Mini-project 1
Noise2Noise PyTorch Implementation

Marie-Alix Gillyboeuf 299327, Rhita Mamou 286507, Assia Ouanaya 288251
Deep Learning - EE-559, EPFL, Switzerland

Abstract

In this project, we aim to implement a Noise2Noise model using the standard PyTorch framework (Lehtinen et al., 2018). The objective of a Noise2Noise model is to clean images by using only corrupted images. We also give in this report an overview of the Deep Learning algorithms implemented and compute their efficiencies. Finally, we were able to show that it is possible to implement a model that learns to restore clean images by only looking at corrupted ones, without priors knowledge of the corruption model.

1 Introduction

In the past few years, Deep Learning has become an increasingly attractive tool for clean signal reconstruction from a downsampled signal. Deep Neural Networks have outperformed multiple times the traditional image processing methods, e.g. (Ronneberger et al., 2015) and (Krizhevsky et al., 2012). Indeed, traditional techniques require manual parameter selection as well as optimization. In addition, Convolutional Neural Networks (CNNs) have shown to be more computationally efficient. Furthermore, with the explosion of available data, CNNs have gained more traction. U-networks, in particular, have been successfully used for biomedical imaging segmentation instead of conventional computer vision approaches.

In this project based on the work of Lehtinen et al., we aim to build a network that learns how to map corrupted observations to clean signals. The dataset represents pixelated images that the network needs to denoise.

The report is organized as follows. The first section describes the data of interest. The second section reviews the core concepts of deep learning used in this work. The key contributions of the present work are introduced in the third section, in which we described the construction of the PyTorch Noise2Noise model. Then, we present the result obtained with the aforementioned methods in the fourth section. Finally, in the last section, we present a summary and a discussion of our work.

2 Data

The training data set consists of 50'000 noisy pairs of colored images with pixel values between 0 and 255. The inputs to the studied networks are of dimension $(3 \times 32 \times 32)$, representing three grayscale images of dimension (32×32) each. The pair of the images represent the same downsampled image; however, with a different noise added. The trained network aims to learn from this data to denoise unseen images.

An additional 1'000 images validation dataset has been provided to test the model implemented. It has the same $(3 \times 32 \times 32)$ size, with different noise characteristics.

Before training the data, a normalization step was performed such that all pixel values are between $[0, 1]$. We simply divide the tensor data sets by 255. Normalizing data sets is a usual good practice as Neural Networks process inputs using small weight values, and inputs with large integer values can disrupt or slow down the learning process. All of our data and models are run on GPUs.

3 Theoretical Background

Our project aims to implement the PyTorch version of a simple image denoiser. Let $\hat{x}_i$ be our input pixelated image, let y_i be a denoised clean image, let f_θ be a mapping function and L a loss function. By training our neural network we aim to minimize the following empirical error:

$$\underset{\theta}{argmin} \sum_i L(f_\theta(\hat{x}_i), y_i)$$

We use various Pytorch modules to minimize the error. First, we apply `Conv2d`, a simple mathematical operation in which we slide the kernel of weights over our 2D images and perform element-wise multiplication with the data that falls under the kernel. Then, we sum up the multiplication result to produce one final output of that operation.

Regarding the activation functions, we used the `LeakyReLU` activation which is a variation of the `ReLU` activation. `ReLU` activation function is a simple function that sets all negative values to zero. However, this function has a problem called *Dying Neurons*: whenever the inputs are negative, the node derivative becomes zero, which implies that the backpropagation cannot be performed on the node. Thus, its learning stops and the node "dies". `LeakyReLU` solves this issue by adding a small slope for negative values to ensure a non-zero gradient.

The learning of a neuron does not stop during back-propagation, avoiding the dying neuron issue.

To down-sample our image inputs, we used `MaxPooling`: it reduces the dimensionality of images by selecting the maximum of an area of the input image. This area is specified by the kernel size.

As an optimizer, we decided to use `ADAM` optimizer which updates network weights iteratively based on training data. It computes individual learning rates γ for different parameters (*i.e.* weights and bias).

To optimize our model, we also used `Batch Normalization`, which is a normalization technique that can be applied at the layer level. Batch normalization fixes the means and variances of layer inputs. It normalizes the inputs of each layer to a learnt representation likely close to $\mu = 0.0$, $\sigma = 1.0$. Thus, all the layer inputs are regularized, and significant outliers are less likely to impact the training process negatively.

4 Architecture

Our first approach was to simply implement an encoder and a decoder with 7 hidden layers for the encoder part and 11 hidden layers for the decoder part. The encoder part consisted of an alternation of convolution followed by `LeakyReLU` activation function with a slow learning rate ($\gamma = 0.1$). For training, we used mini batch where each mini-batch produces estimates of the mean and variance of each activation. The average results for mini-batch of size 100, with 1000 train pairs, 1000 test pairs and 25 epochs were poor as we reached a PSNR of only 12.11 dB.

Then we decided to implement a Unet. Unets architecture consists of a contracting path and an expansive path (the network has a "U" shape that is why it is called a Unet).

The contracting path follows the typical architecture of a convolutional neural network (CNN). It consists of the repeated application of two 3x3 convolutions, each followed by batch normalization, a leaky rectified linear unit (ReLU) activation function and then a 2x2 max pooling operation with a stride of 2 for downsampling. At each downsampling step, we double the number of feature channels. Every step in the expansive path consists of an upsampling of the feature map followed by a 2x2 convolution that halves the number of feature channels, a concatenation with the correspondingly cropped feature map (max-pooling results) from the contracting path, and two 3x3 convolutions, each followed by a LeakyReLU. As a final layer, we used a 1x1 convolution followed by a ReLU activation function.

We used batch normalization for many reasons. Indeed, as we already mentioned in Section 3, it fixes the means and variances of input layers, reduces internal covariates shift, and accelerates the training rate. The above architecture was inspired by the Noise2Noise paper (Lehtinen et al., 2018). The training was done using an `ADAM` optimizer with different learning rates ($\gamma \in [0, 1]$). You can find our architecture in Table 1 for better understanding.

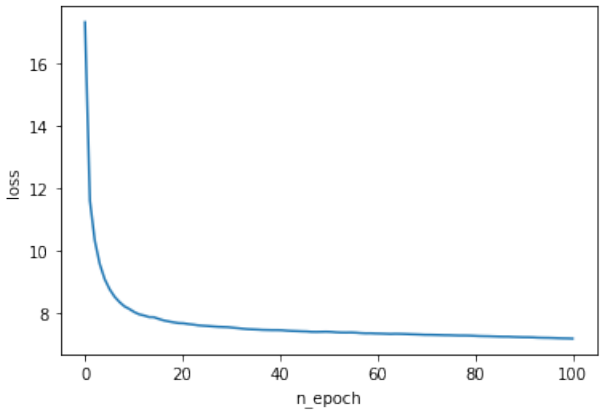

Figure 1: MSE loss as a function of the number of epochs.

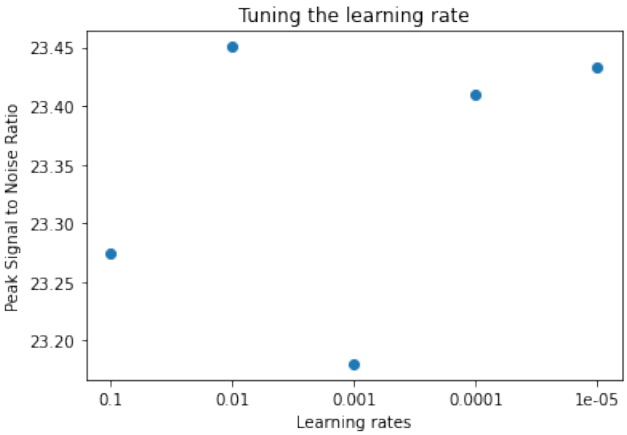

Figure 2: PSNR as a function of the learning rate of the ADAM optimizer.

5 Results

In order to obtain a significant result, the networks have been trained and tested with all of the datasets with a batch size of 100 and 100 epochs. We then calculate the MSE and PSNR.

Figure 1 shows the variation of the MSE versus the number of epochs (n_{epoch}) for training and test data. The gradual decrease of the values of MSE proves that the training and test data are completely accurate. The gradual decrease is because the weights improves after each epoch.

To figure out the best learning rate, we first ran a grid search with multiple learning rates. The obtained results are shown within Figure 2, we see that the best learning rate is $\gamma = 0.01$ which maximises the PSNR metric.

In Figure 3, we see that our denoiser model works well. Indeed, the predicted images denoise the input image and approximate quite well the clean image.

Layer	C_{out}	Function
input	3	
c0	48	Convolution 3x3
c1	48	Convolution 3x3
x1	48	Maxpool 2x2
c2	48	Convolution 3x3
x2	48	Maxpool 2x2
c3	48	Convolution 3x3
x3	48	Maxpool 2x2
c4	48	Convolution 3x3
x4	48	Maxpool 2x2
c5	48	Convolution 3x3
x5	48	Maxpool 2x2
c6	48	Convolution 3x3
up	96	Upampling 2x2
ct5	144	Transpose convolution
dec5a	96	Convolution 3x3
dec5b	96	Convolution 3x3
up	144	Upampling 2x2
ct4	144	Transpose convolution
dec4a	96	Convolution 3x3
dec4b	96	Convolution 3x3
up	144	Upampling 2x2
ct3	144	Transpose convolution
dec3a	96	Convolution 3x3
dec3b	96	Convolution 3x3
up	144	Upampling 2x2
ct2	144	Transpose convolution
dec2a	64	Convolution 3x3
dec2b	64	Convolution 3x3
up	64	Upampling 2x2
ct1	64	Transpose convolution
dec1a	32	Convolution 3x3
dec1b	32	Convolution 3x3
`output`	3	

Table 1: Network architecture of our final model. C_{out} denotes the number of output feature maps for each layer. All convolutions use padding mode "same". Except for the last layer which is followed by ReLU activation function, all the layers are followed by leaky ReLU activation function with $\alpha = 0.1$. Each convolutional layers are followed by BatchNormalisation with the corresponding number of features. After each upsampling, we concatenate the output of the transpose convolution layer with the corresponding maxpooling layer to reach the same number of input features of the next layer. Upsampling is nearest-neighbor.

6 Discussion and Conclusion

With this project, we have shown that it is possible to implement a U-Net that can recover a signal of corrupted images, without seeing clean images and without any explicit characterization of the noise. Our model could achieve a performance score of 22.96 dB. This means that clean data is not necessary to build a denoiser. The results can be further improved by using a bigger data set, more epochs, and more adapted meta-parameter.

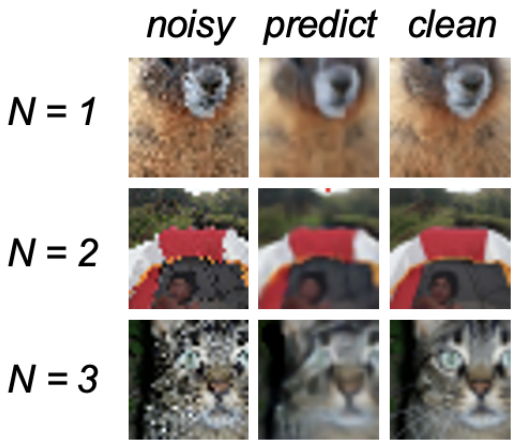

Figure 3: Results images. Noisy images were the input given to train our model, predicted images correspond to the output of our trained network and clean images were the target we tried to reach with our network.

References

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc.

Jaakko Lehtinen, Jacob Munkberg, Jon Hasselgren, Samuli Laine, Tero Karras, Miika Aittala, and Timo Aila. 2018. Noise2Noise: Learning Image Restoration without Clean Data. *arXiv:1803.04189 [cs, stat]*, October. arXiv: 1803.04189.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. 2015. U-Net: Convolutional Networks for Biomedical Image Segmentation. *arXiv:1505.04597 [cs]*, May. arXiv: 1505.04597.