Deep Learning - Coursework

Clive Toh Soon, Wong
Imperial College London
CID: 01044264 cw3915@ic.ac.uk

Abstract

This project aims to develop image representation (descriptors) of N-HPatches (noisy version of HPatches dataset) for measuring similarity between images. The general architecture consists of 2 stages of CNN, with the first denoises the images and the second learns the descriptors of size 128. The performance benchmark used are verification, matching and retrieval (VMR) mean average precision (mAP). With a baseline architecture using shallow U-Net and Hardnet, the VMR mAP are 80.2%, 19.6% and 48.9% respectively. The best performer experimented is the combination of U-Net and HardNet, with VMR mAP being 82.5%, 23.5% and 53.8%.

1. Introduction

This coursework aims to experiment with Neural Network (NN) to develop descriptors of N-HPatches for measuring the similarity between images. The dataset, utility functions and a baseline architecture that consists of a pipeline of denoising model and descriptor model are provided [1]. Together with aforementioned utility functions and baseline architecture, Python packages such as **Keras** [2], **Matplotlib** [3] and **NumPy** [4] are used to facilitate the machine learning experiment.

2. Formulation of the Learning Problem

2.1. Dataset

The data is split into 76 reference image sets for training and 40 reference image sets for validation. For each reference set, there is a large number of small images, which are cropped patches (17 patches) from the larger reference image. Hence, they represent some local parts of the reference image. Each patch is of size 32×32 pixels with 1 channel (grey-scale), consisting of 5 similar images which are related by some kind of geometric transformations. The goal is to learn a descriptor of size 128 that minimises L2 norm between patches that represent the same local part of the scene and maximises those that do not. To achieve this, a pipeline of denoise and descriptor networks are used.

2.2. Denoise Network

The denoise network uses Representation Learning [5] to encode noisy input images with encoder ϕ to a lower dimension, and then decode back with decoder ψ. In the process, the noise which is random will be "penalised" and features which are more consistent will be learned. This essentially denoises the images. Mean absolute error (MAE) is used as the loss function, i.e. $\frac{1}{n}\sum_{i=1}^{n}\left\|x_i - \psi(\phi(x_i)\right\|$, where n is the size of the images, i.e. $32 \times 32 = 1024$. MAE is used instead of a mean squared error (MSE) as MAE is more robust against outliers (the error is not squared).

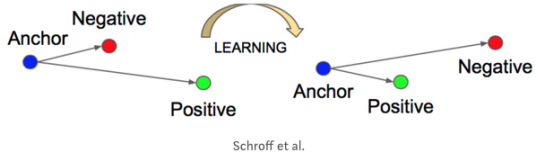

Figure 1. Illustration of the loss function in the triplet network. The network aims to minimise the L2 norm between anchor and positive input and maximise between anchor and negative input

2.3. Descriptor Network

The descriptor network aims to learn a descriptor of size 128. It is achieved using a triplet network using triplet inputs, namely anchor (**a**), positive (**p**) and negative (**n**), into the same network. Triplet loss, which is defined as $loss(\mathbf{a}, \mathbf{p}, \mathbf{n}) = max(\|\mathbf{a} - \mathbf{p}\|^2 - \|\mathbf{a} - \mathbf{n}\|^2 + \alpha, 0))$ will be used as the loss function to minimise the L2 norm between the **a** and **p**, and maximise between the **a** and **n**. The loss function of the network is illustrated in Figure 1 [6].

2.4. Evaluation Metrics

Three evaluation metrics are used, namely verification, matching and retrieval (VMR). Verification is done by classifying whether two images are similar or dissimilar; matching is to find similar images in a small size gallery with mostly difficult distractors; retrieval is to find similar images in a large gallery in which a large portion of them are distractors. To perform the evaluation, the L2 norms between the descriptors of the query image and gallery images are computed. The gallery will then be ranked according to the L2 norms to perform VMR [7]. For each task, mean average precision (mAP) will be calculated, which is defined as the average of the percentage of correct, positive predictions, i.e $precision = \frac{TP}{TP+FP}$, where TP and FP refer to True Positive and False Positive prediction respectively.

3. Baseline Approach

The baseline approach uses a pipeline of shallow version of U-net [8] and HardNet [9] to first denoise the image

patches, then train the denoised images based on the triplet loss to produce descriptors. The baseline architecture is illustrated in Figure 2.

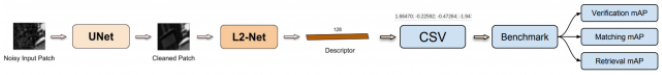

Figure 2. Illustration of the baseline architecture

3.1. Training Denoise Model with Shallow U-Net

Image samples are generated from 76 training sets and 40 validation sets using utility function **DenoiseHPatches** to produce batches where the input data is the noisy image and the label is the clean image. It generates 1,558,960 training samples and 952,512 validation samples. The training is done using **keras** method **fit_generator**, feeding training samples to parameter **generator** and validation samples to **validation_data**. For the optimiser, Stochastic Gradient Descent (SGD) [10] with learning rate of 0.00001, momentum of 0.9 using Nesterov's method [11] is used. The shallow U-Net architecture is illustrated in Figure 3.

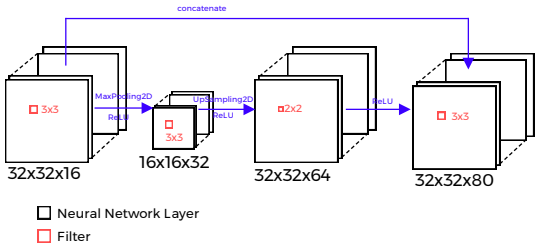

Figure 3. Shallow U-Net Architecture illustration

The shallow U-Net is run with 20 epochs. A plot of validation loss and loss of mean absolute error (MAE) against epoch is shown in Figure 4. It can be seen that both validation loss and loss decrease as the model are trained with more epochs. The lowest validation loss is observed at epoch 18 at 5.4626, and it takes roughly 860s per epoch on Google collab environment (12.9GB RAM, 11.4GB GPU RAM). Overfitting is observed after epoch 18 where the validation loss and training loss start to diverge, i.e. training loss decreases but validation loss increases. Since the validation loss is the lowest at epoch 18, its weights will be used to generate denoised images for the next stage. The denoised images are visualised in Figure 5.

3.2. Training Descriptor Model with HardNet

After the denoise model is trained, it is used to generate denoised images using the utility function **HPatches**. The denoised images are used by **DataGeneratorDesc** to generate 1,000,000 training samples and 100,000 validation samples, in which each sample consists of triplet of **a**, **p** and **n**. The image triplet is visualised in Figure 6. SGD with

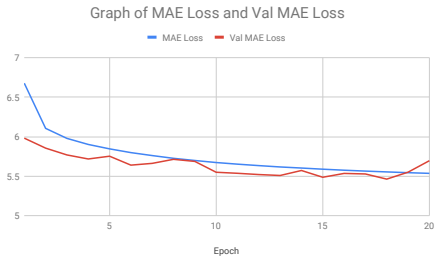

Figure 4. Plot of loss and validation loss of Shallow U-Net

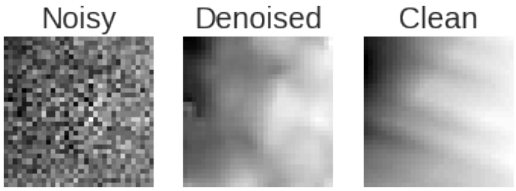

Figure 5. From left to right: Noisy image, image denoised by Shallow U-Net, and the clean image

Figure 6. Triplet image of Anchor, Positive and Negative image generated by **DataGeneratorDesc**

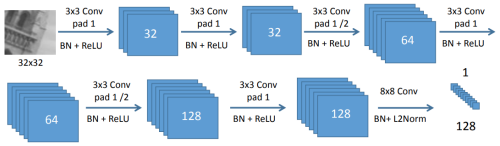

Figure 7. HardNet Architecture illustration, image taken from [9]

learning rate of 0.1 is used as the optimiser. The HardNet architecture is illustrated in Figure 7.

The HardNet is run with 20 epochs. A plot of validation loss and loss of triplet loss is shown in Figure 8. The lowest validation triplet loss is observed at epoch 17 at 0.1242. There is no overfitting observed as there is no divergence between the validation loss and training loss. The descriptor model could be trained with more epochs. However, due to the time constraint, weights from epoch 17 will be chosen for the VMR tasks in Section 5. It is justifiable since the validation loss remains relatively stable after epoch 15.

4. Improved Approach

To improve convergence speed and performance of learning to give lower val loss, Adam optimizer [12] is gen-

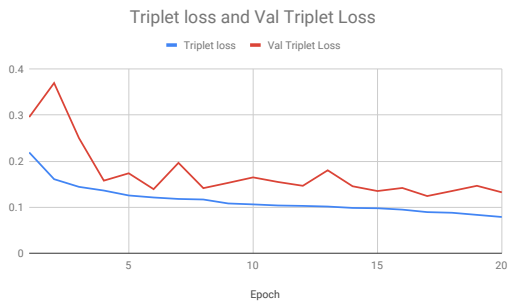

Figure 8. Plot of triplet loss and validation triplet loss of HardNet with images denoised by shallow U-Net

erally recommended [13]. Therefore, the following models will utilise Adam optimiser with default hyperparameters of **lr=0.001**, **beta_1=0.9 beta_2=0.999**, and **epsilon=1e-07**.

4.1. Improve denoise network with Full U-Net

A simple improvement is to use a full U-Net to denoise the images. However, the training time per epoch is increased from the average 20 minutes per epoch to 1.5 hour (4.5 times longer) on the Google Colab environment. With the same input as in Section 3.1, U-Net is run with 10 epochs. A plot of validation loss and loss of mean absolute error (MAE) against epoch is shown in Figure 9.

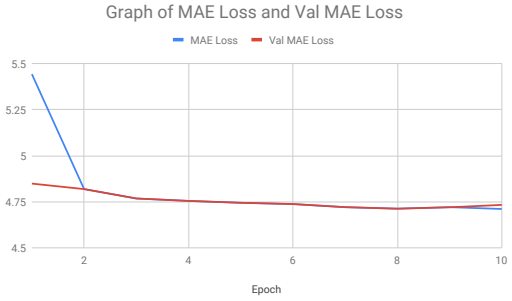

Figure 9. Plot of loss and validation loss of U-Net

Overall, the U-Net achieved better validation loss than the shallow U-Net, with the best validation error of 4.7126 at epoch 8. Although there is no overfitting observed and the U-Net could be trained with more epochs, due to the time constraint the weights at epoch 8 is used. It can be seen that U-Net produces better denoised images both quantitatively (depicted by lower validation error) and qualitatively, which is illustrated in Figure 10.

With the optimised U-Net, the procedures described in Section 3.2 are repeated to produce denoised images for the baseline HardNet. The result, a plot of triplet loss and validation triplet loss is shown in Figure 11. Notice that Adam optimiser is used in this HardNet, which allows the HardNet to perform better, as observed by the validation loss quickly

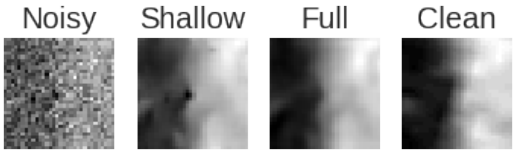

Figure 10. Comparison of image denoised by shallow U-Net and full U-Net. It can be seen that full U-Net produces denoised image closer to the clean image.

converges and stabilises after epoch 2 compared to SGD implementation, which converges after epoch 5 as shown in 8. With images denoised by U-Net, the HardNet is able to achieve a better validation loss of 0.109 at epoch 20. No overfitting is observed and the model could be trained with more epochs, but due to the time constraint the weights at epoch 20 will be chosen for evaluation.

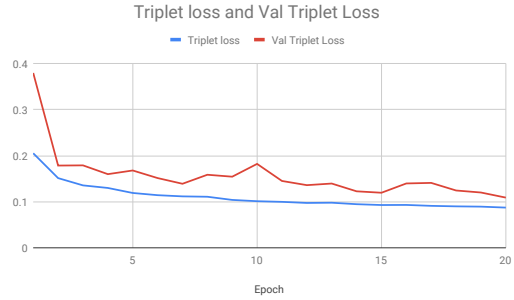

Figure 11. Plot of triplet loss and validation triplet loss of HardNet with images denoised by U-Net

4.2. Improve descriptor model

4.2.1 Deeper HardNetV1

Inspired by the architecture of U-Net, it is attempted to add 2 layers of skip connections to eliminate singularities inherent in the loss landscapes of deep networks[14]. Essentially, skip connections help to pass information through the network to mitigate the vanishing gradient when the architecture is deep. At the end of the network, a fully connected layers using dense layers with 512, 256, and 128 filters are added, connected by Batch Normalisation (BN) and Dropout layers to reduce overfitting. The proposed architecture is shown in Figure 12, named HardNetV1.

With the images denoised by U-Net as input, HardNetV1 is trained with 20 epochs and the result is shown in Figure 13. It is observed that HardNetV1 performed worse than HardNet, with the best validation triplet loss of 0.1685 at epoch 18. Even though skip connections layers could reduce singularities, there might be too many filters in each layer that lead to overfitting. There are 768 filters at the end of the second skip connection layer, and 12,288 ($4\times4\times768$) filters at the first fully connected layer. Even though batch

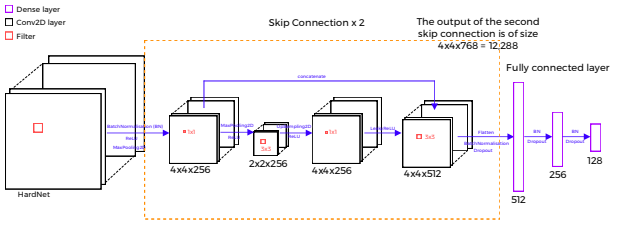

Figure 12. Illustration of proposed HardNetV1. 2 Skip connections are added to the output of the HardNet. At the end of the skip connection, a fully connected layers are added connected with BatchNormalisation and Dropout layers.

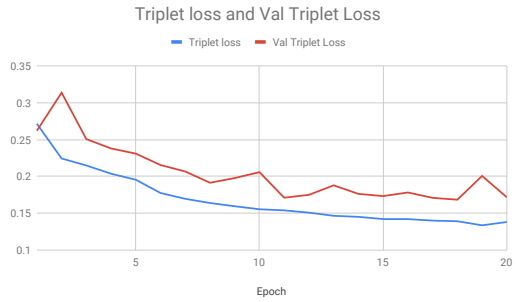

Figure 13. Plot of triplet loss and validation triplet loss of HardNetV1 with images denoised by U-Net

normalisation and dropout are used to deactivate 30% of the neurons to the next dense layer, it might not be sufficient. Therefore, HardNetV2 is proposed.

4.2.2 Deeper HardNetV2

HardNetV2 is an attempt to reduce the number of filters in the skip connections to 128, and replace fully connected layers with Global Average Pooling (GAP) layers [15]. GAP layers are used to reduce the spatial dimensions of a multi-dimensional tensor by taking the average of all dimensions, thereby reducing the total number of parameters in the model and hence minimise overfitting. Operation of GAP is illustrated in Figure 14 [16]. HardNetV2 is highly similar to HardNetV1 except for the size of the skip connection filter being 128 and a GAP layer at the end, hence Figure 12 represents this model substantially.

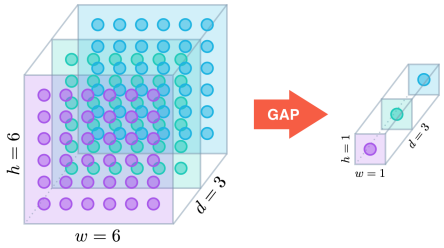

Figure 14. Illustration of GAP reducing tensors of size $h \times w \times d$ into a size of $1 \times 1 \times d$, image taken from [16]

HardNetV2 is trained with 20 epochs, the result is shown in Figure 15. The result improved slightly compared to HardNetV1, with the lowest validation loss of 0.1377 at epoch 19. This shows that overfitting is reduced by reducing filters and using a GAP layer. However, the validation loss is still higher than the HardNet, which could be due to deeper layers deployed in HardNetV2.

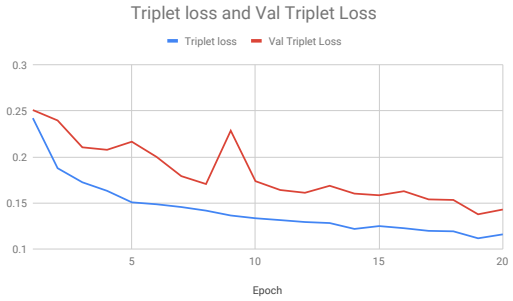

Figure 15. Plot of triplet loss and validation triplet loss of HardNetV2 with images denoised by U-Net

5. Evaluation

For evaluation, VMR will be carried out on the following candidates: Shallow U-Net + HardNet (baseline), U-Net + HardNet, U-Net + HardNetV1 and U-NEt + HardNetV2. The results are summarised in Table 1.

Models	Verification	Matching	Retrieval
Baseline	80.2%	19.6%	48.9%
U-Net + HardNet	82.5%	23.5%	53.8%
U-Net + HardNetV1	74.9%	3.6%	30.5%
U-Net + HardNetV2			

Table 1. Table shows the VMR mAP for baseline, U-Net+HardNet, U-Net+HardNetV1 and U-Net+HardNetV2 models

As argued in Section 4.2.1, HardNetV1 might have been overfitted. This is exemplified in the VMR tasks where HardNetV1 overall performed the worst, especially in Matching. The best performer in the VMR task is the combination of U-Net and HardNet.

6. Conclusion

In conclusion, it is shown that full U-Net improves the quality of the denoised images both quantitatively and qualitatively. For the descriptor network, baseline model (HardNet) performs the best. Attempt to add more layers with HardNetV1 and HardNetV2 lead to slight overfitting with higher validation triplet loss and worse VMR mAP. Also, attempt to experiment with ResNet failed due to broken implementation of ResNet in keras [17]. Shall more time be given, Data Augmentation could be carried out to generate more data to better train both networks.

7. Appendix

7.1. Source code

Source code can be found on my Github repository: `https://github.com/CliveWongTohSoon/EE3-25-Deep-Learning-2018-2019-CW3915`. It contains brief introduction and instructions on how to execute the code and reproduce the result. The source code is at the file **cw3915_DL.ipynb**.

7.2. Instruction to execute the source code

1. It is recommended to run the source code on Google Collab environment. If it is desired to run on local machine, Jupyter Notebook is needed, and comment out the first cell of the source code, i.e.

   from google.collab import drive

   drive.mount('/content/gdrive')

2. If the code is runing on Google Collab, a Google Drive account is needed to persist the data. Create a Google Drive account if you do not have one.

3. Clone dataset and utility funcitons provided by Imperial College London.

4. Create your keras model. For descriptor model, you could use a wrapper function I created called **load_descriptor_model**. It takes in your descriptor model and compile it with triplet loss mentioned in this report.

5. To execute VMR tasks, use the wrapper function called **run_vmr** which takes in descriptor model and denoise model. A output path is also required. Notice the output will be generated and persisted on Gogole Drive, so make sure there is enough space in your drive. The function will print out VMR mAP.

References

[1] Matchlab - imperial. URL https://github.com/MatchLab-Imperial/keras_triplet_descriptor.

[2] Keras: The python deep learning library. URL https://keras.io/.

[3] Matplotlib: Python plotting, 2018. URL https://matplotlib.org.

[4] Numpy, 2018. URL http://www.numpy.org.

[5] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

[6] Marc-Olivier Arsenault. Lossless triplet loss, a more efficient loss function for siamese nn. Feb 2018. URL https://towardsdatascience.com/lossless-triplet-loss-7e932f990b24.

[7] Krystian Mikolajczyk and Carlo Caliberto. Ee3-25: Deep learning. 2019. URL https://bb.imperial.ac.uk/bbcswebdav/pid-1514885-dt-content-rid-4911418_1/courses/DSS-EE3_25-18_19/DL2019_partCoursework.pdf.

[8] Robert L Barkau. Unet: One-dimensional unsteady flow through a full network of open channels. user's manual. Technical report, Hydrologic Engineering Center Davis CA, 1996.

[9] Anastasiia Mishchuk, Dmytro Mishkin, Filip Radenovic, and Jiri Matas. Working hard to know your neighbor's margins: Local descriptor learning loss. In *Advances in Neural Information Processing Systems*, pages 4826–4837, 2017.

[10] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

[11] Yurii Nesterov. *Introductory lectures on convex optimization: A basic course*, volume 87. Springer Science & Business Media, 2013.

[12] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[13] Vitaly Bushaev. Adam - latest trends in deep learning optimization. *Medium*, Oct 2018. URL https://towardsdatascience.com/adam-latest-trends-in-deep-learning-optimization-6be9a291375c.

[14] A Emin Orhan and Xaq Pitkow. Skip connections eliminate singularities. *arXiv preprint arXiv:1701.09175*, 2017.

[15] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.

[16] Alexis Cook. Global average pooling layers for object localization. *Blog*, Apr 2017.

[17] sakvaua. Batchnorm in shared layers goes to nan, Dec 2018. URL https://github.com/keras-team/keras/issues/11927.