ML Group 32: 3.2.2 image upscaling

Johannes Pfennigbauer[1], Adam Skuta[2], and Lukas Fink[3]

[1]**e11902046@student.tuwien.ac.at**
[2]**e12208176@student.tuwien.ac.at**
[3]**e1191069@student.tuwien.ac.at**

ABSTRACT

In this assignment we make ourselves familiar with a Convolutional neural network (CNN) in the field of image upscaling. More concretely, we perform a 3x image upscaling. We pick a dataset used for training, prepare it for the specific CNN and the perform a fine-tuning based on some already pre-trained weights. After the training we perform multiple qualitative and quantitative evaluations comparing the original pre-trained model, with our fine-tuned ones.

Keywords: Deep Learning, CNN, upscaling

USED CNN

We use the suggested CNN from the Assignment: Super-Resolution Convolutional Neural Network (SRCNN)[2]. I.e., concretely its unofficial implementation[1]. The model consists of three convolutional layers, the first two also being activated by the ReLU function. The model visualization can be seen in Figure 1.

We adjusted the code a little bit to fit it to our needs:

- When training, one can now also specify to initialize the weights with a pretrained model. This allows us to finetune a pretrained model to the dataset we are using.
- When testing, the images will be saved actually downscaled. Originally, they were saved with the same resolution as the original (naive upscaling).
- We added a script that lets us quantitatively evaluate a model against an evaluation set of images (h5 file).
- When wanting to just downscale and save an image, we also added a script for that.

[1] `https://github.com/yjn870/SRCNN-pytorch`

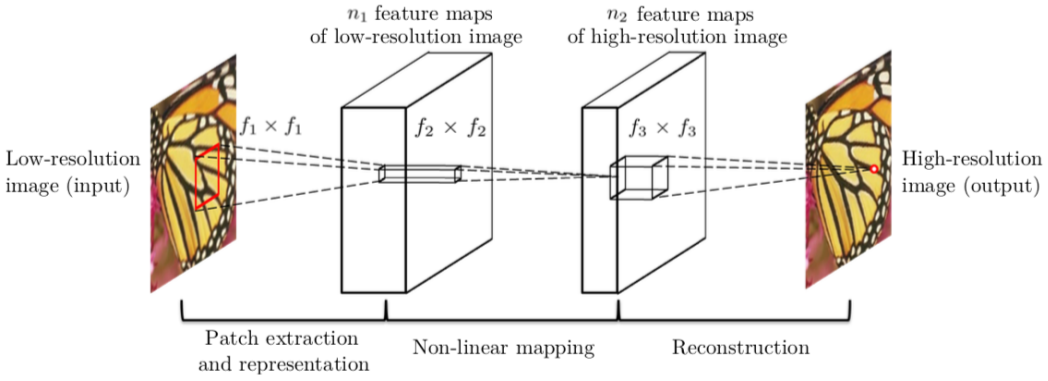


Figure 1. Architecture of the model from [2]

USED DATASET

We also use a dataset from the suggested source COCO[2]. Specifically, we used the first 2000 images of the *2014 Train images [83K/13GB]* dataset that can be downloaded from the official website[3].

FRAMEWORK

We set up a python poetry framework in order to ensure, that it is fully contained. Furthermore, all dependency issues are handled by poetry. The framework can easily be setup by following the instructions in the provided *README* file.

Poetry basically sets up a virtual environment. That means that all the needed packages in its specific versions will be installed there. This avoids dependency issues with e.g. another installed python instance with already installed packages.

FINE-TUNING THE CNN

When wanting to use our Framework together with the implementation of SRCNN, the first step is to prepare training and evaluation data (we will use an existing *h*5-file containing evaluation data from here). This is done by taking a folder containing images and converting it into a *h*5 file. A *h*5 file is basically an efficient way of storing large amount of data in a single file. Furthermore, suitable algorithms can efficiently work with these files, i.e. read and write to them.

The *h*5 file contains a high resolution (hr) and low resolution (lr) version of each image. Furthermore, not the whole image is saved but it is saved in so called patches. A patch is a region of an image, e.g. 33×33 pixel.

After preparing the data, we train the model based on a pretrained model (in our case the $\times 3$ pretrained model). We fine-tuned the model two times, resulting in two new models: The first one was trained with the first 1000 pictures (1k version) and the second one with the first 2000 pictures (2k version).

EVALUATION

In this section we will qualitatively and quantitatively evaluate the models.

Qualitatively

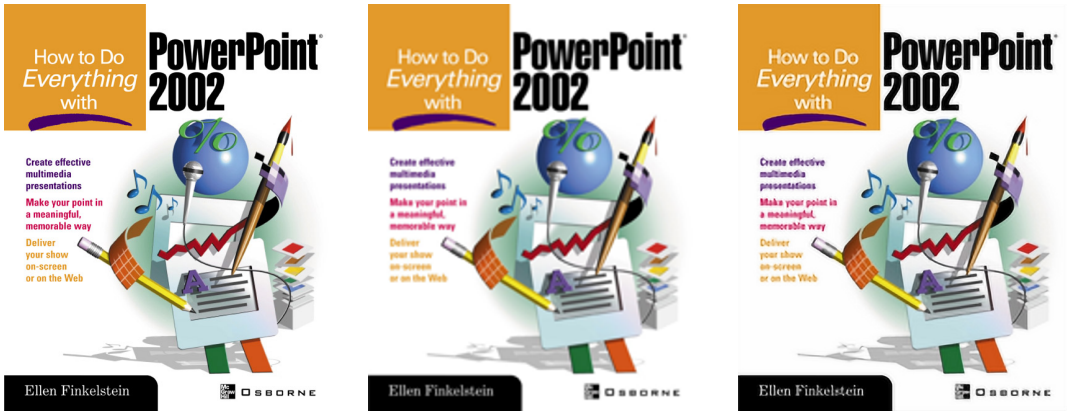
We will first compare original images against the downscaled and upscaled ones.

(a) Original **(b)** Downscaled **(c)** Upscaled (fine-tuned 2k)

Figure 2. Comparing the original, downscaled and upscaled images.

We can see in Figure 2, that the upscaled image looks a little softer than the original. We also clearly see, that some details got lost in the upscaled one, e.g. small light dots in the black borders of the wings of the butterfly. However, compared to the downscaled version, the upscaled version looks very promising. The black borders have now sharper edges and also the dark yellow region in the slight bottom left area of the picture is far more detailed.

[2] https://cocodataset.org/#home
[3] https://cocodataset.org/#download



(a) Original **(b)** Downscaled **(c)** Upscaled (fine-tuned 2k)

Figure 3. Comparing the original, downscaled and upscaled images.

When looking at Figure 3, we see that the upscaling works for text with some limitations. As seen in the pictures, it works best for large text, but far worse for small text. For example, the title is clearly readable and the difference is small compared to the original. However, if we look at the text in the middle left, it is not clearly readable in the upscaled version. We can conclude here, that this upscaling CNN is only useful for upscaling text up to a certain size.

We will compare now qualitatively the pretrained model with our two fine-tuned models.

(a) Pretrained model **(b)** 1k fine-tuned model **(c)** 2k fine-tuned model

Figure 4. Comparing the three models on a sample upscaled image.

If we look closely at Figure 4, we can see that the black boundary (pointed at by the red arrows) is sharper at our fine-tuned models. Other than the fact that the fine-tuned models look sharper, we noticed not much difference.

(a) Pretrained model **(b)** 1k fine-tuned model **(c)** 2k fine-tuned model

Figure 5. Comparing the three models on a sample upscaled image.

We have now a detailed look at the text on the bottom left. We will compare the three models and have a look at Figure 5. If we look very closely at the first ”l” of ”Ellen”, we see that in the pretrained model, there is a mix of grey pixels. However, when looking at the same letter in the 2k fine-tuned model, we see that in the center there are clearly white pixels going straight from top to bottom. Altough this is only a minor difference, we can conclude that our pretrained model is slightly superior at upscaling text.

Quantitatively

In this section, we will evaluate our fine-tuned model quantitatively. We will use on one hand the PSNR score and on the other hand the performance on a classification task.

PSNR scores

The PSNR score, namely Peak signal-to-noise ratio score, measures the quality between the original and the modified image. Concretely, it is "the ratio between the maximum possible value (power) of a signal and the power of distorting noise that affects the quality of its representation"[1].

It was introduced because of the need of an quantitative measurement when evaluating image enhancing techniques. The higher the PSNR, the better the method has reconstructed the original image.

We compute the average PSNR score over the evaluation set using the unofficial implementation[4]. For the first fine-tuned model where 1000 pictures of the COCO 2014 dataset were used, we get an average PSNR score of **33.32**. For the second fine-tuned model where the first 2000 pictures of the COCO 2014 dataset were used, we also get an average PSNR score of **33.32**. That means that with this measurement, we can't tell a difference between the two.

When comparing it to the pretrained model, we can see that the resulting average PSNR of **33.29** is actually slightly worse than both our finetuned models.

Comparing the performance on a classification task

Another possibility to evaluate the models performance is to test if a classification model performs differently on the upscaled images versus their originals. Therefore we used a fine-tuned version of the vit-base-patch16-224-in21k[5] model from google, which can be accessed here[6] and was trained from the user nateraw[7] on the food101[8] dataset. To evaluate the models performance we used 500 random samples from the validation set of the food101 dataset.
On the original images the model achieved an accuracy of 88%, i.e. it classified $440/500$ images correctly, which will be used as the benchmark for the upscaled versions of the images.
By using the fine-tuned upscaling model, which was trained on 1k images, we saw that the accuracy dropped slightly to 87.6% $(438/500)$. By comparing the predicted labels we found that in total four images (0.8%) were classified differently. While two of them were misclassified in both versions, the other two were misclassified because of the upscaling. The later can be seen in figure 6 and 7. However, by looking at the images it is very hard to tell what could be a reason for the missclassification. Therefore, it becomes reasonable to assume that in both cases the probabilites for the two affected classes were already very close together for the original images and the upscaling "broke the camel's back" by pushing the class with the previously lower probability to the top.

Figure 6. Original, downscaled and upscaled version of the first image, where the upscaling led to a missclassification

After that, we performed the same classification task on the upscaled images using the model, which we fine-tuned on 2k images. By using this model for upscaling the classification model achieved 88%

[4] https://github.com/yjn870/SRCNN-pytorch
[5] https://huggingface.co/google/vit-base-patch16-224-in21k
[6] https://huggingface.co/nateraw/food
[7] https://huggingface.co/nateraw
[8] https://huggingface.co/datasets/food101

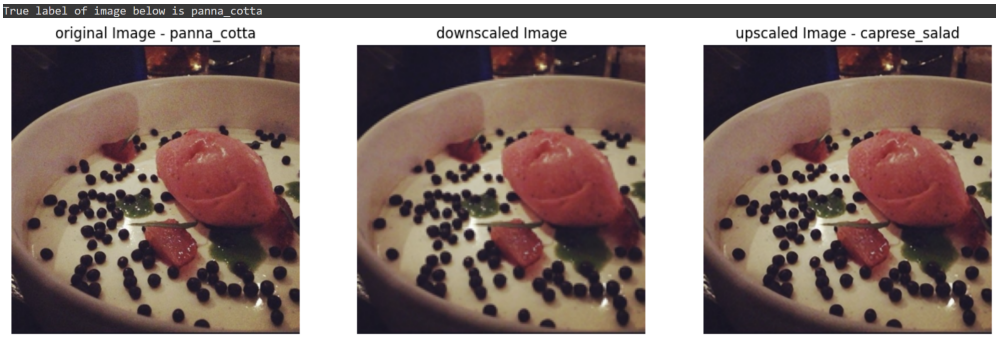


Figure 7. Original, downscaled and upscaled version of the second image, where the upscaling led to a missclassification

$(440/500)$ on the same classification task, which is slightly better than by using the other upscaling model and as good as using the original images! However, as previously four images (0.8%) were classified differently, but this time one image, which was originally missclassified, got predicted correctly in its upscaled version, see figure 8.

Figure 8. Original, downscaled and upscaled version of the image, where the upscaling led to a correct classification

Next to that, the image seen in figure 7 got misclassified in the same way as before. Furthermore, the same two images, like in the previous experiment, which were missclassified anyways, were wrongly predicted again,

We conclude from this experiment that the upscaling lead to minor changes in the probabilities of the individual classes, which potentially ends up in a different prediciton, if the probabilities are already very close together for the original image. However, we have also seen that this small adjustments can actually have a positive and negative impact on the final classification.

REFERENCES

[1] URL: https://www.ni.com/en/shop/data-acquisition-and-control/add-ons-for-data-acquisition-and-control/what-is-vision-development-module/peak-signal-to-noise-ratio-as-an-image-quality-metric.html.

[2] Chao Dong et al. *Image Super-Resolution Using Deep Convolutional Networks*. 2015. arXiv: 1501.00092 [cs.CV].