MASTER'S THESIS 2019

Efficient Deep Learning Architectures for Super-Resolution

Adam Thuvesen

ISSN 1650-2884
LU-CS-EX 2019-23

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

EXAMENSARBETE
Datavetenskap

LU-CS-EX 2019-23

Efficient Deep Learning Architectures for Super-Resolution

Adam Thuvesen

Efficient Deep Learning Architectures for Super-Resolution

(Evaluation and Design)

Adam Thuvesen
dic14ath@student.lu.se

October 2, 2019

Master's thesis work carried out at Sony Mobile in Lund.

Supervisors: Volker Krüger, volker.krueger@cs.lth.se
Sebastian Raase, sebastian.raase@sony.com

Examiner: Pierre Nugues, pierre.nugues@cs.lth.se

Abstract

The purpose of this thesis was to analyze efficient neural network architectures for Super-Resolution. This is a technique used to upscale low resolution images to high resolution. In recent years rapid development has been made with the use of deep convolutional neural networks. Although the networks have achieved impressive visual upscaling performance they have as a consequence also been very deep, resulting in large and slow networks. This has led to difficulties in applying them in real-world applications.

Therefore, the focus of this thesis has been on small and efficient network architectures. The work was divided into a two step process. First, we performed an extensive quantitative evaluation of different techniques. Then we designed a model based on the most promising empirical data. The model was assessed with quantifiable practical and visual metrics. In this lightweight network category, the model achieved satisfying results with similar practical performance but with superior visual quality compared to other models employing efficient state-of-the-art techniques. Subjectively, the perceptual quality of the upscaled images is high, clearly outperforming traditional methods such as B-spline interpolation. Finally, we also showed that it is possible to prime the network towards different visual objectives by adapting the loss function and achieve a greater degree of customizability.

Keywords: Super-Resolution, Machine Learning, Deep Learning, Convolutional Neural Networks, Efficient Architectures

Acknowledgements

First I would like to thank Volker Krüger as supervisor for his general support and meaningful discussion throughout this thesis. I would also like to thank Sebastian Raase at Sony for all the good feedback on the report. Lastly, thanks to Sony and the camera team which provided a friendly and helpful environment for this work.

Contents

Chapter 1
Introduction

This chapter gives an introduction to the thesis, including an initial explanation of the topic and the purpose of this work. It states the research problem and describes the previous research. This includes an overview over the most relevant terminology and concepts, where the most important ones will be explained in more detail in the background chapter. At last, the outline of the report is explained.

1.1 Super-Resolution

Super-resolution (SR) is a technique used to enhance the resolution of an image. Increasing the resolution of an a low resolution image is a notoriously challenging problem due to the fact that it could correspond to multiple high resolution images. This has been proven to be an intractable problem [3]. The field of super-resolution has been gaining attention in recent years because of great improvements in image reconstruction quality. The technique can be applied to a wide variety of areas, such as flight and satellite imaging, medical image analysis, video streaming and surveillance. The field can be divided into *single-image super-resolution* (SISR) and *multi-image super-resolution* (MISR). This thesis focus is on SISR due to its higher efficiency.

1.2 Research Problem

The purpose of this thesis is to analyze efficient neural network architectures for super-resolution; specifically focusing on model size, run time and still maintaining high visual quality of the upscaled images. In recent years great progress has been made. However, the models have been getting larger and larger; from the first successful network with 3 layers to the recent ones with 160 layers [18]. Consequently, the models have been computationally demanding, being both large and slow to run. With the rise of mobile phones and the

Internet-of-Things (IoT), there is an increasing interest in building faster and smaller models. This is crucial to apply deep learning solutions for super-resolution to real-world applications where the memory and computational resources are limited. Currently, these solutions are still mainly in research.

This thesis can be divided into two parts: evaluation and design. The objective of the *evaluation* is to analyze different neural network architecture for super-resolution. Then, based on this evaluation the goal is to use the most successful techniques to *design* a small and efficient network. Hence, we want to investigate if super-resolution is applicable to these light networks and if so, how could you design such efficient architectures?

The objective was to find a suitable balance between the practical performance (like the number of parameters of the model and the run time of the upscaling procedure) and the visual performance (the quality of the generated image). By basing the network on an extensive evaluation of different techniques, we also get a better understanding of why a network performs as it does, an area in deep learning which generally is overlooked. To evaluate the success of the proposed model, existing super-resolution networks will be compared. These will be assessed using the most common image reconstruction metrics. However, judging quality of images is inherently subjective. Therefore a qualitative, i.e. visual, evaluation is also performed. Finally, we investigate the possibility of customization. The purpose of this was to see to what extent it is possible to prime the model to generate images with different visual objectives. This would ease the deployment to fields where the image requirements might differ substantially.

1.3 Previous Research

The research field of deep learning solutions for computer vision tasks exploded after the appearance of *AlexNet* in 2012 [16]. The proposed convolutional neural network (CNN) showed tremendous performance enhancement compared to previous methods for object classification. This led to research into deep learning methods in a wide variety of other computer vision tasks, including super-resolution. To get an idea of the exponential increase in interest of convolutional neural networks see Fig. 1.1 where the Google searches for the term is shown over time.

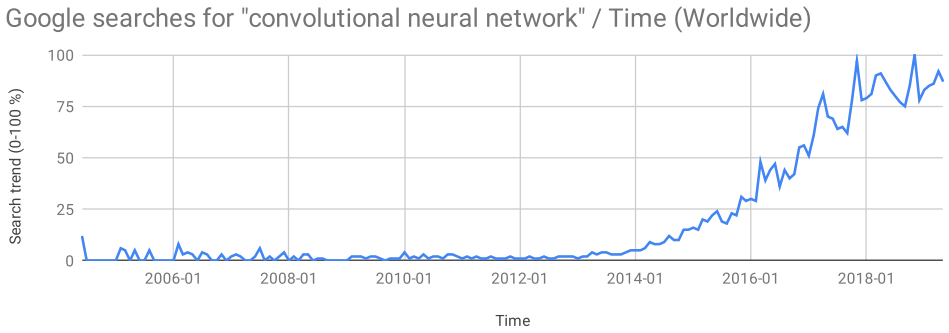

Figure 1.1: Google searches for "convolutional neural network" since 2004 (Data source: Google Trends (https://www.google.com/trends)).

In 2014, the CNN was successfully applied to super-resolution for the first time [4]. The network consisted of only three layers: a feature extraction layer, a non-linear mapping layer and a reconstruction layer. The network, called *Super-Resolution Convolutional Neural Network* (*SRCNN*), approximated the complex mapping between a low resolution input image and a high resolution target image and achieved state-of-the-art performance compared to the traditional methods, such as B-spline interpolation. This marked a shift in super-resolution research towards deep learning methods. Since then, many efficient network architectures and techniques have been suggested. In this report, the architecture is defined as the entire network structure and techniques is used as a broad umbrella term for various improvement methods for the network, such as applying certain layer types and operators (more about this in the background chapter).

A year later, the *Fast Super-Resolution Convolutional Neural Network* (*FSRCNN*) was proposed. This was the first network to use the *transposed convolutional operator* (also known as deconvolution) which is a backward convolutional operator. Thus, instead of upsampling the input image it downsamples it to a lower dimension. This made it possible to train the model on small images and scale up the images in the end [5]. This idea was improved with the *Efficient Sub-Pixel Convolutional Neural Network* (*ESPCN*) in which the deconvolutional layer was substituted with the sub-pixel-convolutional layer; a similar but more efficient upsampling technique [25] which now is the standard. This method expands the input in channel domain (depth) instead of in spatial domain (width and height) and then reorganizes the output in accordance to a given mapping criteria.

Kim et al. presented a learning method for training very deep networks called residual learning [13] in 2015. The model, called *Very Deep Super Resolution* (*VDSR*), used shortcut connections which are connections that skips one or more layers. These were used to build a so-called residual block. Instead of learning the true mapping between the low resolution image and high resolution image directly, with residual learning the network tries to learn the residual (difference) between the two. This was expanded further when researchers proposed the *Residual Convolutional Neural Network* (*ResNet*), which used residual learning to build an even deeper model containing 152 layers [7]. This network has been a popular foundation for new networks since. Although this model initially was developed for image classification (as many are) it has been popular as a base for super-resolution as well.

Mark Sandler et al. released a model type particularly focused on being small and fast in 2016. Two versions have been proposed: *MobileNetV1* and *MobileNetV2* [9][24]. These models' main purpose have been visual tasks like image classification and image detection. The latter network introduced a new interesting concept: the inverted residual block with linear bottlenecks. This was a residual block with shortcut connections between the thin feature maps (meaning low number of feature maps) instead of the thick (meaning high number of feature maps) allowing the model to use a lower number of kernels. The linear bottlenecks were used to drastically reduce the computational complexity. Furthermore, it applied the use of depthwise separable convolution which is a factorized type of convolutional operator that divides the convolution into two smaller sub-parts.

After multiple publications had proven the utility of shortcut connections, Huang et al. evolved them by connecting each layer with all preceding layers [10]. By using densely connected layers, the *Dense Convolutional Neural Network* (*DenseNet*) improved feature propagation and reduced the number of parameters significantly. Also, it alleviated the vanishing gradient problem. This is a common problem that may appear when training a network if the

gradient becomes too small, preventing the weights from updating its values. This basically stops the model from improving. The use of densely connected layers was later developed and adapted for super-resolution resulting in the *Super-Resolution Dense Convolutional Neural Network* (*SRDenseNet*) which employed similar ideas [26]. This was shown to be a very computationally effective approach for super-resolution as well.

Xie et al. proposed an efficient aggregated residual transformation block in 2016. This block used cardinality groups and performed the convolution cardinality-wise [28]. The idea was to exploit wideness instead of deepness and performing the convolution in parallel. To do this they used the grouped convolution operation, first proposed in AlexNet in 2012 [16]. This block consisted of three parts: splitting the input feature maps into smaller groups, transforming each group through convolution and aggregating the output. The model, called ResNeXt, came second in the ILSVRC 2016 competition in object classification.

The same year N. Indola et al. published the work of a small network called *SqueezeNet* [11]. The model used blocks with an initial expansion of features followed by a compression of features. The block was divided into two paths, one for gaining information from the spatial domain and one in the channel domain. This proved to work very well, decreasing the overall computational cost and still keeping the accuracy high.

In 2017, Lim et al. won the NTIRE 2017 super-resolution competition with their *Enhanced Deep Super-Resolution Network* (*EDSR*) [18]. This became the new state-of-the-art by a large margin and the model brought a few new ideas. The most important one was a simplification of the blocks by removing batch normalization layers. Batch normalization is used to normalize the weights with the purpose of speeding up training. These were previously used extensively in deep networks but was proved to distort the quality in super-resolution networks. A second improvement was to significantly increase the depth of the output feature space, i.e. the number of feature maps in each layer. This was followed up the year after in the NTIRE 2018 super-resolution competition when the *Wide Deep Super-Resolution Network* (*WDSR*) won and achieved new state-of-the-art performance [30]. Yu et al. proposed an EDSR-like network architecture but with higher efficiency and accuracy. They applied the use of expansion layers before activation functions which enabled a greater degree of information gain. This was combined with compressing the expanded features with linear low-rank convolution. This enabled the network to have very thin residual mappings. They also trained with weight normalization which is a technique used for reparameterization of the weight vectors, allowing higher learning rates.

Another noticeable network in the same competition was the *Cascading Residual Network* (*CARN*) [2]. This was a residual network with a novel cascading mechanism applied both locally between residual blocks and globally between CARN-blocks. This mechanism resembles the densely connected layers seen in SRDenseNet. The authors' focused not only on an efficient structure but also speed which is an area with limited research.

In 2016 Ledig et al. developed a new innovative approach of solving the task of super-resolution. The purpose was to specifically improve recovering fine details, which has been a difficult problem for conventional methods [17]. The researchers developed a *super-resolution general adversarial network* (*SRGAN*) and a perceptual loss function. Instead of using a pixel-wise loss which is the standard approach the perceptual loss measures the Euclidean distance in high-level feature space between the network being trained and a pre-trained network. This approach was successful and showed great improvement of recreating photo-realistic images which are visually more pleasing.

1.4 Layout

This report is divided into 6 chapters. This chapter contains the introduction to the topic and a brief overview over previous research. Chapter 2 is the background chapter including all necessary theory. It specifically covers the explanation of all the various techniques mentioned here in the previous research section and which later will be used in the evaluation. Chapter 3 explains the method in detail. Here is information about the general approach, evaluation, design, data set and training described. Thereafter is the result presented in Chapter 4. The result chapter is divided into an evaluation part and a design part. The former contains the empirical data gathered from the evaluation. The latter contains the proposed network, including quantitative metrics regarding general performance and qualitative data in the form of the generated images. This is followed up in Chapter 5 by a discussion where different aspects are analyzed, such as efficiency, training and applicability. Finally, in Chapter 6 the conclusion is presented.

Chapter 2
Background

This chapter contains the theory. This includes relevant terminology and background information of which this thesis work builds upon. This part includes an initial overview over the topic and a step-wise walk-through towards more specific techniques for building small and efficient architectures.

2.1 Single Image Super-Resolution

Single image super-resolution (SISR) is a technique used to obtain a high resolution image from a single low resolution image (for convenience super-resolution and single image super-resolution will be used interchangeably). This is a very challenging problem since the low resolution image could correspond to various high resolution images [29].

There are three different categories of approaches to solve this problem. The first are interpolation-based methods such as B-spline interpolation. This simple traditional method has been widely used for a long time. The advantage of this method is speed but it severely lacks in accuracy [29].

The second approach is based on reconstruction. These methods rely heavily on a priori knowledge regarding imaging properties to specify an output space. In general, this approach is superior to the first but is often complex and the performance decreases quickly when the scale factor increases [29].

The third method are learning based methods. Learning-based methods have been gaining more attention in recent years, see for instance the work by He et al. (2015), Huang et al. (2016) and Lim et al. (2017) and is now dominant in research due to their general superior performance [29]. The idea is to use a mathematical approach by exploiting the statistical relations between low and high resolution images. To analyze these mappings a data set of low resolution input images and corresponding high resolution targets is needed. Generally, different classes of machine learning algorithms are applied to find these mappings. The most successful approach in super-resolution has been the use of neural networks. A simple

schematic overview of the process is shown in Fig. 2.1 where the network takes an input image and upscales it to a higher resolution. This learning-based method with the use of neural networks is what will be used in this work. Approaching the problem from an efficiency standpoint, defined as a network with low practical metrics (model size and run time) and high visual metrics (the image reconstruction quality), leads to the question of how to build these models. There are two main approaches: focus on an effective architectural design or apply transfer-learning. Transfer-learning is a way of transferring the knowledge obtained by one model (e.g. a large one) to another model (e.g. a small one). This work is dedicated to the first approach since this attacks the problem explicitly.

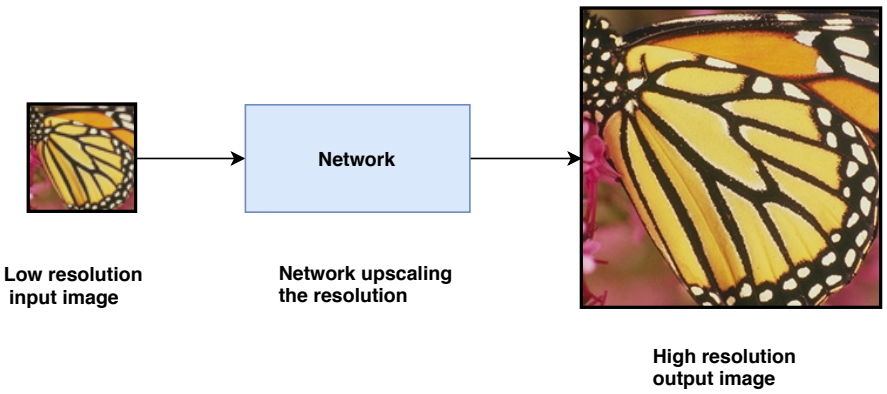

Figure 2.1: A schematic overview over super-resolution where a network takes a low resolution image as input and outputs a corresponding high resolution image.

2.2 Machine Learning

Machine learning (ML) is a branch of the broader field of *artificial intelligence* (AI). The core concept of machine learning is to let the models learn by themselves and without being explicitly programmed on how to solve a problem. The fundamental resource for such a model is data. These mathematical models use algorithms and statistics to find patterns and relations in the data to gain knowledge. Thus, the quality of the data is crucial to build excellent models generalizing well to new data.

There are different categories of machine learning methods, such as *supervised learning*, *unsupervised learning* and *reinforcement learning*. In this work supervised learning will be the approach since this is what super-resolution mainly relies on. Supervised learning can be divided into two separate stages: training and prediction. For the training stage, a *labeled* data set is required. Each data sample in such a set consists of a pair, where each input has a corresponding output class. The model is trained to find the mappings between the two to be able to predict from a given input what the correct output should be. Formally this can be defined as given a data set $\boldsymbol{D}$ with n samples in the form of $\{(\vec{x_1}, y_1), ..., (\vec{x_n}, y_n)\}$, where $\vec{x_i}$ is the input feature vector and y_i the corresponding class, the model searches to learn the approximator function $f(\vec{x}) = y$. An example of this could be a cat breed classifier, where the input $\vec{x}$ is an image of a cat and the corresponding class y is the breed. Once the model has been trained it can be applied to new data for prediction [6].

2.2.1 Loss Function

The *loss function* (or optimization function) is a core component of a machine learning model and it is what the model uses to improve the performance. By minimizing the loss function the difference between the model prediction and correct prediction decreases, leading to better accuracy. There are many different types of loss functions and one of most widely adopted for imaging tasks is the *mean squared error* (***MSE***) [29]. The mathematical definition looks like this:

$$\text{MSE} = \frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}[I(x,y) - J(x,y)]^2 \tag{2.1}$$

where I is the target image and J the predicted image, both with the spatial dimensions M x N (width and height). It is a pixel-wise measurement of the differences between the two images.

A similar loss function is the *mean absolute error* (***MAE***). Instead of squaring the pixel differences we compute the absolute value between them. It is defined as follows:

$$\text{MAE} = \frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}|I(x,y) - J(x,y)| \tag{2.2}$$

where the variables are defined as in Eq. 2.10. The main difference between MSE and MAE is that since the former squares the difference between the predicted value and the correct, the relative error has a higher influence on it (if it is larger than one). This makes it useful if large errors are regarded as much more costly than equivalent small errors. If this is not the case MAE is more robust since it will not being affected as much.

2.2.2 Artificial Neural Network

An *artificial neural network* (ANN) is a machine learning framework heavily inspired by the human brain. It consists of artificial neurons which are connected in layers to enable information to flow through the network. At each neuron a computation is made with the use of an *activation function*, which processes the input and generates an output. With the use of a non-linear activation function the network can learn complex non-linear mappings. Each neuron also has a weight assigned to it. This weight represent the importance of the node in the structure and will be adjusted during training to adapt the network to the given task. The network can therefore be seen as a universal approximator function, by updating the weights any sophisticated function can theoretically be mapped - this is the main power of the ANN. The network shown in Fig. 2.2 has four layers and the data flows from the input neurons to the left, through the hidden layers in the middle, to the output neuron on the right. Using a single output neuron like in this case, the network produces a final binary output prediction [22].

2.2.3 Convolutional Neural Network

A *convolutional neural network* (CNN) is a certain type of ANN, particularly suitable for solving computer vision tasks. The architecture is inspired by the visual cortex in the brain, which

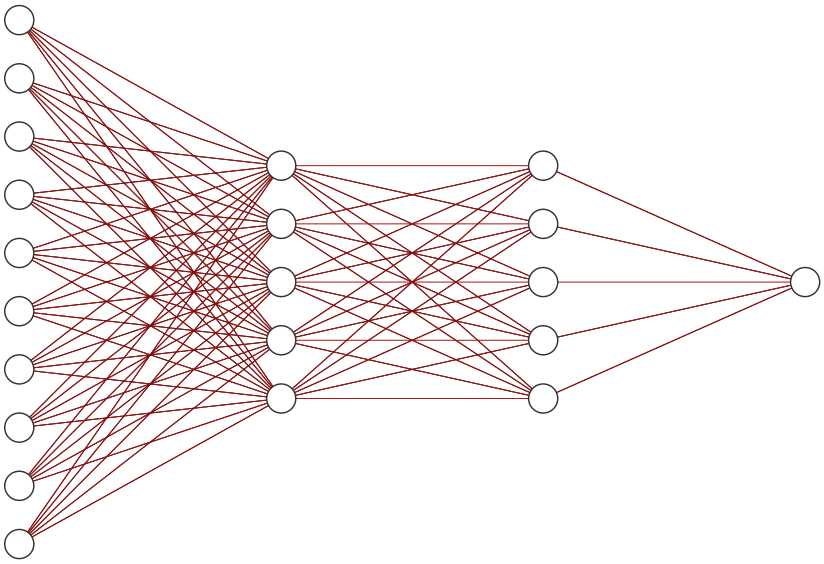

Figure 2.2: An artificial neural network with four layers, one input layer with 10 neurons, two hidden layers with 5 neurons each and an output layer with 1 neuron.

organizes the neural connections to only respond to input in a restricted area of the visual field (the receptive field). This allows the network to have a very efficient shared-weight structure [6].

The most important part of a CNN is the *convolutional* operator (the term operator will be used as a general term for describing various types of convolutions). Convolution is a mathematical operation which produces a new integrable summation function by letting two input functions interact. The interaction can be seen as geometrically sliding one over the other. The continuous version can be defined as:

$$\mathrm{h}(t) = (f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)\,\mathrm{d}\tau = \int_{-\infty}^{\infty} f(t-\tau)g(\tau)\,\mathrm{d}\tau \tag{2.3}$$

where f and g are two integrable functions, t is the time, τ is the shift and $h(t)$ the new integrable function. With images the discrete version is applicable and can formally be expressed as follows:

$$\mathrm{h}(n) = (f * g)[n] = \sum_{m=-\infty}^{\infty} f[m]g[n-m] = \sum_{n=-\infty}^{\infty} f[n-m]g[m] \tag{2.4}$$

where f and g are two discrete functions and $h(n)$ the convoluted result. An example of the operation is shown in Fig. 2.3. In this figure the first function can be seen as the input image, the second function as the *kernel* (also known as *filter*) and the result is the output *feature map*.

In a CNN this operation is computed multiple times at each convolutional layer with different kernels. This allow the network to get a feature representation of the image focusing on different parts, where one kernel could be designed as a filter for certain edges and another for specific lines. Together these feature maps can be seen as the knowledge gained by the network at each layer. For each layer there is also an increasing level of abstraction.

The perhaps second most important part of a CNN is the activation function. This function is employed in between convolutional layers with the purpose of, as previously men-

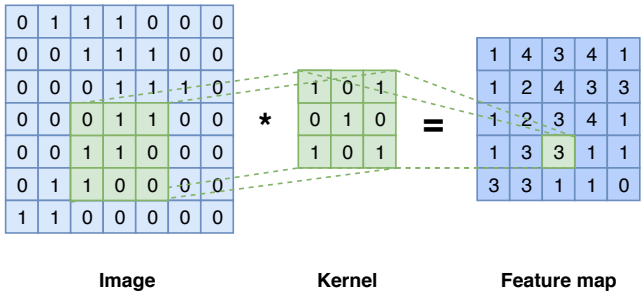

Figure 2.3: A visualization of how the convolutional operator works given an image I, a kernel K and the output feature map $I * K$.

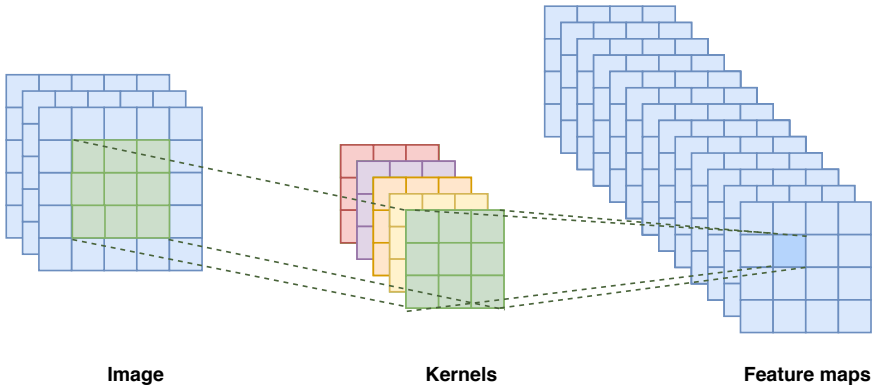

Figure 2.4: A convolutional operation in a CNN: an input image with three channels (RGB) which is passed through a convolutional layer with 5 different kernels and outputs 15 feature maps.

tioned, allow the network to learn complex non-linear mappings. This is crucial for successfully upscaling images to higher resolutions.

2.2.4 Activation Function

The activation function defines the output of a neuron by taking the input and mapping it to a defined output space. There are many different activation function, each with their own advantages and disadvantages. Some of the most popular are the *binary step function*, the *logistic function* and the *rectified linear unit* (ReLU). ReLU has gaining in popularity in recent years due to its speed and was in 2017 the most popular one for deep neural networks. The ReLU is defined as:

$$f(x) = x^+ = \max(0, x) \tag{2.5}$$

hence, it outputs zero for negative input values and x otherwise. A non-linear activation function such as the ReLU allows the model to learn complex non-linear mappings.

2.2.5 Depthwise Separable Convolution

A standard convolution is applied to both the spatial dimensions and the channel dimension simultaneously. Given an image of height H, width W and N color channels, the standard convolutional operator with M kernels of size $K \cdot K$ and M output channels would have the computational cost:

$$C = H \cdot W \cdot N \cdot K^2 \cdot M \tag{2.6}$$

This operator is thus proportional to H, W, N, M and $\sqrt{K}$.

Depthwise separable convolution is another type of convolutional operator. It is a factorized convolution which divides it into two steps: one for the spatial domain (depthwise) and one for the channel domain (pointwise). First the depthwise convolution performs the operation on each input channel independently. Then, a 1x1 pointwise convolution is applied that blends the channels by computing linear combinations between them (a 1x1 pointwise convolution meaning a convolution with a kernel size of 1x1) [24]. Thus, the first step can be regarded as a spatial filtering for the width and height and the second as an depthwise expansion in channel domain.

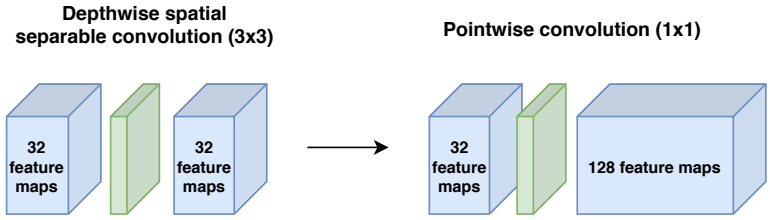

Figure 2.5: Depthwise separable convolution visualized in two steps. First in the spatial domain (left) and then in the channel domain (right).

The computational cost for the depthwise convolution is:

$$C_1 = H \cdot W \cdot N \cdot K^2 \tag{2.7}$$

and for the pointwise convolution:

$$C_2 = H \cdot W \cdot N \cdot M \tag{2.8}$$

giving the total cost as the sum of the two:

$$C = C_1 + C_2 = (H \cdot W \cdot N \cdot K^2) + (H \cdot W \cdot N \cdot M) = H \cdot W \cdot N \cdot (K^2 + M) \tag{2.9}$$

where $M \gg K^2$. With common values for the output channel, e.g. $M = 64$ and $K = 3$, the computational cost decreases with a factor of about $1/8$ compared to the standard operator. It should be noted that with such a big decrease in computational complexity, the drawback of this operation is often a reduction in accuracy. This trade-off between computational complexity and visual performance is very common.

An example of this operation can be visualized in Fig. 2.5 where the left part is the depthwise convolution and the right part the pointwise convolution. In the figure the parts

coloured in blue are the feature maps, i.e. the representation of the image at that stage, and the parts coloured in green are the convolutional layers.

2.2.6 Grouped Convolution

The *grouped convolution* is another efficient operator which is quite similar to the depthwise separable convolution. In grouped convolution the channels of the feature maps are divided into groups and the operation is performed group-wise. With the same previously defined variables in Section 2.2.5 above, the computational cost with a group size of G is:

$$\mathrm{C} = \frac{H \cdot W \cdot N \cdot K^2 \cdot M}{G} \tag{2.10}$$

where $H \cdot W \cdot N \cdot K^2 \cdot M \gg G$. This results in an approximate $1/G$ decrease in computational cost compared to the standard operator [2]. But just as with depthwise separable convolution the drawback is a possible decrease in visual performance.

2.2.7 Linear Low-Rank Convolution

Linear low-rank convolution is another form of convolutional operator that is based on a specific type of factorization. The main idea is to factorize a large convolutional kernel into two smaller. This operation stacks a 1x1 pointwise convolution to reduce the number of feature maps, with a 3x3 convolution for the spatial feature extraction. This operator has specifically been shown to reduce the model size effectively, enabling the network to be deeper or wider with an equal parameter budget. Therefore, it basically functions as a form of bottleneck layer (more about bottleneck layers in Section 2.2.9 below) [30].

2.2.8 Upsampling Layer

To upscale the resolution of an image, the model must apply an upsampling layer to increase the spatial dimensions. One such layer is the *transposed convolutional* layer. This is basically an inverted convolutional operator, mapping the input feature from a lower dimension to a higher dimension. It does this by first expanding the input in the spatial domain and then performing a normal convolution, as shown in Fig. 2.6. The most popular technique is the more efficient *sub-pixel convolutional layer* [25], shown in Fig. 2.7. The sub-pixel convolution does not explicitly increase the spatial dimensions of each feature map as in the transposed layer. Instead it exploits the channel domain by expanding it and then restructure the output through a mapping criteria. The input is the low resolution feature map and the output after the mapping is the high resolution feature map. This decreases the total computation compared to the transposed convolution.

2.2.9 Bottleneck Layer

A *bottleneck layer* is used to compress the feature representation to a lower dimension [24], or more simply put: reduce the number of feature maps. This can be achieved by adding a convolutional layer with fewer feature maps as output than input. The purpose of this

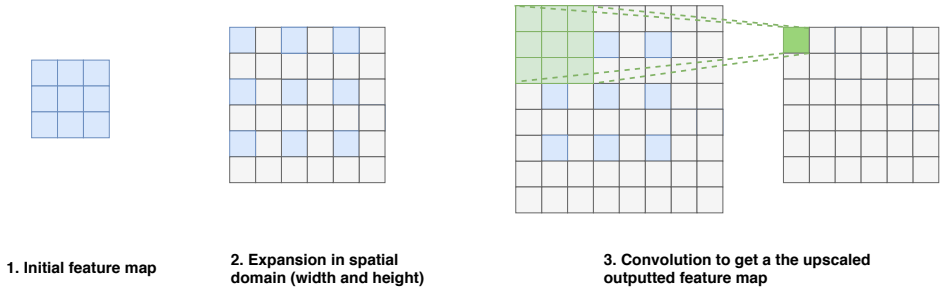

Figure 2.6: Transposed convolution.

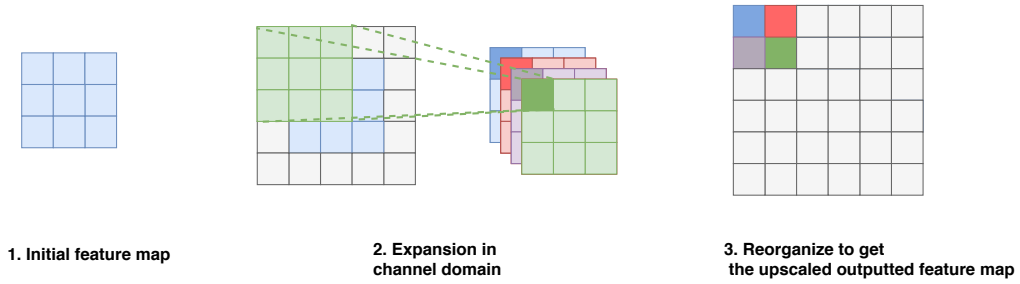

Figure 2.7: Sub-pixel convolution.

operation is to decrease the computational complexity and make the network more efficient. The drawback being that it also leads to less information being gained, leading to a possible loss in accuracy.

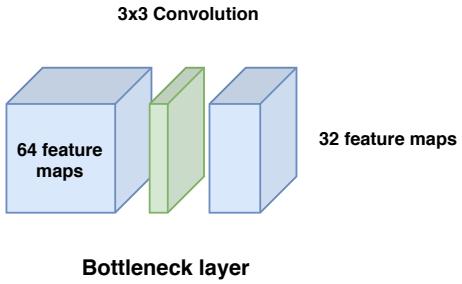

Figure 2.8: A bottleneck layer reducing the number of feature maps.

Another type of bottleneck layer is the 1x1 pointwise convolution. This is a convolutional layer with a kernel size of 1x1. This means that it does not perform any feature extraction in the spatial dimension but only the channel dimension. This operator does not reduce the number of feature maps but greatly decreases the computational cost by the small kernel size and can therefore be regarded as a computational bottleneck layer. In 2.8 a standard bottleneck layer is shown, here reducing the number of feature maps by half.

2.2.10 Expansion Layer

An *expansion layer* is basically the opposite to a bottleneck layer. Instead of compressing the feature representation to a lower dimensional space, an expansion layer increases it to a

higher dimensional space [30].

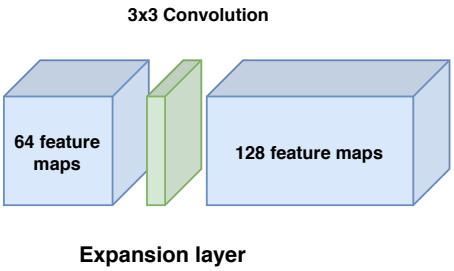

Figure 2.9: A 3x3 convolutional layer with a 2 times expansion factor.

See Fig. 2.9 for a visualization where the feature dimension increases by a factor of 2, i.e. the number of feature maps, through a 3x3 convolutional expansion layer (a 3x3 convolution meaning a convolution with a kernel size of 3x3). The advantage of using a higher number of feature maps is that it allows the network to compute more representations and gain more information which leads to improved visual performance. The drawback is that it increases the computational complexity, hence the expansion factor is often employed in conjunction with some sort of bottleneck layer.

2.2.11 Residual Block

The *residual block* (also referred to as just a block) are for many networks the foundational building stone. A block can be defined as a small coherent structure of multiple layers and activation functions. These are stacked consecutively multiple times forming the main structure of the network. While standard blocks only transform the input to output, a residual block forwards both its inputs and its outputs. The input thus skips a layer. This type of connection is called a *shortcut connection* or (*skip connection*) and is used extensively in deep networks. With an input x (identity mapping) and the true mapping $y(x)$, the standard block would try to learn the mapping function $h(x) = y(x)$. In a residual block, it would instead try to map $h(x) = r(x)$ as:

$$r(x) = Output - Input = y(x) - x \tag{2.11}$$

In other words, it tries to learn the residual $r(x)$ - the difference between the input and output. The benefit of this approach is that it has been observed to enable deeper networks as well as increased accuracy.

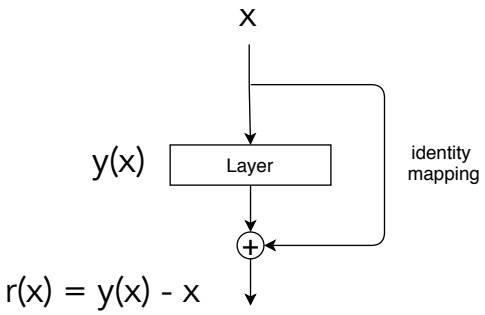

Figure 2.10: A residual layer.

One reason for this is that for a small residual $r(x)$, the true output $y(x)$ is approximately equal to the input x which is easy to compute [7]. A residual layer can be seen in Fig. 2.10. But this concept can also be applied to a block structure. These so called residual blocks are used extensively and an example of such a block can be seen in Fig. 2.11. This block has 64 feature maps and contains a convolutional layer, a ReLU activation function, a second convolutional layer and a residual adding the input and the output together. In this figure the information flows from left to right.

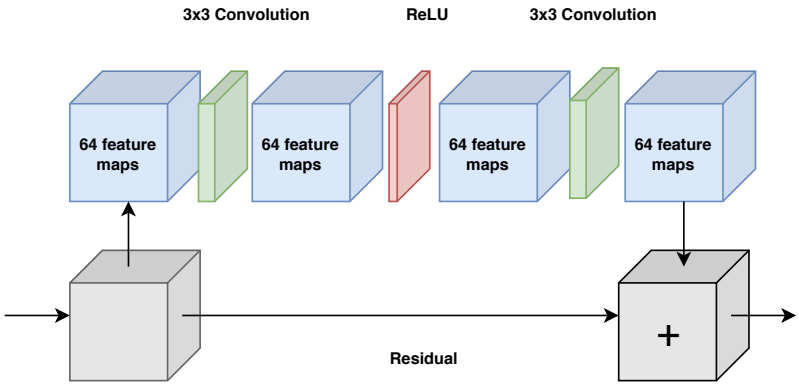

Figure 2.11: A standard residual block.

2.2.12 MobileNet Block

The MobileNet block used *inverted residuals* with *linear bottlenecks*. This block combines bottleneck layers and expansion layers. It takes a low-dimensional input representation which is expanded to a higher dimension by an 1x1 expansion layer. It is then passed through a 3x3 depthwise separable convolution and is subsequently projected back to the low-dimensional input representation by a 1x1 pointwise convolution. In this block the shortcut connections are between the low dimensional bottleneck layers.

For a comparison, take a look at the standard residual block above again, shown in Fig. 2.11. This block has a convolutional layer, an activation function and a second convolutional layer. It keeps the same dimensionality for the output feature maps throughout the block, in this example 64. Another slightly more efficient block structure is the residual bottleneck block shown in Fig. 2.12. This structure can be described as dimensionality-wise moving from high → low → high.

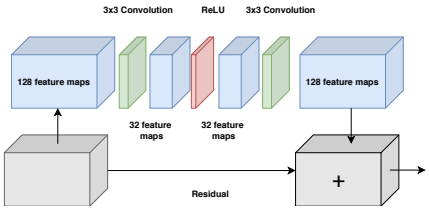

Figure 2.12: A residual bottleneck block.

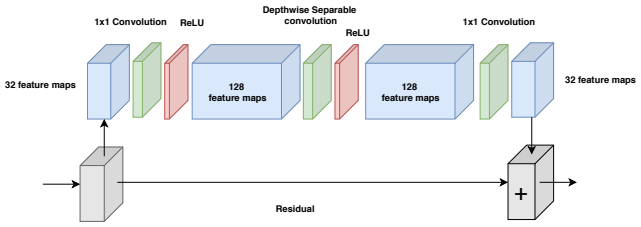

Figure 2.13: A MobileNet based block.

This can now be compared to the structure seen in MobileNet, the inverted residual block where the residuals are between the thin feature maps (low dimension) instead of the thick (high dimension). This gives rise to a low → high → low dimensional design as shown in Fig. 2.13. This block was specifically designed for efficiency regarding both computation and memory consumption [24]. By using this structure it was possible to keep a high visual performance with a lower number of kernels.

2.2.13 WDSR Block

The *Wide Deep Super-Resolution Network* (WDSR) uses two unique block types: WDSR A-blocks and WDSR B-blocks, which are similar to the inverted residual block with residuals between the thin feature maps but do not employ depthwise separable convolution [30]. Instead the block uses standard convolution and expansion layers. The two blocks are visualized in Fig. 2.14 and Fig. 2.15.

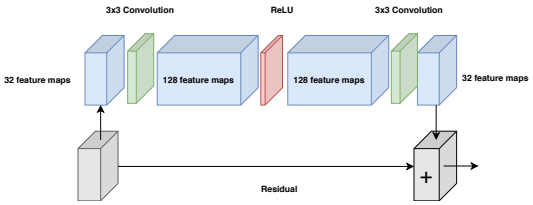

Figure 2.14: WDSR A-block.

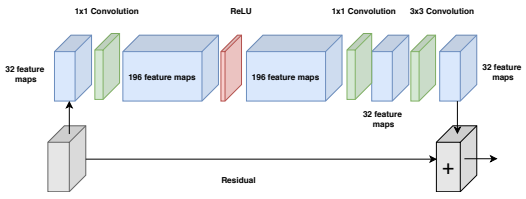

Figure 2.15: WDSR B-block.

The A-block uses a 3x3 expansion layer with a factor of 4, a ReLU activation function and a 3x3 bottleneck layer. The B-block uses a 1x1 expansion layer with a factor of 6, a ReLU activation function and a linear low-rank convolution which factorizes the second layer into a 1x1 convolution and a 3x3 convolution. The expansion layers were only used before the activation functions which proved to increase the visual performance effectively. By applying the linear low-rank convolution in the B-block it was also demonstrated that even greater expansion factor can be used (6x-9x) leading to superior accuracy with the same parameter budget. Both blocks also apply a residual mapping.

2.2.14 SRDenseNet Block

In *Super-Resolution Dense Convolutional Neural Network* (SRDenseNet) a customized dense block was developed. This is a block type with *densely connected layers*, which incorporate features from multiple layers by progressively adding shortcut connections to subsequent layers. Each 3x3 convolutional layer gets and increasing amount of input for each level and the various convolutions are separated by a ReLU activation function. An overview over a block with densely connected layers can be seen in Fig. 2.16, where the black arrows are the shortcut connections. This technique proved to increase the visual performance at a low cost in practical performance. It also enabled the network to not be as deep.

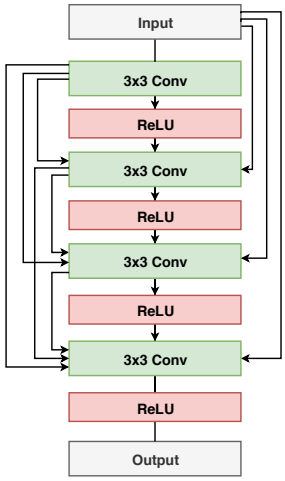

Figure 2.16: A SRDenseNet based block with 4 convolutional layers.

The theory behind the design is that features from different layers reflect various abstraction levels which can be exploited to gain more information. Furthermore, multi-level shortcut connections make the propagation of information faster throughout the network. It also alleviates the *vanishing-gradient* issue. This is a problem occurring in networks employing gradient-based learning methods. If the gradient becomes very small it prevents the weights in the network to change and as a consequence it can stop the model from converging [26].

2.2.15 ResNeXt Block

The ResNeXt block is based on the fact that wider networks are computationally more effective than deeper networks. The philosophy was to use multiple convolutional paths in parallel, which was proven to be a successful approach enabling an efficient block structure.

The researchers used a three step operation for each block in what is called aggregated residual transformation: splitting the input feature maps into smaller groups, transforming each group through convolutional operations and finally aggregating the output to restore the input dimensionality [28]. This can be implemented in multiple different ways. One option is to use grouped convolution by performing the convolution in small independent channel subgroups. Another is to use multiple smaller standard convolutions in parallel. An overview over a ResNeXt based block can be seen in Fig. 2.17. This block consists of three paths and each path is divided into 5 layers. First a 1x1 pointwise convolution and a

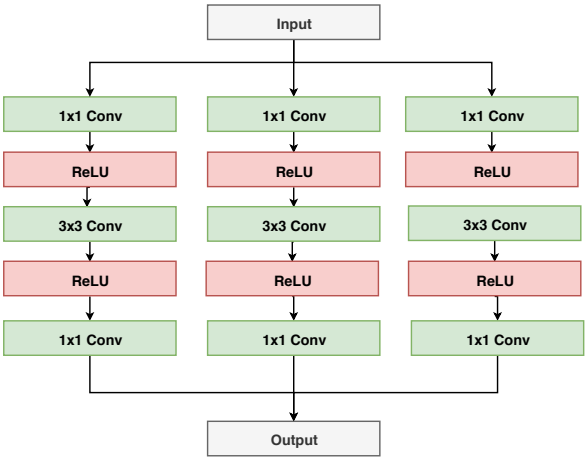

Figure 2.17: A ResNeXt based block with 3 groups.

ReLU activation function. Then a 3x3 convolution for spatial feature extraction followed by another ReLU activation function before the final 1x1 pointwise convolution. The three paths are then added together to get the output.

2.2.16 SqueezeNet Block

The SqueezeNet is another type of network which also was developed for efficiency, specifically regarding size. The block used in the network consists of three layers. First a bottleneck layer with a light 1x1 pointwise convolution. This layer reduces the number of feature maps drastically. Next, two expansion paths: one 1x1 pointwise convolution and one 3x3 convolution. These were used to restore the initial feature dimension of the input. After each operation a ReLU activation function was applied. Finally, the two paths were concatenated to get the final output [11]. An overview of the block is presented in Fig. 2.18.

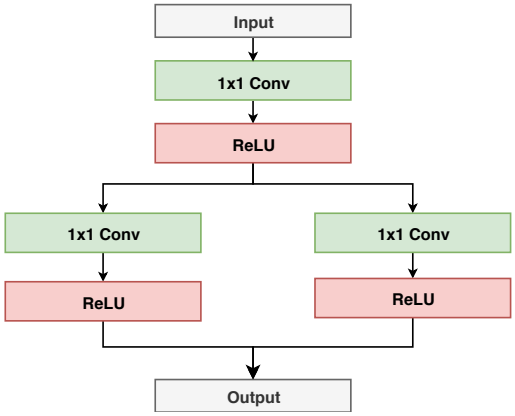

Figure 2.18: A SqueezeNet block.

2.2.17 Cascading Mechanism

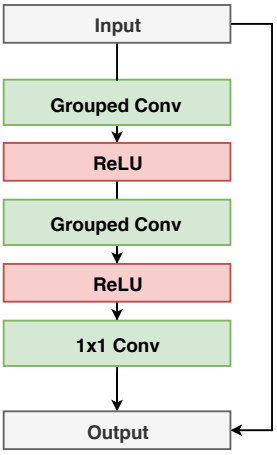

Figure 2.19: A CARN based residual block.

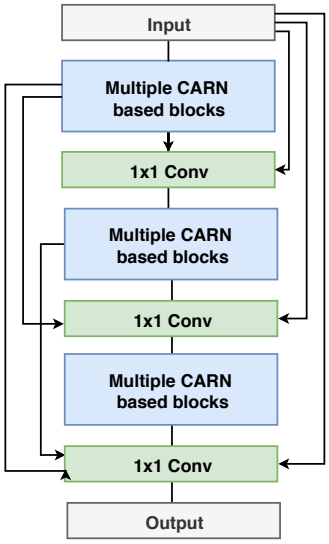

Figure 2.20: A global cascading mechanism between blocks.

The *cascading mechanism* seen in the *Cascading Residual Network* (CARN) is quite similar to the dense layers in SRDenseNet. It is a technique to incorporate features from multiple layers to enable feature reusage. Unlike the dense block the cascading mechanism takes all the input, concatenates them and passes it through a 1x1 pointwise convolution. The mechanism can be applied not just locally inside blocks but also globally between blocks, enabling even more feature information flow between different abstraction levels [2]. This type of cascading was proven to be useful for speed, partly due enabling the use of fewer blocks and still maintaining a high accuracy. An overview over the CARN-block with two grouped convolutions and a 1x1 bottleneck can be seen in Fig. 2.19 and a schematic overview over the cascading mechanism between blocks can be seen in Fig. 2.20.

2.2.18 General Adversarial Network

General adversarial network (GAN) is a type of network consisting of a generative part and a discriminative part. This type of network can be applied to various tasks but are especially good at image generation tasks. The core concept is to let the networks have two opposing goals, which leads to a competition between them. The generator network tries to generate images as similar as possible to the real images while the discriminator network tries to distinguish the generated images from the real ones. The training objective of the generator is to maximize the discriminator loss, i.e fooling the discriminator. The discriminator on the other hand, optimizes its objective function to get as good as possible at differentiating the generated (fake) images from the real images. Thus, the generator will generate images as closely resembling the real as possible [17].

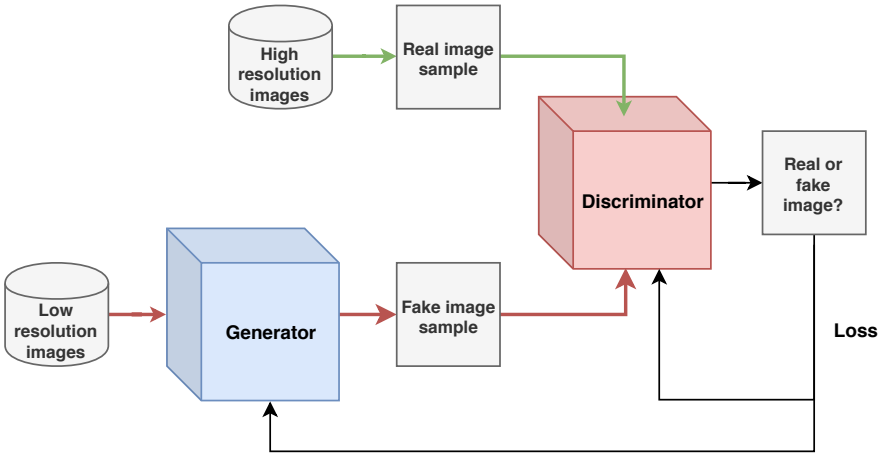

Figure 2.21: A schematic overview of a general adversarial network.

An overview of a GAN can be seen in Fig. 2.21. In this example the generator takes training data in the form of low resolution images and generates high resolution images. These generated images are fed to the discriminator along with the real high resolution images where the discriminator tries to differentiate them.

Content Loss and Perceptual Loss

In 2017, a new type of loss function was developed specifically design to be part of a general adversarial network [17]. Instead of operating in pixel-space as the standard loss functions (using the pixel-wise Euclidean distance), it operates partly in feature-space. This loss function, called *perceptual loss*, consists of two parts: a *content loss* and an *adversarial loss*. For the content loss a pre-trained CNN (*VGG19*) was used for evaluating the performance by extracting high-level features from images produced by this network and comparing them to the images produced by the generator. The idea was to not focus on per-pixel similarity but rather the high-level perceptual similarity.

The definition of the content loss can be stated as:

$$l^{SR}_{VGG/i,j} = \frac{1}{W_{i,j}H_{i,j}} \sum_{x=1}^{W_{i,j}} \sum_{y=1}^{H_{i,j}} (\phi_{i,j}(I^{HR})_{x,y} - \phi_{i,j}(G(I^{LR})_{x,y})^2 \tag{2.12}$$

where $\phi_{i,j}$ is the feature map at convolution layer j and before the max-pooling layer i in the VGG19 network, I^{HR} the high resolution image and I^{LR} the low resolution image. The loss is the Euclidean distance between the features of the high resolution image I^{HR} and the generated image $G(I^{LR})$, both extracted from the pre-trained network. By using this loss function the model tries to minimize the high level feature representations of the target and the prediction.

The second part of the perceptual loss was the *adversarial loss*. This loss function was used to encourage the generator network to produce images which fools the discriminator network, defined as:

$$l^{SR}_{GEN} = \sum_{n=1}^{N} -\log D_{\theta_D}(G_{\theta_G}(I^{LR})) \tag{2.13}$$

Here is N the number of image samples, $G_{\theta_G}(I^{LR})$) the generated image from the generator (parameterized by the weight vector θ_G) and $D_{\theta_D}(G_{\theta_G}(I^{LR}))$ is the probability of the discriminator (parameterized by the weight vector θ_D) falsely predicting the generated image as the real high resolution image.

The perceptual loss is the combination of the content loss and the adversarial loss. This loss works as balance between the two and can be stated as:

$$l^{SR} = I^{SR}_{VGG/i,j} + 10^{-3} \cdot l^{SR}_{GEN} \tag{2.14}$$

For reaching a good balance, the 10^{-3} factor was experimentally found to be most successful.

2.2.19 Network Interpolation

Network interpolation is used to build a model based on a combination of multiple input models. Its purpose is to create a balanced model; a trade-off between the trained networks [27]. Let's say we have a model A trained with mean absolute error as loss function and model B trained with perceptual loss as loss function. Knowing that different loss functions lead to different results it is possible to reach an optimal trade-off between the two. An interpolation network θ_I can be computed with the interpolation parameter $\alpha = [0, 1]$:

$$\theta_I = (1 - \alpha) \cdot \theta_A + \alpha \cdot \theta_B \tag{2.15}$$

By adjusting the interpolation parameter α the network can easily be primed towards the weights of either input network to find the most perceptually pleasing result. This of course requires the input models to be exactly alike in their structure, otherwise the weight vectors would represent different abstraction levels. A benefit of this approach, rather than training a network containing a combined loss, is that it is both simpler and faster, since the technique allows evaluation of different interpolation values instantly without extra training required.

Chapter 3
Method

In this chapter the method is described. It contains the approach; defining what work will be done and specifically how it will be done. The approach was a two-step process with an evaluation part and an architectural design part. It includes information about the training, the data sets and the software and hardware setup. Finally, the evaluation is described explaining both what methods and metrics will be used as well as an analysis of its limitations.

3.1 Approach

The main objective of this work was to:

(a) Evaluate efficient neural network architectures to gain knowledge of what techniques are successful.

(b) Design a small and efficient model based on the result from the evaluation and investigate possible customizability options to prime the model to different visual objectives.

This work can therefore be divided into a two main parts: evaluation and design.

3.1.1 Evaluation

The first step of the evaluation was implementing a network to use as a base network. This network was used to evaluate the different techniques. It was based on the WDSR A network but adapted to a very low parameter budget of 300 k parameters. It was compressed by reducing the depth of the network, i.e. the number of residual blocks. WDSR was chosen since it is the current state-of-the-art and delivers great efficiency and general performance. This network is referred to as WDSR A-XS (XS referring to extra small).

The second step was to evaluate the efficiency of various aspects. For each aspect, a model was implemented and trained for a 2x upscaling factor with WDSR A-XS as foundation. Thus, all evaluated models had identical first and last parts to make the assessments more comparable, i.e maximize the number of constant variables. This includes the initial convolutional layer, the final upsampling layers and a global residual connection. The base architecture is shown in Fig. 3.1 where the various techniques were implemented in the main part of the network coloured in blue. The model was then evaluated for its size and as a trade-off between the practical and visual metrics. The purpose was to find an optimal balance between the two, i.e high efficiency with low run time and parameter count but high visual quality.

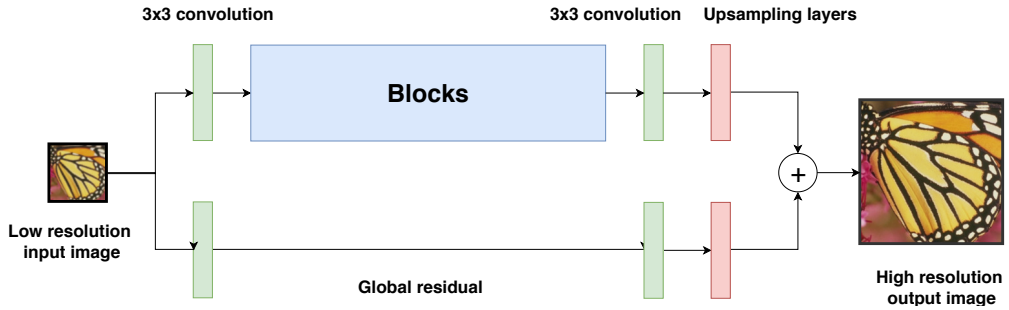

Figure 3.1: The WDSR base architecture.

This assessment included looking into a wide variety of architectural design choices. These are all listed below and for more implementation details see Appendix A.1.1.

General Design

1. Depth (number of residual blocks)
2. Kernels (number of kernels and kernel sizes)
3. Residual blocks or kernels (depth vs width)
4. Activation functions (in each block)
5. Residuals (locally and globally)

Layer Types

1. Expansion layers for various factors
2. Bottleneck layers for various factors

Convolutional Operators

1. Depthwise separable convolution
2. Grouped convolution
3. Linear low-rank convolution

Efficient Residual Blocks

1. Block structure (dimensional shape)
2. WDSR blocks
3. MobileNet block
4. SRDenseNet block
5. ResNeXt block
6. SqueezeNet block
7. Cascading block

These were chosen after the literature study as being highly relevant for building efficient super-resolution networks. The blocks tested are from existing networks, all six being based on quite different design philosophies. The blocks evaluated in this work were not identical to the one proposed in the papers but customized to fit into the WDSR network base structure. They were also adapted to the very low parameter budget and updated in accordance with the latest research as of spring 2019 (such as removing batch normalization layers). This was done to make the blocks more comparable.

3.1.2 Design

The second part of this thesis was the design. Based on the evaluation result and the gained knowledge, we developed an efficient network architecture using the most promising techniques. This network was then evaluated both quantitatively to other models and qualitatively with the generated images. For this assessment multiple networks were implemented based on the previously analyzed blocks. A few extra networks were also built, such as applying the cascading mechanism to the SRDenseNet based model and replacing grouped convolution with standard convolution in the ResNeXt model. The proposed small model was also compared with much larger super-resolution models. The last two winners of the global NTIRE super-resolution competition was used for this purpose (EDSR and WDSR), both approximately 15x as large as the proposed one. All assessed models are listed below and for more implementation details see Appendix A.1.2.

1. WDSR
    (a) WDSR A-XS
    (b) WDSR B-XS
2. MobileNet block
    (a) MobileNet XS
    (b) MobileNet S
3. SRDenseNet block
    (a) SRDenseNet A S
    (b) SRDenseNet B S
    (c) SRDenseNet cascading
    (d) SRDenseNet P
4. ResNeXt block
    (a) ResNeXt SC A XS
    (b) ResNeXt SC A S
    (c) ResNeXt SC B XS
    (d) ResNeXt SC B S
    (e) ResNeXt GC cascading
5. SqueezeNet block
    (a) SqueezeNet XS
    (b) SqueezeNet S
6. Cascading block
    (a) Cascading SC
    (b) Cascading SC
7. Large networks
    (a) EDSR
    (b) WDSR A

Finally, as part of the design process we also investigated customizability. We wanted to see to what extent the networks can be adapted for specific applications since different use cases may require different visual qualities. All training for generating the images was done for 4x upscaling instead of 2x to make the differences between the images from the models more distinct. The proposed network was therefore trained with different loss functions to nudge it towards different visual goals; MAE (mean absolute error) for maximizing PSNR (peak signal-to-noise ratio, see Section 3.5.2 below) and content-loss for maximizing feature

similarity. It was also used a generator network in a GAN (general adversarial network) to minimize the perceptual loss. This GAN setup was an adapted light version of the SRGAN-network in [17] with a discriminator employing half the parameter count and significantly lower run time. To find the optimality between these network types a final interpolation network was built and evaluated for different interpolation values.

3.2 Data set

The data set used was the DIV2K data set which is a high resolution diverse set provided by Timofte et al. from ETH Zürich [1][12]. This data set is widely used and one of the standard sets for super-resolution. It consists of 900 RGB images depicting a wide variety of objects at high resolution. Each sample in the set contains three bicubically downsampled versions for each of the high definition images: 2x, 3x and 4x. In this work we used only the 2x and 4x variants. For visual evaluation purposes another small but popular data set called Set14 was also used, mainly due to its wide use in academia.

3.3 Training

The data set was divided into three independent sets: 700 images were used for training, 100 images for validation and 100 images for testing. Each image was divided into smaller patches of size 48x48 for the input and 96x96 for the 2x upscale target and 196x196 for 4x upscale target. The networks were trained on all three image channels (RGB). Each image was also randomly flipped horizontally and randomly rotated by 90, 180 or 270 degrees. This data augmentation was performed to expand the data set and decreases the risk of possible overfitting, i.e. underperformance due to the model starting to incorporate the noise in the training data. All image patches was subtracted with the RGB-mean values for the data set and then normalized between 0 and 1. The images were cropped with the upscaling factor + 6 pixels from the border. All of this was in line with the training setup used in [30]. The preprocessing and training setup used for this work was the WDSR/EDSR adapted Keras version by [15].

When evaluating networks with the use of a specific operator or layer type, each training session was set to 100 epochs. Since these models were relatively small this was in our case regarded as sufficient to evaluate a technique and draw a conclusion on its usefulness (also due to computational and time limitations). For the assessment of the larger models, the training session was 500 epochs and for the comparison with the heaviest state-of-the-art models it was set to 1000 epochs. This was to make sure the larger models got enough training time and not were severely underfitted to the data, i.e. not reaching its full potential. The number of iterations was in all cases set to 500 and the batch size to 16. If the validation score reached a new high, the model was saved. As loss function was MAE used and as optimizer AdamW with $\beta_1 = 0.9, \beta_2 = 0.999$ and $\epsilon = 10^{-8}$. This is an optimizer based on the Adam optimizer, which in itself is an efficient stochastic optimizer [14]. AdamW is very similar but adds weight normalization, which is a way of reparameterizing the weight vectors [19] which has been proven to enable a higher learning rate [30]. The learning rate was initially set to 0.001 and halved every 200 epochs. All hyperparameters were the same for all evaluations.

3.3.1 General Adversarial Network

Training a GAN differs substantially from training a standard network. However, most of the preprocessing remained the same. A few noticeable differences were the removal of data augmentation which was regarded as unnecessary when training an already complex model which is hard to train. An additional tanh activation function was added before the final upsampling layer in the generator since this has been shown to increase the generators accuracy [23]. This is a non-linear activation function that outputs a value between -1 and 1 instead of max(0, input) as with ReLU. Some label noise of 0.05 was also applied to make the training more stable and the generator more robust. This is a way of randomly assigning some noise both to the labels of the generated images and the high resolution images. Finally was the AdamW optimizer replaced with the Adam optimizer for simplicity reasons.

The GAN consists of two networks: the generator and the discriminator. The generator was pre-trained for 500 epochs with MAE as loss function. This was done to speed up the training time. Thereafter, the weights of this model were loaded onto the generator network. For each iteration, the two networks were trained simultaneously with opposite goals. The discriminator was trained on the real high resolution images and the generated images with the objective of correctly classifying the high resolution images as real and the generated images as fake. The generator on the other hand, was trained as part of the GAN structure by taking low resolution images as input and generating high resolution images to fool the discriminator. The discriminator then classified the images as either real or fake as a measurement of how well the generator performed. The GAN was trained with a perceptual loss (for more details see Section 2.2.18 above). The training session was set to 500 epochs and each epoch consisted of 500 iterations with a batch size of 16, similar to the standard training procedure. The setup used for the GAN training was the Keras implementation of SRGAN provided by [15].

3.4 Setup

All models were implemented in Python 3.6 with Keras as deep learning framework and the Tensorflow backend. The hardware setup was a machine with Ubuntu 18.04, 16 GB RAM and a Nvidia GTX 1080 TI graphics card running CUDA 9.2 and cuDNN 7.3.

3.5 Evaluation

3.5.1 Method

After each training epoch a validation was performed with the the validation set of 100 images, i.e. feeding low resolution images as input to the model, predicting the high resolution image and measuring the PSNR (peak signal-to-noise ratio, see Section 3.5.2 below) and SSIM (structural similarity index, see Section 3.5.2 below) on all three RGB channels. When the model had completed the training the evaluation was done on a separate test set of 100 images. Just as with the validation during training the model predicted the high resolution image from the low resolution input image, measures the PSNR and SSIM but also the run

time of the prediction to get a direct speed measurement.

3.5.2 Metrics

To evaluate the performance of a model a metric is needed. In super-resolution it is possible to divide them into two categories, visual and practical. For the quantitative visual measurements two metrics were used: *Peak signal-to-noise ratio* (PSNR) and *structural similarity index* (SSIM). PSNR is a pixel-wise measurement measuring the reconstruction error after denoising. It is defined as:

$$\text{PSNR} = 10 \log_{10} \frac{MAX_I^2}{MSE} \tag{3.1}$$

where MAX_I^2 is the maximal pixel value and MSE is the mean squared error between the pixels of the images, defined in 2.10 [21]. Thus, by using MSE as loss function the PSNR is per definition maximized.

In addition to PSNR, SSIM was used. This is a reconstruction measurement taking into consideration high frequency details. It is based on three components: luminance (L), contrast (C) and structure (S). Given two images $\boldsymbol{I}$ and $\boldsymbol{J}$ it can mathematically be formulated as:

$$\text{SSIM(I,J)} = \left[L(I,J)^{\alpha} \cdot C(I,J)^{\beta} \cdot S(I,J)^{\gamma} \right] \tag{3.2}$$

where

$$\text{L(x,y)} = \frac{2\mu_x\mu_y + c_1}{\mu_x^2 + \mu_y^2 + c_1} \tag{3.3}$$

$$\text{C(x,y)} = \frac{2\sigma_x\sigma_y + c_2}{\sigma_x^2 + \sigma_y^2 + c_2} \tag{3.4}$$

$$\text{S(x,y)} = \frac{\sigma_{xy} + c_3}{\sigma_x\sigma_y + c_3} \tag{3.5}$$

Regarding the second type of metrics, practical, two of the most important factors for any general use are the number of parameters (size) and the run time (speed). The former was measured in thousands (k) and the latter in milliseconds (ms) as a direct measurement on the used system.

3.5.3 Metric limitations

Due to the inherent subjective nature of image evaluation it is difficult to develop good quantitative assessment criteria. There is ongoing research for the optimal quantitative metrics for super-resolution. Currently, none is completely satisfactory. Models using MSE as loss function will maximize the PSNR, which is in general fine but has proven to be slightly biased towards blurry images that lack in detail recreation. This is due to the loss function measuring the averaged pixel values of the respective images [17]. Also, it has been shown to be slightly sensitive to noise [8].

SSIM was partly developed to also take the reconstruction of the high frequency content into account. Hence, SSIM computes per pixel rather than averaging like in PSNR. A drawback of the SSIM is that it is a much more complex metric and has been demonstrated to perform badly on JPEG images [8]. PSNR is more straightforward and faster to compute. However, there is still a very strong correlation between the two metrics. In general, a high PSNR score leads to a high SSIM score and vice versa.

For evaluating the run time it was directly measured on the used system. This have both advantages and disadvantages. A disadvantage is that it is system dependent making the obtained result harder to reproduce without a similar setup. An advantage is that it in contrast to a more theoretical metric has proven to be more reliable. A theoretical option would be to use *floating operations per second* (FLOPs), but this indirect metric does not necessarily correlate with the actual run time on the platform as noted in [20]. This is partially due to different platform characteristics and the number of memory access operations.

Chapter 4
Result

This chapter is divided into an evaluation part and a design part. The evaluation section 4.1 contains all the empirical data obtained from the various assessments. The data is analyzed and related to previous research and findings. The following design section 4.2 includes the presentation and underlying theory of the proposed network architecture. It also contains all the upscaled images from this network and a comparison between the networks trained with various optimization functions.

All models in the evaluation section were trained for a 2x upscaling factor. It was noted earlier that the evaluations with SSIM (structural similarity index) were very similar to the PSNR (peak signal-to-noise ratio) and was hence omitted and this applies to all evaluations. In the qualitative part where the generated images are presented the models were trained for a 4x upscaling factor to enhance the visual differences.

4.1 Evaluation

4.1.1 Residual Blocks

The first aspect to look into for efficiency gains might be the depth of the model, i.e. the number of residual blocks. What is of interest is at what depth does the relative PSNR improvement becomes too costly, leading to a high parameter count and long run time. It is reasonable to assume that a larger model would achieve better reconstruction quality but poorer practical metrics due to its size.

In Fig. 4.1 the y-axis represents the run time and the x-axis the number of parameters. The marked circles are the number of residual blocks used in that specific network. Fig. 4.2 shows the visual performance in terms of PSNR (y-axis) over the number of parameters (x-axis). This diagram setup is used for all evaluations in this section where the parameter count differs substantially between the compared models, i.e. when model size is of interest.

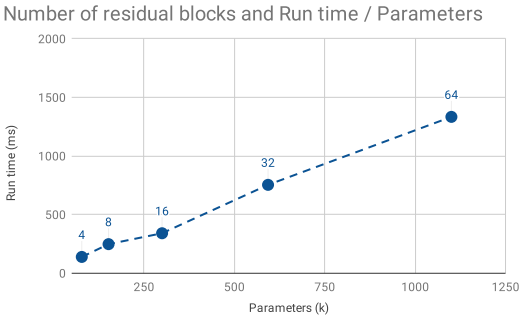

Figure 4.1: Number of residual blocks and the practical metrics for WDSR A.

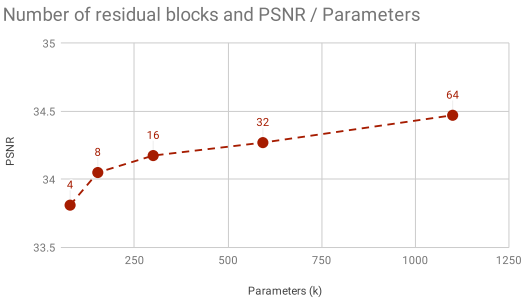

Figure 4.2: Number of residual blocks and the visual metrics for WDSR A.

We are mainly interested in finding values that lead to low practical metrics and relatively high PSNR per parameter. Figs. 4.1 and 4.2 show the result obtained by compressing the WDSR A network to a specific number of residual blocks. In Fig. 4.1, the data indicates as expected that using more blocks increases the practical metrics. There seem to be a positive linear correlation between the parameter count and run time. As Fig. 4.2 shows, increasing the number of blocks greatly increases the PSNR in the beginning, but the gains level off when increasing the depth beyond 4-8 blocks. A deeper network leads to higher relative PSNR but also to significantly larger models. Hence, 4-8 blocks seem to be suitable values to keep the model both small and to achieve a reasonably good PSNR.

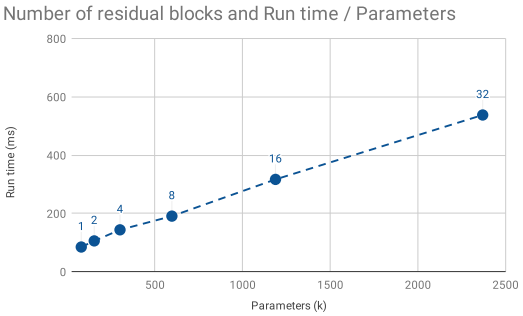

Figure 4.3: Number of residual blocks and the practical metrics for WDSR B.

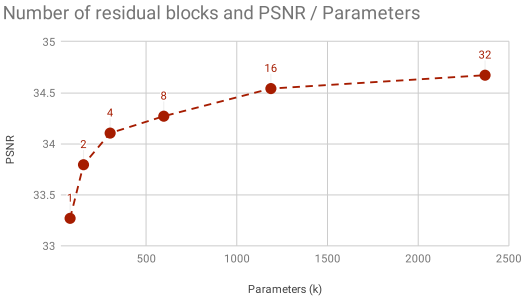

Figure 4.4: Number of residual blocks and the visual metrics for WDSR B.

A very similar pattern emerges when analyzing the depth for the WDSR B-block, as shown in Figs. 4.3 and 4.4. In comparison to the A-block, the B-block uses linear low-rank convolution which makes it approximately 4x as parameter efficient but also slower. As an example, 4 WDSR A-blocks and 16 B-blocks both reach 300 k parameters but the former a 144 ms run time and the latter a 340 ms run time (this is also the reason why the assessment starts at 4 blocks instead of 1 block as in the previous case).

In Fig. 4.3 a similar positive linear correlation between the blocks can be seen. For the reconstruction quality in Fig. 4.4 the relative improvement starts to decrease at 16 blocks. Thus, following an analogous analysis as for the WDSR A-block, 16-32 blocks would be the corresponding suitable values for the WDSR B-block. However, with 32 blocks the model starts to become quite large. Since one of the top priorities is to keep the model small, 16 blocks would probably be a the more preferable choice for this study.

4.1.2 Kernels

Number of Kernels

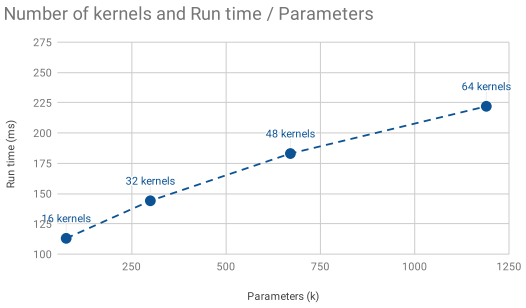

Figure 4.5: Number of kernels and the practical metrics.

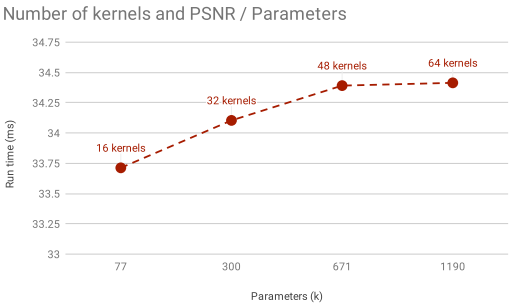

Figure 4.6: Number of kernels and the visual metrics.

Another factor for possible efficiency improvements are the number of kernels used in each layer. Generally, the more kernels used the larger the model and the longer the run time. What is of interest here is analyzing the impact the kernel number has on respective metrics. The data shows a positive linear correlation between the number of parameters and run time as shown in Fig. 4.5. For the visual assessment there is a linear increase until about 48 kernels. For kernel numbers higher than 48, the relative performance gain is reduced. However, with only one data point for kernel numbers higher than 48 this result is not very reliable. But with that in mind, the data indicates that 32-48 kernels would probably be the best range for an efficient trade-off: maintaining a small model while keeping relatively high visual performance. In comparison to most networks these are low values, where 64-128 kernels often are employed [18][24], this saves a lot on the practical performance side.

Kernel Size

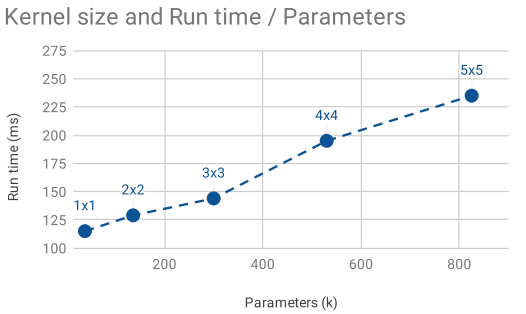

Figure 4.7: Kernel sizes and the practical metrics.

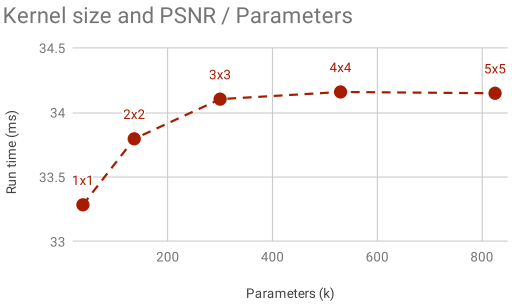

Figure 4.8: Kernel sizes and the visual metrics.

The kernel size also has a clear impact on the model size and run time. In Fig. 4.7, a positive linear correlation between the two can be seen. For the visual metrics, the improvement

of PSNR per parameter increases until the 3x3 kernel. Thereafter, larger kernel sizes have diminishing returns. From the data it appears that using a kernel size larger than 3x3 is not very beneficial. Although the 3x3 convolution is widely used it is unclear why a 4x4 or 5x5 would not be superior regarding PSNR. It might be due to CUDA optimizations for the 3x3 convolution or insufficient training [20].

4.1.3 Residual Blocks or Kernels

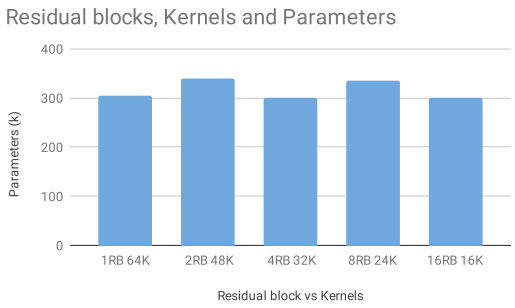

Figure 4.9: Residual block, Kernels and the parameter count.

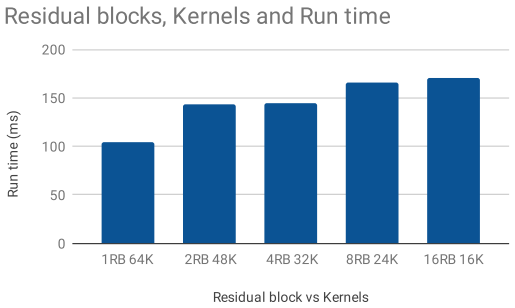

Figure 4.10: Residual block, Kernels and the run time.

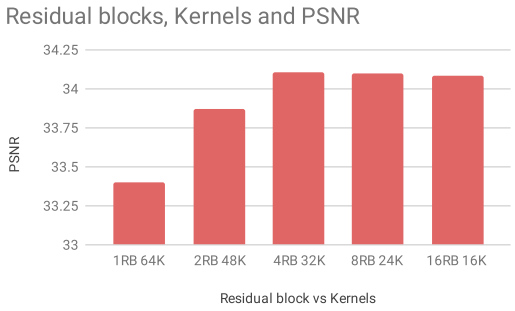

Figure 4.11: Residual block, Kernels and visual metrics.

An interesting trade-off is the one between the number of residual blocks (depth) and the number of kernels in each layer (width). When comparing decreasing or increasing the number of residual blocks versus the number of kernels, with a low fixed parameter budget of 300 k, the data seems to be quite clear. The practical metrics are visualized in Fig. 4.9, where the light blue bar represents the parameter count and Fig. 4.10, where the dark blue bar is the run time. Here does smaller bars imply a lighter network, meaning a low parameter count and low run time. The visual metrics is shown in Fig. 4.11, where the light red bar is the PSNR. The higher the bar the better the reconstruction quality (this diagram setup with the three figures will be used for all evaluations where the size is relatively similar between the compared models).

The use of 4 WDSR A blocks with 32 kernels (noted 4RB 32K) achieves the most optimal trade-off with the highest PSNR and better practical metrics than the comparable 8 residual blocks with 24 kernels and 16 residual blocks with 16 kernels. Using less than 4 residual blocks clearly impacts the visual quality too much. Extrapolating this result for the WDSR B block would lead to an optimal depth-width ratio of 16 residual blocks with 32 kernels.

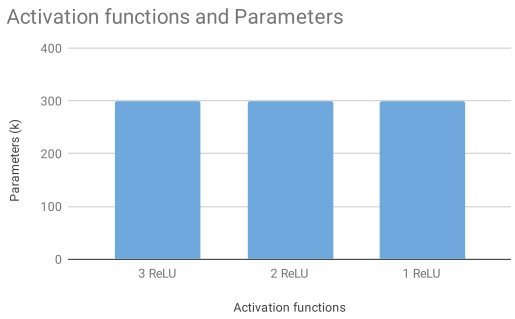

Figure 4.12: Activation functions and the parameter count.

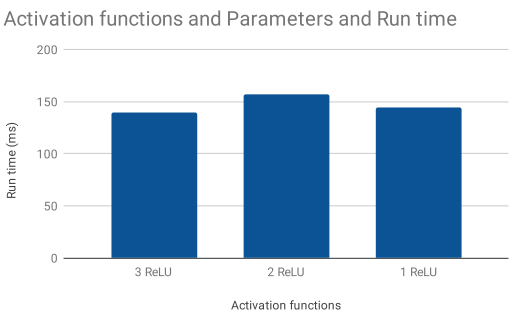

Figure 4.13: Activation functions and the run time.

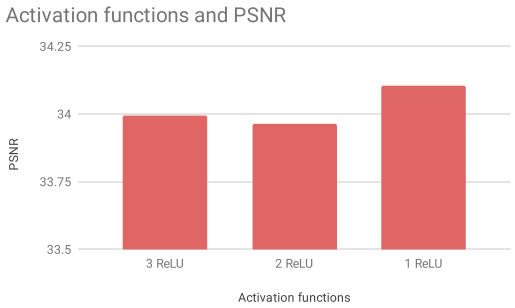

Figure 4.14: Activation functions and the visual metrics.

4.1.4 Activation Function

A various number of ReLU activation functions were applied to each block. One model with only one function between the first and second layer, a second model with an additional ReLU between the second layer and the residual, and a third model with yet another ReLU after the residual. It is possible to conclude that the number of activation functions (ReLU) does not seem to affect neither the parameter count nor the run time significantly as shown in Figs. 4.12 and 4.13. However, applying more than one activation function does reduce the PSNR as shown in Fig. 4.14. Concerning the practical metrics this result was not the expected outcome. Since the ReLU is an element-wise operation a reasonable hypothesis would be a longer run time for more operations. But the ReLU is efficient as pointed out by previous research [20] and may explain why the impact is so low. More activation functions also leads to reduced PSNR. This finding is consistent with what [30] among others have concluded where the reconstruction quality is distorted by too many non-linear functions between the convolutional layers.

4.1.5 Residuals

In this assessment local residuals (in each block) and global residuals (from the initial input layer to the final upsampling layer) were tested. Four models were implemented: one with only a local residual, one with only a global residual, one with neither and one with both. Removing a local or a global residual did not seem to have a significant affect on neither

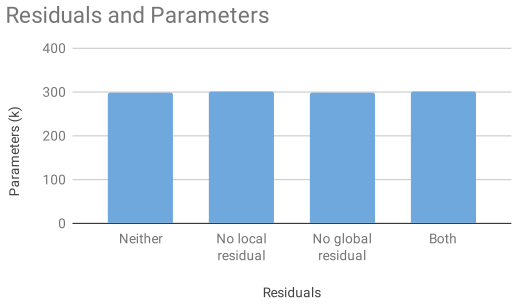

Figure 4.15: Residuals and the parameter count.

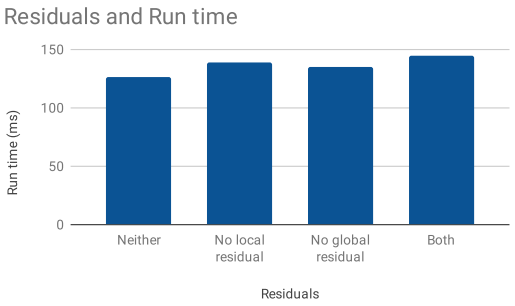

Figure 4.16: Residuals and the run time.

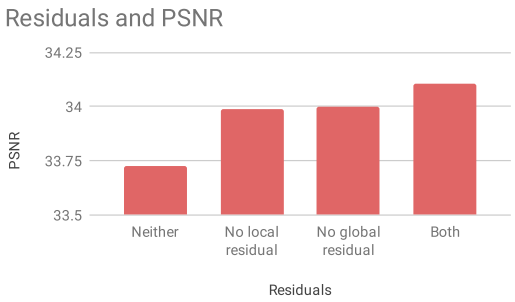

Figure 4.17: Residuals and the visual metrics.

parameters nor run time as seen in Figs. 4.15 and 4.16, but reduces accuracy, see Fig. 4.17. The residuals are evidently important and should be employed on both a local and global level, confirming the theory in [7]. In addition, this also holds from an efficiency standpoint.

4.1.6 Expansion Layer

To evaluate the expansion layer multiple models were built with various expansion factors. When analyzing the plot for the parameter and run time, visualized in Fig. 4.18, it can be inferred that there is a positive linear correlation between them. In the relative PSNR plot, shown in Fig. 4.19, a similar correlation can be derived until an expansion factor of about 6x. For a larger value like 8x, the gain has diminished. This data is in line with what was concluded in [24] and [30] where a 4x-6x expansion factor was chosen as appropriate values.

Although an expansion layer clearly improves the relative PSNR it should preferably be applied in conjunction with some sort of compression layer if a small and effective network is of interest.

4.1.7 Bottleneck Layer

Multiple different bottleneck layers were tested with a various number of output feature maps. Compared in Fig. 4.20, are the different compression levels as well as a 1x1 pointwise bottleneck layer, which reduce the computational complexity in the convolution but not the output dimension. Here does the value define the dimension of the output space (the lower

Figure 4.18: Expansion layers and the practical metrics.

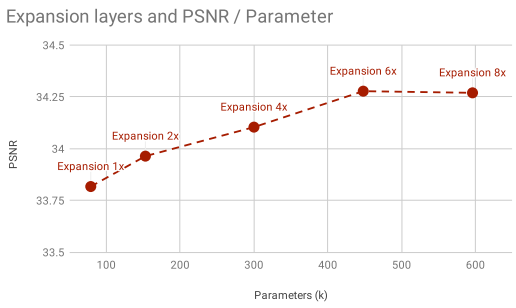

Figure 4.19: Expansion layers and the visual metrics.

the output dimension, the higher the compression). From the data it is possible to infer that the bottleneck layers have a substantial effect; the higher the compression factor the lower the parameter count. Without any bottleneck layer the model becomes much larger and slower (note the logarithmic x-axis).

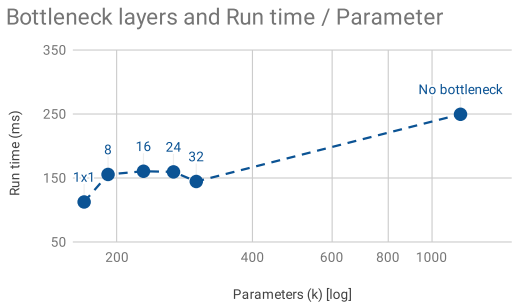

Figure 4.20: Bottleneck layers and the practical metrics.

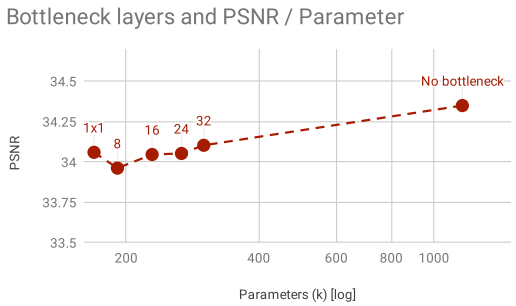

Figure 4.21: Bottleneck layers and the visual metrics.

For the relative PSNR shown in Fig. 4.21 it can be derived that a higher compression factor leads to a lower PSNR (note the logarithmic x-axis). This is expected since it has such a high impact on the practical metrics. Worth noticing in both figures is the 1x1 pointwise bottleneck layer which seems to achieve the best trade-off with the lowest parameter count and run time as well as a relative high PSNR per parameter (comparable to the use of 32 kernels). For small models, the benefit of bottleneck layers are clearly supported by previous research as well as the empirical data obtained here [9][24].

4.1.8 Depthwise Separable Convolution

The depthwise separable convolution was implemented in three different ways. One model applied it in the first layer in each block, one in the second layer and one in both layers. It successfully made the model lighter with a significant reduction of parameters but had no major affect on the run time, see Fig. 4.22 (where the operation is noted DSC). In 4.23 it can also be seen that the operation has a fairly high negative impact on the relative PSNR compared to the model using standard convolutional operators (noted No DSC). If used, it

seems to give the best result if only applied in the first layer since the practical performance is superior and the PSNR similar.

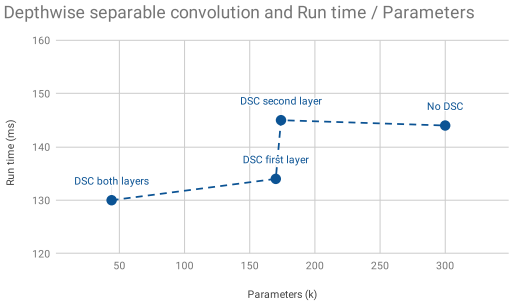

Figure 4.22: Depthwise separable convolution and the practical metrics.

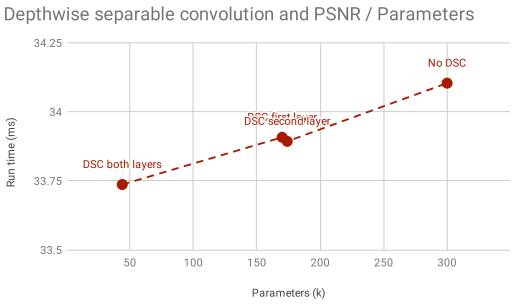

Figure 4.23: Depthwise separable convolution and the visual metrics.

This result is not in line with result in [24], where it was used for other computer vision tasks as an efficient operation with good overall performance. The discrepancy in run time might come from the used system setup, with an inefficient implementation in Keras and CUDA being optimized for the standard 3x3 convolution, as pointed out by [20].

4.1.9 Grouped Convolution

The grouped convolution (GC) was tested with two models. The first used 8 groups and 16 kernels each and the second used 4 groups with 32 kernels each. From Fig. 4.24 the practical performance shows a reduction in parameters but without impact on run time (rather a slight increase). Regarding the visual performance shown in Fig. 4.25 the models perform quite poorly with a drastic decrease in relative PSNR compared to the use of standard convolution (noted No GC). The cost of reducing parameters with the operator seems very high.

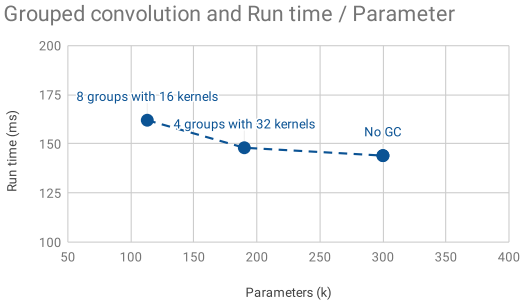

Figure 4.24: Grouped convolution and the practical metrics.

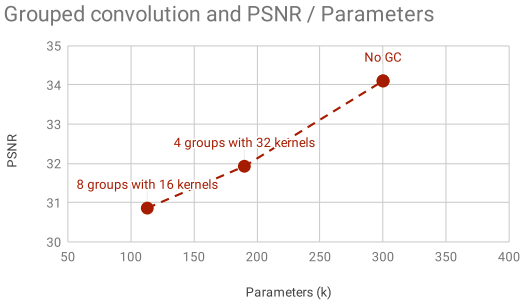

Figure 4.25: Grouped convolution and the visual metrics.

Just like the depthwise separable convolution, grouped convolution has a great theoretical foundation. But in practice, this is not visible in the data and the operator should probably be avoided. Worth considering is the fact that the operator was manually implemented since there was no support for grouped convolution in the Keras framework. This could certainly affect its performance.

4.1.10 Linear Low-Rank Convolution

The linear low-rank convolution (LLRC) is a form of factorized convolution. As previously stated it uses a 1x1 pointwise convolution to reduce the output dimension followed by a 3x3 convolution for the spatial feature extraction. It therefore works as a sort of bottleneck layer and is often used in conjunction with expansion layers.

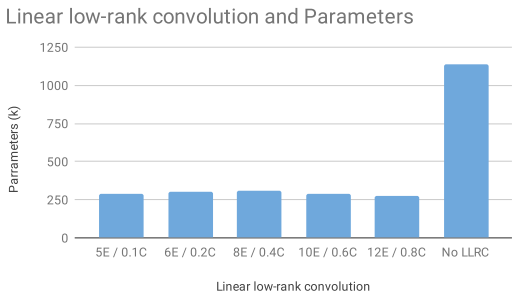

Figure 4.26: Linear low-rank convolutions and the parameter count.

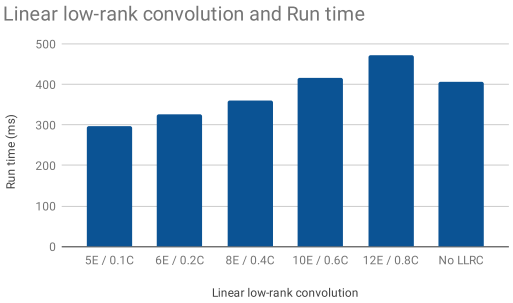

Figure 4.27: Linear low-rank convolutions and the run time.

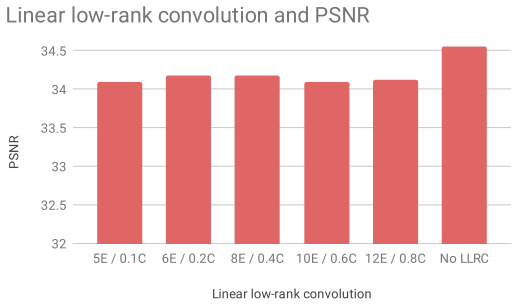

Figure 4.28: Linear low-rank convolution and visual metrics.

In this evaluation various values for the expansion and the compression factor were tested to find an optimal trade-off. Five models were implemented with different values but with the same parameter budget, hence a higher expansion factor (increasing parameters) was used with a higher compression factor (decreasing parameters) to balance them out.

The practical metrics are shown in Figs. 4.26 and 4.27. Here "5E / 0.1C" stands for a 5x expansion layer followed by a 0.1x compression (a higher value indicates a higher expansion or compression factor). It clearly shows that the use of the operator results in substantial reduction in model size. The impact on run time is not at all as large. Also visible is the increase in run time for the larger expansion and compression factors. The larger factors also seem to lead to a reduction in relative PSNR, see Fig. 4.28. Therefore, these should probably be avoided. The most viable option appears to be a 6x-8x expansion and a 0.2x-0.4x compression factor to reach a low run time and a relatively high PSNR. The result here is comparable to what was concluded in [30], where the WDSR B-blocks employ a 6x-9x expansion factor and a 0.2x compression factor. The model without linear low-rank convolution outperform in PSNR due to its large size, as expected.

4.1.11 Block Structure

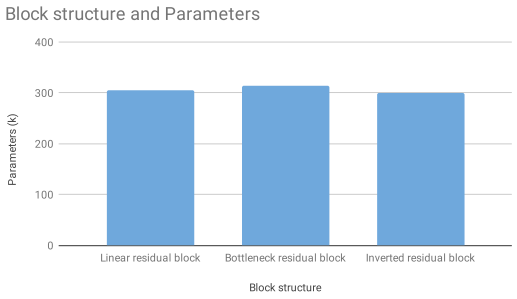

Figure 4.29: Block structures and the parameter count.

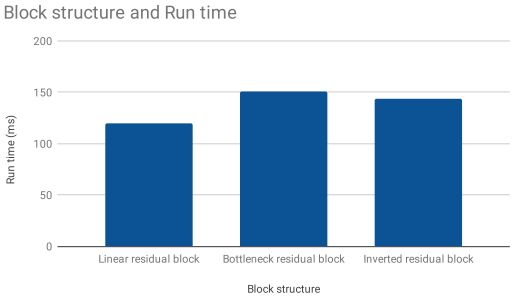

Figure 4.30: Block structures and the run time.

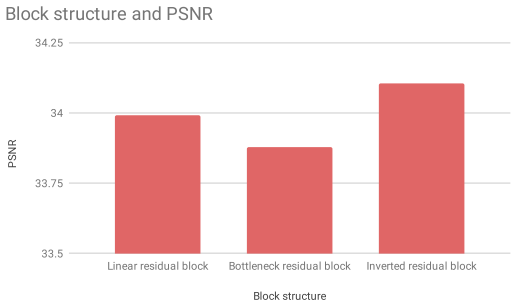

Figure 4.31: Block structures and the visual metrics.

In this assessment different residual block structures were tested: the linear, the bottleneck and the inverted. The structure of these different blocks can be seen as different ways of ordering the dimensions of the feature space in the block.

The parameter count shown in Fig. 4.29 and the run time in Fig. 4.30 are quite similar for all block structures. The visual quality tells a different story, see Fig. 4.31. Here the PSNR is greatly increased with the inverted residual block. This is in accordance with the latest research where the theory suggests enabling low dimensional feature maps as residual connections are more efficient, allowing deeper networks with an equal parameter budget. Thus, these finding are in line with what has been shown in [24] and is also the structure which is used in the WDSR and MobileNet blocks among other.

4.1.12 WDSR Block

The first block types to evaluate were the WDSR blocks (for more details see Section 2.2.13). Two heavily compressed WDSR networks were built, one based on WDSR A-blocks and one on WDSR B-blocks. These models are referred to as WDSR A-XS and WDSR B-XS and are used as references to the other block type evaluations below.

Both have a very tight parameter budget of only 300 k parameters since small and efficient networks are of specific interest in this work. The WDSR A-XS has 4 residual blocks and

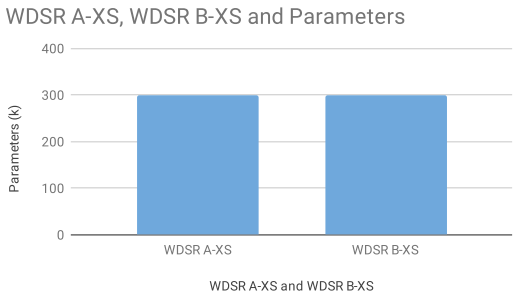

Figure 4.32: WDSR A-XS, WDSR B-XS and the parameter count.

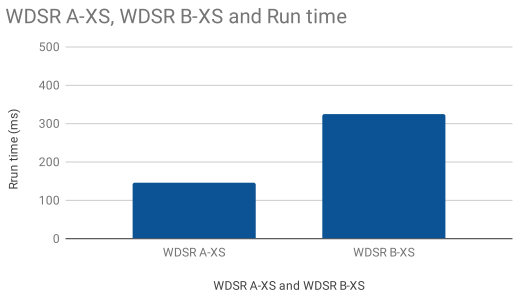

Figure 4.33: WDSR A-XS, WDSR B-XS and the run time.

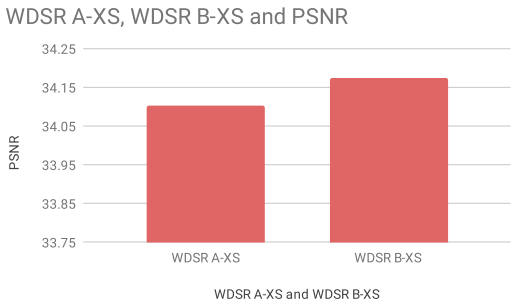

Figure 4.34: WDSR A-XS, WDSR B-XS and the visual metrics.

employs expansion layers. The WDSR B-XS has 16 residual blocks and uses both expansion layers and linear low-rank convolution.

Figs. 4.32 and 4.33 show the result of the practical metrics and visualized in Fig. 4.34 is the visual evaluation. The data shows that the A-block is faster whereas the B-block achieves superior visual quality.

4.1.13 MobileNet Block

Based on the MobileNet block, two models were implemented, a small (MobileNet S) and an extra small (MobileNet XS) variant. The difference being the depth, i.e. number of blocks. The block used inverted residuals and depthwise separable convolution (for more details see Section 2.2.12).

In Figs. 4.35 and 4.36 is the practical performance visualized. Here the MobileNet blocks have an equivalent number of parameters but a drastically higher run time compared to the WDSR A-XS and WDSR B-XS models. For the visual evaluation, see Fig. 4.37, where they achieve worse or similar performance. Overall, the blocks do not reach the level of performance shown with the WDSR XS-models. However, this is partially expected since the use of depthwise separable convolution previously lead to a negative impact on the accuracy.

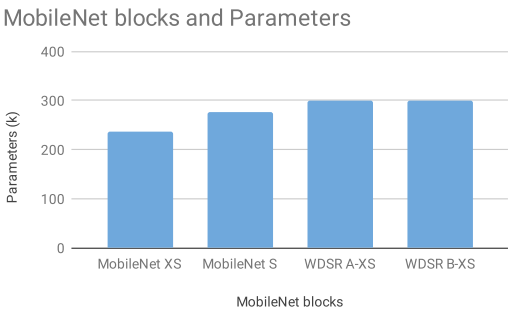

Figure 4.35: MobileNet blocks and the parameter count.

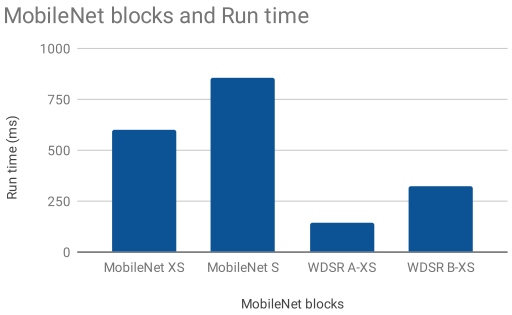

Figure 4.36: MobileNet blocks and the run time.

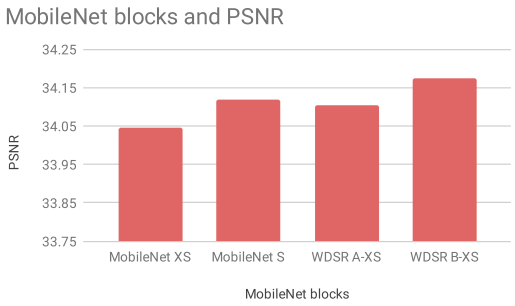

Figure 4.37: MobileNet blocks and the visual metrics.

4.1.14 SRDenseNet Block

Multiple different dense residual blocks were tested, all based on the philosophy of densely connected layers (for more details see Section 2.2.14). Among these were a dense block compressed to 3 layers with a residual mapping (SRDenseNet A S and SRDenseNet A XS) and a second with the use of an extra 1x1 pointwise bottleneck layer (SRDenseNet B S and SRDenseNet B XS).

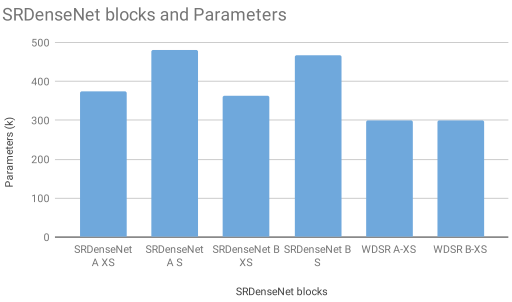

Figure 4.38: SRDenseNet blocks and the parameter count.

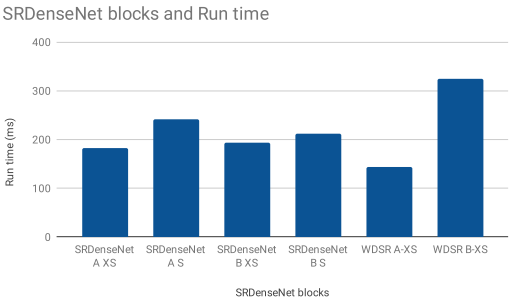

Figure 4.39: SRDenseNet blocks and the run time.

The general performance of the these models are promising. In Fig. 4.38, we can see that the blocks are slightly larger but in Fig. 4.39 that they have a similar or lower run time

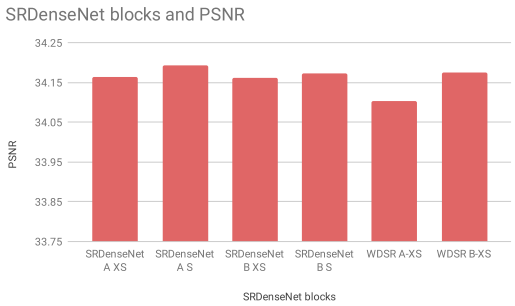

Figure 4.40: SRDenseNet blocks and the visual metrics.

compared to the WDSR XS-models. Furthermore, the visual evaluation in Fig. 4.40 shows slightly higher or similar PSNR for the various models. As pointed out in [26] the technique is quite effective in achieving high reconstruction quality. The use of densely connected layers is certainly of interest for building efficient networks.

4.1.15 ResNeXt Block

Numerous ResNeXt-based blocks were also implemented and evaluated, all built with the idea of exploiting wide networks instead of deep (for more details see Section 2.2.15). Two networks were implemented based on the ResNeXt standard block: one with 4 groups and one with 2 groups (ResNeXt GC XS and ResNeXt GC S). Two additional networks were implemented by substituting the grouped convolution with a standard convolution (ResNeXt SC A XS and ResNeXt SC A S). Finally, two blocks with standard convolution, an additional 1x1 pointwise bottleneck layer and an extra residual mapping were implemented (ResNeXt SC B XS and ResNeXt SC B S). The result from all these six models regarding the practical performance is visualized in Fig. 4.41 and Fig. 4.42. The achieved PSNR is shown in Fig. 4.43.

From the empirical data it can be inferred that the networks with the standard convolution outperform the ones with grouped convolution. This is in line with the results previously obtained for grouped convolution. The other models perform quite well. These models achieve analogous PSNR and run time but with a slightly higher number of parameters in comparison to the WDSR XS-models. From the efficiency perspective the use of wide blocks instead of deep can be concluded to be a viable option.

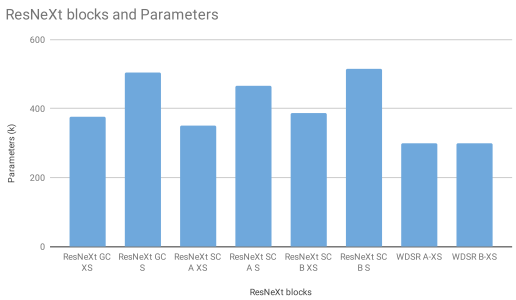

Figure 4.41: ResNeXt blocks and the parameter count.

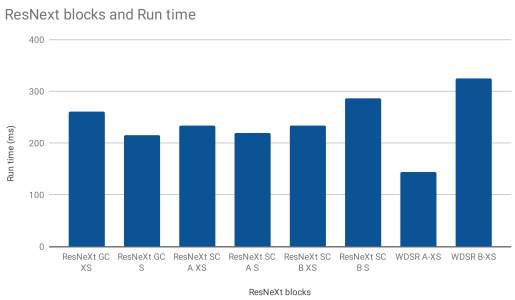

Figure 4.42: ResNeXt blocks and the run time.

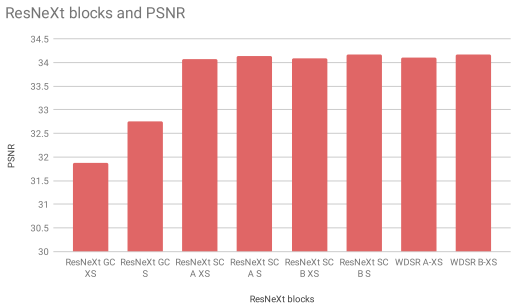

Figure 4.43: ResNeXt blocks and the visual metrics.

4.1.16 SqueezeNet Block

For the SqueezeNet blocks, three variants were implemented of different depths (SqueezeNet S, SqueezeNet XS, SqueezeNet XXS). All models used a 128 kernel expansion layer and a 32 kernel bottleneck layer in each block (for more details see Section 2.2.16).

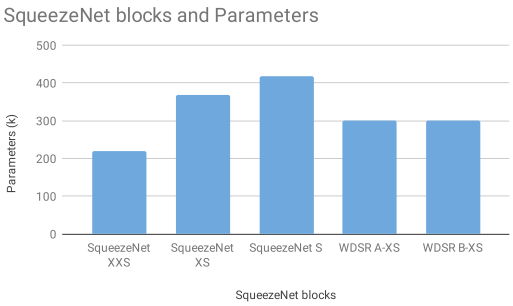

Figure 4.44: SqueezeNet blocks and the parameter count.

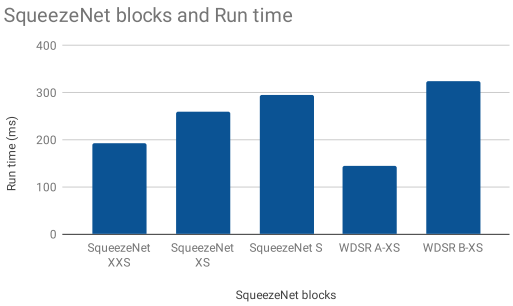

Figure 4.45: SqueezeNet blocks and the run time.

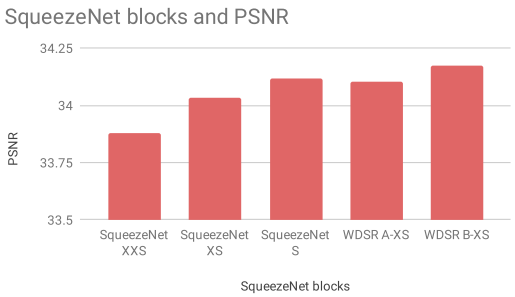

Figure 4.46: SqueezeNet blocks and the visual metrics.

The evaluation shows quite promising practical performance, see Figs. 4.44 and 4.45, where especially the SqueezeNet XXS-model outperforms the others. However, this model does not hold up when analyzing the PSNR visualized in Fig. 4.46 where the data clearly shows that the SqueezeNet-blocks are not really comparable to the WDSR XS-models (or the

previously tested SRDenseNet or ResNeXt models). Only the SqueezeNet S-model manages to reach a high PSNR but has worse practical performance than the WDSR XS-models.

4.1.17 Cascading Mechanism

The cascading mechanism was evaluated on four models (for more details see Section 2.2.17), two with CARN blocks with the use of grouped convolution (Cascading GC S and Cascading GC XS) and two with CARN blocks with the use of standard convolution (Cascading SC S and Cascading SC XS).

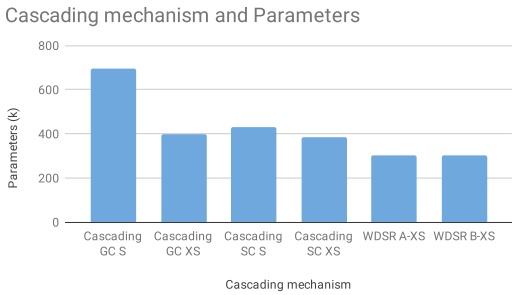

Figure 4.47: Cascading mechanisms and the parameter count.

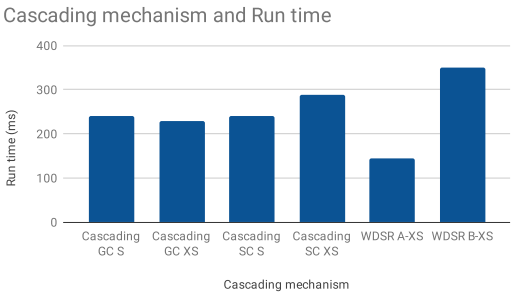

Figure 4.48: Cascading mechanisms and the run time.

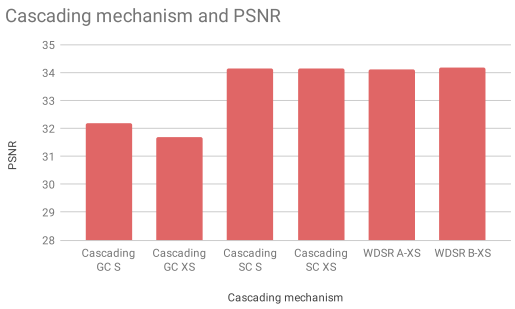

Figure 4.49: Cascading mechanisms and the visual metrics.

The evaluation of parameters and run time can be seen in Fig. 4.47 and in Fig. 4.48 respectively, where it is possible to conclude that they all achieve a low run time but with a slightly higher number of parameters. Especially the Cascading GC S stands out with almost a doubled model size. The visual assessment in Fig. 4.24 indicates that the models using grouped convolution perform poorly. This was also what could be derived from earlier evaluation of the grouped convolution operation. However, applying the cascading mechanism to the other block types with standard convolution achieves equivalent PSNR compared to the WDSR XS-models. The cascading mechanism can be concluded to be viable technique to apply.

4.1.18 Summary of Findings

General Design

From the empirical data some important knowledge can be derived, first concerning some general design choices. To achieve a good efficiency trade-off, the number of residual blocks for the WDSR XS-networks can be inferred to be 4-8 blocks for the A-block and 16 for the B-block. Thereafter, there are diminishing returns. Regarding kernels, using 32-48 with a maximum size of 3x3 seems optimal. By assessing the trade-off between the depth and width the most convincing results were attained with 4 A-blocks with 32 kernels (and 16 B-blocks with 32 kernels). When evaluating the activation functions, which obviously are crucial for building non-linear mappings, we can conclude that using them excessively should be avoided. Best overall performance was reached with 1 ReLU in the block. We can also infer from the data that using residuals, both locally and globally, is strongly recommended since they do not affect the practical metrics significantly but improve the visual metrics.

Layer Types

Concerning the various layer types, the expansion layer performed well where a factor of 6x can be regarded as the maximum value to use from an efficiency stand-point. However, due to the linear correlation between parameters and run time, it should probably be used in conjunction with some sort of compression technique. The bottleneck layer showed substantial affect on the practical evaluation, giving rise to smaller and faster models. Particularly of interest was the 1x1 pointwise convolution, demonstrating the perhaps best balance.

Convolutional Operators

Regarding the convolutional operators, neither depthwise separable convolution nor the grouped convolution gave satisfying results. The former worked well for reducing the model size but not for the run time. It also showed mediocre relative PSNR score. A similar pattern was obtained for grouped convolution but with a larger negative impact on visual performance. Neither should probably be applied from an efficiency standpoint where high visual reconstruction quality is wanted. The assessment of the linear low-rank convolution gave more promising results. We can conclude that it drastically decreased the model size and that the optimal values for expansion and compression factors seemed to be a 6x-8x for expansion and 0.2x-0.4x for compression.

Efficient Residual Blocks

Finally, the most favourable block structure can be deemed to be the inverted block structure. It outperform both the linear structure and the bottleneck structure. Regarding the different evaluated blocks a few achieved satisfying overall performance while others did not. The ones that did not were mainly MobileNet and SqueezeNet. Both underperformed in the visual quality assessment compared to the WDSR XS-models. Noticeable is that none of these block types were specifically developed for super-resolution. However, neither the size nor speed were very convincing either.

One block type that did perform well were the SRDenseNet blocks. The models were marginally larger but had similar or lower run time compared to the WDSR blocks. They also achieved likewise or greater performance in the visual evaluation. Also, the wider blocks based on ResNeXt performed quite similarly to the SRDenseNet blocks (however, only in models without grouped convolution). Lastly, the cascading mechanism proved its viability by a small increase in visual metrics at a reasonable cost in model size. In addition to the WDSR blocks, all of these three block types are definitely worth considering for an efficient architecture.

4.2 Design

In this section the proposed model is first presented along with the theory behind it. The model is then evaluated quantitatively to measure its general performance against comparable models. Then, the model was as previously mentioned trained with various loss functions to nudge it towards different goals. These were:

(a) Mean absolute error loss to maximize PSNR.

(b) Content loss to maximize feature similarity.

(c) General adversarial network (perceptual loss) to maximize perceptual quality.

This section contains all the generated images for these different models and a comparison to the traditional B-spline interpolation technique (for more images take a look at Appendix B).

4.2.1 Proposed Model - Dense WDSR

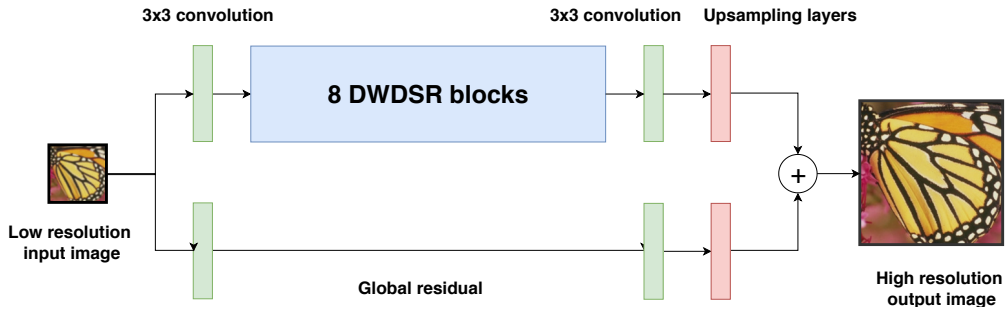

Figure 4.50: The proposed DWDSR network.

The development of an efficient architecture began by analyzing the gathered empirical data. Instead of randomly trying things out the network was built based on the most promising techniques from the quantitative analysis. Here it is proposed as: Dense WDSR, or DWDSR. The network uses the WDSR network as base structure and contains 8 residual blocks, visualized in Fig. 4.50. This is the same structure used in all the previously performed assessments. The design philosophy was based on four core ideas derived from the quantitative analysis:

(a) The efficient increase in reconstruction quality with the use of expansion layers.

(b) The drastic reduction in number of parameter with linear low-rank convolutions.

(c) The superior visual quality achieved through employing densely connected layers.

(d) The reduction in computational complexity by relying mainly on 1x1 pointwise convolutions.

The block can be divided into three identical sub-blocks and each sub-block contains three parts. An overview over the residual block is shown in Fig. 4.51. First, each sub-block uses expansion layers (coloured in green) in conjunction with linear low-rank convolution (LLRC, coloured in blue). The two are separated by a single ReLU activation function (coloured in red).

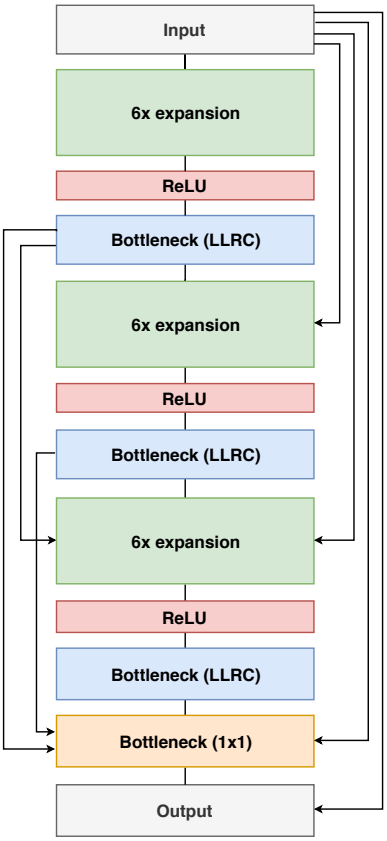

Figure 4.51: The proposed DWDSR block.

In accordance with the best shown trade-off, it uses an expansion factor of 6x for the expansion layer and a compression factor of 0.2x for the linear low-rank convolution. This enables a parameter-efficient network. Second, this block is also densely connected (the black arrows). Each sub-block applies a progressive increase of shortcut connections to subsequent expansion layers, allowing the block to incorporate features from multiple abstraction levels. The connections are fed into the expansion layers and not the bottlenecks. The idea behind this was that these layers have a higher dimensional output space, making them especially suitable for feature extraction with more input data. This is also as the data showed a light technique that increases accuracy efficiently. And as previously pointed out in [26] it enables the use of a less deep network and helps alleviating the vanishing gradient problem. Third, an additional bottleneck layer (coloured in orange) is used to compute the final feature maps from the block. As input this layer takes the output from each sub-block. This was proven to reduce the computational complexity while still maintaining good reconstruction quality.

The empirical results of depth compared to width indicated that the use of 32 kernels with 4 or 16 residual blocks were optimal for the WDSR A-block and B-block respectively. On similar merits were 32 kernels with 8 residual blocks found to achieve a corresponding

optimality for the DWDSR block.

Since neither the depthwise separable convolution nor the grouped convolution was proven very effective, they were discarded in favor of standard convolutions. The network mainly relies on 1x1 pointwise convolutions but also uses a 3x3 convolution in the linear low-rank convolution, since only using the former was shown to reduce the accuracy. Furthermore, the block has an inverted dimensional structure since it was proven the most beneficial. It also employs a relatively sparse usage of activation functions, one in each sub-block, because to many non-linear operations was demonstrated to distort the reconstruction quality. Regarding residuals, both local residuals in each block and a global residual were applied.

So how well did the model perform? A quantitative assessment can be seen in Fig. 4.52 and Fig. 4.54 where DWDSR is compared to models employing previously proposed blocks (the ones assessed in the evaluation section). All models were trained with an identical training setup (for more details see Section 3.3). The green bars in each figure refers to the proposed network. For 2x upscaling the model's parameter count was 316 k, the run time 308 ms and the PSNR 34.51 (for 4x upscaling was the run time 98 ms and the PSNR 28.86). From the diagrams it is possible to conclude that compared to the other models:

(a) The proposed model achieves a low parameter count.

(b) The proposed model achieves a similar run time.

(c) The proposed model achieves superior visual performance.

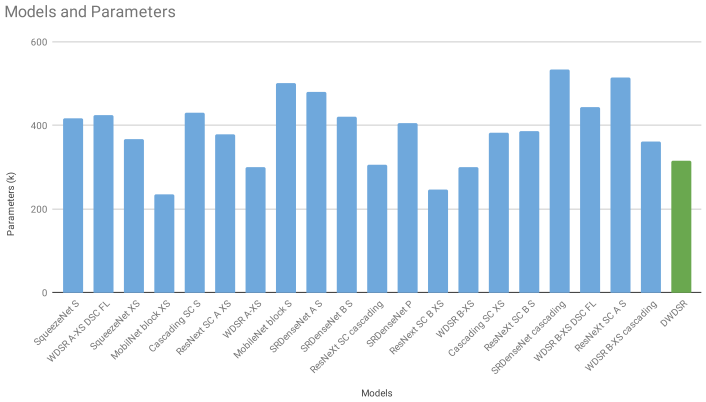

Figure 4.52: Comparison between all models' parameter count.

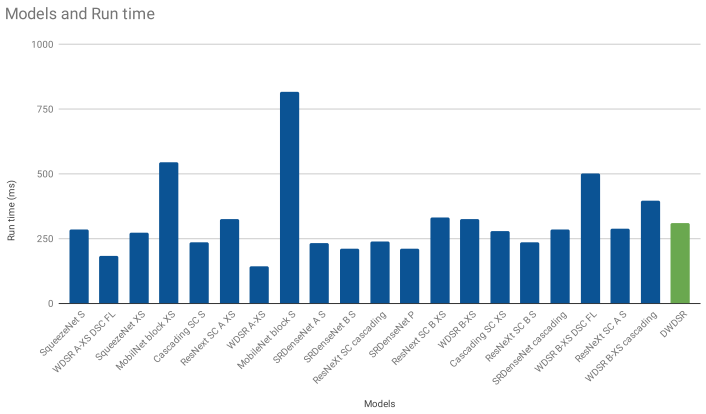

Figure 4.53: Comparison between all models' run time.

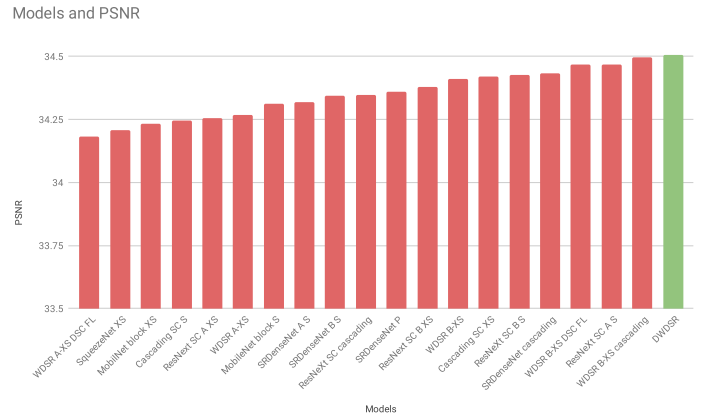

Figure 4.54: Comparison between all models' visual metrics.

4.2.2 Images

Mean Absolute Error Loss

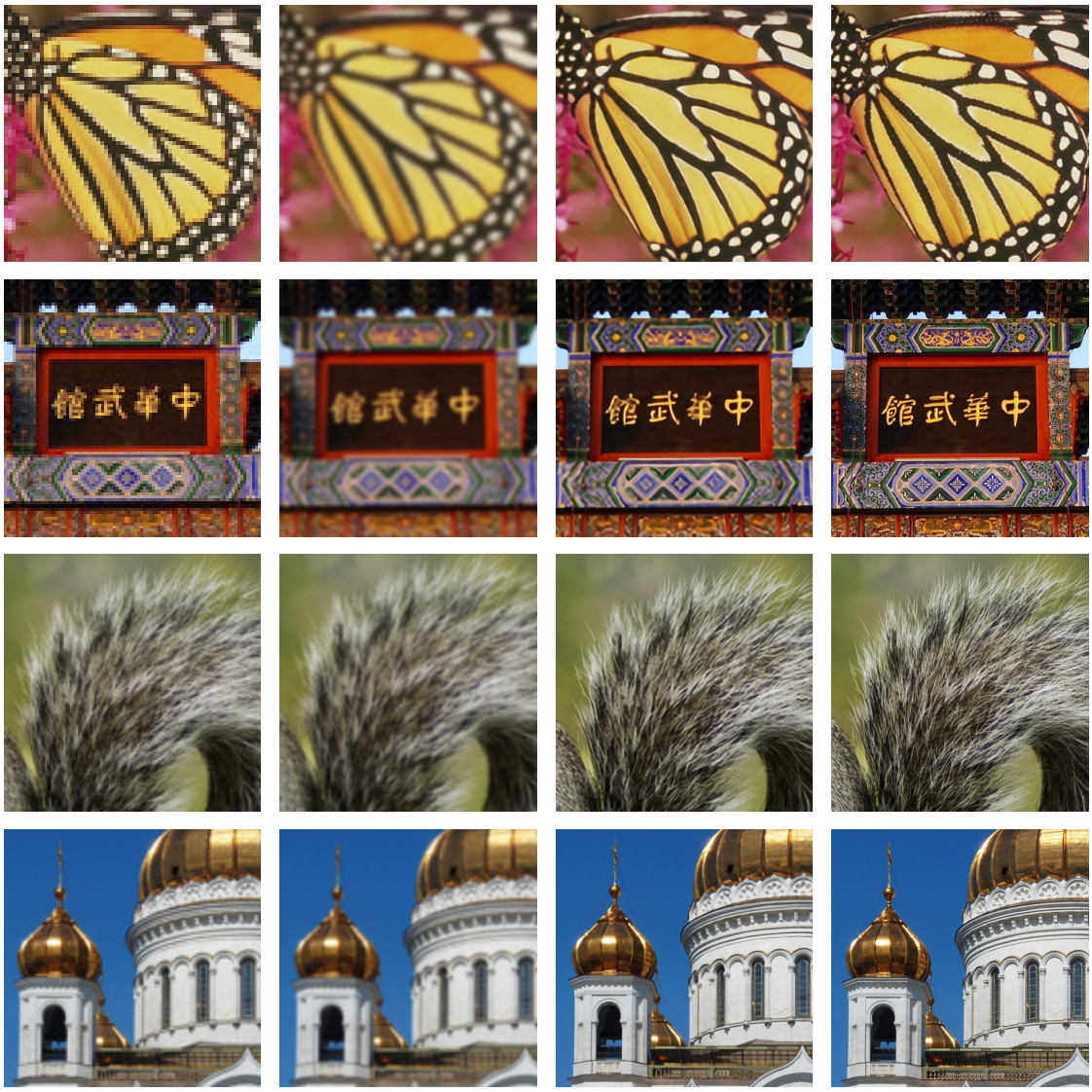

Figure 4.55: Image columns from left to right: the low resolution input images, the B-spline interpolated images, the generated high resolution images with MAE and the real high resolution images.

The proposed network was then trained with various loss functions for a more subjective evaluation of its performance. First by minimizing MAE and the generated images can be viewed in Fig. 4.55. In the figure, the image columns going from left to right corresponds to: the low resolution input images, the traditional B-spline interpolated images (baseline), the generated images from the DWDSR network and the real high resolution images. All images presented here and in the rest of the result section use this figure setup, i.e. the generated images from the model are presented in the third column. All images have been upscaled by a factor of 4 and zoomed in to enhance the visual differences. The generated images are

of size 4-6 MB from DIV2K data set (except a few being of 1 MB from the Set14 data set). Subjectively the model seems to perform well. It clearly outperforms the B-spline interpolation method. The upscaled images also closely resembles the real high resolution images. The images are colourful but slightly too smooth, blurring out some of the details. This can for instance be seen in the lack of graininess in the tail of the squirrel in the third row in Fig. 4.55.

Content Loss

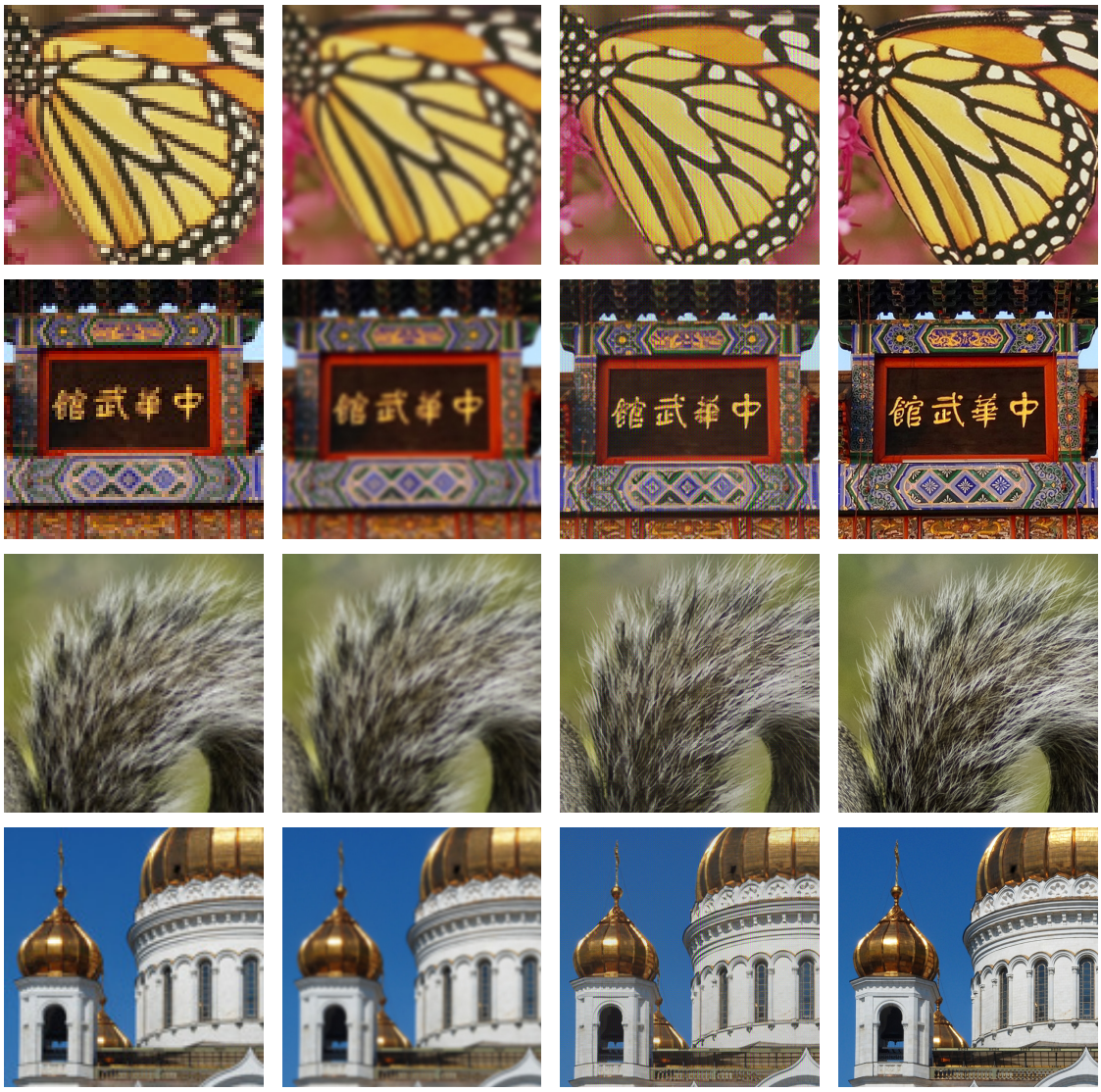

Figure 4.56: Image columns from left to right: the low resolution input images, the B-spline interpolated images, the generated high resolution images with content loss and the real high resolution images.

The DWDSR was also trained with content loss as loss function with the purpose of maximizing the feature similarity (for more details see Section 2.2.18). The images presented

in Fig. 4.56 have been generated with such a model. There are apparent visual differences between the images generated by this model and the previous. This model generate images with better sharpness, being able to recreate more of the details seen in the high frequency content. For instance, this is visible with the edges of the letters seen in the second row in Fig. 4.56. A drawback is the noise that is quite distinct and not as apparent in the previous MAE-model (note that this is not a PDF artifact but appeared in the pure images). This noise can for instance be seen on the wings of the butterfly in the top row in Fig. 4.56 where a grid-like pixel-pattern emerges. Comparing the two, the MAE-model looks to be visually preferable due to better color recreation and less noise.

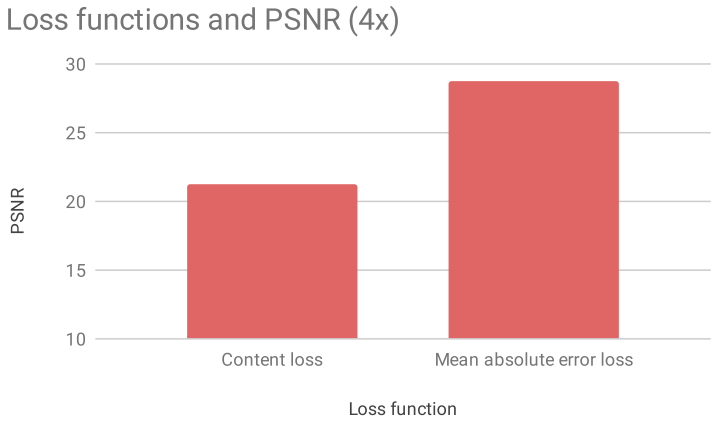

Figure 4.57: PSNR for networks trained with content loss and mean absolute error.

In Fig. 4.57 the PSNR for the two models are compared. As expected, the network optimizing the content loss results in lower PSNR compared to the MAE-network. However, the difference is larger than what we hypothesized. This indicates that PSNR is indeed very sensitive to noise, confirming the research by [8] and previously discussed in 3.5.3.

General Adversarial Network (Perceptual Loss)

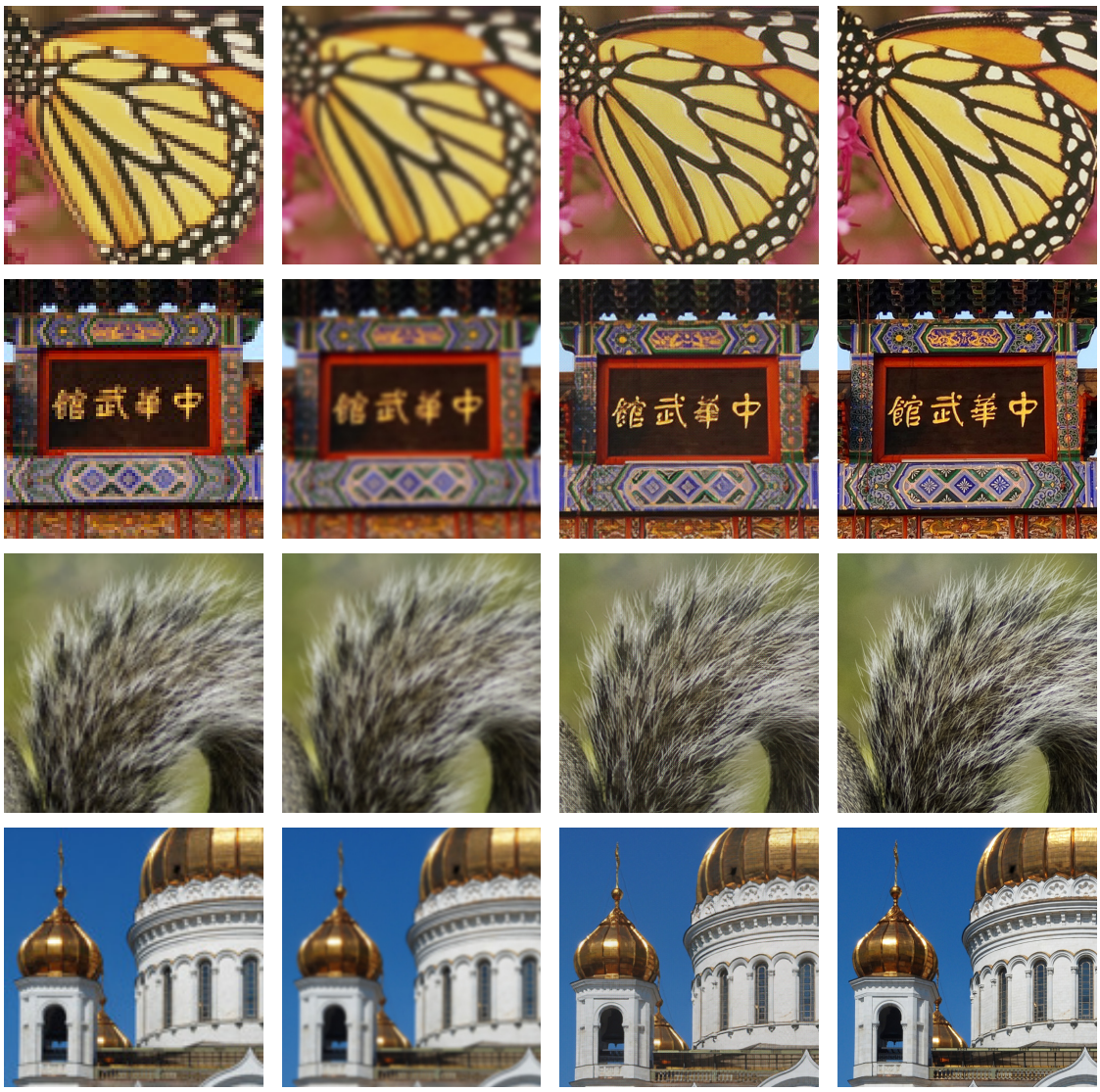

Figure 4.58: Image columns from left to right: the low resolution input images, the B-spline interpolated images, the generated high resolution images with the GAN and the real high resolution images.

The DWDSR was also used as a generator in a general adversarial network (GAN) and trained using perceptual loss to maximize perceptual similarity (for more details see Section 2.2.18 for the GAN and Section 2.2.18 for the perceptual loss). The generated images are presented in Fig. 4.58. Visually, the performance of this model is likely the best of the three. Just as with the content loss-model it is considerably better sharpness than with the MAE-model. This is especially noticeable in the the third row in Fig. 4.58. The GAN is also unarguably superior to the content loss-model due to much less noise in the generated images. Possible drawbacks with this model might be that there still seem to be some noise left and the generated images are not quite as colourful compared to the MAE-model (for a comparison side-by-side between all three models see Appendix B.5).

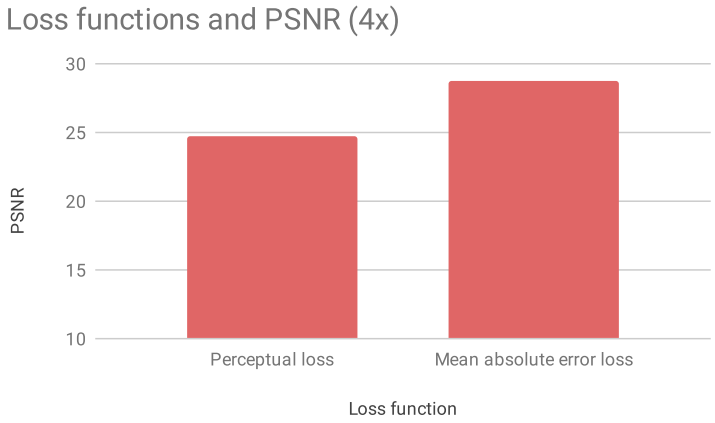

Figure 4.59: PSNR for the networks trained with perceptual loss and mean absolute error.

Presented in Fig. 4.59 is a PSNR comparison between the GAN-network and the MAE-network. As expected, the network optimizing the perceptual loss results in lower PSNR but not at all as low as what was observed with the content loss-network. The GAN actually manages to achieve superior visual reconstruction quality with a lower PSNR (clearly demonstrating the noise sensitivity of PSNR).

Network Interpolation

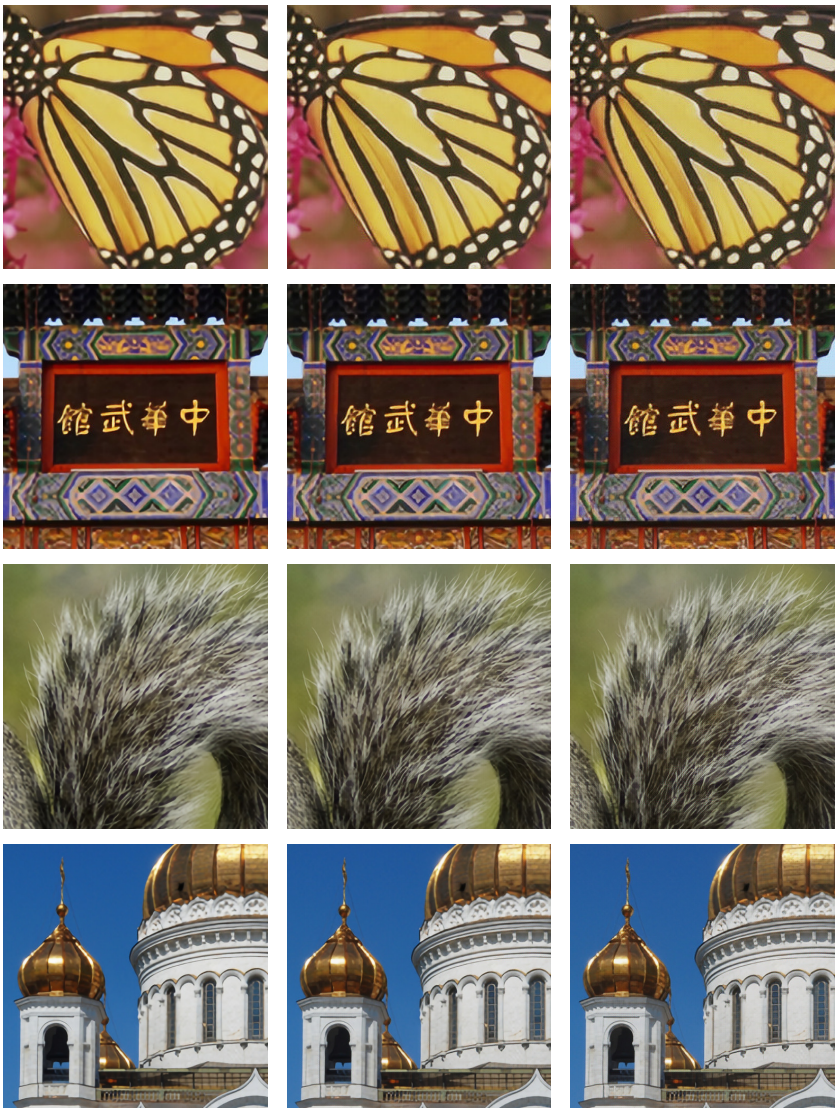

Figure 4.60: The generated images with interpolation networks with three different interpolation values: $\alpha = 0.25$ (left column), $\alpha = 0.50$ (middle column) and $\alpha = 0.75$ (right column).

With three pre-trained DWDSR models primed towards different optimization goals, the perceptually best performing models - the MAE and the GAN-network - were used to implement a few final interpolation networks (for more details about the technique see Section 2.2.19). The goal was to extract the best features from the two models, reaching an optimal visual trade-off between the two since both networks had their advantages and disadvantages. The MAE-network excelled in colour recreation with low noise levels while the GAN manages to recreate the details very well. The generated images are shown in Fig. 4.60. In this figure the three columns show the images from models with three different interpolation values:

(a) $\alpha = 0.25$: priming the model 75 % towards the MAE-model (PSNR) and 25 % towards the GAN (perceptual quality).

(b) $\alpha = 0.50$: keeping the ratio 50 % between them.

(c) $\alpha = 0.75$: priming it 25 % towards the MAE-model and 75 % towards the GAN.

It is possible to see a small gradual visual difference between the images in the left column being more MAE-like to the images in the right column being more GAN-like.

4.2.3 Comparison to Large Networks

At last, a comparison was made between the small DWDSR and the larger state-of-the-art models EDSR and WDSR. Larger in this context is in relation to the DWDSR and not in relation to the competition size. These two are the winner of the last two years global NTIRE SISR-competition (2017, 2018) [18][30]. They were implemented to be of a size of about 4.5 M parameters, roughly 15 times as big as DWDSR.

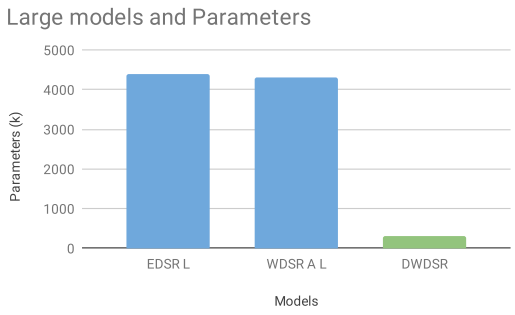

Figure 4.61: EDSR L, WDSR A L, DWDSR and the parameter count.

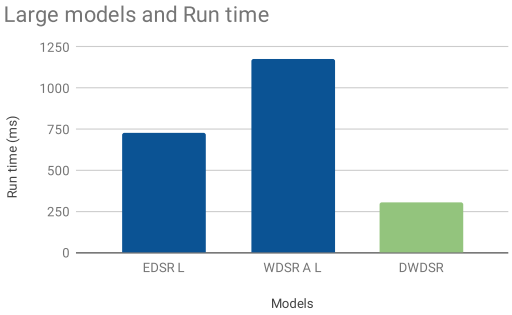

Figure 4.62: EDSR L, WDSR A L, DWDSR and the run time.

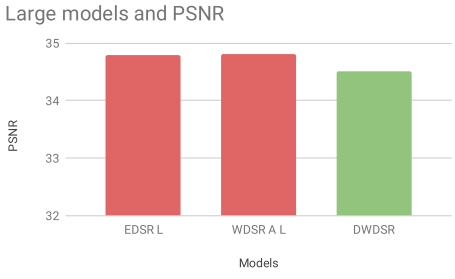

Figure 4.63: EDSR L, WDSR A L, DWDSR and the visual metrics.

The quantitative evaluation is shown in Fig. 4.61 for the model size, Fig. 4.62 for the run time and Fig. 4.63 for the visual performance, where DWDSR is coloured in green. These larger models have comparatively very poor practical performance but achieve impressive reconstruction quality scores for 2x upscaling with an PSNR of 34.76 and 34.80 respectively.

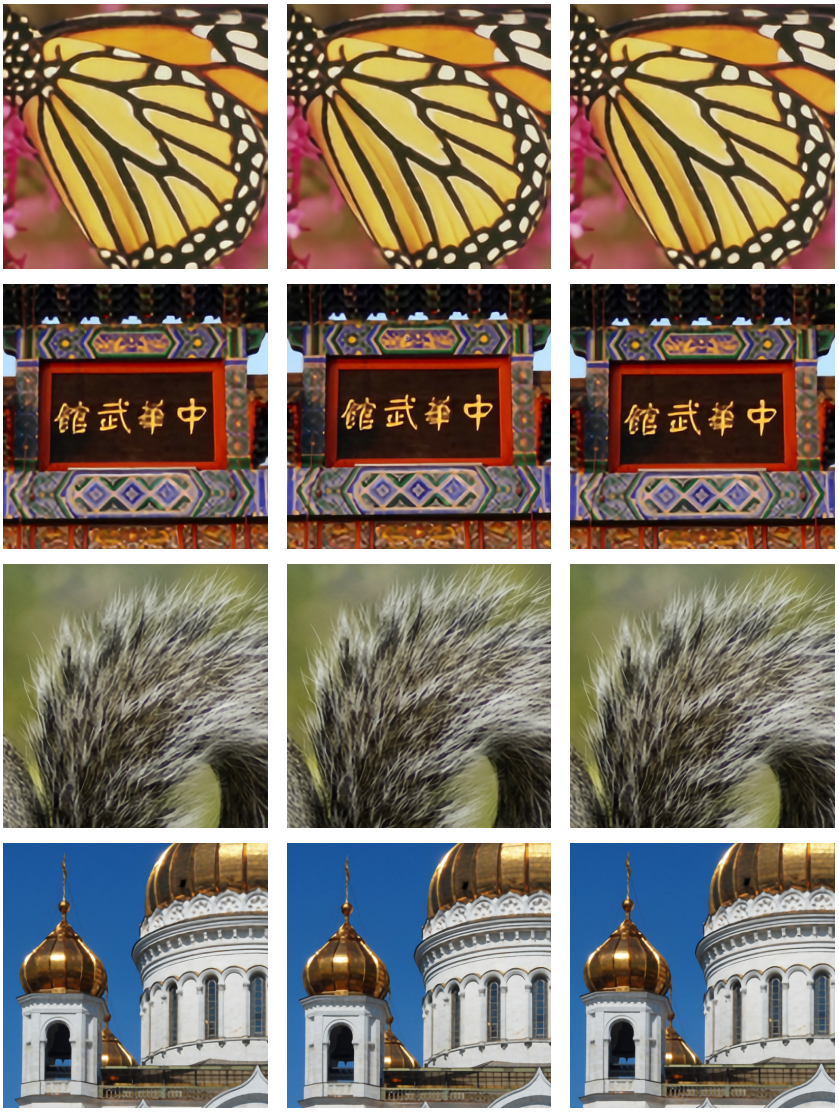

Figure 4.64: The generated images with DWDSR (left column), with WDSR (middle column) and EDSR (right column).

This can be compared to the 34.51 achieved with DWDSR. These models were trained and with MAE as loss function.

A comparison between the upscaled images of the three models is shown in Fig. 4.64. The left column contains the images generated with our DWDSR, the middle column images from WDSR and the right column images from EDSR. The quality of the images seem to be quite similar and it is actually hard to distinguish them. For this reason, see Fig. 4.65 for a closer comparison. Here it is possible to see minor improvements, for instance in the images of the eyes in the top row or in the feathers of the bird in the bottom row.

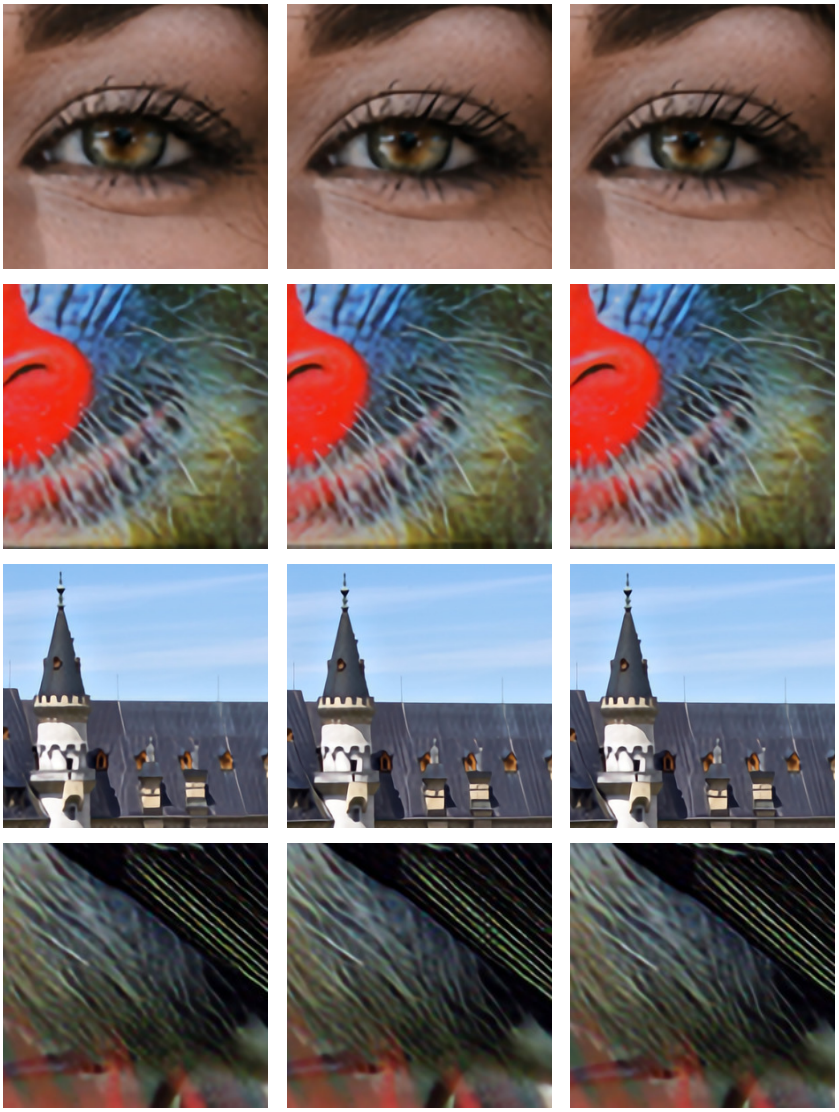

Figure 4.65: The generated images with DWDSR (left column), with WDSR (middle column) and EDSR (right column).

Chapter 5
Discussion

This chapter contains a more detailed discussion and interpretation of the data. It is divided into an analysis from an efficiency perspective, a visual discussion, thoughts about the training procedure, the possible applicability of the technique and finally relevant aspects for improving and expanding this work.

5.1 Efficiency

The proposed DWDSR model achieved satisfying results in the quantitative evaluation. Compared to all other included models regarding the practical metrics it reached a slightly lower number of parameters and a slightly higher run time. The evaluation of the reconstruction quality showed that the model achieved superior performance to all the other models. This indicates that the evaluation of the techniques was in some sense successful since it led to the proposed design performing well.

Concerning the different convolutional operators, neither depthwise separable convolution nor grouped convolution did accomplish the desired result. Both reduced the number of parameters but did not speed up run time. The depthwise separable convolution manage to reach a decent reconstruction quality while the grouped convolution did quite poorly. This is also what was concluded in [30], where both were dropped due to these factors. However, as mentioned previously both have a theoretical foundation which is enticing. The discrepancy between the theoretical and practical run time might partly be due to the implementation in Keras not being efficient (and as previously mentioned, grouped convolution is currently not supported). Furthermore, CUDA is specifically optimized for the 3x3 standard convolution [20]. This certainly affects the run time of the non-standard factorized methods [20]. Nonetheless, both operators have been used successfully in other computer vision tasks. It is too early to discard them for super-resolution although the current state does not look very convincing.

Wide networks rather than deep are also a very interesting area from an efficiency standpoint. Of the five models reaching the highest PSNR (peak signal-to-noise ratio), did three exploit wideness in some way. Either with the use of expansion layers or with wider blocks (e.g. ResNeXt which uses multiple parallel convolutions). According to [30], its superiority might come from expanding the feature space before activation function, leading to more information being passed through each block while still keeping a highly non-linear network. The convolutions done in parallel might be based on similar merits.

The bottleneck layer was proven to be crucial for efficiency as well. It is possible to compress the output feature space quite heavily without affecting the visual performance too much. The 1x1 pointwise convolution and linear low-rank convolution might be the two most preferable options. The former is especially effective in reducing both the number of parameters and run time. The result from the latter was close to what was concluded in [30]. It is also possible to combine these methods as in the proposed network, where both are used used at different stages in the residual block.

Also, just adding techniques which have proven useful individually, does not automatically result in better performance. This was for example seen when adding the cascading mechanism. It worked well for some models but not for others. A reasonable assumption why it did not lead to any performance enhancement for the DWDSR is due to the densely connected layers. This technique is quite similar and also based on increasing the flow of information from different abstraction levels. Thus, adding a cascading mechanism on top of that was not useful.

Finally, it should also be noted that very different architectures can achieve very similar results. Of the best models one used WDSR B-blocks with a cascading mechanism, another employed wide blocks with parallel convolutions and a third used densely connected layers. This is somewhat enticing since it shows that there is not a single path to efficiency. There are certainly a lot more unknown great techniques that will be invented the years to come.

5.2 Visual Quality

The qualitative assessment of the generated images showed good overall results. The generated upscaled images from the best performing models drastically improve the visual quality of the input images and closely resembles the real high resolution images. The super-resolution technique clearly shows its superiority compared to the conventional B-spline interpolation method. We also showed how using various optimization function leads to notably different perceptual results. The standard MAE function did generally achieve good images with a very low noise level and highly colourful images. Its drawback is mainly the lack of recreating details. The content loss-model, which favored feature similarity did generate images with greater detail recreation but at the cost of substantial noise. The GAN-network accomplished impressive results by finding a middle ground, generating images with great detail recreation while not giving rise to too much noise. A theory to why this model led to images with less noise is that the generator network was forced to not only focus on content quality but also fooling the discriminator network. Too much noise would likely be a way for the discriminator to easily classify the generated images as fake, increasing the generator loss. Nonetheless, there is still some noise left which is apparent with certain images in certain areas when zoomed in multiple times. This may be possible to at least partly fix by

adjusting the perceptual loss. The result here is similar to what was concluded in [17], where the GAN was able to outperform other network types in in realistic detail recreation.

The model trained with content loss achieved a very low PSNR. However, the visual difference did not seem to be at all as big. This also applied to the GAN with the perceptual loss, where the PSNR was lower although it arguably generated images with higher quality. This discrepancy does not reflect subjective perception and is a clear indication of the evaluation metric not being optimal.

The interpolation network, favoring partly the MAE-network and partly the GAN-network, also manages to generate convincing results. For instance when using an interpolation value of 0.25, slightly nudging it towards the MAE-network but mainly relying on the GAN, the noise from the latter could be reduced while still managing to recreate the details very well. What is specifically enticing with the technique beyond the customizability, is the ability to build new networks without having to re-train them. This is very time efficient. It can also be difficult do define an appropriate loss function. This interpolation technique allows for building multiple models and simply updating the weights in accordance with the desired visual result. However, this was only tested for combining the MAE and the GAN. It is not clear how well it would turn out for completely different loss functions.

Lastly, it is important to emphasize that although the DWDSR trained as part of a GAN did reach the best visual results it does not seem to be a single model type which is visually superior to all other (at least not in all areas). This type of evaluation is in is essence very subjective. It does not need to be a consensus regarding the visual quality of the images generated from different models and people will likely have different opinions.

5.3 Assessment Criteria

There is clearly a rationale for researching for better assessment criteria. All networks need a loss function and depending on what the objective is, it should be adapted for that particular task. One problem with this is that for super-resolution it can be difficult to explicitly state a precise definition of what the objective of the task actually is. This is most likely the reason why most networks employ MSE or MAE by simplicity, although these models might have vastly different use cases. There is a great need for developing clear and precise criteria and definitions for different applications. This would likely not only increase the performance of a model for the particular task but also making the comparison between models developed for different purposes more fair.

At the moment it should be concluded that there is no one-size-fits-all loss function or evaluation metric. It heavily depends on what the purpose of the network is and in what area it should be applied. Arguably, it may in many cases be best to use a combination of quantitative and qualitative metrics. The standard loss functions MSE and MAE both produces similar results by indirectly maximizing the PSNR. Since the PSNR often is the default metric for the image recreation quality it is understandable that these are used extensively. These functions do work well for most purposes. For instance, for medical image analysis a MAE would probably be preferable since visually would the low noise levels be prioritized. However, as seen there are areas where issues arises. Although the use of a GAN with a perceptual loss results in lower PSNR, the visual quality is often superior as we have seen. This discrepancy does not reflect the subjective perception of the image quality and is hence prob-

lematic. An application where the perceptual loss would be suitable is for instance a mobile photo application where most photos are of people, animals and landscapes.

5.4 Training

Training models are computationally demanding. Despite using a fast graphics card the training takes a lot of time, especially when training so many different models. For this reason the 100 epochs was established as limit for the smaller models and 500 epochs for the larger ones. For the most part this was sufficient, at least to derive a conclusion of its utility in this study. With a training time for the smaller models of 2-8 hours and for the larger 12-64 hours a limit had to be set.

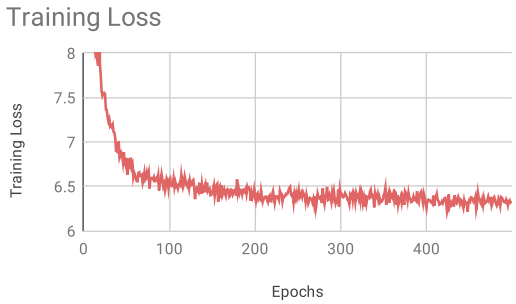

Figure 5.1: Loss during training the DWDSR.

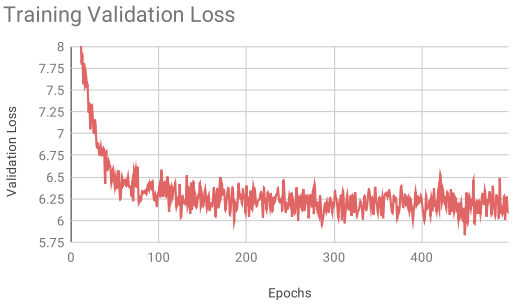

Figure 5.2: Validation loss during training the DWDSR.

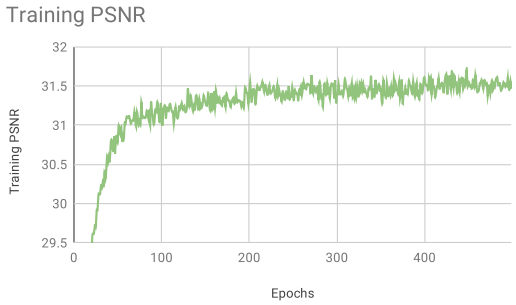

Figure 5.3: PSNR during training the DWDSR.

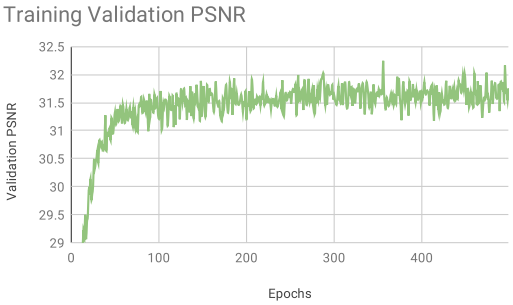

Figure 5.4: Validation PSNR during training the DWDSR.

Figs. 5.1 and 5.2 show the training loss and validation loss during a 500 epoch training for 4x upscaling session visualized. In Figs. 5.3 and 5.4 are the training and validation PSNR during the same training session. It can be seen that both the loss and PSNR during training still seem to make slight improvements. Nonetheless, this is not as clear regarding the validation (which is what matters). Therefor, the model appears to being at or close to its maximum performance.

However, this might not be the case for all evaluations. The smaller models of 100 epoch training session may have needed more training. This may very well apply to the larger EDSR

and WDSR networks as well. Training each network longer would obviously decrease the risk of possible underfitting, leading to a more reliable result. Thus, a possible improvement would have been to train all models for longer. Additionally adding early stopping functionality to prevent overfitting would have been even better. This is a functionality that specifies when the training should be regarded as completed, by setting up a fixed number of epochs without improvement as a stop sign.

Finally, most of the training went well without any major problems. One smaller issue was encountered when training cascading networks. When applying the cascading mechanism an instability sometimes arose where the loss would suddenly increase drastically. It could reach levels so high it could not repair it in a reasonable time. This was solved by decreasing the learning rate and re-start the training with previously saved weights.

5.5 Applicability

It is possible to conclude that even small models can be used for super-resolution. It has been proven that with a small network with relatively fast run time it is still possible to achieve satisfactory image quality for 2x-4x upscaling of low resolution images. By using various loss functions or interpolation techniques it can also be inferred that the network can be adapted for different applications.

A relevant question for real-world applicability is what images sizes can be upscaled. The input images were upscaled from a size of approximately 1000 x 700 (1.4 MB) (for 2x) or 500 x 350 (370 KB) (for 4x) to roughly 2000 x 1400 (3.8 MB). The maximum upscaling size was 1800 x 1350 (4.5 MB) (for 4x) to 7250 x 5450 (45 MB). The run time for upscaling the standard images were about 350 ms (for 2x) and 100 ms (for 4x) and for the large images 1800 ms (for 4x). These are low run times and would for the majority of real-world application be sufficiently fast. However, it should be clearly stated that the machine used was powerful with a GTX 1080 TI graphics card. In many real-world use cases such a machine will not be available. How fast a model needs to be is obviously very dependent on its use case. For medical image processing (e.g. MRI) or satellite image processing, the run time would probably not be of highest priority but rather the end result. Here, the technique would likely be applicable today. For other areas with real-time usage where run time is important, like video surveillance, military imaging equipment or automotive industry applications, it obviously depends on the specific requirements. It is is however clear that neural network solutions should not be overlooked.

To get super-resolution in the hands of consumers, an option would be to apply the technique to applications on lightweight devices. However, this would certainly put boundaries on what could be accomplished. For mobile phones, there are currently two main issues:

(a) Hardware limitations

Memory and processing speed is crucial for a computationally demanding task like super-resolution even for light models. It should certainly be feasible to upscale low resolution images but higher resolutions would be problematic. Theoretically one way to solve the problem of uspcaling higher resolution images would be to split the image into smaller patches and run the network on each of these separately and then reconstruct the image. This would likely lead to a high run time. However, if the result is sufficiently good some delay would

likely be acceptable from a user standpoint. A second solution would be to build extremely small models. This would reduce the run time but also severly affect the visual quality.

(b) Platform support

In recent years the number of deep learning libraries has grown. For mobile platforms it is limited but a few options are Google's TensorFlow Lite, Qualcomm's Snapdragon Neural Processing Engine and Sony's Neural Network Library NNabla. There is ongoing research and development for these platforms. In computer vision, two areas being of most focus are image classification and object detection. For this reason the networks and guides provided by the libraries are often adapted for these tasks. This can be problematic for specialized super-resolution networks since many platforms have limited support for customized layers and operators. This might partly explain why the technique for mobile use is very rare today. But with the proven results achievable with light network architectures, the ongoing development of these type of designs, the increasing support for mobile deployment and progressively more computational power in the devices - it is hopefully just a matter of time before we will see a wider use.

5.6 Improvements

There are numerous areas to which this work could be improved or further expanded. To mention a few:

- Port to a mobile platform. It would be exciting to see how well the model would perform in an environment with limited resources, such as a mobile phone and what resolutions could be upscaled at reasonable run times.
- Hyperparameter tuning. By applying a hyperparameter search the performance of the model might be able to improve.
- Increased training time (or computational resources). With more training time each model would be more probable to reach its maximum performance, leading to more reliable results for all evaluations.
- Train in compressed space. For instance, it would be possible to train in JPEG-space with the purpose of speeding up the models. This would be very experimental but has been demonstrated to work for other computer vision tasks.

Chapter 6
Conclusion

From the quantitative evaluation multiple different techniques proved to be efficient. The four core components used for the proposed model were: expansion layers, linear low-rank convolution, densely connected layers and relying mainly on the 1x1 pointwise convolution. The performance of the DWDSR was satisfying regarding both the quantitative and qualitative evaluations. It achieved a low parameter count and relatively fast run time as well as high visual recreation quality. Subjectively the difference between the generated images and the real high resolution images are small. It also clearly outperformed the traditional B-spline interpolation technique. In comparison to other analyzed models built with state-of-the-art techniques (in this very lightweight class), it obtains similar practical but slightly better visual performance. The initial assessment can therefore be deemed successful, since it enabled the development of a well performing model.

With the use of a GAN the DWDSR network was also used as a generator. This model manages to generate images with greater details. With the additional network interpolation technique it was demonstrated that it is possible to nudge the model towards different visual objectives, giving rise to a greater degree of customizability by finding a suitable balance. The network was also compared to the 15 times as large state-of-the-art EDSR and WDSR-networks. These models achieved better visual performance but the proposed model was not completely outperformed as shown in the generated images in 4.64 in the result chapter. It therefore demonstrated its usability when efficiency is of interest.

Furthermore, regarding efficient super-resolution architectures it clearly does not seem to be a single type of design which is superior to all other. All of the five models achieving the highest visual evaluation scores were quite different from each other. For instance, one was based on a the cascading mechanism applied to WDSR B-blocks, a second relied on wideness with parallel convolutions and a third used densely connected layers in conjunction with expansion layers. We can conclude that there are multiple different ways to design small and efficient networks.

It can also be concluded that more sophisticated models do not necessarily correlate with better performing models. As previously shown in networks like WDSR, it is possible to

reduce the complexity by employing intelligent techniques and still achieve greater overall performance. This was for instance seen with the more complex MobileNet-based model with almost double the parameter count to the proposed network but with worse visual reconstruction quality. This matches the philosophical idea of *Occam's Razor*: given a set of different solutions, the least complex one with the fewest assumptions should be preferred. This might in some sense also be relevant for machine learning.

Finally, model design is hard. It is evident that there can be big theoretical and practical discrepancies that beforehand can be difficult to predict. Some techniques simply work well for certain block types but not for others. Likewise, just because some operators are applicable for another computer vision task does not automatically turn them into a success for a different task. This was seen with operators like the depthwise separable convolution and grouped convolution. For this reason one could argue that a two-step approach, with an initial assessment of various ideas and then basing the development of the obtained result, is a reasonable process. This gives a better understanding of why the network performs as it does, instead of simply randomly trying things out and seeing the result as a magical black box. Nonetheless, to gain knowledge testing is crucial and there does not seem to be a way around actually implementing and evaluate the techniques. Designing machine learning models has been described as an art rather than a science and there might be some truth to that.

Bibliography

[1] Eirikur Agustsson and Radu Timofte. Ntire 2017 challenge on single image super-resolution: Dataset and study. http://www.vision.ee.ethz.ch/ timofter/publications/Agustsson-CVPRW-2017.pdf.

[2] Namhyuk Ahn, Byungkon Kang, and Kyung-Ah Sohn. Fast, accurate, and, lightweight super-resolution with cascading residual network. *CoRR*, abs/1803.08664, 2018.

[3] Yang Chih-Yuan, Ma Chao, and Yang Ming-Hsuan. Single-image super-resolution: A benchmark. pages 372–386, 09 2014.

[4] Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. Image super-resolution using deep convolutional networks. *CoRR*, abs/1501.00092, 2015.

[5] Chao Dong, Chen Change Loy, and Xiaoou Tang. Accelerating the super-resolution convolutional neural network. *CoRR*, abs/1608.00367, 2016.

[6] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn Tensorflow*. O'Reilly Media, Inc., 2017.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[8] A. Hore and D. Ziou. Image quality metrics: Psnr vs. ssim. In *2010 20th International Conference on Pattern Recognition*, pages 2366–2369, Aug 2010.

[9] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017.

[10] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. Densely connected convolutional networks. *CoRR*, abs/1608.06993, 2016.

[11] Forrest N. Iandola, Matthew W. Moskewicz, Khalid Ashraf, Song Han, William J. Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size. *CoRR*, abs/1602.07360, 2016.

[12] Andrey Ignatov, Radu Timofte, et al. Pirm challenge on perceptual image enhancement on smartphones: report. January 2019.

[13] Jiwon Kim, Jung Kwon Lee, and Kyoung Mu Lee. Accurate image super-resolution using very deep convolutional networks. *CoRR*, abs/1511.04587, 2015.

[14] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1809.00219, 2014.

[15] Martin Krasser. Super-resolution, 2018. https://github.com/krasserm/super-resolution.

[16] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. pages 1097–1105, 2012.

[17] Christian Ledig, Lucas Theis, Ferenc Huszar, Jose Caballero, Andrew P. Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, and Wenzhe Shi. Photo-realistic single image super-resolution using a generative adversarial network. *CoRR*, abs/1609.04802, 2016.

[18] Bee Lim, Sanghyun Son, Heewon Kim, Seungjun Nah, and Kyoung Mu Lee. Enhanced deep residual networks for single image super-resolution. *CoRR*, abs/1707.02921, 2017.

[19] Ilya Loshchilov and Frank Hutter. Fixing weight decay regularization in adam. *CoRR*, abs/1711.05101, 2017.

[20] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet V2: practical guidelines for efficient CNN architecture design. *CoRR*, abs/1807.11164, 2018.

[21] Dr. Rajesh Mehra. Estimation of the image quality under different distortions. *International Journal Of Engineering And Computer Science*, 8, 07 2016.

[22] Michael A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015.

[23] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *CoRR*, 2015.

[24] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. *CoRR*, abs/1801.04381, 2018.

[25] Wenzhe Shi, Jose Caballero, Ferenc Huszár, Johannes Totz, Andrew P. Aitken, Rob Bishop, Daniel Rueckert, and Zehan Wang. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. *CoRR*, abs/1609.05158, 2016.

[26] T. Tong, G. Li, X. Liu, and Q. Gao. Image super-resolution using dense skip connections. pages 4809–4817, Oct 2017.

[27] Xintao Wang, Ke Yu, Shixiang Wu, Jinjin Gu, Yihao Liu, Chao Dong, Chen Change Loy, Yu Qiao, and Xiaoou Tang. ESRGAN: enhanced super-resolution generative adversarial networks. *CoRR*, abs/1809.00219, 2018.

[28] Saining Xie, Ross B. Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. *CoRR*, abs/1611.05431, 2016.

[29] Wenming Yang, Xuechen Zhang, Yapeng Tian, Wei Wang, and Jing-Hao Xue. Deep learning for single image super-resolution: A brief review. *CoRR*, abs/1808.03344, 2018.

[30] Jiahui Yu, Yuchen Fan, Jianchao Yang, Ning Xu, Zhaowen Wang, Xinchao Wang, and Thomas S. Huang. Wide activation for efficient and accurate image super-resolution. *CoRR*, abs/1808.08718, 2018.

Appendices

Appendix A
Evaluation Details

A.1 Evaluation Details

A.1.1 Evaluation of Techniques

General Design

1. Depth (number of residual blocks)
   (a) Depth was evaluated for 1, 2, 4, 8, 16 and 32 residual blocks (WDSR A)
   (b) Depth was evaluated for 4, 8, 16, 32 and 64 residual blocks (WDSR B)
2. Kernels (number of kernels and sizes)
   (a) Evaluated number of kernels: 16, 32, 48 and 64
   (b) Evaluated kernel sizes: 1x1, 2x2, 3x3, 4x4 and 5x5
3. Residual Blocks or Kernels (depth vs width)
   (a) 1 residual block with 64 kernels
   (b) 2 residual blocks with 48 kernels
   (c) 4 residual blocks with 32 kernels
   (d) 8 residual blocks with 24 kernels
   (e) 16 residual blocks with 16 kernels
4. Activation functions (in each block)
   (a) 1 ReLU (between first and second layer)
   (b) 2 ReLU (an additional between second layer and residual)

   (c) 3 ReLU (an additional between residual and output)

5. Residuals
   (a) 1 local residual (in each block)
   (b) 1 global residual
   (c) No residuals
   (d) Both local and global residual

Layer Types

1. Expansion layers (applied to the first layer)
   (a) 1x expansion factor
   (b) 2x expansion factor
   (c) 4x expansion factor
   (d) 6x expansion factor
   (e) 8x expansion factor
2. Bottleneck layers (applied to the second layer)
   (a) 8 kernels
   (b) 16 kernels
   (c) 24 kernels
   (d) 32 kernels
   (e) 1x1 pointwise bottleneck

Convolutional Operators

1. Depthwise separable convolution
   (a) Applied to both layers
   (b) Applied to only the first layer
   (c) Applied to only the second layer
2. Grouped convolution
   (a) 8 groups with 16 kernels each
   (b) 4 groups with 32 kernels each
3. Linear low-rank convolution
   (a) 5x expansion and 0.1x compression factors
   (b) 6x expansion and 0.2x compression factors
   (c) 8x expansion and 0.4x compression factors
   (d) 10x expansion and 0.6x compression factors
   (e) 12x expansion and 0.8x compression factors

Efficient Residual Blocks

Here are some used abbreviations:

- RB (residual block)
- K (kernels)
- FM (feature map)
- EXP (expansion factor)
- SC (standard convolution)
- DSC (depthwise separable convolution)
- GC (grouped convolution)
- LLRC (linear low-rank convolution)
- FL (first layer in WDSR A/B block)
- LL (last layer in WDSR A/B block)

1. Block structure (dimensional shape)
   (a) Linear structure with 64 FM
   (b) Bottleneck structure with 128 FM as default and 32 FM as bottleneck
   (c) Inverted bottleneck structure with 32 FM as default and 128 FM as expansion
2. WDSR blocks
   (a) WDSR A block: 4 RB, 32 K, 4x EXP FL
   (b) WDSR B block: 16 RB, 32 K, 6x EXP FL
3. MobileNet block
   (a) MobileNet XS: 12 RB, 32 K, DSC, 6x EXP FL
   (b) MobileNet S: 16 RB, 32 K, DSC, 6x EXP FL
4. SRDenseNet block
   (a) SRDenseNet A XS: 5 RB, 32 K, 3 layers
   (b) SRDenseNet A S: 6 RB, 32 K, 3 layers
   (c) SRDenseNet B XS: 5 RB, 32 K, 3 layers, extra residual, extra 1x1 bottleneck before output
   (d) SRDenseNet B S: 6 RB, 32 K, 3 layers, extra residual, extra 1x1 bottleneck before output

5. ResNeXt block
   (a) ResNeXt GC XS: 6 RB, 32 K, GC,
   (b) ResNeXt GC S: 8 RB, 32 K, GC
   (c) ResNeXt SC XS: 8 RB, 32 K, SC, 3 paths
   (d) ResNeXt SC S: 10 RB, 32 K, SC, 3 paths
   (e) ResNeXt SC B XS: 12 RB, 32 K, 3 paths, extra 1x1 bottleneck before output
   (f) ResNeXt SC B S: 16 RB, 32 K, 3 paths, extra 1x1 bottleneck before output
6. SqueezeNet block
   (a) SqueezeNet XXS: 4 RB, 32 K
   (b) SqueezeNet XS 6 RB, 32 K
   (c) SqueezeNet S 8 RB, 32 K
7. Cascading block
   (a) Cascading GC XS: 18 CARN, 32 K, GC
   (b) Cascading GC S: 9 CARN, 32 K, GC
   (c) Cascading SC XS: 9 CARN, 32 K, SC
   (d) Cascading SC X: 9 CARN, 32 K, SC

A.1.2 Evaluation of The Proposed Model

Here are some used abbreviations:

- RB (residual block)
- K (kernels)
- EXP (expansion factor)
- SC (standard convolution)
- DSC (depthwise separable convolution)
- GC (grouped convolution)
- LLRC (linear low-rank convolution)
- FL (first layer in WDSR A/B block)
- LL (last layer in WDSR A/B block)

1. WDSR
    (a) WDSR A-XS 4 RB, 32K, 4x EXP
    (b) WDSR A-XS 8 RB DSC FL, 4x EXP
    (c) WDSR B-XS 16 RB, 32K, 6x EXP
    (d) WDSR B-XS cascading, 32K
    (e) WDSR B-XS 12 RB, 32K, DSC FL, 6x EXP
2. MobileNet block
    (a) MobileNet XS: 12 RB, 32K, DSC, 6x EXP
    (b) MobileNet S: 16 RB, 32K, DSC, 6x EXP
3. SRDenseNet block
    (a) SRDenseNet A S, 4 RB, 32K, 3 layers,
    (b) SRDenseNet B S, 8 RB, 32K, 3 layers,
    (c) SRDenseNet cascading, 32K, 3 layers
    (d) SRDenseNet P, 8 RB, 32K, 3 layers
4. ResNeXt block
    (a) ResNeXt SC A XS, 12 RB, 32K, 3 paths
    (b) ResNeXt SC A S, 16 RB, 32K, 3 paths
    (c) ResNeXt SC B XS, 12 RB, 32K, 3 paths
    (d) ResNeXt SC B S, 16 RB, 32K, 3 paths
    (e) ResNeXt SC cascading, 32K, 3 paths
5. SqueezeNet block
    (a) SqueezeNet S, 8 RB, 32K, 4x EXP
    (b) SqueezeNet XS, 7 RB, 32K, 4x EXP
6. Cascading block
    (a) Cascading SC XS: 9 CARN block, 32K, SC
    (b) Cascading SC S: 9 CARN block, 32K, SC
7. Large networks
    (a) EDSR, 24 RB, 32K, 96 K
    (b) WDSR A, 64 RB, 32K, 32 K

Appendix B
Additional Images

B.1 Mean Absolute Error Loss

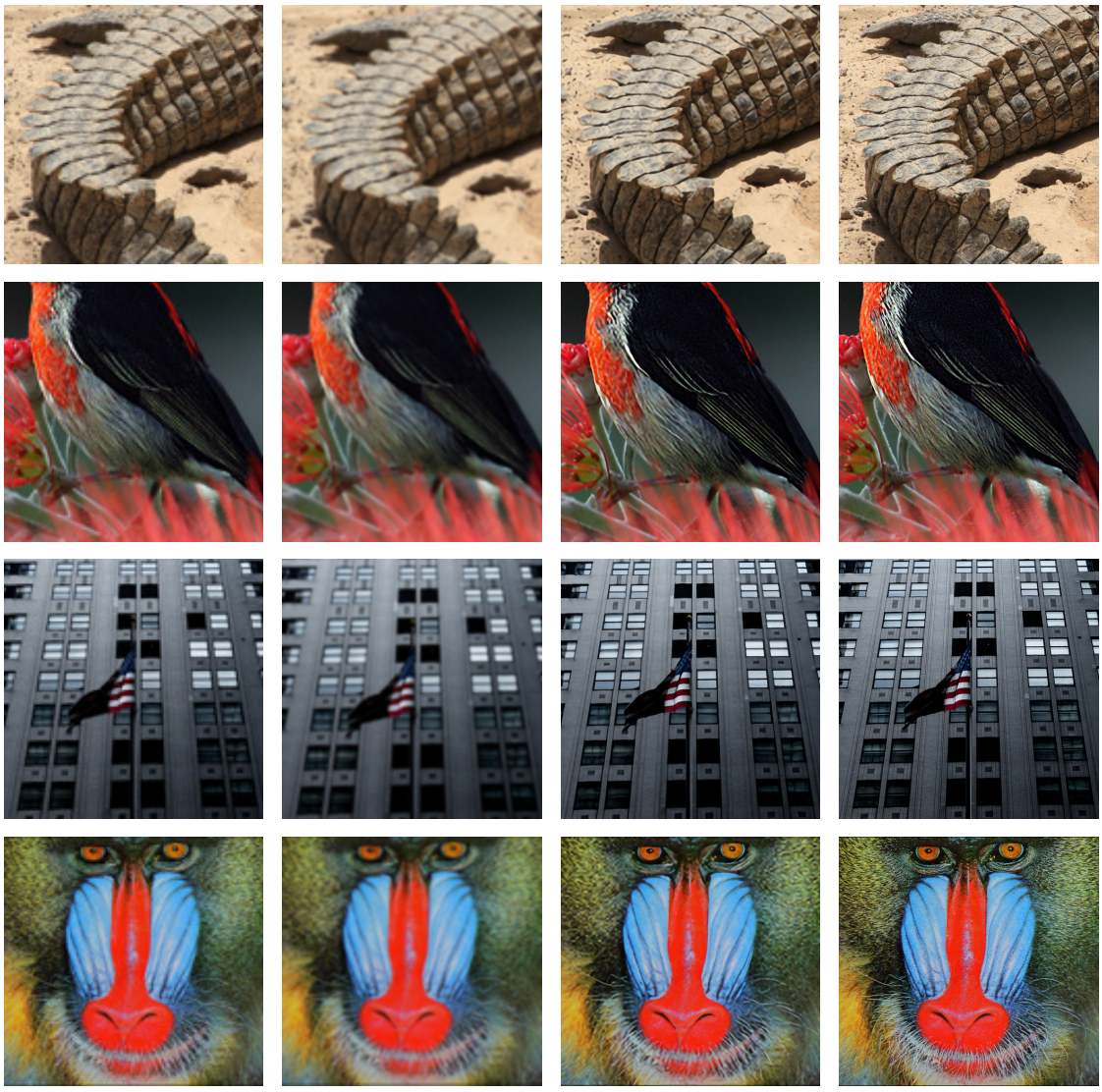

Figure B.1: Image columns from left to right: the low resolution input images, the B-spline interpolated images, the generated high resolution images with MAE and the real high resolution images.

B.2 Content Loss

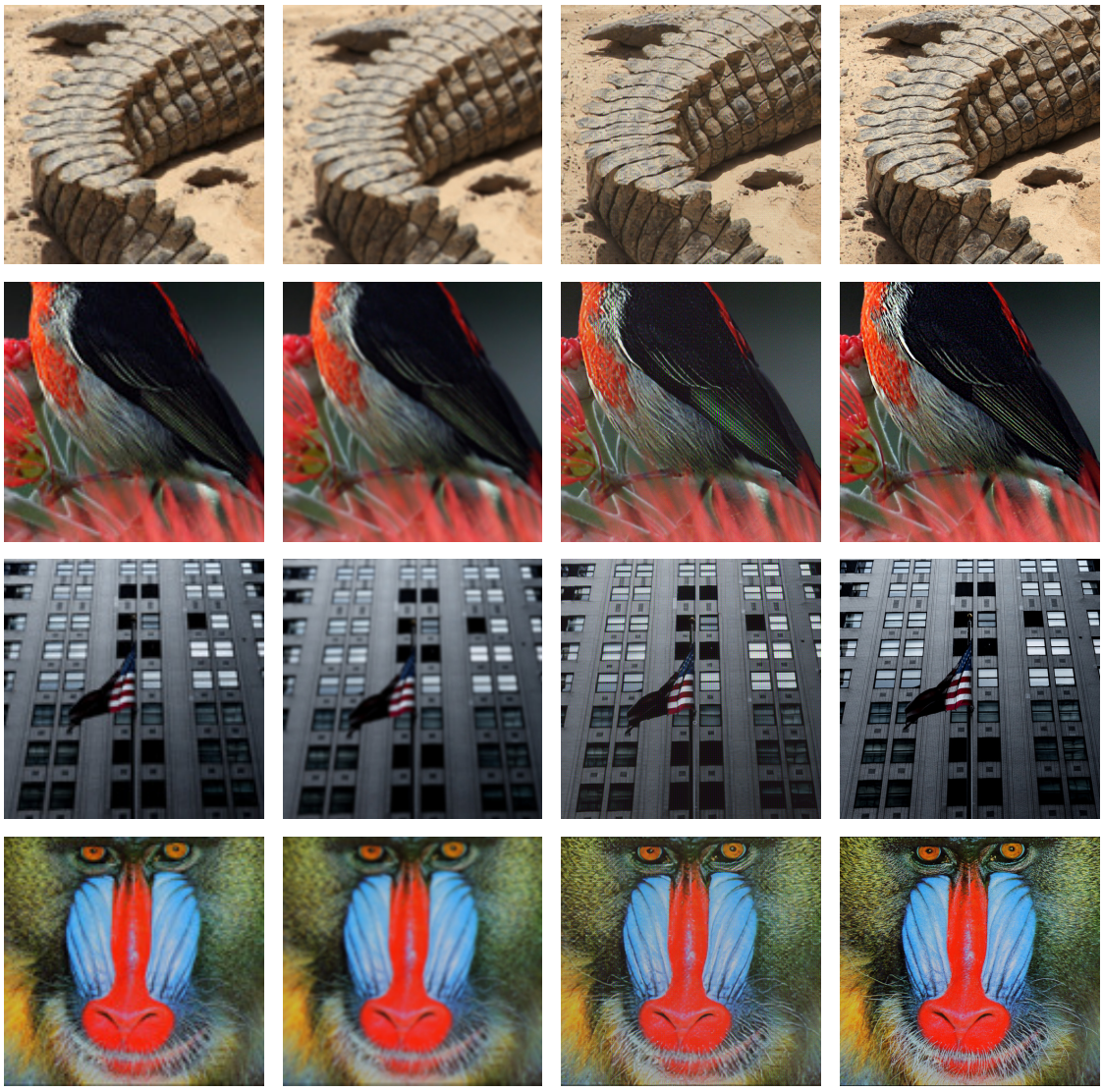

Figure B.2: Image columns from left to right: the low resolution input images, the B-spline interpolated images, the generated high resolution images with content loss and the real high resolution images.

B.3 General Adversarial Network (Perceptual Loss)

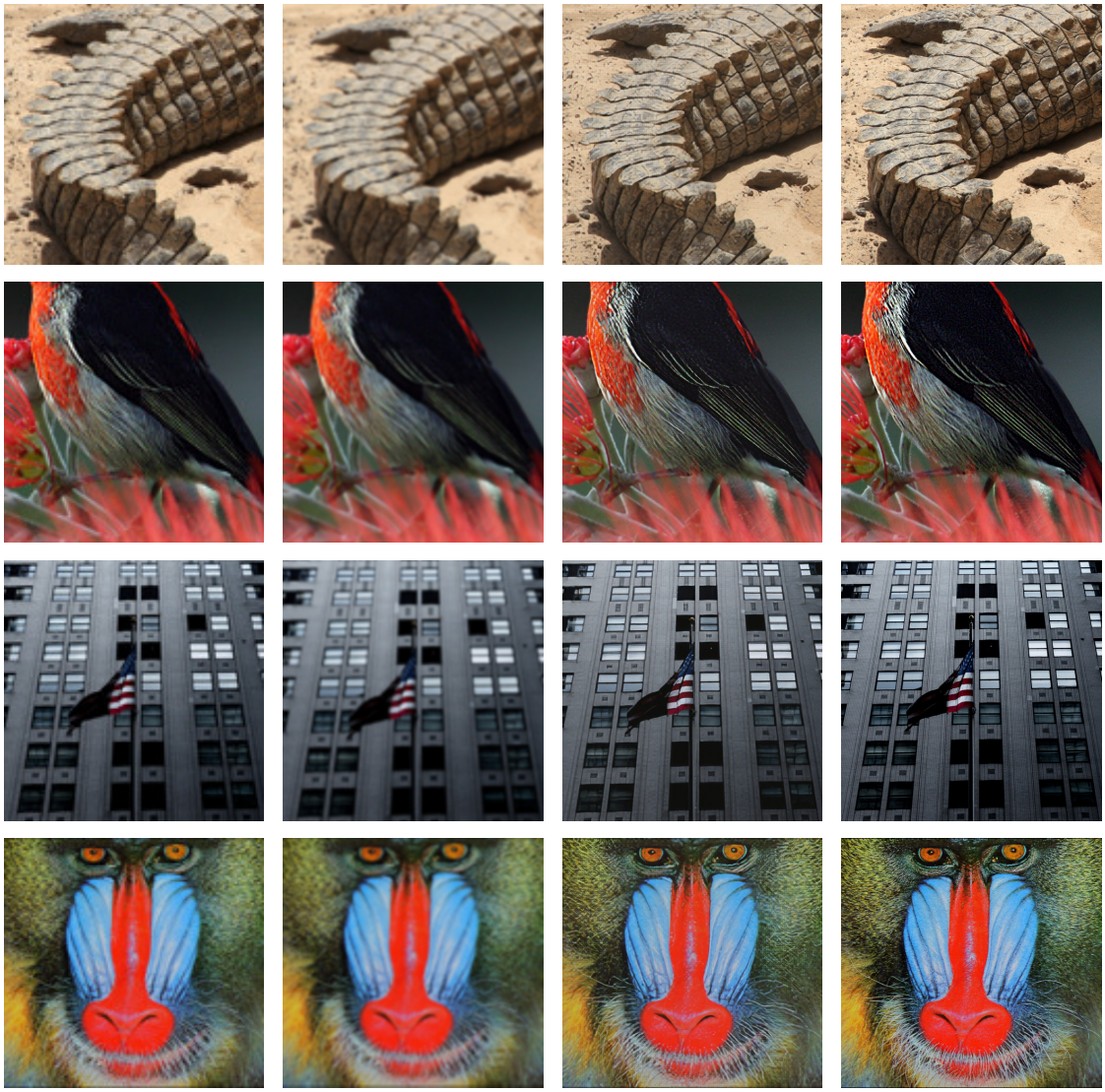

Figure B.3: Image columns from left to right: the low resolution input images, the B-spline interpolated images, the generated high resolution images with the GAN and the real high resolution images.

B.4 Network Interpolation

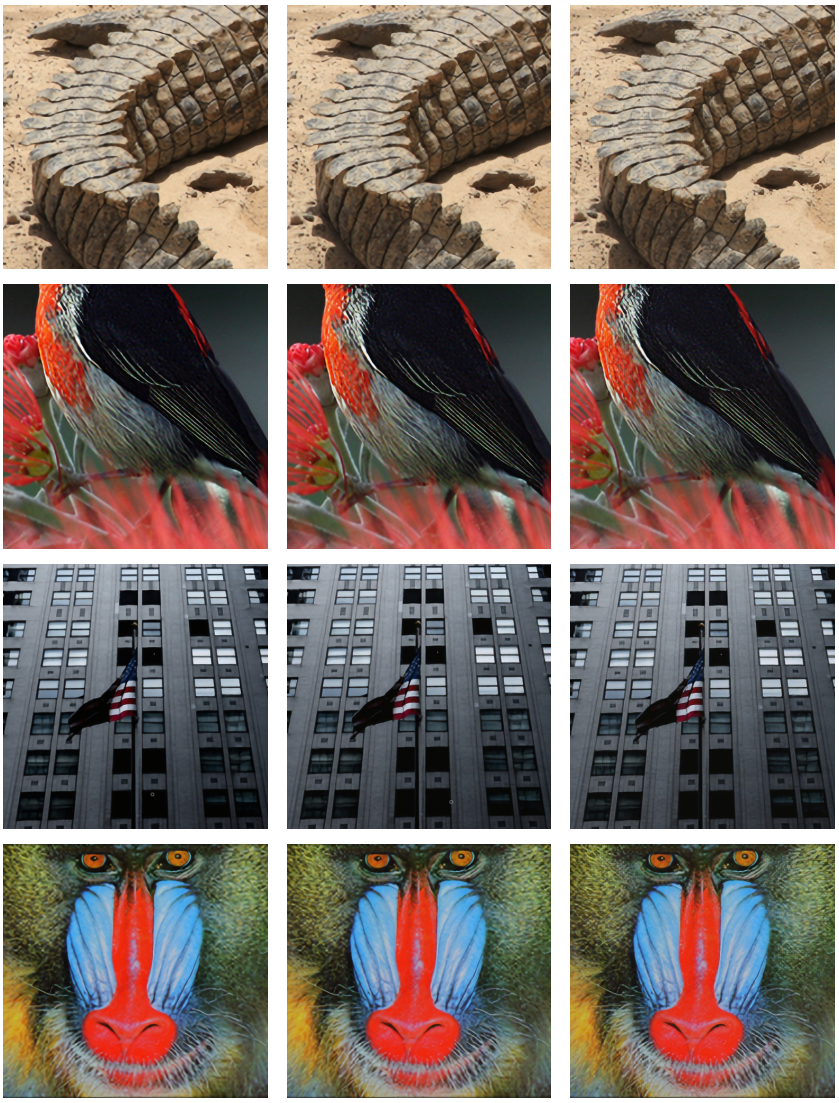

Figure B.4: The generated images with interpolation networks with three different interpolation values: $\alpha = 0.25$ (left column), $\alpha = 0.50$ (middle column) and $\alpha = 0.75$ (right column).

B.5 Comparison Side-by-side

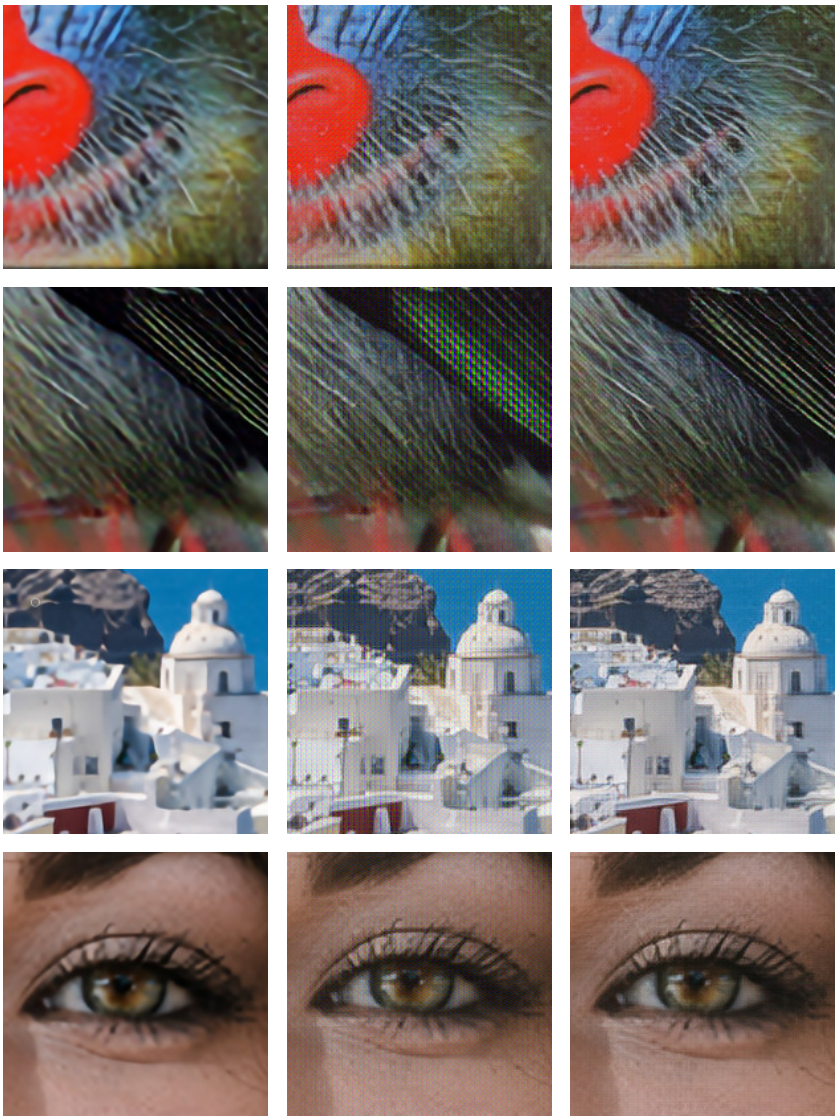

Figure B.5: Generated images with MAE (left column), content loss (middle column) and GAN (right column).

INSTITUTIONEN FÖR DATAVETENSKAP | LUNDS TEKNISKA HÖGSKOLA | PRESENTERAD 2019-06-04

EXAMENSARBETE Efficient Deep Learning Architectures for Super-Resolution
STUDENT Adam Thuvesen
HANDLEDARE Volker Krüger (LTH), Sebastian Raase (Sony)
EXAMINATOR Pierre Nugues (LTH)

Effektiva Deep Learning Arkitekturer för Super-Resolution

POPULÄRVETENSKAPLIG SAMMANFATTNING **Adam Thuvesen**

I Hollywood-filmer har man sedan länge på magiskt vis kunnat förbättra bildupplösningen till arbiträra nivåer utan någon försämrad bildkvalité. I verkligheten är det dock mycket svårt och beräkningstungt. Detta arbete utgår från denna problematik för att analysera effektiva lösningar för Super-Resolution med djupa neurala nätverk.

Ett vanligt problem för alla som någon gång tagit ett foto på långt avstånd, t ex på en konsert, är bilder med dålig upplösning. Men att förbättra bildupplösningen är svårt och resultatet blir med de flesta traditionella algoritmerna inte särskilt imponerande. En teknik för att lösa detta problem kallas Super-Resolution, där stora framgångar uppnåtts genom att använda djupa neurala nätverk. Dessa nätverk har dock i regel varit oerhört komplexa och beräkningstunga, vilket har lett till svårigheter att applicera dem i praktiken. Med denna problematik som utgångspunkt har detta examensarbete fokuserat på att analysera effektiva nätverksarkitekturer för denna uppgift. Målet har varit att undersöka hur man kan designa ett nätverk med god prestanda som både är litet och snabbt men samtidigt uppnår hög kvalitet på de uppskalade bilderna.

För att lösa problemet delades arbetet upp i två delar: först en evalueringsfas av diverse tekniker och därefter en designfas där ett nätverk utvecklades baserat på den mest lovande evalueringsdatan. Resultatet blev ett nätverk som uppnådde subjektivt hög kvalitet på bilderna, vilken kan ses i figuren, samt god prestanda i jämförelse med andra effektiva nätverk i samma storleksklass: (1) färre parametrar än de flesta, (2) liknande snabbhet och (3) bäst visuell kvalité på bilderna.

Figure 1: Lågupplöst bild som input (vänster) och genererad högupplöst bild (höger).

Resultatet tyder på att tekniken definitivt bör vara applicerbar i en rad olika områden redan idag. Men trots att nätverket är relativt litet och effektivt så krävs ändå tillgång till god beräkningskapacitet. Således bör användningsområden kunna vara applikationer där det finns tillgång till detta, som t ex analys av medicinska bilder eller för satellit- och flygfoto. För lättare enheter kvarstår problem, i huvudsak hårdvarubegränsningar samt plattformsupport. För att lösa detta krävs mer forskning på ännu mindre och effektivare nätverk samt utökat support för att implementera tekniken på lättare enheter.