APPLYING TRANSFER LEARNING TO IMAGE SUPER-RESOLUTION USING CONVOLUTIONAL NEURAL NETWORKS

TECHNICAL REPORT

Andreas W. R. Madsen
Department of Computer Science
Aarhus University
Nordre Ringgade 1, 8000 Aarhus

Benno Kossatz
Department of Computer Science
Aarhus University
Nordre Ringgade 1, 8000 Aarhus

Md Sadik Hasan Khan
Department of Computer Science
Aarhus University
Nordre Ringgade 1, 8000 Aarhus

February 27, 2025

ABSTRACT

The potential of a combined transfer learning- and convolutional neural network-model remains largely unexplored for the image super-resolution task. In this paper, we implemented a custom convolutional neural network (CNN) based on the Super Resolution Convolutional Neural Network (SRCNN) as proposed by Dong et al. [1], trained it on the T91 dataset and validated on the union of Set5 and Set14 datasets. To enhance performance, we utilized transfer learning by employing the pre-trained CNN proposed by Simonyan and Zisserman [2], VGG, as a feature extractor. We explored different techniques including implementing a learning rate scheduler, weight decay and AdamW optimizer. Additionally, we investigated the impact of varying the encoder and decoder layer depths, aiming to maximize the Peak Signal-to-Noise Ratio (PSNR), our chosen performance metric. After fine-tuning, our final model achieved an average PSNR of 26.17 dB and 23.87 dB on Set5 and Set14 respectively, representing an improvement over our baseline model's 24.72 dB and 22.87 dB.

Keywords Deep Learning · Convolutional Neural Networks · Image Super-Resolution · Transfer Learning

1 Introduction

The rapid advancements in computer vision have significantly improved image processing tasks, with image super-resolution (SR) emerging as a critical yet challenging application. SR aims to enhance the resolution of low-quality images and has important applications in fields such as medical diagnostics, satellite imagery, surveillance, and sharpening visuals for gaming and media without requiring higher-resolution input. Additionally, it plays a vital role in improving automated system perception (ASP) [3].

Traditional super-resolution methods, such as bicubic interpolation, have been widely used for their simplicity, speed, and reliability as a baseline for evaluating newer techniques. However, despite these strengths, interpolation methods such as bicubic are fundamentally limited. They rely solely on the input image and cannot introduce new information, often resulting in blurry or less detailed outputs compared to modern deep learning-based approaches.

Deep learning-based approaches have shown remarkable performance in generating high-quality upscaled images. Much work with various approaches has already been undertaken, including the use of CNNs [1], nonlinear reaction diffusion [4], residual learning of deep CNN [5], GANs [6], Swin transformers [7] and Hybrid Transformers [8].

In other visual recognition tasks, transfer learning has been widely used to improve performance by leveraging pre-trained models that capture useful features from large-scale datasets [9, 10]. This technique allows models to generalize better on related tasks with limited data.

While both CNNs and transfer learning have individually shown great promise, the potential in combining the two for super-resolution remains largely unexplored. In this work, we incorporate VGG16 [2], a well-known CNN pre-trained on ImageNet, within an encoder-decoder architecture. VGG16 extracts features from low-resolution images, which are then reconstructed through multiple convolutional layers.

While more advanced methods for image super-resolution exist, our objective in using a simpler CNN-based model is to build upon a proven approach and apply transfer learning to it. Through this process, we aim to explore potential improvements and gain valuable practical insights into applying deep learning techniques to the super-resolution task.

2 Related Work

2.1 Image Super-Resolution

Image Super-Resolution is a fundamental task in computer vision that focuses on learning a mapping between low-resolution images and their high-resolution counterparts [1]. The objective is to recover high-resolution details from one [11] or multiple low-resolution input images.

Super-resolution methods can be broadly categorized into two approaches: Classical Multi-Image Super-Resolution, which combines information from multiple images, and Example-Based Super-Resolution, which learns the relationship between low and high resolution images using a database of image patches [11]. Our work focuses on the latter, where high-resolution images are densely cropped into overlapping patches, which are pre-processed and downscaled to generate their low-resolution counterparts. These patches are then encoded into dictionaries and sparse coefficients are used to reconstruct the high-resolution patches, which are aggregated through weighted averaging to form the final output.

Dong et al. [1] demonstrated that this patch-based pipeline could be effectively modeled using a deep convolutional neural network. In their approach, the hidden layers implicitly learn the dictionaries for patch representation, while patch extraction and aggregation are integrated as convolutional operations. This simple yet powerful model outperformed the state-of-the-art example-based methods at the time, setting a new benchmark for accuracy and efficiency in super-resolution.

2.2 Transfer Learning

Training a CNN from scratch is time-consuming and requires a large dataset. To address this, pre-trained CNNs, such as VGG or ResNet, which have been trained on millions of images, are often used as fixed feature extractors. These pre-trained networks provide a rich set of features that can be leveraged to train a classifier on a new, smaller dataset.

Pre-trained CNNs can also be fine-tuned by replacing the original classifier with a new one and retraining it on the target dataset. Fine-tuning involves updating the weights of the pre-trained network through backpropagation, allowing it to adapt to the specific characteristics of the new data.

However, fine-tuning all layers of a CNN can lead to overfitting, especially when the new dataset is small or differs significantly from the original training data. In such cases, a common strategy is to fine-tune only the higher-level layers, which capture more general features like edges and color patterns, while keeping the lower-level layers frozen to reduce the risk of overfitting [12].

2.3 VGG

A critical factor influencing the performance of convolutional networks is their depth. Although deeper networks can improve accuracy in tasks such as image classification, the addition of layers significantly increases computational complexity.

The VGG architecture, proposed by Simonyan and Zisserman [2], addresses this challenge by using small 3x3 convolution filters across all layers, avoiding the use of larger filters that would require more parameters. They demonstrate that this approach not only improves feature learning, but also enhances generalization across datasets. As a result, they were able to design networks with 16 and 19 layers that achieved state-of-the-art performance in various visual recognition challenges at the time.

We have selected the VGG architecture for applying transfer learning to the task of image super-resolution because its constant filter size allows for easy adjustments to the depth of the encoder, without causing significant changes in the receptive field. This is in contrast to other architectures, such as ResNet, that use varying filter sizes.

3 Methods

3.1 Model Architecture

Our model uses an encoder-decoder architecture to extract features from a low-resolution image and reconstruct it through additional convolutional layers. To leverage transfer learning, the encoder is built using multiple layers of the pretrained VGG16 model [2] available in the PyTorch library.

For the baseline model, denoted as VGGSR1, we include all layers up to and including the fourth convolutional layer, along with the pooling layer after the second convolutional layer. The pretrained VGG16 implementation in PyTorch also includes batch normalization, which we retain in the encoder of our model.

The decoder consists of a transpose convolution that upscales the extracted features to match the original image's width and height while reducing the number of channels. A final convolutional layer then reduces the channels to the original 3 RGB channels. ReLU activations are used to introduce non-linearity, and the kernel size of 3 is maintained throughout, in line with the VGG16 architecture.

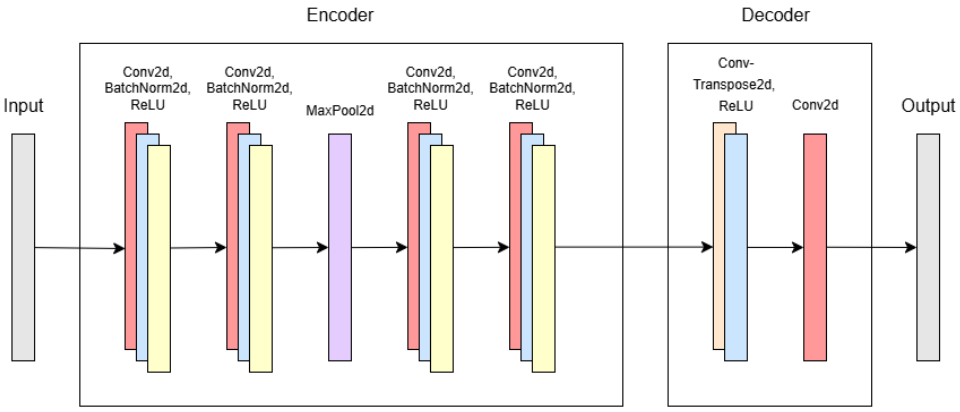

Figure 1: Architecture of the baseline model using the first layers of VGG-16 as encoder.

3.2 Dataset

Similar to the approach used by Dong et al. [1] we use the T91 dataset to generate patches on which we train our model. This dataset consists of 91 photos with varying motives containing mainly close-up shots of plants but also cars, people and man-made structures. The patches are extracted using the method described by Rath [13] with a size of 32 by 32 pixels at a stride of 14 pixels. This gives us 22.227 patches to train our network on.

For validation during training we use the union of the Set5 and Set14 datasets which are commonly used for evaluation in super-resolution tasks. These images are not turned into patches but passed through the network at their original resolution.

To calculate the loss, we use the generated patches and test images as ground truth. For each training patch and validation image, we create a lower-resolution version by scaling it down and then back up by a factor of four using bicubic interpolation. This process results in a low-resolution version of each patch with the same dimensions as the ground truth but with fewer details. By training the model on this dataset, it learns to perform 4x upscaling, recovering the missing details from the low-resolution input.

3.3 Evaluation

To evaluate the performance of our model, we use the Peak Signal-to-Noise Ratio (PSNR). This metric compares the super-resolved image with the original high-resolution image by taking the Mean-Squared-Error and transforming it to a logarithmic scale measured in decibels (dB). A higher PSNR means that the super-resolved image is closer to the original. The PSNR can be calculated using the following formula, where MAX is the maximum possible value of a pixel.

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(x_i - y_i)^2 \tag{1}$$

$$\text{PSNR} = 10 \cdot \log_{10}\left(\frac{\text{MAX}^2}{\text{MSE}}\right) \tag{2}$$

Due to being based on the Mean-Squared-Error, PSNR only takes into account local pixel-by-pixel differences and does not consider broader structural aspects of human perception.

To be able to compare the performance of the model to the results of other papers, the average PSNR for Set5 and Set14 are being used as our main benchmarks.

3.4 Experiments

Our baseline model when trained for 150 epochs delivers a PSNR of 24.72 dB on Set5 and of 22.87 dB on Set14 for 4x upscaling. This is suboptimal compared to other approaches of using CNNs for image super-resolution tasks like the model proposed by Dong et al. [1] which achieves a PSNR of 30.49 dB on Set5 and 27.50 dB on Set14. This leaves us room for further experiments to improve the model's performance.

3.4.1 Varying Model Depth

The depth of a neural network is critical for its ability to capture the relevant patterns in the training data to make accurate predictions. This holds true for the task of Image Super-Resolution.

The baseline model, VGGSR1, consists of a total of 6 convolutional layers and a total of 336,451 parameter. As seen in Table 1 it gets outperformed even by the basic upscaling technique bicubic interpolation which might indicate that the model is not able to capture all relevant features of the underlying training data needed to produce high-quality reconstructions.

To combat this, an additional model, denoted as VGGSR2, is introduced that includes two additional layers from the VGG16 model in the encoder leading to an encoder-output that is half the width and height but twice the channels. This has to be taken into account in the decoder where an additional transpose convolution is added to keep the overall output the same size. This model leads to a performance decrease, as seen in Table 1, suggesting that further abstraction in the encoder is not beneficial. Decreasing the depth of the encoder was considered but would go against our approach of using transfer learning as an even shallower encoder could easily be trained without using pre-trained weights.

Since we are so far only training the decoder, the missing ability to capture the features could also be combatted by adding additional layers to the decoder. For this, two additional models denoted VGGSR1+ and VGGSR2+, are implemented, where two additional convolutional layers are added in the decoder. These two models both show better performance, indicating that additional layers in the decoder are benefitial to accurately reconstruct the image from the extracted features.

Table 1: Different model architectures and their performance for 4x upscaling. For bicubic upscaling, the images were produced using the corresponding function included in the Python Imaging Library (Pillow). The exact implementation of the upscaling can influence the final PSNR values, which is why they might vary across different papers.

Model	Encoder Parameters	Decoder Parameters	Set5	Set14
Bicubic	-	-	*26.69*	*24.238*
VGGSR1	260,928	75,523	24.18	22.65
VGGSR2	556,608	149,251	22.47	20.99
VGGSR1+	260,928	260,035	**25.45**	**23.36**
VGGSR2+	556,608	518,147	23.85	22.01

3.4.2 Integrating Perceptual Loss

We enhanced the model's ability to generate high-quality images by incorporating a perceptual loss function based on a pre-trained VGG network. The perceptual loss captures high-level feature differences between the super-resolved images and the ground truth images, encouraging the model to produce outputs that are perceptually closer to the target.

We utilized the VGG16 network with batch normalization (`VGG16_BN`) as the feature extractor. The perceptual loss $L_{\text{perceptual}}$ is computed by comparing the feature representations of the output image $\hat{Y}$ and the ground truth image Y at selected layers of the VGG network:

$$L_{\text{perceptual}} = \sum_{l \in \mathcal{L}} \lambda_l \left\| \phi_l(\hat{Y}) - \phi_l(Y) \right\|_2^2,$$

where:

- $\phi_l(\cdot)$ denotes the feature map obtained from the l-th layer of the VGG network.
- $\mathcal{L}$ is the set of layers selected for feature extraction (we used layers after ReLU activations: `relu1_1`, `relu2_1`, `relu3_1`, and `relu4_1`).
- λ_l is the weighting factor for the loss at layer l (we set $\lambda_l = 1$ for all l for simplicity).

The total loss function L_{total} used to train the model combines the traditional pixel-wise Mean Squared Error (MSE) loss L_{pixel} with the perceptual loss:

$$L_{\text{total}} = L_{\text{pixel}} + \alpha L_{\text{perceptual}},$$

where:

- $L_{\text{pixel}} = \left\| \hat{Y} - Y \right\|_2^2$ is the MSE loss between the output and ground truth images.
- α is a hyperparameter that balances the contribution of the perceptual loss (we set $\alpha = 0.3$).

By incorporating the perceptual loss, the model is guided to minimize not only the pixel-wise differences but also the discrepancies in high-level feature representations. This results in super-resolved images that are sharper and more visually appealing, with better preservation of textures and edges. [14]

To ensure compatibility with the VGG network, both input and target images were normalized using ImageNet statistics. During training, the VGG network's parameters were kept fixed, allowing it to function solely as a feature extractor. The perceptual loss was computed in each training iteration and combined with the pixel-wise loss to update the model weights, guiding the network toward generating outputs that align more closely with both low-level pixel accuracy and high-level perceptual quality.

Training VGGSR1+ again with the perceptual loss implemented leads to a small improvement in the PSNR by raising it to 25.74 for Set5 and 23.54 for Set14. This is only a small difference but was to be expected as the previous pixel-wise loss was already geared to improve the PSNR which is also based on local pixel by pixel differences and does not include other perceptual aspects.

To better capture the overall image quality, we introduce the Structural Similarity Index Measure (SSIM) as an additional benchmark. SSIM is a method to calculate the perceived quality of the upscaled image by comparing its luminance, contrast and structure and taking into account local patterns. This emphasizes the factors relevant to human visual perception rather than pixel-wise differences.

Including the perceptual loss function increases the SSIM on Set5 to 0.757 (higher is better) and Set14 to 0.648 from 0.744 and 0.643 respectively. This leads to higher quality images that are closer to the original when perceived by a human.

3.4.3 Optimizing the Training Process

As seen in Figure 2 the PSNR for VGGSR1+ varies widely during training and does not seem to clearly converge. Moreover, the validation PSNR does not seem to improve after around 100 epochs. Both of these issues could be caused by the fixed learning rate of 0.001 preventing to the model from finding local minima in the loss landscape effectively.

To counteract this, a learning rate scheduler is implemented that reduces the learning rate by a factor of 10 whenever the validation loss plateaus for more than 25 epochs. A long training of 500 epochs is performed on the most promising model VGGSR1+. This helps the model converge as seen in Figure 2 but does not yield higher performance when testing on Set14 and Set5 indicating that the learning rate is not the main limiting factor during training and the model might be prone to overfitting.

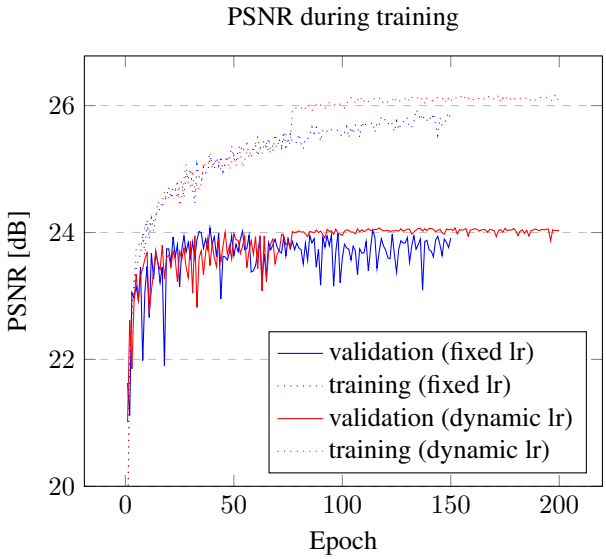

Figure 2: PSNR for VGGSR1+ using fixed learning rate (blue) versus a dynamic learning rate (red).

To deal with the issue of overfitting, weight decay is implemented and the optimizer is changed from Adam to AdamW. This should help the model generalize during the final training by punishing large weight values in the decoder during the training process.

Additionally, batch normalization is added to the decoder after each convolutional layers to combat the high variance of the validation PSNR during training and further stabilize the converging of the model. This works by dynamically normalizing the input of each layer to have a mean of zero and standard deviation of one.

After the final training of the decoder is performed, the model is fine-tuned by un-freezing the encoder layers and training the whole model with a fixed lowered learning rate of 0.00001, compared to 0.001 when training with the frozen encoder, for another 50 epochs. While fine-tuning typically carries a risk of overfitting, this was mitigated by the combination of the low learning rate and the relatively small number of additional epochs, ensuring the model retained its generalization capabilities.

4 Results

Our final model, after implementing various experiments and optimizations, achieved a PSNR of 25.97 dB on the Set5 dataset and 23.70 dB on Set14, with fine-tuning further improving these scores to 26.17 dB and 23.87 dB, respectively, as shown in Table 2. While these results demonstrate a clear improvement over the baseline model, they remain significantly lower than those achieved by dedicated CNN-based approaches like SRCNN [1] and state-of-the-art transformer architectures such as HAT-L [8]. The gap in performance highlights the limitations of our simpler encoder-decoder design based on transfer learning.

Table 2: Different model architectures and their performance for 4x upscaling

	Set5		Set14	
Model	PSNR	SSIM	PSNR	SSIM
Bicubic	*26.69*	*0.789*	*24.24*	*0.683*
VGGSR1 (baseline)	24.18	0.689	22.65	0.631
VGGSR (final)	26.17	0.766	23.87	0.651
SRCNN [1]	30.49	0.8628	27.50	0.7513
HAT-L [8]	33.30	0.9083	29.47	0.8015

While overall performance metrics provide a general overview of the model's effectiveness in upscaling tasks, a qualitative analysis of selected test images offers deeper insights into its strengths and weaknesses. As illustrated in Figure 3, the model's performance is highly dependent on the type of image being upscaled. This variability is also

evident in bicubic interpolation, which performs better on simpler images, such as the bird, where the background is mostly out of focus and contains few sharp edges. In contrast, for more complex images like the butterfly, with many distinct lines and fine details, bicubic interpolation produces poorer results. Although the PSNR and SSIM scores for the bird image are higher with bicubic upscaling than with VGGSR, a closer examination of magnified portions reveals that our model excels in reconstructing sharp edges, offering better perceptual quality. This edge reconstruction capability is likely the reason VGGSR outperforms bicubic interpolation on the butterfly image, producing more accurate and visually appealing black edge details.

Figure 3: Example images from Set5 (4x upscaling).

5 Discussion

While we successfully improved the model's performance over the baseline, it was still outperformed by other CNN-based models like SRCNN and significantly more by modern transformer-based approaches such as HAT-L.

The most noticeable improvement came from increasing the depth of the decoder, which proved more effective than adding layers to the pretrained encoder, as shown in Table 1. While the perceptual loss function had minimal impact on PSNR, it did enhance SSIM, leading to better perceived image quality. Additional adjustments to the training process only resulted in minor improvements.

Several factors may have contributed to the model's underperformance. First, hardware and time constraints limited our ability to train the model for thousands of epochs, as commonly seen in other studies, and restricted the use of larger datasets, which could have improved performance, as noted by Dong et al [1]. Additionally, the scope of this work did not allow for experimenting with more advanced architectures like GANs or transformers, which may better capture the complex patterns needed for high-quality image reconstruction.

Finally, it is worth considering whether transfer learning with a network designed for image classification is optimal for super-resolution tasks. Although VGG16 was effective as a feature extractor, certain components, such as max-pooling layers, may have limited its ability to accurately reconstruct high-resolution details.

6 Conclusion & Future Work

In this work, we evaluated a convolutional neural network with an encoder-decoder architecture using VGG as encoder and feature extractor for image super-resolution. The proposed approach showed to underperform in comparison with SRCNN by Dong et al. [1] and state-of-the-art architectures like HAT-L [8]. Increasing the number of layers in the decoder yielded the most substantial performance improvement, while incorporating the perceptual loss function contributed minor improvements. Given more time, one could investigate how VGGSR trained on a bigger dataset and for longer would perform. One could also explore the possibility of using a GAN in a transfer learning context.

References

[1] Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. Image super-resolution using deep convolutional networks, 2015.

[2] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

[3] Amanjot Singh and Jagroop Singh. Survey on single image based super-resolution —implementation challenges and solutions, 2018.

[4] Thomas Pock Yunjin Chen. Trainable nonlinear reaction diffusion: A flexible framework for fast and effective image restoration, 2015.

[5] Kai Zhang, Wangmeng Zuo, Yunjin Chen, Deyu Meng, and Lei Zhang. Beyond a gaussian denoiser: Residual learning of deep cnn for image denoising. *IEEE Transactions on Image Processing*, 26(7):3142–3155, July 2017.

[6] Christian Ledig, Lucas Theis, Ferenc Huszar, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, and Wenzhe Shi. Photo-realistic single image super-resolution using a generative adversarial network, 2017.

[7] Rui-Yang Ju, Chih-Chia Chen, Jen-Shiun Chiang, Yu-Shian Lin, Wei-Han Chen, and Chun-Tse. Resolution enhancement processing on low quality images using swin transformer based on interval dense connection strategy, 2023.

[8] Xiangyu Chen, Xintao Wang, Jiantao Zhou, Yu Qiao, and Chao Dong. Activating more pixels in image super-resolution transformer, 2023.

[9] Bahram Jalali Sifeng He. Brain mri image super resolution using phase stretch transform and transfer learning, 2018.

[10] Yongsong Huang, Zetao Jiang, Rushi Lan, Shaoqin Zhang, and Kui Pi. Infrared image super-resolution via transfer learning and psrgan, 2021.

[11] Michal Irani Daniel Glasner, Shai Bagon. Super-resolution from a single image, 2009.

[12] http://vision.stanford.edu/teaching/cs231n/. Transfer learning, https://cs231n.github.io/transfer-learning/, (accessed 24-11-2024).

[13] Sovit Ranjan Rath. Srcnn implementation in pytorch for image super resolution, 2022.

[14] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution, 2016.