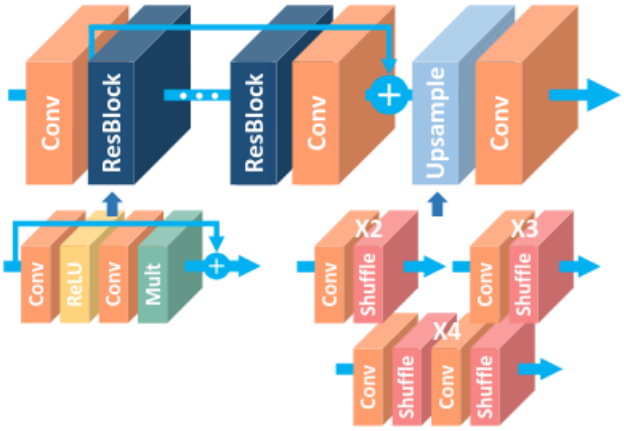
Figure 1: Super Resolution Architecture



1 Disclaimer

In this project, we've implemented two models with different hardware setups. The first model trained on a GPU using the standard PyTorch modules. It handles upscaling images from 128×64 to 256×128. The second model is trained on a CPU, where we wrote our own modules. Even though the implementations differ, both models keep the same method signatures. This means you can easily switch between them , allowing you to use checkpoints from either model interchangeably. For the smaller CPU model, we focused on upscaling images from 32×16 to 64×32. In this report we will present the results of both models.

2 Introduction

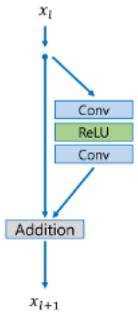
The goal of a Single Image Super Resolution Network is to take a low-resolution image and enhance its resolution by a factor of 2 or more. This is a regression task. The network takes an image in tensor form as input and outputs a larger tensor based on the scaling factor. The output tensor can be converted back into an image to visualize the upscaling result.
We can utilize a Convolutional Neural Network that employs several convolutional filters (learned during training) to leverage the regular structure of an image. A proposed architecture for this type of problem is shown in Figure 1. The network contains 3 main layers:

- Convolutional
- Residual Block
- Upsample

2.1 Convolutional And Residual Block Layers

As shown in Figure 1, the first part of the network consists of an alternation between Convolutional and Residual blocks. The architecture of the Residual Blocks is illustrated in Figure 2.

Figure 2: Residual Block Architecture



In the residual block layer, the input is processed through a convolutional filter, followed by a ReLU activation, another convolutional filter, and then summed with the original input for that layer before being passed to the next layer. The number of residual blocks in the network determines the depth of the network and is a parameter to be chosen during model selection. The implementation of the Residual Block is shown in listing 1 .

```
from torch import nn


class ResidualBlock(nn.Module):

        def __init__(self, num_channels):
                super().__init__()
                self.conv1 = nn.Conv2d(num_channels, num_channels,
                    kernel_size=3, padding=1)
                self.relu = nn.ReLU()
                self.conv2 = nn.Conv2d(num_channels, num_channels,
                    kernel_size=3, padding=1)

        def forward(self, x):
                res = x
                out = self.conv1(x)
                out = self.relu(out)
                out = self.conv2(out)
                out += res
        return out
```

Listing 1: Residual Block implementation

2.2 Upsample and Output Layer

The Upsample layer is the last layer responsible for increasing the dimensions of the tensor before passing it through the output convolutional layer. The implementation of the Upsample layer is shown in Listing 2. The component that actually performs the upscaling is the covolutional layer that "generates" the new pixels and the PixelShuffle that rearange it with the proper dimentions.

```
from torch import nn

class Upsample(nn.Module):

        def __init__(self, num_channel):
                super().__init__()
                self.conv = nn.Conv2d(num_channel, num_channel * 4,
                     kernel_size=3, padding=1)
                self.shuffle = nn.PixelShuffle(2)

        def forward(self, x):
                out = self.conv(x)
                out = self.shuffle(out)
        return out
```

Listing 2: Upsample implementation