Data representation, reduction and analysis

IMAGE SUPER RESOLUTION PROJECT

Implementing a deep convolutional neural network (DCNN) for image super resolution

ZHEKOV KAMEN, VAN HELDEN PHILEMON

2019-2020

Applied Computer Science: Smart Cities

Introduction

Super-resolution is defined as the process of reconstructing a high-resolution (HR) image from a lower resolution (LR) one. It is known as an ill-posed problem because there is no straightforward way to extrapolate the HR image pixels from the LR image. When upscaling by a factor of 2 (e.g. a 400x400 image giving us an 800x800 image), every pixel from the LR image becomes 4 pixels in the HR one, and the RGB values of those 4 pixels need to be deduced from the LR image. This means that we are trying to augment the amount of information contained in the original image by 4, while using only the LR image.

Simple algorithms like bilinear or bicubic interpolation exist and will be used as a benchmark for the Deep Convolutional Neural Network (DCNN) solution proposed in this work. Those techniques work only with local information contained in the LR pixels to obtain an HR image, and the results are often blurry and visually unpleasant.

DCNNs however learn to map LR pixels to HR pixels from a large number of examples, as they are trained with LR images as input and HR images as output. In our case, the DCNN trained for the project learns the inversed function of the downsampling used to create the LR images, which is the inversed bicubic interpolation.

The general expectations of the DCNN approach to the super-resolution problem is that it outperforms the simple algorithms, and that the results are not only better in terms of peak signal-to-noise ratio (PSNR), but also much more visually pleasing.

Literature and references

A very good comprehensive overview of the image super-resolution problem can be found in the following paper: Deep Learning for Image Super-resolution: A Survey. The authors take a deep learning perspective to review and benchmark the recent advances of the super-resolution techniques and underline the advantages and limitations of each component for an effective solution, but the article goes way deeper than the scope of this implementation. It is important to remember that the training of such a network will be done using a single GPU with merely 12 GB of RAM, or a computer with a 6-core CPU and 32 GB of RAM, meaning that most architectures presented in papers will be impossible to reproduce, simply because of hardware limitations.

There are two key concepts for the creation of a super-resolution DCNN. First, it is important to understand how upsampling works in different DCNNs (Figure 1 - Pre-upsampling and Figure 2 - Post-upsampling) and what kind of light architectures can be used for solving the problem. In the implementation discussed further, the architecture uses post-upsampling, which is a computationally efficient approach that forms most of the mappings in low-dimensional space, reducing greatly the number of computations required for the feature extraction process that happens in the convolutional layers. This contributes to a lighter architecture.

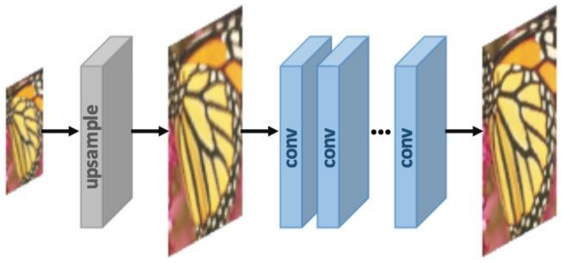

Figure 1 - Pre-upsampling

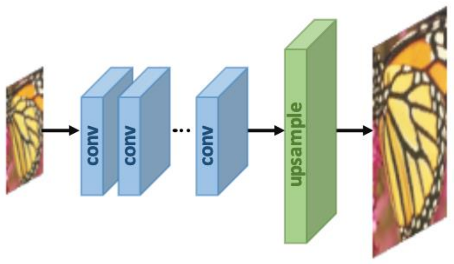

Figure 2 - Post-upsampling

Another important concept to understand is the way residual learning (Figure 3 - Residual Learning) functions. Since super-resolution is an image-to-image translation task where the low-resolution input is highly correlated with the target high-resolution image, a popular approach is to learn only the residuals between the two images, instead of the whole transformation from one complete image to another. This also contributes to a computationally lighter network.

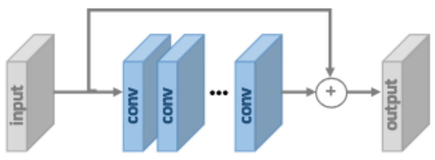

Figure 3 - Residual Learning

As there are many approaches to the super-resolution problem, the chosen implementation for the project was based on the winner of the NTIRE2017 Challenge on Single Image Super-

Resolution: Enhanced Deep Residual Networks for Single Image Super-Resolution. The paper tackles the problem using residual learning techniques and the architecture developed there achieves better than 2017 state-of-the art performance by optimizing and simplifying existing residual building block architectures (Figure 4 - Residual Blocks Architectures).

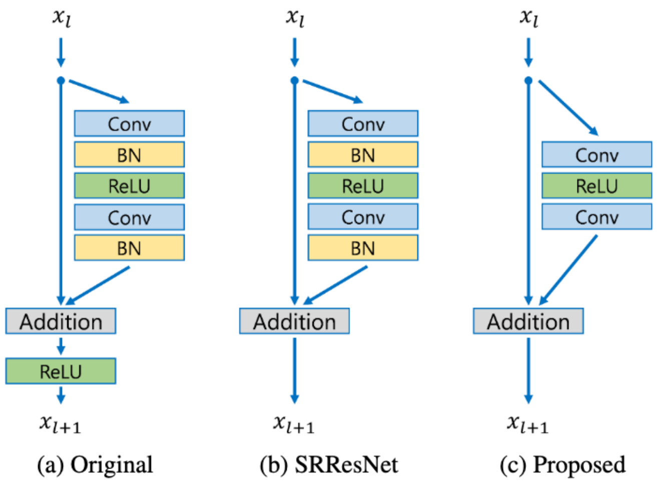

Figure 4 - Residual Blocks Architectures

The proposed single-scale architecture (Figure 4 - Residual Blocks Architectures and Figure 5 - EDSR Architecture) uses the above-described residual blocks followed by a final upsampling layer (previously mentioned post-upsampling) which is built differently based on the scaling factor, but essentially follows the same idea of using a convolutional layer, followed by a shuffle, followed by a convolutional layer.

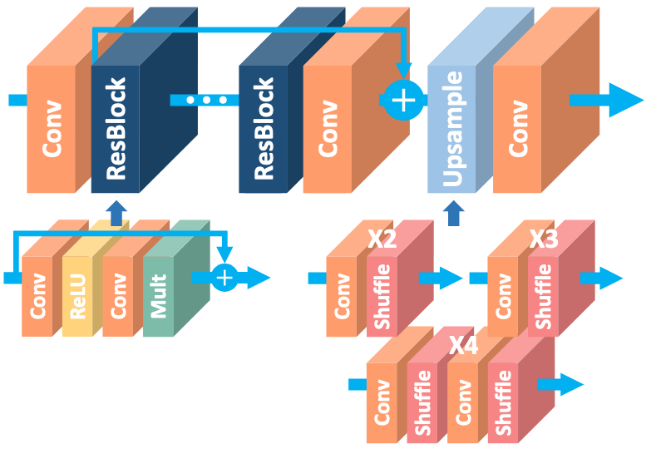

Figure 5 - EDSR Architecture

The way the architecture is designed allows for a variable number of residual blocks, defined as the network depth, and a variable number of feature channels of the convolutional layers, defined as the width of the network, and the authors state that for maximizing performance, increasing the width is preferable to increasing the depth where there are limited resources. The best model presented using the single-scale architecture has a depth of 32 layers and a width of 256 feature channels. This, of course, has been impossible to reproduce because of hardware limitations, so a more lightweight simplified version is presented in this work. The baseline model that the paper presented has a depth of 16 layers and a width of 64 feature channels, which was still too much to reproduce and train.

Implementation

The chosen implementation is based on the network described in Enhanced Deep Residual Networks for Single Image Super-Resolution whose architecture is shown in Figure 5 - EDSR Architecture, and is written in Python 3.6, using Keras with Tensorflow 2 as backend. A simple overview of the class that creates, trains and manipulates the model can be seen in Figure 6 - EDSR Model Creator, as well as the parameters that influence the created model's architecture.

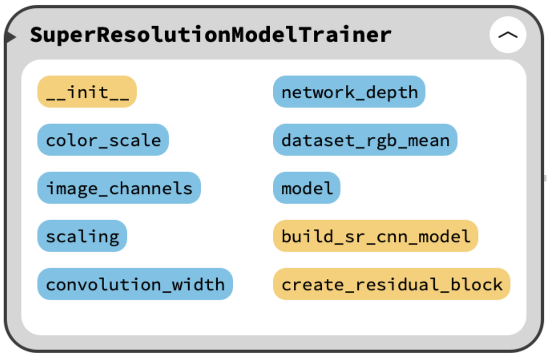

Figure 6 - EDSR Model Creator

The yellow blocks show the methods that are available through the creation class, the most important ones for the architecture being "*build_sr_cnn_model*" which handles the creation of the model layer-by-layer, and the "*create_residual_block*", which handles the creation of chained residual blocks.
The blue blocks show the variables that are important to the model's architecture, notably its scaling (x2, x3 or x4), network width (number of feature channels of the convolutional layers) and network depth (number of residual blocks). The other parameters influence the network architecture but should not be changed in most cases. For example, image channels could go from 3 to 1 if the model is only used for upscaling grayscale images.
A rendered version of the submitted x2 scaling model implemented for this project can be seen in Figure 7 - EDSR x2 scaling architecture. Other architectures implemented and trained were for x3 scaling, which was the same architecture as the x2, and x4 scaling which had a convolutional width of 64 and network depth of 8, and its architecture can be seen in the visuals folder submitted with this report.

Training

All models were trained for 100 epochs using the parameters specified in the paper: ADAM optimizer with settings β1= 0.9, β2= 0.999, and ϵ = 10−8. Unlike the paper, which trains x3 and x4 models using weights from the x2 model, which was trained from scratch, we train the three architectures from scratch, so that the training could be done in parallel.

The pixel-wise L2 loss and the pixel-wise L1 loss are frequently used loss functions for training super-resolution models. They measure the pixel-wise mean squared error and the pixel-wise mean absolute error, respectively, between the target HR image and the model-made super-resolution image. The pixel-wise L2 loss directly optimizes PSNR, an evaluation metric often used in super-resolution competitions but experiments have shown that the pixel-wise L1 loss can often achieve even better performance and is therefore used for EDSR training.

Batch training was used for the x2 and x3 architectures, where each batch was a single image and its augmented variations. One horizontal flip and one brightness change was the most augmentation possible with the available hardware. More than that would make the script crash.

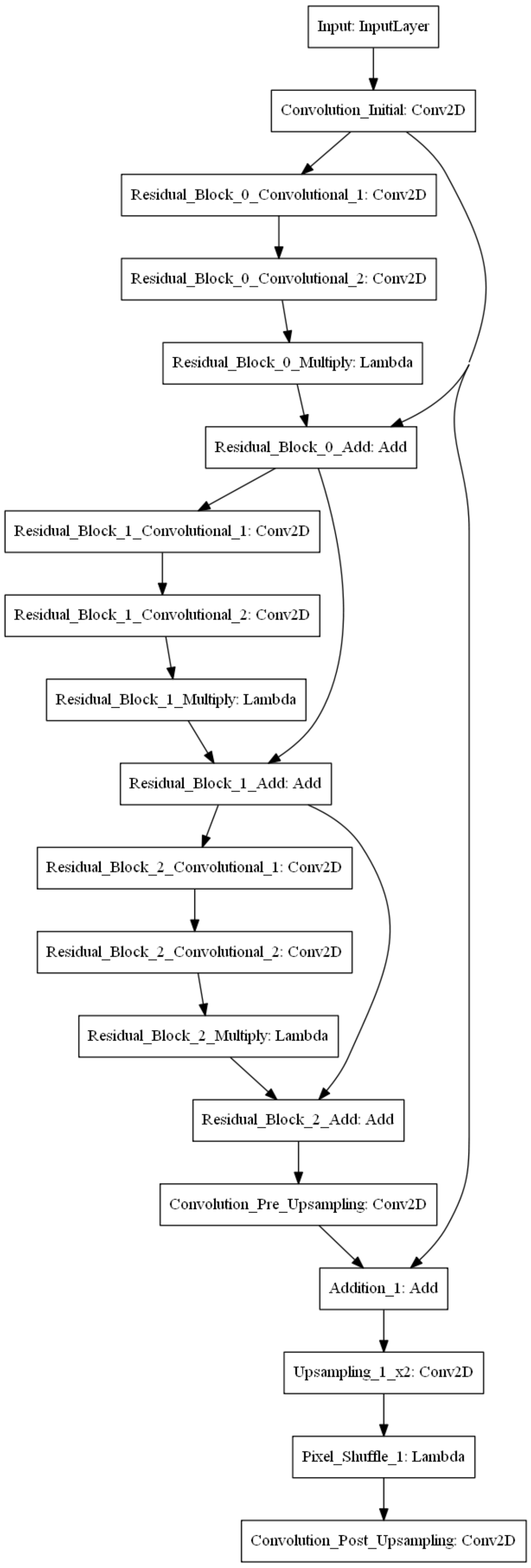
Input: InputLayer
Convolution_Initial: Conv2D
Residual_Block_0_Convolutional_1: Conv2D
Residual_Block_0_Convolutional_2: Conv2D
Residual_Block_0_Multiply: Lambda
Residual_Block_0_Add: Add
Residual_Block_1_Convolutional_1: Conv2D
Residual_Block_1_Convolutional_2: Conv2D
Residual_Block_1_Multiply: Lambda
Residual_Block_1_Add: Add
Residual_Block_2_Convolutional_1: Conv2D
Residual_Block_2_Convolutional_2: Conv2D
Residual_Block_2_Multiply: Lambda
Residual_Block_2_Add: Add
Convolution_Pre_Upsampling: Conv2D
Addition_1: Add
Upsampling_1_x2: Conv2D
Pixel_Shuffle_1: Lambda
Convolution_Post_Upsampling: Conv2D

Figure 7 - EDSR x2 scaling architecture

Correction algorithm

After training the models and observing the results on the validation data set, there was a major problem that was due to the lack of augmentation possibilities during training. The model had yet to see pixel transformations for some pixel values, which is something that could be fixed with enough augmented data. Unfortunately, making batches of enough augmented data was impossible, because of hardware limitations. The Google Colab servers have an Nvidia Tesla K80 GPU which has 12GB of RAM, and the most augmentation possible was a singe flip and a single brightness change for x2 and x3 scaling models, and only a flip or only a brightness change for the x4 scaling model.
Therefore, a practical correction approach was designed for the problem. Its performance can be seen in the results section below. The way the correction algorithm works is by correcting pixels that have aberrant values by replacing them with the pixel values of a bilinear interpolation of the LR image. The correction algorithm makes the upscaling significantly slower but gives results that are on average a lot better than results without it. If we do not use correction, the model's PSNR results are worse than plain bilinear interpolation, because even if the detailed parts are visually more pleasing and contain more details, the noise that appears in the super-resolution image ruins the result. This phenomenon and the correction at work can be seen in Figure 8 – Image 900 upscaling comparison Figure 9 – Image 879 upscaling comparison where the first image is the result of bilinear upscaling, the second is model only, the third is with correction.

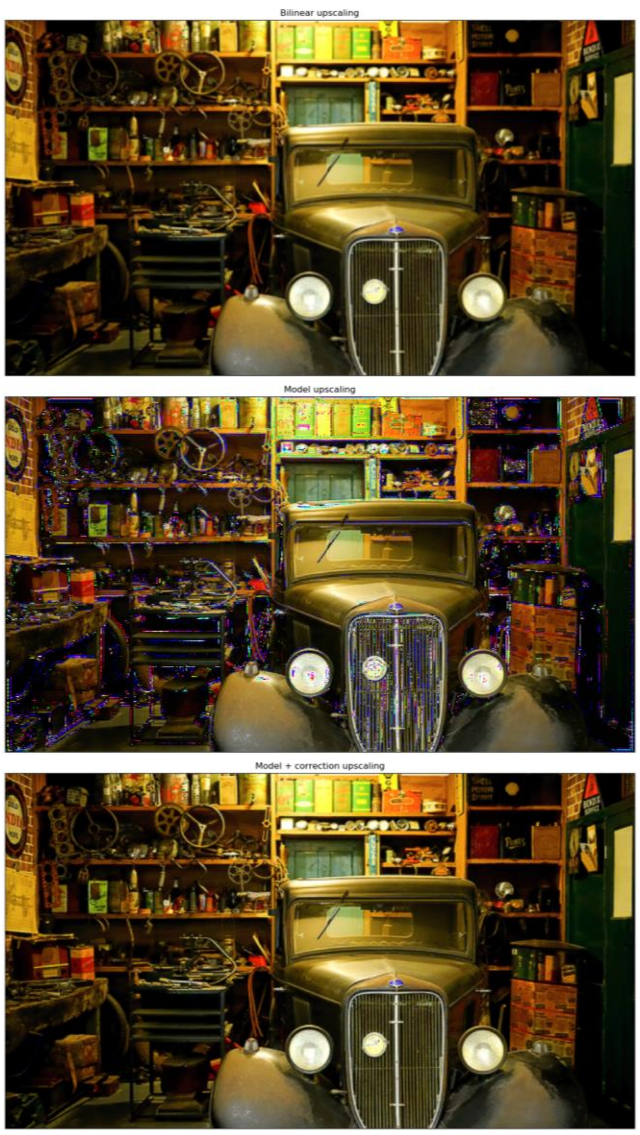

Figure 8 – Image 900 upscaling comparison

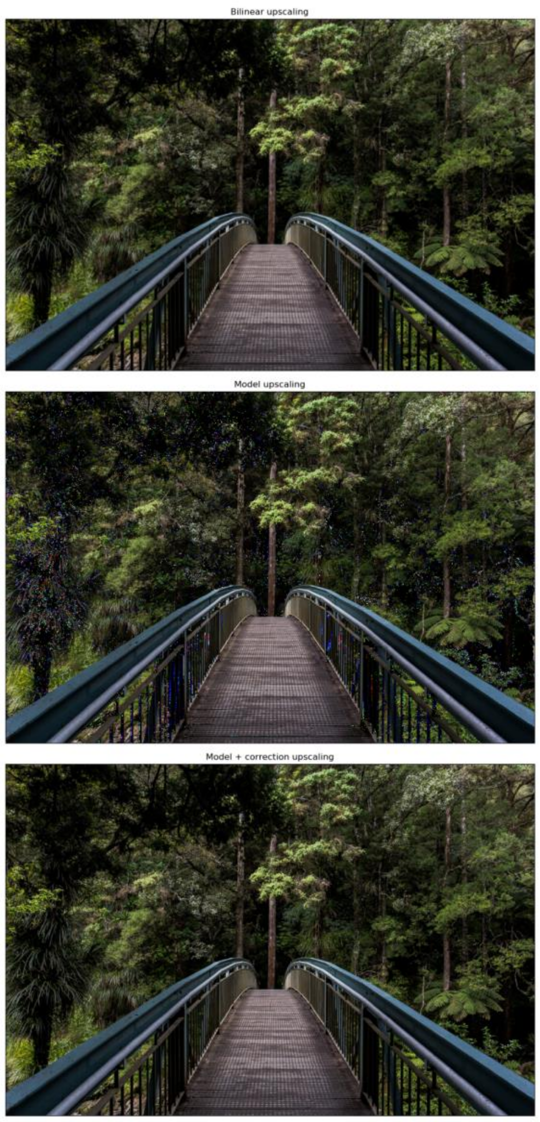

Figure 9 – Image 879 upscaling comparison

Results

PSNR was used as a metric to compare the performance of the model. The original performance of the model, plagued by RGB artefacts, is lesser than plain bilinear interpolation. Nevertheless, PSNR is just a metric, and even the bad results from the model give better details for the image, but the artefacts cannot be ignored.

Fixing them with the above-described correction algorithm drastically improves performance, and that performance is probably what the model could reach with image-augmentation techniques, if the hardware permitted it. The PSNR results from testing on the 100 images from the validation set of the DIV2K data can be seen in the table below. For all individual images' PSNR and the graph of the super-resolution experiment, refer to Figure 10 - x2 and x3 upscaling PSNR comparison and Figure 11 - x4 PSNR upscaling comparison.

	Model upscaling	Bilinear Upscaling	Corrected upscaling
x2 upscaling	28.43dB PSNR	30.03dB PSNR	32.41dB PSNR
x3 upscaling	25.56dB PSNR	27.57dB PSNR	29.11dB PSNR
x4 upscaling	23.89dB PSNR	26.48dB PSNR	27.39dB PSNR

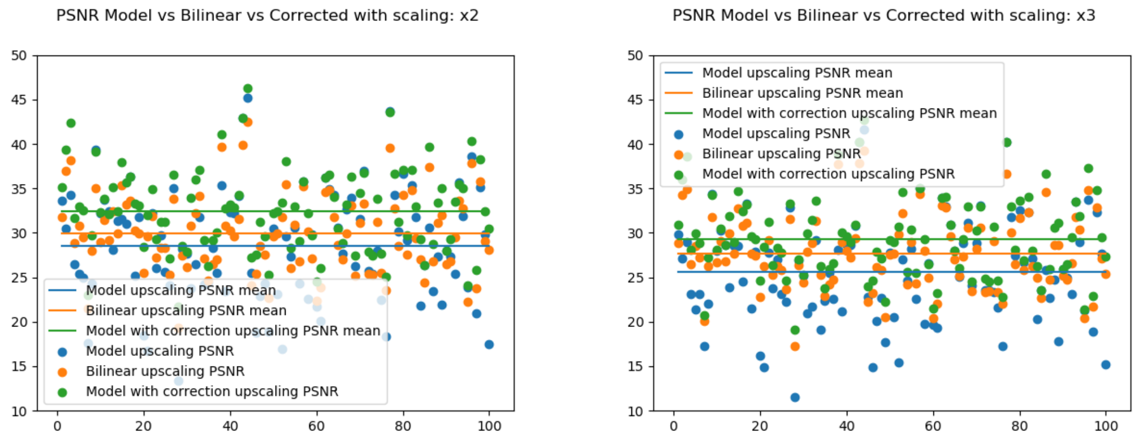

Figure 10 - x2 and x3 upscaling PSNR comparison

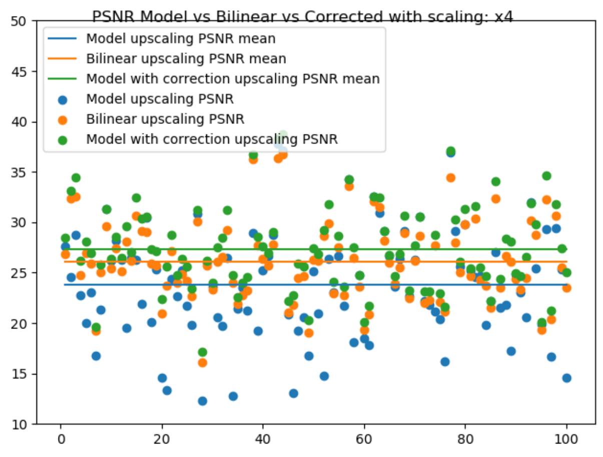

Figure 11 - x4 PSNR upscaling comparison

Upsampling details comparison

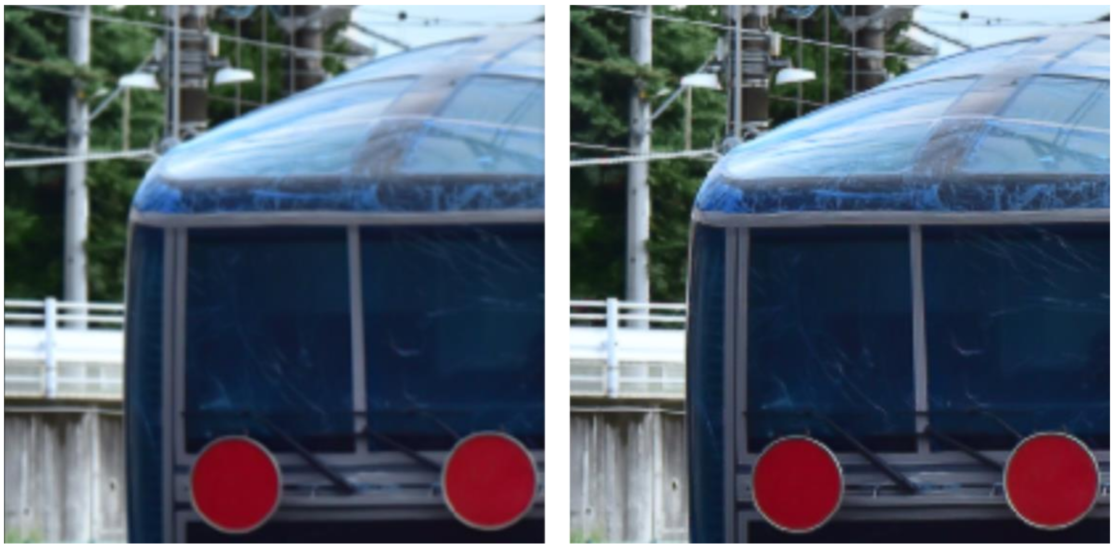

Figure 12 - Bilinear (left) vs model (right) upscaling on validation image 820

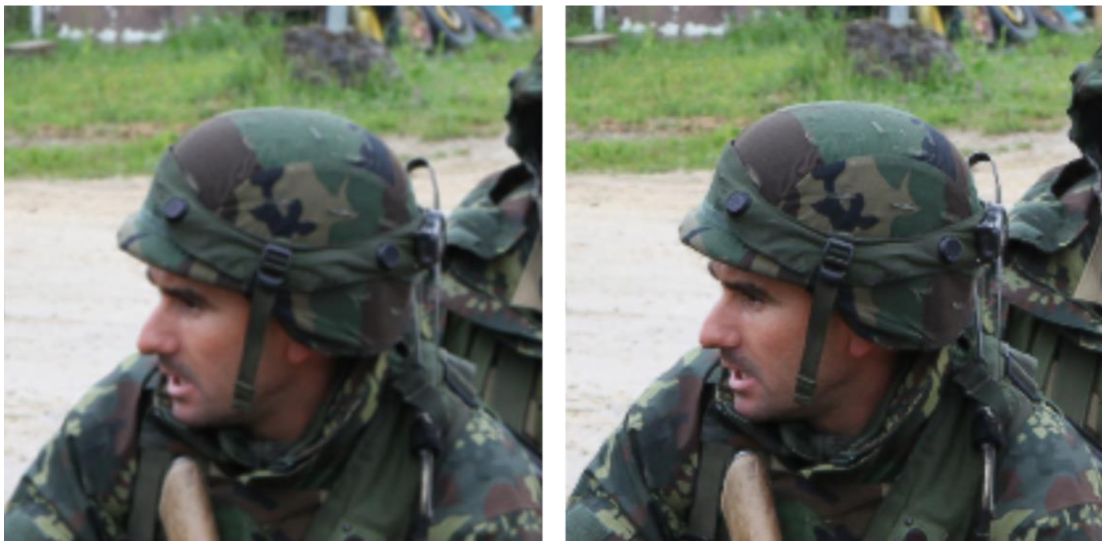

Figure 13 - Bilinear (left) vs model (right) upscaling on validation image 819

Conclusion

Solving the super-resolution problem using deep learning yealds promising results, which are definitely better than regular interpolation methods, and give more detailed upscaled images. The SR DCNN implemented in this work shows better results than linear interpolation when the output images are corrected and present no more noise.

A large number of augmented images is necessary when training the network, and without it, the network presents artefacts in the output images. To fix those, either more training is necessary, or a correction algorithm is introduced. We’ve opted for the second choice, and the correction algorithm indeed brings the output to an almost state-of-the art performance for lighter architectures.

A bigger network structure gives better performance, but requires exponentially more memory to train. Having limited hardware, state-of-the art performance cannot be achieved, and the results from the original paper cannot be reproduced, as training cannot even be initialized with the optimal parameters, let alone be run for 100 epochs.

The correction algorithm performs admirably, but can definitely be improved in terms of speed. It slows down the super-resolution significantly, as it has to go through every pixel for every color channel and replace it with the bilinear version if it exceeds certain thresholds. Perhaps a search algorithm could replace the cycling through all the pixels, but it exceeds the scope of this project.

All in all, super-resolution using deep learning can become one of the best techniques available, as long as the hardware permits training and using the right model. Correction algorithms can definitely help for artefacts, and probably for sharpening and improving the final image. As our hardware improves over time, DCNNs may be the go-to algorithms for upscaling images, but for now they are not flexible enough, as a new model is required for every different scale of architecture.