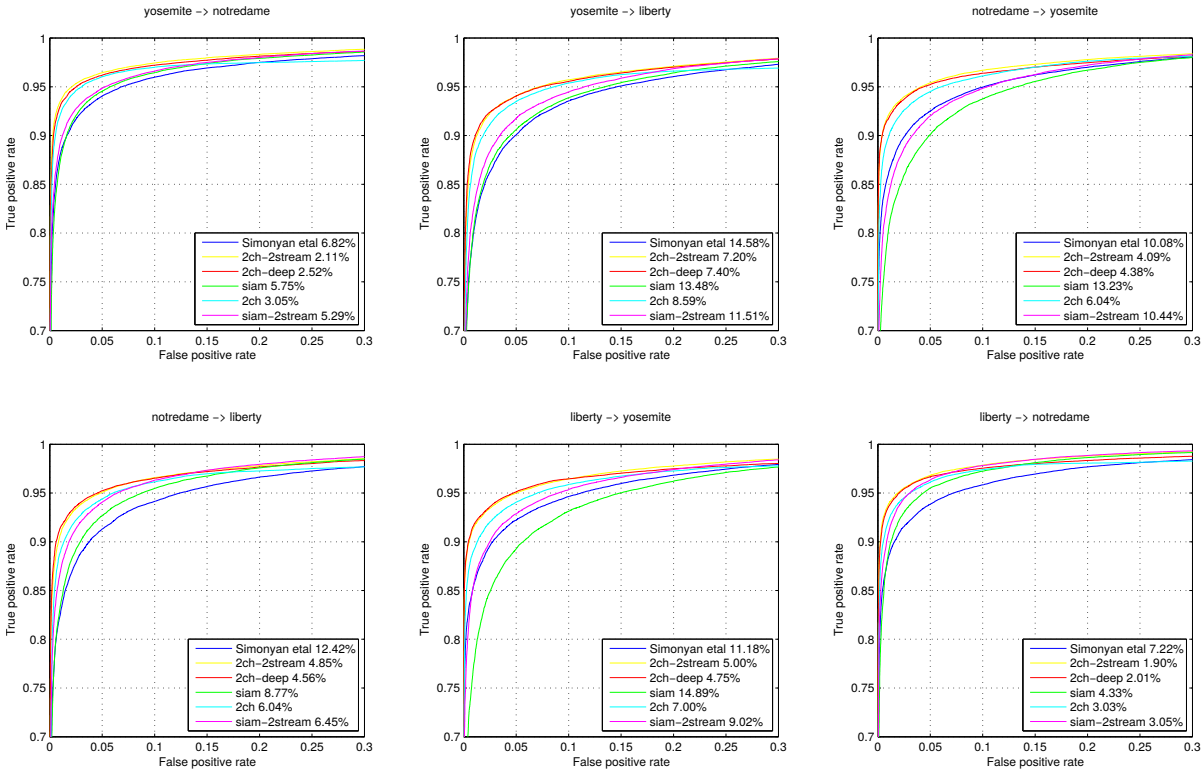



Figure 5. ROC curves for various models (including the state-of-the-art descriptor [19]) on the local image patches benchmark. Numbers in the legends are corresponding FPR95 values

## 5.2. Wide baseline stereo evaluation

For this evaluation we chose the dataset by Strecha *et al.* [23], which contains several image sequences with ground truth homographies and laser-scanned depthmaps. We used "fountain" and "herzjesu" sequences to produce 6 and 5 rectified stereo pairs respectively. Baselines in both sequences we chose are increasing with each image making matching more difficult. Our goal was to show that a photometric cost computed with neural network competes favorably against costs produced by a state-ot-the-art hand-crafted feature descriptor, so we chose to compare with DAISY [26].

Since our focus was not on efficiency, we used an un-

optimized pipeline for computing the photometric costs. More specifically, for 2-channel networks we used a brute-force approach, where we extract patches on corresponding epipolar lines with subpixel estimation, construct batches (containing a patch from the left image $I_1$ and all patches on the corresponding epipolar line from the right image $I_2$) and compute network outputs, resulting in the cost:

$$C(\mathbf{p}, d) = -o^{net}(I_1(\mathbf{p}), I_2(\mathbf{p} + d)) \quad (2)$$

Here, $I(\mathbf{p})$ denotes a neighbourhood intensity matrix around a pixel $\mathbf{p}$, $o^{net}(P_1, P_2)$ is the output of the neural network given a pair of patches $P_1$ and $P_2$, and $d$ is the distance between points on epipolar line.

For siamese-type networks, we compute descriptors for each pixel in both images once and then match them with decision top layer or $l_2$ distance. In the first case the formula for photometric cost is the following:

$$C(\mathbf{p}, d) = -o^{top}(D_1(I_1(\mathbf{p})), D_2(I_2(\mathbf{p} + d))) \quad (3)$$

where $o^{top}$ is output of the top decision layer, and $D_1$, $D_2$ are outputs of branches of the siamese or pseudo-siamese network, i.e. descriptors (in case of siamese network $D_1 = D_2$). For $l_2$ matching, it holds:

$$C(\mathbf{p}, d) = \|D_1(I_1(\mathbf{p})) - D_2(I_2(\mathbf{p} + d))\|_2 \quad (4)$$

It is worth noting that all costs above can be computed a lot more efficiently using speed optimizations similar with [28]. This essentially means treating all fully connected layers as $1 \times 1$ convolutions, computing branches of siamese network only once, and furthermore computing the outputs of these branches as well as the final outputs of the network at all locations using a number of forward passes on full images (e.g., for a 2-channel architecture such an approach of computing the photometric costs would only require feeding the network with $s^2 \cdot d_{\max}$ full 2-channel images of size equal to the input image pair, where $s$ is the stride at the first layer of the network and $d_{\max}$ is the maximum disparity).

Once computed, the photometric costs are subsequently used as unary terms in the following pairwise MRF energy

$$E(\{d_p\}) = \sum_p C(\mathbf{p}, d_p) + \sum_{(p,q)\in\mathcal{E}} (\lambda_1 + \lambda_2 e^{-\frac{\|\nabla I_1(\mathbf{p})\|_2}{\sigma^2}}) \cdot |d_p - d_q|\,,$$

minimized using algorithm [8] based on FastPD [12] (we set $\lambda_1 = 0.01$, $\lambda_2 = 0.2$, $\sigma = 7$ and $\mathcal{E}$ is a 4-connected grid).

We show in Fig. 9 some qualitative results in terms of computed depth maps (with and without global optimization) for the "fountain" image set (results for "herzjesu" appear in supp. material due to lack of space). Global MRF optimization results visually verify that photometric cost computed with neural network is much more robust than with hand-crafted features, as well as the high quality of the depth maps produced by 2-channel architectures. Results without global optimization also show that the estimated depth maps contain much more fine details than DAISY. They may exhibit a very sparse set of errors for the case of siamese-based networks, but these errors can be very easily eliminated during global optimization.

Fig. 8 also shows a quantitative comparison, focusing in this case on siamese-based models as they are more efficient. The first plot of that figure shows (for a single stereo pair) the distribution of deviations from the ground truth across all range of error thresholds (expressed here as a fraction of the scene's depth range). Furthermore, the other plots of the same figure summarize the corresponding distributions of errors for the six stereo pairs of increasing baseline (in this case we also show separately the error distributions when only unoccluded pixels are taken into account). The error thresholds were set to 3 and 5 pixels in these plots (note that the maximum disparity is around 500 pixels in the largest baseline). As can be seen, all siamese models perform much better than DAISY across all error thresholds and all baseline distances (*e.g.*, notice the difference in the curves of the corresponding plots).

### 5.3. Local descriptors performance evaluation

We also test our networks on Mikolajczyk dataset for local descriptors evaluation [16]. The dataset consists of 48 images in 6 sequences with camera viewpoint changes, blur, compression, lighting changes and zoom with gradually increasing amount of transfomation. There are known ground truth homographies between the first and each other image in sequence.

Testing technique is the same as in [16]. Briefly, to test a pair of images, detectors are applied to both images to extract keypoints. Following [10], we use MSER detector. The ellipses provided by detector are used to exctract patches from input images. Ellipse size is magnified by a factor of 3 to include more context. Then, depending on the type of network, either descriptors, meaning outputs of siamese or pseudo-siamese branches, are extracted, or all patch pairs are given to 2-channel network to assign a score.

A quantitative comparison on this dataset is shown for several models in Fig. 10. Here we also test the CNN network `siam-SPP-`$l_2$, which is an SPP-based siamese architecture (note that `siam-SPP` is same as `siam` but with the addition of two SPP layers - see also Fig. 4). We used an inserted SPP layer that had a spatial dimension of $4 \times 4$. As can be seen, this provides a big boost in matching performance, suggesting the great utility of such an architecture when comparing image patches. Regarding the rest of the models, the observed results in Fig. 10 reconfirm the conclusions already drawn from previous experiments. We simply note again the very good performance of `siam-2stream-`$l_2$, which (although not trained with

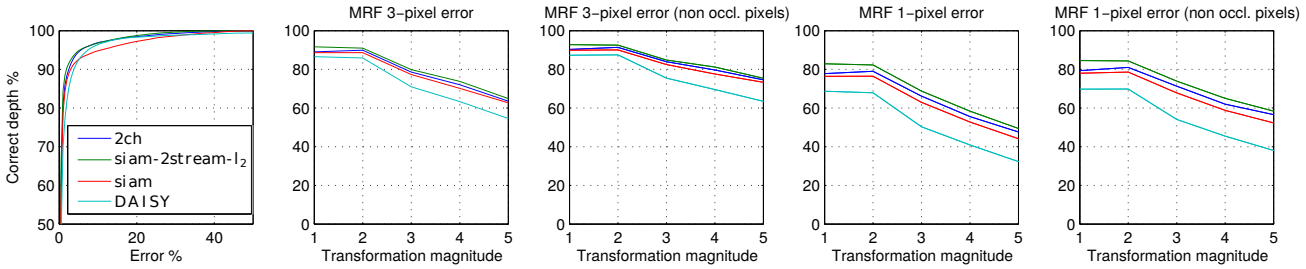



Figure 8. Quantitative comparison for wide-baseline stereo on "fountain" dataset. (Leftmost plot) Distribution of deviations from ground truth, expressed as a fraction of scene's depth range. (Other plots) Distribution of errors for stereo pairs of increasing baseline (horizontal axis) both with and without taking into account occluded pixels (error thresholds were set equal to 1 and 3 pixels in these plots - maximum disparity is around 500 pixels).

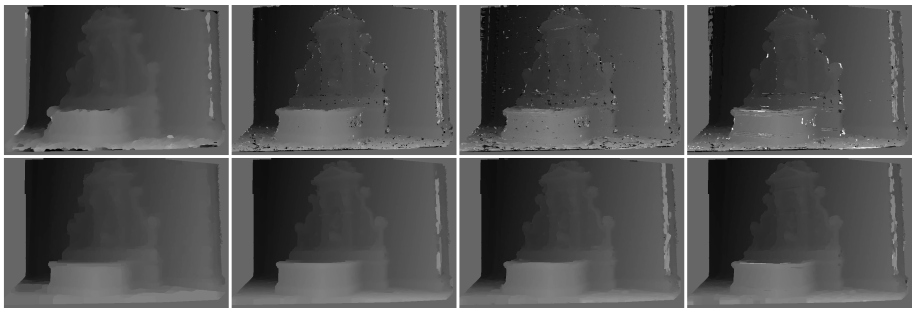


Figure 9. Wide baseline stereo evaluation. From left to right: DAISY, `siam-2stream-`$l_2$, siam, `2ch`. First row - "winner takes all" depthmaps, second row - depthmaps after MRF optimization.

$l_2$ distances) is able to significantly outperform SIFT and to also match the performance of imagenet-trained features (using, though, a much lower dimensionality of 512).

## 6. Conclusions

In this paper we showed how to learn directly from raw image pixels a general similarity function for patches, which is encoded in the form of a CNN model. To that end, we studied several neural network architecures that are specifically adapted to this task, and showed that they exhibit extremely good performance, significantly outperforming the state-of-the-art on several problems and benchmark datasets.

Among these architectures, we note that 2-channel-based ones were clearly the superior in terms of results. It is, therefore, worth investigating how to further accelerate the evaluation of these networks in the future. Regarding siamese-based architectures, 2-stream multi-resolution models turned out to be extremely strong, providing always a significant boost in performance and verifying the importance of multi-resolution information when comparing patches. The same conclusion applies to SPP-based siamese networks, which also consistently improved the quality of results[1].

[1]In fact, SPP performance can improve even further, as no multiple aspect ratio patches were used during the training of SPP models (such patches appear only at test time).

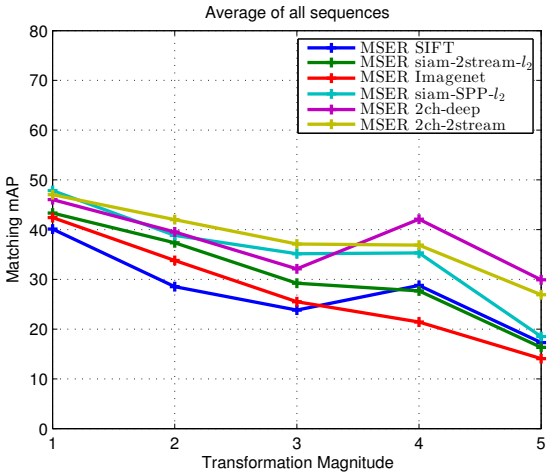



Figure 10. Evaluation on the Mikolajczyk dataset [16] showing the mean average precision (mAP) averaged over all types of transformations in the dataset (as usual, the mAP score measures the area under the precision-recall curve). More detailed plots are provided in the supplemental material due to lack of space.

Last, we should note that simply the use of a larger training set can potentially benefit and improve the overall performance of our approach even further (as the training set that was used in the present experiments can actually be considered rather small by today's standards).

## References


[1] X. Boix, M. Gygli, G. Roig, and L. Van Gool. Sparse quantization for patch description. In *CVPR*, 2013. 5

[2] J. Bromley, I. Guyon, Y. Lecun, E. Sckinger, and R. Shah. Signature verification using a "siamese" time delay neural network. In *NIPS*, 1994. 2

[3] M. Brown, G. Hua, and S. Winder. Discriminative learning of local image descriptors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2010. 2, 4, 5

[4] M. Brown, G. Hua, and S. Winder. Discriminative learning of local image descriptors. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 33(1):43–57, Jan 2011. 4

[5] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer. cudnn: Efficient primitives for deep learning. *CoRR*, abs/1410.0759, 2014. 4

[6] S. Chopra, R. Hadsell, and Y. LeCun. Learning a similarity metric discriminatively, with application to face verification. In *CVPR*, 2005. 2

[7] R. Collobert, K. Kavukcuoglu, and C. Farabet. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, 2011. 4

[8] B. Conejo, N. Komodakis, S. Leprince, and J.-P. Avouac. Inference by learning: Speeding-up graphical model optimization via a coarse-to-fine cascade of pruning classifier. In *NIPS*, 2014. 7

[9] D. Eigen, C. Puhrsch, and R. Fergus. Depth map prediction from a single image using a multi-scale deep network. In *NIPS*, 2014. 2

[10] P. Fischer, A. Dosovitskiy, and T. Brox. Descriptor matching with convolutional neural networks: a comparison to SIFT. *CoRR*, abs/1405.5769, 2014. 2, 4, 7

[11] K. He, X. Zhang, S. Ren, and J. Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. In *ECCV14*, pages III: 346–361, 2014. 4

[12] N. Komodakis, G. Tziritas, and N. Paragios. Fast, approximately optimal solutions for single and dynamic MRFs. In *CVPR*, 2007. 7

[13] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. Burges, L. Bottou, and K. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012. 1, 2

[14] Y. LeCun. A theoretical framework for back-propagation. In *Proceedings of the 1988 Connectionist Models Summer School*, pages 21–28, 1988. 1

[15] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60:91–110, 2004. 1, 2

[16] K. Mikolajczyk and C. Schmid. A performance evaluation of local descriptors. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 27(10):1615–1630, 2005. 2, 7, 8

[17] E. Nowak and F. Jurie. Learning Visual Similarity Measures for Comparing Never Seen Objects. In *CPVR 2007 - IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, Minneapolis, United States, June 2007. IEEE Computer society. 1

[18] A. S. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson. CNN features off-the-shelf: An astounding baseline for recognition. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR Workshops 2014, Columbus, OH, USA, June 23-28, 2014*, pages 512–519, 2014. 2

[19] K. Simonyan, A. Vedaldi, and A. Zisserman. Learning local feature descriptors using convex optimisation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2014. 2, 5, 6

[20] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. 3

[21] N. Snavely, S. M. Seitz, and R. Szeliski. Photo tourism: Exploring photo collections in 3d. *ACM Trans. Graph.*, 25(3):835–846, July 2006. 1

[22] N. Snavely, S. M. Seitz, and R. Szeliski. Modeling the world from internet photo collections. *Int. J. Comput. Vision*, 80(2):189–210, Nov. 2008. 1

[23] C. Strecha, W. von Hansen, L. J. V. Gool, P. Fua, and U. Thoennessen. On benchmarking camera calibration and multi-view stereo for high resolution imagery. In *CVPR*. IEEE Computer Society, 2008. 6

[24] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. J. Goodfellow, and R. Fergus. Intriguing properties of neural networks. *CoRR*, abs/1312.6199, 2013. 2

[25] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf. Deepface: Closing the gap to human-level performance in face verification. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014. 2

[26] E. Tola, V.Lepetit, and P. Fua. A Fast Local Descriptor for Dense Matching. In *Proceedings of Computer Vision and Pattern Recognition*, Alaska, USA, 2008. 2, 6

[27] T. Trzcinski, M. Christoudias, V. Lepetit, and P. Fua. Learning Image Descriptors with the Boosting-Trick. In *NIPS*, 2012. 2

[28] J. Zbontar and Y. LeCun. Computing the stereo matching cost with a convolutional neural network. *CoRR*, abs/1409.4326, 2014. 2, 7