Deep Learning - EE-559

Report of Project 2

Implementation of several Deep Learning modules

Louise Coppieters
Suhan Narayana Shetty
Centamori Frank

May 27, 2022

1 The Modules

The different modules are classes constructed with an init, a forward, a backward and a param function. With this structure, it is possible to create a model composed of a series of modules, in the same way the torch.nn.Sequential works. The forward operation takes the images as input and return the output. The backward operation takes the grad wrt the output as input and returns the grad wrt the input. Each module was tested to assess the correctness of the implementation compared to regular Pytorch solution. Specificities of the different modules are shown in their subsections.

1.1 Relu and Sigmoid

The two activation functions modules have no parameters. Their forward pass is the activation function itself, applied to the input. As for the backward pass, it is the derivative applied on the back-propagated loss.

1.2 Convolution

This module computes the convolution between the input image and the filter as a matrix vector multiplication. The working of our implementation of the convolution is illustrated on Figure 1 and for the matrix multiplications, the operations and matrix sizes are illustrated in Table 1.

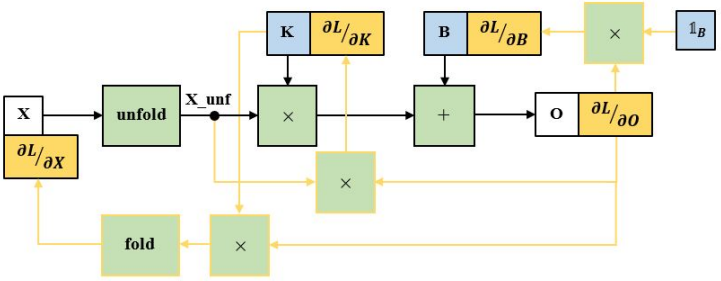

Figure 1: Schematics of the convolution (forward pass in black, backward pass in yellow)

The convolution can be mathematically expressed as:

$$O = K * X + B$$

For the input X the implementation uses the unfold function. After this operation, each of the columns corresponds to the elements of the original image which would be convoluted with the kernels at one iteration. The kernels, K, are reshaped with each line being one kernel. The unfolded input and kernels can then be multiplied. Finally, the bias term B is added to each channel.

For the backward pass, three different gradients of the loss should be computed: one wrt the input ($^{dL}/_{dX}$), one wrt the kernels ($^{dL}/_{dK}$) and one wrt the bias ($^{dL}/_{dB}$). The gradient wrt the input will be the gradient wrt the output given to the previous module in the model. While the gradient wrt the kernels and the bias are needed for the optimizer step.

The gradient wrt the input is given by $^{dL}/_{dX} = {}^{dL}/_{dO} \cdot {}^{dO}/_{dX}$. For the matrix operations, the inverse logic of the forward pass must be applied. The gradient wrt the output and kernels are reshaped and multiplied. The obtained result is then folded to match the size of the input matrix.

The gradient wrt the kernel is given by: $^{dL}/_{dK} = {}^{dL}/_{dO} \cdot {}^{dO}/_{dK}$ with $^{dO}/_{dK} = X_{unf}$. For the bias, the derivation of the output wrt the bias is equal to one. The gradient wrt to the bias can thus be computed: $^{dL}/_{dB} = {}^{dL}/_{dO} \cdot {}^{dO}/_{dB}$ with $^{dO}/_{dB} = \mathbb{1}_B$.

An important operation in the convolution for the module to work correctly is the initialization of the weight and the bias. Both parameters are initialized with a uniform distribution with the boundaries: $\left[-\frac{1}{\sqrt{\text{fan\_in}}}, \frac{1}{\sqrt{\text{fan\_in}}}\right]$, fan_in corresponds to the number of inputs of every picture fan_in $= ch_{in} \cdot k \cdot k$. This initialization is also used in the Pytorch implementation of the convolution.

	Initial size	Operation		Unfolded or expanded
X	Bs, ch_{in}, s_{in}, s_{in}	unfold	$\longrightarrow$	Bs, $(ch_{in} \cdot \mathrm{k} \cdot \mathrm{k})$, $(s_{out} \cdot s_{out})$
K	ch_{out}, ch_{in}, k, k	view	$\longrightarrow$	ch_{out}, $(ch_{in} \cdot \mathrm{k} \cdot \mathrm{k})$
O	Bs, ch_{out}, s_{out}, s_{out}	view	$\longleftarrow$	Bs, ch_{out}, $(s_{out} \cdot s_{out})$
${}^{dL}/_{dO}$	Bs, ch_{out}, s_{out}, s_{out}	view	$\longrightarrow$	Bs, ch_{out}, $(s_{out} \cdot s_{out})$
${}^{dO}/_{dX} = K$	ch_{out}, ch_{in}, k, k	view and $\top$	$\longrightarrow$	$(ch_{in} \cdot \mathrm{k} \cdot \mathrm{k})$, ch_{out}
${}^{dL}/_{dX}$	Bs, ch_{in}, s_{in}, s_{in}	fold	$\longleftarrow$	Bs, $(ch_{in} \cdot \mathrm{k} \cdot \mathrm{k})$, $(s_{out} \cdot s_{out})$
${}^{dL}/_{dO}$	Bs, ch_{out}, s_{out}, s_{out}	$\top$ and reshape	$\longrightarrow$	ch_{out}, $(\mathrm{Bs} \cdot s_{out} \cdot s_{out})$
${}^{dO}/_{dK} = X_{\mathrm{unf}}$	Bs, $(ch_{in} \cdot \mathrm{k} \cdot \mathrm{k})$, $(s_{out} \cdot s_{out})$	$\top$ and reshape	$\longrightarrow$	$(\mathrm{Bs} \cdot s_{out} \cdot s_{out})$, $(ch_{in} \cdot \mathrm{k} \cdot \mathrm{k})$
${}^{dL}/_{dK}$	ch_{out}, $ch_{in} \cdot \mathrm{k} \cdot \mathrm{k}$	view	$\longleftarrow$	ch_{out}, $(ch_{in} \cdot \mathrm{k} \cdot \mathrm{k})$
${}^{dL}/_{dO}$	Bs, ch_{out}, s_{out}, s_{out}	$\top$ and view	$\longrightarrow$	ch_{out}, $(\mathrm{Bs} \cdot s_{out} \cdot s_{out})$
${}^{dO}/_{dB} = 1$	$(\mathrm{Bs} \cdot s_{out} \cdot s_{out})$	-	$\longrightarrow$	$(\mathrm{Bs} \cdot s_{out} \cdot s_{out})$
${}^{dL}/_{dB}$	ch_{out}	-	$\longleftarrow$	ch_{out}

Table 1: This table gives an overview of the different sizes of the matrices and operations needed for the matrix multiplication of the convolution.

1.3 Transposed convolution

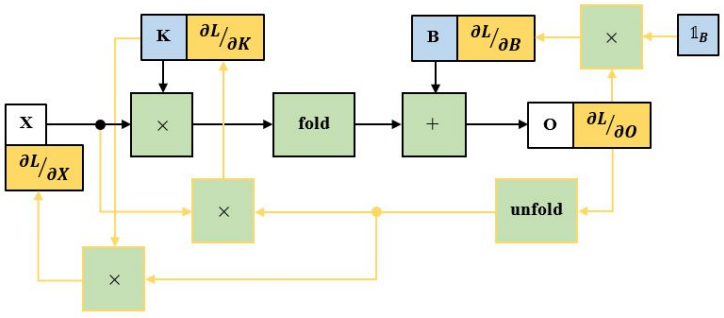

Figure 2: Schematics of the transposed convolution (forward pass in black, backward pass in yellow)

The transposed convolution is implemented with the same logic as the convolution. The only difference is the inversion of the order of operations compared to the convolution. Thus, the forward structure of the transposed convolution is based on the backward pass of the convolution and vice-versa. Figure 2 illustrates the operations of the forward and backward pass as well as the computations of the gradients of the kernel and the bias.

2 The loss computation: Mean square error

The MSE module has no parameters and computes the mean-squared error between the output of the model and a target element-wise. The backward pass takes the derivative of the MSE given its input and the target. It serves as the loss of the model.

3 A container: sequential

The sequential is a class used to create and interface a model composed of several interconnected modules. It takes the list of modules as input. In the forward pass, it gives the input to the first model,which returns a value that is given to the next module in the list and this is carried on iteratively through the different modules, before returning the output of the model. The Sequential block is also used by the optimizer to get the gradients and the parameters for the different steps.

4 The gradient step: Stochastic gradient descend

The SGD takes the model as input and can apply a zero_grad (that zeroes the gradients) or step on them. The step function iterates over the different modules and apply a gradient descent operation: the

gradients g are multiplied with the learning rate η and substracted from the weights w (kernels and bias). Besides, our SGD module also takes the momentum as input parameter, this introduces the velocity u and the damping γ. This adapts the gradients in order to converge faster and in a more stable way.

$$u_t = \gamma u_{t-1} + \eta g_t \quad (1)$$
$$w_{t+1} = w_t - u_t \quad (2)$$

5 Results

This structure allows us to create a simple model for denoising, using the Noise2Noise approach, to assess the performance of the homemade modules. In combination with the results using a complete model, assessment was undertaken of each modules individually in comparison with their Pytorch counterpart, to make sure the results are coherent.

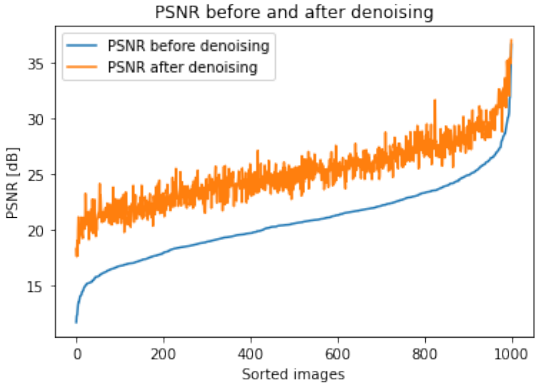

Figure 3: This figure illustrates the PSNR result for the different images of the test set. The blue curve shows the initial PSNR between the noisy and clean image. The orange curve is the PSNR obtained at the output af the model for the corresponding image. The PSNR range shows that some images perform naturally better than others, figure 4 illustrates typical images we can get with a low, average and high PSNR. The mean distance between the 2 plots is 4.37 dB

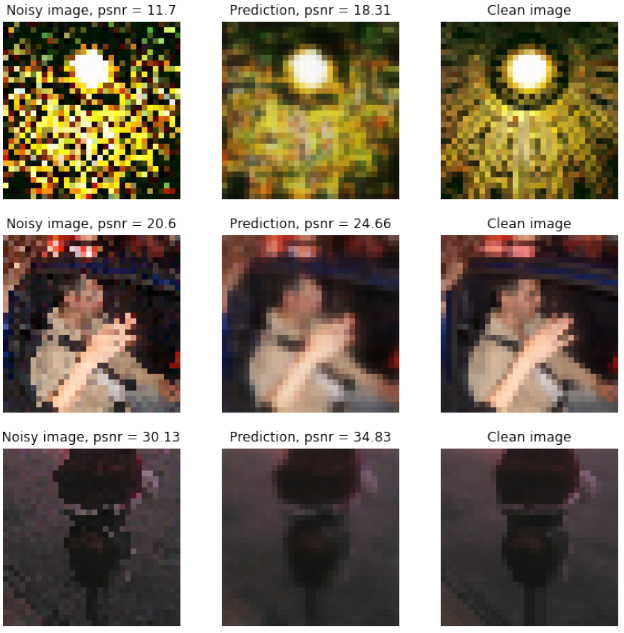

Figure 4: Examples of noisy, denoised and clean images.

The images on figure 4 illustrate which types of features can be retrieved and which not. Some patterns such as in the upper figure or small details such as the eye and hand of the man of the middle image are complicated to get. Bigger color patches can be denoised more easily, as they are composed of regular pixels. This shows the impact of the noise on loss of information. Whenever the objects have fine details, the noise will likely destroy the information of the shape or color of the object. While big patches of uniform color will likely keep more overall information when noisy.

6 Conclusion

The result is convincing with the homemade modules and structure, looking alike those which would result from a Pytorch implementation. The overall effects obtained seem to be quite comprehensible given the different types of images.