Deep Learning Mini-project 1: Noise2Noise model

Stephan Weber, Laurent Gürtler, Taha Zakariya
EE-559 Deep Learning, EPFL

***Abstract*– The goal of this project is to to implement a Noise2Noise model. A Noise2Noise model is an image denoising network trained without clean reference images, that learns to capture only the relevant information of the noisy image, the denoised image. The state-of-the-art for this kind of problem is resolved by a UNet architecture. To perform this task, we train and optimize a model from noisy images. In this report we present our implementation consisting of a UNet with residual connections. We explain our approach and the different strategies that we used such as data augmentation and hyperparameter tuning of the model. In the end, we achieve a PSNR score of 24.6.**

I. Introduction

Given a dataset containing 50000 noisy images from different objects and shapes, we train a ResUNet to achieve the denoising of the images without reference of denoised images.

To test the performance of our model, we have also a testing set containing fewer images.

In section II, we will present the ResUNet under a theoretical point of view to convince you why this model is the state of the art for denoising model. In section III, we will talk about the different techniques that were used to increase the performance of our model as well as the techniques that were promising but didn't have a significant impact on the model. In section IV, we will discuss the results we obtained and how we could have increase the accuracy of our predictions.

II. Model

The backbone of the model used in this project comes from the paper *Noise2Noise: Learning Image Restoration without Clean Data*.

A. ResUNet

ResUNet uses a UNet encoder/decoder backbone, in combinaison with residual connections. This improvement over classic UNet takes the advantage of having a UNet architecture and the Deep Residual Learning.

In section IV, we show how ResUNet achieves the denoising of the different images.

Architecture of ResUNet.

ResUNet is inspired by the fully convolutional and is a better version of UNet. Its U-shaped architecture is what gives it this name. The left branch of the "U" is called contracting path or encoder and its major role is to capture context of objects in the image. It has the typical architecture of a convolutional network consisted of the repeated application of two 3x3 convolutions, each followed by a rectified linear unit (ReLU) and a 2x2 max pooling operation with stride 2 for downsampling which doubles the number of feature channels at each step.

After 5 blocks, the encoder yields what we call a bottom layer. This bottom layer is the input to the second part of the network which is called expanding path or decoder (right branch of the "U"). Every step in the expansive path consists of 2 convolutions, each followed by a ReLU function, and an upsampling of the feature map by a transpose convolution. Then, a concatenation with the correspondingly feature map from the contracting path is performed. The role of this second branch is to enable precise localization as well as allowing the propagation of context information by using a large number of feature channels.

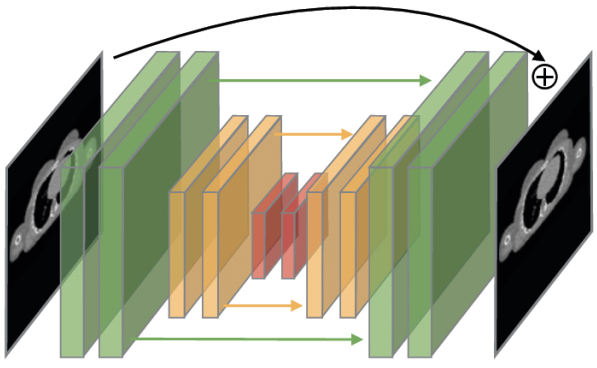

Fig. 1: Example of ResUNet.

Our implementation. Table I summarizes our implementation of ResUNet and shows the parameters chosen for the encoding and decoding blocks. (3x3) are the sizes of filters of the corresponding convolutional layer, and ReLU (Rectified Linear Unit) is the activation function used. It is also judicious to mention that we used same-padding to extrapolate the missing pixels in the border region of the image in each convolution.

TABLE I: Configuration of each layer of our implementation of ResUNet.

Layer	Configuration	Output shape
Input	-	n
Enc0	Conv2D(3x3) + ReLU	48
Enc1	Conv2D(3x3) + ReLU	48
Pool1	MaxPooling2D(2x2)	48
Enc2	Conv2D(3x3) + ReLU	48
Pool2	MaxPooling2D(2x2)	48
Enc3	Conv2D(3x3) + ReLU	48
Pool3	MaxPooling2D(2x2)	48
Enc4	Conv2D(3x3) + ReLU	48
Pool4	MaxPooling2D(2x2)	48
Enc5	Conv2D(3x3) + ReLU	48
Pool5	MaxPooling2D(2x2)	48
Enc6	Conv2D(3x3) + ReLU	48
UpSample5	Conv2DTranspose(2x2) + ReLU	48
Concat5	Concat Pool 4	96
Dec5_A	Conv2D(3x3) + ReLU	96
Dec5_B	Conv2D(3x3) + ReLU	96
UpSample4	Conv2DTranspose(2x2) + ReLU	96
Concat4	Concat Pool 4	144
Dec4_A	Conv2D(3x3) + ReLU	96
Dec4_B	Conv2D(3x3) + ReLU	96
UpSample3	Conv2DTranspose(2x2) + ReLU	96
Concat3	Concat Pool 4	144
Dec3_A	Conv2D(3x3) + ReLU	96
Dec3_B	Conv2D(3x3) + ReLU	96
UpSample2	Conv2DTranspose(2x2) + ReLU	96
Concat2	Concat Pool 4	144
Dec2_A	Conv2D(3x3) + ReLU	96
Dec2_B	Conv2D(3x3) + ReLU	96
UpSample1	Conv2DTranspose(2x2) + ReLU	96
Concat1	Concat Pool 4	96 + n
Dec1_A	Conv2D(3x3) + ReLU	64
Dec1_B	Conv2D(3x3) + ReLU	32
FinalConv	Conv2D(1x1) + ReLU	m

Metrics. For the performance metric, our model uses the mean squared error loss, which is the most commonly used loss function for regression.

$$L(y, \hat{y}) = \frac{1}{N}\sum_{i=0}^{N}(y - \hat{y}_i)^2$$

where y is the set of ground truth and $\hat{y}$ the set of predictions.

We use Adam as the optimizer of our model which is by definition a stochastic gradient descent method based on adaptive estimates of lower-order moments.

III. Optimization

In this section, we outline the different techniques we used to improve the performance of our model.

Data augmentation. The provided dataset counts 50000 training images which is a good amount of data to learn a model. To increase the score of the model, we decided to perform data augmentation to increase the generalization of the model. We do that by applying different transformations on the original images. Since the scale of the pixels need to stay the same through the transformations, we decided to only use rotations as transformations. The final dataset used for training contains the original images and a flip on 90° of every images.

Normalization The normalization of the data is done once for the input of the model to re-scale the values of the pixels. This helps to stabilise the

Learning rate. By using cross-validation, the learning rate is hyper-tuned and the best parameter is a learning rate of 0.0008 as we can see in the figure 2.

Batch size. This method was applied after every convolution to re-scale the values of the pixels. It is commonly used to make neural networks faster and more stable and also to prevent calculations from exploding because of very large numbers. A mini-batch size of 8 is the optimal value for this parameter as we can see on the figure 3.

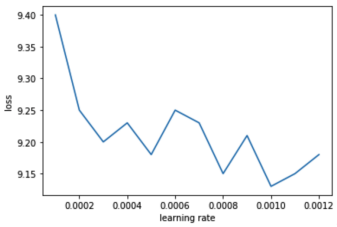

Fig. 2: Learning Rate tuning

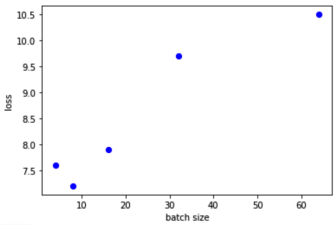

Fig. 3: mini-batches tuning

IV. Results and Discussion

Table II shows the best result that we managed to obtain with the ResUNet model. The metric used for the score is called the Peak signal-to-noise ratio (PSNR). This metric is broadly used as an image quality metric, especially to compare original vs compressed images.

The PSNR (in dB) is defined as:

$$PSNR = 10 \cdot \log_{10} \left(\frac{MAX_I^2}{MSE} \right) \quad (1)$$

$$= 20 \cdot \log_{10} \left(\frac{MAX_I}{\sqrt{MSE}} \right) \quad (2)$$

$$= 20 \cdot \log_{10}(MAX_I) - 10 \cdot \log_{10}(MSE). \quad (3)$$

TABLE II: Performance of the model.

Model	PNSR score
ResUNet	24.6

Figures 4 and 5 show the prediction of a noisy image denoised by our model and the result we should obtain, respectively. The prediction gives a smoother image due to the averaging of the pixels but it looks really similar to the image we are supposed to obtain.

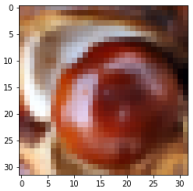

Fig. 4: Prediction

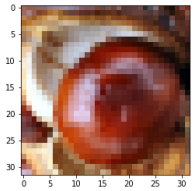

Fig. 5: Real image

Here are some ideas to reach a better score:

Hyper-parameter tuning. The main idea to perform better is the hyper-parameter tuning, there are many different parameters such as the learning rate, the batch-size, as well as the parameter of the model itself (kernel size, padding, number of layer, ...). Hyper-parameter tuning is always a good idea but it gets rapidly limited by the computation time with nowadays large datasets.

Pre-processing and data augmentation. Especially when working with images, the pre-processing might be a solution to get more accurate results. It is always possible to play around with the colors of the images, the intensity of the pixels or any transformations. This can lead to data augmentation to enable better generalization of unseen images during training.

V. Conclusion

Noise2Noise models are typical machine learning tools which have many useful applications such as in the medical domain or in biology. Typically, images taken directly from the microscope have natural noise, which can perturb the analysis of the images. We already know from literature that some models are extremely precise, making denoising a resolved problem. The knowledge acquired during this project helped us to understand one of the applications of deep learning models on images.

For further work, we thought about some solutions to improve our model. The most obvious one is to tune our model to specialize it for denoising by implementing a different architecture that is more adapted to the present problem and which could catch more relevant information of the image. A second option would be to increase even more the number of images for training in the dataset by finding other sources. To finalize the model, we have thought about applying post-processing operations on the predictions but the results that we obtained was already satisfying.

References

[1] Lehtinen, J. et al. (2018) 'Noise2Noise: Learning Image Restoration without Clean Data' Available at: http://arxiv.org/abs/1803.04189