Invariant Descriptor Learning Using a Siamese Convolutional Neural Network

L. Chen \*, F. Rottensteiner, C. Heipke

Institute of Photogrammetry and GeoInformation, Leibniz Universität Hannover - Germany

(chen, rottensteiner, heipke)@ipi.uni-hannover.de

Commission III, WG III/1

**KEY WORDS:** Descriptor Learning, Convolutional Neural Networks, Image Matching, Gradient Descent,

ABSTRACT:

In this paper we learn descriptor based on Siamese Convolutional Neural Network (CNN) architecture and evaluate our descriptor both on standard viewpoint change descriptor evaluation benchmark and real images. The descriptor learning architecture is composed of a input module, a CNN descriptor module and a cost computation module that is based on L2 Norm. The cost function we used pull the descriptor of patches in matched pairs close to each other in feature space while push those in unmatched pairs far away from each other. The training is in a mini-batch manner. By using a moving average strategy on gradients and momentum term we get the result of our training result. Experiments shows that the learned descriptor can cope with severe viewpoint change better than SIFT.

# introDuction

Feature based matching for finding pairs of homologous points in two different images is a fundamental problem in computer vision and photogrammetry, required for different tasks such as automatic relative orientation, image mosaicking or image retrieval. In general, for a feature based matching algorithms one needs to define a *feature detector*, a *feature descriptor* and a *matching strategy*. Each of these three modules is relatively independent from the others, therefore a combination of different detectors, descriptors and matching strategies is always possible and a good combination might adapt to some specific data configurations or applications. The key problem of image matching is to achieve *invariance against possible photometric or geometric transformations between images*. The list of photometric transformations that could affect the matching performance comprises illumination change or the use of different spectral bands in the two images. Geometric transformations comprise translation, rotation and scaling as well as affine and perspective transformation; besides, the matching performance may also be affected by self occlusion caused by a viewpoint change. In most cases, features for matching are extracted locally in the image, and the feature vectors (*descriptors*) used to represent the local image structure of a feature is extracted from a relatively small local image patch centered at the feature. Consequently, it is usually sufficient to design a matching strategy that is invariant to affine distortion, because a global perspective transformation can be approximated well by affine transformation locally (Morel and Yu, 2009). Such distortions are likely to occur in case of large changes of the view points and the viewing directions.

Classical descriptors, like SIFT (Lowe, 2004) and SURF (Bay et al., 2008) are designed manually; they are invariant to shift, scale and rotation, but not to affine distortions. Some authors (Mikolajczyk and Schmid, 2005; Moreels and Perona, 2007; Aanæs et al., 2012) have evaluated the performance of detectors and descriptors against different types of transformations in planar or 3D scenes, using recall and matching precision as the main evaluation criteria (Mikolajczyk and Schmid, 2005). As discussed in (Moreels and Perona, 2007), their results show that the performance of classical detectors and descriptors drops sharply when the viewpoint change becomes large, because the local patches vary severely in appearance, so that the tolerance of classical feature detectors and descriptors is exceeded.

One strategy to improve the invariance of descriptors to view point changes is to convert the descriptor design and descriptor matching into a pattern classification problem. By collecting the patches of the same feature in different images, one could clearly capture the real differences between these patches. The process of designing invariant feature descriptors is equal to finding a mapping of those patches into a proper feature space where they are more located closely to the descriptors of the homologous features. By using an appropriate machine learning model, a loss based on the similarity of the learned descriptors is designed. In this case, decreasing the loss by learning would help to achieve a higher level of invariance.

In this paper, we present a new method for defining descriptors based on machine-learning. It extends our previous descriptor learning work to Convolutional Neural Networks (CNN). As CNN has a natural "deep" architecture, we expect this architecture to have a stronger modelling ability that could be used to produce invariance against more challenging transformations, which classical manually designed descriptor cannot cope with.

# Related Work

A substantial body of classical descriptors are designed in a manual manner, for instance SIFT (Lowe, 2004) or SURF (Bay et al., 2008). More recent manually designed features like DAISY (Tola et al., 2010) introduced a more complex pattern of pooling operations to make descriptors more robust. These descriptors have been considered to be a standard for quite some time. However, they cannot deal with large viewpoint changes. This is why affine-invariant frameworks for feature based matching have been descigned (Morel & Yu, 2009). By using an affine view-sphere simulation strategy, ASIFT transforms the two original images that require for matching into many affine versions and then features and descriptors are computed based on those affine version images, afterwards the descriptors are matched against affine version images of the two original images. As each feature has many different descriptors that are built on simulated affine views, it cope with affine distortions better than only detect features and build descriptors in the original image. However, it is also computationally expensive.

An alternative to using hand-crafted features and strategies such a sampling many potentials viewpoints synthetically is to use descriptor learning (Bengio et al., 2013). To test if machine learning model could achieve better representations, Brown et al. (2011) proposed a descriptor learning framework, in which a descriptor is composed of four different modules: 1) Gaussian smoothing; 2) non-linear transformation; 3) spatial pooling or embedding; 4) normalization,by optimizing the configuration of the second and the third modules. An extension of their work to be able to apply convex optimization in the training process is given in (Simonyan et al., 2012). In (Trzcinski et al., 2012; 2015), a descriptor learning architecture based on the combination of weak learners by boosting is designed, in which the weak learners rely on comparisons of simple features. In the training process, the optimal features for the weak learners are determined along with the optimal matching score function. Their descriptor outperforms SIFT under nearly every type of transformation on the benchmark data set proposed in (Mikolajczyk and Schmid, 2005). However, those are all one layer based method and has a risk of under-fitting the real transformation in data.

Another category of descriptor learning framework is built on CNN, which are becoming increasingly popular in computer vision. CNN consist of multiple convolutional layers (LeCun et al. 1998). Invariant feature representation learning based on a so-called *Siamese CNN* has originally been proposed in (Bromley et al., 1993) to extract feature representation for signature verification, where the signature from one person may changes complexly that no explicit model could describe. Later in (Hadsell et al., 2006), the siamese CNN architecture is used to learn feature representation for digit recognition; as the same digit may vary from each other a lot, a Siamese CNN architecture is used to find invariant feature representation that could map the high dimensional input data into a more discriminative lower manifold feature space where "similar" digits locates more close to each other. As the architecture use the same CNN architecture to obtain and train representations, it is called "Siamese". The CNN finds a proper mapping that can cope with complex transformations and map those quite differing patterns (in pixel or lower level feature space) to a closer position in a more sophisticated feature space, which is defined by the output features of the final convonlutional layer in the CNN. The use of multiple layers (i.e., its *deep architecture*) is the reason for the strong modelling ability of CNNs. This property of CNN fits well with the requirements for learning descriptors that are invariant against various types of transformations. Consequently, they have been used to train descriptors for patch comparison.

The first patch comparison work based on the Siamese CNN is present in (Jahrer et al., 2008). Jahrer et al used the Siamese CNN to train the descriptor and compare the patches, but the training data is generated from image warps and dependent on input images, which makes this method less practical while it always needs a prior simulation and training before image matching. In (Osendorfer et al., 2013), a Siamese CNN is used to train descriptor and they focus on the comparison of four different types of loss functions. More recently, this Siamese architecture are used to train patch descriptor to cope with dynamic lighting conditions (Carlevaris-Bianco and Eustice, 2014) and ground-to-aerial viewpoint change (Lin et al., 2015).In Carlevaris-Bianco and Eustice's work, patches taken from severe illumination change are fed into a Siamese CNN and they achieve illumination invariance that exceeds any hand-crafted descriptors. In (Lin et al., 2015), images containing same scene (mostly streetview) taken from aerial and terrestrial view are fed into a Siamese CNN network and a similarity function is therefore trained via this architecture. By that way they achieved a performance improvement in ground-to-aerial geolocalization.

Our work is closely linked to the work in (Han et al., 2015; Zagoruyko and Komodakis, 2015; Zbontar and Lecun, 2015). In (Han et al., 2015; Zagoruyko and Komodakis, 2015, they did not only train the descriptor, but also the matching function, which is called the *decision layer* (Zagoruyko and Komodakis, 2015) and metric network (Han et al., 2015) in their work, which makes their model more complicated than ours. As we focus on feature based image matching, we do not train matching function for each pair of patches. In (Zbontar and Lecun, 2015) they also calculate a four extra layers of metric network but apply them in wide baseline stereo matching and achieve currently the best result on KITTI benchmark. In contrast to these works, we only train descriptor without metric function for 2 patches. If one trained similarity function for a pair of patches, then when this descriptor is applied in real image matching or large scale image retrieval, every pair of feature patches should be fed into the network with metric layers. In this case, the highly efficient search strategies such as Best Bin First (Beis and Lowe, 1997) in a KD tree could not be used and the real matching speed will be seriously influenced. This reduces the practical value of a learned descriptor in feature based image matching. In addition, we also present a clear detail of our optimizing method, while other related works give quite few information about this part, that is might because of the usage of open-source CNN library which could run the learning procedure automatically without instructions about the parameters updating.

# METHODOLOGY

## Siamese Descriptor Learning Architecture

In order to learn the CNN-based descriptor, we need pairs of training patches of which we know whether they represent homologous image features or not. In this context, it is important that the set of positive examples (the pairs that correspond to homologous key points) contains transformations that the learned descriptor should be tolerant to. The basic idea of the Siamese architecture for descriptor learning is to apply the same type of CNN using the same set of parameters  to the each of the patches that should be checked for correspondence and determine these parameters  by optimising a loss function of the L2 norm of the differences of the resultant descriptors. That is, by adjusting the parameters so that the L2 norm is as discriminative as possible we obtain a descriptor that should be tolerant to the type of geometric distortions that occur between positive examples in the training data. Please refer to Figure 1 for an illustration of the whole architecture.

right patch

CNN

left patch

CNN

L2 Norm Distance

Loss *L*

*θ*





Figure 1. The architecture for Siamese CNN descriptor learning used in this paper. Green: input patches; Red: a CNN as depicted in figure 1; Blue: loss function. The two CNNs share the learned parameters  (orange).

In the training process, the following loss function based on the L2 norm of the differences of the CNN descriptors of both patches are used:

 (1)

where *N*  *=* number oftraining samples

*i =* index of a training sample

*yi* *=* label for a patch pair: 1 for matching training

pairs, 0 for unmatched pairs.

*Dik =* CNN descriptors for patch *k*, with *k ∈ {l, r}*  
 indicating the left or right patch, respectively

*|| . ||2 =* L2 norm of the differences between the   
 descriptors of the two patches

*lpush* *=* Push radius for dissimlar pairs

*lpull* *=* Pull radius for similar pairs

The above loss function creates a margin between matching pairs and pairs that do not correspond. For matching pairs, a distance larger than a “pull radius” *lpull* will be penalised, whereas for the negative training examples, penalisation occurs for distances smaller than a “push radius” *lpush*. The two radii are parameters that have to be set by the user. Before the learning, the CNN parameters are initialised at random, so that the distances of descriptors from matching pairs are not close to each other. The learning procedure will try to find parameters  of the CNN that could push the unmatched pairs far away while pull the matched pairs close to each other. An illustration of this idea is shown in Figure 2.

**Before Learning**

**After Learning**



feature space

feature space

Figure 2. Descriptor learning. In the top part, each coloured dot represents a descriptor; identical colours indicate the patches from the same features (patches extracted from multi-view images). In the "after learning" part, the radius of the inner concentric circle equals to *lpull* and the radius of the outer one equals to *lpush*.

## CNN Descriptor

The concept of CNNs was proposed by (LeCun et al. 1998); it is a multi layer neural network. A CNN may have one or several stages consisting of a convolution layer, a nonlinear layer and a feature pooling layer each. Compared to general multi layer neural networks, there are two main differences:

1) In the convolution layer, the neurons of the input layer are not fully connected to those of the next layer and weights are shared, so that the same weights are repeatedly used across the different position of the input layer. This is the reason for using the term "convolutional" network. The weight sharing strategy dramatically decreases the number of parameters and makes possible deep architectures consisting of larger numbers of stages.

2) The network is able to deal with a large input layer, for instance related to images taken from consumer cameras, and it decreases the layer size in successive stages by pooling layers. Thus, the input can be compressed into a meaningful feature representation, which reduces the dimension of the original data considerably.

In essence, a CNN can be seen as is a nonlinear mapping function, transforming the input (a given image patch) to a high-level feature representation .

In this paper, we use a CNN architecture consisting of three stages to learn feature descriptors (cf. figure 3). Details about the architecture and the learning parameters are listed in table 1. The input patch size is 32 by 32 pixels and the unit of the whole table is pixel. It contains three stages, in which both of the first two stages has a [convolution - nonlinear - pooling] structure and the third one contains only one convolution layer. The parameters *w1*, *w2*, *w3* and *b1*, *b2*, *b3* are the convolution kernel and bias term used for stage 1, 2 and 3 respectively. We also record the parameters as *k=1,2,3* and *wk*, *bk* for short. The nonlinearity is calculated by sigmoid function and the pooling method is max pooling (without overlapping) where the biggest value in the 2 x 2 input is preserved as the output. The final output of the CNN used in our work is a 125 dimensional vector. This 125 dimensional vector is the learned descriptor that is used to represent the current feature whose surrounding patch is the input of the CNN we used.

Figure 3. The CNN used in this paper to learn descriptor

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Input | Convolution kernels | Nonlinear | Pooling | Output | Learning parameters |
| Stage 1 | 32 x 32 | 5 x 5 x 5 | sigmoid | max (2 x 2) | 14 x 14 x 5 | *w1 , b1* |
| Stage 2 | 14 x 14 x5 | 25 x 5 x 5 x 5 | sigmoid | max (2 x 2) | 5 x 5 x 25 | *w2 , b2* |
| Stage 3 | 5 x 5 x 25 | 125 x 5 x 5 x 25 | ~ | ~ | 1 x 1 x 125 | *w3 , b3* |

Table 1. detailed architecture and learning parameters for used CNN in this paper.

This CNN architecture is different from (Han et al., 2015; Zagoruyko and Komodakis, 2015; Zbontar and Lecun, 2015) and this is the first time it is used in this context. In comparison, a smaller input with 32 x 32 pixels instead of 64 x 64 pixels that are used in their work is used. Additionally, sigmoid function is applied to achieve nonlinearity because we found it performs better than Rectified Linear Unit (ReLU) in our experiment.

## Training of the Siamese CNN

Training of the CNN is based on online gradient descent to find the optimum of the loss function. In this context, the well-known back propagation algorithm (Rumelhart et al., 1986) can be used to get the derivatives of the loss with respect to the parameters. In our network, back-propagation is a little more complicated than usually, because the gradients are influenced by each of the two subnets in the Siamese CNN as we introduced before. In Section 3.3.1 the online gradient training procedure is described, whereas Section 3.3.2 gives details about the way in which gradients are computed.

groups of homologous patches. Before learning, they are distributed randomly in the feature space, while after learning the descriptor from patches of the same interest point are close to each other.

**3.3.1 Online gradient descent:** In general, after calculating the gradient of the loss with respect to the learning parameters, the learning parameters are updated according to the gradient with a learning rate. As the gradient can be calculated for each single training sample, the learning parameters can be updated after each training case, that is called online learning. However, when the training data contains a huge number of training samples, that updating strategy is quite inefficiency and unstable as the sampling error exists in training data. A better way is to update the parameter after a small group of training cases, typically several hundred of training samples are formed as a small training data group and they are called a mini-batch. The gradients are calculated on those several hundred samples and then averaged. In this work, the mini-batch training is used for updating parameters.

To update the learning parameter, we need also consider how to adjust the parameter. One naive way is gradient descent with only a learning rate. However, we observe that it doesn't work for our cases because of oscillation and rather slow decrease of error. To tackle them, we used the momentum method to have an accumulated gradient and it enables a more consistent decrease of error during training. Another observation is that the loss in adjacent batches oscillates a lot; the moving average of the root mean (Tieleman and Hinton, 2012) squared gradient is used to ensure that the gradient calculated by adjacent mini-batches are divided by a more similar value. The details of those methods are explained in Algorithm 1.

Algorithm 1. The training algorithm used in this paper

|  |  |  |  |
| --- | --- | --- | --- |
| Data: | | Training patch pairs | |
| Hyper-parameters: | | *Num\_Epochs* | Number of epochs |
|  | | *Num\_Batches* | Number of batches |
|  | | *β* | Momentum term |
|  | | *γ* | Decay term for root mean gradients |
|  | | *α* | Learning rate |
|  | | *α\_decrease* | Decrease rate of the learning rate after each epoch |
| ***for epoch\_index = 1:Num\_Epochs*** | | | |
|  | Random divide training samples into small batches | | |
|  | ***for batch\_index=1:Num\_Batches*** | | |
|  | * Read the current batch data * Central normalization of data * Calculate *Dr* and *Dl* by forward propagation * Calculate Current loss *L* * Calculate gradients *g(θ)* by back propagation ( see last section) * Update learning parameters *θ*      * Record parameters for next round | | |
|  | ***end (for batch\_index =1:Num\_Batches)*** | | |
|  | Decrease the learning rate  *α = α\_decrease\*α* | | |
| ***end (for epoch\_index = 1:Num\_Epochs)*** | | | |
| Output: Optimized parameters *wk*, *bk,* where *k=1,2,3* | | | |

**3.3.2 Gradient computation:** The loss function is calculated based on the distance. To decrease the risk of over-fitting, [regularization](javascript:void(0);) for the parameters to be learned *w1*, *w2*, *w3* are also used. Thus the loss function in (1) is added by regularization terms.



where *s* is the shrink rate in the regularization term.

The derivative the loss with respect to the distance *di* = *|| Dil - Dir||2* is calculated by:



where (.) is a indicator function that it equals to 1 if the argument is true and 0 otherwise. The derivatives of the distance *di* with respect to the descriptors D*il* and D*ir* are:



The derivatives of D*il* and D*ir* with respect to each parameter *wk* to be learned are calculated by normal back propagation. Since both subnets contribute to the loss, the derivatives of the loss function with respect to each parameter *wk* must be summed over the two subnets.



where *k=1,2,3* and *wk*, *bk* represents all the learning parameters.

# Experiments

We first report the training and testing of our model on standard image patch dataset, then the trained model is used to test its matching performance against viewpoint change on standard local descriptors performance evaluation data. Furthermore, we also test the performance of the learned descriptor model on real photogrammetric image matching.

## Experiment Setup

To train the network we introduced before, the Brown dataset (Brown et al., 2011) is used. The dataset contains three different subsets three separate datasets - Notre Dame, Yosemite and Liberty. Those patches are extracted surrounding Difference of Gaussian (DoG) feature points on real multi-view images, thus the real transformation in 3D scene with viewpoint changes is contained in those datasets. The original patch size is 64 x 64 pixels, to cope with our model it is resized to 32 x 32 pixels. The number of positive and negative training pairs are both 250,000 in each dataset and the training data is same to the one used in (Brown et al., 2011).

To implement the method, matconvnet (Vedaldi and Lenc, 2014), which is an opensource convnet[[1]](#footnote-1), is used to conduct the convolution, pooling, sigmoid and back-propagation for those basic layers. We implement the Siamese model, back-propagation for the whole architecture and optimizing strategy ourselves. The local descriptor performance evaluation is implemented based on vlfeat[[2]](#footnote-2) and vlbenchmarks[[3]](#footnote-3) (Lenc, Gulshan et al., 2012).

## Descriptor Learning

The descriptor is trained in each of the three datasets separately, and the other two datasets are test datasets. The training hyper-parameters we used are listed here. *Num\_Epochs* = 100, *Num\_Batches* = 1000, *β* = 0.9,γ = 0.9, *α = 3e-5*, *α\_decrease* = 0.99, *lpull* = 3, *lpush* = 10. Addtionaly, the size of mini-batch is 500.

In order to get the ROC curve of features, we first calculate the descriptor for each patch in the test datasets using CNN with learned parameters. Then the L2 Norms of the two descriptors in the training pairs is calculated as the similarity measure of the pair of patches. For a distance threshold *t*, a patch pair with L2 Norm under *t* are judged as matched pair, otherwise the current pair is judged as unmatched pair. Afterwards the true positive and false positive rate could be calculated for *t*. By changing *t* the ROC curve is achieved. When the true positive rate equals to 95%, the corresponding false positive rate is also reported for evaluation and comparison.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Train | Test | Our Method | Simonyan | SIFT | BinBoost |
| ND | Yos | 18.5% | 10.08% | 29.15% |  |
| ND | Lib | 14.9% | 12.42% | 36.27% |  |
| Lib | ND | 7.7% (7.66%) | 7.22% | 28.09% |  |
| Lib | Yos | 18.9% (18.84%) | 11.18% | 29.15% |  |
| Yos | ND | 9.3% | 6.82% | 28.09% |  |
| Yos | Lib | 17.1% | 14.58% | 36.27% |  |
|  |  |  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Train Test | Notre Dame | Yosemite | Liberty |
| Notre Dame |  |  |  |
| Yosemite |  |  |  |
| Liberty | 7.7%(7.66%) |  |  |

## Evaluation for Performance against view-point Change.

Following the work in (Mikolajczyk and Schmid, 2005), we use *recall* and *1-precision* as evaluation criteria.





where *#correspondences* is the number of ground truth correspondences, whereas *#correct matches* is the number of correct matches from the matching result based on the descriptor to be evaluated. This criterion gives the proportion of all potential matches that are detected based on the descriptor. A perfect result would give a *recall* equal to 1, which means the tested descriptor can find all potential matches between the two tested images. (refer to the PIA 2015 paper while I use almost the same words here?)

The second criterion, *1-precision*, is defined as the number of false matches divided by the total number of matches. By varying the nearest neighbor distance ratio, one can get different *recall* and *1-precision* values that are used to generate performance curve. For the interpretation of the figures and possible curve shapes please refer to (Mikolajczyk and Schmid, 2005).

## Real Image Pairs Matching

3 stage and 4 stage model could be compared together to

# Discussion and Conclusion

*ability of invariance.*

*training data.*

*transfer learning.*

*could be used to learn descriptor. It could also be extended to train descriptor for different source matching tasks like laser intensity image to optical image matching and SAR image to optical remote sensing image, with the proper feature detectors*.

# Acknowledgement

The author Lin Chen would like to thank the China Scholarship Council (CSC) for supporting his PhD study in Leibniz Universität Hannover, Germany.

# References

Aanæs, H., Dahl, A. L., & Pedersen, K. S., 2012. Interest-ing interest points. International Journal of Computer Vision, 97(1), 18-35.

Bay, H., Ess, A., Tuytelaars, T., et al., 2008. Speeded-up robust features (SURF). Computer Vision and Image Understanding, 110(3), pp. 346-359.

Beis, J. S., & Lowe, D. G., 1997. Shape indexing using approximate nearest-neighbour search in high-dimensional spaces. In IEEE Computer Society Conference on Computer Vision and Pattern Recognition, pp. 1000-1006.

Bengio, Y., Courville, A., & Vincent, P., 2013. Repre-sentation learning: A review and new perspectives. IEEE Transactions on Pattern Analysis and Machine Intelligence 35(8), 1798-1828.

Bromley, J., Bentz, J. W., Bottou, L., Guyon, I., LeCun, Y., Moore, C., ... & Shah, R., 1993. Signature verification using a “Siamese” time delay neural network. *International Journal of Pattern Recognition and Artificial Intelligence*, *7*(04) pp., 669-688.

Brown, M., Hua, G., Winder, S., 2011. Discriminative learning of local image descriptors. IEEE Transactions on Pattern Analysis and Machine Intelligence, 33(1), pp. 43-57.

Carlevaris-Bianco, N., & Eustice, R. M. (2014). Learning visual feature descriptors for dynamic lighting conditions. In International Conference on Intelligent Robots and Systems (IROS 2014), pp. 2769-2776.

Fischer, P., Dosovitskiy, A., & Brox, T. (2014). De-scriptor matching with convolutional neural networks: a comparison to SIFT. arXiv preprint arXiv:1405.5769.

Hadsell, R., Chopra, S., & LeCun, Y. (2006). Dimension-ality reduction by learning an invariant mapping. In IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Vol. 2, pp. 1735-1742.

Han, X., Leung, T., Jia, Y., Sukthankar, R., & Berg, A. C., 2015. MatchNet: Unifying Feature and Metric Learning for Patch-Based Matching. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. pp. 3279-3286.

Jahrer, M., Grabner, M., and Bischof, H., 2008. Learned local descriptors for recognition and matching. In Computer Vision Winter Workshop. Moravske Toplice, Slovenia.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P., 1998. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324.

K. Lenc, V. Gulshan, and A. Vedaldi, 2012. VLBenchmarks,

<http://www.vlfeat.org/benchmarks/> (visited 25/11/2015).

Lin, T. Y., Cui, Y., Belongie, S., Hays, J., & Tech, C., 2015. Learning Deep Representations for Ground-to-Aerial Geolocalization. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 5007-5015.

Lowe, D. G., 2004. Distinctive image features from scale-invariant keypoints. International Journal of Computer Vision, 60(2), pp. 91-110.

Mikolajczyk, K., Schmid, C., 2005. A performance evaluation of local descriptors. IEEE Transactions on Pattern Analysis and Machine Intelligence, 27(10), pp. 1615-1630.

Moreels, P., & Perona, P., 2007. Evaluation of features detectors and descriptors based on 3d objects. International Journal of Computer Vision, 73(3), 263-284.

Morel, J. M., & Yu, G. (2009). ASIFT: A new framework for fully affine invariant image comparison. SIAM Journal on Imaging Sciences, 2(2), 438-469.

Osendorfer, C., Bayer, J., Urban, S., & van der Smagt, P., 2013. Convolutional Neural Networks learn compact local image descriptors. In Neural Information Processing, Springer Berlin Heidelberg, pp. 624-630.

Rumelhart, D. E., Hintont, G. E., & Williams, R. J., 1986. Learning representations by back-propagating errors. *NATURE*, 323(9), pp. 533-536.

Simonyan, K., Vedaldi, A., Zisserman, A., 2012. Descriptor learning using convex optimisation. In: European Conference on Computer Vision, pp. 243-256.

Tieleman, T. and Hinton, G. (2012), Lecture 6.5 - rmsprop, COURSERA: Neural Networks for Machine Learning

Tola, E., Vincent, L., Fua, P., 2010. Daisy: An efficient dense descriptor applied to wide-baseline stereo. IEEE Transactions on Pattern Analysis and Machine Intelligence, 32(5), pp. 815-830.

Trzcinski, T., Christoudias, M., Lepetit, V., et al, 2012. Learning image descriptors with the boosting-trick. Advances in neural information processing systems, pp. 269-277.

Trzcinski, T., Christoudias, M., & Lepetit, V.. 2015. Learning image descriptors with boosting. IEEE Transactions on Pattern Analysis and Machine Intelligence, 37(3) pp. 597-610.

Tuytelaars, T., & Mikolajczyk, K., 2008. Local invariant feature detectors: a survey. Foundations and Trends® in Computer Graphics and Vision, 3(3), 177-280.

Vedaldi, A., & Lenc, K.. 2014. MatConvNet-convolutional neural networks for MATLAB. arXiv preprint arXiv:1412.4564.

Zagoruyko, S., & Komodakis, N., 2015. Learning to Compare Image Patches via Convolutional Neural Net-works. arXiv preprint arXiv:1504.03641.

Zbontar, J., & LeCun, Y., 2015. Computing the Stereo Matching Cost With a Convolutional Neural Network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. pp. 1592-1599.

1. http://www.vlfeat.org/matconvnet/ (accessed 25 November 2015) [↑](#footnote-ref-1)
2. http://www.vlfeat.org/ (accessed 25 November 2015) [↑](#footnote-ref-2)
3. http://www.vlfeat.org/benchmarks/ (accessed 25 November 2015) [↑](#footnote-ref-3)