Deep Learning Final Report

Koral Hassan
Imperial College London
Kensington, London SW7 2AZ
kbh15@ic.ac.uk

Abstract

De-noising and representation learning is to be used for generating a patch descriptor that is able to perform tasks such as matching, retrieval or verification successfully. HPatches is used as the benchmark. We first establish baseline models, then seek improvements by varying hyperparameters and architectures.

1. Introduction

1.1. Problem Formulation

We will be judging the performance of a patch descriptor on matching, retrieval and verification tasks. However, our patches contain random non-smooth perturbations produced by synthetic noise.

This noise is critical when training the descriptor, so we introduce a denoising model to help as a preprocessing stage. The objective of this stage is for the *denoised* images obtained from the *noisy* ones to resemble their *clean* counterparts as closely as possible.

The denoised images will then be fed into the patch descriptor, which will output 128-dimensional feature vectors (also called descriptors) that should be accurate representations of the input patches.

For a given set of patches $Patch_A$, $Patch_B$, $Patch_C$, if $Patch_A$ is more similar to $Patch_B$ than $Patch_C$, then its feature vector should be closer to (i.e. have a lower euclidian distance to) that of B:

$$\|D_A - D_B\| \leq \|D_A - D_C\|$$

where D are feature vectors/descriptors.

1.2. Dataset

A modified version of the HPatches dataset [1] will be used for training and evuating the models, which will be referred to as the N-HPatches dataset. The modified images contain random non-smooth perturbations produced by a synthetic noise. All images are 32×32 pixels and grayscale. The noisy and clean versions of one particular image are shown in Appendix A.

The original HPatches dataset has several splits, which are used to separate the available sequences in train sequences and test sequences. For our experiments in N-HPatches we use the same splits as in HPatches. Specifically, we load (and report results) using the split 'a'.

1.3. Evaluation

Evaluation is performed on 3 different tasks;

- verification: classifying whether two patches are extracted from the same measurement,
- matching: identifying correspondances in two images,
- and identification: matching a query match to a pool of patches extracted from many images.

The definition of the tasks is taken from HPatches and HPatches benchmark code is used to compute the results for the models. [1] The score for each task is computed using the mean Average Precision (mAP) metric.

2. Baseline Models

2.1. Denoiser

The denoising is done by a shallow U-Net. [5] Its architecture can be seen in Figure 1 and is visualised in Appendix B.

The model contains a skip connection between encoder and decoder. This feeds spatial information from the primary (and wider) layers to the output explicitly. It also alleviates the vanishing gradient.

All pooling windows and upsampling factors are 2×2. Since there is only one of each the size of the bottleneck is 16×16. All convolutional layers use 3×3 kernels sampled from truncated normal distributions.

Up to 80 filters are stacked up and then reduced back to one which produces our denoised ouput image. Each layer (except the last one) uses a ReLu activation function and

Layer Type	Output Shape	Param #
InputLayer	(32, 32, 1)	0
Conv2D	(32, 32, 16)	160
MaxPooling2D	(16, 16, 16)	0
Conv2D	(16, 16, 32)	4640
UpSampling2D	(32, 32, 32)	0
Conv2D	(32, 32, 64)	8256
Concatenate	(32, 32, 80)	0
Conv2D	(32, 32, 64)	46144
Conv2D	(32, 32, 1)	577

Figure 1: Architecture of baseline denoiser.

same-padding. All in all, the model contains 59777 trainable parameters.

Figure 2 shows the noisy, denoised and clean versions of one image.

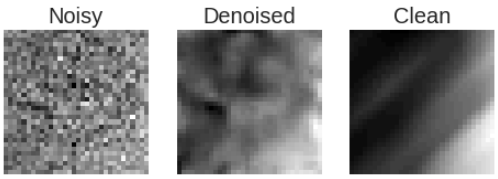


Figure 2: Noisy, denoised and clean versions of a training datapoint after 1 epoch.

2.2. Descriptor Network

The architecture of the descriptor network is copied from HardNet. [3] It is summarised in Figure 3 and visualised in Appendix C.

It has 7 convolutional layers, each of them (except the last one) followed by batch normalizations and ReLu activation functions. They all use same-padding. The main advantage of the batch normalizations is allow faster convergence. It has also been shown to enable higher learning rates.

The model utilises strides of 2 to reduce the width of the image from 32 to 16 to 8, and then uses a final convolutional layer with an 8×8 kernel which essentially functions as a dense layer with one node. Meanwhile, the number of filters goes from 32 to 64 to 128. The 128 channels, each with one output node, gives us our final feature vector.

The model also uses dropout just before the final layer. This is especially important before dense layers because it prevents over-fitting by making the neurons less dependent on specific connections.

All inall, the model contains 1336928 trainable parameters.

Layer Type	Output Shape	Param #
Conv2D	(32, 32, 32)	320
BatchNormalization	(32, 32, 32)	128
Activation	(32, 32, 32)	0
Conv2D	(32, 32, 32)	9248
BatchNormalization	(32, 32, 32)	128
Activation	(32, 32, 32)	0
Conv2D	(16, 16, 64)	18496
BatchNormalization	(16, 16, 64)	256
Activation	(16, 16, 64)	0
Conv2D	(16, 16, 64)	36928
BatchNormalization	(16, 16, 64)	256
Activation	(16, 16, 64)	0
Conv2D	(8, 8, 128)	73856
BatchNormalization	(8, 8, 128)	512
Activation	(8, 8, 128)	0
Conv2D	(8, 8, 128)	147584
BatchNormalization	(8, 8, 128)	512
Activation	(8, 8, 128)	0
Dropout	(8, 8, 128)	0
Conv2D	(1, 1, 128)	1048704
Reshape	(128)	0

Figure 3: Architecture of baseline descriptor.

2.3. Training

Each model uses Stochastic Gradient Descent (SGD) as their optimizer.

The denoiser is trained for one epoch with batches of 50 images. It uses an initial learning rate of 0.00001 and a Nesterov momentum [4] of 0.9 . The loss computation is mean absolute error.

The descriptor on the other hand, uses a learning rate of 0.1 (thanks in part due to batch normalization) and does not use a momentum term (i.e. momentum is 0, learning rate is constant).

Each of the patches in our dataset is related to a subset of other patches of the dataset by some kind of geometric transformation (e.g. rotation). For a given patch, we want the descriptor to output a vector that is close to the vectors of the patches that represent the same local part of a scene, while being far from patches do not represent that local part of a scene.

To this end, our descriptor network uses a triplet loss function. We compute the descriptor for an *anchor point*, a related *positive point* and an unrelated *negative point*. We then aim to minimize the distance between the anchor point and positive point, while maximizing the distance between the anchor point and the negative point. Figure 4 provides some intuition about the reasoning behind the triplet loss calculation.

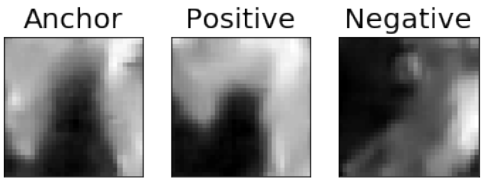


Figure 4: Random triplet in the form of anchor, positive and negative sample.

Verification	Matching	Retrieval
0.69	0.10	0.34

Figure 5: mAP scores after 1 epoch.

2.4. Evaluation

Figure 5 show the results our baseline model achieved in the 3 categories.

3. Advanced Methods

3.1. Converging Further

First of all, we will run the model for more epochs and see if this improves our validation scores. It is likely that we are far from overfitting or even converging given the size of our dataset and the complexity of our models.

Figure 6 shows the mAP scores improving. However, we can see that our model converged before 30 epochs so we could have spent less time and computational resources training.

The final scores after 30 epochs can be seen in Figure 7. We can also see that our denoising is working much better by having a look at Figure 8.

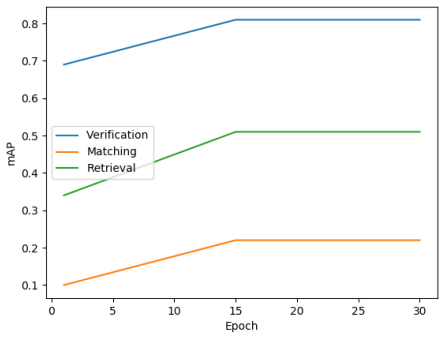


Figure 6: mAP scores as a function of number of epochs.

Verification	Matching	Retrieval
0.81	0.22	0.51

Figure 7: mAP scores after 30 epochs.

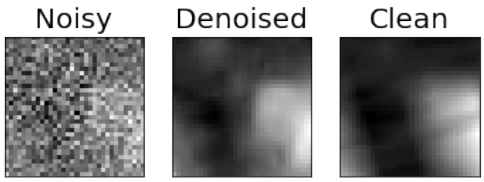


Figure 8: Noisy, denoised and clean versions of a training datapoint after 30 epochs.

3.2. DnCNN

It is likely that the weakest link in the baseline method is the denosier, since it is an overly simplistic model compared to the discriminator. We will now use a different model as our denoiser. The new model is a type of feed-forward denoising convolutional network (DnCNN) that utilises residual learning and batch normalisation.

The architecture [6] is summarised in Figure 9 and visualised in Appendix D. All in all, it has 556096 trainable parameters. Further details on it can be found at the source repository.

We train the DnCNN for only one epoch and this takes approximately one hour. It is worthwhile to note that this is prohibitively longer than our baseline denoiser. We also change the optimizer from SGD to Adam [2] and the initial learning rate from 0.00001 to 0.001 .

Figure 10 shows the mAP scores and Figure 11 show the denoiser in action. The results are very impressive.

4. Conclusion

We can improve our mAP scores as well the qualitative success of our denoising by running more epochs on our baseline. It will converge before 15 epochs which is very managable. Using a more complicated DnCNN for denoising is possible and would outperform our baseline denoiser. However this method is very computationally expensive.

The code for this research is at https://imperiallondon-my.sharepoint.com/:f:/g/personal/kbh15_ic_ac_uk/ along with insctructions for use (within .zip file).

Layer Type	Output Shape	Param #
Conv2D	64	640
Activation	64	0
Conv2D	64	36864
BatchNormalization	64	256
Activation	64	0
Conv2D	64	36864
BatchNormalization	64	256
Activation	64	0
Conv2D	64	36864
BatchNormalization	64	256
Activation	64	0
Conv2D	64	36864
BatchNormalization	64	256
Activation	64	0
Conv2D	64	36864
BatchNormalization	64	256
Activation	64	0
Conv2D	64	36864
BatchNormalization	64	256
Activation	64	0
Conv2D	64	36864
BatchNormalization	64	256
Activation	64	0
Conv2D	64	36864
BatchNormalization	64	256
Activation	64	0
Conv2D	64	36864
BatchNormalization	64	256
Activation	64	0
Conv2D	64	36864
BatchNormalization	64	256
Activation	64	0
Conv2D	64	36864
BatchNormalization	64	256
Activation	64	0
Conv2D	64	36864
BatchNormalization	64	256
(Activation	64	0
Conv2D	64	36864
BatchNormalization	64	256
Activation	64	0
Conv2D	64	36864
BatchNormalization	64	256
Activation	64	0
Conv2D	64	36864
BatchNormalization	64	256
Activation	64	0
Conv2D	1	576
Subtract	1	0

Figure 9: Architecture of DnCNN.

Verification	Matching	Retrieval
0.82	0.23	0.52

Figure 10: mAP scores of DnCNN.

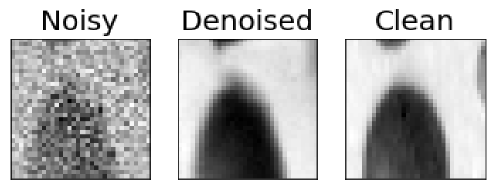


Figure 11: Noisy, denoised and clean versions of a training datapoint after being processed by the DnCNN.

References

[1] V. Balntas, K. Lenc, A. Vedaldi, and K. Mikolajczyk. Hpatches: A benchmark and evaluation of handcrafted and learned local descriptors. In *CVPR*, 2017.

[2] D. Kingma and J. Ba. Adam: a method for stochastic optimization (2014). *arXiv preprint arXiv:1412.6980*, 15, 2015.

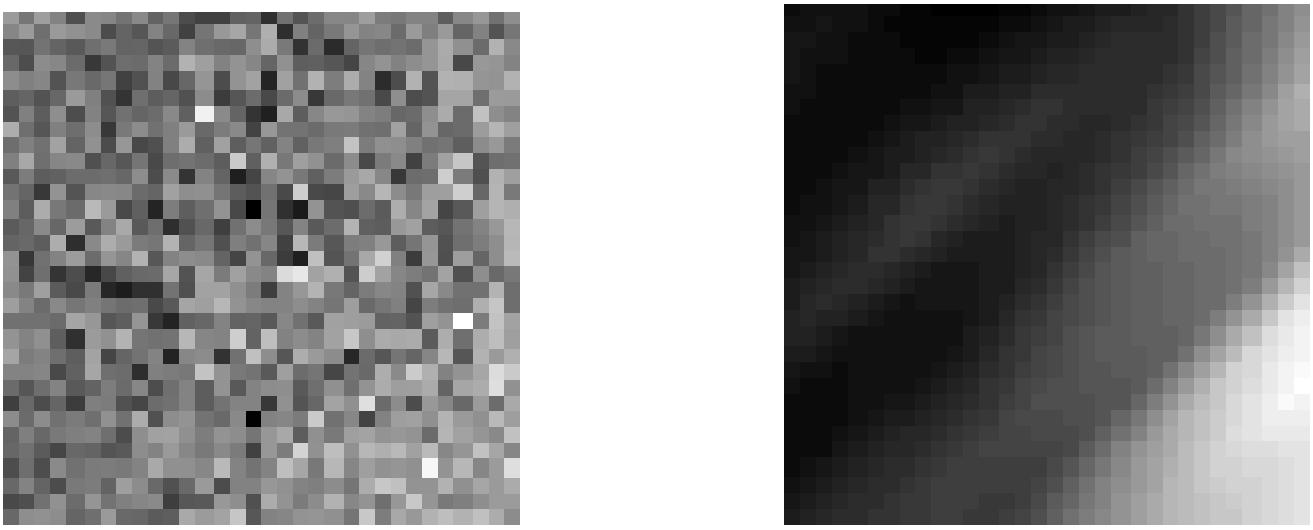
[3] A. Mishchuk, D. Mishkin, F. Radenovic, and J. Matas. Working hard to know your neighbor's margins: Local descriptor learning loss. In *Advances in Neural Information Processing Systems*, pages 4826–4837, 2017.

[4] Y. E. Nesterov. A method for solving the convex programming problem with convergence rate o (1/kˆ 2). In *Dokl. akad. nauk Sssr*, volume 269, pages 543–547, 1983.

[5] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. *Medical Image Computing and Computer-Assisted Intervention MICCAI 2015*, page 234241, 2015.

[6] K. Zhang, W. Zuo, Y. Chen, D. Meng, and L. Zhang. Beyond a gaussian denoiser: Residual learning of deep cnn for image denoising. *IEEE Transactions on Image Processing*, 26(7):3142–3155, 2017.

A. Exemplar Noisy and Clean Image

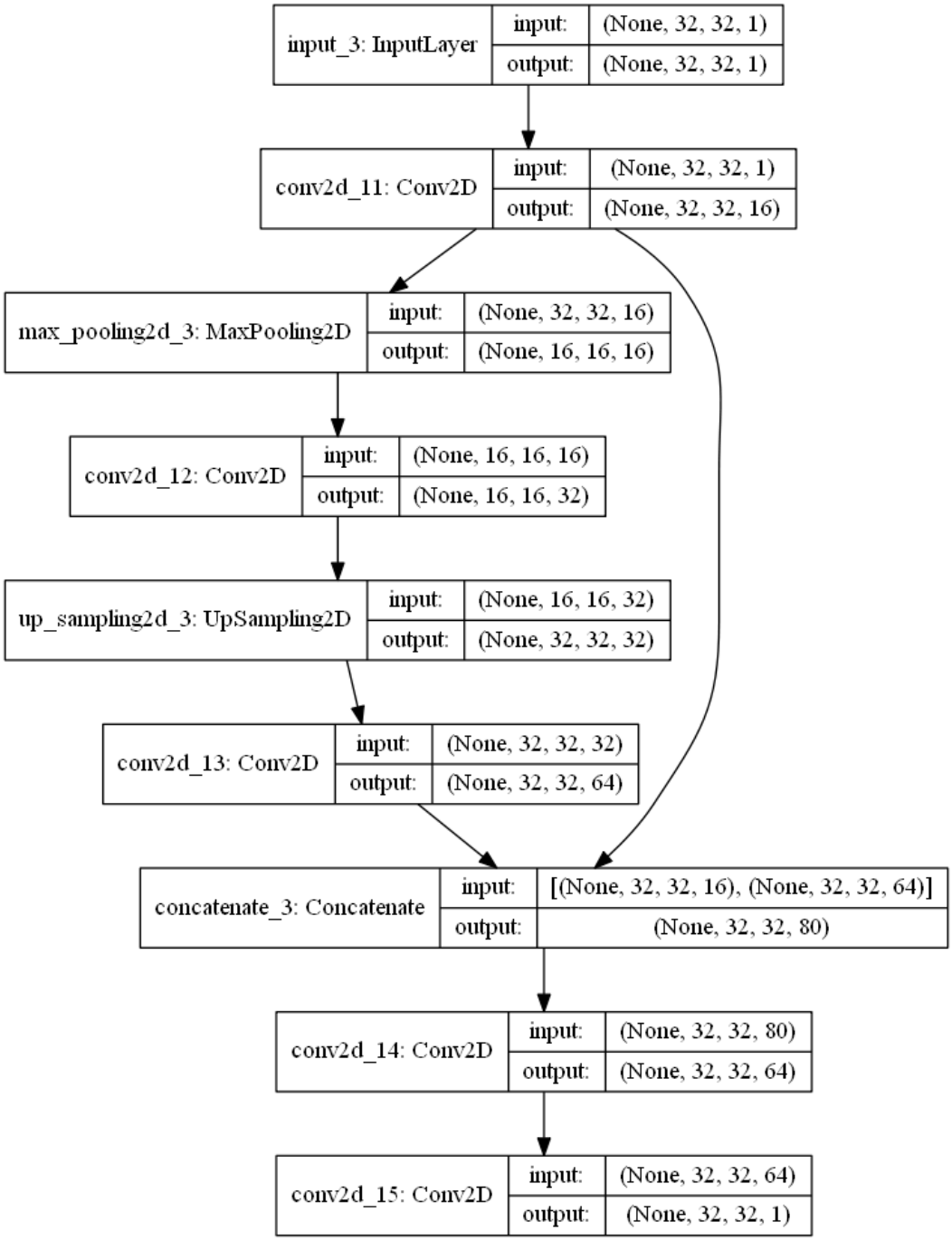


(a) Noisy version.

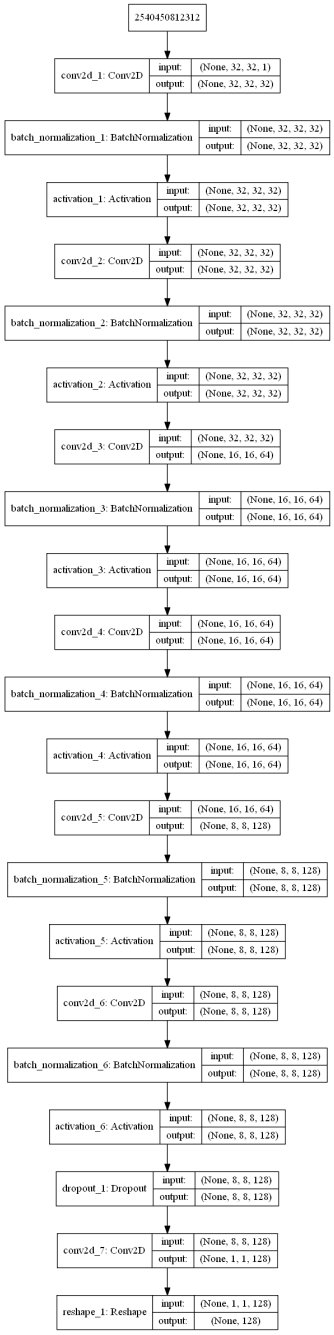
(b) Clean version.

Figure 12: Noisy and clean versions of one image.

B. Architecture of Base Denoiser Model



C. Architecture of Base Descriptor Model



D. Architecture of Base Denoiser Model