Mini Project 1

Giovanni La Cagnina
École Polytechnique
Fédérale de Lausanne
Lausanne, Switzerland
Email: giovanni.lacagnina@epfl.ch
SCIPER: 342352

Pau Autrand Caballero
École Polytechnique
Fédérale de Lausanne
Lausanne, Switzerland
Email: pau.autrandicaballero@epfl.ch
SCIPER: 343812

Joshua Voelkel
École Polytechnique
Fédérale de Lausanne
Lausanne, Switzerland
Email: joshua.voelkel@epfl.ch
SCIPER: 343212

I. Introduction

The approach used in this Mini Project is the following: firstly we have reconstruct the network architecture used in the Noise2Noise paper [4] in order to use it as a benchmark and as a starting for additional improvements. Parallely we try to implement two additional networks: one shallow network based on the Unet structure (citation) and one deep network composed by Unet block, inspired by the Noise2Noise architecture, but gradually increasing the number of channel in the encoder.

II. Noise2Noise

We used as benchmark the network architecture used in the Noise2Noise paper, except for the fact that we have substituted the last activation function, that was a LeakyReLU function with a simple ReLU activation function, in order to guarantee positive output [4]. The loss function used to train the network was the MSE. This network structure increases the amount of channels sharply in the beginning from 3 to 48 and increases them to 96 roughly in the middle of the network. At the end, the amount of channels decrease to 64, then 32 and then 3. Using this benchmark model without any adjustments, we achieved a 25.47 dB PSNR by using the following hyperparameters.

Weight Initialization	Kaiming normal
Optimizer	ADAM
Learning Rate	0.001
Mini Batch Size	16
Epochs	10
Loss function	MSE

Note that we did not use batch normalization or any other regularization techniques.

A. Benchmark improvements

To further improve the performance of the benchmark model, we tried various adaptions. First, we wanted to see if batch normalization has impact on the performance of the model. Therefore, we applied batch normalization for 4 different Noise2Noise with differen channels (16, 32, 64, 128) and achieved a very slight increase in PSNR to 25.49 dB for the 32-batch model. In the next step, we wanted to check whether using Leaky ReLU instead of the standard ReLU makes a difference. For the aforementioned four different batch sizes, we therefore tested the performance of the models using Leaky ReLU (except in the last block) and observed an increase of the PSNR value for both the 16-batch and 32-batch model to 25.52 dB each. To see if batch normalization works well with Leaky ReLU we tested the performance of the four models using both batch normalization and Leaky ReLU but none of the models had an increased PSNR value. We therefore decided to not keep batch normalizations in the upcoming models. In the following step, we used the two best performing models with Leaky ReLU (batch size of 16 and 32) and tried different learning rates to test for a possible improvement. However, both models did not increase their performance using learning rates $1e{-}4$ and $1e{-}2$ compared to the previous learning rate of $1e{-}3$. In the following, we thought that an increase in epochs could enhance the performance of the models and thus tested both models with 20 epochs instead of initially 10. But also here, the performance did not increase measured by the PSNR value. Next, we tried to improve the performance by changing the weight initialization. So far, we used Kaiming normal initialization as proposed in the Noise2Noise paper. Using the two best performing models, we initialized the weights using Xavier normal, Xavier uniform and orthogonal. We could see clear improvements in both models using Xavier normal and uniform while the Xavier normal initialization had the strongest positive impact. We achieved for the 16-batch-size and the 32-batch-size models a 25.57dB and 25.54dB respectively. We decided then to focus on improving the best model (16-batch-size) and did so by trying different optimizers. In particular, we tried standard SGD, NADAM [1], RADAM [5] and ADAMAX [3]. While standard SGD performed very poorly, which was to be expected as it is the most simplest optimizer out of these, NADAM performed the best with an increase in PSNR values to 25.59dB. Both ADAMAX and RADAM did not increase the PSNR. In the last step, we applied two different data augmentation strategies. In the first one, we doubled the size of the train data set by adding vertically flipped images. Using ADAM and NADAM, the best perfoming models managed to increase the PSNR value to 25.60dB each. We also tried data augmentation using horizontally flipped images, which also increased the PSNR slightly to 25.58dB but not as significantly as by using the vertically flipped one. This result could be explained by the

fact that many objects in the images are horizontally centered but many are not vertically centered, such that a horizontal flipping does add as much new training potential as the vertical flipping. Eventually, we were able to increase the base case performance from 25.47dB to 25.60dB by using Leaky ReLU instead of standard ReLU, changing the weight initialization to Xavier normal, the optimizer to NADAM and applying data augmentation by adding vertically flipped images. The hyperparameters can be summarized as follows.

Weight Initialization	Xavier normal
Optimizer	NADAM
Learning Rate	0.001
Mini Batch Size	16
Epochs	10
Loss function	MSE

III. Short model

The first architecture created is a U-Net model inspired by [6]. The architecture of the model is illustrated in (III-A). The encoder part is structured as a composition of two convolutional blocks made by two 3×3 convolutional layers, with stride and padding equal to 1, where each convolutional layer is followed by a LeakyReLU activation function, and a 2×2 Maxpool operator. The decoder part is composed by a two of the following blocks: 2×2 Upsample layer followed by a 3×3 Convolutional layer, concatenation of the output from the encoder part, and then two 3×3 layers. The first difference that is possible to note with respect to the benchmark model is that we double up the number of channels at each encoder convolutional block. This network architecture permits to have a shallower model that reaches a larger number of channels (196) in the middle of the network. The first model uses a learning rate equal to $1e{-}3$, the optimizer used is ADAM [3], and batch size equal to 16 and we get a PSNR of 25.51dB. The loss function used to train the network was the MSE. We summarize in the following table the hyperparameters used to train the short model.

Weight Initialization	Xavier normal
Optimizer	ADAM
Learning Rate	0.001
Mini Batch Size	16
Epochs	10
Loss function	MSE

A. Short model improvements

The first step we took in an attempt to improve the Short model was to change the mini batch size using 32 instead of 16. Using this mini batch size we obtained an improvement in performance on the validation set by obtaining a PSNR of 25.53dB. We have also tried to change the increase in channels done in the first step in the model, lowering it from 48 to 32, but this resulted in a worse performance getting a PSNR of 24.97dB. Regarding the optimizer choice the final choice is to use ADAM optimizer because it has performed the best during our tests. In fact the SGD optimizer, using the same number of epochs equal to 10, leads to a poor performance both in terms of time and also in terms of PSNR. We also observed that changing the weight initialization using the Kaiming [2] normal method performs worse than Xavier normal for all of the models. On top of that, we tried to use the NADAM [1] but we found negative results in the convergence of the model.

TABLE I
Short Model

NAME	N_{out}	FUNCTION
INPUT	3	
E–CONV0	48	Convolution 3×3
E-CONV1	48	Convolution 3×3
POOL0	48	Maxpool 2×2
E-CONV2	96	Convolution 3×3
E-CONV3	96	Convolution 3×3
POOL2	96	Maxpool 2×2
E-CONV4	192	Convolution 3×3
E-CONV5	192	Convolution 3×3
UPSAMPLE0	192	Upsample 2×2
D-CONV0	96	Convolution 3×3
CONCAT0	96	Concat output of E-CONV3
D-CONV1	96	Convolution 3×3
D-CONV2	96	Convolution 3×3
UPSAMPLE1	96	Upsample 2×2
D-CONV3	48	Convolution 3×3
CONCAT1	48	Concat output of E-CONV1
D-CONV4	48	Convolution 3×3
D-CONV5	48	Convolution 3×3
D-CONV6	3	Convolution 1×1

The biggest improvements were found using data augmentation: as described in *Benchmark improvements* we have doubled the samples in our training dataset using vertically flipped images. Using this data augmentation, we were able to achieve a PSNR of 25.57dB which is very close to the improved benchmark model performance. We have also tried to use horizontally flipped images and we obtained the best performance of 25.58dB on the validation set. We have, in addition, tried to apply Batch Normalization to the model but it did not lead to improvements on the performance on the validation set and therefore we did not introduce it in our final model.

IV. Long Model

To see if increasing the depth of the network enhances the performance, we created a model using the same blocks as in the short model but using more of them. As in the short model, we double up the number of channels at each encoder convolution block. Starting with an initial amount of channels of 32, we go up to 512 channels and then gradually decrease the amount of channels. The model architecture can be seen in table (IV-A).

A. Long model improvements

We tried various hyperparameters to increase the performance of the model. That is, we tried different learning rates, epochs, weight initialization and optimizers. The loss function used to train the network was the MSE. Up to this point, the

TABLE II
LONG MODEL

NAME	N_{out}	FUNCTION
INPUT	3	
E–CONV0	32	Convolution 3×3
E-CONV1	32	Convolution 3×3
POOL0	32	Maxpool 2×2
E-CONV2	64	Convolution 3×3
E-CONV3	64	Convolution 3×3
POOL2	64	Maxpool 2×2
E-CONV4	128	Convolution 3×3
E-CONV5	128	Convolution 3×3
POOL3	128	Maxpool 2×2
E-CONV6	256	Convolution 3×3
E-CONV7	256	Convolution 3×3
POOL4	256	Maxpool 2×2
E-CONV8	512	Convolution 3×3
E-CONV9	512	Convolution 3×3
UPSAMPLE0	512	Upsample 2×2
D-CONV0	256	Convolution 3×3
CONCAT0	256	Concat output of E-CONV7
D-CONV1	256	Convolution 3×3
D-CONV2	256	Convolution 3×3
UPSAMPLE1	128	Upsample 2×2
D-CONV3	128	Convolution 3×3
CONCAT1	128	Concat output of E-CONV5
D-CONV4	128	Convolution 3×3
D-CONV5	128	Convolution 3×3
UPSAMPLE2	64	Upsample 2×2
D-CONV6	64	Convolution 3×3
CONCAT2	64	Concat output of E-CONV3
D-CONV7	64	Convolution 3×3
D-CONV8	64	Convolution 3×3
UPSAMPLE3	32	Upsample 2×2
D-CONV9	32	Convolution 3×3
CONCAT3	32	Concat output of E-CONV1
D-CONV10	32	Convolution 3×3
D-CONV11	32	Convolution 3×3
D-CONV12	3	Convolution 3×3

best performing model achieved 25.29dB PSNR with a batch size of 16. Using vertical data augmentation as described in the previous chapters, we managed to increase PSNR of this model to 25.41dB. Furthermore, we could observe that all models performed better with Xavier initalization compared to the Kaiming [2] normal initialization which was used in the Noise2Noise standard model. The hyperparameters of this best performing model can be summarized as follows.

V. COMPARISON BETWEEN THE MODELS

Comparing each of the PSNR scores of the best of the three different models, we observe that the short model's performance on the validation set using data augmentation are very similar to the performance of the benchmark built based on the Noise2Noise paper [4]. We can note that the short model is shallower respect to the benchmark model, in the sense that it has less layers, also another difference between the two models comes from the fact that in the short model the number of channels is increased directly in the encoder part of the architecture: indeed the short model passes from 3 to 48 channels, as the benchmark, but then the second convolutional block changes the number of channels to 96, while the benchmark model keeps it fixed to 48. The good performance of a shallower model could be explained by the fact that the images that we aim to denoise are quite small, 32×32 pixels, while in the Noise2Noise paper they have trained their model using images having 256×256 pixel, and therefore a shallower structure in terms of number of layer can be sufficient to get a similar performance on the validation set. Also compared to long model, the short model is clearly preferable as it is performing better under the aspects of performance and training time. Given the limited available computational power at our disposal we could not improve our results trying further techniques such as data augmentation, which performed well in our research. This leaves space for improvement.

REFERENCES

[1] Timothy Dozat. Incorporating nesterov momentum into adam. 2016.

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1026–1034, 2015.

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

[4] Jaakko Lehtinen, Jacob Munkberg, Jon Hasselgren, Samuli Laine, Tero Karras, Miika Aittala, and Timo Aila. Noise2noise: Learning image restoration without clean data. *arXiv preprint arXiv:1803.04189*, 2018.

[5] Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the variance of the adaptive learning rate and beyond. *arXiv preprint arXiv:1908.03265*, 2019.

[6] Olaf Ronneberger and Philipp Fischer and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015.