École Polytechnique Fédérale de Lausanne

Deep Learning

Professor François Fleuret

MiniProject 2

Students	**Sciper**
Gabriel Bessette	283364
Louis Ferment	288331

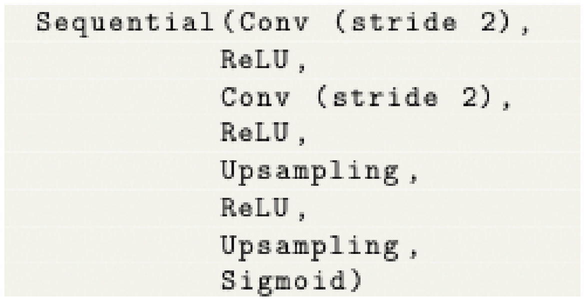
May 27, 2022

Table of Contents

1 Mini-Project 2

1.1 Introduction

The main purpose of this project was to create a denoiser from scratch without implicitly using Pytorch modules. The network is composed of Convolution, Relu, Upsampling and Sigmoid blocks. It has the following structure on Figure 1. **Note that in the provided code, the Upsampling module is followed by a Convolution module of stride 1.**



```
Sequential(Conv (stride 2),
           ReLU,
           Conv (stride 2),
           ReLU,
           Upsampling,
           ReLU,
           Upsampling,
           Sigmoid)
```

Figure 1: *Denoiser Network*

The main challenge was in building a convolution module, not only for the forward pass but also for the backward pass. In the following paragraphs, the main structure of the code will be described as well as the implementation of the module Conv2d. Finally, an investigation to find the best parameters of the denoiser has been led in order to obtain the smallest possible PSNR.

1.2 Code Structure

The main components of the code provided to generate and train such a network are the following:

- **Blocks:** network operations and activation functions were coded the same way (**Conv2d, Upsampling, Relu, Sigmoid**). They include a forward and a backward method as well as a method storing the weights w and biases b in a list of parameters.
- **Optimizer:** the **SGD** optimizer has been computed to update the weights of the network by updating them using the Stochastic Gradient Descent method.
- **Loss Function:** the **MSE** loss function has been implemented in order to compute the loss of the network's output with respect to a target image. It also includes the expression of its derivative to compute the gradient of the loss with respect to output.
- **Sequential:** a sequential module has been elaborated to generate the sequence of the network's blocks (see Figure 1). This class is also used as the model for this network and calls the forward and backward methods of all other classes for training.

The code takes as input a batch of images of size [N,C,W,H], with N the number of images, C the number of channels (RGB = 3), and H & W the respective height and width of the images. Training the network is done thanks to a training function over a certain number of epochs and, for a specified mini-batch-size and target batch of images. The network outputs an image of size [N,C,W,H] to be compared to the target images.

1.3 Conv2d

Focus is made on this block since it is the main module of the network. The following paragraphs describe the forward as well as the backward pass of the convolution operation on the image samples.

1.3.1 Forward Pass

The convolution operation has been performed by using matrix multiplication between the weights of the kernels and the value of the input images. This has been done thanks to torch.nn.functional.unfold and torch.nn.functional.fold, in order to rearrange the dimensions of the inputs and kernels before the multiplication. Additionally, a bias has been added to each kernel.

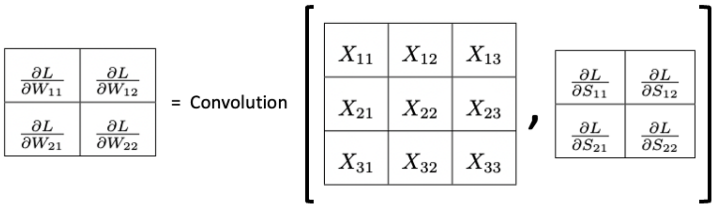
1.3.2 Backward Pass

According to S. Maheshwari [1] and P. Solai [2], the backward operation can also be seen as a convolution operation. Let L be the loss function, l be the l^{th} layer of the network, x^{l-1} be the input of the convolution, s^l be the output of the convolution, σ be the activation function after the convolution, and x^l be the output after having computed the activation function. A convolution operation followed by an activation function is given by the following sequence, with w and b the weights and biases of the convolution.

$$x^{l-1} \xrightarrow[w,b]{conv} s^l \xrightarrow{\sigma} x^l$$

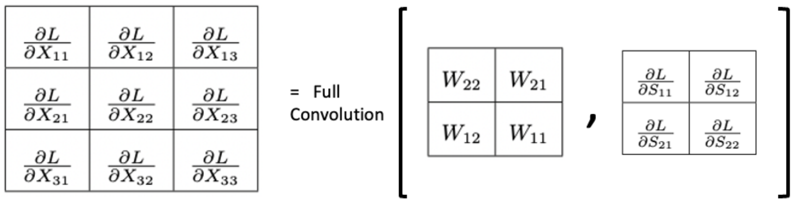
- Gradient of loss wrt. weights $\frac{dL}{dW}$:

For a stride of 1, the gradient of the loss with respect to the weights of the kernels $\frac{dL}{dW}$ is given by a simple convolution between the input x^{l-1} and the gradient of the loss with respect to the output s^l. For a stride of 2, a dilation of 2 must be added to dL/dS.


$\frac{\partial L}{\partial W_{11}}$ $\frac{\partial L}{\partial W_{12}}$
$\frac{\partial L}{\partial W_{21}}$ $\frac{\partial L}{\partial W_{22}}$
= Convolution
X_{11} X_{12} X_{13}
X_{21} X_{22} X_{23}
X_{31} X_{32} X_{33}
$\frac{\partial L}{\partial S_{11}}$ $\frac{\partial L}{\partial S_{12}}$
$\frac{\partial L}{\partial S_{21}}$ $\frac{\partial L}{\partial S_{22}}$

- Gradient of loss wrt. inputs $\frac{dL}{dX^{(l-1)}}$:

For a stride of 1, the gradient of the loss with respect to the input x^{l-1} is given by the full convolution of the rotated weights W and the gradient of the loss with respect to the output $\frac{dL}{dS}$. For a stride of 2, a dilation of 2 must be added to $\frac{dL}{dS}$ and a certain padding must be added to the weights W depending on the size of the output.


$\frac{\partial L}{\partial X_{11}}$ $\frac{\partial L}{\partial X_{12}}$ $\frac{\partial L}{\partial X_{13}}$
$\frac{\partial L}{\partial X_{21}}$ $\frac{\partial L}{\partial X_{22}}$ $\frac{\partial L}{\partial X_{23}}$
$\frac{\partial L}{\partial X_{31}}$ $\frac{\partial L}{\partial X_{32}}$ $\frac{\partial L}{\partial X_{33}}$
= Full Convolution
W_{22} W_{21}
W_{12} W_{11}
$\frac{\partial L}{\partial S_{11}}$ $\frac{\partial L}{\partial S_{12}}$
$\frac{\partial L}{\partial S_{21}}$ $\frac{\partial L}{\partial S_{22}}$

- Gradient of loss wrt. biases $\frac{dL}{dB}$:

Since there is one bias for each kernel, the gradient of the loss with respect to the bias of the kernels is given by the sum of the elements of the gradient with respect to the output dL/dS.

1.4 Parameter Investigation

In order to improve the model's performance, an investigation has been led on different parameters such as the number of channels C, the mini-batch's size B, as well as on the learning rate η. Each of these parameters have been modified one at a time according to a reference configuration: $\eta = 0.5$, $C = 3$, $B = 5$. For this study, the number of samples for training has been set to 5'000 and the number of epochs has been fixed to 5. The performance was evaluated once again by using the mean of the PSNR on the ouput and the target images. For this task, the convolutions have been applied with a kernel size of (3,3) a padding of 1 and a stride of 2. This way, the convolution reduced the output size by a factor of 2 for the down-sampling operations. The image size has been obtained once again thanks to the up-sampling blocks. The results are shown in the tables below:

Learning Rate	$\eta = 0.05$	$\eta = 0.1$	$\eta = 0.5$
PSNR [dB]	17.84	18.23	18.73

Table 1: *PSNR wrt. learning rate η, with $C = 3$, $B = 5$*

Nb of Channels	$C = 3$	$C = 6$	$C = 12$	$C = 32$
PSNR [dB]	18.73	19.12	21.45	21.94

Table 2: *PSNR wrt. number of channels C, with $\eta = 0.5$, $B = 5$*

Nb of Channels	$B = 2$	$B = 5$	$B = 20$
PSNR [dB]	19.48	18.73	18.32

Table 3: *PSNR wrt. number of channels C, with $\eta = 0.5$, $B = 5$*

From the results in the tables above, the selected parameters for an optimal model with a maximum PSNR are $\eta = 0.5$, $B = 2$ and $C = 32$. To speed up the calculations, the mini-batch size has been set to 5 and the number of channels have been increased progressively throughout the convolutions: from 3 to 16 to 32 for the down-sampling blocks and reversely for the up-sampling blocks. By training the model with these parameters, we obtain a final mean PSNR of **21.9** [dB] on the entire clean test sample. The optimal model can therefore be defined defined by the sequence :

```
Sequential(
Conv2d(3,16,3,padding = 1, stride=2), ReLU(),
Conv2d(16,32,3,padding = 1, stride=2), ReLU(),
NearestUpsampling(scale_factor=2), Conv2d(32,16,3,padding = 1, stride=1), ReLU(),
NearestUpsampling(scale_factor=2), Conv2d(16,3,3,padding = 1, stride=1), Sigmoid()
)
```

1.5 Discussion

The main challenge of this study was to compute the backward pass of the convolution operator. The provided code only works for a kernel size of (3,3), a padding of either 0 or 1 and a stride of either 1 or 2. During the backward operation, the kernel's size was hard to find back since the convolution operation is not reversible: for example, for a kernel size of (4,4) a padding of 1 and stride of 2, the output is the same as a convolution with identical parameters except for the kernel size of (3,3). One had to think of a solution to correctly crop the output of the backward pass in order to obtain the corresponding size of the respective gradients. But this solution was not generalized to all kernel sizes, paddings, strides and dilations.

References

[1] *S. Maheshwari, Backpropagation in Convolutional Networks, 2021.* URL: `https://codeandfire.github.io/assets/2021-07-16-backpropagation/article.pdf`.

[2] *P. Solai, Convolutions and Backpropagations, 2018.* URL: `https://pavisj.medium.com/convolutions-and-backpropagations-46026a8f5d2c`.