Deep Learning Report

Heran Zhang
hz4915
01054578
hz4915@ic.ac.uk

Abstract

This report aims to evaluate a baseline architecture in developing image descriptors using Shallow-UNet denoise model and HardNet descriptor model and thus develop further optimization methods to improve the image representations. The task of the project is to match noisy patches to the correct neighbour while keeping them away from patches of different physical view point. The method will be evaluated using HPatches as benchmarks. Performance will be verified by measuring mAP on verification, matching and retrieval tasks.

1. Formulation of the Problem

The goal of the project is to develop good image representation known as the descriptors for measuring similarity between images, for which differences between similar images should be kept small and differences between dissimilar images should be maximised.

$$\min_{i,j \in S} d(x_i, x_j) \quad \text{and} \quad \max_{i,j \in D} d(x_i, x_j) \tag{1}$$

Where S is the similar set and D denotes the dissimilar set.

The data set N-HPatches being used for this project is a noisy version of the original HPatches[1] data set. HPatches is based on 116 image sequences, in which 59 of them shows geometric deformations due to viewpoint changes and the other 57 sequences presenting photometric distortions[1].

On the other hand, N-HPatches contain images obtained from applying random synthetic transformations on different levels which can critically affect the descriptor training results. Therefore, for the first part of the project, the aim is to evaluate the upper bound using clean patches for the descriptor training network by hyper-parameter tuning with different optimizer. Then select the descriptor training model with best accuracy as a control variable and manipulate the denoise training model to minimize the impacts made by noisy patches.

The patches are downsampled from original size of 65x65 to 32x32 and the performance will be evaluated by testing the descriptor in the noisy patches.

The input $x \in \mathcal{X}$ are patches of size 1x32x32, each representing the depth, width and height of the input image while the output $y \in \mathcal{Y}$ is the descriptor of dimension 128x1. The objective is to learn a hypothesis class $\mathcal{H} \subset \{h : \mathcal{X} \to \mathcal{Y}\}$ such that the loss between $g \in \mathcal{H}$ and $f : x \to y$ is minimized. The loss function we used for training is the validation loss measured by MAE(mean absolute error).

$$MAE = \frac{1}{n}\sum_{i=1}^{n} |y_i - \hat{y_i}| \tag{2}$$

where y_i is the test output value and $\hat{y_i}$ is the predicted output value.

The verification process will be performed by measuring the classification accuracies on *verification*, *matching* and *retrieval* tasks using mAP (mean Average Precision).

2. Descriptor Training Model Evaluation

2.1. Baseline Approach

The baseline code is using a L2-Net architecture with 7 layers of CNN as illustrated in Figure 3[6] and SGD optimizer with a static learning rate of 0.1. The model was fitted through 1 epoch only and the result is shown below.

val_loss	Verification	Matching	Retrieval	Mean
0.1949	0.763667	0.176211	0.436518	0.458799

The above result was retrieved using a train-test split of 19:10 (76, 40) base of the 116 image sequences given. The matching verification is expected to be the lowest score since the gallery data set contains more difficult distractors then the others. Given that the training only went through one epoch, the loss function may not converge and hence increase the number of epochs can be a simple improvement. Moreover, fixed learning rate could have the potential to stuck at the boundary and does not converge, thus introducing an annealing schedule where the learning rate decreases with a fixed factor if the loss function doesn't improve can be a better approach.

2.2. Hyper-parameter Tuning

2.2.1 Epochs

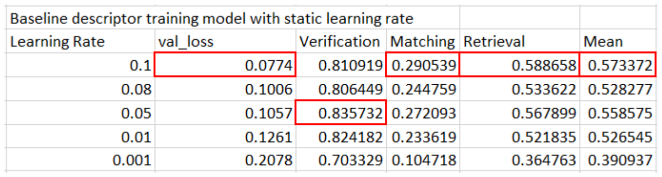
Now we introduce a simple improvement to the baseline learning by increasing the epochs to 10 instead of running through just 1 epoch. Thus the descriptor model of the previous epoch will be fed back to input of the next epoch for optimization improvement. The result is shown below:

val_loss	Verification	Matching	Retrieval	Mean
0.0774	0.840919	0.290539	0.588658	0.573372

The overall accuracy for the model had a significant improvement. However, greater value of epoch could result in over-fitting to the training data causing degradation in performance and require much longer runtime. Thus a value of 10 is being used throughout the report.

2.2.2 Static Learning Rate

Another common hyper-parameter adjustment can be made on the learning rate of the model. Learning rate determines the how much we are adjusting the weights of our network with respect to the loss gradient. Thus a small value will slow down the process where we travel along the downward slope. While a large value can overshoot the minimum and fail to converge, or even diverge[3]. Thus choosing a suitable learning rate for the task is essential. The below table has shown the results for baseline SGD approach using different static learning rate:



Baseline descriptor training model with static learning rate

Learning Rate	val_loss	Verification	Matching	Retrieval	Mean
0.1	0.0774	0.810919	0.290539	0.588658	0.573372
0.08	0.1006	0.806449	0.244759	0.533622	0.528277
0.05	0.1057	0.835732	0.272093	0.567899	0.558575
0.01	0.1261	0.824182	0.233619	0.521835	0.526545
0.001	0.2078	0.703329	0.104718	0.364763	0.390937

Figure 1. Descriptor training model with baseline SGD using different static learning rates

The result indicates that an initial learning rate of 0.1 already yields the best result among all. Therefore in order to improve the result further, we can employ a annealing schedule for the learning rate and adjust the learning rate by a constant factor if the loss function cease to improve.

2.2.3 Learning Rate with Annealing Schedule

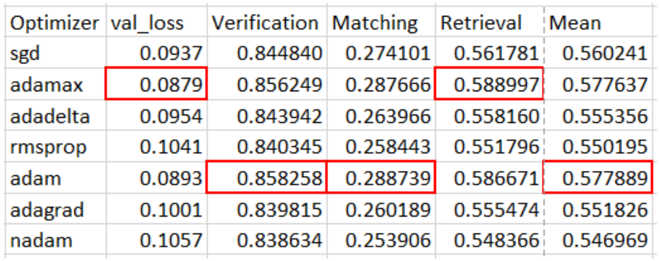
Moreover, looking at the validation loss and training loss over the epochs when learning rate is 0.1 shown in Figure 4, we can notice that the validation loss of the model fluctuates while the training loss is smoothly descending. This is because the learning was performed on the training set performance but has less contribution to the validation set. Thus we propose a annealing schedule that tracks the validation loss instead of training loss, if the changes in validation loss is less than a threshold value of 0.01, the learning rate will be decreased by a factor of 0.2. This method can also avoid over fitting to the training set and generate a more generic descriptor to the data.

By running the baseline approach alone with the annealing schedule through 10 epochs, there is a significant improvements compare to the baseline model but slightly under-performed compare to the static learning rate.

val_loss	Verification	Matching	Retrieval	Mean
0.0984	0.844840	0.274101	0.561781	0.560241

2.3. Optimizer Evaluation

Same methodology was apply across other optimizers and the result summary table is shown in Figure 2, where the highlighted boxes are the top performance of that measurement.



Optimizer	val_loss	Verification	Matching	Retrieval	Mean
sgd	0.0937	0.844840	0.274101	0.561781	0.560241
adamax	0.0879	0.856249	0.287666	0.588997	0.577637
adadelta	0.0954	0.843942	0.263966	0.558160	0.555356
rmsprop	0.1041	0.840345	0.258443	0.551796	0.550195
adam	0.0893	0.858258	0.288739	0.586671	0.577889
adagrad	0.1001	0.839815	0.260189	0.555474	0.551826
nadam	0.1057	0.838634	0.253906	0.548366	0.546969

Figure 2. Descriptor model training result summary for different optimizers using clean patches

The hyper-parameter configurations for all the optimizers are shown in Figure 6. We can deduce that $adam$ and $adamx$ are the best choices for descriptor training. Moreover, if we look at the validation loss over the epochs as shown in Figure 7. $adam$ and $adamax$ does converge faster than the other optimizers and yields a better validation loss.

2.4. Triplet Loss Evaluation

The descriptor model was trained by learning with triplets which involves training from samples of the form $\{\mathbf{a}, \mathbf{p}, \mathbf{n}\}$, where $\mathbf{a}$ is the $anchor$, $\mathbf{p}$ is positive and $\mathbf{n}$ is negative. For our usage, **a** and **p** are different viewpoints of the same image, and **n** is a viewpoint from different image. Thus the aim is to optimise the parameters of the network which brings **a** and **p** closer in the feature space, and brings **a** and **n** further apart[7].

$$\delta_+ = \|f(\mathbf{a}) - f(\mathbf{p})\|_2 \quad \text{and} \quad \delta_- = \|f(\mathbf{a}) - f(\mathbf{n})\|_2 \tag{3}$$

A diagram can be seen in Figure 8

2.4.1 Triplet Margin Ranking loss

This was invoked in the baseline triplet loss function first proposed in [8], which is defined as

$$\lambda(\sigma_+, \sigma_-) = \max(0, \mu + \sigma_+ - \sigma_-) \tag{4}$$

where μ is a margin parameter. The margin ranking loss measures the violation of the ranking order of the embedded features inside the triplet to ensure $\delta_- > \delta_+ + \mu$.

2.4.2 Triplet Ratio loss

Different to ranking loss, a ratio loss aims to optimize the ratio distances within triplets, which learns embeddings such that $\frac{\delta_-}{\delta_+} \rightarrow \inf$ illustrated in [2]

$$\hat{\lambda}(\delta_+, \delta_-) = (\frac{e^{\delta_+}}{e^{\delta_+} + e^{\delta_-}})^2 + (1 - \frac{e^{\delta_-}}{e^{\delta_+} + e^{\delta_-}})^2 \tag{5}$$

The goal of this loss function is to force $(\frac{e^{\delta_+}}{e^{\delta_+}+e^{\delta_-}})^2$ to 0, and $(\frac{e^{\delta_-}}{e^{\delta_+}+e^{\delta_-}})^2$ to 1. The result is shown below:

Verification	Matching	Retrieval	Mean
0.841905	0.268845	0.558933	0.556561

2.4.3 In-Triplet Hard Negative Mining

The previous ideas of triplet loss ignore the third distance $\delta'_- = \|f(\mathbf{p}) - f(\mathbf{n})\|_2$. Note that since the feature embedding network already computes the representations for $f(\mathbf{a})$, $f(\mathbf{p})$, $f(\mathbf{n})$, thus no need for extra convolutional overhead to compute δ'_- except evaluating the L_2 distance. Thus we can introduce the third constraint known as $in - triplethardnegative$ as $\delta_* = min(\delta_-, \delta'_-)$ and subsequently, the margin ranking loss becomes $\lambda(\delta_+, \delta_*) = max(0, \mu + \delta_+ - \delta_*)$. This approach can result in improvement in performance without computational overhead as shown below[7]:

Verification	Matching	Retrieval	Mean
0.878279	0.338602	0.639451	0.618777

2.5. Number of Triplet Pairs

Although the training of descriptor model was performed on all available training data and validation data, it did not make use of all the triplet loss pairs available. The default number of triplet loss pairs generated was 100000 for training and 10000 for validation. Thus increasing the number of triplet loss pair samples for training can boost the learning indefinitely. For comparison, I had increased both validation samples and training samples for 5 times, the result is shown below:

train pairs = 100000, val pairs = 10000

val_loss	Verification	Matching	Retrieval	Mean
0.14360	0.860462	0.304832	0.601346	0.588880

train pairs = 500000, val pairs = 50000

val_loss	Verification	Matching	Retrieval	Mean
0.09520	0.898772	0.400098	0.702471	0.667114

Both training were performed using $Adam$ optimizer with $lr = 0.001$ and epochs of 5 for time efficiency with a relatively good convergence.

2.6. Batch Size

We are going to investigate the influence of batch size on the final descriptor performance. It is known that small batch sizes are beneficial to faster convergence and better generalization, while large batches allow better GPU utilization[9]. The triplet loss function should benefit from seeing more hard negative patches to learn to distinguish them from true positive patches. We had recorded results for batch sizes of 50, 128, 256, 512, 1024 as shown in Table 1:

batch_size	val_loss	Verification	Matching	Retrieval	Mean
50	0.09520	0.898772	0.400098	0.702471	0.667114
128	0.09539	0.901719	0.411886	0.712482	0.675362
256	0.09012	0.905304	0.421604	0.719248	0.682052
512	0.08622	0.907225	0.431845	0.728063	0.689044
1024	0.10054	0.900345	0.410986	0.705504	0.672278

Table 1. Descriptor performance by varying batch sizes, models are trained using $adam$ optimizer with lr=0.001 through 5 epochs

As we expected, model performance improves with increasing batch size up til 512 and a degradation at 1024 due to slow convergence.

3. Denoise Training Model Evaluation

3.1. Baseline Approach

The baseline for the denoising model utilized a shallow UNet architecture consist of a encoder layer which encodes the image and reduce it to a vector of smaller size, forcing it to capture important features of the image by use of a pooling layer. Then the image features are decoded by upsampling convolutional layer. The denoise model uses a subset of the training sequences for training and a subset of the validation sequences for validation purpose. The baseline model took a random sample of 3 training sequences and 1 validation sequence, each with a batch size of 50. It invokes a SGD optimizer with a learning rate of 0.00001, momentum of 0.9 and applying nesterov accelerated gradient which prevent our gradient from steep descent and results in increased responsiveness[10]. The result is shown below:

Verification	Matching	Retrieval	Mean
0.475657	0.037140	0.215969	0.242922

The score of the model is poor compare to the ones above. This is because the denoise model was trained only on a very small subset of the training data, thus the training is pointless as the model is not a generic representation of the vast data set. Moreover, training over just 1 epoch simply does not allow the learning curve to converge.

3.2. Hyper-parameter Tunning

3.2.1 Increase Epochs and Sample Size

As discussed earlier, the first step is to increase the epochs to allow convergence for the model and generate a larger subset of the training data for a good overall representation of the training set. We chose to run the network for 5 epochs with 40 training sequences and 10 validation sequences. The result is shown below:

Verification	Matching	Retrieval	Mean
0.795588	0.164924	0.454415	0.471666

The findings above support our argument in which the baseline may be improved. The score for the overall performance shows a promising advancement over the baseline model.

3.2.2 Optimizer Evaluation

Here we investigated mainly two optimizers, a generative adversarial learning method SGD and a adaptive learning method $Adam$. The results are shown below:

SGD optimizer

val_mse	Verification	Matching	Retrieval	Mean
5.8363	0.795588	0.164924	0.454415	0.471666

Adam optimizer

val_mse	Verification	Matching	Retrieval	Mean
5.1156	0.808443	0.179724	0.467872	0.485346

From the results, we can denote that $Adam$ presents a better performance in our case.

3.3. Full UNet

The current architecture for the denoise model is a Shallow-UNet which consists of an encoder layout with one $MaxPooling$ layer sandwiched by two 3x3 $Convolution$ layer each followed by a rectified linear unit (ReLU). Then a decoder layout consist 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 one 3x3 $Convolution$ layers, followed by a ReLU. Finally a single filter $Convolution$ layer at the end[4], which made up 6 convolution layers in total. An illustration of the Shallow-UNet is shown below in Figure 9. Although the shallow network offer shorter runtime during training, it provides less feature channels in each layer and less convolution networks. Therefore key context information may not be extracted and result in poor reconstruction of the key features for the denoised image. Therefore we attempted to use Full UNet (network diagram shown in Figure 10) to train the denoise model and the result is shown below.

Verification	Matching	Retrieval	Mean
0.797833	0.173226	0.458381	0.476480

Above summary has shown only a slight improvement over the Shallow-UNet. This is because the purpose of the Full UNet was for image segmentation where large size image is provided. Since our data patch size is very small, thus invoking Full UNet does not provide any advantage. The feature extracted at the deep layers (the feature space is only 2x2 at the bottom layer) are possibly being overfitted as they should not be considered as a good representation of the overall structure. This finding can also be observed from the validation mean absolute error of the denoise model shown below.

Shallow val_mse	Full val_mse
5.4859	5.9801

4. Performance Comparison

The final model invoke a Shallow-UNet as the denoise model CNN along with $adam$ optimizer using learning rate of 0.001. In order to achieve better denoising performance, the model was trained using all training images through 4 epochs. For descriptor training, we utilized In-Triplet Hard Negative Mining as loss function with HardNet CNN architecture using $adam$ optimizer. We employed 500000 triplet loss pairs for training and 50000 triplet loss pairs for validation with batch size of 512. The result are shown in Table 2:

	Baseline Approach	Optimized Approach
Verification	0.714930	0.901205
Matching	0.104292	0.391247
Retrieval	0.358950	0.699873
Mean	0.392724	0.664108

Table 2. Performance evaluation comparing baseline approach and optimized approach

5. Conclusion

This report discussed a pipeline method using a shallow UNet to train a denoise model and feed to local image descriptor that relies on the $hardnegativemining$ within a batch that maximizes the distance between close positives and close negatives. Both training model were found to perform well using $adam$ optimizer. Moreover, a relatively large batch size of 512 can improve the descriptor training and greater number of triplet loss pairs provide a better representative of the sample thus improve performance.

References

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

[2] E. Hoffer and N. Ailon. Deep metric learning using triplet network. Dec 20, 2014.

[3] K. Mikolajczyk and C. Carlo. Ee3-25: Deep learning lecture 2. page 8, 2019.

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

[5] F. Schroff, D. Kalenichenko, and J. Philbin. Facenet: A unified embedding for face recognition and clustering. pages 815–823. IEEE, Jun 2015.

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

[7] D. P. Vassileios Balntas, Edgar Riba and K. Mikolajczyk. Learning local feature descriptors with triplets and shallow convolutional neural networks. In E. R. H. Richard C. Wilson and W. A. P. Smith, editors, *Proceedings of the British Machine Vision Conference (BMVC)*, pages 119.1–119.11. BMVA Press, September 2016.

[8] J. Wang, Y. Song, T. Leung, C. Rosenberg, J. Wang, J. Philbin, B. Chen, and Y. Wu. Learning fine-grained image similarity with deep ranking. pages 1386–1393. IEEE, Jun 2014.

[9] D. R. Wilson and T. R. Martinez. The general inefficiency of batch training for gradient descent learning. *Neural Networks*, 16(10):1429–1451, 2003.

[10] N. Yurii. A method for unconstrained convex minimization problem with the rate of convergence $o(1/k^2). Doklady ANUSSR, (269) : 543 - -547, 1983.$

Appendices

The code is being pushed to remote repository at https://github.com/tommyzbear/Deep-Learning. Optimization are done in Deep Learning.ipynb. Thank you.

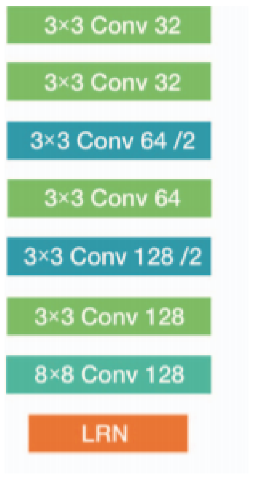


Figure 3. L2-Network illustration

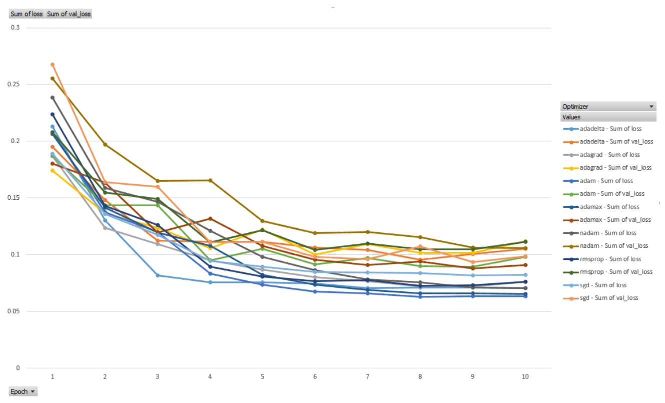


Figure 4. Descriptor training model loss variation with increasing epoch using baseline sgd with static learning rate of 0.1

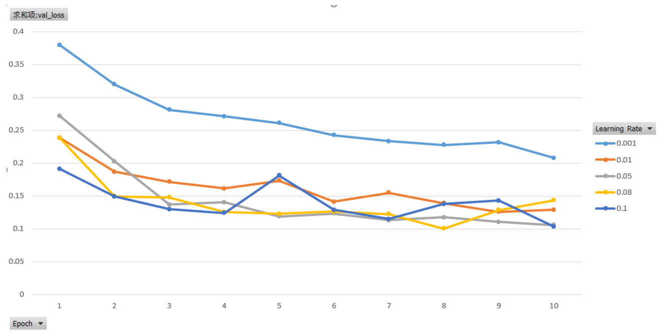
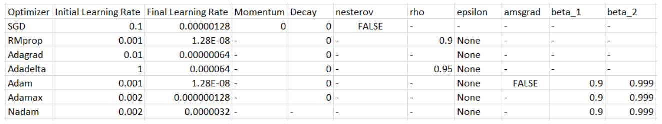


Figure 5. Validation loss for descriptor training model with static learning rate



Optimizer	Initial Learning Rate	Final Learning Rate	Momentum	Decay	nesterov	rho	epsilon	amsgrad	beta_1	beta_2
SGD	0.1	0.00000128	0	0	FALSE	-	-	-	-	-
RMprop	0.001	1.28E-08	-	0	-	0.9	None	-	-	-
Adagrad	0.01	0.00000064	-	0	-	-	None	-	-	-
Adadelta	1	0.000064	-	0	-	0.95	None	-	-	-
Adam	0.001	1.28E-08	-	0	-	-	None	FALSE	0.9	0.999
Adamax	0.002	0.000000128	-	0	-	-	None	-	0.9	0.999
Nadam	0.002	0.0000032	-	-	-	-	None	-	0.9	0.999

Figure 6. Hyper-parameters used for descriptor training model

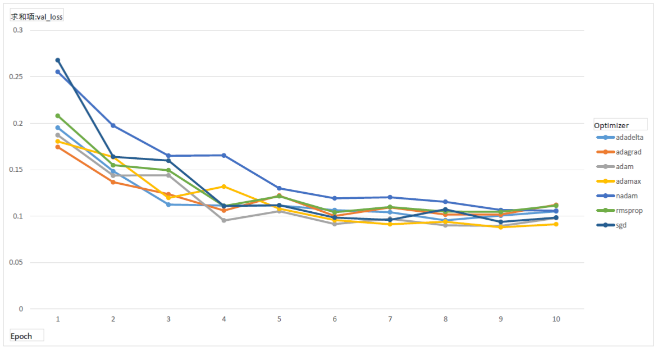


Figure 7. Validation loss for different optimizers performance on descriptor training using clean patches

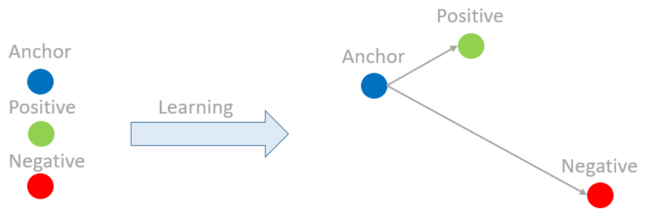


Figure 8. The triplet loss attempts to minimize the distance between *anchor* and *positive* which both of them has the same physical view. Also maximizes the distance between *anchor* and *negative* for which they have different identities[5]

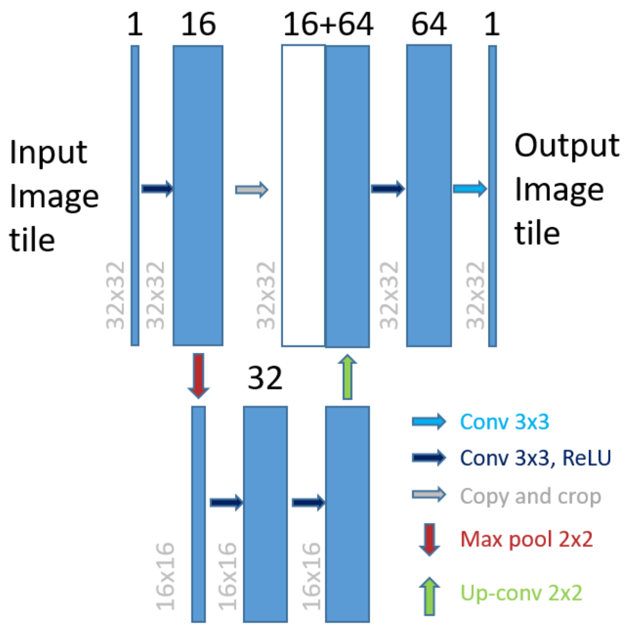


Figure 9. Shallow-UNet architecture invoked in baseline denoise model. Each blue box corresponds to a multi-filter feature map. The number of filters is denoted on the top of the box. The x-y sizes is provided at the bottom left corner of the box. The white box represents copied feature map. Arrows denote different operations.

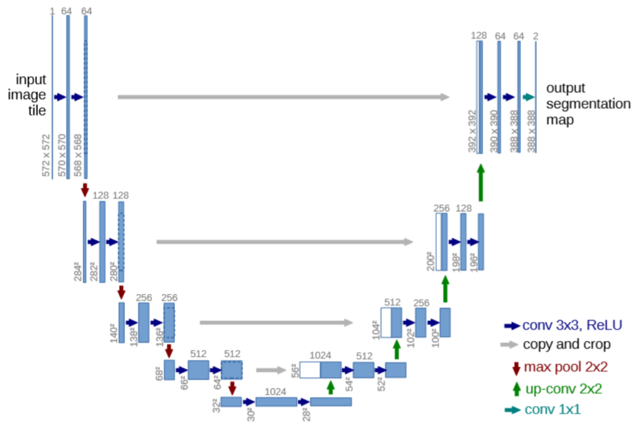


Figure 10. Full-UNet architecture. Each blue box corresponds to a multi-filter feature map. The number of filters is denoted on the top of the box. The x-y sizes is provided at the bottom left corner of the box. The white box represents copied feature map. Arrows denote different operations[4].