EE-559 Mini-project 2: Custom Implementation of Noise2Noise-like Encoder-Decoder Denoiser

Xianjie DAI
Dept. of Computer Science
EPFL
Lausanne, Switzerland
xianjie.dai@epfl.ch

Guanqun LIU
Dept. of Computer Science
EPFL
Lausanne, Switzerland
guanqun.liu@epfl.ch

I. Introduction

In the second mini-project, our group implemented our own Noise2Noise-like encoder-decoder framework for image denoising without clean images. The original paper can be found at [1]. We implement the convolution and the transpose convolution through image-to-column and column-to-image [2] and achieve a Peak Signal-to-Noise Ratio (PSNR) of (dB) with limited training time. We will briefly introduce the techniques implementing the layers in section II and show our network framework and training results in section III. Finally, we will recap our implementation and discuss factors that may further improve our current project.

II. Methodology

A. Convolution

The original 2D convolution operation starts with a kernel or mask that slides over the 2D input data and performs element-wise multiplication with the part of the input that it currently covers. Finally, we sum up the results to obtain a single output pixel. All such pixels comprise the output image. However, the sliding process of the kernel requires $O(L)$ matrix multiplications and additions, where L represents the number of possible kernel positions (eigensubmatrices) on the input. To improve computational efficiency, we implement image-to-column (img2col, unfolding) that flattens the input into row/column vectors according to each eigensubmatrix and reconstruct the vectors into an input feature matrix. A similar step is also implemented to the kernels except transforming them into column/row vectors. The above operations are also applied to the input and kernel of each channel and we conduct a large scale matrix multiplication and fold (col2img, folding) the resulting convolution matrix back to the actual image after convolution. Both forward and backward operations are performed in the above matrix form.

1) ***Forward****:* At layer l, the forward pass can be expressed as: $z^{(l)} = w^{(l)}x^{(l-1)} + b^{(l)}$, where $x^{(l-1)}$ is the flattened and padded input feature matrix, $w^{(l)}$ is layer l's flattened weight matrix, and $b^{(l)}$ is layer l's bias vector with length of l's output channels.

2) ***Backward****:* The backward propagation at layer l can be expressed as:

$$\begin{cases} (\frac{\partial L}{\partial w})^{(l)} = (\frac{\partial L}{\partial z})^{(l)}x^{(l-1)} \\ (\frac{\partial L}{\partial b})^{(l)} = (\frac{\partial L}{\partial z})^{(l)} \\ (\frac{\partial L}{\partial x})^{(l)} = (w^{(l)})^T(\frac{\partial L}{\partial z})^{(l)} \end{cases}$$

where $\frac{\partial L}{\partial w}$, $\frac{\partial L}{\partial b}$, and $\frac{\partial L}{\partial x}$ represents the derivative of the loss with respect to w, b, and x (input of layer l). The layer's backward function receives $\frac{\partial L}{\partial z}$ as input from its activation function and is cached during back-propagation. $\frac{\partial L}{\partial z}$ is flattened and padded, $x^{(l)}$ and $w^{(l)}$ are flattened to conduct matrix multiplication to get the gradients. It outputs a folded $\frac{\partial L}{\partial x}$ reshaped to the size of x^{l-1} before unfolding. that is sent to the previous layer's activation function to continue the backward process.

B. Transpose Convolution

The transpose convolution can be seen as the opposite operation of normal convolution, which we "propagate" the convolution result back to its original input. Although transpose convolution cannot restore pixel information due to one-to-many mapping, it can upsample the spatial dimensions of intermediate feature maps to recover the input size at the model output. Similarly, transpose convolution is still a convolution process, with the kernel rotates by 180°.

1) ***Forward****:* At layer l, the forward pass can be expressed as: $z^{(l)} = w^{R(l)}x^{(l)} + b^{(l)}$, where w^R is the flattened weight matrix rotated by 180°(central flipping) and $b^{(l)}$ is the bias vector with length of l's output channels.

2) ***Backward****:* The backward propagation at layer l can be expressed as:

$$\begin{cases} (\frac{\partial L}{\partial w})^{(l)} = ((\frac{\partial L}{\partial z})^{(l)}x^{(l-1)})^R \\ (\frac{\partial L}{\partial b})^{(l)} = (\frac{\partial L}{\partial z})^{(l)} \\ (\frac{\partial L}{\partial x})^{(l)} = (w^{R(l)})^T(\frac{\partial L}{\partial z})^{(l)} \end{cases}$$

$\frac{\partial L}{\partial w}$, $\frac{\partial L}{\partial b}$, and $\frac{\partial L}{\partial x}$ represents the derivative of the loss with respect to w, b, and x (input of layer l). The calculation process is similar as stated in *Convolution*'s backward pass.

C. ReLU

The output of ReLU is asymmetric around 0. Compared with Sigmoid and other activation functions, ReLU is simpler, converges faster, and does not saturate at the positive interval, which solves the vanishing gradient problem.

1) ***Forward****:*

$$R(x) = \begin{cases} 0, x < 0 \\ x, x \geq 0 \end{cases}$$

2) ***Backward****:*

$$R(x)^{'} = \begin{cases} 0, x < 0 \\ x, x \geq 0 \end{cases}$$

Notice that the gradient will always be 0 if $x < 0$, and parameters will never be updated.

D. Sigmoid

Sigmoid is a non-linear activation function that could add non-linearity to compensate for the deficiency of linear function.

1) ***Forward****:* Sigmoid can be expressed as:

$$S(x) = \frac{1}{1 + e^{-x}}$$

Sigmoid is continuous and differentiable everywhere, and constraints the output to the range The output of Sigmoid is within the range $[0, 1]$,

2) ***Backward****:* Since Sigmoid is the last layer of the model, in the backward process:

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial S} * \frac{\partial S}{\partial x} = \frac{\partial L}{\partial S} * S(x)^{'} = \frac{\partial L}{\partial S} * S(x) * S(1 - x),$$

where L is MSE loss, S is the output of Sigmoid, x is the input of Sigmoid, and $\frac{\partial L}{\partial S}$ is the backward of MSE loss.

E. MSE Loss

Since the output of our framework is still an image with the same size as input images. Also, the last layer of our framework is Sigmoid, which leads to loss saturation and

1) ***Forward****:*

$$L(Y, f(X \mid \theta)) = (Y - f(X \mid \theta))^2,$$

where Y is the label, $f(X \mid \theta)$ is the output of our model with parameter θ.

2) ***Backward****:* In the backward process, the derivative of the loss function with respect to the output of the model is given as:

$$\frac{\partial}{\partial f(X \mid \theta)} L(Y, f(X \mid \theta)) = 2 * (Y - f(X \mid \theta)).$$

F. SGD Optimizer

The Stochastic Gradient Descent algorithm can be written as:

$$X_{t+1} := X_t - \gamma_t \nabla f_i(X_y),$$

where sample $i \in [n]$ uniformly at random, t is time, and γ_t is step size at time t. The SGD optimizer takes the parameters of each module and step size as input, and conducts update iteratively.

III. Experiments & Results

A. Network Framework

We followed the instruction given in the project description and build the following framework:

Layers	Configuration
Input	$channel_{input} = 3$
Convolution	$kernel = (3, 3), stride = 2, padding = 1$
ReLU	$channel_{input} = 48$
Convolution	$kernel = (3, 3), stride = 2, padding = 1$
ReLU	$channel_{input} = 96$
ConvTranspose2d	$kernel = (3, 3), stride = 2, padding_{out} = 1$
ReLU	$channel_{input} = 48$
ConvTranspose2d	$kernel = (3, 3), stride = 2, padding_{out} = 1$
Sigmoid	$channel_{input} = 3$

TABLE I: Network Structure

B. Configurations & Results

We set the learning rate (or step size) to a very large one, 0.1, because this large learning rate provides the fast converging with a fixed batch size of 16. Our model could achieve a PSNR of 21.62 after 10 epochs within 10 minutes.

We provide a sample of our training results as Fig. 1 shows.

(a) Noisy (b) Clean (c) Output of N2N

Fig. 1: A sample of our N2N output

IV. Conclusion

To summarize, we implement a simplified auto-encoder of Noise2Noise-like image denoiser through convolution layers and transpose convolution layers with img2col and col2img speeding techniques. With fine-tuning on feature maps in Conv/Transconv layers, SGD learning rate, and momentum, we achieve a PSNR of (dB) with limited training time that could compete with the complex U-Net models we implemented in project 1. Further improvements may include different layer initializations and data augmentation approaches.

References

[1] J. Lehtinen, J. Munkberg, J. Hasselgren, S. Laine, T. Karras, M. Aittala, and T. Aila, "Noise2Noise: Learning image restoration without clean data," arXiv.org, 29-Oct-2018. [Online]. Available: https://arxiv.org/abs/1803.04189.

[2] K. Chellapilla, S. Puri, P. Simard, et al. High performance convolutional neural networks for document processing. In Tenth International Workshop on Frontiers in Handwriting Recognition, 2006.