EE3-25 Deep Learning Report

Kaiyue Sun
ICL login: ks4016
CID: 01197813
kaiyue.sun16@imperial.ac.uk

Abstract

In this report, the objective is to propose an image descriptor for measuring similarity between images. The descriptor is trained and tested on N-HPatches, a noisy version of HPatches dataset [1]. The method of 2 Convolutional Neural Networks (CNN) pipeline is used in the baseline appraoch. The first network aims to denoise the input patches. The second one is used to train the descriptor, which is then evaluated on three tasks: patch verification, image matching and patch retrieval. Firstly, the baseline approach is demonstrated and implemented. Based on that, an improved approach is proposed and evaluated. Then an insightful aanalysis of the improvement upon the baseline approach is discussed in the final part.

1. Formulation of the ML Problem

The original HPatches is based on 116 sequences of images, 57 sequences presenting photometric changes and the remaining 59 sequences show geometric deformation due to viewpoint change. The N-HPatches dataset contains many sets of corresponding patches extracted with different settings of noise described as EASY, HARD and TOUGH. The patches are downsampled from the size of 65×65 to 32×32. Therefore, the input is a patch $x \in \mathcal{X} \in \mathbb{R}^{32\times32}$, the output is a descriptor of size 128×1, $y \in \mathcal{Y} \in \mathbb{R}^{128\times1}$. The hypothesis set is $\mathcal{H} \in \{h : \mathcal{X} \to \mathcal{Y}\}$. The goal is to find a $g \in \mathcal{H}$ that approximates the target function $f : \mathcal{X} \to Y$ such that $f(x) = y$.

2. The baseline approach

The baseline pipeline has 2 consecutive networks. The first denoising network is a shallow UNet[2] which is trained using the Mean Absolute Error (MAE) as the loss function:

$$MAE = \frac{\sum_{i=1}^{n} |y_i - y_i^p|}{n} \tag{1}$$

where y_i^p is the pixel of the denoised patch output from the network for the input of a noisy patch, y_i is the pixel of the corresponding cleaned patch. n, in our case, is the total number of pixels $32 \times 32 = 1024$. The second network is the L2-Net[3] architecture which takes the denoised input patch to create a learned descriptor to represent the input. The network is trained using samples of triplet in the form $\mathbf{a}, \mathbf{p}, \mathbf{n}$, which correspond to Anchor, Positive (same local part of the scene as the anchor with geometric transformation) and Negative (not the same) inputs, The loss function is the triplet loss[4]:

$$\mathcal{L}(\mathbf{a}, \mathbf{p}, \mathbf{n}) = max(0, \mu + ||f(\mathbf{a}) - f(\mathbf{p})||_2 - ||f(\mathbf{a}) - f(\mathbf{n})||_2) \tag{2}$$

This loss makes sure that the distance between descriptors from anchor and positive inputs is smaller than that from anchor and negative inputs by at least a margin μ. The advantage of using the margin is that, while all inputs of the same local part will cluster eventually, they are not required to converge to a single point, they merely need to be closer to each other than to other inputs from a different class.
For the evaluation metrics, we use the mean Average Precision (mAP) on three tasks to evaluate the descriptor. They are Patch Verification, which tests the ability of the descriptor to classify whether 2 patches are in correspondence or not; Image Matching, which tests how well the descriptor can match patches from a reference image to a target one; and Patch Retrieval, that states how well the descriptor is in finding correspondences in a large collection of patches. The score is the average of the mAPs on all tasks.

2.1. Training the baseline approach

In the 116 sequences of images, 76 sequences are used for training, 40 sequences are used for validatioin. The baseline code takes randomly 3 training sequences and 1 validation sequence in order to speed up the training of the denoise model. In the first strategy, the partial data is used to train the denoise model for 50 epochs and train the descriptor for 10 epochs. Figure 1 shows the losses of the 2 models. In denoise model, the validation error levels off at the 20^{th} epoch while the training error is still decreasing.

Table 1. Optimizer

Optimizer	parameter
SGD (denoise)	lr=0.00001, momentum=0.9, nesterov=True)
SGD (descriptor)	lr=0.1
Adagrad	lr=0.01, epsilon=None, decay=0.0
Adadelta	lr=1.0, rho=0.95, epsilon=None, decay=0.0
Adam(1)	lr=0.001, beta$_1 = 0.9, beta_2 = 0.999, epsilon = None, decay = 0.0, amsgrad = True$
Adam(2)	lr=0.001, beta$_1 = 0.9, beta_2 = 0.999, epsilon = None, decay = 0.0, amsgrad = False$
Adam(3)	lr=0.001, beta$_1 = 0.9, beta_2 = 0.999, epsilon = 1e-9, decay = 0.0, amsgrad = True$
Adam(4)	lr=0.002, beta$_1 = 0.9, beta_2 = 0.999, epsilon = None, decay = 0.0, amsgrad = False$

This indicates that the network might be overfitting to the training data, which should be avoided. As for the descriptor, the validation error fluctuates vigorously. This might be due to the fact that the validation split is selected at random and its size is too small for testing, so the validation error is from some extreme cases and cannot reflect the average level of the testing error. The final score is: Retrieval: 0.410774. Matching: 0.125483. Verification: 0.672233. Mean: 0.40283. By repeating the same experiment several times, different scores are obtained. This implies that the size of the partial dataset used is too small to be reliable.

In the second strategy, the whole dataset is used to train the denoise network for only 8 epochs since it takes much longer. Figure 2 graphs the model losses. In the denoise model, the trainning error drops to 6 at the 3^{rd} epoch, it converges much faster than using partial data. Its validation error seems to saturate at the 4^{th} epoch. The training method for the descriptor is unchanged. This time its validation loss is much smoother than using partial data. The final score is: Retrieval: 0.35895. Matching: 0.104292. Verification: 0.71493. Mean: 0.392724, similar to the first strategy and very reasonable since the denoise network is not fully trained. This proves that using a larger dataset dose improve the training of the network. Moreover, the validation error of the descriptor decreases across all the 10 epochs, which means an epoch size of 10 is not enough to fully train the model. Then clean patches is used to train the descriptor for 30 epochs in order to see the best case scenario. As shown in Figure 3, the validation error is still dropping at the 30^{th} epoch, but in order to speed up the training when trying different setups, a smaller epoch size is prefered.

As such, an epoch size between 15 and 20 would be appropriate for parameter tuning.

In summary, a compromise between time and reliability should be made for better parameter tuning. I chose 35 training sequences and 18 validation sequences to train the denoise model. As discussed above, the point of overfitting should be between 4 and 20 epochs. As such, an epoch size of 12 seems appropriate is selected for parameter tuning. And an epoch size of 15 is used to train the descriptor for the following optimizer selection stage of the denoise network. Since training the descriptor with clean patches does not need the training of the denoise model, the epoch size of 30 could still be used.

2.2. changing optimizers

The optimizer selection for both models is separated. First of all, clean patches are used to train the descriptor for comparison with the upper bound of each optimizer. Lists of optimizers with corresponding parameters is shown in Table 1. Table 2 shows the score achieved by the optimizers. SGD seems to be the best and achieves 0.633644. Adam(1) and Adam(3) achieve 0.619771 and 0.618576 which are comparable to SGD. For the denoising model, denoised patches and SGD(descriptor) are used for training the descriptor.The performance is shown in Table 3. It can be seen that Adagrad, Adadelta and Adam(2) are the best ones.

3. Improved approach and the evaluation

3.1. Modifying UNet[2]

The denoisng network used in the baseline code is a shallow UNet. It follows the typical architecture[**?**] of the UNet, but with only one contracting layer and one expansion layer.In the bottleneck, the size is 16×16, the number of feature channel is 32.

I can modify the architecture to be a deeper UNet with multiple contracting and expansive layers. The maximum number of layers in each path is 5, since the HPatches is of the size 32×32, five layers will reduce the image size by $2^5 = 32$, so the image size will be 1×1 in the bottleneck.

The performance of using different UNet architectures for training the denoise network is shown in the table at the bottom in this section. The training epoch is 12. Although in Table 3 the best optimizer is Adagrad, Adam can achieve good results fast and is much easier to tune. Therefore, Adam(1), which achieves a really close score to Adagrad, is selected to train the modified UNet.

The modified networks consists of a contracting path and an expansive path. The contracting path follows the typical architecture of a convolutional network. It consists of five repeated application of a single 3x3 convolution

Table 2. Performance of using different optimizer for training descriptor network, epoch=30

Optimizer	SGD(descriptor)	Adagrad	Adadelta	Adam(2)	Adam(1)	Adam(3)
Patch Verification	0.871581	0.864996	0.870762	0.86003	0.87153	0.869749
Image Matching	0.358279	0.311804	0.335912	0.308001	0.338652	0.337417
Patch Retrieval	0.671072	0.617927	0.63956	0.620758	0.649131	0.648561
Mean	0.633644	0.598242	0.615411	0.596263	0.619771	0.618576

Table 3. Performance of using different optimizer for training denoise network, epoch=12

Optimizer	SGD(denoise)	Adagrad	Adadelta	Adam(2)	Adam(1)	Adam(3)
Patch Verification	0.774065	0.805768	0.806664	0.79176	0.81138	0.785766
Image Matching	0.186951	0.211124	0.211495	0.189246	0.210634	0.194227
Patch Retrieval	0.491477	0.509356	0.502069	0.481764	0.504025	0.48643
Mean	0.484164	0.508749	0.506743	0.48759	0.508680	0.488808

(unpadded convolutions), each followed by a rectified linear unit (ReLU) and a 2x2 max pooling operation with stride 2 for downsampling. At each downsampling step the number of feature channels is doubled starting from a specific number which is 8 in UNet(1), 16 in UNet(2), 32 in UNet(3), 64 in UNet(4). Every step in the expansive path consists of an upsampling of the feature map followed by a 2x2 convolution ("up-convolution") that halves the number of feature channels, a concatenation with the correspondingly cropped feature map from the contracting path, and a single 3x3 convolution, each followed by a ReLU. The cropping is necessary due to the loss of border pixels in every convolution. At the final layer a 1x1 convolution is used to map each 64-component feature vector to the desired number of classes. In total the network has 11 convolutional layers.

	UNet(1)	UNet(2)	UNet(3)	UNet(4)
Verification	0.813676	0.813113	0.814368	0.813324
Matching	0.208585	0.209236	0.212187	0.214894
Retrieval	0.500286	0.507034	0.505547	0.514279
Mean	0.507516	0.509794	0.510367	0.514166

It can be seen that generally the performance improves as the number of feature channels in each layer increases. When the size of the image reduces following the down sampling in the contracting path, the filter in the deeper laters could focus on a larger receptive field. However, at the same time, more number of channels are required to extract more complex features from the image. Therefore, more channels means less features to be ignored, which is the reason of this observation.

In addition, the performances of the modified architectures except UNet(1) are better than the baseline approach with the same optimizer. intuitively we would expect the architecture with many more layers to be more powerful. For example, the neurons in the first layer might learn to recognise brightness, the neurons in the second layer could learn to recognise edges, built up from brightness. The third layer would then recognise still complex shapes like triangle and rectangle, and so on. These multiple abstraction layers seem likely to give deep networks a compelling advantage in learning to solve complex image identification problems. However, I would suggest that there is also an upper limit of the complexity of the network, just like a higher degree of freedom may introduce more deterministic noise. At some point, it will lead to overfitting.

3.2. Triplet Loss

As discussed in [5], from the definition of the triplet loss used in the training of the descriptor model, it can be pretended that: the positive distance is 1.5, the negative distance is 3.5. Then $1.5 - 3.5 + 1 = -1$. By taking the maximum between -1 and 0, the loss function results will end up with 0. This means a positive distance of 1.5 will not contribute to the loss and the network will not try to minimise this distance. Therefore, it is going to be very difficult for the algorithm to effectively reduce the distance patches from the same local part.
An improved approach could be taking a greater weight for the positive distance in the loss function. For example:

$$\mathcal{L}(\mathbf{a}, \mathbf{p}, \mathbf{n}) = max(0, \mu + \alpha||f(\mathbf{a}) - f(\mathbf{p})||_2 - ||f(\mathbf{a}) - f(\mathbf{n})||_2) \tag{3}$$

in which α should be a constant larger than 1. If it is set to be 2, in the example above, $2 \times 1.5 - 3.5 + 1 = 0.5$, the positive distance of 1.5 will be taken into account by the loss function. Therefore, the algorithm could reduce more positive distances.

According to the deduction above, an experiment is carried out with setting $\alpha = 2$. Again, the descriptor model is trained for 30 epochs using clean patches. The score is: Retrieval: 0.499507. Matching: 0.128313. Verification: 0.828551. Mean: 0.485457. The result is worse than the using the original loss function as shown in Table 2. One reason could be that, as the algorithm needs to put more ef-

fort to reduce the positive distances, it will take longer time and more epochs to achieve a better results.

4. Future work

Due to the limitation of time, more improvement could only be done in the future, for example data augmentation can apply more elastic deformations to the available training images. This allows the network to learn invariance to such deformations, without the need to see these transformations in the annotated image corpus as discussed in [2] Another thing is that as the dataset gets larger, the possible number of triplets growscubically, rendering a long enough training impractical. The algorithm relatively quickly learns to correctlymap most trivial triplets, rendering a large fraction of all triplets uninformative. Thus mining hard triplets becomes crucial for learning. This can be implemented by using the algorithm Batch All to change the way in which triplets are formed to improvement the effectiveness of triplet loss as discussed in[6].

References

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

[2] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation, May 18, 2015.

[3] Y. Tian, B. Fan, and F. Wu. L2-net: Deep learning of discriminative patch descriptor in euclidean space. pages 6128–6136. IEEE, Jul 2017.

[4] V.Balntas,E.Riba,D.Ponsa,andK.Mikolajczyk.Learning local feature descriptors with triplets and shallow convolu- tional neural networks. Proc. BMVC, 2016.

[5] Marc-Olivier Arsenault () Lossless Triplet loss, Available at: https://towardsdatascience.com/lossless-triplet-loss-7e932f990b24 (Accessed: 22nd March 2019).

[6] In Defense of the Triplet Loss for Person Re-IdentificationArXiv2017 Alexander HermansLucas BeyerBastian Leibe

Appendices

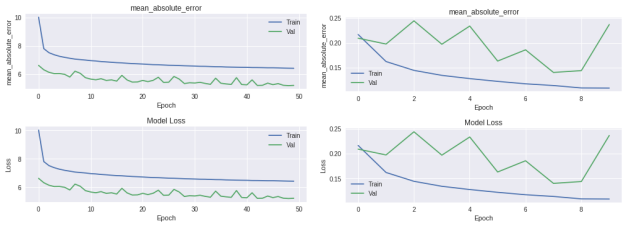

Figure 1. denoise and descriptor model loss, epoch: 50 &10

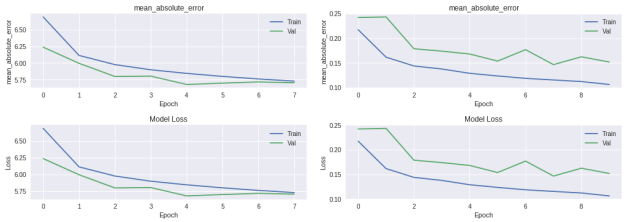

Figure 2. denoise and descriptor model loss, epoch: 8 &10

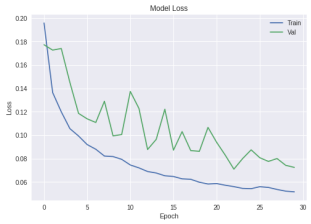

Figure 3. descriptor model loss, epoch: 30