Deep Learning Report : Learning Patch Descriptors

Patrick John Chia
pjc316
01259148
pjc316@ic.ac.uk

Abstract

This work reports the findings from experiments conducted to develop a robust vector descriptor for patches, which is an important task in computer vision. The work builds upon a proposed approach that utilises Convolution Neural Networks (CNN) in various architectures and is trained to serve as a baseline that new approaches will be measured against. These new approaches include architectural modification and hard negative mining to deal with the limitations in the baseline approach. The main performance metrics will be the mAP from performing 3 tasks – Verification, Matching and Retrieval, which is performed using the HPatches [1] Testbench.

1. Introduction

The objective is to develop a Deep Neural Network (DNN) which takes as input a patch and generates a descriptor for that patch. A patch is an image that is cropped from a larger image and hence represents a local part of that scene. A descriptor is lower dimension vector representation of the input patch and should be able to accurately represent a patch to perform the 3 tasks - Verification, Matching and Retrieval.

In *patch verification*, descriptors are used to determine if two patches are in correspondence or not. *Image matching*, tests the extend to which descriptors can correctly identify correspondence in tow images. *Image retrieval*, tests how well descriptors can find corresponding patches from a large collection of patches many of which are distractors. For all 3 tasks, Mean Average Precision (mAP) is used as the performance metric. Further details can be found in [1].

2. Problem Definition

Consider a set of patches $\mathbf{x} = \{\mathbf{x}_1^1, \mathbf{x}_1^2, ...\mathbf{x}_j^i\}$ where $\mathbf{x}_j^i \in R^{32\times32\times1}$, where i and j refers to the i^{th} patch obtained from patch group j. A patch group refers to patches that were obtained from the same point in space.

Define $\mathbf{y}_j^i = h(\mathbf{x}_j^i) \in R^{1\times128}$ to be the descriptor, for a patch $\mathbf{x}_j^i$, that is generated from some model h .

The goal is to find some model g that minimises the dissimilarity of patches from the same patch group and minimises the similarity between patches from different patch groups. The level of dissimilarity will be measured by the L_2 distance, $d_{\mathbf{ab}} = ||\mathbf{a} - \mathbf{b}||_2$, between two patch descriptors .

Let Δ_j be the total dissimilarity of patches from the same patch group j and let $\bar{\Delta}_{jl}$ be the total similarity (negation of dissimilarity) between patches from the groups j and l . Hence, for a given n patch groups, we want to find g such that,

$$g = \underset{h}{\operatorname{argmin}}(\sum_{j=1}^{n} \Delta_j + \sum_{a=1}^{n-1} \sum_{b=a+1}^{n} \bar{\Delta}_{ab}) \tag{1}$$

Equation 1 is computationally intractable to optimise and hence it is more feasible to consider the Triplet Loss,

$$l_{trip}(\mathbf{a}, \mathbf{p}, \mathbf{n}) = \max(0, d_{\mathbf{ap}} - d_{\mathbf{an}}) \tag{2}$$

where the triplet $(\mathbf{a}, \mathbf{p}, \mathbf{n})$, refers to the descriptors for an anchor patch, a positive patch that is from the same patch group as $\mathbf{a}$ and a negative patch that is from a different patch group as $\mathbf{a}$. Hence the learning problem using N such triplets becomes,

$$g = \underset{h}{\operatorname{argmin}} \frac{1}{N} \sum_{i=0}^{N-1} l_{trip}(h(\mathbf{x}_a^i), h(\mathbf{x}_p^i), h(\mathbf{x}_n^i)) \tag{3}$$

The patches, $\mathbf{x}$, that will be used are from the N-HPatches dataset. The dataset contains 116 sequences each of which include a reference image and 5 target images with either photometric or geometric changes. The same patches are extracted for images in the same sequence, with geometric and signal perturbations of levels: *EASY, HARD* and *TOUGH* added. Different from the original HPatches dataset is the addition of signal perturbations.

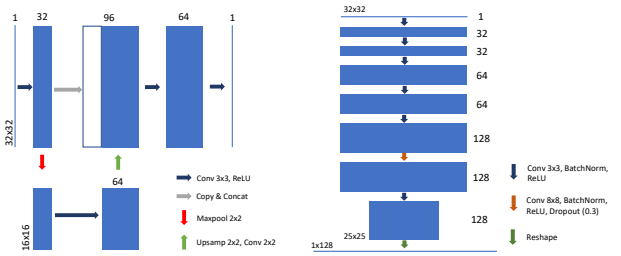

Figure 1: Baseline Denoising (Left) and Descriptor (Right) Architecture

3. Baseline Approach

3.1. Baseline Architecture

The baseline approach consists of two concatenated networks. The Denoising network is based on a shallow U-Net [5] and is trained with noisy and clean image pairs. The U-Net consists of a contracting path which features Convolutions and Maxpooling and an expansive path which features Upsampling, Convolutions and Concatenation with the corresponding feature map from the contracting path.

The Descriptor network is based on L2-Net [6] and is trained with with anchor, positive, negative triplets. It features successive Convolution, Batch Norm and ReLU layers with the number of CNN filters increasing along the network up till 128 filters. It then reshapes the final output to a 1x128 vector which will be the descriptor for the input. Figure 1 describes the two architectures in more detail.

3.2. Training and Evaluation

A. *Experimental setting*

1) *Training and Testing Data*: Training is performed using the N-HPatches dataset. For the Denoising dataset, a sample is drawn from the $\sim 1,500,000$ patches available to form a training set and validation set of $\sim 40,000$ and $\sim 30,000$ patches respectively. For the Descriptor dataset, $100,000$ and $10,000$ triplets are used for training and validation respectively.

2) *Parameter Setting and Network Training*: He Normal Initialisation is used to initialise the weights for all layers. For the Denoising network, SGD with learning rate of 0.00001, momentum of 0.9, mini-batch size of 50 and Mean Absolute Error (MAE) as the loss function is used. PSNR is used additionally as a performance metric. For the Descriptor network, SGD with learning rate of 0.1, mini-batch size of 50 and Mean Triplet Loss as the loss function is used. Training is done for 50 epoch for Baseline Model. The Descriptor network will also be trained for 50 epochs on clean and denoised patches.

B. *Evaluation*

Results for the baseline model can be found in Tables 1 and 2. In Figure 2, the loss for the Denoising network shows asymptotic behaviour suggesting that the current model complexity may need to be increased for better performance or that more epochs of training is required.

When comparing the performance to the Descriptor on clean and denoised patches, there is a 2.9% drop in Verification mAP but $\sim 10\%$ drop for Matching and Retrieval mAP.

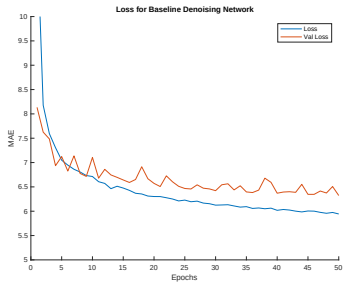

Figure 2: Baseline Denoising Loss

4. Improved Approach

Modifications were made to improve the performance of each of the two networks individually. These modifications are detailed in Sections 4.1 and 4.2.

4.1. Improved Denoising Network

The following improvements were made to the Denoising Network.

1. *Increased Architecture Complexity*

   The baseline network for Denoising uses a shallow U-Net. At this level of complexity, it is already able to produce a PSNR of 30.503 after 50 Epochs of training. A deeper U-Net with longer contractive and expansive paths is tested for the improved approach. Theoretically, this will increase the solution space and hence the number of hypothesis $h \in H$ that the network is able to represent. In the context of the U-Net, a longer contractive path yields increased feature information extraction which is postulated to improve Denoising Performance.

2. *Adam Optimiser*

   Adam [3] is deployed as an improved optimiser. Adam is an optimisation algorithm with adaptive learning rate unlike SGD which requires fine tuning of learning rate. It has also been shown in [3] that Adam is able to achieve faster training time.

4.2. Improved Descriptor Network

Improvements to the Descriptor Network were focused on a different approach to sampling. Specifically, *In-batch Hard Mining* is used in place the one used in the baseline. Similar approaches has been proposed in [2], [4] and [7]

1. *Descriptor Normalisation*

   A final layer that performs normalisation of the descriptors to unit length is added. This is critical in the calculation of the distances between descriptors which simplifies to $d_{\mathbf{ab}} = \sqrt{2(1 - \mathbf{ab}^T)}$ for any two descriptors $\mathbf{a}$ and $\mathbf{b}$.

2. *In-batch Hard Mining*

   The baseline network randomly samples triplets and applies the triplet loss to each sample. For each positive patch, only one corresponding negative patch is used to calculate the loss. However, in reality the number of negative patches for a given positive patch is orders of magnitude greater than the number of positive patches. From learning theory, we know that it is important to have a sample that is representative of reality. In addition, the loss used in the baseline averages the batch triplet loss. This suppresses the loss from the hard examples as the network improves, which slows the rate of further improvement.

   As such, a different sampling approach, know as *In-batch Hard Mining*, is used in the improved Descriptor network.

   In-batch Hard Mining utilises anchor and positive pairs, and generates many more $(2N - 2)$ negative distances per positive distance. This will be in a matrix where the diagonal elements are the positive distances. The off-diagonal elements along the column and row of the positive distance are its negative examples. The hardest example for a given positive distance is the one with the minimum value. The hardest example along the row and column for each positive distance is chosen and triplet loss applied to their average. The average triplet loss of this forms the loss for each batch. Full derivation of this can be found in Appendix A.

3. *Training With Clean Images then Noisy Images*

   Another potential improvement is to first train the descriptor network with clean patches that do not have signal noise perturbations. This may allow the network to converge to a good representation of the features of the patches first and not of the noise. Then, the trained network is fine-tuned with de-noised patches.

5. Evaluation of Improved Approach

This section details the results of the improved networks.

5.1. Evaluation of Improved Denoising Network

A. *Experimental setting*

1) *Training and Testing Data*: Follows Section 3.2.A.1.

2) *Parameter Setting and Network Training*: Follows Section 3.2.A.1 but with training done for 25 epochs and Adam is used as part of experiment.

3) *Experimental Approach*: The two proposed improvements are tested individually to determine their effect on performance, followed by testing their combined effect. Two U-Net architectures of increasing depth are tested. One with 1.3M (Deep) parameters another with 13M (Deeper). Architecture details for Deep U-Net can be found in Appendix B.

B. *Evaluation*

Results from the experiments can be found in Table 1. It is observed that with Baseline+Adam, there is appreciable improvement in the validation PSNR performance to 32.16. This gives some credence to the claim that Adam is able to attain faster convergence since only 25 epochs was used.

The results from an increased network complexity in Deep U-Net + SGD shows modest gains in the validation PSNR to 30.73. However, this is expected since more training may be required for a more complex network. Deep U-Net + Adam and Deeper U-Net + Adam also experience an increased PSNR. There is a general trend of increasing validation PSNR with increasing model complexity. However, the trade-off is a longer training time and also the need for more epochs.

5.2. Evaluation of Improved Descriptor Network

A. *Experimental setting*

1) *Training and Testing Data*: Unlike in the provided Baseline, precautions were made so that the training and validation data used do not come from the test split to prevent contamination of data used in the HPatches testbench. A custom data generator was written to support the *In-Batch Hard Mining* proposed in Section 4.2.2. The training and validation sets use 100 and 10 mini-batches of size of 128 (anchor-positive pairs) respectively.

2) *Parameter Setting and Network Training*: Training is performed in a Siamese Network rather than Triplet Network to accommodate the proposed sampling approach for 50 epochs. The loss function is as described in Section 4.2.2 with the triplet loss parameter $\alpha = 0.01$. Adam Optimiser is used. He Normal Initialisation is used for all Convolutional layers.

Denoising Network Performance	
Model	Val PSNR
Baseline	30.5
Baseline + Adam	32.16
Deep U-Net + SGD	30.73
Deep U-Net + Adam	32.32
Deeper U-Net + Adam	32.47

Table 1: Summary of Denoising Performance for Baseline and Improved Networks

3) *Experimental Approach*: Training is performed with clean, denoised and noisy patches. Additionally the proposed fine-tuning approach is also tested.

B. *Evaluation*

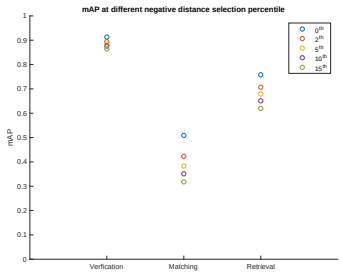

Figure 3: Testbench performance for different selection of negative distance training example based on percentile

The testbench results for the experiments performed on the improved Descriptor network are shown in Table 2. The initial results from training with clean patches is promising. An improvement in the mAP for all three tasks is observed, demonstrating the importance of *In-Batch Hard Mining*.

However, it is difficult to directly compare this to the Baseline network. The Baseline network was trained with $100,000$ triplets. On the other hand, the improved network is trained with $12,800$ pairs but is able to generate distances for $3,251,200$ triplets when using 100 mini-batches of size 128 pairs. It is intractable to train the Baseline network with 3M triplets due to computational inefficiency, and it is also unfair to take the batch average triplet loss when using In-Batch Hard Mining as it would have a mini-batch size of 32512 which is orders larger than the 50 used in the Baseline.

Therefore, to further assess the effect of Hard Mining, the improved model is trained at $q = 0, 2, 5, 10, 15$, where q refers to the percentile of the negative examples that are chosen. Hard Mining effectively uses $q = 0$. The results are shown in Figure 3. The benefits of In-Batch Hard Mining can be seen, where even at $q = 2$, there is significant drop in *Matching* and *Retrieval* performance.

HPatches Testbench Results (mAP)					
	Baseline		Improved		
	Clean	Denoised	Clean	Denoised	Noisy
Verifi	0.857	0.829	0.913	0.860	0.851
Match	0.341	0.251	0.509	0.344	0.342
Retriv	0.662	0.561	0.758	0.620	0.607

Table 2: Summary of Descriptor Performance for Baseline and Improved Networks trained and evaluated on Clean, Denoised or Noisy Patches

Additionally, by only using the hardest examples in the batch, it promotes continuous and faster learning even as the network improves. This is unlike the Baseline loss, which took the average of the triplet loss, many of which will be small or near zero, of the entire batch - this results in a small error and a poor gradient as the network improves. Overfitting is not observed since mining is done In-Batch and not on the entire dataset. [4] has also shown that In Batch Hard Mining does not lead to overfitting unlike classical hard-mining.

The results for the proposed fine-tuning are $[0.863, 0.346, 0.624]$ for Verification, Matching and Retrieval respectively. This is a slight improvement when compared to training with the denoised patches only. It is difficult to credit the improvement to the proposed fine-tuning method as it may be due to the variance of the network.

It is interesting to note that the performance when trained with noisy patches is very close to when trained with denoised patches. This suggests that the Descriptor Network is able to find a representation which accounts for the noise and suggests that a denoising network may not be needed. A patch group can be thought of as a point in space, and with nosie added or when denoised, it increases the variance of the patches to ground truth which makes learning a low variance representation harder. Therefore, whether noisy or denoised, the variance is still higher which may explain why their performance is very similar. The denoiser also does not account for the features which are effective for the descriptor and hence is less effective than expected.

6. Conclusion

The results show the importance of proper sampling and understanding of the loss function. The loss function determines how the network learns, and consideration of its response during training is a critical. Model complexity is also an influencing factor, but increasing complexity may not always be the solution and often comes with trade-offs. In deep learning, data and its efficient use, is still a big determining factor in the success of a network as seen in the use of a smaller training set in the improved model to effectively generate many negative examples and significantly improve the network's performance.

References

[1] V. Balntas, K. Lenc, A. Vedaldi, and K. Mikolajczyk. Hpatches: A benchmark and evaluation of handcrafted and learned local descriptors. *CoRR*, abs/1704.05939, 2017.

[2] M. Keller, Z. Chen, F. Maffra, P. Schmuck, and M. Chli. Learning deep descriptors with scale-aware triplet networks. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 2762–2770, 2018.

[3] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[4] A. Mishchuk, D. Mishkin, F. Radenovic, and J. Matas. Working hard to know your neighbor's margins: Local descriptor learning loss. *CoRR*, abs/1705.10872, 2017.

[5] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015.

[6] Y. Tian, B. Fan, and F. Wu. L2-net: Deep learning of discriminative patch descriptor in euclidean space. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 00, pages 6128–6136, July 2017.

[7] Y. Tian, B. Fan, and F. Wu. L2-net: Deep learning of discriminative patch descriptor in euclidean space. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

A. Loss Derivation for In-Batch Hard Mining

In *In-batch Hard Mining*, a training batch is formed with n anchor and positive pairs $(\mathbf{a}_i, \mathbf{p}_i)$ for $i = 1..n$ from n distinct patch groups. The descriptors $(\mathbf{y}_a^i, \mathbf{y}_p^i)$ are obtained for each pair, resulting in $2n$ descriptors, where $\mathbf{y}_a^i, \mathbf{y}_p^i \in R^{1\times 128}$. Let $\mathbf{Y}_a$ and $\mathbf{Y}_p$ be matrices formed from to the anchor and positive descriptors such that $\mathbf{Y}_a = [\mathbf{y}_a^{1^T} \dots \mathbf{y}_a^{n^T}]$ and $\mathbf{Y}_p = [\mathbf{y}_p^{1^T} \dots \mathbf{y}_p^{n^T}]$. Take the product of $\mathbf{Y}_a$ and $\mathbf{Y}_p^T$ to form,

$$\mathbf{Y} = \mathbf{Y}_a^T \mathbf{Y}_p \tag{4}$$

$\mathbf{Y}$ is a matrix of inner products of all combinations of $\mathbf{y}_a^i$ and $\mathbf{y}_p^j$. Given that the descriptors $\mathbf{y}$ have been normalised to unit length, a matrix of distances, $\mathbf{D}$, between the descriptor $\mathbf{y}_a^i$ and $\mathbf{y}_p^j$ in matrix $\mathbf{Y}$ can be formed by,

$$\mathbf{D} = \sqrt{2(1 - \mathbf{Y})} \tag{5}$$

which follows from the law of cosines. Hence, $\mathbf{D}$ is a $n \times n$ matrix with n anchor-positive distances, $\mathbf{d}_{ap}$, along its diagonal entries and a total of $2n - 2$ anchor-negative distances and positive-negative distances $\mathbf{d}_{aj}$ and $\mathbf{d}_{ip}$, along the row and column of each anchor-positive distance respectively[1].

For each $\mathbf{d}_{ap}$ (diagonal element), the hardest corresponding training example along its row and column is given by the $\mathbf{d}_{aj}$ or $\mathbf{d}_{ip}$ (off-diagonal element) with minimum value. The Triplet loss (Eqn 2) is applied then applied as

$$l_{trip} = \max(0, \mathbf{d}_{ap} - \frac{\min\limits_{j,j\neq a} \mathbf{d}_{aj} + \min\limits_{i,i\neq p} \mathbf{d}_{ip}}{2} + \alpha) \tag{6}$$

Which uses the average of the hardest examples in the row and column for a given $\mathbf{a}$ and $\mathbf{p}$ pair. The average of the total triplet loss for each anchor-positive pair is then used as the loss for each batch.

[1]Note that this approach generates many negative examples without significant computational overhead

B. Architecture of Deep U-Net

All Convolutional layers use He Normal Initialisation with padding set to SAME.

Layer Name	Layer Details
input	Input
enc_conv1	Conv2D 64,3x3 ReLU Maxpool 2x2
enc_conv2	Conv2D 128,3x3 ReLU Maxpool 2x2
enc_conv3	Conv2D 256,3x3 ReLU Maxpool 2x2
enc_conv4	Conv2D 512,3x3 ReLU
upsamp3	UpSampling2D 2x2 Conv2D 256, 2x2
dec_conv3	Concat(upsamp3,enc_conv3) Conv2D 256,3x3 ReLU
upsamp2	UpSampling2D 2x2 Conv2D 128,2x2
dec_conv2	Concat(upsamp2,enc_conv2) Conv2D 128,3x3 ReLU
upsamp1	UpSampling2D 2x2 Conv2D 64,2x2
dec_conv1	Concat(upsamp1,enc_conv1) Conv2D 64,3x3 ReLU
out	Conv2D 1,3x3

Table 3: Architecture Details for Deep U-Net

C. Experimentation Code

Link: www.dropbox.com/s/0znpsu4egvot9kg/pjc316.zip?dl=0
The IPython notebook is run as per normal. The different configurations are commented out and should be uncommented when required. User can load pre trained model if required.