Deep Learning - Final Report

Yee Hong Low
yhl116
01202613

Abstract

This report outlines the proposed improved approach to create a vector of learned descriptor that is able to perform matching, verification and retrieval tasks successfully on noisy image patches (N-HPatches). This is done by matching and mapping similar images into distinct clusters on Euclidian space. Two convolutional neural nets are used for this purpose: one for denoising the noisy N-HPatches into clean H-Patches, and one for descriptor generation. The benchmark used to evaluate the descriptors is separated into matching, verification and retrieval, each measuring performance of the generated descriptors in different aspects.

This report is split into three parts: Section one explains the formulation of the learning problem. Part two outlines the baseline approached used and its performance. Part three summarises the proposed approach to improve upon the baseline approach.

1. Introduction

The project proposes a learned descriptor that is able to perform matching, verification and retrieval tasks successfully on N-HPatches. N-HPatches is a noisy version of HPatches dataset.

The problem at hand is a supervised multiclass classification problem. The input data is given as 32 by 32 pixel noisy samples in 1 channel (grey-scale). The output is a vector of 128 descriptors which represents the features of the input images.

vs Bad.jpg

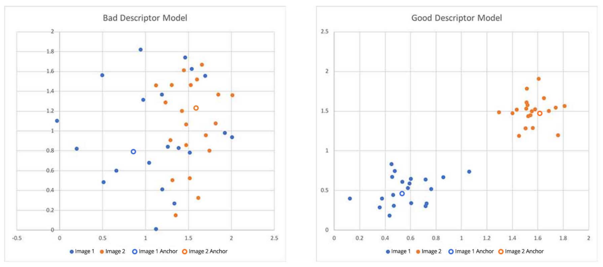

Figure 1. Good Descriptor Model vs Bad Descriptor Model

Figure 1 shows a simple example of a descriptor of two types of images on a 2D space. A good descriptor model maps similar images into compact, easily distinguishable clusters, centered around the anchor. Note that in a good descriptor model, the Euclidian distance between similar images are minimal while distance between dissimilar images are large. The same concept is used for the proposed descriptor model but the descriptor model has 128 dimensions. The Euclidian distance, l2, between two points, p and q, is calculated using the formula below:

$$l2 = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2}$$

where n, the dimension of the Euclidian space, is 128 in this report.

1.1. Evaluation

The learned descriptor is evaluated with three metrics: Verification, matching and retrieval (VMR):

1. Retrieval: The descriptor is able to find similar images from a large pool of images consisting of large numbers of distractors.

2. Matching: The descriptor is able to find similar images from a small pool of images with difficult distractors.

3. Verification: The descriptor is able to deduce if two images given are similar.

All the three metrics mentioned uses mean average precision (maP). The benchmark used is based off the paper *HPatches: A benchmark and evaluation of handcrafted and learned local descriptors* (Balntas et al. 2017) which has been designed to imitate real world uses of local descriptors.

$$mAP = \frac{TP}{TP + FP}$$

where TP = True Positive and FP = False Positive.

2. Baseline Approach

The baseline approach is made up of two models: the denoising model and the descriptor model.

Stochastic gradient descent is used as the optimizer for both the baseline denoise model and the baseline descriptor model. The hyperparameters used in for the optimizer is summarised in Table 1.

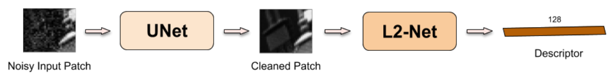

Figure 2. Baseline Descriptor Model Pipeline

Model	Denoise Model	Descriptor Model
Learning Rate	0.00001	0.1
Momentum	0.9	0
Nesterov	True	False
Decay	0	0

Table 1. Baseline Model Optimizers Hyperparameters

2.1. Baseline Denoise Model

The baseline denoisie model features a four-layer shallow UNet (Ronneberger et al. 2015), a convolutional autoencoder. The autoencoder uses an encoder ϕ to lower the dimension of noisy input images and then uses a decoder ψ to get the denoised images.

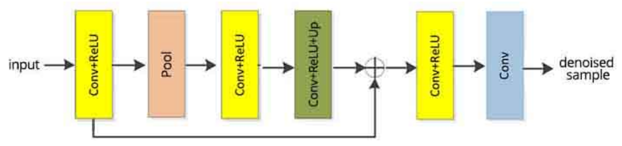

Figure 3. Baseline Denoise Model Architecture

The baseline denoise model uses mean absolute error (MAE) as the loss function:

$$MAE = \frac{1}{n}\sum_{i=1}^{n} \| x_i - \psi(\phi(x_i)) \|$$

where n is the number of inputs where in this case is given by the number of pixels of each images i.e. 32 x 32 = 1024.

2.2. Baseline Descriptor Model

The baseline descriptor model is built based off HardNet (Mishchuk et al. n.d.). It takes in the denoised sample and produces a descriptor: a vector of 128 elements which acts as representations for the input images.

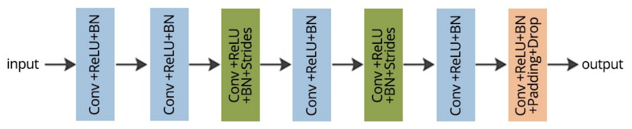

Figure 4. Baseline Descriptor Model Architecture

The descriptor model uses Triplet loss as the loss function which takes in three inputs: anchor (a), positive (p) and negative (n).

$$TripletLoss(a,p,b) = max(\| a-p \|^2 - \| a-n \|^2 + \alpha, 0)$$

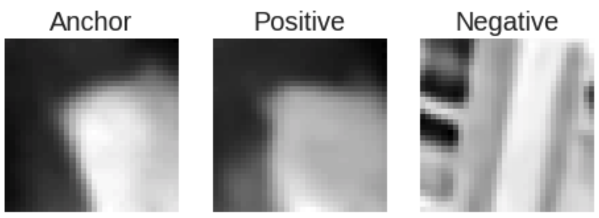

Figure 5. Anchor (a), Positive (p) and Negative (n)

2.3. Baseline Performance

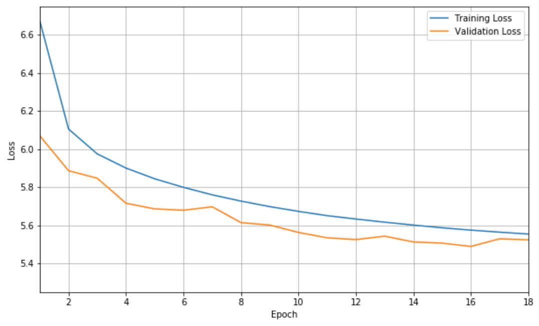

Figure 6. Baseline Denoise Loss vs Epoch

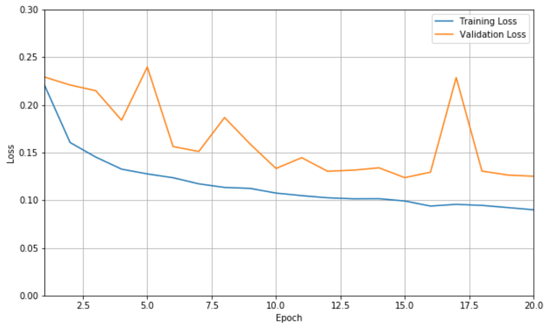

Figure 7. Baseline Descriptor Loss vs Epoch

From Figure 6, it is deduced that the baseline denoise model overfits the data starting from the 17th Epoch as the value loss starts to increase. Whilst from Figure 7, it is deduced that the baseline descriptor starts overfitting from the 16th epoch. Hence early stopping is introduced and for the final baseline model, the denoise model is trained for 16 epochs while the descriptor model is trained for 15 epochs. The final VMR score are 0.801, 0.204 and 0.484.

3. Proposed Methods

All the models are trained with 76 reference image sets while 40 reference image sets are used for validation. The hyperparameters used for all the models are listed as follows:

Initializer	he_normal
Denoise Model Loss Function	Mean Absolute Error
Descriptor Model Loss Function	Triplet Loss

Table 2. Testing Parameters

The loss function used for both models are kept the same as the ones used in the baseline approach so that the performance of all the models tested can be objectively compared.

3.1. Proposed Denoising Model

A few architectures were studied to improve upon the baseline denoise model. For all the proposed denoise models, Adam (Kingma & Ba 2017) is used as the activation function with default hyperparameters i.e. lr = 0.001, beta_1 = 0.9, beta_2 = 0.999, epsilon = None, decay = 0.0, amsgrad = False. The number of convolutional layers for each model studied, which reflects the complexity of the model, are summarised in Table 3.

Denoising Model	**Number of Convolutional Layers**
Baseline	5
Shallow UNet	7
Moderate Dilated UNet	12
UNet with Drop Layers	15
Moderate Dilated UNet	15
Dilated UNet	15
UNet	15

Table 3. Number of Convolutional Layers

Convolutional layers are added in incremental stages to fully study its effects. Deeper models are not studied in this report due to overly lengthy training times. Training one epoch for UNet takes about 80 minutes while training one epoch for Dilated UNet takes about 100 minutes.

A dilation rate of 2 is introduced in the dilated models for every convolutional layers. Dropout layers were introduced to some of the models at a rate proportional to the number of neurons of each layer. Note that early stopping was introduced to Dilated UNet and UNet as the models have started to overfit the data before the 10th epoch.

UNet was found to be the most effective out of all the models studied, giving minimal loss without overfitting as

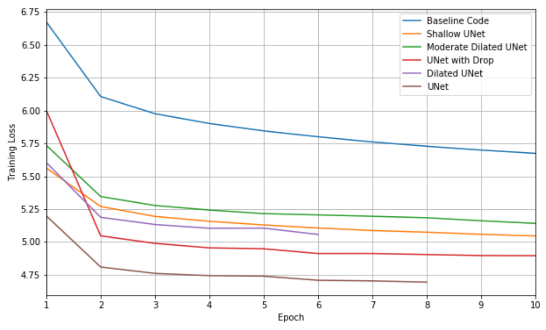

Figure 8. Proposed Denoising Model Loss vs Epoch

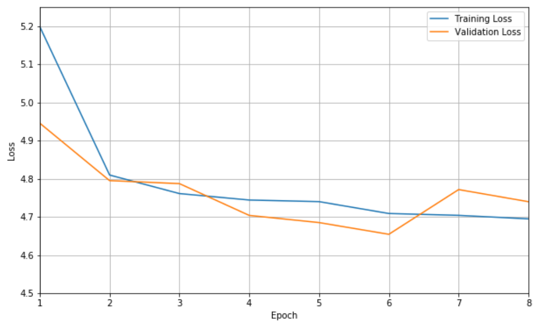

Figure 9. UNet Loss vs Epoch

reflected in Figure 8 and Figure 9. In Figure 9 shows that UNet has minimal validation loss at the 6th epoch and starts to overfit on the 7th epoch. The minimum validation loss is 4.6542 with a training loss of 4.7087. The architechture of UNet is summarised in Table 4.

3.2. Proposed Descriptor Model

A few models are tested on top of the baseline descriptor model and the results is illustrated in Figure.

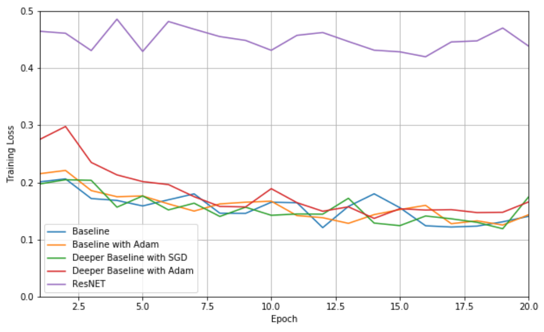

Figure 10. Descriptor Model Training Loss vs Epoch

Layer Name	Layer Type	Connected To
input	Input Layer	-
conv2D_1	Convolutional	input
max_pooling2d_1	Max Pooling	conv2D_1
conv2d_2	Convolutional	max_pooling2d_1
max_pooling2d_2	Max Pooling	conv2d_2
conv2d_3	Convolutional	max_pooling2d_2
max_pooling2d_3	Max Pooling	conv2d_3
conv2d_4	Convolutional	max_pooling2d_3
max_pooling2d_4	Max Pooling	conv2d_4
conv2d_5	Convolutional	max_pooling2d_4
up_sampling_1	Upsampling	conv2d_5
conv2d_6	Convolutional	up_sampling_1
concatenate_1	Concatenate	conv2d_4 conv2d_6
conv2d_7	Convolutional	concatenate_1
up_sampling_2	Upsampling	conv2d_7
conv2d_8	Convolutional	up_sampling_2
concatenate_2	Concatenate	conv2d_3 conv2d_8
conv2d_9	Convolutional	concatenate_2
up_sampling_3	Upsampling	conv2d_9
conv2d_10	Convolutional	up_sampling_3
concatenate_3	Concatenate	conv2d_3 conv2d_10
conv2d_11	Convolutional	up_sampling_3
up_sampling_4	Upsampling	conv2d_11
conv2d_12	Convolutional	up_sampling_4
concatenate_4	Concatenate	conv2d_1 conv2d_12
conv2d_13	Convolutional	concatenate_4
conv2d_14	Convolutional	conv2d_13
conv2d_15	Convolutional	conv2d_14

Table 4. UNet Architechure

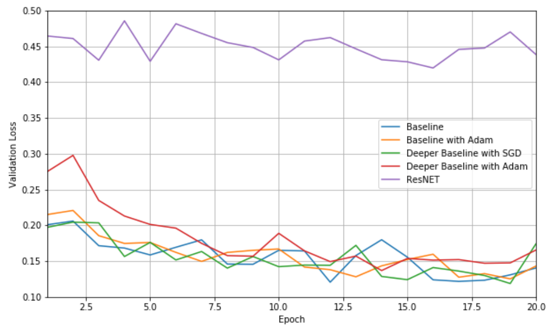

Figure 11. Descriptor Model Validation Loss vs Epoch

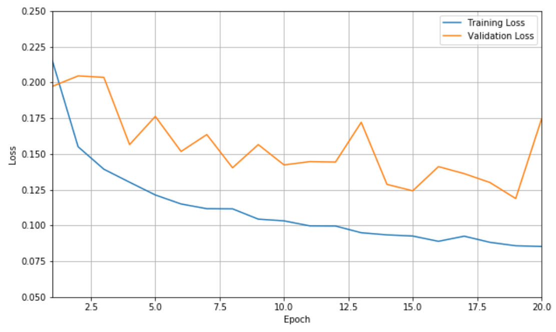

Figure 12. Deeper HardNet Loss vs Epoch

Deeper Baseline is a slightly deeper version of the Baseline Descriptor model with 3 additional concolutional layers. ResNET (He et al. 2015) used is imported from the Keras library with only the last two layers trained using the data provided while the rest of the layers are kept as the loaded weights from the library. It is seen that the pre-trained ResNET model, though is a much deeper model, failed to outperform even the baseline model. It is hence speculated that the pre-trained weights do not generalize well with the given data set. From figure 10, 11 and 12, it is deduced that Deeper HardNet is the most effective descriptor model but overfitting is observed starting from the 20th epoch hence early stopping is again introduced to stop training on the 19th epoch.

3.3. Best Proposed Model

The best proposed model is a combination of UNet and Deeper HardNet. The two models are trained to the point before they overfit i.e. 6 epoch for UNet and 19 epoch for Deeper HardNet. The final model is then evaluated using the evaluation method mentioned in section 1.1 and it gives a VRM score of 0.83, 0.232 and 0.54 which is a decent improvement from the baseline approach.

4. Conclusion

Out of all the models tested, UNet turns out to be the best denoising model while Deeper HardNet is the best descriptor model. The combination of the two gave a decent improvement under strict evaluation as compared to the baseline approach. If allow more time, additional features can be added to the improved model including hyperparameters tuning, data augmentation, increase in the number of layers of the neural net etc which might lead to even better models. ResNET might also perform better if all the layers are trained, not just the last two layers. Though significant time and computation power will be needed to fully train ResNET.

References

Balntas, V., Lenc, K., Vedaldi, A. & Mikolajczyk, K. (2017), 'HPatches: A benchmark and evaluation of handcrafted and learned local descriptors'.

He, K., Zhang, X., Ren, S. & Sun, J. (2015), 'Deep residual learning for image recognition;', **1**.

Kingma, D. P. & Ba, J. L. (2017), 'ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION'.

Mishchuk, A., Mishkin, D., Radenovic, F. & Matas, J. (n.d.), 'Working hard to know your neighbors margins: Local descriptor learning loss', **4**.

Ronneberger, O., Fischer, P. & Brox, T. (2015), 'U-net: Convolutional networks for biomedical image segmentation', **1**.

Appendix

Dataset

The data used in this report can be downloaded via https://imperialcollegelondon.box.com/shared/static/ah40eq7cxpwq4a6l4f62efzdyt8rm3ha.zip.

GitHub Repository

THe gitHub repository for this project can be found here https://github.com/yhl116/Deep-Learning-Coursework.

Instruction to Execute Source Code

1. The source code provided is written and tested under the Colab environment and is recommended to be run under the stated environment.

2. If source code is run in any other environment, comment out the 1st two code blocks which includes importing Google Drive and changing of directory.

3. The models tested are included in the source code. By default, the best proposed models: UNet and , are uncommented. Comment and uncomment the models needed and before running the source code.

4. Number of data samples used to train the models can also be modified with a commented segment of the code.

5. Change the optimizers from sgd to Adam if needed.

6. Comment out the saving of files into file.io or into Google Drive if unneeded.