Deep Learning Coursework Final Report

Tim Hung Wu
thw116
CID: 01195701
thw116@ic.ac.uk

Abstract

In this report, we explore methods of improving descriptor generation for images. A baseline architecture was provided consisting of two steps: initial denoising followed by description calculations for each image using a convolutional neural network. The images are from the HPatches dataset[1], which also provides a companion benchmark for evaluating local image descriptors. The metrics are the mean Average Precisions (mAP) for three tasks: Verification, Matching and Retrieval. We will study the iterative design process for machine learning models, focusing on model architecture designs, hyperparameter adjustments and training methods.

1. Problem Formulation

The problem revolves around trying to build a pipeline to recognise how similar or different a selected patch is to a queried patch. Synthetic noise is also added to inputted patches and so they will initially require a denoising model to restore them to as close to the original patch as possible. The objective is then to output a good descriptor that will have images similar to the query closer in vector space to it than other more dissimilar images. This will be assessed by calculating the L2 distance between the image and query: $\sqrt{(x_{query} - x_{image})^2}$. In the case where multiple patches are compared to the query, a gallery is formed, ranking them in order of similarity. Figure 9 is a visual example. Real world applications of this problem extend to facial recognition, robotics navigation, 3D model design etc.[11]

1.1. Data

HPatches data is structured into a sequence of patches, of which each sequence contains the same patch with various illumination or homography transformations. Local image patches are used rather than whole images to allow experiments to be more replicable as it helps avoid biases, such as semi-local constraints which may not be present from one image to another. Data is then augmented by adding noise into the patches with underlying geometric distributions. This dataset is called N-HPatches[1] and are 32x32 pixels in size. They are categorised into three levels of difficulty: Easy, Hard and Tough.

1.2. Evaluation Metrics

The three metrics involve diverse tasks. Verification looks at two patches and determines whether they match, forming a binary classification problem. Retrieval attempts to find similar images in a large gallery whilst Matching attempts this in a small gallery but with more difficult distractors. The overall performance of a pipeline is judged through the mAP, defined as $\frac{1}{n}\sum_{i=1}^{n}\frac{p^i_{relevant}}{p^i_{retrieved}}$ where n is the number of queries and $p^i_{relevant}, p^i_{retrieved}$ are the number of relevant and total retrieved patches to a query i.

2. Baseline Approach

The baseline consists of two consecutive networks: a shallow UNet[12] for denoising and a L2-Net[14] for generating descriptors. Below are diagrams of the architectures. The denoiser input is the noisy 32x32 patch, the output is a denoised version. The descriptor input is the 32x32 denoised patch, the output is a 128 vector descriptor.

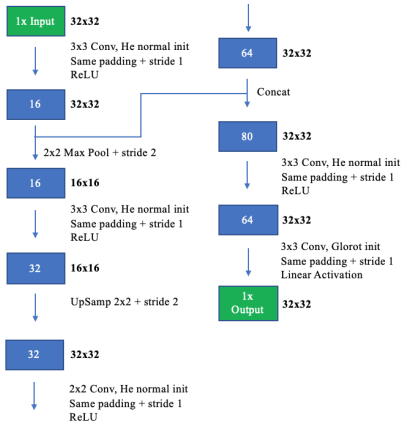

Figure 1. Baseline Denoiser Architecture (Shallow UNet)

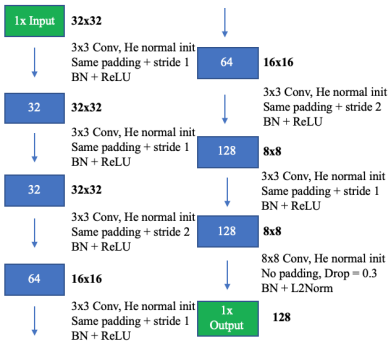

Figure 2. Baseline Descriptor Architecture (L2-Net)

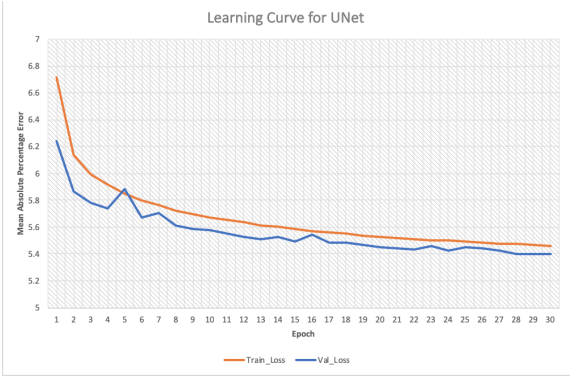

Figure 3. Learning curve of UNet

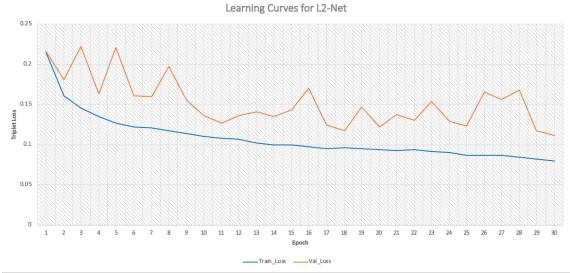

Figure 4. Learning curve of L2-Net

2.1. Baseline Training

The denoiser trained using the Mean Absolute Error (MAE) loss function: $\frac{1}{n}\sum_{i=1}^{n}|y_i - x_i|$ where y_i is the actual value and x_i is the predicted value. The descriptor used the Triple Loss Function, formula from Balntas et al[3]: $max(0, \delta_+ - \delta_- + 1)$, where $\delta_+ = ||f(a) - f(p)||_2, \delta_- = ||f(a) - f(n)||_2$ and a, p, n are the anchor, positive and negative patches respectively.

The default training and validation sets splits was type $'a'$[2] which is a random mix of illuminatory and homographic transformed patches. For the whole dataset, there were 76 training and 40 validation sequences. The default random sampling split for the denoiser was 3:1 training to validation. For the descriptor, the number of triplets formed splits rather than sequences and this was initialised to 10:1 training to validation. Testing also uses the validation set and so consists of 40 sequences.

The optimiser in both cases was Stochastic Gradient Descent (SGD), with learning rate and Nestorov momentum initialised to 0.00001 and 0.9 respectively in the UNet and learning rate initialised to 0.1 in the L2-Net. Batch size was initially set at 50. Early stopping was used to determine the number of Epoch runs required. Each epoch consisted of 31179 batches. He normal weight initialisation ($Var(W_l) = \frac{2}{d^l}$) where l is the layer and d is the number of layer outputs) was used since activation functions were ReLU. The learning curves of both models and final metric performance results are provided below. Figure 10 in the appendix highlights performances by other models for comparison.

	Verification	Matching	Retrieval
mAP	0.821939	0.231602	0.536964

Table 1. Mean Average Precision for the baseline model.

2.2. Evaluation

Surprisingly, the training error for the denoiser was actually higher than validation error. This could stem from Keras as its training error is averaged across all batches in the epoch whereas validation error is calculated on the last: $\frac{1}{b}\sum_{b=1}^{n} l_b^{train} > l_n^{valid}$ where l is loss b is batch number. This suggests that the model might still be learning and underfitted meaning, in theory, should train for more epochs but this wasn't possible due to GPU limitations.

Then, hyperparameters were tuned. In order to speed up the process, this was completed on 10 epochs and sample training-validation sizes of 8:2. The final epoch's validation losses were then compared to determine a relative performances between models. This could potentially consequently penalise models with initial poorer learning rates but better later generalisation. Efforts were made to use a number of epochs where relative performance is already determined i.e. $\lim_{e\to\infty} validLoss(f_1(e)) < validLoss(f_2(e))$ where e is epoch number.

Batch sizing was adjusted. Powers of 2 were used to improve performance by aligning virtual processors onto the physical processors of the GPU, which are often a power of 2. The tradeoff for batch sizes are that smaller ones are noisier and therefore may not be moving down the error function in the direction of steepest descent. However, Keskar et al[6] reported that they can help navigate out of sharp minima, characterised by large positive eigenvalues in $\nabla_2 f(x)$,

that often generalise less well. Large batch sizes often fail to navigate out of these. 16 was established to be the best size and results are in Table 3. of the appendix. However, 32 was used instead to reduce computational time of the epochs.

The optimiser was then changed to see if an adaptive learning method would improve the model. Adam[8] was selected as Ruder[13] assesses it to be better than Adagrad and RMSProp due to its component that mimics momentum. The weight updates for SGD and Adam are as follows: $W_{t+1} = W_t - \eta Z_{t+1}$ where W is weight, η is the learning rate, Z_{t+1} is the momentum update and $W_{t+1} = W_t - \frac{\eta}{\sqrt{v_t+\epsilon}} \odot m_t$ where m_t, v_t are the beta 1 and 2 decay parameters and ϵ is to prevent division by 0.

The loss function was then changed to Mean Squared Error (MSE): $\frac{1}{n}\sum_{i=1}^{n}(y_i - x_i)^2$ where y_i is the actual value and x_i is the predicted value. This penalises outliers more but upon testing, this produced markedly higher training and validation losses (approx 2.5x).

3. Proposed Improvements

Architecturally, it was decided to deepen the shallow UNet to enhance denoising performance. This was based on Mhaskar's findings[10] which suggested that deep networks can fit functions better with less parameters than a shallow network with the same computational power i.e. number of neurons. This lowers the VC dimensionality since $d_{VC}(H)$ = number of parameters where H is the Hypothesis space, which in turn lowers the theoretical test error bound: $e_{test}(h) <= e_{train}(h) + \sqrt{\frac{8d_{VC}(H)}{n} log(2n+1) + \frac{8}{n} log\frac{4}{\delta}}$. This is because, while both networks can approximate any function, the structure of a deep network is better at approximating compositional functions and thus extracting some additional types of features. Figure 11 depicts two networks. The shallow one can approximate functions of the form $f(x_1, .., x_8)$. The deep one is of the form $f(n_3)$ where $n_3 = (n_{21}, n_{22})$, which in turn are $n_{21} = (n_{11}, n_{12}) = (n_{11}(x_1, x_2), n_{12}(x_3, x4))$ etc. This structuring helps with compositional function formulation e.g. $h(g(f(x)))$. Shallow networks are also typically wider, meaning they are poorer generalisers and are susceptible to overfitting the training data. For example, a one-layer polynomial regression network with $n-1$ nodes can always perfectly fit to n datapoints i.e. $y = ax^2 + bx + c$ can fit 3 data points.

The proposed architecture is based off the DeepUNet[9]. This conveniently builds on the current model, providing more layers in its contracting and expanding paths (4 MaxPools + UpSamples compared to 1). The original had approximately 21 million parameters so we reduced the depths of each layer by a factor of 4 to reduce this down to 1.3 million. This was done since the original version trained on much larger images which may require contain more complex features than our 32x32 patches, which may have resulted in overfitting had we used the original.

It was difficult to implement a new architecture for our descriptor given the Siamese structure of the network. This made the impact of triplet loss more complex when updating the weights during backpropogation in training as its effects three inputs: anchor, positive and negative patches. Instead, we looked at rectifying the instability of validation loss in the baseline model, by adjusting the train-validation splits, optimiser and its hyper-parameters.

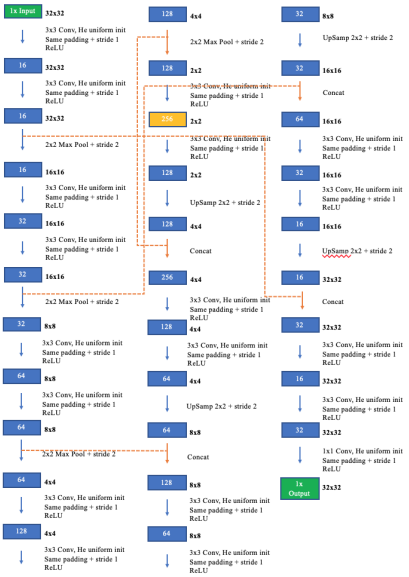

Figure 5. Proposed Denoiser Architecture (DeepUNet)

3.1. Training and Evaluation

Because of GPU limitations, a smaller data set had to be used. This meant picking a ratio of training-validation sets. Guyon[4] specifies a theoretical method of deriving good ratios. The concept follows: $minimise\ p_{val}(t) - p_{opt}(t) = (p_{val}(t) - p_{val}(gt)) + (p_{val}(gt) - p_{opt}(gt)) + (p_{opt}(gt) - p_{opt}(t))$ where t, gt are are all and training examples respectively and p_{val}, p_{opt} are the probabilities of error in the validation and test sets. We focus on the second and third bracketed terms, which are positive, and are known as the 'uncertainty' and 'learning' curves. Figure 12 shows the trade off between these to minimise the difference between error in the optimal and validation cases with respect to proportion of validation set allocation. These terms are then equated to: $(p_{val}(gt) - p_{opt}(gt)) = \frac{Cln(N/\alpha^2)}{t}\frac{1}{f}$ and $(p_{opt}(gt) - p_{opt}(t)) = \frac{h_{max}}{t}\frac{f}{1-f}$ where C is the Chernoff bound constant, N is the number of families of learners considered, α is the probability of being incorrect, t is the number of test cases, h_{max} is the highest complexity of the learners and f is the proportion allocated to the validation

set. The optimal value is found by differentiating and locating the minima but we can see in our case that h_{max} has increased in the deeper UNet and therefore f_{opt} should decrease relatively. We therefore decided to increase the base ratio of 2:1 train-validation to 4:1, often referred to as the Pareto principle.

The case of weights initialisation was also considered due to the additional depth of the network. The original DeepUNet used Glorot uniform initialisation, which samples uniformly from limits: $\pm\sqrt{\frac{6}{n_{in}+n_{out}}}$ where n_{in}, n_{out} are number of input and outputs to the layer. Adjusting these to He normal initialisation caused a very large decrease in performance whereas He uniform (limits: $\pm\sqrt{\frac{6}{n_{in}}}$) marginally improved performance.

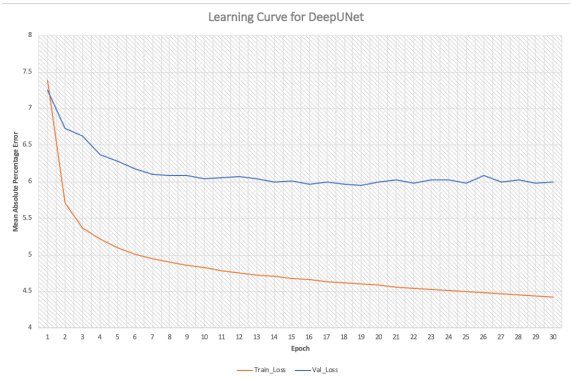

Figure 6. Initial DeepUNet Training, 8:2 train-validation split.

Initial monitoring showed validation loss decrease rapidly initially but converging quickly at a high loss of approximately 6%. This showed signs of too high a learning rate. Because we were unable to use the whole dataset, Adam optimisation was used in order to obtain a faster learning rate. However, this was set to $\frac{1}{10}$ of the default value of 0.0001 to compensate for the high validation loss convergence experienced. This can be found below, training on as much of the dataset as possible (28 sequences total).

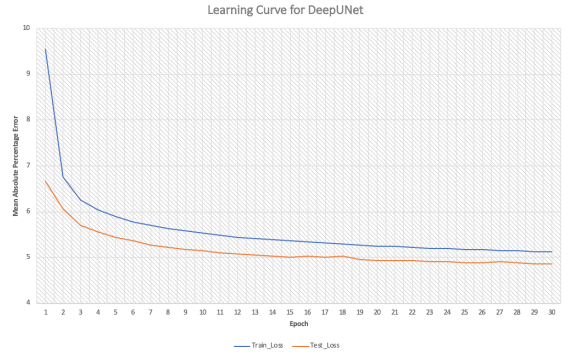

Figure 7. Optimised DeepUNet, 28:7 train-validation split.

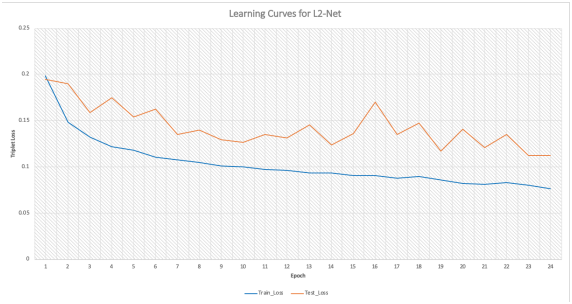

Figure 8. Optimised L2-Net, 120k:30k train-validation split.

In order to improve the stability of validation loss in the L2-Net, we focused on the optimiser, learning rate and training-validation splits. The number of total training-validation cases was increased from 100k:10k to 120k:30k, thus also increasing the split from 9:1 to 8:2. This was done on the theory that the fluctuations in validation loss may come from noise variance due to a small absolute sample size. The optimiser was changed to Adam and learning rate was increased to 0.001, seeing that the baseline model had a high SGD learning rate of 0.01. Below are the results of the evaluation benchmark for the upgraded denoiser and descriptor for 28 training and 7 test sequences.

	Verification	Matching	Retrieval
mAP	0.806913	0.217416	0.523319

Table 2. Mean Average Precision for the improved model.

4. Conclusion

The improved model did not beat the baseline in absolute terms with respect to mAP metrics. However, given the fact that it was trained on only 30% of the whole dataset due to GPU limitations, it performed better relatively i.e. it is better if the baseline trained on the same amount of data. It should be noted that early stopping was never triggered because of the limited number of epochs that could be run. Additionally, Keskar et al[7] indicate that SGD may in fact generalise better than Adam but requires many more epochs to train with due to the lower learning rate. This could indicate our hyperparameter tuning was not the most optimal given the small sample and epoch sizes that were used.

If it were possible to run higher dimensioned models well in Colab, a more expressive denoiser could be used. This could lead to using dropout regularisers, which were neglected in this experimentation. In variations of UNet[5], dropouts of 0.5 were used in the deepest layer (depths of 1024+), of the bottleneck to avoid overfitting. Finally, the descriptor could also be trained on the clean images, along with adjusting the bias of the triplet loss function to see if that would further improve the model.

5. Appendix

Figure 13 shows the experimental process taken.

Figure 9. Example of gallery rankings for patch query

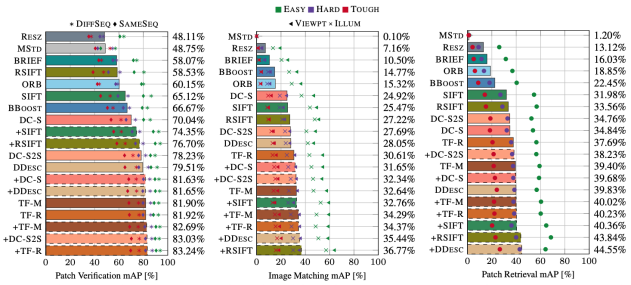

Figure 10. Results of various models, with bars indicating mAP

Batch Size	Validation Error
16	5.8736
32	6.2644
50	7.1674
64	6.4122

Table 3. Batch size testing on baseline denoiser model.

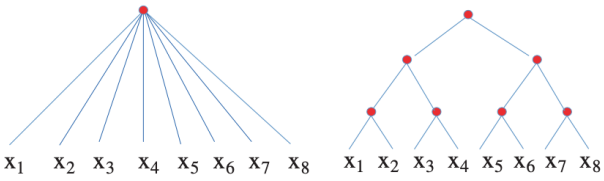

Figure 11. LHS: shallow network, RHS: deep network

5.1. Code Base

The code base can be found in GitHub repository with the link here.The zip file can be downloaded by clicking the green button. The code base is within the Jupyter notebook titled $DL\_CW.ipynb$.

References

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

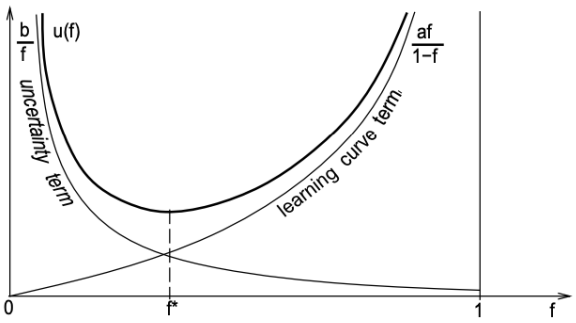

Figure 12. How loss (u(f)), is affected by proportion of validation set (f)

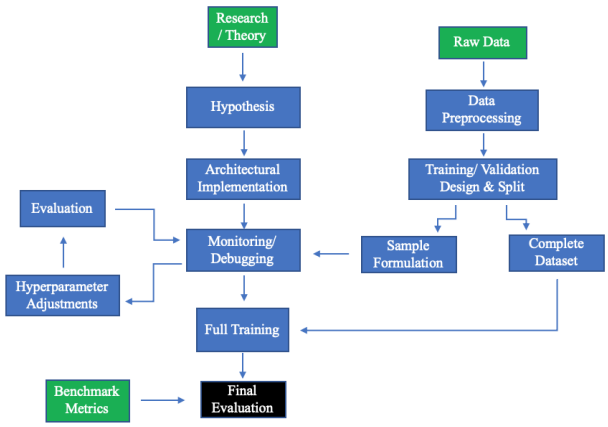

Figure 13. Experimentation process cycle

[2] V. Balntas, K. Lenc, A. Vedaldi, and K. Mikolajczyk. Hpatches, homography patches dataset: Training/test splits. Github: https://github.com/hpatches/hpatches-benchmark/blob/master/docs/splits.md, 2017.

[3] V. Balntas, E. Riba, D. Ponsa, and K. Mikolajczyk. *Learning local feature descriptors with triplets and shallow convolutional neural networks*. BMVA, 2016.

[4] I. Guyon. *A scaling law for the validation-set training-set size ratio*. ATandT Bell Laboratories, 1996.

[5] Y. Han and J. C. Ye. *Framing U-Net via Deep Convolutional Framelets: Application to Sparse-view CT*. arXiv, 2018.

[6] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang. *On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima.* arXiv, 2017.

[7] N. S. Keskar and R. Socher. *Improving Generalization Performance by Switching from Adam to SGD*. arXiv, 2017.

[8] D. P. Kingma and J. Ba. *Adam: A Method for Stochastic Optimization*. arXiv, 2014.

[9] R. Li, W. Liu, L. Yang, S. Sun, W. Hu, F. Zhang, and W. Li. *DeepUNet: A Deep Fully Convolutional Network for Pixel-level Sea-Land Segmentation*. arXiv, 2017.

[10] H. Mhaskar, Q. Liao, and T. Poggio. *When and Why Are Deep Networks Better than Shallow Ones?* arXiv, 2017.

[11] K. Mikolajczyk and C. Ciliberto. Ee3-25: Deep learning, 2019.

[12] O. Ronneberger, P. Fischer, and T. Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. arXiv, 2015.

[13] S. Ruder. *An overview of gradient descent optimization algorithms*. arXiv, 2017.

[14] Y. Tian, B. Fan, and F. Wu. *L2-Net: Deep Learning of Discriminative Patch Descriptor in Euclidean Space*. CVPR, 2017.